Machbase 7.5 Manual Home

새 스페이스를 시작합니다

Confluence 공간은 콘텐츠와 소식을 팀과 공유하기에 매우 적합합니다. 이곳은 홈 페이지입니다. 지금은 최근의 공간 활동을 표시하지만, 원하는 대로 이 페이지를 사용자 정의할 수 있습니다.

시작하려면 다음을 수행하십시오.

- **이 개요 사용자 지정** 이 페이지의 오른쪽 상단에 있는 **편집 아이콘** 을 사용하여
- **새 페이지 만들기** 스페이스 사이드바에서 **+** 을 클릭한 다음 계속해서 계획, 아이디어 또는 다른 원하는 것으로 채우십시오.

영감이 필요하신가요?

- 스페이스가 무엇이고, 가장 잘 활용하는 방법에 대한 간단한 소개를 보려면 다음을 참조하십시오: Confluence 101: 스페이스에서 작업을 체계화
- 가이드에서 다음 방법에 대한 아이디어를 확인하십시오: 스페이스 설정 개요.
- 빈 스페이스에서 시작하기가 어려울 경우 대신 스페이스 템플릿 중 하나를 사용해 보십시오.

Machbase 7.5 Manual

개요

- 마크베이스 소개
- 마크베이스 특징
- 마크베이스 제품군 소개

주요 기능/테이블

- 태그 테이블 (Tag Table)
- 로그 테이블 (Log Table)
- 참조 테이블 (Lookup Table)
- 휘발성 테이블 (Volatile Table)
- 스트림 (STREAM)
- 백업 및 마운트

설치

- 패키지 개요
- Linux 설치
- Windows 설치
- 라이선스 설치
- Cluster Edition 설치

업그레이드/복구

- Cluster Edition 업그레이드 및 복구

설정/모니터링

- Meta Table
- Virtual Table
- Property
- Property (Cluster)

도구

- 유틸리티 모음
- MWA (Machbase Web Analytics)
- 태그 분석기 (Tag Analyzer)
- machcoordinatoradmin
- machdeployeradmin
- 컬렉터 (Collector)

SQL 레퍼런스

- 자료형
- DDL
- DML
- SELECT
- SELECT Hint
- 사용자 관리
- 지원 함수
- 시스템/세션 관리

SDK

- CLI/ODBC
- CLI/ODBC 예제
- JDBC
- Python
- RESTful API
- .NET Connector
- Timezone

자주 묻는 질문(FAQ)

- 쿼리 에러를 Property 수정하여 해결하는 방법

개요

마크베이스 소개

마크베이스(Machbase)는 다양한 IoT 환경에서 발생하는 대량의 "센서 데이터"를 실시간으로 저장할 뿐만 아니라, 실시간 데이터 분석이 가능한 시계열 데이터베이스이다.

마크베이스는 기존의 솔루션으로는 해결할 수 없었던 센서 데이터에 대한 데이터 저장과 처리에 대한 대량의 부하 문제를 해결하였으며, 다양한 기능을 통해 앞으로 폭증하는 센서 데이터에 대한 훌륭한 솔루션을 제공한다.

새로운 데이터의 출현

최근 정보통신 분야가 유무선 통신에 기반한 사물인터넷(IoT: Internet of Things) 환경으로 발전하고 있으며, 이를 통해 다양하고 방대한 양의 데이터가 축적되고 있다.

따라서 이런 데이터를 실시간으로 분석할 수 있다면 이전에 발생하였거나 앞으로 발생할 수 있는 수많은 장애들을 파악하고 방지할 수 있을 것이다.

특히 몇 년 사이에 데이터를 생성하는 원천 디바이스의 숫자가 급격하게 늘어나고, 이에 뒤따르는 데이터의 발생량도 몇 년 만에 수십 배로 증가하고 있다.

반면 데이터를 처리하는 전통적인 소프트웨어에는 이에 대한 적절한 해결책을 제시하지 못하고 있다.

전통적인 솔루션이 현재 데이터의 처리에 적합하지 않은 이유는 다음과 같다.

첫째, 데이터 발생 속도가 어마어마하게 증가하고 있다.

데이터 원천 소스가 늘어나고 처리해야 할 정보의 종류가 증가함에 따라 서버에 데이터가 저장되는 속도가 기존과는 완전히 다른 속도로 도착하고 있다.

초당 수만 건에서 수십만 건의 데이터를 저장하기 위해 최적화된 소프트웨어 솔루션은 파일 시스템에 일반 Text 파일로 저장하는 것 외에는 해결책이 없는 실정이다.

둘째, 데이터의 발생 속도와 비례하여 실시간 데이터 분석의 수요가 증가하고 있다.

이런 빅데이터를 활용하여 의사 결정에 도움을 주는 것이 매우 중요하지만 기존 솔루션은 초당 수십만 건에 이르는 데이터를 실시간으로 인덱싱하고 검색 및 분석을 통해 결과를 사용자에게 전달하는 기술적 성숙도가 높지 않다.

셋째, 앞에서 언급했던 "센서 데이터"의 데이터 특성이 완전히 다르다.

전통적인 데이터베이스는 센서형 머신 데이터를 처리하기에 부적당한 아키텍쳐를 가지고 있으며, 이를 해결하기 위해서는 기술적으로 새로운 접근이 필요하다.

이런 이유로 마크베이스는 새로운 형태의 "센서형 머신 데이터"를 처리하는 최고의 아키텍쳐를 통해 새롭게 개발되었으며, 전통적인 기술로 해결하지 못했던 실시간 데이터에 대한 저장과 처리 및 분석이 가능한 유일한 데이터베이스 솔루션이다.

센서 데이터의 형태

ID	Time	Data
Temperature01	2020-01-01 00:00:00	20
Pressure01	2020-01-01 00:00:00	0.98

위의 그림은 전형적인 센서 데이터의 형태를 나타낸 것이다. 크게 3가지의 속성으로 구분할 수 있으며 각각의 성질과 특성은 다음과 같다.

ID

이 값은, 해당 머신 데이터가 발생한 디바이스(원천 소스)의 유일성을 나타내는 기호 및 숫자를 나타낸다. 해당 머신이나 센서의 일련 번호로 구성되며 32비트 혹은 64비트의 정수로 표현된다.

TIME

이 값은, 해당 머신 데이터가 발생한 순간의 시간을 나타낸다. 이 시간은 지속적으로 증가하는 경향이 있으며, 일반적으로 초 단위까지 표현하고 특별한 경우 나노초(nanosecond)까지 표현되기도 한다.

DATA

이 값은, 2진 데이터로 주로 정수형, 실수형 숫자 혹은 IP 주소값 등의 형식의 데이터이다. 대표적으로 이 영역에는 특정 센서에서 나오는 온도, 가속도, 밝기 등의 숫자형 값이나, IPv4 혹은 IPv6와 같은 4바이트 혹은 16 바이트의 고정 데이터가 포함된다.

마크베이스는 Tag Table을 제공하여, 위 형태의 센서 데이터를 초당 수십만건을 받아들일 수 있는 최적화된 아키텍처를 제공한다.

PLC에서의 센서 데이터 구조

실제로 현장 (PLC : Programmable Logic Control) 에서 사용되는 센서 데이터는 위의 세 가지 속성으로만 표현되지 않는다.

대부분의 경우 아래와 같이 다수의 센서 데이터가 모여진 형태로 저장되는 것이 일반적이다.

따라서 아래 형태의 데이터도 함께 받아 들일 수 있고, 쉽게 변환이 되어야 한다.

```
data from PLC
```

Time	SN01M	SN02M	SN03M	SN04M	SN05M	SN06M	SN07M	SN08M	SN09M	SN10M
04:01:56.005	11.1	1	0	0	0	1	0	0	0	0
04:01:56.057	11.3	1	0	0	0	1	0	0	0	1
04:01:56.109	11.1	1	0	1	0	1	0	1	1	0
04:01:56.161	12.3	1	0	0	0	1	0	0	0	1
04:01:56.213	9.1	1	1	0	0	1	0	0	0	0
04:01:56.266	0.9	1	0	0	1	1	0	0	0	0
04:01:56.319	8.9	1	0	1	0	1	0	1	0	1
04:01:56.370	1.3	1	0	0	0	1	0	0	0	0
04:01:56.422	33.1	1	0	0	0	1	0	0	0	0
04:01:56.474	3.3	1	0	0	0	1	0	0	0	0
04:01:56.526	5.6	1	0	0	1	1	0	0	0	1
04:01:56.578	5.5	1	0	0	0	1	0	0	1	0
04:01:56.630	4.5	1	0	0	0	1	0	0	0	0
04:01:56.682	5.3	1	0	0	0	1	0	0	0	0
04:01:56.733	1.2	1	0	0	0	1	0	0	0	1
04:01:56.785	3.4	1	0	0	1	1	0	0	0	0
04:01:56.838	11.3	1	0	1	0	1	0	1	0	0
04:01:56.890	11.2	1	0	0	0	1	0	0	0	0
04:01:56.942	9.9	1	0	0	0	1	0	0	0	1
04:01:56.994	8.4	1	0	0	0	1	0	0	0	0
04:01:57.046	8.4	1	0	1	0	1	0	0	0	0
04:01:57.097	1.2	1	0	0	1	1	0	1	1	1
04:01:57.149	1.3	1	0	0	0	1	0	0	0	0
04:01:57.200	11.2	1	0	0	0	1	0	0	0	0
04:01:57.252	3.1	1	0	0	0	1	0	0	0	0

마크베이스는 Log Table을 제공하여, 위와 같은 형태의 PLC 데이터를 저장할 수 있는 구조 또한 제공한다.

이 테이블 역시 초당 수만건의 데이터를 입력하고, 분석할 수 있는 능력을 제공한다.

마크베이스 특징

다양한 테이블 구조 지원

마크베이스는 사용자의 용도에 따라 네가지 형태의 테이블을 각각 제공한다. (Tag, Log, Volatile, Lookup)

이는 센서데이터를 저장하는 고객의 요구 사항이 매우 다양하고, 하나의 비지니스가 하나의 특정 데이터 패턴 만을 담고 있지 않기 때문이기도 하다.

따라서, 그러한 비지니스 요구 사항을 잘 이해하고, 적절한 테이블을 선택하는 것이 중요하다.

아래는 각 테이블의 특성을 구분하여 나타낸 것이다.

테이블 종류	Tag Table	Log Table	Volatile Table	Lookup Table
목적	*<센서명, 시간, 센서값>* 형태의 센서 시계열 데이터 처리에 최적화	PLC 형태 로그 시계열 데이터 처리에 최적화 (텍스트 포함)	휘발성 메모리 데이터 실시간 처리	영구 저장 가능한 마스터 데이터 관리
설명	고속으로 센서 데이터를 저장하고, 고속으로 해당 데이터를 추출하거나, 실시간으로 통계 테이블을 생성할 경우 활용 주로, 실시간 센서 데이터를 저장	텍스트를 포함한 로그성 데이터를 저장하고, 이를 일반 DBMS 형태로 분석하고자 할 경우 활용 주로, 히스토리성 사용자 데이터를 저장	Insert, Delete, Update, Select 가 메모리 기반의 성능으로 필요한 경우 사용 (초당 수만건) **시스템 종료시 모든 데이터가 사라지며,** 주로 key-value 기반의 모니터링 용도로 활용함.	사용자의 변경 가능한 주요 마스터 데이터를 영구 저장할 목적으로 사용함. SELECT 성능은 고속이지만, 나머지 INSERT, UPDATE, DELETE는 디스크 기반의 성능을 제공함.
테이블 구조	*<센서명, 시간, 센서값>* 이 기본형이며, 추가로 컬럼을 지정할 수 있다.	임의의 스키마 가능	임의의 스키마 가능 (Primary Key 지정 가능)	
INSERT (입력) 성능	초당 수백만건	초당 수백만건	초당 수만건	초당 수백건
SELECT (질의)	센서명 + 시간 범위 한정	모든 질의 가능		
DELETE (삭제)	임의 시점 이전 데이터 실시간 삭제 가능	임의 시점/구간 데이터 실시간 삭제 가능	Primary Key 기준 Record Delete 지원 (※ Primary Key 지정 필요)	
UPDATE (변경)	지원 불가 (※ 메타데이터 컬럼 에 한해서 변경 가능)	지원 불가	Primary Key 기준 Update 지원 (※ Primary Key 지정 필요)	
저장소 크기 한계	디스크 한계		메모리 한계	
INDEX 구조	3 단계의 Partitioning 실시간 인덱스 (※ 기본 생성)	LSM 인덱스	Red/Black 메모리 인덱스	
STREAM 지원	타겟 대상으로만 가능 (저장 대상)	소스/타겟 대상 모두 가능 (읽기 및 저장대상)	불가능	
고려 사항	과거 데이터 삭제를 고려한 충분한 스토리지 확보 고려	Tag 입력을 위한 임시 저장소로 고려	메모리 한계 고려	

다양한 크기의 하드웨어 지원

마크베이스는 사용자의 환경에 따른 아래와 같은 다양한 제품 Edition을 제공한다.

Edge Edition

이 제품은 ARM 혹은 인텔의 ATOM 급 CPU를 기반으로 동작하는 소규모 Edge 장비에서 동작한다.

그러나, 이런 소규모 장비에서도 초당 수만건의 센서 데이터를 저장하고, 필터링을 하고자 하는 경우 마크베이스가 유용하게 활용될 수 있다.

주로, 로봇이나 공장의 생산 설비, 빌딩 등의 단말 단계에서의 다양한 센서를 고속 및 고용량으로 저장하고자 하는 경우 필요한 제품이다.

Fog Edition

이 제품은 단일 서버에서 고속의 데이터 처리를 달성하고자 하는 경우 활용된다.

주로 인텔 x86 CPU 기반의 윈도우나 리눅스 운영체제에서 동작하며, 타 DBMS가 제공하지 못하는 매우 빠른 센서 데이터 저장과 분석을 제공한다.

대부분의 경우 수백대 이상의 Edge 장비로부터 실시간으로 입력되는 데이터를 저장하고, 이를 2차로 분석하기 위한 용도로 활용된다.

Cluster Edition

이 제품은 거대 제조 공장을 위한 초거대규모의 센서 데이터를 저장하기 위한 목적으로 개발되었다.

반도체 혹은 디스플레이, 발전, 철강 생산 공정에서 발생하는 초당 천만건 이상의 데이터를 저장하기 위해 다수의 물리적 서버가 클러스터 형태로 동작한다.

데이터가 늘어나는 환경에서 처리 용량과 성능을 지속적으로 유지해야 하는 환경에서 활용된다.

Tag analyzer : 데이터 시각화 솔루션 지원

마크베이스는 버젼 5부터 마크베이스에 저장된 수백억건의 센서 데이터에 대한 실시간 시각화를 제공한다.

즉, 임의의 태그 아이디를 지정하며, 그 아이디가 입력된 기간동안의 트렌드 차트를 순식간에 웹 기반으로 확인할 수 있도록 한다.

또한, 단순한 태그 데이터 뿐만 아니라 그 기간동안의 통계 챠트도 함께 볼 수 있도록 제공하기 때문에 단순 시각화를 넘어 일정 수준의 통계 분석도 가능하다.

Write Once, Read Many

센서 데이터는 일단 데이터베이스에 입력되면 변경 또는 삭제되는 경우가 거의 없다.

따라서, 마크베이스는 머신 데이터에 대한 특성을 최대한 살리기 위해 한번 입력된 주요 시계열 데이터에 대해서는 UPDATE가 발생할 수 없도록 설계 되었다.

한번 입력된 로그 데이터는, 악의적 사용자에 의해 변조되거나 삭제되지 않으므로 걱정할 필요가 없다.

Lock-free 아키텍쳐 지원

센서 데이터 처리하는데 가장 중요한 것은 데이터의 입력, 변경, 삭제 연산과 읽기 연산이 서로 충돌하지 않고가능한 독립적으로 처리되어야 한다는 것이다.

이 때문에 마크베이스는 SELECT 연산에 대한 어떠한 Lock도 할당 받지 않도록 설계되었고, 변경 연산인 입력 혹은 삭제와도 서로 절대로 충돌하지 않는 고성능 구조로 설계되었다.

따라서 수십만 건의 데이터가 입력되고, 실시간으로 일부가 삭제되는 상황에서도 SELECT 연산은 수백만 건의 레코드에 대한 통계 연산을 빠른 속도록 진행할 수 있다.

초고속 데이터 저장

마크베이스는 기존의 데이터베이스보다 수십배의 빠른 데이터 저장 성능을 제공한다. 특정 데이블에 인덱스가 다수 존재하는 상황에서도 최소 초당 300,000 건에서 최고 2,000,000 건까지 데이터를 받아들일 수 있다.

이것이 가능한 이유는 마크베이스가 시계열 데이터를 최적화하는 구조로 설계되었기 때문이다.

STREAM 기능 지원

마크베이스는 버젼 5 부터 Edge Edition과 Fog Edition에 대해서 실시간 데이터 필터링을 지원하기 위해 STREAM 기능을 제공한다.

이 STREAM은 DBMS 내부에서 실시간으로 입력되는 데이터에 대해 고속으로 조건 평가를 수행하고, 그 결과를 임의의 테이블로 전송하는 역할을 수행한다.

이 기능은 특정 센서의 값이 특정 범위를 넘었을 경우 경고를 발생시키거나 내부적으로 입력된 데이터에 대한 실시간 평가를 하는 경우 매우 유용하다.

실시간 인덱스 구성

마크베이스는 인덱스가 많으면 많을수록 데이터 입력 성능이 비례적으로 느려지는 전통적인 데이터베이스 구조를 혁신적으로 개선해, 초당 수십만건의 데이터가 입력되더라도 거의 실시간으로 인덱스를 구성할 수 있다.

이 특징은 실제 데이터가 발생하는 순간의 즉시 검색할 수 있는 강력한 기능적인 토대를 제공해 주기 때문에 머신 데이터와 같은 시계열 데이터 분석에 있어서는 핵심적인 기술이다.

실시간 데이터 압축

머신 데이터와 같은 시계열 데이터의 특징은 끊임없이 데이터가 발생한다는 것이다. 이 사실은 필연적으로 해당 데이터베이스의 저장 공간이 언젠가는 부족해질 뿐만 아니라, 처리해야 할 데이터를 충분하게 보유하지 못한다는 의미이다.

특히, 전통적인 데이터베이스는 고속 입력 시 데이터 뿐만 아니라 인덱스가 늘어남에 따라 차지하는 데이터 공간 역시 급격하게 증가한다. 이 때문에 머신 데이터의 저장과 분석에 매우 부적절한 구조이다.

마크베이스는 쏟아져 들어오는 데이터에 대해 혁신적인 실시간 압축 기술 2 가지를 통해 성능 저하 없이 적게는 수십배에서 수백배까지 데이터를 압축하여 저장한다.

논리적 실시간 데이터 압축 기술 지원

첫 번째로 마크베이스는 논리적 실시간 데이터 압축 기술을 지원한다.

이는 컬럼형 데이터베이스에서 유래한, 머신 데이터의 데이터 중복성을 이용한 것으로서 동일한 값을 갖는 데이터가 많으면 많을수록 데이터의 중복을 코드화하여, 데이터 저장공간을 혁신적으로 줄이는 기술이다. 이를 통해 데이터의 중복성이 높은 데이터에 대해 수백배까지 데이터를 압축할 수 있다.

물리적 데이터 압축 기술 (특허 기술)

두 번째는 마크베이스의 특허 기술인 물리적 데이터 압축기술이다.

이는 디스크에 저장될 물리적인 데이터 블럭을 미리 일정한 크기의 파티션으로 나누어 압축하여 디스크에 별도로 내림으로써 저장될 물리적 데이터의 량을 줄이고, 더불어 시스템이 유발시키는 I/O 비용을 급격하게 낮추는 기술이다. 이를 통해 실제 논리적으로 압축된 데이터를 다시한번 더 압축하여 저장 공간의 효율성을 높이는데 일조한다.

탁월한 질의 성능

마크베이스의 혁신적인 기술적 우월성은 초당 수십만 건의 데이터를 입력하는 와중에도 이미 저장된 과거의 수백만 혹은 수천만 건의 데이터에 대한 검색 및 통계 분석 성능이 매우 빠르다는 것이다.

삽입과 분석 모두에 탁월한 성능을 제공하는 마크베이스 만의 인덱싱 기술 때문에 가능한 것이며 실시간 비즈니스 의사 결정에 핵심적인 역할을 수행할 것이다.

전통적인 데이터베이스와 달리, 마크베이스는 두 개 이상의 인덱스를 하나의 질의문에서 처리할 수 있기 때문에 병렬로 데이터를 처리할 경우 몇 배나 더 빠른 성능을 기대할 수 있다.

아래는 다음과 같은 질의문에 대해 두 개 이상의 인덱스를 활용하는 경우를 나타낸 것이다.

```
SELECT * FROM table1 WHERE c1 = 1 and c2 = 2;
```

시계열 데이터 특성 SQL 구문 지원

센서 데이터의 경우 최신 데이터가 예전의 데이터보다 몇 배 더 가치가 있으며 데이터의 접근 빈도도 최근 데이터가 예전 데이터보다 몇 배 더 많은 특징이 있다.

이런 이유로 마크베이스는 두 종류의 테이블 즉, 태그 (Tag) Table과 로그 (Log) Table을 통해 시계열 데이터 특징을 지원한다.

로그 테이블

마크베이스에서 지원하는 로그 테이블 (Log Table) 은 다음 특징을 가지고 있다.

첫째, 입력 시간을 자동으로 저장

레코드가 데이터베이스에 저장되는 순간 나노 세컨드 단위의 timestamp를 _arrival_time이라는 필드로 저장한다.

이 의미는 마크베이스가 저장하는 모든 레코드는 시간을 기준으로 검색하거나 조건을 줄 수 있다는 것이다.

둘째, 최근 데이터 우선 조회

데이터 검색시 최근 시간이 예전 시간 보다 먼저 출력된다. 즉, SELECT를 수행할 때 최근 데이터가 먼저 출력된다는 것이다.

앞에서 언급한 _arrival_time 컬럼 기준으로 내림차순 정렬 (descending sort) 를 한 것과 같은 결과이다.

셋째, DURATION 키워드

DURATION 키워드를 제공해, 특정 시간 범위 데이터를 입력 시간 기준으로 빠르게 조회할 수 있는 기능을 탑재했다.

머신 데이터 분석의 경우 특정 시간 범위를 지정하는 경우가 많기 때문에 SQL 레벨에서 이러한 특성을 제공한다.

이를 통해 복잡한 시간 연산자를 where 절에 주지 않더라도 편리하게 데이터를 분석할 수 있다.

```
-- 예1) 지금 부터 10분 전까지의 데이터 통계 조회
SELECT SUM(traffic) FROM t1 DURATION 10 MINUTE;

-- 예2) 지금 부터 1시간 전에 30분간의 데이터 통계 조회
SELECT SUM(traffic) FROM t1 DURATION 30 MINUTE BEFORE 1 HOUR;
```

태그 테이블

마크베이스 5.0 부터 지원하는 태그 테이블 (Tag Table) 은 다음 특징을 가지고 있다.

첫째, 고속 TAGID/시간 조건 검색 성능

태그 테이블은 임의의 시간 및 임의의 ID 기반 검색 성능이 탁월하다.

기존의 RDBMS로는 도달할 수 없는 초고속의 데이터 추출 성능을 자랑하며, 이는 수십억건의 센서 데이터가 저장된 상황에서도 동일한 속도를 보장한다.

둘째, 고속 태그 데이터 입력

태그 테이블은 고속의 데이터 입력을 지원한다.

앞의 로그 테이블과 마찬가지로, 초당 수십만건의 센서데이터 입력에 있어서도 무리없이 데이터를 입력할 수 있다.

셋째, 실시간 통계 기능

태그 테이블은 실시간 통계 기능을 지원한다.

마크베이스는 이 태그 테이블에 저장된 데이터의 경우 실시간으로 다섯 종류의 통계를 자동적으로 생성하고, 이를 실시간으로 접근할 수 있는 기능을 제공한다.

텍스트 검색 기능 지원

로그성 시계열 데이터를 저장하고 활용하는 사용자의 가장 중요한 실제 용도 중 하나는 특정 시점에 특정 event 가 발생했는지를 확인하는 것이다.

특정 시점의 경우 시계열 데이터 처리로 가능하지만, 특정 event가 발생한 것은 대부분의 경우 특정 컬럼에 저장된 text field에서 특정 "단어"를 찾는 행위가 필요하다.

그러나, 전통적 데이터베이스에서는 특정 필드의 단어를 검색하기 위해서는 B+ Tree를 통해 exact match 혹은 LIKE 절을 통해 최초 일부 캐릭터의 조건을 검사하게 되는데, 대부분의 경우 이는 매우 느린 응답 결과를 초래한다.

그런 이유로 전통적인 데이터베이스에서 특정 단어에 대한 검색은 매우 취약하다.

반면, 마크베이스에서는 로그 테이블 기반의 SEARCH라는 SQL 키워드를 제공함으로써 실시간 단어 검색이 가능하도록 하였다.

이를 통해 장비로부터 발생된 임의의 에러 텍스트를 순식간에 검색할 수 있게 되었다.

```
-- 예1) msg 필드에 Error 혹은 102를 포함하는 레코드를 출력
SELECT id, ipv4 FROM devices WHERE msg SEARCH 'Error' or msg SEARCH '102';

-- 예2) msg 필드에 Error 그리고 102를 포함하는 레코드를 출력
SELECT id, ipv4 FROM devices WHERE msg SEARCH 'Error 102';
```

선택적 삭제 지원

센서 데이터의 경우에는 삽입 이후에 삭제 연산이 거의 발생하지 않는 것이 현실이다.

그러나 embedded 장비의 경우에는 저장 공간의 제약이 분명히 존재하고, 사용자에 의해 주의 깊게 관리되지 않는 것이 그 특징이다.

이 경우 혹시나 머신 데이터에 의해 Disk full이 발생하거나 장애가 발생하게 되면, 기업 입장에서 많은 손해를 감수해야 한다.

마크베이스는 이런 환경에서 주어진 특정 조건에 레코드를 삭제할 수 있도록 기능을 제공한다.

따라서 embedded 개발사는 CRON 혹은 주기적인 프로그램을 통해서 마크베이스가 일정 크기 이상의 데이터를 유지하지 않도록 손쉽게 관리할 수 있다.

로그 테이블의 경우

아래 모든 문법 지원

```
-- 사용 예1) 가장 오래된 마지막 100건을 삭제하라.
DELETE FROM devices OLDEST 100 ROWS;

-- 사용 예2) 최근 1000건을 제외하고 모두 삭제하라.
DELETE FROM devices EXCEPT 1000 ROWS;

-- 사용 예3) 지금부터 하루치를 남기고 모두 삭제하라.
DELETE FROM devices EXCEPT 1 DAY;

-- 사용 예4) 2014년 6월 1일 이전의 데이터를 모두 삭제하라.
DELETE FROM devices BEFORE TO_DATE('2014-06-01', 'YYYY-MM-DD');
```

태그 테이블의 경우

아래 한가지 문법 지원

```
-- 2016년 6월 15일 이전의 데이터를 모두 삭제하라.
DELETE FROM tag BEFORE TO_DATE('2016-06-15', 'YYYY-MM-DD');
```

자동화된 데이터 수집

마크베이스는 산재해 있는 머신 데이터 로그 파일로부터 데이터를 읽어 자동으로 전송해주는 기능인 "컬렉터 (Collector)" 기능을 제공한다.

이를 통해 syslog 나 웹서버 로그 등의 이미 정형화된 데이터를 수집할 수 있을 뿐만 아니라 사용자가 임의로 정의한 로그 포맷의 경우에도 매우 쉽게 변환하여 자동으로 수집할 수 있는 기능을 제공한다.

마크베이스 제품군 소개

마크베이스는 다음과 같은 3가지의 제품군을 가지고 각각에 맞는 형태의 비지니스 환경에 적용할 수 있다.

- Edge Edition
- Fog Edition
- Cluster Edition

Edge Edition

필요성

Edge Edition은 한정된 리소스를 가진 소규모 장비에서 고속으로 동작하는 마크베이스 제품군을 일컫는다.

2010년대 후반부터 어느정도의 컴퓨팅 파워를 가진 Edge 장비들이 출시되고 있으며, Industrial IoT 분야에서 이 Edge 장비를 통한 다양한 비지니스가 개발되고 있다.

특정 환경에서 생산 설비의 상태를 모니터링하거나, 로봇 혹은 대량의 센싱이 필요한 곳에서 발생하는 데이터를 1차적으로 저장하고, 필요한 경우 상위 서버로 전송하는 구조가 일반적이다.

그러나, 실제로 이러한 소규모 Edge 장비에서 원활하게 동작하기 위해서는 고속으로 데이터를 저장하고, 실시간으로 데이터를 추출할 수 있는 데이터베이스가 핵심이다.

마크베이스는 본연의 혁신적인 기술을 통해 이 요구가 가능하게 하였으며, 아래와 같은 하드웨어를 지원한다.

지원 하드웨어

	Rasberry PI 3	Samsung ARTIK 7	LattePanda	nvidia Jetson TX2
형태				
CPU	ARM Cortex-A53 (64bit)	ARM Cortex-A53 (64bit)	Intel ATOM (x5-Z8350)	ARM Coretx-A57 (64bit)
MEMORY	1 GB	1 GB	4 GB	8 GB

기능 제약

한정된 리소스를 가진 소규모 장비에서 동작해야 함에 따라 다음과 같은 기능 제약을 가진다.

```
- rollup 기능 없음
- backup 기능 없음
- mount/restore 기능 없음
```

Fog Edition

필요성

Fog Edition은 Edge에서 전송되는 대량의 데이터를 저장하고, 분석하기 위해 사용되는 제품이다.

이 Fog Edition은 단일 하드웨어 장비에서 최대한의 성능을 낼 수 있도록 설계 되었기 때문에 복잡한 설치와 구성 필요 없이 매우 쉽고 빠르게 설치, 관리, 운용할 수 있다.

특히, 단일 Appliance에 내장되어 공장 및 건물의 장비로 설치 및 활용되며, 고객의 데이터 수집을 1차적으로 담당하는 역할을 주로 수행한다.

지원 하드웨어 및 운영체제

Fog Edition은 인텔 CPU 기반의 64-bit Linux 및 Windows 2000 이상의 운영체제를 지원한다.

Cluster Edition

Cluster Edition은, 빠른 입력 속도와 표준 SQL로 조회가 가능한 Machbase Fog Edition 으로도 처리할 수 없는 초대용량의 데이터 입력/조회를 분산 환경에서 처리할 수 있는 제품이다.

왜 Cluster Edition을 써야 하는가?

Machbase 초고속으로 시계열 데이터를 입력받을 수 있는 Fog Edition 을 서비스하고 있다. 그러나, 다음의 단점이 존재한다.

- 하나의 프로세스로 구성되어 있기 때문에, 가용성이 떨어진다.
- 데이터를 분석할 때 하나의 프로세스가 전담하므로, 대용량 데이터 분석 성능을 늘리는 데 한계가 있다.

위의 단점을 극복하기 위해, 즉 가용성을 확보하고 대용량 데이터를 저장하고 분석하는데 확장성을 확보할 수 있는 새로운 제품군이 필요하다. 이런 요구사항을 만족할 수 있는 것이 Machbase Cluster Edition 이다.

용어

Host

물리적인 서버 1대, 또는 클라우드/VM 에서의 OS 인스턴스 1대를 나타낸다.

Node

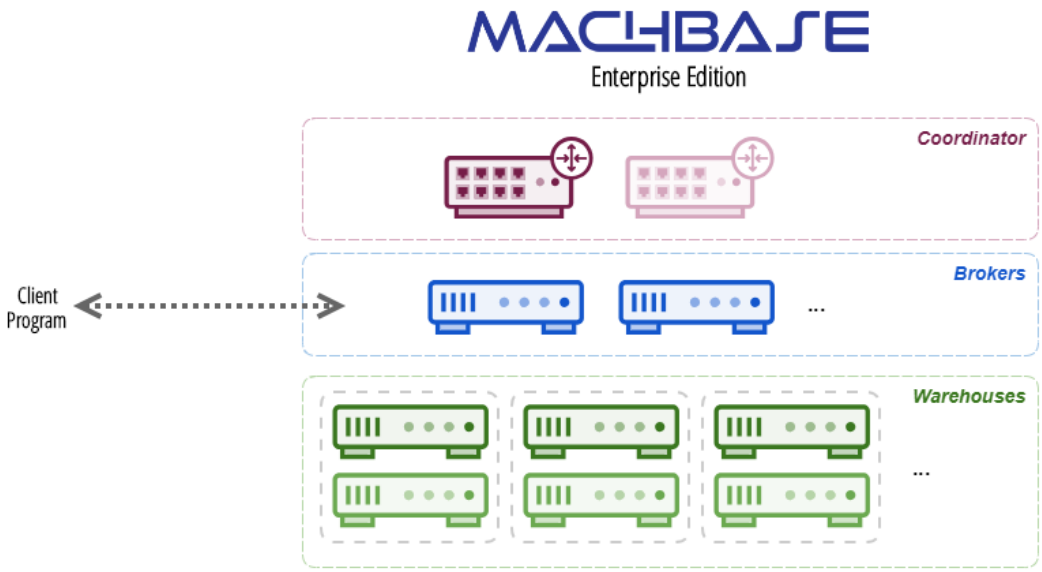
서버에 상주하는 Machbase Process 를 나타낸다.

Process 타입은 아래의 Node Type 과 동일하다.

- Coordinator
- Deployer
- Lookup
- Broker
- Warehouse

구조

Machbase Cluster Edition 은, Host 에 상주하는 여러 Node 가 하나의 클러스터를 구성한다.



고가용성

내부의 모든 Node 중 하나가 중단되어도 서비스가 지속될 수 있도록 한다.

고확장성

데이터 저장을 분산할 수 있고, 분산된 데이터에서 병렬 분석이 가능하므로 클러스터를 확장할 수록 성능이 증가한다.

Node의 분류

각 Node는 다음과 같이 분류할 수 있다.

분류	설명	프로세스 이름
Coordinator	모든 범용 서버와 Node를 관리하는 프로세스.	machcoordinatord
Deployer	Host 마다 하나씩 상주하는 프로세스. Broker/Warehouse 의 설치와 업그레이드, 상태 관찰을 담당한다.	machdeployerd
Lookup	Lookup 테이블 데이터를 가지고 있는 프로세스.	machlookupd
Broker	실제 클라이언트 프로그램을 맞이하는 프로세스. 클라이언트의 데이터 입력/데이터 조회 쿼리를 Warehouse 에 분산 수행시키는 역할을 한다.	machbased
Warehouse	실제 데이터를 저장하고 있는 프로세스. 전체 클러스터 데이터 중 일부를 저장하고 있으며, Broker 로부터 전달받은 명령을 수행한다.	machbased

Coordinator

Coordinator는 모든 Node의 상태를 관리하는 Process로, 최대 2개를 가질 수 있다.

먼저 생성된 Coordinator를 Primary Coordinator, 그 다음을 Secondary Coordinator라 하고 Primary Coordinator만이 모든 Node의 상태를 관리한다.

Primary Coordinator가 종료하면 Secondary Coordinator가 Primary Coordinator로 격상된다.

Special Node : Deployer

Coordinator에 의해 관리되지만, 단순히 **Broker/Warehouse/Lookup Node의 설치/제거를 담당하는 Process** 이다.
보통은 Node 를 설치할 Host 에 1대씩 추가하지만, 설치 성능을 위해 여러 Deployer를 추가할 수도 있다.

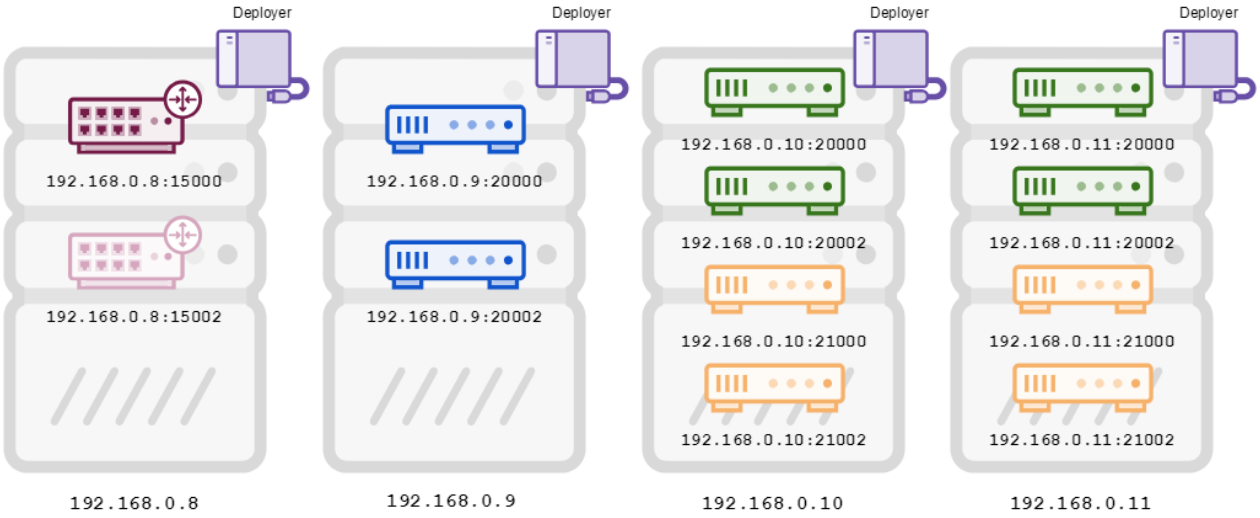


Figure (a) Deploying MachBase Nodes in Commodity Servers

⚠ 서버에서 설치 예제

아래 그림(a)는, 범용 서버 4대에 2개의 Coordinator, 2개의 Broker, 4개의 Warehouse Active, 4개의 Warehouse Standby Node를 설치한 것을 나타낸 것이다. 그림 처럼, 범용 서버의 호스트 이름과 할당된 포트 번호가 이어진 **'hostname:port'** 로 각 Node를 구분할 수 있다.

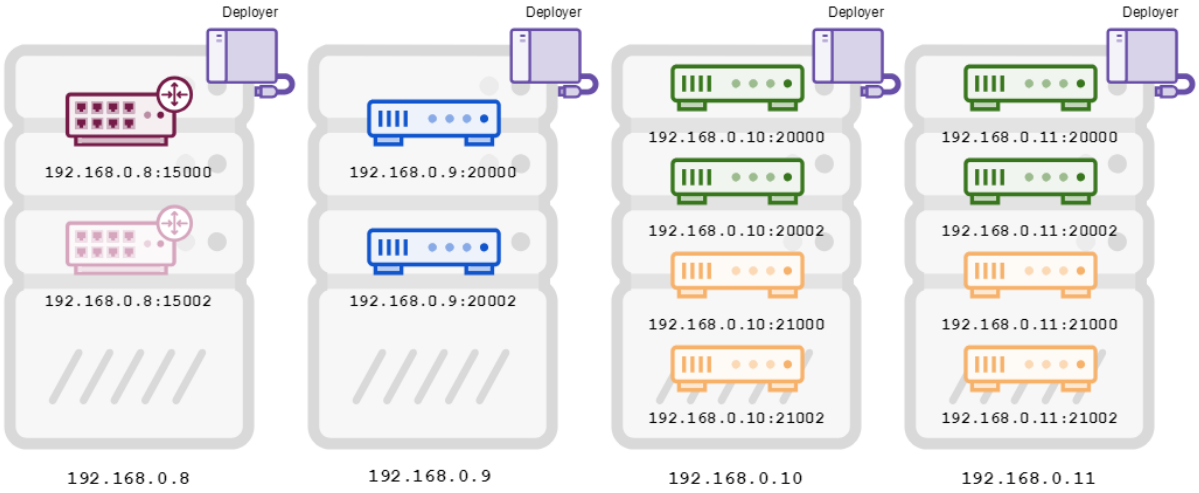


Figure (a) Deploying MachBase Nodes in Commodity Servers

Lookup

Lookup은 Lookup Table 데이터 관리를 위해 존재한다.

Broker

Broker는 말 그대로 Client의 명령을 Warehouse에게 전달하고, Warehouse의 결과를 Client에 모아서 전달하는 역할을 한다.

- 데이터 입력 시, Broker는 Warehouse에게 데이터 입력을 골고루 가도록 한다.
- 데이터 조회 시, Broker는 Warehouse에게 데이터를 가져오도록 한 뒤에 모든 결과를 모아서 전달한다.

Broker는 Log Table의 데이터를 가지고 있지 않지만, Volatile Table의 데이터는 가지고 있는다.

Warehouse

Warehouse 는 Log Table 데이터를 직접 저장하게 되고, Broker가 전달한 명령을 실제로 수행하는 역할을 한다.

Broker 처럼 Warehouse 에도 직접 클라이언트 접속이 가능하지만, 데이터 입력/갱신/삭제는 할 수 없고 오로지 해당 Warehouse 데이터 조회만 가능하다.

ⓘ Warehouse Group

Warehouse 는, 자신이 속할 Group 을 지정할 수 있다.

- Broker 가 데이터를 입력할 때, 같은 Group 에 있는 모든 Warehouse는 동일한 레코드를 입력받는다.
- Group 의 특정 Warehouse 가 탈락하더라도, 데이터 조회는 이상 없다.
- Group 에 새로운 Warehouse 가 추가되면, 이중화 (Replication) 를 통해 동일한 레코드를 유지한다.

Warehouse Group 의 상태

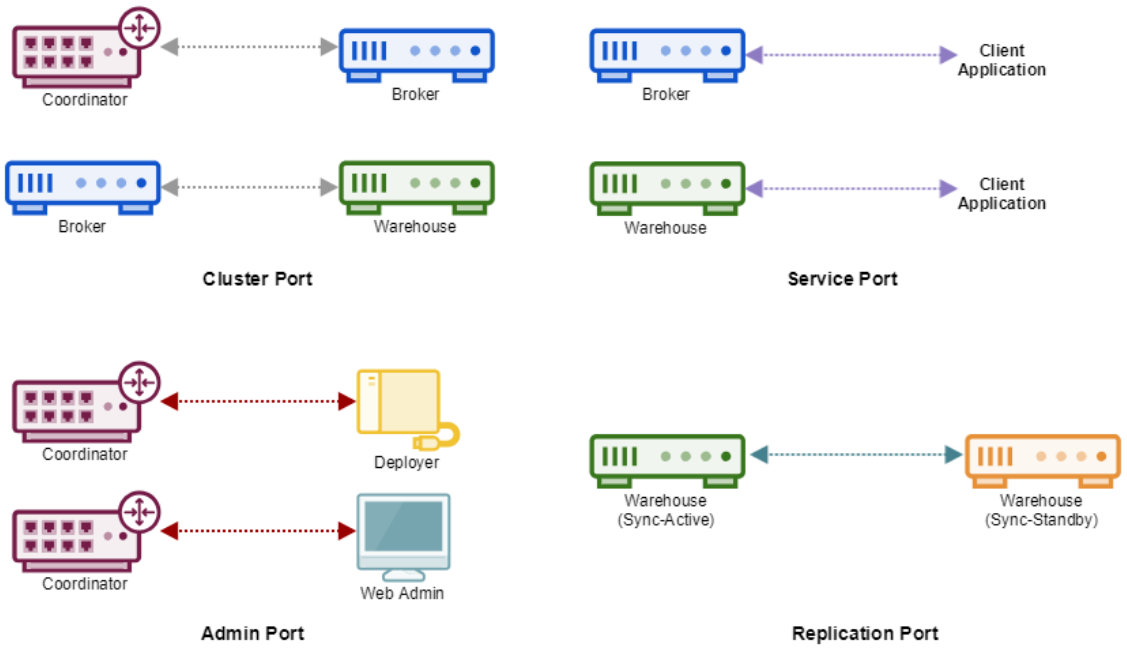
상태	INSERT / APPEND	SELECT
Normal	O	O
Readonly	X	O

Readonly 상태로 변하는 조건은 다음과 같다.

- INSERT/APPEND 도중, Group 의 일부 Warehouse 가 입력에 실패하는 경우
  - 실패한 Warehouse 와 성공한 Warehouse 간의 데이터 불일치가 발생했기 때문에,
    실패한 Warehouse 는 Scrapped 상태로 만들고 해당 Group 은 더 이상의 입력을 받지 않기 위해 Readonly 상태로 전환된다. ⚠️
- 새로운 Warehouse 가 추가된 경우
  - 이중화 과정이 진행되는 동안에도 입력을 받게 되면, 이중화 끝을 알 수 없기 때문에 Readonly 상태로 전환된다. ⚠️

Node의 Port 관리

각 Node는 Port를 여러 개 열어두고 있어야 하는데, 다음과 같이 구분된다.



Port 구분	설명	필요한 Node
Cluster Port	Node 간 통신을 위한 Port	모든 Node
Service Port	클라이언트가 직접 접속하게 되는 Port	Broker / Warehouse

Port 구분	설명	필요한 Node
Admin Port	관리 목적으로 통신하기 위한 Port	Coordinator / Deployer
Replication Port	Warehouse 간에 Replication 용 통신을 위한 Port	Warehouse

ⓘ 직접 접속 후, 수행 가능한 명령

다음은 각 Node에 직접 접속해서, 명령 수행이 가능한 것과 불가능한 것을 표로 나타낸 것이다.
모든 Node는 Client 를 통한 접속이 가능하지만, Node 종류에 따라 불가능한 쿼리가 존재한다.

	Broker (Leader)	Broker (non-leader)	Warehouse Standby
Client 접속	O	O	O
DDL	O	X	X
DELETE	O	O	X
INSERT	O	O	X [1)]
APPEND	O	O	X [1)]
SELECT	O	O	O

데이터 저장/조회

Machbase Cluster Edition 은 데이터를 분산 저장하고, 분산 쿼리 수행으로 계산되는 결과를 수집할 수 있다. 여기서는 테이블 종류에 따라 어떻게 저장되고 조회되는지 설명한다.

데이터 저장

Log Table

Broker를 통해 Log Table에 데이터를 입력하는 경우, 모든 Warehouse에 분산 저장된다. (입력을 수행하는 Broker에는 데이터가 저장되지 않는다.) Coordinator가 각 Warehouse의 데이터베이스 크기를 판단, 그 기준으로 Broker가 분산 저장한다.

Warehouse를 통해 직접 Log Table에 데이터를 입력하는 경우, 해당 Warehouse에만 저장된다. 분산 알고리즘, 네트워크 병목으로 인한 성능 저하를 피하고자 하는 경우에 선택할 수 있다.

Volatile Table

Broker를 통해 Volatile Table에 데이터를 입력하는 경우, 해당 Broker에 저장된다. 즉, 다른 Broker에는 해당 데이터가 입력되지 않고 동기화되지도 않는다.

Volatile Table에 대한 이중화를 지원하지 않는 이유는, DELETE가 가능한 Volatile Table의 특성에 맞추면 이중화 성능에 영향을 미치기 때문이다.

Volatile Table은 Broker에서만 생성되므로, Warehouse에서 입력할 수 없다.

Lookup Table

Broker를 통해 Lookup 테이블에 데이터를 입력하는 경우, 입력한 데이터는 Lookup 노드에에 저장되며, Replication을 통해 다른 Broker들에게 복제된다.

Tag Table

Log 테이블의 저장 방식과 동일하다. 단, 신규 TagID를 포함하는 데이터를 입력할 경우 Leader Broker를 통해서만 입력할 수 있다.

데이터 조회

Log Table

Broker를 통해 Log Table에 데이터를 조회하는 경우, 모든 Warehouse에 쿼리가 분산된다. 각 Warehouse는 쿼리를 실제로 수행하는데, 필요한 경우엔 Warehouse 간 중간 결과를 교환한다. 이렇게 생성된 부분 결과를 Broker가 수집, 최종 결과를 반환한다.

Warehouse를 통해 Log Table에 데이터를 조회하는 경우, 해당 Warehouse에서만 쿼리가 수행된다. 이 과정은 Fog Edition 에서의 쿼리 수행과 동일하다.

Lookup / Volatile Table

Broker를 통해 Volatile Table에 데이터를 조회하는 경우, Broker에서만 쿼리가 수행된다. 이 과정은 Fog Edition 에서의 쿼리 수행과 동일하다.

Warehouse를 통해서는 JOIN을 할 수 없는데, Volatile Table이 생성되지 않기 때문이다.

두 테이블 간 JOIN

Broker를 통해서 Log Table과 Volatile Table 간 JOIN을 하는 경우, 접속한 Broker와 나머지 Warehouse 가 동시에 쿼리를 수행한다. Broker는 Volatile Table 결과를 Warehouse에게 나눠주며, Warehouse는 Broker가 전달한 데이터를 JOIN 해서 결과를 반환한다. 이렇게 생성된 부분 결과를 Broker가 수집, 최종 결과를 반환한다.

Warehouse를 통해서는 JOIN을 할 수 없는데, Volatile Table이 생성되지 않기 때문이다.

이중화

이중화란, 기존에 설치된 Node 의 실패를 대비해 똑같은 Node 를 복제하는 과정 또는 그 상태를 의미한다.

Coordinator Replication

Cluster Edition 에서 Coordinator는 최대 2개까지 생성이 가능하다.

두 Coordinator는 지속적으로 Cluster Node 정보를 계속 유지한다.
어느 한 쪽이 비정상적으로 종료되더라도, 나머지 Coordinator가 Cluster Node 관리를 계속 할 수 있다.

> Primary Coordinator가 재시작되면 기존의 secondary coordinator가 primary로 격상되고, 재시작하는 coordinator는 secondary가 된다.

Lookup Replication

기본적으로 Lookup Master가 Lookup Table 데이터를 관리하지만, Lookup Slave를 추가하여 데이터를 복제하도록 할 수 있다.

Broker Replication

Broker는 Replication 대상이 아니다.

따라서 Broker A에 들어있는 Volatile Table의 데이터 레코드는 Broker B에 똑같이 유지되지 않는다. (not synchronized)
다만, Cluster 전체의 테이블/인덱스 스키마는 모두 동일하므로, Broker A에 Volatile Table **`VOL_TBL1`** 이 존재하면 Broker B에도 Volatile Table **`VOL_TBL1`** 이 존재한다.

Warehouse Replication

Group 에 새로운 Warehouse 가 추가되는 경우, 다음 과정을 통해서 Warehouse 가 이중화된다.

1. Coordinator 가, 새로운 Warehouse 에게 DDL 이중화를 시작한다.
2. Group 이 Readonly 상태로 전환된다.
3. Group 중 1개의 Warehouse 가, 새로운 Warehouse 에게 데이터 이중화를 시작한다.
4. 데이터 이중화가 끝나면 Group 은 Normal 상태로 전환된다.

데이터 입력의 경우, Broker 가 같은 Group 에는 같은 데이터를 전송함으로써 이중화를 보장한다.

복구 방법

Node가 비정상적으로 종료되어도, 아래와 같은 방법으로 서비스를 계속할 수 있다.

자세한 내용은 운영 가이드를 참고한다.

종류	Fail-over 방법
Coordinator	Primary Coordinator가 비정상 종료되어도, Secondary Coordinator 가 Primary Coordinator가 되면서 클러스터 관리를 계속 할 수 있다. 최악의 상황으로 Coordinator가 모두 종료되어도, 클러스터 관리를 할 수 없을 뿐 전체 서비스 (데이터 입력/조회) 는 계속 할 수 있다. (물론, 이 때 Broker나 Warehouse가 종료되면 클러스터 관리를 할 수 없다.)
Deployer	해당 Host 로 Node Operation (ADD, REMOVE..) 을 할 수 없고, 해당 Host 의 통계 정보를 수집할 수 없다.
Lookup	Lookup Master에세 장애가 발생하면 Lookup Monitor가 자동으로 감지하여 Lookup Slave 중 하나를 Lookup Master로 변경하여 계속적인 서비스 이용이 가능하게 한다. ⚠ 기존에 Lookup Slave가 존재하지 않았다면 데이터 복제가 되지 않는 상태이기에 안정적인 HA를 위해 Lookup Slave는 하나 이상 존재하는 것을 권장한다.
Broker	Broker가 종료되어도, 다른 Broker가 존재한다면 계속 서비스를 지속할 수 있다. 단, 종료된 Broker와 연결된 클라이언트의 접속은 끊어지기 때문에 다른 Broker로 재접속해야 한다.

종류	Fail-over 방법
Warehouse	Group 에 다른 Warehouse(s) 가 존재하면, 해당 Warehouse(s) 가 SELECT 와 APPEND 를 그대로 참여한다.

지원되지 않는 기능

Query Statement

TABLESPACE

현재 Cluster Edition 에서는 테이블 스페이스 구분을 하지 않는다.

BACKUP / MOUNT

현재 Cluster Edition 에서는 데이터베이스 구분을 하지 않는다.

LOAD IN FILE

CSV 파일을 읽어 분산하는 기능은 현재 구현되어 있지 않다.

ALTER TABLE FORGERY CHECK

고객의 데이터가 변경되었는지 검증하기 위한 구문인데, Result File을 한 곳에 모을 수 없다.

Clause / Function

UNION ALL

실행 단위가 복잡하게 생성되므로, 현재 지원되지 않는다.

GROUP_CONCAT() function

각 Warehouse 에서 수집한 부분 그룹에 대한 CONCAT 내용 전체를, 단순 누적으로 처리할 수 없다.
(GROUP CONCAT에서 ORDER BY를 하는 경우)

TS_CHANGE_COUNT() function

각 Warehouse 에서 수집한 부분 그룹에 대한 TS_CHANGE_COUNT 결과를 단순 누적으로 처리할 수 없다.
게다가 TS_CHANGE_COUNT() 는 전체 결과가 정렬되어 있어야 의미가 있는데, Warehouse에 분산된 결과를 대상으로 하면 의미가 없다.

지원 하드웨어 및 운영체제

CPU	Intel Core i Series (Nehalem~) 이상 권장
Memory	설치될 Node 1개 당 2GB 이상 권장
운영체제	Linux (Any distribution)

설치

패키지 개요

패키지 종류

MACHBASE 는 매뉴얼 설치, 패키지 설치 파일을 제공한다.

설치 형태	특징	비고
매뉴얼 설치	압축파일 형태를 띄고 있으며, 유닉스의 경우에는 확장자가 tgz 이다. 사용자는 tar 및 GNU gzip 을 이용하여 압축을 풀어서 설치를 진행해야 한다.	콘솔 환경에서만 설치 가능
패키지 설치	각 운영체제 환경에 맞는 설치 패키지를 제공한다. • Windows : msi • RHEL/CentOS Linux : rpm • Debian/Ubuntu : deb • Docker : (Dockerhub image)	콘솔 환경에서 설치 가능

패키지 파일명 구조

패키지 파일명은 다음과 같이 구성된다.

> *"machbase-EDITION-VERSION-OS-CPU-BIT-MODE-OPTIONAL.EXT"*

항목	설명
EDITION	패키지의 에디션을 나타낸다. edge : Edge Edition fog : Fog Edition cluster : Cluster Edition
VERSION	버전을 나타낸다. 세부적으로는 *MajorVersion.MinorVersion.FixVersion.AUX* 로 숫자 및 문자로 구분된다. Major Version : 제품 메인 버전 (숫자) Minor Version : 같은 메인 버전에서 비교적 큰 기능이 추가된 버전. DB 파일/프로토콜 호환을 보장하지 않는다. (숫자) Fix Version : 같은 메인 버전에서 버그/사소한 기능이 추가된 버전. DB 파일/프로토콜 호환을 보장한다. (숫자) AUX : 패키지의 구분을 나타낸다. (문자) • official : 일반 패키지 • community : 커뮤니티 에디션 용 패키지 • develop : 내부 개발용 패키지 (비공개)
OS	운영체제 명을 나타낸다. (예) LINUX, WINDOWS
CPU	해당 운영체제에 설치된 CPU 의 타입을 나타낸다. (예) X86, IA64
BIT	컴파일된 바이너리가 32비트 혹은 64비트 인지 나타낸다. (예) 32, 64
MODE	컴파일시 해당 바이너리의 릴리즈 모드를 나타낸다. 예) release, debug, prerelease

<table>
<tr><th>항목</th><th>설명</th></tr>
<tr><td>OPTIONAL</td><td>ⓘ Enterprise Edition 에서만 표시된다.
<table><tr><td>(공란)</td><td>-</td></tr><tr><td>lightweight</td><td>Coordinator 에 추가할 경량화 Package 를 나타낸다.</td></tr></table></td></tr>
<tr><td>EXT</td><td>패키지 파일 확장자이다. 패키지에 따라 tgz, rpm, deb, msi 로 제공된다.</td></tr>
</table>

Linux 설치

Linux 환경 설치 준비

파일 최대 개수 확인 및 설정

1. 리눅스 파일 최대 개수를 아래 명령어로 확인한다.

```
[machbase@localhost ~] ulimit -Sn
1024
```

2. 결과값이 65535보다 작다면, limit.conf 와 user.conf 를 아래와 같이 수정하고 서버를 재시작 한다.

```
[machbase@localhost ~] sudo vi /etc/security/limits.conf


#<domain>      <type>  <item>         <value>
#

*               hard    nofile          65535
*               soft    nofile          65535


[machbase@localhost ~] sudo vi /etc/systemd/user.conf

DefaultLimitNOFILE=65535
```

3. 서버를 재시작하고 다시 값을 확인한다.

```
[machbase@localhost ~] ulimit -Sn
65535
```

서버의 시간 확인 및 설정

Machbase는 시계열 데이터를 다루는 데이터베이스이므로 Machbase가 설치될 서버의 시간 값을 정확하게 설정해야 한다.

타임존 설정하기

Machbase는 모든 데이터를 해당 서버가 위치한 곳의 지역 시간을 이용하기 때문에 현재 서버의 시간과 Timezone이 맞는지 꼭 확인해야 한다.

아래 명령어로 자신이 위치한 Timezone과 맞는지 확인한다. 만약 다르다면, /usr/share/zoneinfo 에서 맞는 지역을 선택하여 링크한다.

```
[machbase@localhost ~] ls -l /etc/localtime
lrwxrwxrwx 1 root root 32 Sep 27 14:08 /etc/localtime -> ../usr/share/zoneinfo/Asia/Seoul


# date 명령어를 통해 설정된 Timezone을 확인할 수 있다.
[machbase@localhost ~] date
Wed Jan  2 11:12:44 KST 2019
```

시간 설정하기

만약 현재의 지역 시간이 맞지 않다면 다음 명령을 통해 시간을 재설정한다.

```
[machbase@localhost ~] sudo date -s '2018/12/25 12:34:56'
```

포트 설정

Machbase가 사용할 운영체제 포트는 다른 프로그램에서 사용이 안되도록 포트 예약을 사용해야 한다.

아래 명령어로 예약 포트를 설정하면 운영체제가 다른 프로그램에 해당 포트를 할당하지 않게 되어 포트 충돌 문제를 피할 수 있다.

```
[machbase@localhost ~] sudo echo 예약포트범위~예약포트범위 > /proc/sys/net/ipv4/ip_local_reserved_ports
```

위 방법은 일시적인 방법으로 영구적으로 설정하려면 /etc/sysctl.conf 파일을 수정해야 한다.

```
[machbase@localhost ~] sudo vim /etc/sysctl.conf
```

```
# 해당 내용 추가
net.ipv4.ip_local_reserved_ports = 예약포트범위-예약포트범위
```

자세한 내용은 안내 페이지를 참고하면 된다.

Tarball 설치

사용자 생성

마크베이스 설치 및 사용을 위한 리눅스 사용자 '**machbase**' 를 생성한다.

```
sudo useradd machbase
```

패스워드를 설정한 다음, machbase 계정으로 접속한다.

패키지 설치

machbase_home 이라는 디렉터리를 생성하고, 마크베이스 다운로드 사이트에서 패키지를 다운로드 받아서 설치한다.

```
[machbase@localhost ~]$ wget http://www.machbase.com/dist/machbase-fog-x.x.x.of
[machbase@localhost ~]$ mkdir machbase_home
[machbase@localhost ~]$ mv machbase-fog-x.x.x.official-LINUX-X86-64-release.tgz
[machbase@localhost ~]$ cd machbase_home/
[machbase@localhost machbase_home]$ tar zxf machbase-fog-x.x.x.official-LINUX-X

[machbase@loclahost machbase_home]$ ls -l
drwxrwxr-x  5 machbase machbase        64 Oct 30 16:10 3rd-party
drwxrwxr-x  2 machbase machbase      4096 Oct 30 16:10 bin
drwxrwxr-x  6 machbase machbase       189 Dec 21 14:04 collector
drwxrwxr-x  2 machbase machbase       306 Jan  2 11:36 conf
drwxrwxr-x  2 machbase machbase       136 Jan  2 11:37 dbs
drwxrwxr-x  3 machbase machbase        22 Oct 30 16:10 doc
drwxrwxr-x  2 machbase machbase        96 Oct 30 16:10 include
drwxrwxr-x  2 machbase machbase        29 Oct 30 16:10 install
drwxrwxr-x  2 machbase machbase       283 Oct 30 16:10 lib
-rw-rw-r--  1 machbase machbase 139888377 Dec 20 11:33 machbase-fog-x.x.x.offic
drwxrwxr-x  2 machbase machbase        22 Dec 21 15:43 msg

drwxrwxr-x  2 machbase machbase         6 Oct 30 16:10 package
drwxrwxr-x 12 machbase machbase       140 Oct 30 16:10 sample
drwxrwxr-x  2 machbase machbase      4096 Jan  2 09:37 trc
drwxrwxr-x 10 machbase machbase       160 Oct 30 16:10 tutorials
drwxrwxr-x  3 machbase machbase        19 Oct 30 16:10 webadmin

[machbase@loclahost machbase_home]$
```

설치된 디렉터리 설명은 다음과 같다.

디렉터리	설명
bin	실행 파일들
collector	로그 수집기 파일들
conf	설정 파일들
dbs	데이터 저장 공간
doc	라이선스 파일들
include	CLI 프로그램을 위한 각종 헤더 파일들
install	Makefile을 위한 mk 파일
lib	각종 라이브러리
msg	Machbase 서버 에러 메시지
package	(Cluster edition) 추가된 패키지를 저장할 경로

디렉터리	설명
sample	각종 예제 파일들
trc	Machbase 서버 로그 및 추적 내용들
webadmin	MWA 웹서버 파일들
3rd-party	grafana 연동 파일들

환경변수 설정

.bashrc 파일에 마크베이스 관련 환경 변수를 추가한다.

```
export MACHBASE_HOME=/home/machbase/machbase_home
export PATH=$MACHBASE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$MACHBASE_HOME/lib:$LD_LIBRARY_PATH

# 변경사항을 아래 명령어로 적용한다.
source .bashrc
```

마크베이스 프로퍼티 설정

$MACHBASE_HOME/conf 디렉터리에 machbase.conf.sample 파일이 있다.

```
[machbase@localhost ~]$ cd $MACHBASE_HOME/conf
[machbase@localhost conf]$ ls -l
-rw-rw-r-- 1 machbase machbase   106 Oct 30 16:10 machtag.sql.sample
-rw-rw-r-- 1 machbase machbase 17556 Oct 30 16:10 machbase.conf.sample
-rw-rw-r-- 1 machbase machbase  1706 Oct 30 16:10 machcollector.conf.sample

[machbase@localhost conf]$
```

또한 리눅스 환경 변수를 이용하여 마크베이스 접속 포트번호를 변경할 수도 있다. 아래는 디폴트값(5656)이 아닌 다른 포트번호(7878)로 변경하는 예이다.

```
export MACHBASE_PORT_NO=7878
```

마크베이스 간단 사용

데이터베이스 생성

데이터베이스 생성은 **machadmin** 유틸리티를 이용한다. --help 옵션으로 명령어를 볼 수 있다.

```
[machbase@localhost machbase_home]$ machadmin --help
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
<< available option lists >>
  -u, --startup                       Startup Machbase server.
      --recovery[=simple,complex,reset] Recovery mode. (default: simple)
  -s, --shutdown                      Shutdown Machbase server.
  -c, --createdb                      Create Machbase database.
  -d, --destroydb                     Destroy Machbase database.
  -k, --kill                          Terminate Machbase server.
  -i, --silence                       Produce less output.
  -r, --restore                       Restore Machbase database.
```

```
  -x, --extract                         Extract BackupFile to BackupDirectory.
  -w, --viewimage                       Display information of BackupImageFile.
  -e, --check                           Check whether Machbase Server is running.
  -t, --licinstall                      Install the license file.
  -f, --licinfo                         Display information of installed license file.

[machbase@localhost machbase_home]$
```

-c 옵션으로 데이터베이스를 생성한다.

```
[machbase@localhost machbase_home]$ machadmin -c
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Database created successfully.
[machbase@localhost machbase_home]$
```

마크베이스 서버 실행

-u 옵션으로 마크베이스 서버를 실행한다.

```
[machbase@localhost machbase_home]$ machadmin -u
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.
[machbase@localhost machbase_home]$
```

ps 명령을 통해 아래와 같이 서버 데몬인 machbased 가 구동된 것을 확인할 수 있다.

```
[machbase@localhost machbase_home]$  ps -ef |grep machbased
machbase 11178     1  2 11:25 ?        00:00:01 /home/machbase/machbase_home/bin/machbased -s --recovery=simple
machbase 11276  9867  0 11:26 pts/1    00:00:00 grep --color=auto machbased
[machbase@localhost machbase_home]$
```

마크베이스 서버 접속

machsql 이라는 접속 유틸리티를 이용하여 마크베이스 서버에 접속한다.

관리자 계정인 **SYS** 가 준비되어 있으며, 패스워드는 **MANAGER** 로 설정되어 있다.

```
[machbase@localhost machbase_home]$  machsql
==================================================================
     Machbase Client Query Utility
     Release Version x.x.x.official
     Copyright 2014 MACHBASE Corporation or its subsidiaries.
     All Rights Reserved.
==================================================================
Machbase server address (Default:127.0.0.1) :
Machbase user ID  (Default:SYS)
Machbase User Password :
MACHBASE_CONNECT_MODE=INET, PORT=5656
Type 'help' to display a list of available commands.
Mach>
```

간단한 테이블 생성 및 데이터를 입력, 출력 해보자.

```
create table hello( id integer );
insert into hello values( 1 );
insert into hello values( 2 );
select * from hello;
select _arrival_time, * from hello;
```

```
Mach> create table hello( id integer );
Created successfully.
Elapsed time: 0.054
Mach> insert into hello values( 1 );
1 row(s) inserted.
Elapsed time: 0.000
Mach> insert into hello values( 2 );
1 row(s) inserted.
Elapsed time: 0.000
Mach> select * from hello;
ID
--------------
2
1
[2] row(s) selected.
Elapsed time: 0.000
Mach> select _arrival_time, * from hello;
_arrival_time                   ID
-----------------------------------------------
2019-01-02 11:33:00 122:806:804 2
2019-01-02 11:32:57 383:848:361 1
[2] row(s) selected.
Elapsed time: 0.000
Mach>
```

위의 SELECT 결과를 보면 최근에 입력된 데이터가 가장 먼저 표시되는 것을 확인할 수 있다.
또한 _arrival_time 칼럼을 통해 해당 레코드가 입력된 시간이 나노초 단위까지 설정된 것을 알 수 있다.

마크베이스 서버 중단

-s 옵션으로 마크베이스 서버를 종료한다.

```
[machbase@localhost machbase_home]$ machadmin -s
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server shut down...
Machbase server shut down successfully.
[machbase@localhost machbase_home]$
```

데이터베이스 삭제

-d 옵션으로 데이터베이스를 삭제한다.

> 모든 데이터가 삭제되므로 매우 주의해서 사용해야 한다.

```
[machbase@localhost machbase_home]$ machadmin -d
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
```

```
     All Rights Reserved
----------------------------------------------------------------
Destroy Machbase database. Are you sure?(y/N) y
Database destoryed successfully.
[machbase@localhost machbase_home]$
```

RPM 설치

마크베이스 설치하기

Redhat/CentOS 계열의 리눅스는 rpm 파일을 통해서 Machbase를 설치할 수 있다.

먼저 최신의 RPM 패키지를 다운로드 받는다. 아래 명령으로 다운로드 할 수 있다.

```
$ wget http://www.machbase.com/dist/machbase-fog-x.x.x.official-LINUX-X86-64-re
```

해당 파일을 다운로드 받은 후에 설치하는 명령어는 다음과 같다.

```
$ sudo yum install machbase-fog-x.x.x.official-LINUX-X86-64-release.rpm

Loaded plugins: fastestmirror, refresh-packagekit, security
Setting up Install Process
Examining machbase-fog-x.x.x.official-LINUX-X86-64-release.rpm: machbase-x.x.x-
Marking machbase-fog-x.x.x.official-LINUX-X86-64-release.rpm to be installed
Loading mirror speeds from cached hostfile
 * base: data.aonenetworks.kr
 * extras: data.aonenetworks.kr
 * updates: data.aonenetworks.kr
Resolving Dependencies
--> Running transaction check
---> Package machbase.x86_64 0:x.x.x-official will be installed
--> Finished Dependency Resolution

Dependencies Resolved
================================================================================================================================
 Package         Arch          Version                 Repository                                                        S
================================================================================================================================
Installing:
 machbase        x86_64        x.x.x-official         /machbase-fog-x.x.x.official-LINUX-X86-64-release        632

Transaction Summary
================================================================================================================================
Install       1 Package(s)

Total size: 632 M
Installed size: 632 M
Is this ok [y/N]: y
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Warning: RPMDB altered outside of yum.
  Installing : machbase-x.x.x-official.x86_64                                                                    1

Create database
----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
----------------------------------------------------------------
Database created successfully.

Ulimit check
65535 PASS

Machbase startup
----------------------------------------------------------------
```

```
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
-----------------------------------------------------------------
-----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
-----------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.

MWA startup
SERVER HAS BEEN RESET
SERVER STARTED, PID : 7757
     Connection URL : http://192.168.0.55:5001
Machbase has been installed in : /opt/machbase/
To start Machbase, run the command : service machbased start
To change server port, run the command : service machbased port
To use interactive SQL, execute : machsql
Documentation is available at http://www.machbase.com/document
  Verifying  : machbase-x.x.x-official.x86_64                                     1

Installed:
  machbase.x86_64 0:x.x.x-official

Complete!
```

설치가 완료되면 /opt/machbase 폴더가 생성이 되고 기본 포트는 5656 으로 설정된다. **이후 데이터베이스가 생성되고 마크베이스 서버와 MWA 웹서버가 자동으로 실행된다.**

마크베이스 디렉터리 내부에는 최신 버전으로 심볼릭 링크가 되어 있는 current 라는 디렉터리가 있고, versions 디렉터리에는 마크베이스 버전 별로 파일들이 위치해 있다.

```
[root@localhost ~]$ cd /opt/machbase
[root@localhost machbase]# ls -l
total 4
lrwxrwxrwx. 1 root     root       28 Jan  2 14:12 current -> /opt/machbase/versions/x.x.x
drwxr-xr-x. 3 machbase machbase 4096 Jan  2 14:12 versions
[root@localhost machbase]$
```

/etc/init.d 디렉터리내에 machbased 라는 셀 스크립트가 설치가 되고, 이를 이용하여 마크베이스를 관리할 수 있다.

```
[root@localhost ~]$ cd /etc/init.d
[root@localhost init.d]# ls -l machbased
-rwxr-xr-x. 1 root root 8446 Oct 30 16:11 machbased
[root@localhost machbase]$
```

마크베이스 삭제하기

마크베이스 삭제는 아래 명령어로 수행하면 된다.

```
[root@localhost ~]$ sudo yum remove machbase
```

마크베이스 사용하기

마크베이스를 rpm 으로 설치하면 **/etc/init.d/machbased** 스크립트 파일이 설치되고 이 파일을 이용하여 마크베이스를 편리하게 관리할 수 있다.

지원하는 기본 기능들은 아래 명령어로 확인하면 된다.

```
[root@localhost init.d]$ service machbased
Usage: /etc/init.d/machbased {start|stop|kill|restart|createdb|destorydb|check|MWA|console|port|exe|collector|help]
```

서버 시작

마크베이스 서버를 시작한다. machadmin -u 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased start
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.
[root@localhost ~]$
```

서버 종료

마크베이스 서버를 정상 종료한다. machadmin -s 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased stop
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server shut down...
Machbase server shut down successfully.
[root@localhost ~]$
```

서버 중단

마크베이스 서버를 강제 종료한다. machadmin -k 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased kill
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server terminated.
Machbase server terminated successfully.
[root@localhost ~]$
```

서버 재시작

마크베이스 서버를 정상 종료하고 재실행한다.

```
[root@localhost ~]$ sudo service machbased restart
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server shut down...
Machbase server shut down successfully.
```

```
----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
----------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.
[root@localhost ~]$
```

데이터베이스 생성

마크베이스 데이터베이스를 생성한다. machadmin -c 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased createdb
----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
----------------------------------------------------------------
Database created successfully.
[root@localhost ~]$
```

데이터베이스 삭제

마크베이스 데이터베이스를 삭제한다. machadmin -d 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased destroydb
----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
----------------------------------------------------------------
Destroy Machbase database. Are you sure?(y/N) y
Database destoryed successfully.
[root@localhost ~]$
```

서버 상태 체크

마크베이스 서버 구동 상태를 확인한다. machadmin -e 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased check
----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
----------------------------------------------------------------
Machbase server is running with PID(23542).
[root@localhost ~]$
```

MWA 관리

마크베이스 MWA (Machbase Web Analytics) 웹서버와 관련된 명령어이다.

```
[root@localhost ~]$ sudo service machbased  MWA
start | restart | stop | reset | port

# MWA 서버를 실행한다.
```

```
[root@localhost ~]$ sudo service machbased MWA start
SERVER ALREADY STARTED
Connection URL : http://192.168.0.10:5001

# MWA 서버를 종료한다.
[root@localhost ~]$ sudo service machbased MWA stop
SERVER STOPPED

# MWA 서버 포트를 변경한다.
[root@localhost ~]$ sudo service machbased MWA port 5050
WEBSERVER PORT CHANGED : 5050

# MWA 서버를 종료하고 재실행한다.
[root@localhost ~]$ sudo service machbased MWA restart
SERVER IS RESTARTING
SERVER STOPPED
SERVER STARTED, PID : 23810
     Connection URL : http://192.168.0.10:5001
[root@localhost ~]$
```

서버 포트 변경

마크베이스 서버의 포트를 변경한다. 해당 명령어를 실행시 변경할 포트를 입력하면 해당 포트로 변경된다. 포트 변경 이후에는 마크베이스 서버를 재시작하여야 적용된다.

```
[root@localhost ~]$ sudo service machbased port
The default port for the Machbase server is 5656. If you want to use 5656 as Machbase server port, press return key
Use current port.
[root@localhost ~]$
```

Collector 관리

Machcollector를 관리하기 위한 명령어들이다.

```
[root@localhost ~]$ sudo service machbased collector
List of commands:
  * machbased collector start
    Machcollectormanager starts.
  * machbased collector stop
    Machcollectormanager shutdown.
  * machbased collector kill
    Terminate Machcollectormanager.
  * machbased collector destroy
    Destroy Machcollectormanager meta data.
  * machbased collector add_server
    Add an Machbase server to Machcollectormanager.
  * machbased collector port
    Change the default port. Now: 9999
[root@localhost ~]$
```

DEB 설치

마크베이스 설치하기

마크베이스 설치 패키지를 다운로드 사이트에서 받아서 설치한다.

```
root@ubuntu:/usr/local/src# wget http://www.machbase.com/dist/machbase-fog-x.x.
root@ubuntu:/usr/local/src# sudo dpkg -i machbase-fog-x.x.x.community-LINUX-X86
Selecting previously unselected package machbase.
(Reading database ... 464623 files and directories currently installed.)
Preparing to unpack machbase-fog-x.x.x.community-LINUX-X86-64-release.deb ...
Group machbase exist
Unpacking machbase (5.1.9) ...
Setting up machbase (5.1.9) ...

Create database
-----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
-----------------------------------------------------------------
Database created successfully.

Ulimit check
65535 PASS

Machbase startup
-----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
-----------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.

MWA startup
SERVER HAS BEEN RESET
SERVER STARTED, PID : 1957
     Connection URL : http://192.168.0.183:5001
Machbase has been installed in : /opt/machbase/
To start Machbase, run the command : service machbased start
To change server port, run the command : service machbased port
To use interactive SQL, execute : machsql
Documentation is available at http://www.machbase.com/document
Processing triggers for systemd (229-4ubuntu21.1) ...
Processing triggers for ureadahead (0.100.0-19) ...
root@ubuntu:/usr/local/src#
```

설치과정에서 의존성 문제가 발생하면, 다음 명령어로 설치한다.

```
root@ubuntu:/usr/local/src# sudo apt-get install -f
root@ubuntu:/usr/local/src# sudo dpkg -i machbase-fog-x.x.x.community-LINUX-X86-64-release.deb
```

설치가 완료되면 /opt/machbase 폴더가 생성이 되고 기본 포트는 5656 으로 설정된다. **이후 데이터베이스가 생성되고 마크베이스 서버와 MWA 웹서버가 자동으로 실행된다.**

마크베이스 디렉터리 내부에는 최신 버전으로 심볼릭 링크가 되어 있는 current 라는 디렉터리가 있고, versions 디렉터리에는 마크베이스 버전 별로 파일들이 위치해 있다.

```
root@ubuntu:/usr/local/src# cd /opt/machbase
root@ubuntu:/opt/machbase# ls -l
total 4
```

```
lrwxrwxrwx 1 root     root       28 Jan  3 00:25 current -> /opt/machbase/versions/5.1.9
drwxrwxr-x 3 machbase machbase 4096 Jan  3 00:25 versions
root@ubuntu:/opt/machbase#
```

마크베이스 삭제하기

마크베이스 삭제는 아래 명령어로 수행하면 된다.

```
root@ubuntu:/opt/machbase# sudo dpkg -r machbase
```

마크베이스 사용하기

마크베이스를 deb 로 설치하면 **/etc/init.d/machbased** 스크립트 파일이 설치되고 이 파일을 이용하여 마크베이스를 편리하게 관리할 수 있다.

지원하는 기본 기능들은 아래 명령어로 확인한다.

```
root@ubuntu:/opt/machbase# cd /etc/init.d
root@ubuntu:/etc/init.d# sudo service machbased
Usage: /etc/init.d/machbased {start|stop|kill|restart|createdb|destroydb|check|MWA|console|port|exe|collector|help}
root@ubuntu:/etc/init.d#
```

서버 시작

마크베이스 서버를 시작한다. machadmin -u 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased start
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.
[root@localhost ~]$
```

서버 종료

마크베이스 서버를 정상 종료한다. machadmin -s 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased stop
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server shut down...
Machbase server shut down successfully.
[root@localhost ~]$
```

서버 중단

마크베이스 서버를 강제 종료한다. machadmin -k 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased kill
------------------------------------------------------------------
     Machbase Administration Tool
```

```
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------
Waiting for Machbase server terminated.
Machbase server terminated successfully.
[root@localhost ~]$
```

서버 재시작

마크베이스 서버를 정상 종료하고 재실행한다.

```
[root@localhost ~]$ sudo service machbased restart
--------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------
Waiting for Machbase server shut down...
Machbase server shut down successfully.
--------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.
[root@localhost ~]$
```

데이터베이스 생성

마크베이스 데이터베이스를 생성한다. machadmin -c 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased createdb
--------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------
Database created successfully.
[root@localhost ~]$
```

데이터베이스 삭제

마크베이스 데이터베이스를 삭제한다. machadmin -d 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased destroydb
--------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------
Destroy Machbase database. Are you sure?(y/N) y
Database destoryed successfully.
[root@localhost ~]$
```

서버 상태 체크

마크베이스 서버 구동 상태를 확인한다. machadmin -e 와 같은 기능이다.

```
[root@localhost ~]$ sudo service machbased check
----------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.official
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
----------------------------------------------------------------
Machbase server is running with PID(23542).
[root@localhost ~]$
```

MWA 관리

마크베이스 MWA (Machbase Web Analytics) 웹서버와 관련된 명령어이다.

```
[root@localhost ~]$ sudo service machbased  MWA
start | restart | stop | reset | port

# MWA 서버를 실행한다.
[root@localhost ~]$ sudo service machbased MWA start
SERVER ALREADY STARTED
Connection URL : http://192.168.0.10:5001

# MWA 서버를 종료한다.
[root@localhost ~]$ sudo service machbased MWA stop
SERVER STOPPED

# MWA 서버 포트를 변경한다.
[root@localhost ~]$ sudo service machbased MWA port 5050
WEBSERVER PORT CHANGED : 5050

# MWA 서버를 종료하고 재실행한다.
[root@localhost ~]$ sudo service machbased MWA restart
SERVER IS RESTARTING
SERVER STOPPED
SERVER STARTED, PID : 23810
     Connection URL : http://192.168.0.10:5001
[root@localhost ~]$
```

서버 포트 변경

마크베이스 서버의 포트를 변경한다. 해당 명령어를 실행시 변경할 포트를 입력하면 해당 포트로 변경된다. 포트 변경 이후에는 마크베이스 서버를 재시작하여야 적용된다.

```
[root@localhost ~]$ sudo service machbased port
The default port for the Machbase server is 5656. If you want to use 5656 as Machbase server port, press return key
Use current port.
[root@localhost ~]$
```

Collector 관리

Machcollector를 관리하기 위한 명령어들이다.

```
[root@localhost ~]$ sudo service machbased collector
List of commands:
  * machbased collector start
    Machcollectormanager starts.
  * machbased collector stop
    Machcollectormanager shutdown.
  * machbased collector kill
```

```
    Terminate Machcollectormanager.
  * machbased collector destroy
    Destroy Machcollectormanager meta data.
  * machbased collector add_server
    Add an Machbase server to Machcollectormanager.
  * machbased collector port
    Change the default port. Now: 9999
[root@localhost ~]$
```

DOCKER 설치

마크베이스는 Docker 이미지를 제공한다. Docker가 이미 설치되어 있다고 가정하고 마크베이스를 Docker로 설치하는 과정을 설명한다.

> Docker 설치는 **Docker 설치 페이지**를 참조하여 진행하면 된다. 마크베이스의 Docker Hub는 **이 페이지**를 참고한다.

```
$ docker pull machbase/machbase
Using default tag: latest
latest: Pulling from machbase/machbase
3a291d7fe8d1: Pull complete
f1e7bd0ef2d1: Pull complete
78632f9cbb53: Pull complete
f4f6c5358244: Pull complete
a3e04b27f9cd: Pull complete
a3ed95caeb02: Pull complete
e03e135c0eda: Pull complete
26612cd7ebc1: Pull complete
b61e71cf4bc2: Pull complete
09c9c411b936: Pull complete
2b1cdec8c664: Pull complete
fd9a9a288691: Pull complete
d8852dedc8a1: Pull complete
cba7e30dbb6f: Pull complete
c7ead0fa7c49: Pull complete
6af02fe4c01f: Pull complete
d18db958464f: Pull complete
1fb93627ec0f: Pull complete
265b8b73294a: Pull complete
f122e6396b46: Pull complete
3b2f248fb414: Pull complete
07ed5a8f0935: Pull complete
44ec57c5ed31: Pull complete
59383e5f4c61: Pull complete
542101ec7002: Pull complete
Digest: sha256:aa6a982d35946b3fb33930de91cad61bfe7d3e9a559080526ed8e9a511c82c2b
Status: Downloaded newer image for machbase/machbase:latest
```

```
# 설치된 마크베이스 이미지를 확인한다.
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
machbase/machbase   latest              dfb90844e7da        2 months ago        1.09 GB
```

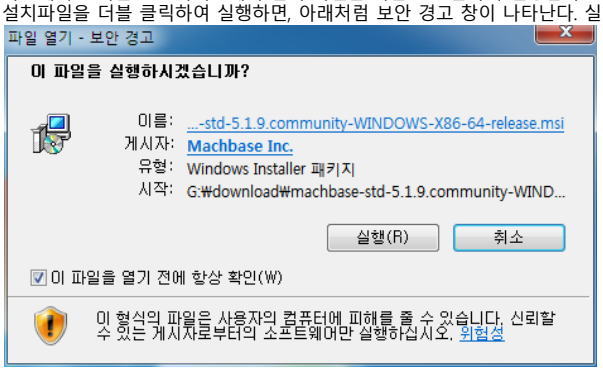
```
# 마크베이스 이미지를 실행한다.
$ docker run -it machbase/machbase
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Database created successfully.
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - x.x.x.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase server start.
Machbase server started successfully.
SERVER HAS BEEN RESET
SERVER STARTED, PID : 56
     Connection URL : http://172.17.0.2:5001
```

```
machbase@5ba45a22d140:~$
```

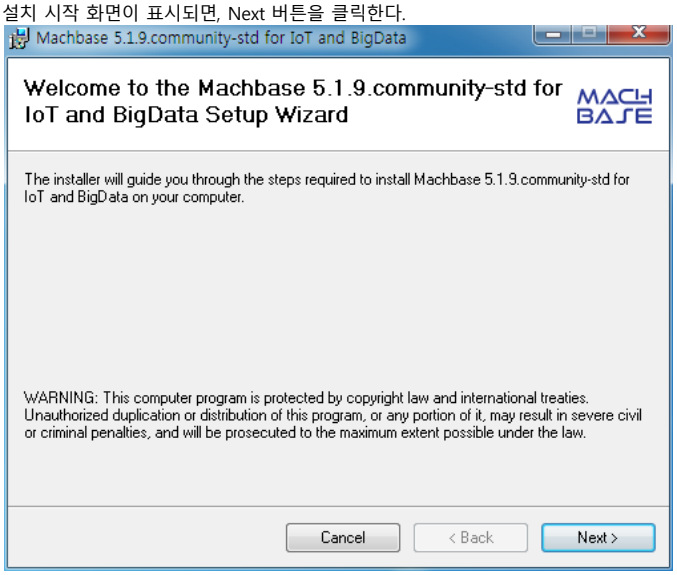
Windows 설치

MSI 설치

1. 마크베이스 다운로드 사이트에서 설치 파일을 다운로드 받아서 실행한다.
   설치파일을 더블 클릭하여 실행하면, 아래처럼 보안 경고 창이 나타난다. 실행 버튼을 클릭한다.

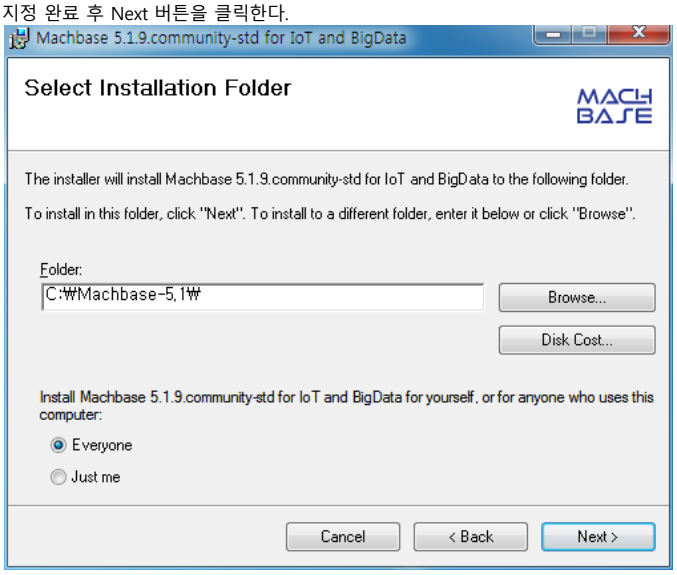


2. 설치 시작 화면이 표시되면, Next 버튼을 클릭한다.

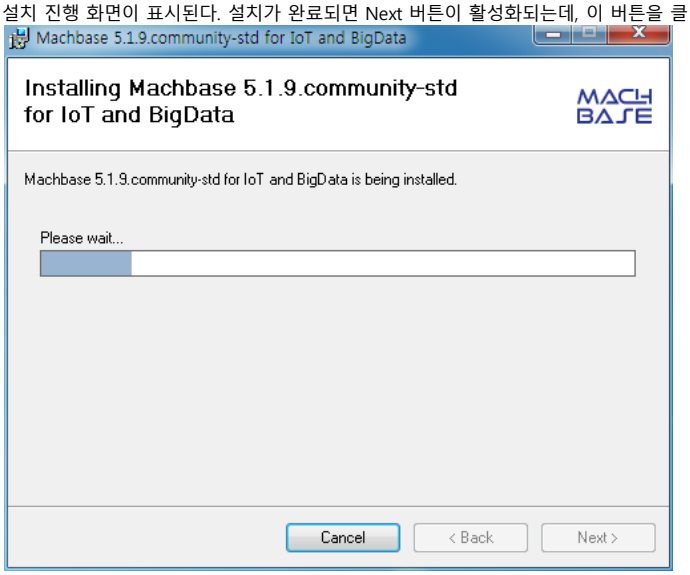


3. 설치할 디렉터리를 선택하는 화면이 표시되는데, 기본적으로 `"C:\Machbase-5.1\"` 디렉터리에 설치가 된다.
   다른 디렉터리에 설치하려면, 해당 경로 변경하여 설치한다.

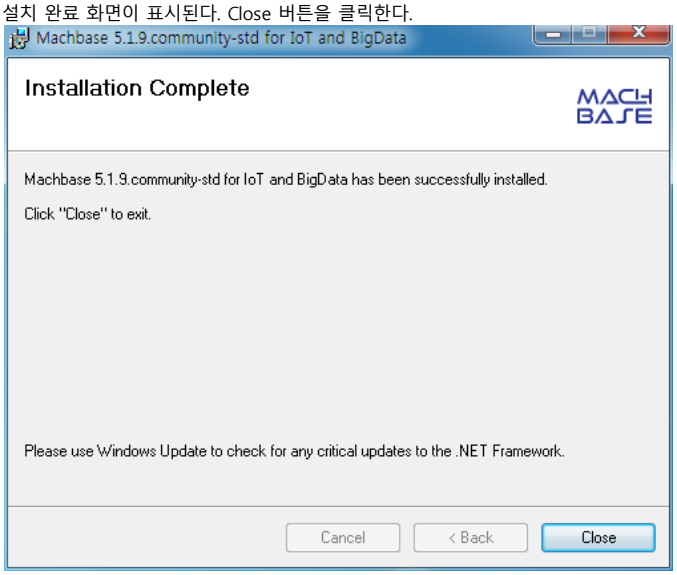
   지정 완료 후 Next 버튼을 클릭한다.



4. 설치 진행 화면이 표시된다. 설치가 완료되면 Next 버튼이 활성화되는데, 이 버튼을 클릭한다.



5. 설치 완료 화면이 표시된다. Close 버튼을 클릭한다.

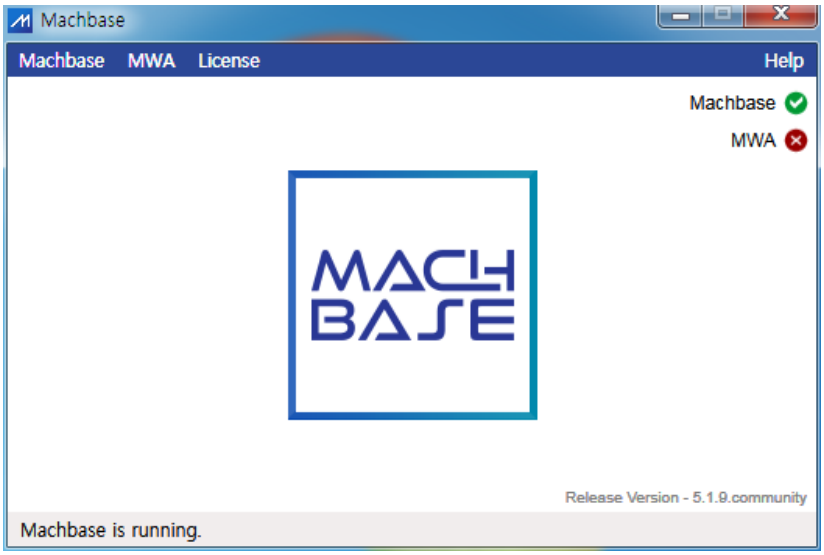


실행하기

1. 마크베이스 설치가 완료되면, 바탕화면에 마크베이스 실행 아이콘이 표시된다.

   더블 클릭하면 마크베이스 서버가 실행된다.

2. 마크베이스 서버를 관리하는 윈도우 인터페이스 화면이다.
   메뉴를 클릭해서 마크베이스 서버와 MWA 웹서버를 제어할 수 있다.


Machbase
Machbase MWA License Help
Machbase
MWA
MACH
BASE
Release Version - 5.1.9.community
Machbase is running.

Windows 환경 설치 준비

방화벽 포트 개방

마크베이스를 윈도우에 설치하는 경우 윈도우 방화벽에서 마크베이스가 사용하는 포트를 열어주어야 한다.

기본적으로 마크베이스는 **5656, 5001** 2개의 포트를 사용한다.

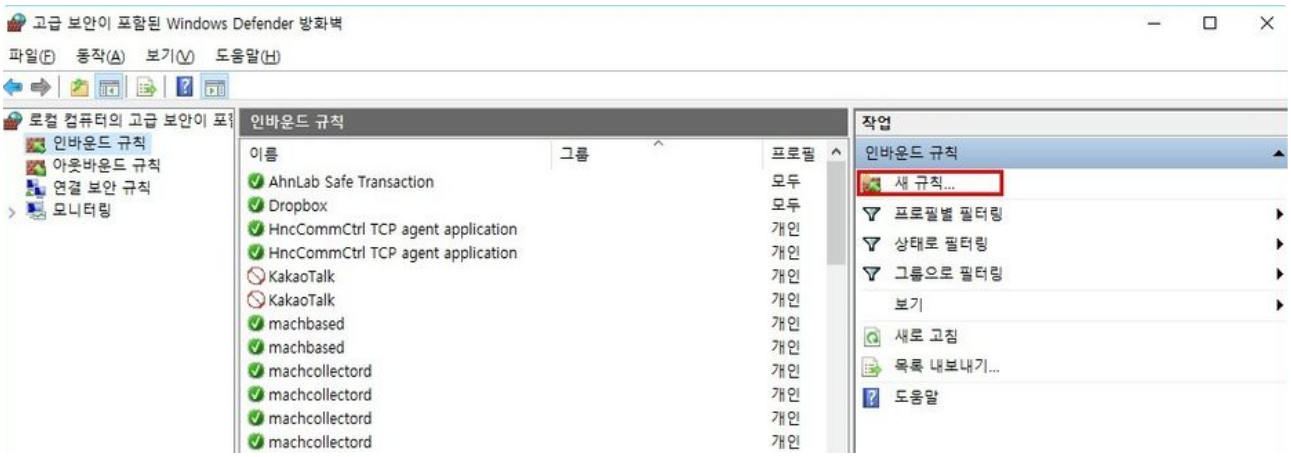
1. 방화벽에 해당 포트를 등록하기 위해서는 **제어판 – Windows 방화벽** 또는 **Windows Defender 방화벽** 을 선택하여 실행한다. 실행화면에서 "고급 설정" 메뉴를 클릭한다.

Windows 7

Windows 10

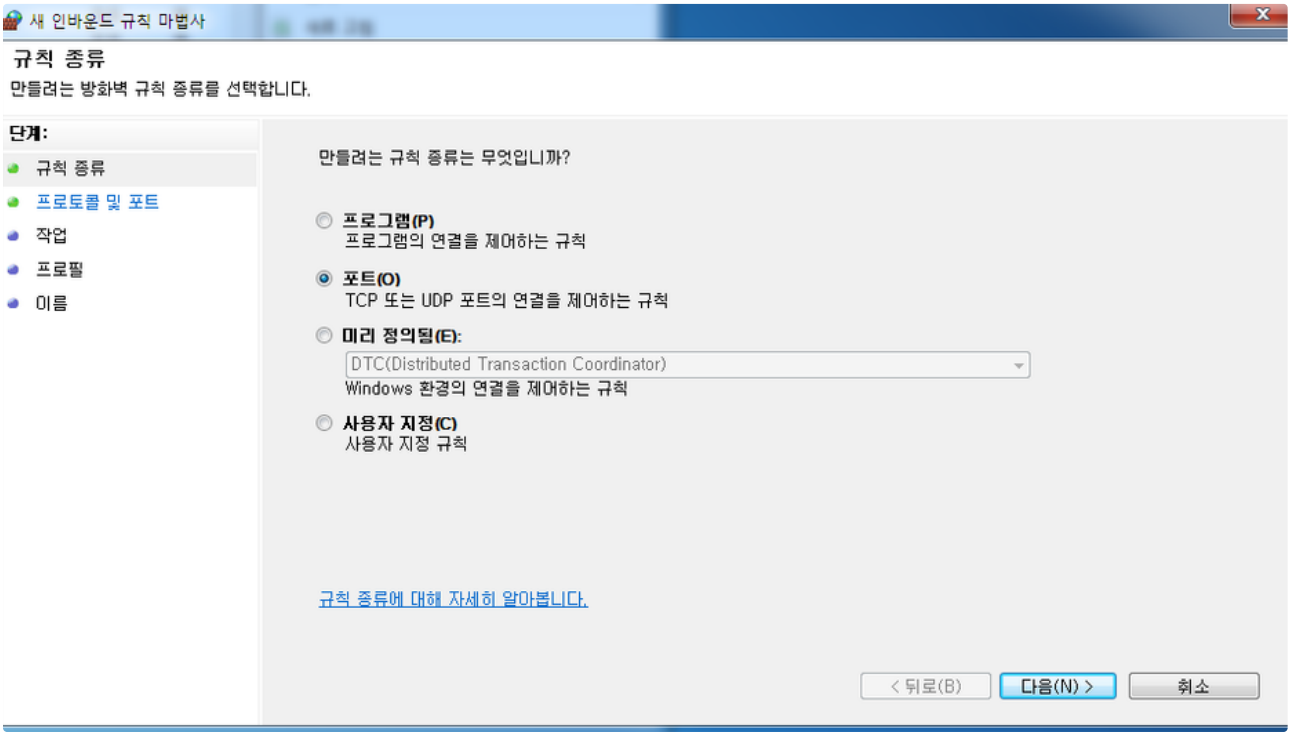
2. 고급설정에서 **인바운드 규칙 - 새 규칙** 을 선택하여 클릭한다.

Windows 7

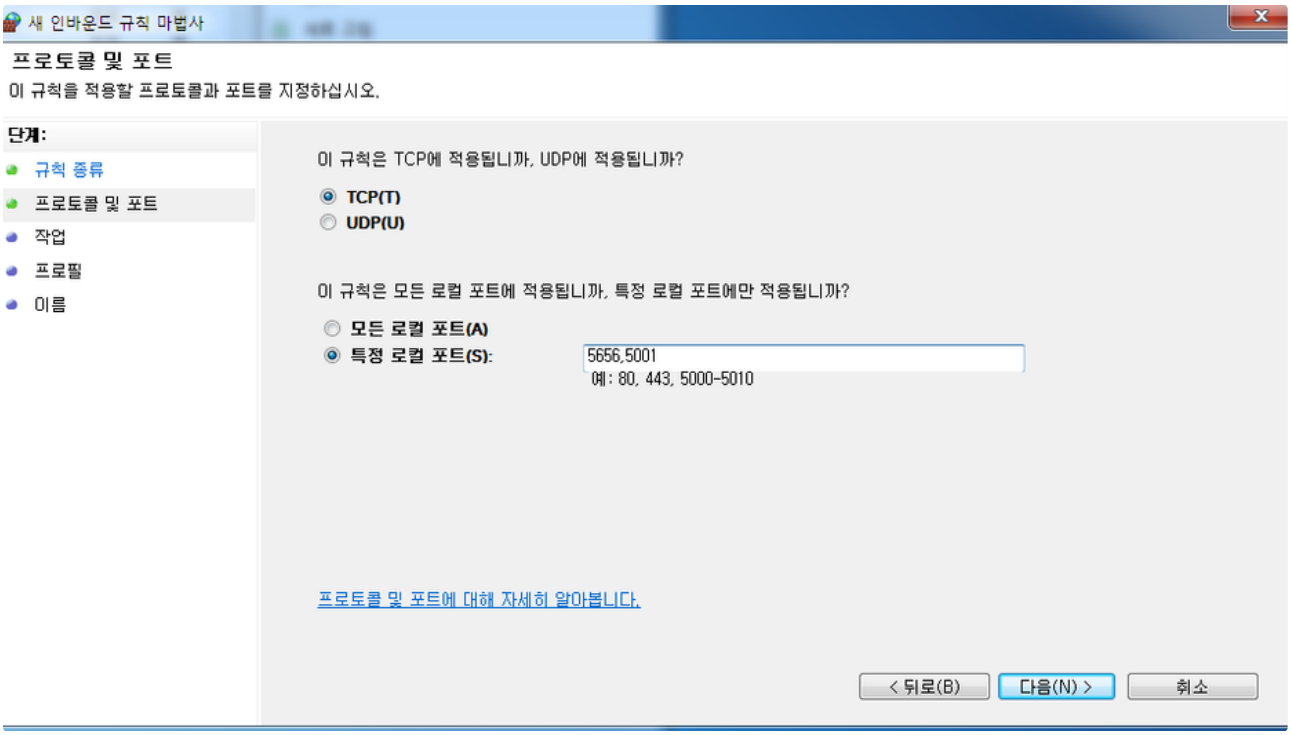


Windows 10

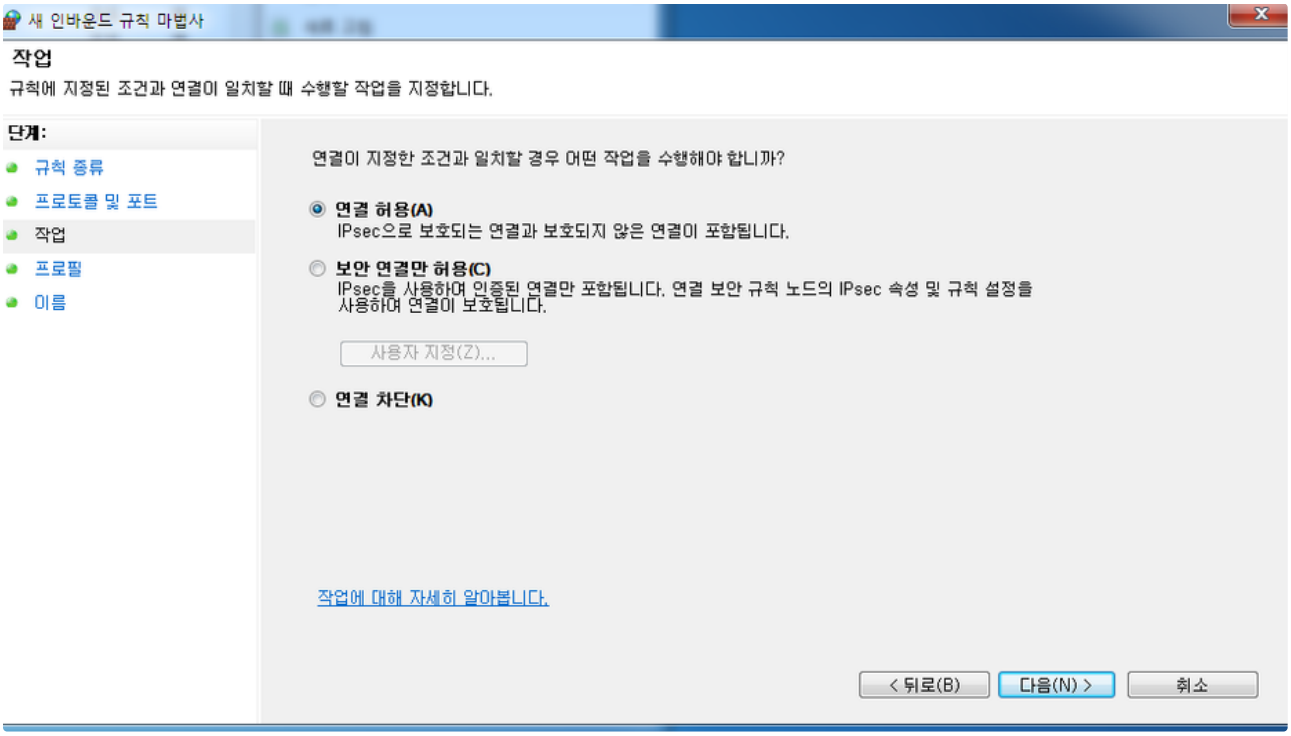
3. 새 규칙 설정 마법사 창이 표시되면 아래와 같이 포트 옵션을 선택하고 **다음**을 클릭한다.



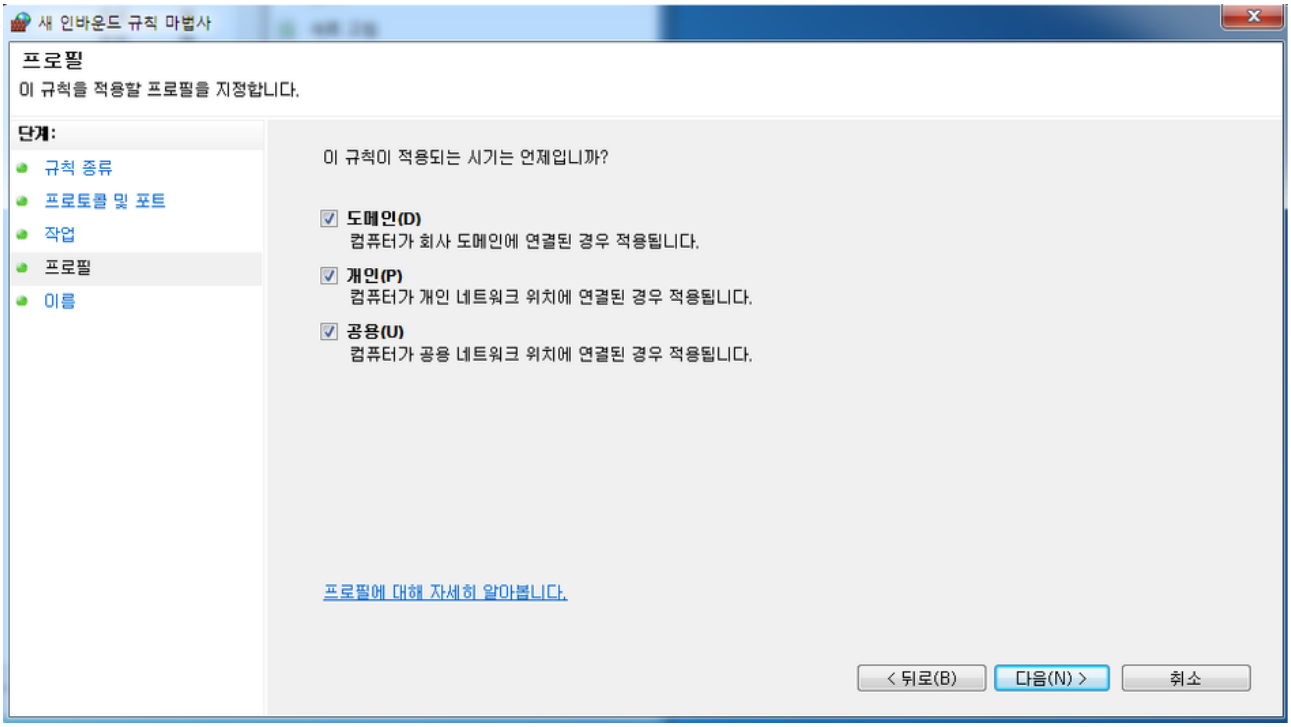
4. TCP(T) 옵션을 선택하고 **특정 로컬 포트** 입력란에 5656,5001 을 입력한 후 **다음**을 클릭한다.



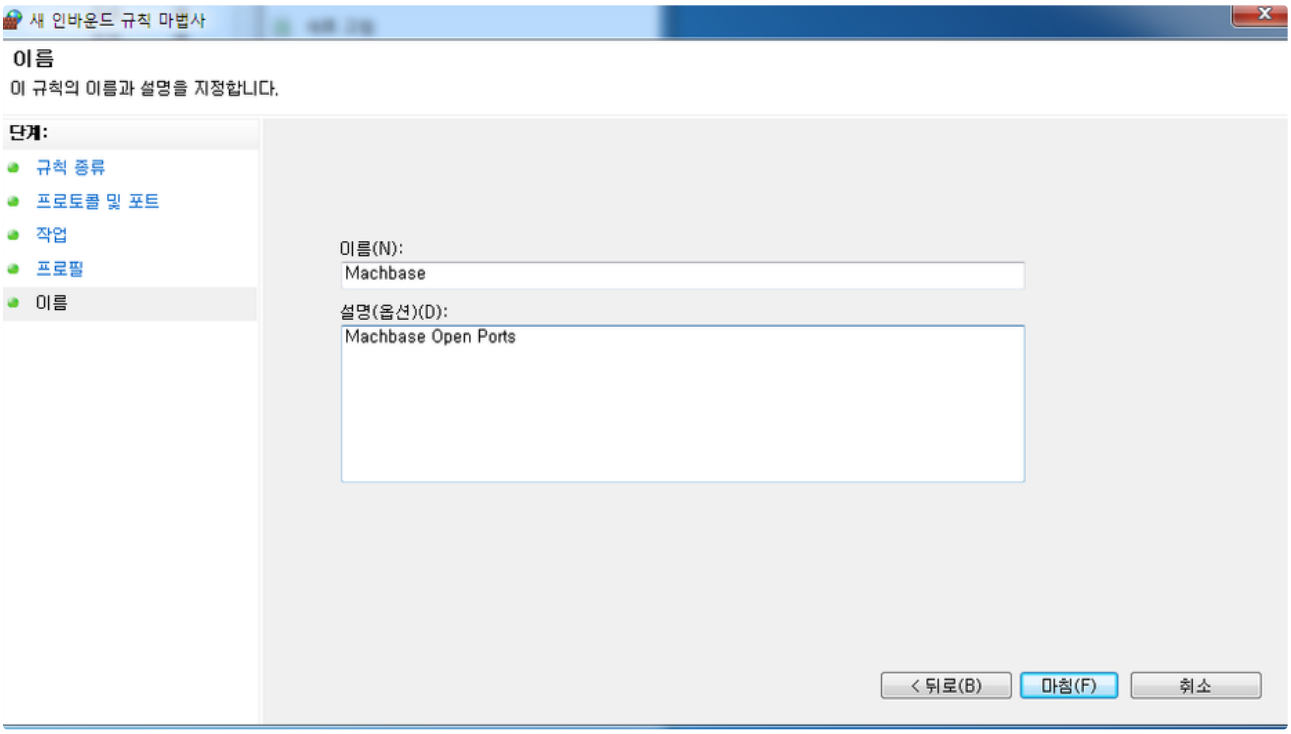
5. **연결 허용** 옵션을 선택하고 **다음**을 클릭한다.



6. **도메인**, **개인**, **공용**을 체크하고 **다음**을 클릭한다.



7. **이름**과 **설명** 입력란에 내용을 입력한 후 **마침**을 클릭한다.



라이선스 설치

라이선스 키 설치는 일반적으로 마크베이스 설치가 끝난 후에 수행한다. 설치 이후에 라이선스를 설치하지 않았다면 일부 제약이 있는 상태로 마크베이스의 사용이 가능하다. 이 장은 마크베이스의 라이선스 정책과 구조 및 설치 방법 등에 대해 기술한다.

라이선스 파일 구조

마크베이스의 라이선스는 license.dat 파일로 관리된다. 제품을 구매하거나 혹은 테스트를 위해 받는 라이선스는 텍스트 파일의 형태이다.

```
mach@localhost:~$ cat license.dat

\#Company\#ID-ProjectName: test\#0-Machbase
 \#License Policy: SIZE4DAY
 \#License Type \(Version 2\): OFFICIAL
 \#Issue DATE: 20160216
 \#Expiry DATE: 20160319
 VBz5h4TC-d3+Bf3Efkpdp/Tx873PpZA-78LRSdrxbPY-xhGf4355/iXaY5/jfnn+Jdpjn+N+ef4l2m
```

라이선스 파일이 없는 경우

라이선스가 없는 경우에도 서버가 구동되지만 일부 제약 사항이 있다. 평가 용도로만 사용할 수 있으며, 정식으로 사용하려는 경우에는 적법한 절차로 라이선스를 취득하여야 한다.

라이선스 파일이 없을 경우에는 아래와 같은 기능적인 제약이 존재한다.

1. 한 세션에서 Append 프로토콜을 통해 1억건 이상의 레코드를 입력할 경우 경고 메시지가 출력된다. 이후 Append 입력이 정지된다. 서버를 재시작할 경우에만 입력 제한 상태가 해제된다.
2. 테이블스페이스를 생성할 때 2개 이상의 디스크 디렉터리에 대해 생성할 수 없으며, 만약 2개 이상 사용할 경우 아래와 같은 오류가 출력된다. 즉, 고성능 데이터 입력을 위한 병렬 I/O 기능을 사용할 수 없다.

```
CREATE TABLESPACE tbs1 DATADISK disk1 (disk_path="tbs1_disk1"), disk2 (disk_path="tbs1_disk2"), disk3 (disk_path="t
[ERR-00867 : Error in adding disk to tablespace. You cannot use multiple disks for tablespace without valid license
```

라이선스 설치

마크베이스의 라이선스는 반드시 *$MACHBASE_HOME/conf* 에 설치하고, *license.dat*를 기본 이름으로 한다. 라이선스를 설치하는 방법에는 아래처럼 세 가지가 있다.

라이선스 파일을 $MACHBASE_HOME/conf에 복사

이때 발급 받은 라이선스 파일의 이름을 반드시 **license.dat** 로 바꾼 후 복사해야 한다. 이후 서버 구동시 해당 라이선스가 적합한지 판별하여 서버를 구동한다.

`machadmin -t 'licensefile_path'` 실행

이 방법의 장점은 라이선스 파일 이름이나 위치를 맞춰줄 필요가 없이 명령어로 손쉽게 설치가 가능하다.

쿼리문으로 설치: 이 방법은 서버 구동 중에 쿼리문을 이용하여 라이선스를 설치하는 방법이다.

라이선스 설치 확인

라이선스가 설치된 경우

라이선스 파일이 설치된 경우 서버 구동 이후에 machbase.trc에 아래와 같이 출력된다.

```
[2016-02-17 14:51:00 P-20913 T-140709874054912][INFO] LICENSE [License Type (Version 2)][OFFICIAL]
[2016-02-17 14:51:00 P-20913 T-140709874054912][INFO] LICENSE [License Policy] [CORE]
[2016-02-17 14:51:00 P-20913 T-140709874054912][INFO] LICENSE [Host ID] [FFFFFFFFFFFFFFF]
[2016-02-17 14:51:00 P-20913 T-140709874054912][INFO] LICENSE [Expiry DATE] [25300318]
```

```
[2016-02-17 14:51:00 P-20913 T-140709874054912][INFO] Machbase Logs Signature! : OFFICIAL:CORE:FFFFFFFFFFFFFF:253(
```

그리고 machadmin -f 명령어로도 확인 할 수 있다.

라이선스가 없는 경우

라이선스 파일이 설치되지 않았거나 비정상 파일을 사용할 경우에는 아래와 같이 출력된다.

```
[2016-02-17 14:49:54 P-6620 T-140539052701440][INFO] LICENSE [License Type(Version 2)][Only for evaluation (No lice
[2016-02-17 14:49:54 P-6620 T-140539052701440][INFO] LICENSE [License Policy] [None]
[2016-02-17 14:49:54 P-6620 T-140539052701440][INFO] LICENSE [Host ID] [Unknown]
[2016-02-17 14:49:54 P-6620 T-140539052701440][INFO] LICENSE [Expiry DATE] [N/A]
```

Cluster Edition 설치

Cluster Edition 설치 준비

목차

파일 LIMIT 확인 및 변경

열 수 있는 최대 파일 개수를 늘려야 하므로, 아래와 같이 수행한다.

1. `/etc/security/limits.conf` 파일을 아래와 같이 수정한다.

```
sudo vi /etc/security/limits.conf
*       hard    nofile      65535
*       soft    nofile      65535
```

2. 재부팅한다.

```
sudo reboot
# 또는
sudo shutdown -r now
```

3. 아래 명령어를 실행하여 결과를 확인한다. 65535 가 출력되면 성공적으로 변경된 것이다.

```
ulimit -Sn
```

서버 시간 동기화

각 Host 간 서버 시간을 동기화해야 한다. 이미 동기화를 하고 있다면 확인 차원에서 점검하도록 하자.

1. 모든 서버의 시간을 time 서버와 동기화한다.

```
# 다음 명령어로 동기화한다.
/usr/bin/rdate -s time.bora.net && /sbin/clock -w
```

2. Time 서버를 활용할 수 없는 경우 직접 명령어로 수정한다.

```
# 다음 명령어로 수정한다.
date -s "2017-10-31 11:15:30"
```

3. 변경된 시간을 확인한다

```
# 다음 명령어로 확인한다.
date
```

네트워크 커널 파라미터 변경

1. 현재 설정된 값을 확인한다.

```
# 다음 명령어로 확인한다.
sysctl -a | egrep 'mem_(max|default)|tcp_.*mem'
```

2. 아래 명령어로 설정값을 변경한다.(64GB Memory 기준)

```
sysctl -w net.core.rmem_default = "33554432"     # 32MB
sysctl -w net.core.wmem_default = "33554432"
sysctl -w net.core.rmem_max     = "268435456"    # 256MB
sysctl -w net.core.wmem_max     = "268435456"    # 최소 256KB 기본 32MB 최대 256MB
sysctl -w net.ipv4.tcp_rmem     = "262144 33554432 268435456"
sysctl -w net.ipv4.tcp_wmem     = "262144 33554432 268435456"

# 8388608 Page * 4KB = 32GB (가용 메모리에 따라 변경 필요)
sysctl -w net.ipv4.tcp_mem      = "8388608 8388608 8388608"
```

3. 변경값을 유지하려면 /etc/sysctl.conf 파일에 추가하고 호스트 OS 를 재시작한다

3. 변경값을 테스트하려면 /etc/sysctl.conf 파일에 추가하고 호스트 OS 를 재부팅한다.

```
# /etc/sysctl.conf 파일 수정한다.
net.core.rmem_default = "33554432"
net.core.wmem_default = "33554432"
net.core.rmem_max     = "268435456"
net.core.wmem_max     = "268435456"
net.ipv4.tcp_rmem     = "262144 33554432 268435456"
net.ipv4.tcp_wmem     = "262144 33554432 268435456"
net.ipv4.tcp_mem      = "8388608 8388608 8388608"
```

사용자 생성

1. Machbase 설치를 위한 리눅스 OS 사용자 machbase를 생성한다.
   사용자 계정 디렉토리는 /home/machbase 로 생성되도록 한다.

```
# 다음 명령어로 사용자 "machbase"를 추가한다.
$ sudo useradd machbase --home-dir "/home/machbase"
```

2. 사용자 "machbase"의 비밀번호를 설정한다.

```
sudo passwd machbase
```

Cluster Edition 설치 (Command-line)

(1) Coordinator / Deployer 설치, Package 추가

Coordinator 설치

목차

환경 설정

machbase 계정으로 로그인한 후에, machbase 권한으로 다음과 같이 파일을 수정하여 설치 디렉터리와 경로 정보에 대한 환경을 설정한다.

```
# .bashrc 편집한다.
export MACHBASE_COORDINATOR_HOME=~/coordinator
export MACHBASE_DEPLOYER_HOME=~/deployer
export MACHBASE_HOME=~/coordinator
export PATH=$MACHBASE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$MACHBASE_HOME/lib:$LD_LIBRARY_PATH

# 변경된 내용을 반영한다.
source .bashrc
```

디렉터리 생성 및 압축 해제

전용 디렉터리를 생성하고 패키지 압축 파일을 해당 디렉터리에 압축 해제한다.

```
# 디렉터리 생성한다.
mkdir $MACHBASE_COORDINATOR_HOME

# 압축 해제한다.
tar zxvf machbase-ent-x.y.z.official-LINUX-X86-64-release.tgz -C $MACHBASE_COORDINATOR_HOME
```

포트 설정 및 서비스 구동

machbase.conf 파일을 수정하여 포트를 설정하고 서비스를 구동한다.

```
# machbase.conf 파일에서 포트번호를 설정한다.
cd $MACHBASE_COORDINATOR_HOME/conf
cp machbase.conf.sample machbase.conf
vi machbase.conf
CLUSTER_LINK_HOST       = 192.168.0.83 (추가할 노드 ip)
CLUSTER_LINK_PORT_NO    = 5101
HTTP_ADMIN_PORT         = 5102

# 메타 정보를 생성하고 서비스 구동한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin -c
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin -u
```

노드 등록 및 확인

Coordinator 노드를 추가하고 확인한다.

```
# 노드 등록.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.83:5101" --node-type=coordinator

# 노드 확인.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --cluster-status
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로 "IP:PORT" 형식으로 지정한다. PORT값은 CLUSTER_LINK_PORT_NO 값이다.	192.168.0.83:5101
--node-type	노드 종류를 지정한다. coordinator / deployer / broker / warehouse 4종류가 있다.	coordinator

Coordinator 삭제

Coordinator가 설치된 서버로 접속하여 Coordinator 프로세스를 정상 종료시킨 후 해당 Coordinator 디렉토리를 삭제한다.

```
# coordinator를 종료하고 디렉토리를 삭제한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin -s
rm -rf $MACHBASE_COORDINATOR_HOME
```

Secondary Coordinator 설치

Primary Coordinator 외에 추가 Coordinator 를 설치하는 경우, 다음을 주의한다.

- Secondary Coordinator 의 Startup 이전에, Primary Coordinator 에 가서 Secondary Coordinator 를 Add-Node 해야 한다.
- Secondary Coordinator 의 Startup 을 할 때, **`--primary`** 옵션으로 Primary Coordinator 를 지정해야 한다.
- Secondary Coordinator 에 Primary Coordinator 를 Add-Node 해서는 안 된다.

이 경우를 지키지 않는다면, Secondary Coordinator 역시 Primary Coordinator 처럼 행동한다.

디렉터리 생성 및 압축 해제

전용 디렉터리를 생성하고 패키지 압축 파일을 해당 디렉터리에 해제한다.

```
# 디렉터리 생성한다.
mkdir $MACHBASE_COORDINATOR_HOME

# 압축 해제한다.
tar zxvf machbase-ent-x.y.z.official-LINUX-X86-64-release.tgz -C $MACHBASE_COORDINATOR_HOME
```

포트 설정

machbase.conf 파일을 수정하여 포트 설정만 한다. **서비스 구동하면 Primary Coordinator 처럼 작동한다.**

```
# machbase.conf 파일에서 포트 설정한다.
cd $MACHBASE_COORDINATOR_HOME/conf
vi machbase.conf
CLUSTER_LINK_HOST       = 192.168.0.83 (추가할 노드 ip)
CLUSTER_LINK_PORT_NO    = 5111
HTTP_ADMIN_PORT         = 5112
```

노드 등록 및 확인

Primary Coordinator 에서, Secondary Coordinator 노드를 추가하고 확인한다.

```
# 노드 등록.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.83:5111" --node-type=coordinator

# 노드 확인.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --cluster-status
```

서비스 구동

이제 Secondary Coordinator를 구동한다. Startup을 할 때, **`--primary`** 옵션으로 Primary Coordinator를 지정해야 한다.

```
# 메타 정보를 생성하고 서비스를 구동한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin -c
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin -u --primary="192.168.0.83:5101"
```

Secondary Coordinator 삭제

Primary Coordinator에 등록된 Secondary Coordinator를 삭제한 후 Secondary Coordinator의 프로세스를 정상 종료시켜야 한다.

```
# 노드 삭제.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --remove-node="192.168.0.83:5101"

# secondary coordinator를 종료하고 디렉토리를 삭제한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin -s
rm -rf $MACHBASE_COORDINATOR_HOME

# 노드 확인.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --cluster-status
```

옵션 항목	설명	예시
--remove-node	삭제할 노드명으로, "IP:PORT" 형식으로 지정한다. PORT 값은 CLUSTER_LINK_PORT_NO 값이다.	192.168.0.84:5201

Deployer 설치

> **정보**
>
> Deployer는 broker와 warehouse가 설치되는 모든 Host, 즉 서버에 미리 설치해야 한다.

환경 설정

다음과 같이, 설치 디렉터리와 경로에 대한 환경을 설정한다.

```
# .bashrc 편집한다.
export MACHBASE_DEPLOYER_HOME=~/deployer
export MACHBASE_HOME=~/deployer
export PATH=$MACHBASE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$MACHBASE_HOME/lib:$LD_LIBRARY_PATH

# 변경된 내용을 반영한다.
source .bashrc
```

디렉터리 생성 및 압축 해제

전용 디렉터리를 생성하고 패키지 압축 파일을 해당 디렉터리에 압축 해제한다.

```
# 디렉터리를 생성한다.
mkdir $MACHBASE_DEPLOYER_HOME

# 압축을 해제한다.
tar zxvf machbase-ent-x.y.z.official-LINUX-X86-64-release.tgz -C $MACHBASE_DEPLOYER_HOME
```

포트 설정 및 서비스 구동

machbase.conf 파일을 수정하여 포트를 설정하고 서비스를 구동한다.

```
# machbase.conf 파일에서 포트를 설정한다.
cd $MACHBASE_DEPLOYER_HOME/conf
vi machbase.conf
CLUSTER_LINK_HOST       = 192.168.0.84
```

```
CLUSTER_LINK_PORT_NO    = 5201
HTTP_ADMIN_PORT         = 5202

# 메타 정보를 생성하고 서비스를 구동한다.
$MACHBASE_DEPLOYER_HOME/bin/machdeployeradmin -c
$MACHBASE_DEPLOYER_HOME/bin/machdeployeradmin -u
```

노드 등록 및 확인

> **주의**
> 이 작업은 coordinator 노드에서 수행해야 한다.

Deployer 노드를 추가하고 확인한다.

```
# 노드 등록.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.84:5201" --node-type=deployer

# 노드 확인.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --cluster-status
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로, "IP:PORT" 형식으로 지정한다. PORT 값은 CLUSTER_LINK_PORT_NO 값이다.	192.168.0.84:5201
--node-type	노드 종류를 지정한다. coordinator / deployer / broker / warehouse 4종류가 있다.	deployer

Deployer 삭제

Coordinator 노드에서 Deployer 노드를 삭제하고, Deployer가 있는 서버에서 Deployer 프로세스를 정상 종료시켜야 한다.

```
# 노드 삭제.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --remove-node="192.168.0.84:5201"

# deployer를 종료하고 디렉토리를 삭제한다.
$MACHBASE_DEPLOYER_HOME/bin/machdeployeradmin -d
rm -rf $MACHBASE_DEPLOYER_HOME

# 노드 확인.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --cluster-status
```

패키지 추가

Coordinator에 broker와 warehouse로 설치될 패키지를 추가 등록한다. 이때 등록되는 패키지로 MWA가 제외된 lightweight 버전을 사용한다.

```
# 설치 패키지를 추가 등록한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-package=machbase \
    --file-name="/home/machbase/machbase-ent-x.y.z.official-LINUX-X86-64-release-lightweight.tgz"
```

옵션 항목	설명	예시
--add-package	추가할 패키지명을 지정한다.	machbase
--file-name	패키지 파일의 전체 경로와 파일명을 지정한다.	/home/machbase/machbase-ent-5.0.0.official-LINUX-X86-64-release-lightweight.tgz

옵션 항목	설명	예시
	Broker와 warehouse 설치만을 위한 패키지이므로, MWA 파일이 제외된 lightweight 패키지를 지정한다.	

패키지 삭제

Coordinator에 등록한 패키지를 삭제한다.

```
# 등록한 패키지를 삭제한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --remove-package=machbase
```

(2) Lookup/Broker / Warehouse 설치

Lookup 설치

Coordinator 노드에서 lookup 노드를 추가한다. 여러 개의 lookup 노드 등록이 가능하다.

해당 서버에 deployer 노드가 미리 설치되어 있어야 한다.

Deployer 노드가 설치되면, 모든 작업은 coordinator 노드에서 수행하게 되며 해당 서버에 접속해서 설정하는 것은 없다.

```
# lookup master 노드를 추가한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.84:53
        --node-type=lookup --lookup-type=master --deployer="192.168.0.84:5201"
        --home-path="/home/machbase/lookup1"


# lookup monitor 노드를 추가한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.84:53
        --node-type=lookup --lookup-type=monitor --deployer="192.168.0.84:5201"
        --home-path="/home/machbase/lookupm1"


# lookup slave 노드를 추가한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.84:53
        --node-type=lookup --lookup-type=slave --deployer="192.168.0.84:5201"
        --home-path="/home/machbase/lookup3"


# lookup 노드를 실행한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --startup-node="192.168.0.8

# lookup 노드를 일괄적으로 실행할 수 있다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --startup-lookup
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로, "IP:PORT" 형식으로 지정한다.	192.168.0.84:5301
--node-type	노드 종류를 지정한다. coordinator, deployer, broker, lookup,warehouse 5종류가 있다.	lookup
--deployer	설치할 서버의 deployer node 정보를 등록한다.	192.168.0.84:5201
--lookup-type	lookup 노드 종류를 지정한다. master,slave,monitor 3종류가 있다.	master
--home-path	설치할 경로를 지정한다. machbase 계정에서 /home/machabse/lookup1 로 지정한다	/home/machabse/lookup

설치 조건

Lookup 노드는 Master, Slave, Monitor 3 종류가 존재하는데 아래 조건에 맞게 설치해야 한다.

1. Lookup Master 노드
    a. 반드시 1개가 존재해야 하는 Lookup 노드이다.
    b. Monitor, Slave 노드보다 먼저 설치되어야 한다.
2. Lookup Monitor 노드
    a. 반드시 존재해야 하는 Lookup 노드이다.
    b. 안정적인 HA를 위해 각 서버에 1개씩 존재해야 한다.
3. Lookup Slave 노드
    a. HA를 위해 1개 이상 존재하는 것을 권장한다. (없을경우 HA를 보장할 수 없다)

목차

Lookup 삭제

Coordinator 노드에서 lookup 노드를 삭제한다.

```
# lookup 노드를 삭제한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --remove-node="192.168.0.84
```

Lookup 종료/중단

Coordinator 노드에서 lookup 노드를 종료/중단하는 방법이 있다.

```
# lookup 노드를 종료한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --shutdown-node="192.168.0.

# lookup 노드를 일괄적으로 종료할 수 있다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --shutdown-lookup
```

Lookup Master 변경

Coordinator 노드에서 lookup master 노드를 변경하는 방법이 있다.

lookup slave에 한해서 lookup master로 변경 가능하며, 기존에 lookup master는 lookup slave가 된다.

```
# lookup master를 변경한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --set-lookup-master="192.16
```

Broker 설치

Coordinator 노드에서 broker 노드를 추가한다. 여러 개의 broker 노드 등록이 가능하다.

해당 서버에 deployer 노드가 미리 설치되어 있어야 한다.

Deployer 노드가 설치되면, 모든 작업은 coordinator 노드에서 수행하게 되며 해당 서버에 접속해서 설정하는 것은 없다.

최초에 등록되는 노드가 leader broker가 되고 이후에 추가적으로 등록되는 노드는 follower broker가 된다.

```
# broker 노드를 추가한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.84:53
        --node-type=broker --deployer="192.168.0.84:5201" --port-no="5656"
        --home-path="/home/machbase/broker" --package-name=machbase

# broker 노드를 실행한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --startup-node="192.168.0.8
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로, “IP:PORT” 형식으로 지정한다. PORT 값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.84:5301
--node-type	노드 종류를 지정한다. coordinator, deployer, broker, lookup,warehouse 5종류가 있다.	broker
--deployer	설치할 서버의 deployer node 정보를 등록한다.	192.168.0.84:5201
--port-no	machbased 구동 포트를 지정한다. Broker는 디폴트값인 5656을 지정한다. client와 machsql이 접속할 때 이 포트를 이용한다.	5656
--home-path	설치할 경로를 지정한다. machbase 계정에서 /home/machabse/broker 로 지정한다	/home/machabse/broker

옵션 항목	설명	예시
--package-name	패키지 추가할 때 지정한 package 명을 설정한다.	machbase

Broker 삭제

Coordinator 노드에서 broker 노드를 삭제한다.

```
# broker 노드를 삭제한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --remove-node="192.168.0.84:5301"
```

Broker 종료/중단

Coordinator 노드에서 broker 노드를 종료/중단하는 방법이 있다.

```
# broker 노드를 종료한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --shutdown-node="192.168.0.84:5301"

# broker 노드를 중단한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --kill-node="192.168.0.84:5301"
```

또는, broker가 설치된 서버에서 직접 그 프로세스를 종료/중단하는 방법도 있다.

```
# broker 노드를 종료한다.
$MACHBASE_HOME/bin/machadmin -s

# broker 노드를 종료한다.
$MACHBASE_HOME/bin/machadmin -k
```

Warehouse 설치

Coordinator 노드에서 active 노드와 standby 노드를 설치한다.

사전에 설치된 deployer 를 통해서 설치된다.

Group1 설치

첫번째 Warehouse 그룹인 Group1 노드를 설치한다.

```
# group1 warehouse를 설치한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.83:5401"  \
        --node-type=warehouse --deployer="192.168.0.83:5201" --port-no="5400"       \
        --home-path="/home/machbase/warehouse_g1" --package-name=machbase           \
        --replication="192.168.0.83:5402"  --group="group1" --no-replicate

# 설치된 노드를 구동한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --startup-node="192.168.0.84:5401"
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로 "IP:PORT" 형식으로 지정한다. PORT값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.84:5401
--node-type	노드 종류를 지정한다. coordinator, deployer, broker, warehouse,lookup 5종류가 있다.	warehouse
--deployer	설치할 서버의 deployer node 정보를 등록한다.	192.168.0.84:5201

옵션 항목	설명	예시
--port-no	machbased 구동 포트를 지정한다. Broker에서 5656값을 설정하였으므로 동일 서버에 설치되는 경우 다른 포트를 지정해야 한다. 따라서 warehouse 사용 포트 대역인 5400 을 지정한다. client와 machsql 접속할 때 이 포트를 이용한다.	5400
--home-path	설치할 경로를 지정한다. 그룹을 구분하기 위해서 warehouse_g1, g2, g3 순으로 설정한다.	/home/machbase/warehouse_g1
--package-name	패키지 추가할 때 지정한 package 명을 설정한다.	machbase
--replication	Replication을 담당할 노드를 "IP:PORT" 형식으로 지정한다. PORT값은 해당 warehouse 사용 포트대역인 5402로 지정한다.	192.168.0.84:5402
--group	Group명을 지정한다.	group1
--no-replicate	Group내의 warehouse데이터가 있는 경우, 노드추가 시, 데이터를 복제할 것 인지 지정한다.	

Group1 에 노드 추가 설치

Warehouse Group1에 노드를 한 개 더 추가 설치한다.

```
# group1에 warehouse node를 추가 설치한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.84:5401"  \
        --node-type=warehouse --deployer="192.168.0.84:5201" --port-no="5400"       \
        --home-path="/home/machbase/warehouse_g1" --package-name=machbase           \
        --replication="192.168.0.84:5402" --group="group1" --no-replicate

# 설치된 노드를 구동한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --startup-node="192.168.0.84:5401"
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로 "IP:PORT" 형식으로 지정한다. PORT값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.84:5401
--node-type	노드 종류를 지정한다. coordinator, deployer, broker, warehouse,lookup 5종류가 있다.	warehouse
--deployer	설치할 서버의 deployer node 정보를 등록한다.	192.168.0.84:5201
--port-no	machbased 구동 포트를 지정한다. Broker에서 5656값을 설정하였으므로 동일 서버에 설치되는 경우 다른 포트를 지정해야 한다. 따라서 warehouse 사용 포트 대역인 5400 을 지정한다. client와 machsql 접속할 때 이 포트를 이용한다.	5400
--home-path	설치할 경로를 지정한다. 그룹을 구분하기 위해서 warehouse_g1, g2, g3 순으로 설정한다.	/home/machabse/warehouse_g1
--package-name	패키지 추가할 때 지정한 package 명을 설정한다.	machbase
--replication	Replication 을 담당할 노드를 "IP:PORT" 형식으로 지정한다. PORT값은 해당 warehouse 사용 포트대역인 5402로 지정한다.	192.168.0.84:5402
--group	Group명을 지정한다.	group1
--no-replicate	Group내의 warehouse데이터가 있는 경우, 노드추가 시, 데이터를 복제할 것 인지 지정한다.	
--set-group-state	그룹의 상태를 normal과 readonly 두가지로 지정한다. Normal은 읽기,쓰기 / Readonly 읽기만 가능	

Group2 설치

두 번째 Warehouse 그룹인 Group2 노드를 설치한다.

```
# group1 warehouse를 설치한다.
```

```
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.84:5411"  \
        --node-type=warehouse --deployer="192.168.0.84:5201" --port-no="5410"       \
        --home-path="/home/machbase/warehouse_g2" --package-name=machbase           \
        --replication="192.168.0.84:5412"  --group="group2" --no-replicate

# 설치된 노드를 구동한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --startup-node="192.168.0.84:5411"
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로 "IP:PORT" 형식으로 지정한다. PORT값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.84:5411
--node-type	노드 종류를 지정한다. coordinator, deployer, broker, warehouse,lookup 5종류가 있다.	warehouse
--deployer	설치할 서버의 deployer node 정보를 등록한다.	192.168.0.84:5201
--port-no	machbased 구동 포트를 지정한다. Broker에서 5656값을 설정하였으므로 동일 서버에 설치되는 경우 다른 포트를 지정해야 한다. 따라서 warehouse 사용 포트 대역인 5410 을 지정한다. client와 machsql 접속할 때 이 포트를 이용한다.	5410
--home-path	설치할 경로를 지정한다. 그룹을 구분하기 위해서 warehouse_g1, g2, g3 순으로 설정한다.	/home/machbase/warehouse_g2
--package-name	패키지 추가할 때 지정한 package 명을 설정한다.	machbase
--replication	Replication을 담당할 노드를 "IP:PORT" 형식으로 지정한다. PORT값은 해당 warehouse 사용 포트대역인 5412로 지정한다.	192.168.0.84:5412
--group	Group명을 지정한다.	group2
--no-replicate	Group내의 warehouse데이터가 있는 경우, 노드추가 시, 데이터를 복제할 것 인지 지정한다.	
--set-group-state	그룹의 상태를 normal과 readonly 두가지로 지정한다. Normal은 읽기,쓰기 / Readonly 읽기만 가능	

Group2에 노드 추가 설치

Warehouse Group2에 노드를 한 개 더 추가 설치한다.

```
# group1에 warehouse node를 추가 설치한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --add-node="192.168.0.83:5411"  \
        --node-type=warehouse --deployer="192.168.0.83:5201" --port-no="5410"       \
        --home-path="/home/machbase/warehouse_g2" --package-name=machbase           \
        --replication="192.168.0.83:5412" --group="group2" --no-replicate

# 설치된 노드를 구동한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --startup-node="192.168.0.83:5411"
```

옵션 항목	설명	예시
--add-node	추가할 노드명으로 "IP:PORT" 형식으로 지정한다. PORT값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.83:5411
--node-type	노드 종류를 지정한다. coordinator, deployer, broker, warehouse, lookup 5종류가 있다.	warehouse
--deployer	설치할 서버의 deployer node 정보를 등록한다.	192.168.0.83:5201
--port-no	machbased 구동 포트를 지정한다. Broker에서 5656값을 설정하였으므로 동일 서버에 설치되는 경우 다른 포트를 지정해야 한다. 따라서 warehouse 사용 포트 대역인 5410 을 지정한다.	5410

옵션 항목	설명	예시
	client와 machsql 접속할 때 이 포트를 이용한다.	
--home-path	설치할 경로를 지정한다. 그룹을 구분하기 위해서 warehouse_g1, g2, g3 순으로 설정한다.	/home/machbase/warehouse_g2
--package-name	패키지 추가할 때 지정한 package 명을 설정한다.	machbase
--replication	Replication 을 담당할 노드를 "IP:PORT" 형식으로 지정한다. PORT값은 해당 warehouse 사용 포트대역인 5412로 지정한다.	192.168.0.83:5412
--group	Group명을 지정한다.	group2
--no-replicate	Group내의 warehouse데이터가 있는 경우, 노드추가 시, 데이터를 복제할 것 인지 지정한다.	
--set-group-state	그룹의 상태를 normal과 readonly 두가지로 지정한다. Normal은 읽기,쓰기 / Readonly 읽기만 가능	

Warehouse 삭제

Coordinator 노드에서 warehouse 노드를 삭제한다.

```
# warehouse 노드를 삭제한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --remove-node="192.168.0.83:5401"
```

Warehouse 종료/중단

Coordinator 노드에서 warehouse 노드를 종료/중단하는 방법이 있다.

```
# warehouse 노드를 종료한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --shutdown-node="192.168.0.83:5401"

# warehouse 노드를 중단한다.
$MACHBASE_COORDINATOR_HOME/bin/machcoordinatoradmin --kill-node="192.168.0.83:5401"
```

또는 warehouse가 설치된 서버에서 직접 그 프로세스를 종료/중단하는 방법이 있다.

```
# warehouse 노드를 종료한다.
$MACHBASE_HOME/bin/machadmin -s

# warehouse 노드를 종료한다.
$MACHBASE_HOME/bin/machadmin -k
```

Cluster Edition 설치 (Web Admin)

(1) MWA 설치

Cluster Edition 설치 (MWA) 목차

MWA 설치를 위해서는 Command-line 환경에서 패키지 압축을 풀어야 한다.

목차

환경 설정

먼저, 서버에 접속한다. 여기서는 machbase/machbase 라는 계정이 있다고 가정한다.

이후, 설치 디렉터리와 경로 정보에 대한 환경 설정을 한다.

```
export MACHBASE_HOME=~/coordinator
export PATH=$MACHBASE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$MACHBASE_HOME/lib:$LD_LIBRARY_PATH
```

Coordinator 디렉터리 생성 및 압축 해제

`$MACHBASE_HOME` 에 지정한 디렉터리를 생성하고, 패키지 파일을 해당 디렉터리에 압축 해제한다.

```
# 디렉터리 생성
mkdir $MACHBASE_HOME

# 설치 패키지를 다운로드 받아서 MACHBASE_HOME에 복사
cp -f machbase-ent-x.x.x.official-LINUX-X86-64-release.tgz $MACHBASE_HOME

# MACHBASE_HOME 으로 이동하여 압축 해제
cd $MACHBASE_HOME
tar zxvf machbase-ent-x.x.x.official-LINUX-X86-64-release.tgz
```

Cluster Edition 설치를 위한 MWA 설정

Cluster Edition 설치에는 많은 항목들의 입력이 요구된다. MWA는 설치 시에 이런 항목들에 기본값을 제공한다.
이 기본값은 사용자가 원하는 값으로 변경할 수 있다.
`$MACHBASE_HOME/webadmin/flask/MWA.conf`파일에서 필요한 부분을 수정하고 MWA를 재시작하면 된다.

MWA.conf에 설정된 기본값은 아래와 같다.

```
31
32  ################################################################################
33  # Default value in the node installation of cluster admin.
34  # - You can use $, # for the DEFAULT_[BROKER|WAREHOUSE]_HOME_PATH.
35  #   $ : Group Name
36  #   # : Warehouse index in same group
37  # - PORT_INCREMENT_VALUE           : When installing to a new group,
38  #                                    increase the port number by this value.
39  # - PORT_INCREMENT_VALUE_IN_GROUP  : When installing to an existing group,
40  #                                    increase the port number by this value.
41  # - CLUSTER_ADMIN_REFRESH_INTERVAL : Information update interval
42  #                                    for Cluster Admin (second)
43  ################################################################################
44  DEFAULT_COORDINATOR_SERVICE_PORT = 5100
45  DEFAULT_COORDINATOR_LINK_PORT    = 5101
46  DEFAULT_COORDINATOR_ADMIN_PORT   = 5102
47  DEFAULT_COORDINATOR_HOME_PATH    = /home/machbase/coordinator
```

```
48
49  DEFAULT_DEPLOYER_SERVICE_PORT    = 5200
50  DEFAULT_DEPLOYER_LINK_PORT       = 5201
51  DEFAULT_DEPLOYER_ADMIN_PORT      = 5202
52  DEFAULT_DEPLOYER_HOME_PATH       = /home/machbase/deployer
53
54  DEFAULT_BROKER_SERVICE_PORT      = 5656
55  DEFAULT_BROKER_LINK_PORT         = 5301
56  DEFAULT_BROKER_HOME_PATH         = /home/machbase/broker
57
58  DEFAULT_WAREHOUSE_SERVICE_PORT   = 5400
59  DEFAULT_WAREHOUSE_LINK_PORT      = 5401
60  DEFAULT_WAREHOUSE_REPL_PORT      = 5402
61
62  # Use if there is no same group in the server
63  DEFAULT_WAREHOUSE_HOME_PATH      = /home/machbase/warehouse_$
64  PORT_INCREMENT_VALUE             = 10
65  PORT_INCREMENT_VALUE_IN_GROUP    = 0
66
67  # Use if there is a same group in the server
68  #DEFAULT_WAREHOUSE_HOME_PATH      = /home/machbase/warehouse_$_w#
69  #PORT_INCREMENT_VALUE             = 100
70  #PORT_INCREMENT_VALUE_IN_GROUP    = 10
71
72  DEFAULT_SSH_USER_ID              = machbase
    CLUSTER_ADMIN_REFRESH_INTERVAL   = 5
```

- BROKER와 WAREHOUSE의 설치 폴더 기본값에는 $와 #을 사용할 수 있다.
  $는 Group명으로 치환되고, #은 Group내에 있는 Warehouse의 수로 치환된다.
- PORT_INCREMENT_VALUE는 Group이 추가될 때 Port 번호에 증가되는 값이다.
  PORT_INCREMENT_VALUE가 10인 경우, Group이 추가될 때마다 각 Port 번호에 10이 증가된다.
- PORT_INCREMENT_VALUE_IN_GROUP은 Group 내의 Warehouse가 추가될 때 Port 번호에 증가되는 값이다.
  PORT_INCREMENT_VALUE_IN_GROUP이 10인 경우, 같은 Group내에서 Warehouse가 추가될 때마다 각 Port 번호에 10이 증가된다.
  실제로 하나의 서버에 같은 Group의 Warehouse가 설치되는 경우는 거의 없으므로 테스트 용도로 주로 사용된다.
- DEFAULT_SSH_USER_ID은 SSH 접속에 사용되는 ID의 기본값이다.
  Coordinator와 Deployer 설치에는 SSH 접속이 필요하므로 설치시에 SSH ID와 Password를 입력해야 하는데 이때 기본값으로 사용된다.
- CLUSTER_ADMIN_REFRESH_INTERVAL은 Cluster Admin에서 Cluster의 상태를 체크하는 주기를 second 단위로 입력한다.
  MWA는 이 값을 주기로 Coordinator에 접속해서 Cluster의 상태를 얻어온다.

⚠ # 으로 시작하면 주석으로 인식한다.

MWA 서비스 구동

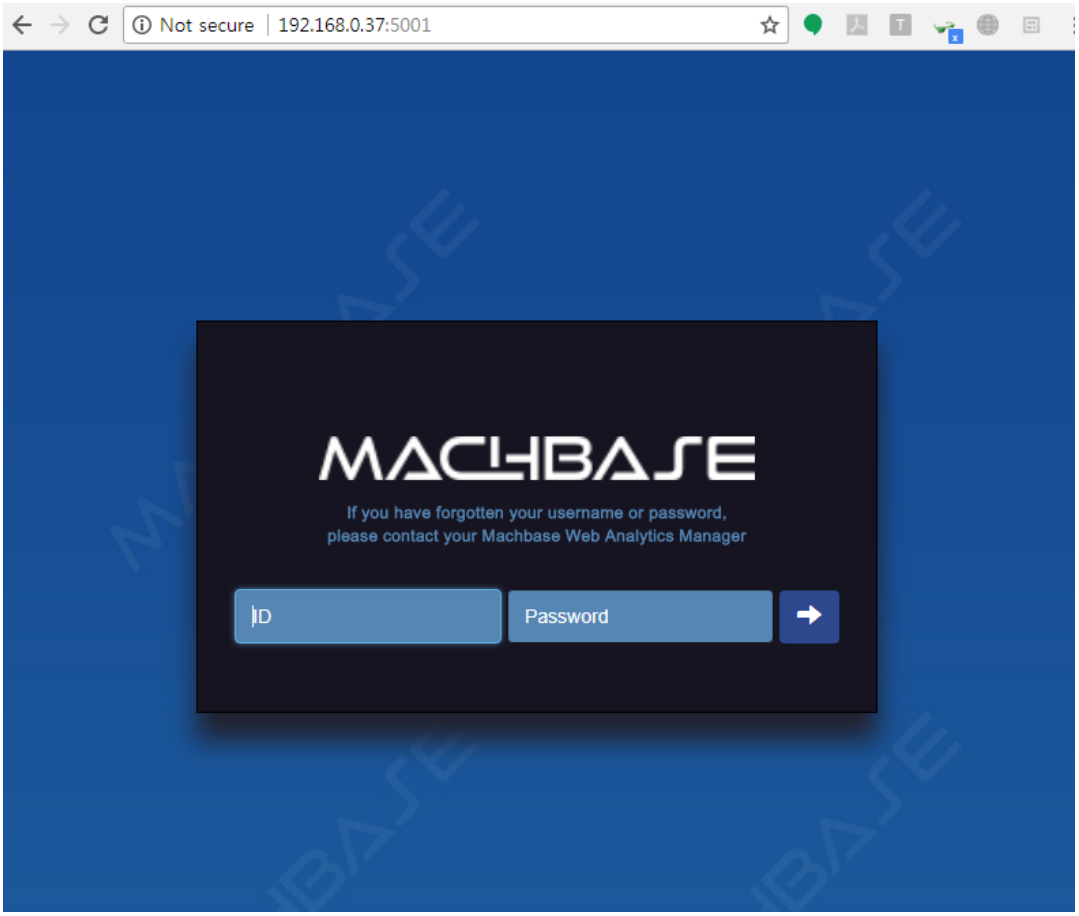
아래 명령을 통해 MWA 서비를 구동하면, 서비스 접속 주소가 출력된다.

```
# 디렉터리 이동
cd $MACHBASE_HOME/bin

# MWA 서비스 구동
MWAserver start
```

MWA 서비스 접속

브라우저를 열고, 출력된 접속 주소와 포트로 접속한다.

초기 웹 로그인 계정정보는 admin / machbase 이다.


Not secure
192.168.0.37:5001
MACHBASE
If you have forgotten your username or password,
please contact your Machbase Web Analytics Manager
ID
Password

(2) Coordinator / Deployer 설치

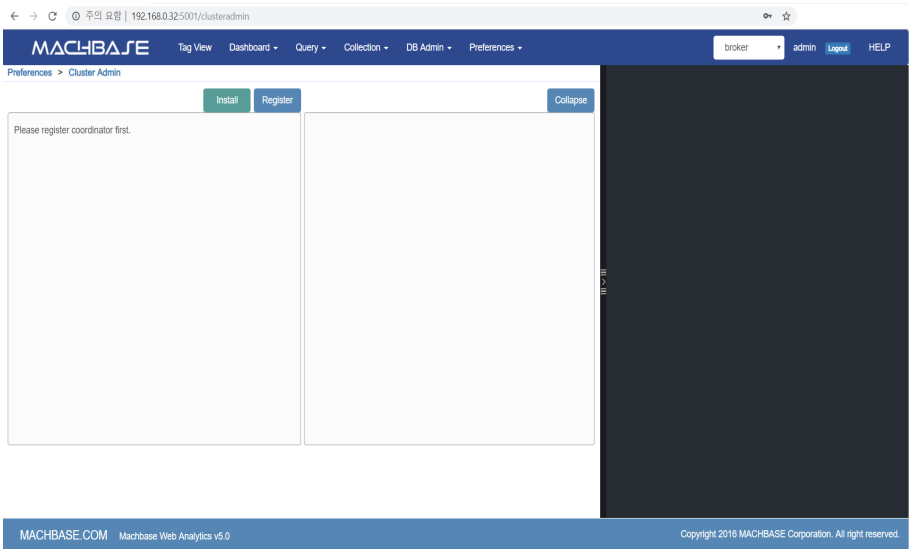
Cluster Edition 설치 (MWA) 목차

목차

Cluster Admin Page 이동

Web Admin에 접속하면 Cluster Admin 화면이 나타난다. 상단의 메뉴 Preferences를 클릭해도 나타난다.

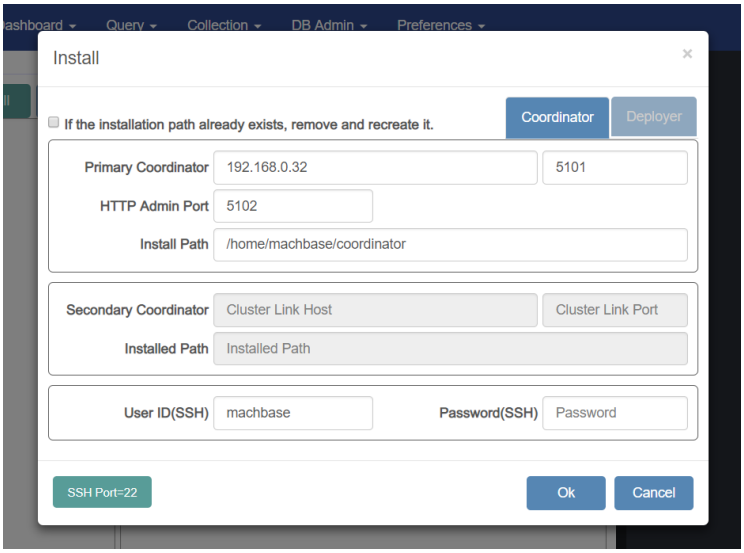
- Coordinator / Deployer 를 처음 설치하는 경우는 상단의 **[Install]** 버튼을 클릭해 진행한다.
- Command Line으로 이미 설치한 Coordinator / Deployer 를 등록하기 위해서는 상단의 **[Register]** 버튼을 클릭한다.



Coordinator 설치

Cluster Admin 페이지에서 **[Install]** 을 클릭한다.

각 항목들을 입력한 후 **[Ok]** 를 클릭한다.



항목	설명	예시
Primary Coordinator / Secondary Coordinator	추가할 Node 명으로 "IP:PORT" 형식으로 구성된다. PORT 값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.32:5101

항목	설명	예시
HTTP Admin Port	PORT 값은 HTTP_ADMIN_PORT 값으로 설정된다.	5102
Install Path	설치할 경로를 지정한다.	/home/machbase/coordinator
User ID(SSH) / Password(SSH)	설치하고자 하는 서버가 MWA가 동작하는 서버와 다를 경우, SSH의 User ID, Password를 입력한다.	

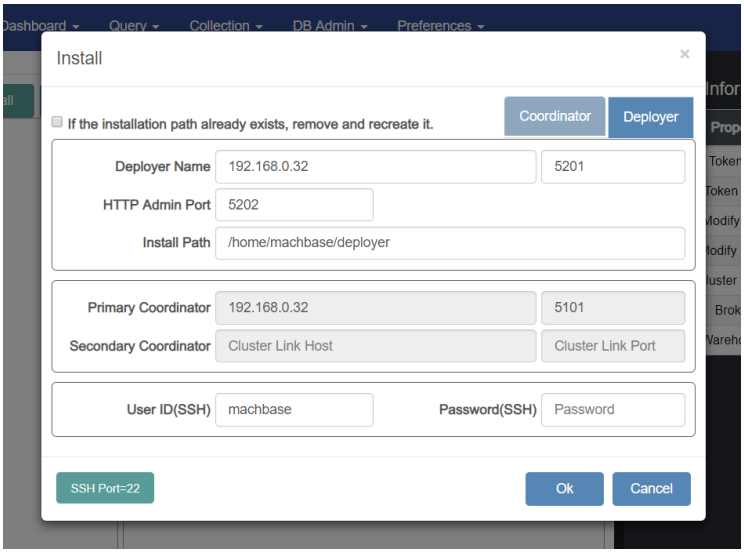
Coordinator 삭제

⚠ MWA에서는 coordinator 삭제 기능이 없다. 직접 command-line으로 삭제해야 한다.

Deployer 설치

Cluster Admin 페이지에서 **[Install]** 을 클릭한다.

각 항목들을 입력한 후 **[Ok]** 를 클릭한다.



항목	설명	예시
Deployer Name	추가할 Node 명으로 "IP:PORT" 형식으로 구성된다. PORT 값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.32:5201
HTTP Admin Port	PORT 값은 HTTP_ADMIN_PORT 값으로 설정된다.	5202
Install Path	설치할 경로를 지정한다.	/home/machbase/deployer
User ID(SSH) / Password(SSH)	설치하고자 하는 서버가 MWA가 동작하는 서버와 다를 경우, SSH의 User ID, Password를 입력한다.	

Deployer 삭제

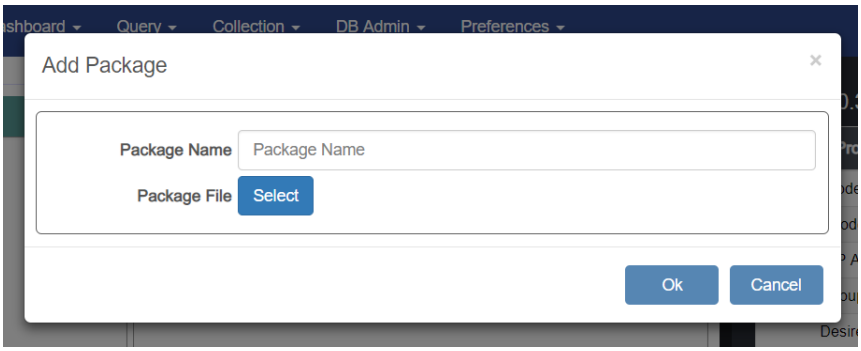
Cluster Admin 페이지에서 Deployer 아이콘(Ⓓ)을 클릭한 후, **[Remove Deployer]** 를 클릭한다.

그리고 command-line으로 해당 deployer를 정상 종료하고 디렉토리를 삭제한다.

패키지 추가

Coordinator에 Broker와 Warehouse로 설치될 패키지를 추가 등록한다. 이때 등록되는 패키지는 MWA가 제외된 lightweight 버전을 등록한다.

Cluster Admin 페이지에서 Coordinator 아이콘 (Ⓒ)을 클릭한 후 **[Add Package]** 를 클릭한다.

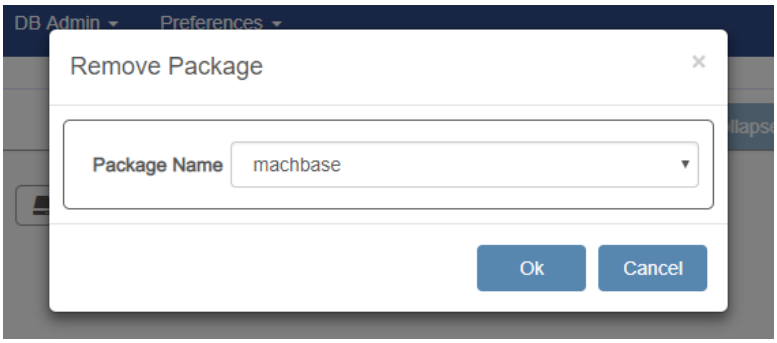


항목	설명	예시
Package Name	추가할 패키지명을 지정한다.	machbase
Package File	패키지 파일을 선택한다. Broker와 warehouse 설치만을 위한 패키지이므로, MWA 파일이 제외된 lightweight 패키지를 지정한다.	/home/machbase/machbase-ent-x.x.x.official-LINUX-X86-64-release-lightweight.tgz

패키지 삭제

Coordinator에 등록한 패키지를 삭제한다.

Cluster Admin 페이지에서 Coordinator 아이콘 ( )을 클릭한 후 **[Remove Package]** 를 클릭한다.



항목	설명	예시
Package Name	삭제할 패키지명을 지정한다.	machbase

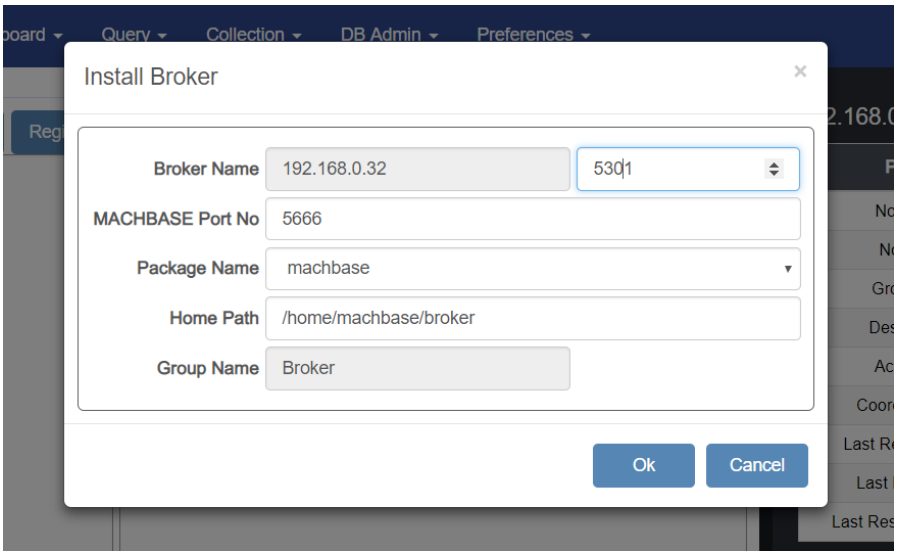
(3) Broker / Warehouse 설치

Cluster Edition 설치 (MWA) 목차

목차

Broker 설치

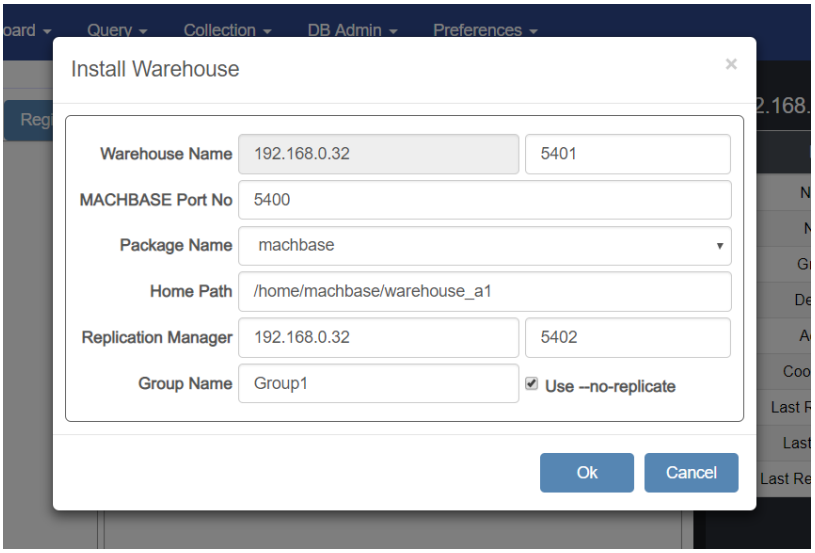
Cluster Admin 페이지에서 Deployer 아이콘(D)을 클릭한 후, **[Install Broker]** 를 클릭한다.
각 항목들을 입력하고 **[Ok]** 를 클릭한다.



항목	설명	예시
Broker Name	추가할 Node 명으로 "IP:PORT" 형식으로 구성된다. IP 값은 Deployer의 IP 값으로 자동 설정된다. PORT 값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.32:5301
MACHBASE Port No	machbased 구동 포트를 지정한다.	5656
Package Name	패키지 추가할 때 지정한 package 명을 설정한다.	machbase
Home Path	설치할 경로를 지정한다.	/home/machbase/broker

Warehouse 설치

Cluster Admin 페이지에서 Deployer 아이콘(D)을 클릭한 후, **[Install Warehouse]** 를 클릭한다.
각 항목들을 입력하고 **[Ok]** 를 클릭한다.



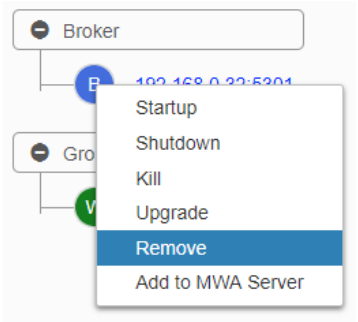
항목	설명	예시
Warehouse Name	추가할 Node 명으로 "IP:PORT" 형식으로 구성된다. IP 값은 Deployer의 IP 값으로 자동 설정된다. PORT 값은 CLUSTER_LINK_PORT_NO 값으로 설정된다.	192.168.0.32:5401
MACHBASE Port No	machbased 구동 포트를 지정한다.	5400
Package Name	패키지 추가할 때 지정한 package 명을 설정한다.	machbase
Home Path	설치할 경로를 지정한다.	/home/machbase/warehouse_a1
Replication Manager	Replication을 담당할 Node 를 "IP:PORT" 형식으로 지정한다.	192.168.0.32:5402
Group Name	Group명을 지정한다.	group1

설치할 Warehouse가 replication 을 하기 원한다면 **[Use --no-replicate] 체크박스를 해제**하면 된다.

이미 데이터가 입력된 상태에서 클러스터를 확장할 목적으로 Warehouse 를 추가할 때, 체크박스를 해제하면 된다.
체크박스를 해제한 채로, 빈 클러스터에 Warehouse 를 추가해도 무방하지만 replication 기능이 의미없이 진행되므로 설치 과정에 성능 저하를 유발할 수 있다.

Broker / Warehouse 삭제

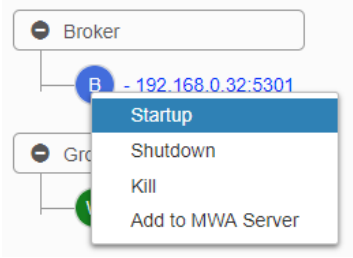
Cluster Admin 페이지에서 Broker / Warehouse 아이콘(/)을 클릭한 후, **[Remove]** 를 클릭한다.



만일, Cluster Status가 Service 상태라면 이를 Deactivate 상태로 바꿔야 한다. 아이콘 을 클릭하면, Cluster Status를 Service / Deactivate 상태로 변경할 수 있다.

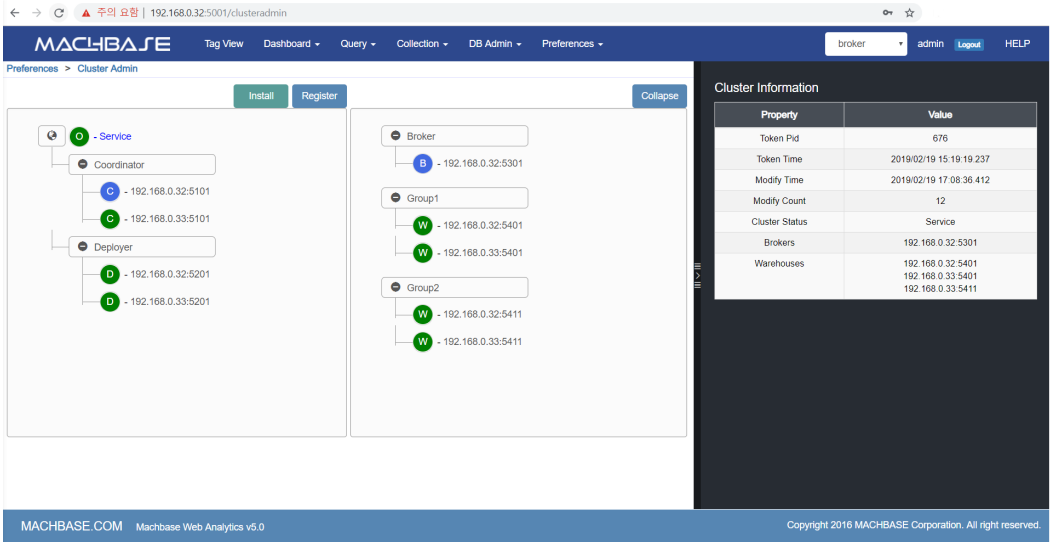
Broker / Warehouse 구동/종료/중단

Cluster Admin 페이지에서 Broker / Warehouse 아이콘(/)을 클릭한 후, **[Startup / Shutdown/ Kill]** 을 클릭한다.



클러스터 구성 완성

2개의 Coordinator, 2개의 Deployer, 1개의 Broker, 2개의 group으로 구성된 4개의 Warehouse들을 설치하면 아래의 사진과 같이 나타난다.



업그레이드

준비 : 호환 버전 확인

업그레이드를 하려면, 현재 설치된 마크베이스의 버전과 설치하려는 마크베이스 버전을 확인해야 한다.

- 메이저 버전이나 마이너 버전이 차이가 나는 경우, 마크베이스 기술지원을 통해 이전 도구를 요청해야 한다.
- 메이저 버전과 마이너 버전은 동일하고 패치 버전만 차이가 나는 경우, 업그레이드가 가능하다.

업그레이드 방법

- Cluster Edition 업그레이드 및 복구

Cluster Edition 업그레이드 및 복구

목차

Coordinator 업그레이드

Coordinator / Deployer 는 부득이하게 수동으로 업그레이드 해야 한다.

주의사항

- DDL 또는 DELETE 수행 중이 아니어야 한다. (INSERT, APPEND, SELECT 는 상관없다.)
- 업그레이드 중 Node 추가/구동/종료/삭제 등의 명령을 내릴 수 없다.

Coordinator 종료

Coordinator / Deployer 는 종료되어도 Broker / Warehouse 의 INSERT, APPEND, SELECT 에 영향을 주지 않는다.
다만, 종료된 동안에는 Broker / Warehouse 가 도중에 죽는 것을 감지하지 못한다. (재시작 후에는 정상 감지된다.)

```
machcoordinatoradmin --shutdown
```

(Optional) Coordinator 백업

`$MACH_COORDINATOR_HOME` 에 있는 `dbs/`, `conf/` 디렉토리를 백업한다.

Coordinator 업그레이드

lightweight package 가 아닌 full package 로 진행한다.

Package 를 `$MACH_COORDINATOR_HOME` 에 압축을 풀어 덮어쓴다.

```
tar zxvf machbase-ent-new.official-LINUX-X86-64-release.tgz -C $MACHBASE_COORDINATOR_HOME
```

Coordinator 시작

```
machcoordinatoradmin --startup
```

Deployer 업그레이드

Coordinator 와 동일하다.

주의 사항

- 업그레이드 중 Node 추가/구동/종료/삭제 등의 명령을 내릴 수 없다.

Deployer 종료

```
machdeployeradmin --shutdown
```

(Optional) Deployer 백업

`$MACH_DEPLOYER_HOME` 에 있는 `dbs/`, `conf/` 디렉토리를 백업한다.

Deployer 업그레이드

> Deployer 가 설치된 Host 에서 MWA 를 수행하거나 Collector 를 수행하지 않는다면, lightweight package 로 진행해도 무방하다.

Package 를 `$MACH_DEPLOYER_HOME` 에 압축을 풀어 덮어쓴다.

```
tar zxvf machbase-ent-new.official-LINUX-X86-64-release.tgz -C $MACH_DEPLOYER_HOME
```

Deployer 시작

```
machdeployeradmin --startup
```

Package 등록

Broker/Warehouse 업그레이드를 위한 작업으로, Package 를 Coordinator 에 등록해서 업그레이드를 진행한다.

> lightweight package 로 등록하는 것이 좋다.

먼저, Package 를 `$MACH_COORDINATOR_HOME` 이 위치한 host 에 옮긴다.
그 다음, 아래 명령으로 패키지를 추가한다.

```
machcoordinatoradmin --add-package=new_package --file-name=./machbase-ent-new.official-LINUX-X86-64-release-lightwe
```

옵션	설명
--add-package	추가할 패키지의 이름을 지정한다.
--file-name	추가할 패키지 파일의 경로를 지정한다. ⓘ 이미 같은 파일 이름의 package 가 추가되어 있다면 에러가 발생하므로, 파일 이름을 확인해야 한다.

Broker/Warehouse 업그레이드

Coordinator 에서 다음 명령을 수행한다.

Node 종료

```
machcoordinatoradmin --shutdown-node=localhost:5656
```

Node 업그레이드

```
machcoordinatoradmin --upgrade-node=localhost:5656 --package-name=new_package
```

옵션	설명
--upgrade-node	업그레이드 대상 Node 이름을 입력한다.
--package-name	업그레이드할 Package 이름을 입력한다.

Node 종료 없이 Node 업그레이드를 수행하면, 자동으로 Node 를 종료시키고 Node 업그레이드를 수행한다.

하지만, 안정성을 위해서 Node 종료를 명시적으로 수행하도록 한다.

Node 구동

```
machcoordinatoradmin --startup-node=localhost:5656
```

Snapshot Failover

Machbase 6.5 Cluster Edition에 Snapshot Failover 기능이 추가되었다.

Snapshot Failover는 DBMS가 정상적인 상황일 때 Snapshot 을 기록해두고 특정 Warehouse의 장애 발생 시 정상인 Snapshot은 제외하고 문제가 발생한 부분에 대해서만 Failover를 수행함으로써 빠른 복구를 제공하는 기능이다.

Snapshot 기본 개념

Cluster Edition의 각 Group별로 Group 내 존재하는 Warehouse들 사이의 정상 데이터의 위치를 기록하는 개념이다.

Group 내의 Warehouse 에 생성된 Snapshot 이전의 데이터는 모두 정상 상태의 Data임을 보장하며 각 Group 별로 각각의 Snapshot을 기록한다.

Snapshot Failover 동작 방식

특정 Warehouse에 문제가 발생했을 경우 이 Warehouse는 Scrapped 상태로 데이터 복구가 필요한 상황이 된다.

Snapshot Recovery를 수행하게 되면 문제가 발생한 Warehouse에서의 정상 Snapshot을 기준으로 Snapshot 이후의 Data는 Clear하고 같은 Group 내 정상 상태의 Warehouse의 기준 Snapshot 이후 Data를 문제가 발생한 Warehouse로 Replication해서 복구를 완료한다.

Snapshot 자동 수행

Default로 Snapshot 자동 수행이 Enable 되어 있으며 Snapshot을 수행하는 주기는 60초로 설정 되어 있으며 Clustser에 Warehouse Group이 여러 개일 경우 Snapshot 주기마다 하나의 Group만 순서대로 Snapshot을 수행한다.

수행 주기를 0으로 설정하면 Snapshot 자동 수행이 Disable된다.

Snapshot 주기 설정은 명령어를 실행하면 즉시 반영된다.

```
# Snapshot 주기 설정
machcoordinatoradmin --snapshot-interval=[sec]

# 현재 Snapshot 주기 확인
machcoordinatoradmin --configuration
```

Snapshot 수동 수행

machcoordinatoradmin tool을 이용해 group_name을 지정하고 수동으로 Snapshot을 수행한다.

group_name은 group1, group2와 같이 미리 설정되어 있다.

Cluster에 Group이 여러 개일 경우 전체 Snapshot을 찍기 위해서는 모든 각각의 Group에 Snapshot을 수행해줘야 한다.

```
# group_name에 대한 Snapshot 수동 수행
machcoordinatoradmin --exec-snapshot --group='group_name'
```

Scrapped node 를 Snapshot 기반으로 복구

Scrapped node 가 발생한 경우 아래와 같이 복구한다.

```
# 해당 group 을 readonly 로 변경
```

```
# 이후의 단계에서 group이 normal로 변경되는 것을 방지
machcoordinatoradmin --set-group-state=readonly --group=[groupname]

# snapshot 기반으로 복구
machcoordinatoradmin --snapshot-recover=[nodename]

# replication을 통해 snapshot 이후의 최신 data를 복제
# replication이 끝나면 warehouse 의 상태는 normal로 자동 변경
machcoordinatoradmin --exec-sync=[nodename]

# group 상태를 normal 로 변경
machcoordinatoradmin --set-group-state=normal --group=[groupname]
```

Scrapped node의 Snapshot 기반 복구 과정

Snapshot으로 Scrapped node를 복구시 아래와 같은 과정이 수행된다.

```
/*최초 cluster 상태*/
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30420 | group1          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------

/*group1의 warehouse 0이 죽었을 때*/
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | readonly        | scrapped      | **unknown**   | -----------
| warehouse   | localhost:30420 | group1          | readonly        | normal        | normal        | -----------
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------

# 해당 group 을 readonly 로 변경
machcoordinatoradmin --set-group-state=readonly --group=[groupname]

kellen@kellen-ku:~$ machcoordinatoradmin --set-group-state=readonly --group=group1
-------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - 321a012d05.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
-------------------------------------------------------------------------
Group Name: group1
Flag      : 1
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
```

```
| warehouse   | localhost:30410 | group1          | readonly         | scrapped      | **unknown**   | -----------
| warehouse   | localhost:30420 | group1          | readonly         | normal        | normal        | -----------
| warehouse   | localhost:30510 | group2          | normal           | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal           | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+

#죽은 Warehouse를 다시 startup 수행한다
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State   |
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | readonly         | scrapped      | scrapped      | -----------
| warehouse   | localhost:30420 | group1          | readonly         | normal        | normal        | -----------
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+

# snapshot 기반으로 복구
machcoordinatoradmin --snapshot-recover=[nodename]

kellen@kellen-ku:~$ machcoordinatoradmin --snapshot-recover=localhost:30410
-------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - 321a012d05.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
-------------------------------------------------------------------------
Node-Name: localhost:30410
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State   |
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | readonly         | scrapped      | scrapped      | -----------
| warehouse   | localhost:30420 | group1          | readonly         | normal        | normal        | -----------
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+

# replication을 통해 snapshot 이후의 최신 data를 복제
machcoordinatoradmin --exec-sync=[nodename]

kellen@kellen-ku:~$ machcoordinatoradmin --exec-sync=localhost:30410
-------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - 321a012d05.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
-------------------------------------------------------------------------
Node-Name: localhost:30410
Source:
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State   |
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------+
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | readonly         | scrapped      | scrapped      | stopped
| warehouse   | localhost:30420 | group1          | readonly         | normal        | normal        | stopped
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
```

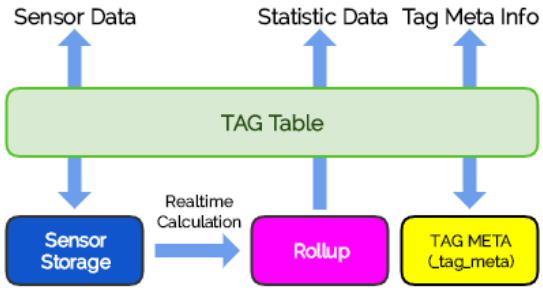
```
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | readonly         | sync-standby  | sync-standby  | running
| warehouse   | localhost:30420 | group1          | readonly         | sync-active   | sync-active   | running
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | readonly         | normal        | normal        | stopped
| warehouse   | localhost:30420 | group1          | readonly         | normal        | normal        | stopped
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------

# group 상태를 normal 로 변경
machcoordinatoradmin --set-group-state=normal --group=[groupname]

kellen@kellen-ku:~$ machcoordinatoradmin --set-group-state=normal --group=group1
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - 321a012d05.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Group Name: group1
Flag      : 0
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
|  Node Type  |    Node Name    |   Group Name    |   Group State   |    Desired & Actual State     |  RP State
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
| coordinator | localhost:30110 | Coordinator     | normal          | primary       | primary       | -----------
| coordinator | localhost:30120 | Coordinator     | normal          | normal        | normal        | -----------
| deployer    | localhost:30210 | Deployer        | normal          | normal        | normal        | -----------
| broker      | localhost:30310 | Broker          | normal          | leader        | leader        | -----------
| broker      | localhost:30320 | Broker          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30410 | group1          | normal          | normal        | normal        | stopped
| warehouse   | localhost:30420 | group1          | normal          | normal        | normal        | stopped
| warehouse   | localhost:30510 | group2          | normal          | normal        | normal        | -----------
| warehouse   | localhost:30520 | group2          | normal          | normal        | normal        | -----------
+-------------+-----------------+-----------------+-----------------+-------------------------------+-------------
```

Snapshot 관련 Property

Property	설명	설정 위치
GROUP_SNAPSHOT_TIMEOUT_SEC	Snapshot 실행 시의 timeout 시간을 결정 Default : 60 (초) 최소 값 : 0 (무한 대기) 최대 값 : uint32_max (초)	Coordinator, Broker, Warehouse 각각의 machbase.conf 파일 내 작성

태그 테이블 (Tag Table)

개념



태그 테이블은 센서 데이터 처리를 위한 데이터 저장소 및 관련 부가 정보 관리를 담당한다.
TAG 테이블에서는 아래의 세가지 개념적인 데이터 처리 공간을 제공하며, 세부적인 설명은 다음과 같다.

Sensor Partition

이는 TAG 테이블이 생성될 때 사용자가 정의한 스키마를 기준으로 저장되는 내부 센서 데이터 테이블이다.
이 데이터는 TAG 테이블에 대한 SELECT 질의를 통해서 추출이 가능하다.
고속으로 초당 수만건에서 수십 만건의 센서 데이터를 로딩할 수 있다.
고속으로 초당 수만건의 센서 데이터를 시간 범위의 조건으로 검색할 수 있다.
실시간 압축을 통해 오랜 기간 동안의 센서 데이터를 저장할 수 있다.
시간 순으로 오래된 센서 데이터에 대한 순차적인 삭제가 가능하다.
저장되는 사용자의 센서 데이터는 기본적으로 시계열 데이터로서 해당 태그의 이름과 시간 그리고 64비트 실수 값을 갖는 특정한 데이터형이다.

태그이름(사용자 지정 길이 스트링)	시간(64 비트)	실수 값(64 비트)	(사용자 확장 컬럼들..)

ROLLUP Partition

이것은 Sensor storage에 저장된 센서 데이터를 바탕으로 자동으로 통계 데이터를 생성하는 내부 테이블이다.
이는 수일 혹은 수년의 긴 시간동안의 통계 데이터를 수초내의 실시간으로 얻을 목적으로 개발되었다.
하나의 Sensor storage 당 Hour, Minute, Second 단위로 3개의 내부 ROLLUP 테이블이 별도로 생성된다.
이 테이블에서는 MIN, MAX, AVG, SUM, COUNT 다섯개의 통계 데이터를 지원한다.
이 ROLLUP 결과값을 얻기 위해서는 TAG 테이블에 대한 SELECT 질의의 Hint 지정을 통해서 가능하다.

META 테이블

이것은 Sensor Storage에 저장될 태그의 이름 및 부가 메타 정보를 저장하는 별도의 테이블이다.
사용자는 이 테이블에 대해 명시적으로 INSERT를 통해 태그의 메타 정보를 등록할 수 있다.
또한, 사용자는 이 테이블에 대한 수정과 삭제도 가능하다.

작업 방법

- 태그 테이블 생성 및 삭제
- 태그 메타 (태그 이름) 관리
- 태그 데이터의 조작
- 롤업 테이블의 생성 및 조회
- 태그 테이블 활용 샘플 예제
- 태그 테이블 인덱스 생성 및 관리

태그 테이블 생성 및 삭제

목차

사용자는 테이블 타입으로 TAG라고 명시적으로 지정하여야 하며, 이후에 이 테이블을 조작함으로써 센서 데이터를 다양한 형태로 활용할 수 있다.

과거 버전과 달리 테이블 이름을 TAG로 하지 않아도 되며, 자유롭게 지정 가능하다.

데이터베이스가 처음 설치되었을 때는 TAG 테이블이 없다는 점에 유의하라.

TAG 테이블은 기본적으로 사용자의 센서 데이터를 저장하기 위한 목적이므로, 아래의 세가지 필수 항목은 반드시 포함되어야 한다.

- 이름
- 입력 시간
- 값

그러나, 마크베이스의 TAG 테이블은 위의 세가지 뿐만 아니라 부가 컬럼의 입력도 허용하기 때문에 위의 필수 컬럼을 위한 키워드를 동반한다.

7.5 버전부터는 태그 값에 SUMMARIZED 키워드는 선택 사항이다.

- **태그 이름** : PRIMARY KEY
- **태그 입력 시간** : BASETIME

그리고, 이 태그 이름은 다음 장에 설명될 태그 메타 정보로서 활용된다.

태그 테이블 생성

가장 간단한 태그 테이블은 아래와 같이 생성된다.

```
Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME, value DOUBLE);
[ERR-02253: Mandatory column definition (PRIMARY KEY / BASETIME) is missing.]
==> 위와 같이 키워드를 넣지 않으면, 위와 같은 에러가 발생한다.

Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE SUMMARIZED);
Executed successfully.
==> 통계 정보 활용을 위해서는 태그 값에 SUMMARIZED 키워드를 추가해야 한다.

Mach> desc tag;
[ COLUMN ]
---------------------------------------------------------------
NAME      TYPE        LENGTH
---------------------------------------------------------------
NAME      varchar       20
TIME      datetime      31
VALUE     double        17

Mach> CREATE TAG TABLE other_tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE);
Executed successfully.

Mach> desc other_tag;
[ COLUMN ]
---------------------------------------------------------------
NAME      TYPE        LENGTH
---------------------------------------------------------------
NAME      varchar       20
TIME      datetime      31
VALUE     double        17
```

즉, TAG 라는 이름을 가진 테이블이 생성되었다. 성능 향상을 위해 4개의 파티션으로 나뉘어진 내부 테이블이 생성된다.

추가 센서 컬럼

실제로 TAG 테이블을 활용할 때 단지 3개의 컬럼만으로는 주어진 문제를 해결하기 힘든 경우가 있다.

특히, 입력되는 센서 데이터의 정보가 이름과 시간, 값 뿐만 아니라 특정 그룹이나 인터넷 주소의 경우도 있기 때문에 아래와 같이 추가할 수 있다.

```
Mach> create tag table TAG (name varchar(20) primary key, time datetime basetime, value double, grpid short, myip
Executed successfully.
```

```
Mach> desc tag;
[ COLUMN ]
---------------------------------------------------------------
NAME            TYPE        LENGTH
---------------------------------------------------------------
NAME            varchar         20
TIME            datetime        31
VALUE           double          17
GRPID           short            6       <=== 추가됨
MYIP            ipv4            15      <=== 추가됨
```

그러나, 5.5를 포함한 구버전에서는 VARCHAR 타입의 값은 부가 컬럼에 들어갈 수 없다는 점에 유의하자.

```
Mach> create tag table TAG (name varchar(20) primary key, time datetime basetime, value double summarized, myname v
[ERR-01851: Variable length columns are not allowed in tag table.]
```

문자열 타입의 경우에는 위와 같이 에러가 발생한다. 5.6 이후의 버전에서는TAG 테이블의 추가 컬럼에서도 VARCHAR를 지원한다.

추가 메타데이터 컬럼

TAG 테이블에는 센서 컬럼 추가만 가능한 것이 아니라, 각 태그 이름에 종속된 정보를 함께 입력할 수 있다.

이 정보는 센서 데이터에 중복 저장할 필요가 없는 정보이기 때문에, 효율적으로 관리하기 위한 별도의 컬럼 정의 구문인 `METADATA (...)` 을 추가해야 한다.

```
Mach> create tag table TAG (name varchar(20) primary key, time datetime basetime, value double)
   2  metadata (room_no integer, tag_description varchar(100));
```

여기서 room_no, tag_description 은 name 에 종속된 정보이다. 예를 들면, 이런 정보를 입력해 둘 수 있다.

name	room_no	tag_description
temp_001	1	It reads current temperature as Celsius
humid_001	1	It reads current humidity as percentage

입력한 이후에는, TAG 테이블에서 SELECT 를 통해 같이 조회할 수 있다.

```
Mach> SELECT name, time, value, tag_description FROM tag LIMIT 1;
name                time                            value
-------------------------------------------------------------------------------------
tag_description
-------------------------------------------------------------------------------------
temp_001            2019-03-01 09:52:17 000:000:000 25.3
It reads current temperature as Celsius
```

테이블 프로퍼티 지정

태그 테이블 생성 시, 아래 3가지 프로퍼티를 지정할 수 있다.

이름	설명	값
TAG_PARTITION_COUNT	메모리 및 CPU 사용량을 조절하기 위해 파티션 개수를 지정할 수 있다.	• Default: 4 • Min: 1 • Max: 1024
TAG_DATA_PART_SIZE	파티션 별 메모리 및 CPU 사용량을 조절하기 위해 데이터 크기를 지정할 수 있다. BYTE 단위로 지정하며, MB 단위로 ALIGN 된다.	• Default: 16MB (16 * 1024 * 1024) • Min: 1MB (1024 * 1024) • Max: 1GB (1024 * 1024 * 1024)
TAG_STAT_ENABLE	TAG ID별 통계 정보를 저장하는 기능의 활성화 여부를 지정할 수 있다.	• Default : 1 • Min: 0 (disable) • Max: 1 (enable)

```
Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE) TAG_PARTITION_COUNT
Executed successfully.

Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE) TAG_DATA_PART_SIZE=
Executed successfully.

Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE) TAG_STAT_ENABLE=0;
Executed successfully.

Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE) TAG_PARTITION_COUNT
Executed successfully.
```

태그 테이블 삭제

만일 생성된 태그 테이블을 다시 만들거나, 필요가 없어져 디스크 공간을 확보해야 하는 경우에는 다음과 같은 DROP 명령어를 통해 삭제할 수 있다.

TAG 테이블에 관련된 모든 자료, 즉 태그 데이터, 메타데이터 테이블이 모두 삭제되므로 유의해야 한다.

```
Mach> DROP TABLE tag;
Dropped successfully.

Mach> DESC tag;
tag does not exist.
```

태그 메타 (태그 이름) 관리

목차

태그 메타의 개념

태그 메타는 마크베이스에서 저장될 임의의 태그가 가질 이름과 부가 정보를 나타낸다.

즉, 특정 장비에 존재하는 태그가 3개라고 한다면, 이 태그를 나타내는 임의의 이름과 관련 부가 정보가 필요한데, 이것을 모두 태그의 메타 정보라고 한다.

이 태그 메타는 최소한 이름이 존재할 수 있으며, 부가적으로 필요하다면 해당 장비에 맞는 다양한 종류의 데이터 타입을 지정할 수 있도록 되어 있다.

이름만으로 이뤄진 태그 메타

태그 메타의 생성

아래는 가장 기본적인 태그 메타가 생성되는 TAG 테이블의 생성 명령어이다.

```
create tag table TAG (name varchar(20) primary key, time datetime basetime, val
Mach> desc tag;
[ COLUMN ]
---------------------------------------------------------------
NAME                          TYPE               LENGTH
---------------------------------------------------------------
NAME                          varchar            20
TIME                          datetime           31
VALUE                         double             17
```

위는 기본적인 TAG 테이블을 생성한 것이며, 태그 메타에 대한 별도의 정보는 보이지 않는다.

이 경우 태그 메타는 VARCHAR(20)의 기본적인 이름만을 가진다.

태그 메타의 입력

이제 TAG1 이라는 이름을 갖는 하나의 태그 정보를 입력해 보자.

```
Mach> insert into tag metadata values ('TAG_0001');
1 row(s) inserted.
```

위의 질의를 통해서 TAG_0001 이라는 이름을 갖는 하나의 태그를 생성하였다.

태그 메타의 출력

마크베이스에서는 입력된 태그 메타의 정보를 확인하기 위한 특별한 테이블인 _tag_meta 를 제공한다.

따라서, 사용자는 다음과 같은 질의를 통해서 마크베이스에 입력된 모든 태그의 정보를 확인할 수 있다.

```
Mach> select * from _tag_meta;
ID                  NAME
-----------------------------------------------
1                   TAG_0001
[1] row(s) selected.
```

위의 질의를 통해서 TAG_0001 이라는 NAME을 갖는 하나의 태그를 생성하였다.

ID는 내부적으로 관리되는 값으로서 자동으로 부여된다.

태그 메타의 수정

마크베이스는 입력된 태그 메타 정보를 수정할 수 있도록 해 주는데, 다음과 같이 이름이 수정 가능하다.

```
Mach> update tag metadata set name = 'NEW_0001' where NAME = 'TAG_0001';
1 row(s) updated.

Mach> select * from _tag_meta;
```

```
ID                  NAME
------------------------------------------------
1                   NEW_0001
[1] row(s) selected.
```

위와 같이 이름이 TAG_0001에서 NEW_0001로 수정된 것을 확인할 수 있다.

태그 메타의 삭제

아래와 같이 실제 태그 메타의 정보를 삭제할 수 있다.

```
Mach> delete from tag metadata where name = 'NEW_0001';
1 row(s) deleted.

Mach> select * from _tag_meta;
ID                  NAME
------------------------------------------------
[0] row(s) selected.
```

주의할 점은 태그 테이블에 실제 데이터가 해당 태그 메타를 참조하지 않을 때 태그 메타 삭제가 가능하다.

추가 정보를 갖는 태그 메타

태그 메타의 생성

아래는 태그 메타의 정보에 16비트 정수와 시간 그리고, IPv4 의 정보를 부가적으로 더 추가해서 만들어 본다.

주의할 점은 일단 생성된 태그 메타에 대해 값은 수정할 수 있지만, 그 구조는 수정할 수 없다는 것이다.

```
create tag table TAG (name varchar(20) primary key, time datetime basetime, value double summarized)
metadata (type short, create_date datetime, srcip ipv4) ;

Mach> desc tag;
[ COLUMN ]
-------------------------------------------------------------------
NAME                        TYPE                LENGTH
-------------------------------------------------------------------
NAME                        varchar             20
TIME                        datetime            31
VALUE                       double              17
[ META-COLUMN ]
-------------------------------------------------------------------
NAME                        TYPE                LENGTH
-------------------------------------------------------------------
TYPE                        short               6
CREATE_DATE                 datetime            31
SRCIP                       ipv4                15
```

태그 메타의 입력

이름 뿐만 아니라 부가 정보가 있는 상태에서 아래와 같이 입력해서 정보를 확인할 수 있다.

```
Mach> insert into tag metadata(name) values ('TAG_0001');
1 row(s) inserted.

Mach> select * from _tag_meta;
ID                  NAME                TYPE        CREATE_DATE                 SRCIP
-------------------------------------------------------------------------------------------------------------------
1                   TAG_0001            NULL        NULL                        NULL
[1] row(s) selected.
```

위와 같이 NAME 외 다른 컬럼에는 NULL이 입력된 것을 알 수 있다.

이제 부가 정보를 아래와 같이 더 넣어 보자.

```
Mach> insert into tag metadata values ('TAG_0002', 99, '2010-01-01', '1.1.1.1');
1 row(s) inserted.
```

```
Mach> select * from _tag_meta;
ID                  NAME                TYPE        CREATE_DATE                    SRCIP
-------------------------------------------------------------------------------------------------------
1                   TAG_0001            NULL        NULL                           NULL
2                   TAG_0002            99          2010-01-01 00:00:00 000:000:000 1.1.1.1
[2] row(s) selected.
```

부가 정보를 위와 같이 넣었고, 각 태그 메타가 주어진 풍부한 정보를 가질 수 있게 되었다.

태그 메타의 수정

이제 TAG_0001의 타입을 NULL에서 11로 수정해 보자.

```
Mach> update tag metadata set type = 11 where name = 'TAG_0001';
1 row(s) updated.

Mach> select * from _tag_meta;
ID                  NAME                TYPE        CREATE_DATE                    SRCIP
-------------------------------------------------------------------------------------------------------
2                   TAG_0002            99          2010-01-01 00:00:00 000:000:000 1.1.1.1
1                   TAG_0001            11          NULL                           NULL
[2] row(s) selected.
```

위와 같이 수정되었다.

즉, UPDATE 구문을 통해 모든 필드의 값을 수정할 수 있다.

단, 반드시 WHERE 절에 NAME이 지정되어야 하는 것은 공통적인 제약 사항이다.

RESTful API를 통한 태그 메타 조회

모든 태그 리스트 얻기

아래는 마크베이스 포함된 모든 태그의 리스트를 얻는 예제이다.

```
Host:~$ curl  -G  "http://192.168.0.148:5001/machiot-rest-api/tags/list"
{"ErrorCode": 0,
 "ErrorMessage": "",
 "Data": [{"NAME": "TAG_0001"},
          {"NAME": "TAG_0002"}]}
Host:~$
```

특정 태그의 시간 범위 얻기

아래는 원하는 태그가 가지고 있는 데이터의 최소 및 최대 시간 범위를 얻는 예제이다.

이기능은 특정 태그의 차트를 그릴 때 매우 유용하다.

문법

```
{MWA URL}/machiot-rest-api/tags/range/  # Time Range of whole DB
{MWA URL}/machiot-rest-api/tags/range/{TagName}  # Time Range of a specific Tag
```

전체 시간 범위

```
Host:~$ curl  -G  "http://192.168.0.148:5001/machiot-rest-api/tags/range/"
{"ErrorCode": 0,
 "ErrorMessage": "",
 "Data": [{"MAX": "2018-02-10 10:00:00 000:000:000", "MIN": "2018-01-01 01:00:00 000:000:000"}]}
```

특정 태그의 시간 범위

```
Host:~$ curl  -G  "http://192.168.0.148:5001/machiot-rest-api/tags/range/TAG_0001"
```

```
{"ErrorCode": 0, "ErrorMessage": "", "Data": [{"MAX": "2018-01-10 10:00:00 000:000:000", "MIN": "2018-01-01 01:00:0
Host:~$
Host:~$ curl  -G  "http://192.168.0.148:5001/machiot-rest-api/tags/range/TAG_0002"
{"ErrorCode": 0, "ErrorMessage": "", "Data": [{"MAX": "2018-02-10 10:00:00 000:000:000", "MIN": "2018-02-01 01:00:0
```

태그 데이터의 조작

태그 데이터의 입력

태그 데이터를 입력하기 위해서는 아래와 같은 다양한 방법을 활용할 수 있다.

목차

INSERT 구문을 통해 입력하기

가장 간단한 방법으로 아래와 같이 INSERT 구문을 통해 입력할 수 있다.

간단하게 테스트 용도로 할 수 있는 방법이고, 만일 대량의 데이터를 빨리 넣고자 할 경우에는 다른 방법을 활용한다.

```
Mach> create tag table TAG (name varchar(20) primary key, time datetime basetim
Executed successfully.

Mach>  insert into tag metadata values ('TAG_0001');
1 row(s) inserted.

Mach> insert into tag values('TAG_0001', now, 0);
1 row(s) inserted.

Mach> insert into tag values('TAG_0001', now, 1);
1 row(s) inserted.

Mach> insert into tag values('TAG_0001', now, 2);
1 row(s) inserted.

Mach> select * from tag where name = 'TAG_0001';
NAME                 TIME                            VALUE
------------------------------------------------------------------------------
TAG_0001             2018-12-19 17:41:37 806:901:728 0
TAG_0001             2018-12-19 17:41:42 327:839:368 1
TAG_0001             2018-12-19 17:41:43 812:782:202 2
[3] row(s) selected.
```

위와 같이 3개의 TAG 값을 현재의 시간으로 넣어 보았다.

CSV 파일을 통해 한꺼번에 로딩하기

마크베이스는 csvimport 라는 도구를 통해서 CSV 파일 대량으로 로딩할 수 있도록 해 준다.

더 자세한 내용은 실제 예제를 통해서 파악할 수 있으며, 아래에 간단하게 기술한다.

CSV 파일 형태 (data.csv)

```
TAG_0001, 2009-01-28 07:03:34 0:000:000, -41.98
TAG_0001, 2009-01-28 07:03:34 1:000:000, -46.50
TAG_0001, 2009-01-28 07:03:34 2:000:000, -36.16
....
```

위와 같이 <태그명, 시간, 값> 으로 구성된 csv 파일 준비한다.

물론, 태그명 TAG_0001이 존재해야 한다.

로딩 프로그램 csvimport 사용

```
csvimport -t TAG -d data.csv -F "time YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn" -l error.log
```

TAG라는 테이블에 data.csv를 로딩한다.

그리고, -F 옵션은 data.csv에 저장된 시간 포맷을 지정하는 것인데, 현재 파일은 나노 단위까지 값을 넣을 수 있도록 되어 있다.

또한, -l error.log 는 입력시 발생한 에러에 대해 별도의 파일로 기록하는 것이다.

RESTful API를 통해 입력하기

RESTful API의 더 자세한 사용법은 다음의 활용 예제를 참고하도록 한다.

입력 API 문법

마크베이스는 다음과 같이 RESTful API를 제공한다.

```
{
 "values":[
     [TAG_NAME, TAG_TIME, VALUE],  # 태그명,시간,값을 입력한다. TAG 형태에 따라 부가 컬럼 추가 필요
     [ .... ]....
 ],
 "date_format":"Date Format"       # date_format은 생략시 'YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn' 로 설정된다.
}
```

정의된 TAG 스키마의 컬럼 갯수만큼의 값을 위의 구조와 일치되도록 요청한다.

SDK를 통해 데이터 입력하기

마크베이스는 아래와 같은 다양한 언어의 표준 개발 툴을 제공하고 있다.

- C/C++ library
- JAVA library
- Python library
- C# library

이러한 라이브러리를 통해서 사용자는 자신의 환경에 따라 다양한 형태의 응용 프로그램을 작성하여 마크베이스에 대한 데이터 입력이 가능하다.

태그 데이터의 추출

마크베이스는 고속의 태그 데이터 추출 성능을 제공하며, 특히 특정 태그의 시간 범위에 대한 탁월한 성능을 제공한다.

목차

샘플 스키마

이후의 샘플은 아래와 같이 TAG 테이블이 생성되고, 두개의 태그를 생성하였다.

각 태그에 대해 각각 2018년 1월 1일부터 2월 10일까지의 데이터를 입력하였다.

```
create tag table TAG (name varchar(20) primary key, time datetime basetime, val

insert into tag metadata values ('TAG_0001');
insert into tag metadata values ('TAG_0002');

insert into tag values('TAG_0001', '2018-01-01 01:00:00 000:000:000', 1);
insert into tag values('TAG_0001', '2018-01-02 02:00:00 000:000:000', 2);
insert into tag values('TAG_0001', '2018-01-03 03:00:00 000:000:000', 3);
insert into tag values('TAG_0001', '2018-01-04 04:00:00 000:000:000', 4);
insert into tag values('TAG_0001', '2018-01-05 05:00:00 000:000:000', 5);
insert into tag values('TAG_0001', '2018-01-06 06:00:00 000:000:000', 6);
insert into tag values('TAG_0001', '2018-01-07 07:00:00 000:000:000', 7);
insert into tag values('TAG_0001', '2018-01-08 08:00:00 000:000:000', 8);
insert into tag values('TAG_0001', '2018-01-09 09:00:00 000:000:000', 9);
insert into tag values('TAG_0001', '2018-01-10 10:00:00 000:000:000', 10);

insert into tag values('TAG_0002', '2018-02-01 01:00:00 000:000:000', 11);
insert into tag values('TAG_0002', '2018-02-02 02:00:00 000:000:000', 12);
insert into tag values('TAG_0002', '2018-02-03 03:00:00 000:000:000', 13);
insert into tag values('TAG_0002', '2018-02-04 04:00:00 000:000:000', 14);
insert into tag values('TAG_0002', '2018-02-05 05:00:00 000:000:000', 15);
insert into tag values('TAG_0002', '2018-02-06 06:00:00 000:000:000', 16);
insert into tag values('TAG_0002', '2018-02-07 07:00:00 000:000:000', 17);
insert into tag values('TAG_0002', '2018-02-08 08:00:00 000:000:000', 18);
insert into tag values('TAG_0002', '2018-02-09 09:00:00 000:000:000', 19);
insert into tag values('TAG_0002', '2018-02-10 10:00:00 000:000:000', 20);
```

전체 TAG 데이터 추출

```
Mach> select * from tag;
NAME TIME VALUE
--------------------------------------------------------------------------------
TAG_0001 2018-01-01 01:00:00 000:000:000 1
TAG_0001 2018-01-02 02:00:00 000:000:000 2
TAG_0001 2018-01-03 03:00:00 000:000:000 3
TAG_0001 2018-01-04 04:00:00 000:000:000 4
TAG_0001 2018-01-05 05:00:00 000:000:000 5
TAG_0001 2018-01-06 06:00:00 000:000:000 6
TAG_0001 2018-01-07 07:00:00 000:000:000 7
TAG_0001 2018-01-08 08:00:00 000:000:000 8
TAG_0001 2018-01-09 09:00:00 000:000:000 9
TAG_0001 2018-01-10 10:00:00 000:000:000 10
TAG_0002 2018-02-01 01:00:00 000:000:000 11
TAG_0002 2018-02-02 02:00:00 000:000:000 12
TAG_0002 2018-02-03 03:00:00 000:000:000 13
TAG_0002 2018-02-04 04:00:00 000:000:000 14
TAG_0002 2018-02-05 05:00:00 000:000:000 15
TAG_0002 2018-02-06 06:00:00 000:000:000 16
TAG_0002 2018-02-07 07:00:00 000:000:000 17
TAG_0002 2018-02-08 08:00:00 000:000:000 18
TAG_0002 2018-02-09 09:00:00 000:000:000 19
```

```
TAG_0002 2018-02-10 10:00:00 000:000:000 20
[20] row(s) selected.
```

위와 같이 특별한 조건이 없으면, 각 시간 순으로 정렬된 태그별로 데이터를 추출할 수 있다.

임의 TAG명에 대한 데이터 추출

아래는 TAG 이름이 TAG_0002 인 데이터를 출력하는 예제이다. WHERE 절에 주어진 name에 대한 조건을 설정한다.

```
Mach> select * from tag where name='TAG_0002';
NAME                 TIME                           VALUE
------------------------------------------------------------------------------------
TAG_0002             2018-02-01 01:00:00 000:000:000 11
TAG_0002             2018-02-02 02:00:00 000:000:000 12
TAG_0002             2018-02-03 03:00:00 000:000:000 13
TAG_0002             2018-02-04 04:00:00 000:000:000 14
TAG_0002             2018-02-05 05:00:00 000:000:000 15
TAG_0002             2018-02-06 06:00:00 000:000:000 16
TAG_0002             2018-02-07 07:00:00 000:000:000 17
TAG_0002             2018-02-08 08:00:00 000:000:000 18
TAG_0002             2018-02-09 09:00:00 000:000:000 19
TAG_0002             2018-02-10 10:00:00 000:000:000 20
[10] row(s) selected.
```

시간 범위에 대한 쿼리

아래는 TAG_0002에 대한 시간 범위를 주고, 데이터를 받아오는 쿼리이다.

between 절을 활용해서 시간 범위를 주는 것이 일반적인 방법이다. 물론, time을 < 혹은 > 기호로 시간 범위를 입력해도 같은 결과를 얻을 수 있다.

```
Mach> select * from tag where name = 'TAG_0002' and time between to_date('2018-02-01') and to_date('2018-02-05');
NAME                 TIME                           VALUE
------------------------------------------------------------------------------------
TAG_0002             2018-02-01 01:00:00 000:000:000 11
TAG_0002             2018-02-02 02:00:00 000:000:000 12
TAG_0002             2018-02-03 03:00:00 000:000:000 13
TAG_0002             2018-02-04 04:00:00 000:000:000 14
[4] row(s) selected.

Mach> select * from tag where name = 'TAG_0002' and time > to_date('2018-02-01') and time < to_date('2018-02-05');
NAME                 TIME                           VALUE
------------------------------------------------------------------------------------
TAG_0002             2018-02-01 01:00:00 000:000:000 11
TAG_0002             2018-02-02 02:00:00 000:000:000 12
TAG_0002             2018-02-03 03:00:00 000:000:000 13
TAG_0002             2018-02-04 04:00:00 000:000:000 14
[4] row(s) selected.
```

다중 태그에 대한 시간 범위 검색

아래는 2개 이상의 태그에 대해서 동일한 시간 범위 데이터를 검색하는 예제이다.

만일 한번의 질의 수행으로 다수의 태그에 대해 한꺼번에 빠른 결과를 받고 싶을 경우에는 아래와 같은 형태의 수행이 바람직하다.

```
Mach> select * from tag where name in ('TAG_0002', 'TAG_0001') and time between to_date('2018-01-05') and to_date(
NAME                 TIME                           VALUE
------------------------------------------------------------------------------------
TAG_0001             2018-01-05 05:00:00 000:000:000 5
TAG_0001             2018-01-06 06:00:00 000:000:000 6
TAG_0001             2018-01-07 07:00:00 000:000:000 7
TAG_0001             2018-01-08 08:00:00 000:000:000 8
TAG_0001             2018-01-09 09:00:00 000:000:000 9
```

```
TAG_0001              2018-01-10 10:00:00 000:000:000 10
TAG_0002              2018-02-01 01:00:00 000:000:000 11
TAG_0002              2018-02-02 02:00:00 000:000:000 12
TAG_0002              2018-02-03 03:00:00 000:000:000 13
TAG_0002              2018-02-04 04:00:00 000:000:000 14
[10] row(s) selected.
```

특정 값 이상의 태그만 출력하기

간단한 예제이긴 하지만, 태그 값에 대한 조건도 함께 아래와 같이 줄 수 있다.

TAG_0002의 값 중에 12보다 크고, 15보다 작을 것들에 대해 필터링을 수행했다.

```
Mach> select * from tag where name = 'TAG_0002' and value > 12 and value < 15 and time between to_date('2018-02-01
NAME                  TIME                          VALUE
--------------------------------------------------------------------------------
TAG_0002              2018-02-03 03:00:00 000:000:000 13
TAG_0002              2018-02-04 04:00:00 000:000:000 14
[2] row(s) selected.
```

특정 태그 아이디별 통계 정보 출력하기

태그 테이블을 생성하면 기본적으로 태그 테이블의 태그 아이디별 간단한 통계 정보를 수집하는 별도의 가상 테이블이 생성된다.

가상 테이블의 이름은 **v$**{태그테이블 이름}**_stat** 의 규칙을 따르게 된다.

해당 테이블을 사용하면 태그 테이블의 통계 정보를 빠르게 얻어올 수 있다.

통계 정보 대상 칼럼은 자동으로 3번째 칼럼이 지정된다.

```
Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE SUMMARIZED);
Executed successfully.

Mach> DESC v$tag_stat;
[ COLUMN ]
-------------------------------------------------------------------------------------------------------
NAME                                                    NULL?    TYPE                LENGTH
-------------------------------------------------------------------------------------------------------
NAME                                                             varchar             100
ROW_COUNT                                                        ulong               20
MIN_TIME                                                         datetime            31
MAX_TIME                                                         datetime            31
MIN_VALUE                                                        double              17
MIN_VALUE_TIME                                                   datetime            31
MAX_VALUE                                                        double              17
MAX_VALUE_TIME                                                   datetime            31
RECENT_ROW_TIME                                                  datetime            31
```

수집하는 통계 정보는 아래와 같다.

컬럼 이름	정보
NAME	태그 아이디의 이름
ROW_COUNT	Row 개수
MIN_TIME	해당 태그 아이디 Row 중 가장 작은 Basetime 컬럼 값
MAX_TIME	해당 태그 아이디 Row 중 가장 큰 Basetime 컬럼 값
MIN_VALUE	해당 태그 아이디 Row 중 가장 작은 Summarized 컬럼 값
MIN_VALUE_TIME	MIN_VALUE 값과 같이 입력된 Basetime 컬럼 값
MAX_VALUE	해당 태그 아이디 Row 중 가장 큰 Summarized 컬럼 값
MAX_VALUE_TIME	MAX_VALUE 값과 같이 입력된 Basetime 컬럼 값
RECENT_ROW_TIME	해당 태그 아이디 Row 중 가장 최근에 입력된 Basetime 컬럼 값

3번째 칼럼에 SUMMARIZED 키워드가 없으면, VALUE 관련 정보(MIN_VALUE, MAX_VALUE, MIN_VALUE_TIME, MAX_VALUE_TIME)는 저장하지 않는다.

조회 예시는 아래와 같다.

```
1. SUMMARIZED 칼럼이 존재하는 경우
Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE SUMMARIZED);
Executed successfully.

Mach> INSERT INTO tag VALUES('tag-0', TO_DATE('2021-08-12'), 10);
Mach> INSERT INTO tag VALUES('tag-0', TO_DATE('2021-08-13'), 10);
Mach> INSERT INTO tag VALUES('tag-0', TO_DATE('2021-08-14'), 20);
Mach> INSERT INTO tag VALUES('tag-0', TO_DATE('2021-08-11'), 5);
Mach> INSERT INTO tag VALUES('tag-1', TO_DATE('2022-08-12'), 100);
Mach> INSERT INTO tag VALUES('tag-1', TO_DATE('2022-08-11'), 200);
Mach> INSERT INTO tag VALUES('tag-1', TO_DATE('2022-08-10'), 50);

Mach> SELECT * FROM v$tag_stat;
NAME                                                                                                    ROW_COUNT            MIN_TIME
--------------------------------------------------------------------------------------------------------------------------------------
MIN_VALUE_TIME                  MAX_VALUE                   MAX_VALUE_TIME                  RECENT_ROW_TIME
--------------------------------------------------------------------------------------------------------------------------------------
tag-0                                                                                                   4                    2021-08-11 (
2021-08-11 00:00:00 000:000:000 20                          2021-08-14 00:00:00 000:000:000 2021-08-11 00:00:00 000
tag-1                                                                                                   3                    2022-08-10 (
2022-08-10 00:00:00 000:000:000 200                         2022-08-11 00:00:00 000:000:000 2022-08-10 00:00:00 000
[2] row(s) selected.

2. SUMMARIZED 칼럼이 존재하지 않는 경우
Mach> CREATE TAG TABLE other_tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE);
Executed successfully.

Mach> INSERT INTO other_tag VALUES('tag-0', TO_DATE('2021-08-12'), 10);
Mach> INSERT INTO other_tag VALUES('tag-0', TO_DATE('2021-08-13'), 10);
Mach> INSERT INTO other_tag VALUES('tag-0', TO_DATE('2021-08-14'), 20);
Mach> INSERT INTO other_tag VALUES('tag-0', TO_DATE('2021-08-11'), 5);
Mach> INSERT INTO other_tag VALUES('tag-1', TO_DATE('2022-08-12'), 100);
Mach> INSERT INTO other_tag VALUES('tag-1', TO_DATE('2022-08-11'), 200);
Mach> INSERT INTO other_tag VALUES('tag-1', TO_DATE('2022-08-10'), 50);

Mach> SELECT * FROM v$other_tag_stat;
NAME                                                                                                    ROW_COUNT            MIN_TIME
--------------------------------------------------------------------------------------------------------------------------------------
MIN_VALUE_TIME                  MAX_VALUE                   MAX_VALUE_TIME                  RECENT_ROW_TIME
--------------------------------------------------------------------------------------------------------------------------------------
tag-0                                                                                                   4                    2021-08-11 (
NULL                            NULL                        NULL                            2021-08-11 00:00:00 000
tag-1                                                                                                   3                    2022-08-10 (
NULL                            NULL                        NULL                            2022-08-10 00:00:00 000
[2] row(s) selected.
```

RESTful API를 통한 추출

RESTful API를 위한 준비 사항

반드시 MWA (Machbase Web Analyzer) 를 수행해서 웹 서비스가 가능한 상태로 만든 이후에 아래를 수행해야 한다.

MWA의 수행

```
$ MWAserver start
SERVER STARTED, PID : 27307
     Connection URL : http://192.168.0.148:5001
```

RESTful API 호출 규약

SELECT FORM

```
{MWA URL}/machiot-rest-api/datapoints/raw/{TagName}/{Start}/{End}/{Direction}/{Count}/{Offset}/
```

```
TagName    : Tag Name. 복수의 Tag 지원(,로 구분하여 사용)
Start, End : 기간, YYYY-MM-DD HH24:MI:SS 또는 YYYY-MM-DD 또는 YYYY-MM-DD HH24:MI:SS,mmm (mmm: millisecond, 생략시 sta
실제 스트링으로 지정할 때는 날짜와 시간 사이에 T를 넣어서 빈공간을 없애준다.
Direction  : 0(ascending), 추후 지원 (시간이 증가하는 방향)
Count      : LIMIT, 0이면 전체
Offset     : offset (기본값 = 0)
```

CURL을 통한 단일 태그 데이터 가져오기 샘플

아래와 같이 192.168.0.148에 설치된 마크베이스에 대한 호출을 수행하면, 해당 데이터를 웹으로 부터 가져올 수 있다.

Single Tag

```
$ curl -G "http://192.168.0.148:5001/machiot-rest-api/v1/datapoints/raw/TAG_0001/2018-01-01T00:00:00/2018-01-06T00

{"ErrorCode": 0,
 "ErrorMessage": "",
 "Data": [{"DataType": "DOUBLE",
 "ErrorCode": 0,
 "TagName": "TAG_0001",
 "CalculationMode": "raw",
 "Samples": [{"TimeStamp": "2018-01-01 01:00:00 000:000:000", "Value": 1.0, "Quality": 1},
             {"TimeStamp": "2018-01-02 02:00:00 000:000:000", "Value": 2.0, "Quality": 1},
             {"TimeStamp": "2018-01-03 03:00:00 000:000:000", "Value": 3.0, "Quality": 1},
             {"TimeStamp": "2018-01-04 04:00:00 000:000:000", "Value": 4.0, "Quality": 1},
             {"TimeStamp": "2018-01-05 05:00:00 000:000:000", "Value": 5.0, "Quality": 1}]}]
}
```

CURL을 통한 다중 태그 데이터 가져오기

아래는 두개의 태그에 대한 값을 가져오는 샘플 예제이다.

```
$ curl -G "http://192.168.0.148:5001/machiot-rest-api/datapoints/raw/TAG_0001,TAG_0002/2018-01-05T00:00:00/2018-02-
{"ErrorCode": 0,
 "ErrorMessage": "",
 "Data": [{"DataType": "DOUBLE",
           "ErrorCode": 0,
           "TagName": "TAG_0001,TAG_0002",
           "CalculationMode": "raw",
           "Samples": [{"TimeStamp": "2018-01-05 05:00:00 000:000:000", "Value": 5.0, "Quality": 1},
                       {"TimeStamp": "2018-01-06 06:00:00 000:000:000", "Value": 6.0, "Quality": 1},
                       {"TimeStamp": "2018-01-07 07:00:00 000:000:000", "Value": 7.0, "Quality": 1},
                       {"TimeStamp": "2018-01-08 08:00:00 000:000:000", "Value": 8.0, "Quality": 1},
                       {"TimeStamp": "2018-01-09 09:00:00 000:000:000", "Value": 9.0, "Quality": 1},
                       {"TimeStamp": "2018-01-10 10:00:00 000:000:000", "Value": 10.0, "Quality": 1},
                       {"TimeStamp": "2018-02-01 01:00:00 000:000:000", "Value": 11.0, "Quality": 1},
                       {"TimeStamp": "2018-02-02 02:00:00 000:000:000", "Value": 12.0, "Quality": 1},
                       {"TimeStamp": "2018-02-03 03:00:00 000:000:000", "Value": 13.0, "Quality": 1},
                       {"TimeStamp": "2018-02-04 04:00:00 000:000:000", "Value": 14.0, "Quality": 1}
]}]}
```

힌트(hint)를 이용한 검색 방향 지정하기

태그 테이블은 일반적으로 입력한 순서가 오래된 레코드부터 조회가 가능하다. 가장 최근에 입력한 레코드부터 조회하고 싶을 때에는 힌트를 이용해 조회 방향을 제어할 수 있다.

정방향 검색

기본값이며, /*+ SCAN_FORWARD(table_name) */ 힌트를 추가하여 조회가 가능하다.

```
Mach> SELECT * FROM tag WHERE t_name='TAG_99' LIMIT 10;
T_NAME               T_TIME                          T_VALUE
--------------------------------------------------------------------------------
TAG_99               2017-01-01 00:00:49 500:000:000 0
TAG_99               2017-01-01 00:01:39 500:000:000 1
TAG_99               2017-01-01 00:02:29 500:000:000 2
TAG_99               2017-01-01 00:03:19 500:000:000 3
TAG_99               2017-01-01 00:04:09 500:000:000 4
```

```
TAG_99                2017-01-01 00:04:59 500:000:000 5
TAG_99                2017-01-01 00:05:49 500:000:000 6
TAG_99                2017-01-01 00:06:39 500:000:000 7
TAG_99                2017-01-01 00:07:29 500:000:000 8
TAG_99                2017-01-01 00:08:19 500:000:000 9
[10] row(s) selected.
Elapsed time: 0.001

Mach> SELECT /*+ SCAN_FORWARD(tag) */  * FROM tag WHERE t_name='TAG_99' LIMIT 10;
T_NAME                T_TIME                        T_VALUE
--------------------------------------------------------------------------------
TAG_99                2017-01-01 00:00:49 500:000:000 0
TAG_99                2017-01-01 00:01:39 500:000:000 1
TAG_99                2017-01-01 00:02:29 500:000:000 2
TAG_99                2017-01-01 00:03:19 500:000:000 3
TAG_99                2017-01-01 00:04:09 500:000:000 4
TAG_99                2017-01-01 00:04:59 500:000:000 5
TAG_99                2017-01-01 00:05:49 500:000:000 6
TAG_99                2017-01-01 00:06:39 500:000:000 7
TAG_99                2017-01-01 00:07:29 500:000:000 8
TAG_99                2017-01-01 00:08:19 500:000:000 9
[10] row(s) selected.
Elapsed time: 0.001
Mach>
```

역방향 검색

/*+ SCAN_BACKWARD(table_name) */ 힌트를 추가하여 조회가 가능하다.

```
Mach> SELECT /*+ SCAN_BACKWARD(tag) */ * FROM tag WHERE t_name='TAG_99' LIMIT 10;
T_NAME                T_TIME                        T_VALUE
--------------------------------------------------------------------------------
TAG_99                2017-02-27 20:53:19 500:000:000 9
TAG_99                2017-02-27 20:52:29 500:000:000 8
TAG_99                2017-02-27 20:51:39 500:000:000 7
TAG_99                2017-02-27 20:50:49 500:000:000 6
TAG_99                2017-02-27 20:49:59 500:000:000 5
TAG_99                2017-02-27 20:49:09 500:000:000 4
TAG_99                2017-02-27 20:48:19 500:000:000 3
TAG_99                2017-02-27 20:47:29 500:000:000 2
TAG_99                2017-02-27 20:46:39 500:000:000 1
TAG_99                2017-02-27 20:45:49 500:000:000 0
[10] row(s) selected.
Elapsed time: 0.001
Mach>
```

기본 스캔 방향 프로퍼티로 설정

TABLE_SCAN_DIRECTION 프로퍼티로 SELECT 문에 힌트가 없을 때 태그 테이블의 스캔 방향을 설정할 수 있다.

태그 데이터의 삭제

태그 데이터 삭제 제약 사항

마크베이스는 태그 테이블 전체에 대해서 특정 시간 이전의 전체 데이터에 대한 삭제만을 지원한다.

불가능한 태그 데이터 삭제 조건

- 특정 태그 아이디 삭제
- 특정 시간 범위의 데이터를 삭제
- 특정 태그의 특정 시간 범위 데이터 삭제

가능한 태그 데이터 삭제 조건

- 특정 시간 이전의 전체 태그에 대한 삭제
- 전체 삭제

목차

DELETE 구문 수행

특정 시간 이전의 전체 태그에 대한 삭제

BEFORE 구문의 시간을 지정하면 그 시간 이전의 태그는 모두 삭제한다.

```
DELETE FROM TAG BEFORE TO_DATE('Time-string');
```

```
# 현재 데이터
Mach> select * from tag;
NAME TIME VALUE
--------------------------------------------------------------------------------
TAG_0001 2018-01-01 01:00:00 000:000:000 1
TAG_0001 2018-01-02 02:00:00 000:000:000 2
TAG_0001 2018-01-03 03:00:00 000:000:000 3
TAG_0001 2018-01-04 04:00:00 000:000:000 4
TAG_0001 2018-01-05 05:00:00 000:000:000 5
TAG_0001 2018-01-06 06:00:00 000:000:000 6
TAG_0001 2018-01-07 07:00:00 000:000:000 7
TAG_0001 2018-01-08 08:00:00 000:000:000 8
TAG_0001 2018-01-09 09:00:00 000:000:000 9
TAG_0001 2018-01-10 10:00:00 000:000:000 10
TAG_0002 2018-02-01 01:00:00 000:000:000 11
TAG_0002 2018-02-02 02:00:00 000:000:000 12
TAG_0002 2018-02-03 03:00:00 000:000:000 13
TAG_0002 2018-02-04 04:00:00 000:000:000 14
TAG_0002 2018-02-05 05:00:00 000:000:000 15
TAG_0002 2018-02-06 06:00:00 000:000:000 16
TAG_0002 2018-02-07 07:00:00 000:000:000 17
TAG_0002 2018-02-08 08:00:00 000:000:000 18
TAG_0002 2018-02-09 09:00:00 000:000:000 19
TAG_0002 2018-02-10 10:00:00 000:000:000 20
[20] row(s) selected.

Mach> delete from tag before to_date('2018-02-01');
10 row(s) deleted.

Mach> select * from tag;
NAME TIME VALUE
--------------------------------------------------------------------------------
TAG_0002 2018-02-01 01:00:00 000:000:000 11
TAG_0002 2018-02-02 02:00:00 000:000:000 12
TAG_0002 2018-02-03 03:00:00 000:000:000 13
TAG_0002 2018-02-04 04:00:00 000:000:000 14
TAG_0002 2018-02-05 05:00:00 000:000:000 15
TAG_0002 2018-02-06 06:00:00 000:000:000 16
TAG_0002 2018-02-07 07:00:00 000:000:000 17
TAG_0002 2018-02-08 08:00:00 000:000:000 18
TAG_0002 2018-02-09 09:00:00 000:000:000 19
```

```
TAG_0002 2018-02-10 10:00:00 000:000:000 20
[10] row(s) selected.
```

전체 삭제

아무 조건이 없는 경우 전체 데이터가 삭제된다.

```
Mach> delete from tag;
10 row(s) deleted.

Mach> select * from tag;
NAME TIME VALUE
--------------------------------------------------------------------------------
[0] row(s) selected.
```

ROLLUP 데이터의 삭제

문법

```
DELETE FROM TAG ROLLUP BEFORE TO_DATE('Time-string');
```

위에서 BEFORE 구문의 시간을 지정하면 그 시간 이전의 시, 분, 초 ROLLUP 데이터가 삭제된다.

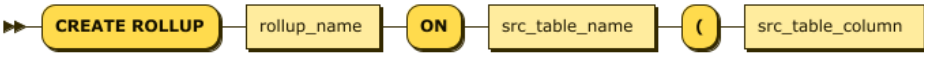
만일 BEFORE 구문을 지정하지 않는 경우에는 ROLLUP의 모든 데이터를 삭제한다.

롤업 테이블의 생성 및 조회

목차

ROLLUP 테이블 생성

Tag Table 생성시 Rollup이 기본으로 생성되지 않고, 사용자가 직접 생성하는 방식으로 변경되었으며 문법은 아래와 같다.



- rollup name : 생성될 rollup table의 이름 (40자 이내의 문자열로 자유롭게 생성 가능)
- source table name : 생성될 rollup이 데이터를 집계할 source table 이름
- src_table_column : rollup 대상 데이터 칼럼 이름
  - 숫자형 타입의 칼럼만 가능
  - source table이 rollup table인 경우 생략하며, source table의 rollup 대상 칼럼으로 자동 지정
- *number sec/min/hour : 집계할 시간 숫자와 시간 단위
  ex) 1초 단위 집계 : 1 sec
  ex) 30초 단위 집계 : 30 sec
  ex) 1분 단위 집계 : 1 min
  ex) 1시간 단위 집계 : 1 hour
- 제약조건
  - 집계할 source table은 tag table 또는 rollup table만 지정 가능하다.
  - 집계할 source table이 rollup table일 경우 생성될 rollup table의 시간은 source table의 시간보다 크며, 배수여야 한다.

롤업 테이블 생성 예시

```
Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIN
Executed successfully.

-- tag table의 value 칼럼 대상 1초 rollup 생성
Mach> CREATE ROLLUP _tag_rollup_sec ON tag(value) INTERVAL 1 SEC;

-- tag table의 value 칼럼 대상 1분 rollup 생성
Mach> CREATE ROLLUP _tag_rollup_min ON tag(value) INTERVAL 1 MIN;

-- tag table 대상 1시간 rollup 생성
Mach> CREATE ROLLUP _tag_rollup_hour ON tag(value) INTERVAL 1 HOUR;

-- tag table 대상 30초 rollup 생성
Mach> CREATE ROLLUP _tag_rollup_30sec ON tag(value) INTERVAL 30 SEC;

-- rollup table(위 30초 rollup) 대상 10분 rollup 생성
Mach> CREATE ROLLUP _tag_rollup_10min ON _tag_rollup_30sec INTERVAL 10 MIN;

-- 숫자형 타입이 아닌 칼럼에 대해 rollup 생성 시 에러
Mach> CREATE ROLLUP _tag_rollup_sec ON tag(strvalue) INTERVAL 1 SEC;
[ERR-02671: Invalid type for ROLLUP column (STRVALUE).]
```

ROLLUP 테이블 시작/중지

rollup을 생성해도 rollup thread가 자동으로 시작되지 않아 사용자가 직접 rollup 을 시작해야한다.

```
-- 특정 rollup 시작
EXEC ROLLUP_START(rollup_name)


-- 특정 rollup 중지
```

```
EXEC ROLLUP_STOP(rollup_name)
```

ROLLUP 테이블 즉시 수집

rollup은 기본적으로 설정된 시간 단위마다 데이터 집계를 시작한다.

- ex) 1시간 단위 rollup이라면 1시간 마다 한번씩 데이터 집계를 하고, 나머지 시간은 대기한다.

사용자가 수동으로 대기 시간을 무시하고 강제로 데이터 집계를 실행할 수 있다.

```
-- 특정 rollup 즉시 수집
EXEC ROLLUP_FORCE(rollup_name)
```

ROLLUP 테이블 삭제

Rollup을 삭제한다.

```
DROP ROLLUP rollup_name
```

- rollup_name : 삭제할 rollup 이름
- 제약조건: 삭제할 rollup table을 source table로 참조하고 있는 rollup이 존재할 경우 삭제 할 수 없으며 rollup 간의 의존성이 있는 경우 rollup 을 생성한 역순으로 삭제해야 한다.

```
mach> create tag table tag (name varchar(20) primary key, time datetime basetim
mach> create rollup _tag_rollup_1 on tag(value) interval 1 sec;
mach> create rollup _tag_rollup_2 on _tag_rollup_1 interval 1 min;
mach> create rollup _tag_rollup_3 on _tag_rollup_2 interval 1 hour;

위와 같이 생성했을 경우 참조 순서는 아래와 같다.

tag -> _tag_rollup_1 -> _tag_rollup_2 -> _tag_rollup_3

이 때 tag table이나, 중간에 있는 rollup을 삭제하려고 하면 에러가 발생한다.

mach> drop rollup tag
> [ERR-02651: Dependent ROLLUP table exists.]
mach> drop rollup _tag_rollup_1
> [ERR-02651: Dependent ROLLUP table exists.]

아래 순서대로 삭제해야 정상적으로 삭제할 수 있다.

mach> drop rollup _tag_rollup_3;
mach> drop rollup _tag_rollup_2;
mach> drop rollup _tag_rollup_1;
mach> drop table tag;
```

조회 문법

```
SELECT TIME ROLLUP 3 SECOND, AVG(VALUE) FROM TAG WHERE ...;
```

위와 같이 BASETIME 속성으로 지정된 Datetime 형 컬럼 뒤에 ROLLUP 절을 붙여 지정하면 롤업 테이블 조회가 된다.

```
[BASETIME_COLUMN] ROLLUP [PERIOD] [TIME_UNIT]
```

- BASETIME_COLUMN : BASETIME 속성으로 지정된 TAG 테이블의 Datetime 형 컬럼
- PERIOD : DATE_TRUNC() 함수에서 사용 가능한 시간 단위별 범위를 지정할 수 있다. (아래 참고)
- TIME_UNIT : DATE_TRUNC() 함수에서 사용 가능한 모든 시간 단위를 사용할 수 있다. (아래 참고)

TIME_UNIT 의 선택에 따라, 조회되는 롤업 테이블이 달라진다.

시간 단위 (축약어)	시간 범위	조회 대상 롤업 테이블
nanosecond (nsec)	1000000000 (1초)	SECOND
microsecond (usec)	60000000 (60초)	SECOND
milisecond (msec)	60000 (60초)	SECOND
second (sec)	86400 (1일)	SECOND
minute (min)	1440 (1일)	MINUTE
hour	24 (1일)	HOUR
day	1	HOUR
month	1	HOUR
year	1	HOUR

ROLLUP 절을 사용하는 것은 롤업 테이블 조회를 직접 하는 것이기 때문에, 집계 함수를 사용하려면 다음의 특징이 있다.

- **숫자형 타입의 컬럼에 집계 함수를 호출해야 한다.** 단, 롤업 테이블에서 지원하는 여섯 가지 집계 함수 (SUM, COUNT, MIN, MAX, AVG, SUMSQ) 만 지원한다.
- **ROLLUP 하는 BASETIME 컬럼으로 GROUP BY 를 직접 해야 한다.**
  - 같은 의미의 ROLLUP 절을 그대로 사용해도 된다.
  - 또는, ROLLUP 절에 별명 (alias) 를 붙이고, 별명으로 GROUP BY 에 작성해도 된다.

```
SELECT   time rollup 3 sec mtime, avg(value)
FROM     TAG
GROUP BY time rollup 3 sec mtime;

-- 또는
SELECT   time rollup 3 sec mtime, avg(value)
FROM     TAG
GROUP BY mtime;
```

데이터 샘플

아래는 롤업 테스트를 위한 샘플 데이터이다.

```
create tag table TAG (name varchar(20) primary key, time datetime basetime, value double summarized);

insert into tag metadata values ('TAG_0001');

insert into tag values('TAG_0001', '2018-01-01 01:00:01 000:000:000', 1);
insert into tag values('TAG_0001', '2018-01-01 01:00:02 000:000:000', 2);
insert into tag values('TAG_0001', '2018-01-01 01:01:01 000:000:000', 3);
insert into tag values('TAG_0001', '2018-01-01 01:01:02 000:000:000', 4);
insert into tag values('TAG_0001', '2018-01-01 01:02:01 000:000:000', 5);
insert into tag values('TAG_0001', '2018-01-01 01:02:02 000:000:000', 6);

insert into tag values('TAG_0001', '2018-01-01 02:00:01 000:000:000', 1);
insert into tag values('TAG_0001', '2018-01-01 02:00:02 000:000:000', 2);
insert into tag values('TAG_0001', '2018-01-01 02:01:01 000:000:000', 3);
insert into tag values('TAG_0001', '2018-01-01 02:01:02 000:000:000', 4);
insert into tag values('TAG_0001', '2018-01-01 02:02:01 000:000:000', 5);
insert into tag values('TAG_0001', '2018-01-01 02:02:02 000:000:000', 6);

insert into tag values('TAG_0001', '2018-01-01 03:00:01 000:000:000', 1);
insert into tag values('TAG_0001', '2018-01-01 03:00:02 000:000:000', 2);
insert into tag values('TAG_0001', '2018-01-01 03:01:01 000:000:000', 3);
insert into tag values('TAG_0001', '2018-01-01 03:01:02 000:000:000', 4);
insert into tag values('TAG_0001', '2018-01-01 03:02:01 000:000:000', 5);
insert into tag values('TAG_0001', '2018-01-01 03:02:02 000:000:000', 6);
```

태그 하나에 대해서 3시간 동안 초단위의 각기 다른 값을 입력해 놓았다.

ROLLUP 평균값 얻기

아래는 해당 태그에 대해 초, 분, 시 단위의 평균값을 얻는 예제이다.

```
Mach> SELECT time rollup 1 sec mtime, avg(value) FROM TAG WHERE name = 'TAG_0001' group by mtime order by mtime;
mtime                          avg(value)
---------------------------------------------------------------
2018-01-01 01:00:01 000:000:000 1
2018-01-01 01:00:02 000:000:000 2
2018-01-01 01:01:01 000:000:000 3
2018-01-01 01:01:02 000:000:000 4
2018-01-01 01:02:01 000:000:000 5
2018-01-01 01:02:02 000:000:000 6
2018-01-01 02:00:01 000:000:000 1
2018-01-01 02:00:02 000:000:000 2
2018-01-01 02:01:01 000:000:000 3
2018-01-01 02:01:02 000:000:000 4
2018-01-01 02:02:01 000:000:000 5
2018-01-01 02:02:02 000:000:000 6
2018-01-01 03:00:01 000:000:000 1
2018-01-01 03:00:02 000:000:000 2
2018-01-01 03:01:01 000:000:000 3
2018-01-01 03:01:02 000:000:000 4
2018-01-01 03:02:01 000:000:000 5
2018-01-01 03:02:02 000:000:000 6
[18] row(s) selected.

Mach> SELECT time rollup 1 min mtime, avg(value) FROM TAG WHERE name = 'TAG_0001' group by mtime order by mtime;
mtime                          avg(value)
---------------------------------------------------------------
2018-01-01 01:00:00 000:000:000 1.5
2018-01-01 01:01:00 000:000:000 3.5
2018-01-01 01:02:00 000:000:000 5.5
2018-01-01 02:00:00 000:000:000 1.5
2018-01-01 02:01:00 000:000:000 3.5
2018-01-01 02:02:00 000:000:000 5.5
2018-01-01 03:00:00 000:000:000 1.5
2018-01-01 03:01:00 000:000:000 3.5
2018-01-01 03:02:00 000:000:000 5.5
[9] row(s) selected.

Mach> SELECT time rollup 1 hour mtime, avg(value) FROM TAG WHERE name = 'TAG_0001' group by mtime order by mtime;
mtime                          avg(value)
---------------------------------------------------------------
2018-01-01 01:00:00 000:000:000 3.5
2018-01-01 02:00:00 000:000:000 3.5
2018-01-01 03:00:00 000:000:000 3.5
[3] row(s) selected.
```

ROLLUP 최소/최대값 얻기

아래는 해당 태그의 시간 범위에 따른 최소/최대값을 얻는 예제를 나타낸다. 이전 예제와 다른 점은, 쿼리 한 번에 최대값과 최소값을 동시에 얻을 수 있다는 것이다.

```
Mach> SELECT time rollup 1 hour mtime, min(value), max(value) FROM TAG WHERE name = 'TAG_0001' group by mtime order
mtime                          min(value)                max(value)
-------------------------------------------------------------------------------------------
2018-01-01 01:00:00 000:000:000 1                        6
2018-01-01 02:00:00 000:000:000 1                        6
2018-01-01 03:00:00 000:000:000 1                        6
[3] row(s) selected.

Mach> SELECT time rollup 1 min mtime, min(value), max(value) FROM TAG WHERE name = 'TAG_0001' group by mtime order
mtime                          min(value)                max(value)
-------------------------------------------------------------------------------------------
2018-01-01 01:00:00 000:000:000 1                        2
2018-01-01 01:01:00 000:000:000 3                        4
2018-01-01 01:02:00 000:000:000 5                        6
```

```
2018-01-01 02:00:00 000:000:000 1                          2
2018-01-01 02:01:00 000:000:000 3                          4
2018-01-01 02:02:00 000:000:000 5                          6
2018-01-01 03:00:00 000:000:000 1                          2
2018-01-01 03:01:00 000:000:000 3                          4
2018-01-01 03:02:00 000:000:000 5                          6
[9] row(s) selected.
```

ROLLUP 합계/개수 얻기

아래는 합계 및 데이터 개수 값을 얻는 예제이다. 역시 하나의 쿼리에 합계와 개수를 얻을 수 있다.

```
Mach> SELECT time rollup 1 min  mtime, sum(value), count(value) FROM TAG WHERE name = 'TAG_0001' group by mtime orc
mtime                          sum(value)                 count(value)
-------------------------------------------------------------------------------
2018-01-01 01:00:00 000:000:000 3                          2
2018-01-01 01:01:00 000:000:000 7                          2
2018-01-01 01:02:00 000:000:000 11                         2
2018-01-01 02:00:00 000:000:000 3                          2
2018-01-01 02:01:00 000:000:000 7                          2
2018-01-01 02:02:00 000:000:000 11                         2
2018-01-01 03:00:00 000:000:000 3                          2
2018-01-01 03:01:00 000:000:000 7                          2
2018-01-01 03:02:00 000:000:000 11                         2
[9] row(s) selected.
```

ROLLUP 제곱합 얻기

아래는 제곱합 값을 얻는 예제이다.

```
Mach> SELECT time ROLLUP 1 SEC mtime, SUMSQ(value) FROM tag GROUP BY mtime ORDER BY mtime;
mtime                          SUMSQ(value)
---------------------------------------------------------------
2018-01-01 01:00:01 000:000:000 1
2018-01-01 01:00:02 000:000:000 4
2018-01-01 01:01:01 000:000:000 9
2018-01-01 01:01:02 000:000:000 16
2018-01-01 01:02:01 000:000:000 25
2018-01-01 01:02:02 000:000:000 36
2018-01-01 02:00:01 000:000:000 1
2018-01-01 02:00:02 000:000:000 4
2018-01-01 02:01:01 000:000:000 9
2018-01-01 02:01:02 000:000:000 16
2018-01-01 02:02:01 000:000:000 25
2018-01-01 02:02:02 000:000:000 36
2018-01-01 03:00:01 000:000:000 1
2018-01-01 03:00:02 000:000:000 4
2018-01-01 03:01:01 000:000:000 9
2018-01-01 03:01:02 000:000:000 16
2018-01-01 03:02:01 000:000:000 25
2018-01-01 03:02:02 000:000:000 36
[18] row(s) selected.

Mach> SELECT time ROLLUP 1 MIN mtime, SUMSQ(value) FROM tag GROUP BY mtime ORDER BY mtime;
mtime                          SUMSQ(value)
---------------------------------------------------------------
2018-01-01 01:00:00 000:000:000 5
2018-01-01 01:01:00 000:000:000 25
2018-01-01 01:02:00 000:000:000 61
2018-01-01 02:00:00 000:000:000 5
2018-01-01 02:01:00 000:000:000 25
2018-01-01 02:02:00 000:000:000 61
2018-01-01 03:00:00 000:000:000 5
2018-01-01 03:01:00 000:000:000 25
```

```
2018-01-01 03:02:00 000:000:000 61
[9] row(s) selected.
```

다양한 시간 간격으로 그룹화

ROLLUP 절의 장점은, DATE_TRUNC() 를 의도적으로 사용해서 시간 간격을 다변화할 필요가 없다는 것이다.

3초 간격의 합계와 데이터 개수를 얻으려면 아래와 같이 하면 된다.
예제 시간 범위가 0초, 1초, 2초 뿐이라 전부 0초로 수렴된 것을 확인할 수 있다. 결과적으로는 '분 단위 롤업' 조회 결과와 일치한다.

```
Mach> SELECT time rollup 3 sec  mtime, sum(value), count(value) FROM TAG WHERE name = 'TAG_0001' GROUP BY mtime ORD
mtime                          sum(value)                count(value)
-------------------------------------------------------------------------------
2018-01-01 01:00:00 000:000:000 3                        2
2018-01-01 01:01:00 000:000:000 7                        2
2018-01-01 01:02:00 000:000:000 11                       2
2018-01-01 02:00:00 000:000:000 3                        2
2018-01-01 02:01:00 000:000:000 7                        2
2018-01-01 02:02:00 000:000:000 11                       2
2018-01-01 03:00:00 000:000:000 3                        2
2018-01-01 03:01:00 000:000:000 7                        2
2018-01-01 03:02:00 000:000:000 11                       2
```

JSON 타입 대상의 ROLLUP 활용

7.5 버전부터 JSON 타입을 대상으로 ROLLUP을 사용할 수 있다.

생성 구문에 JSON PATH를 OPERATOR와 연결하면 된다.

JSON 타입 특성상, 하나의 JSON 칼럼에 PATH 별로 ROLLUP을 생성할 수 있다.

```
-- create tag table
CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, jval JSON);

-- insert data
insert into tag values ('tag-01', '2022-09-01 01:01:01', "{ \"x\": 1, \"y\": 1.1}");
insert into tag values ('tag-01', '2022-09-01 01:01:02', "{ \"x\": 2, \"y\": 1.2}");
insert into tag values ('tag-01', '2022-09-01 01:01:03', "{ \"x\": 3, \"y\": 1.3}");
insert into tag values ('tag-01', '2022-09-01 01:01:04', "{ \"x\": 4, \"y\": 1.4}");
insert into tag values ('tag-01', '2022-09-01 01:01:05', "{ \"x\": 5, \"y\": 1.5}");
insert into tag values ('tag-01', '2022-09-01 01:02:00', "{ \"x\": 6, \"y\": 1.6}");
insert into tag values ('tag-01', '2022-09-01 01:03:00', "{ \"x\": 7, \"y\": 1.7}");
insert into tag values ('tag-01', '2022-09-01 01:04:00', "{ \"x\": 8, \"y\": 1.8}");
insert into tag values ('tag-01', '2022-09-01 01:05:00', "{ \"x\": 9, \"y\": 1.9}");
insert into tag values ('tag-01', '2022-09-01 01:06:00', "{ \"x\": 10, \"y\": 2.0}");

-- create rollup
CREATE ROLLUP _tag_rollup_jval_x_sec ON tag(jval->'$.x') INTERVAL 1 SEC;
CREATE ROLLUP _tag_rollup_jval_y_sec ON tag(jval->'$.y') INTERVAL 1 SEC;
```

ROLLUP 조회도 동일하게 사용하면 된다.

```
Mach> SELECT time ROLLUP 2 SEC mtime, MIN(jval->'$.x'), MAX(jval->'$.x'), SUM(jval->'$.x'), COUNT(jval->'$.x'), SUM
mtime                          min(jval->'$.x')          max(jval->'$.x')          sum(jval->'$.x')
-------------------------------------------------------------------------------------------------------------------
2022-09-01 01:01:00 000:000:000 1                        1                         1
2022-09-01 01:01:02 000:000:000 2                        3                         5
2022-09-01 01:01:04 000:000:000 4                        5                         9
2022-09-01 01:02:00 000:000:000 6                        6                         6
2022-09-01 01:03:00 000:000:000 7                        7                         7
2022-09-01 01:04:00 000:000:000 8                        8                         8
2022-09-01 01:05:00 000:000:000 9                        9                         9
2022-09-01 01:06:00 000:000:000 10                       10                        10
[8] row(s) selected.

Mach> SELECT time ROLLUP 2 SEC mtime, MIN(jval->'$.y'), MAX(jval->'$.y'), SUM(jval->'$.y'), COUNT(jval->'$.y'), SUM
mtime                          min(jval->'$.y')          max(jval->'$.y')          sum(jval->'$.y')
```

```
---------------------------------------------------------------------------------------------------------------------
2022-09-01 01:01:00 000:000:000 1.1                    1.1                    1.1
2022-09-01 01:01:02 000:000:000 1.2                    1.3                    2.5
2022-09-01 01:01:04 000:000:000 1.4                    1.5                    2.9
2022-09-01 01:02:00 000:000:000 1.6                    1.6                    1.6
2022-09-01 01:03:00 000:000:000 1.7                    1.7                    1.7
2022-09-01 01:04:00 000:000:000 1.8                    1.8                    1.8
2022-09-01 01:05:00 000:000:000 1.9                    1.9                    1.9
2022-09-01 01:06:00 000:000:000 2                      2                      2
[8] row(s) selected.
```

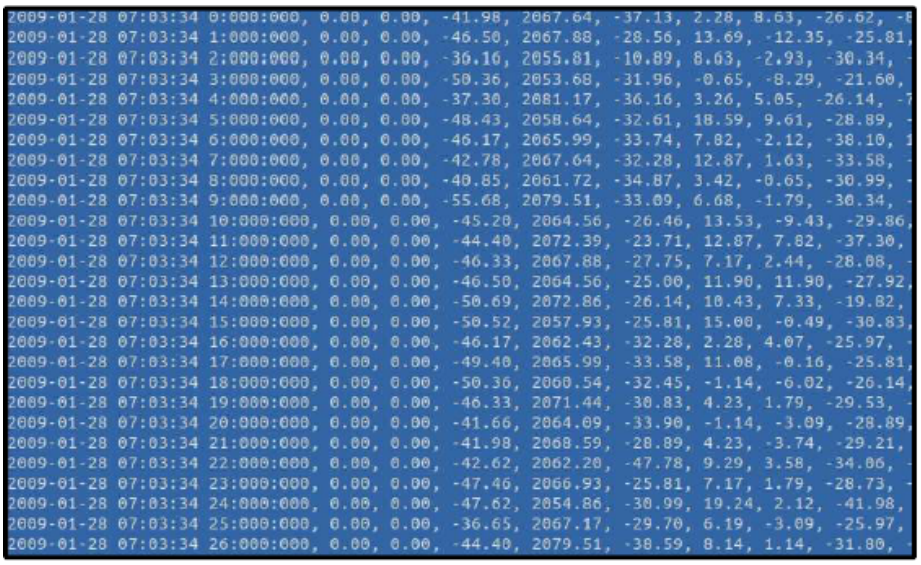
태그 테이블 활용 샘플 예제

개요

태그 테이블은 일반적인 센서 데이터가 저장된 파일의 구조 형태를 로딩할 수 있다.

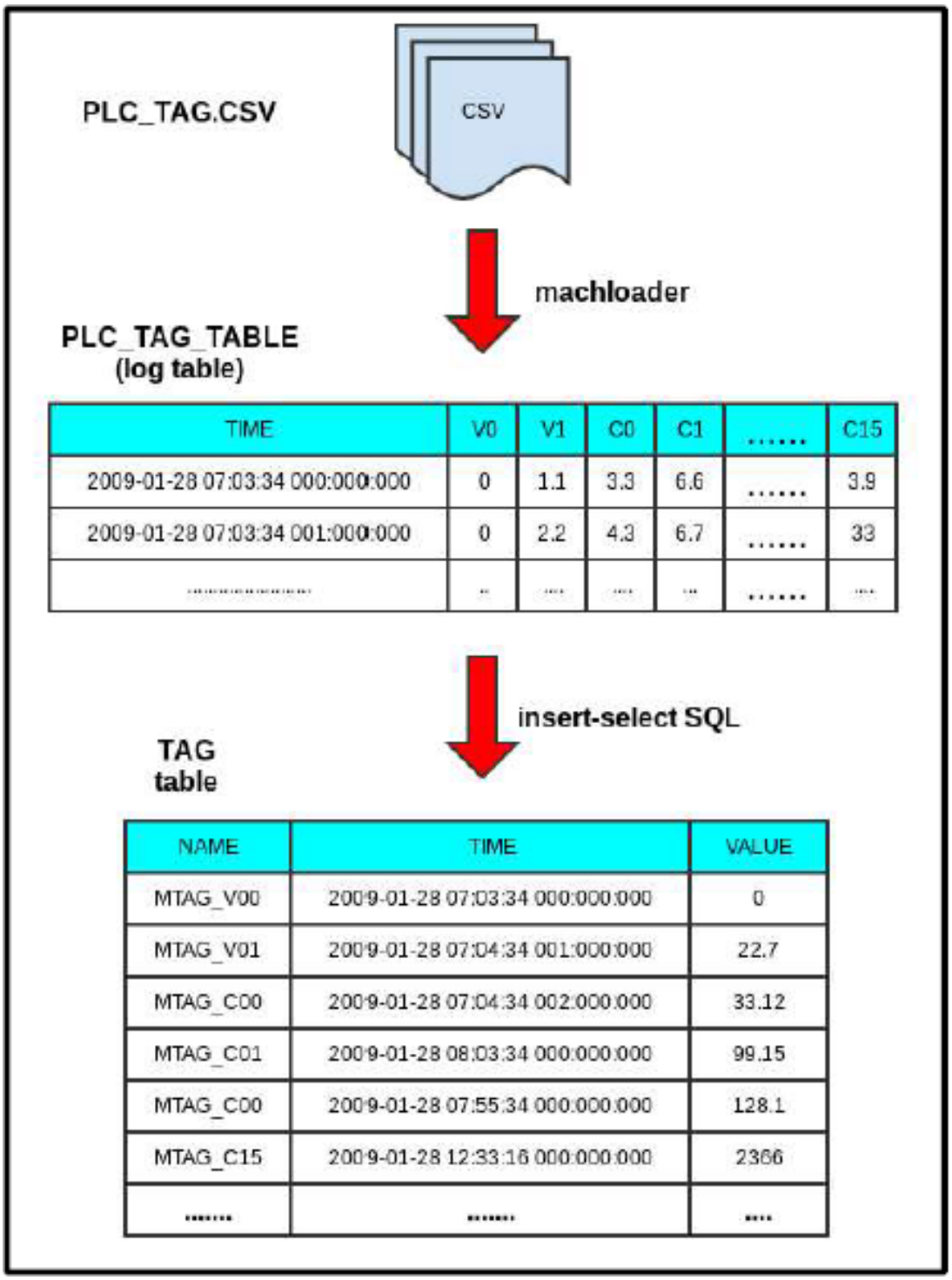
가장 흔히 볼 수 있는 형태의 텍스트 저장 파일은 아무런 설명 없이, (콤마)나 나뉘어진 다수의 숫자형 값을 그냥 나열한 무작위의 파일 내용인 <값,값,값><값,값,값><반복..> 형태가 대표적이고, 시간을 포함한 파일의 경우에는 <시간,값,값,값> <시간, 값,값,값><반복..>이 있다.

이런 파일의 데이터는 PLC (programmable Logic Controller)라고 불리는 장비에서 1개 이상의 센서 값을 지속적으로 입력된 데이터를 오랜 기간동안 수집했을 경우에 만들어진다. 아래의 사진이 그 예이다.



이제 이 파일을 어떻게 마크베이스의 태그 테이블로 한꺼번에 배치 형태로 로딩하겠다.

데이터 변환 순서도



위의 그림에서 볼 수 있듯이 원시 CSV 파일을 마크베이스의 로그 테이블로 한꺼번에 로딩을 한 이후, 이를 태그 테이블로 변환할 것이다.

태그 테이블 생성 및 태그 메타 로딩

아래와 같이 태그 테이블을 생성하고 tagmetaimport라는 도구를 이용해서 CSV 파일에 저장된 태그 이름(태그 메타)들을 한꺼번에 로딩한다.

아래 기술된 Option이외에도 machloader에서 사용할 수 있는 옵션을 모두 사용 가능하다.

```
Mach> create tag table tag (name varchar(32) primary key, time datetime basetime, value double
summarized);
Executed successfully.
Elapsed time: 3.032

$ cat tag_meta.csv
MTAG_V00
```

```
MTAG_V01
MTAG_C00
MTAG_C01
MTAG_C02
MTAG_C03
MTAG_C04
MTAG_C05
MTAG_C06
MTAG_C07
MTAG_C08
MTAG_C09
MTAG_C10
MTAG_C11
MTAG_C12
MTAG_C13
MTAG_C14
MTAG_C15

$ tagmetaimport -d tag_meta.csv
Import time : 0 hour 0 min 0.340 sec
Load success count : 18
```

(Machbase 포트번호를 기본값에서 변경했으면, tagmetaimport에 -P 옵션을 사용해서 변경된 포트번호를 사용해야한다.)
위와 같이 성공적으로 태그 메타 정보(이름) 18개가 로딩되었다.

PLC 데이터 로딩을 위한 테이블 생성

아래의 쿼리를 수행해 로그 테이블을 생성한다.

```
create table plc_tag_table(
    tm datetime,
    V0 DOUBLE ,
    V1 DOUBLE ,
    C0 DOUBLE ,
    C1 DOUBLE ,
    C2 DOUBLE ,
    C3 DOUBLE ,
    C4 DOUBLE ,
    C5 DOUBLE,
    C6 DOUBLE ,
    C7 DOUBLE ,
    C8 DOUBLE ,
    C9 DOUBLE ,
    C10 DOUBLE ,
    C11 DOUBLE ,
    C12 DOUBLE ,
    C13 DOUBLE ,
    C14 DOUBLE ,
    C15 DOUBLE
);
```

주의할 점은 이 테이블은 로그 테이블 타입이라는 것이다(파일명 때문에 헷갈리지 않도록 하자). 마크베이스에서는 별도의 테이블 지정자를 명시하지 않으면, 로그 테이블로 생성된다.

PLC 데이터 로딩

아래와 같이 machloader를 사용해 200만 건의 원시 PLC 데이터가 저장된 plc_tag.csv 파일을 위에서 생성한 로그 테이블 plc_tag_table에 PLC 입력 형태로 입력한다. plc_tag.csv 파일은 첫 컬럼은 시간이며, 이후 순서대로 V0, V1, ...C15 까지 컬럼이 나뉘어져 있다. 데이터의 패턴은 1초에 0~99mili second까지 약 100개의 데이터가 입력되고, 100 mili second에서 999까지는 입력이 없다가, 다음 1초 동안 동일한 패턴으로 입력된다.

```
$ machloader -t plc_tag_table -i -d plc_tag.csv -F "tm YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn"
------------------------------------------------------------------
Machbase Data Import/Export Utility.
Release Version 5.5.0.official
Copyright 2014, MACHBASE Corporation or its subsidiaries.
```

```
All Rights Reserved.
----------------------------------------------------------------
NLS : US7ASCII EXECUTE MODE : IMPORT
TARGET TABLE : plc_tag_table DATA FILE : 4_plc_tag.csv
IMPORT MODE : APPEND FIELD TERM : ,
ROW TERM : \n ENCLOSURE : "
ESCAPE : \ ARRIVAL_TIME : FALSE
ENCODING : NONE HEADER : FALSE
CREATE TABLE : FALSE
Progress bar Imported records Error records
============================== 2000000 0
Import time : 0 hour 0 min 26.544 sec
Load success count : 2000000
Load fail count : 0
```

태그 메타 이름 생성 규칙

이제 태그 테이블에 데이터를 넣어서 실제로 Tag Analyzer를 통해서 데이터를 확인할 수 있도록 한다. 이를 위해서 plc_tag_table 의 정보를 모두 태그 테이블에 넣어야 하는데, 이를 위해서 insert-select 구문을 이용해서 한꺼번에 넣는다. 그리고, 각 컬럼의 값이 모든 태그 테이블의 이름과 맵핑이 되어야 하기 때문에 다음과 같이 매타 태그의 이름 정보를 미리 결정하였다.

로그 테이블의 컬럼명	태그 테이블의 Name 컬럼에 입력되는 이름
V0	MTAG_V00
V1	MTAG_V01
C0	MTAG_C00
C1	MTAG_C01
...	
C15	MTAG_C15

태그 테이블 데이터 로딩

이제 마지막으로 실제 데이터를 태그 테이블로 로딩할 차례이다. 아래의 쿼리를 수행하면 하나씩 순차적으로 태그 테이블에 넣는다.

```
Mach> insert into tag select 'MTAG_V00', tm, v0 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 4.898
Mach> insert into tag select 'MTAG_V01', tm, v1 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 5.577
Mach> insert into tag select 'MTAG_C00', tm, c0 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 6.327
Mach> insert into tag select 'MTAG_C01', tm, c1 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 7.445
Mach> insert into tag select 'MTAG_C02', tm, c2 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 6.898
Mach> insert into tag select 'MTAG_C03', tm, c3 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 7.078
Mach> insert into tag select 'MTAG_C04', tm, c4 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 6.799
Mach> insert into tag select 'MTAG_C05', tm, c5 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 7.210
Mach> insert into tag select 'MTAG_C06', tm, c6 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 9.232
Mach> insert into tag select 'MTAG_C07', tm, c7 from plc_tag_table;
```

```
2000000 row(s) inserted.
Elapsed time: 6.398
Mach> insert into tag select 'MTAG_C08', tm, c8 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 6.432
Mach> insert into tag select 'MTAG_C09', tm, c9 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 6.734
Mach> insert into tag select 'MTAG_C10', tm, c10 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 7.692
Mach> insert into tag select 'MTAG_C11', tm, c11 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 8.628
Mach> insert into tag select 'MTAG_C12', tm, c12 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 8.229
Mach> insert into tag select 'MTAG_C13', tm, c13 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 9.517
Mach> insert into tag select 'MTAG_C14', tm, c14 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 7.231
Mach> insert into tag select 'MTAG_C15', tm, c15 from plc_tag_table;
2000000 row(s) inserted.
Elapsed time: 7.830
```

총 3600 만건의 데이터가 로딩된 것을 확인할 수 있다.

태그 테이블 인덱스 생성 및 관리

마크베이스의 태그테이블에는 추가 칼럼에 대한 TAG 인덱스 타입을 생성할 수 있다.

자세한 내용은 SQL 레퍼런스의 DDL 페이지의 CREATE INDEX 문단을 참조하면 된다.

- TAG 인덱스: 기본칼럼(time, name)등을 제외한 추가 칼럼에 대한 인덱스

목차

인덱스 생성

CREATE INDEX 구문을 이용하여 특정 컬럼에 대해서 인덱스를 생성한다.

```
CREATE INDEX index_name ON table_name (column_name) [index_type]
    index_type ::= INDEX_TYPE { TAG }
```

```
Mach> CREATE INDEX id_index ON tag (id) INDEX_TYPE TAG;
Created successfully.
```

7.5 버전부터 tag table에 한해 json 타입의 칼럼에 대해서 json path 별로 인덱스를 생성할 수 있다.

기존의 인덱스 생성 구문에 json path를 operator와 연결하면 된다.

json operator의 return 타입이 varchar이므로, varchar 비교 시 인덱스를 사용할 수 있다.

```
Mach> CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, jval JSON);
Executed successfully.

Mach> CREATE INDEX idx_jval_value1 ON tag (jval->'$.value1');
Created successfully.

Mach> CREATE INDEX idx_jval_value2 ON tag (jval->'$.value2');
Created successfully.

Mach> EXPLAIN SELECT * FROM tag WHERE jval->'$.value1' = '10';
PLAN
-------------------------------------------------------------------------------------
 PROJECT
  TAG READ (RAW)
   KEYVALUE INDEX SCAN (_TAG_DATA_0)
    [KEY RANGE]
     * jval->'$.value1' = '10'
   VOLATILE FULL SCAN (_TAG_META)
[6] row(s) selected.
```

인덱스 삭제

DROP INDEX 구문을 이용하여 지정된 인덱스를 삭제한다. 단, 해당 테이블을 검색 중인 다른 세션이 존재할 경우에는 에러를 내면서 실패한다.

```
DROP INDEX index_name;
```

```
Mach> DROP INDEX id_index;
Dropped successfully.
```

로그 테이블 (Log Table)

개념

로그 테이블은 입력되는 데이터가 시계열 데이터인 머신 로그 데이터를 저장할 수 있는 테이블이다.

이 테이블에는 데이터가 무한히 입력되며, 각 필드의 값에는 고유한 의미가 있다.
또한 텍스트 필드(varchar 혹은 text) 를 통해 데이터를 검색할 수 있으며, 빠른 통계 연산이 가능하다.

마크베이스에서 테이블이라고 하면 기본적으로 "로그 테이블"을 가리킨다.

로그 테이블의 특징은 다음과 같다.

숨은 시간 컬럼 존재

모든 로그 테이블에는 _arrival_time이라는 숨겨진 컬럼이 있다.
이 컬럼에는 해당 레코드가 생성된 시간이 저장되어 있으며, 나노초 단위 정밀도를 지원한다.

시간 역순 검색

일반 데이터베이스는 데이터 검색 시 입력 순서와 무관하게 출력된다.
그러나 Machbase의 로그 테이블은 별도의 ORDER BY를 통한 정렬 옵션을 주지 않는 한 언제나 최신 데이터가 먼저 출력된다.
이는 _arrival_time 컬럼을 통해서도 확인할 수 있다.

이렇게 설계된 이유는 머신 로그 데이터에서는 최근 데이터의 중요도가 이전의 데이터에 비해 훨씬 높기 때문이다.

입력 후 조회 전용

Machbase의 로그 테이블은 변경(Update) 연산이 존재하지 않는다. 다시 말해 사용자 로그 데이터가 일단 Machbase에 저장되고 나면,

해당 데이터에 대한 변경 연산을 불허함으로써 데이터의 안정성과 로그 데이터 자체의 무결성을 엔진 레벨에서 지원하는 것이다.

제한된 삭제 허용

Machbase가 비록 데이터의 변경은 불허하더라도 특수한 상황에서 필요한 데이터의 삭제는 허용한다.
그러나 전통적인 데이터베이스에서처럼 임의의 데이터를 삭제할 수는 없으며, 가장 오래된 데이터부터 순차적으로 삭제하는 것만 가능하다.
이 기능을 통해 저장공간에 제약이 있는 임베디드 장비나 관리가 쉽지 않은 형태의 장비에서 편리하게 주기적으로 데이터를 삭제하여 관리할 수 있다.

텍스트 검색 기능 지원

Machbase는 일반 데이터베이스의 문자열을 취급하는 방식에서 한걸음 더 나아가 단어 기반의 검색 기능을 제공한다.
이 기능은 머신 로그 데이터의 용도에 가장 잘 부합하는 것으로서, 특정 시간대에 저장된 로그 데이터에 대한 검색이 비즈니스 상황에서 주요한 기능으로 사용된다.

이를 위해서 Machbase는 실시간 역인덱스를 제공함으로써 데이터의 삽입과 동시에 실시간으로 텍스트 검색이 가능하게 함으로써 빠른 장애 진단 및 장애 상황 해결에 큰 도움을 줄 수 있다.

특수 데이터 타입 지원

Machbase는 IPv4, IPv6를 지원한다. 이는 인터넷 주소를 나타내는 특수한 타입으로서 수많은 머신 로그 데이터가 주로 표현하는 주소 체계를 반영한 것이다.
이 데이터 타입을 활용하여 특정 주소의 검색과 추출을 손쉽게 할 수 있다.

또한, `select * from t1 where ipaddr = '192.168.0.*'` 과 같은 확장 문법을 활용하여, 특정 주소 체계의 일부 주소 범위를 검색하거나 추출할 수 있는 기능도 함께 제공한다.
또한 netmask 연산자를 제공하여 특정 인터넷 주소가 특정한 주소 범위에 포함되는지 쉽게 판단할 수 있다.

LOB(Large Object) 데이터 지원

로그 테이블은 64MB 바이트까지 저장 가능한 Text 및 Binary 타입을 제공한다.

- 만일 해당 데이터가 텍스트 문서 형태로서 검색이 필요한 경우에는 Text 타입으로 저장하고 데이터를 검색할 수 있다.
- 만일 해당 데이터가 그림이나 음악과 같은 2진 데이터 형태인 경우 binary 타입으로 저장할 수 있다.

시간 기반 파티셔닝

로그 테이블은 시간 축을 기준으로 특정 개수의 레코드 및 인덱스를 유지하고 있는 파티션 파일의 연속체이다.
다시 말해 데이터가 계속 입력됨에 따라 새로운 파티션 파일이 생성되고, 그 파티션에 특정 개수의 레코드가 모두 차게 되면 다음 파티션이 생성된다는 의미이다.

파티션으로 관리하는 이유는 주로 시간을 기준으로 검색이 발생하는 로그 데이터의 특성을 반영한 것이며 데이터 입력 성능에 대단히 큰 장점이 있다.
통계 분석을 위한 초고속 데이터 접근에 용이한 구조이기 때문이다.

작업 방법

- 로그 테이블 생성 및 관리
- 로그 데이터의 입력
- 로그 데이터의 추출
- 로그 데이터의 삭제
- 로그 인덱스 생성 및 관리
- 로그 테이블 활용 샘플 예제

로그 테이블 생성 및 관리

로그 테이블은 다음과 같이 간단하게 생성할 수 있다. sensor_data 라는 테이블을 생성하고 삭제해 보도록 하자.
마크베이스에서 사용가능한 데이터 타입은 SQL 레퍼런스의 자료형 에서 확인하면 된다.

목차

로그 테이블 생성

'CREATE TABLE' 구문으로 로그 테이블을 생성한다.

```
Mach> CREATE TABLE sensor_data
      (
          id VARCHAR(32),
          val DOUBLE
       );
Created successfully.

Mach> DROP TABLE sensor_data;
Dropped successfully.
```

로그 테이블 삭제

'DROP TABLE' 구문으로 로그 테이블을 삭제한다

```
-- DROP은 데이터와 테이블 모두 삭제한다.
Mach> DROP TABLE sensor_data;
Dropped successfully.


-- TRUNCATE는 데이터만 삭제하고 테이블은 유지한다.
Mach> TRUNCATE TABLE sensor_data;
Truncated successfully.
```

로그 데이터의 입력

마크베이스에 로그 데이터를 입력하는 방법은 여러 가지가 있다.

- 데이터 입력 : Insert
- 데이터 입력 : Append
- 데이터 불러오기 : Import
- 데이터 불러오기 : Load

데이터 입력 : Insert

다른 상용 RDBMS와 유사하게, 테이블을 먼저 생성하고 데이터는 INSERT INTO 문을 이용하여 데이터를 입력할 수 있다.

마크베이스는 'machsql' 도구를 대화형 질의 처리기로 제공한다.

목차

테이블 생성

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ....
);
```

```
CREATE TABLE sensor_data
(
    id VARCHAR(32),
    val DOUBLE
);
```

데이터 입력

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO sensor_data VALUES('sensor1', 10.1);
INSERT INTO sensor_data VALUES('sensor2', 20.2);
INSERT INTO sensor_data VALUES('sensor3', 30.3);
```

데이터 입력 확인

```
SELECT column1, column2, ...
FROM table_name;
```

```
SELECT * FROM sensor_data;
```

전체 과정

아래는 machsql 을 사용한 예제이다.

```
Mach> CREATE TABLE sensor_data (id VARCHAR(32), val DOUBLE);
Created successfully.
Mach> INSERT INTO sensor_data VALUES('sensor1', 10.1);
1 row(s) inserted.
Mach> INSERT INTO sensor_data VALUES('sensor2', 20.2);
1 row(s) inserted.
Mach> INSERT INTO sensor_data VALUES('sensor3', 30.3);
1 row(s) inserted.
```

```
Mach> SELECT * FROM sensor_data;
ID                              VAL
---------------------------------------------------------------
sensor3                         30.3
sensor2                         20.2
sensor1                         10.1
[3] row(s) selected.
```

데이터 입력 : Append

마크베이스에서 제공하는 빠른 실시간 데이터 입력 API이다.
C, C++, C#, Java, Python, PHP, Javascript 를 이용하여 입력할 수 있다.
세부 내용은 SDK 가이드를 참고한다.

데이터 불러오기 : Import

machloader 도구를 이용하여, CSV 또는 다른 구분자로 구별된 텍스트 파일을 입력할 수 있다.
machloader 도구에 대한 자세한 설명은 machloader 문서를 참조한다.

목차

테이블 생성

```
CREATE TABLE import_sample
(
    srcip     IPV4,
    srcport   INTEGER,
    dstip     IPV4,
    dstport   INTEGER,
    protocol  SHORT,
    eventlog  VARCHAR(1024),
    eventcode SHORT,
    eventsize LONG
);
```

데이터 불러오기

machloader 도구를 이용하여 csv 파일을 입력한다.

```
machloader  -i  -t  import_sample   -d  sample_data.csv
```

입력 데이터 확인

입력 데이터를 확인한다.

```
SELECT  COUNT(*)    FROM    import_sample;
```

샘플 예제

아래는, 실제 machloader 와 machsql 을 이용한 샘플 예제 과정을 나타낸 것이다.

```
Mach> CREATE TABLE import_sample
     (
         srcip     IPV4,
         srcport   INTEGER,
         dstip     IPV4,
         dstport   INTEGER,
         protocol  SHORT,
         eventlog  VARCHAR(1024),
         eventcode SHORT,
         eventsize LONG
     );
Created successfully.
Mach> quit
```

```
[mach@localhost ~]$ cd $MACHBASE_HOME/sample/quickstart
```

```
[mach@localhost ~]$ ls -l sample_data.csv
-rw-r--r--- 1 mach mach 110477124 2017-02-23 15:18 sample_data.csv

[mach@localhost ~]$ machloader -i -t import_sample -d sample_data.csv
------------------------------------------------------------------
     Machbase Data Import/Export Utility.
     Release Version x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries.
     All Rights Reserved.
------------------------------------------------------------------
NLS            : US7ASCII            EXECUTE MODE   : IMPORT
TARGET TABLE   : import_sample
DATA FILE      : sample_data.csv
IMPORT_MODE    : APPEND
FILED TERM     : ,                   ROW TERM       : \n
ENCLOSURE      : "                   ARRIVAL_TIME   : FALSE
ENCODING       : NONE                HEADER         : FALSE
CREATE TABLE   : FALSE
 Progress bar                       Imported records        Error records
                                           1000000                     0
Import time         :  0 hour  0 min  2.39 sec
Load success count  : 1000000
Load fail count     : 0
[mach@localhost ~]$
```

```
Mach> SELECT COUNT(*) FROM import_sample;
COUNT(*)
-----------------------
1000000
[1] row(s) selected.
Mach>
```

데이터 불러오기 : Load

'Load Data' 문은 csv 파일을 마크베이스에 입력한다.

먼저 데이터를 저장할 테이블을 생성하는데, csv 파일의 첫 번째 라인을 이용하여 칼럼들을 생성한다.

- 생성된 칼럼들의 데이터 타입은 VARCHAR(32768)이다.
- 데이터 파일 경로는 $MACHBASE_HOME을 기준으로 한 상대 경로이다. 절대경로로 설정할 수도 있다.

테이블의 데이터들을 csv 파일로 저장하기 위해서는 SAVE DATA 문을 이용한다.

만약 csv파일의 각 필드에 대한 데이터 타입을 미리 알고 있다면, 테이블을 미리 생성하여 데이터를 입력할 수 있다.

'load_sample.csv' 파일을 LOAD DATA 문으로 입력하면, 테이블 'load_sample' 이 자동으로 생성된다.

목차

데이터 불러오기

```
LOAD DATA INFILE 'sample/quickstart/load_sample.csv' INTO TABLE load_sample AUTO HEADUSE;
```

입력 데이터 확인

```
SELECT * FROM load_sample;
```

샘플 예제

샘플 파일을 이용하여, 다음과 같이 실행할 수 있다.

```
[mach@localhost ~]$ cd $MACHBASE_HOME/sample/quickstart
[mach@localhost ~]$ ls -l load_sample.csv
-rw-r
--r--- 1 root root 2827 2017-02-23 15:01 load_sample.csv

[mach@localhost ~]$ machsql
=================================================================
     Machbase Client Query Utility
     Release Version x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries.
     All Rights Reserved
=================================================================
Machbase server address (Default:127.0.0.1) :
Machbase user ID  (Default:SYS)
Machbase User Password :
MACH_CONNECT_MODE=INET, PORT=5656

Mach> LOAD DATA INFILE 'sample/quickstart/load_sample.csv' INTO TABLE load_sample AUTO HEADUSE;
50 row(s) loaded. Failed to load 0 row(s).
Mach> DESC load_sample;
-----------------------------------------------------------------
NAME                          TYPE               LENGTH
-----------------------------------------------------------------
SENSOR_ID                     varchar            32767
EPOCH_TIME                    varchar            32767
E_YEAR                        varchar            32767
E_MONTH                       varchar            32767
E_DAY                         varchar            32767
E_HOUR                        varchar            32767
E_MINUTE                      varchar            32767
E_SECOND                      varchar            32767
VALUE                         varchar            32767
Mach> SELECT COUNT(*) FROM load_sample;
```

```
COUNT(*)
------------------------
50
[1] row(s) selected.
Mach>
```

로그 데이터의 추출

마크베이스는 표준 ANSI SQL 구문을 이용하여 데이터를 추출할 수 있고, 시계열 데이터를 편리하게 다룰 수 있는 확장 구문을 제공한다.

- 데이터 조회
- 시계열 데이터 조회
- 텍스트 검색
- 단순 Join
- 네트워크 데이터 타입 / 연산자

데이터 조회

ANSI 표준 SQL로 데이터를 검색할 수 있다.
아래 예제 모두는 인덱스를 생성하지 않고 검색한 예이다. 즉, 마지막으로 입력된 데이터가 제일 먼저 출력된다.
보다 자세한 내용은 SQL 레퍼런스의 SELECT 부분을 참조하면 된다.

목차

기본 조회

```
SELECT * FROM table_name;
```

```
Mach> SELECT * FROM mach_log;
DEVICE          TM                              TEMP
-----------------------------------------------------------------
MSG
-------------------------------------------------------------------------------
192.168.0.1     NULL                            NULL
NULL
192.168.0.2     2014-06-15 19:50:03 484:382:010 82
error code = 20, critical warning
192.168.0.2     2014-06-15 19:50:03 484:382:008 57
error code = 20
192.168.0.1     2014-06-15 19:50:03 484:382:006 99
error code = 10, critical bug
192.168.0.1     2014-06-15 19:50:03 484:382:004 55
error code = 10
192.168.0.2     2014-06-15 19:50:03 484:382:002 31
normal state
192.168.0.1     2014-06-15 19:50:03 484:382:000 32
normal state
[7] row(s) selected.
Mach>
```

조건 절 조회

```
SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;
```

```
Mach> SELECT * FROM mach_log WHERE device = '192.168.0.1';
DEVICE          TM                              TEMP
-----------------------------------------------------------------
MSG
-------------------------------------------------------------------------------
192.168.0.1     NULL                            NULL
NULL
192.168.0.1     2014-06-15 19:50:36 488:663:006 99
error code = 10, critical bug
192.168.0.1     2014-06-15 19:50:36 488:663:004 55
error code = 10
192.168.0.1     2014-06-15 19:50:36 488:663:000 32
normal state
[4] row(s) selected.

Mach> SELECT * FROM mach_log WHERE device = '192.168.0.1' AND temp > 30 AND temp < 50;
DEVICE          TM                              TEMP
-----------------------------------------------------------------
MSG
-------------------------------------------------------------------------------
```

```
192.168.0.1     2014-06-15 19:50:36 488:663:000 32
normal state
[1] row(s) selected.

Mach> SELECT * FROM mach_log where device > '192.168.0.1';
DEVICE          TM                              TEMP
-----------------------------------------------------------------
MSG
-------------------------------------------------------------------------------------
192.168.0.2     2014-06-15 19:50:36 488:663:010 82
error code = 20, critical warning
192.168.0.2     2014-06-15 19:50:36 488:663:008 57
error code = 20
192.168.0.2     2014-06-15 19:50:36 488:663:002 31
normal state
[3] row(s) selected.

Mach> SELECT * FROM mach_log WHERE msg LIKE '%error%';
DEVICE          TM                              TEMP
-----------------------------------------------------------------
MSG
-------------------------------------------------------------------------------------
192.168.0.2     2014-06-15 19:50:36 488:663:010 82
error code = 20, critical warning
192.168.0.2     2014-06-15 19:50:36 488:663:008 57
error code = 20
192.168.0.1     2014-06-15 19:50:36 488:663:006 99
error code = 10, critical bug
192.168.0.1     2014-06-15 19:50:36 488:663:004 55
error code = 10
[4] row(s) selected.
```

힌트(hint)를 이용한 검색 방향 지정하기

로그 테이블은 일반적으로 최근에 입력한 레코드부터 조회가 가능하다. 가장 먼저 입력한 레코드부터 조회하고 싶을 때에는 힌트를 이용해 조회 방향을 제어할 수 있다.

역방향 검색

기본값이며, /*+ SCAN_BACKWARD(table_name) */ 힌트를 추가하여 조회가 가능하다.

```
Mach> SELECT * FROM LOG;
TIME
---------------------------------
2021-01-04 00:00:00 000:000:000
2021-01-03 00:00:00 000:000:000
2021-01-02 00:00:00 000:000:000
2021-01-01 00:00:00 000:000:000
[4] row(s) selected.
Elapsed time: 0.001

Mach> SELECT /*+ SCAN_BACKWARD(LOG) */ * FROM LOG;
TIME
---------------------------------
2021-01-04 00:00:00 000:000:000
2021-01-03 00:00:00 000:000:000
2021-01-02 00:00:00 000:000:000
2021-01-01 00:00:00 000:000:000
[4] row(s) selected.
Elapsed time: 0.001
```

정방향 검색

/*+ SCAN_FORWARD(table_name) */ 힌트를 추가하여 조회가 가능하다.

```
Mach> SELECT /*+ SCAN_FORWARD(LOG) */ * FROM LOG;
TIME
---------------------------------
```

```
2021-01-01 00:00:00 000:000:000
2021-01-02 00:00:00 000:000:000
2021-01-03 00:00:00 000:000:000
2021-01-04 00:00:00 000:000:000
[4] row(s) selected.
Elapsed time: 0.001
```

기본 스캔 방향 프로퍼티로 설정

TABLE_SCAN_DIRECTION 프로퍼티로 SELECT 문에 힌트가 없을 때 로그 테이블의 스캔 방향을 설정할 수 있다.

시계열 데이터 조회

SELECT문의 DURATION절은 검색 대상 시간 조건절을 정의한다. DURATION절을 이용하는 가장 큰 이유는 검색 대상을 줄여서 대량의 데이터를 검색하더라도 성능을 향상시키기 위함이다.

마크베이스는 입력 시간을 기준으로 데이터를 파티션하여 저장하므로, 시간 조건으로 데이터를 쉽게 검색하도록 하였다. 입력 시간은 사용자가 정의한 칼럼이 아니라 '_ARRIVAL_TIME'이라는 자동 생성 칼럼에 저장된다. 그러므로 마크베이스를 가장 효율적으로 사용하기 위해서는 추가로 시간 칼럼을 지정하지 않고 내장된 '_ARRIVAL_TIME' 칼럼을 이용하는 것이 좋다.

마크베이스는 입력된 순서의 역순으로 데이터를 출력한다. 즉, 최근 데이터가 먼저, 오래전 데이터가 나중에 출력되는 것이다. 일반적으로 시계열 데이터를 검색할 때, 최근 데이터가 더 중요하고 먼저 얻어야 하는 경우가 많으므로 이와 같은 순서로 출력한다. 또한 모든 DURATION조건절에 의한 데이터 출력은 최근에서 과거 순으로 출력된다. 과거에서 최근 순으로 출력하려면 AFTER 절을 이용하여야 한다. 문법은 다음과 같다.

목차

문법

```
DURATION    time_expression [BEFORE time_expression | TO_DATE(time) ];
DURATION    time_expression [AFTER TO_DATE(time)];
time_expression
 -  ALL
 -  n   year
 -  n   month
 -  n   week
 -  n   day
 -  n   hour
 -  n   minute
 -  n   second
```

DURATION...BEFORE

앞서 말한 것 처럼, BEFORE를 명시적 이용하거나 정의되지 않은 경우(자동으로 BEFORE를 적용)에는 최근에서 과거 순으로 데이터를 출력한다.

절대 시간 값 또는 상대 시간 값을 기준으로 데이터를 조회할 수 있다.

절대 시간 값 기준 검색

```
Mach> CREATE TABLE time_table (id INTEGER);
Created successfully.

Mach> INSERT INTO time_table(_arrival_time, id) VALUES(TO_DATE('2014-6-12 10:00:00', 'YYYY-MM-DD HH24:MI:SS'), 1);
1 row(s) inserted.

Mach> INSERT INTO time_table(_arrival_time, id) VALUES(TO_DATE('2014-6-12 11:00:00', 'YYYY-MM-DD HH24:MI:SS'), 2);
1 row(s) inserted.

Mach> INSERT INTO time_table(_arrival_time, id) VALUES(TO_DATE('2014-6-12 12:00:00', 'YYYY-MM-DD HH24:MI:SS'), 3);
1 row(s) inserted.

Mach> INSERT INTO time_table(_arrival_time, id) VALUES(TO_DATE('2014-6-12 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 4);
1 row(s) inserted.

Mach> INSERT INTO time_table VALUES(5);
1 row(s) inserted.

Mach> SELECT _arrival_time, * FROM time_table DURATION 1 MINUTE;
_arrival_time                   ID
-----------------------------------------------
2017-02-16 12:17:01 880:937:028 5
[1] row(s) selected.

Mach> SELECT _arrival_time, * FROM time_table DURATION 1 DAY BEFORE TO_DATE('2014-6-12 12:00:00', 'YYYY-MM-DD HH24
_arrival_time                   ID
-----------------------------------------------
2014-06-12 12:00:00 000:000:000 3
```

```
2014-06-12 11:00:00 000:000:000 2
2014-06-12 10:00:00 000:000:000 1
[3] row(s) selected.
```

상대 시간 값 기준 검색

상대 시간 값을 기준으로 한 검색은, 바로 현재를 기준으로 한 검색으로 볼 수 있다.

```
Mach> CREATE TABLE relative_table(id INTEGER);
Created successfully.

Mach> INSERT INTO relative_table values(1);
1 row(s) inserted.

------ WAIT for 30 SECONDS before the second value ------

Mach> INSERT INTO relative_table values(2);
1 row(s) inserted.

Mach> SELECT _arrival_time, * FROM relative_table;
_arrival_time                   ID
-----------------------------------------------
2017-02-16 12:35:34 476:055:014 2
2017-02-16 12:35:04 430:802:356 1
[2] row(s) selected.

Mach> SELECT id FROM relative_table DURATION 30 second ;
id
--------------
2
[1] row(s) selected.

Mach> SELECT id FROM relative_table DURATION 60 second ;
id
--------------
2
1
[2] row(s) selected.

Mach> SELECT id FROM relative_table DURATION 30 second BEFORE 30 second;
id
--------------
1
[1] row(s) selected.
```

DURATION...AFTER

AFTER를 적용한 경우, 데이터는 과거에서 최근순으로 출력된다.

BEFORE명령은, 최근에서 과거로 출력하는것에 비교하면 데이터가 입력 시간을 기준으로 자동으로 역순으로 출력된다.

```
Mach> CREATE TABLE after_table (id INTEGER);
Created successfully.

Mach> INSERT INTO after_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 10:00:00', 'YYYY-MM-DD HH24:MI:SS'), 1);
1 row(s) inserted.

Mach> INSERT INTO after_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 11:00:00', 'YYYY-MM-DD HH24:MI:SS'), 2);

Mach> INSERT INTO after_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 12:00:00', 'YYYY-MM-DD HH24:MI:SS'), 3);
1 row(s) inserted.

Mach> INSERT INTO after_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 4);
1 row(s) inserted.

Mach> INSERT INTO after_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 14:00:00', 'YYYY-MM-DD HH24:MI:SS'), 5);
1 row(s) inserted.
```

```
Mach> select _arrival_time, * from after_table duration ALL after TO_DATE('2016-6-12 11:00:00', 'YYYY-MM-DD HH24:MI

_arrival_time                   ID
-----------------------------------------------
2016-06-12 11:00:00 000:000:000 2
2016-06-12 12:00:00 000:000:000 3
2016-06-12 13:00:00 000:000:000 4
2016-06-12 14:00:00 000:000:000 5
[4] row(s) selected.

Mach> select _arrival_time, * from after_table duration ALL before TO_DATE('2016-6-12 13:00:00', 'YYYY-MM-DD HH24:M
_arrival_time                   ID
-----------------------------------------------
2016-06-12 13:00:00 000:000:000 4
2016-06-12 12:00:00 000:000:000 3
2016-06-12 11:00:00 000:000:000 2
2016-06-12 10:00:00 000:000:000 1
[4] row(s) selected.
```

DURATION...FROM/TO

사용자가 두개의 절대 시간을 기준으로 데이터를 검색하려고 할 때, "DURATION FROM A TO B" 형태의 조건절을 이용한다.

A와 B는 절대적 시간이며 TO_DATE함수를 이용하여 표현된다. A와 B는 사용자의 의도에 따라 다르게 설정될 수 있다. 예를 들어,

- A가 B보다 이후의 시간일 경우 BEFORE를 사용한 것과 같이, 검색 방향은 최근에서 과거 순으로 데이터를 출력한다.
- B가 A보다 과거인 경우 AFTER를 사용한 것과 같이, 검색 방향은 과거에서 최근 순으로 데이터를 출력한다.

아래의 예제를 보면 데이터의 출력 방식을 쉽게 이해할 수 있다.

```
Mach> CREATE TABLE from_table (id INTEGER);
Created successfully.

Mach> INSERT INTO from_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 10:00:00', 'YYYY-MM-DD HH24:MI:SS'), 1);
1 row(s) inserted.

Mach> INSERT INTO from_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 11:00:00', 'YYYY-MM-DD HH24:MI:SS'), 2);
1 row(s) inserted.

Mach> INSERT INTO from_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 12:00:00', 'YYYY-MM-DD HH24:MI:SS'), 3);
1 row(s) inserted.

Mach> INSERT INTO from_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 4);
1 row(s) inserted.

Mach> INSERT INTO from_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 14:00:00', 'YYYY-MM-DD HH24:MI:SS'), 5);
1 row(s) inserted.

Mach> INSERT INTO from_table(_arrival_time, id) VALUES(TO_DATE('2016-6-12 15:00:00', 'YYYY-MM-DD HH24:MI:SS'), 6);
1 row(s) inserted.

Mach> SELECT _arrival_time, * FROM from_table DURATION FROM TO_DATE('2016-6-12 12:00:00', 'YYYY-MM-DD HH24:MI:SS')
_arrival_time                   ID
-----------------------------------------------
2016-06-12 12:00:00 000:000:000 3
2016-06-12 13:00:00 000:000:000 4
2016-06-12 14:00:00 000:000:000 5
[3] row(s) selected.

Mach> SELECT _arrival_time, * FROM from_table limit 2 DURATION FROM TO_DATE('2016-6-12 12:00:00', 'YYYY-MM-DD HH24
'YYYY-MM-DD HH24:MI:SS');
_arrival_time                   ID
-----------------------------------------------
2016-06-12 12:00:00 000:000:000 3
2016-06-12 13:00:00 000:000:000 4
[2] row(s) selected.

Mach> SELECT _arrival_time, * FROM from_table DURATION FROM TO_DATE('2016-6-12 15:00:00', 'YYYY-MM-DD HH24:MI:SS')
```

```
_arrival_time                   ID
-----------------------------------------------
2016-06-12 15:00:00 000:000:000 6
2016-06-12 14:00:00 000:000:000 5
2016-06-12 13:00:00 000:000:000 4
2016-06-12 12:00:00 000:000:000 3
[4] row(s) selected.

Mach> SELECT _arrival_time, * FROM from_table LIMIT 2 duration FROM TO_DATE('2016-6-12 15:00:00', 'YYYY-MM-DD HH24
'YYYY-MM-DD HH24:MI:SS');
_arrival_time                   ID
-----------------------------------------------
2016-06-12 15:00:00 000:000:000 6
2016-06-12 14:00:00 000:000:000 5
[2] row(s) selected.

Mach> SELECT _arrival_time, * from from_table duration FROM TO_DATE('2016-6-12 13:00:00', 'YYYY-MM-DD HH24:MI:SS')
_arrival_time                   ID
-----------------------------------------------
2016-06-12 13:00:00 000:000:000 4
[1] row(s) selected.

Mach> SELECT _arrival_time, * from from_table duration FROM TO_DATE('2016-6-12 13:00:00', 'YYYY-MM-DD HH24:MI:SS')
_arrival_time                   ID
-----------------------------------------------
2016-06-12 13:00:00 000:000:000 4
2016-06-12 14:00:00 000:000:000 5
2016-06-12 15:00:00 000:000:000 6
[3] row(s) selected.

Mach> SELECT _arrival_time, * from from_table duration FROM TO_DATE('2016-6-12 20:00:00', 'YYYY-MM-DD HH24:MI:SS')
_arrival_time                   ID
-----------------------------------------------
2016-06-12 15:00:00 000:000:000 6
2016-06-12 14:00:00 000:000:000 5
2016-06-12 13:00:00 000:000:000 4
[3] row(s) selected.
```

텍스트 검색

이 문서는 키워드 인덱스를 이용한 텍스트 검색을 다룬다.

텍스트 검색은 "reverse index"라는 특수한 종류의 인덱스를 탐색하여 원하는 문자열 패턴을 검색하기 때문에, 일반적인 DBMS의 LIKE검색과 비교할 수 없을 정도로 빠른 성능을 낸다. 키워드 인덱스는 가변길이 문자형 칼럼인 varchar 타입과 text 타입 칼럼에 대해서만 생성할 수 있다. 단, 검색 대상 문자열이 반드시 정확히 일치해야 한다. 마크베이스는 특수문자를 기반한 키워드나, 형태소 분석등을 수행하지는 않는다.

목차

SEARCH

```
SELECT  column_name(s)
FROM    table_name
WHERE   column_name
SEARCH  pattern;
```

```
Mach> CREATE TABLE search_table (id INTEGER, name VARCHAR(20));
Created successfully.

Mach> CREATE INDEX idx_SEARCH ON SEARCH_table (name) INDEX_TYPE KEYWORD;
Created successfully.

Mach> INSERT INTO search_table VALUES(1, 'time flys');
1 row(s) inserted.

Mach> INSERT INTO search_table VALUES(1, 'time runs');
1 row(s) inserted.

Mach> SELECT * FROM search_table WHERE name SEARCH 'time' OR name SEARCH 'runs2' ;
ID          NAME
-------------------------------------
1           time runs
1           time flys
[2] row(s) selected.

Mach> SELECT * FROM search_table WHERE name SEARCH 'time' AND name SEARCH 'runs2' ;
ID          NAME
-------------------------------------
[0] row(s) selected.

Mach> SELECT * FROM search_table WHERE name SEARCH 'flys' OR name SEARCH 'runs2' ;
ID          NAME
-------------------------------------
1           time flys
[1] row(s) selected.
```

다중 언어 검색

마크베이스는 ASCII와 UTF-8로 저장된 여러 가지 종류의 언어의 가변길이 문자열에 대한 검색이 가능하다. 한국어나 일본어와 같은 언어의 문장에서 일부분만을 검색하기 위해서, 2-gram 기법을 이용한다.

```
SELECT  column_name(s)
FROM    table_name
WHERE   column_name
SEARCH  pattern;
```

```
Mach> CREATE TABLE multi_table (message varchar(100));
Created successfully.

Mach> CREATE INDEX idx_multi ON multi_table(message)INDEX_TYPE KEYWORD;
Created successfully.

Mach> INSERT INTO multi_table VALUES("Machbase is the combination of ideal solutions");
1 row(s) inserted.

Mach> INSERT INTO multi_table VALUES("Machbase is a columnar DBMS");
1 row(s) inserted.

Mach> INSERT INTO multi_table VALUES("Machbaseは理想的なソリューションの組み合わせです");
1 row(s) inserted.

Mach> INSERT INTO multi_table VALUES("Machbaseは円柱状のDBMSです");
1 row(s) inserted.

Mach>  SELECT * from multi_table WHERE message SEARCH 'Machbase DBMS';
MESSAGE
----------------------------------------------------------------------------------
Machbaseは円柱状のDBMSです
Machbase is a columnar DBMS
[2] row(s) selected.

Mach> SELECT * from multi_table WHERE message SEARCH 'DBMS is';
MESSAGE
----------------------------------------------------------------------------------
Machbase is a columnar DBMS
[1] row(s) selected.

Mach> SELECT * from multi_table WHERE message SEARCH 'DBMS' OR message SEARCH 'ideal';
MESSAGE
----------------------------------------------------------------------------------
Machbaseは円柱状のDBMSです
Machbase is a columnar DBMS
Machbase is the combination of ideal solutions
[3] row(s) selected.

Mach> SELECT * from multi_table WHERE message SEARCH '組み合わせ';
MESSAGE
----------------------------------------------------------------------------------
Machbaseは理想的なソリューションの組み合わせです
[1] row(s) selected.
Elapsed time: 0.001
Mach> SELECT * from multi_table WHERE message SEARCH '円柱';
MESSAGE
----------------------------------------------------------------------------------
Machbaseは円柱状のDBMSです
[1] row(s) selected.
```

입력된 데이터가 "대한민국" 인 경우, "대한", "한민", "민국"의 세 단어가 인덱스에 기록된다. 그러므로 "대한" 또는 "민국" 키워드로도 "대한민국"을 검색할 수 있다.

기본적으로 search문에서 입력받은 키워드들은 AND조건으로 검색되므로, 세 단어만 입력하더라도 결과는 매우 정확하게 표시된다. 예를 들어, 검색 대상 키워드가 "computer utilization guide"인 경우, 세 단어 "computer", "utilization", "guide"가 AND 조건으로 설정되므로 세 단어가 한 데이터에서 모두 사용된 칼럼값만 표시된다.

ESEARCH

ESEARCH 연산자는 검색 대상 키워드를 확장하여 검색하기 위해 사용된다. 검색 대상 키워드는 반드시 ASCII여야 한다. 검색 키워드를 %문자를 이용하여 설정할 수 있다. LIKE조건절처럼 %문자로 시작되는 키워드를 이용하면, 모든 레코드를 검색해야 하지만 키워드 색인 내의 단어들에 대해서 이 조건을 검색하기 때문에, LIKE보다 빠르게 검색할 수 있다. 이 기능은 알파벳 문자열(에러 문장 또는 코드등)을 빠르게 검색할 경우에 유용하다.

```
SELECT  column_name(s)
FROM    table_name
WHERE   column_name
ESEARCH pattern;
```

```
Mach> CREATE TABLE esearch_table(id INTEGER, name VARCHAR(20), data VARCHAR(40));
```

```
Created successfully.

Mach> CREATE INDEX idx1 ON esearch_table(name) INDEX_TYPE KEYWORD;
Created successfully.

Mach> CREATE INDEX idx2 ON esearch_table(data) INDEX_TYPE KEYWORD;
Created successfully.

Mach> INSERT INTO esearch_table VALUES(1, 'machbase', 'Real-time search technology');
1 row(s) inserted.

Mach> INSERT INTO esearch_table VALUES(2, 'mach2flux', 'Real-time data compression');
1 row(s) inserted.

Mach> INSERT INTO esearch_table VALUES(3, 'DB MS', 'Memory cache technology');
1 row(s) inserted.

Mach> INSERT INTO esearch_table VALUES(4, 'ファ ッションアドバイザー、', 'errors');
1 row(s) inserted.

Mach> INSERT INTO esearch_table VALUES(5, '인피 니 플럭스', 'socket232');
1 row(s) inserted.

Mach> SELECT * FROM esearch_table WHERE name ESEARCH '%mach%';
ID          NAME                DATA
-------------------------------------------------------------------------------
2           mach2flux           Real-time data compression
1           machbase            Real-time search technology

Mach> SELECT * FROM esearch_table where data ESEARCH '%echn%';
ID          NAME                DATA
-------------------------------------------------------------------------------
3           DB MS                Memory cache technology
1           machbase           Real-time search technology
[2] row(s) selected.

Mach> SELECT * FROM esearch_table where name ESEARCH '%피니%럭스';
ID          NAME                DATA
-------------------------------------------------------------------------------
[0] row(s) selected.

Mach> SELECT * FROM esearch_table where data ESEARCH '%232';
ID          NAME                DATA
-------------------------------------------------------------------------------
5           인피 니 플럭스  socket232
[1] row(s) selected.
```

REGEXP

REGEXP 연산자는 정규표현식을 통하여 데이터에 대한 텍스트 검색을 수행하기 위해서 사용된다. REGEXP 연산자는 대상 칼럼에 정규표현식을 수행하여 실행되며, 색인을 사용할 수 없기 때문에, 검색 성능이 저하될 수 있다. 따라서 검색 속도를 향상시키기 위해 색인을 사용할 수 있는 다른 검색 조건을 AND 연산자로 추가하여 사용하는 것이 좋다.

특정한 정규표현식 패턴으로 검색하기 전에 인덱스를 사용할 수 있는 SEARCH 또는 ESEARCH 연산자를 먼저 적용하고, 결과집합을 축소시킨 다음 REGEXP를 사용하는 것이 검색 성능을 향상시킬 수 있는 방법이다.

```
Mach> CREATE TABLE regexp_table(id INTEGER, name VARCHAR(20), data VARCHAR(40));
Created successfully.

Mach> INSERT INTO regexp_table VALUES(1, 'machbase', 'Real-time search technology');
1 row(s) inserted.

Mach> INSERT INTO regexp_table VALUES(2, 'mach2base', 'Real-time data compression');
1 row(s) inserted.

Mach> INSERT INTO regexp_table VALUES(3, 'DBMS', 'Memory cache technology');
1 row(s) inserted.
```

```
Mach> INSERT INTO regexp_table VALUES(4, 'ファ ッショ', 'errors');
1 row(s) inserted.

Mach> INSERT INTO regexp_table VALUES(5, '인피니플럭스', 'socket232');
1 row(s) inserted.

Mach> SELECT * FROM regexp_table WHERE name REGEXP 'mach';
ID          NAME                  DATA
------------------------------------------------------------------------------
2           mach2base           Real-time data compression
1           machbase            Real-time search technology
[2] row(s) selected.

Mach> SELECT * FROM regexp_table WHERE data REGEXP 'mach[1]';
ID          NAME                  DATA
------------------------------------------------------------------------------
[0] row(s) selected.

Mach> SELECT * FROM regexp_table WHERE data REGEXP '[A-Za-z]';
ID          NAME                  DATA
------------------------------------------------------------------------------
5           인피니플럭스  socket232
4           ファ ッショ      errors
3           DBMS                  Memory cache technology
2           mach2base           Real-time data compression
1           machbase            Real-time search technology
[5] row(s) selected.
```

LIKE

마크베이스는 SQL표준의 LIKE연산자도 지원한다. LIKE연산자에 한국어, 일본어, 중국어도 사용 가능하다.

```
SELECT  column_name(s)
FROM    table_name
WHERE   column_name
LIKE    pattern;
```

Example:

```
Mach> CREATE TABLE like_table (id INTEGER, name VARCHAR(20), data VARCHAR(40));
Created successfully.

Mach> INSERT INTO like_table VALUES(1, 'machbase', 'Real-time search technology');
1 row(s) inserted.

Mach> INSERT INTO like_table VALUES(2, 'mach2base', 'Real-time data compression');
1 row(s) inserted.

Mach> INSERT INTO like_table VALUES(3, 'DBMS', 'Memory cache technology');
1 row(s) inserted.

Mach> INSERT INTO like_table VALUES(4, 'ファ ッションアドバイザー、', 'errors');
1 row(s) inserted.

Mach> INSERT INTO like_table VALUES(5, '인피 니 플럭스', 'socket232');
1 row(s) inserted.

Mach> SELECT * FROM like_table WHERE name LIKE 'mach%';
ID          NAME                  DATA
------------------------------------------------------------------------------
2           mach2base           Real-time data compression
1           machbase            Real-time search technology
[2] row(s) selected.

Mach> SELECT * FROM like_table WHERE name LIKE '%니%';
ID          NAME                  DATA
------------------------------------------------------------------------------
```

```
5           인피 니 플럭스  socket232
[1] row(s) selected.

Mach> SELECT * FROM like_table WHERE data LIKE '%technology';
ID          NAME                  DATA
------------------------------------------------------------------------------
3           DBMS                  Memory cache technology
1           machbase            Real-time search technology
[2] row(s) selected.
```

단순 Join

로그 테이블, 휘발성 테이블, 참조 테이블과 메타 테이블은 Join 하여 검색할 수 있다.

목차

단순 Join

```
Mach> CREATE TABLE logtable (code INT,value INT);
Created successfully.

Mach> INSERT INTO logtable VALUES(1,20 );
1 row(s) inserted.

Mach> INSERT INTO logtable VALUES(2,10 );
1 row(s) inserted.

Mach> INSERT INTO logtable VALUES(3,15 );
1 row(s) inserted.

Mach> INSERT INTO logtable VALUES(4,20 );
1 row(s) inserted.

Mach> INSERT INTO logtable VALUES(5,10 );
1 row(s) inserted.

Mach> CREATE VOLATILE table VTABLE (code INT,name VARCHAR(32));
Created successfully.

Mach> INSERT INTO vtable VALUES(1, 'Sam');
1 row(s) inserted.

Mach> INSERT INTO vtable VALUES(3, 'Thomas');
1 row(s) inserted.

Mach> INSERT INTO vtable VALUES(5, 'Micheal');
1 row(s) inserted.

Mach> INSERT INTO vtable VALUES(7, 'Jessica');
1 row(s) inserted.

Mach> SELECT name,value FROM logtable, vtable WHERE logtable.code=vtable.code;
name                              value
-----------------------------------------------
Micheal                           10
Thomas                            15
Sam                               20
[3] row(s) selected.
```

Alias 를 이용한 Join

Join을 사용할 때, join 대상 테이블에 alias를 사용할 수 있다.

```
SELECT c.name FROM m$sys_tables t, m$sys_columns c WHERE t.id = c.table_id AND t.name = 'T1'
AND c.id NOT IN(0, 65534) ORDER BY c.name;

c.name
-------------------------------------------
ADDR
ISTYPE
SRCIP
[3] row(s) selected.
```

GROUP BY/ORDER BY 사용

GROUP BY, ORDER BY 와 집계 함수도 사용 가능하다.

```
Mach> SELECT t.name, COUNT(c.name) FROM m$sys_columns c, m$sys_tables t WHERE t.id = c.table_id GROUP BY t.name ORI
t.name                                   count(c.name)
-------------------------------------------------------------------
COMMON_TABLE                             5
DURATIONT                                3
[2] row(s) selected.
```

조건절 없는 Join

JOIN 조건절이 없는 join 질의는 에러를 발생시킨다. 로그 테이블에 너무나 많은 데이터가 있기 때문에, join 조건절이 없는 질의의 속도는 예측할 수 없을 정도로 느리기 때문이다.

또한, 두개의 로그 테이블 join은 매우 성능이 느릴 수 있다. 그래서 데이터베이스를 설계할 때, 역정규화(denormalization)를 고려하여 join이 발생하지 않도록 설계하는 것이 좋다.

```
Mach> CREATE TABLE log_table1(i1 INTEGER);
Created successfully.
Mach> INSERT INTO log_table1 VALUES(1);
1 row(s) inserted.
Mach> INSERT INTO log_table1 VALUES(20);
1 row(s) inserted.
Mach> INSERT INTO log_table1 VALUES(30);
1 row(s) inserted.


Mach>CREATE TABLE log_table2(i1 INTEGER);
Created successfully.
Mach> INSERT INTO log_table2 VALUES(1);
1 row(s) inserted.
Mach> INSERT INTO log_table2 VALUES(30);
1 row(s) inserted.
Mach> INSERT INTO log_table2 VALUES(50);
1 row(s) inserted.

Mach> SELECT log_table1.i1 FROM log_table1, log_table2;
[ERR-02101 : Error in joining tables. Cannot join without join predicate.]

Mach> SELECT log_table1.i1 FROM log_table1, log_table2 where log_table1.i1 = 1;
[ERR-02101 : Error in joining tables. Cannot join without join predicate.]

Mach> SELECT log_table1.i1 from log_table1, log_table2 WHERE log_table1.i1 = log_table2.i1;
i1
--------------
30
1
[2] row(s) selected.
```

Inner Join / Outer Join

ANSI 타입의 INNER, LEFT OUTER, RIGHT OUTER join을 사용할 수 있으나 FULL OUTER JOIN은 사용할 수 없다.

```
FROM    TABLE_1 [INNER|LEFT OUTER|RIGHT OUTER]  JOIN    TABLE_2 ON  expression
```

```
SELECT t1.i1 t2.i1 FROM t1 LEFT OUTER JOIN t2 ON (t1.i1 = t2.i1) WHERE t2.i2 = 1;
```

위 질의는 where절의 t2.i2 = 1 조건에 의해서 Inner Join 으로 변경되어 실행된다.

네트워크 데이터 타입 / 연산자

마크베이스는 네트워크 데이터 타입을 지원함과 동시에 SELECT 문에서 사용할 수 있는 함수들을 지원한다.

- IPv4 형식 : 4바이트 주소 타입
- IPv6 형식 : 16바이트 주소 타입
- 네트워크 마스크 : IPv4 또는 IPv6 에 대한 네트워크 마스크 지정 형식(/비트수)

목차

IPv4

INSERT

```
INSERT INTO table_name VALUES (value1,value2,value3,...);
```

```
CREATE TABLE addrtable (addr IPV4);
INSERT  INTO addrtable VALUES ('127.0.0.1');
INSERT  INTO addrtable VALUES ('127.0' || '.0.2');
INSERT  INTO addrtable VALUES ('127.0.0.3');
INSERT  INTO addrtable VALUES ('127.0.0.4');
INSERT  INTO addrtable VALUES ('127.0.0.5');
INSERT  INTO addrtable VALUES ('255.255.255.255');
```

SELECT

```
SELECT  column_name,column_name FROM    table_name;
```

```
Mach> SELECT addr FROM addrtable WHERE addr = '127.0.0.3' or addr = '127.0.0.5';
addr
------------------
127.0.0.5
127.0.0.3
[2] row(s) selected.

Mach> SELECT addr FROM addrtable WHERE addr > '127.0.0.3' AND addr < '127.0.0.5';
addr
------------------
127.0.0.4
[1] row(s) selected.

Mach> SELECT addr FROM addrtable WHERE addr <> '127.0.0.3';
addr
------------------
255.255.255.255
127.0.0.5
127.0.0.4
127.0.0.2
127.0.0.1
[5] row(s) selected.

Mach> SELECT addr FROM addrtable WHERE addr = '127.0.0.*';
addr
------------------
127.0.0.5
127.0.0.4
127.0.0.3
127.0.0.2
127.0.0.1
[5] row(s) selected.
```

```
Mach> SELECT addr FROM addrtable WHERE addr = '*.0.0.*';
addr
------------------
127.0.0.5
127.0.0.4
127.0.0.3
127.0.0.2
127.0.0.1
[5] row(s) selected.
```

IPv6

INSERT

```
INSERT  INTO    table_name  VALUES  (value1,value2,value3,...);
```

```
CREATE TABLE addrtable6 (addr ipv6);
INSERT INTO addrtable6 VALUES ('::0.0.0.0');
INSERT INTO addrtable6 VALUES ('::127.0' || '.0.1');
INSERT INTO addrtable6 VALUES ('::127.0.0.3');
INSERT INTO addrtable6 VALUES ('::127.0.0.4');
INSERT INTO addrtable6 VALUES ('21DA:D3:0:2F3B:2AA:FF:FE28:9C5A');
INSERT INTO addrtable6 VALUES ('::FFFF:255.255.255.255');
```

SELECT

```
SELECT  column_name,column_name FROM    table_name;
```

```
Mach> SELECT addr FROM addrtable6 WHERE addr = '::127.0.0.3' or addr = '::127.0.0.5';
addr
-----------------------------------------------------------------
::127.0.0.3
[1] row(s) selected.

Mach> SELECT addr FROM addrtable6 WHERE addr > '::127.0.0.3' and addr < '::127.0.0.5';
addr
-----------------------------------------------------------------
::127.0.0.4
[1] row(s) selected.

Mach> SELECT addr FROM addrtable6 WHERE addr <> '::127.0.0.3';
addr
-----------------------------------------------------------------
::ffff:255-255.255.255
21da:d3::2f3b:2aa:ff:fe28:9c5a
::127.0.0.4
::127.0.0.1
::
[5] row(s) selected.

Mach> SELECT addr FROM addrtable6 WHERE addr >= '21DA::';
addr
-----------------------------------------------------------------
21da:d3::2f3b:2aa:ff:fe28:9c5a
[1] row(s) selected.

Mach> SELECT addr FROM addrtable6 order by addr desc;
addr
-----------------------------------------------------------------
21da:d3::2f3b:2aa:ff:fe28:9c5a
```

```
::ffff:255.255.255.255
::127.0.0.4
::127.0.0.3
::127.0.0.1
::
[6] row(s) selected.
```

네트워크 마스크

네트워크 마스크는 특정 주소가 특정한 네트워크에 포함되는지를 지정하는 표현 형식이다. 마크베이스는 네트워크 마스크 타입과 관련 연산자를 지원한다.

마스크의 표현 형태

일반 네트워크 표현과 마찬가지로 네트워크 주소 마지막에 / 기호와 비트 개수를 표현하는 형식으로 나타낸다.

```
'192.128.0.0/16'
'FFFF::192.128.99.0/32'
```

마스크 연산자

CONTAINS

이 연산자는 왼쪽에 네트워크 마스크와 오른쪽에 네트워크 주소 데이터 타입이 나와야 한다. 즉 입력된 주소가 주어진 네트워크 마스크에 포함되는지를 검사한다. NOT 연산자도 함께 사용할 수 있다.

```
SELECT addr FROM addrtable  WHERE '192.0.0.0/16'    CONTAINS    addr;
SELECT addr FROM addrtable  WHERE '192.128.99.0/32' NOT CONTAINS    addr;
```

CONTAINED

CONSTAINS와 반대로, 네트워크 주소가 왼쪽, 네트워크 마스크가 오른쪽이다. 왼쪽 주소가 오른쪽 마스크의 일부인지를 검사한다.

```
SELECT addr FROM addrtable  WHERE addr CONTAINED '192.0.0.0/16';
SELECT addr FROM addrtable  WHERE addr NOT CONTAINED '192.128.99.0/32';
```

마스크 사용 예제

네트워크 마스크 타입을 이용한 검색의 예는 다음과 같다.

```
CREATE TABLE ip_table (addr4 IPV4, addr6 IPV6);

INSERT INTO ip_table VALUES ('192.0.0.1','FFFF::192.0.0.1');
INSERT INTO ip_table VALUES ('192.0.10.1','FFFF::192.0.10.1');
INSERT INTO ip_table VALUES ('192.128.0.1','FFFF::192.128.0.1');
INSERT INTO ip_table VALUES ('192.128.99.128','FFFF::192.128.99.128');
INSERT INTO ip_table VALUES ('192.128.99.64','FFFF::192.128.99.64');
INSERT INTO ip_table VALUES ('192.128.99.32','FFFF::192.128.99.32');
INSERT INTO ip_table VALUES ('192.128.99.16','FFFF::192.128.99.16');
INSERT INTO ip_table VALUES ('192.128.99.8','FFFF::192.128.99.8');
INSERT INTO ip_table VALUES ('192.128.99.4','FFFF::192.128.99.4');
INSERT INTO ip_table VALUES ('192.128.99.2','FFFF::192.128.99.2');
INSERT INTO ip_table VALUES ('192.128.99.1','FFFF::192.128.99.1');

Mach> SELECT addr4 FROM ip_table WHERE '192.0.0.0/16' CONTAINS addr4;
addr4
-----------
192.0.10.1
192.0.0.1
[2] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE '192.128.0.0/16' CONTAINS addr4;
```

```
addr4
-----------
192.128.99.1
192.128.99.2
192.128.99.4
192.128.99.8
192.128.99.16
192.128.99.32
192.128.99.64
192.128.99.128
192.128.0.1
[9] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE '192.0.10.0/24' CONTAINS addr4;
addr4
----------------------------------------------------------------
192.0.10.1
[1] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE '192.128.99.0/31' CONTAINS addr4;
addr4
-------------------------------------------------------
192.128.99.1
[1] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE '192.128.99.0/32' NOT CONTAINS addr4;
addr4
-----------
192.128.99.1
192.128.99.2
192.128.99.4
192.128.99.8
192.128.99.16
192.128.99.32
192.128.99.64
192.128.99.128
192.128.0.1
192.0.10.1
192.0.0.1
[11] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE addr4 CONTAINED '192.0.0.0/16';
addr4
-------------------------------------
192.0.10.1
192.0.0.1
[2] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE addr4 CONTAINED '192.128.0.0/16';
addr4
-------------------------------------
192.128.99.1
192.128.99.2
192.128.99.4
192.128.99.8
192.128.99.16
192.128.99.32
192.128.99.64
192.128.99.128
192.128.0.1
[9] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE addr4 CONTAINED '192.0.10.0/24';
addr4
----------------------------
192.0.10.1
[1] row(s) selected.

Mach> SELECT addr4 FROM ip_table WHERE addr4 not CONTAINED '192.128.99.0/32';
addr4
```

```
------------------------------------------------
192.128.99.1
192.128.99.2
192.128.99.4
192.128.99.8
192.128.99.16
192.128.99.32
192.128.99.64
192.128.99.128
192.128.0.1
192.0.10.1
192.0.0.1
[11] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE 'FFFF::192.0.0.0/104' CONTAINS addr6;
addr6
------------------------------------
ffff::c080:6301
ffff::c080:6302
ffff::c080:6304
ffff::c080:6308
ffff::c080:6310
ffff::c080:6320
ffff::c080:6340
ffff::c080:6380
ffff::c080:1
ffff::c000:a01
ffff::c000:1
[11] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE 'FFFF::192.128.0.0/112' CONTAINS addr6;
addr6
------------------------------------
ffff::c080:6301
ffff::c080:6302
ffff::c080:6304
ffff::c080:6308
ffff::c080:6310
ffff::c080:6320
ffff::c080:6340
ffff::c080:6380
ffff::c080:1
[9] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE 'FFFF::192.0.10.0/120' CONTAINS addr6;
addr6
------------------------------------------------
ffff::c000:a01
[1] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE 'FFFF::192.128.99.0/31' CONTAINS addr6;
addr6
------------------------------------------------
ffff::c080:6301
ffff::c080:6302
ffff::c080:6304
ffff::c080:6308
ffff::c080:6310
ffff::c080:6320
ffff::c080:6340
ffff::c080:6380
ffff::c080:1
ffff::c000:a01
ffff::c000:1
[11] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE 'FFFF::192.128.99.0/32' not CONTAINS addr6;
addr6
------------------------------------
[0] row(s) selected.
```

```
Mach> SELECT addr6 FROM ip_table WHERE addr6 CONTAINED 'FFFF::192.0.0.0/104';
addr6
-------------------------------------
ffff::c080:6301
ffff::c080:6302
ffff::c080:6304
ffff::c080:6308
ffff::c080:6310
ffff::c080:6320
ffff::c080:6340
ffff::c080:6380
ffff::c080:1
ffff::c000:a01
ffff::c000:1
[11] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE addr6 CONTAINED 'FFFF::192.128.0.0/112';
addr6
-------------------------------------
ffff::c080:6301
ffff::c080:6302
ffff::c080:6304
ffff::c080:6308
ffff::c080:6310
ffff::c080:6320
ffff::c080:6340
ffff::c080:6380
ffff::c080:1
[9] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE addr6 CONTAINED 'FFFF::192.0.10.0/120';
addr6
-------------------------------------
ffff::c000:a01
[1] row(s) selected.

Mach> SELECT addr6 FROM ip_table WHERE addr6 not CONTAINED 'FFFF::192.128.99.0/32';
addr6
-------------------------------------
[0] row(s) selected.
```

로그 데이터의 삭제

마크베이스에서의 DELETE 구문은 로그 테이블에 대해서 수행 가능하다.

또한, 중간의 임의 위치에 있는 데이터를 삭제할 수 없으며, 임의의 위치부터 연속적으로 마지막(가장 오래된 로그) 레코드까지 지울 수 있도록 구현되었다. 이는 로그 데이터의 특성을 살린 정책으로서 한번 입력되면 수정이 없고, 공간 확보를 위해 파일을 삭제하는 행위를 DB 형식으로 표현한 것이다.

아래는 사용할 수 있는 표현의 종류이다.

목차

문법

```
DELETE FROM table_name;
DELETE FROM table_name OLDEST number ROWS;
DELETE FROM table_name EXCEPT number ROWS;
DELETE FROM table_name EXCEPT number [YEAR | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND];
DELETE FROM table_name BEFORE datetime_expr;
```

예제

```
-- 모든 데이터를 삭제한다.
mach>DELETE FROM devices;
10 row(s) deleted.

-- 가장 오래된 5건을 삭제한다.
mach>DELETE FROM devices OLDEST 5 ROWS;
10 row(s) deleted.

-- 최근 5건을 제외하고 모두 삭제한다.
mach>DELETE FROM devices EXCEPT 5 ROWS;
15 row(s) deleted.

-- 2018년 6월 1일 이전의 데이터를 모두 삭제한다.
mach>DELETE FROM devices BEFORE TO_DATE('2018-06-01', 'YYYY-MM-DD');
50 row(s) deleted.
```

로그 인덱스 생성 및 관리

마크베이스의 로그테이블에는 2가지 인덱스 타입을 생성할 수 있다.

자세한 내용은 SQL 레퍼런스의 DDL 페이지의 CREATE INDEX 문단을 참조하면 된다.

- BITMAP 인덱스 : BITMAP인덱스는 Text, Binary 타입을 제외한 모든 컬럼에 생성할 수 있다.
- KEYWORD 인덱스 : Varchar, Text 컬럼에만 생성 가능하며 문자열을 검색할 때 사용한다.

목차

인덱스 생성

CREATE INDEX 구문을 이용하여 특정 컬럼에 대해서 인덱스를 생성한다.

```
CREATE INDEX index_name ON table_name (column_name) [index_type] [tablespace] [index_prop_list]
    index_type ::= INDEX_TYPE { BITMAP | KEYWORD }
    tablespace ::= TABLESPACE tablesapce_name
    index_prop_list ::= value_pair, value_pair, ...
    value_pair ::= property_name = property_value
```

```
Mach> CREATE INDEX id_index ON log_data(id) INDEX_TYPE BITMAP TABLESPACE tbs_data MAX_LEVEL=3;
Created successfully.
```

인덱스 변경

ALTER INDEX 구문을 이용하여 인덱스 속성을 변경한다.

```
ALTER INDEX index_name SET KEY_COMPRESS = { 0 | 1 }
```

```
Mach> ALTER INDEX id_index SET KEY_COMPRESS = 1;
```

인덱스 삭제

DROP INDEX 구문을 이용하여 지정된 인덱스를 삭제한다. 단, 해당 테이블을 검색 중인 다른 세션이 존재할 경우에는 에러를 내면서 실패한다.

```
DROP INDEX index_name;
```

```
Mach> DROP INDEX id_index;
Dropped successfully.
```

로그 테이블 활용 샘플 예제

마크베이스 패키지를 설치하면 로그 테이블을 생성하고 로그 데이터를 생성된 테이블에 입력하고 조회하는 튜토리얼을 제공한다.

아래 경로에서 확인할 수 있다.

```
[machbase@localhost tutorials]$ cd $MACHBASE_HOME/tutorials
[machbase@localhost tutorials]$ ls -l
total 0
drwxrwxr-x 2 machbase machbase 103 Oct 30 16:10 backup_mount
drwxrwxr-x 2 machbase machbase  44 Oct 30 16:10 connect_r
drwxrwxr-x 2 machbase machbase 177 Oct 30 16:10 csvload
drwxrwxr-x 2 machbase machbase  49 Oct 30 16:10 export_data
drwxrwxr-x 2 machbase machbase  32 Oct 30 16:10 install_docker_image
drwxrwxr-x 2 machbase machbase  49 Oct 30 16:10 ip_address
drwxrwxr-x 2 machbase machbase  75 Oct 30 16:10 searchtext
drwxrwxr-x 2 machbase machbase  93 Oct 30 16:10 time_series
[machbase@localhost tutorials]$
```

목차

로그 테이블 생성

입력할 로그 데이터는 다음은 csv 포맷의 파일이다.

```
[machbase@localhost csvload]$ cd $MACHBASE_HOME/tutorials/csvload
[machbase@localhost csvload]$ more sample_data.csv
2015-05-20 06:00:00,63.214.191.124,2296,122.195.164.32,5416,12,GET /twiki/bin/view/Main/TWikiGroups?rev=1.2 HTTP/1
2015-05-20 06:00:07,212.237.153.79,6203,71.129.68.118,8859,67,GET /twiki/bin/view/Main/WebChanges HTTP/1.1,200,405
2015-05-20 06:00:07,243.9.49.80,344,122.195.164.32,6203,46,GET /twiki/bin/view/Main/TWikiGroups?rev=1.2 HTTP/1.1,2
2015-05-20 06:00:07,232.191.241.129,5377,174.47.129.59,1247,17,GET /mailman/listinfo/hsdivision HTTP/1.1,200,6291
2015-05-20 06:00:07,121.67.24.216,2296,212.237.153.79,6889,68,GET /twiki/bin/view/TWiki/WebTopicEditTemplate HTTP/1
2015-05-20 06:00:07,31.224.72.52,450,100.46.183.122,10541,20,GET /twiki/bin/view/Main/WebChanges HTTP/1.1,200,4052
2015-05-20 06:00:07,210.174.159.227,6180,173.149.119.202,6927,2,GET /twiki/bin/rdiff/TWiki/AlWilliams?rev1=1.2&rev2
2015-05-20 06:00:07,210.174.159.227,10124,16.194.51.72,10512,69,GET /twiki/bin/rdiff/TWiki/AlWilliams?rev1=1.2&rev2
2015-05-20 06:00:07,60.48.99.15,12333,85.183.139.166,12020,64,GET /robots.txt HTTP/1.1,200,68
```

로그 데이터의 각각의 필드 값을 확인하고 테이블을 생성한다. machsql 상에서 'CREATE TABLE' 구문을 이용하여 생성하면 된다.

```
CREATE TABLE SAMPLE_TABLE
(
    srcip        IPV4,
    srcport      INTEGER,
    dstip        IPV4,
    dstport      INTEGER,
    protocol     SHORT,
    eventlog     VARCHAR(1204),
    eventcode    SHORT,
    eventsize    LONG
);
```

또는 테이블 생성 스크립트 파일을 만들어서 OS 커맨드 라인상에서 machsql 을 실행해도 된다.

```
[machbase@localhost csvload]$ machsql -s localhost -u sys -p manager -f create_sample_table.sql
================================================================
     Machbase Client Query Utility
     Release Version x.x.x.official
     Copyright 2014 MACHBASE Corporation or its subsidiaries.
     All Rights Reserved.
================================================================
MACHBASE_CONNECT_MODE=INET, PORT=5656
Type 'help' to display a list of available commands.
Mach> CREATE TABLE SAMPLE_TABLE
(
```

```
    srcip        IPV4,
    srcport      INTEGER,
    dstip        IPV4,
    dstport      INTEGER,
    protocol     SHORT,
    eventlog     VARCHAR(1204),
    eventcode    SHORT,
    eventsize    LONG
);
Created successfully.
```

로그 데이터 입력

로그 데이터는 csv 포맷 파일이므로 csvimport 를 이용하여 로딩하면 된다.

로그 파일의 첫번째 필드가 날짜인데, 이 값을 _arrival_time 컬럼에 입력하도록 옵션을 지정한다.

```
[machbase@localhost csvload]$ csvimport -t sample_table -d sample_data.csv -a -F "_arrival_time YYYY-MM-DD HH24:MI
----------------------------------------------------------------
     Machbase Data Import/Export Utility.
     Release Version x.x.x.official
     Copyright 2014, MACHBASE Corporation or its subsidiaries.
     All Rights Reserved.
----------------------------------------------------------------
NLS            : US7ASCII            EXECUTE MODE   : IMPORT
TARGET TABLE   : sample_table        DATA FILE      : sample_data.csv
IMPORT_MODE    : APPEND              FILED TERM     : ,
ROW TERM       :
                   ENCLOSURE      : "
ESCAPE         : "                   ARRIVAL_TIME   : TRUE
ENCODING       : NONE                HEADER         : FALSE
CREATE TABLE   : FALSE

 Progress bar                       Imported records        Error records
                                            1000000                    0

Import time         :  0 hour  0 min  5.728 sec
Load success count  : 1000000
Load fail count     : 0

[machbase@localhost csvload]$
```

로그 데이터 조회

데이터 조회는 machsql 상에서 확인한다.

```
[machbase@localhost csvload]$ machsql
================================================================
     Machbase Client Query Utility
     Release Version x.x.x.official
     Copyright 2014 MACHBASE Corporation or its subsidiaries.
     All Rights Reserved.
================================================================
Machbase server address (Default:127.0.0.1) :
Machbase user ID  (Default:SYS)
Machbase User Password :
MACHBASE_CONNECT_MODE=INET, PORT=5656
Type 'help' to display a list of available commands.
Mach> show tables;
NAME                                                                                    TYPE
-----------------------------------------------------------------------------------------------------
SAMPLE_TABLE                                                                            LOG
[1] row(s) selected.
```

```
Mach> desc sample_table;
[ COLUMN ]
---------------------------------------------------------------
NAME                          TYPE               LENGTH
---------------------------------------------------------------
SRCIP                         ipv4               15
SRCPORT                       integer            11
DSTIP                         ipv4               15
DSTPORT                       integer            11
PROTOCOL                      short              6
EVENTLOG                      varchar            1204
EVENTCODE                     short              6
EVENTSIZE                     long               20

Mach> SELECT COUNT(*) FROM SAMPLE_TABLE;
COUNT(*)
----------------------
1000000
[1] row(s) selected.

Mach> SELECT SRCIP, COUNT(*) FROM SAMPLE_TABLE GROUP BY SRCIP ORDER BY 2 DESC LIMIT 10;
SRCIP           COUNT(*)
---------------------------------------
96.128.212.177  13594
173.149.119.202 13546
219.229.142.218 13537
69.99.246.62    13511
239.81.105.222  13501
86.45.186.17    13487
231.146.69.51   13483
248.168.229.34  13472
105.9.103.49    13472
115.18.128.171  13468
[10] row(s) selected.
Mach>
```

인덱스 생성 및 조회

생성된 sample_table 컬럼 중에서 varchar 형인 eventlog 컬럼에 대해서 keyword 인덱스를 생성하고 텍스트 검색을 해본다.

```
-- eventlog_index 인덱스를 생성한다.
Mach> CREATE INDEX eventlog_index ON SAMPLE_TABLE( eventlog) INDEX_TYPE KEYWORD;
Created successfully.
Elapsed time: 0.442

-- 생성된 인덱스를 확인한다.
Mach> desc sample_table;
[ COLUMN ]
---------------------------------------------------------------
NAME                          TYPE               LENGTH
---------------------------------------------------------------
SRCIP                         ipv4               15
SRCPORT                       integer            11
DSTIP                         ipv4               15
DSTPORT                       integer            11
PROTOCOL                      short              6
EVENTLOG                      varchar            1204
EVENTCODE                     short              6
EVENTSIZE                     long               20

[ INDEX ]
---------------------------------------------------------------
NAME                          TYPE               COLUMN
---------------------------------------------------------------
EVENTLOG_INDEX                KEYWORD_LSM        EVENTLOG
```

```
-- SEARCH 구문을 이용하여 'view' 가 들어간 데이터를 검색한다.
Mach> SELECT EVENTLOG FROM SAMPLE_TABLE WHERE EVENTLOG SEARCH 'view' LIMIT 10;
EVENTLOG
--------------------------------------------------------------------------------
GET /twiki/bin/view/TWiki/ManagingWebs?skin=print HTTP/1.1
GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1
GET /twiki/bin/view/TWiki/ManagingWebs?rev=1.22 HTTP/1.1
GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1
GET /twiki/bin/view/TWiki/WebTopicEditTemplate HTTP/1.1
GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1
GET /twiki/bin/view/TWiki/WikiCulture HTTP/1.1
GET /twiki/bin/view/Main/MikeMannix HTTP/1.1
GET /twiki/bin/view/TWiki/WikiCulture HTTP/1.1
GET /twiki/bin/view/TWiki/WikiCulture HTTP/1.1
[10] row(s) selected.

-- 'robots.txt'가 포함된 데이터 건수를 구한다.
Mach> SELECT COUNT(*) FROM SAMPLE_TABLE WHERE EVENTLOG SEARCH 'robots.txt';
COUNT(*)
----------------------
40283
[1] row(s) selected.

-- 'robots.txt'가 포함된 데이터를 SRCIP 별로 집계해서 상위 10개만 출력한다.
Mach> SELECT SRCIP, COUNT(*) FROM SAMPLE_TABLE WHERE EVENTLOG SEARCH 'robots.txt' GROUP BY SRCIP ORDER BY 2 DESC LI
SRCIP           COUNT(*)
---------------------------------------
81.227.25.139   616
162.80.44.96    596
7.234.88.67     595
227.106.13.91   578
220.192.100.45  570
46.201.48.18    570
231.146.69.51   564
185.22.195.164  564
64.58.31.79     561
50.5.206.126    561
[10] row(s) selected.
```

시계열 데이터 조회

마크베이스는 시계열 데이터를 조회하는데 편리한 구문을 제공하고 있다. DURATION을 이용하여 빠른 데이터를 조회하는 방법을 알아본다.

```
-- _arrival_time 컬럼에 입력된 최대,최소값을 확인한다.
Mach> SELECT MIN(_ARRIVAL_TIME), MAX(_ARRIVAL_TIME) FROM SAMPLE_TABLE;
MIN(_ARRIVAL_TIME)              MAX(_ARRIVAL_TIME)
-------------------------------------------------------------------
2015-05-20 06:00:00 000:000:000 2015-05-20 06:40:10 000:000:000
[1] row(s) selected.

-- DATE_TRUNC() 함수를 이용하여 분당 건수를 구한다.
Mach> SELECT DATE_TRUNC('minute', _ARRIVAL_TIME) as TIME, COUNT(*) as COUNT FROM SAMPLE_TABLE GROUP BY TIME ORDER B
TIME                            COUNT
-----------------------------------------------------
2015-05-20 06:00:00 000:000:000 32001
2015-05-20 06:01:00 000:000:000 28000
2015-05-20 06:02:00 000:000:000 24000
2015-05-20 06:03:00 000:000:000 32000
2015-05-20 06:04:00 000:000:000 16000
2015-05-20 06:05:00 000:000:000 16000
2015-05-20 06:06:00 000:000:000 32000
2015-05-20 06:07:00 000:000:000 32000
2015-05-20 06:08:00 000:000:000 20000
2015-05-20 06:09:00 000:000:000 24000
2015-05-20 06:10:00 000:000:000 20000
2015-05-20 06:11:00 000:000:000 20000
2015-05-20 06:12:00 000:000:000 24000
```

```
2015-05-20 06:13:00 000:000:000 20000
2015-05-20 06:14:00 000:000:000 32000
2015-05-20 06:15:00 000:000:000 24000
2015-05-20 06:16:00 000:000:000 32000
2015-05-20 06:17:00 000:000:000 28000
2015-05-20 06:18:00 000:000:000 32000
2015-05-20 06:19:00 000:000:000 12000
2015-05-20 06:20:00 000:000:000 24000
2015-05-20 06:21:00 000:000:000 28000
2015-05-20 06:22:00 000:000:000 28000
2015-05-20 06:23:00 000:000:000 24000
2015-05-20 06:24:00 000:000:000 28000
2015-05-20 06:25:00 000:000:000 28000
2015-05-20 06:26:00 000:000:000 32000
2015-05-20 06:27:00 000:000:000 20000
2015-05-20 06:28:00 000:000:000 20000
2015-05-20 06:29:00 000:000:000 20000
2015-05-20 06:30:00 000:000:000 28000
2015-05-20 06:31:00 000:000:000 32000
2015-05-20 06:32:00 000:000:000 32000
2015-05-20 06:33:00 000:000:000 28000
2015-05-20 06:34:00 000:000:000 20000
2015-05-20 06:35:00 000:000:000 24000
2015-05-20 06:36:00 000:000:000 24000
2015-05-20 06:37:00 000:000:000 16000
2015-05-20 06:38:00 000:000:000 24000
2015-05-20 06:39:00 000:000:000 16000
2015-05-20 06:40:00 000:000:000 3999
[41] row(s) selected.

-- DURATION 구문을 이용하여 특정시각 기준 1분 이전 시간 범위를 지정하여 조회한다.
Mach> SELECT MIN(_ARRIVAL_TIME), MAX(_ARRIVAL_TIME), COUNT(*) as COUNT FROM SAMPLE_TABLE DURATION 1 MINUTE BEFORE T
MIN(_ARRIVAL_TIME)              MAX(_ARRIVAL_TIME)              COUNT
-------------------------------------------------------------------------------
2015-05-20 06:29:05 000:000:000 2015-05-20 06:29:45 000:000:000 20000
[1] row(s) selected.

-- DURATION 구문을 이용하여 특정시각 기준 1분 이후 시간 범위를 지정하여 조회한다.
Mach> SELECT MIN(_ARRIVAL_TIME), MAX(_ARRIVAL_TIME), COUNT(*) as COUNT FROM SAMPLE_TABLE DURATION 1 MINUTE AFTER TO
MIN(_ARRIVAL_TIME)              MAX(_ARRIVAL_TIME)              COUNT
-------------------------------------------------------------------------------
2015-05-20 06:30:04 000:000:000 2015-05-20 06:30:57 000:000:000 28000
[1] row(s) selected.


-- DURATION 구문을 이용하여 FROM ~ TO 시간 범위를 지정하여 조회한다.
Mach> SELECT MIN(_ARRIVAL_TIME), MAX(_ARRIVAL_TIME), COUNT(*) as COUNT FROM SAMPLE_TABLE DURATION FROM TO_DATE('201
MIN(_ARRIVAL_TIME)              MAX(_ARRIVAL_TIME)              COUNT
-------------------------------------------------------------------------------
2015-05-20 06:20:03 000:000:000 2015-05-20 06:29:45 000:000:000 252000
[1] row(s) selected.
```

인터넷 주소형 데이터 조회

마크베이스는 인터넷 주소에 대해서 데이터 타입으로 제공하고 편리하게 검색할 수 있다.

```
-- Netmask 형식으로 IP 대역을 설정하여 조회한다.
Mach> SELECT COUNT(*) FROM SAMPLE_TABLE WHERE SRCIP CONTAINED '100.195.159.0/24';
COUNT(*)
----------------------
13097
[1] row(s) selected.


-- '*' 를 이용하여 Equal(=) 검색도 가능하다.
Mach> SELECT COUNT(*) FROM SAMPLE_TABLE WHERE SRCIP = '100.195.159.*';
COUNT(*)
```

```
------------------------
13097
[1] row(s) selected.
```

휘발성 테이블 (Volatile Table)

개념

휘발성(Volatile) 테이블은 모든 데이터가 임시 메모리 공간에 상주하고, 로그 테이블과의 Join을 통해 데이터 결과를 풍부하게 만드는 임시 테이블이다.

휘발성 테이블은 로그 테이블에 간략한 기호나 숫자로 표현된 특정 디바이스나 장비의 다양한 정보를 담고 있는 부가 정보 테이블이라고 할 수 있다. 고속으로 입력 및 갱신이 가능하여 데이터의 현재 상태 (시계열 데이터와는 맞지 않음) 를 실시간으로 유지해야 하는 경우에 많이 사용한다.

이 테이블의 특성은 다음과 같다

스키마 보존

휘발성 테이블의 구조(스키마) 정보는 Machbase 서버가 종료되었다가 다시 구동이 되더라도 유지된다. 해당 테이블을 삭제하기 위해서는 명시적으로 DROP table을 수행하면 된다.

데이터 휘발성

휘발성 테이블에 담긴 데이터는 서버를 종료하는 즉시 모두 사라진다. 따라서 서버의 구동 시에 휘발성 테이블의 내용을 다시 INSERT 해야 한다.

인덱스 제공 및 Join 기능

휘발성 테이블의 빠른 데이터 접근을 위해 실시간에 최적화된 인덱스인 RED-BLACK 인덱스를 제공한다. 따라서 로그 테이블과의 Join 이나 검색 과정에서 효율적으로 활용될 수 있다.

- 테이블 컬럼에서 기본 키(Primary Key) 를 지정할 수 있다.
- 중복되는 기본 키 값을 가진 데이터를 삽입하는 경우, 기존 데이터의 값을 갱신(UPDATE)할 수 있다.
- 조건 절 (`WHERE` 절)을 이용해, 기본 키 값 조건과 일치하는 데이터를 삭제할 수 있다.
- `_ARRIVAL_TIME` 컬럼이 존재하지 않는다.

기본 키(Primary Key)

기본 키는 테이블 컬럼 값의 유일성(Uniqueness) 제약 조건을 형성하고 테이블 데이터를 구별할 키 컬럼을 지정하는 목적으로 생성할 수 있다.

기본 키가 지정된 휘발성 테이블에 데이터를 삽입할 때, 삽입 데이터의 기본 키 컬럼 값은 테이블 안의 다른 기본 키 컬럼 값들과 반드시 달라야 한다. 이 제약사항을 유일성 조건이라고 할 수 있다.

기본 키의 생성 제약사항은 다음과 같다.

- 기본 키는 휘발성 테이블에서만 생성할 수 있다.
- 기본 키 컬럼은 1개만 지정할 수 있으며, 2개 이상의 컬럼을 함께 기본 키로 지정할 수 없다.

갱신(UPDATE) 기능

다른 테이블 유형과는 다르게, 휘발성 테이블은 제한적이나마 갱신 기능을 제공한다.

삽입하고자 하는 데이터의 기본 키 값이 이미 다른 데이터의 기본 키 값 중 하나와 중복되는 경우에는 '삽입' 이 아니라 '갱신' 모드로 전환되며, 중복된 키 값을 가진 기존 데이터의 다른 컬럼 값이 삽입하고자 하는 데이터의 컬럼 값으로 변경된다. 기본 키가 지정된 휘발성 테이블에서만 갱신 기능을 사용할 수 있으며, 삽입 과정에서 기본 키 값을 지정하지 않았을 때는 갱신 기능을 사용할 수 없다.

삭제(DELETE) 기능

휘발성 테이블은 기본 키 값을 이용해 특정 데이터를 삭제할 수 있는 기능을 제공한다.

DELETE 구문에서 조건 절(`WHERE` 절)을 추가해 기본 키 값을 지정하면, 해당 기본 키 값에 해당하는 데이터가 존재하는 경우에 한하여 이를 삭제할 수 있다. 기본 키가 지정된 휘발성 테이블에서만 삭제 기능을 사용할 수 있으며, 이 때 조건 절에 들어갈 수 있는 조건은 `(기본 키 컬럼) = (값)` 으로 제한되어 있다.

작업 방법

휘발성 테이블 생성 및 관리

휘발성 테이블의 생성과 삭제 방식은 다음과 같다.

생성

```
create volatile table vtable (id1 integer, name varchar(20));
```

삭제

```
drop table vtable;
```

휘발성 데이터의 입력 및 갱신

데이터 입력 (Insert)

휘발성 테이블의 데이터 입력(Insert)은 다음과 같다.

```
Mach> create volatile table vtable (id integer, name varchar(20));
Created successfully.
Mach> insert into vtable values(1, 'west device');
1 row(s) inserted.
Mach> insert into vtable values(2, 'east device');
1 row(s) inserted.
Mach> insert into vtable values(3, 'north device');
1 row(s) inserted.
Mach> insert into vtable values(4, 'south device');
1 row(s) inserted.
```

데이터 입력 (Append)

마크베이스에서 제공하는 빠른 실시간 데이터 입력 API이다.

C, C++, C#, Java, Python, PHP, Javascript 를 이용하여 입력할 수 있다.

```
Mach> create volatile table vtable (id integer, value double);
```

```
SQL_APPEND_PARAM sParam[2];
for(int i=0; i<10000; i++)
{
    sParam[0].mInteger  = i;
    sParam[1].mDouble   = i;
    SQLAppendDataV2(stmt, sParam) != SQL_SUCCESS)
}
```

Cluster Edition의 경우 Append 입력은 Leader Broker에서 수행해야 한다.

세부 내용은 SDK 가이드를 참고한다.

데이터 갱신

휘발성 테이블의 데이터 입력 시, ON DUPLICATE KEY UPDATE 절을 사용해 중복된 기본 키 값을 가진 데이터의 갱신을 할 수 있다.

삽입할 데이터 값으로 갱신

INSERT 구문에서 삽입할 데이터를 지정했지만, 삽입 데이터의 기본 키 값과 일치하는 다른 데이터가 존재하는 경우에는 INSERT 구문이 실패하게 되고 해당 데이터는 삽입되지 않는다. 삽입 데이터의 기본 키 값과 일치하는 다른 데이터가 존재하는 경우에, 삽입이 아닌 해당 데이터를 갱신하고자 하는 경우에는 `ON DUPLICATE KEY UPDATE` 절을 추가할 수 있다.

- 기본 키 중복 데이터가 존재하지 않는 경우, 삽입할 데이터 내용이 그대로 삽입된다.
- 기본 키 중복 데이터가 존재하는 경우, 삽입할 데이터 내용으로 기존의 데이터가 갱신된다.

이 기능을 사용하기 위한 제약 조건은 다음과 같다.

- **휘발성 테이블에 기본 키가 지정되어 있어야 한다.**
- 삽입하고자 하는 값에, 기본 키 값이 반드시 포함되어야 한다.

```
Mach> create volatile table vtable (id integer primary key, direction varchar(10), refcnt integer);
Created successfully.
Mach> insert into vtable values(1, 'west', 0);
1 row(s) inserted.
Mach> insert into vtable values(2, 'east', 0);
1 row(s) inserted.
Mach> select * from vtable;
ID          DIRECTION   REFCNT
----------------------------------------
1           west       0
2           east        0
```

```
[2] row(s) selected.

Mach> insert into vtable values(1, 'south', 0);
[ERR-01418 : The key already exists in the unique index.]
Mach> insert into vtable values(1, 'south', 0) on duplicate key update;
1 row(s) inserted.

Mach> select * from vtable;
ID          DIRECTION   REFCNT
----------------------------------------
1           south        0
2           east        0
[2] row(s) selected.
```

갱신할 데이터 값을 지정

위와 비슷하지만, 삽입할 데이터 값과 다른 컬럼 값으로 갱신해야 하는 경우에는 `ON DUPLICATE KEY UPDATE SET` 절을 통해 지정할 수 있다. `SET` 절 아래에 갱신할 데이터 값을 지정할 수 있다.

- 기본 키 중복 데이터가 존재하지 않는 경우, *삽입 데이터* 내용이 그대로 삽입된다.
- 기본 키 중복 데이터가 존재하는 경우, `SET` 절에 명시된 *갱신 데이터*만으로 기존의 데이터가 갱신된다.
- **기본 키 값을 갱신할 데이터 값으로 지정할 수 없다.**
- `SET` 절에서 명시되지 않은 컬럼들의 값은 갱신되지 않는다.

```
Mach> create volatile table vtable (id integer primary key, direction varchar(10), refcnt integer);
Created successfully.
Mach> insert into vtable values(1, 'west', 0);
1 row(s) inserted.
Mach> insert into vtable values(2, 'east', 0);
1 row(s) inserted.
Mach> select * from vtable;
ID          DIRECTION   REFCNT
----------------------------------------
1           west        0
2           east        0
[2] row(s) selected.

Mach> insert into vtable values(1, 'west', 0) on duplicate key update set refcnt = 1;
1 row(s) inserted.

Mach> select * from vtable;
ID          DIRECTION   REFCNT
----------------------------------------
1           west        1
2           east        0
[2] row(s) selected.
```

휘발성 데이터의 추출

데이터 조회

데이터 조회는 다른 테이블 유형과 마찬가지로, 아래와 같이 수행할 수 있다.

```
Mach> create volatile table vtable (id integer primary key, name varchar(20));
Created successfully.
Mach> insert into vtable values(1, 'west device');
1 row(s) inserted.
Mach> insert into vtable values(2, 'east device');
1 row(s) inserted.
Mach> insert into vtable values(3, 'north device');
1 row(s) inserted.
Mach> insert into vtable values(4, 'south device');
1 row(s) inserted.
Mach> select * from vtable;
ID          NAME
-------------------------------------
1           west device
2           east device
3           north device
4           south device
[4] row(s) selected.
Mach> select * from vtable where id = 1;
ID          NAME
-------------------------------------
1           west device
[1] row(s) selected.
Mach> select * from vtable where name like 'west%';
ID          NAME
-------------------------------------
1           west device
[1] row(s) selected.
```

휘발성 데이터의 삭제

데이터 삭제

휘발성 테이블은 조건 절(WHERE 절)에서 기본 키 값 조건을 이용해 데이터를 삭제할 수 있다.

- **휘발성 테이블에 기본 키 컬럼이 지정되어 있어야 한다.**
- (기본 키 컬럼) = (값)의 조건만 허용하며, 다른 조건과 함께 사용할 수 없다.
- 기본 키 컬럼이 아닌 다른 컬럼을 사용할 수 없다.

```
Mach> create volatile table vtable (id integer primary key, name varchar(20));
Created successfully.
Mach> insert into vtable values(1, 'west device');
1 row(s) inserted.
Mach> insert into vtable values(2, 'east device');
1 row(s) inserted.
Mach> insert into vtable values(3, 'north device');
1 row(s) inserted.
Mach> insert into vtable values(4, 'south device');
1 row(s) inserted.
Mach> select * from vtable;
ID          NAME
--------------------------------------
1           west device
2           east device
3           north device
4           south device
[4] row(s) inserted.
Mach> delete from vtable where id = 2;
[1] row(s) deleted.
Mach> select * from vtable;
ID          NAME
--------------------------------------
1           west device
3           north device
4           south device
[3] row(s) selected.
```

휘발성 인덱스 생성 및 관리

인덱스 생성 및 활용

휘발성 테이블은 실시간 검색에 최적화된 레드블랙(RED-BLACK) 트리를 기본으로 제공하고 있으며, 모든 데이터 타입에 대해서 인덱스를 설치할 수 있다. 단, 하나의 컬럼에 대해 하나의 인덱스가 생성 가능하며, 복합(composite) 인덱스는 제공하지 않는다.

```
Mach> create volatile table vtable (id integer, name varchar(10));
Created successfully.
Mach> create index idx_vrb on vtable (name) index_type redblack;
Created successfuly.
Mach> desc vtable;
-------------------------------------------------------------------
NAME                          TYPE                LENGTH
-------------------------------------------------------------------
ID                            integer             11
NAME                          varchar             10

[ INDEX ]
-------------------------------------------------------------------
NAME                          TYPE                COLUMN
-------------------------------------------------------------------
IDX_VRB                       REDBLACK            NAME
iFlux>
```

기본 키 인덱스

또한 휘발성 테이블의 특정 컬럼에 기본 키를 부여하게 되면, 여기에 레드블랙 트리 인덱스를 자동으로 생성하게 된다. 이 때는 유일성(Uniqueness) 속성을 지닌 특별한 인덱스가 생성되며 중복된 값을 허용하지 않는다.

```
Mach> create volatile table vtable (id integer primary key, name varchar(20));
Created successfully.
Mach> desc vtable;
-------------------------------------------------------------------
NAME                          TYPE                LENGTH
-------------------------------------------------------------------
ID                            integer             11
NAME                          varchar             20

[ INDEX ]
-------------------------------------------------------------------
NAME                          TYPE                COLUMN
-------------------------------------------------------------------
__PK_IDX_VTABLE               REDBLACK            ID

iFlux>
```

다른 인덱스 유형

로그 테이블에서 사용하던 비트맵 혹은 키워드 인덱스는 휘발성 테이블에서 사용할 수 없다.

```
Mach> create bitmap   index idx_1237 on vtable(id1);
[ERR-02069 : Error in index for invalid table. BITMAP Index can only be created for LOG Table.]
Mach> create keyword  index idx_1238 on vtable(name);
[ERR-02069 : Error in index for invalid table. KEYWORD Index can only be created for LOG Table.]
```

휘발성 테이블 활용 샘플 예제

센서 데이터의 현재 값을 저장

휘발성 테이블의 데이터는 메모리 상에만 존재하여 기본 키에 의한 갱신 연산이 매우 빠른 특징이 있다. 이 특징을 이용하여, 매우 빠르게 변화하는 센서의 현재 값을 저장하는 테이블을 만든다. 테이블 생성 스크립트의 예는 아래와 같다.

```
create volatile table sensor_current (sensor_id varchar(40) primary key, value double);
```

휘발성 데이터의 입력 및 갱신

테이블을 생성하였으므로, 데이터 입력과 갱신 연산을 통하여 센서의 현재 값을 반영할 수 있다. 입력되는 센서값은 기본 키 sensor_id 컬럼을 기준으로 입력 혹은 갱신을 수행할 것인지가 결정된다. 입력 혹은 갱신은 다음의 질의문으로 수행할 수 있다.

```
insert into sensor_current values('SENSOR_001',100.0) on duplicate key update set value=100.0;
```

위 질의문에서 입력되는 데이터는 기본 키에 해당하는 컬럼인 sensor_id 값이 'SENSOR_001'인 데이터가 있으면 그 레코드의 value 컬럼 값을 100.0으로 갱신하고, 없으면 Insert문의 문법 대로 새로운 레코드를 삽입한다.

휘발성 데이터의 검색

특정 센서 데이터의 현재 값을 알려면 다음의 질의를 이용하여 검색한다. 일반적인 SQL질의문과 동일한 문법을 사용하여 검색을 수행할 수 있다.

```
SELECT value FROM sensor_current WHERE sensor_id = 'SENSOR_001'
```

참조 테이블 (Lookup Table)

개념

참조 테이블은 휘발성 테이블과 마찬가지로 모든 데이터를 메모리에 상주시킴으로써 빠른 질의 처리를 수행할 수 있다.

또한 데이터의 입력 및 변경에 대해서 디스크에 반영함으로써 데이터의 영구성을 보장한다. 휘발성 테이블과 비교하여 질의 처리 성능은 동일하지만 데이터 입력 및 변경 성능은 다소 떨어진다.

이 테이블의 특성은 다음과 같다.

낮은 입력/갱신 성능

휘발성 테이블과는 달리, 디스크 데이터 이미지 유지에 따른 입력 및 갱신 성능이 낮으므로 대시보드 (Dashboard) 등 실시간 데이터 표현을 위한 테이블에는 부적합하다.

스키마 보존

역시 휘발성 테이블과는 달리, 참조 테이블의 구조 (스키마) 정보는 서버 재시작 후에도 유지된다. 해당 테이블을 삭제하기 위해서는 명시적으로 `DROP TABLE` 를 수행해야 한다.

데이터 보존

참조 테이블은 휘발성 테이블과 달리 서버 재시작 시 데이터가 서버 종료 직전의 상태로 복구된다.

인덱스 제공

휘발성 테이블과 마찬가지로 RED-BLACK 인덱스를 제공한다. 따라서 검색 과정이나 로그 테이블과의 Join 과정에서 효율적으로 활용될 수 있다.

작업 방법

- 참조 테이블 생성 및 관리
- 참조 데이터의 입력
- 참조 데이터의 추출
- 참조 데이터의 삭제
- 참조 인덱스 생성 및 관리
- 참조 테이블 활용 샘플 예제

참조 테이블 생성 및 관리

참조 테이블의 생성 방법은 다음과 같다.

생성

```
CREATE LOOKUP TABLE lktable (id INTEGER PRIMARY KEY, name VARCHAR(20));
```

참조 테이블은 Primary Key를 반드시 지정해야한다.

삭제

```
DROP TABLE lktable;
```

참조 데이터의 입력

휘발성 테이블의 입력 및 갱신 방법과 대부분 동일하다.

한가지 차이점이 있다면 참조 테이블에 Append를 통해 데이터를 입력할 경우 'LOOKUP_APPEND_UPDATE_ON_DUPKEY ' Property 를 설정해 Primary Key가 중복일 경우에 해당 Row를 Update 하도록 설정 가능하다.

'LOOKUP_APPEND_UPDATE_ON_DUPKEY'에 대한 세부 내용은 Property 가이드를 참고한다.

참조 테이블 재로딩

마크베이스 6.7 부터 참조 테이블 데이터를 룩업 노드가 관리하도록 수정되었다.

룩업 노드로부터 참조 테이블 데이터를 다시 불러오고 싶다면

EXEC TABLE_REFRESH 명령어를 사용하면된다.

```
EXEC TABLE_REFRESH(lktable);
```

참조 데이터의 추출

SQL문을 이용하여 데이터를 추출하며 사용 방법은 휘발성 테이블과 동일하다.

참조 데이터의 삭제

휘발성 테이블과 동일하다.

참조 인덱스 생성 및 관리

휘발성 테이블과 마찬가지로 RED-BLACK 인덱스를 기본으로 제공하고 있으며, 사용 방법은 휘발성 테이블과 동일하다. 휘발성 테이블과 같이 RED-BLACK인덱스만 생성 가능하며 keyword, LSM인덱스의 생성은 불가능하다.

참조 테이블 활용 샘플 예제

참조 테이블의 생성

참조 테이블은 갱신이 가능하며, 원본 로그 데이터가 갖고 있지 않은 데이터를 join을 통하여 추가하는데 사용된다. 아래 예제는 로그 테이블과 참조 테이블을 생성하는 예를 보여준다.

```
-- 로그 테이블의 생성
create table weblog (addr ipv4, msg varchar(100));
-- 샘플 데이터 입력
insert into weblog values ('127.0.0.1', 'a test msessage');
-- 참조 테이블의 생성
create lookup table dnslookup (addr ipv4 primary key, hostname varchar (100));
```

참조 테이블에 데이터를 삽입 혹은 갱신해 보자.

```
insert into dnslookup values ('127.0.0.1', 'localhost') on duplicate key update set hostname = '127.0.0.1'
```

참조 테이블과 로그 테이블에 join을 통하여 데이터를 검색할 수 있다.

```
select msg, hostname from weblog, dnslookup where weblog.addr = dnslookup.addr;
```

스트림 (STREAM)

개념

STREAM이란 마크베이스 5부터 새롭게 지원되는 CQL (Continuous Query Language) 기반의 실시간 데이터 처리 기능이다.

즉, 로그 테이블로 입력되는 증분 데이터를 대상으로, 이 중 조건을 만족하는 데이터를 추출해 다른 테이블로 실시간 입력할 수 있다.

만일 스트림 기능을 이용하지 않고 전체 데이터를 대상으로 조건 검색을 하게 되면 누적된 데이터의 조회가 느릴 뿐만 아니라 시스템에 큰 부담이 생길 수 있다.

스트림을 이용하면 실시간으로 입력되는 특정 로그 데이터에 대해 조건을 검색하여 이벤트에 빠르게 대응할 수 있다.

제약 사항

- 현재 마크베이스는 Edge Edition과 Fog Edition에서만 STREAM을 지원하고 있다.
    - Cluster Edition은 차후 버전에서 지원할 예정이다.
- 데이터 입력 소스는 로그 테이블만 가능하다.
    - 태그 테이블을 소스로 쓸 수 없다.
- 데이터 출력 대상은 로그 및 태그 테이블이 가능하다.

스트림 생성 및 삭제

스트림 생성

스트림 질의는 Insert... Select 의 형태로만 생성 가능하며, 스트림을 생성할 때, 질의문을 검사하여 정상 실행이 가능한 질의인지를 확인한다.

스트림을 생성하기 위해서 아래의 저장 프로시저를 이용한다.

```
EXEC STREAM_CREATE(stream_name, stream_query_string);
```

스트림을 성공적으로 생성하더라도 바로 실행이 시작되지 않는다. 관련 사항은 스트림 시작 및 종료를 참조하라.

스트림 질의

스트림 질의는 Insert... Select문의 형태를 하고 있다. 기본적인 스트림 질의는 매 입력 데이터에 대해서 실행되는 것이 기본으로, 이 경우 SUM, AVG등의 통계 질의를 사용할 수 없다.

```
EXEC STREAM_CREATE(normal_query, 'INSERT INTO CEP_LOG_TABLE SELECT * FROM EVENT WHERE C1 = 0');
```

하지만 Insert select문의 마지막에 스트림 질의가 수행될 주기를 설정하면 일정 주기 마다 입력 데이터에 대한 통계 질의문을 이용할 수 있다.

```
EXEC STREAM_CREATE(aggr_1_sec, 'insert into aggr select sum(i1), i2 from base group by i2 BY 1 SECOND');
```

위 stream 질의는 매 1초마다 수행하여 group by 질의를 마지막 수행 이후에 입력된 최신 데이터에 대해 실행하고, 그 결과를 aggr 로그 테이블에 입력하게 된다.

만약 스트림 질의의 수행 시점을 사용자가 정의하여 수행하고 싶다면 실행 주기 설정 절에 아래와 같이 지정하면, 스트림 질의는 사용자의 명시적인 호출 이전에는 수행되지 않는다.

```
EXEC STREAM_CREATE(base_trig, 'insert into aggr select sum(i1), i2 from base group by i2 BY USER');
```

스트림 질의 실행조건절을 BY USER로 하면 STREAM_EXECUTE 프로시저를 이용하여 그 스트림 질의를 명시적으로 호출될 때 까지 실행되지 않는다.
STREAM_EXECUTE로 호출된 STREAM은 이전에 읽어들인 부분을 제외하고 실행 기간 동안 추가된 증분 데이터에 대해서만 스트림 질의를 수행한다.

스트림 삭제

생성된 스트림의 목록은 V$STREAMS 메타 테이블을 이용하여 조회할 수 있다. 스트림을 삭제하려면, 스트림을 생성하였을 때 결정한 스트림의 이름을 매개변수로 다음의 저장 프로시저를 이용한다.

```
EXEC STREAM_DROP(stream_name);
```

실행중인 스트림은 삭제가 되지 않으며, 스트림을 삭제하기 전에 먼저 스트림의 실행을 종료시켜야 한다. 관련 사항은 스트림 시작 및 종료를 참조하라.

스트림 메타 테이블 V$STREAMS

DB서버에 등록된 스트림들의 현재 상태를 조회하기 위한 메타 테이블이다. 자세한 설명은 메뉴얼의 virtual table에 기술되어 있다.

스트림 실행 및 종료

스트림 실행

저장 프로시저를 이용하여 등록된 스트림을 실행한다. 한번 실행한 스트림은 지속적으로 실행되며 서버를 재시작하더라도 마지막으로 실행한 시점 이후에 입력된 데이터에 대해서 계속 스트림 질의를 실행한다.

```
EXEC STREAM_START(stream_name);
```

스트림 종료

실행중인 스트림을 종료시키기 위해서 아래의 저장 프로시저를 이용한다.

```
EXEC STREAM_STOP(stream_name);
```

스트림의 직접 실행

스트림 실행 조건을 BY USER로 설정한 경우, 사용자에 의한 명시적 호출 없이는 해당 질의가 수행되지 않는다. 이 스트림 질의를 실행하기 위해 다음의 저장프로시저를 이용한다.

```
EXEC STREAM_EXECUTE(stream_name);
```

호출할 스트림 질의를 생성할 때, BY USER조건으로 생성하지 않았거나, STREAM_START로 실행 상태로 전환하지 않은 경우, 오류가 발생한다.

스트림 활용을 위한 샘플 예제

샘플 데이터 다운로드

아래 가이드를 따라 샘플 데이터를 다운로드한다.

```
## 1. MACHBASE git repository에서 샘플 데이터를 clone한다.
$ git clone https://www.github.com/MACHBASE/TagTutorial.git MyTutorial

## 2. 필요한 데이터를 압축 해제한다.
$ cd MyTutorial/
$ gunzip edu_3_plc_stream/*.gz

## 3. 샘플 데이터가 있는 이렉토리로 이동한다.
$ cd edu_3_plc_stream/
```

TAG, LOG 테이블의 생성

STREAM 기능을 사용하기 위해 아래 명령어들을 환경에 맞게 수정 후 실행해 TAG, LOG 테이블을 생성한다.

```
$ pwd
~/MyTutorial/edu_3_plc_stream

## 1-1. TAG 테이블 생성
$ machsql --server=127.0.0.1 --port=${MACHBASE_PORT_NO} --user=SYS --password=MANAGER --script=1_create_tag.sql
## 1-2. TAG Meta 로드
$ sh 2_load_meta.sh

## 2. LOG 테이블 생성
$ machsql --server=127.0.0.1 --port=${MACHBASE_PORT_NO} --user=SYS --password=MANAGER --script=3_create_plc_tag_tab
```

STREAM 생성, 구동

생성된 TAG, LOG 테이블에 맞게 작성된 샘플 파일을 수행해 STREAM을 시작한다.

```
$ machsql --server=127.0.0.1 --port=${MACHBASE_PORT_NO} --user=SYS --password=MANAGER --script=4_plc_stream_tag.sq
```

해당 샘플 파일에 포함된 쿼리에는 두 종류가 있는데 각각 STREAM을 생성하고 생성된 STREAM을 구동하는 역할을 한다.

```
## STREAM 생성 Query 예
## event_v0라는 이름의 MTAG_V00를 name으로 가지고 plc_tag_table에 입력되는 데이터 중 tm과 v0 column 데이터를 time, value로
EXEC STREAM_CREATE(event_v0, 'insert into tag select ''MTAG_V00'', tm, v0 from plc_tag_table;');
## STREAM 구동 Query 예
EXEC STREAM_START(event_v0);
```

STREAM을 정상적으로 구동시켰다면 이제부터 plc_tag_table에 데이터가 입력되는 순간 각 STREAM이 동작해 해당되는 데이터를 TAG 테이블에 입력한다.

STREAM 상태 확인

Machbase에서 지원하는 가상 테이블인 v$streams를 통해 수행중인 STREAM의 갯수, 사용 쿼리, 상태, 에러 메시지 등을 확인할 수 있다.

```
Mach> desc v$streams;
[ COLUMN ]
-------------------------------------------------------------------------------------------------------
NAME                                                    NULL?    TYPE                LENGTH
-------------------------------------------------------------------------------------------------------
NAME                                                             varchar             100
LAST_EX_TIME                                                     datetime            31
TABLE_NAME                                                       varchar             100
END_RID                                                          long                20
STATE                                                            varchar             10
QUERY_TXT                                                        varchar             2048
ERROR_MSG                                                        varchar             2048
```

```
FREQUENCY                                                    ulong              20
```

다음과 같이 모든 STREAM의 상태를 확인할 수 있다.

```
Mach> select state, name, table_name, query_txt from v$streams;
STATE   NAME     TABLE_NAME    QUERY_TXT
-------------------------------------------------------------------------------------------------
RUNNING EVENT_V0 PLC_TAG_TABLE insert into tag select 'MTAG_V00', tm, v0 from plc_tag_table;
RUNNING EVENT_V1 PLC_TAG_TABLE insert into tag select 'MTAG_V00', tm, v1 from plc_tag_table;
RUNNING EVENT_C0 PLC_TAG_TABLE insert into tag select 'MTAG_C00', tm, c0 from plc_tag_table;
RUNNING EVENT_C1 PLC_TAG_TABLE insert into tag select 'MTAG_C01', tm, c1 from plc_tag_table;
RUNNING EVENT_C2 PLC_TAG_TABLE insert into tag select 'MTAG_C02', tm, c2 from plc_tag_table;
RUNNING EVENT_C3 PLC_TAG_TABLE insert into tag select 'MTAG_C03', tm, c3 from plc_tag_table;
RUNNING EVENT_C4 PLC_TAG_TABLE insert into tag select 'MTAG_C04', tm, c4 from plc_tag_table;
RUNNING EVENT_C5 PLC_TAG_TABLE insert into tag select 'MTAG_C05', tm, c5 from plc_tag_table;
RUNNING EVENT_C6 PLC_TAG_TABLE insert into tag select 'MTAG_C06', tm, c6 from plc_tag_table;
RUNNING EVENT_C7 PLC_TAG_TABLE insert into tag select 'MTAG_C07', tm, c7 from plc_tag_table;
RUNNING EVENT_C8 PLC_TAG_TABLE insert into tag select 'MTAG_C08', tm, c8 from plc_tag_table;
RUNNING EVENT_C9 PLC_TAG_TABLE insert into tag select 'MTAG_C09', tm, c9 from plc_tag_table;
RUNNING EVENT_C10 PLC_TAG_TABLE insert into tag select 'MTAG_C10', tm, c10 from plc_tag_table;
RUNNING EVENT_C11 PLC_TAG_TABLE insert into tag select 'MTAG_C11', tm, c11 from plc_tag_table;
RUNNING EVENT_C12 PLC_TAG_TABLE insert into tag select 'MTAG_C12', tm, c12 from plc_tag_table;
RUNNING EVENT_C13 PLC_TAG_TABLE insert into tag select 'MTAG_C13', tm, c13 from plc_tag_table;
RUNNING EVENT_C14 PLC_TAG_TABLE insert into tag select 'MTAG_C14', tm, c14 from plc_tag_table;
RUNNING EVENT_C15 PLC_TAG_TABLE insert into tag select 'MTAG_C15', tm, c15 from plc_tag_table;
```

데이터 로드

STREAM이 모두 구동중임을 확인했으니 Machloader를 사용해 데이터를 입력, 작동을 확인한다.

STREAM은 입력 방식에 관계 없이 작동하므로 CLI, JDBC, Collector 등 어떤 방식의 입력이 발생하더라도 TAG 테이블로 자동 입력된다.

```
$ cat 5_plc_tag_load.sh
machloader  -t plc_tag_table -i -d 5_plc_tag.csv -F "tm YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn"

$ sh 5_plc_tag_load.sh
----------------------------------------------------------------
     Machbase Data Import/Export Utility.
     Release Version 6.5.1.official
     Copyright 2014, MACHBASE Corporation or its subsidiaries.
     All Rights Reserved.
----------------------------------------------------------------
NLS            : US7ASCII            EXECUTE MODE   : IMPORT
TARGET TABLE   : plc_tag_table       DATA FILE      : 5_plc_tag.csv
IMPORT MODE    : APPEND              FIELD TERM     : ,
ROW TERM       : \n                  ENCLOSURE      : "
ESCAPE         : \                   ARRIVAL_TIME   : FALSE
ENCODING       : NONE                HEADER         : FALSE
CREATE TABLE   : FALSE

 Progress bar                      Imported records        Error records
                                             80000                    0
```

데이터 로드 중 TAG 테이블 데이터를 확인해보면 실시간으로 데이터가 입력됨을 확인할 수 있다.

```
Mach> select count(*) from TAG;
count(*)
----------------------
16775979
[1] row(s) selected.
Mach> select count(*) from TAG;
count(*)
----------------------
17609187
[1] row(s) selected.
Mach> select count(*) from TAG;
count(*)
----------------------
18238357
```

```
[1] row(s) selected.
Elapsed time: 0.000
```

STREAM 동작의 결과 확인

아래와 같이 STREAM이 소스 테이블(plc_tag_table)의 데이터를 어디까지 읽었는지를 확인할 수 있다.

```
Mach> select name, state, end_rid from v$streams;
name      state   end_rid
-----------------------------------------------------------
EVENT_V0  RUNNING 909912
EVENT_V1  RUNNING 1584671
EVENT_C0  RUNNING 1312416
EVENT_C1  RUNNING 1268520
EVENT_C2  RUNNING 1636800
EVENT_C3  RUNNING 1197840
EVENT_C4  RUNNING 622728
EVENT_C5  RUNNING 972780
EVENT_C6  RUNNING 1021512
EVENT_C7  RUNNING 1287474
EVENT_C8  RUNNING 826956
EVENT_C9  RUNNING 1639032
EVENT_C10 RUNNING 725954
EVENT_C11 RUNNING 1511436
EVENT_C12 RUNNING 531079
EVENT_C13 RUNNING 1004400
EVENT_C14 RUNNING 741768
EVENT_C15 RUNNING 746604
[18] row(s) selected.
```

end_rid column의 값이 소스 테이블의 레코드 갯수와 동일하면 소스 테이블에서 더 이상 읽을 것이 없다는 뜻이다.

```
Mach> select name, state, end_rid from v$streams;
name      state   end_rid
-----------------------------------------------------------
EVENT_V0  RUNNING 2000000
EVENT_V1  RUNNING 2000000
EVENT_C0  RUNNING 2000000
EVENT_C1  RUNNING 2000000
EVENT_C2  RUNNING 2000000
EVENT_C3  RUNNING 2000000
EVENT_C4  RUNNING 2000000
EVENT_C5  RUNNING 2000000
EVENT_C6  RUNNING 2000000
EVENT_C7  RUNNING 2000000
EVENT_C8  RUNNING 2000000
EVENT_C9  RUNNING 2000000
EVENT_C10 RUNNING 2000000
EVENT_C11 RUNNING 2000000
EVENT_C12 RUNNING 2000000
EVENT_C13 RUNNING 2000000
EVENT_C14 RUNNING 2000000
EVENT_C15 RUNNING 2000000
[18] row(s) selected.
```

TAG 테이블의 데이터 갯수가 소스 테이블의 갯수 * STREAM의 갯수와 같으므로 STREAM이 정상적으로 모든 데이터를 읽었음을 확인할 수 있다.

```
Mach> select count(*) from TAG;
count(*)
-----------------------
36000000
[1] row(s) selected.
```

입력된 데이터의 시간 범위도 다음과 같이 확인할 수 있다.

```
Mach> select min(time), max(time) from TAG;
min(time)                       max(time)
-------------------------------------------------------------------
```

```
2009-01-28 07:03:34 000:000:000 2009-01-28 12:36:58 020:000:000
[1] row(s) selected.
```

데이터 추가

STREAM이 실제로 각 데이터 입력마다 반응하는지 확인하기 위해 insert 구문을 통해 확인해볼 수 있다.

```
Mach> insert into plc_tag_table values(TO_DATE('2009-01-28 12:37:00 000:000:000'), 50000, 50000, 50000, 50000, 500(
1 row(s) inserted.
```

PLC_TAG_TABLE에 레코드 하나를 더 추가한 순간 아래와 같이 각 스트림의 end_rid가 1건 늘어 2000001 건이 된 것을 확인할 수 있다.

```
Mach> select name, state, end_rid from v$streams;
name      state   end_rid
----------------------------------------------------------------
EVENT_V0  RUNNING 2000001
EVENT_V1  RUNNING 2000001
EVENT_C0  RUNNING 2000001
EVENT_C1  RUNNING 2000001
EVENT_C2  RUNNING 2000001
EVENT_C3  RUNNING 2000001
EVENT_C4  RUNNING 2000001
EVENT_C5  RUNNING 2000001
EVENT_C6  RUNNING 2000001
EVENT_C7  RUNNING 2000001
EVENT_C8  RUNNING 2000001
EVENT_C9  RUNNING 2000001
EVENT_C10 RUNNING 2000001
EVENT_C11 RUNNING 2000001
EVENT_C12 RUNNING 2000001
EVENT_C13 RUNNING 2000001
EVENT_C14 RUNNING 2000001
EVENT_C15 RUNNING 2000001
[18] row(s) selected.
```

TAG Analyzer 그래프

STREAM으로 입력된 데이터들의 그래프를 Tag Analyzer로 확인하면 아래와 같다.

마지막으로 입력한 데이터의 값이 다른 데이터의 값에 비해 크기 때문에 부각되어 보이는 것을 확인할 수 있다.

백업 및 마운트

데이터를 정기적으로 백업하는 것은 매우 중요하다. 이 챕터에서는 마크베이스의 데이터를 어떻게 백업하며, 백업한 데이터를 복구하는지를 기술한다. 또한 마크베이스는 백업된 데이터를 복구시키지 않고 검색할 수 있는 마운트 기능을 제공한다. 마운트 기능은 백업된 데이터를 읽어야 할 때 매우 빠르게 실행할 수 있다.

- 백업 및 복구
- 마운트

백업 개요

BACKUP/MOUNT

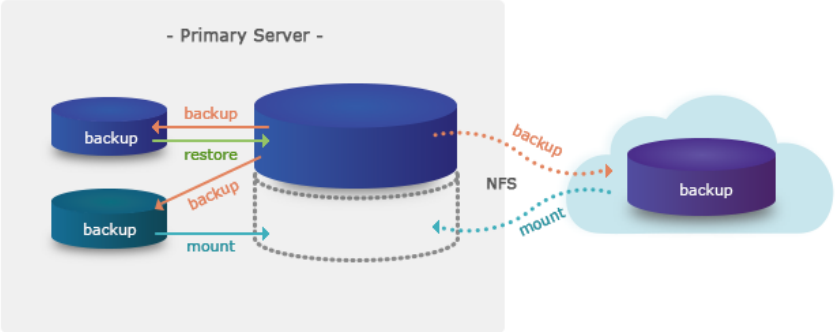
데이터베이스의 영속성을 보장하기 위해서 메모리에 저장된 데이터는 최대한 빨리 Disk에 저장된다. 그리고 Process Failure와 같은 일반적인 장애상황이 발생할 경우, Restart Recovery를 통해서 데이터베이스를 Consistent한 상태로 만든다. 하지만 Power Failure나 화재에 의한 Hardware의 피해가 발생할 경우, 데이터베이스의 복구는 불가하다. 이런 문제를 해결하기 위해서 별도의 디스크나 Hardware에 데이터를 주기적으로 다른 영역에 저장하여, 유사시 해당 데이터를 이용하여 데이터를 복구하는 기능이 데이터베이스 백업과 복구 기능이다.

데이터베이스 백업은 언제 수행하느냐에 따라서 크게 두 가지로 나누어 진다.

- Offline Backup
- Online Backup

첫번째, Offline Backup 기능은 DBMS를 Shutdown하고 데이터베이스를 복사하는 기능으로 Cold Backup이라고 부르기도 한다. 매우 간단하지만, 사용자의 서비스가 중단되는 단점이 있으므로 운영 중에는 사용하는 경우가 거의 없으며 초기 테스트나 데이터 구축 시에만 사용하는 경향이 있다.

두 번째, Online Backup은 DBMS가 동작 중일 때, 데이터베이스를 Backup하는 기능으로 Hot Backup이라고 부르기도 한다. 이 기능은 서비스를 중단하지 않고 수행될 수 있어 사용자의 Service Availability를 증가시켜 대부분의 DBMS Backup은 Online Backup을 의미한다. 다른 데이터베이스의 Backup과 달리 시계열 데이터베이스인 Machbase 는 Duration Backup을 제공한다. 이는 Backup시 백업될 Database의 시간을 지정하여 원하는 시간대의 데이터만 Backup할 수 있다.




```
backup database into disk = 'backup';
backup database from to_date('2015-07-14 00:00:00','YYYY-MM-DD HH24:MI:SS') to to_date('2015-07-14 23:59:59 999:999
               into disk = 'backup_20150714';
```

Backup된 데이터베이스는 장애 복구 과정을 거쳐서 기존 데이터베이스처럼 사용될 수 있다. 이 복구 방법을 Restore라고 한다. 이 Restore 기능은 파손된 데이터베이스를 삭제하고 백업된 데이터베이스 이미지를 Primary Database로 복구한다. 때문에 복구시 기존 데이터베이스를 삭제하고 machadmin -r 기능을 이용하여 복구한다.

```
machadmin -r 'backup'
```

Mount/unmount 기능은 Online으로 동작하는 기능으로 Backup된 데이터베이스를 현재 운영 중인 데이터베이스에 Attach하는 기능이다.

```
mount database 'backup' to mountName;
umount database mountName;
```

Database Backup

Machbase 에서는 데이터 백업을 할 때 두 가지 옵션을 제공한다. 운영 중인 DB의 정보를 백업하는 DATABASE 백업과 필요한 Table만 선택하여 백업할 수 있는 TABLE 백업 기능을 제공한다.

DB에서 제공하는 백업 명령은 다음과 같다.

```
BACKUP [ DATABASE | TABLE table_name ]  [ time_duration ] INTO DISK = 'path/backup_name';
time_duration = FROM start_time TO end_time
path = 'absolute_path' or  'relative_path'
```

```
# Directory backup
       BACKUP DATABASE INTO DISK = 'backup_dir_name';
# Set backup duration
      - Directory backup
       BACKUP DATABASE FROM TO_DATE('2015-07-14 00:00:00','YYYY-MM-DD HH24:MI:SS')
```

```
                    TO TO_DATE('2015-07-14 23:59:59','YYYY-MM-DD HH24:MI:SS')
                    INTO DISK = '/home/machbase/backup_20150714'
```

DB 백업을 수행할 때 옵션은 백업 타입, Time duration, 경로를 입력해야 한다. DATABASE 전체를 백업할 때는 백업 타입에 DATABASE를 입력하고, 특정 Table만 백업하려면 TABLE을 입력한 후 백업하려는 Table_Name을 입력한다. TIME_DURATION 구문은 필요한 기간의 데이터만 백업하도록 설정할 수 있다. FROM 항목에 백업을 원하는 날짜의 시작 시간을 입력하고 TO 항목에 백업의 마지막 날짜의 시간을 입력하면 그 기간의 데이터만 선택하여 백업할 수 있다. 예제 3번을 보면 TIME_DURATION 항목의 FROM에 '2015년 7월 14일 0시 0분 0초'로 설정하고 TO에 '2015년 7월 14일 23시 59분 59초'로 설정하여 2015년 7월 14일의 데이터만 백업되도록 설정하였다. 만약 DURATION 항목에 대한 정보를 입력하지 않으면 FROM 항목에는 '1970년 1월 1일 9시 0분 0초'로 설정되고 TO 항목에는 명령을 수행하는 시간으로 자동 설정된다. DURATION 절을 이용한 시간범위 백업은 TAG테이블과 TAG table을 포함한 데이터베이스에서 사용할 수가 없으며, 증분 데이터를 백업하는 기능인 INCREMENTAL BACKUP기능을 이용해야 한다.

마지막으로, 백업 수행의 결과를 저장할 저장 매체에 대한 설정이 필요하다. 디렉터리 단위로 생성하려면 DISK를 입력한다. 주의할 점은 생성물 저장되는 PATH 정보를 지정할 수 있는데 만약 상대 경로를 입력하면 현재 운영 중인 DB의 환경설정의 DB_PATH 항목에 지정된 경로에 생성된다. 만약 DB_PATH가 아닌 다른 곳에 저장하고 싶다면 '/'로 시작하는 절대 경로를 입력해야 한다.

Incremental Backup

증분 백업이란 이전에 수행한 백업 이후에 입력된 데이터만을 백업하는 기능이다. 증분 백업이 수행되는 대상은 Log, Tag 테이블의 데이터만 해당하며 lookup 테이블은 항상 모든 데이터를 백업한다. 증분 백업을 수행하기 위해서는 이전에 수행한 증분 백업 디렉토리나 전체 백업 디렉토리가 필요하다. 증분 백업은 다음과 같이 수행한다.

```
Mach> BACKUP DATABASE INTO DISK = 'backup1'; /* full backup 수행 */
Executed successfully.
Mach> ...

Mach> BACKUP DATABASE AFTER 'backup1' INTO DISK = 'backup2'; /* backup1 이후에 입력한 데이터만 증분 백업을 수행함 */
Executed successfully.
Mach> ...
```

증분 백업은 데이터베이스 전체(이때 lookup 테이블은 전체 백업이 됨), Log 테이블, Tag table에 대해서 가능하며 RESTORE기능을 이용하여 복구할 경우 증분 백업 이전에 백업한 백업 데이터도 필요하다. 현재 데이터를 삭제하고 이전 상태로 되돌리기 싫은 경우 아래에서 설명하는 MOUNT기능을 이용하면 된다.

Incremental Backup 주의 사항

위와 같이 backup1을 기준으로 증분 백업으로 backup2를 만든 경우, backup1이 유실(disk failure 등의 이유)되면, backup2를 사용하여 복구할 수 없다.

같은 이유로, 증분 백업을 하였을 때 이전 백업이 유실되면 이후 백업을 사용해서 복구할 수 없다.

아래와 같이 백업을 3번 진행하면 backup3의 이전 백업은 backup2가 되고 backup2의 이전 백업은 backup1이 된다.

따라서, backup1이 유실되면 backup2 와 backup3 모두 사용할 수 없고, backup2가 유실되면 backup3를 사용하여 복구할 수 없다.

```
Mach> BACKUP DATABASE INTO DISK = 'backup1'; /* full backup 수행 */
Executed successfully.
Mach> ...

Mach> BACKUP DATABASE AFTER 'backup1' INTO DISK = 'backup2'; /* backup1 이후에 입력한 데이터만 증분 백업을 수행함 */
Executed successfully.
Mach> ...

Mach> BACKUP DATABASE AFTER 'backup2' INTO DISK = 'backup3'; /* backup2 이후에 입력한 데이터만 증분 백업을 수행함 */
Executed successfully.
Mach> ...
```

Database Restore

Database Restore기능은 구문으로 제공되지 않고, Offline으로 machadmin -r 기능을 통해 복구할 수 있다. 복구전에 다음 사항을 체크해야 한다.

- Machbase 가 종료되었는가?
- 이전에 생성한 DB를 삭제하였는가?

```
machadmin -r backup_database_path;
```

```
backup database into disk = '/home/machbase/backup';
```

```
machadmin -k
machadmin -d
machadmin -r /home/machbase/backup;
```

Database Mount

장애상황을 대비하여 대량의 데이터베이스를 주기적으로 Backup 하고 데이터를 계속 추가하는 경우, 다음과 같은 문제점이 발생한다.

- 데이터를 저장하기 위한 디스크 비용 증가
- 운영 중인 Machine의 물리적 Disk공간의 한계

이 문제점을 해결하기 위해서 주기적으로 현재 서비스를 위해 필요한 데이터만을 남기고 삭제를 수행한다. 그러나 과거 데이터에 대한 참조가 필요할 경우에는 Backup 된 데이터베이스를 Restore하여 참조해야 하는데, 대단히 큰 Backup Image일 경우 복구시간이 많이 걸리고 또한 별도의 장비도 필요하다. 왜냐하면 Restore 기능은 현재 운영 중인 데이터베이스를 삭제해야 수행할 수 있기 때문이다. 이런 문제를 해결하기 위해서 Machbase 는 Database Mount 기능을 제공한다.

Database Mount기능은 Online으로 동작하는 기능으로 Backup된 데이터베이스를 현재 운영 중인 데이터베이스에 Attach하는 기능이다. 그리고 여러 개의 Backup Database을 운영 중인 Primary Database에 Attach하여 사용자는 여러 개의 Backup Database를 하나의 Database인 것처럼 참조 가능한다. 단 Mount한 Database에 대해서는 Read만 가능하다.

Mount DATABASE 명령은 기존에 Backup으로 생성된 데이터베이스 혹은 테이블 DATA를 현재 운영 중인 데이터베이스에서 조회 가능한 상태로 준비시켜 주는 기능이다. 그래서 Mount 된 DATABASE는 동일한 DB 명령어를 사용하여 데이터를 조회할 수 있다.

현재 Database Mount 기능의 제약 사항은 다음과 같다.

- Backup 정보는 Mount할 Database와 DB의 Major 번호와 Meta의 Major 번호가 호환 가능한 버전이어야 한다.
- Backup Data를 Mount할 경우 읽기만 가능하여 Index 생성, 데이터 입력 및 삭제 등은 지원하지 않는다.
- 현재 Mount된 DATABASE의 정보는 V$STORAGE_MOUNT_DATABASES를 조회하여 확인할 수 있다.
- 증분 백업 데이터를 mount하는 경우, 그 백업데이터에 기록된 증분 데이터만 검색되며 이전에 수행한 증분데이터를 따라가서 mount해 주지는 않는다.

Mount

Mount 명령을 수행하기 위해서는 Backup_database_path 정보와 DatabaseName이 필요하다. Backup_database_path는 Backup 명령을 통하여 생성된 DB의 위치 정보를 입력해야 하고, DatabaseName에는 Database에 Mount 할 때 구분할 수 있는 이름을 지정한다. Backup_database_path는 Backup 할 때와 동일하게 상대 경로를 입력할 경우 DB의 환경변수에 설정된 DB_PATH에 지정된 디렉터리를 기준으로 검색한다.

```
MOUNT DATABASE 'backup_database_path' TO mount_name;
```

```
MOUNT DATABASE '/home/machbase/backup' TO mountdb;
```

Unmount

Mount된 Database를 더 이상 사용하지 않을 경우 Umount 명령을 사용하여 제거할 수 있다.

```
UNMOUNT DATABASE mount_name;
```

```
UNMOUNT DATABASE mountdb;
```

MOUNT DB에서 데이터 조회

Backup DB의 DATA를 조회할 때 운영 중인 DB의 DATA를 조회하는 것과 동일한 SQL문을 이용하여 조회할 수 있다.

Mount 된 DB는 운영중인 DB의 SYS 권한의 사용자만 데이터를 조회할 수 있다. 데이터를 조회하기 위해서는 조회할 TableName 앞에 MountDBName과 UserName을 입력하고, 각각의 구분자로 '.'을 붙여서 사용해야 한다. MountDBName은 현재 Mount된 DB들 중 특정 DB를 지칭하기 위해 사용하고, UserName은 Mount된 DB의 Table을 소유한 User의 정보를 지칭하는 것이다.

```
SELECT column_name FROM mount_name.user_name.table_name;
```

```
SELECT * FROM mountdb.sys.backuptable;
```

데이터베이스 마운트

- MOUNT
- UNMOUNT
- 마운트된 데이터베이스에서 데이터 읽기

데이터를 분석하기 위해서 대량의 데이터를 지속적으로 저장하면 그 양이 매우 증가하므로 다음의 문제가 발생한다.

- 대량의 데이터 저장에 의한 디스크 비용 증가
- 데이터 분석용 장비의 디스크 한계

문제 해결을 위해서는 오래된 데이터를 백업하고, 주기적으로 삭제할 필요가 있다. 이후에 오래된 데이터를 읽을 필요가 있을 때, 백업된 데이터베이스를 읽기 위해서 데이터 복구를 실행하면 복구 과정에서 실행 시간이 오래 걸리는 것 뿐만 아니라, 데이터베이스를 오프라인 상태로 변환하고 현재 데이터를 모두 삭제한 상태에서 복구를 실행해야 하므로 서비스를 계속 진행하기 위해서는 별도의 장비가 필요한 문제점이 있다. 마크베이스는 이 문제를 해결하기 위해서 MOUNT 명령을 지원한다.

MOUNT명령은 데이터베이스가 서비스를 진행하면서도 백업된 데이터를 읽어들여서 현재 실행중인 데이터베이스와 별개로 새로운 데이터베이스를 생성한다. 하나의 서버에서 여러 개의 백업된 데이터베이스를 추가하여 동시에 데이터를 검색할 수 있으나, 마운트된 데이터베이스는 읽기 전용으로 데이터의 추가와 삭제는 불가능하다.

데이터베이스 MOUNT명령은 백업 데이터와 주 데이터베이스 내용을 동시에 읽을 수 있도록 한다. 따라서 마운트된 데이터베이스는 기존의 데이터 검색 방법과 동일하게 데이터를 검색할 수 있다.

MOUNT 명령을 실행하기 위해서는 다음의 제약조건을 만족시켜야 한다.

- 백업 데이터베이스의 버전과 메타데이터 버전이 호환 가능해야 한다.
- 마운트된 백업 데이터베이스에는 테이블 생성, 인덱스 생성 및 삭제, 데이터 추가 및 삭제를 실행할 수 없다.

마운트된 데이터베이스들에 대한 정보는 V$STORAGE_MOUNT_DATABASES 메타 테이블에서 얻을 수 있다.

MOUNT

마운트 명령을 실행하기 위해서는 백업 데이터베이스 경로명과 마운트할 데이터베이스 이름을 입력해야 한다.

백업 데이터베이스 경로는 백업 명령으로 실행한 디렉토리의 위치를 설정한다. 마운트할 데이터베이스 이름은 운영중인 데이터베이스와 구별하기 위해서 별도의 이름을 부여해야 한다.

백업 데이터베이스 경로명은 절대 경로명 ("/"문자로 시작되는 경로명)을 입력하거나, 백업 명령과 동일한 규칙으로 $MACHBASE_HOME/dbs를 기준으로 한 상대 경로명을 이용할 수 있다.

Syntax:

```
MOUNT DATABASE 'backup_database_path' TO mount_name;
```

Example:

```
MOUNT DATABASE '/home/machbase/backup' TO mountdb;
```

UNMOUNT

마운드된 데이터베이스 데이터가 더 이상 읽을 필요가 없다면, 마운트 상태를 해제하기 위해 UNMOUNT 명령을 사용한다.

Syntax:

```
UNMOUNT DATABASE mount_name;
```

Example:

```
UNMOUNT DATABASE mountdb;
```

마운트된 데이터베이스에서 데이터 읽기

마운트된 데이터베이스에서 데이터를 검색할 때는 기존과 동일한 SQL문을 이용한다.

SYS유저만 마운트된 데이터를 읽을 수 있다. SQL문에서 마운트된 데이터베이스의 테이블을 지정하기 위해서는 mount_name과 user_name을 "." 문자로 연결하여 지정해야 한다.

Syntax:

```
SELECT column_name FROM mount_name.user_name.table_name;
```

Example:

```
SELECT * FROM mountdb.sys.backuptable;
```

백업 및 복구

- 데이터베이스 백업
- 데이터베이스 복구
  - 단일 백업 파일 추출
  - 단일 백업 파일의 정보 확인

데이터베이스 백업

마크베이스의 백업 방법은 두 가지가 있다.

1. 현재 DB의 전체 백업
2. 특정 테이블만 선택해서 백업

Syntax:

```
BACKUP [ DATABASE | TABLE table_name ]  [ time_duration ] INTO [ DISK ] = 'path/backup_name';
time_duration = FROM start_time TO end_time
path = 'absolute_path' or  'relative_path'
BACKUP [ DATABASE | TABLE table_name ] AFTER 'previous_backup_dir'
```

Example:

```
# Directory backup
       BACKUP DATABASE INTO DISK = 'backup_dir_name';

# Set backup duration
      - Directory backup
       BACKUP DATABASE FROM TO_DATE('2015-07-14 00:00:00','YYYY-MM-DD HH24:MI:SS')
                        TO TO_DATE('2015-07-14 23:59:59','YYYY-MM-DD HH24:MI:SS')
                        INTO DISK = '/home/machbase/backup_20150714'
```

백업 명령을 실행할 때, 백업 타입과 duration 시간 조건절, 백업 대상 경로를 반드시 정의하여야 한다. 전체 데이터베이스를 백업하려면 "DATABASE"를, 특정 테이블을 백업하려면 "TABLE"을 백업 타입에 지정하고, 특정 테이블을 백업할 때에는 테이블 이름을 지정하여야 한다.

DURATION 조건절을 이용하여 백업 대상을 지정할 수 있다. 백업 대상 데이터의 시작 시간과 끝 시간을 FROM 및 TO 절에서 지정한다. 위 예제에서 "2015-07-14 00:00:00" 가 FROM으로 정의되었고, "2015-07-14 23:59:59" 이 TO로 정의되었으므로, 사용자는 2015년 7월 14일의 전체 데이터를 백업하는 것이다. duration 시간 조건절을 지정하지 않으면 "1970-01-01 00:00:00" 이 FROM으로 설정되고 실행되는 현재 시점이 TO절에 설정된다.

DURATION절은 Tag 테이블과 Tag 테이블을 포함하는 DATABASE에서는 이용할 수 없으며, 추가된 데이터만을 백업하려면 증분 백업 (Backup .... AFTER 'previous_backup') 문을 수행해야 한다.

백업 경로를 지정할 때, 상대 경로를 지정하면 "$MACHBASE_HOME/dbs" 아래에 백업 파일들이 생성되므로 주의하여야 한다. 절대 경로를 지정하려면 항상 "/"로 시작하는 경로를 설정하여야 한다.

데이터베이스 복구

백업 파일에서 데이터 복원을 수행할 때는 질의 명령으로 수행할 수 없으며, "machadmin -r" 명령을 데이터베이스가 동작하지 않는 상황에서 실행해야 한다. 백업 실행 이전에 다음의 조건들을 검토해야 한다.

- 마크베이스 데이터베이스가 정지 상태인가
- 현재의 데이터는 삭제되고 복구할 데이터로 대치되므로, 현재 데이터베이스의 삭제가 허용되는가
- 증분 백업에 대해서는 이전에 백업한 full backup까지의 증분 백업 파일이 필요하다.

Syntax:

```
machadmin -r backup_database_path
```

Example:

```
backup database into disk = '/home/machbase/backup';

machadmin -k
machadmin -d
machadmin -r /home/machbase/backup;
```

도구

마크베이스는 다양한 커맨드 라인 도구와 웹 기반 관리 도구, 데이터 수집 도구를 제공한다.

- Utilities
- Collector Guide
- Machbase Web Analytics(MWA)
- 태그 분석기 (Tag Analyzer)

유틸리티 모음

마크베이스 서버의 시작, 종료, 데이터 입력 및 출력을 위해서 사용되는 유틸리티들이다.

- machadmin
- machsql
- machloader
- csvimport/csvexport

CSVIMPORT/CSVEXPORT

'csvimport'와 'csvexport' 는 CSV 파일을 마크베이스 서버에 import/export 하기 위해 사용되는 도구이다.
CSV 파일에 대해서 machloader 를 이용하여 보다 간단하게 사용할 수 있도록 옵션을 단순화하였다.
아래 기술된 Option이외에도 machloader에서 사용할 수 있는 옵션을 모두 사용 가능하다.

csvimport

csvimport를 이용하면 쉽게 CSV 파일을 서버에 입력할 수 있다.

기본 사용법

테이블 명과 데이터 파일명을 다음의 옵션에 따라 입력하여 수행한다.

Option:

```
-t: 테이블명 지정 옵션
-d: 데이터 파일명 지정 옵션
* 옵션을 지정하지 않고 테이블명과 데이터 파일명만으로도 수행할 수 있다.
```

Example:

```
csvimport -t table_name -d table_name.csv
csvimport table_name file_path
csvimport file_path table_name
```

CSV 헤더 제외

입력시에 CSV 파일의 헤더를 제외하고 입력하려면 다음의 옵션을 사용한다.

Option:

```
-H: csv 파일의 첫번째 라인을 헤더로 인식하고 입력하지 않는다.
```

Example:

```
csvimport -t table_name -d table_name.csv -H
```

테이블 자동 생성

입력시 입력할 테이블을 생성하지 않은 경우, 다음 옵션을 통해서 테이블 생성도 동시에 수행할 수 있다.

Option:

```
-C: import할 때 테이블을 자동 생성한다. 칼럼명은 c0, c1, ... 자동으로 생성된다. 생성되는 칼럼은 varchar(32767) 타입이다.
-H: import할 때 csv 헤더명으로 칼럼명을 생성한다.
```

Example:

```
csvimport -t table_name -d table_name.csv -C
csvimport -t table_name -d table_name.csv -C -H
```

csvexport

'csvexport'로 데이터베이스 테이블 데이터를 CSV 파일로 쉽게 export할 수 있다.

기본 사용법

Option:

```
-t: 테이블명 지정 옵션
-d: 데이터 파일명 지정 옵션
* 옵션을 지정하지 않고 테이블명과 데이터 파일명만으로도 수행할 수 있다.
```

Example:

```
csvexport -t table_name -d table_name.csv
csvexport table_name file_path
csvexport file_path table_name
```

CSV 헤더 사용

다음의 옵션을 이용하면, export할 CSV 파일에 칼럼명으로 헤더를 추가할 수 있다.

Option:

```
-H: 테이블 칼럼명으로 csv 파일의 헤더를 생성한다.
```

Example:

```
csvexport -t table_name -d table_name.csv -H
```

MACHADMIN

마크베이스 서버를 시작, 종료하거나 생성, 삭제 및 실행 상태를 체크하기 위해서는 machadmin을 사용한다.

옵션 및 기능

machadmin의 옵션은 아래와 같다. 앞의 설치 절에서 설명한 기능은 생략한다.

```
mach@localhost:~$ machadmin -h
```

옵션	설명
-u, --startup/ --recovery[=simple,complex,reset]	마크베이스 서버 시작/ 복구 mode (기본: simple)
-s, --shutdown	마크베이스 서버 정상 종료
-c, --createdb	마크베이스 데이터베이스 생성
-d, --destroydb	마크베이스 데이터베이스 삭제
-k, --kill	마크베이스 서버 강제 종료
-i, --silence	출력 없이 실행
-r, --restore	백업에서 데이터베이스 복구
-x, --extract	백업 파일을 백업 디렉터리로 변환
-e, --check	마크베이스 서버 실행 체크
-t, --licinstall	라이선스 파일 설치
-f, --licinfo	설치된 라이선스 정보 출력

복구 모드

Syntax:

```
machadmin -u --recovery=[simple | complex | reset]
```

복구 모드는 다음과 같다.

- simple: 서버가 동작중일때 전원이 끊어지는 문제가 발생하지 않았다면, simple recovery 모드가 기본 실행된다.
- complex: complex recovery 모드는 simple 모드에 비해서 실행시간이 더 오래 걸린다. 전원이 끊어진 이후 재시작시에 기본으로 실행된다.
- reset: simple 혹은 complex 모드로 복구가 수행되지 않을 때, 모든 테이블의 모든 데이터를 검사하여 데이터베이스를 복구한다. 이 경우, 데이터의 일부 유실이 발생할 수 있다.

서버 정상 종료

Example:

```
mach@localhost:~$ machadmin -s

------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for the server shut down...
Server shut down successfully.
```

데이터베이스 생성

Example:

```
mach@localhost:~$ machadmin -c
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Database created successfully.
```

데이터베이스 삭제

Example:

```
mach@localhost:~$ machadmin -d
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Destroy Machbase database- Are you sure?(y/N) y
Database destroyed successfully.
```

서버 강제 종료

Syntax:

```
machadmin -k
```

Example:

```
mach@localhost:~$ machadmin -k
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for Machbase terminated...
Server terminated successfully.
```

침묵 모드 실행

machadmin 실행시 출력되는 메시지를 없앤다.

Syntax:

```
machadmin -i
```

데이터베이스 복구

Syntax:

```
machadmin -r backup_database_path
```

Example:

```
mach@localhost:~$ machadmin -r 'backup'
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Backed up database restored successfully.
```

서버 실행 유무 확인

Syntax:

```
machadmin -e
```

서버가 실행중이지 않을 때의 출력 예

```
mach@localhost:~$ machadmin -e
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
[ERR] Server is not running.
```

서버가 실행중일 때의 출력 예

```
mach@localhost:~$ machadmin -e
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Machbase server is already running with PID (14098).
```

라이선스 파일 설치

Syntax:

```
machadmin -t license_file
```

Example:

```
mach@localhost:~$ machadmin -t license.dat
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
License installed successfully.
```

라이선스 확인

Example:

```
mach@localhost:~$ machadmin -f
------------------------------------------------------------------
     Machbase Administration Tool
     Release Version - 5.1.9.community
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
                   INFORMATION
Install Date                      : 2018-12-20 11:34:43
Company#ID-ProjectName            : machbase
License Policy                    : CORE
License Type(Version 2)           : OFFICIAL
Host ID                           : FFFFFFFFFFFFFF
Issue Date                        : 2013-03-25
Expiry Date                       : 2037-03-18
Max Data Size For a Day(GB)       : 0
Percentage Of Data Addendum(%)    : 0
Overflow Action                   : 0
Overflow Count to Stop Per Month  : 0
Stop Action                       : 0
Reset Flag                        : 0
------------------------------------------------------------------
                   STATUS
Usage Of Data(GB)                 : 0.000000
Previous Checked Date             : 2018-12-22
Violation Count                   : 0
Stop Enabled                      : 0
------------------------------------------------------------------
License information displayed successfully.
```

MACHLOADER

마크베이스 서버에 텍스트 파일 데이터를 import/export하기 위해서 machloader를 사용한다. 기본적으로 CSV 파일을 이용하여 동작하지만, 다른 포맷도 지원한다.

machloader의 특징은 다음과 같다.

- machloader는 datetime 형식을 스키마 파일에서 지정할 수 있다. 지정하는 datetime 형식은 마크베이스 서버에서 지원하는 형식이어야 한다. 하나의 datetime 형식을 모든 필드에 적용할 수도 있고, 각 필드마다 다른 형식을 지정할 수도 있다.
- 입력 대상 테이블의 데이터를 삭제하고 입력하려면 "-m replace" 옵션을 사용하면 된다.
- machloader는 스키마와 데이터 파일의 정합성을 검증하지 않는다. 사용자는 스키마, 테이블, 데이터 파일이 정합성을 만족하는지 검사해야만 한다.
- machloader는 APPEND 모드를 기본으로 지원한다.
- machloader는 기본적으로는 "_ARRIVAL_TIME" 컬럼을 사용하지 않는다. 해당 컬럼 데이터를 import/export하려면 "-a" 옵션을 사용하여야 한다.

machloader의 옵션은 다음 명령으로 볼 수 있다.

```
[mach@localhost]$ machloader -h
```

옵션	설명
-s, --server=SERVER	마크베이스 서버의 IP 주소를 입력한다.(default : 127.0.0.1)
-u, --user=USER	접속할 사용자명을 입력한다.(default : SYS)
-p, --password=PASSWORD	접속할 사용자의 패스워드 (default : MANAGER)
-P, --port=PORT	마크베이스 서버의 포트 번호 (default : 5656)
-i, --import	데이터 import 명령 옵션
-o, --export	데이터 export 명령 옵션
-c, --schema	데이터베이스의 테이블 정보를 이용하여 스키마 파일을 만드는 명령 옵션
-t, --table=TABLE_NAME	스키마 파일을 생성할 테이블 명을 설정
-f, --form=SCHEMA_FORM_FILE	스키마 파일명을 지정
-d, --data=DATA_FILE	데이터 파일명을 지정
-l, --log=LOG_FILE	machloader 실행 로그 파일을 지정
-b, --bad=BAD_FILE	-i 옵션 실행시 입력 오류가 발생한 데이터를 기록하며, 에러 설명을 기록하는 파일명을 지정한다.
-m, --mode=MODE	-i 옵션 실행시 import 방법을 지시한다. append 또는 replace 옵션이 사용가능 하다. append는 기존 데이터 이후에 데이터를 입력하고 replace는 기존 데이터를 삭제하고 데이터를 입력한다.
-D, –delimiter=DELIMITER	각 필드 구분자를 설정한다. 기본값은 ','이다.
-n, --newline=NEWLINE	각 레코드 구분자를 설정한다. 기본값은 '₩n'이다.
-e, --enclosure=ENCLOSURE	각 필드의 enclosing 구분자를 설정한다.
-r, --format=FORMAT	파일 입력/출력 시 포맷을 지정한다. (default : csv)
-a, --atime	내장 컬럼 "_ARRIVAL_TIME"을 사용할 것인지를 결정한다. 기본값은 사용하지 않는 것이다.
-z, --timezone	Set timezone ex) +0900 -1230
-I, --silent	저작권 관련 출력 및 import/export 상태 정보를 표시하지 않는다.
-h, --help	옵션 리스트를 표시한다.
-F, --dateformat=DATEFORMAT	컬럼 dateformat을 설정한다. ("_arrival_time YYYY-MM-DD HH24:MI:SS") • dateformat 대신 'unixtimestamp' 을 설정하면, 입력되는 값을 unix timestamp 값으로 간주한다. ("time_column unixtimestamp") • dateformat 대신 'nanotimestamp' 을 설정하면, 입력되는 값을 nanosecond 단위의 timestamp 값으로 간주한다. ("time_column nanotimestamp") ⓘ unixtimestamp, nanotimestamp format 은 5.7 부터 지원합니다.
-E, --encoding=CHARACTER_SET	입/출력하는 파일의 인코딩을 설정한다. 지원되는 인코딩은 UTF8(기본값), ASCII, MS949, KSC5601, EUCJP, SHIFTJIS, BIG5, GB231280, UTF16이다.

옵션	설명
-C, --create	import시에 table이 없으면 생성한다.
-H, --header	import/export시에 헤더 정보의 유무를 설정한다. 기본값은 미설정이다.
-S, --slash	backslash 구분자를 지정한다.

기본 사용법

아래 사용법을 실행하기 전에 테이블을 먼저 생성해야 한다.

CSV 파일 Import

마크베이스 서버에 CSV 파일을 import한다.

Option:

```
-i: import 지정 옵션
-d: 데이터 파일명 지정 옵션
-t: 테이블명 지정 옵션
```

Example:

```
machloader -i -d data.csv -t table_name
```

CSV 파일 Export

데이터를 CSV 파일에 기록한다.

Option:

```
-o: export 지정 옵션
-d: 데이터 파일명 지정 옵션
-t: 테이블명 지정 옵션
```

Example:

```
machloader -o -d data.csv -t table_name
```

CSV 파일 헤더 사용

CSV 파일의 헤더 관련 설정이다.

Option:

```
-i -H: import 할 때 csv 파일의 첫번째 라인을 헤더로 인식한다. 따라서 첫번째 라인은 입력에서 제외된다.
-o -H: export 할 때 테이블의 컬럼명으로 csv 헤더를 생성한다.
```

Example:

```
machloader -i -d data.csv -t table_name -H
machloader -o -d data.csv -t table_name -H
```

테이블 자동 생성

테이블 자동 생성에 관련한 내용이다.

Option:

```
-C: import할 때 테이블을 자동 생성한다. 컬럼명은 c0, c1, ... 자동으로 생성된다. 생성되는 컬럼은 varchar(32767) 타입이다.
```

```
-H: import할 때 csv 헤더명으로 컬럼명을 생성한다.
```

Example:

```
machloader -i -d data.csv -t table_name -C
machloader -i -d data.csv -t table_name -C -H
```

CSV 포맷 이외 파일

CSV 포맷이 아닌 파일에 대해서 구분자를 설정하여 사용한다.

Option:

```
-D: 각 필드의 구분자 지정 옵션
-n: 각 레코드 구분자 지정 옵션
-e: 각 필드의 enclosing character 지정 옵션
```

Example:

```
machloader -i -d data.txt -t table_name -D '^' -n '\n' -e '"'
machloader -o -d data.txt -t table_name -D '^' -n '\n' -e '"'
```

입력 모드 지정

import 시 (-i 옵션 설정 시) REPLACE와 APPEND의 두 가지 모드가 있다. APPEND가 기본값이다. REPLACE 모드인 경우, 기존 데이터를 삭제하므로 주의해야 한다.

Option:

```
-m: import 모드 지정
```

Example:

```
machloader -i -d data.csv -t table_name -m replace
```

접속 정보 지정

서버 IP, 사용자, 패스워드를 별도로 지정한다.

Option:

```
-s: 서버 IP 주소 지정 (default: 127.0.0.1)
-P: 서버 포트 번호 지정 (default: 5656)
-u: 접속할 사용자명 지정 (default: SYS)
-p: 접속할 사용자의 패스워드 지정 (default: MANAGER)
```

Example:

```
machloader -i -s 192.168.0.10 -P 5656 -u mach -p machbase -d data.csv -t table_name
```

로그 파일 생성

machloader의 실행 로그 파일을 생성한다.

Option:

```
-b: import할 때 입력되지 않은 데이터를 생성할 로그 파일명 설정한다.
-l: import할 때 입력되지 않은 데이터와 에러 메시지를 생성할 로그 파일명을 설정한다.
```

Example:

```
machloader -i -d data.csv -t table_name -b table_name.bad -l table_name.log
```

스키마 파일 생성

machloader의 스키마 파일을 생성할 수 있다. 스키마 파일을 이용하여 데이터 타입 형식을 바꾸거나 테이블과 데이터 파일의 컬럼 수가 다른 경우에도 import/export 가 가능하다.

Option:

```
-c: 스키마 파일 생성 옵션
-t: 테이블명 지정 옵션
-f: 생성될 스키마 파일명 지정 옵션
```

Example:

```
machloader -c -t table_name -f table_name.fmt
machloader -c -t table_name -f table_name.fmt -a
```

스키마 파일에서 datetime 형식 설정

DATEFORMAT 옵션으로 dateformat을 원하는 대로 설정할 수 있다.

Syntax:

```
# 모든 datetime 컬럼에 대해서 설정한다.
DATEFORMAT <dateformat>

# 개별 datetime 컬럼에 대해서 설정한다.
DATEFORMAT <column_name> <format>
```

Example:

```
-- 스키마 파일(datetest.fmt)에 datetest.csv 파일의 각 필드에 맞게 dateformat을 설정한다.
datetest.fmt
table datetest
{
INS_DT datetime;
UPT_DT datetime;
}
DATEFORMAT ins_dt "YYYY/MM/DD HH12:MI:SS"
DATEFORMAT upt_dt "YYYY DD MM HH12:MI:SS"

datetest.csv
2017/02/20 11:05:23,2017 20 02 11:05:23
2017/02/20 11:06:34,2017 20 02 11:06:34



-- datetest.csv 파일을 import 하고 입력된 데이터를 확인한다.
machloader -i -f datetest.fmt -d datetest.csv
----------------------------------------------------------------
Machbase Data Import/Export Utility.
Release Version 5.1.9.community
Copyright 2014, MACHBASE Corporation or its subsidiaries.
All Rights Reserved.
----------------------------------------------------------------
Import time : 0 hour 0 min 0.39 sec
Load success count : 2
Load fail count : 0

mach> SELECT * FROM datetest;
INS_DT UPT_DT
----------------------------------------------------------------
```

```
2017-02-20 11:06:34 000:000:000 2017-02-20 11:06:34 000:000:000
2017-02-20 11:05:23 000:000:000 2017-02-20 11:05:23 000:000:000
[2] row(s) selected.
Elapsed time: 0.000
```

IGNORE

CSV 파일의 특정 필드를 입력하려 하지 않을 때, IGNORE 옵션을 fmt 파일에 설정할 수 있다.

ignoretest.csv 파일은 세 개의 필드를 갖지만, 마지막 필드가 필요 없을 경우, fmt 파일에 필요 없는 컬럼에 IGNORE를 명시한다.

Example:

```
-- ignoretest.fmt 파일에 마지막 필드에 대해서 ignore 옵션을 설정한다.
ignoretest.fmt
table ignoretest
{
ID integer;
MSG varchar(40);
SUB_ID integer IGNORE;
}

ignoretest.csv
1, "msg1", 3
2, "msg2", 4


-- ignoretest.csv 파일을 import 하고 입력된 데이터를 확인한다.
machloader -i -f ignoretest.fmt -d ignoretest.csv
------------------------------------------------------------------
Machbase Data Import/Export Utility.
Release Version 5.1.9.community
Copyright 2014, MACHBASE Corporation or its subsidiaries.
All Rights Reserved.
------------------------------------------------------------------
NLS : US7ASCII EXECUTE MODE : IMPORT
SCHMEA FILE : ignoretest.fmt DATA FILE : ignoretest.csv
IMPORT_MODE : APPEND FILED TERM : ,
ROW TERM : \n ENCLOSURE : "
ARRIVAL_TIME : FALSE ENCODING : NONE
HEADER : FALSE CREATE TABLE : FALSE

Progress bar Imported records Error records
2 0

Import time : 0 hour 0 min 0.39 sec
Load success count : 2
Load fail count : 0


mach> SELECT * FROM ignoretest;
ID MSG
------------------------------------------------------------
2 msg2
1 msg1
[2] row(s) selected.
Elapsed time: 0.000
```

컬럼 개수가 필드 개수보다 많은 경우

테이블의 컬럼 개수가 데이터 파일의 필드 개수보다 많은 경우에는 스키마 파일에 지정된 컬럼만 입력되고 다른 컬럼은 NULL로 입력된다.

컬럼 개수가 필드 개수보다 적은 경우

테이블의 컬럼 개수가 데이터 파일의 필드 개수보다 적은 경우에는 테이블에 없는 필드는 IGNORE 옵션을 제외하고 입력하여야 한다.

Example:

```
-- 마지막 필드에 대해서 ignore 옵션을 설정해서 제외한다.
loader_test.fmt
table loader_test
{
ID integer;
MSG varchar (40);
SUB_ID integer IGNORE;
}
```

MACHSQL

MACHSQL은 터미널 화면을 통해 SQL질의를 수행하는 대화형 도구이다.

구동 옵션 설명

```
[mach@localhost]$ machsql -h
```

짧은옵션	긴 옵션	설명
-s	--server	접속할 서버의 IP 주소 (default : 127.0.0.1)
-u	--user	사용자명 (default : SYS)
-p	--password	사용자 패스워드 (default : MANAGER)
-P	--port	서버의 포트 번호 (default : 5656)
-n	--nls	NLS 설정
-f	--script	실행할 SQL 스크립트 파일
-z	--timezone=+-HHMM	Timezone 설정 ex) +0900 -1230
-o	--output	질의 결과를 저장할 파일명
-i	--silent	저작권 출력 없이 실행
-v	--verbose	상세 출력
-r	--format	출력 파일 포맷 지정 (default: csv)
-h	--help	옵션 출력
-c	N/A	Connection 매개변수 추가(6.1이후 버전부터 지원)

Example:

```
machsql -s localhost -u sys -p manager
machsql --server=localhost --user=sys --password=manager
machsql -s localhost -u sys -p manager -f script.sql
# 6.1 이후버전부터 지원
machsql -s 127.0.0.1 -u sys -p manager -P 8888 -c ALTERNATIVE_SERVERS=192.168.0.147:9209;CONNECTION_TIMEOUT=10
```

환경변수 MACHBASE_CONNECTION_STRING

기본 접속 매개변수를 지정한다. 예를 들어 CONNECTION_TIMEOUT 값 설정 및 ALTERNATIVE_SERVERS 설정을 추가하기 위해 다음의 환경변수를 설정할 수 있다.

```
export MACHBASE_CONNECTION_STRING=ALTERNATIVE_SERVERS=192.168.0.148:8888;CONNECTION_TIMEOUT=3
```

-c 옵션으로 접속 매개변수를 지정하면 환경변수보다 우선하여 수행된다. 이 기능은 6.1 이후 버전부터 지원한다.

SHOW 명령어

테이블, 테이블스페이스, 인덱스 등의 정보를 출력한다.

SHOW 명령어 목록

- SHOW INDEX
- SHOW INDEXES
- SHOW INDEXGAP
- SHOW LSM
- SHOW LICENSE
- SHOW STATEMENTS

- SHOW STORAGE
- SHOW TABLE
- SHOW TABLES
- SHOW TABLESPACE
- SHOW TABLESPACES
- SHOW USERS

SHOW INDEX

인덱스 정보를 출력한다.

Syntax:

```
SHOW INDEX index_name
```

Example:

```
Mach> CREATE TABLE t1 (c1 INTEGER, c2 VARCHAR(10));
Created successfully.
Mach> CREATE VOLATILE TABLE t2 (c1 INTEGER, c2 VARCHAR(10));
Created successfully.
Mach> CREATE INDEX t1_idx1 ON t1(c1) INDEX_TYPE LSM;
Created successfully.
Mach> CREATE INDEX t1_idx2 ON t1(c1) INDEX_TYPE BITMAP;
Created successfully.
Mach> CREATE INDEX t2_idx1 ON t2(c1) INDEX_TYPE REDBLACK;
Created successfully.
Mach> CREATE INDEX t2_idx2 ON t2(c2) INDEX_TYPE REDBLACK;
Created successfully.

Mach> SHOW INDEX t1_idx2;
TABLE_NAME                              COLUMN_NAME                             INDEX_NAME
-----------------------------------------------------------------------------------------------------------------------------
INDEX_TYPE   BLOOM_FILTER  KEY_COMPRESS  MAX_LEVEL   PART_VALUE_COUNT BITMAP_ENCODE
-------------------------------------------------------------------------------
T1                                      C1                                      T1_IDX2
LSM          ENABLE   COMPRESSED    2           100000      EQUAL
[1] row(s) selected.
```

SHOW INDEXES

인덱스 전체 리스트를 출력한다.

Syntax:

```
SHOW INDEXES
```

Example:

```
Mach> CREATE TABLE t1 (c1 INTEGER, c2 VARCHAR(10));
Created successfully.
Mach> CREATE VOLATILE TABLE t2 (c1 INTEGER, c2 VARCHAR(10));
Created successfully.
Mach> CREATE INDEX t1_idx1 ON t1(c1) INDEX_TYPE LSM;
Created successfully.
Mach> CREATE INDEX t1_idx2 ON t1(c1) INDEX_TYPE BITMAP;
Created successfully.
Mach> CREATE INDEX t2_idx1 ON t2(c1) INDEX_TYPE REDBLACK;
Created successfully.
Mach> CREATE INDEX t2_idx2 ON t2(c2) INDEX_TYPE REDBLACK;
Created successfully.

Mach> SHOW INDEXES;
TABLE_NAME                              COLUMN_NAME                             INDEX_NAME
-----------------------------------------------------------------------------------------------------------------------------
INDEX_TYPE
---------------
T1                                      C1                                      T1_IDX1
```

```
LSM
T1                                      C1                                      T1_IDX2
LSM
T2                                      C2                                      T2_IDX2
REDBLACK
T2                                      C1                                      T2_IDX1
REDBLACK
[4] row(s) selected.
```

SHOW INDEXGAP

인덱스 생성 GAP 정보를 출력한다.

Example:

```
Mach> SHOW INDEXGAP
TABLE_NAME                              INDEX_NAME                              GAP
-----------------------------------------------------------------------------------------------------------------
INDEX_TABLE                             T1_IDX1                                 0
INDEX_TABLE                             T1_IDX2                                 0
```

SHOW LSM

LSM 인덱스 생성 정보를 출력한다.

Example:

```
Mach> SHOW LSM;
TABLE_NAME                              INDEX_NAME                              LEVEL       COUNT
-----------------------------------------------------------------------------------------------------------------
T1                                      IDX1                                    0           0
T1                                      IDX1                                    1           100000
T1                                      IDX1                                    2           0
T1                                      IDX1                                    3           0
T1                                      IDX2                                    0           100000
T1                                      IDX2                                    1           0
[6] row(s) selected.
```

SHOW LICENSE

라이선스 정보를 출력한다.

Example:

```
Mach> SHOW LICENSE
INSTALL_DATE          ISSUE_DATE            EXPIRY_DATE  TYPE        POLICY
-------------------------------------------------------------------------------
2016-07-01 10:24:37   20160325              20170325    2           0
[1] row(s) selected.
```

SHOW STATEMENTS

서버에 등록(Prepare, Execute, Fetch)된 모든 질의문을 출력한다.

Example:

```
Mach> SHOW STATEMENTS
USER_ID     SESSION_ID  QUERY
-----------------------------------------------------------------------------------------------------------------
0           2           SELECT ID USER_ID, SESS_ID SESSION_ID, QUERY FROM V$STMT
[1] row(s) selected.
```

SHOW STORAGE

사용자가 생성한 테이블 별 디스크 사용량을 출력한다.

Syntax:

```
SHOW STORAGE
```

Example:

```
Mach> CREATE TAGDATA TABLE TAG (name varchar(20) primary key, time datetime basetime, value double summarized);
Created successfully.

Mach> SHOW STORAGE
TABLE_NAME                                              DATA_SIZE           INDEX_SIZE          TOTAL_SIZE
-------------------------------------------------------------------------------------------------------------
_TAG_DATA_0                                             50335744            0                   50335744
_TAG_DATA_1                                             50335744            0                   50335744
_TAG_DATA_2                                             50335744            0                   50335744
_TAG_DATA_3                                             50335744            0                   50335744
_TAG_META                                               0                   0                   0
```

SHOW TABLE

사용자가 생성한 테이블의 정보를 출력한다.

Syntax:

```
SHOW TABLE table_name
```

Example:

```
Mach> CREATE TABLE t1 (c1 INTEGER, c2 VARCHAR(10));
Created successfully.
Mach> CREATE INDEX t1_idx1 ON t1(c1) INDEX_TYPE LSM;
Created successfully.
Mach> CREATE INDEX t1_idx2 ON t1(c1) INDEX_TYPE BITMAP;
Created successfully.

Mach> SHOW TABLE T1
[ COLUMN ]
-----------------------------------------------------------------
NAME                          TYPE                LENGTH
-----------------------------------------------------------------
C1                            integer             11
C2                            varchar             10

[ INDEX ]
-----------------------------------------------------------------
NAME                          TYPE                COLUMN
-----------------------------------------------------------------
T1_IDX1                       LSM                 C1
T1_IDX2                       LSM                 C1
```

SHOW TABLES

사용자가 생성한 테이블 전체 목록을 출력한다.

Example:

```
Mach> SHOW TABLES
NAME
-------------------------------------------
BONUS
```

```
DEPT
EMP
SALGRADE
[4] row(s) selected.
```

SHOW TABLESPACE

테이블 스페이스 정보를 출력한다.

Example:

```
Mach> CREATE TABLE t1 (id integer);
Created successfully.
Mach> CREATE INDEX t1_idx_id ON t1(id);
Created successfully.

Mach> SHOW TABLESPACE SYSTEM_TABLESPACE;
[TABLE]
NAME                                    TYPE
-------------------------------------------------------
T1                                      LOG
[1] row(s) selected.

[INDEX]
TABLE_NAME                              COLUMN_NAME                             INDEX_NAME
-------------------------------------------------------------------------------------------------------------------------
T1                                      ID                                      T1_IDX_ID
[1] row(s) selected.
```

SHOW TABLESPACES

테이블스페이스 전체 목록을 출력한다.

Example:

```
Mach> CREATE TABLESPACE tbs1 DATADISK disk1 (DISK_PATH="tbs1_disk1"), disk2 (DISK_PATH="tbs1_disk2"), disk3 (DISK_P
Created successfully.

-- 데이터를 입력한다
...
...


Mach> SHOW TABLESPACES;
NAME                                                                                                    DISK_COUNT  USAGE
-------------------------------------------------------------------------------------------------------------------------
SYSTEM_TABLESPACE                                                                                       1           0
TBS1                                                                                                    3           25824256
[2] row(s) selected.
```

SHOW USERS

사용자 목록을 출력한다.

Example:

```
Mach> CREATE USER testuser IDENTIFIED BY 'test1234';
Created successfully.

Mach> SHOW USERS;
USER_NAME
-----------------------------------------
SYS
TESTUSER
```

```
[2] row(s) selected.
```

MWA (Machbase Web Analytics)

Machbase Web Analytics (MWA)는 Python 2.7과 Flask기반의 Werkzeug, Jinja2로 개발된 Web application이다.

Configuration

MWA는 5001포트를 이용하여 클라이언트와 통신을 수행한다. 해당 포트를 사용할 수 있는지 확인하려면, linux 운영체제는 iptables 명령을 이용하고, windows는 방화벽 설정을 참조하여 통신 문제를 해결해야 한다.

추가로, $MACHBASE_HOME/lib 폴더를 $LD_LIBRARY_PATH에 추가하고 libmachbasecli_dll.so 파일이 그 폴더내에 있는지 확인해야 한다.

How to run the Server

MWA 서버는 $MACHBASE_HOME/bin/MWAserver 스크립트 파일을 이용해 실행한다. 이 스크립트는 $MACHBASE_HOME 환경변수를 이용하므로 해당 환경 변수를 반드시 설정해야 한다. 이 스크립트는 START, STOP, RESTART, RESET, PORT 명령 옵션을 지원한다. 'MWAserver start' 명령으로 서버를 실행할 수 있다. port 명령 옵션을 이용하면 MWAserver가 이용하는 기본 포트 번호인 5001이 아닌 다른 포트를 이용하여 MWA가 동작하도록 설정할 수 있다.

Example:

```
[mach@localhost ]$./MWAserver port 1234
WEBSERVER PORT CHANGED : 1234
```

명령 옵션은 아래와 같다.

List of commands

```
[mach@localhost flask]$./MWAserver help
List of commands:
  * MWAserver start; Startup WebServer
  * MWAserver restart; Restart WebServer
  * MWAserver stop; Shutdown WebServer
  * MWAserver reset; Reset WebServer database
  * MWAserver port NUMBER; Change WebServer port.
```

Connect to MWA with Web Browser

웹 브라우저를 통해서 MWA 서버 IP와 포트 번호(예 http:127.0.0.1:5001)를 입력하여 MWA에 접속할 수 있다. 웹 브라우저로는 최신 크롬 브라우저를 추천한다. 접속에 성공하였다면, 로그인 화면이 표시될 것이다. 기본 로그인 계정과 패스워드는 "admin"/"machbase"이다.

How to Use MWA

MWA 의 주 메뉴는 Dashboard, Query, Collection, DB Admin, Preferences 이다.

Overview

사용자 인증

MWA에는 ADMIN과 USER의 두 가지 종류의 사용자가 있다. ADMIN 사용자 계정은 모든 메뉴와 기능을 사용할 수 있고, USER 계정은 ADMIN 사용자가 허가한 기능만 사용할 수 있다.

Group

사용자 권한부여 및 설정을 쉽게 하기 위해서 여러 사용자들을 그룹을 설정할 수 있다. 화면의 좌측 위에서 원하는 그룹 카테고리를 선택할 수 있다. 그룹 카테고리는 "Users"메뉴의 "Available Groups" 메뉴에서 설정할 수 있다. ADMIN 계정 사용자는 모든 권한을 갖는다. ADMIN 계정 사용자만 그룹을 등록할 수 있다.

Permission

저장된 Bookmark 질의, 그리드, 대시보드등의 리소스에 대한 권한을 설정한다. 권한은 ALL, USER, OWNER 세 가지이다.

ALL : 모든 사용자는 로그인하지 않고 리소스에 접근 가능하다.

USER : 로그인 한 사용자만 리소스에 접근할 수 있다.

OWNER : 그 리소스를 생성한 사용자만 접근할 수 있다.

리소스 생성자가 아니면 ADMIN 권한을 갖는 사용자만 권한 없이 변경할 수 있다.

Change a Server

화면 오른쪽 구석에서 등록한 서버목록을 볼 수 있고, 선택에 따라 다른 서버로 접속할 수 있다. 서버의 IP주소와 포트번호를 이용하여 접속 서버를 확인할 수 있다.

Dashboard

MWA는 질의 결과를 Grid, 차트로 나타내고 이 결과를 대시보드 형태로 표시하는 기능이 있다. 대시보드를 생성하려면 먼저 그리드, 차트를 생성하여 등록하고, 대시보드를 생성하여 등록된 차트, 그리드등을 원하는 형태로 배치하고, 대시보드를 등록하는 것이다. 대시보드를 한번 등록하면 원본 그리드, 차트를 변경하더라도 그 대시보드에 등록된 차트, 그리드는 변경되지 않는다.

사용자는 오른쪽의 대시보드와 관련된 정보를 생성하고, 변경하고, 삭제하거나 선택할 수 있다. 그리고 대시보드에 외부 url을 우하단의 "Link URL"항목에 입력하여 대시보드에 외부 페이지를 입력할 수 있다. 사용자마다 권한여부에 따라 표시될 수 있는 대시보드 항목이 다르게 돌 수 있으므로, 로그인하지 않아도 데이터를 표출하려면 권한을 ALL로 설정해야 한다.

Dashboard 생성 및 편집

대시보드를 생성하고 차트나 그리드등을 추가하기 위해서 대시보드 편집화면에서 "Row+"버튼을 이용하여 row를 생성하고, "Row-"버튼을 이용하여 아래쪽부터 row를 삭제할 수 있다. Row에 패널을 생성했다면 그 패널을 삭제한 이후에 삭제할 수 있다. Chart+, Grid+, HTML+, URL+ 버튼을 이용하여 row에 패널을 추가할 수 있다. 패널의 내용으로 저장해 둔 차트나 그리드를 선택할 수 있다. 단, 사용자로부터 입력을 받는 입력 매개변수를 갖는 차트와 그리드는 패널에 등록할 수 없다.

HTML+ 버튼을 이용하여 HTML 또는 자바스크립트, chart.js를 이용하는 jquery 데이터를 표시할 수 있다. ID 애트리뷰트를 설정했다면, 다른 대시보드 구성요소와 동일한 값이 설정될 수 있으므로, ID를 "_ID"로 지정하여 사용하면 된다.

URL을 이용하여 접근할 때에는 'X-Frame-Options' 값에 따라 접근할 수 없는 경우가 있으므로 주의해야 한다. 데쉬보드의 각 패널은 각각 리프래쉬 설정을 할 수 있다.

Chart

SQL질의를 그래프로 표시할 수 있다. Chart의 Result탭에서 질의결과를 확인할 수 있고 Setting tab에서 차트 표시에 관련한 파라메터를 설정할 수 있으며, 결과로 표시되는 차트는 Chart 탭에서 확인할 수 있다.

MWA는 시계열 차트를 쉽게 생성할 수 있는 "Builder"를 제공한다. 사용자는 집계할 시간 간격을 설정할 수 있고, X축과 Y축에 해당되는 데이터를 설정할 수 있다. Y축 값이 숫자인 경우, SUM 또는 AVERAGE와 같은 함수를 사용할 수 있다. 숫자가 아닌 경우, 레코드 카운트를 이용하여 표시할 수 있다. 시계열 그래프를 생성하면 기본으로 라인 차트를 생성한다. 기본 설정은 Setting 탭에서 변경할 수 있다. 차트 화면의 아래쪽에 그리드를 표시하려면 Config 버튼을 클릭하고 원하는 컬럼을 Columns 탭에서 선택한다. 컬럼 데이터를 그리드에 표시할 때 좌우 폭은 비율에 따라 설정된다. 예를 들어 Width에 1, 2, 1의 값을 주면 화면 표시 비율은 25%, 50%, 25%의 순으로 설정된다.

질의문을 실행할 때, 입력 매개변수를 설정할 수 있다. 입력매개변수는 Variables 탭에서 Config 버튼으로 등록해야 한다. 매개변수의 type은 Text, number, date, time, datetime, 및 select query를 선택할 수 있다.

질의문에서 매개변수의 이름은 {} 문자로 둘러싸서 기술한다. 질의가 실행될 때, 매개변수는 사용자가 입력한 값으로 치환된다. 단순히 치환하므로 문자열을 사용할 경우 따옴표를 이용해야 한다. SELECT문의 경우 옵션에서 ',' 문자를 이용하여 분리해야 한다.

```
e.g.: SELECT * FROM TEST_TABLE WHERE C3 = '{V3}'
```

Preview 버튼으로 차트 설정을 저장하지 않고 차트 데이터를 표시할 수 있다. View 탭에서 Output 버튼으로 결과값을 Excel, JSON, csv, TDE(Tableau data extract) 타입의 파일로 저장할 수 있다.

Grid

질의 결과를 테이블 형태로 표시하기 위해서 Grid를 사용한다. 질의 결과에서 필요하지 않은 컬럼을 제외하고 표시할 수도 있다. Builder 버튼을 이용하여 WHERE절을 작성하는데 도움을 받을 수 있다. 단, 이 기능은 단순 조건절만을 지원한다. 결과 데이터에서 특정 컬럼만을 선택하려면 Config 버튼을 클릭하여 Columns 탭에서 "+"버튼을 클릭한 후, 타이틀과 출력 너비 설정을 하고 사용할 수 있다.

Column width는 chart의 grid 설정에서 설명한 것처럼 백분율 비율로 처리된다. 질의 매개변수 설정 또한 Chart와 같은 기능을 수행한다. Output에서 질의 결과를 다양한 파일로 저장할 수 있는 점도 차트와 동일한 기능이다.

Query

SQL

질의를 수행하여 결과를 표시한다. LIMIT또는 DURATION 관련 설정이 오른쪽 패널에 표시된다.

질의문을 저장해 두고 재실행할 수 있다. 또한 실행한 질의문들을 히스토리창에서 불러 올 수 있다. 질의 실행창에서 결과를 그리드나 차트 화면으로 전환할 수 있다. 북마크된 질의는 "Bookmark Quries" 메뉴에서 확인할 수 있다.

"Tables"탭에서 생성한 테이블 리스트와 각 테이블 스키마를 확인할 수 있다. 테이블 명을 클릭하면 그 테이블의 일부 데이터가 표시된다. 질의를 매 5초마다 재실행하여 결과를 리프레쉬한다. "Results"탭에서 질의 결과를 확인할 수 있다. "Input selected text at cursor position" 체크박스를 선택한 경우, 클릭한 위치의 문자열이 질의 입력창에 자동으로 입력된다. Grid 또는 Chart와 같은 방법으로 질의 결과를 다양한 포맷의 파일에 저장할 수 있다.

Table Explorer

테이블 탐색자는 테이블 데이터의 입력 상황을 표시한다. 입력한 시간에 의해 데이터 입력 수의 합계를 구하고, 이를 그래프 형태로 표시한다. 그래프에서 일정 영역을 drag로 선택하면 그 기간 동안의 입력 레코드의 수를 볼 수 있다.

화면의 오른쪽 상단 부분에 Zoom mode(시간 범위 조정을 위해서 사용)의 ON/OFF버튼이 있다.

Bookmark Queries

SQL 화면에서 실행 후 북마크한 질의문의 관리 화면이다. 질의문 리스트를 클릭하면, 질의문의 상세 내역을 확인할 수 있다. 또한 "to SQL" 버튼을 이용하여 SQL 화면으로 전환할 수 있다.

Collection

Data Collection

마크베이스 서버에 등록된 collector manager와 collecotor 리스트가 트리 형태로 표시된다.

템플릿 파일(.tpl), 정규 표현식 파일(.rgx) 과 전처리 스크립트 파일(.py)은 MACHBASE_HOME 경로를 기준으로 다음의 경로에 위치해야 한다.

- 템플릿 파일(.tpl) : $MACHBASE_HOME/collector
- 정규 표현식 파일(.tpl) : $MACHBASE_HOME/collector/regex
- 전처리 스크립트 파일(.py) : $MACHBASE_HOME/collector/preprocess

화면에 표시된 테이블의 우측에 눈(eye)모양의 아이콘 오른쪽에 레코드의 수가 표시된다. 눈 모양의 아이콘 위로 마우스 커서를 옮기면, "View Table" 윈도우가 표시된다. 눈 모양의 아이콘을 클릭하면, "Table Explorer" 화면으로 전환되어 테이블 내용을 확인 할 수 있다. 테이블 하단에는 컬렉터와 컬렉터 관리자의 실행 상태와, 데이터 수집 속도를 볼 수 있다. 각 컬렉터 이름의 오른쪽에 컬렉터에 실행, 중지, 삭제 명령을 실행하는 아이콘이 있다.

"add Manager" 버튼을 누르면 별도의 윈도우에 컬렉터 관리자를 실행하는 윈도우가 열린다. 이 윈도우에서 "Create Manager" 버튼으로 신규 컬렉터 관리자를 등록할 수 있고, 오른쪽에 RENAME, DROP, LIST 기능을 실행할 수 있는 버튼이 있다. LIST 버튼은 그 컬렉터 관리자가 관리하고 있는 컬렉터의 목록을 표시한다.

특정 컬렉터 관리자가 관리하는 컬렉터를 "add Manager" 버튼을 이용하여 생성할 수 있다. 컬렉터 관리자를 선택한 다음, 생성할 컬렉터의 이름을 입력한다. 컬렉터의 이름은 컬렉터 관리자에 대해서 유일한 값이어야 한다.

"Template" 버튼을 클릭하면 새로운 윈도우가 표시된다. "New" 버튼을 이용하여 새로운 템플릿 파일을 생성할 수 있다. 각 템플릿 파일 명의 오른쪽에 템플릿 파일을 수정하기 위한 버튼이 있다. 이 기능을 이용하여 기존에 만들어진 템플릿 파일을 이용하여 새로운 템플릿 파일을 생성할 수 있다. 템플릿 파일을 생성, 변경하고 저장하기 전에 DB_ADDR 및 DB_PORT필드가 적절한 값인지 확인해야 한다. 화면에 표시된 각 구성요소에 마우스 커서를 가져가면, 구성요소에 대한 설명이 표시된다.

Preprocess

이 기능은 전처리 스크립트 파일을 관리하기 위해 사용된다. 컬렉터 관리자를 화면의 우측 구석에서 선택하면, 전처리 스크립트 파일의 목록을 표시한다. "Reload" 버튼을 누르면 파일 목록을 다시 읽어서 표시한다.

스크립트 파일명을 클릭하여 파일을 편집할 수 있고, "New" 버튼을 이용하여 새로운 스크립트 파일을 생성할 수 있다. "Save" 버튼을 이용하여 변경하거나 생성한 파일을 기록할 수 있다.

Regular Expression

템플릿 파일에서 사용되는 정규 표현식 파일을 관리하는 기능이다. 우측에서 컬렉터 관리자를 선택하면, 정규 표현식 파일의 목록을 표시한다. "Reload" 버튼을 이용하여 목록을 다시 읽어 올 수 있다.

정규 표현식 파일 목록에서 파일을 클릭하면 파일 내용이 표시되고 그 파일을 변경할 수 있다. "New" 버튼을 이용하여 신규 정규표현식 파일을 생성할 수 있다.

정규 표현식 파일은 다음의 구성 요소를 포함해야 한다.

- REGEX : 데이터를 분석하기 위한 정규 표현식이다.
- START_REGEX : 분석 대상 데이터의 시작 지점을 지정하는 정규 표현식이다. 이 데이터는 REGEX 분석데이터에 포함된다.
- END_REGEX : 분석 데이터의 끝을 나타내는 정규 표현식이다. 이 이후 값은 분석에 포함되지 않는다. 이 값이 지정되지 않으면 컬렉터는 동작하지 않는다.

아래쪽 화면에 분석 데이터가 입력될테이블의 컬럼 목록이 표시된다. Regex No는 machregex의 결과로 생성되는 일련번호이며 그 이후에 컬럼 이름, 타입, 사이즈가 표시된다. Test 탭을 눌러서 샘플 테스트를 실행해 볼 수 있다. 테스트 결과 성공했다면 Columns 탭에서 원하는 컬럼을 선택한다. 테스트를 실행하면 창에 테스트 실행이 성공했는지를 표시한다. 성공 이후에, 원하는 컬럼 이름, 타입, 사이즈를 설정하고 "Apply" 버튼을 누르면 컬럼 리스트를 다시 표시하게 된다.

DB Admin

Tables

테이블과 테이블 스페이스를 관리하는 화면이다. 테이블 스키마와 테이블 인덱스를 확인할 수 있다.

Running Queries

현재 실행중인 질의문들을 표시한다. 실행중인 질의를 중단시킬 수 있다.

System Monitoring

MWA서버가 실행되고 있는 시스템에 관련한 정보를 표시한다. 정보를 얻을 수 없는 경우에는 공백으로 표시된다. MWA를 현재 실행하는 장비가 아닌 장비를 모니터링하기 위해서는 원하는 장비에 MWA를 실행하고 그 장비의 "Web URL"을 등록해야 한다.

Preferences

Servers

마크베이스 서버를 등록하기 위한 메뉴이다. MWA 사용자 중 ADMIN 권한을 갖는 사용자만 접속할 수 있다.

서버 이름	서버의 이름
Host and Port	Machbase DB 서버의 IP 주소와 포트 번호를 설정한다. MWA 서버와 같은 장비에서 마크베이스 DB를 실행중인 경우, 127.0.0.1 또는 localhost를 사용할 수 있다.
Web URL	이 URL을 이용하여 접속할 수 있다.
UserID and Password	Machbase DB 서버의 사용자명과 암호

최소 하나의 서버는 등록해야 한다. MWA 서버가 실행중인 장비에 Machbase DB를 실행중인 경우 "Command" 버튼을 이용하여 서버 시작과 종료, DB 생성 등의 명령을 수행할 수 있다. 다른 장비에 실행중인 Machbase DB 서버에 대해선 "Command" 버튼으로 실행하는 명령을 제외한 기능들을 이용할 수 있다. 현재 실행중인 MWA 서버가 아닌 다른 장비에도 MWA 서버가 DB 서버와 같이 실행중이면 "Command" 버튼의 명령들을 실행할 수 있다.

Groups

MWA사용자 그룹을 관리한다. ADMIN 권한을 갖는 MWA 사용자만 접근할 수 있다. Chart, Grid, Dashboard, Bookmark query를 그룹마다 등록하여 관리할 수 있다.

Users

사용자 계정을 관리한다. 이 사용자 계정은 Machbase DB사용자가 아니라 MWA 사용자를 의미한다. 대시보드의 경우, 이전에 생성한 대시보드를 선택하는 화면이 표시된다.

ADMIN 사용자는 각 사용자와 서버를 등록하고 관리할 수 있다.

Available Groups: set ADMIN/USER to Authority.
Available Servers : set default server.

USER 권한의 사용자는 PASSWORD만 갱신이 가능하다.

태그 분석기 (Tag Analyzer)

Tag Analyzer는 Tag Table의 ROLLUP 기능을 활용하여 데이터들을 Chart로 조회/분석할 수 있는 기능을 제공한다.

설정

MWA에서 호출이 되므로 MWA가 사용가능해야 하며, 사용되는 Table은 TAG Table로 지정되어 있다.

$MACHBASE_HOME/webadmin/flask/MWA.conf 파일에 정의된 설정 중에서 Tag Analyzer에서 사용되는 항목은 아래와 같다.

항목	설명	기본값
USE_TAG_ANALYZER_AUTO_DRILLDOWN	Rollup 데이터가 발견되지 않으면 아래 단계의 Rollup에서 데이터를 찾는다.	Y
MAX_TAG_COUNT	하나의 Chart에서 사용할 수 있는 최대 Tag의 수	12

시작

MWA를 구동하면 보이는 상단의 메뉴중 제일 좌측의 "Tag View"를 클릭하면 새로운 탭(창)에 Tag Analyzer가 실행된다.

MWA에 로그인되어 있는 상태라면 **MWA URL**/tagview(예: 127.0.0.1:5001/tagview)를 사용하여 Tag Analyzer에 직접 접속할 수 있으며, 저장된 Dashboard가 있는 경우 **MWA URL**/tagview?id=*dashboard_id*(예: 127.0.0.1:5001/tagview?id=board1)를 입력하면 해당 Dashboard를 바로 조회할 수 있다.

사용법

Tag Analyzer는 여러 개의 Chart로 구성된 Dashboard 형태로 구성되며 Dashboard의 각 열은 하나의 Chart로 구성된다.

개요

좌측 상단에 Tag Analyzer를 관리하는 메뉴가 표시된다.

Dashboard 선택

저장된 Dashboard중에서 현재 화면에 표시되는 Dashboard를 선택한다.

New Dashboard

새로운 Dashboard를 생성한다.

Set

Preference

Tag Analyzer의 환경을 설정한다.

항목	설명	기본값
UI Thema	Dashboard의 배경색을 설정한다. (machIoTchartBlack / machIoTchartWhite)	machIoTchartBlack
Home Dashboard	Dashboard 지정없이 Tag Analyzer를 호출했을때 선택되는 Dashboard를 선택한다.	없음.
Query Timeout	Query 호출시 설정한 시간동안 응답이 없으면 Timeout 오류를 발생시킨다.	20

Manage Dashboard

저장된 Dashboard를 관리한다.

버튼	설명	비고
	Dashboard를 새 탭(창)에서 조회한다.	
	Dashboard의 Title을 수정한다.	
	Dashboard를 삭제한다.	

Request Rollup

ROLLUP 강제 업데이트 명령인 "**EXEC ROLLUP_FORCE**" 명령을 실행시킨다. 약 6초간의 시간이 소요된다.

Logout

MWA에서 logout 한다.

Dashboard

Chart 생성

Dashboard의 각 열은 하나의 Chart로 구성되며 제일 하단의 패널에 표시된 + 버튼을 눌러서 Chart를 생성할 수 있다.

사용할 Tag들과 집계방법을 선택하고 Chart 유형을 선택하면 Chart가 생성된다. 선택된 Tag 목록에 있는 Tag를 Click하면 선택이 취소된다.

설정

Dashboard에 관련된 버튼은 우측 상단에서 찾을 수 있다.

버튼	설명	비고
Save Dashboard	현재의 Dashboard를 저장한다. Board ID는 조회시 URL에 사용된다. Board ID를 변경하면, "Save as Copy"가 체크된 경우에는 새로운 Dashboard가 생성되며 그렇지 않으면 Board ID가 변경된다.	Board ID에는 영문, 숫자 및 underscore(_)만 사용 가능

Time Range	조회할 시간범위와 Refresh 주기를 설정한다.
	
	• 시간범위 : 시간 선택(입력) 또는 now를 이용한 상대 시간 지정이 가능하다.
	(예: now, now-5d, now-3M)
	  • 단위 : now를 기준으로 y(year), M(month), w(week : Mon - Sun), d(day), h(hour), m(minute), s(second) 사용 가능하다.
	  • 옵션 : "/"를 사용하여 특정기간을 쉽게 지정할 수 있다.
	예 : now = 2021-03-04
	Today
	설정 : from = now/d, to = now/d
	결과 : 2021-03-04 00:00:00 ~ 2021-03-04 23:59:59
	Previous year
	설정 : from = now-1y/y, to = now-1y/y
	결과 : 2020-01-01 00:00:00 ~ 2020-12-31 23:59:59,
	Last ~ This week (2weeks)
	설정 : from = now-1w/w, to = now/w
	결과 : 2021-02-22 00:00:00 ~ 2021-03–07 23:59:59
	• Refresh 주기 : h(hour), m(minute), s(second) 단위로 입력이 가능하며 입력된 주기로 Dashboard가 다시 그려진다.
	
	Chart 설정에서 별도로 시간범위나 Refresh 주기를 설정하면 Dashboard의 설정과 무관하게 자체 설정 값을 기준으로 동작한다.
		YYYY-MM-DD HH24:MI:SS

Refresh all	모든 Chart를 Refresh 한다.	
Share Dashboard	현재의 Dashboard를 새로운 탭(창)에서 조회한다.(조회용 URL로 open)	Save된 경우에만 가능

Chart

선택된 Tag들과 집계방법으로 Chart가 그려진다. 마우스를 Chart에서 움직이면 시간 및 Series별 값을 확인 할 수 있다.

아래 부분에 표시된 범례(legend)를 클릭하면 선택한 Tag의 그래프만 볼 수 있으며 같은 Tag를 다시 클릭하면 다시 모든 Series를 볼 수 있다.

또한 범례 중 Chart 색 부분을 클릭하면 선택한 Tag의 Alias를 설정할 수 있다.

Zoom

Chart에서 확대해서 보고 싶은 부분을 마우스로 Drag해서 선택하면 해당 되는 부분을 자세히 볼 수 있다. Chart의 아래 부분에 Time Range에 따른 전체 그래프가 나타나고 그 중에서 현재 Chart에 그려지는 부분이 표시된다. Chart에 표시되는 부분이 점점 확대가 되면 초단위 ROLLUP 데이터까지 표시가 되다가 선택된 부분이 아주 작아지면 원시자료(raw data)를 표시하는 Chart가 그려진다.

Raw data chart의 경우 Chart Header에 표시되는 Time Range의 좌우에 시간을 이동할 수 있는 버튼이 표시가 되어 Chart를 이동하면서 조회할 수 있다. Raw data chart는 선택 범위가 아주 작으므로 viewport를 사용하여 이동할 수 없다.

Zoom모드에 있는 경우 Auto Refresh는 동작하지 않는다.

- viewport : Chart 아래 부분에 전체 그래프가 표시되는 부분. Chart의 일부를 확대해서 볼 때만 표시된다.(Zoom 모드로 시작하는 것이 기본 값이다.) viewport 좌측에 있는 [x]를 누르면 Zoom 모드를 종료하고 viewport를 닫는다.
- window : viewport 중에서 현재 Chart에 그려지는 부분을 표시한다. 마우스로 Drag하여 이동할 수 있으며, 좌우측 끝부분을 Drag하면 크기를 변경하여 Chart를 확대 축소할 수 있다.
- viewport buttons : viewport를 이동하거나 조작하는데 사용되는 button들이다.

버튼	설명	비고
Time Range	시간 및 기간을 사용하여 Time Range를 설정한다.	From & 6M
Undo	Zoom 조작을 취소한다. 마우스 우측 버튼도 같은 기능을 수행한다.	현재는 **직전의 Zoom만 가능**
Reset	Time Range를 전체 데이터의 minimum ~ maximum 으로 설정한다.	
Center	현재의 window가 viewport의 중앙에 위치하도록 Time Range를 조정한다.	

버튼	설명	비고
Resize	현재의 window가 20%의 크기로 중앙에 위치하도록 Time Range를 조정한다.	
- , +	window를 좌측(우측)으로 Zoom out / Zoom in 한다.	Zoom out / Zoom in
<< , < , > , >>	window를 좌우로 100%(50%) 이동한다.	

Chart buttons

Chart의 우측 상단에 표시되는 버튼으로 아래와 같다.

버튼	설명	비고
Preview ( )	해당 Chart만 새로운 탭(창)에서 조회한다.	
Edit options ( )	Chart의 property를 수정한다. Chart의 모양을 변경하거나 Tag를 추가/수정할 수 있다.	수정후 를 눌러야 적용
Refresh ( )	Chart를 다시 그린다.	
Delete ( )	Chart를 Dashboard에서 제거한다.	

Chart properties

Property는 하단의 패널에서 수정하고 를 눌러서 수정된 것을 확인한 다음 상단의 를 누르면 적용이 된다. 각 탭별 Property는 아래와 같다.

- General

Property	설명	기본값	비고
Chart Title	Chart의 제목	Chart Title	
Width	Chart의 너비(0 : 전체 크기)	0	현재는 0외에는 의미없음
Height	Chart의 높이(0: 전체 크기)	300	현재 0은 300으로 계산
Action On Click	Chart의 node를 클릭했을 때의 동작 - No Action - Show Raw data chart - Show Raw data table	No Action	Show Raw data chart는 클릭한 node의 시간 범위에 해당하는 Chart를 의미한다.
Zoom	Chart를 Drag 했을때 Zoom을 하는지 여부	Y	
Drill down	Zoom을 할때 Drill down을 하는지 여부	Y	
Start with Zoom	Chart를 그릴 때 Zoom모드에서 시작한다.	Y	Zoom 상태에서는 Auto Refresh가 동작하지 않으므로, Auto Refresh를 사용하려면 이 기능은 꺼두어야 한다.
Normalize	Chart에 Normalize 사용 여부	N	
Raw data time range	Raw Data Chart의 기준 시간 범위	5000	값은 millisecond(ms) 단위로 작성해야 한다.

- Data
  사용할 Tag와 집계방법 등을 변경한다.

Property	설명	기본값	비고
Calc. mode	Tag Data 집계 방법	Average	
Tag Names	사용할 Tag의 이름		여러 개의 Tag를 ","로 연결해서 입력하면 하나의 Series로 그려진다. 시간대에 해당하는 Tag가 여러 개라면 Alphabet 순으로 빠른 Tag의 값이 선택된다.
Alias	Chart에서 Tag Name 대신 사용하는 Alias		빈칸이면 기존의 Tag Name을 사용한다.
Weight	Normalize를 사용 시 Tag 별 가중치	1	0~1사이의 값을 입력해야 한다.

- Axes

Property	설명	기본값	비고
Interval	X축의 시간 간격을 설정한다. 지정하지 않으면 시간범위 및 화면의 크기에 따라서 자동으로 계산한다.		h, m, s 지원
Show tick line (X-axis)	X축의 눈금에 대응하는 Grid를 보여준다.(세로선)	Y	

Property	설명	기본값	비고
Pixels between tick marks	X축 눈금 간의 pixel을 설정한다.	3	
Start at zero (Y-axis)	Y축 눈금이 0부터 시작한다.	N	
Show tick line (Y-axis)	Y축의 눈금에 대응하는 Grid를 보여준다.(가로선)	Y	
Custom scale	Y축의 눈금의 범위를 설정한다. 지정하지 않으면 최소값과 최대값을 사용하여 자동으로 지정된다.		
Custom scale for drill down chart	Drill down Chart의 Y축의 눈금의 범위를 설정한다. 지정하지 않으면 최소값과 최대값을 사용하여 자동으로 지정된다.		
Set additional Y-axis.	Y축을 하나 추가해서 사용한다.	N	
Position of Y-axis	추가된 Y축이 그려지는 위치를 설정한다.	Right	
Select tags for Y-axis 2	추가된 Y축을 사용할 Data를 선택한다. 'Data' 탭에 등록된 항목에서 선택한다.		

- Display

Property	설명	기본값	비고
Show data points	Chart의 node에 point를 표시한다.	N	
Point radius	위의 경우 표시되는 point의 반지름(pixel)	3	
Legend	범례(Legend)를 표시한다.	Y	
Opacity of fill area	그래프의 fill 영역의 투명도를 설정한다.	0.15	0:투명, 1:불투명
Line thickness	line chart의 선두께를 설정한다.	1.5	0:line없음
Border color	Chart의 테두리 색상을 설정한다. 입력하지 않으면 Background Color로 설정된다.		변경시에는 #을 붙여서 입력 공백으로 돌릴때는 "none"을 입력

- Time Range
  이 Chart에만 적용되는 Time Range와 Refresh 주기를 입력한다. 이 값들이 설정되면 Dashboard의 설정과는 무관하게 이 값으로만 Chart가 동작한다.

machcoordinatoradmin

machcoordinatoradmin

Coordinator 에서 클러스터 전체에 대한 관리 도구이다.

Cluster Edition 패키지에만 존재한다.

옵션 및 기능

machcoordinatoradmin의 옵션은 아래와 같다. 앞의 설치 절에서 설명한 기능은 생략한다.

```
mach@localhost:~$ machcoordinatoradmin -h
```

옵션	설명
-u, --startup	Coordinator 프로세스를 구동
-s, --shutdown	Coordinator 프로세스를 종료
-k, --kill	Coordinator 프로세스를 중단
-c, --createdb	Coordinator의 메타를 생성
-d, --destroydb	Coordinator의 메타를 삭제하고, $MACHBASE_COORDINATOR_HOME/package에 있는 패키지 파일들을 삭제
-e, --check	Coordinator 프로세스가 작동 중인지 확인
-i, --silence	출력 없이 실행
--configuration[=name]	configuration 설정에서의 키와 값 출력(특정 키만 출력 가능)
--activate	Cluster status를 Service로 전환
--deactivate	Cluster status를 Deactivate로 전환
--list-package[=package]	등록한 Package들의 정보를 나열(특정 Package만 출력 가능)
--add-package=package	Package를 추가
--remove-package=package	Package를 삭제
--list-node[=node]	Node들의 정보를 나열(특정 Node만 출력 가능)
--add-node=node	Node를 추가
--remove-node=node	Node를 삭제
--upgrade-node=node	Node를 업그레이드
--startup-node=node	Node를 구동
--shutdown-node=node	Node를 종료
--kill-node=node	Node를 중단
--cluster-status	Cluster의 각 Node 상태를 출력
--cluster-status-full	Cluster의 각 Node 상태를 상세하게 출력
--cluster-node	Cluster의 정보를 출력
--set-group-state=[normal | readonly]	특정 warehouse 그룹의 상태를 변경
--get-host-resource	각 Node가 위치한 Host 자원 정보를 출력

옵션	설명
--host-resource-enable	각 노드의 Host 자원 정보 수집을 시작
--host-resource-disable	각 노드의 Host 자원 정보 수집을 멈춤

부가 옵션	설명	필수 옵션
--file-name=filename	파일 이름	--add-package
--port-no=portno	포트 번호	--add-node
--deployer=node	Deployer의 노드 이름	--add-node
--package-name=packagename	설치 원본이 될 Package 이름	--add-package
--home-path=path	Deployer 서버 기준, 현재 Node의 설치 경로	--add-node
--node-type=[broker | warehouse]	설치할 노드의 타입(broker / warehouse 중 선택)	--add-node
--group=groupname	설치할 노드의 그룹 이름	--add-node
--replication=host:port	Replication을 주고 받을 host:port	--add-node
--no-replicate	설치할 노드의 Replication을 사용하지 않음	--add-node
--primary=host:port	Secondary Coordinator 설치 시 Primary Coordinator의 노드 이름을 지정	-u, --startup
--host=host	Host 자원 정보를 출력할 특정 Host 지정	--get-host-resource
--metric=[cpu|memory|disk|network]	Host 자원 정보를 출력할 특정 Metric 지정	--get-host-resource

동작 여부 확인

Example:

```
mach@localhost:~$ machcoordinatoradmin -e
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Machbase Coordinator is running with pid(29245)!
```

메타 생성 / 삭제

Example:

```
mach@localhost:~$ machcoordinatoradmin -c
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Coordinator metadata created successfully.

mach@localhost:~$ machcoordinatoradmin -d
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Coordinator metadata destroyed successfully.
```

Configuration 설정 출력

Syntax:

```
machcoordinatoradmin --configuration[=name]
```

Example:

```
mach@localhost:~$ machcoordinatoradmin --configuration
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Name  : CLUSTER
Value : 3

Name  : DECISION
Value : ON

Name  : HOST-RESOURCE
Value : OFF

mach@localhost:~$ machcoordinatoradmin --configuration=decision
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
              Name : DECISION
             Value : ON
            Format : text/plain
```

Cluster Status 변경

Example:

```
mach@localhost:~$ machcoordinatoradmin --activate
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
              Name : CLUSTER
             Value : 3
            Format : text/plain


mach@localhost:~$ machcoordinatoradmin --deactivate
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
              Name : CLUSTER
             Value : 0
            Format : text/plain
```

패키지 정보 나열

Syntax:

```
machcoordinatoradmin --list-package[=package]
```

Example:

```
mach@localhost:~$ machcoordinatoradmin --list-package
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Package Name : machbase
File Name    : machbase-cluster-6bab497c9.develop-LINUX-X86-64-release-lightweight.tgz
File Size    : 64630670 bytes

Package Name : machbase2
File Name    : machbase-cluster-e3c0717.develop-LINUX-X86-64-release-lightweight.tgz
File Size    : 64677030 bytes


mach@localhost:~$ machcoordinatoradmin --list-package=machbase
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Package Name : machbase
File Name    : machbase-cluster-6bab497c9.develop-LINUX-X86-64-release-lightweight.tgz
File Size    : 64630670 bytes
```

노드 정보 나열

Syntax:

```
machcoordinatoradmin --list-node[=node]
```

Example:

```
mach@localhost:~$  machcoordinatoradmin --list-node
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Node Name             : 192.168.0.32:5101
Node Type             : coordinator
HTTP Admin Port       : 5102
Group Name            : Coordinator
Desired State         : primary
Actual State          : primary
Coordinator Host      : 192.168.0.32:5101
Last Response Time    : 497590
Last Modify Time      : 421020408
Last Response Elapsed : 1006148

Node Name             : 192.168.0.32:5201
Node Type             : deployer
Group Name            : Deployer
```

```
Desired State         : normal
Actual State          : normal
Coordinator Host      : 192.168.0.32:5101
Last Response Time    : 497594
Last Modify Time      : 404915419
Last Response Elapsed : 1006128

Node Name             : 192.168.0.32:5301
Node Type             : broker
Port Number           : 5757
Deployer              : 192.168.0.32:5201
Package Name          : machbase
Home Path             : /home/machbase/broker1
Group Name            : Broker
Desired State         : leader
Actual State          : leader
Coordinator Host      : 192.168.0.32:5101
Last Response Time    : 497544
Last Modify Time      : 353606480
Last Response Elapsed : 1006157

Node Name             : 192.168.0.32:5401
Node Type             : warehouse
Port Number           : 5400
Deployer              : 192.168.0.32:5201
Package Name          : machbase
Home Path             : /home/machbase/warehouse_a1
Group Name            : Group1
Desired State         : normal
Actual State          : normal
Coordinator Host      : 192.168.0.32:5101
Last Response Time    : 497556
Last Modify Time      : 332480933
Last Response Elapsed : 1006160

mach@localhost:~$  machcoordinatoradmin --list-node=192.168.0.32:5401
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Node Name             : 192.168.0.32:5401
Node Type             : warehouse
Port Number           : 5400
Deployer              : 192.168.0.32:5201
Package Name          : machbase
Home Path             : /home/cumulus/warehouse_a1
Group Name            : Group1
Desired State         : normal
Actual State          : normal
Coordinator Host      : 192.168.0.32:5101
Last Response Time    : 648879
Last Modify Time      : 419153148
Last Response Elapsed : 1005962
```

Cluster의 Node 상태 출력

Example:

```
mach@localhost:~$ machcoordinatoradmin --cluster-status
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
+-------------+------------------+------------------+------------------+--------------+
```

```
|  Node Type  |     Node Name     |    Group Name     |    Group State    |     State    |
+-------------+-------------------+-------------------+-------------------+--------------+
| coordinator | 192.168.0.32:5101 | Coordinator       | normal            | primary      |
| deployer    | 192.168.0.32:5201 | Deployer          | normal            | normal       |
| broker      | 192.168.0.32:5301 | Broker            | normal            | leader       |
| warehouse   | 192.168.0.32:5401 | Group1            | normal            | normal       |
+-------------+-------------------+-------------------+-------------------+--------------+

mach@localhost:~$ machcoordinatoradmin --cluster-status-full
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
+-------------+-------------------+-------------------+-------------------+-------------------------------+--------
|  Node Type  |     Node Name     |    Group Name     |    Group State    |    Desired & Actual State     |  RP Sta
+-------------+-------------------+-------------------+-------------------+-------------------------------+--------
| coordinator | 192.168.0.32:5101 | Coordinator       | normal            | primary       | primary       | -------
| deployer    | 192.168.0.32:5201 | Deployer          | normal            | normal        | normal        | -------
| broker      | 192.168.0.32:5301 | Broker            | normal            | leader        | leader        | -------
| warehouse   | 192.168.0.32:5401 | Group1            | normal            | normal        | normal        | -------
+-------------+-------------------+-------------------+-------------------+-------------------------------+--------
```

Cluster 정보 출력

Example:

```
mach@localhost:~$ machcoordinatoradmin --cluster-node
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Token Pid      : 29245
Token Time     : 1553153902646178
Modify Time    : 1553154010296715
Modify Count   : 8
Cluster Status : Service
Broker         : 192.168.0.32:5301
Warehouse      : 192.168.0.32:5401
```

Group State 변경

Syntax:

```
machcoordinatoradmin --set-group-state=[ normal | readonly ] --group=group
```

Example:

```
mach@localhost:~$ machcoordinatoradmin --set-group-state=readonly --group=Group1
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Group Name: Group1
Flag      : 1

mach@localhost:~$ machcoordinatoradmin --cluster-status
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
```

```
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------------
+-------------+------------------+------------------+-------------------+--------------+
|  Node Type  |     Node Name    |    Group Name    |    Group State    |     State    |
+-------------+------------------+------------------+-------------------+--------------+
| coordinator | 192.168.0.32:5101 | Coordinator      | normal            | primary      |
| deployer    | 192.168.0.32:5201 | Deployer         | normal            | normal       |
| broker      | 192.168.0.32:5301 | Broker           | normal            | leader       |
| warehouse   | 192.168.0.32:5401 | Group1           | readonly          | normal       |
+-------------+------------------+------------------+-------------------+--------------+
```

Host Resource 출력

Syntax:

```
machcoordinatoradmin --host-resource-enable [--metric=metric] [host=host]
```

Example:

```
mach@localhost:~$ machcoordinatoradmin --host-resource-enable
--------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------------
              Name : HOST-RESOURCE
             Value : ON
            Format : text/plain

mach@localhost:~$ machcoordinatoradmin --get-host-resource
--------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------------
Host Name : 192.168.0.32
   CPU Info :
      Model Name          : Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz
      Number of CPUs      : 8
      Number of CPU Cores : 4
      CPU Utilization     : 14.0%
      CPU IOWait Ratio    : 0.0%
   Memory Info :
      Physical Memory Utilization : 99.1%
      Virtual Memory Utilization  : 98.6%
   Network Info :
      Receive Bytes(per second)    : 42809
      Receive Packets(per second)  : 337
      Transmit Bytes(per second)   : 42885
      Transmit Packets(per second) : 332
   Disk Info :
      /dev/sda1 : 87.4%
         |-> 192.168.0.32:5101   /home/cumulus/coordinator1
         |-> 192.168.0.32:5301   /home/cumulus/broker1
         |-> 192.168.0.32:5401   /home/cumulus/warehouse_a1
Host Name : 192.168.0.33
   CPU Info :
      Model Name          : Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz
      Number of CPUs      : 8
      Number of CPU Cores : 4
      CPU Utilization     : 2.0%
      CPU IOWait Ratio    : 0.0%
```

```
   Memory Info :
      Physical Memory Utilization : 46.9%
      Virtual Memory Utilization  : 22.8%
   Network Info :
      Receive Bytes(per second)    : 12336
      Receive Packets(per second)  : 103
      Transmit Bytes(per second)   : 13500
      Transmit Packets(per second) : 103
   Disk Info :
      /dev/sda1 : 64.2%
         |-> 192.168.0.33:5101   /home/cumulus/coordinator2
         |-> 192.168.0.33:5401   /home/cumulus/warehouse_a2

mach@localhost:~$ machcoordinatoradmin --get-host-resource --metric=cpu
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Host Name : 192.168.0.32
   CPU Info :
      Model Name          : Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz
      Number of CPUs      : 8
      Number of CPU Cores : 4
      CPU Utilization     : 13.9%
      CPU IOWait Ratio    : 0.0%
Host Name : 192.168.0.33
   CPU Info :
      Model Name          : Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz
      Number of CPUs      : 8
      Number of CPU Cores : 4
      CPU Utilization     : 1.9%
      CPU IOWait Ratio    : 0.0%

mach@localhost:~$ machcoordinatoradmin --get-host-resource --host=192.168.0.33
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
Host Name : 192.168.0.33
   CPU Info :
      Model Name          : Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz
      Number of CPUs      : 8
      Number of CPU Cores : 4
      CPU Utilization     : 2.0%
      CPU IOWait Ratio    : 0.0%
   Memory Info :
      Physical Memory Utilization : 46.9%
      Virtual Memory Utilization  : 22.8%
   Network Info :
      Receive Bytes(per second)    : 12588
      Receive Packets(per second)  : 106
      Transmit Bytes(per second)   : 13330
      Transmit Packets(per second) : 100
   Disk Info :
      /dev/sda1 : 64.2%
         |-> 192.168.0.33:5101   /home/cumulus/coordinator2
         |-> 192.168.0.33:5401   /home/cumulus/warehouse_a2

mach@localhost:~$ machcoordinatoradmin --host-resource-disable
------------------------------------------------------------------------
     Machbase Coordinator Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------------
              Name : HOST-RESOURCE
```

```
 Value : OFF
Format : text/plain
```

machdeployeradmin

machdeployeradmin

Deployer의 상태를 확인하거나, Deployer의 구동/종료/중단 명령을 직접 내릴 수 있다.

보통은 machcoordinatoradmin 을 통해서 내리는 편이 가장 빠르지만, 불가능한 경우에는 아래와 같이 수행해야 한다.

Cluster Edition 패키지에만 존재한다.

옵션 및 기능

machdeployeradmin의 옵션은 아래와 같다. 앞의 설치 절에서 설명한 기능은 생략한다.

```
mach@localhost:~$ machdeployeradmin -h
```

옵션	설명
-u, --startup	Deployer 프로세스 구동
-s, --shutdown	Deployer 프로세스 종료
-k, --kill	Deployer 프로세스 중단
-c, --createdb	Deployer 메타 생성
-d, --destroydb	Deployer 메타 삭제
-i, --silence	출력 없이 실행
-e, --check	Deployer 프로세스가 작동중인지 확인

동작 여부 확인

Example:

```
mach@localhost:~$ machdeployeradmin -e
--------------------------------------------------------------------------------
     Machbase Deployer Administration Tool
     Release Version - e3c0717.develop
     Copyright 2014, MACHBASE Corp. or its subsidiaries
     All Rights Reserved
--------------------------------------------------------------------------------
Machbase Deployer is running with pid(29373)!
```

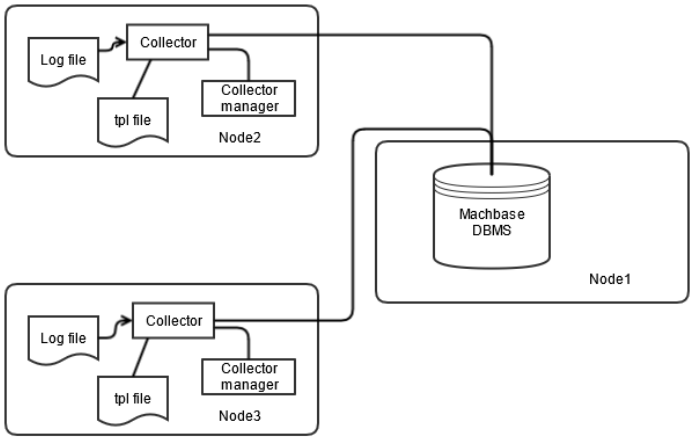
컬렉터 (Collector)

마크베이스 Collector 는 로그 데이터를 추출하여 변환 후 마크베이스 데이터베이스에 실시간으로 입력하는 도구다.

마크베이스 서버와 분리된 장비에 설치하여 실시간으로 로그 데이터를 수집해서 네트워크를 통해 입력할 수 있는 것이 마크베이스 Collector 다. 이는, 마크베이스 서버와 분리된 프로세스로 동작하며 동시에 여러 Collector 를 실행할 수 있다. 각 Collector 프로세스가 데이터 소스 하나를 처리한다.

목차

개념



위 그림은 Node-2 와 Node-3 의 Collector 가 데이터를 수집하여 데이터베이스 서버가 설치된 Node-1에 입력하는 것을 나타낸다.

- Node-2와 Node-3에서 Collector 는 **특정 로그 파일에 대하여 별개의 프로세스로 실행**되어 데이터를 전송하는 것을 볼 수 있다.
- 각 Collector 프로세스는 **로그 데이터에 대한 상세 정보를 주어진 tpl 파일을 이용**하여 얻는 것을 알 수 있다.
- **Collector manager** 는 각 Node에 설치되어 해당 노드에서 동작 중인 Collector 프로세스를 관리하고 모니터링한다.

특징

아래에 마크베이스 Collector 의 주요 기능을 설명한다.

일관성 있는 인터페이스

마크베이스는 Collector 를 실행하기 위해 SQL기반의 명령어 외에 추가 프로그램을 요구하지 않는다. 간단히 다음 명령어로 Collector 를 관리 및 모니터링 할 수 있다.

```
1 CREATE Collector MANAGER LOCALHOST AT '127.0.0.1:9999';
2 CREATE Collector LOCALHOST.MYADP FROM 'syslog.tpl';
3 ALTER Collector LOCALHOST.MYADP START;
```

데이터 수집 성능 향상

마크베이스 Collector 는 로그 데이터 타입 마다 별개의 Collector 가 데이터를 수집하도록 설계되었다. 그래서 각 프로세스가 고속으로 각각의 로그 파일을 처리할 수 있다.

별개의 프로세스들이 각각 로그 데이터를 처리하므로, 다른 로그 파일 처리에 영향을 받지 않는다.

Collector 는 각 로그 타입에 대해 최적화된 코드로 실행되고, 자원 사용량을 최소화한 전용 프로토콜로 데이터를 입력하므로 최고의 성능을 얻을 수 있다.

수집 방법

Collector 를 이용하여 다양한 방법으로 로그 데이터를 수집할 수 있다. 데이터 수집 방법은 tpl 파일을 수정하여 설정할 수 있다. 지원되는 수집 방법은 아래와 같다.

방식 이름	설명
FILE	로컬 호스트의 파일을 수집한다.
SFTP	리모트 호스트의 파일을 수집한다.
SOCKET	포트로 들어오는 데이터를 수집한다.
ODBC	타 데이터 베이스로부터 데이터를 수집한다.

로그 데이터 종류

마크베이스 Collector 는 다양한 타입의 로그 데이터를 위해서 정규 표현식을 지원한다.
사용자는 기존에 제공되는 정규 표현식을 간단히 수정하여 다양한 로그 파일을 분석할 수 있다. 현재 지원하는 로그 타입은 다음과 같다.

정규 표현식 파일 이름	지원하는 타입	데이터의 기본 위치 (수정 가능)
machbase.rgx	Machbase의 트레이스 로그	$MACHBASE_HOME/trc/machbase.trc
apache_access.rgx	Apache web server의 Access 파일	/var/log/apache2/access.log
apache_error.rgx	Apache web server의 Error 파일	/var/log/apache2/access.log
syslog.rgx	sysglog 파일	/var/log/syslog
custom.rgx	유저 정의 타입	유저 정의 파일

사용자 정의 로그를 쉽게 지원

마크베이스 Collector 는 정규 표현식으로 표현될 수 있는 다양한 종류의 로그파일을 처리할 수 있다.

로그 파일이 없더라도 machregex를 이용하여 샘플 로그 메시지와 정규 표현식을 테스트 해 볼 수 있다.

장애 발생시에도 데이터 유실 방지

마크베이스 Collector 는 장애 발생 시 전송하지 못한 데이터를 장애 해결 후에 정확하게 재전송하는 기능을 제공한다.

장애가 발생하면 Collector 는 서버에 전송한 마지막 위치를 기록하고, 장애를 해결한 다음은 그 위치부터 데이터를 재 전송한다.

그래서 장애 극복을 위한 추가 조작이나 코드를 작성하지 않더라도, 모든 데이터의 유실없이 서버에 전송할 수 있다.

고가용성 보장

서비스의 고가용성을 보장하기 위해서 여러 Collector 들을 동일한 데이터 소스에 대해서 동시에 동작 시킬 수 있으며 이 Collector 들은 서로 다른 마크베이스 서버에 데이터를 전송한다.

이렇게 하면 마크베이스 서버에 오류가 발생하더라도 다른 서버에 동일 데이터를 지속적으로 저장하고 있으므로 서비스를 지속시킬 수 있으며,

오류를 해결하고 나서 서버를 재기동하면 Collector 는 미전송 로그 데이터를 정확히 재전송하므로 자동으로 데이터를 복제하여 고가용성을 제공할 수 있다.

MWA를 통한 통합 모니터링

마크베이스 Collector manager는 Collector 의 실행 정보를 마크베이스 서버에 동기화 한다.

이를 이용하여 MWA (Machbase Web Admin) 를 통한 통합 모니터링을 수행할 수 있다.

MWA를 이용하면, 실행 중인 Collector 의 다양한 상태 정보와 Collector 를 수행 중인 서버의 상태 정보를 실시간으로 모니터링할 수 있다.

Python 스크립트를 이용한 로그 전처리

Python 스크립트를 작성하여 Collector 가 데이터를 처리하기 전에 조작할 수 있다.

입력한 데이터를 처리하여 필요 없는 데이터는 입력하지 않도록 설정할 수도 있고 파싱한 데이터를 변경하도록 할 수도 있다.

Collector 설치

수동 설치

목차

$MACHBASE_COLLECTOR_HOME 디렉터리를 생성하고 다운로드한 collector 패키지를 복사한 후, 압축을 해제한다.

```
[mach@localhost ~]$ mkdir mach_collector_home
[mach@localhost ~]$ cp machcollector-x.x.x.official-LINUX-X86-64-release.tgz ma
[mach@localhost ~]$ cd mach_collector_home/
[mach@localhost ~/mach_collector_home]$ tar -zxvf machcollector-x.x.x.official-
bin/
bin/machregex
bin/machcollectord
bin/machcollector
bin/machcollectoradmin
collector/
collector/sign/
collector/apache_access.tpl
collector/samples/
collector/samples/test.rgx
collector/samples/test.tpl
collector/samples/test.log
collector/preprocess/
collector/preprocess/custom.py
collector/preprocess/skip.py
collector/preprocess/replace.py
collector/preprocess/pypyodbc_sample.py
collector/preprocess/trace.py
collector/custom.tpl
collector/syslog.tpl
collector/machbase.tpl
collector/regex/
collector/regex/meta.rgx
collector/regex/syslog.rgx
collector/regex/nxlog.rgx
collector/regex/machbase.rgx
collector/regex/unparse.rgx
collector/regex/apache_error.rgx
collector/regex/apache_access.rgx
conf/
doc/
meta/
trc/
webadmin/
webadmin/flask/
webadmin/flask/Python/
...
```

파일의 압축을 해제한 후, 다음의 디렉토리가 생성될 것이다.

```
[mach@localhost ~/mach_collector_home]$ ls -al
total 60372
drwxrwxr-x  9 mach mach     4096 Jun 27 23:58 .
drwx------ 20 mach mach     4096 Jun 27 23:40 ..
drwxrwxr-x  2 mach mach     4096 Jun 27 23:31 bin
drwxrwxr-x  6 mach mach     4096 Jun 27 23:31 collector
drwxrwxr-x  2 mach mach     4096 Jun 27 23:31 conf
drwxrwxr-x  2 mach mach     4096 Jun 27 23:31 doc
-rw-r--r--  1 mach mach 61783606 Jun 27 23:41 machcollector-x.x.x.official-LINUX-X86-64-release.tgz
drwxrwxr-x  2 mach mach     4096 Jun 27 23:31 meta
drwxrwxr-x  2 mach mach     4096 Jun 27 23:31 trc
drwxrwxr-x  3 mach mach     4096 Jun 27 23:31 webadmin
.. 생략
```

Name of directory	Description
bin	실행 파일
collector	설정 파일 예제
conf	설정 파일
doc	라이선스 파일
lib	라이브러리 파일
msg	메세지 파일
webadmin	Python 패키지

환경 변수 설정

실행 파일과 shared object 파일, 경로 환경 변수를 설정해야 한다.

```
export MACHBASE_COLLECTOR_HOME=/home/machbase/machbase_home
export PATH=$MACHBASE_COLLECTOR_HOME/bin:$PATH
export LD_LIBRARY_PATH=$MACHBASE_COLLECTOR_HOME/lib:$LD_LIBRARY_PATH
export MACH_COLLECTOR_PORT=9999
```

위 환경 변수를 적용하기 위해서 다음의 명령을 실행한다.

```
source .bashrc
```

`$MACHBASE_COLLECTOR_PORT` 를 지정하지 않은 경우, `$MACH_COLLECTOR_HOME/conf/machcollector.conf` 의 **PORT_NO** 값이 사용된다.

Collector 생성

리눅스 시스템의 syslog 에서 실시간으로 데이터를 수집하는 예제를 살펴보자.

보통 collector와 마크베이스 서버는 다른 장비에서 실행되지만, 이 예제에서는 편의를 위해 동일한 장비에서 실행하는 것을 가정한다.

목차

Collector Manager 실행

Collector manager는 collector들을 제어한다. Collector를 제어하기 위해서 collector manager가 먼저 실행되어야 한다. 아래의 명령으로 collector manager를 실행한다.

```
[mach@localhost ~]$ machcollectoradmin --startup
Waiting for collector manager starts.
Collector Manager started successfully.
```

Collector manager가 실행중인지를 알기 위해서 다음의 netstat명령을 실행한다.

```
[mach@localhost ~]$ netstat -anp | grep "LISTEN "
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
...
tcp 0 0 0.0.0.0:9999 0.0.0.0:* LISTEN 21163/machcollecto
...
[mach@localhost ~]$
```

Collector Manager 등록

Collector manager를 마크베이스 서버와 연결하기 위해서 마크베이스 서버에 collector manager를 등록한다. 아래의 명령을 machsql을 이용하여 실행한다.

```
CREATE COLLECTORMANAGER manager_name AT "host_addr:host_port";
```

- manager_name : collector manager의 이름. 중복된 값은 허용되지 않는다.
- host_addr : collector manager가 실행중인 서버의 ip
- host_port : collector manager가 실행중인 서버의 포트 번호

```
[mach@localhost ~/mach]$ machsql
================================================================
    Machbase Client Query Utility
    Release Version x.x.x.official
    Copyright 2015, Machbase Inc. or its subsidiaries.
    All Rights Reserved.
================================================================
Machbase server address (Default:127.0.0.1):
Machbase user ID (Default:SYS)
Machbase user password:
MACHBASE_CONNECT_MODE=INET, PORT=5656
mach>CREATE COLLECTORMANAGER LOCALHOST AT "127.0.0.1:9999";
Created successfully.
```

마크베이스 서버에 collector manager를 등록한 이후에는m$sys_collectormanagers 테이블에서 상태를 조회할 수 있다.

```
mach> SELECT * FROM m$sys_collectormanagers;
MANAGER_ID MANAGER_NAME MANAGER_HOST MANAGER_PORT MANAGER_STATE
-------------------------------------------------------------------------------------------------------------------------
1 LOCALHOST 127.0.0.1 9999 1
[1] row(s) selected.
```

해당 테이블에서 collector manager의 식별자, 명칭, 포트번호와 주소 및 실행 상태를 조회할 수 있다.

Collector 생성

Collector manager를 등록한 이후, collector객체를 collector manager를 통해서 생성한다.

Collector에 대한 정보는 마크베이스 서버에 저장되고, 그 정보를 조회할 수 있다. 아래의 명령을 machsql을 통해서 실행하여 collector를 생성한다.

```
CREATE COLLECTOR manager_name.collector_name FROM "path_for_template.tpl";
```

- manager_name : 해당 collector를 실행하는 collector manager의 이름
- collector_name : collector 객체의 이름
- path_for_template.tpl : collector의 설정 파일 경로. 다양한 샘플 설정파일이 "$MACHBASE_COLLECTOR_HOME/collector" 디렉토리에 있다. 원하는 샘플 파일을 선택하여 수정한 다음, 다른 파일로 저장하여 사용하는 것을 추천한다.

템플릿 파일 준비

템플릿 파일은 Collector의 데이터 소스, 처리 방법, 저장 방법을 기술한 text file이다. 샘플 파일들이 $MACHBASE_COLLECTOR_HOME/collector 디렉토리에 제공된다.

템플릿 파일의 구조

템플릿 파일은 마크베이스 프로퍼티 파일과 유사한 "variable name = value"의 구조로 되어 있다. 각 설정 변수의 자세한 정보는 아래 표와 같다.

> Machbase 3.5 버전 이후의 설정 파일은 이전 버전과 호환되지 않는다.

변수 이름	설명	비고
COLLECT_TYPE	데이터 수집 방법	데이터 수집 방법을 설정한다. 데이터 수집방법은 다음과 같다. FILE 은 기본 값으로 collector가 설치된 장비의 특정 파일을 지정한다. SFTP: 원격 SFTP파일 경로, SOCKET: socket 입력 데이터 입력. ODBC: ODBC로 설정된 데이터베이스에서 데이터 입력한다.
LOG_SOURCE	읽을 로그 파일의 위치	읽어들일 데이터 파일의 위치. SFTP 모드에서는 원격 호스트의 절대 경로를 지정해야 한다. SOKET 및 ODBC 모드에서는 사용되지 않는다. 원본 파일을 여러개로 설정하거나 정규 표현식으로 설정하는 것도 가능하다.
SFTP_HOST	SFTP_HOST	
SFTP_PORT	SFTP_PORT	설정하지 않은 경우 22가 기본으로 설정된다.
SFTP_USER	SFTP 사용자명	anonymous가 기본값으로 설정된다.
SFTP_PASS	SFTP 패스워드	anonymous가 기본값으로 설정된다.
SOCKET_PORT	Collector가 데이터를 입력받는 소켓 포트 번호	
SOCKET_PROTOCOL	Collector 소켓 프로토콜 종류	사용 가능한 값은 TCP와 UDP이며 기본값은 TCP이다.
ODBC_DSN	ODBC 모드의 DSN	".odbc.ini"의 값
ODBC_QUERY	ODBC 모드의 질의문	ODBC 데이터 소스에서 입력 데이터를 얻기 위해 실행하는 Query string
ODBC_SEQ_COLUMN	ODBC 모드의 증가되는 컬럼 명	숫자 컬럼만 가능하다.
LIB_NAME	외부 링크 라이브러리 패스	아직 사용되지 않는다.
REGEX_PATH	입력 데이터를 분석하기 위한 정규 표현식 파일	ODBC모드에서는 사용되지 않는다.
PREPROCESS_PATH	데이터 선처리용 Python 스크립트 파일의 위치	
SLEEP_TIME	데이터를 입력 완료후 대기 시간	밀리초 단위이며 기본값은 1000이다.
DB_TABLE_NAME	입력할 테이블 명	
DB_ADDR	입력할 데이터베이스의 IP주소	
DB_PORT	데이터베이스 포트 번호	
DB_USER	데이터베이스 사용자명	
DB_PASS	데이터베이스 패스워드	
APPEND_MODE	데이터 입력 방법 설정	과거 버전과의 호환성을 위한 값으로 사용되지 않는다.
AUTO_ADD_COLUMN	테이블 컬럼이 없는 경우 이를 자동으로 생성할 지의 여부 0이면 생성하지 않고 1이면 자동 생성함.	기본값은 1이다.

변수 이름	설명	비고
CREATE_TABLE_MODE	입력 테이블에 대한 연산을 설정한다. (0:아무것도 실행하지 않는다. 1: 기존 테이블을 truncate한다 2 : 테이블을 생성한다. 에러가 발생하면 에러를 trc에 기록하고 계속 진행한다. 3: 테이블을 drop하고 테이블을 재생성한다.)	일반적으로 2로 설정하는 것을 추천한다.
LANG	입력 데이터 파일의 인코딩을 지정한다.	사용가능 값은 UTF-8(기본값), CP949(MS949), KSC5601, EUCJP, SHIFTJIS, BIG5, BG231280가 지원된다.
REGEX_SORT	입력 파일의 순서를 결정한다.	기본값은 ASC이며 DESC도 가능한다.
ROTATE_FILE_PATH	Rotation 파일 경로명 설정	
ROTATE_FILE_COUNT	Rotation 파일 숫자 설정	
ROTATE_REGEX_SORT	Rotation 파일의 순서 설정	기본값은 ASC임 DESC도 가능한다.

REGEX_PATH, PREPROCESS_PATH는 collector가 실행시에 참조하는 파일이다. 아래는 REGEX_PATH에 설정되는 rgx파일에 대한 설명이다.

변수 이름	설명	비고
LOG_TYPE	정규식 이름	수정될 수 있는 값이지만, 데이터 베이스에 함께 저장되므로 값을 유지하는 것이 좋다.
COL_LIST	테이블내의 컬럼들의 리스트	테이블에 속해있는 컬럼들의 정보
REGEX	데이터 분석을 위한 정규표현식	
END_REGEX	한 레코드의 끝을 의미하는 정규표현식	각각의 레코드를 구분하기위한 정규표현식. 설정하지않으면 "\n" 줄바꿈을 기본으로 사용한다.

COL_LIST는 로그 파일과 데이터베이스 컬럼을 연결하는 정보를 기술한다. 정규표현식의 결과와 컬럼을 설정하는 다양한 정보를 설정해야 한다. COL_LIST를 이용하여 복잡한 로그 데이터를 구조화된 테이블 컬럼에 입력할 수 있다.

변수 이름	설명	비고
NAME	컬럼 이름	스페이스를 포함하지 않은 문자열이다.
TYPE	컬럼의 데이터 타입	타입의 이름을 의미한다.
SIZE	컬럼의 사이즈	컬럼의 실제 지정된 크기를 의미한다. 문자열은 생성된 혹은 생성될 크기에 따라 다른 값을 지정한다.(short (6), int (11), long (20), float (17), double (17), datetime (31), varchar (User-defined), ipv4 (15), ipv6 (45), text (64MB), binary (64MB))
DATE_FORMAT	타입이 datetime일 때, datetime 데이터의 형식	내부적으로 "strptime" 함수를 이용해 값을 파싱한다. e.g.) 'Aug 19 07:56:16' 는 '월 일 시간:분:초'의 포맷을 가진다. 따라서 사용되는 포맷값은 다음과 같다. "%b %d %H:%M:%S" add) 추가적인 인자로 %0,%1,%2가 존재하며 각각 밀리초, 마이크로초,나노초를 의미를 가지는 세자리 숫자가 입력된다.
USE_INDEX	인덱스 생성 여부	타입기반으로 LSM 또는 KEYWORD LSM 인덱스를 생성한다. 0: 생성하지 않는다./ 1: 생성한다.
REGEX_NO	정규표현식 내에서의 토큰 번호	정규식 파일에서 지정한 REGEX 문법중에 "()" 괄호로 싸인 영역이 하나의 토큰이 된다. 0번은 해당 레코드 데이터 전체를 의미한다. 이후 첫번째 위치한 괄호부터 1번 토큰이 된다.

syslog.tpl 예제

아래에 syslog.tpl파일의 예제를 기술한다. 해당 파일은 $MACHBASE_COLLECTOR_HOME/collector/syslog.tpl에 샘플로 제공된다.

```
##############################################################################
# Copyright of this product 2013-2023,
# Machbase Inc. or its subsidiaries.
# All Rights reserved
##############################################################################
```

```
#
#  This file is for Machbase collector template file.
#

########################################################################
# Input setting
########################################################################

COLLECT_TYPE=FILE <== It specifies a method to collect local data.
LOG_SOURCE=/var/log/syslog <== It specifies a location of source file.

########################################################################
# Process setting
########################################################################

REGEX_PATH=syslog.rgx        <== 정규 표현식 파일의 위치.
                                 $MACHBASE_HOME/collector/regex/ 를 root 로 둔다.

########################################################################
# Output setting
########################################################################

DB_TABLE_NAME = "syslogtable" <== 테이블 이름 : 데이터가 여기 입력됨
DB_ADDR       = "127.0.0.1"   <== 마크베이스 서버가 작동 중인 IP/PORT
DB_PORT       = 5656
DB_USER       = "SYS"
DB_PASS       = "MANAGER"

# 0: Direct insert
# 1: Prepared insert
# 2: Append
APPEND_MODE=2 <== Data insertion in APPEND mode.

# 0: None, just append.
# 1: Truncate.
# 2: Try to create table. If table already exists, warn it and proceed.
# 3: Drop and create.
CREATE_TABLE_MODE=2 <== Create a table if there is none.
```

syslog.rgx 파일은 syslog.tpl파일에 설정된 정규표현식 파일이다. rgx 파일을 설정할 때는 절대 경로로 설정하거나 $MACHBASE_COLLECTOR_HOME/collector/regex를 기준으로 한 상대 경로를 설정하면 된다.

```
##############################################################################
# Copyright of this product 2013-2023,
# Machbase Corporation (Incorporation) or its subsidiaries.
# All Rights reserved
##############################################################################

#
#  This file is for Machbase collector regex file.
#

LOG_TYPE=syslog

COL_LIST= (
     (
        REGEX_NO = 0 <== REGEX 정규표현식의 토큰 번호
        NAME = tm
        TYPE = datetime
        SIZE = 8
        DATE_FORMAT="%b %d %H:%M:%S" <== strptime 함수에서 사용하는 datetime 포맷
         ),
     (
        REGEX_NO = 4
        NAME = host
        TYPE = varchar
        SIZE = 128
```

```
        USE_INDEX = 1 <== 인덱스의 사용 여부
         ),
     (
        REGEX_NO = 5
        NAME = msg
        TYPE = varchar
        SIZE = 512
        USE_INDEX = 1
         )
)

# Below is the regular expression to pares syslog data. It may not work properly if it is modified.
REGEX="(([a-zA-Z]+)\s+([0-9]+)\s+([0-9:]*))\s(\S+)\s+([^\n]+)"

END_REGEX="\n"
```

Collector 생성

아래와 같이 collector "syslog_test"를 생성한다.

```
mach> CREATE COLLECTOR localhost.syslog_test FROM "/home/mach/mach_collector_home/collector/syslog.tpl";
Created successfully.
```

Collector 확인

M$SYS_COLLECTORS 테이블에 등록된 collector들의 정보가 있다. "RUN_FLAG" 컬럼 값이 1인 collector는 실행중이며 0이면 실행이 중지된 상태이다.

```
mach> SELECT collector_name, run_flag FROM m$sys_collectors;
collector_name                          run_flag
------------------------------------------------------------
SYSLOG_TEST                             0
[1] row(s) selected.
mach> SELECT * FROM m$sys_collectors;
COLLECTOR_ID MANAGER_NAME                                COLLECTOR_NAME
-------------------------------------------------------------------------------------------------------------
LOG_TYPE                                TABLE_NAME
-------------------------------------------------------------------------------------
TEMPLATE_NAME                                                                    COLLECT_TYPE
-------------------------------------------------------------------------------------------------------------------------------------------
COLLECTOR_SOURCE
-------------------------------------------------------------------------------
COLLECTOR_LIB                                                                    COL_COUNT
-------------------------------------------------------------------------------------------------------
PREPROCESS_PATH
-------------------------------------------------------------------------------
REGEX_PATH
-------------------------------------------------------------------------------
REGEX
-------------------------------------------------------------------------------
END_REGEX
-------------------------------------------------------------------------------
DEFAULT_ADDR                                                                     LANGUAGE
-------------------------------------------------------------------------------------------------------------------------------------------
SLEEP_TIME  DB_ADDR                                 DB_PORT     DB_USER
-------------------------------------------------------------------------------------------------------------------------------------
DB_PASS                                 RUN_FLAG
------------------------------------------------------------
1           LOCALHOST                               SYSLOG_TEST
syslog                                  syslogtable
/home/mach/mach_collector_home/collector/syslog.tpl                              FILE
/var/log/syslog
NULL                                                                               7
NULL
syslog.rgx
```

```
(([a-zA-Z]+)\s+([0-9]+)\s+([0-9:]*))\s(\S+)\s+([^\n]+)
\n
192.168.122.1                                                                    UTF-8
1000        127.0.0.1                                5656        SYS
MANAGER                                  0
[1] row(s) selected.
```

Collector 실행

```
ALTER COLLECTOR manager_name.collector_name START [TRACE];
```

등록된 collector를 시작하려면 ALTER COLLECTOR 문을 이용한다.

- manager_name : 등록된 collector manager의 이름
- collector_name : 실행할 collector의 이름

Collector 실행시 오류가 발생한 경우 $MACHBASE_COLLECTOR_HOME/trc/machcollector.trc 파일을 참고하여 문제 해결을 할 수 있다.

```
mach> ALTER COLLECTOR localhost.syslog_test START;
Altered successfully.
mach> SELECT collector_name, run_flag FROM m$sys_collectors;
collector_name                           run_flag
-----------------------------------------------------------
SYSLOG_TEST                              1
[1] row(s) selected.
```

ALTER COLLECTOR문으로 collector를 시작하면 RUN_FLAG 컬럼 값이 1 변경된 것을 알 수 있다.

Collector를 시작하면 수집된 데이터가 저장되는 log 테이블도 데이터베이스 서버에 생성된다. collector_type, collector_addr, collector_origin, collector_offset 값들은 기본 값으로 설정된다. syslog.tpl 파일에 설정된 tmp, host, msg 컬럼 또한 생성된다.

```
mach> ALTER COLLECTOR localhost.syslog_test START;
Altered successfully.
mach> SELECT collector_name, run_flag FROM m$sys_collectors;
collector_name                           run_flag
-----------------------------------------------------------
SYSLOG_TEST                              1
[1] row(s) selected.
```

Machsql 을 이용하여 질의문을 수행할 때, 마크베이스 서버에 접속하여 실행되는지를 확인해야 한다. 마크베이스 서버와 collector가 다른 장비에 설치된 경우, machsql이 접속하는 서버가 collector인 경우에는 정상적으로 실행이 안될 수 있다.

Collector 실행시 오류가 발생하여 중지된 이후, 재 실행되면 콜렉터는 마지막으로 입력한 데이터의 위치를 읽어 들여서 데이터를 유실하지 않고 나머지 데이터를 계속 입력한다.

데이터 확인

아래 내용은 원본 syslog의 마지막 10건과 데이터와 입력된 데이터를 비교한 것이다.

```
[mach@localhost ~/mach]$ tail -n 10 /var/log/syslog
Jun 28 21:05:01 localhost CROND[12285]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 28 21:10:01 localhost CROND[12442]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 28 21:10:01 localhost CROND[12443]: (root) CMD (/usr/lib64/sa/sa1 1 1)
Jun 28 21:15:01 localhost CROND[12527]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 28 21:20:01 localhost CROND[12609]: (root) CMD (/usr/lib64/sa/sa1 1 1)
Jun 28 21:20:01 localhost CROND[12608]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 28 21:25:01 localhost CROND[12707]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 28 21:25:01 localhost CROND[12708]: (pcp) CMD ( /usr/libexec/pcp/bin/pmlogger_check -C)
Jun 28 21:25:43 localhost su: pam_unix(su:session): session opened for user root by mach(uid=506)
Jun 28 21:26:02 localhost su: pam_unix(su:session): session closed for user root
```

아래 내용은 마크베이스 서버에 입력된 마지막 10건의 데이터이다.

```
mach> SELECT tm, msg FROM syslogtable LIMIT 10;
tm                              msg
-----------------------------------------------------------------------------------------------------------------
```

```
2016-06-28 21:26:02 000:000:000 su: pam_unix(su:session): session closed for user root
2016-06-28 21:25:43 000:000:000 su: pam_unix(su:session): session opened for user root by mach(uid=506)
2016-06-28 21:25:01 000:000:000 CROND[12708]: (pcp) CMD ( /usr/libexec/pcp/bin/pmlogger_check -C)
2016-06-28 21:25:01 000:000:000 CROND[12707]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
                            k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
2016-06-28 21:20:01 000:000:000 CROND[12608]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
                            k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
2016-06-28 21:20:01 000:000:000 CROND[12609]: (root) CMD (/usr/lib64/sa/sa1 1 1)
2016-06-28 21:15:01 000:000:000 CROND[12527]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
                            k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
2016-06-28 21:10:01 000:000:000 CROND[12443]: (root) CMD (/usr/lib64/sa/sa1 1 1)
2016-06-28 21:10:01 000:000:000 CROND[12442]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
                            k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
2016-06-28 21:05:01 000:000:000 CROND[12285]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
                            k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
[10] row(s) selected.
```

Collector 정지

```
ALTER COLLECTOR manager_name.collector_name STOP;
```

다음 명령으로 collector를 중지시킬 수 있다.

```
mach> ALTER COLLECTOR localhost.syslog_test STOP;
Altered successfully.
```

Collector 삭제

```
DROP COLLECTOR manager_name.collector_name;
```

```
mach> DROP COLLECTOR localhost.syslog_test;
Dropped successfully.
```

Collector가 삭제된 상황은 다음의 질의로 확인할 수 있다.

```
mach> SELECT collector_name, run_flag FROM m$sys_collectors;
collector_name                          run_flag
------------------------------------------------------------
[0] row(s) selected.
```

Collector 갱신

```
ALTER COLLECTOR manager_name.collector_name RELOAD;
```

콜렉터를 생성한 이후, 템플릿 파일을 변경하고 그 내용을 새로 적용할 경우에 사용한다. 실행시 갱신된 템플릿 파일의 내용이 적용된다. 아래 예제는 데이터를 입력할 테이블을 원래 값 대신에 "anothertable"로 바꾼 경우이다.

```
mach> ALTER COLLECTOR localhost.custom RELOAD;
Altered successfully.

mach> SELECT * FROM m$sys_collectors;
COLLECTOR_ID MANAGER_NAME                                   COLLECTOR_NAME
-------------------------------------------------------------------------------------------------------
LOG_TYPE                                TABLE_NAME
-------------------------------------------------------------------------------
TEMPLATE_NAME                                                                   COLLECT_TYPE
-------------------------------------------------------------------------------------------------------------------------
```

```
COLLECTOR_SOURCE
--------------------------------------------------------------------------------
COLLECTOR_LIB                                                                   COL_COUNT
--------------------------------------------------------------------------------------------------
PREPROCESS_PATH
--------------------------------------------------------------------------------
REGEX_PATH
--------------------------------------------------------------------------------
REGEX
--------------------------------------------------------------------------------
END_REGEX                                                                       LANGUAGE
--------------------------------------------------------------------------------------------------------------------------------------------------
SLEEP_TIME  DB_ADDR                                 DB_PORT     DB_USER
--------------------------------------------------------------------------------------------------------------------------------------------------
DB_PASS                                 PROCESS_BYTE         PROCESS_RECORD       RUN_FLAG
--------------------------------------------------------------------------------------------------------------------------------
4           LOCALHOST                               CUSTOM
syslog                                  anothertable
syslog.tpl                                                                      FILE
/var/log/syslog
NULL                                                                            7
NULL
syslog.rgx
(([a-zA-Z]+)\s+([0-9]+)\s+([0-9:]*))\s(\S+)\s+([^\n]+)
\n                                                                              UTF-8
1000        127.0.0.1                               5656        SYS
MANAGER                                 0                   0                   0
[1] row(s) selected.
```

메타 테이블을 조회하면, 입력 테이블이 anothertable 로 바뀐 것을 알 수 있다.

데이터 수집 방법 (1) FILE/SFTP

마크베이스 collector가 지원하는 다양한 데이터 수집 모드와 이를 적용하는 방법을 기술한다. Collector가 지원하는 데이터 수집 모드는 FILE, SFTP, SOCKET, ODBC이다.

아래 설명은 collector와 마크베이스 서버가 동일한 장비에 설치되어 있고, 서버의 hostname이 "localhost"이며, colletor는 127.0.0.1:9999 에서 동작하는 것을 가정한다.

> **데이터 입력 모드 설정**
>
> 데이터 입력 모드는 COLLECT_TYPE 변수를 조정하면 바꿀 수 있다.
> 현재는 FILE과 SFTP, SOCKET, ODBC 등을 설정할 수 있다. 각 모드에 대해서 추가 변수를 설정해야 한다.

목차

FILE 방식

파일 모드 입력을 설정하면 collector가 실행 중인 서버의 파일을 읽어서 처리한다.

추가적인 옵션 설정

파일 모드에서는 데이터를 localhost에서 읽기 때문에, 그 파일의 경로명과 파일 읽기 권한만 체크하면 된다.

옵션 이름	설명	비고
LOG_SOURCE	로그 파일이 위치한 경로	절대경로로 작성해야 한다.

예제 (1) 로그 파일 확인

아래의 예제는 파일 모드 입력 방법으로 "/var/log/syslog" 파일의 데이터를 수집하여 마크베이스 서버에 입력하도록 하는 것이다.

먼저 입력 파일을 collector 프로세스가 읽을 수 있는지 확인해야 한다.

```
[mach@localhost ~]$ head /var/log/syslog
head: cannot open '/var/log/syslog' for reading: Permission denied
```

위 결과를 보면 입력 파일에 대해서 읽기 권한이 없는 것을 알 수 있다.

입력 파일(/var/log/syslog)에 읽기 권한을 collector를 실행하는 사용자에게 부여하여 collector프로세스가 입력 파일을 읽을 수 있도록 해야 한다.

아래의 예제는 그 과정을 보여준다.

```
[mach@localhost ~]$ su
# chmod 744 /var/log/syslog
# exit
[mach@localhost ~]$
```

다시, head 명령을 이용하여 입력 파일을 읽을 수 있는지 검사하면 다음의 결과를 얻는다.

```
[mach@localhost ~]$ head /var/log/syslog
Jun 20 04:31:43 localhost kernel: imklog 5.8.10, log source = /proc/kmsg started.
Jun 20 04:31:43 localhost rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="15062" x-info="http://www
Jun 20 04:31:46 localhost kernel: imklog 5.8.10, log source = /proc/kmsg started.
Jun 20 04:35:01 localhost CROND[15111]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 20 04:40:01 localhost CROND[15188]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 20 04:40:01 localhost CROND[15187]: (root) CMD (/usr/lib64/sa/sa1 1 1)
Jun 20 04:45:01 localhost CROND[15265]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 20 04:50:01 localhost CROND[15341]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
Jun 20 04:50:01 localhost CROND[15342]: (root) CMD (/usr/lib64/sa/sa1 1 1)
Jun 20 04:55:01 localhost CROND[15419]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /v
[mach@localhost ~]$
```

예제 (2) 정규 표현식 파일 작성

입력 파일에 대한 권한을 확인하고 나서, 데이터를 파싱하기 위한 정규 표현식 파일을 작성해야 한다.

작성한 정규 표현식이 데이터 포맷과 맞는지 확인하기 위해서 마크베이스가 제공하는 machregex 도구를 이용할 수 있다.
machregex 도구는 두 개의 정규 표현식을 매개변수로 설정하고 데이터를 입력한다.

- 첫 번째 정규 표현식(REGEX)은 입력 데이터를 파싱하기 위해서 사용된다.
- 두 번째 정규 표현식(END_REGEX)는 입력 데이터를 한 건씩 분리하기 위해서 사용된다.

아래의 예제는 machregex를 정규 표현식과 데이터 파일을 이용하여 실행해 본 것이다.

```
[mach@localhost ~]$ head /var/log/syslog > syslog
[mach@localhost ~]$ machregex "^(([a-zA-Z]+)\\s+([0-9]+)\\s+([0-9:]*))\\s(\\S*)\\s+((?:[^\\0])*)$" ".*" < syslog
Pattern => (^(([a-zA-Z]+)\s+([0-9]+)\s+([0-9:]*))\s(\S*)\s+((?:[^\0])*)$)
================================================================================
SUCCESS[2] (rc=7)(Jun 20 04:31:43 localhost kernel: imklog 5.8.10, log source = /proc/kmsg started.
)
  ALL (0:82) => [Jun 20 04:31:43 localhost kernel: imklog 5.8.10, log source = /proc/kmsg started.
]
  0 (0:15) => [Jun 20 04:31:43]
  1 (0:3) => [Jun]
  2 (4:6) => [20]
  3 (7:15) => [04:31:43]
  4 (16:25) => [localhost]
  5 (26:82) => [kernel: imklog 5.8.10, log source = /proc/kmsg started.
]
```

파싱 결과로 출력되는 값은 6개의 토큰으로 표시된다. 이 파일에 따르면, 마크베이스는 0, 4, 5의 토큰을 이용하고 나머지 토큰들은 버린다.

- 0: 시간 표현 문자열 이 문자열을 datetime으로 변환하려면 format을 지정해야 한다.
- 4: 호스트 명칭
- 5: 로그 데이터 메시지.

ⓘ 분석하려는 데이터가 위의 파싱 규칙에 따라 문제없이 처리된다면, 마크베이스가 제공하는 syslog.rgx 파일을 그대로 사용해도 좋다.
(이 파일은 `$MACHBASE_HOME/collector/regex/` 폴더에 있다.)

이 폴더에 있는 정규 표현식 파일을 이용할 경우, 템플릿 파일의 **REGEX_PATH** 변수에 파일 경로를 쓰지 않고 파일명 만을 설정해도 된다.

예제 (3) 템플릿 파일 작성

아래에 syslog.tpl 템플릿 파일의 예를 기술한다.

```
##############################################################################
# Copyright of this product 2013-2023,
# Machbase Inc. or its subsidiaries.
# All Rights reserved
##############################################################################

#
#  This file is for Machbase collector template file.
#

#########################################################################
# Collect setting
#########################################################################

COLLECT_TYPE=FILE

LOG_SOURCE=/var/log/syslog

#########################################################################
# Process setting
#########################################################################

REGEX_PATH=syslog.rgx

#########################################################################
# Output setting
#########################################################################
```

```
DB_TABLE_NAME = "file_syslogtable"
DB_ADDR       = "127.0.0.1"
DB_PORT       = 5656
DB_USER       = "SYS"
DB_PASS       = "MANAGER"

# 0: Direct insert
# 1: Prepared insert
# 2: Append
APPEND_MODE=2

# 0: None, just append.
# 1: Truncate.
# 2: Try to create table. If table already exists, warn it and proceed.
# 3: Drop and create.
CREATE_TABLE_MODE=2
```

컬렉터 설정에서 기본 파일 경로가 아닌 파일들은 반드시 절대 경로('/'로 시작하는 경로)와 파일명을 명시해야 한다.

읽어들일 파일 이름은 LOG_SOURCE 변수에 지정하고,그 데이터를 파싱하기 위한 정규 표현식 파일도 설정해야 한다.

예제 (4) Collector 실행

마크베이스 서버에 접속하기 위한 정보와 테이블 생성 방식을 설정하여 템플릿 파일 설정이 끝나면 다음과 같이 collector 를 실행한다.

```
Mach> create collector localhost.file_syslog from "/home/machbase_home/collector/syslog.tpl";
Created successfully.

Mach> ALTER COLLECTOR localhost.file_syslog START;
Altered successfully.
```

collector의 생성 및 실행이 성공했다면, (테이블이 없을 경우에 한해) 테이블이 생성되고 데이터가 입력되기 시작한다.

데이터가 정상적으로 입력되고 있는지 확인하려면, 생성된 테이블에 SELECT 쿼리를 실행하여 확인할 수 있다.

```
Mach> SELECT * FROM file_syslogtable ORDER BY _arrival_time asc LIMIT 10;
COLLECTOR_TYPE                   COLLECTOR_ADDR
--------------------------------------------------
COLLECTOR_ORIGIN                                                                                    COLLECTOR_OFFSET
-----------------------------------------------------------------------------------------------------------------------
TM                              HOST
-----------------------------------------------------------------------------------------------------------------------------------------
MSG
-------------------------------------------------------------------------------
FILE                             127.0.0.1
/var/log/syslog                                                                                     81
2016-06-20 04:31:43 000:000:000 localhost
kernel: imklog 5.8.10, log source = /proc/kmsg started.
FILE                             127.0.0.1
/var/log/syslog                                                                                     217
2016-06-20 04:31:43 000:000:000 localhost
rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="15062" x-info="h
ttp://www.rsyslog.com"] start
FILE                             127.0.0.1
/var/log/syslog                                                                                     256
2016-06-20 04:31:46 000:000:000 localhost
kernel: imklog 5.8.10, log source = /proc/kmsg started.
FILE                             127.0.0.1
/var/log/syslog                                                                                     431
2016-06-20 04:35:01 000:000:000 localhost
CROND[15111]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
FILE                             127.0.0.1
/var/log/syslog                                                                                     606
2016-06-20 04:40:01 000:000:000 localhost
CROND[15188]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
FILE                             127.0.0.1
/var/log/syslog                                                                                     681
```

```
2016-06-20 04:40:01 000:000:000 localhost
CROND[15187]: (root) CMD (/usr/lib64/sa/sa1 1 1)
FILE                            127.0.0.1
/var/log/syslog                                                                 856
2016-06-20 04:45:01 000:000:000 localhost
CROND[15265]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
FILE                            127.0.0.1
/var/log/syslog                                                                1031
2016-06-20 04:50:01 000:000:000 localhost
CROND[15341]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
FILE                            127.0.0.1
/var/log/syslog                                                                1106
2016-06-20 04:50:01 000:000:000 localhost
CROND[15342]: (root) CMD (/usr/lib64/sa/sa1 1 1)
FILE                            127.0.0.1
/var/log/syslog                                                                1281
2016-06-20 04:55:01 000:000:000 localhost
CROND[15419]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
[10] row(s) selected.
```

SFTP 방식

원격 파일에서 데이터를 수집하기 위해서 SFTP 모드를 사용할 수 있다.

원격 파일을 SFTP를 통해서 접근할 수 있어야 한다. FILE 모드와 유사하나 파일을 SFTP를 통하여 접근하므로 SFTP 관련 변수를 설정하여야 한다.

추가적인 옵션 설정

SFTP 모드로 데이터를 수집하려면 다음의 변수들을 추가 설정하여야 한다.

옵션 이름	설명	비고
LOG_SOURCE	원격 위치의 데이터 파일 경로및 파일명	절대경로로 작성해야 한다.
SFTP_HOST	SFTP 서버의 IP 주소	
SFTP_PORT	SFTP서버의 포트 번호	포트 번호를 설정하지 않으면 22번이 기본으로 사용됨
SFTP_USER	SFTP 사용자명	
SFTP_PASS	SFTP 패스워드	

예제 (1) SFTP 접근 확인

SFTP로 로그인 및 수집 파일을 읽을 수 있는지를 확인하고, 읽을 수 없다면 다음을 시도해야 한다.

1. SFTP 로그인 문제 해결
2. 파일 권한 관련 문제 해결 (FILE 방식을 참고)

예제 (2-3) 정규 표현식/템플릿 파일 작성

이 작업이 완료되었다면, 정규 표현식 파일과 템플릿 파일을 작성한다.

이 작업 설명은 위의 FTP 방식을 참고한다.

```
##############################################################################
# Copyright of this product 2013-2023,
# Machbase Corporation (Incorporation) or its subsidiaries.
# All Rights reserved
##############################################################################

#
#  This file is for Machbase collector template file.
#
```

```
########################################################################
# Collect setting
########################################################################

COLLECT_TYPE=SFTP

SFTP_HOST=127.0.0.1
SFTP_PORT=22
SFTP_USER=mach
SFTP_PASS=mach

LOG_SOURCE=/var/log/syslog

########################################################################
# Process setting
########################################################################

REGEX_PATH=syslog.rgx

########################################################################
# Output setting
########################################################################

DB_TABLE_NAME = "sftp_syslogtable"
DB_ADDR       = "127.0.0.1"
DB_PORT       = 5656
DB_USER       = "SYS"
DB_PASS       = "MANAGER"

# 0: Direct insert
# 1: Prepared insert
# 2: Append
APPEND_MODE=2

# 0: None, just append
# 1: Truncate
# 2: Try to create table. If table already exists, warn it and proceed.
# 3: Drop and create
CREATE_TABLE_MODE=2
```

예제 (4) Collector 실행

아래 예제는 위 템플릿 파일로 SFTP 를 이용한 컬렉터를 생성하는 것을 보여준다.

```
Mach> create collector localhost.sftp_syslog from "/home/mach/mach_collector_home/collector/sftp_syslog.tpl";
Created successfully.

Mach> alter collector localhost.sftp_syslog start;
Altered successfully.
```

Collector 생성 및 시작을 성공하였다면, 다음과 같이 컬렉터가 수집한 데이터를 확인할 수 있다.

```
Mach> select * from sftp_syslogtable order by _arrival_time asc limit 10;
COLLECTOR_TYPE                   COLLECTOR_ADDR
-------------------------------------------------
COLLECTOR_ORIGIN                                                                                    COLLECTOR_OFFSET
-----------------------------------------------------------------------------------------------------------------------
TM                               HOST
-----------------------------------------------------------------------------------------------------------------------------------------------------------
MSG
-------------------------------------------------------------------------------
SFTP                             127.0.0.1
/var/log/syslog                                                                                     81
2016-06-20 04:31:43 000:000:000 localhost
kernel: imklog 5.8.10, log source = /proc/kmsg started.
SFTP                             127.0.0.1
/var/log/syslog                                                                                     217
```

```
2016-06-20 04:31:43 000:000:000 localhost
rsyslogd: [origin software="rsyslogd" swVersion="5.8.10" x-pid="15062" x-info="h
ttp://www.rsyslog.com"] start
SFTP                            127.0.0.1
/var/log/syslog                                                              256
2016-06-20 04:31:46 000:000:000 localhost
kernel: imklog 5.8.10, log source = /proc/kmsg started.
SFTP                            127.0.0.1
/var/log/syslog                                                              431
2016-06-20 04:35:01 000:000:000 localhost
CROND[15111]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
SFTP                            127.0.0.1
/var/log/syslog                                                              606
2016-06-20 04:40:01 000:000:000 localhost
CROND[15188]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
SFTP                            127.0.0.1
/var/log/syslog                                                              681
2016-06-20 04:40:01 000:000:000 localhost
CROND[15187]: (root) CMD (/usr/lib64/sa/sa1 1 1)
SFTP                            127.0.0.1
/var/log/syslog                                                              856
2016-06-20 04:45:01 000:000:000 localhost
CROND[15265]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
SFTP                            127.0.0.1
/var/log/syslog                                                             1031
2016-06-20 04:50:01 000:000:000 localhost
CROND[15341]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
SFTP                            127.0.0.1
/var/log/syslog                                                             1106
2016-06-20 04:50:01 000:000:000 localhost
CROND[15342]: (root) CMD (/usr/lib64/sa/sa1 1 1)
SFTP                            127.0.0.1
/var/log/syslog                                                             1281
2016-06-20 04:55:01 000:000:000 localhost
CROND[15419]: (root) CMD (LANG=C LC_ALL=C /usr/bin/mrtg /etc/mrtg/mrtg.cfg --loc
k-file /var/lock/mrtg/mrtg_l --confcache-file /var/lib/mrtg/mrtg.ok)
[10] row(s) selected.
```

데이터 수집 방법 (2) Socket/ODBC

Socket 방식

Collector가 소켓에서 데이터를 읽어서 이를 파싱하고 데이터베이스 서버에 입력하는 방법이다. 소켓 모드를 이용하려면 포트 번호를 설정해야 한다. 이 모드를 이용하면 rsyslog, logstash, nxlog등의 프로그램에서 처리한 로그 데이터를 수신하여 처리할 수 있다.

추가 값 설정

SOCKET 모드로 데이터를 수신하기 위해서 SOCKET_PORT변수의 설정이 필요하다.

옵션 이름	설명	비고
SOCKET_PORT	데이터를 수신할 포트 번호	다른 프로그램이 사용하지 않는 포트를 지정해야 한다.

목차

예제 (1) 정규 표현식/템플릿 파일 작성

이 예제는 SOCKET 모드로 데이터를 수신하는 것이다. SOCKET 모드는 소켓을 열고 외부 프로그램이 데이터를 입력하는 것을 기다린다.

Syslog를 수집하기 위해서는 rsyslog를 이용할 것을 추천한다. 이 예제는 rsyslog를 이용하여 데이터를 수집하여 분석한다.

데이터의 송신은 다른 프로그램에서 실행하는 것으로 아래의 설정 파일에서는 단순히 rgx 파일과 tpl파일의 설정만을 다룬다.

> FILE 모드 데이터 처리 예제에서 사용한 syslog.rgx는 그대로 사용 가능하다.

아래 예제에서 소켓 포트 번호는 33333 을 이용하였다. 이 포트는 방화벽에서 접속 허용을 해 주어야 한다.

```
################################################################################
# Copyright of this product 2013-2023,
# Machbase Corporation (Incorporation) or its subsidiaries.
# All Rights reserved
################################################################################

# This file is for Machbase collector template file.

#########################################################################
# Collect setting
#########################################################################

COLLECT_TYPE=SOCKET

SOCKET_PORT=33333

#########################################################################
# Process setting
#########################################################################

REGEX_PATH=syslog.rgx

#########################################################################
# Output setting
#########################################################################

DB_TABLE_NAME = "socket_syslogtable"
DB_ADDR = "127.0.0.1"
DB_PORT = 5656
DB_USER = "SYS"
DB_PASS = "MANAGER"

# 0: Direct insert
```

```
# 1: Prepared insert
# 2: Append
APPEND_MODE=2

# 0: None, just append.
# 1: Truncate.
# 2: Try to create table. If table already exists, warn it and proceed.
# 3: Drop and create.
CREATE_TABLE_MODE=2
```

예제 (2) Collector 실행

위 예제의 socket_syslog.tpl 파일을 이용하여 콜렉터를 생성하고 이를 실행한다. (생성 및 실행은 FILE모드와 동일하다.)

```
Mach> create collector localhost.file_syslog from "/home/machbase_home/collector/socket_syslog.tpl";
Created successfully.

Mach> ALTER COLLECTOR localhost.file_syslog START;
Altered successfully.
```

Collector가 정상적으로 실행되어 연결 중인지 확인하려면 다음의 명령을 실행한다.

```
[mach@localhost ~]$ netstat -anp | grep "LISTEN " | grep 33333
(Not all processes could be identified, non-owned process info will not be shown, you would have to be root to see
tcp 0 0 0.0.0.0:33333 0.0.0.0:* LISTEN 20818/machcollecto
[mach@localhost ~]$
```

위 실행 결과를 보면 33333 포트에서 collector가 데이터의 입력을 대기하고 있는 것을 확인할 수 있다.

예제 (3) 데이터 입력

데이터를 rsyslog를 이용하여 입력해 보자.

rsyslog 프로그램의 설정을 root 권한이 있는 사용자로 로그인하여 rsyslog 설정 파일(/etc/rsyslog.d/.conf)를 다음과 같이 작성한다.

```
*.* @@127.0.0.1:33333
```

또는 아래와 같이 자세한 설정을 할 수 있다.

아래의 내용을 "/etc/rsyslog.d/127.0.0.1_syslog.conf" 파일로 작성한다. rsyslog를 재시작하면 syslog 데이터가 생성될 때 마다 socket으로 데이터를 출력해 준다.

```
$ModLoad imfile

$InputFileName /var/log/syslog
$InputFileTag syslog_file:
$InputFileStateFile stat-syslog
$InputFilePollInterval 1
$InputRunFileMonitor

if $programname == 'syslog_file' then @@127.0.0.1:33333
```

그 다음, rsyslog 데몬을 재시작한다.

```
# service rsyslog restart
rsyslog stop/waiting
rsyslog start/running, process 22936
```

rsyslog가 재시작되면 /var/log/syslog데이터를 127.0.0.1:33333포트에 기록한다.

collector는 소켓 연결을 통해 데이터를 수집하고 이를 분석하여 마크베이스 서버에 입력한다.
rsyslog 설정에서 **$InputFilePollInterval** 변수를 1로 설정하였으므로 초기에 데이터 입력 속도가 느릴 수 있다.

데이터가 정상적으로 입력되었다면 데이터베이스 서버에서 아래의 SELECT 쿼리를 이용하여 해당 테이블에서 데이터를 확인할 수 있다.

```
Mach> select * from socket_syslogtable order by _arrival_time asc limit 10;
COLLECTOR_TYPE COLLECTOR_ADDR
-----------------------------------------------------
```

```
COLLECTOR_ORIGIN COLLECTOR_OFFSET
-------------------------------------------------------------------------------------------------------------
TM HOST
-------------------------------------------------------------------------------------------------------------------------
MSG
-------------------------------------------------------------------------------
SOCKET 127.0.0.1
NULL 1
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:31:43 localhost kernel: imklog 5.8.10, log source = /proc
/kmsg started.
SOCKET 127.0.0.1
NULL 2
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:31:43 localhost rsyslogd: [origin software="rsyslogd" swV
ersion="5.8.10" x-pid="15062" x-info="http://www.rsyslog.com"] start
SOCKET 127.0.0.1
NULL 3
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:31:46 localhost kernel: imklog 5.8.10, log source = /proc/kmsg started.
SOCKET 127.0.0.1
NULL 4
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:35:01 localhost CROND[15111]: (root) CMD (LANG=C LC_ALL=C
/usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /var/lock/mrtg/mrtg_l --confcache-
file /var/lib/mrtg/mrtg.ok)
SOCKET 127.0.0.1
NULL 5
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:40:01 localhost CROND[15188]: (root) CMD (LANG=C LC_ALL=C
/usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /var/lock/mrtg/mrtg_l --confcache-
file /var/lib/mrtg/mrtg.ok)
SOCKET 127.0.0.1
NULL 6
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:40:01 localhost CROND[15187]: (root) CMD (/usr/lib64/sa/s
a1 1 1)
SOCKET 127.0.0.1
NULL 7
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:45:01 localhost CROND[15265]: (root) CMD (LANG=C LC_ALL=C
/usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /var/lock/mrtg/mrtg_l --confcache-
file /var/lib/mrtg/mrtg.ok)
SOCKET 127.0.0.1
NULL 8
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:50:01 localhost CROND[15341]: (root) CMD (LANG=C LC_ALL=C
/usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /var/lock/mrtg/mrtg_l --confcache-
file /var/lib/mrtg/mrtg.ok)
SOCKET 127.0.0.1
NULL 9
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:50:01 localhost CROND[15342]: (root) CMD (/usr/lib64/sa/s
a1 1 1)
SOCKET 127.0.0.1
NULL 10
2016-06-28 23:50:17 000:000:000 localhost
syslog_file: Jun 20 04:55:01 localhost CROND[15419]: (root) CMD (LANG=C LC_ALL=C
/usr/bin/mrtg /etc/mrtg/mrtg.cfg --lock-file /var/lock/mrtg/mrtg_l --confcache-
file /var/lib/mrtg/mrtg.ok)
[10] row(s) selected.
```

Log Collector 설정

소켓 입력 모드 예제는 *rsyslog, logstash, nxlog* 를 이용하여 실행된다.

이들 프로그램으로 로그 데이터를 소켓을 통해 입력 받으면 collector는 이를 수집하여 데이터베이스 서버에 입력한다.

rsyslog

rsyslog는 최근의 리눅스 배포판에 기본으로 포함되는 경우가 많다.
그래서 추가로 설치할 필요 없이 설정파일을 /etc/rsyslog.d/ 디렉토리에 추가한 다음 rsyslog 데몬을 재시작 해 주면 된다.

rsyslog는 이미 기록되어 있는 로그 데이터만 전송하는 것이 아니라, 신규로 로그 데이터가 기록될 때마다 데이터를 전송해 준다.

아래에 설정관련 내용이 있다.

간단한 설정 : 로그 전달 주소 설정

로그가 생성될 때 전달할 주소만 지정하는 방법이다. 다른 방법에 비해서 간단하다.
아래의 내용을 /etc/rsyslog.d/.conf 파일에 작성후 rsyslog를 재시작한다.

```
*.* @@<Collector host>:<Collector port>
```

이후 syslog 데이터는 **collector host:port** 로 전송된다.

복잡한 설정 : 입력 로그 파일 / 전송 주기 설정

좀 더 복잡한 방법은 입력 로그 파일 및 전송 주기를 설정할 수 있는 방법이다.

```
$ModLoad imfile

$InputFileName /var/log/syslog
$InputFileTag syslog_file:
$InputFileStateFile stat-syslog
$InputFilePollInterval 1
$InputRunFileMonitor

if $programname == 'syslog_file' then @@<Collector host>:<Collector port>
```

/etc/rsyslog.d/ 폴더에 .conf로 끝나는 파일을 작성한 후, rsyslog를 재시작하면 적용된다.

자세한 내용은 rsyslog 설명을 참고하라.

logstash

logstash를 설치하기 위해서는 Getting Started with Logstash를 참고하라.

logstash의 conf 파일을 수정하여 원하는 데이터를 소켓으로 전송할 수 있다.

아래의 예제를 참고한다.

```
input {
   file {
      path => "<Absolute path of log file>"
      }
   }
output {
   tcp {
      host => "<Collector host>"
      port => "<Collector port>"
        }
      }
```

- "input" 부분에 입력 데이터 파일의 위치를 설정한다. syslog를 입력하고 싶다면 /var/log/syslog를 설정한다.
- "output"부분에는 collector의 tcp 소켓을 입력해야 하므로 tcp 를 설정하고 ip와 포트번호를 설정한다.

Nxlog

Nxlog는 Windows를 위한 로그 수집기이다.

소켓 입력 모드를 위한 collector를 위한 rgx, tpl파일의 설정은 동일하고, nxlog를 위한 설정 파일의 예제는 다음과 같다.

보통 nxlog는 "C:₩Program Files₩nxlog" 또는 "C:₩Program Files (x86)₩nxlog"에 설치된다.

위 경로에 위치한 설정 파일을 다음과 같이 작성한다.

```
## This is a sample configuration file. See the nxlog reference manual about the
## configuration options. It should be installed locally and is also available
## online at http://nxlog-ce.sourceforge.net/nxlog-docs/en/nxlog-reference-manual.html
```

```
## Set the ROOT to the folder your nxlog was installed,
## otherwise it will not start.


#define ROOT C:\Program Files\nxlog
define ROOT C:\Program Files\nxlog


Moduledir %ROOT%\modules
CacheDir %ROOT%\data
Pidfile %ROOT%\data\nxlog.pid
SpoolDir %ROOT%\data
LogFile %ROOT%\data\nxlog.log


<Input in>
Module im_msvistalog
# For windows 2003 and earlier use the following:
# Module im_mseventlog
</Input>


<Output out>
Module om_tcp
Host <Collector host>
Port <Collector port>
</Output>


<Route 1>
Path in => out
</Route>
```

위의 예제에서 im_msvistalog 파일에 데이터가 기록되면 <Collector ip>:<collector port>로 소켓을 통한 데이터 전송을 설정한다.

설정 파일을 변경하고 서비스를 재시작하면 데이터를 소켓을 통해 collector에 전송한다.

자세한 설정 방법은 nxlog 메뉴얼을 참고하라.

ODBC 방식

ODBC모드는 ODBC연결로 접속할 수 있는 데이터베이스의 데이터를 collector로 수집하는 방법이다.

Linux환경에서는 unixODBC패키지를 설치해야 사용할 수 있다.
아래의 예제는 MySQL데이터베이스의 데이터를 unixODBC를 통해서 수집하는 것이다.

unixODBC 와 MyODBC 의 설치 방법은 각각의 웹사이트를 참조하라.

추가 값 설정

다음의 변수들을 설정해야 한다.

옵션 이름	설명	비고
ODBC_DSN	데이터베이스에 접근하기 위한 DSN	ODBC 설정에 기술된 DSN을 이용해야 한다.
ODBC_QUERY	데이터 검색을 위한 질의문	
ODBC_SEQ_COLUMN	순차 증가값의 컬럼 명	질의 대상 컬럼중의 하나여야 함. 숫자형만 지원함.

예제 (1) 데이터 생성

먼저 MySQL 데이터를 입력해야 한다. 다음 내용으로 데이터를 입력한다.

```
0,2015-05-20 06:00:00,16.194.51.72,6790,183.103.50.46,5281,20,GET /twiki/bin/view/TWiki/KlausWriessnegger HTTP/1.1,
1,2015-05-20 06:00:02,96.40.75.42,11011,31.224.72.52,12069,55,GET /twiki/bin/search/Main/SearchResult?scope=text®ex
2,2015-05-20 06:00:02,174.47.129.59,6032,96.40.75.42,6442,72,GET /twiki/bin/edit/Main/WebSearch?t=1078669682 HTTP/1
3,2015-05-20 06:00:02,153.199.166.54,4220,86.45.186.17,2245,1,GET /twiki/bin/oops/TWiki/RichardDonkin?template=oops
4,2015-05-20 06:00:02,226.7.237.25,10805,50.230.44.173,179,70,GET /twiki/bin/oops/TWiki/AppendixFileSystem?template
```

```
5,2015-05-20 06:00:02,183.103.50.46,7175,96.128.212.177,7175,73,GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1,200,4
6,2015-05-20 06:00:02,123.198.82.192,6784,63.214.191.124,10825,21,GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1,200
7,2015-05-20 06:00:02,214.153.107.182,5562,85.183.139.166,1367,8,GET /twiki/bin/oops/TWiki/RichardDonkin?template=o
8,2015-05-20 06:00:02,245.13.24.17,7451,69.99.246.62,4497,20,GET /twiki/bin/view/Main/SpamAssassin HTTP/1.1,200,408
9,2015-05-20 06:00:02,239.81.105.222,2245,71.129.68.118,1641,59,GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1,200,3
```

수집 대상 테이블 컬럼은 seq, at, srcip, srcport, dstip, dstport, protocol, eventlog, eventcode, eventsize 이다. seq를 순차증가 컬럼으로 설정한다. 이 구조로 테이블을 생성하려면 mysql에서 다음의 질의를 수행한다.

```
mysql> create table odbc_seq_int_10 (seq int(9), at timestamp, srcip varchar(20), srcport int(6), dstip varchar(20)
```

테이블을 정상적으로 생성하고 나서 다음의 명령으로 MySQL 데이터베이스에 데이터를 입력한다.

```
mysql> load data infile '<File Path>' into table <Table Name> fields terminated by ',';
Query OK, 10 rows affected (0.00 sec)
Records: 10 Deleted: 0 Skipped: 0 Warnings: 0
```

MySQL 데이터베이스 실행 관련 내용은 MySQL 매뉴얼을 참고하라.

예제 (2) ODBC 설정

MySQL 데이터베이스 접속을 위해 ODBC 설정 파일을 작성한다.

```
[MYSQL]
Driver=MySQL
Server=<Database host address>
Port=<Database host port>
Database=<Name of default database to access>
UID=<User ID>
PWD=<User password>
```

unixODBC에서는 USER DATA SOURCES가 먼저 이용되므로 collector를 실행하는 사용자 홈 디렉토리에서 .odbc.ini파일에 위 내용을 작성한다.

예제 (3) ODBC 설정 검사

ODBC 설정을 정상적으로 진행 되었는지 확인하기 위해서 unixODBC의 isql프로그램을 이용한다. 설정한 DSN인 MYSQL 을 매개변수로 해서 다음과 같이 실행한다.

```
$ isql -v MYSQL
```

정상적으로 설치되고 설정이 되었다면 다음과 같은 결과를 얻을 수 있을 것이다.

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

isql을 통해서, 입력된 데이터를 조회해 볼 수 있다.

```
SQL> select * from odbc_seq_int_10;
+-----+---------------------+-----------------+---------+----------------+---------+----------+--------------------
| seq | at                  | srcip           | srcport | dstip          | dstport | protocol | eventlog
+-----+---------------------+-----------------+---------+----------------+---------+----------+--------------------
| 0   | 2015-05-20 06:00:00 | 16.194.51.72    | 6790    | 183.103.50.46  | 5281    | 20       | GET /twiki/bin/view
| 1   | 2015-05-20 06:00:02 | 96.40.75.42     | 11011   | 31.224.72.52   | 12069   | 55       | GET /twiki/bin/sear
| 2   | 2015-05-20 06:00:02 | 174.47.129.59   | 6032    | 96.40.75.42    | 6442    | 72       | GET /twiki/bin/edit
| 3   | 2015-05-20 06:00:02 | 153.199.166.54  | 4220    | 86.45.186.17   | 2245    | 1        | GET /twiki/bin/oops
| 4   | 2015-05-20 06:00:02 | 226.7.237.25    | 10805   | 50.230.44.173  | 179     | 70       | GET /twiki/bin/oops
| 5   | 2015-05-20 06:00:02 | 183.103.50.46   | 7175    | 96.128.212.177 | 7175    | 73       | GET /twiki/bin/view
| 6   | 2015-05-20 06:00:02 | 123.198.82.192  | 6784    | 63.214.191.124 | 10825   | 21       | GET /twiki/bin/view
| 7   | 2015-05-20 06:00:02 | 214.153.107.182 | 5562    | 85.183.139.166 | 1367    | 8        | GET /twiki/bin/oops
```

```
| 8   | 2015-05-20 06:00:02 | 245.13.24.17    | 7451    | 69.99.246.62   | 4497    | 20       | GET /twiki/bin/viev
| 9   | 2015-05-20 06:00:02 | 239.81.105.222  | 2245    | 71.129.68.118  | 1641    | 59       | GET /twiki/bin/viev
+-----+---------------------+-----------------+---------+----------------+---------+----------+--------------------
SQLRowCount returns 10
10 row(s) fetched.
SQL>
```

위와 같은 결과를 얻을 수 없었다면, unixODBC 와 DSN 명을 확인해 보라. 관련된 자세한 내용은 unixODBC 문서를 참고하라.

예제 (4) Collector 설정

위에서 사용한 쿼리, DSN, 사용자명과 패스워드를 이용하여 tpl 파일을 생성한다.
ODBC 모드 입력 방식에서는 데이터가 컬럼 별로 분리되어 제공되므로 REGEX_PATH가 필요하지 않다.

```
##############################################################################
# Copyright of this product 2013-2023,
# Machbase Corporation (Incorporation) or its subsidiaries.
# All Rights reserved
##############################################################################

#
# This file is for Machbase collector template file.
#

#####################################################################
# Collect setting
#####################################################################

COLLECT_TYPE=ODBC
ODBC_DSN=MYSQL <= Name of driver specified in "odbc.ini"
ODBC_QUERY="select * from sample_seq_int_10" <= Not required to input other queries except select <Columns> from <
ODBC_SEQ_COLUMN=seq <= The base column name that determines the order in the query results

#####################################################################
# Process setting
#####################################################################

#PREPROCESS_PATH=Python script file path

#####################################################################
# Output setting
#####################################################################

DB_TABLE_NAME = "sample_seq_int"
DB_ADDR = <Machbase Server Host>
DB_PORT = <Machbase Server Port>
DB_USER = "SYS"
DB_PASS = "MANAGER"

# 0: Direct insert
# 1: Prepared insert
# 2: Append
APPEND_MODE=2

# 0: None, just append.
# 1: Truncate.
# 2: Try to create table. If table already exists, warn it and proceed.
# 3: Drop and create.
CREATE_TABLE_MODE=2
```

위의 tpl파일을 이용하여 collector를 생성하고 시작한 다음 결과를 machsql 을 이용하여 확인해 보자.

```
Mach> select * from sample_seq_int limit 50;
COLLECTOR_TYPE COLLECTOR_ADDR
-----------------------------------------------------
COLLECTOR_ORIGIN COLLECTOR_OFFSET SEQ
-----------------------------------------------------------------------------------------------------------------------
AT SRCIP SRCPORT DSTIP DSTPORT PROTOCOL
```

```
-----------------------------------------------------------------------------------------------------------------------------
EVENTLOG EVENTCODE EVENTSIZE
-----------------------------------------------------------------------------------------------------------------------------
ODBC 192.168.122.1
MYSQL 9 9
1900-01-03 19:54:55 000:000:000 239.81.105.222 2245 71.129.68.118 1641 59
GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1 200 3853
ODBC 192.168.122.1
MYSQL 8 8
1900-01-03 19:54:55 000:000:000 245.13.24.17 7451 69.99.246.62 4497 20
GET /twiki/bin/view/Main/SpamAssassin HTTP/1.1 200 4081
ODBC 192.168.122.1
MYSQL 7 7
1900-01-03 19:54:55 000:000:000 214.153.107.182 5562 85.183.139.166 1367 8
GET /twiki/bin/oops/TWiki/RichardDonkin?template=oopsmore¶m1=1.2¶m2=1.2 HTTP/1
.1 200 11281
ODBC 192.168.122.1
MYSQL 6 6
1900-01-03 19:54:55 000:000:000 123.198.82.192 6784 63.214.191.124 10825 21
GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1 200 4140
ODBC 192.168.122.1
MYSQL 5 5
1900-01-03 19:54:55 000:000:000 183.103.50.46 7175 96.128.212.177 7175 73
GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1 200 4140
ODBC 192.168.122.1
MYSQL 4 4
1900-01-03 19:54:55 000:000:000 226.7.237.25 10805 50.230.44.173 179 70
GET /twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore¶m1=1.12¶m2=1.12
HTTP/1.1 200 11382
ODBC 192.168.122.1
MYSQL 3 3
1900-01-03 19:54:55 000:000:000 153.199.166.54 4220 86.45.186.17 2245 1
GET /twiki/bin/oops/TWiki/RichardDonkin?template=oopsmore¶m1=1.2¶m2=1.2 HTTP/1
.1 200 11281
ODBC 192.168.122.1
MYSQL 2 2
1900-01-03 19:54:55 000:000:000 174.47.129.59 6032 96.40.75.42 6442 72
GET /twiki/bin/edit/Main/WebSearch?t=1078669682 HTTP/1.1 401 12846
ODBC 192.168.122.1
MYSQL 1 1
1900-01-03 19:54:55 000:000:000 96.40.75.42 11011 31.224.72.52 12069 55
GET /twiki/bin/search/Main/SearchResult?scope=text®ex=on&search=Office%20*Locat
ions[notA-Za-z] HTTP/1.1 200 7771
ODBC 192.168.122.1
MYSQL 0 0
1900-01-03 19:54:55 000:000:000 16.194.51.72 6790 183.103.50.46 5281 20
GET /twiki/bin/view/TWiki/KlausWriessnegger HTTP/1.1 200 3848
[10] row(s) selected.
```

MySQL의 결과 출력과 역순으로 데이터가 출력 되는 것을 볼 수 있다.

사용자 지정 로그 수집

이 장은 사용자 정의 로그데이터의 정의와 이 데이터를 collector를 통해서 어떻게 수집할지에 대해서 기술한다.

목차

데이터 변환 개념

다음 그림은 collector가 동작하는 순서를 나타낸 것이다. 먼저, 원본 데이터를 읽어서 정규 표현식 파일을 이용하여 파싱한다. (이 정규 표현식 파일의 위치는 템플릿 파일에 기술한다.) 그리고 파싱된 데이터는 rgx 파일에 정의된 COL_LIST값을 이용하여 컬럼에 지정된다.

사용자 정의 로그데이터를 수집하기 위해서는 원본 데이터를 파싱하기 위한 정규 표현식을 생성하고, 템플릿 파일에 COL_LIST와 정규 표현식 파일을 기술하고, 템플릿을 이용하여 collector를 생성 및 실행하면 된다.

machregex

Collector로 데이터를 수집하기 위해서는 입력 데이터를 파싱하기 위한 정규 표현식을 생성해야 한다. 마크베이스는 작성한 정규 표현식이 원하는 입력 데이터를 정확히 파싱할 수 있는지를 검사하기 위한 도구인 machregex를 제공한다.

마크베이스는 SYSLOG, 아파치 웹서버의 ACCESS Log, 마크베이스 서버의 TRACE LOG를 파싱할 수 있는 정규 표현식 예제를 제공한다. 마크베이스는 정규 표현식을 지원하기 위해 PCRE (Perl Compatible Regular Expressions) 라이브러리를 이용하였다.

machregex 실행

```
[mach@localhost ~/mach_collector_home/bin]$ ./machregex
================================================================
     Machbase Collector Regex Utility
     Release Version 3.0.0.8634.official
     Copyright 2015, Machbase Inc. or its subsidiaries.
     All Rights Reserved.
================================================================

Usage> ./machregex Pattern NewlinePattern

Result file : machregex.ok machregex.err

<< APACHE access log >>
  => machregex "^([0-9.:]+)\\s([\\w.-]+)\\s([\\w.-]+)\\s(\\[[^\\[\\]]+\\])\\s\"((?:[^\"]|\")+)\"\\s(\\d{3})\\s(\\d+
((?:[^\"]|\")*)\"$" "^([0-9.:]+)\s" < DATA.LOG

<< MACH trace log >>
  => machregex "^\\[(\\d+[-]\\d+[-]\\d+\\s\\d+[:]\\d+[:]\\d+)+\\s([P][-]\\d+)+\\s([T][-]\\d+)+\\]\\s((?:[^\\0])*)$"

<< syslog >>
  => machregex "^(([a-zA-Z]+)\\s+([0-9]+)\\s+([0-9:]*))\\s(\\S*)\\s+((?:[^\\0])*)$" ".*" < DATA.LOG
```

machregex 실행 화면의 예제이다.

machregex 테스트

Syslog 데이터를 정규 표현식을 이용하여 machregex로 파싱 테스트한 내용이다.

```
[mach@localhost bin]$ machregex "^(([a-zA-Z]+)\\s+([0-9]+)\\s+([0-9:]*))\\s(\\S*)\\s+((?:[^\\0])*)$" ".*" </var/log
machregex "^(([a-zA-Z]+)\\s+([0-9]+)\\s+([0-9:]*))\\s(\\S*)\\s+((?:[^\\0])*)$" ".*" </var/log/syslog
Pattern => (^(([a-zA-Z]+)\s+([0-9]+)\s+([0-9:]*))\s(\S*)\s+((?:[^\0])*)$)
================================================================================
.............
================================================================================
SUCCESS[107] (rc=7)(Aug 19 18:17:01 localhost CRON[6553]: (root) CMD (   cd / && run-parts --report /etc/cron.hourl
)
  ALL (0:110) => [Aug 19 18:17:01 localhost CRON[6553]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
]
  0 (0:15) => [Aug 19 18:17:01]
  1 (0:3) => [Aug]
```

```
  2 (4:6) => [19]
  3 (7:15) => [18:17:01]
  4 (16:37) => [localhost]
  5 (38:110) => [CRON[6553]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
]
======================================================================
SUCCESS[107] (rc=7)(Aug 19 18:39:01 localhost CRON[6616]: (root) CMD (  [ -x /usr/lib/php5/maxlifetime ] && [ -x /u
[ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/php5/maxlifetime))
)
  ALL (0:232) => [Aug 19 18:39:01 localhost CRON[6616]: (root) CMD (  [ -x /usr/lib/php5/maxlifetime ] && [ -x /usr
[ -d /var/lib/php5 ] && /usr/lib/php5/sephp5/maxlifetime))
]
  0 (0:15) => [Aug 19 18:39:01]
  1 (0:3) => [Aug]
  2 (4:6) => [19]
  3 (7:15) => [18:39:01]
  4 (16:37) => [localhost]
  5 (38:232) => [CRON[6616]: (root) CMD (  [ -x /usr/lib/php5/maxlifetime ] && [ -x /usr/lib/php5/sessionclean ] &&
]
Summary : Success(107), Failure(0) <== It shows that all of them were successfully completed.
```

위 실행 예에서, machregex는 syslog text파일을 주어진 정규 표현식으로 파싱하여 6개의 토큰으로 분리하였다. 이 토큰중 0, 4, 5를 데이터베이스 입력으로 사용하기 위해서 템플릿 파일의 COL_LIST 변수를 사용하여 토큰과 데이터베이스 컬럼을 연결한다.

사용자 정의 템플릿 생성 예제

이 장에서는 예제 text log파일을 이용하여 이 파일에서 데이터를 수집하는 collector template를 생성하는 것을 다룬다.

test.log

입력 샘플 text 파일은 다음과 같다.

```
[2014-08-18 13:51:19] spiderman message-1 : This is the best machine data DBMS ever.
[2014-08-18 13:51:19] superman  message-2 : This is the best machine data DBMS ever.
[2014-08-18 13:51:33] spiderman message-3 : This is the best machine data DBMS ever.
[2014-08-18 13:51:33] superman  message-4 : This is the best machine data DBMS ever.
[2014-08-18 13:51:34] batman    message-5 : This is the best machine data DBMS ever.
[2014-08-18 13:52:34] superman  message-6 : This is the best machine data DBMS ever.
[2014-08-18 13:53:34] batman    message-7 : This is the best machine data DBMS ever.
[2014-08-18 13:54:31] superman  message-8 : This is the best machine data DBMS ever.
[2014-08-18 13:55:30] batman    message-9 : This is the best machine data DBMS ever.
[2014-08-18 13:56:44] spiderman message-10 : This is the best machine data DBMS ever.
[2014-08-18 13:57:59] superman  message-11 : This is the best machine data DBMS ever.
```

위 샘플 파일은 tm, user, msg의 3개 컬럼으로 변환될 수 있다. 각 컬럼의 데이터 타입은 각각 datetime, varchar(16), varchar(512)로 지정될 수 있다.

정규 표현식 생성 예제

정규 표현식 작성

`\[([0-9-: ]+)\]`: 첫번째, 대괄호로 싸인 날짜 데이터가 들어온다. 대괄호는 제외하고 내부의 숫자 값만 토큰으로 받아오기위해서 아래의 표현식을 사용한다.

`(\S+)`: 두번째, 유저이름 데이터가 들어오고, 공백을 제외한 문자열들을 입력받는다.

`([^\0]*)`: 세번째, 끝까지의 문자열을 입력받는다.

`\[([0-9-: ]+)\]\s(\S+)\s+([^\0]*)`: 세개의 토큰 사이에서 공백을 입력받도록 합친다.

`"\\[([0-9-: ]+)\\]\\s(\\S+)\\s+([^\\0]*)"`: 헬에서 문자열을 사용하기위해 더블 슬래쉬 처리를 한다.

`"^\\["`: 줄바꿈 정규표현식으로는 시간의 시작부분인 대괄호로 합니다.

정규 표현식 검증

```
[mach@localhost ~/mach_collector_home/bin]$ machregex "\\[([0-9-: ]+)\\]\\s(\\S+)\\s+([^\\0]+)" "\\[" <test.log
Pattern => (\[([0-9-: ]+)\]\s(\S+)\s+([^\0]+))
```

```
================================================================================
SUCCESS[2] (rc=4)([2014-08-18 13:51:19] spiderman message-1 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:51:19] spiderman message-1 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:51:19]
  1 (22:31) => [spiderman]
  2 (32:85) => [message-1 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[3] (rc=4)([2014-08-18 13:51:19] superman  message-2 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:51:19] superman  message-2 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:51:19]
  1 (22:30) => [superman]
  2 (32:85) => [message-2 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[4] (rc=4)([2014-08-18 13:51:33] spiderman message-3 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:51:33] spiderman message-3 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:51:33]
  1 (22:31) => [spiderman]
  2 (32:85) => [message-3 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[5] (rc=4)([2014-08-18 13:51:33] superman  message-4 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:51:33] superman  message-4 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:51:33]
  1 (22:30) => [superman]
  2 (32:85) => [message-4 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[6] (rc=4)([2014-08-18 13:51:34] batman    message-5 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:51:34] batman    message-5 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:51:34]
  1 (22:28) => [batman]
  2 (32:85) => [message-5 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[7] (rc=4)([2014-08-18 13:52:34] superman  message-6 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:52:34] superman  message-6 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:52:34]
  1 (22:30) => [superman]
  2 (32:85) => [message-6 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[8] (rc=4)([2014-08-18 13:53:34] batman    message-7 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:53:34] batman    message-7 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:53:34]
  1 (22:28) => [batman]
  2 (32:85) => [message-7 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[9] (rc=4)([2014-08-18 13:54:31] superman  message-8 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:54:31] superman  message-8 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:54:31]
  1 (22:30) => [superman]
```

```
  2 (32:85) => [message-8 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[10] (rc=4)([2014-08-18 13:55:30] batman    message-9 : This is the best machine data DBMS ever.
)
  ALL (0:85) => [[2014-08-18 13:55:30] batman    message-9 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:55:30]
  1 (22:28) => [batman]
  2 (32:85) => [message-9 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[11] (rc=4)([2014-08-18 13:56:44] spiderman message-10 : This is the best machine data DBMS ever.
)
  ALL (0:86) => [[2014-08-18 13:56:44] spiderman message-10 : This is the best machine data DBMS ever.
]
  0 (1:20) => [2014-08-18 13:56:44]
  1 (22:31) => [spiderman]
  2 (32:86) => [message-10 : This is the best machine data DBMS ever.
]
================================================================================
SUCCESS[11] (rc=4)([2014-08-18 13:57:59] superman  message-11 : This is the best machine data DBMS ever.)
  ALL (0:85) => [[2014-08-18 13:57:59] superman  message-11 : This is the best machine data DBMS ever.]
  0 (1:20) => [2014-08-18 13:57:59]
  1 (22:30) => [superman]
  2 (32:85) => [message-11 : This is the best machine data DBMS ever.]
Summary : Success(11), Failure(0)
```

test.rgx 생성

작성한 정규 표현식을 위 과정을 거쳐서 입력 파일을 정상적으로 파싱하는 지를 확인 한 후, 파싱에 문제가 없었다면 정규 표현식과 컬럼 바인딩을 위해서 rgx파일을 다음과 같이 작성한다. 이 파일은 `$MACHBASE_HOME/collector/samples/test.rgx` 에 작성한다.

```
##############################################################################
# Copyright of this product 2013-2023,
# Machbase Corporation (Incorporation) or its subsidiaries.
# All Rights reserved
##############################################################################

#
# This file is for Machbase trace collector regex file.
#

LOG_TYPE=custom

COL_LIST= (
     (
        REGEX_NO = 0
        NAME = tm
        TYPE = datetime
        SIZE = 8
        DATE_FORMAT="%Y-%m-%d %H:%M:%S"
         ),
     (
        REGEX_NO = 1
        NAME = user
        TYPE = varchar
        SIZE = 16
        USE_INDEX = 1
         ),
     (
        REGEX_NO = 2
        NAME = msg
        TYPE = varchar
        SIZE = 512
        USE_INDEX = 1
         )
)
```

```
REGEX="\[([0-9-: ]+)\]\s(\S+)\s+([^\0]+)"

END_REGEX="\["
```

test.tpl 생성

$MACHBASE_HOME/collector/custom.tpl 을 $MACHBASE_HOME/collector/test.tpl 의 이름으로 복사하고 그 파일을 아래와 같이 수정한다.

```
##############################################################################
# Copyright of this product 2013-2023,
# Machbase Corporation(Incorporation) or its subsidiaries.
# All Rights reserved
##############################################################################

#
# This file is for Machbase collector template file.
#

########################################################################
# Collect setting
########################################################################

COLLECT_TYPE=FILE
LOG_SOURCE=/home/mach/machbase_home/collector/samples/test.log

########################################################################
# Process setting
########################################################################

REGEX_PATH=/home/mach/machbase_home/collector/samples/test.tpl

########################################################################
# Output setting
########################################################################

DB_TABLE_NAME = "custom_table"
DB_ADDR       = "127.0.0.1"
DB_PORT       = 5656
DB_USER       = "SYS"
DB_PASS       = "MANAGER"

# 0: Direct insert
# 1: Prepared insert
# 2: Append
APPEND_MODE=2

# 0: None, just append.
# 1: Truncate.
# 2: Try to create table. If table already exists, warn it and proceed.
# 3: Drop and create.
CREATE_TABLE_MODE=2

Create and Execute a Collector
```

Collector 생성/실행

"myclt" collector를 생성하고 이를 실행한다.

```
Mach> create collector localhost.myclt from "/home/mach/mach_collector_home/collector/samples/test.tpl";
Created successfully.
Elapsed Time : 0.106
Mach>
Mach> alter collector localhost.myclt start;
Altered successfully.
```

Collector 디버깅

입력 데이터를 기록할 TESTTABLE이 생성되지 않았다.

```
Mach> select * from custom_table;
[ERR-02025 : Table CUSTOM_TABLE does not exist.]
```

Collector의 오류를 추적 파일에 기록하여 오류 해결을 하기 위해서 trace파일을 생성한다. trace파일의 생성하기 위해 다음의 명령을 수행한다.

```
Mach> alter collector localhost.myclt stop;
Altered successfully.
Mach> alter collector localhost.myclt start trace;
Altered successfully.
```

Trace Log 를 통한 문제 탐색/해결

Collector실행시 오류가 발생한 경우, `$MACHBASE_HOME/trc/machbase.trc` 파일을 조사하면 데이터베이스 실행 오류를 찾을 수 있다. collector에서 실행 오류가 발생한 경우에는 collector를 TRACE모드로 실행해야 한다.

```
[2016-03-13 23:44:35 P-29741 T-139982693979904][INFO] PREPARE Error [create table custom_table ( collector_type va
collector_offset long, tm datetime, user varchar(16), msg varchar(512))] (100007DA:Error in parse (syntax): near t
```

위 메시지를 살펴보면 테이블 생성 질의가 실패했고, 그 원인은 컬럼명으로 설정된 user가 built-in 키워드여서 컬럼명으로 쓸 수가 없기 때문이다. 따라서 rgx 파일의 COL_LIST 부분에서 user컬럼을 myuser로 변경하고 콜렉터를 다시 실행한다.

```
A partial contents from "test.rgx"
...........

COL_LIST= (
     (
        REGEX_NO = 0
        NAME = tm
        TYPE = datetime
        SIZE = 8
        DATE_FORMAT="%Y-%m-%d %H:%M:%S"
         ),
     (
        REGEX_NO = 1
        NAME = myuser   <== 수정된 부분
        TYPE = varchar
        SIZE = 16
        USE_INDEX = 1
         ),
     (
        REGEX_NO = 2
        NAME = msg
        TYPE = varchar
        SIZE = 512
        USE_INDEX = 1
         )
)
..................
```

실행/결과 확인

수정한 rgx파일로 재실행한다.

```
Mach> alter collector localhost.myclt stop; <== Stop the TRACE mode.
Altered successfully.
Mach> alter collector localhost.myclt start; <== Execute it again in a normal mode after the modification
Altered successfully.
```

정상적으로 실행되었다면 콜렉터가 데이터를 저장한 테이블 내용을 조회할 수 있다.

```
Mach> select tm, myuser, msg from custom_table;
```

```
tm                              myuser
-------------------------------------------------
msg
-------------------------------------------------------------------------------
2014-08-18 13:57:59 000:000:000 superman
message-11 : This is the best machine data DBMS ever.

2014-08-18 13:56:44 000:000:000 spiderman
message-10 : This is the best machine data DBMS ever.

2014-08-18 13:55:30 000:000:000 batman
message-9 : This is the best machine data DBMS ever.

2014-08-18 13:54:31 000:000:000 superman
message-8 : This is the best machine data DBMS ever.

2014-08-18 13:53:34 000:000:000 batman
message-7 : This is the best machine data DBMS ever.

2014-08-18 13:52:34 000:000:000 superman
message-6 : This is the best machine data DBMS ever.

2014-08-18 13:51:34 000:000:000 batman
message-5 : This is the best machine data DBMS ever.

2014-08-18 13:51:33 000:000:000 superman
message-4 : This is the best machine data DBMS ever.

2014-08-18 13:51:33 000:000:000 spiderman
message-3 : This is the best machine data DBMS ever.

2014-08-18 13:51:19 000:000:000 superman
message-2 : This is the best machine data DBMS ever.

2014-08-18 13:51:19 000:000:000 spiderman
message-1 : This is the best machine data DBMS ever.

[11] row(s) selected.
```

Collector 전처리 프레임워크

목차

마크베이스 collector는 로그 데이터를 수집하여 분석하고 마크베이스 서버에 전송한다.

데이터 수집 및 분석 기능 외에 추가적인 데이터 처리를 위해, 마크베이스 collector는 python 을 이용한 데이터 전처리 프레임워크를 제공한다.

전처리를 위한 환경변수 설정

전처리 프레임워크로 python 2.6 버전을 사용한다. 이 버전의 python은 마크베이스 서버와 같이 설치된 것을 사용하는 것을 추천한다.

설치된 python은 `$MACH_COLLECTOR_HOME/webadmin/flask/Python/bin` 경로에 있다.
Python 라이브러리 추가 설치를 위한 python 실행도 위 디렉토리에서 실행해야 기존에 설치되어 있는 다른 버전의 python과 충돌을 방지할 수 있다.

> ⓘ 마크베이스 collector와 같이 제공된 python을 기본으로 사용하려면 PATH 환경변수를 정확히 설정하고, USER_PREPROCESS_LIB_PATH를 설정하여야 한다. USER_PREPROCESS_LIB_PATH에 경로를 추가로 등록할 때 경로 값의 분리를 위해 경로들 사이에 ":" 문자를 넣어줘야 한다.

전처리 순서

로그 데이터를 변환 및 조작하기 위한 전처리 실행 순서를 기술한다.

1. 메시지 전처리

원본 로그 데이터 파일에 데이터가 입력되면 각 로그 데이터는 로그 단위들로 분리된다.

가령 로그 데이터를 origin_msg라고 하자. 각 origin_msg는 한번에 한 단계씩 처리 과정을 거친다.

- 예를 들어 입력된 첫번째 메시지가 **"Aug 19 15:37:12 localhost NetworkManager[1340]: (eth1): bringing up device."** 라고 하면, 입력된 origin_msg는 정규 표현식에 의해 토큰들로 분리된다. 이를 메시지 파싱이라고 한다.
- 메시지 파싱 이전에 origin_msg를 전처리 할 수 있다.
- 전처리 스크립트를 이용해 origin_msg를 변경하는 경우에는, 변경된 결과 메시지가 파싱 가능한 메시지여야 한다.

2. 컬럼 전처리

로그 메시지를 파싱한 이후, 결과 토큰값들이 생성된다. 아무런 처리 과정이 없다면 이 값이 데이터베이스에 저장된다.

파싱된 토큰들을 데이터베이스에 전달하기 전에 두번째 단계의 전처리 과정을 실행할 수 있다.
이때 rgx파일에 기술된 필드명을 이용하여 변경하거나 이를 이용할 수 있다. rgx파일에 기술된 데이터형과 다른 유형으로 토큰을 변경하면 에러가 발생할 수 있다.

자세한 내용은, 아래 예제 스크립트를 참조하라.

전처리 스크립트

전처리 스크립트는 Python 으로 작성하여야 한다. 쉽게 사용하려면 "custom.py"파일을 원하는 형태로 변경하는 것을 추천한다.

```
PRS_SUCCESS       = ( 0, None )
PRS_SUCCESS_INFO  = ( 1, "Info Msg")
PRS_SUCCESS_SKIP  = ( 2, None )
PRS_FAILURE       = (-1, "Error Msg" )

class mach_preprocess:
    def __init__(self):
        return
    def mach_msg_preprocess(self, dict):
        return PRS_SUCCESS;
    def mach_column_preprocess(self, dict):
        return PRS_SUCCESS;
    def __del__(self):
        return
```

결과값 정의

전처리 스크립트의 실행 결과를 collector에 전달하기 위해서 리턴값을 이용한다. collector가 전처리 스크립트를 실행한 이후에 참조하는 결과값은 (code, message)의 튜플 (tuple) 유형이다.

```
PRS_SUCCESS       = ( 0, None )
PRS_SUCCESS_INFO  = ( 1, "Info Msg")
PRS_SUCCESS_SKIP  = ( 2, None )
PRS_FAILURE       = (-1, "Error Msg" )
```

여기서 PRS_SUCCESS, PRS_SUCCESS_INFO, PRS_SUCCESS_SKIP, PRS_FAILURE는 collector가 참조하는 결과값이다.

- 결과값이 PRS_SUCCESS인 경우에는, collector는 정상적으로 데이터를 입력한다.
- 결과값이 PRS_SUCCSS_INFO인 경우에는, 데이터를 정상적으로 처리하고 전달된 메시지를 trc 파일에 기록한다.
- 결과값이 PRS_SUCCESS_SKIP인 경우에는, 해당 데이터를 버리고 새로운 데이터 처리를 시작한다.
- 결과값이 PRS_FAILURE인 경우에는, 에러 메시지를 trc 파일에 기록하고 다음 메시지를 처리한다.

데이터 처리의 제어를 위해서는 PRS_SUCCESS, PRS_SUCCESS_SKIP 결과값을 이용하고, 메시지를 trc에 남기기 위해서는 PRS_SUCCESS_INFO나 PRS_FAILURE 결과값을 이용하면 된다.

클래스 정의

마크베이스 collector는 python언어로 작성된 사전 정의된 클래스의 함수를 호출하여 전처리를 수행한다.

아래의 예제를 보면 클래스의 각 함수와 "dict" 매개변수와 리턴값에 대해서 알 수 있다.

> ⓘ 클래스 명이나 함수 이름을 바꾸면 실행되지 않는다. 따라서 작성시 유의하여야 한다.

```
class mach_preprocess:
    def __init__(self):
        return
    def mach_msg_preprocess(self, dict):
        return PRS_SUCCESS;
    def mach_column_preprocess(self, dict):
        return PRS_SUCCESS;
    def __del__(self):
        return
```

사전 정의된 클래스명은 "mach_preprocess"이다.

- 매개변수는 메서드를 호출할 때 "self" 인스턴스로 전달된다.
- __init__과 __del__은 python언어의 기본 객체 생성자/제거자이다.
  따라서 __init__은 collector의 프로세스가 생성될 때 호출되고, __del__은 collector가 종료될 때 호출된다.
- __init__에서 변수들을 초기화하고, __del__에서 할당한 자원을 해제할 수 있다.
  이 두 메서드는 리턴값이 없다.

데이터 전처리에 호출되는 메서드는 "mach_msg_preprocess"와 "mach_column_preprocess"이다.
각 메서드의 설명은 아래와 같다.

mach_msg_preprocess

이 메서드는 입력 메시지가 토큰으로 분리되기 전에 호출된다.

메시지를 파싱하기 전에 실행되므로, 전달되는 값은 컬렉터 관련 메타 데이터와 원본 메시지인 "origin_msg"이다.
컬렉터 메타 데이터는 테이블 이름, 컬렉터 유형, 현재 실행 중인 컬렉터의 이름과 오프셋 (offset) 이다. 이 정보들은 참조 정보로 제공되어 변경하더라도 컬렉터에 반영되지는 않는다.

"origin_msg"는 변경되면 컬렉터에 변경사항이 반영된다.
rgx파일에 설정된 정규표현식을 통과하지 못하도록 메시지를 변경하면, 이후 파싱 과정에서 오류가 발생할 수 있다.

키	설명	변경 값 반영 가능 여부
table_name	테이블 이름	X
collect_type	컬렉터 유형	X
collector_name	현재 실행 중인 컬렉터 이름	X
data_source	데이터 원본이 되는 소스 파일 경로	X

키	설명	변경 값 반영 가능 여부
origin_msg	소스 파일의 원본 데이터 (Raw data) 메시지	O

필요 없는 메시지는 PRS_SUCCESS_SKIP 을 반환시켜서, 이후 파싱 과정을 생략하여 처리를 빠르게 할 수 있다.
필요 없는 메시지를 이 단계에서 파악할 수 있다면, 먼저 PRS_SUCCESS_SKIP 으로 처리하는 것이 좋다.

mach_column_preprocess

이 메서드는 입력 메시지를 파싱한 이후 토큰으로 분해된 값을, 데이터베이스에 입력하기 전에 호출된다.
"mach_msg_preprocess"와 마찬가지로, 전달된 메타데이터는 컬렉터에 반영되지 않는다.

키	설명	변경 값 반영 가능 여부
table_name	테이블 이름	X
collect_type	컬렉터 이름	X
collect_name	현재 실행 중인 컬렉터 이름	X
data_source	데이터 원본이 되는 소스 파일 경로	X
origin_msg	소스 파일의 원본 데이터 (Raw data) 메시지	X
column_name	n번째 컬럼 토큰	O

예제 스크립트

기본으로 제공되는 예제는 syslog파일에 대한 것이며 `$MACH_COLLECTOR_HOME/collector` 디렉토리에 있다.

예제 템플릿인 syslog.tpl 에서 전처리를 수행하는 방법을 살펴보자.

```
##############################################################################
Copyright of this product 2013-2023,
Machbase Inc. or its subsidiaries.
All Rights reserved
##############################################################################
#
This file is for Machbase collector template file.
#
#########################################################################
Collect setting
#########################################################################

COLLECT_TYPE=FILE
LOG_SOURCE=/var/log/syslog

#########################################################################
Process setting
#########################################################################

REGEX_PATH=syslog.rgx
PREPROCESS_PATH=script_path

#########################################################################
Output setting
#########################################################################
DB_TABLE_NAME = "syslogtable"
DB_ADDR = "127.0.0.1"
DB_PORT = 5656
DB_USER = "SYS"
DB_PASS = "MANAGER"
#
0: Direct insert
1: Prepared insert
2: Append
APPEND_MODE=2
#
0: None, just append.
1: Truncate.
```

```
2: Try to create table. If table already exists, warn it and proceed.
3: Drop and create.
CREATE_TABLE_MODE=2
```

전처리 스크립트 파일의 위치를 지정하기 위해서, PREPROCESS_PATH를 tpl 파일에 설정한다. 경로명은 절대 경로 (/로 시작되는 경로)를 지정하거나 $MACH_COLLECTOR_HOME/collector/preprocess의 기본 경로(파일명만 지정한 경우)가 된다.

SKIP

입력 메시지를 검사하여 특정한 단어가 있는 경우에 이를 입력하지 않는 스크립트이다.

컬렉터 템플릿 파일에 PREPROCESS_PATH=skip.py를 설정하면 된다.
경로명을 지정하지 않았으므로, $MACH_COLLECTOR_HOME/collector/preprocess/ 디렉토리에 그 파일을 작성해야 한다.

```
PRS_SUCCESS       = ( 0, None )
PRS_SUCCESS_INFO  = ( 1, "Info Msg" )
PRS_SUCCESS_SKIP  = ( 2, None )
PRS_FAILURE       = (-1, "Error Msg" )

class mach_preprocess:
    def __init__(self):
        return
    def mach_msg_preprocess(self, dict):
        if dict['origin_msg'].find("CMD") is not -1: <== 문자열 검색 "CMD"
            return PRS_SUCCESS_SKIP <== 문자열에 "CMD"가 없으면 생략
        else:
            return PRS_SUCCESS;
    def mach_column_preprocess(self, dict):
        return PRS_SUCCESS;
    def __del__(self):
        return

#Test code
if __name__ == "__main__":
    pre_obj = mach_preprocess()
    dict = {"origin_msg":"Jul 16 07:09:01 mach-Precision-T1700 CRON[1220]: (root) CMD (  [ -x /usr/lib/php5/maxlife
[ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/php5/maxlifetime))"}
    print pre_obj.mach_msg_preprocess(dict)

    dict = {"origin_msg":"Jul 16 07:39:31 mach-Precision-T1700 cracklib: no dictionary update necessary."}
    print pre_obj.mach_msg_preprocess(dict)
```

파싱 과정을 실행하지 않은 "mach_msg_preprocess" 메서드에서 "origin_msg"매개변수에 "CMD" 문자열이 있는지 검사하여 있다면 그 메시지를 스킵하도록 설정한 예제이다.
"if __name__ == "__main__"이후의 소스라인은 스크립트가 정상적으로 동작하는지 테스트하기 위해서 작성한 코드이다.

관련 내용은, 아래 '전처리 스크립트 테스트' 부분을 참고하라.

REPLACE

파싱을 거친 이후에 msg 칼럼에 "CRON"이라는 문자열이 있는 경우, 그것을 "cron-exectue"문자열로 변환하는 예제이다.
이 또한 $MACH_COLLECTOR_HOME/collector/preprocess/ 디렉토리의 replace.py 파일을 tpl 파일에서 다음과 같이 지정하면 실행된다.

sample.tpl

```
PREPROCESS_PATH=replace.py
```

```
PRS_SUCCESS       = ( 0, None )
PRS_SUCCESS_INFO  = ( 1, "Info Msg" )
PRS_SUCCESS_SKIP  = ( 2, None )
PRS_FAILURE       = (-1, "Error Msg" )

class mach_preprocess:
    def __init__(self):
        return
    def mach_msg_preprocess(self, dict):
        return PRS_SUCCESS;
    def mach_column_preprocess(self, dict):
        dict['msg'] = dict['msg'].replace("CRON", "cron-execute") <== "CRON"을 "cron-execute"로 치환
        return PRS_SUCCESS;
```

```
    def __del__(self):
        return

#Test code
if __name__ == "__main__":
    pre_obj = mach_preprocess()
    dict = {"tm":"Jul 16 07:39:01", "host":"mach-Precision-T1700", "msg":"CRON[1377]: (root) CMD (  [ -x /usr/lib/p
[ -x /usr/lib/php5/sessionclean ] && [ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/ph
    (code, msg) = pre_obj.mach_column_preprocess(dict);
    if code >= 0:
        print dict
    else:
        print msg
```

입력된 원본 메시지는 파싱 과정을 거쳐 토큰으로 분리된다.이 토큰은 mach_column_ preprocess메서드에서 처리할 수 있다.
위의 예제는 "CRON"문자열을 "cron-execute"로 변환하는 예제이다. "if name == "__main__"이하의 코드는 스크립트 실행을 디버깅하기 위한 것이다.

TRACE

TRACE 스크립트는 입력 데이터를 "mach_msg_preprocess" 및 "mach_column_preprocess"메서드에서 파일에 기록하는 것이다.
PREPROCESS_PATH=trace.py를 tpl 파일에 추가하고, 해당 스크립트 파일을 `$MACH_COLLECTOR_HOME/collector/preprocess` 디렉토리에 작성해 두면 동작한다.

```
PRS_SUCCESS       = ( 0, None )
PRS_SUCCESS_INFO  = ( 1, "Info Msg" )
PRS_SUCCESS_SKIP  = ( 2, None )
PRS_FAILURE       = (-1, "Error Msg" )

class mach_preprocess:
    def __init__(self):
        self.msg_file = open("/tmp/msg.log", 'a') <== 파일 지정
        self.column_file = open("/tmp/column.log", 'a')
        return
    def mach_msg_preprocess(self, dict):
        self.msg_file.write(str(dict)+"\n"); <== 인자값을 파일에 기록
        self.msg_file.write("\n");
        return PRS_SUCCESS;
    def mach_column_preprocess(self, dict):
        self.column_file.write(str(dict)+"\n");
        self.column_file.write("\n");
        return PRS_SUCCESS;
    def __del__(self):
        self.msg_file.close() <== 파일 닫기
        self.column_file.close()
        return

#Test code
if __name__ == "__main__":
    pre_obj = mach_preprocess()
    dict = {"origin_msg":"Jul 16 06:39:01 mach-Precision-T1700 CRON[1149]: (root) CMD (  [ -x /usr/lib/php5/maxlife
[ -x /usr/lib/php5/sessionclean ] && [ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/ph
    pre_obj.mach_msg_preprocess(dict)
    dict = {"origin_msg":"Jul 16 06:39:01 mach-Precision-T1700 CRON[1149]: (root) CMD (  [ -x /usr/lib/php5/maxlife
[ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/php5/maxlifetime))", "tm":"Jul 16 06:39
"msg":"CRON[1149]: (root) CMD (  [ -x /usr/lib/php5/maxlifetime ] && [ -x /usr/lib/php5/sessionclean ] && [ -d /var
/usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/php5/maxlifetime))"}
    pre_obj.mach_column_preprocess(dict)
```

init 및 del 이 시작 및 종료시에 실행되는 점을 이용하여 collector 실행시 msg_file, column_file 객체를 생성 및 개방하며, 종료시 각 파일을 닫도록 할 수 있다.

이 두개의 변수는 객체의 다른 메서드에서 접근할 수 있다. if __name__ == "__main__" 이하의 코드는 스크립트가 정상적으로 동작하는지 테스트하는 코드이다.

ODBC

아래의 ODBC 스크립트는 검색 키를 데이터베이스에서 검색하여 검색 키가 입력 메시지에 존재한다면, 그 값을 지정된 테이블에 입력하는 예제이다.

이 예제를 동작시키는 방법은 **PREPROCESS_PATH** 값을 템플릿 파일에 지정하는 것으로 기존과 동일하다.

이 예제에서 pypyodbc가 사용되었다. 컬렉터 스크립트에서 ODBC를 사용하려면 pypyodbc를 사용하여야 한다.

> ⓘ import 된 pypyodbc는 기본 모듈이 아니므로 미리 설치할 필요가 있다.

$MACH_COLLECTOR_HOME/webadmin/flask/Python/bin/python 경로에서 pypyodbc를 설치하여야 한다.
설치 경로는 $MACH_COLLECTOR_HOME/webadmin/flask/Python/lib/python2.7/site-packages/pypyodbc-1.3.3-py2.7.egg 이다.

그 다음 임포트 모듈의 경로를 설정하여야 한다. 경로 설정은 두 가지 방법이 있다.

- 첫 번째 방법은, sys module의 경로를 수정하거나 collector에게 모듈 경로를 제공하는 것이다.
- 두 번째 방법은, 환경 변수 USER_PREPROCESS_LIB_PATH 를 설정하는 것이다.

```
export USER_PREPROCESS_LIB_PATH=$MACH_COLLECTOR_HOME/webadmin/flask/Python/lib/python2.7/site-packages/pypyoc
```

```
import pypyodbc

PRS_SUCCESS       = ( 0, None )
PRS_SUCCESS_INFO  = ( 1, "Info Msg" )
PRS_SUCCESS_SKIP  = ( 2, None )
PRS_FAILURE       = (-1, "Error Msg" )

class mach_preprocess:
    def __init__(self):
        self.con = pypyodbc.connect("DSN=MYSQL") <== 미리 선언된 MySQL DNS값을 입력.
        self.cursor = self.con.cursor()
        self.table_name = "error_msg"
        self.test_data_make(); <== 랜덤 데이터 생성.
        return
    def mach_msg_preprocess(self, dict):
        return PRS_SUCCESS;
    def mach_column_preprocess(self, dict):
        result = self.cursor.execute("select code, msg from %s where code = %d"%(self.table_name, int(dict['code_ty
        if result is not None: <== 결과값이 존재할 때.
            dict['code_type'] = result.fetchall()[0][1] <== 관련 데이터 치환.
        else:
            print "failure "+str(dict)
        return PRS_SUCCESS;
    def __del__(self):
        self.cursor.close()
        self.con.close()
        return

    #for test
    def test_data_make(self):
        self.table_check("create table %s (code integer, msg varchar(255))");
        return
    def table_check(self, query):
        self.tables = self.cursor.tables().fetchall()
        self.table_list = []
        for (db, user, table, info, none) in self.tables:
            self.table_list.append(table.upper())
        if self.table_name.upper() in self.table_list: <== 이미 테이블이 존재할 경우, 테이블을 지우고 새로 생성.
            self.cursor.execute("drop table %s"%self.table_name)
        self.cursor.execute(query%self.table_name);
        self.insert_error_msg()
        self.cursor.commit()
        return
    def insert_error_msg(self): <== 코드와 메세지를 치환.
        error = ((0, "SUCCESS"), (1, "SUCCESS_WITH_INFO"), (-1, "FAILURE"))
        for (code, msg) in error:
            self.cursor.execute("insert into %s values ( %d, '%s')"%(self.table_name, code, msg))
        return

# Test code
if __name__ == "__main__":
    pre_obj = mach_preprocess()
    pre_obj.test_data_make()
    dict = {"tm":"Jul 16 07:39:01","host":"mach-Precision-T1700","msg":"CRON[1377]: (root) CMD (  [ -x /usr/lib/php
[ -x /usr/lib/php5/sessionclean ] && [ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/ph
    pre_obj.mach_column_preprocess(dict)
    print dict
    dict = {"tm":"Jul 15 11:31:54","host":"mach-Precision-T1700","msg":"NetworkManager[1340]: <error> [1405391514.2
nm_system_iface_get_flags(): (unknown): failed to get interface link object","code_type":"0"}
    pre_obj.mach_column_preprocess(dict)
```

```
    print dict
    dict = {"tm":"Jul 15 11:31:54","host":"mach-Precision-T1700","msg":"NetworkManager[1340]: <warn> sysctl: faile
open '/proc/sys/net/ipv6/conf/eth1/use_tempaddr': (2) No file in directory","code_type":"1"}
```

전처리 스크립트 테스트

위 ODBC예제에서 사용할 pypyodbc 모듈을 지정된 환경변수에서 로딩할 수 있다. 새로운 스크립트를 작성한 이후, 스크립트가 정확히 동작하는지 확인해야 한다.

스크립트 테스트를 위한 방법으로, 직접 실행 방법과 간접 실행 방법이 있다.
직접 실행 방법은 스크립트가 테스트를 직접 실행하는 것이며, 간접 실행 방법은 그 스크립트를 import 하여 테스트 하는 것이다.

직접 실행

예제 전처리 스크립트 하단에 추가되어 있는 테스트 코드들은 전처리 클래스 객체를 직접 호출하여 결과를 확인한다. 아래 예제는 skip.py 스크립트의 테스트 코드이다.

```
if __name__ == "__main__":
    pre_obj = mach_preprocess()
    dict = {"origin_msg":"Jul 16 07:09:01 mach-Precision-T1700 CRON[1220]: (root) CMD (  [ -x /usr/lib/php5/maxlife
[ -x /usr/lib/php5/sessionclean ] && [ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/ph
    print pre_obj.mach_msg_preprocess(dict)

    dict = {"origin_msg":"Jul 16 07:39:31 mach-Precision-T1700 cracklib: no dictionary update necessary."}
    print pre_obj.mach_msg_preprocess(dict)
```

테스트 스크립트는 "__name__ == "__main__":" 으로 시작한다.

Python 인터프리터로 이 스크립트를 바로 실행할 경우 __name__변수를 __main__으로 설정하므로 테스트 코드를 실행한다.
collector에서 호출될 경우에는 위 테스트 코드가 실행되지 않는다.

테스트 코드의 실행 과정을 살펴보면 먼저 mach_process() 함수의 호출로 프리프로세스 객체인 pre_obj를 생성한다.
이 때, __init__메서드가 호출된다. mach_msg_preprocess에 전달되는 매개변수인 dict를 설정한 후, 메서드를 호출하여 테스트를 수행한다.
dict 값을 개발자가 설정해야 하지만, 실제 스크립트에 전달되는 값을 알기 어려운 경우, "trace.py" 스크립트를 이용하여 메서드에 전달되는 값을 얻어서 테스트 할 수 있다.

trace.py 스크립트가 생성하는 데이터는 /tmp/msg.log 및 /tmp/column.log에 기록된다.

간접 실행

이미 생성한 전처리 스크립트를 import 를 통해서 읽어들여 실행하는 방법이다.

```
import skip <== 이미 작성된 script를 import.

if __name__ == "__main__":
    pre_obj = skip.mach_preprocess() <== 클래스 생성 시, mach_preprocess 함수 호출.
    dict = {"origin_msg":"Jul 16 07:09:01 mach-Precision-T1700 CRON[1220]: (root) CMD (  [ -x /usr/lib/php5/maxlife
[ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/php5/maxlifetime))"}
    print pre_obj.mach_msg_preprocess(dict)

    dict = {"origin_msg":"Jul 16 07:39:31 mach-Precision-T1700 cracklib: no dictionary update necessary."}
    print pre_obj.mach_msg_preprocess(dict)
```

MACHCOLLECTORADMIN

machcollectoradmin 은 Collector Manger 를 관리하는 도구이다.

목차

옵션 목록

```
[mach@localhost ~]$ machcollectoradmin --help
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
<< Available option lists >>
  -u, --startup                          Startup collectormanager.
  -s, --shutdown                         Shutdown collectormanager.
  -d, --destroy                          Destroy meta data.
  -k, --kill                             Terminate collectormanager.
      --showcollectors                   Show collector list.
      --showservers                      Show server list.
  -h, --help                             Print collector argument guide.
      --createcollector=collector_name   Create collector.
      --dropcollector=collector_name     Drop collector.
      --startcollector=collector_name    Start collector.
      --stopcollector=collector_name     Stop collector.
  -m, --template=template_path           Set template path. Template path is rec
  -t, --trace=[0,1]                      Set trace option. Start collector mode
      --createserver=host:port           Manager registers itself to the server.
      --managername=managername          Specify a name of the manager. If not s
```

Collector Manager 실행

Collector Manager 는 직접 실행할 수 없고, machcollectoradmin을 통해서 실행해야 한다.

```
[mach@localhost ~]$ machcollectoradmin --startup
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
Waiting for collectormanager start.
Collectormanager started successfully.
```

Collector Manager 종료

Collector Manager 를 중지하면 그 collectormanager가 관리하는 collector프로세스도 동시에 중지된다.

```
[mach@localhost ~]$ machcollectoradmin --shutdown
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved
------------------------------------------------------------------
CollectorManager server shutdown successfully.
```

Collector Manager 메타 정보 삭제

Collector Manager 가 관리하는 collector, 데이터베이스 서버 등에 대한 정보를 삭제한다. Collector Manager 가 실행중이면 이 명령은 오류로 처리된다.

```
[mach@localhost ~]$ machcollectoradmin --destroy
------------------------------------------------------------------
machcollector administration tool
Release Version - x.x.x.official
Copyright 2014, Machbase Inc. or its subsidiaries
All Rights Reserved.
------------------------------------------------------------------
Destroyed meta data successfully.
```

Collector Manager 강제 종료

Collector Manager 가 shutdown과정을 대기하지 않고 즉시 중지한다.

```
[mach@localhost ~]$ machcollectoradmin --kill
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
Waiting for collectormanager terminated.
Collectormanager terminated successfully.
```

마크베이스 서버 등록

Collector Manager 에 연결된 마크베이스 데이터베이스 서버를 설정한다. 이후 machsql 을 통한 관리가 가능하다.

```
[mach@localhost ~]$ machcollectoradmin --createserver=127.0.0.1:5757
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
Created the server successfully.
```

Collector Manager 이름 지정

마크베이스 서버에 Collector Manager 를 등록할 때, 등록하는 Collector Manager 의 이름을 설정할 수 있다.

```
[mach@localhost ~]$ machcollectoradmin --createserver=127.0.0.1:5757 --managername=mach_manager
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
Created the server successfully.
```

등록된 마크베이스 서버 목록 출력

Collector Manager 에 등록된 데이터베이스 서버 리스트를 표시한다.

```
[mach@localhost ~]$ machcollectoradmin --showservers
```

```
----------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
----------------------------------------------------------------
ID   1
NAME 192.168.0.34:5757
ADDR 192.168.0.34
PORT 5757
```

Collector 생성

Collector Manager 가 관리하는 collector 를 생성한다.

collector 를 생성할 때, 이름과 템플릿 파일의 경로를 필수적으로 입력받는다.

collector의 이름은 첫 번째 인자로 입력 받고, 템플릿 경로는 -m 또는 --template 옵션으로 지정해야 한다.

```
[mach@localhost ~]$ machcollectoradmin --createcollector=syslog --template=/home/hanchi/work/nfx/machbase_home/coll
----------------------------------------------------------------
machcollector administration tool
Release Version - x.x.x.official
Copyright 2014, Machbase Inc. or its subsidiaries
All Rights Reserved.
----------------------------------------------------------------
Created the collector successfully.
```

템플릿 파일 지정

collector 를 생성할 때, collector설정 파일인 템플릿 파일의 경로를 지정한다.
상대 경로인 경우 `$MACHBASE_HOME/collector/` 디렉토리를 기준으로 한다.

```
[mach@localhost ~]$ machcollectoradmin --createcollector=syslog --template=syslog.tpl
----------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
----------------------------------------------------------------
Created the collector successfully.
```

Collector 실행

등록된 collector 프로세스를 실행한다. 실행 시 collector 이름을 명시해야 한다.

```
[mach@localhost ~]$ machcollectoradmin --startcollector=syslog
----------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
----------------------------------------------------------------
Started the collector successfully.
```

Collector 로그 메세지 설정

사용자가 collector 실행 시에 로그 메시지를 생성하도록 한다.

값을 1로 설정하면 생성되고 0으로 설정하면 생성하지 않는다.

```
[mach@localhost ~]$ machcollectoradmin --startcollector=syslog --trace=1
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
Started the collector successfully.
```

Collector 중지

실행중 인 collector 를 중지시킨다.

```
[mach@localhost ~]$ machcollectoradmin --stopcollector=syslog
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
Stopped the collector successfully.
```

Collector 제거

collector 에 관련된 메타 정보를 삭제한다. 실행 시 collector 의 이름을 명시해야 한다.

```
[mach@localhost ~]$ machcollectoradmin --dropcollector=syslog
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbasae Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
Dropped the collector successfully.
```

Collector 리스트 출력

등록된 collector의 리스트를 표시한다.

```
[mach@localhost ~]$ machcollectoradmin --showcollectors
------------------------------------------------------------------
     machcollector administration tool
     Release Version - x.x.x.official
     Copyright 2014, Machbase Inc. or its subsidiaries
     All Rights Reserved.
------------------------------------------------------------------
ID                  1
NAME                  SYSLOG
TEMPLATE_PATH       syslog.tpl
COLLECT_TYPE        FILE
SOURCE_FILE         /var/log/syslog
LOG_TYPE            syslog
PREPROCESS_PATH
REGEX_PATH          syslog.rgx
REGEX               (([a-zA-Z]+)\s+([0-9]+)\s+([0-9:]*))\s(\S+)\s+([^\n]+)
END_REGEX           \n
LANGUAGE            UTF-8
SLEEP_TIME          1000
PROCESS_BYTE        0
PROCESS_RECORD      0
```

```
PREV_PROCESS_BYTE   0
PREV_PROCESS_RECORD 0
RUN_FLAG            0
```

원격 Collector Node 관리

이 장은 마크베이스 collector 를 마크베이스 서버를 통해서 관리하는 방법을 기술한다.

목차

Fixed Table

마크베이스 서버에 있는 Fixed table 을 조회해, collector 상태를 확인할 수 있다.

m$sys_collector_sources

마크베이스 서버에 등록된 collector manager가 관리하는 설정 파일들의 정보를 유지하는 테이블이다. 테이블의 컬럼 정보는 아래와 같다.

Name	Description
MANAGER_ID	Collector manager 의 식별자
MANAGER_NAME	Collector manager 의 이름
SOURCE_TYPE	파일 종류 (Template, Regular Expression, and Python script)
SOURCE_PATH	파일의 절대 경로
CONTEXT	파일 내용

Procedure

마크베이스 서버에서 수행하는 프로시저 (Procedure) 로 collector 제어가 가능하다.

INSERT_COLLECTOR_SOURCE

Insert_collector_source procedure로 원격 서버에 실행중인 콜렉터에 설정 파일들을 전송할 수 있다.

```
EXECUTE INSERT_COLLECTOR_SOURCE ("manager_name", "path", "context");
```

각 파라메터의 의미는 다음과 같다.

- manager_name : 파일을 전송할 Collector manager의 이름
- path : 전송할 파일의 경로명
- context : 파일의 내용

만약 원격 서버에 파일이 이미 존재한다면 원래 파일은 파일명.bak 로 변경된다.

RENAME_COLLECTORMANAGER

이미 등록된 collector manager의 이름을 변경하는 Procedure 이다.

```
EXECUTE RENAME_COLLECTORMANAGER ("old_name", "new_name");
```

설정/모니터링

이 장은 마크베이스의 프로퍼티의 의미와 설정 방법, 메타 테이블, 버추얼 테이블에 대해서 기술한다.

Meta Table

메타 테이블은 마크베이스의 스키마 정보를 제시해 주는 테이블들로 테이블 명이 M$로 시작된다.

이 테이블들은 테이블의 이름과, 컬럼 정보, 인덱스 정보들을 유지하고 있고, DDL문에 의해서 생성, 변경, 삭제된 상황을 반영한다.
이 메타 테이블은 사용자에 의해서 추가, 삭제, 변경될 수 없다.

목차

User Objects

M$SYS_TABLES

사용자가 생성한 테이블을 표시한다.

컬럼 이름	설명
NAME	테이블 이름
TYPE	테이블 유형 • 0: Log • 1: Fixed • 3: Volatile • 4: Lookup • 5: Key Value • 6: Tag
DATABASE_ID	데이터베이스 식별자
ID	테이블 식별자
USER ID	테이블을 생성한 사용자
COLCOUNT	컬럼의 개수
FLAG	테이블 타입 구분 • 1 : Tag Data Table • 2 : Rollup Table • 4 : Tag Meta Table • 8 : Tag Stat Table

M$SYS_TABLE_PROPERTY

각 테이블에 적용된 테이블 프로퍼티 정보를 표시한다.

컬럼 이름	설명
ID	대상 테이블 ID
NAME	프로퍼티 이름
VALUE	프로퍼티 값

M$SYS_COLUMNS

M$SYS_TABLES 에 표시된 사용자 테이블의 컬럼 정보를 표시한다.

컬럼 이름	설명
NAME	컬럼 이름
TYPE	컬럼 타입
DATABASE_ID	데이터베이스 식별자
ID	컬럼 식별자
LENGTH	컬럼 길이

컬럼 이름	설명
TABLE_ID	컬럼이 속한 테이블의 식별자
FLAG	(서버 내부 사용을 위한 정보)
PART_PAGE_COUNT	파티션 당 페이지 수
PAGE_VALUE_COUNT	페이지 당 데이터의 수
MINMAX_CACHE_SIZE	MIN-MAX 캐쉬의 크기
MAX_CACHE_PART_COUNT	파티션 캐쉬의 최대 개수

M$SYS_INDEXES

사용자가 생성한 인덱스 정보를 표시한다.

컬럼 이름	설명
NAME	인덱스의 이름
TYPE	인덱스의 종류
DATABASE_ID	데이터베이스 식별자
ID	인덱스 식별자
TABLE_ID	인덱스를 생성한 테이블 식별자
COLCOUNT	인덱스를 생성한 컬럼의 수
PART_VALUE_COUNT	인덱스가 속한 테이블의 파티션당 데이터 수
BLOOM_FILTER	Bloom Filter 사용 가능 여부
KEY_COMPRESS	키값의 압축 여부
MAX_LEVEL	인덱스의 최대 레벨 (LSM 만 가능)
PAGE_SIZE	페이지 크기
MAX_KEYWORD_SIZE	최대 키워드 길이 (Keyword 만 가능)
BITMAP_ENCODE	Bitmap 인코딩 유형 (RANGE / EQUAL)

M$SYS_INDEX_COLUMNS

M$SYS_INDEXES 에 표시된 사용자 인덱스의 컬럼 정보를 표시한다.

컬럼 이름	설명
INDEX_ID	인덱스 식별자
INDEX_TYPE	인덱스 종류
NAME	컬럼의 이름
COL_ID	컬럼 식별자
DATABASE_ID	데이터베이스 식별자
TABLE_ID	테이블 식별자
TYPE	컬럼의 데이터타입

M$SYS_TABLESPACES

사용자가 생성한 테이블스페이스 정보를 표시한다.

컬럼 이름	설명
NAME	테이블스페이스 이름
ID	테이블스페이스 식별자
DISK_COUNT	테이블스페이스에 속한 디스크의 수

M$SYS_TABLESPACE_DISKS

테이블스페이스가 사용하는 디스크 정보를 유지한다.

컬럼 이름	설명
NAME	디스크 이름
ID	디스크 식별자
TABLESPACE_ID	디스크가 속한 테이블스페이스 식별자
PATH	디스크의 경로
IO_THREAD_COUNT	이 디스크에 할당된 IO 스레드의 수
VIRTUAL_DISK_COUNT	이 디스크에 할당된 Virtual Disk 단위 개수

M$SYS_USERS

마크베이스에 등록된 사용자 정보를 유지한다.

컬럼 이름	설명
USER_ID	사용자의 식별자
NAME	사용자의 이름

Collectors

M$SYS_COLLECTORS

마크베이스 서버에 등록된 Collector 정보를 표시한다.

컬럼 이름	컬럼 이름
COLLECTOR_ID	Collector의 식별자
MANAGER_NAME	Collector의 매니저 이름
COLLECTOR_NAME	Collector의 이름
LOG_TYPE	정규표현식 로그 타입의 이름
TABLE_NAME	Collector가 데이터를 입력할 테이블 이름
TEMPLATE_NAME	템플릿 파일의 이름
COLLECTOR_TYPE	Collector 타입
COLLECTOR_SOURCE	입력 로그 파일의 위치
COLLECTOR_LIB	입력 라이브러리 이름
COL_COUNT	컬럼의 수
PREPROCESS_PATH	Preprocessor 파일의 경로
REGEX_PATH	Regular Expression 파일의 경로
REGEX	정규표현식
END_REGEX	정규표현식의 끝을 나타낼 표현
LANGUAGE	로그파일의 언어 설정 (UTF-8)
SLEEP_TIME	Collector 입력 주기
DB_ADDR	마크베이스 서버 IP address
DB_PORT	마크베이스 서버 PORT
DB_USER	데이터베이스 사용자명
DB_PASS	데이터베이스 사용자 패스워드
PROCESS_BYTE	한번에 입력받은 데이터 크기 (재입력 시 초기화)

컬럼 이름	컬럼 이름
PROCESS_RECORD	한번에 입력받은 데이터 레코드 수 (재입력 시 초기화)
TOTAL_PROCESS_BYTE	기동 후 입력받은 데이터 총 크기
TOTAL_PROCESS_RECORD	기동 후 입력받은 데이터 레코드 총 개수
LAST_PROCESS_TIME	마지막으로 입력했던 시각
RUN_FLAG	Collector 실행 여부 (0:STOP, 1:START)

M$SYS_COLLECTOR_COLUMNS

Collector가 입력하는 테이블의 컬럼 정보를 표시한다.

컬럼 이름	설명
COLLECTOR_ID	Collector 식별자
COL_ID	컬럼 식별자
NAME	컬럼 이름
TYPE_NAME	컬럼 데이터타입
DATE_FORMAT	데이터타입이 Datetime 인 경우, 변환할 문자열 포맷
TYPE_CODE	컬럼 데이터타입 코드
SIZE	컬럼 길이
USE_INDEX	인덱스 사용 여부
REGEX_NO	등록된 정규표현식의 번호

M$SYS_COLLECTOR_OFFSETS

Collector가 입력하는 테이블의 마지막 Offset 정보와, 당시의 소스 체크섬을 표시한다.

> 5.5 부터 사용 가능하다.
> 이전까지는 테이블에 저장된 Offset 정보를 직접 조회했지만, 5.5 부터는 해당 정보를 계속 메모리에서 관리하도록 수정했다.

컬럼 이름	설명
USER_NAME	사용자 이름
TABLE_NAME	테이블 이름
ADDRESS	Collector 의 위치
CHECKSUM1	Collector Source 의 체크섬1
CHECKSUM2	Collector Source 의 체크섬2
OFFSET	Collector Source 의 Offset

M$SYS_COLLECTORMANAGERS

Collector 를 관리하는 Collector Manager 의 정보를 표시한다.

컬럼 이름	설명
MANAGER_ID	Collector 매니저 식별자
MANAGER_NAME	Collector 매니저 이름
MANAGER_HOST	Collector 매니저의 호스트
MANAGER_PORT	Collector 매니저의 포트 번호
MANAGER_LAST_PROCESS_TIME	마지막으로 Collector 매니저가 동작한 시간

M$SYS_COLLECTOR_SOURCES

Collector 가 취합하는 소스의 형식을 표시한다. 이 소스는 Collector Manager 가 관리한다.

컬럼 이름	설명
MANAGER_ID	Collector 매니저 식별자
MANAGER_NAME	컬렉터 매니저 이름
SOURCE_TYPE	로그 파일의 형식
SOURCE_PATH	로그 파일이 위치한 경로
CONTEXT	

Others

M$TABLES

M$로 시작하는 모든 메타테이블을 표시한다.

컬럼 이름	설명
NAME	메타 테이블의 이름
TYPE	테이블 유형
DATABASE_ID	데이터베이스 식별자
ID	메타 테이블의 식별자
USER ID	테이블을 생성한 사용자 (여기서는 SYS)
COLCOUNT	컬럼의 개수

M$COLUMNS

M$TABLES 에 표시된 메타테이블의 컬럼 정보를 표시한다.

컬럼 이름	설명
NAME	컬럼 이름
TYPE	컬럼 타입
DATABASE_ID	데이터베이스 식별자
ID	컬럼 식별자
LENGTH	컬럼 길이
TABLE_ID	컬럼이 속한 테이블의 식별자
FLAG	(서버 내부 사용을 위한 정보)
PART_PAGE_COUNT	파티션당 페이지 수
PAGE_VALUE_COUNT	페이지 당 데이터의 수
MINMAX_CACHE_SIZE	MIN-MAX 캐쉬의 크기
MAX_CACHE_PART_COUNT	파티션 캐쉬의 최대 개수

Virtual Table

Virtual Table은 마크베이스 서버의 다양한 운영 정보들을 테이블 형태로 표현하는 가상 테이블이다. 이 테이블들의 이름은 V$문자로 시작한다.

마크베이스 서버가 어떤 상태로 동작하고 있는지를 알기 위해서 이 데이터를 읽어서 저장해 두고 이용할 수 있다.
추가로, 이 Virtual Table 을 다른 테이블들과 JOIN 연산을 통해서 다양한 정보를 얻을 수 있다.

Virtual Table 은 읽기 전용으로 사용자가 추가/삭제/갱신할 수 없다.

목차

Session/System

V$PROPERTY

서버에 설정된 프로퍼티 정보를 표시한다.

컬럼 이름	설명
NAME	프로퍼티명
VALUE	프로퍼티 값
TYPE	데이터 타입
DEFLT	기본 값
MIN	설정할 수 있는 최소값
MAX	설정할 수 있는 최대값

V$SESSION

MACHBASE 서버에 접속된 세션 정보를 표시한다.

컬럼 이름	설명
HOSTNAME (Cluster Only)	세션 연결된 HOST 이름
ID	세션 식별자
CLOSED	연결이 닫혀있는지 여부
USER_ID	사용자 식별자
LOGIN_TIME	접속 시각
CLIENT_TYPE	접속 Client 타입
USER_NAME	사용자 이름
USER_IP	사용자 IP
SQL_LOGGING	해당 세션의 Trace Log 에 메시지를 남길지 여부 1. Parsing, Validation, Optimization 단계에서 발생하는 에러를 남긴다. 2. DDL을 수행한 결과를 남긴다. 3. (위의 두 케이스 모두 남긴다)
SHOW_HIDDEN_COLS	SELECT 시, 숨겨진 컬럼을 나타낼 것인지 여부
FEEDBACK_APPEND_ERROR	APPEND 시 에러를 찾으면 곧바로 실패할 것인지 여부
DEFAULT_DATE_FORMAT	Datetime 입력 시 기본 입력 포맷
HASH_BUCKET_SIZE	쿼리 수행 시 생성할, Temp Hashtable 의 Bucket 개수
MAX_QPX_MEM	쿼리 수행 시 가용할 최대 메모리 크기
RS_CACHE_ENABLE	Result Cache 사용 여부

컬럼 이름	설명
HOSTNAME (Cluster Only)	세션 연결된 HOST 이름
RS_CACHE_TIME_BOUND_MSEC	Result Cache 사용 시, 결과를 저장하기 위한 최대 경과 시간
RS_CACHE_MAX_MEMORY_PER_QUERY	Result Cache 사용 시, 쿼리 마다 사용할 최대 메모리 크기
RS_CACHE_MAX_RECORD_PER_QUERY	Result Cache 사용 시, 쿼리 마다 사용할 최대 결과 개수
RS_CACHE_APPROXIMATE_RESULT_ENABLE	Result Cache 사용 시, 정확하지 않은 쿼리의 결과를 캐싱해 갈 것인지 여부
IDLE_TIMEOUT	세션 연결 후 해당 시간 동안 Client 가 아무일도 하지 않을 시 세션 종료
QUERY_TIMEOUT	쿼리 수행 시 응답 대기 시간

V$SESMEM

세션 메모리 정보를 표시한다.

컬럼 이름	설명
SID	세션 식별자
ID	메모리 매니저 식별자
USAGE	사용 크기

V$SESSTAT

세션의 통계 정보를 표시한다.

컬럼 이름	설명
SID	세션 식별자
ID	통계 정보 식별자
VALUE	통계 정보 값

V$SESTIME

세션의 시간 정보를 표시한다.

컬럼 이름	설명
SID	세션 식별자
ID	수행 단위 식별자
ACCUM_TICK	누적 시간
MAX_TICK	(각 수행 중) 최대 시간

V$SYSMEM

시스템의 메모리 정보를 표시한다.

컬럼 이름	설명
ID	메모리 매니저 식별자
NAME	메모리 매니저 이름
USAGE	현재 사용량
MAX_USAGE	(기록된) 최대 사용량

V$SYSSTAT

시스템의 통계 정보를 표시한다.

컬럼 이름	설명
ID	통계 정보 식별자
NAME	통계 정보 이름
VALUE	통계 정보 값

V$SYSTIME

시스템의 시간 정보를 표시한다.

컬럼 이름	설명
ID	수행 단위 식별자
NAME	수행 단위 이름
ACCUM_TICK	누적 시간
AVG_TICK	(각 수행 중) 평균 시간
MIN_TICK	(각 수행 중) 최소 시간
MAX_TICK	(각 수행 중) 최대 시간
COUNT	수행 횟수

V$STMT

사용자가 현재 실행중인 쿼리문에 대한 정보를 표시한다.

컬럼 이름	설명
ID	쿼리 식별자
SESS_ID	쿼리를 수행한 세션 식별자
STATE	쿼리 상태
RECORD_SIZE	SELECT 구문 수행 중인 경우, 결과 레코드 크기
QUERY	쿼리 구문

V$VERSION

MACHBASE 의 버전에 대한 정보를 표시한다.

컬럼 이름	설명
BINARY_DB_MAJOR_VERSION	DB 메이저 버전
BINARY_DB_MINOR_VERSION	DB 마이너 버전
BINARY_META_MAJOR_VERSION	META 메이저 버전
BINARY_META_MINOR_VERSION	META 마이너 버전
BINARY_CM_MAJOR_VERSION	Client (Communication Level) 메이저 버전
BINARY_CM_MINOR_VERSION	Client (Communication Level) 마이너 버전
BINARY_SIGNATURE	DB서버 파일의 버전 명
FILE_DB_MAJOR_VERSION	File DB 메이저 버전
FILE_DB_MINOR_VERSION	File DB 메이저 버전
FILE_META_MAJOR_VERSION	File META 메이저 버전
FILE_META_MINOR_VERSION	File META 마이너 버전
FILE_CM_MAJOR_VERSION	File Client (Communication Level) 메이저 버전
FILE_CM_MINOR_VERSION	File Client (Communication Level) 마이너 버전

컬럼 이름	설명
FILE_CREATE_TIME	파일 생성 시각
EDITION	MACHBASE 유형

Result Cache

V$RS_CACHE_LIST

결과 캐시 목록을 표시한다.

컬럼 이름	설명
TOUCH_TIME	캐시를 사용하거나 생성한 시각
USER_ID	캐시를 생성한 사용자 식별자
QUERY	캐시를 만든 쿼리문
TIME_SPENT	결과를 생성하기까지 경과 시간
TABLE_COUNT	쿼리문과 연관된 테이블 개수
RECORD_COUNT	결과 레코드 개수
REFERENCE_COUNT	현재 참조중인 세션의 개수
HIT_COUNT	캐시 히트 횟수
AGGR_TOUCH_TIME	집계 결과인 경우, 캐시를 사용하거나 생성한 시각
AGGR_HIT_COUNT	집계 결과인 경우, 캐시 히트 횟수

V$RS_CACHE_STAT

하나의 세션에서의 결과 캐시의 통계 정보를 표시한다.

컬럼 이름	설명
CACHE_COUNT	결과 캐시의 개수
CACHE_HIT	총 캐시 히트 횟수
AGGR_HIT	집계 결과의 총 캐시 히트 횟수
CACHE_REPLACED	캐시 교체 횟수
CACHE_MEMORY_USAGE	캐시로 사용된 메모리 크기

Storage

V$STORAGE

저장 시스템의 내부 정보를 표시한다.

컬럼 이름	설명
DC_TABLE_FILE_SIZE	디스크 컬럼 데이터의 총 용량
DC_INDEX_FILE_SIZE	인덱스 파일 데이터의 총 용량
DC_TABLESPACE_DWFILE_SIZE	모든 컬럼데이터를 위한 DWFILE의 총 용량
DC_KV_TABLE_FILE_SIZE	TAGDATA 테이블의 파티션 테이블이 가지는 데이터파일 총 용량

V$STORAGE_MOUNT_DATABASES

마운트 기능을 이용하여 마운트한 백업 데이터베이스의 정보를 표시한다.

컬럼 이름	설명
NAME	마운트된 데이터베이스의 이름
PATH	백업 파일의 위치
BACKUP_TBSID	백업 데이터베이스의 테이블스페이스 식별자
BACKUP_SCN	백업 데이터베이스의 식별자
MOUNTDB	백업 시간
DB_BEGIN_TIME	백업 데이터베이스의 최초입력 시간
DB_END_TIME	백업 데이터베이스의 최종 입력 시간
BACKUP_BEGIN_TIME	백업 실행시 시작 시간
BACKUP_END_TIME	백업 실행시 종료 시간
FLAG	프로퍼티 플래그

V$CACHE

Storage Manager 에서 읽은 결과를 캐싱한, 캐시 객체에 대한 종합 정보를 표시한다.

컬럼 이름	설명
OBJ_COUNT	결과집합 캐시 객체의 현재 수

V$CACHE_OBJECTS

저장 시스템에서 읽은 결과를 캐싱한, 각 캐시 객체에 대한 정보를 표시한다.

컬럼 이름	설명
OID	객체식별자
REF_COUNT	참조 카운트
FLAG	(서버 내부 사용 플래그)

V$STORAGE_DC_TABLESPACES

저장 시스템의 테이블스페이스 정보를 표시한다.

컬럼 이름	설명
NAME	테이블스페이스 이름
ID	테이블스페이스 식별자
FLAG	테이블스페이스 Property 를 나타내는 Flag
REF_COUNT	테이블스페이스 참조 횟수
DISK_COUNT	테이블스페이스에 속한 디스크 개수

V$STORAGE_DC_TABLESPACE_DISKS

저장 시스템의 테이블스페이스 정보를 표시한다.

컬럼 이름	설명
NAME	디스크 이름
ID	디스크 식별자
TABLESPACE_ID	디스크가 속한 테이블스페이스 식별자
PATH	디스크의 경로
IO_THREAD_COUNT	I/O Thread 개수
IO_JOB_COUNT	I/O Job 개수
VIRTUAL_DISK_COUNT	가상 디스크 개수

V$STORAGE_DC_DWFILES

저장 시스템에서 운용하는 Double-write 파일 (DW File) 의 정보를 표시한다.

컬럼 이름	설명
TBS_ID	테이블스페이스 식별자
DISK_ID	디스크 식별자
FILE	파일의 경로
TABLE_ID	테이블 식별자
COLUMN_ID	컬럼 식별자
PARTITION_ID	파티션 식별자
PAGE_ID	페이지 식별자
DISK_OFFSET	디스크 오프셋
DISK_IMAGE_SIZE	디스크 이미지 크기
HEAD_CRC32CODE_IMAGE	CRC32 Code 의 Head Image
TAIL_CRC32CODE_IMAGE	CRC32 Code 의 Tail Image
CRC32CODE_PAGE	CRC32 Code 의 Page
HEAD_TIMESTAMP_PAGE	Timestamp 의 Head Page
TAIL_TIMESTAMP_PAGE	Timestamp 의 TailPage

V$STORAGE_DC_PAGECACHE

저장 시스템에서 운용하는 Page Cache 에 대한 정보를 표시한다.

컬럼 이름	설명
MAX_MEM_SIZE	Page Cache 의 최대 메모리 크기
CUR_MEM_SIZE	Page Cache 의 현재 메모리 크기
PAGE_CNT	캐싱된 페이지 개수
CHECK_TIME	검사 시간

V$STORAGE_DC_PAGECACHE_LRU_LST

저장 시스템에서 운용하는 Page Cache 의 LRU List 에 대한 정보를 표시한다.

컬럼 이름	설명
OBJECT_ID	객체 식별자
LEVEL	파티션 레벨
PARTITION_ID	파티션 식별자
OFFSET	Page Cache 의 Offset
SIZE	페이지 크기
REF_CNT	참조 횟수

V$STORAGE_USAGE

저장 시스템에서 사용 중인 스토리지의 사용량을 표시한다.

컬럼 이름	설명
TOTAL_SPACE	$MACHBASE_HOME/dbs 디렉터리가 위치한 스토리지의 총 용량
USED_SPACE	$MACHBASE_HOME/dbs 디렉터리가 위치한 스토리지의 사용량
USED_RATIO	사용량 비율(%)

컬럼 이름	설명
RATIO_CAP	스토리지 사용량 한계. USED_RATIO이 이 한계에 도달하면 데이터 입력/인덱스 구축이 멈춤.

V$STORAGE_TABLES

테이블의 상세 정보를 표시한다.

컬럼 이름	설명
ID	테이블의 ID
TYPE	테이블 형태 • Persistent: LOG 테이블과 TAG 테이블 • Volatile: 휘발성(Volatile) 테이블 • Key-Value: TAG 테이블의 부속 테이블
STATUS	현재 상태 • Creating...: CREATE TABLE로 테이블 생성 진행중 • Normal: 정상 • Predrop: DROP TABLE 명령 접수 상태 • Dropping...: DROP TABLE 명령 수행 상태 • Dropped: DROP TABLE 명령 완료 상태 • Mounted: 백업된 데이터베이스를 mount 명령으로 불러온 상태
STORAGE_USAGE	해당 테이블이 스토리지에서 점유한 용량

Log Table

V$STORAGE_DC_TABLES

Log Table 에 대한 내부 정보를 표시한다.

컬럼 이름	설명
ID	테이블의 식별자
DATABASE_ID	데이터베이스 식별자
CREATE_SCN	생성 당시의 시스템 변경 번호 (System Change Number)
UPDATE_SCN	최근 변경 당시의 시스템 변경 번호 (System Change Number)
DDL_REF_COUNT	DDL 구문 수행으로, 해당 테이블을 참조하고 있는 세션의 개수.
BEGIN_RID	테이블의 최소 RID
END_RID	테이블의 마지막 Row ID + 1
BEGIN_META_RID	메타 정보를 기록하기 시작한 시점의 ID
END_META_RID	메타 정보의 기록이 종료한 시점의 ID
END_SYNC_RID	디스크에 기록된 마지막 Row ID + 1
FLAG	Table Property 를 나타내는 Flag
COLUMN_COUNT	테이블의 컬럼 수
INDEX_COUNT	테이블의 인덱스 수
INDEX_MIN_END_RID	인덱스에 기록된 마지막 RID + 1
LAST_ARRIVAL_TIME	마지막으로 기록된 _arrival_time 값
LAST_CHECKPOINT_TIME	마지막으로 Checkpoint 를 지난 시점
TYPE	테이블 유형
REMAINING_ROW_COUNT	자동 삭제 기능 사용시, 삭제되지 않는 레코드의 수
KEPT_DURATION	자동 삭제 기능 사용시, 데이터를 유지할 기간

V$STORAGE_DC_TABLES_STAT

Log Table 에 대한 내부 정보를 표시한다.

컬럼 이름	설명
TABLESPACE_ID	테이블스페이스 식별자
TABLE_ID	테이블 식별자
COLUMN_ID	컬럼 식별자
COUNT	레코드 개수

V$STORAGE_DC_TABLE_COLUMNS

Log Table 의 컬럼에 대한 정보를 표시한다.

컬럼 이름	설명
TABLE_ID	테이블 식별자
DATABASE_ID	데이터베이스 식별자
ID	컬럼 식별자
FLAG	프로퍼티 플래그
SIZE	컬럼의 데이터 크기
PARTITION_VALUE_COUNT	파티션에 저장되는 최대 데이터 수
PAGE_VALUE_COUNT	페이지에 저장되는 최대 데이터 수
CACHE_VALUE_COUNT	캐시 값의 최대 수
MINMAX_CACHE_SIZE	컬럼 파티션에 대한 MIN/MAX 캐시의 최대 크기
CUR_APPEND_PARTITION_ID	현재 입력을 진행중인 파티션의 식별자
CUR_CACHE_PARTITION_COUNT	현재 캐시에 데이터를 읽어들인 파티션의 수
CUR_MINMAX_CACHE_SIZE	현재 MIN/MAX캐시의 크기
END_RID_FOR_DEFAULT_VALUE	이 값보다 작은 RID를 갖는 컬럼은 디폴트값으로 지정됨
DISK_FILE_SIZE	해당 컬럼에 대한 컬럼 파티션 데이터 파일의 전체 크기
MEMORY_TOTAL_SIZE	테이블이 사용 중인 메모리 크기
MEMORY_ALLOC_SIZE	테이블이 할당받은 메모리 크기

V$STORAGE_DC_TABLE_COLUMN_PARTS

Log Table 의 컬럼 파티션 정보를 표시한다.

컬럼 이름	설명
TABLE_ID	테이블 식별자
DATABASE_ID	데이터베이스 식별자
COLUMN_ID	컬럼 식별자
ID	파티션 식별자
FLAG	컬럼 Property 를 나타내는 Flag
BEGIN_RID	파티션에 저장된 최소 RID
END_RID	파티션에 저장된 최종 RID
END_SYNC_RID	SYNC가 끝난 최종 RID. 시작 RID 보다 크고 마지막 SYNC RID 보다 작은 RID 를 갖는 데이터는 파티션 파일에 기록되어 있다.
MIN_TIME	컬럼 파티션에 최초로 데이터를 입력한 시간
MAX_TIME	컬럼 파티션에 마지막으로 데이터를 입력한 시간
MAX_VALUE_COUNT_PER_PARTITION	파티션의 최대 데이터 수
MAX_VALUE_COUNT_PER_PAGE	페이지당 최대 데이터 수

컬럼 이름	설명
MAX_PAGE_COUNT	파티션당 최대 페이지의 수
PAGE_SIZE	컬럼 파티션에 저장된 페이지의 크기
PAGE_COUNT	현재 컬럼 파티션에 생성된 페이지의 수
COMPRESS_RATIO	컬럼 파티션의 압축률. 0이면 아직 데이터 압축이 실행되지 않은 경우이다.
DISK_FILENAME	파티션 파일의 이름
EXTERNAL_PART_SIZE	데이터의 양이 큰 값은 외부 파티션 파일에 기록하는데, 그 파일의 크기를 표시
MIN_VALUE	컬럼 파티션의 최소값
MAX_VALUE	컬럼 파티션의 최대값

V$STORAGE_DC_TABLE_INDEXES

Log Table 에 생성된 인덱스 정보를 표시한다.

컬럼 이름	설명
TABLE_ID	테이블 식별자
DATABASE_ID	데이터베이스 식별자
ID	인덱스 식별자
FLAG	인덱스 Property 를 나타내는 Flag
TABLE_BEGIN_RID	테이블의 입력된 최소 RID
TABLE_END_RID	테이블의 마지막 RID
BEGIN_RID	인덱스의 최소 RID
END_RID	인덱스의 최대 RID
END_SYNC_RID	파일에 기록된 최대 RID+1
COLUMN_COUNT	인덱스 컬럼 수
BEGIN_PART_ID	인덱스의 최초 파티션 식별자
END_PART_ID	인덱스의 최종 파티션 식별자
FLUSH_REQUEST_COUNT	디스크에 반영요청된 인덱스 파티션의 수
MAX_KEY_SIZE	최대 키 크기
INDEX_TYPE	인덱스 유형
DISK_FILE_SIZE	해당 인덱스에 대한 인덱스 파티션 파일의 전체 크기
LAST_CHECKPOINT_TIME	마지막으로 Checkpoint 를 지난 시점

LSM Index

V$STORAGE_DC_LSMINDEX_LEVEL_PARTS

LSM Index 파티션에 대한 정보를 표시한다.

컬럼 이름	설명
TABLE ID	인덱스가 생성된 테이블의 식별자
TABLESPACE_ID	테이블스페이스 식별자
INDEX_ID	인덱스 식별자
LEVEL	인데스 파티션의 LSM 레벨
PARTITION_ID	파티션 식별자
BEGIN_RID	파티션에 입력된 최소 RID
END_RID	파티션에 입력된 최대 RID+1

컬럼 이름	설명
KEY_VALUE_COUNT	파티션에 입력된 키값의 수
KEY_VALUE_TABLE_SIZE	키값을 저장하는 페이지 크기
KEY_VALUE_TABLE_PAGE_COUNT	키값을 저장하는 페이지의 수
MIN_KEY_VALUE	최소 키 값
MAX_KEY_VALUE	최대 키 값
BITMAP_TABLE_SIZE	비트맵 값을 저장하는 페이지의 합계
BITMAP_TABLE_PAGE_COUNT	비트맵 값을 저장하는 페이지의 수
META_SIZE	메타 정보를 저장하는 페이지의 합계
META_PAGE_COUNT	메타 정보를 저장하는 페이지의 수
TOTAL_BUILD_MSEC	해당 파티션을 완성하기 까지의 총 시간
KEYVAL_BUILD_MSEC	KeyValue Mode 에서, 해당 파티션을 완성하기 까지의 총 시간
BITMAP_BUILD_MSEC	Bitmap Mode 에서, 해당 파티션을 완성하기 까지의 총 시간

V$STORAGE_DC_LSMINDEX_LEVEL_PARTS_CACHE

LSM Index 파티션 캐시에 대한 정보를 표시한다.

컬럼 이름	설명
TABLESPACE_ID	테이블스페이스 식별자
TABLE_ID	인덱스가 생성된 테이블의 식별자
INDEX_ID	인덱스 식별자
LEVEL	인데스 파티션의 LSM 레벨
PARTITION_ID	파티션 식별자
BEGIN_RID	파티션에 입력된 최소 RID
END_RID	파티션에 입력된 최대 RID+1
KEY_VALUE_COUNT	파티션에 입력된 키값의 수
KEY_VALUE_TABLE_SIZE	키값을 저장하는 페이지의 크기
KEY_VALUE_TABLE_PAGE_COUNT	키값을 저장하는 페이지의 수
BITMAP_TABLE_SIZE	비트맵 값을 저장하는 페이지의 합계
BITMAP_TABLE_PAGE_COUNT	비트맵 값을 저장하는 페이지의 수
META_SIZE	메타 정보를 저장하는 페이지의 합계
META_PAGE_COUNT	메타 정보를 저장하는 페이지의 수
MEMORY_SIZE	메모리 사용량
MEMORY_SIZE_RBTREE	Redblack Tree 가 사용한 메모리 사용량

V$STORAGE_DC_LSMINDEX_LEVELS

LSM 인덱스의 레벨에 관한 정보를 표시한다.

컬럼 이름	설명
TABLE ID	테이블 식별자
DATABASE_ID	데이터베이스 식별자
INDEX_ID	인덱스 식별자
LEVEL	레벨
BEGIN_RID	파티션의 첫번째 RID
END_RID	파티션의 마지막 RID+1
META_BEGIN_RID	메타정보를 기록하기 시작한 시점의 RID

컬럼 이름	설명
META_END_RID	메타정보의 기록이 끝난 시점의 RID
DELETE_END_RID	삭제된 RID 최대값 +1

V$STORAGE_DC_LSMINDEX_FILES

LSM Index 를 구성하는 파일에 대한 정보를 표시한다.

컬럼 이름	설명
TABLE_ID	테이블 식별자
DATABASE_ID	데이터베이스 식별자
INDEX_ID	인덱스 식별자
LEVEL	인데스 파티션의 LSM 레벨
PARTITION_ID	파티션 식별자
BEGIN_RID	파티션의 첫번째 RID
END_RID	파티션의 마지막 RID+1
PATH	인덱스 파일의 위치

V$STORAGE_DC_LSMINDEX_AGER_JOBS

LSM Index 의 삭제를 담당하는 Ager 의 작업 상태를 표시한다.

컬럼 이름	설명
TABLE_ID	테이블 식별자
INDEX_ID	인덱스 식별자
LEVEL	인데스 파티션의 LSM 레벨
BEGIN_RID	파티션의 첫번째 RID
END_RID	파티션의 마지막 RID+1
STATE	Index Ager 의 작업 상태

Volatile Table

V$STORAGE_DC_VOLATILE_TABLE

Volatile Table 에 대한 정보를 표시한다.

컬럼 이름	설명
MAX_MEM_SIZE	Volatile Tablespace 의 최대 크기
CUR_MEM_SIZE	Volatile Tablespace 의 현재 크기

Tag Table

V$STORAGE_TAG_TABLES

Tagdata Table 의 파티션 테이블에 대한 정보를 표시한다.

컬럼 이름	설명
ID	테이블 식별자
TABLE_BEGIN_RID	테이블 시작 RID

컬럼 이름	설명
TABLE_END_RID	테이블 끝 RID
WRITE_END_RID	데이터 파일에 기록된 마지막 RID
EXT_ROW_COUNT	VARCHAR 레코드 중 외부 파티션에 입력된 개수
EXT_WRITE_COUNT	VARCHAR 레코드 중 데이터파일에 기록된 개수
DISK_INDEX_END_RID	스토리지에 저장된 인덱스의 끝 RID
MEMORY_INDEX_END_RID	메모리 인덱스에 상주한 테이블 끝 RID
DELETE_MIN_DATE	DELETE ... BETWEEN ... 수행시 삭제 대상의 최소 시각
DELETE_MAX_DATE	DELETE ... BETWEEN ..., 혹은 DELETE ... BEFORE ... 수행시 삭제 대상의 최대 시각
INDEX_STATE	현재 인덱스 구축 상태 • IDLE: 구축 완료, 대기중. • PROGRESS: 구축 진행중 • IOWAIT: 스토리지에 입출력 연산 대기. • PENDING: 테이블에 읽기 잠금 대기중 • SHUTDOWN: 정지됨. DELETE 연산, 혹은 DROP 연산 진행중. • ABNORMAL: 비정상 종료
DELETE_STATE	현재 DELETE 연산의 상태. DELETE 명령이 들어올 때에만 수행되므로 IDLE이 없다. • PROGRESS: 삭제 진행중 • IOWAIT: 스토리지에 입출력 연산 대기. • PENDING: 테이블에 읽기/쓰기 잠금 대기중 • SHUTDOWN: 정지됨. DELETE 연산이 진행되지 않음. • ABNORMAL: 비정상 종료
SAVE_STATE	현재 테이블 저장 연산의 상태. • IDLE: 저장 완료, 대기중. • PROGRESS: 저장 진행중 • IOWAIT: 스토리지에 입출력 연산 대기. • PENDING: 테이블에 읽기 잠금 대기중 • SHUTDOWN: 정지됨. DELETE 연산, 혹은 DROP 연산 진행중. • ABNORMAL: 비정상 종료

V$STORAGE_TAG_CACHE

Tagdata Table 의 파티션 테이블에서 사용하는 캐시 정보를 표시한다.

컬럼 이름	설명
CATEGORY	캐쉬되고 있는 객체 분류
USED_MEMORY	사용중인 메모리 크기
BLOCK_COUNT	데이터 캐시 개수
CACHE_HIT	데이터 캐시 히트 횟수
CACHE_MISS	데이터 캐시 미스 횟수
FLUSHOUT	데이터 캐시 충돌로 페이지를 비운 횟수
COLDREAD	스토리지에서 직접 읽어온 데이터 페이지 개수
MEMORY_WAIT	데이터 메모리가 캐시 충돌로 대기한 횟수
IO_WAIT	데이터 읽기 연산 대기 횟수

V$STORAGE_TAG_CACHE_OBJECTS

Tagdata Table의 파티션 테이블에서 사용하는 각각의 캐시 블럭에 대한 상세정보를 표시한다.

컬럼 이름	설명
CATEGORY	캐쉬되고 있는 객체 분류
LATEST_HIT	마지막 접근 시각

컬럼 이름	설명
STATUS	캐시 상태 • None: 메모리 할당을 마친 상태 • Resides: 캐시에 보존된 상태 • Loading: 스토리지에서 테이블 데이터를 불러 오는 중 • ERROR!: 데이터를 불러오는 중 오류 발생
WAIT_COUNT	Loading 상태에서 해당 캐시를 읽지 못해 대기한 회수
REF_COUNT	현재 캐시 블럭을 참조 중인 세션 수
HIT_COUNT	캐시 블럭을 참조한 회수
TABLE_ID	테이블 식별자
FILE_ID	파일 식별자
PART_ID	데이터파일 내부의 파티션 식별자
SAVE_SCN	테이블 저장 SCN
VSAVE_SCN	테이블 저장 SCN
DELETE_SCN	DELETE 연산 SCN
OFFSET	데이터파일 오프셋
DATA_SIZE	압축 이전 데이터 크기, 혹은 0

V$STORAGE_TAG_TABLE_FILES

Tagdata Table 의 파티션 테이블의 파일 정보를 표시한다.

컬럼 이름	설명
TABLE_ID	테이블 식별자
FILE_ID	파일 식별자
STATE	인덱싱 상태 • COMPLETE: 데이터 저장, 인덱싱 완료 • INDEXING: 인덱스 구축 중. • FILLED: 데이터가 꽉 찬 상태, 인덱싱 대기 중 • PARTIAL: 아직 데이터가 꽉 차지 않았음. 인덱싱 대기 중.
REF_COUNT	현재 파일을 참조 중인 세션 수
ROW_COUNT	삭제됐던 레코드를 포함하여 파일에 저장된 레코드 개수
DEL_COUNT	파일에서 삭제된 레코드 개수
MIN_DATE	해당 파일에 기록된 데이터의 최소 일자
MAX_DATE	해당 파일에 기록된 데이터의 최대 일자

V$STORAGE_TAG_INDEX

Tagdata Table 에 생성된 인덱스 정보를 표시한다.

컬럼 이름	설명
TABLE_ID	테이블 식별자
INDEX_ID	인덱스 식별자(INDEX_ID가 4294967295인 경우 tag테이블 생성시 자동으로 생성되는 기본 인덱스를 의미함)
INDEX_STATE	인덱싱 상태 • IDLE: 인덱싱이 완료되어 대기중인 상태 • INDEXING: 인덱싱이 진행중인 상태 • STORAGE FULL: Disk full상태로 인덱싱이 중단된 상태

컬럼 이름	설명
DISK_INDEX_END_RID	마지막으로 disk에 반영된 인덱스의 EndRID
MEMORY_INDEX_END_RID	마지막으로 memory에 반영된 인덱스의 EndRID
TABLE_END_RID	테이블에 마지막으로 반영된 데이터의 EndRID

Tag Rollup

V$ROLLUP

Tagdata 테이블의 Rollup 정보를 표시한다.

컬럼 이름	설명
ID	Rollup 작업 ID
ROLLUP_TABLE	Rollup 이 저장할 테이블 이름
SOURCE_TABLE	Rollup 이 조회할 테이블 이름
USER_ID	Rollup Table과 Source Table의 User ID
ROOT_TABLE	최상위 Source Table 이름
INTERVAL	Rollup의 실행 주기 (sec)
ENABLED	Rollup 진행 여부를 나타냄
END_RID	Source Table의 마지막 RID
LAST_ELAPSED_MSEC	최근에 진행했던 Rollup의 경과 시간

Stream

V$STREAMS

칼럼 이름	설명
NAME	서버에 등록된 stream질의의 이름. 서버내에서 유일해야 함.
LAST_EX_TIME	해당 STREAM질의가 마지막으로 수행된 시간
TABLE_NAME	STREAM질의의 검색 대상 테이블 이름
END_RID	STREAM 질의가 마지막으로 읽어 들인 RID
STATE	STREAM질의의 현재 상태
QUERY_TXT	사용자가 입력한 STREAM질의의 원본
ERROR_MSG	마지막으로 실행했을 때의 에러 메시지
FREQUENCY	질의 수행의 최소 대기 시간. 0이면 매 레코드마다 실행되며 0이 아니면 해당 시간이 지날 때 마다 수행된다. 단위는 나노초이다.

License

V$LICENSE_INFO

라이선스 정보를 표시한다.

컬럼 이름	설명
INSTALL_DATE	설치일
TYPE	라이선스 유형
POLICY	라이선스 정책 유형
CUSTOMER	고객사 이름
ISSUE_DATE	발행일
ID	호스트 ID
EXPIRY_DATE	만료일
SIZE_LIMIT	일 입력제한량
ADDENDUM	추가 데이터 비율
VIOLATION_ACTION	위반 시 행동
VIOLATION_LIMIT	서비스가 중단될 위반 횟수 (매월 갱신)
STOP_ACTION	중단 행동
RESET_FLAG	(서버 내부 사용)

V$LICENSE_STATUS

라이선스 상태를 표시한다.

컬럼 이름	설명
USER_DATA_PER_DAY	하루에 입력할 수 있는 데이터 제한량
PREVIOUS_CHECK_DATE	직전 라이선스 검사일
VIOLATION_COUNT	라이선스 위반 횟수
STOP_ENABLED	(제거됨)

Mutex

V$MUTEX

현재 뮤텍스 상태를 보여준다.

필드명	설명	비고
OBJECT	뮤텍스 객체의 주소	
NAME	뮤텍스 생성시 부여한 이름	
TYPE	뮤텍스 타입	• Mutex: pmuMutex • RW Mutex: pmuRWMutex
OWNER	뮤텍스를 획득한 스레드의 ID	• Mutex: 뮤텍스를 획득한 스레드가 없으면 0. • RW Mutex w/ Read-Lock: 0 • RW Mutex w/ Write-Lock: Write Lock을 획득한 스레드의 ID
LOCK_COUNT	뮤텍스를 획득한 스레드 개수	• RW Mutex는 2 이상이 될 수 있음.
PEND_COUNT	뮤텍스를 획득하려고 대기 중인 스레드 개수	• TRACE_MUTEX_WAIT_STATUS=1일 때에만 수집
TRY_COUNT	뮤텍스를 획득하려고 시도한 회수	• TRACE_MUTEX_WAIT_STATUS=1일 때에만 수집
CONFLICT_COUNT	뮤텍스 획득에 실패한 회수	• TRACE_MUTEX_WAIT_STATUS=1일 때에만 수집
WAIT_TICK	뮤텍스 획득 대기 시간의 총합	• TRACE_MUTEX_WAIT_STATUS=1일 때에만 수집 • RW Mutex에는 기록하지 않음
WAIT_TICK_AVG	뮤텍스 획득 시도 후 성공까지의 평균 시간	• TRACE_MUTEX_WAIT_STATUS=1일 때에만 수집 • RW Mutex에는 기록하지 않음
HELD_TICK	뮤텍스를 획득한 이후 해제할 때까지의 시간 총합	• TRACE_MUTEX_WAIT_STATUS=1일 때에만 수집 • RW Mutex에는 기록하지 않음
HELD_TICK_AVG	뮤텍스 획득 이후 해제까지의 시간 평균	• TRACE_MUTEX_WAIT_STATUS=1일 때에만 수집 • RW Mutex에는 기록하지 않음

V$MUTEX_WAIT_STAT

현재 대기중인 뮤텍스의 콜스택을 보여준다.

필드	설명	비고
THREAD_ID	뮤텍스 획득 대기 중인 스레드 ID	
OBJECT	획득 시도 중인 뮤텍스의 주소	• V$MUTEX의 OBJECT와 동일
DEPTH	호출 깊이	• TRACE_MUTEX_WAIT_STACK=1일 때에만 수집
SYMBOL	뮤텍스 획득을 호출한 함수의 심볼	• TRACE_MUTEX_WAIT_STACK=1일 때에만 수집

Cluster

V$NODE_STATUS

Cluster 각 Node 의 상태를 표시한다. 1건만 표시된다.

컬럼 이름	설명
NODETYPE	Node 의 유형. 쿼리로 조회 가능한 Type 은 두 가지 뿐이다. • Broker • Warehouse
STATE	Node 의 상태

V$DDL_INFO

Cluster 에서 수행한 DDL 정보를 표시한다.

컬럼 이름	설명
SEQUENCENUMBER	DDL 순서 번호
TIME	DDL 수행 시간
VALUE	DDL 쿼리 결과 값 (서버 내부 사용)
CLIENT	클라이언트 이름
BROKER	Leader Broker 의 Node 이름
USER	사용자 이름
SQL	DDL 쿼리 값

V$REPLICATION

Replication 작동에 대한 정보를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
MODE	(서버 내부 사용)
STATE	Node 의 상태
ADDR	Replication Manager 의 주소
PORT_NO	Replication Manager 의 포트번호
MAX_SENDER_COUNT	생성 가능한 Sender 최대 개수
RUN_SENDER_COUNT	작동중인 Sender 최대 개수

V$REPL_SENDER

Replication 작동 시, Sender 의 정보를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
ID	Sender 식별자
STATUS	Sender Thread 의 작동상태
PAYLOAD_RECV_COUNT	Sender 로부터 받은 페이로드 개수
PAYLOAD_RECV_BYTES	Sender 로부터 받은 페이로드 크기 총합
QUEUE_REMAIN_COUNT	Receive Queue 에 남은 버퍼의 개수
NET_SEND_COUNT	전체 전송 횟수
NET_SEND_SIZE	전체 전송 크기 총합
NET_RECV_COUNT	전체 수신 횟수
NET_RECV_SIZE	전체 수신 크기 총합

V$REPL_SENDER_META

Replication 작동 시, Sender 의 메타데이터를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
SENDER_ID	Sender 식별자
TABLE_ID	대상 테이블 식별자
TABLE_TYPE	대상 테이블 유형
BEGIN_RID	대상 레코드의 시작 RID
END_RID	대상 레코드의 끝 RID

V$REPL_RECEIVER

Replication 작동 시, Receiver 의 정보를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
STATUS	Receiver Thread 의 작동상태
PAYLOAD_RECV_COUNT	Sender 로부터 받은 페이로드 개수
PAYLOAD_RECV_BYTES	Sender 로부터 받은 페이로드 크기 총합
QUEUE_REMAIN_COUNT	Receive Queue 에 남은 버퍼의 개수
NET_SEND_COUNT	전체 전송 횟수
NET_SEND_SIZE	전체 전송 크기 총합
NET_RECV_COUNT	전체 수신 횟수
NET_RECV_SIZE	전체 수신 크기 총합

V$REPL_RECEIVER_META

Replication 작동 시, Receiver 의 메타데이터를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
TABLE_ID	대상 테이블 식별자
TABLE_TYPE	대상 테이블 유형
BEGIN_RID	대상 레코드의 시작 RID
END_RID	대상 레코드의 끝 RID

V$REPL_READER

Replication 작동 시, Reader 의 정보를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
SENDER_ID	Sender 식별자
ID	Reader 식별자
STATUS	Reader Thread의 작동상태
FETCH_COUNT	FETCH 수행 횟수

V$REPL_READER_META

Replication 작동 시, Reader 의 메타데이터를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
SENDER_ID	Sender 식별자
ID	Reader 식별자
TABLE_ID	대상 테이블 식별자
TABLE_TYPE	대상 테이블 유형
BEGIN_RID	대상 레코드의 시작 RID
END_RID	대상 레코드의 끝 RID

V$REPL_WRITER

Replication 작동 시, Writer 의 정보를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
ID	Writer 식별자
STATUS	Writer Thread 의 작동상태
APPEND_COUNT	APPEND 수행 횟수

V$REPL_WRITER_META

Replication 작동 시, Writer 의 메타데이터를 표시한다.

컬럼 이름	설명
HOSTNAME	Replication 이 작동되는 Node 의 Hostname
ID	Writer 식별자
TABLE_ID	대상 테이블 식별자
TABLE_TYPE	대상 테이블 유형
BEGIN_RID	대상 레코드의 시작 RID
END_RID	대상 레코드의 끝 RID

Others

V$TABLES

V$로 시작하는 모든 Virtual Table 을 표시한다.

컬럼 이름	설명
NAME	테이블 이름
TYPE	테이블 유형
DATABASE_ID	데이터베이스 식별자
ID	테이블 식별자
USER ID	테이블을 생성한 사용자
COLCOUNT	컬럼의 갯수

V$COLUMNS

Virtual Table 의 컬럼 정보를 표시한다.

컬럼 이름	설명
NAME	컬럼명
TYPE	컬럼의 데이터 타입
DATABASE_ID	데이터베이스 식별자
ID	컬럼의 식별자
LENGTH	컬럼의 크기
TABLE_ID	테이블 식별자
FLAG	비공개 데이터
PART_PAGE_COUNT	(사용되지 않음)
PAGE_VALUE_COUNT	(사용되지 않음)
MINMAX_CACHE_SIZE	(사용되지 않음)
MAX_CACHE_PART_COUNT	(사용되지 않음)

Property

프로퍼티란 $MACHBASE_HOME/conf/machbase.conf 파일에 정의되어 있는 키-값 의 쌍을 의미한다.

이 값들은 마크베이스 서버가 시작할 때 설정되고 실행시 지속적으로 이용된다. 성능 튜닝을 위해서 이 값을 변경하려면 이 값들에 대한 의미를 이해하고, 주의 깊게 설정하여야 한다.

목차

CPU_AFFINITY_BEGIN_ID

마크베이스 서버가 사용할 CPU의 시작 번호이다. 마크베이스 서버의 CPU사용량을 조절하기 위해서 사용한다.

	Value
최소값	0
최대값	2 ^ 32 - 1
기본값	0

CPU_AFFINITY_COUNT

마크베이스 서버가 사용할 CPU의 수이다. 0으로 설정하면 마크베이스 서버가 모든 CPU를 사용한다.

	Value
최소값	0
최대값	2 ^ 32 - 1
기본값	0

CPU_COUNT

시스템에 설정된 CPU의 수를 지정한다. 이 값을 기반으로 마크베이스의 스레드 수를 결정한다. 0으로 지정한 경우에는 시스템의 모든 CPU를 사용한다.

	Value
최소값	0 (시스템에 물리적으로 설치된 CPU수)
최대값	2^32 -1
기본값	0

CPU_PARALLEL

CPU당 생성할 스레드의 수를 지정한다. 만약 이 값이 2이고 cpu의 수가 2인 경우, 두개의 CPU마다 병렬 스레드가 2개씩 생성되므로 병렬처리 스레드의 수가 4가 된다. 이 값이 너무 큰 경우, 메모리가 빨리 소모될 수 있다.

	Value
최소값	1
최대값	2^32 -1
기본값	1

DBS_PATH

마크베이스 서버의 기본 데이터가 저장될 경로를 지정한다. 기본값은 "?/dbs"로, `$MACHBASE_HOME/dbs` 를 의미한다.

	Value
기본값	?/dbs

DEFAULT_LSM_MAX_LEVEL

LSM인덱스의 기본 레벨을 설정한다. 인덱스를 생성할 때 MAX_LEVEL값을 입력하지 않으면 이 값이 적용된다.

	Value
최소값	0
최대값	3
기본값	2

DISK_BUFFER_COUNT

디스크 입출력을 위한 버퍼의 수를 지정한다.

	Value
최소값	1
최대값	2^32 - 1
기본값	16

DISK_COLUMNAR_INDEX_CHECKPOINT_INTERVAL_SEC

인덱스에 대한 체크포인트 주기를 설정한다. 너무 길게 설정할 경우, 인덱스 빌드에 오류가 발생할 수 있다.

	Value
최소값	1 (sec)
최대값	2^32 -1 (sec)
기본값	120 (sec)

DISK_COLUMNAR_INDEX_FDCACHE_COUNT

오픈한 인덱스 파티션 파일 디스크립터의 수를 지정한다.

	Value
최소값	0
최대값	2^32 -1
기본값	0

DISK_COLUMNAR_INDEX_SHUTDOWN_BUILD_FINISH

마크베이스 서버를 종료할 때, 인덱스 정보를 디스크에 모두 반영할 것인지를 설정한다. 이 값을 '1'로 설정하면 모든 인덱스 정보를 디스크에 반영하고 종료하므로 종료시 대기 시간이 길어질 수 있다.

	Value
최소값	0 (False)
최대값	1 (True)
기본값	0 (False)

DISK_COLUMNAR_PAGE_CACHE_MAX_SIZE

페이지 캐쉬의 최대 크기를 설정한다.

- TAG_TABLE_META_MAX_SIZE
- TRACE_LOGFILE_COUNT
- TRACE_LOGFILE_PATH
- TRACE_LOGFILE_SIZE
- UNIX_PATH
- VOLATILE_TABLESPACE_MEMORY_MAX_SIZE

	Value
최소값	0
최대값	2^64 - 1
기본값	2 * 1024 * 1024 * 1024

DISK_COLUMNAR_TABLE_CHECKPOINT_INTERVAL_SEC

테이블 데이터의 체크포인트 주기를 설정한다. 이 값이 너무 크면 재시작시 복구 시간이 매우 길어지고, 이 값이 너무 작으면 I/O가 자주 발생하여 전체 성능이 저하될 수 있다.

	Value
최소값	1 (sec)
최대값	2^32 -1 (sec)
기본값	120 (sec)

DISK_COLUMNAR_TABLE_COLUMN_FDCACHE_COUNT

테이블의 컬럼 데이터에 대한 오픈된 파일 설명자의 최대 수를 지정한다.

	Value
최소값	0
최대값	2^32 - 1
기본값	0

DISK_COLUMNAR_TABLE_COLUMN_MINMAX_CACHE_SIZE

_ARRIVAL_TIME 컬럼에 설정되는 기본 MINMAX 캐쉬의 크기를 설정한다.

	Value
최소값	0
최대값	2^64 - 1
기본값	100 *1024 * 1024

DISK_COLUMNAR_TABLE_COLUMN_PART_FLUSH_MODE

컬럼 파티션 파일의 자동 플러쉬 주기를 설정한다.

	Value
최소값	0 (sec)
최대값	2^32-1 (sec)
기본값	60 (sec)

DISK_COLUMNAR_TABLE_COLUMN_PART_IO_INTERVAL_MIN_SEC

파티션 파일을 디스크에 반영하는 주기를 설정한다. 파티션이 설정된 갯수보다 더 많은 데이터를 입력받으면 이 주기와 관계없이 디스크에 반영된다.

	Value
최소값	0 (sec)
최대값	2^32-1 (sec)
기본값	3 (sec)

DISK_COLUMNAR_TABLE_COLUMN_PARTITION_PRECREATE_COUNT

테이블에 대해서 사용할 예정인 컬럼 파티션 객체의 사전 생성 수를 정의한다.

	Value
최소값	1
최대값	2^32-1
기본값	3

DISK_COLUMNAR_TABLE_TIME_INVERSION_MODE

설정된 값 만큼 _ARRIVAL_TIME컬럼의 값이 감소하더라도 입력을 허용한다. 만약 0인 경우 _ARRIVAL_TIME컬럼 값의 최대값보다 작은 값이 입력되면 이는 오류로 처리된다.

	Value
최소값	0 (False)
최대값	1 (True)
기본값	1 (True)

DISK_COLUMNAR_TABLESPACE_DWFILE_EXT_SIZE

시작시 복구를 위해서 사용되는 더블 라이트 파일이 한번에 증가하는 크기를 지정한다.

	Value
최소값	1024 * 1024
최대값	2^32 - 1
기본값	1024 * 1024

DISK_COLUMNAR_TABLESPACE_DWFILE_INT_SIZE

파일 생성시에 더블라이트 파일이 확보하는 용량을 지정한다.

	Value
최소값	1024 * 1024
최대값	2^32 - 1
기본값	2* 1024 * 1024

DISK_COLUMNAR_TABLESPACE_MEMORY_EXT_SIZE

컬럼 파티션을 위해서 확보하는 메모리의 블록 크기를 지정한다.

	Value
최소값	1024 * 1024
최대값	2^64 - 1
기본값	2* 1024 * 1024

DISK_COLUMNAR_TABLESPACE_MEMORY_MAX_SIZE

로그 테이블에 의하여 할당된 최대 메모리 크기를 지정한다. 만약 서버가 이 값 이상의 메모리를 할당하게 되면, 메모리 사용량이 이 값 이하로 줄어들 때 까지 메모리 할당이 대기하므로 성능이 저하된다. 이 값은 물리적 메모리의 50~80% 정도로 설정할 것을 추천한다.

	Value
최소값	256 * 1024 * 1024
최대값	2^64 - 1
기본값	8 * 1024 * 1024 * 1024

DISK_COLUMNAR_TABLESPACE_MEMORY_MIN_SIZE

마크베이스 서버가 시작할 때, 메모리 할당에 의한 성능 저하를 막기 위해서 이 값 만큼 메모리를 사전 확보한다. 데이터 입력 버퍼로만 이 메모리를 사용하므로, 메모리가 충분할 경우에만 사용할 것을 추천한다.

Table 24. Range of values

	Value
최소값	1024 * 1024
최대값	2^64 - 1
기본값	100 * 1024 * 1024

DISK_COLUMNAR_TABLESPACE_MEMORY_SLOWDOWN_HIGH_LIMIT_PCT

컬럼 데이터 파일을 위한 메모리 사용량이 제한 값을 이 값을 다음과 같이 이용하여 계산하고, 초과한 경우 입력 성능을 저하시킨다.

```
DISK_COLUMNAR_TABLESPACE_MEMORY_MAX_SIZE * (DISK_COLUMNAR_TABLESPACE_MEMORY_SLOWDOWN_HIGH_LIMIT_PCT / 100)
```

	Value
최소값	0
최대값	100
기본값	80

DISK_COLUMNAR_TABLESPACE_MEMORY_SLOWDOWN_MSEC

컬럼 데이터 파일을 위한 메모리 사용량이 기준을 초과한 경우, 매 레코드 입력시에 다음의 대기 시간을 설정한다.

	Value
최소값	0 (msec)
최대값	2^32 - 1 (msec)
기본값	1 (msec)

DISK_IO_THREAD_COUNT

데이터를 디스크에 기록하는 입출력 스레드의 수를 설정한다.

	Value
최소값	1
최대값	2^32 -1
기본값	3

DISK_TABLESPACE_DIRECT_IO_FSYNC

Direct I/O를 실행할 경우, 데이터 파일에 대해서 fsync는 불필요하다. Direct I/O 를 사용할 경우 fsync를 사용하지 않도록 하면 데이터 I/O 성능을 향상시킬 수 있다 (0으로 설정).
Fsync를 수행하지 않아도 일반적 상황에서는 데이터 유실이 없으나 전원이 꺼지는 등의 장애 상황이 발생할 수 있는 경우에는 fsync를 수행하도록 설정해야 한다.

	Value
최소값	0
최대값	1
기본값	0

DISK_TABLESPACE_DIRECT_IO_READ

데이터 읽기 연산에 DIRECT I/O 를 사용할 것인지를 설정한다.

	Value
최소값	0
최대값	1
기본값	0

DISK_TABLESPACE_DIRECT_IO_WRITE

데이터 쓰기 연산에 DIRECT I/O 를 사용할 것인지 설정한다. 파일 시스템에 따라서 DIRECT I/O 지원하지 않는 경우(ex: ZFS), 0으로 설정한다.

- 참고 : ZFS DIRECT I/O https://github.com/openzfs/zfs/issues/224

	Value
최소값	0
최대값	1
기본값	1

DISK_TAG_AUTO_RECLAIM

TAG 데이터에 대해서 사용되지 않는 공간을 자동 확보할 것인지의 여부를 결정한다. 기본값인 1인 경우, 자동 공간 확보 기능이 동작하고 0 인 경우에는 동작하지 않으며 사용자가 ALTER TABLE문을 이용하여 해당 기능을 직접 수행해야 한다.

	Value
최소값	0
최대값	1
기본값	1

DUMP_APPEND_ERROR

이 값을 1로 설정하면 Append API 가 실패한 경우 `$MACHBASE_HOME/trc/machbase.trc` 파일에 에러 내용을 기록한다.
이 상황에서 append 성능이 매우 저하될 수 있으므로 테스트용으로만 사용할 것을 권장한다.

사용자 application에서 에러를 검사하고 싶으면 SQLAppendSetErrorCallback API 를 사용하는 것이 도움이 된다.

	Value
최소값	0
최대값	1
기본값	0

DUMP_TRACE_INFO

서버는 일정한 주기로 DBMS 시스템 상태 정보를 machbase.trc 파일에 주기적으로 기록하는데, 이 주기를 설정한다.
0으로 설정하면 기록하지 않는다.

	Value
최소값	0 (sec)
최대값	2^32 - 1 (sec)
기본값	60 (sec)

DURATION_BEGIN

DURATION 절을 지정하지 않은 SELECT 문에 대해서 기본을 설정하는 duration 값 중 시작시점을 설정한다.
만약 60을 설정해 두었다면, 현재 시각에서 60초 이전의 데이터를 검색하게 된다.

기본값은 0으로 모든 데이터를 검색한다.

	Value
최소값	0
최대값	2^32 - 1

	Value
기본값	0

DURATION_GAP

DURATION 절을 지정하지 않은 SELECT 문에 대해서 기본을 설정하는 duration 값 중 기간을 설정한다.

- 만약 60을 설정해 두었다면, 현재 시각에서 60초 동안의 데이터를 검색하게 된다.
- DURATION_BEGIN 값도 60이라면, 현재 시각에서 60초 이전부터 60초 동안의 데이터를 검색하게 된다.

기본값은 0으로 모든 데이터를 검색한다.

	Value
최소값	0
최대값	Non-zero
기본값	0

FEEDBACK_APPEND_ERROR

Append API 실행시 오류가 발생하였을 경우, 오류 데이터를 클라이언트에 전송할지를 설정한다. 0이면 클라이언트에 오류 데이터를 전송하지 않으며 1이면 클라이언트에 오류 정보를 전송한다.

	Value
최소값	0
최대값	1
기본값	1

GRANT_REMOTE_ACCESS

원격지에서 데이터베이스에 접근할 수 있는지를 결정한다. 0이면 원격지 접속이 차단된다.

	Value
최소값	0 (False)
최대값	1 (True)
기본값	1 (True)

HTTP_THREAD_COUNT

마크베이스의 웹 서버가 사용할 스레드의 개수를 설정 가능하다.

	Value
최소값	0
최대값	1024
기본값	2(Edge) 32(Fog/Cluster)

INDEX_BUILD_MAX_ROW_COUNT_PER_THREAD

인덱스 빌드 스레드가 인덱싱 되지 않은 레코드의 수가 이 값 이상이 되면 인덱스를 추가하기 시작한다.

	Value
최소값	1
최대값	2^32 - 1
기본값	100000

INDEX_BUILD_THREAD_COUNT

인덱스 생성 스레드의 수를 지정한다. 0으로 설정되면 인덱스를 생성하지 않는다.

	Value
최소값	0
최대값	2 ^ 32 - 1
기본값	3

INDEX_FLUSH_MAX_REQUEST_COUNT_PER_INDEX

인덱스당 최대 flush 요청 수를 지정한다.

	Value
최소값	0
최대값	2 ^ 32 - 1
기본값	3

INDEX_LEVEL_PARTITION_AGER_THREAD_COUNT

LSM 인덱스 생성시에 필요없는 인덱스 파일의 삭제를 위한 스레드의 갯수를 지정한다.

	Value
최소값	0
최대값	1024
기본값	1

INDEX_LEVEL_PARTITION_BUILD_MEMORY_HIGH_LIMIT_PCT

LSM 인덱스 생성을 위한 최대 메모리 사용량의 퍼센트로 설정한다. 이 퍼센트는 마크베이스가 사용하는 최대 메모리사용량 대비하여 설정된다. 메모리 사용량이 한도를 초과하면, LSM 파티션 병합은 중지된다.

	Value
최소값	0
최대값	100
기본값	70

INDEX_LEVEL_PARTITION_BUILD_THREAD_COUNT

LSM 인덱스의 생성을 위한 병합 연산을 수행하는 스레드의 수를 결정한다.

	Value
최소값	1
최대값	1024
기본값	3

LOOKUP_APPEND_UPDATE_ON_DUPKEY

Lookup 테이블에 Append 할 때 Primary Key가 중복일 경우 어떻게 처리할지 지정한다.

- 0 : Append 실패
- 1 : 해당 Primary Key 에 대해서 Row를 Update 한다.

	Value
최소값	0
최대값	1

	Value
기본값	0

MAX_QPX_MEM

GROUP BY, DISTINCT, ORDER BY 절을 수행하기 위해서 질의처리기가 이용하는 메모리의 최대 양을 설정한다.
하나의 질의문이 이보다 큰 값으로 메모리를 사용하게 되면 질의는 취소된다. 이때, 에러메시지를 클라이언트에 전송하고, machbase.trc 파일에 관련 내용이 기록된다.

	Value
최소값	1024 * 1024
최대값	2^64 - 1
기본값	500 * 1024 * 1024

MEMORY_ROW_TEMP_TABLE_PAGESIZE

Volatile table및 lookup 테이블을 위한 임시 테이블 스페이스의 페이지 크기를 설정한다. Volatile 테이블 및 lookup 테이블의 레코드들은 페이지에 저장되므로 volatile 을 위한 최대 레코드 크기보다 커야 한다.
페이지에 N개의 레코드를 입력하고 싶으면 이 값을 최대 레코드 크기 * N으로 설정해야 한다.

	Value
최소값	8 * 1024
최대값	2^32 - 1
기본값	32 * 1024

PID_PATH

마크베이스 서버 프로세스의 PID파일이 기록되는 위치를 지정한다. 기본값은 "?/conf"이며 이는 `$MACHBASE_HOME/conf` 를 의미한다.

	Value
기본값	?/conf

PID_PATH 값	PID 파일 위치 경로
지정되지 않음	$MACHBASE_HOME/conf/machbase.pid
?/test	$MACHBASE_HOME/test/machbase.pid
/tmp	/tmp/machbase.pid

PORT_NO

마크베이스 서버 프로세스가 클라이언트와 통신하기 위한 TCP/IP 포트를 지정한다. 기본값은 5656이다.

	Value
최소값	1024
최대값	65535
기본값	5656

PROCESS_MAX_SIZE

마크베이스 서버 프로세스인 machbased 프로그램이 사용하는 최대 메모리 사이즈를 지정한다. 이 제한값 이상의 메모리를 사용하려고 하면 서버는 다음과 같이 동작하여 메모리의 사용량을 줄이려고 시도한다. 메모리 제한을 초과한 경우, 다음의 방법으로 메모리 사용량을 줄인다.

- 데이터 입력을 중지하거나 오류로 처리
- 인덱스 생성 속도를 떨어뜨림

이 경우, 성능이 매우 저하되므로, 메모리 과다 사용 원인을 찾아서 해결하여야 한다.

	Value
최소값	1024 * 1024 * 1024

	Value
최대값	2^64 - 1
기본값	8 * 1024 * 1024 * 1024

QUERY_PARALLEL_FACTOR

병렬 질의 실행기의 실행 스레드의 수를 지정한다.

	Value
최소값	1
최대값	100
기본값	8

ROLLUP_FETCH_COUNT_LIMIT

롤업 스레드가 한번에 패치해올 데이터양을 제한한다.

0으로 설정할 경우 제한이 없다.

	Value
최소값	0
최대값	2^32 - 1
기본값	3000000

RS_CACHE_APPROXIMATE_RESULT_ENABLE

결과값 캐쉬의 추측 모드(approximate result mode)를 사용할지의 여부를 결정한다. 이 값이 1이면 결과값 캐쉬를 사용할 때, 추측 값을 얻고(매우 빠르지만 데이터가 부정확할 수 있다.) 0 이면 정확한 값을 얻는다.

	Value
최소값	0 (False)
최대값	1 (True)
기본값	0 (False)

RS_CACHE_ENABLE

결과값 캐쉬를 사용할 지의 여부를 결정한다.

	Value
최소값	0 (False)
최대값	1 (True)
기본값	1 (True)

RS_CACHE_MAX_MEMORY_PER_QUERY

결과값 캐쉬가 사용할 메모리의 양을 설정한다. 특정 질의 결과의 메모리 사용량이 이 값을 초과하면, 해당 질의의 결과는 결과값 캐쉬에 저장되지 않는다.

	Value
최소값	1024
최대값	2^64 - 1
기본값	16 * 1024 * 1024

RS_CACHE_MAX_MEMORY_SIZE

결과값 캐쉬의 최대 메모리 사용량을 지정한다.

	Value
최소값	32 * 1024
최대값	2^64 - 1
기본값	512 * 1024 * 1024

RS_CACHE_MAX_RECORD_PER_QUERY

결과값 캐쉬에 저장되는 최대 레코드 갯수이다. 만약 질의의 결과 레코드의 수가 이 값 이상이면 해당 질의 결과값은 캐쉬에 저장하지 않는다.

	Value
최소값	1
최대값	2^64 - 1
기본값	10000

RS_CACHE_TIME_BOUND_MSEC

특정 질의가 매우 빠르게 실행된 경우에는 그 결과값을 결과값 캐쉬에 저장하지 않는 것이 메모리 사용량을 줄일 수 있으므로 캐쉬에 저장하지 않는것이 좋다.
이 값은 어느 정도 빨리 실행된 질의를 캐쉬에 저장하지 않을지를 결정한다. 0으로 설정된 경우에는 모든 질의결과를 결과집합캐쉬에 저장한다.

	Value
최소값	0 (msec)
최대값	2^64 - 1 (msec)
기본값	1000 (msec)

SHOW_HIDDEN_COLS

_ARRIVAL_TIME 컬럼은 기본 설정으로는 SELECT * FROM 질의에 의해서 표시되지 않는다. 그러나 이 값이 1로 설정된 경우에는 해당 컬럼을 표시한다.

	Value
최소값	0
최대값	1
기본값	0

TABLE_SCAN_DIRECTION

태그 테이블의 스캔 방향을 설정할 수 있다. 프로퍼티 값은 -1,0, 1중 택일이며 기본값은 0이다.

- -1 : 역방향 스캔
- 0 : Tag Table(정방향 스캔), Log Table(역방향 스캔)
- 1 : 정방향 스캔

	Value
최소값	-1
최대값	1
기본값	0

TAGDATA_AUTO_META_INSERT

> 5.5 에서는 TAGDATA_AUTO_NAME_INSERT 이다. 값의 범위도 0/1 이다.
> 5.7 이하에서는 기본값이 1 이다.

TAGDATA 테이블에 APPEND/INSERT 를 통해 데이터를 입력할 때, 일치하는 TAG_NAME 이 없을 경우 어떻게 처리할 것인지를 정한다.

- 0 : 입력이 실패한다.
- 1 : 입력을 원하는 TAG_NAME 값을 입력한다. 추가 메타데이터 컬럼이 존재할 경우, 해당 컬럼의 값은 모두 NULL 로 입력된다.
- 2 : 입력을 원하는 TAG_NAME 값과 함께, 추가 메타데이터 컬럼 값도 같이 입력한다.

- APPEND 에서만 유효한 설정이며, INSERT 는 추가 메타데이터 컬럼 값을 입력할 수 없기 때문에 1과 같이 작동한다.
- 이 설정을 한 이후에는, APPEND 에서 반드시 메타데이터 컬럼 값까지 포함시킨 APPEND Parameter 로 입력해야 한다.

	Value
최소값	0
최대값	2
기본값	2

TAG_TABLE_META_MAX_SIZE

TAGDATA Table 생성 시 Metadata 영역을 보관할 메모리의 최대 크기를 설정한다.

	Value
최소값	1024*1024
최대값	2^32-1
기본값	100*1024*1024

TRACE_LOGFILE_COUNT

TRACE_LOGFILE_PATH에 생성되는 로그 트레이스 파일의 최대 수를 지정한다. 디스크 공간을 절약하기 위해서, 최대 개수 이상의 로그파일이 생성되면 가장 오래된 로그파일을 삭제한다.

로그 트레이스 파일의 최대 개수 이상의 로그파일이 생성되어 가장 오래된 파일이 삭제될 경우 삭제된 파일의 이름이 가장 최신의 로그파일로 저장이 된다.

	Value
최소값	1
최대값	2^32 - 1
기본값	1000

TRACE_LOGFILE_PATH

로그 트레이스 파일들(machbase.trc, machadmin.trc, machcollector.trc, machsql.trc)의 경로를 설정한다.
이 파일들은 마크베이스의 시작, 종료, 실행시 내부 정보를 지속적으로 기록한다. 기본값인 ?/trc의 의미는 `$MACHBASE_HOME/trc` 를 의미한다.

	Value
기본값	?/conf

TRACE_LOGFILE_PATH 값	trc 디렉터리 위치
지정되지 않음	$MACHBASE_HOME/trc/
?/test	$MACHBASE_HOME/test/
/tmp	/tmp/

TRACE_LOGFILE_SIZE

로그 트레이스 파일의 최대 크기를 설정한다. 만약 크기 이상의 데이터를 기록하여야 한다면, 신규로 log 파일을 생성할 것이다.

	Value
최소값	10 * 1024 * 1024
최대값	2^32 - 1
기본값	10 * 1024 * 1024

UNIX_PATH

Unix domain socket 파일의 경로를 설정한다. 사용자가 설정하지 않았을 경우의 기본 값은 ?/conf/machbase-unix 이다.

	Value
기본값	?/conf/machbase-unix

VOLATILE_TABLESPACE_MEMORY_MAX_SIZE

시스템의 모든 volatile, lookup 테이블의 메모리 사용량 총계의 한도를 설정한다.

	Value
최소값	0
최대값	2^64 - 1
기본값	2 * 1024 * 1024 * 1024

Property (Cluster)

Property 와 별개로, Cluster Edition 에서만 사용 가능한 Property 를 정리한다.

목차

CLUSTER_LINK_ACCEPT_TIMEOUT

특정 Node와 연결할 때, Accept 후 Handshake 메시지를 수신할 때까지의 Timeout.

Timeout 이후까지 수신에 실패하면, 해당 연결은 실패한다.

기본값은 5초.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	5000000

CLUSTER_LINK_BUFFER_SIZE

ⓘ 5.6 부터 사용 가능합니다.

이 크기가 모자라면 송신시 버퍼가 비워질 때 까지 재시도하게 된다.송신/수신 버퍼의 크기.

기본값은 32M.

(byte)	Value
최소값	1024768
최대값	2^32 - 1
기본값	33554432 (32M)

CLUSTER_LINK_CHECK_INTERVAL

특정 Node와 연결된 Socket들을 검사하는, Timeout Thread의 검사 주기.

RECEIVE_TIMEOUT, SESSION_TIMEOUT 을 검사하는 Timeout Thread가 존재한다.
주기를 짧게 할 수록, 자주 검사하지만 Timeout 판단은 아래의 값에 따라 이루어진다.

기본값은 1초.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	1000000

CLUSTER_LINK_CONNECT_RETRY_TIMEOUT

특정 Node와 연결이 실패한 이후, 재연결 시도를 반복하는 Timeout

Timeout 이후까지 재연결되지 않는다면, 완전히 단절되었다고 판단한다.

기본값은 1분.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	60000000

CLUSTER_LINK_CONNECT_TIMEOUT

특정 Node와 연결을 시도할 때, 기다리는 시간.

Timeout 이후까지 연결되지 않는다면, CLUSTER_LINK_CONNECT_RETRY_TIMEOUT 이 지나기 전 까지 재연결을 시도한다.

기본값은 5초.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	5000000

CLUSTER_LINK_ERROR_ADD_ORIGIN_HOST

Cluster 간 통신 중 발생하는 에러 메시지에, 오류가 발생한 호스트 이름을 추가할지 여부를 선택할 수 있다.

자세한 에러 메시지를 표시하고자 한다면, 해당 Property를 켜야 한다.

기본값은 'No' (=0). 호스트 이름이 출력되지 않는다.

(boolean)	Value
최소값	0
최대값	1
기본값	0

CLUSTER_LINK_HANDSHAKE_TIMEOUT

특정 Node와 Cluster Socket으로 연결된 상태에서, Handshake 메시지를 수신할 때까지의 Timeout

연결이 막 완료된 두 Node는, 연결 상태를 점검하는 차원에서 작은 크기의 Handshake 메시지를 주고 받는다.
Accept한 Node가 Handshake 메시지를 먼저 보내는데, 그 응답을 기다리는 시간을 여기서 설정한다.

기본값은 5초.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	5000000

CLUSTER_LINK_HOST

특정 Node와 Cluster Socket 을 연결하기 위한, 현재 Node의 호스트 이름

(string)	Value
기본값	localhost

CLUSTER_LINK_LONG_TERM_CALLBACK_INTERVAL

Cluster Socket 으로 수신 되는 메시지를 처리할 Receive Callback 이 수행한 시간을 Long-Term Callback 으로 인식할 시간

수신 Thread의 개수가 제한적이므로, 가급적이면 Receive Callback은 오랜 시간 동안 메시지를 처리하고 있으면 안 된다.
이 시간이 지나도록 Receive Callback이 메시지를 처리하고 있다면, Long-Term Callback 으로 인식하고 Trace Log에 그 기록을 남긴다.

기본값은 1초.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	1000000

CLUSTER_LINK_LONG_WAIT_INTERVAL

Cluster Socket 으로 수신 되는 메시지가 도착할 때 까지의 시간을 Long-Wait Message 로 인식할 시간

수신 시작~수신 종료 까지의 시간이 길면 네트워크 환경의 문제로 볼 수 있다.
이 시간이 지나도록 수신 메시지가 도착하지 않는다면, Long-Wait Message 로 인식하고 Trace Log에 그 기록을 남긴다.

기본값은 1초.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	1000000

CLUSTER_LINK_MAX_LISTEN

특정 Node와 연결할 때, Socket의 Accept Queue 의 최대 숫자

(count)	Value
최소값	1
최대값	2^32 - 1
기본값	512

CLUSTER_LINK_MAX_POLL

특정 Node와 통신할 때, Poll에 의해서 한번에 조회할 수 있는 최대 Event 수

(count)	Value
최소값	1
최대값	2^32 - 1
기본값	4096

CLUSTER_LINK_PORT_NO

특정 Node와 Cluster Socket 을 연결하기 위한, 현재 Node의 포트 번호

(port)	Value
최소값	1024
최대값	65535
기본값	3868

CLUSTER_LINK_RECEIVE_TIMEOUT

Timeout Thread가, 마지막 수신 이후로 연결이 끊긴 것을 판단할 때 까지의 Timeout

Cluster Node 간 연결은, 수신이 완료되면 종료되기 때문에 '연결 리스트' 에 존재하는 연결들은 지속적으로 수신을 받고 있어야 한다.
이 시간이 지나도록 마지막 수신 시각이 갱신되지 않으면, Timeout Thread는 Trace Log에 기록을 남기고 해당 Socket을 닫는다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	30000000

CLUSTER_LINK_REQUEST_TIMEOUT

Cluster Socket에서 요청 메시지를 보냈을 때, 요청에 대한 응답이 올 때 까지의 Timeout

특정 메시지의 경우 Request 이후 Answer 전송까지 대기할 수 있는 시간을 따로 지정한다.
이 시간이 지나도록 응답 메시지가 도착하지 않으면, Trace Log에 기록을 남기고 해당 Socket을 닫는다.

기본값은 60초. 메시지 종류와 수신 처리가 어떻게 될지 모르므로, Timeout이 길다.

(usec)	Value
최소값	0

최대값	2^64 - 1
기본값	60000000

CLUSTER_LINK_SEND_RETRY_COUNT

> 5.6 부터 사용 가능합니다.

재시도를 할 때 마다 1ms 씩 쉬게 된다. 이 회수를 넘어서서 재시도하게 될 경우 연결을 해제하게 된다.송신 버퍼가 비워질 때 까지 송신을 재시도하는 회수.

기본값은 5000.

(count)	Value
최소값	0
최대값	2^32 - 1
기본값	5000

CLUSTER_LINK_SEND_TIMEOUT

Cluster Socket을 통해 메시지를 송신할 때 설정하는 Timeout

송신할 때 해당 Timeout 을 설정하며,
Timeout 까지 송신이 완료되지 않으면 Trace Log에 그 기록을 남긴다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	30000000

CLUSTER_LINK_SESSION_TIMEOUT

Timeout Thread가, 특정 세션에서 마지막 수신 이후로 연결이 끊긴 것을 판단할 때 까지의 Timeout

Cluster 연결은, 내부적으로 모든 메시지의 세션을 관리하고 있다. 갑자기 세션 정리를 하지 못하게 된 상황에서 필요한 Property 이다.
이 시간이 지나도록 해당 세션에 대한 마지막 수신 시각이 갱신되지 않으면, Timeout Thread는 Trace Log에 기록을 남기고 해당 세션을 닫는다.

기본값은 1시간.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	3600000000

CLUSTER_LINK_THREAD_COUNT

특정 Node와 통신할 때, 수신된 메시지를 처리할 Thread의 수

Cluster의 규모가 커지거나, 처리해야 할 연산의 개수가 많아져서 수신 가능한 Thread 가 여유가 없을 때 늘릴 수 있다.

(count)	Value
최소값	1
최대값	4096
기본값	8

CLUSTER_QUERY_STAT_LOG_ENABLE

수행한 질의에 대한 통계정보를 trace log에 출력한다.

(boolean)	Value

최소값	0
최대값	1
기본값	0

CLUSTER_REPLICATION_BLOCK_SIZE

Cluster Edition 에서, Node 추가로 Replication 을 진행할 때, 한번에 실어 보내는 데이터 크기.

Replication Active 가 되는 Warehouse (=전송을 하는 Warehouse) 에 직접 Property 를 적용해야 한다.

기본값은 640KB 이다.

(size)	Value
최소값	64 * 1024
최대값	100 * 1024 * 1024
기본값	640 * 1024 (655360)

CLUSTER_WAREHOUSE_DIRECT_DML_ENABLE

Cluster Edition 에서, Warehouse 에 곧바로 접속해 DML 을 수행할 수 있도록 한다.

- 1 : 수행 가능
- 2 : 수행 불가능. 에러가 반환된다.

Warehouse 에 직접 DML 을 수행할 경우 Broker 를 통한 것보다 성능 이점이 있지만, 동일 Group 에 DML 이 전파되지 않는 문제가 있다.
따라서, 데이터 불일치로 인한 비상 복구용 혹은 Group 의 데이터 불일치를 감안해도 되는 경우에 한해 사용한다.

원하는 특정 Warehouse 에 직접 Property 를 적용해야 한다.

기본값은 0이다.

> 해당 Property 를 켠 채로 Group 내 Warehouse 간의 데이터 차이가 발생하더라도, Coordinator 는 데이터 불일치 여부를 별도로 검사하지 않는다.

(boolean)	Value
최소값	0
최대값	1
기본값	0

COORDINATOR_DBS_PATH

Coordinator 의 데이터 파일이 생성될 디렉터리를 지정한다.

기본값은 ?/dbs 로 설정되어 있으며, ? 는 $MACHBASE_COORDINATOR_HOME 환경변수로 치환된다.
이는 환경변수 `$MACHBASE_COORDINATOR_HOME/dbs` 디렉터리라는 의미이다.

Coordinator 에 적용해야 하며, 다른 Node 에는 아무런 효과가 없다.

(path)	Value
기본값	?/dbs

COORDINATOR_DDL_REQUEST_TIMEOUT

Coordinator가 Node에게 DDL 수행을 요청한 후 대기할 때 까지의 Timeout

이 값은 Coordinator가 각 Node에게 DDL 수행을 요청한 후 대기할 때 까지를 말한다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	300000000

COORDINATOR_DDL_TIMEOUT

Broker가 CC를 통해 Coordinator에게 DDL 수행을 요청한 후 대기할 때 까지의 Timeout

이 값은 Broker가 Cluster 전체에 대한 DDL 수행을 Coordinator에게 요청한 후 대기할 때 까지를 말한다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	3600000000

COORDINATOR_DECISION_DELAY

Coordinator가 상태 변경을 요청하고 실제로 반영할 때 까지의 Timeout.

이 시간이 지나도록 실제로 상태가 변경되지 않는 경우, Cluster 상태를 비활성화시킨다.
만약 Warehouse Active의 상태가 변경되지 않았는데 연결된 Standby가 존재하는 경우, Fail-Over 작업을 시작한다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	1000000

COORDINATOR_DECISION_INTERVAL

Coordinator가 상태 변경을 얼마나 자주 할지 결정할 시간.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	1000000

COORDINATOR_HOST_RESOURCE_ENABLE

Coordinator가 Cluster Node들의 Host Resource 수집 여부

(boolean)	Value
최소값	0(false)
최대값	1(true)
기본값	0(false)

COORDINATOR_HOST_RESOURCE_COLLECT_INTERVAL

Cluster Node들이 Host Resource를 수집하는 주기

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	1000000

COORDINATOR_HOST_RESOURCE_INTERVAL

Coordinator가 Node들과 Host Resource를 주고받는 주기

(usec)	Value

최소값	0
최대값	2^64 - 1
기본값	1000000

COORDINATOR_HOST_RESOURCE_REQUEST_TIMEOUT

Coordinator가 Node들에게 Host Resource 정보를 요청한 이후 대기할 시간

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	10000000

COORDINATOR_NODE_REQUEST_TIMEOUT

Coordinator가 Node에게 명령을 수행하도록 요청한 후 대기할 때 까지의 Timeout

Add/Remove-node, Add/Remove-Package 등의 Node 명령 수행이 포함되어 있어, 짧은 시간으로 잡을 경우 해당 명령 처리가 완료되지 못할 수 있다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	600000000

COORDINATOR_NODE_TIMEOUT

Coordinator가 Node의 장애를 판단하기 까지 기다릴 시간.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	30000000

COORDINATOR_STARTUP_DELAY

Coordinator 시작 직후 Decision Thread를 작동시킬 때 까지의 유예 시간.

Cluster 전체 구동에 오랜 시간이 소요되는 경우, 해당 값을 크게 설정해서 Coordinator의 Node 제어를 더욱 늦게 시작할 수 있다.
전체 구동도 하기 전에 Decision Thread가 작동하는 경우, Coordinator가 오판할 가능성이 높아진다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	3000000

COORDINATOR_STATUS_NODE_INTERVAL

Coordinator가 Node들과 상태 조회 메시지를 주고 받을 주기

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	1000000

COORDINATOR_STATUS_NODE_REQUEST_TIMEOUT

Coordinator가 Node들에게 상태 조회 요청을 한 이후 대기할 시간.

해당 시간동안 상태 조회 응답이 없으면, Coordinator는 해당 Node의 상태를 갱신하지 않고 계속 진행한다.
네트워크 상황이 좋지 않은데 상태 갱신을 반드시 해야 하는 경우엔, 값을 늘리는 것을 고려해 볼 수 있다.
대신, 상태 조회 응답이 없을 경우엔 값을 늘린 만큼 Coordinator에서 반드시 기다리게 된다.

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	5000000

COORDINATOR_DISK_FULL_UPPER_BOUND_RATIO

Cluster 로 구성중인 일부 서버의 디스크 사용량이 프로퍼티 값을 넘어가면 해당 host 가 속한 group 이 DISKFULL 상태로 전환된다.
DISKFULL 상태의 group 에 대해서는 입력이 제한되고 조회 및 삭제만 가능하다.

프로퍼티 값이 0 인 경우 해당 기능이 disable 된다.

(percent)	Value
최소값	0
최대값	99
기본값	0

COORDINATOR_DISK_FULL_LOWER_BOUND_RATIO

DISKFULL 상태로 동작중인 서버의 디스크 사용량이 프로퍼티 값 이하로 떨어질 경우 해당 group 이 normal 상태로 전환된다.

프로퍼티 값이 0 인 경우 해당 기능이 disable 된다.

(percent)	Value
최소값	0
최대값	99
기본값	0

DEPLOYER_DBS_PATH

Deployer 의 데이터 파일이 생성될 디렉터리를 지정한다.

기본값은 ?/dbs 로 설정되어 있으며, ? 는 $MACHBASE_DEPLOYER_HOME 환경변수로 치환된다.
이는 환경변수 `$MACHBASE_DEPLOYER_HOME /dbs` 디렉터리라는 의미이다.

Deployer 에 적용해야 하며, 다른 Node 에는 아무런 효과가 없다.

(path)	Value
기본값	?/dbs

EXECUTION_STAGE_MEMORY_MAX

Cluster Edition 에서, SELECT 쿼리를 수행하는 Stage Thread 가 사용하는 Memory 의 최대 크기.

각 Stage 의 최대 크기이므로, Stage 개수가 늘어나는 복잡한 SELECT 쿼리의 경우 요구 메모리가 더 커질 수 있다.
최대 크기를 넘는 Stage 가 존재하는 경우, 해당 Stage 는 취소되고 Query 역시 에러와 함께 취소된다.

원하는 특정 Warehouse 에 직접 Property 를 적용해야 한다.

기본값은 1GB 이다.

(size)	Value
최소값	1024
최대값	2^64 - 1
기본값	1024 *1024 * 1024

HTTP_ADMIN_PORT

MWA 또는 machcoordinatoradmin 으로부터 요청을 받을 port number

(port)	Value
최소값	1024
최대값	65535
기본값	5779

HTTP_CONNECT_TIMEOUT

machcoordinatoradmin 과 연결할 때 사용하는 timeout

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	30000000

HTTP_RECEIVE_TIMEOUT

machcoordinatoradmin 과 통신할 때 사용하는 timeout

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	3600000000

HTTP_SEND_TIMEOUT

machcoordinatoradmin 과 통신할 때 사용하는 timeout

(usec)	Value
최소값	0
최대값	2^64 - 1
기본값	60000000

INSERT_BULK_DATA_MAX_SIZE

Append 또는 INSERT-SELECT 수행 시 입력 data block의 최대 크기

(size)	Value
최소값	1024
최대값	10 * 1024 * 1024
기본값	1024 * 1024

INSERT_RECORD_COUNT_PER_NODE

입력 수행시 warehouse group 전환을 유도하는 data 입력 개수.

(count)	Value
최소값	1
최대값	2^64 - 1
기본값	10000

LOOKUPNODE_COMMAND_RETRY_MAX_COUNT

Lookup 노드에 명령 및 접속 실패시 재시도 횟수

(count)	Value
최소값	1
최대값	3600
기본값	30

STAGE_RESULT_BLOCK_SIZE

하나의 stage 에서 만드는 최대 block 크기

(size)	Value
최소값	1024
최대값	2^64 - 1
기본값	1024 * 1024

SQL 레퍼런스

자료형

데이터 타입 테이블

목차

타입 명	설명	값 범위	NULL 값
short	16비트 부호 있는 정수형 데이터 타입	-32767 ~ 32767	-32768
ushort	16비트 무부호 정수형 데이터 타입	0 ~ 65534	65535
integer	32비트 부호 있는 정수형 데이터 타입	-2147483647 ~ 2147483647	-2147483648
uinteger	32비트 무부호 정수형 데이터 타입	0 ~ 4294967294	4294967295
long	64비트 부호 있는 정수형 데이터 타입	-9223372036854775807 ~ 9223372036854775807	-9223372036854775808
ulong	64비트 무부호 정수형 데이터 타입	0~18446744073709551614	18446744073709551615
float	32비트 부동 소수점 데이타 타입	-	-
double	64비트 부동 소수점 데이타 타입	-	-
datetime	시간 및 날짜	1970-01-01 00:00:00 000:000:000 ~ 2262-04-11 23:47:16.854:775:807	-
varchar	가변길이 문자열 (UTF-8)	길이 : 1 ~ 32768 (32K)	-
ipv4	Version 4의 인터넷 주소 타입 (4 바이트)	"0.0.0.0" ~ "255.255.255.255"	-
ipv6	Version 6의 인터넷 주소 타입 (16 바이트)	"0000:0000:0000:0000:0000:0000:0000:0000" ~ "FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF"	-
text	텍스트 데이터형 (키워드 인덱스 생성가능)	길이 : 0 ~ 64M	-
binary	바이너리 데이터형 (인덱스 생성 불가능)	길이 : 0 ~ 64M	-
json	json 데이터 타입	json data 길이 : 1 ~ 32768 (32K) json path 길이 : 1 ~ 512	-

short

C 언어의 16비트 부호있는 정수형 데이터와 동일하다. 최소 음수값에 대해서는 NULL로 인식한다. "int16" 이라고 표시해도 된다.

integer

C 언어의 32비트 부호있는 정수형 데이터와 동일하다. 최소 음수값에 대해서는 NULL로 인식한다. "int32" 또는 "int" 라고 표시해도 된다.

long

C 언어의 64비트 부호있는 정수형 데이터와 동일하다. 최소 음수값에 대해서는 NULL로 인식한다. "int64" 라고 표시해도 된다.

float

C 언어의 32비트 부동 소수점 데이터타입 float와 동일하다. 양수 최대값에 대해 NULL로 인식한다.

double

C 언어의 64비트 부동 소수점 데이터타입 double과 동일하다. 양수 최대값에 대해 NULL로 인식한다.

datetime

마크베이스에서는 이 타입은 1970년 1월 1일 자정 이후에 흘러간 시간의 나노값을 유지한다.

따라서, 마크베이스는 datetime 타입 관련 모든 함수에 대해서 nano 단위까지 값을 처리할 수 있도록 제공한다.

varchar

가변 문자열 데이터 타입이며, 길이는 최대 32K byte까지 생성이 가능하다.

이 길이의 기준은 영문 1자를 기준으로 한 것이기 때문에 UTF-8에서 표현하는 실제 출력되는 문자 개수와는 서로 다르며, 적절한 길이로 설정해야 한다.

IPv4

이 타입은 인터넷 프로토콜 버전 4에서 사용되는 주소를 저장할 수 있는 타입이다.

내부적으로 4바이트를 사용하여 표현하고 있으며, "0.0.0.0" 부터 "255.255.255.255"까지 모두 표현 가능하다.

IPv6

이 타입은 인터넷 프로토콜 버전 6에서 사용되는 주소를 저장할 수 있는 타입이다.

내부적으로 16바이트를 사용하여 표현하고 있으며, "0000:0000:0000:0000:0000:0000:0000:0000" 부터 "FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF" 까지 표현 가능하다.

데이터 입력시에는 축약형도 지원하기 때문에 : 기호를 활용하여 다음과 같이 표현할 수 있다.

- "::FFFF:1232" : 앞자리가 모두 0일 경우
- "::FFFF:192.168.0.3" : IPv4 형 호환형 지원
- "::192.168.3.1" : deprecated 된 IPv4 형 호환형 지원

text

이 타입은 VARCHAR 의 크기를 넘어선 문자열 혹은 문서를 저장하기 위한 데이터 타입이다.

이 데이터 타입은 키워드 인덱스를 통해 검색이 가능하며, 최대 64메가 바이트의 텍스트를 저장할 수 있다.
이 타입은 주로 큰 텍스트 파일을 별도의 컬럼으로 저장하고, 검색하기 위한 용도로 사용된다.

binary

이 타입은 비정형 데이터를 컬럼형태로 저장하기 위해 지원되는 타입이다.

이미지나 동영상 혹은 음성과 같은 바이너리 데이터를 저장하는데 사용되는데 이 타입에 대해 인덱스를 생성하여 검색할 수 없다.
저장하기 위한 최대 데이터 크기는 TEXT 타입과 동일하게 64 메가 바이트까지 가능하다.

json

이 타입은 json 데이터를 저장하기 위해 지원되는 타입이다.

json이란, "Key-Value"의 쌍으로 이루어진 데이터 오브젝트를 텍스트형으로 저장한 포맷이다.
저장하기 위한 최대 데이터 크기는 varchar 타입과 동일하게 32K byte까지 생성이 가능하다.

SQL 자료형 표

아래는, 마크베이스 자료형과 대응되는 SQL 자료형, C 자료형을 표로 나타냈다.

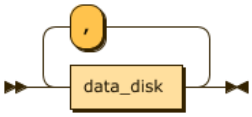
Machbase Datatype	Machbase CLI Datatype	SQL Datatype	C Datatype	Basic types for C	Description
short	SQL_SMALLINT	SQL_SMALLINT	SQL_C_SSHORT	int16_t (short)	16비트 부호 있는 정수형 데이터 타입
ushort	SQL_USMALLINT	SQL_SMALLINT	SQL_C_USHORT	uint16_t (unsigned short)	16비트 무부호 정수형 데이터 타입

Machbase Datatype	Machbase CLI Datatype	SQL Datatype	C Datatype	Basic types for C	Description
integer	SQL_INTEGER	SQL_INTEGER	SQL_C_SLONG	int32_t (int)	32비트 부호 있는 정수형 데이터 타입
uinteger	SQL_UINTEGER	SQL_INTEGER	SQL_C_ULONG	uint32_t (unsigned int)	32비트 무부호 정수형 데이터 타입
long	SQL_BIGINT	SQL_BIGINT	SQL_C_SBIGINT	int64_t (long long)	64비트 부호 있는 정수형 데이터 타입
ulong	SQL_UBIGINT	SQL_BIGINT	SQL_C_UBIGINT	uint64_t (unsigned long long)	64비트 무부호 정수형 데이터 타입
float	SQL_FLOAT	SQL_REAL	SQL_C_FLOAT	float	32비트 부동 소수점 데이터 타입
double	SQL_DOUBLE	SQL_FLOAT, SQL_DOUBLE	SQL_C_DOUBLE	double	64비트 부동 소수점 데이터 타입
datetime	SQL_TIMESTAMP SQL_TIME	SQL_TYPE_TIMESTAMP SQL_BIGINT SQL_TYPE_TIME	SQL_C_TYPE_TIMESTAMP SQL_C_UBIGINT SQL_C_TIME	char * (YYYY-MM-DD HH24:MI:SS 출력 포맷) int64_t (timestamp: nano seconds) struct tm	시간 및 날짜
varchar	SQL_VARCHAR	SQL_VARCHAR	SQL_C_CHAR	char *	문자열
ipv4	SQL_IPV4	SQL_VARCHAR	SQL_C_CHAR	char * (ip 문자열 입력) unsigned char[4]	Version 4 인터넷 주소 타입
ipv6	SQL_IPV6	SQL_VARCHAR	SQL_C_CHAR	char * (ip 문자열 입력) unsigned char[16]	Version 6 인터넷 주소 타입
text	SQL_TEXT	SQL_LONGVARCHAR	SQL_C_CHAR	char *	텍스트
binary	SQL_BINARY	SQL_BINARY	SQL_C_BINARY	char *	바이너리 데이터
json	SQL_JSON	SQL_JSON	SQL_C_CHAR	json_t	json 데이터 타입

CREATE TABLESPACE

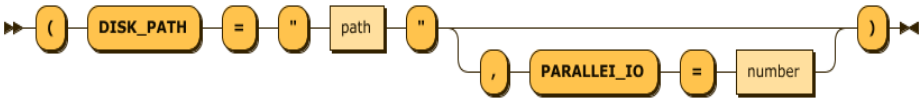
create_tablespace_stmt:

datadisk_list:



data_disk:

data_disk_property:




```
create_tablespace_stmt ::= 'CREATE TABLESPACE' tablespace_name 'DATADISK' datac
datadisk_list ::= data_disk ( ',' data_disk )*
data_disk ::= disk_name data_disk_property
data_disk_property ::= '(' 'DISK_PATH' '=' '"' path '"' ( ',' 'PARALLEL_IO' '='
```

```
-- 예제
create tablespace tbs1 datadisk disk1 (disk_path=""); -- $MACHBASE_HOME/dbs/ 에
create tablespace tbs1 datadisk disk1 (disk_path="tbs1_disk1"); -- $MACHBASE_HO
create tablespace tbs2 datadisk disk1 (disk_path="tbs2_disk1", parallel_io = 5)
create tablespace tbs1 datadisk disk1 (disk_path="tbs1_disk1", parallel_io = 16
```

목차

CREATE TABLESPACE 구문은 로그 Table 또는 로그 Table의 Index가 저장될 Tablespace를 $MACHBASE_HOME/dbs/ 에 생성한다.

Tablespace는 여러 개의 Disk를 가질 수 있다. Table과 Index의 데이터를 저장하는 각각의 Partition File들이 저장될 때, Tablespace에 속한 Data Disk들에 분산되어 저장된다.

만약 2개 이상의 Disk를 사용 시 Index와 Table의 여러 File이 각 Disk에 분산 저장되고, 각각의 Device에서 Parallel 하게 IO가 수행되어 Disk 개수가 늘어날수록 Disk I/O Throughput 이 높아져 다량의 Data를 빠르게 Disk에 저장할 수 있는 이점이 있다.

또한, Table과 Index의 Tablespace를 별도로 생성하고 각각 다른 Disk를 정의할 경우, Physical Disk의 재구성 없이, Logical 하게 Table과 Index의 I/O를 분리할 수 있다.

DATA DISK

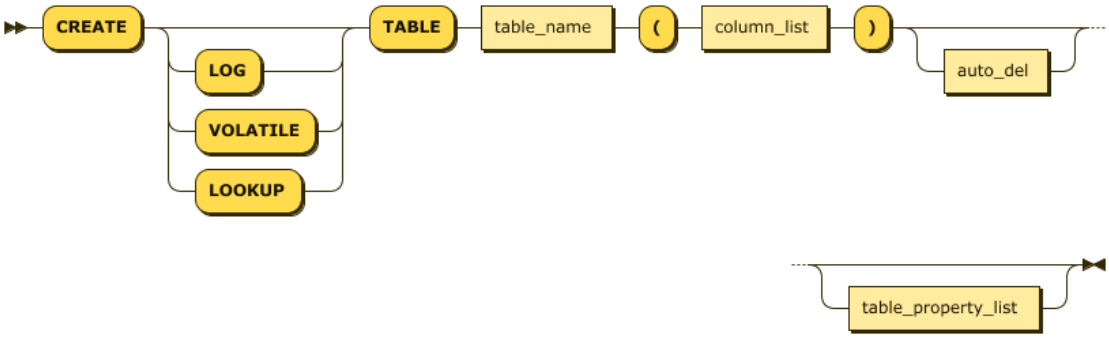
Tablespace에 속한 Disk를 정의한다. 각 Disk는 다음과 같은 속성을 가진다.

속성	설명
disk_name	Disk 객체의 이름을 지정한다. 추후에 Alter Tablespace구문을 통해서 Disk객체의 속성을 변경할 때 사용한다.

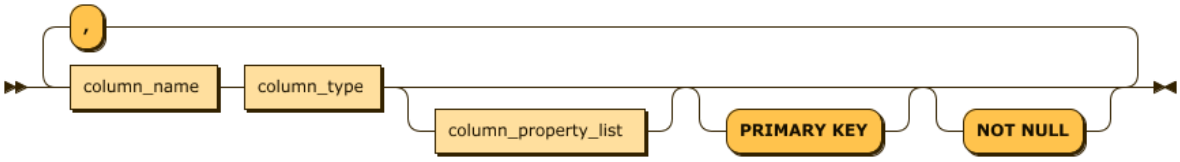
속성	설명
data_disk_property	Disk의 속성을 지정한다.
disk_path	Disk의 Directory Path를 지정한다. 이 Directory는 Create 되어 있어야 한다. 상대 Path로 Path를 지정시 $MACHBASE_HOME/dbs 기준으로 PATH를 찾는다. 예를 들어 PATH='disk1'일 경우 Disk Path를 $MACHBASE_HOME/dbs/disk1으로 인식한다.
parallel_io	Disk의 IO Request를 Parallel하게 몇 개까지 허용할 지를 결정한다. (DEF: 3, MIN: 1, MAX: 128)

CREATE TABLE

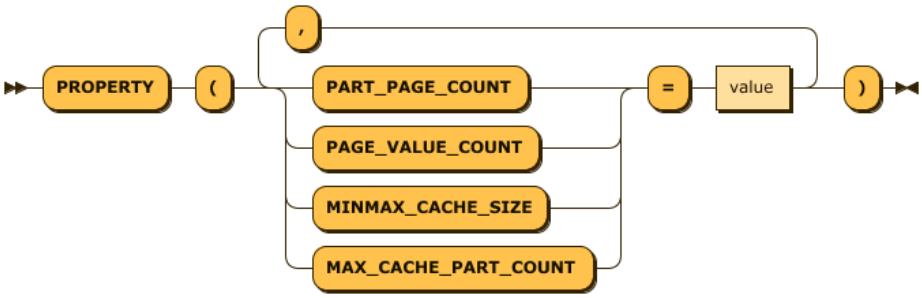
create_table_stmt:



column_list:

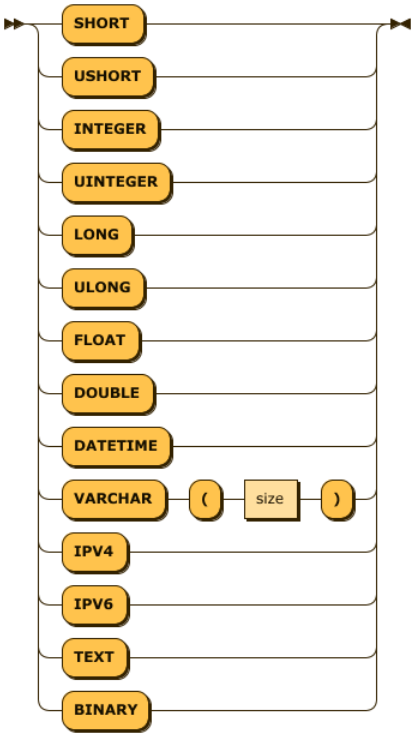


column_property_list:

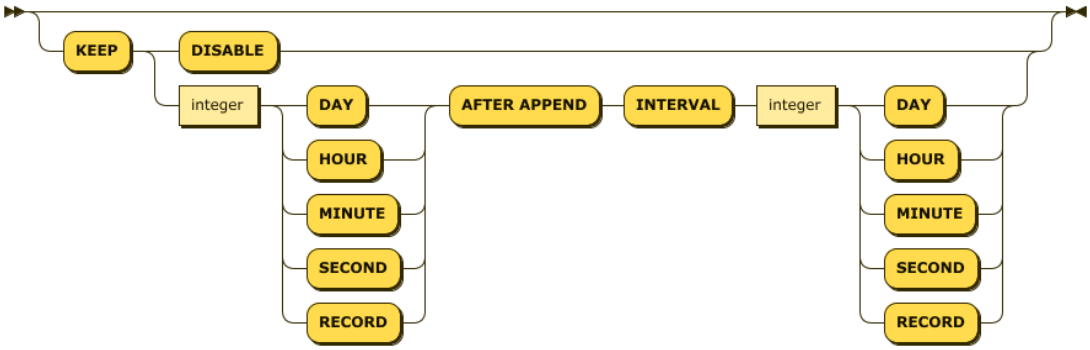


table_property_list:

column_type:



auto_del:




```
create_table_stmt ::= 'CREATE' ( 'LOG' | 'VOLATILE' | 'LOOKUP' )? 'TABLE' table_name '(' column_list ')' autodel? t
column_list ::= column_name column_type column_property_list? 'PRIMARY KEY'? 'NOT NULL'? ( ',' column_name column_t
column_property_list ::= 'PROPERTY' '(' ( 'PART_PAGE_COUNT' | 'PAGE_VALUE_COUNT' | 'MINMAX_CACHE_SIZE' | 'MAX_CACHE
table_property_list ::=  ('TAG_PARTITION_COUNT' ) '=' value
column_type ::= 'SHORT' | 'USHORT' | 'INTEGER' | 'UINTEGER' | 'LONG' | 'ULONG' | 'FLOAT' | 'DOUBLE' | 'DATETIME' |
```

```
-- 5개의 column을 가진 ctest table을 만든다.
Mach> CREATE TABLE ctest (id INTEGER, name VARCHAR(20), sipv4 IPV4, dipv6 IPV6, comment TEXT);
Created successfully.
```

테이블 종류

테이블 종류	설명
LOG_TABLE	CREATE TABLE 사이에 아무런 키워드를 넣지 않았다면 Log Table이 생성된다.
VOLATILE_TABLE	VOLATILE_TABLE은 모든 데이터가 임시 메모리에 상주하는 임시 테이블이며 로그 테이블을 조인하여 결과를 향상시킵니다만, Marbase 서버가 종료 되자마자 사라집니다.
LOOKUP_TABLE	VOLATILE_TABLE과 마찬가지로 LOOKUP_TABLE은 메모리의 모든 데이터를 저장함으로써 빠른 쿼리 처리를 수행 할 수 있습니다.

테이블 컬럼

테이블 컬럼은 문자열을 그대로 넣는 것이 보통이지만, 특수 문자를 넣기 위해 홑따옴표 (') 또는 쌍따옴표 (") 로 감싸서 넣을 수도 있다.

```
Mach> CREATE TABLE special_tbl ( with.dot INTEGER );
[ERR-02010: Syntax error: near token (.dot INTEGER )).]
CREATE TABLE special_tbl ( "with.dot" INTEGER ); -- 가능
CREATE TABLE special_tbl ( 'with.dot' INTEGER ); -- 가능
```

해당 컬럼을 SELECT 쿼리를 통해 조회할 때도 홑따옴표 또는 쌍따옴표로 감싸야 하는데, 두 방법엔 차이가 존재한다.

- 홑따옴표로 감싸게 되면, 컬럼이 아닌 String Literal 로 취급된다.
- 쌍따옴표로 감싸게 되면, SELECT 쿼리 결과 컬럼의 이름 그대로 출력된다.

```
Mach> SELECT 'with.dot' FROM special_tbl;
'with.dot'
--------------
[0] row(s) selected.

Mach> SELECT "with.dot" FROM special_tbl;
with.dot
--------------
[0] row(s) selected.
```

테이블 프로퍼티(Table Property)

Table에 대한 속성을 지정한다.

프로퍼티 이름	사용 가능한 테이블 종류
TAG_PARTITION_COUNT	TAG TABLE
TAG_DATA_PART_SIZE	TAG TABLE
TAG_STAT_ENABLE	TAG TABLE

TAG_PARTITION_COUNT(Default:4)

TAG Table에 대해 지원되는 속성으로 TAG 테이블을 내부적으로 몇 개의 파티션 테이블에 저장할 것인지 결정한다. TAG의 수나 서버의 성능에 따라 설정하여야 한다.

TAG_DATA_PART_SIZE(Default:16MB)

TAG Table에 대해 지원되는 속성으로 파티션 테이블 별 데이터 사이즈를 결정한다.

TAG_STAT_ENABLE(Default:1)

TAG Table에 대해 지원되는 속성으로 TAG ID 별 통계 정보 저장 여부를 결정한다.

컬럼 프로퍼티(Column Property)

Column에 대한 속성을 지정한다.

프로퍼티 이름	사용 가능한 테이블 종류
PART_PAGE_COUNT	LOG TABLE
PAGE_VALUE_COUNT	LOG TABLE
MAX_CACHE_PART_COUNT	LOG TABLE
MINMAX_CACHE_SIZE	LOG TABLE

PART_PAGE_COUNT

이 프로퍼티는 하나의 파티션이 가지는 Page의 개수를 나타낸다. 하나의 파티션이 가지는 Value의 개수는 PART_PAGE_COUNT * PAGE_VALUE_COUNT가 된다.

PAGE_VALUE_COUNT

이 프로퍼티는 하나의 Page가 가지는 Value의 개수를 나타낸다.

MAX_CACHE_PART_COUNT (Default : 0)

이 프로퍼티는 성능 향상을 위한 캐시 영역을 설정하는 것이다.

마크베이스 가 파티션에 접근할 때 해당 파티션의 메타 정보를 메모리에 담고 있는 구조체를 먼저 찾게 되는데, 몇 개의 파티션 정보를 메모리에 담고 있을지 결정한다. 크면 클수록 성능에 도움이 될 것이나, 메모리 사용량이 늘어난다. 최소값은 1 최대값은 65535이다.

MINMAX_CACHE_SIZE (Default : 10240)

이 프로퍼티는 해당 Column의 MINMAX를 위한 캐시 메모리를 얼마나 사용할 것인지 지정하는 것이다. 0번째 Hidden Column인 _ARRIVAL_TIME의 경우 기본적으로 100MB으로 지정이 된다. 하지만 다른 Column들은 기본적으로 10KB로 지정되어 있다. 이 크기는 Table의 생성 이후에도 "ALTER TABLE MODIFY" 구문을 통해서 이 값은 변경이 가능하다.

NOT NULL Constraint

컬럼 값에 NULL을 허용하지 않을 경우 NOT NULL을 지정하고, 허용할 경우(Default)에는 생략한다.

테이블 생성 이후에 정의된 이 제약조건을 삭제하거나 추가하기 위해서는 ALTER TABLE MODIFY COLUMN 명령으로 제약조건을 변경할 수 있다.

```
# 컬럼 c1은 not null로, c2는 not null 제약 조건 없이 생성한다.
CREATE TABLE t1(c1 INTEGER NOT NULL, c2 VARCHAR(200));
```

Pre-defined System Columns

Create Table 문을 이용하여 테이블을 생성하면 시스템은 두 개의 사전 정의된 시스템 컬럼을 추가로 생성한다. _ARRIVAL_TIME 및 _RID컬럼이다.

_ARRIVAL_TIME 컬럼은 DATETIME 타입의 컬럼으로 INSERT 문이나 AppendData로 데이터를 삽입하는 시점의 시스템 time을 기준으로 삽입되며, 해당 값은 생성된 레코드의 unique key로 사용될 수 있다. 이 컬럼의 값은 순서가 보장되는 경우(과거-현재 순으로) machloader나 INSERT 문에서 값을 지정하여 삽입할 수 있다. DURATION 조건절을 이용하여 데이터를 검색할 경우, 이 컬럼의 값을 기준으로 데이터를 검색한다.

_RID 컬럼은 특정 레코드가 갖는 유일한 값으로 시스템이 생성한다. 이 컬럼의 데이터 타입은 64bit 정수이며, 이 컬럼에 대해서는 사용자가 값을 지정할 수 없고 인덱스도 생성할 수 없다. 데이터 INSERT시에 자동으로 생성된다. _RID 컬럼의 값으로 레코드를 검색할 수 있다.

```
create volatile table t1111 (i1 integer);
Created successfully.
Mach> desc t1111;

----------------------------------------------------------------
NAME                          TYPE               LENGTH
----------------------------------------------------------------
_ARRIVAL_TIME                 datetime           8
I1                            integer            4

Mach>insert into t1111 values (1);
1 row(s) inserted.
Mach>select _rid from t1111;
_rid
------------------------
0
[1] row(s) inserted.

Mach>select i1 from t1111 where _rid = 0;
i1
--------------
1
[1] row(s) selected.
```

MINMAX Cache 개념

일반적으로 Disk DBMS에서는 특정 값을 인덱스를 활용하여 검색할 경우 해당 인덱스가 포함된 디스크 영역에 대해 접근하고, 해당 값이 포함된 최종 디스크 페이지를 찾아가도록 구현되어 있다.

반면, 마크베이스는 시계열 정보를 유지하기 위해 시간순으로 파티션 되어있는 구조이며, 이것은 특정한 하나의 인덱스 정보가 시간순으로 조각조각의 파일로 나누어져 있다는 의미이다. 따라서, 마크베이스의 인덱스를 이용할 때는 이러한 파티션으로 조각나 있는 인덱스 파일을 순차적으로 검색한다.

만일 검색해야 할 대상 데이터의 범위가 1000개의 파티션으로 나뉘어져 있다면 1000번의 파일을 매번 열어서 검색해야 한다는 의미이다. 비록 효율적인 컬럼형 데이터베이스 구조로 설계되어 있긴 하나, 이러한 I/O 비용이 인덱스 파티션 개수의 크기에 비례하기 때문에 그 성능을 향상하기 위한 방법이 MINMAX_CACHE 구조이다.

이 MINMAX_CACHE는 해당 파티션의 인덱스 파일 정보를 메모리에 담고 있는 구조체로서 해당 컬럼의 최소 및 최대 값을 메모리에 유지하는 연속된 메모리 공간이다. 이런 구조를 유지함으로써 특정 값이 포함된 파티션을 검색할 경우 그 값이 해당 인덱스의 최소값 보다 작거나 최대 값보다 클 경우에는 아예 해당 파티션을 건너뛸 수 있기 때문에 고성능의 데이터 분석이 가능해진다.

위의 그림에서 볼 수 있듯이 85라는 값을 찾기 위해서 5개의 파티션 중에서 MIN/MAX에 포함된 1번과 5번 파티션만을 실제로 검색하게 되며, 2, 3, 4 번 파티션은 아예 건너뛰는 모습을 볼 수 있다.

MINMAX Cache 컬럼

테이블 생성 시에 특정 컬럼에 대해 MINMAX Cache를 사용할 것인지 결정할 수 있다.

만일 이 컬럼이 minmax_cache_size가 0이 아닌 값으로 설정되었으면, 해당 컬럼에 인덱스 검색 시 MINMAX Cache가 동작하게 되며, `MINMAX_CACHE_SIZE = 0`일 경우에는 동작하지 않는다.

이런 MINMAX Cache를 사용할 때 다음과 같은 사항에 주의한다.

1. MINMAX Cache 는 해당 컬럼에 인덱스를 명시적으로 생성하지 않아도 적용된다.
2. 모든 컬럼의 default는 `MINMAX_CACHE_SIZE`가 10KB로 설정되있고 Alter Table 구문을 활용하여 적정한 크기의 메모리 크기를 재설정할 수 있다.
3. 숨어있는 컬럼인 _arrival_time은 디폴트로 100MB이며, 자동으로 MINMAX Cache 메모리를 사용한다.
4. VARCHAR 타입의 경우에는 MINMAX Cache 의 대상이 되지 않는다. 따라서 VARCHAR 타입을 명시적으로 캐시 사용 여부를 지정하면, 에러가 발생한다.
5. 해당 테이블이 하나 생성될 때 프로퍼티에 설정된 `MINMAX_CACHE_SIZE` 만큼 최대 메모리가 더 사용될 수 있다. 파티션 개수가 늘수록 메모리가 점진적으로 늘어나 위의 최대 메모리만큼 늘어난다.
6. 만일 해당 테이블에 레코드가 하나도 들어있지 않으면, MINMAX Cache 메모리는 전혀 할당되지 않는다.

아래는 실제 MINMAX를 활용한 테이블 생성 예를 나타낸다.

```
-- VARCHAR에 대한 MINMAX_CACHE_SIZE = 0은 의미상으로 허용된다.
CREATE TABLE ctest (id INTEGER, name VARCHAR(100) PROPERTY(MINMAX_CACHE_SIZE = 0));
Created successfully.
Mach>

-- id 컬럼에 캐시가 적용되었다.
CREATE TABLE ctest2 (id INTEGER PROPERTY(MINMAX_CACHE_SIZE = 10240), name VARCHAR(100) PROPERTY(MINMAX_CACHE_SIZE =
Created successfully.
Mach>

-- id1, id2, id3 컬럼에 적용되었다.
CREATE TABLE ctest3 (id1 INTEGER PROPERTY(MINMAX_CACHE_SIZE = 10240), name VARCHAR(100) PROPERTY(MINMAX_CACHE_SIZE
Created successfully.
Mach>

-- Column단위로 MINMAX_CACHE_SIZE가 지정되거나, 0으로 설정되었다.
CREATE TABLE ctest4 (id1 INTEGER PROPERTY(MINMAX_CACHE_SIZE=10240), name VARCHAR(100) PROPERTY(MINMAX_CACHE_SIZE=0)
Created successfully.
Mach>
```

기본 키(Primary Key)

Volatile/Lookup 테이블의 컬럼에 부여할 수 있는 제약 사항으로, 해당 컬럼의 값이 중복되는 것을 방지한다. Volatile/Lookup 테이블이 항상 기본 키를 가지고 있을 필요는 없으나, 기본 키가 없으면 INSERT ON DUPLICATE KEY UPDATE 구문을 사용할 수 없다.

기본 키를 부여하면, 기본 키에 대응되는 레드-블랙 트리 인덱스가 생성된다.

AUTO_DEL 절

데이터의 양을 제한하여 디스크 저장 공간을 유지할 수 있다. Log 테이블에 대해서만 지원하며, CREATE TABLE문에서 Table property를 지정하기 전에 AUTO_DEL절을 이용하여 설정한다. AUTO_DEL절은 저장 시간 기준 혹은 레코드 수 기준으로 설정할 수 있다.

```
CREATE TABLE t1 (c1 INT) KEEP 30 DAY AFTER APPEND INTERVAL 5 SECOND;
```

위 예제는 자동 삭제가 수행된 이후 5초가 지나고 추가 입력이 있는 경우, 입력된지 30일이 지난 데이터를 삭제한다. interval로 지정된 기간이 길면 자동 delete가 수행되는 시간이 길어져서 입력에 영향을 끼칠 가능성이 있으며, 너무 짧으면 시스템 성능에 전체적으로 영향을 줄 수 있다.

아래의 예제는 보존할 데이터의 건 수로 자동 삭제 기능을 지정하는 예이다.

```
CREATE TABLE t1 (c1 INT) KEEP 3 RECORD AFTER APPEND INTERVAL 5 RECORD;
```

입력 데이터 5건 마다 해당 테이블의 레코드의 수를 검사하여 3건 이상이면 3건만 남기도록 자동 삭제를 수행한다.

SEQUENCE for Lookup Table

Lookup 테이블의 Unique한 Record를 생성하고 Data의 입력 순서를 결정하기 위해 위해 Sequence가 추가되었다.

이 기능은 Lookup 테이블에서 datetime column을 사용하여 Record의 순서를 구분하는 방식을 사용했을 때

datetime 값이 중복될 경우 Record의 순서를 구분하기 어렵고 데이터 중복으로 인한 Application의 오류가 발생하는 등의 문제점을 해결하기 위해 추가되었다.

Sequence 지원 Machbase Edition

Edge / Fog / Cluster

Sequence 지원 Table 종류

Lookup Table

Lookup 테이블 생성 시 Sequence 설정

Create Table SQL 문으로 Lookup 테이블을 생성할 때 Sequence로 사용할 컬럼에 PROPERTY 절을 추가하여 Sequence를 설정하겠다고 명시하면 된다.

Sequence로 설정할 컬럼은 LONG datatype(64bit, unsigned)만 지원하며 이외의 datatype은 지원하지 않는다.

추가로, Sequence의 시작값을 설정할 수 있는데 1로 설정한 경우 Sequence가 1부터 시작이 된다. (0이나 음수는 지원 안함)

```
CREATE LOOKUP TABLE table_name (v1 LONG PROPERTY(SEQUENCE=1), v2 VARCHAR(10));
```

Sequence 컬럼의 사용

Lookup 테이블의 Sequence 컬럼은 기본적으로 일반 Long 컬럼과 동일하게 사용이 가능하며 이렇게 사용할 경우 Sequence 값은 자동으로 증가하지 않는다.

Sequence 컬럼에 직접 값을 입력하는 것이 허용되며 심지어 중복 값을 입력하는 것도 가능하다.

대신, Sequence 기능을 사용하려면 nextval 이라는 새로 추가된 Sequence 전용 Function을 사용하여 Sequence값을 증가시키는 방식으로 사용해야 한다.

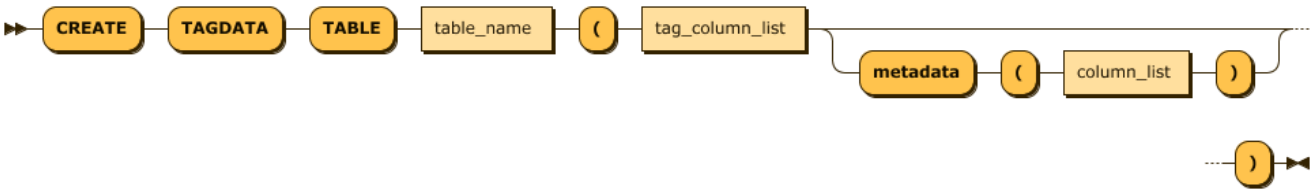
내부적으로는 Sequence로 설정된 컬럼의 값 중 가장 큰 값에 대해 저장하고 있기 때문에 이후에 nextval Function을 사용하여 입력할 때 Sequence 컬럼 값 중 가장 큰 값 + 1 의 값이 저장된다.

Sequence 컬럼 사용 예

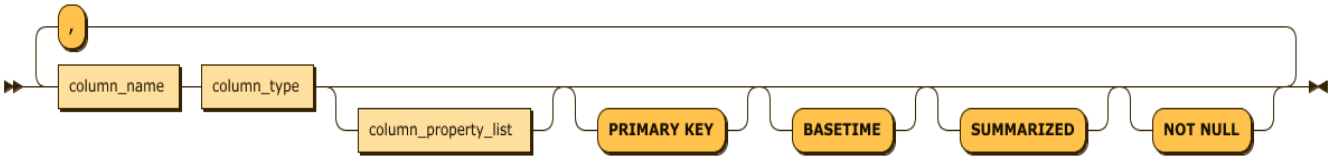
```
-- Sequence 컬럼에 nextval Function을 사용하여 다음 Sequence 값 입력
INSERT INTO table_name (v1, v2) values (nextval(v1), 'aaaa');

-- Sequence 컬럼에 직접 값을 입력
INSERT INTO table_name (v1, v2) values (100, 'aaaa');

-- Sequence 컬럼에 연산을 통한 값을 입력
INSERT INTO table_name (v1, v2) values (1 + 99, 'aaaa');

-- Sequence 컬럼을 포함한 Lookup 테이블에 대한 정상 Select
SELECT v1, v2 FROM table_name;

-- Sequence 컬럼에 대한 잘못된 Select (nextval 컬럼은 insert 시에만 사용 가능)
SELECT nextval(v1), v2 FROM table_name;
```

CREATE TAG Table

create_tag_table_stmt:



tag_column_list:



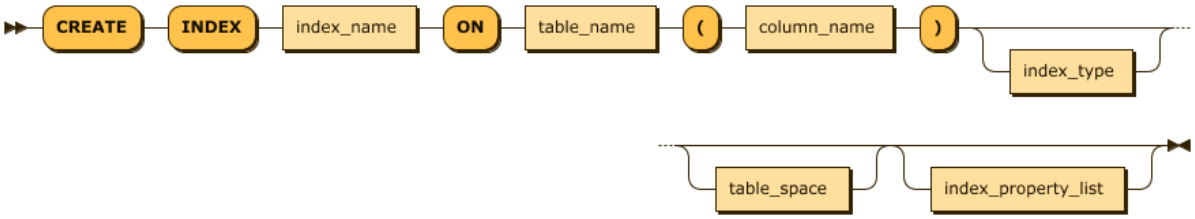

```
create_tag_table_stmt ::= 'CREATE' 'TAGDATA' 'TABLE' table_name '(' tag_column_list ( 'metadata' '(' column_list '
tag_column_list ::= column_name column_type column_property_list? 'PRIMARY KEY'? 'BASETIME'? 'SUMMARIZED'? 'NOT NUL
```

태그 테이블 생성시 반드시 primary key, basetime, summarized 가 들어가야 한다.

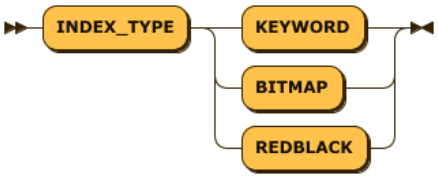
```
--예제
CREATE TAGDATA TABLE tag (name varchar(20) primary key, time datetime basetime, value double summarized);
CREATE TAGDATA TABLE tag (name varchar(20) primary key, int_column int, time datetime basetime, value double summar
CREATE TAGDATA TABLE tag (name varchar(20) primary key, time datetime basetime, value double summarized, value2 flo
```

CREATE INDEX

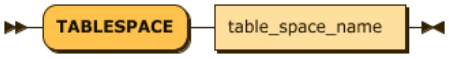
create_index_stmt:



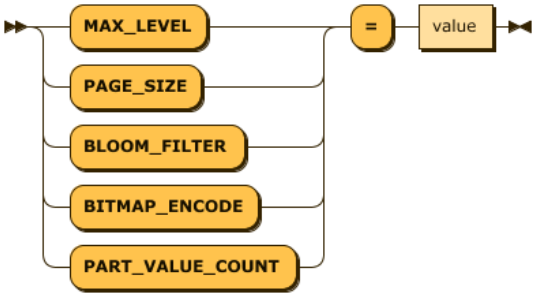
index_type:



table_space:



index_property_list:




```
create_index_stmt ::= 'CREATE' 'INDEX' index_name 'ON' table_name '(' column_name ')' index_type? table_space? inde
index_type ::= 'INDEX_TYPE' ( 'KEYWORD' | 'BITMAP' | 'REDBLACK' )
table_space ::= 'TABLESPACE' table_space_name
```

```
index_property_list ::= ( 'MAX_LEVEL' | 'PAGE_SIZE' | 'BITMAP_ENCODE' | 'PART_VALUE_COUNT' ) '=' value
```

Index Type

생성할 Index Type을 지정한다. Keyword Index가 아닌 경우 Index Type을 지정하지 않으면 Table Type에 따라서 Default Index Type으로 Index가 생성된다.

Table Type	Default Index Type
Volatile Table	REDBLACK
Lookup Table	REDBLACK
Log Table	LSM

KEYWORD Index

텍스트 검색을 위한 인덱스로써 로그 테이블의 varchar와 text 컬럼에만 생성 가능하며, 단일 컬럼에 대해서만 생성할 수 있다.

LSM Index

LSM(Log Structure Merge) Index로 Big Data에 저장 및 검색에 최적화된 Index이다. LSM Index들의 Partition들은 Level 별로 유지되고 하위 Level의 Partition들이 Merge되어 상위 Level로 이동한다. 그리고 상위 Level의 Partition 생성에 사용된 하위 Partition들은 삭제된다.

이러한 Index Level Partition Building은 Background Thread 에 의해서 수행된다. 상위 Level Partition은 하위 Level의 Partition 들이 Merge되어 하나의 Partition으로 생성되기 때문에 Index를 통한 검색 시 다음과 같은 장점이 존재한다.

1. Key가 중복된 경우, 한 번만 저장되기 때문에 Key 저장을 위한 Disk Space가 절약된다.
2. 여러 개의 Partition에 대한 Searching보다 하나의 Index Partition에 대한 검색 시 File Open 및 Close 비용이 줄어들고, 접근하는 Index Page의 개수 또한 줄어든다.

LSM Index Property

항목	설명
MAX_LEVEL (DEFAULT = 3, MIN = 0, MAX = 3)	LSM Index의 최대 Level로서 현재 3이 최대값이다. 그리고 하나의 Partition의 최대 Record 건수는 2억건을 초과할 수 없다. 각 Level의 Partition 크기는 이전 Partition의 Value 개수 * 10이다. 예를 들면 MAX_LEVEL = 3, PART_VALUE_COUNT가 100,000 이면 Level 0 = 100,000, Level 1 = 1,000,0000, Level 2 = 10,000,000, Level 3 = 100,000,000 이다. 만약 마지막 Level의 Partition Size가 2억건을 초과하면 Index 생성이 실패한다.
PAGE_SIZE (DEFAULT = 512 * 1024, MIN = 32 * 1024, MAX = 1 * 1024 * 1024)	Index의 Key Value와 Bitmap 값이 저장되는 Page의 크기를 지정한다. Default는 512K이다.
BITMAP_ENCODE (DEFAULT = EQUAL, RANGE)	인덱스의 Bitmap 타입을 설정한다. BITMAP_ENCODE=EQUAL(기본값)의 경우 키값과 같은 값에 대한 bitmap을 생성하고 BITMAP=RANGE인 경우 키값의 range에 따른 bitmap을 생성한다. 질의 조건으로 = 을 주로 사용하는 경우 BITMAP_ENCODE=EQUAL로, 특정 범위값을 질의 조건으로 주로 사용하는 경우 BITMAP_ENCODE=RANGE로 설정하는 편이 좋다. BITMAP=RANGE인 경우 생성 비용은 EQUAL에 비해서 약간 증가한다.

BITMAP 인덱스

데이터 분석을 위한 인덱스로서, 로그 테이블에만 생성 가능하다. 그리고 varchar, text, binary를 제외한 모든 컬럼에 생성 가능하며, 단일 컬럼에 대해서만 생성할 수 있다.

RED-BLACK 인덱스

실시간 데이터 검색을 위한 메모리 인덱스로서, Volatile/Lookup 테이블에만 생성 가능하다. 그리고 이 테이블의 모든 컬럼에 생성 가능하며 단일 컬럼에 대해서만 생성할 수 있다.

Index Property

LSM Index 에서 적용할 수 있는 Property 는 다음과 같다.

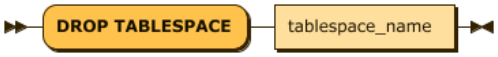
PART_VALUE_COUNT

Index의 Partition에 저장되는 Row 개수를 나타낸다.

```
--예제
-- c1 컬럼에 index가 적용되었다.
CREATE INDEX index1 on table1 ( c1 )
-- varchar type의 var_column에 keyword index가 적용되고 page_size의 단위는 100000가 되었다.
CREATE INDEX index2 on table1 (var_column) INDEX_TYPE KEYWORD PAGE_SIZE=100000;
```

DROP TABLESPACE

drop_table_stmt:



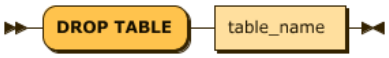

```
drop_table_stmt ::= 'DROP TABLESPACE' tablespace_name
```

지정된 Tablespace를 삭제한다. 하지만 Tablespace에 생성된 객체가 존재하는 경우, 삭제가 실패한다.

```
--예제
DROP TABLESPACE TablespaceName;
```

DROP TABLE

drop_table_stmt:



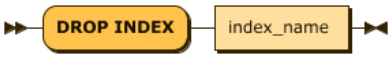

```
drop_table_stmt ::= 'DROP TABLE' table_name
```

지정된 테이블을 삭제한다. 단, 해당 테이블을 검색 중인 다른 세션이 존재할 경우에는 에러를 내면서 실패한다.

```
--예제
DROP TABLE TableName;
```

DROP INDEX

drop_index_stmt:




```
drop_index_stmt ::= 'DROP INDEX' index_name
```

지정된 인덱스를 삭제한다. 단, 해당 테이블을 검색 중인 다른 세션이 존재할 경우에는 에러를 내면서 실패한다.

```
--예제
DROP INDEX IndexName;
```

ALTER TABLESPACE

ALTER TABLESPACE 구문은 지정된 Tablespace에 관련된 정보를 변경하는데 사용된다.

ALTER TABLESPACE MODIFY DATADISK

이 구문은 Tablespace의 DATADISK의 속성을 변경하는데 사용된다.

alter_tablespace_stmt:


```
alter_tablespace_stmt ::= 'ALTER TABLESPACE' table_name 'MODIFY DATADISK' disk_name 'SET' 'PARALLEL_IO' '=' value
```

```
-- 예제
ALTER TABLESPACE tbs1 MODIFY DATADISK disk1 SET PARALLEL_IO = 10;
```

ALTER TABLE

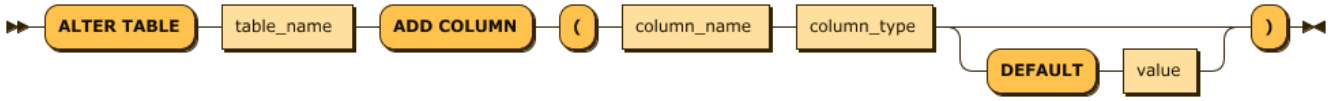
ALTER TABLE 구문은 지정된 테이블의 스키마 정보를 변경시키기 위한 용도로 사용되며 Log Table 만 사용 가능하다.

ALTER TABLE SET

이 구문은 Table의 Property를 변경하는 구문이다. 현재 동적으로 변경 가능한 Property는 없다.

ALTER TABLE ADD COLUMN

alter_table_add_stmt:




```
alter_table_add_stmt ::= 'ALTER TABLE' table_name 'ADD COLUMN' '(' column_name column_type ( 'DEFAULT' value )? ')
```

이 구문은 테이블에 특정 컬럼을 실시간으로 추가하는 기능이다. 컬럼의 이름과 타입을 추가하고, DEFAULT 구문을 통해 기본 데이터 값을 설정할 수 있다.

```
-- 예제-1
alter table atest2 add column (id4 float);

-- 예제-2
alter table atest2 add column (id6 double  default 5);
alter table atest2 add column (id7 ipv4  default '192.168.0.1');
alter table atest2 add column (id8 varchar(4) default 'hello');
```

ALTER TABLE DROP COLUMN

alter_table_drop_stmt:


```
alter_table_drop_stmt ::= 'ALTER TABLE' table_name 'DROP COLUMN' '(' column_name ')'
```

이 구문은 테이블에 특정 컬럼을 실시간으로 삭제하는 기능이다.

```
-- 예제
alter table atest2 drop column (id4);
alter table atest2 drop column (id8);
```

ALTER TABLE RENAME COLUMN

alter_table_column_rename_stmt:

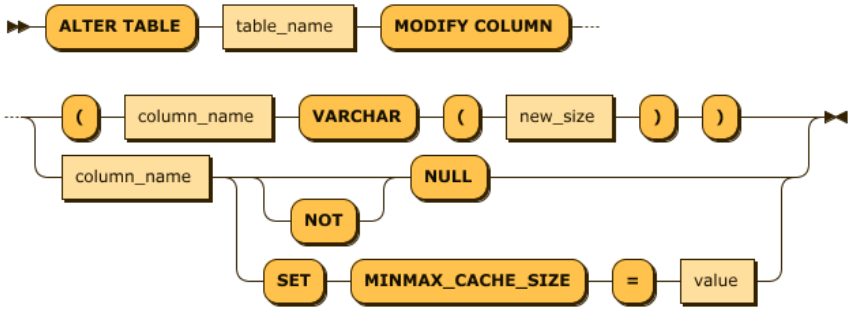

```
alter_table_column_rename_stmt ::= 'ALTER TABLE' table_name 'RENAME COLUMN' old_column_name 'TO' new_column_name
```

이 구문은 테이블의 특정 컬럼명을 변경하는 기능이다.

```
-- 예제
alter table atest2 rename column id7 to id7_rename;
```

ALTER TABLE MODIFY COLUMN

alter_table_modify_stmt:




```
alter_table_modify_stmt ::= 'ALTER TABLE' table_name 'MODIFY COLUMN' ( '(' column_name 'VARCHAR' '(' new_size ')'
```

이 구문은 테이블의 특정 컬럼의 속성을 변경하는 것이다. 현재는 VARCHAR 타입의 컬럼 길이와 그외 타입에 대한 MINMAX CACHE 속성과 NOT NULL 제약조건을 수정하는 것이 가능하다.

VARCHAR SIZE

이 구문은 VARCHAR 타입의 컬럼 길이만 변경하는 것을 지원한다. 이 동작은 기존의 데이터를 보존하기 위해 그 길이가 줄어들 수는 없으며, 언제나 증가해야 한다.

```
ALTER TABLE table_name MODIFY COLUMN (column_name VARCHAR(new_size));
```

```
-- 예제 : TABLE 이 이렇게 만들어졌다고 가정하자.
-- create table atest5 (id integer, name varchar(5), id3 double, id4 float);

-- 에러 발생: 다른 타입으로 변경할 수 없음.
alter table atest5 modify column (id varchar(10));

-- 에러 발생: VARCHAR 길이를 더 작게 할 수 없음.
alter table atest5 modify column (name varchar(3));

-- 에러 발생: VARCHAR의 최대 크기 32767 이상 넘을 수 없음.
alter table atest5 modify column (name varchar(32768));

-- 성공
alter table atest5 modify column (name varchar(128));
```

MINMAX_CACHE_SIZE

이 구문은 특정 컬럼에 대해 MINMAX_CACHE_SIZE를 변경한다.

```
ALTER TABLE table_name MODIFY COLUMN column_name SET MINMAX_CACHE_SIZE=value;
```

```
-- 예제 : TABLE 이 이렇게 만들어졌다고 가정하자.
create table atest9 (id integer, name varchar(100));
```

```
-- 에러: VARCHAR에는 적용 안됨.
alter table atest9 modify column name set minmax_cache_size=0;
[ERR-02139 : MINMAX CACHE is not allowed for VARCHAR column(NAME).]

-- 변경 성공
alter table atest9 modify column id set minmax_cache_size=10240;
```

NOT NULL

컬럼에 NOT NULL 제약 조건을 추가한다. NOT NULL 제약 조건을 추가할 경우 NULL값이 있는 컬럼에 대해서는 DDL연산이 실패한다.

만약 컬럼에 NULL값을 허용하고 싶은 경우에는 다음 절의 MODIFY COLUMN NULL 명령어를 이용한다.

```
ALTER TABLE table_name MODIFY COLUMN column_name NOT NULL;
```

```
-- t1.c1에 NOT NULL 제약조건을 추가한다.
alter table t1 modify column c1 not null;
```

NULL

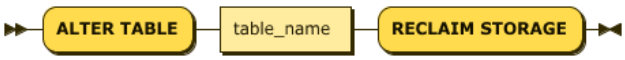
NOT NULL 제약조건을 해제한다. LSM 인덱스의 min_max 캐시로 인한 성능 향상을 얻을 수 없다. NULL 값의 입력이 가능해 진다.

```
ALTER TABLE table_name MODIFY COLUMN column_name NULL;
```

```
-- t1.c1에 NOT NULL 제약조건을 해제한다.
alter table t1 modify column c1 null;
```

ALTER TABLE RECLAIM STORAGE

alter_table_reclaim_storage_stmt:



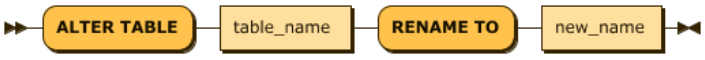

```
alter_table_reclaim_storage_stmt ::= 'ALTER TABLE' table_name 'RECLAIM STORAGE'
```

Tag 테이블에서 사용되지 않는 데이터를 삭제하여 가용공간을 확보한다. 시스템 프로퍼티 DISK_TAG_AUTO_RECLAIM 가 1 인 경우(기본값) 특별히 수행하지 않아도 자동으로 수행된다. 이 값을 0으로 설정한 경우에는 원하는 시점에 이 질의를 수행하여 저장 공간을 확보할 수 있다. Tag 테이블에 대해서만 사용가능하다.

```
-- tag table의 저장공간을 확보한다.
alter table tag reclaim storage;
```

ALTER TABLE RENAME TO

alter_table_rename_stmt:




```
alter_table_rename_stmt ::= 'ALTER TABLE' table_name 'RENAME TO' new_name
```

테이블의 이름을 변경한다.

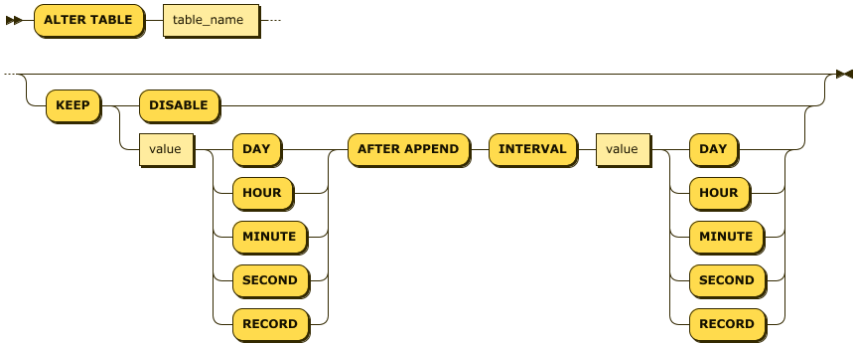
메타 테이블들은 이름을 변경할 수 없고, 변경될 이름에 $문자는 사용할 수 없다. 테이블 이름 변경은 log 테이블에 대해서만 가능하다.

```
-- user 테이블의 이름을 employee로 변경한다.
ALTER TABLE user RENAME TO employee
```

ALTER TABLE AUTO DELETE

alter_table_auto_del_stmt:

```
alter_table_auto_del_stmt ::=  'ALTER TABLE' table_name ('KEEP' 'DISABLE' || 'KEEP' value ('DAY' | 'HOUR' | 'MINUTE
```



테이블의 자동 삭제 주기를 설정하거나 사용 여부를 결정할 수 있다.

log 테이블에서만 가능하다.

```
-- logtbl 테이블의 자동 삭제를 30일간의 데이터만 유지하고 이를 append 이후 5초 간격으로 검사하도록 변경한다.
ALTER TABLE logtbl keep 30 day after append interval 5 second;
```

TRUNCATE TABLE

truncate_table_stmt:

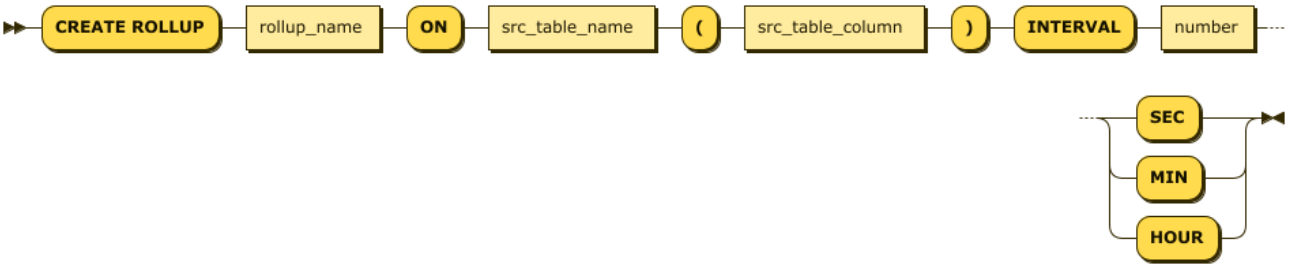

```
truncate_table_stmt ::= 'TRUNCATE TABLE' table_name
```

```
-- ctest 테이블의 모든 데이터를 삭제한다.
Mach> truncate table ctest;
Truncated successfully.
```

지정된 테이블에 존재하는 모든 데이터를 삭제한다. 단, 해당 테이블을 검색 중인 다른 세션이 존재할 경우에는 에러를 내면서 실패한다.

CREATE ROLLUP

create_rollup_stmt:

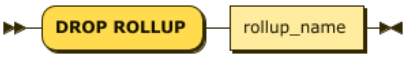



```
create_rollup_stmt ::= 'CREATE ROLLUP' rollup_name 'ON' src_table_name '('src_table_column')' 'INTERVAL' number ('S
```

```
-- tag table의 value 칼럼을 대상으로 rollup을 생성한다.
Mach> CREATE ROLLUP _rollup_tag_value_sec ON tag(value) INTERVAL 1 SEC;
Executed successfully
```

DROP ROLLUP

drop_rollup_stmt



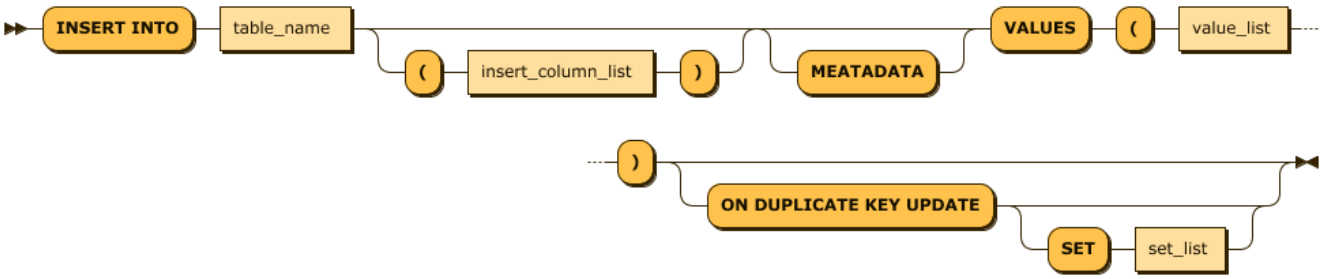

```
drop_rollup_stmt ::= 'DROP ROLLUP' rollup_name
```

```
-- rollup을 삭제한다.
Mach> DROP ROLLUP _rollup_tag_value_sec;
Executed successfully
```

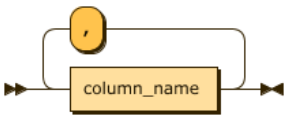
DML

INSERT

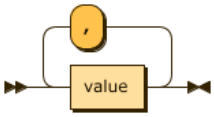
insert_stmt:



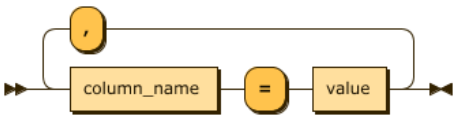
insert_column_list:



value_list:



set_list:




```
insert_stmt ::= 'INSERT INTO' table_name ( '(' insert_column_list ')' )? 'METADATA'? 'VALUES' '(' value_list ')' (
insert_column_list ::= column_name ( ',' column_name )*
value_list ::= value ( ',' value )*
set_list ::= column_name '=' value ( ',' column_name '=' value )*
```

```
create table test (number int,name varchar(20));
Created successfully.
insert into test values (1,"test");
1 row(s) inserted.
insert into test(name,number) values ("test",2);
1 row(s) inserted.
```

특정 테이블에 값을 입력하는 구문이다. 한 가지 특이한 점은 Column_List에서 지정되지 않은 컬럼에는 모두 NULL 값으로 채워진다는 것이다. 이는 입력의 편의성과 저의 효율화를 위해 채택된 로그 파일의 특성을 고려한 정책이다.

METADATA는 tag table에만 사용이 가능하다.

INSERT ON DUPLICATE KEY UPDATE

마크베이스는, 흔히 알려진 UPSERT 기능과 유사한 구문을 지원한다.

기본 키가 지정된 Lookup/Volatile 테이블에 값을 입력할 때 사용할 수 있는 특수 구문으로, 기본 키 값이 중복되는 데이터가 이미 테이블에 존재하는 경우에는 기존 데이터의 값이 변경된다.
물론, 키 값이 중복되는 데이터가 존재하지 않는 경우에는 새로운 데이터로 삽입된다.

이 구문을 사용하기 위해서, 휘발성 테이블에 기본 키가 지정되어 있어야 한다.

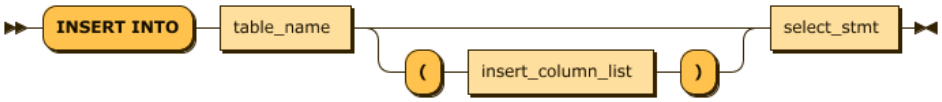
삽입되는 데이터의 컬럼 값과 갱신되는 데이터의 컬럼 값을 다르게 하고자 하는 경우, 또는 삽입되는 데이터의 컬럼 값이 아닌 다른 컬럼 값을 갱신하고자 하는 경우에는 SET 절을 추가로 입력할 수 있다.

- SET 절에는 '컬럼=값'으로 구성되며, 각각을 콤마로 구분해야 한다.

- SET 절에서 기본 키 값을 변경해서는 안 된다.

INSERT SELECT

insert_select_stmt:




```
insert_select_stmt ::= 'INSERT INTO' table_name ( '(' insert_column_list ')' )? select_stmt
```

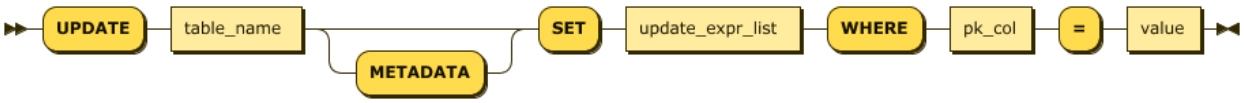
특정 table에 대해서 SELECT 문의 수행 결과를 삽입하는 문장이다. 기본적으로는 다른 DBMS와 유사하지만 다음의 차이점이 있다.

1. _ARRIVAL_TIME 컬럼 값은 select 및 INSERT 컬럼 리스트에서 지정되지 않으면 INSERT SELECT 문이 수행되는 시점의 시간 값으로 입력된다.
2. VARCHAR 타입의 컬럼에 대해서 삽입되는 입력값이 컬럼의 최대 길이보다 큰 경우, 오류를 발생시키지 않고 해당 컬럼의 최대 길이만큼 잘라서 입력된다.
3. 형 변환이 가능한 경우(숫자형->숫자형)에는 입력되는 컬럼 값에 맞게 삽입된다.
4. 수행 도중 오류가 발생한 경우 ROLLBACK되지 않는다.
5. _ARRIVAL_TIME 컬럼의 값을 지정하여 삽입하는 경우, 새로 입력되는 값이 기존의 값보다 이전 시간을 갖고 있으면 입력되지 않는다.

```
create table t1 (i1 integer, i2 varchar(60), i3 varchar(5));
Created successfully.

insert into t1 values (1, 'a', 'ddd' );
1 row(s) inserted.
insert into t1 values (2, 'kkkkkkkkkkkkkkkkkkkk', 'c');
1 row(s) inserted.

insert into t1 select * from t1;
2 row(s) inserted.
create table t2 (i1 integer, i2 varchar(60), i3 varchar(5));

insert into t2 (_arrival_time, i1, i2, i3) select _arrival_time, * from t1;
4 row(s) inserted.
```

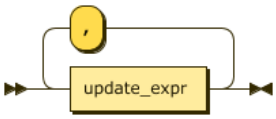
UPDATE

5.5 부터 제공되는 기능입니다.

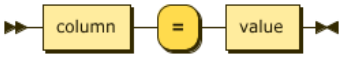


update_stmt:

update_expr_list:



update_expr:




```
update_stmt ::= 'UPDATE' table_name ( 'METADATA' )? 'SET' update_expr_list 'WHERE' primary_key_column '=' value
update_expr_list ::= update_expr ( ',' update_expr)*
update_expr ::= column '=' value
```

INSERT ON DUPLICATE KEY UPDATE 를 통한 UPSERT 가 아닌, UPDATE 구문도 제공된다.

역시, 기본 키 (Primary Key) 가 지정된 Lookup/Volatile 테이블에 값을 입력할 때 사용할 수 있다. WHERE 절에는 기본 키의 일치 조건식을 작성해야 한다.

UPDATE METADATA

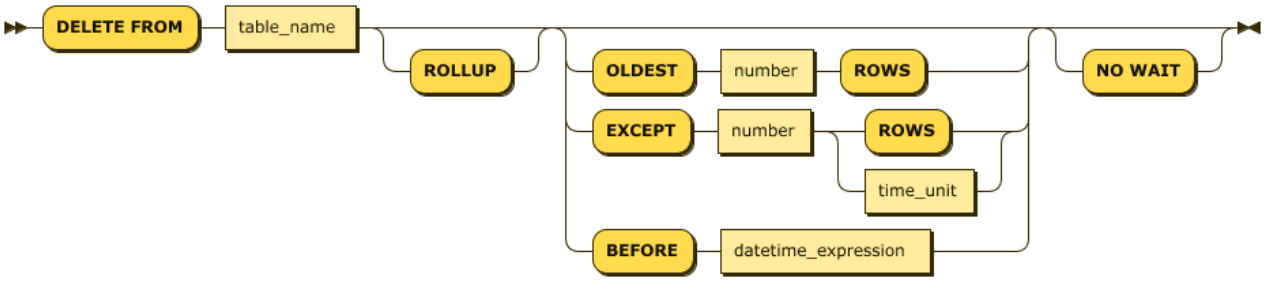
TAGDATA 테이블에 한해서, 메타데이터를 업데이트 하고자 할 때 사용한다.

```
UPDATE TAG METADATA SET ...
```

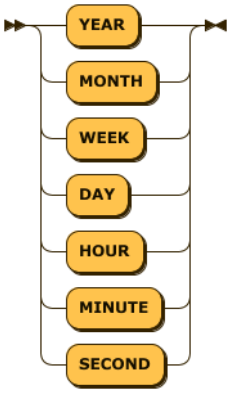
⚠ TAGDATA 테이블의 메타데이터는 INSERT ON DUPLICATE KEY UPDATE 를 통해 입력/수정할 수 없다.

DELETE

delete_stmt:



time_unit:




```
delete_stmt ::= 'DELETE FROM' table_name ( 'OLDEST' number 'ROWS' | 'EXCEPT' number ( 'ROWS' | time_unit ) | 'BEFOR
time_unit ::= 'DURATION' number time_unit ( ( 'BEFORE' | 'AFTER' ) number time_unit )?
```

마크베이스에서의 DELETE BEFORE 구문은 로그 테이블, Tag 테이블, Rollup table에 대해서 수행 가능하다. 중간의 임의 위치에 있는 데이터를 삭제할 수 없으며, 임의의 위치부터 연속적으로 마지막(가장 오래된 로그) 레코드까지 지울 수 있도록 구현되었다.

이는 로그 데이터의 특성을 살린 정책으로서 한번 입력되면 수정이 없고, 공간 확보를 위해 파일을 삭제하는 행위를 DB 형식으로 표현한 것이다.

DURATION, OLDEST, EXCEPT 구문은 TAG 및 Rollup 테이블에 대해서는 사용할 수 없다.

```
-- 모두 삭제하라.
DELETE FROM devices;

-- 가장 오래된 마지막 N건을 삭제하라.
DELETE FROM devices OLDEST N ROWS;

-- 최근 N건을 제외하고 모두 삭제하라.
DELETE FROM devices EXCEPT N ROWS;

-- 지금부터 N일치를 남기고 모두 삭제하라.
DELETE FROM devices EXCEPT N DAY;

-- 2014년 6월 1일 이전의 데이터를 모두 삭제하라.
DELETE FROM devices BEFORE TO_DATE('2014-06-01', 'YYYY-MM-DD');
```

```
-- tag 데이터의 시간 기준 삭제
DELETE FROM tag BEFORE TO_DATE('2014-06-01', 'YYYY-MM-DD');

-- tag rollup 데이터의 시간 기준 삭제
DELETE FROM tag ROLLUP BEFORE TO_DATE('2014-06-01', 'YYYY-MM-DD');
```

DELETE WHERE

delete_where_stmt:

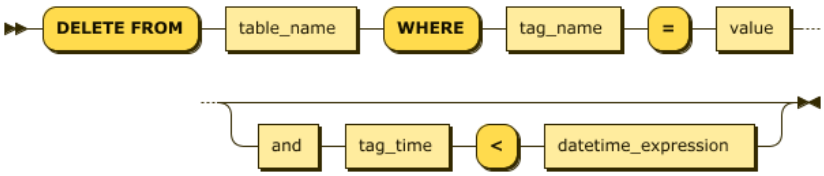

```
delete_where_stmt ::= 'DELETE FROM' table_name 'WHERE' column_name '=' value
```

```
create volatile table t1 (i1 int primary key, i2 int);
Created successfully.
insert into t1 values (2,2);
1 row(s) inserted.
delete from t1 where i1 = 2;
1 row(s) deleted.
```

휘발성 테이블에 대해서만 수행 가능한 구문으로, WHERE 절에 작성된 조건에 일치하는 레코드만 삭제할 수 있다.

- 기본 키가 지정된 휘발성 테이블에 대해서만 수행 가능하다.
- WHERE 절에는 (기본 키 컬럼) = (값) 의 조건만 허용되며, 다른 조건과 함께 작성할 수 없다.
- 기본 키 컬럼이 아닌 다른 컬럼을 조건에 사용할 수 없다.

delete_from_tag_where_stmt:




```
delete_from_tag_where_stmt ::= 'DELETE FROM' table_name 'WHERE' tag_name '=' value ( and tag_time '<' datetime_expr
```

Tag 테이블은 아래와 같이 2가지 방식의 삭제쿼리가, 추가적으로 지원된다.

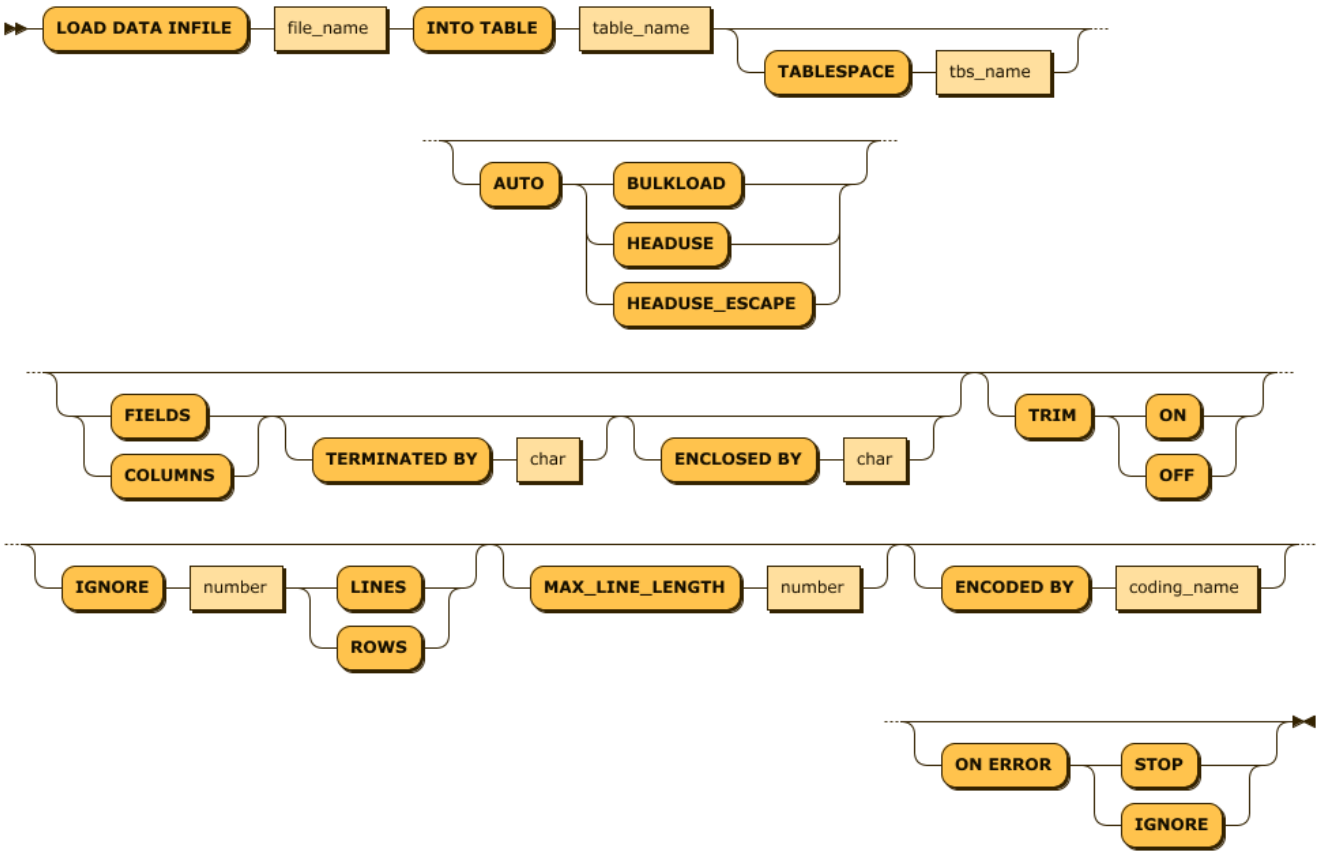
- Tag name 기준 삭제
- Tag name과 Tag time 기준 삭제

```
-- tag name 기준 삭제
DELETE FROM tag where tag_name = 'my_tag_2021'

-- tag name 와 tag time 기준 삭제
DELETE FROM tag where tag_name = 'my_tag_2021' and tag_time < TO_DATE('2021-07-01', 'YYYY-MM-DD');
```

- 삭제 쿼리가 실행된 후에, 삭제된 row가 저장공간에서 물리적으로 삭제되기 까지 걸리는 시간은, DBMS의 동작상황에 따라서 다를 수 있다.

LOAD DATA INFILE

load_data_infile_stmt:




```
load_data_infile_stmt: 'LOAD DATA INFILE' file_name 'INTO TABLE' table_name ( 'TABLESPACE' tbs_name )? ( 'AUTO' (
```

CSV 포맷의 데이터 파일을 서버에서 직접 읽어서, 옵션에 따라 서버에서 직접 테이블 및 컬럼들을 생성하여 이를 입력하는 기능이다.

각 옵션에 대해서 설명하면 다음과 같다.

옵션	설명
AUTO mode_string mode_string = (BULKLOAD | HEADUSE | HEADUSE_ESCAPE)	해당 테이블을 생성하고 컬럼 타입(자동 생성시 varchar type) 및 컬럼명을 자동으로 생성한다. BULKLOAD: 데이터 한 개의 row를 하나의 컬럼으로 입력한다. 컬럼으로 구분할 수 없는 데이터에 대해서 사용한다. HEADUSE: 데이터 파일의 첫 번째 라인에 기술되어 있는 컬럼 명을 테이블의 컬럼명으로 사용하고, 그 라인에 기술된 수 만큼의 컬럼을 생성한다. HEADUSE_ESCAPE: HEADUSE 옵션과 유사하지만, 컬럼명이 DB의 예약어와 같을 경우 발생할 수 있는 오류를 회피하기 위해 컬럼명의 앞뒤로 '_' 문자를 덧붙이고, 컬럼명에 특수문자가 존재하면 그 문자를 '_' 문자로 변경한다.
(FIELDS|COLUMNS) TERMINATED BY 'term_char' ESCAPED BY 'escape_char'	데이터 라인을 파싱하기 위한 구분 문자(term_char)와 이스케이프 문자(escape_char)를 지정한다. 일반적인 CSV 파일의 경우 구분 문자는 , 이며 이스케이프 문자는 '이다.
ENCODED BY coding_name coding_name = { UTF8(default) | MS949 | KSC5601 | EUCJP | SHIFTJIS | BIG5 | GB231280 }	데이터 파일의 인코딩 옵션을 지정한다. 기본 값은 UTF-8이다.
TRIM (ON | OFF)	컬럼의 빈 공간을 제거하거나 유지한다. 기본값은 ON이다.
IGNORE number (LINES | ROWS)	숫자로 지정된 라인 또는 행 만큼의 데이터를 무시한다. CSV 포맷 파일의 헤더 등을 무시하거나 VCF 헤더를 무시하기 위해서 사용한다.
MAX_LINE_LENGTH	한 라인의 최대 길이를 지정한다. 기본값은 512K이며, 데이터가 더 큰 경우에는 더 큰 값을 지정할 수 있다.
ON ERROR (STOP | IGNORE)	입력 도중 에러가 발생할 경우 수행할 동작을 지정한다. STOP인 경우 입력을 중단하고 IGNORE인 경우 에러가 발생한 라인을 건너뛰고 계속 입력한다. 기본값은 IGNORE이다.

```
-- default field delimiter(,)  field encloser (") 를 사용하여 데이터를 입력한다.
LOAD DATA INFILE '/tmp/aaa.csv' INTO TABLE Sample_data ;
```

```
-- 하나의 컬럼을 갖는 NEWTABLE을 생성해서 한 라인을 한 컬럼으로 입력한다.
LOAD DATA INFILE '/tmp/bbb.csv' INTO TABLE NEWTABLE AUTO BULKLOAD;

-- csv의 첫번째 라인을 컬럼 정보로 이용하여 NEWTABLE을 생성하고, 이를 그 테이블에 입력한다.
LOAD DATA INFILE '/tmp/bbb.csv' INTO TABLE NEWTABLE AUTO HEADUSE;

-- 첫번째 라인은 무시하고 필드 구분자는 ; enclosing 문자는 ' 로 지정해서 입력한다.
LOAD DATA INFILE '/tmp/ccc.csv' INTO TABLE Sample_data FIELDS TERMINATED BY ';' ENCLOSED BY '\''  IGNORE 1 LINES OI
```

AUTO 옵션을 사용하지 않는 경우 테이블의 모든 컬럼은 VARCHAR 또는 TEXT 타입으로 생성해야 한다.

SELECT

SELECT는 마크베이스에서 각종 테이블로부터 데이터를 찾거나 필터링 및 조작하는 데 사용되는 구문이다.

목차

SELECT Syntax

```
select_stmt UNION ALL select_stmt
```

```
SELECT target_list FROM TableList WHERE Condition GROUP BY Expr ORDER BY Expr [
```

SET OPERATOR

여러 개의 Select문의 결과를 하나의 질의 결과로 전달받을 경우에 사용한다. 마크베이스는 UNION ALL 집합 연산자만을 지원한다. 집합 연산자는 좌우에 기술된 Select문이 (1) 같거나 호환가능한 타입이며 (2) 질의 결과값의 개수가 동일한 경우에만 실행이 가능하며 두 조건 중 하나라도 일치하지 않은 경우에는 오류로 처리된다.

다음의 기준으로 데이터 타입 변환이나 호환성 검증을 수행한다.

- 부호 있는 정수형 타입과 부호 없는 정수형 타입은 호환이 되지 않는다.
- 정수형 타입은 실수형 타입과 호환이 되며 질의 결과는 실수형 타입으로 변환되어 반환된다.
- 문자형 타입은 길이가 달라도 호환이 된다.
- IPv6 타입과 IPv4 타입은 호환이 되지 않는다.
- 두 개의 SELECT 문 중 항상 왼쪽 질의의 컬럼명이 사용된다.

실행 예제

```
SELECT i1, i2 FROM table_1
UNION ALL
SELECT c1, c2 FROM table_2
```

TARGET LIST

Select 문이 대상으로 하는 **컬럼** 또는 **Subquery 의 리스트**이다.

Target list에 사용된 Subquery는 WHERE 조건절에서 사용되는 Subquery와 같이 두 개 이상의 값을 갖거나 두 개 이상의 결과 컬럼을 갖는 경우에는 오류로 처리된다.

```
SELECT i1, i2 ...
SELECT i1 (Select avg(c1) FROM t1), i2 ...
```

CASE 구문

```
CASE <simple_case_expression|searched_case_expression> [else_clause] END

simple_case_expression ::=
    expr WHEN comparison_expr THEN return_expr
        [WHEN comparison_expr THEN return_expr ...]

searched_case_expression ::=
    WHEN condtion_expr THEN return EXPR [WHEN condtion_expr THEN return EXPR ...]

else_clause ::=
    ELSE else_value_expr
```

일반적인 프로그램 언어의 IF... THEN... ELSE블록을 지원하는 표현식이다. simple_case_expression은 하나의 컬럼이나 표현식이 when 뒤에 오는 comparison_expr 값과 같은 경우 return_expr을 반환하는 형태로 수행되며 이 when ... then 절은 원하는 만큼 반복하여 기술할 수 있다.

searched_case_expression은 CASE 이후에 표현식을 지정하지 않고 when절에 비교연산자를 포함한 조건절을 기술한다. 각 비교연산의 결과가 참이면 then 절의 값을 반환한다. else 절은 when 절들의 값이 만족하지 않을 경우(expression 결과가 NULL인 경우에도) else_value를 반환한다.

```
select * from t1;
I1          I2
--------------------------
2           2
1           1
[2] row(s) selected.

select case i1 when 1 then 100 end from t1;
case i1 when 1 then 100 end
------------------------------
NULL
100
[2] row(s) selected.
```

simple_case_expression의 예제에서 i1 컬럼의 값이 2인 경우에 해당하는 값이 없으면 NULL을 반환한다.

```
select case when i1 > 0 then 100 when i1 > 1 then 200 end from t1;
case when i1 > 0 then 100 when i1 > 1 then 200 end
----------------------------------------
100
100
[2] row(s) selected.
```

searched_case_expression에서 만족하는 첫 번째 조건절을 반환하므로 첫 번째 조건절의 반환값인 100이 반환되며 두 번째 조건절은 실행이 되지 않는다.

FROM

FROM 절에는 테이블 이름이나 Inline view를 지정할 수 있다. 테이블 간의 Join을 수행하려면 테이블 혹은 Inline view를 쉼표(,)로 구분해서 나열한다.

```
FROM table_name
```

table_name로 지정한 테이블 내의 데이터를 검색한다.

SUBQUERY(INLINE VIEW) 사용

```
FROM (Select statement)
```

괄호로 둘려쳐진 subquery의 내용에 대하여 데이터를 검색한다.

> 마크베이스 서버는 correlated subquery를 지원하지 않으므로 outer query에서 subquery 내의 column을 참조할 수 없다.

JOIN(INNER JOIN)

```
FROM TABLE_1, TABLE_2
```

두 개의 테이블 table_1 과 table_2를 JOIN한다. INNER JOIN은 테이블이 3개 이상 나열될 때에도 사용이 가능하며 WHERE 절에 검색 조건절과 JOIN 조건절을 모두 기술하여 사용한다.

```
SELECT t1.i1, t2.i1 FROM t1, t2 WHERE t1.i1 = t2.i1 AND t1.i1 > 1 AND t2.i2 = 3;
```

INNER JOIN 및 OUTER JOIN

ANSI 스타일의 INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN을 지원한다. FULL OUTER JOIN은 지원하지 않는다.

```
FROM TABLE_1 [INNER|LEFT OUTER|RIGHT OUTER] JOIN TABLE_2 ON expression
```

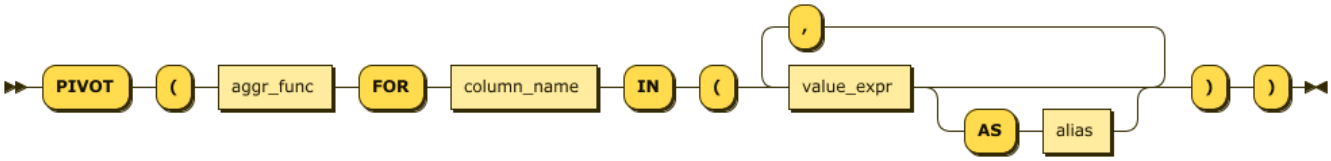
ANSI 스타일 JOIN절의 ON절에는 JOIN에서 수행하는 조건절을 사용한다. OUTER JOIN 질의에서 where절에 Inner table(ON 절의 조건을 만족하지 않으면 NULL이 채워지는 테이블)에 대한 조건절이 있는 경우, 해당 질의는 INNER JOIN으로 변환된다.

```
SELECT t1.i1 t2.i1 FROM t1 LEFT OUTER JOIN t2 ON (t1.i1 = t2.i1) WHERE t2.i2 = 1;
```

위 질의는 WHERE 절의 조건 t2.i2 = 1에 의하여 INNER JOIN으로 변환된다.

PIVOT

> PIVOT 구문은 마크베이스 5.5.6 버전부터 지원한다.



pivot_clause:

PIVOT 구문은 ROW로 출력되는 GROUP BY에 대한 집계 결과를 컬럼으로 재배열하여 보여준다.

Inline view와 함께 사용되며 다음과 같이 수행된다.

- Inline view의 결과 컬럼 중 PIVOT 절에 사용되지 않은 컬럼으로 GROUP BY를 수행한 후 PIVOT IN 절에 나열된 값 별로 집계함수를 수행한다.
- 결과로 나온 grouping 컬럼과 집계 결과를 회전하여 컬럼으로 보여준다.

예) 여러 센서로부터 수집된 데이터에서 각 device 별로 value 값을 집계해서 출력하라.

CASE 구문을 통해 수행해야하는 질의를 PIVOT 구문을 통해 간단히 표현할 수 있다.

```
-- w/o PIVOT
SELECT * FROM (
    SELECT
             regtime,
             SUM(CASE WHEN tagid = 'FRONT_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS front_axis_torque,
             SUM(CASE WHEN tagid = 'REAR_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS rear_axis_torque,
             SUM(CASE WHEN tagid = 'HOIST_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS hoist_axis_torque,
             SUM(CASE WHEN tagid = 'SLIDE_AXIS_TORQUE' THEN dvalue ELSE 0 END)  AS slide_axis_torque
    FROM     result_d
    WHERE    regtime BETWEEN TO_DATE('2018-12-07 00:00:00') AND TO_DATE('2018-12-08 05:00:00')
    GROUP BY regtime
) WHERE front_axis_torque >= 40 AND rear_axis_torque >= 20;

-- w/ PIVOT
SELECT * FROM (
    SELECT regtime, tagid, dvalue FROM result_d
    WHERE  regtime BETWEEN TO_DATE('2018-12-07 00:00:00') AND TO_DATE('2018-12-08 05:00:00')
) PIVOT (SUM(dvalue) FOR tagid IN ('FRONT_AXIS_TORQUE', 'REAR_AXIS_TORQUE', 'HOIST_AXIS_TORQUE', 'SLIDE_AXIS_TORQUE
WHERE front_axis_torque >= 40 AND rear_axis_torque >= 20;

-- Result
regtime                         'FRONT_AXIS_TORQUE'         'REAR_AXIS_TORQUE'          'HOIST_AXIS_TORQUE'
-------------------------------------------------------------------------------------------------------------------
2018-12-07 16:42:29 840:000:000 12158                       7244                        NULL
2018-12-07 14:56:26 220:000:000 3308                        663                         NULL
2018-12-07 12:20:13 844:000:000 3804                        113                         NULL
2018-12-07 11:10:01 957:000:000 8729                        5384                        NULL
2018-12-07 17:46:57 812:000:000 7500                        4559                        NULL
2018-12-07 14:30:06 138:000:000 5080                        6817                        NULL
2018-12-07 13:09:20 464:000:000 5233                        1869                        -7253
2018-12-07 15:43:03 539:000:000 7491                        4453                        NULL
...
```

WHERE

SUBQUERY의 사용

조건절에 대해서 subquery의 사용이 가능하다. IN 구문을 제외한 조건절에서 subquery가 두 개 이상의 레코드를 리턴하거나, subquery의 결과 컬럼이 두 개 이상인 경우는 지원하지 않는다.

```
WHERE i1 = (SELECT MAX(c2) FROM T1)
```

subquery를 조건연산자 오른쪽에 괄호를 둘러쳐서 사용한다.

> 마크베이스 서버는 correlated subquery를 지원하지 않으므로 outer query에서 subquery 내의 column을 참조할 수 없다.

SEARCH 구문

일반 데이터베이스와의 문법이 동일하다. 단, 반드시 keyword index를 등록해야 하며, 텍스트 검색을 위한 연산자 키워드인 "SEARCH"를 추가하여, 부가적인 검색 연산이 가능하다.

```
-- drop table realdual;
create table realdual (id1 integer, id2 varchar(20), id3 varchar(20));

create keyword index idx1 on realdual (id2);
create keyword index idx2 on realdual (id3);

insert into realdual values(1, 'time time2', 'series series2');

select * from realdual;

select * from realdual where id2 search 'time';
select * from realdual where id3 search 'series' ;
select * from realdual where id2 search 'time' and id3 search 'series';
```

수행 결과는 다음과 같다.

```
Mach> create table realdual (id1 integer, id2 varchar(20), id3 varchar(20));
Created successfully.

Mach> create keyword index idx1 on realdual (id2);
Created successfully.

Mach> create keyword index idx2 on realdual (id3);
Created successfully.

Mach> insert into realdual values(1, 'time time2', 'series series2');
1 row(s) inserted.

Mach> select * from realdual;
ID1         ID2                  ID3
-------------------------------------------------------------
1           time time2           series series2
[1] row(s) selected.

Mach> select * from realdual where id2 search 'time';
ID1         ID2                  ID3
-------------------------------------------------------------
1           time time2           series series2
[1] row(s) selected.

Mach> select * from realdual where id3 search 'series';
ID1         ID2                  ID3
-------------------------------------------------------------
1           time time2           series series2
[1] row(s) selected.

Mach> select * from realdual where id2 search 'time' and id3 search 'series';
ID1         ID2                  ID3
-------------------------------------------------------------
1           time time2           series series2
[1] row(s) selected.
```

ESEARCH 구문

ESEARCH 구문은 ASCII 문자 텍스트에 대한 확장 검색을 가능하게 해주는 검색 키워드이다. 이러한 확장을 위해 % 문자를 이용하여 원하는 패턴의 검색을 수행한다. 이 Like 연산에서 앞에 %가 오는 경우 모든 레코드를 검사해야 하지만, ESEARCH의 장점은 이 경우에도 빠르게 해당 단어를 찾을 수 있다는 데 있다. 이 기능은 영문 문자열(에러 문자열 혹은 코드)의 일부를 찾을 때 매우 유용하게 사용할 수 있다.

```
예제

select id2 from realdual where id2 esearch 'bbb%';
id2
-----------------------------------------
bbb ccc1
aaa bbb1

[2] row(s) selected.

검색 pattern 'bbb%'에 의하여 bbb1도 검색 결과에 포함된다.


select id3 from realdual where id3 esearch '%cd%';
id3
-----------------------------------------
cdf def1
bcd/cdf1ad
abc, bcd1
[3] row(s) selected.

% 문자는 검색 pattern의 처음, 끝 뿐만 아니라 가운데에 있어도 동작한다.

select id3 from realdual where id3 esearch '%cd%';
id3
-----------------------------------------
cdf def1
bcd/cdf1ad
abc, bcd1
[3] row(s) selected.
```

NOT SEARCH 구문

NOT SEARCH는 SEARCH구문에서 검색되는 조건 이외의 레코드들에 대해서 참을 리턴하는 구문이다.

NOT ESEARCH는 사용할 수 없다.

```
create table t1 (id integer, i2 varchar(10));
create keyword index t1_i2 on t1(i2);
insert into t1 values (1, 'aaaa');
insert into t1 values (2, 'bbbb');

select id from t1 where i2 not search 'aaaa';

id
-----------------------------------------
2
[1] row(s) selected.
```

REGEXP 구문

REGEXP 구문은 정규표현식을 사용하여 데이터에 대한 검색을 수행하는데 사용된다. 일반적으로 특정 컬럼의 패턴을 정규표현식을 사용하여 데이터를 필터링하게 된다.

한가지 주의할 점은 REGEXP 구문을 사용할 경우 인덱스를 활용할 수 없기 때문에 전체 검색 범위를 줄이기 위해 반드시 다른 컬럼에 대한 인덱스 조건을 넣어서 전체적인 검색 비용을 낮춰야 한다.

특정 패턴을 검사하고자 할 때에는 SEARCH 혹은 ESEARCH를 통해 인덱스를 활용하도록 하고, 이를 통해 전체적인 데이터 건수가 작아진 상태에서 다시 REGEXP를 이용하는 것이 시스템 전체 효율 향상에 도움이 된다

```
Mach>
create table realdual (id1 integer, id2 varchar(20), id3 varchar(20));
create table dual (id integer);
```

```
insert into dual values(1);
insert into realdual values(1, 'time1', 'series1 series21');
insert into realdual values(1, 'time2', 'series2 series22');
insert into realdual values(1, 'time3', 'series3 series32');


Mach> select * from realdual where id2 REGEXP 'time' ;
ID1         ID2                  ID3
------------------------------------------------------------
1           time3                series3 series32
1           time2                series2 series22
1           time1                series1 series21
[3] row(s) selected.

Mach> select * from realdual where id2 REGEXP 'time[12]' ;
ID1         ID2                  ID3
------------------------------------------------------------
1           time2                series2 series22
1           time1                series1 series21
[2] row(s) selected.

Mach> select * from realdual where id2 REGEXP 'time[13]' ;
ID1         ID2                  ID3
------------------------------------------------------------
1           time3                series3 series32
1           time1                series1 series21
[2] row(s) selected.

Mach> select * from realdual where id2 regexp 'time[13]' and id3 regexp 'series[12]';
ID1         ID2                  ID3
------------------------------------------------------------
1           time1                series1 series21
[1] row(s) selected.

Mach> select * from realdual where id2 NOT REGEXP 'time[12]';
ID1         ID2                  ID3
------------------------------------------------------------
1           time3                series3 series32
[1] row(s) selected.

Mach> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e' from dual;
'abcde' REGEXP 'a[bcd]{1,10}e'
---------------------------------
1
[1] row(s) selected.
```

IN 구문

```
column_name IN (value1, value2,...)
```

IN 구문은 뒤의 value 리스트에서 만족할 경우 TRUE를 리턴한다. OR로 연결된 구문과 동일하다.

IN 구문과 SUBQUERY의 사용

조건절의 IN 구문의 오른쪽에 subquery를 사용할 수 있다. 단, IN 조건절의 왼쪽에는 컬럼 두 개 이상의 컬럼을 지정하면 오류로 처리하고 오른쪽의 subquery에서 리턴되는 결과 집합이 왼쪽 컬럼값에 존재하는지를 검사한다.

```
WHERE i1 IN (Select c1 from ...)
```

ⓘ 마크베이스 서버는 correlated subquery를 지원하지 않으므로 outer query에서 subquery 내의 column을 참조할 수 없다.

BETWEEN 구문

```
column_name BETWEEN value1 AND value2
```

BETWEEN 구문은 column의 값이 value1과 value2 범위에 있을 경우, TRUE를 리턴한다.

RANGE 구문

```
column_name RANGE duration_spec;

-- duration_spec : integer (YEAR | WEEK | HOUR | MINUTE | SECOND);
```

지정된 컬럼에 대해 시간 조건절을 쉽게 지정하는 Range 연산자를 제공한다. Range 연산자는 (BEFORE 키워드로 지정하는 것처럼) 특정 시점을 지정하는 게 아니라 현재 시점부터의 시간 범위를 연산의 대상 조건으로 지정한다. 이 연산자를 사용하면 손쉽게 원하는 시간 범위 내의 결과 레코드들을 검색할 수 있다.

```
select * from test where id < 2 and c1 range 1 hour;
ID          C1
----------------------------------------------
1           2014-07-25 09:28:53 706:707:001
[1] row(s) selected.
```

GROUP BY / HAVING

GROUP BY 절은 SELECT 문으로 검색한 결과를 특정 컬럼을 기준으로 그룹화하기 위해 사용하며, 그룹별로 정렬을 수행하거나 집계 함수를 사용하여 그룹별 집계를 구할 때 사용한다. 그룹이란 GROUP BY 절에 명시된 컬럼에 대해 동일한 컬럼 값을 가지는 레코드들을 의미한다.

GROUP BY 절 뒤에 HAVING 절을 결합하여 그룹 선택을 위한 조건식을 설정할 수 있다. 즉, GROUP BY 절로 구성되는 모든 그룹 중 HAVING 절에 명시된 조건식을 만족하는 그룹만 조회한다.

```
SELECT ...
GROUP BY { col_name | expr } ,...[ HAVING <search_condition> ]

select id1, avg(id2) from exptab where id2 group by id1 order by id1;
id1 컬럼을 기준으로 id2의 평균값을 구한다.
```

ORDER BY

ORDER BY 절은 질의 결과를 오름차순 또는 내림차순으로 정렬하며, ASC 또는 DESC와 같은 정렬 옵션을 명시하지 않으면 디폴트로 오름차순으로 정렬한다. ORDER BY 절을 지정하지 않으면, 조회되는 레코드의 순서는 질의에 따라 다르다.

```
SELECT ...
ORDER BY {col_name | expr} [ASC | DESC]

select id1, avg(id2) from exptab where id2 group by id1 order by id1;
id1 컬럼을 기준으로 id2의 평균값을 구한다.
```

SERIES BY

SERIES BY 절은 정렬된 결과집합을 SERIES BY 조건절을 만족하는 연속된 결과값들로 추출한다. 만약 ORDER BY 절이 지정되지 않은 경우에는 _ARRIVAL_TIME 컬럼값을 이용하여 정렬된 결과를 생성하므로, _ARRIVAL_TIME 컬럼이 없는 휘발성 테이블이나 참조 테이블에 대한 질의나, GROUP BY 절을 이용하는 경우에는 반드시 ORDER BY 절을 이용해야 한다.

조건절을 만족하는 결과값들은 같은 SERIESNUM() 함수의 반환값을 갖게 된다.

```
예를 들어 다음의 데이터에 대해서
CREATE TABLE T1 (C1 INTEGER, C2 INTEGER);
INSERT INTO T1 VALUES (0, 1);

INSERT INTO T1 VALUES (1, 2);

INSERT INTO T1 VALUES (2, 3);
```

```
INSERT INTO T1 VALUES (3, 2);

INSERT INTO T1 VALUES (4, 1);

INSERT INTO T1 VALUES (5, 2);

INSERT INTO T1 VALUES (6, 3);

INSERT INTO T1 VALUES (7, 1);


아래의 질의는 다음의 결과를 출력한다.
SELECT C1,C2 FROM T1 ORDER BY C1 SERIES BY C2>1;
C1          C2
---------------------------
1           2
2           3
3           2
5           2
6           3

C2 컬럼의 값이 1 보다 큰 C1의 RANGE값을 알고 싶은 경우, SERIESNUM 함수로 각 레코드가 어느 그룹에 포함되는지를 출력하여 RANGE를 결
```

LIMIT

LIMIT 절은 출력되는 레코드의 개수를 제한할 때 사용한다. 결과 집합의 특정 행부터 마지막 행까지 출력하기 위해 정수를 지정할 수 있다

```
LIMIT [offset,] row_count

select id1, avg(id2) from exptab where id2 group by id1 order by id1 LIMIT 10;
```

DURATION

DURATION은 _arrival_time을 기준으로 데이터 검색 범위를 손쉽게 결정하도록 해 주는 키워드이다. BEFORE 구문과 함께 사용되어 특정 시점의 특정 데이터 범위를 설정하게 해 준다. 이 DURATION을 잘 활용하면 검색 성능을 현격하게 올림과 동시에 시스템 부하를 획기적으로 낮출 수 있다. 더 자세한 활용 용도는 다음을 참조한다.

```
DURATION Number TimeSpec [BEFORE/AFTER Number TimeSpec]
TimeSpec : YEAR | MONTH | WEEK |  DAY | HOUR | MINUTE | SECOND
```

```
create table t8(i1 integer);
insert into t8 values(1);
insert into t8 values(2);

select i1 from t8;

# BEFORE 절 없이
select i1 from t8 duration 2 second;
select i1 from t8 duration 1 minute;
select i1 from t8 duration 1 hour;
select i1 from t8 duration 1 day;
select i1 from t8 duration 1 week;
select i1 from t8 duration 1 month;
select i1 from t8 duration 1 year;

# DURATION 구문 전체를 써서
select i1 from t8 duration 1 second before 1 day;
select i1 from t8 duration 1 minute before 1 day;
select i1 from t8 duration 1 hour before 1 day;
select i1 from t8 duration 1 day before 1 day;
select i1 from t8 duration 1 week before 1 day;
select i1 from t8 duration 1 month before 1 day;
```

```
select i1 from t8 duration 1 year before 1 day;
```

수행 결과는 다음과 같다.

```
Mach> create table t8(i1 integer);
Created successfully.

Mach> insert into t8 values(1);
1 row(s) inserted.

Mach> insert into t8 values(2);
1 row(s) inserted.

Mach> select i1 from t8;
i1
--------------
2
1
[2] row(s) selected.

# BEFORE 절 없이
Mach> select i1 from t8 duration 2 second;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 minute;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 hour;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 day;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 week;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 month;
i1
--------------
2
1
[2] row(s) selected.

Mach> select i1 from t8 duration 1 year;
i1
--------------
2
1
[2] row(s) selected.
```

```
# DURATION 구문 전체를 써서
Mach> select i1 from t8 duration 1 second before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 minute before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 hour before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 day before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 week before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 month before 1 day;
i1
--------------
[0] row(s) selected.

Mach> select i1 from t8 duration 1 year before 1 day;
i1
--------------
[0] row(s) selected.
```

SAVE DATA

질의의 결과를 CSV 데이터 파일로 바로 저장한다.

```
SAVE DATA INTO 'file_name.csv' [HEADER ON|OFF] [(FIELDS | COLUMNS) [TERMINATED BY 'char'] [ENCLOSED BY 'char']] [EN
```

옵션의 설명은 다음과 같다.

옵션	설명
HEADER (ON|OFF)	생성할 csv 파일의 첫번째 라인에 컬럼명을 입력할지를 결정한다. 기본값은 OFF이다.
(FIELDS|COLUMNS) TERMINATED BY 'term_char' ENCLOSED BY 'escape_char'	생성할 csv 파일의 컬럼 구분자와 이스케이프 구분자를 지정한다.
ENCODED BY coding_name coding_name = (UTF8, MS949, KSC5601, EUCJP, SHIFTJIS, BIG5, GB231280)	출력 데이터 파일의 인코딩 포맷을 지정한다. 기본값은 UTF8이다.

```
SAVE DATA INTO '/tmp/aaa.csv' AS select * from t1;
-- select 문을 실행하여 그 결과를 '/tmp/aaa.csv' 파일에 csv 포멧으로 기록한다.

SAVE DATA INTO '/tmp/ccc.csv' HEADER ON FIELDS TERMINATED BY ';' ENCLOSED BY '\'' ENCODED BY MS949 AS select * fron
-- select 문을 실행하여 그 결과를 /tmp/ccc.csv파일에 기록한다. 필드 구분자와 이스케이프 구분자를 각각 지정하고 저장되는 데이터의 인
```

SELECT Hint

개요

SELECT 쿼리에 함께 사용할 수 있는 Hint를 설명한다.

- Cluster / Fog Edition에 따라서 지원되는 Hint는, 지원/미지원 여부가 표기 되어 있다
- 별도의 표기가 없는 경우에는, 모든 Edition에서 지원한다

목차

PARALLEL

Parallel query 수행을 위한 parallel factor를 지정한다.

- Cluster : 지원함

- Fog : 미지원

```
SELECT /*+ PARALLEL(table_name, parallel_factor) */ ...
```

주어진

```
Mach> CREATE TABLE log_parallel_test (sensor VARCHAR(32), frequency DOUBLE, val
Mach> CREATE INDEX idx_ts ON log_parallel_test (ts);

Mach> EXPLAIN SELECT /*+ PARALLEL(log_parallel_test, 8) */ sensor, frequency, a
      FROM log_parallel_test
      WHERE ts >= TO_DATE('2007-07-01', 'YYYY-MM-DD') and ts <= TO_DATE('2007-0
      GROUP BY sensor, frequency;

PLAN
-------------------------------------------------------------------------------
 PROJECT
  GROUP AGGREGATE
   PARALLEL INDEX SCAN
    *BITMAP RANGE (table id:3, column id:4, index id:4)
    [KEY RANGE]
     * ts >= TO_DATE('2007-07-01', 'YYYY-MM-DD')
     * ts <= TO_DATE('2007-07-31', 'YYYY-MM-DD')
[7] row(s) selected.
```

NOPARALLEL

병렬로 수행되지 않도록 한다.

- Cluster : 지원함

- Fog : 미지원

```
SELECT /*+ NOPARALLEL(table_name) */ ...
```

```
Mach> CREATE TABLE log_parallel_test (sensor VARCHAR(32), frequency DOUBLE, value DOUBLE, ts DATETIME);
Mach> CREATE INDEX idx_ts ON log_parallel_test (ts);

Mach> EXPLAIN SELECT /*+ NOPARALLEL(log_parallel_test) */ sensor, frequency, avg(value)
      FROM log_parallel_test
      WHERE ts >= TO_DATE('2007-07-01', 'YYYY-MM-DD') and ts <= TO_DATE('2007-07-31', 'YYYY-MM-DD')
      GROUP BY sensor, frequency;

PLAN
-------------------------------------------------------------------------------
 PROJECT
  GROUP AGGREGATE
   INDEX SCAN
    *BITMAP RANGE (table id:5, column id:4, index id:6)
    [KEY RANGE]
     * ts >= TO_DATE('2007-07-01', 'YYYY-MM-DD')
     * ts <= TO_DATE('2007-07-31', 'YYYY-MM-DD')
```

```
[7] row(s) selected.
```

FULL

INDEX SCAN을 사용하지 않고, FULL SCAN으로 쿼리를 실행한다.

```
SELECT /*+ FULL(table_name) */ ...
```

아래 스키마와 같이, idx_I1의 인덱스를 사용할 수 있는 쿼리에서도, FULL Hint를 사용하면, 인덱스를 사용하지 않고, FULL SCAN을 한다.

```
Mach> CREATE TABLE log_full_test (sensor VARCHAR(32), I1 INTEGER);
Mach> CREATE INDEX idx_I1 ON log_full_test (I1);

Mach> EXPLAIN SELECT * FROM log_full_test WHERE I1 = 1;
PLAN
------------------------------------------------------------------------------------
 PROJECT
  INDEX SCAN
   *BITMAP RANGE (table id:14, column id:2, index id:15)
   [KEY RANGE]
    * I1 = 1
[5] row(s) selected.

Mach> EXPLAIN SELECT /*+ FULL(log_full_test) */ * FROM log_full_test WHERE I1 = 1;
PLAN
------------------------------------------------------------------------------------
 PROJECT
  FULL SCAN
[2] row(s) selected.
```

NO_INDEX

쿼리에서 index_name의 INDEX를 사용하지 않는다.

```
SELECT /*+ NO_INDEX(table_name,index_name) */ ...
```

```
Mach> CREATE TABLE log_no_index_test (sensor VARCHAR(32), I1 INTEGER, I2 INTEGER);
Mach> CREATE INDEX idx_I1 ON log_no_index_test (I1);
Mach> CREATE INDEX idx_I2 ON log_no_index_test (I2);

Mach> EXPLAIN SELECT * FROM TEST WHERE I1 = 1;
PLAN
------------------------------------------------------------------------------------
 PROJECT
  INDEX SCAN
   *BITMAP RANGE (t:7, c:1, i:8) with BLOOMFILTER
   [KEY RANGE]
    * I1 = 1
[5] row(s) selected.

Mach> EXPLAIN SELECT /*+ NO_INDEX(TEST,TEST_IDX) */ * FROM TEST WHERE I1 = 1;
PLAN
------------------------------------------------------------------------------------
 PROJECT
  FULL SCAN
[2] row(s) selected.

Mach> EXPLAIN SELECT * FROM log_no_index_test WHERE I1 = 1 or I2 = 2;
PLAN
------------------------------------------------------------------------------------
 PROJECT
  INDEX SCAN
   INDEX (OR)
    *BITMAP RANGE (table id:21, column id:2, index id:22)
    *BITMAP RANGE (table id:21, column id:3, index id:23)
   [KEY RANGE]
```

```
    * I1 = 1 or I2 = 2
[7] row(s) selected.

Mach> EXPLAIN SELECT /*+ NO_INDEX(log_no_index_test, idx_I1) */ * FROM log_no_index_test WHERE I1 = 1 or I2 = 2;
PLAN
--------------------------------------------------------------------------------
 PROJECT
  FULL SCAN (LOG_NO_INDEX_TEST)
[2] row(s) selected.

Mach> EXPLAIN SELECT * FROM log_no_index_test WHERE I1 = 1 and I2 = 2;
PLAN
--------------------------------------------------------------------------------
 PROJECT
  INDEX SCAN
   *BITMAP RANGE (table id:21, column id:2, index id:22)
   *BITMAP RANGE (table id:21, column id:3, index id:23)
   [KEY RANGE]
    * I1 = 1
    * I2 = 2
[7] row(s) selected.

Mach> EXPLAIN SELECT /*+ NO_INDEX(log_no_index_test, idx_I1) */ * FROM log_no_index_test WHERE I1 = 1 and I2 = 2;

PLAN
--------------------------------------------------------------------------------
 PROJECT
  INDEX SCAN
   *BITMAP RANGE (table id:21, column id:3, index id:23)
   [KEY RANGE]
    * I2 = 2
   [FILTER]
    * I1 = 1
[7] row(s) selected.
Elapsed time: 0.001
Mach>
Mach>
Mach> EXPLAIN SELECT /*+ NO_INDEX(log_no_index_test, idx_I2) */ * FROM log_no_index_test WHERE I1 = 1 and I2 = 2;

PLAN
--------------------------------------------------------------------------------
 PROJECT
  INDEX SCAN
   *BITMAP RANGE (table id:21, column id:2, index id:22)
   [KEY RANGE]
    * I1 = 1
   [FILTER]
    * I2 = 2
[7] row(s) selected.
```

RID_RANGE

RID 범위 내에서 수행한다.

```
SELECT /*+ RID_RANGE(table_name,number,number) */ ...
```

```
Mach> SELECT /*+ RID_RANGE(TEST,45,50) */ _RID, * FROM TEST;
_RID                 I1
-----------------------------------
49                   1
48                   1
47                   1
46                   1
45                   1
[5] row(s) selected.
```

SCAN_FORWARD, SCAN_BACKWARD

테이블의 스캔 방향을 지정한다. SCAN_FORWARD를 힌트로 사용하면 가장 먼저 입력한 레코드 우선으로, SCAN_BACKWARD를 힌트로 사용하면 가장 나중에 입력한 레코드 우선으로 조회한다.

> Fog edition의 LOG 테이블에 대해서만 지원된다.

```
SELECT /*+ SCAN_FORWARD(table_name) */ ...
SELECT /*+ SCAN_BACKWARD(table_name) */ ...
```

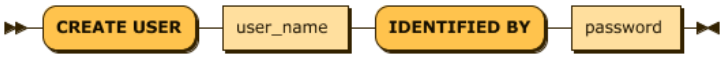
```
Mach> SELECT /*+ SCAN_FORWARD(mytbl) */  _ARRIVAL_TIME, VALUE FROM mytbl LIMIT 10;
_ARRIVAL_TIME                   VALUE
---------------------------------------------------------------
2017-01-01 00:00:49 500:000:000 0
2017-01-01 00:01:39 500:000:000 1
2017-01-01 00:02:29 500:000:000 2
2017-01-01 00:03:19 500:000:000 3
2017-01-01 00:04:09 500:000:000 4
2017-01-01 00:04:59 500:000:000 5
2017-01-01 00:05:49 500:000:000 6
2017-01-01 00:06:39 500:000:000 7
2017-01-01 00:07:29 500:000:000 8
2017-01-01 00:08:19 500:000:000 9
[10] row(s) selected.

Mach> SELECT /*+ SCAN_BACKWARD(mytbl) */ _ARRIVAL_TIME, VALUE FROM mytbl LIMIT 10;
_ARRIVAL_TIME                   VALUE
---------------------------------------------------------------
2017-02-27 20:53:19 500:000:000 9
2017-02-27 20:52:29 500:000:000 8
2017-02-27 20:51:39 500:000:000 7
2017-02-27 20:50:49 500:000:000 6
2017-02-27 20:49:59 500:000:000 5
2017-02-27 20:49:09 500:000:000 4
2017-02-27 20:48:19 500:000:000 3
2017-02-27 20:47:29 500:000:000 2
2017-02-27 20:46:39 500:000:000 1
2017-02-27 20:45:49 500:000:000 0
[10] row(s) selected.

Mach>
```

사용자 관리

목차

CREATE USER

create_user_stmt:



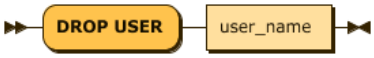

```
create_user_stmt ::= 'CREATE USER' user_name 'IDENTIFIED BY' password
```

사용자를 생성하는 구문은 다음과 같다.

```
--예제
CREATE USER new_user IDENTIFIED BY password
```

DROP USER

drop_user_stmt:



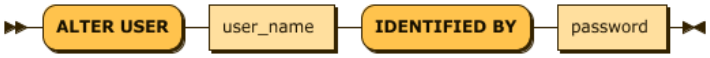

```
drop_user_stmt ::= 'DROP USER' user_name
```

사용자를 삭제하는 구문은 다음과 같다. SYS 사용자는 삭제할 수 없으며, 삭제 대상 사용자가 이미 생성한 테이블이 있을 경우에는 에러를 나타낸다.

```
--예제
DROP USER old_user
```

ALTER USER

alter_user_pwd_stmt:




```
alter_user_pwd_stmt ::= 'ALTER USER' user_name 'IDENTIFIED BY' password
```

사용자는 아래의 구문을 통해 비밀번호를 변경할 수 있다.

```
--예제
ALTER USER user1 IDENTIFIED BY password
```

CONNECT

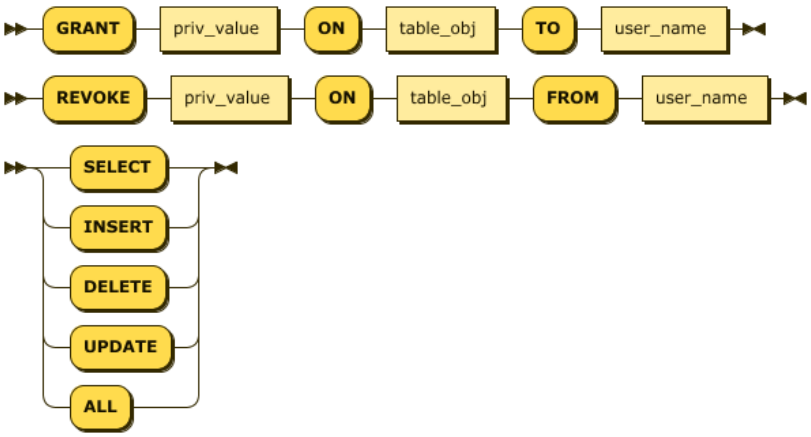
user_connect_stmt:


```
user_connect_stmt: 'CONNECT' user_name '/' password
```

사용자는 응용 프로그램을 종료하지 않고, 다음의 구문을 통해 다른 사용자로 재접속할 수 있다.

```
--예제
CONNECT user1/password;
```

GRANT/REVOKE



GRANT 구문을 통하여 사용자에게 테이블에 대한 권한을 부여한다.

```
-- user1 에게 mytable 에 대한 SELECT 권한 부여
GRANT SELECT ON mytable TO user1;

-- user1 에게 mytable 에 대한 모든 권한 부여
GRANT ALL ON mytable TO user1;
```

REVOKE 구문을 통하여 사용자에게 부여된 권한을 회수한다.

```
-- user1 에게 부여된 mytable 에 대한 UPDATE 권한 회수
REVOKE UPDATE ON mytable FROM user1;

-- user1 에게 부여된 mytable 에 대한 모든 권한 회수
REVOKE ALL ON mytable FROM user1;
```

사용자 관리 예제

위의 쿼리를 수행한 예제와 그 결과를 나타냈다.

```
###############################################
# Connect SYS : SYS 계정으로 접속함.
###############################################
Mach> create user demo identified by 'demo';
Created successfully.

Mach> drop user demo;
Dropped successfully.

Mach> create user demo1 identified by 'demo1';
Created successfully.

Mach> create user demo2 identified by 'demo2';
Created successfully.

Mach> alter user demo2 identified by 'demo22';
Altered successfully.

Mach> create table demo1_table (id integer);
```

```
Created successfully.

Mach> create bitmap index demo1_table_index1 on demo1_table(id);
Created successfully.

Mach> insert into demo1_table values(99991);
1 row(s) inserted.

Mach> insert into demo1_table values(99992);
1 row(s) inserted.

Mach> insert into demo1_table values(99993);
1 row(s) inserted.

Mach> select * from demo1_table;
ID
--------------
99993
99992
99991
[3] row(s) selected.

#Error: 자기 자신을 Drop 할 수 없음.
Mach> drop user SYS;
[ERR-02083 : Drop user error. You cannot drop yourself(SYS).]

###############################################
# Connect DEMO1
###############################################
Mach> connect demo1/demo1;
Connected successfully.

#Error: 일반 유저는 다른 사용자의 비밀번호를 바꿀 수 없다.
Mach> alter user demo2 identified by 'demo22';
[ERR-02085 : ALTER user error. The user(DEMO2) does not have ALTER privileges.]

Mach> alter user demo1 identified by demo11;
Altered successfully.

Mach> connect demo2/demo22;
Connected successfully.

#Error: wrong password
Mach> connect demo1/demo11234;
 [ERR-02081 : User authentication error. Invalid password (DEMO11234).]

# Correct password
Mach> connect demo1/demo11;
Connected successfully.

Mach> create table demo1_table (id integer);
Created successfully.

Mach> create bitmap index demo1_table_index1 on demo1_table(id);
Created successfully.

Mach> insert into demo1_table values(1);
1 row(s) inserted.

Mach> insert into demo1_table values(2);
1 row(s) inserted.

Mach> insert into demo1_table values(3);
1 row(s) inserted.

Mach> select * from demo1_table;
ID
--------------
3
2
```

```
1
[3] row(s) selected.

Mach> select * from demo1.demo1_table;
ID
--------------
3
2
1
[3] row(s) selected.

############################################
# Connect SYS again
############################################
Mach> connect SYS/MANAGER;
Connected successfully.

Mach> select * from demo1_table;
ID
--------------
99993
99992
99991
[3] row(s) selected.

Mach> select * from demo1.demo1_table;
ID
--------------
3
2
1
[3] row(s) selected.

#Error: demo1 유저에 속한 테이블이 존재함.
Mach> drop user demo1;
[ERR-02084 : DROP user error. The user's tables still exist. Drop those tables first.]

Mach> connect demo1/demo11;
Connected successfully.

Mach> drop table demo1_table;
Dropped successfully.

Mach> connect SYS/MANAGER;
Connected successfully.

Mach> drop user demo1;
Dropped successfully.
```

지원 함수

ABS

이 함수는 숫자형 컬럼에 대해 동작하고, 양의 값으로 변환해서 실수형으로 값을 리턴한다.

```
ABS(column_expr)
```

```
Mach> CREATE TABLE abs_table (c1 INTEGER, c2 DOUBLE, c3 VARCHAR(10));
Created successfully.

Mach> INSERT INTO abs_table VALUES(1, 1.0, '');
1 row(s) inserted.

Mach> INSERT INTO abs_table VALUES(2, 2.0, 'sqltest');
1 row(s) inserted.

Mach> INSERT INTO abs_table VALUES(3, 3.0, 'sqltest');
1 row(s) inserted.

Mach> SELECT ABS(c1), ABS(c2) FROM abs_table;
SELECT ABS(c1), ABS(c2) from abs_table;
ABS(c1)                     ABS(c2)
------------------------------------------------------------
3                           3
2                           2
1                           1
[3] row(s) selected.
```

ADD_TIME

이 함수는 주어진 datetime 컬럼에 대해 날짜 가감연산을 수행한다. 연, 월, 일, 시간, 분, 초 까지의 가감 연산을 지원하며, 밀리, 마이크로, 나노초에 대한 연산은 지원하지 않는다. Diff format은 다음과 같다. "Year/Month/Day Hour:Minute:Second" 각 항목은 양수 혹은 음수 값을 가진다.

```
ADD_TIME(column,time_diff_format)
```

```
Mach> CREATE TABLE add_time_table (id INTEGER, dt DATETIME);
Created successfully.

Mach> INSERT INTO  add_time_table VALUES(1, TO_DATE('1999-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(2, TO_DATE('2000-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(3, TO_DATE('2012-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(4, TO_DATE('2013-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(5, TO_DATE('2014-12-30 11:22:33 444:55
1 row(s) inserted.

Mach> INSERT INTO  add_time_table VALUES(6, TO_DATE('2014-12-30 23:22:33 444:55
1 row(s) inserted.

Mach> SELECT ADD_TIME(dt, '1/0/0 0:0:0') FROM add_time_table;
ADD_TIME(dt, '1/0/0 0:0:0')
---------------------------------
```

목차

```
2015-12-30 23:22:33 444:555:666
2015-12-30 11:22:33 444:555:666
2014-11-11 01:02:03 004:005:006
2013-11-11 01:02:03 004:005:006
2001-11-11 01:02:03 004:005:006
2000-11-11 01:02:03 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '0/0/0 1:1:1') FROM add_time_table;
ADD_TIME(dt, '0/0/0 1:1:1')
---------------------------------
2014-12-31 00:23:34 444:555:666
2014-12-30 12:23:34 444:555:666
2013-11-11 02:03:04 004:005:006
2012-11-11 02:03:04 004:005:006
2000-11-11 02:03:04 004:005:006
1999-11-11 02:03:04 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '1/1/1 0:0:0') FROM add_time_table;
ADD_TIME(dt, '1/1/1 0:0:0')
---------------------------------
2016-01-31 23:22:33 444:555:666
2016-01-31 11:22:33 444:555:666
2014-12-12 01:02:03 004:005:006
2013-12-12 01:02:03 004:005:006
2001-12-12 01:02:03 004:005:006
2000-12-12 01:02:03 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '-1/0/0 0:0:0') FROM add_time_table;
ADD_TIME(dt, '-1/0/0 0:0:0')
---------------------------------
2013-12-30 23:22:33 444:555:666
2013-12-30 11:22:33 444:555:666
2012-11-11 01:02:03 004:005:006
2011-11-11 01:02:03 004:005:006
1999-11-11 01:02:03 004:005:006
1998-11-11 01:02:03 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '0/0/0 -1:-1:-1') FROM add_time_table;
ADD_TIME(dt, '0/0/0 -1:-1:-1')
---------------------------------
2014-12-30 22:21:32 444:555:666
2014-12-30 10:21:32 444:555:666
2013-11-11 00:01:02 004:005:006
2012-11-11 00:01:02 004:005:006
2000-11-11 00:01:02 004:005:006
1999-11-11 00:01:02 004:005:006
[6] row(s) selected.

Mach> SELECT ADD_TIME(dt, '-1/-1/-1 0:0:0') FROM add_time_table;
ADD_TIME(dt, '-1/-1/-1 0:0:0')
---------------------------------
2013-11-29 23:22:33 444:555:666
2013-11-29 11:22:33 444:555:666
2012-10-10 01:02:03 004:005:006
2011-10-10 01:02:03 004:005:006
1999-10-10 01:02:03 004:005:006
1998-10-10 01:02:03 004:005:006
[6] row(s) selected.

Mach> SELECT * FROM add_time_table WHERE dt > ADD_TIME(TO_DATE('2014-12-30 11:2
ID          DT
-----------------------------------------------
6           2014-12-30 23:22:33 444:555:666
5           2014-12-30 11:22:33 444:555:666
[2] row(s) selected.
```

```
Mach> SELECT * FROM add_time_table WHERE dt > ADD_TIME(TO_DATE('2014-12-30 11:2
ID          DT
----------------------------------------------
6           2014-12-30 23:22:33 444:555:666
5           2014-12-30 11:22:33 444:555:666
4           2013-11-11 01:02:03 004:005:006
[3] row(s) selected.

Mach> SELECT ADD_TIME(TO_DATE('2000-12-01 00:00:00 000:000:001'), '-1/0/0 0:0:-
ADD_TIME(TO_DATE('2000-12-01 00:00:00 000:000:001'), '-1/0/0 0:0:-1')
----------------------------------------
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
1999-11-30 23:59:59 000:000:001
[6] row(s) selected.

Mach> SELECT * FROM add_time_table WHERE dt > ADD_TIME(TO_DATE('2014-12-30 11:2
ID          DT
----------------------------------------------
6           2014-12-30 23:22:33 444:555:666
5           2014-12-30 11:22:33 444:555:666
4           2013-11-11 01:02:03 004:005:006
[3] row(s) selected.
```

AVG

이 함수는 집계 함수로써, 숫자형 컬럼에 대해 동작하고 해당 컬럼의 평균 값을 출력한다.

```
AVG(column_name)
```

```
Mach> CREATE TABLE avg_table (id1 INTEGER, id2 INTEGER);
Created successfully.

Mach> INSERT INTO avg_table VALUES(1, 1);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(1, 2);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(1, 3);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(2, 1);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(2, 2);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(2, 3);
1 row(s) inserted.

Mach> INSERT INTO avg_table VALUES(null, 4);
1 row(s) inserted.

Mach> SELECT id1, AVG(id2) FROM avg_table GROUP BY id1;
id1         AVG(id2)
----------------------------------------
2                2
NULL             4
1                2
```

BITAND / BITOR

이 함수는 두 입력 값을 64-bit 의 부호 있는 정수로 변환한 뒤, 비트별 and/or 을 수행한 결과를 반환한다. 입력 값은 반드시 정수형이어야 하며, 출력 값은 64비트 부호 있는 정수가 된다.

0보다 작은 정수값에 대해서는 플랫폼에 따라 다른 결과를 얻을 수 있으므로 uinteger, ushort 타입만 사용하기를 권장한다.

```
BITAND (<expression1>, <expression2>)
BITOR (<expression1>, <expression2>)
```

```
Mach> CREATE TABLE bit_table (i1 INTEGER, i2 UINTEGER, i3 FLOAT, i4 DOUBLE, i5 SHORT, i6 VARCHAR(10));
Created successfully.

Mach> INSERT INTO bit_table VALUES (-1, 1, 1, 1, 2, 'aaa');
1 row(s) inserted.

Mach> INSERT INTO bit_table VALUES (-2, 2, 2, 2, 3, 'bbb');
1 row(s) inserted.

Mach> SELECT BITAND(i1, i2) FROM bit_table;
BITAND(i1, i2)
-----------------------
2
1
[2] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITAND(i2, 1) = 1;
I1          I2          I3                          I4                          I5          I6
-------------------------------------------------------------------------------------------------------------------
-1          1           1                           1                           2           aaa
[1] row(s) selected.

Mach> SELECT BITOR(i5, 1) FROM bit_table WHERE BITOR(i5, 1) = 3;
BITOR(i5, 1)
-----------------------
3
3
[2] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITOR(i2, 1) = 1;
I1          I2          I3                          I4                          I5          I6
-------------------------------------------------------------------------------------------------------------------
-1          1           1                           1                           2           aaa
[1] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITAND(i3, 1) = 1;
I1          I2          I3                          I4                          I5          I6
-------------------------------------------------------------------------------------------------------------------
[ERR-02037 : Function [BITAND] argument data type is mismatched.]
[0] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITAND(i4, 1) = 1;
I1          I2          I3                          I4                          I5          I6
-------------------------------------------------------------------------------------------------------------------
[ERR-02037 : Function [BITAND] argument data type is mismatched.]
[0] row(s) selected.

Mach> SELECT BITAND(i5, 1) FROM bit_table WHERE BITAND(i5, 1) = 1;
BITAND(i5, 1)
-----------------------
1
[1] row(s) selected.

Mach> SELECT * FROM bit_table WHERE BITOR(i6, 1) = 1;
I1          I2          I3                          I4                          I5          I6
-------------------------------------------------------------------------------------------------------------------
[ERR-02037 : Function [BITOR] argument data type is mismatched.]
[0] row(s) selected.
```

```
Mach> SELECT BITOR(i1, i2) FROM bit_table;
BITOR(i1, i2)
-----------------------
-2
-1
[2] row(s) selected.

Mach> SELECT BITAND(i1, i3) FROM bit_table;
BITAND(i1, i3)
-----------------------
[ERR-02037 : Function [BITAND] argument data type is mismatched.]
[0] row(s) selected.

Mach> SELECT BITOR(i1, i6) FROM bit_table;
BITOR(i1, i6)
-----------------------
[ERR-02037 : Function [BITOR] argument data type is mismatched.]
[0] row(s) selected.
```

COUNT

이 함수는 집계 함수로써, 해당 컬럼의 레코드 개수를 구하는 함수이다.

```
COUNT(column_name)
```

```
Mach> CREATE TABLE count_table (id1 INTEGER, id2 INTEGER);
Created successfully.

Mach> INSERT INTO count_table VALUES(1, 1);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(1, 2);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(1, 3);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(2, 1);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(2, 2);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(2, 3);
1 row(s) inserted.

Mach> INSERT INTO count_table VALUES(null, 4);
1 row(s) inserted.

Mach> SELECT COUNT(*) FROM count_table;
COUNT(*)
-----------------------
7
[1] row(s) selected.

Mach> SELECT COUNT(id1) FROM count_table;
COUNT(id1)
-----------------------
6
[1] row(s) selected.
```

DATE_TRUNC

이 함수는 주어진 datetime 값을 '시간 단위'와 '시간 범위'까지만 표시된 새로운 datetime 값으로 반환한다.

```
DATE_TRUNC (field, date_val [, count])
```

```
Mach> CREATE TABLE trunc_table (i1 INTEGER, i2 DATETIME);
Created successfully.

Mach> INSERT INTO trunc_table VALUES (1, TO_DATE('1999-11-11 1:2:0 4:5:1'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (2, TO_DATE('1999-11-11 1:2:0 5:5:2'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (3, TO_DATE('1999-11-11 1:2:1 6:5:3'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (4, TO_DATE('1999-11-11 1:2:1 7:5:4'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (5, TO_DATE('1999-11-11 1:2:2 8:5:5'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (6, TO_DATE('1999-11-11 1:2:2 9:5:6'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (7, TO_DATE('1999-11-11 1:2:3 10:5:7'));
1 row(s) inserted.

Mach> INSERT INTO trunc_table VALUES (8, TO_DATE('1999-11-11 1:2:3 11:5:8'));
1 row(s) inserted.

Mach> SELECT COUNT(*), DATE_TRUNC('second', i2) tm FROM trunc_table group by tm ORDER BY 2;
COUNT(*)             tm
-------------------------------------------------------
2                    1999-11-11 01:02:00 000:000:000
2                    1999-11-11 01:02:01 000:000:000
2                    1999-11-11 01:02:02 000:000:000
2                    1999-11-11 01:02:03 000:000:000
[4] row(s) selected.

Mach> SELECT COUNT(*), DATE_TRUNC('second', i2, 2) tm FROM trunc_table group by tm ORDER BY 2;
COUNT(*)             tm
-------------------------------------------------------
4                    1999-11-11 01:02:00 000:000:000
4                    1999-11-11 01:02:02 000:000:000
[2] row(s) selected.

Mach> SELECT COUNT(*), DATE_TRUNC('nanosecond', i2, 2) tm FROM trunc_table group by tm ORDER BY 2;
COUNT(*)             tm
-------------------------------------------------------
1                    1999-11-11 01:02:00 004:005:000
1                    1999-11-11 01:02:00 005:005:002
1                    1999-11-11 01:02:01 006:005:002
1                    1999-11-11 01:02:01 007:005:004
1                    1999-11-11 01:02:02 008:005:004
1                    1999-11-11 01:02:02 009:005:006
1                    1999-11-11 01:02:03 010:005:006
1                    1999-11-11 01:02:03 011:005:008
[8] row(s) selected.

Mach> SELECT COUNT(*), DATE_TRUNC('nsec', i2, 1000000000) tm FROM trunc_table group by tm ORDER BY 2; //DATE_TRUNC(
COUNT(*)             tm
-------------------------------------------------------
2                    1999-11-11 01:02:00 000:000:000
2                    1999-11-11 01:02:01 000:000:000
2                    1999-11-11 01:02:02 000:000:000
2                    1999-11-11 01:02:03 000:000:000
[4] row(s) selected.
```

시간 단위와, 시간 단위별 허용되는 시간 범위는 다음과 같다.

> nanosecond, microsecond, milisecond 단위와 축약어는, 5.5.6 부터 사용 가능하다.

시간 단위 (축약어)	시간 범위
nanosecond (nsec)	1000000000 (1초)
microsecond (usec)	60000000 (60초)
milisecond (msec)	60000 (60초)
second (sec)	86400 (1일)
minute (min)	1440 (1일)
hour	24 (1일)
day	1
month	1
year	1

예를 들어, `DATE_TRUNC('second', time, 120)` 으로 입력하면, 반환되는 값은 **2분 간격**으로 표시될 것이며 이는 `DATE_TRUNC('minute', time, 2)` 와 동일하다.

DAYOFWEEK

이 함수는 주어진 datetime 값의 요일을 나타내는 자연수를 반환한다.

TO_CHAR(time, 'DAY') 와 의미상 비슷한 값을 반환하지만, 여기서는 정수를 반환한다.

```
DAYOFWEEK(date_val)
```

반환되는 자연수는 다음의 요일을 나타낸다.

반환값	요일
0	일요일
1	월요일
2	화요일
3	수요일
4	목요일
5	금요일
6	토요일

DECODE

이 함수는 주어진 Column 값을 Search와 같은지 비교하고, 같으면 바로 다음의 Return 값을 되돌린다. 만일 만족하는 Search 값이 없을 경우에는 Default 값을 리턴한다. Default가 생략되었을 경우에는 NULL이 리턴된다.

```
DECODE(column, [search, return],.. default)
```

```
Mach> CREATE TABLE decode_table (id1 VARCHAR(11));
Created successfully.

Mach> INSERT INTO decode_table VALUES('decodetest1');
```

```
1 row(s) inserted.

Mach> INSERT INTO decode_table VALUES('decodetest2');
1 row(s) inserted.

Mach> SELECT id1, DECODE(id1, 'decodetest1', 'result1', 'decodetest2', 'result2', 'DEFAULT') FROM decode_table;
id1          DECODE(id1, 'decodetest1', 'result1', 'decodetest2', 'result2', 'DEFAULT')
-----------------------------------------------------------
decodetest2  result2
decodetest1  result1
[2] row(s) selected.

Mach> SELECT id1, DECODE(id1, 'codetest', 2, 99) FROM decode_table;
id1          DECODE(id1, 'codetest', 2, 99)
-----------------------------------------------
decodetest2  99
decodetest1  99
[2] row(s) selected.

Mach> SELECT DECODE(id1, 'decodetest1', 2) FROM decode_table;
DECODE(id1, 'decodetest1', 2)
--------------------------------
NULL
2
[2] row(s) selected.

Mach> SELECT DECODE(id1, 'codetest', 2) FROM decode_table;
DECODE(id1, 'codetest', 2)
------------------------------
NULL
NULL
[2] row(s) selected.
```

FIRST / LAST

이 함수는 집계 함수로써, 각 Group 에서 '기준 값'이 순서상 가장 앞선 (또는 가장 나중의) 레코드의 특정 값을 반환한다.

- FIRST : 순서상 가장 앞선 레코드에서 특정 값을 반환
- LAST : 순서상 가장 나중의 레코드에서 특정 값을 반환

```
FIRST(sort_expr, return_expr)
LAST(sort_expr, return_expr)
```

```
Mach> create table firstlast_table (id integer, name varchar(20), group_no integer);
Created successfully.
Mach> insert into firstlast_table values (1, 'John', 0);
1 row(s) inserted.
Mach> insert into firstlast_table values (2, 'Grey', 1);
1 row(s) inserted.
Mach> insert into firstlast_table values (5, 'Ryan', 0);
1 row(s) inserted.
Mach> insert into firstlast_table values (4, 'Andrew', 0);
1 row(s) inserted.
Mach> insert into firstlast_table values (7, 'Kyle', 1);
1 row(s) inserted.
Mach> insert into firstlast_table values (6, 'Ross', 1);
1 row(s) inserted.

Mach> select group_no, first(id, name) from firstlast_table group by group_no;
group_no    first(id, name)
------------------------------------
1           Grey
0           John
[2] row(s) selected.


Mach> select group_no, last(id, name) from firstlast_table group by group_no;
```

```
group_no    last(id, name)
------------------------------------
1           Kyle
0           Ryan
```

FROM_UNIXTIME

이 함수는 정수형으로 입력된 32비트 UNIXTIME 값을 datetime 자료형의 값으로 변환한다. (UNIX_TIMESTAMP 는 datetime 자료형을 32비트 UNIXTIME 정수형 데이터로 변환한다.)

```
FROM_UNIXTIME(unix_timestamp_value)
```

```
Mach> SELECT FROM_UNIXTIME(315540671) FROM TEST;
FROM_UNIXTIME(315540671)
---------------------------------
1980-01-01 11:11:11 000:000:000

Mach> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP('2001-01-01')) FROM unix_table;
FROM_UNIXTIME(UNIX_TIMESTAMP('2001-01-01'))
-----------------------------------------
2001-01-01 00:00:00 000:000:000
```

FROM_TIMESTAMP

이 함수는 1970-01-01 09:00 부터 경과된 nanosecond 값을 입력받아 datetime 자료형의 값으로 변환한다. (TO_TIMESTAMP() 는 datetime 자료형을 1970-01-01 09:00 부터 경과된 nanosecond 데이터로 변환한다.)

```
FROM_TIMESTAMP(nanosecond_time_value)
```

```
Mach> SELECT FROM_TIMESTAMP(1562302560007248869) FROM TEST;
FROM_TIMESTAMP(1562302560007248869)
-----------------------------------
2019-07-05 13:56:00 007:248:869
```

sysdate, now 는 모두 현재 시각의 1970-01-01 09:00 부터 경과된 nanosecond 값을 나타내므로, 곧바로 FROM_TIMESTAMP() 를 사용해도 된다.

물론, 사용하지 않아도 결과는 같다. sysdate 와 now 에 nanosecond 단위로 연산을 한 경우에 유용하게 사용할 수 있다.

```
Mach> select sysdate, from_timestamp(sysdate) from test_tbl;
sysdate                         from_timestamp(sysdate)
-------------------------------------------------------------------
2019-07-05 14:00:59 722:822:443 2019-07-05 14:00:59 722:822:443
[1] row(s) selected.

Mach> select sysdate, from_timestamp(sysdate-1000000) from test_tbl;
sysdate                         from_timestamp(sysdate-1000000)
-------------------------------------------------------------------
2019-07-05 14:01:05 130:939:525 2019-07-05 14:01:05 129:939:525      -- 1 ms (1,000,000 ns) 차이가 발생함
[1] row(s) selected.
```

GROUP_CONCAT

이 함수는 집계 함수로써, 그룹 안에 존재하는 해당 컬럼의 값을 문자열로 이어 붙여서 출력한다.

> 📌 Cluster Edition 에서는 사용할 수 없는 함수이다.

```
GROUP_CONCAT(
```

```
    [DISTINCT] column
    [ORDER BY { unsigned_integer | column }
    [ASC | DESC] [, column ...]]
    [SEPARATOR str_val]
)
```

- DISTINCT: 이어 붙일 컬럼의 값이 중복되는 경우, 중복된 값은 이어 붙이지 않는다.
- ORDER BY: 지정된 컬럼 값들에 따라, 이어 붙이는 컬럼 값의 순서를 정렬한다.
- SEPARATOR: 컬럼 값을 이어 붙일 때 사용하는 구분자 문자열로, 기본 값은 콤마(,)이다.

문법에 대한 주의사항은 아래와 같다.

- 이어 붙일 컬럼은 1개만 지정할 수 있으며, 2개 이상 지정하고자 하는 경우에는 TO_CHAR() 함수와 CONCAT 연산자 (||)를 활용해서 1개의 표현식으로 만들어서 입력해야 한다.
- ORDER BY 에는 이어 붙이는 컬럼 외에 다른 컬럼을 지정할 수 있으며, 여러 컬럼을 지정할 수 있다.
- SEPARATOR 에는 반드시 문자열 상수를 입력해야 하며, 문자열 컬럼은 입력할 수 없다.

```
Mach> CREATE TABLE concat_table(id1 INTEGER, id2 DOUBLE, name VARCHAR(10));
Created successfully.

Mach> INSERT INTO concat_table VALUES (1, 2, 'John');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (2, 1, 'Ram');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (3, 2, 'Zara');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (4, 2, 'Jill');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (5, 1, 'Jack');
1 row(s) inserted.

Mach> INSERT INTO concat_table VALUES (6, 1, 'Jack');
1 row(s) inserted.


Mach> SELECT GROUP_CONCAT(name) AS G_NAMES FROM concat_table GROUP BY id2;
G_NAMES
--------------------------------------------------------------------------------
Jack,Jack,Ram
Jill,Zara,John
[2] row(s) selected.

Mach> SELECT GROUP_CONCAT(DISTINCT name) AS G_NAMES FROM concat_table GROUP BY Id2;
G_NAMES
--------------------------------------------------------------------------------
Jack,Ram
Jill,Zara,John
[2] row(s) selected.

Mach> SELECT GROUP_CONCAT(name SEPARATOR '.') G_NAMES FROM concat_table GROUP BY Id2;
G_NAMES
--------------------------------------------------------------------------------
Jack.Jack.Ram
Jill.Zara.John
[2] row(s) selected.

Mach> SELECT GROUP_CONCAT(name ORDER BY id1) G_NAMES, GROUP_CONCAT(id1 ORDER BY id1) G_SORTID FROM concat_table GR
G_NAMES
--------------------------------------------------------------------------------
G_SORTID
--------------------------------------------------------------------------------
Ram,Jack,Jack
2,5,6
John,Zara,Jill
1,3,4
[2] row(s) selected.
```

INSTR

이 함수는 입력된 문자열 패턴이, 함께 입력된 문자열에서 몇 번째 문자에 있는지 그 인덱스를 반환한다. 인덱스 시작은 1 이다.

- 문자열 패턴을 찾을 수 없는 경우, 0 이 반환된다.
- 찾고자 하는 문자열 패턴의 길이가 0이거나 NULL 인 경우, NULL 이 반환된다.

```
INSTR(target_string, pattern_string)
```

```
Mach> CREATE TABLE string_table(c1 VARCHAR(20));
Created successfully.

Mach> INSERT INTO string_table VALUES ('abstract');
1 row(s) inserted.

Mach> INSERT INTO string_table VALUES ('override');
1 row(s) inserted.

Mach> SELECT c1, INSTR(c1, 'act') FROM string_table;
c1                   INSTR(c1, 'act')
-----------------------------------------
override             0
abstract             6
[2] row(s) selected.
```

LEAST / GREATEST

두 함수는 입력 매개변수로 여러 개의 컬럼 또는 값들을 지정하면 그중 가장 작은 값(LEAST) 또는 가장 큰 값(GREATEST)을 반환한다.

만약 입력값이 1개 또는 없는 경우에는 오류로 처리되며 입력값들 중에 NULL이 있는 경우에는 NULL을 반환하므로 입력값이 컬럼인 경우 함수등을 이용하여 미리 변환하여야 한다.

입력값의 비교가 불가능한 컬럼(BLOB, TEXT) 등이 포함되어 있거나 대소비교를 위한 형 변환이 불가능한 경우 오류로 처리된다.

```
LEAST(value_list, value_list,...)
GREATEST(value_list, value_list,...)
```

```
Mach> CREATE TABLE lgtest_table(c1 INTEGER, c2 LONG, c3 VARCHAR(10), c4 VARCHAR(5));
Created successfully.

Mach> INSERT INTO lgtest_table VALUES (1, 2, 'abstract', 'ace');
1 row(s) inserted.

Mach> INSERT INTO lgtest_table VALUES (null, 100, null, 'bag');
1 row(s) inserted.

Mach> SELECT LEAST (c1, c2) FROM lgtest_table;
LEAST (c1, c2)
-----------------------
NULL
1
[2] row(s) selected.

Mach> SELECT LEAST (c1, c2, -1) FROM lgtest_table;
LEAST (c1, c2, -1)
-----------------------
NULL
-1
[2] row(s) selected.

Mach> SELECT GREATEST(c3, c4) FROM lgtest_table;
GREATEST(c3, c4)
--------------------
```

```
NULL
ace
[2] row(s) selected.

Mach> SELECT LEAST(c3, c4) FROM lgtest_table;
LEAST(c3, c4)
-----------------
NULL
abstract
[2] row(s) selected.

Mach> SELECT LEAST(NVL(c3, 'aa'), c4) FROM lgtest_table;
LEAST(NVL(c3, 'aa'), c4)
----------------------------
aa
abstract
[2] row(s) selected.
```

LENGTH

이 함수는 문자열 컬럼의 길이를 구한다. 구해진 값은 영문 기준으로 바이트 수를 출력한다.

```
LENGTH(column_name)
```

```
Mach> CREATE TABLE length_table (id1 INTEGER, id2 DOUBLE, name VARCHAR(15));
Created successfully.

Mach> INSERT INTO length_table VALUES(1, 10, 'Around the Horn');
1 row(s) inserted.

Mach> INSERT INTO length_table VALUES(NULL, 20, 'Alfreds Futterkiste');
1 row(s) inserted.

Mach> INSERT INTO length_table VALUES(3, NULL, 'Antonio Moreno');
1 row(s) inserted.

Mach> INSERT INTO length_table VALUES(4, 40, NULL);
1 row(s) inserted.

Mach> select * FROM length_table;
ID1         ID2                         NAME
-------------------------------------------------------------
4           40                          NULL
3           NULL                        Antonio Moreno
NULL        20                          Alfreds Futterk
1           10                          Around the Horn
[4] row(s) selected.

Mach> select id1 * 10 FROM length_table;
id1 * 10
-----------------------
40
30
NULL
10
[4] row(s) selected.

Mach> select * FROM length_table Where id1 > 1 and id2 < 50;
ID1         ID2                         NAME
-------------------------------------------------------------
4           40                          NULL
[1] row(s) selected.

Mach> select name || ' with null concat' FROM length_table;
name || ' with null concat'
-----------------------------------
```

```
NULL
Antonio Moreno with null concat
Alfreds Futterk with null concat
Around the Horn with null concat
[4] row(s) selected.

Mach> select LENGTH(name) FROM length_table;
LENGTH(name)
---------------
NULL
14
15
15
[4] row(s) selected.
```

LOWER

이 함수는 영문 문자열을 소문자로 변환한다.

```
LOWER(column_name)
```

```
Mach> CREATE TABLE lower_table (name VARCHAR(20));
Created successfully.

Mach> INSERT INTO lower_table VALUES('');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES('James Backley');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES('Alfreds Futterkiste');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES('Antonio MORENO');
1 row(s) inserted.

Mach> INSERT INTO lower_table VALUES (NULL);
1 row(s) inserted.

Mach> SELECT LOWER(name) FROM lower_table;
LOWER(name)
------------------------
NULL
antonio moreno
alfreds futterkiste
james backley
NULL
[5] row(s) selected.
```

LPAD / RPAD

이 함수는 입력 값을 주어진 길이(length)가 될때까지 문자(char)를 왼쪽(LPAD) 또는 오른쪽(RPAD)에 덧붙이는 함수이다.

마지막 매개변수인 char는 생략이 가능하며 생략된 경우에는 공백 문자 ' ' 를 이용한다.
입력된 컬럼값이 length로 주어진 길이보다 긴 경우에는 문자를 덧붙이지 않고 앞에서부터 length 만큼만 가져온다.

```
LPAD(str, len, padstr)
RPAD(str, len, padstr)
```

```
Mach> CREATE TABLE pad_table (c1 integer, c2 varchar(15));
Created successfully.

Mach> INSERT INTO pad_table VALUES (1, 'Antonio');
```

```
1 row(s) inserted.

Mach> INSERT INTO pad_table VALUES (25, 'Johnathan');
1 row(s) inserted.

Mach> INSERT INTO pad_table VALUES (30, 'M');
1 row(s) inserted.

Mach> SELECT LPAD(to_char(c1), 5, '0') FROM pad_table;
LPAD(to_char(c1), 5, '0')
-----------------------------
00030
00025
00001
[3] row(s) selected.

Mach> SELECT RPAD(to_char(c1), 5, '0') FROM pad_table;
RPAD(to_char(c1), 5, '0')
-----------------------------
30000
25000
10000
[3] row(s) selected.

Mach> SELECT LPAD(c2, 5) FROM pad_table;
LPAD(c2, 5)
---------------
    M
Johna
Anton
[3] row(s) selected.

Mach> SELECT RPAD(c2, 5) FROM pad_table;
RPAD(c2, 5)
---------------
M
Johna
Anton
[3] row(s) selected.

Mach> SELECT RPAD(c2, 10, '***') FROM pad_table;
RPAD(c2, 10, '***')
-----------------------
M*********
Johnathan*
Antonio***
[3] row(s) selected.
```

LTRIM / RTRIM

이 함수는 첫 번째 매개변수에서 pattern 문자열에 해당하는 값을 제거하는 역할을 수행한다. LTRIM 함수는 컬럼 값의 왼쪽에서 오른쪽으로, RTRIM 함수는 오른쪽에서 왼쪽으로 문자들이 pattern에 있는지 확인하고 pattern에 없는 문자를 만날 때까지 잘라낸다. 만약 모든 문자열이 pattern에 존재한다면 NULL을 리턴한다.

Pattern 표현식을 명시하지 않은 경우 공백 문자 ' '를 기본으로 사용하여 공백 문자를 제거한다.

```
LTRIM(column_name, pattern)
RTRIM(column_name, pattern)
```

```
Mach> CREATE TABLE trim_table1(name VARCHAR(10));
Created successfully.

Mach> INSERT INTO trim_table1 VALUES ('   smith   ');
1 row(s) inserted.

Mach> SELECT ltrim(name) FROM trim_table1;
ltrim(name)
---------------
```

```
smith
[1] row(s) selected.

Mach> SELECT rtrim(name) FROM trim_table1;
rtrim(name)
---------------
   smith
[1] row(s) selected.

Mach> SELECT ltrim(name, ' s') FROM trim_table1;
ltrim(name, ' s')
---------------------
mith
[1] row(s) selected.

Mach> SELECT rtrim(name, 'h ') FROM trim_table1;
rtrim(name, 'h ')
---------------------
   smit
[1] row(s) selected.

Mach> CREATE TABLE trim_table2 (name VARCHAR(10));
Created successfully.

Mach> INSERT INTO trim_table2 VALUES ('ddckaaadkk');
1 row(s) inserted.

Mach> SELECT ltrim(name, 'dc') FROM trim_table2;
ltrim(name, 'dc')
---------------------
kaaadkk
[1] row(s) selected.

Mach> SELECT rtrim(name, 'dk') FROM trim_table2;
rtrim(name, 'dk')
---------------------
ddckaaa
[1] row(s) selected.

Mach> SELECT ltrim(name, 'dckak') FROM trim_table2;
ltrim(name, 'dckak')
------------------------
NULL
[1] row(s) selected.

Mach> SELECT rtrim(name, 'dckak') FROM trim_table2;
rtrim(name, 'dckak')
------------------------
NULL
[1] row(s) selected.
```

MAX

이 함수는 집계 함수로써, 해당 숫자 컬럼의 최대 값을 구하는 함수이다.

```
MAX(column_name)
```

```
Mach> CREATE TABLE max_table (c INTEGER);
Created successfully.

Mach> INSERT INTO max_table VALUES(10);
1 row(s) inserted.

Mach> INSERT INTO max_table VALUES(20);
1 row(s) inserted.
```

```
Mach> INSERT INTO max_table VALUES(30);
1 row(s) inserted.

Mach> SELECT MAX(c) FROM max_table;
MAX(c)
--------------
30
[1] row(s) selected.
```

MIN

이 함수는 집계 함수로써, 해당 숫자 컬럼의 최소값을 구하는 함수이다.

```
MIN(column_name)
```

```
Mach> CREATE TABLE min_table(c1 INTEGER);
Created successfully.

Mach> INSERT INTO min_table VALUES(1);
1 row(s) inserted.

Mach> INSERT INTO min_table VALUES(22);
1 row(s) inserted.

Mach> INSERT INTO min_table VALUES(33);
1 row(s) inserted.

Mach> SELECT MIN(c1) FROM min_table;
MIN(c1)
--------------
1
[1] row(s) selected.
```

NVL

이 함수는 컬럼의 값이 NULL이면 value를 리턴하고, NULL이 아니면 원래 컬럼의 값을 출력한다.

```
NVL(string1, replace_with)
```

```
Mach> CREATE TABLE nvl_table (c1 varchar(10));
Created successfully.

Mach> INSERT INTO nvl_table VALUES ('Johnathan');
1 row(s) inserted.

Mach> INSERT INTO nvl_table VALUES (NULL);
1 row(s) inserted.

Mach> SELECT NVL(c1, 'Thomas') FROM nvl_table;
NVL(c1, 'Thomas')
----------------------
Thomas
Johnathan
```

ROUND

이 함수는 정수형 입력 값에서 (입력된 자릿수+1) 의 자릿수에서 반올림한 결과를 반환한다. 자릿수가 입력되지 않은 경우, 반올림은 0의 자리에서 이뤄진다. 소수점 자리를 반올림하기 위해서 `decimals` 자리에 음수를 입력하는 것이 가능하다.

```
ROUND(column_name, [decimals])
```

```
Mach> CREATE TABLE round_table (c1 DOUBLE);
Created successfully.

Mach> INSERT INTO round_table VALUES (1.994);
1 row(s) inserted.

Mach> INSERT INTO round_table VALUES (1.995);
1 row(s) inserted.

Mach> SELECT c1, ROUND(c1, 2) FROM round_table;
c1                          ROUND(c1, 2)
------------------------------------------------------------
1.995                       2
1.994                       1.99
```

ROWNUM

이 함수는 SELECT 쿼리 결과 Row에 번호를 부여하는 함수이다.

SELECT 쿼리 내부에 사용되는 Subquery 또는 Inline View 내부에서도 사용이 가능하며, Inline View 에서 ROWNUM() 함수를 Target List에 사용하는 경우엔 Alias를 부여해야 외부에서 참조가 가능하다.

```
ROWNUM()
```

사용 가능한 절

해당 함수는 SELECT 쿼리의 Target List, GROUP BY, 또는 ORDER BY 절에서 사용이 가능하다. 하지만 SELECT 쿼리의 WHERE와 HAVING 절에서는 사용할 수 없다. ROWNUM() 결과 번호로 WHERE나 HAVING 절을 통해 제어하고자 한다면, ROWNUM() 을 포함한 SELECT 쿼리를 Inline View로 사용한 다음 외부에 있는 WHERE나 HAVING 절에서 참조하면 된다.

사용할 수 있는 절	사용할 수 없는 절
Target List / GROUP BY / ORDER BY	WHERE / HAVING

```
Mach> CREATE TABLE rownum_table(c1 INTEGER, c2 DOUBLE, c3 VARCHAR(10));
Created successfully.

Mach> INSERT INTO rownum_table VALUES(1, 1.0, '');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(2, 2.0, 'Second Row');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(3, 3.3, 'Third Row');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(4, 4.3, 'Fourth Row');
1 row(s) inserted.

Mach> SELECT INNER_RANK, c3 AS NAME
    2 FROM   (SELECT ROWNUM() AS INNER_RANK, * FROM rownum_table)
    3 WHERE  INNER_RANK < 3;
INNER_RANK           NAME
-----------------------------------
1                    Fourth Row
2                    Third Row
[2] row(s) selected.
```

정렬로 인한 결과 변화

SELECT 쿼리에 ORDER BY 절이 존재하는 경우, Target List에 있는 ROWNUM()의 결과 번호가 순차적으로 부여되지 않는 경우가 발생할 수 있다. 이는 ROWNUM() 연산이 ORDER BY 절의 연산 이전에 이루어지기 때문이다. 순차적으로 부여하고자 할 경우, ORDER BY 절을 포함한 쿼리를 Inline View로 사용한 다음 ROWNUM()을 외부 SELECT 문에서 호출하면 된다.

```
Mach> CREATE TABLE rownum_table(c1 INTEGER, c2 DOUBLE, c3 VARCHAR(10));
Created successfully.

Mach> INSERT INTO rownum_table VALUES(1, 1.0, '');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(2, 2.0, 'John');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(3, 3.3, 'Sarah');
1 row(s) inserted.

Mach> INSERT INTO rownum_table VALUES(4, 4.3, 'Micheal');
1 row(s) inserted.

Mach> SELECT ROWNUM(), c2 AS SORT, c3 AS NAME
    2 FROM   ( SELECT * FROM rownum_table ORDER BY c3 );
ROWNUM()             SORT                        NAME
-------------------------------------------------------------------
1                    1                           NULL
2                    2                           John
3                    4.3                         Micheal
4                    3.3                         Sarah
[4] row(s) selected.
```

SERIESNUM

각 레코드가, SERIES BY 로 그룹지어진 시리즈의 몇 번째에 속해있는지를 나타낸 번호를 반환한다. 반환형은 BIGINT 형이며, SERIES BY 절이 사용되지 않았을 경우엔 항상 1을 반환한다.

```
SERIESNUM()
```

```
Mach> CREATE TABLE T1 (C1 INTEGER, C2 INTEGER);
Created successfully.

Mach> INSERT INTO T1 VALUES (0, 1);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (1, 2);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (2, 3);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (3, 2);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (4, 1);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (5, 2);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (6, 3);
1 row(s) inserted.

Mach> INSERT INTO T1 VALUES (7, 1);
1 row(s) inserted.


Mach> SELECT SERIESNUM(), C1, C2 FROM T1 ORDER BY C1 SERIES BY C2 > 1;
SERIESNUM() C1 C2
-----------------------------------------------
1 1 2
1 2 3
```

```
1 3 2
2 5 2
2 6 3
[5] row(s) selected.
```

STDDEV / STDDEV_POP

이 함수는 집계 함수로써, 입력된 컬럼의 (샘플) 표준 편차와 모집단 표준 편차를 반환한다. 각각 VARIANCE 와 VAR_POP 값의 제곱근과 같다.

```
STDDEV(column)
STDDEV_POP(column)
```

```
Mach> CREATE TABLE stddev_table(c1 INTEGER, C2 DOUBLE);

Mach> INSERT INTO stddev_table VALUES (1, 1);
1 row(s) inserted.

Mach> INSERT INTO stddev_table VALUES (2, 1);
1 row(s) inserted.

Mach> INSERT INTO stddev_table VALUES (3, 2);
1 row(s) inserted.

Mach> INSERT INTO stddev_table VALUES (4, 2);
1 row(s) inserted.

Mach> SELECT c2, STDDEV(c1) FROM stddev_table GROUP BY c2;
c2                          STDDEV(c1)
------------------------------------------------------------
1                           0.707107
2                           0.707107
[2] row(s) selected.

Mach> SELECT c2, STDDEV_POP(c1) FROM stddev_table GROUP BY c2;
c2                          STDDEV_POP(c1)
------------------------------------------------------------
1                           0.5
2                           0.5
[2] row(s) selected.
```

SUBSTR

이 함수는 가변 문자열 컬럼의 데이터를 START 부터 SIZE 만큼 잘라낸다.

- START 는 1부터 시작하며, 0일 경우에는 NULL을 리턴한다.
- SIZE가 만일 해당 문자열의 크기보다 클 경우에는 그 문자열의 최대값까지만 되돌린다.

SIZE는 생략 가능하며, 생략할 경우에는 해당 문자열 크기만큼 내부적으로 지정된다.

```
SUBSTRING(column_name, start, [length])
```

```
Mach> CREATE TABLE substr_table (c1 VARCHAR(10));
Created successfully.

Mach> INSERT INTO substr_table values('ABCDEFG');
1 row(s) inserted.

Mach> INSERT INTO substr_table values('abstract');
1 row(s) inserted.

Mach> SELECT SUBSTR(c1, 1, 1) FROM substr_table;
SUBSTR(c1, 1, 1)
--------------------
```

```
a
A
[2] row(s) selected.

Mach> SELECT SUBSTR(c1, 3, 3) FROM substr_table;
SUBSTR(c1, 3, 3)
--------------------
str
CDE
[2] row(s) selected.

Mach> SELECT SUBSTR(c1, 2) FROM substr_table;
SUBSTR(c1, 2)
-----------------
bstract
BCDEFG
[2] row(s) selected.

Mach> drop table substr_table;
Dropped successfully.

Mach> CREATE TABLE substr_table (c1 VARCHAR(10));
Created successfully.

Mach> INSERT INTO substr_table values('ABCDEFG');
1 row(s) inserted.

Mach> SELECT SUBSTR(c1, 1, 1) FROM substr_table;
SUBSTR(c1, 1, 1)
--------------------
A
[1] row(s) selected.

Mach> SELECT SUBSTR(c1, 3, 3) FROM substr_table;
SUBSTR(c1, 3, 3)
--------------------
CDE
[1] row(s) selected.

Mach> SELECT SUBSTR(c1, 2) FROM substr_table;
SUBSTR(c1, 2)
-----------------
BCDEFG
[1] row(s) selected.
```

SUBSTRING_INDEX

주어진 구분자(delim)가 입력한 count만큼 발견될 때까지 복제한 문자열을 반환한다. 만약 count를 음수값으로 입력하면 입력한 문자열의 끝에서부터 구분자를 검사해서 구분자가 발견된 위치에서 문자열의 끝까지 반환한다.

만약 count를 0으로 입력하거나 문자열에 구분자가 존재하지 않는다면 함수는 NULL을 리턴할 것이다.

```
SUBSTRING_INDEX(expression, delim, count)
```

```
Mach> CREATE TABLE substring_table (url VARCHAR(30));
Created successfully.

Mach> INSERT INTO substring_table VALUES('www.machbase.com');
1 row(s) inserted.

Mach> SELECT SUBSTRING_INDEX(url, '.', 1) FROM substring_table;
SUBSTRING_INDEX(url, '.', 1)
---------------------------------
www
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(url, '.', 2) FROM substring_table;
```

```
SUBSTRING_INDEX(url, '.', 2)
---------------------------------
www.machbase
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(url, '.', -1) FROM substring_table;
SUBSTRING_INDEX(url, '.', -1)
---------------------------------
com
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(SUBSTRING_INDEX(url, '.', 2), '.', -1) FROM substring_table;
SUBSTRING_INDEX(SUBSTRING_INDEX(url, '.', 2), '.', -1)
-----------------------------------------
machbase
[1] row(s) selected.

Mach> SELECT SUBSTRING_INDEX(url, '.', 0) FROM substring_table;
SUBSTRING_INDEX(url, '.', 0)
---------------------------------
NULL
[1] row(s) selected.
```

SUM

이 함수는 집계 함수로써, 숫자형 컬럼의 총합을 나타낸다.

```
SUM(column_name)
```

```
Mach> CREATE TABLE sum_table (c1 INTEGER, c2 INTEGER);
Created successfully.

Mach> INSERT INTO sum_table VALUES(1, 1);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(1, 2);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(1, 3);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(2, 1);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(2, 2);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(2, 3);
1 row(s) inserted.

Mach> INSERT INTO sum_table VALUES(3, 4);
1 row(s) inserted.

Mach> SELECT c1, SUM(c1) from sum_table group by c1;
c1          SUM(c1)
-----------------------------------
2           6
3           3
1           3
[3] row(s) selected.

Mach> SELECT c1, SUM(c2) from sum_table group by c1;
c1          SUM(c2)
-----------------------------------
2           6
3           4
```

```
1           6
[3] row(s) selected.
```

SUMSQ

SUMSQ 함수는 숫자형 컬럼값에 대한 제곱 합을 반환한다.

```
SUMSQ(value)
```

```
Mach> CREATE TABLE sumsq_table (c1 INTEGER, c2 INTEGER);
Created successfully.

Mach> INSERT INTO sumsq_table VALUES (1, 1);
1 row(s) inserted.

Mach> INSERT INTO sumsq_table VALUES (1, 2);
1 row(s) inserted.

Mach> INSERT INTO sumsq_table VALUES (1, 3);
1 row(s) inserted.

Mach> INSERT INTO sumsq_table VALUES (2, 4);
1 row(s) inserted.

Mach> INSERT INTO sumsq_table VALUES (2, 5);
1 row(s) inserted.

Mach> SELECT c1, SUMSQ(c2) FROM sumsq_table GROUP BY c1;
c1          SUMSQ(c2)
-------------------------------------
2           41
1           14
[2] row(s) selected.
```

SYSDATE / NOW

SYSDATE 는 함수가 아니라 의사컬럼 (pseudocolumn) 으로, 시스템의 현재 시각을 반환한다.

NOW 는 SYSDATE 와 동일한 함수이며, 사용자 편의를 위해 같이 제공한다.

```
SYSDATE
NOW
```

```
Mach> SELECT SYSDATE, NOW FROM t1;

SYSDATE                         NOW
-------------------------------------------------------------------
2017-01-16 14:14:53 310:973:000 2017-01-16 14:14:53 310:973:000
```

TO_CHAR

이 함수는 주어진 데이터 타입을 문자열 타입으로 변환하는 함수이다. 타입에 따라 format_string을 지정할 수도 있으며, Binary 타입에 대해서는 지원하지 않는다.

```
TO_CHAR(column)
```

TO_CHAR : 기본 자료형

아래와 같이 기본 자료형에서는 그대로 문자열 형식의 데이터로 변환된다.

```
Mach> CREATE TABLE fixed_table (id1 SHORT, id2 INTEGER, id3 LONG, id4 FLOAT, id5 DOUBLE, id6 IPV4, id7 IPV6, id8 VA
Created successfully.

Mach> INSERT INTO fixed_table values(200, 19234, 1234123412, 3.14, 7.8338, '192.168.0.1', '::127.0.0.1', 'log varch
1 row(s) inserted.

Mach> SELECT '[ ' || TO_CHAR(id1) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id1) || ' ]'
--------------------------------------------------------------------------------
[ 200 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id2) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id2) || ' ]'
--------------------------------------------------------------------------------
[ 19234 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id3) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id3) || ' ]'
--------------------------------------------------------------------------------
[ 1234123412 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id4) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id4) || ' ]'
--------------------------------------------------------------------------------
[ 3.140000 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id5) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id5) || ' ]'
--------------------------------------------------------------------------------
[ 7.833800 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id6) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id6) || ' ]'
--------------------------------------------------------------------------------
[ 192.168.0.1 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id7) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id7) || ' ]'
--------------------------------------------------------------------------------
[ 0000:0000:0000:0000:0000:0000:7F00:0001 ]
[1] row(s) selected.

Mach> SELECT '[ ' || TO_CHAR(id8) || ' ]' FROM fixed_table;
'[ ' || TO_CHAR(id8) || ' ]'
--------------------------------------------------------------------------------
[ log varchar ]
[1] row(s) selected.
```

TO_CHAR : 부동소수형

ⓘ 5.5.6 부터 지원됩니다.

float, double 컬럼의 값을 임의의 문자열로 변환하는 함수이다.

포맷 표현은 중복해서 사용할 수 없고, '[문자][숫자]' 의 형태로 입력해야 한다.

포맷 표현식	설명
F / f	컬럼 값의 소수점 자릿수를 지정한다. 입력 최대 숫자값은 30이다.
N / n	컬럼 값의 소수점 자릿수를 지정하고, 정수 부분은 세자리마다 쉼표 (,) 를 입력한다. 입력 최대 숫자값은 30이다.

```
Mach> create table float_table (i1 float, i2 double);
Created successfully.

Mach> insert into float_table values (1.23456789, 1234.5678901234567890);
1 row(s) inserted.

Mach> select TO_CHAR(i1, 'f8'), TO_CHAR(i2, 'N9') from float_table;
TO_CHAR(i1, 'f8')       TO_CHAR(i2, 'N9')
----------------------------------------------------------------
1.23456788              1,234.567890123
[1] row(s) selected.
```

TO_CHAR : DATETIME 형

datetime 컬럼의 값을 임의의 문자열로 변환하는 함수이다. 이 함수를 이용하여 다양한 형태의 문자열을 만들고 조합할 수 있다.

format_string이 생략된 경우에는 기본값으로 "YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn" 을 사용해서 변환한다.

포맷 표현식	설명
YYYY	연도를 4자리 숫자로 변환한다.
YY	연도를 2자리 숫자로 변환한다.
MM	해당 월을 2자리 숫자로 변환한다.
MON	해당 월을 3자리 축약 알파벳으로 변환한다. (e.g. JAN, FEB, MAY, ...)
DD	해당 일을 2자리 숫자로 변환한다.
DAY	해당 요일을 3자리 축약 알파벳으로 변환한다. (e.g. SUN, MON, ...)
IW	ISO 8601 규칙에 의해 (요일을 고려한), 1부터 53까지의 특정 연도의 주 수 (Week Number) 를 변환한다. • 한 주의 시작은 월요일이다. • 첫 주는 이전 연도의 마지막 주로 간주할 수 있다. 마찬가지로, 마지막 주는 다음 연도의 첫 주로 간주할 수 있다. 자세한 내용은 ISO 8601 규칙을 참고한다.
WW	요일을 고려하지 않은, 1부터 53까지의 특정 연도의 주 수 (Week Number) 를 변환한다. 즉, 1월 1일부터 1월 7일까지는 1로 변환된다.
W	요일을 고려하지 않은, 1부터 5까지의 특정 달의 주 수 (Week Number) 를 변환한다. 즉, 3월 1일부터 3월 7일까지는 1로 변환된다.
HH	해당 시간을 2자리 숫자로 변환한다.
HH12	해당 시간을 2자리 숫자로 변환하되, 1~12까지 표현한다.
HH24	해당 시간을 2자리 숫자로 변환하되, 1~23까지 표현한다.
HH2, HH3, HH6	해당 시간을 HH 다음에 오는 숫자로 절삭한다. 즉, HH6을 사용한 경우 0시부터 5시까지는 0으로 6~11시까지는 6으로 표현한다. 이 표현은 시계열 상의 특정 시간 단위 통계를 계산할 때 유용하다. 이 값은 24시 기준으로 표현된다.
MI	해당 분을 두자리 숫자로 표현한다.
MI2, MI5, MI10, MI20, MI30	해당 분을 MI 다음에 오는 숫자로 절삭한다. 즉, MI30을 사용한 경우에는 0분에서 29분까지는 0으로 표현되고, 30~59분까지는 30으로 표현된다. 이 표현은 시계열 상의 특정 시간 단위 통계를 계산할 때 유용하다.
SS	해당 초를 두자리 숫자로 표현한다.
SS2, SS5, SS10, SS20, SS30	해당 초를 이어지는 숫자로 절삭한다. 즉, SS30을 사용한 경우에는 0초에서 29초까지는 0으로 표현되고, 30~59초까지는 30으로 표현된다. 이 표현은 시계열 상의 특정 시간 단위 통계를 계산할 때 유용하다.
AM	현재 시간이 오전, 오후에 따라 각각 AM 혹은 PM으로 표현한다.
mmm	해당 시간의 mili second를 3자리 숫자로 표현한다. 값의 범위는 0~999이다.
uuu	해당 시간의 micro second를 3자리 숫자로 표현한다.

포맷 표현식	설명
nnn	값의 범위는 0~999이다. 해당 시간의 nano second를 3자리 숫자로 표현한다. 값의 범위는 0~999이다.

```
Mach> CREATE TABLE datetime_table (id integer, dt datetime);
Created successfully.

Mach> INSERT INTO  datetime_table values(1, TO_DATE('1999-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  datetime_table values(2, TO_DATE('2012-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  datetime_table values(3, TO_DATE('2013-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  datetime_table values(4, TO_DATE('2014-12-30 11:22:33 444:555:666'));
1 row(s) inserted.

Mach> SELECT id, dt FROM datetime_table WHERE dt > TO_DATE('2000-11-11 1:2:3 4:5:0');
id          dt
-----------------------------------------------
4           2014-12-30 11:22:33 444:555:666
3           2013-11-11 01:02:03 004:005:006
2           2012-11-11 01:02:03 004:005:006
[3] row(s) selected.

Mach> SELECT id, dt FROM datetime_table WHERE dt > TO_DATE('2013-11-11 1:2:3') and dt < TO_DATE('2014-11-11 1:2:3')
id          dt
-----------------------------------------------
3           2013-11-11 01:02:03 004:005:006
[1] row(s) selected.

Mach> SELECT id, TO_CHAR(dt) FROM datetime_table;
id          TO_CHAR(dt)
-------------------------------------------------------------------------------------------------
4           2014-12-30 11:22:33 444:555:666
3           2013-11-11 01:02:03 004:005:006
2           2012-11-11 01:02:03 004:005:006
1           1999-11-11 01:02:03 004:005:006
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY')
-------------------------------------------------------------------------------------------------
4           2014
3           2013
2           2012
1           1999
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM')
-------------------------------------------------------------------------------------------------
4           2014-12
3           2013-11
2           2012-11
1           1999-11
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD')
-------------------------------------------------------------------------------------------------
4           2014-12-30
3           2013-11-11
2           2012-11-11
1           1999-11-11
[4] row(s) selected.
```

```
Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD TO_CHAR') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD TO_CHAR')
-------------------------------------------------------------------------------------------
4           2014-12-30 TO_CHAR
3           2013-11-11 TO_CHAR
2           2012-11-11 TO_CHAR
1           1999-11-11 TO_CHAR
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS')
-------------------------------------------------------------------------------------------
4           2014-12-30 11:22:33
3           2013-11-11 01:02:03
2           2012-11-11 01:02:03
1           1999-11-11 01:02:03
[4] row(s) selected.

Mach> SELECT id, TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS mmm.uuu.nnn') FROM datetime_table;
id          TO_CHAR(dt, 'YYYY-MM-DD HH24:MI:SS mmm.
-------------------------------------------------------------------------------------------
4           2014-12-30 11:22:33 444.555.666
3           2013-11-11 01:02:03 004.005.006
2           2012-11-11 01:02:03 004.005.006
1           1999-11-11 01:02:03 004.005.006
[4] row(s) selected.
```

TO_CHAR : 지원하지 않는 타입

현재 Binary 타입에 대해서는 TO_CHAR를 지원하지 않는다.

왜냐하면, 일반 텍스트로 변환이 불가능하기 때문이다. 만일 이를 화면에 출력하고자 할 경우에는 TO_HEX() 함수를 통해 16진수 값을 출력해서 확인할 수 있다.

TO_DATE

이 함수는 주어진 포맷 문자열로 표현된 문자열을 datetime 타입으로 변환한다.

format_string이 생략된 경우에는 기본값으로 "YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn" 을 사용해서 변환한다.

```
-- default format is "YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn" if no format exists.
TO_DATE(date_string [, format_string])
```

```
Mach> CREATE TABLE to_date_table (id INTEGER, dt datetime);
Created successfully.

Mach> INSERT INTO  to_date_table VALUES(1, TO_DATE('1999-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  to_date_table VALUES(2, TO_DATE('2012-11-11 1:2:3 4:5:6'));
1 row(s) inserted.

Mach> INSERT INTO  to_date_table VALUES(3, TO_DATE('2014-12-30 11:22:33 444:555:666'));
1 row(s) inserted.

Mach> INSERT INTO  to_date_table VALUES(4, TO_DATE('2014-12-30 23:22:34 777:888:999', 'YYYY-MM-DD HH24:MI:SS mmm:uu
1 row(s) inserted.

Mach> SELECT id, dt FROM to_date_table WHERE dt > TO_DATE('1999-11-11 1:2:3 4:5:0');
id          dt
-----------------------------------------------
4           2014-12-30 23:22:34 777:888:999
3           2014-12-30 11:22:33 444:555:666
2           2012-11-11 01:02:03 004:005:006
1           1999-11-11 01:02:03 004:005:006
[4] row(s) selected.
```

```
Mach> SELECT id, dt FROM to_date_table WHERE dt > TO_DATE('2000-11-11 1:2:3 4:5:0');
id          dt
----------------------------------------------
4           2014-12-30 23:22:34 777:888:999
3           2014-12-30 11:22:33 444:555:666
2           2012-11-11 01:02:03 004:005:006
[3] row(s) selected.

Mach> SELECT id, dt FROM to_date_table WHERE dt > TO_DATE('2012-11-11 1:2:3','YYYY-MM-DD HH24:MI:SS') and dt < TO_D
id          dt
----------------------------------------------
2           2012-11-11 01:02:03 004:005:006
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999', 'YYYY') FROM to_date_table LIMIT 1;
id          TO_DATE('1999', 'YYYY')
----------------------------------------------
4           1999-01-01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12', 'YYYY-MM') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12', 'YYYY-MM')
----------------------------------------------
4           1999.12.01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999', 'YYYY') FROM to_date_table LIMIT 1;
id          TO_DATE('1999', 'YYYY')
----------------------------------------------
4           1999-01-01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12', 'YYYY-MM') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12', 'YYYY-MM')
----------------------------------------------
4           1999-12-01 00:00:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12', 'YYYY-MM-DD HH24:MI') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12-31 13:12', 'YYYY-MM-DD HH24:MI')
----------------------------------------------------------
4           1999-12-31 13:12:00 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32', 'YYYY-MM-DD HH24:MI:SS') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12-31 13:12:32', 'YYYY-MM-DD HH24:MI:SS')
----------------------------------------------------------
4           1999-12-31 13:12:32 000:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32 123', 'YYYY-MM-DD HH24:MI:SS mmm') FROM to_date_table LIMIT 1;
id          TO_DATE('1999-12-31 13:12:32 123', 'YYYY-MM-DD HH24:MI:SS mmm')
----------------------------------------------------------
4           1999-12-31 13:12:32 123:000:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32 123:456', 'YYYY-MM-DD HH24:MI:SS mmm:uuu') FROM to_date_table LIMIT 1
id          TO_DATE('1999-12-31 13:12:32 123:456', 'YYYY-MM-DD HH24:MI:SS mmm:uuu')
----------------------------------------------------------
4           1999-12-31 13:12:32 123:456:000
[1] row(s) selected.

Mach> SELECT id, TO_DATE('1999-12-31 13:12:32 123:456:789', 'YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn') FROM to_date_table
id           TO_DATE('1999-12-31 13:12:32 123:456:789', 'YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn')
----------------------------------------------------------
4           1999-12-31 13:12:32 123:456:789
[1] row(s) selected.
```

TO_DATE_SAFE

TO_DATE() 와 비슷하지만, 변환에 실패할 경우 에러를 내지 않고 NULL 을 반환한다.

```
TO_DATE_SAFE(date_string [, format_string])
```

```
Mach> CREATE TABLE date_table (ts DATETIME);
Created successfully.

Mach> INSERT INTO date_table VALUES (TO_DATE_SAFE('2016-01-01', 'YYYY-MM-DD'));
1 row(s) inserted.
Mach> INSERT INTO date_table VALUES (TO_DATE_SAFE('2016-01-02', 'YYYY'));
1 row(s) inserted.
Mach> INSERT INTO date_table VALUES (TO_DATE_SAFE('2016-12-32', 'YYYY-MM-DD'));
1 row(s) inserted.

Mach> SELECT ts FROM date_table;
ts
----------------------------------
NULL
NULL
2016-01-01 00:00:00 000:000:000
[3] row(s) selected.
```

TO_HEX

이 함수는 컬럼의 값이 NULL이면 value를 리턴하고, NULL이 아니면 원래 컬럼의 값을 출력한다. 출력의 일관성을 보장하기 위해 short, int, long 타입의 경우에는 BIG ENDIAN 형태로 변환해서 출력해 준다.

```
TO_HEX(column)
```

```
Mach> CREATE TABLE hex_table (id1 SHORT, id2 INTEGER, id3 VARCHAR(10), id4 FLOAT, id5 DOUBLE, id6 LONG, id7 IPV4, i
id11 DATETIME);
Created successfully.

Mach> INSERT INTO hex_table VALUES(256, 65535, '0123456789', 3.141592, 1024 * 1024 * 1024 * 3.14, 13513135446, '192
'binary', TO_DATE('1999', 'YYYY'));
1 row(s) inserted.

Mach> SELECT TO_HEX(id1), TO_HEX(id2), TO_HEX(id3), TO_HEX(id4), TO_HEX(id5), TO_HEX(id6), TO_HEX(id7), TO_HEX(id8)
FROM hex_table;
TO_HEX(id1)  TO_HEX(id2)  TO_HEX(id3)            TO_HEX(id4)  TO_HEX(id5)        TO_HEX(id6)        TO_HEX(id7)
-------------------------------------------------------------------------------------------------------------------
TO_HEX(id8)                         TO_HEX(id9)
-------------------------------------------------------------------------------------------------------------------
TO_HEX(id10)                                                                        TO_HEX(id11)
-----------------------------------------------------------------------------------------------
0100   0000FFFF   30313233343536373839   D80F4940   1F85EB51B81EE941   0000000325721556   04C0A80001
060000000000000000000000000C0A80001   74657874657874
62696E617279                                                                        0CB325846E226000
[1] row(s) selected.
```

TO_IPV4 / TO_IPV4_SAFE

이 함수는 주어진 문자열을 IP 버전4 타입으로 변환한다. 만일 해당 문자열이 숫자형으로 변환이 불가능할 경우에는 TO_IPV4() 함수는 에러를 리턴하고, 동작을 종료한다.

그러나, TO_IPV4_SAFE() 함수의 경우에는 에러 상황일 경우 NULL을 리턴하고, 동작을 계속할 수 있게 되어 있다.

```
TO_IPV4(string_value)
TO_IPV4_SAFE(string_value)
```

```
Mach> CREATE TABLE ipv4_table (c1 varchar(100));
```

```
Created successfully.

Mach> INSERT INTO ipv4_table VALUES('192.168.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipv4_table VALUES('     192.168.0.2    ');
1 row(s) inserted.

Mach> INSERT INTO ipv4_table VALUES(NULL);
1 row(s) inserted.

Mach> SELECT c1 FROM ipv4_table;
c1
--------------------------------------------------------------------------------
NULL
     192.168.0.2
192.168.0.1
[3] row(s) selected.

Mach> SELECT TO_IPV4(c1) FROM ipv4_table;
TO_IPV4(c1)
------------------
NULL
192.168.0.2
192.168.0.1
[3] row(s) selected.

Mach> INSERT INTO ipv4_table VALUES('192.168.0.1.1');
1 row(s) inserted.

Mach> SELECT TO_IPV4(c1) FROM ipv4_table limit 1;
TO_IPV4(c1)
------------------
[ERR-02068 : Invalid IPv4 address format (192.168.0.1.1).]
[0] row(s) selected.

Mach> SELECT TO_IPV4_SAFE(c1) FROM ipv4_table;
TO_IPV4_SAFE(c1)
-------------------
NULL
NULL
192.168.0.2
192.168.0.1
[4] row(s) selected.
```

TO_IPV6 / TO_IPV6_SAFE

이 함수는 주어진 문자열을 IP 버전6 타입으로 변환한다. 만일 해당 문자열이 숫자형으로 변환이 불가능할 경우에는 TO_IPV6() 함수는 에러를 리턴하고, 동작을 종료한다.

그러나, TO_IPV6_SAFE() 함수의 경우에는 에러 상황일 경우 NULL을 리턴하고, 동작을 계속할 수 있게 되어 있다.

```
TO_IPV6(string_value)
TO_IPV6_SAFE(string_value)
```

```
Mach> CREATE TABLE ipv6_table (id varchar(100));
Created successfully.

Mach> INSERT INTO ipv6_table VALUES('::0.0.0.0');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('::127.0.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('::127.0' || '.0.2');
1 row(s) inserted.
```

```
Mach> INSERT INTO ipv6_table VALUES('   ::127.0.0.3');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('::127.0.0.4  ');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('   ::FFFF:255.255.255.255   ');
1 row(s) inserted.

Mach> INSERT INTO ipv6_table VALUES('21DA:D3:0:2F3B:2AA:FF:FE28:9C5A');
1 row(s) inserted.

Mach> SELECT TO_IPV6(id) FROM ipv6_table;
TO_IPV6(id)
----------------------------------------------------------------
21da:d3::2f3b:2aa:ff:fe28:9c5a
::ffff:255.255.255.255
::127.0.0.4
::127.0.0.3
::127.0.0.2
::127.0.0.1
::
[7] row(s) selected.

Mach> INSERT INTO ipv6_table VALUES('127.0.0.10.10');
1 row(s) inserted.

Mach> SELECT TO_IPV6(id) FROM ipv6_table limit 1;
TO_IPV6(id)
----------------------------------------------------------------
[ERR-02148 : Invalid IPv6 address format.(127.0.0.10.10)]
[0] row(s) selected.

Mach> SELECT TO_IPV6_SAFE(id) FROM ipv6_table;
TO_IPV6_SAFE(id)
----------------------------------------------------------------
NULL
21da:d3::2f3b:2aa:ff:fe28:9c5a
::ffff:255.255.255.255
::127.0.0.4
::127.0.0.3
::127.0.0.2
::127.0.0.1
::
[8] row(s) selected.
```

TO_NUMBER / TO_NUMBER_SAFE

이 함수는 주어진 문자열을 숫자형 double 타입으로 변환한다. 만일 해당 문자열이 숫자형으로 변환이 불가능할 경우에는 TO_NUMBER() 함수는 에러를 리턴하고, 동작을 종료한다.

그러나, TO_NUMBER_SAFE() 함수의 경우에는 에러 상황일 경우 NULL을 리턴하고, 동작을 계속할 수 있게 되어 있다.

```
TO_NUMBER(string_value)
TO_NUMBER_SAFE(string_value)
```

```
Mach> CREATE TABLE number_table (id varchar(100));
Created successfully.

Mach> INSERT INTO number_table VALUES('10');
1 row(s) inserted.

Mach> INSERT INTO number_table VALUES('20');
1 row(s) inserted.

Mach> INSERT INTO number_table VALUES('30');
1 row(s) inserted.
```

```
Mach> SELECT TO_NUMBER(id) from number_table;
TO_NUMBER(id)
------------------------------
30
20
10
[3] row(s) selected.

Mach> CREATE TABLE safe_table (id varchar(100));
Created successfully.

Mach> INSERT INTO safe_table VALUES('invalidnumber');
1 row(s) inserted.

Mach> SELECT TO_NUMBER(id) from safe_table;
TO_NUMBER(id)
------------------------------
[ERR-02145 : The string cannot be converted to number value.(invalidnumber)]
[0] row(s) selected.

Mach> SELECT TO_NUMBER_SAFE(id) from safe_table;
TO_NUMBER_SAFE(id)
------------------------------
NULL
[1] row(s) selected.
```

TO_TIMESTAMP

이 함수는 datetime 자료형을 1970-01-01 09:00 부터 경과된 nanosecond 데이터로 변환한다.

```
TO_TIMESTAMP(datetime_value)
```

```
Mach> create table datetime_tbl (c1 datetime);
Created successfully.

Mach> insert into datetime_tbl values ('2010-01-01 10:10:10');
1 row(s) inserted.

Mach> select to_timestamp(c1) from datetime_tbl;
to_timestamp(c1)
-----------------------
1262308210000000000
[1] row(s) selected.
```

TRUNC

TRUNC 함수는 소수점 아래 n번째 자리에서 버림한 number를 반환한다.

n이 생략될 경우 이를 0으로 취급하여 소수점 아래 자리는 모두 삭제한다. n이 음수일 경우 소수점 앞 n자리에서 버림한 값을 반환한다.

```
TRUNC(number [, n])
```

```
Mach> CREATE TABLE trunc_table (i1 DOUBLE);
Created successfully.

Mach> INSERT INTO trunc_table VALUES (158.799);
1 row(s) inserted.

Mach> SELECT TRUNC(i1, 1), TRUNC(i1, -1) FROM trunc_table;
```

```
TRUNC(i1, 1)                TRUNC(i1, -1)
------------------------------------------------------------
158.7                       150
[1] row(s) selected.

Mach> SELECT TRUNC(i1, 2), TRUNC(i1, -2) FROM trunc_table;
TRUNC(i1, 2)                TRUNC(i1, -2)
------------------------------------------------------------
158.79                      100
[1] row(s) selected.
```

TS_CHANGE_COUNT

이 함수는 집계 함수로써, 특정 컬럼 값의 변경 횟수를 얻는다.

입력되는 데이터의 순서가 시간 순으로 입력된다는 것을 보장할 수 있어야 원하는 결과를 얻을 수가 있기 때문에 이 함수는 1) Join을 사용하거나 2) Inline view 를 사용한 경우에는 사용할 수가 없다.

현재 버전에서는 varchar를 제외한 타입만 지원한다.

> Cluster Edition 에서는 사용할 수 없는 함수이다.

```
TS_CHANGE_COUNT(column)
```

```
Mach> CREATE TABLE ipcount_table (id INTEGER, ip IPV4);
Created successfully.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.2');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.1');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (1, '192.168.0.2');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.3');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.3');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.4');
1 row(s) inserted.

Mach> INSERT INTO ipcount_table VALUES (2, '192.168.0.4');
1 row(s) inserted.

Mach> SELECT id, TS_CHANGE_COUNT(ip) from ipcount_table GROUP BY id;
id          TS_CHANGE_COUNT(ip)
-----------------------------------
2           2
1           4
[2] row(s) selected.
```

UNIX_TIMESTAMP

UNIX_TIMESTAMP는 date타입의 값을 unix의 time() 시스템 호출이 변환하는 32비트 정수 값으로 변환하는 함수이다. (FROM_UNIXTIME은 반대로 정수형 데이터를 date 타입 값으로 변환하는 함수이다.)

```
UNIX_TIMESTAMP(datetime_value)
```

```
Mach> CREATE table unix_table (c1 int);
Created successfully.

Mach> INSERT INTO unix_table VALUES (UNIX_TIMESTAMP('2001-01-01'));
1 row(s) inserted.

Mach> SELECT * FROM unix_table;
C1
--------------
978274800
[1] row(s) selected.
```

UPPER

이 함수는 주어진 영문 컬럼의 내용을 대문자로 변환한다.

```
UPPER(string_value)
```

```
Mach> CREATE TABLE upper_table(id INTEGER,name VARCHAR(10));
Created successfully.

Mach> INSERT INTO upper_table VALUES(1, '');
1 row(s) inserted.

Mach> INSERT INTO upper_table VALUES(2, 'James');
1 row(s) inserted.

Mach> INSERT INTO upper_table VALUES(3, 'sarah');
1 row(s) inserted.

Mach> INSERT INTO upper_table VALUES(4, 'THOMAS');
1 row(s) inserted.

Mach> SELECT id, UPPER(name) FROM upper_table;
id          UPPER(name)
----------------------------
4           THOMAS
3           SARAH
2           JAMES
1           NULL
[4] row(s) selected.
```

VARIANCE / VAR_POP

이 함수는 집계 함수로써, 주어진 숫자형 컬럼 값들의 분산값을 리턴한다. Variance함수는 샘플에 대한 분산을, VAR_POP함수는 모집단에 대한 분산값을 리턴한다.

```
VARIANCE(column_name)
VAR_POP(column_name)
```

```
Mach> CREATE TABLE var_table(c1 INTEGER, c2 DOUBLE);
Created successfully.

Mach> INSERT INTO var_table VALUES (1, 1);
1 row(s) inserted.

Mach> INSERT INTO var_table VALUES (2, 1);
1 row(s) inserted.
```

```
Mach> INSERT INTO var_table VALUES (1, 2);
1 row(s) inserted.

Mach> INSERT INTO var_table VALUES (2, 2);
1 row(s) inserted.

Mach> SELECT VARIANCE(c1) FROM var_table;
VARIANCE(c1)
------------------------------
0.333333
[1] row(s) selected.

Mach> SELECT VAR_POP(c1) FROM var_table;
VAR_POP(c1)
------------------------------
0.25
[1] row(s) selected.
```

YEAR / MONTH / DAY

이 함수들은 입력된 datetime 컬럼값에서 해당하는 연, 월, 일을 추출하여 정수형 값으로 반환하는 함수이다.

```
YEAR(datetime_col)
MONTH(datetime_col)
DAY(datetime_col)
```

```
Mach> CREATE TABLE extract_table(c1 DATETIME, c2 INTEGER);
Created successfully.

Mach> INSERT INTO extract_table VALUES (to_date('2001-01-01 12:30:00 000:000:000'), 1);
1 row(s) inserted.

Mach> SELECT YEAR(c1), MONTH(c1), DAY(c1) FROM extract_table;
year(c1)    month(c1)   day(c1)
---------------------------------------
2001        1           1
```

ISNAN / ISINF

이 함수는 인자로 받은 숫자형 값이 NaN / Inf 값인지를 판단한다. NaN / Inf 값인 경우 1 그렇지 않은 경우 0 을 반환한다.

syntax

```
ISNAN(number)
ISINF(number)
```

example

```
Mach> SELECT * FROM test;
I1                          I2                          I3
------------------------------------------------------------------------------
1                           1                           1
nan                         inf                         0
NULL                        NULL                        NULL
[3] row(s) selected.


Mach> SELECT ISNAN(i1), ISNAN(i2), ISNAN(i3), i3 FROM test ;
ISNAN(i1)   ISNAN(i2)   ISNAN(i3)   i3
-----------------------------------------------------
0           0           0           1
1           0           0           0
```

```
NULL        NULL        NULL        NULL
[3] row(s) selected.

Mach> SELECT * FROM test WHERE ISNAN(i1) = 1;
I1                          I2                          I3
------------------------------------------------------------------------
nan                         inf                         0
[1] row(s) selected.
```

내장 함수 지원 타입

	Short	Integer	Long	Float	Double	Varchar	Text	Ipv4	Ipv6	D
ABS	o	o	o	o	o	x	x	x	x	x
ADD_TIME	x	x	x	x	x	x	x	x	x	o
AVG	o	o	o	o	o	x	x	x	x	x
BITAND / BITOR	o	o	o	x	x	x	x	x	x	x
COUNT	o	o	o	o	o	o	x	o	o	o
DATE_TRUNC	x	x	x	x	x	x	x	x	x	o
DECODE	o	o	o	o	o	o	x	o	x	o
FIRST / LAST	o	o	o	o	o	o	x	o	o	o
FROM_UNIXTIME	o	o	o	o	o	x	x	x	x	x
FROM_TIMESTAMP	o	o	o	o	o	x	x	x	x	x
GROUP_CONCAT	o	o	o	o	o	o	x	o	o	o
INSTR	x	x	x	x	x	o	o	x	x	x
LEAST / GREATEST	o	o	o	o	o	o	x	x	x	x
LENGTH	x	x	x	x	x	o	o	x	x	x
LOWER	x	x	x	x	x	o	x	x	x	x
LPAD / RPAD	x	x	x	x	x	o	x	x	x	x
LTRIM / RTRIM	x	x	x	x	x	o	x	x	x	x
MAX	o	o	o	o	o	o	x	o	o	o
MIX	o	o	o	o	o	o	x	o	o	o
NVL	x	x	x	x	x	o	x	o	x	x
ROUND	o	o	o	o	o	x	x	x	x	x
ROWNUM	o	o	o	o	o	o	o	o	o	o
SERIESNUM	o	o	o	o	o	o	o	o	o	o

	Short	Integer	Long	Float	Double	Varchar	Text	Ipv4	Ipv6	D
STDDEV / STDDEV_POP	o	o	o	o	o	x	x	x	x	x
SUBSTR	x	x	x	x	x	o	x	x	x	x
SUBSTRING_INDEX	x	x	x	x	x	o	o	x	x	x
SUM	o	o	o	o	o	x	x	x	x	x
SYSDATE / NOW	x	x	x	x	x	x	x	x	x	x
TO_CHAR	o	o	o	o	o	o	x	o	o	o
TO_DATE / TO_DATE_SAFE	x	x	x	x	x	o	x	x	x	x
TO_HEX	o	o	o	o	o	o	o	o	o	o
TO_IPV4 / TO_IPV4_SAFE	x	x	x	x	x	o	x	x	x	x
TO_IPV6 / TO_IPV6_SAFE	x	x	x	x	x	o	x	x	x	x
TO_NUMBER / TO_NUMBER_SAFE	x	x	x	x	x	o	x	x	x	x
TO_TIMESTAMP	x	x	x	x	x	x	x	x	x	o
TRUNC	o	o	o	o	o	x	x	x	x	x
TS_CHANGE_COUNT	o	o	o	o	o	x	x	o	o	o
UNIX_TIMESTAMP	o	o	o	o	o	x	x	x	x	x
UPPER	x	x	x	x	x	o	x	x	x	x
VARIANCE / VAR_POP	o	o	o	o	o	x	x	x	x	x
YEAR / MONTH / DAY	x	x	x	x	x	x	x	x	x	o
ISNAN / ISINF	o	o	o	o	o	x	x	x	x	x

JSON 관련 함수

이 함수들은 입력된 JSON 데이터 타입을 인자로 받아 사용한다.

함수명	설명	비고
JSON_EXTRACT(JSON 칼럼명, 'json path')	해당 값을 string type으로 출력한다. (해당 객체가 없을 경우 ERROR를 출력한다.)	• JSON 객체 혹은 배열형 : 포함된 모든 객체를 문자열로 변환해서 리턴 • 문자열 : 그대로 리턴 • 숫자형 :문자열로 변환하여 리턴 • boolean 형 : "True" or "False" 리턴

JSON_EXTRACT_DOUBLE(JSON 칼럼명, 'json path')	해당 값을 부동소수점 64비트 double type으로 출력한다. (해당 객체가 없을 경우 NULL을 출력한다.)	• JSON 객체 혹은 배열형 : NULL 리턴 • 문자열 : 변환하여 리턴하고, 변환 실패시 NULL 리턴 • 숫자형 : 64비트 실수 리턴 • boolean 형 : "True"는 1.0, "False"는 0.0 리턴
JSON_EXTRACT_INTEGER(JSON 칼럼명, 'json path')	해당 값을 64비트 integer type으로 출력한다. (해당 객체가 없을 경우 NULL을 출력한다.)	• JSON 객체 혹은 배열형 : NULL 리턴 • 문자열 : 변환하여 리턴하고, 변환 실패시 NULL 리턴 • 숫자형 : 64비트 정수 리턴 • boolean 형 : "True"는 1 , "False"는 0 리턴
JSON_EXTRACT_STRING(JSON 칼럼명, 'json path')	해당 값을 string type으로 출력한다. (해당 객체가 없을 경우 NULL을 출력한다.) operator(->)와 같은 결과를 출력한다.	• JSON 객체 혹은 배열형 : 포함된 모든 객체를 문자열로 변환해서 리턴 • 문자열 : 그대로 리턴 • 숫자형 :문자열로 변환하여 리턴 • Boolean 형 : "True", "Fase" 리턴
JSON_IS_VALID('json string')	json string이 json format에 유효한지 확인한다.	• 0 : False • 1 : True
JSON_TYPEOF(JSON 칼럼명, 'json path')	해당 값의 타입을 반환한다.	• None : 해당 키가 존재하지 않음 • Object : 객체형 • Integer : 정수형 • Real : 실수형 • String : 문자형 • True/False : Boolean • Array :배열형 • Null : NULL

example

```
Mach> CREATE TABLE jsontbl (name VARCHAR(20), jval JSON);
Created successfully.

Mach> INSERT INTO jsontbl VALUES("name1", '{"name":"test1"}');
1 row(s) inserted.
Mach> INSERT INTO jsontbl VALUES("name2", '{"name":"test2", "value":123}');
1 row(s) inserted.
Mach> INSERT INTO jsontbl VALUES("name3", '{"name":{"class1": "test3"}}');
1 row(s) inserted.
Mach> INSERT INTO jsontbl VALUES("name4", '{"myarray": [1, 2, 3, 4]}');
1 row(s) inserted.
Mach> INSERT INTO jsontbl VALUES("name5", '{"name":"error"');
[ERR-02233: Error occurred at column (2): (Error in json load.)]

Mach> SELECT name, JSON_EXTRACT_STRING(jval, '$.name') FROM jsontbl;
name                 JSON_EXTRACT_STRING(jval, '$.name')
-----------------------------------------------------------------------------------------------------------
name4                NULL
name3                {"class1": "test3"}
name2                test2
name1                test1
[4] row(s) selected.

Mach> SELECT name, JSON_EXTRACT_INTEGER(jval, '$.myarray[1]') FROM jsontbl;
name                 JSON_EXTRACT_INTEGER(jval, '$.myarray[1]')
-------------------------------------------------------------------
name4                2
name3                NULL
name2                NULL
name1                NULL
[4] row(s) selected.

Mach> SELECT name, JSON_TYPEOF(jval, '$.name') FROM jsontbl;
name                 JSON_TYPEOF(jval, '$.name')
-----------------------------------------------------------------------------------------------------------
name4                None
name3                Object
name2                String
name1                String
[4] row(s) selected.
```

JSON Operator

JSON 데이터의 object에 접근할 때 '->' operator를 사용할 수 있다.

JSON_EXTRACT_STRING과 같은 결과를 반환한다.

```
json_col -> 'json path'
```

example

```
Mach> SELECT name, jval->'$.name' FROM jsontbl;
name                 JSON_EXTRACT_STRING(jval, '$.name')
-----------------------------------------------------------------------------------------------------------
name4                NULL
name3                {"class1": "test3"}
name2                test2
name1                test1
[4] row(s) selected.

Mach> SELECT name, jval->'$.myarray[1]' FROM jsontbl;
name                 JSON_EXTRACT_INTEGER(jval, '$.myarray[1]')
-------------------------------------------------------------------
name4                2
name3                NULL
name2                NULL
name1                NULL
[4] row(s) selected.

Mach> SELECT name, jval->'$.name.class1' FROM jsontbl;
name                 jval->'$.name.class1'
-----------------------------------------------------------------------------------------------------------
name4                NULL
name3                test3
name2                NULL
name1                NULL
[4] row(s) selected
```

시스템/세션 관리

ALTER SYSTEM

시스템 단위의 자원을 관리하거나 설정을 변경하는 구문이다.

목차

KILL SESSION

alter_system_kill_session_stmt:


```
alter_system_kill_session_stmt: 'ALTER SYSTEM KILL SESSION' number
```

SessionID를 가진 특정 세션을 종료시킨다.

단, SYS 유저만이 구문을 수행할 수 있으며 자기 자신의 세션에 대해서는 KILL할 수 없다.

CANCEL SESSION

alter_system_cancel_session_stmt:


```
alter_system_cancel_session_stmt ::= 'ALTER SYSTEM CANCEL SESSION' number
```

SessionID를 가진 특정 세션을 취소시킨다.

접속이 끊어지는 대신 수행중인 동작을 취소하고, 사용자에게 해당 수행이 취소되었다는 에러 코드를 되돌린다. 단, KILL과 마찬가지로 자기 자신이 연결된 세션에 대해서는 취소를 할 수 없다.

CHECK DISK_USAGE

alter_system_check_disk_stmt:


```
alter_system_check_disk_stmt ::= 'ALTER SYSTEM CHECK DISK_USAGE'
```

V$STORAGE에서 Log Table의 디스크 사용량을 나타내는 DC_TABLE_FILE_SIZE 의 값을 보정한다.

Process Failure나 Power Failure 발생시 디스크 사용량이 부정확할 수 있다. 이 명령어를 통해서 파일 시스템으로부터 정확한 값을 읽어온다. 하지만 파일 시스템에 상당한 부하를 줄 수 있기 때문에 지양해야 한다.

INSTALL LICENSE

alter_system_install_license_stmt:


```
alter_system_install_license_stmt ::= 'ALTER SYSTEM INSTALL LICENSE'
```

라이선스 파일의 기본위치($MACHBASE_HOME/conf/license.dat)에 라이선스 파일을 설치한다.

해당 라이선스가 설치에 적합한지 판별 후 설치된다.

INSTALL LICENSE (PATH)

alter_system_install_license_path_stmt:

```
alter_system_install_license_path_stmt: ::= 'ALTER SYSTEM INSTALL LICENSE' '=' "'" path "'"
```

특정 위치에 있는 라이선스 파일을 설치한다.

해당 위치에 존재하지 않거나 올바르지 않은 라이선스 파일을 입력했을 시에는 에러가 발생한다. 경로는 반드시 절대경로로 입력해야 한다. 해당 라이선스가 설치에 적합한지 판별 후 설치된다.

SET

> 5.6 이후 버전 부터 지원되는 기능입니다.

alter_system_set_stmt:

```
alter_system_set_stmt ::= 'ALTER SYSTEM SET' prop_name '=' value
```

System 의 Property 를 수정할 수 있다. 수정 가능한 Property 는 다음과 같다.

- QUERY_PARALLEL_FACTOR
- DEFAULT_DATE_FORMAT
- TRACE_LOG_LEVEL
- PAGE_CACHE_MAX_SIZE
- MAX_SESSION_COUNT
- SESSION_IDLE_TIMEOUT_SEC
- PROCESS_MAX_SIZE
- TAG_CACHE_MAX_MEMORY_SIZE

ALTER SESSION

세션 단위의 자원을 관리하거나 설정을 변경하는 구문이다.

SET SQL_LOGGING

alter_session_sql_logging_stmt:


```
alter_session_sql_logging_stmt ::= 'ALTER SESSION SET SQL_LOGGING' '=' flag
```

해당 세션의 Trace Log에 메시지를 남길지 여부를 결정한다.

이 메시지를 Bit Flag 로서 다음의 값을 사용하면 된다.

- 0x1 : Parsing, Validation, Optimization 단계에서 발생하는 에러를 남긴다.
- 0x2 : DDL을 수행한 결과를 남긴다.

즉, 해당 플래그의 값이 2일 경우에는 DDL만 로깅하고, 3일 경우에는 에러 및 DDL을 함께 로깅하는 것이다.

아래는 해당 세션의 로깅 플래그를 변경하고, 에러 로깅을 남기는 예제이다.

```
Mach> alter session set SQL_LOGGING=1;
Altered successfully.
Mach> exit
```

SET DEFAULT_DATE_FORMAT

alter_session_set_defalut_dateformat_stmt:


```
alter_session_set_defalut_dateformat_stmt ::= 'ALTER SESSION SET DEFAULT_DATE_FORMAT' '=' date_format
```

해당 세션의 Datetime 자료형의 기본 포맷을 설정한다.

서버가 구동되면, Property 인 DEFAULT_DATE_FORMAT 의 값이 세션 속성으로 설정이 된다.
Property 의 속성이 바뀌지 않았다면, 세션의 값 또한 "YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn"이 될 것이다.
시스템과 무관하게, 특정 사용자에 한해 Datetime 자료형의 기본 포맷을 수정할 경우에 이 명령어를 사용한다.

v$session 에 해당 세션마다 설정된 Default Date Format 이 있고 확인도 할 수 있다. 아래는 해당 세션의 값을 확인 및 변경하는 예제이다.

```
Mach> CREATE TABLE time_table (time datetime);
Created successfully.

Mach> SELECT DEFAULT_DATE_FORMAT from v$session;
default_date_format
-----------------------------------------------
YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn
[1] row(s) selected.

Mach> INSERT INTO time_table VALUES(TO_DATE('2016-11-11'));
[ERR-00300 : Invalid date format or input string.([2016-11-11]:[%Y-%m-%d %H:%M:%S %0:%1:%2])]

Mach> ALTER SESSION SET DEFAULT_DATE_FORMAT='YYYY-MM-DD';
Altered successfully.

Mach> SELECT DEFAULT_DATE_FORMAT from v$session;

default_date_format
-----------------------------------------------
YYYY-MM-DD
[1] row(s) selected.

Mach> INSERT INTO time_table VALUES(TO_DATE('2016-11-11'));
1 row(s) inserted.

Mach> SELECT * FROM time_table;

TIME
---------------------------------
2016-11-11

[1] row(s) selected.
```

SET SHOW_HIDDEN_COLS

alter_session_set_hidden_column_stmt:


```
alter_session_set_hidden_column_stmt ::= 'ALTER SESSION SET SHOW_HIDDEN_COLS' '=' ( '0' | '1' )
```

해당 세션의 select 수행시 *로 표현된 컬럼에서 숨은 컬럼 (_arrival_time)을 출력할 것인지를 결정한다.

서버가 구동되면, 전역 프로퍼티인 SHOW_HIDDEN_COLS의 값이 세션 속성으로 0이 설정된다.
만일 사용자가 자기 세션의 기본 동작을 변경하고자 할 경우에는 이 값을 1로 설정하면 된다.

v$session에 해당 세션마다 설정된 SHOW_HIDDEN_COLS 값이 있으며, 확인할 수 있다.

```
Mach> SELECT * FROM  v$session;
ID                  CLOSED      USER_ID     LOGIN_TIME                    SQL_LOGGING SHOW_HIDDEN_COLS
-------------------------------------------------------------------------------------------------------------
DEFAULT_DATE_FORMAT                                                              HASH_BUCKET_SIZE
-------------------------------------------------------------------------------------------------------------
1                   0           1           2015-04-29 17:23:56 248:263:000 3           0
YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn                                                20011
[1] row(s) selected.
Mach> ALTER SESSION SET SHOW_HIDDEN_COLS=1;
Altered successfully.
Mach> SELECT * FROM v$session;
_ARRIVAL_TIME                   ID                   CLOSED      USER_ID     LOGIN_TIME                      SQL_LO
-------------------------------------------------------------------------------------------------------------
```

```
SHOW_HIDDEN_COLS DEFAULT_DATE_FORMAT                                                                                  HASH_BUCKET_SIZE
-------------------------------------------------------------------------------------------------------------------------------------------
1970-01-01 09:00:00 000:000:000 1                   0           1           2015-04-29 17:23:56 248:263:000 3
1           YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn                                                    20011
[1] row(s) selected.
```

SET FEEDBACK_APPEND_ERROR

alter_session_set_feedback_append_err_stmt:


```
alter_session_set_feedback_append_err_stmt ::= 'ALTER SESSION SET FEEDBACK_APPEND_ERROR' '=' ( '0' | '1' )
```

해당 세션의 Append 에러 메시지를 Client program으로 보낼 것인지를 설정한다.

에러 메시지는 다음의 값을 사용하면 된다.

- 0 = 에러 메시지를 보내지 않는다.
- 1 = 에러 메시지를 보낸다.

아래는 사용 예제이다.

```
mach> ALTER SESSION SET FEEDBACK_APPEND_ERROR=0;
Altered successfully.
```

SET MAX_QPX_MEM

alter_session_set_max_qpx_mem_stmt:


```
alter_session_set_max_qpx_mem_stmt ::= 'ALTER SESSION SET MAX_QPX_MEM' '=' value
```

해당 세션의 하나의 SQL Statement가 GROUP BY, DISTINCT, ORDER BY 연산을 수행할때 사용하는 최대 메모리의 크기를 지정한다.

만약 최대 메모리 이상의 메모리 할당을 시도하면, 시스템은 그 SQL문의 수행을 취소하고 오류로 처리한다.
오류 발생 시 machbase.trc에 해당 질의문을 포함한 에러 코드 및 에러 메시지를 기록한다.

```
Mach> ALTER SESSION SET MAX_QPX_MEM=1073741824;
Altered successfully.

Mach> SELECT * FROM v$session;
ID                  CLOSED      USER_ID     LOGIN_TIME                    CLIENT_TYPE
-------------------------------------------------------------------------------------------------------------------------------
USER_NAME                                                                        USER_IP
-------------------------------------------------------------------------------------------------------------------------------
FEEDBACK_APPEND_ERROR DEFAULT_DATE_FORMAT                                                                          HASH_BUCKET
-------------------------------------------------------------------------------------------------------------------------------
RS_CACHE_MAX_MEMORY_PER_QUERY RS_CACHE_MAX_RECORD_PER_QUERY RS_CACHE_APPROXIMATE_RESULT_ENABLE IDLE_TIMEOUT
-------------------------------------------------------------------------------------------------------------------------------
14                  0           1           2021-03-08 16:33:01 503:181:809 CLI
NULL                                                                             192.168.0.194
1                    YYYY-MM-DD HH24:MI:SS mmm:uuu:nnn                                                    20011
16777216                    50000                         0                                0
[1] row(s) selected.
Elapsed time: 0.001
```

- 최대 메모리 크기 이상을 SQL문에서 사용했을 때, trc 에러

```
[2021-03-08 16:36:32 P-69000 T-140515328653056][INFO] DML FAILURE (2E10000084:Memory allocation error (alloc'd: 104
```

- 최대 메모리 크기 이상을 SQL문에서 사용했을 때, machsql 에러 메세지

```
Mach> select * from tag order by value DESC, time ASC;
NAME                  TIME                            VALUE
------------------------------------------------------------------------------------
[ERR-00132: Memory allocation error (alloc'd: 1048595, max: 1048576).]
[0] row(s) selected.
Elapsed time: 0.447
```

(* 위의 trc 메세지를 출력하기 위해서, 임의로 최대 메모리 크기를 1MB로 변경한 후 에러를 확인하였다.)

SET SESSION_IDLE_TIMEOUT_SEC

alter_session_set_session_idle_timeout_sec_stmt:


```
alter_session_set_session_idle_timeout_sec_stmt ::= 'ALTER SESSION SET SESSION_IDLE_TIMEOUT_SEC' '=' value
```

해당 세션이 유휴 상태일 때의 연결 유지 시간을 지정한다.

초단위로 지정하며 유휴 상태로 설정된 시간이 지나게 되면 세션이 종료된다.

v$session 에서 세션에 설정된 idle timeout 시간을 조회할 수 있다.

```
Mach> ALTER SESSION SET SESSION_IDLE_TIMEOUT_SEC=200;
Altered successfully.


Mach> SELECT IDLE_TIMEOUT FROM V$SESSION;
IDLE_TIMEOUT
-----------------------
200
[1] row(s) selected.
```

SET QUERY_TIMEOUT

alter_session_set_query_timeout_stmt:


```
alter_session_set_query_timeout_stmt ::= 'ALTER SESSION SET QUERY_TIMEOUT' '=' value
```

세션에서 Query를 수행 시 서버의 응답을 대기하는 시간이다.

초단위로 지정하며 Query 수행 후 서버에서의 응답이 지정된 시간을 초과하면 Query가 종료된다.

v$session에서 세션에 설정된 Query timeout 시간을 조회할 수 있다.

```
Mach> ALTER SESSION SET QUERY_TIMEOUT=200;
Altered successfully.


Mach> SELECT QUERY_TIMEOUT FROM V$SESSION;
QUERY_TIMEOUT
-----------------------
200
[1] row(s) selected.
```

SDK

MACHBASE SDK (Software Development Kit) 에 대한 문서이다.

CLI/ODBC

목차

CLI란 ISO/IEC 9075-3:2003에 정의된 소프트웨어 개발 표준이다.

CLI는 데이터베이스에 어떻게 SQL을 전달하고, 결과 값을 어떻게 받고 분석해야 하는지에 대한 함수 및 명세를 정의하고 있다. 이 CLI는 1990년 초창기에 개발되었고, C 와 COBOL 언어 만을 위해 개발되었고, 현재까지 그 스펙이 유지되고 있다.

현재까지 가장 널리 알려진 표준 인터페이스는 ODBC(Open Database Connectivity)로서 클라이언트 프로그램이 데이터베이스의 종류와 무관하게 데이터베이스 접속할 수 있는 방법을 제시해 주고 있다. 현재 최신 ODBC API 버전은 3.52 로서 ISO와 X/Open 표준에 정의되어 있다.

표준 CLI 함수

표준 함수의 사용법에 대해서는 다음과 같은 링크를 참조한다.

- 위키피디아
- 오픈그룹 문서

다음의 함수를 참고하면 된다.

SQLAllocConnect	SQLDisconnect	SQLGetDescField	SQLPrepare
SQLAllocEnv	SQLDriverConnect	SQLGetDescRec	SQLPrimaryKeys
SQLAllocHandle	SQLExecDirect	SQLGetDiagRec	SQLStatistics
SQLAllocStmt	SQLExecute	SQLGetEnvAttr	SQLRowCount
SQLBindCol	SQLFetch	SQLGetFunctions	SQLSetConnectAttr
SQLBindParameter+	SQLFreeConnect	SQLGetInfo	SQLSetDescField
SQLColAttribute	SQLFreeEnv	SQLGetStmtAttr	SQLSetDescRec
SQLColumns	SQLFreeHandle	SQLGetTypeInfo	SQLSetEnvAttr
SQLConnect	SQLFreeStmt	SQLNativeSQL	SQLSetStmtAttr
SQLCopyDesc	SQLGetConnectAttr	SQLNumParams	SQLStatistics
SQLDescribeCol	SQLGetData	SQLNumResultCols	SQLTables

접속을 위한 연결 스트링

CLI를 통해 접속을 하기 위해서는 연결 스트링을 만들어야 하며, 각각의 내용은 다음과 같다.

연결 스트링 항목명	항목 설명
DSN	데이터 소스 명을 지정한다. ODBC에서는 리소스가 담긴 파일의 섹션 명을 기술하고, CLI에서는 서버명 혹은 IP 주소를 지정한다.
DBNAME	Machbase의 DB명을 기술한다.
SERVER	Machbase가 위치하는 서버의 호스트 명 혹은 IP 주소를 가리킨다.
NLS_USE	서로 사용할 언어 종류를 설정한다.(현재 사용되지 않으며, 차후 확장을 위해 유지한다.)
UID	사용자 아이디
PWD	사용자 패스워드
PORT_NO	접속할 포트 번호
PORT_DIR	유닉스에서 Unix domain으로 접속할 경우 사용되는 파일 경로를 지정한다. (서버에서 수정했을 경우에 지정하며, 디폴트로는 지정하지 않아도 동작한다.)
CONNTYPE	클라이언트와 서버의 접속 방법을 지정한다. 1: TCP/IP INET 으로 접속 2: Unix Domain 으로 접속
COMPRESS	Append 프로토콜을 압축할 것인지 나타낸다.

연결 스트링 항목명	항목 설명
	이 값이 0일 경우에는 압축하지 않고 전송한다. 이 값이 0보다 큰 임의의 값일 경우에는 그 값보다 Append 레코드가 클 경우에만 압축한다. 예) COMPRESS=512 레코드 사이즈가 512보다 클 경우에만 압축하여 동작한다. 원격 접속일 경우 압축하면 전송 성능이 향상된다.
SHOW_HIDDEN_COLS	숨겨진 컬럼(_arrival_time)을 select * 로 수행시 보여줄 것인지 결정한다. 0일 경우에는 보이지 않으며, 1일 경우에 해당 컬럼의 정보가 출력된다.
CONNECTION_TIMEOUT	최초 연결시에 얼마나 대기할 것이지 설정한다. 디폴트로는 30초가 설정되어 있다. 만일 최초 연결시 서버의 응답이 30초 보다 더 느려지는 경우를 고려하면, 이 값을 더 크게 설정해야 한다. CONNECTION_TIMEOUT에서 0 값은 Timeout에 제한이 없음을 의미하며 연결이 실패할 때에도 무한정으로 대기하므로 되도록이면 사용하지 않는 편이 좋다.
SOCKET_TIMEOUT	Protocol I/O에 시간이 걸리면 발생하는 timeout이다. Client에서 검사하여 대기 후 Disconnect를 수행한다. ORACLE의 Read Timeout과 같다. (MYSQL, MSSQL에서는 동일하게 SOCKET_TIMEOUT이라는 이름으로 사용한다.) Connection String에서 SOCKET_TIMEOUT=NN(초)로 설정하며 기본값은 30분(1800)으로 설정된다.
ALTERNATIVE_SERVERS	cluster 버전을 사용 시, 여러 대의 브로커의 정보를 추가적으로 가지게 되는 설정이다. 다중의 브로커를 등록해두었을 시, 접속되어있던 브로커가 혹시 내려가게 된 경우에도 다른 브로커에 접속한 뒤, 입력하던 데이터를 계속해서 입력하게 된다. 여러개의 브로커를 등록할 수 있으며, <서버 주소>:<서버 포트>의 값을 ',' 단위로 이어서 작성한다. ex) ALTERNATIVE_SERVERS=192.168.0.10:20320,192.168.0.11:20320;

CLI 접속 예제는 다음과 같다.

```
sprintf(connStr,"SERVER=127.0.0.1;COMPRESS=512;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

if (SQL_ERROR == SQLDriverConnect( gCon, NULL, (SQLCHAR *)connStr, SQL_NTS, NULL, 0, NULL, SQL_DRIVER_NOPROMPT )) {
   ...
}
```

확장 CLI 함수 (APPEND)

CLI 확장 함수는 Machbase 서버에 데이터를 초고속으로 입력하기 위해 제공되는 Append 프로토콜을 구현하기 위한 함수이다.

이 함수는 크게 4가지의 함수로 구성되어 있는데, 채널의 오픈, 채널에 대한 데이터 입력, 채널의 플러쉬, 채널 클로징이다.

Append 프로토콜의 이해

Machbase에서 제공하는 Append 프로토콜은 비동기 방식으로 동작한다. 비동기라 함은 클라이언트가 서버에게 요청한 특정 작업에 대한 응답이 서로 완전히 동기화되지 않고, 임의의 이벤트가 발생하는 순간에 발생하는 것을 의미한다. 즉, 클라이언트가 Append를 수행했다고 하더라도, 그 수행에 대한 결과를 바로 얻거나 확인할 수 없으며, 서버에서 준비가 되는 임의의 시점에 그것을 확인할 수 있다는 것이다. 이런 이유로 Append 프로토콜을 활용해서 응용 프로그램을 개발하는 개발자는 다음과 같은 내부 동작에 대한 이해를 가져야 한다. 이후의 설명은 클라이언트가 언제 어떻게 서버에서 발생하는 비동기 에러를 검출하고 사용자에게 되돌여주는지에 대한 것이다.

Append 데이터의 전송

SQLExecute 혹은 SQLExecDirect()와 같은 일반적인 호출에서 Machbase는 즉시 그 결과를 클라이언트에게 되돌려주는 동기화 방식을 사용한다. 그러나, SQLAppendDataV2()는 사용자 데이터가 입력된 이후 즉시 요청을 보내지 않는다. 대신, 클라이언트 통신 버퍼가 모두 찰 때 까지 대기하고 있다가 모두 차면 그 이후에 한꺼번에 데이터를 클라이언트로 전송하게 된다. 이렇게 설계된 이유는 Append를 사용하는 클라이언트의 입력 데이터가 초당 수만에서 수십만 레코드를 가정하였기 때문에 고속의 데이터 전송을 위한 버퍼링 방식을 활용한 것이다. 이런 이유로 만일 사용자가 임의로 해당 버퍼의 내용을 전송하고자 할 경우에는 SQLAppendFlush() 함수를 호출하여, 명시적으로 데이터를 입력할 수 있다.

Append 데이터의 에러 확인

앞에서 언급한 바와 같이 Append 프로토콜은 버퍼링되어 비동기로 동작한다. 특히, 서버에서 에러가 발생하지 않았을 경우에는 아무런 응답을 받지 않고, 에러가 발생했을 경우에만 에러를 검출하는 방식을 취하기 때문에 에러가 언제 어떻게 검출되는지 이해하는 것이 매우 중요하다. 또한, 에러를 검출하는 비용이 상대적으로 매우 크기 때문에 레코드 입력시마다 매번 검사하는 것이 매우 비효율적으로 판단되어, 현재 Machbase에서는 명시적으로 다음과 같은 경우에만 에러를 검출하도록 되어 있다. 에러가 검출될 경우에는 사용자가 설정한 에러 콜백 함수를 매번 호출하게 된다.

1. 전송 버퍼가 모두 차고, 서버에게 명시적으로 데이터를 전송한 이후 검사
2. SQLAppendFlush() 내부에서 서버에게 명시적으로 데이터를 전송한 이후 검사
3. SQLAppendClose() 내부에서 종료 직전에 검사

즉, 기본적으로 위의 3가지 경우에만 에러를 검출하도록 되어 있어, I/O의 발생을 최소화하도록 설계되었다.

서버 에러 검사를 위한 부가 옵션

성능을 최대한으로 달성하기 위해 기본적으로 설정된 에러 검출 기법은 사용자가 원하는 경우 좀 더 빈번하게 검사하고, 이를 활용할 수 있다. 즉, SQLAppendOpen() 함수의 마지막 인자인 aErrorCheckCount를 조절함으로서 가능한다. 이 값이 0일 경우에는 별도의 확인 동작을 하지 않고, 기본으로 동작한다. 그러나, 만일 이 값이 0보다 클 경우에는 SQLAppendData()의 호출 횟수마다 명시적으로 에러를 검사하도록 되어있다. 다시 말해 이 값이 10일 경우에는 10번의 Append 동작마다 에러를 검사하는 비용을 지불한다. 따라서, 이 값이 작을 경우에는 에러 검출을 위한 시스템 리소스를 많이 사용하기 때문에 적절한 숫자로 조절하여 사용해야 한다.

서버 에러 발생시 Trace 로그 남기기

만일 에러가 발생한 Append 데이터에 대해서 별도로 Trace 로그를 남기고자 할 경우에는 서버에 준비된 프로퍼티 DUMP_APPEND_ERROR 를 1로 설정한다. 이렇게 설정하면, mach.trc 파일에 해당 에러를 발생시킨 레코드에 대한 명세가 파일로 기록된다. 단, 에러의 횟수가 과도할 경우 시스템 리소스의 사용량이 급격히 늘어나, Machbase의 전체 성능을 떨어뜨릴 수 있으므로 주의하여 사용해야 한다.

APPEND 함수 설명

SQLAppendOpen

```
SQLRETURN SQLAppendOpen(SQLHSTMT   aStatementHandle,
                        SQLCHAR   *aTableName,
                        SQLINTEGER aErrorCheckCount );
```

이 함수는 대상 테이블에 대한 채널을 오픈한다. 이후 이 채널을 닫아 주지 않으면 지속적으로 열린 상태가 유지된다.

하나의 연결에 대해 최대 1024개의 Statement 설정이 가능하다. 각 Statement마다 SQLAppendOpen을 사용하면 된다.

1. aStatementHandle : Append를 수행할 Statement의 핸들을 나타낸다.
2. aTableName : Append를 수행할 대상 테이블의 이름을 나타낸다.
3. aErrorCheckCount : 몇 건의 데이터가 입력될 때 마다 서버의 에러를 검사할 것인지 결정한다. 이 값이 0일 경우에는 임의로 에러를 검사하지 않는다.

SQLAppendData (deprecated)

```
SQLRETURN  SQLAppendData(SQLHSTMT StatementHandle, void *aData[]);
```

이 함수는 해당 채널에 대해 데이터를 입력하는 함수이다.

- aData는 입력될 데이터의 포인터를 담고 있는 배열이다. 배열의 개수는 Open시에 지정한 테이블이 보유하고 있는 컬럼의 개수와 일치해야 한다.
- 리턴값은 SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR가 가능하다.
  특히, SQL_SUCCESS_WITH_INFO가 반환되었을 경우에는 입력된 특정 컬럼의 길이가 길어 잘리는 등의 오류가 있을 수 있으므로 결과를 다시 확인하여야 한다.

데이터 타입에 따른 설정

숫자형 및 문자형

- float, double, short, int, long long, char * 과 같은 타입은 해당 값에 대한 포인터 설정 만으로 잘 동작한다.

주소형

- ipv4 의 경우에는 5 바이트 무부호 문자(unsigned char)의 배열로 넘긴다.
- 첫 번째 바이트는 4로 설정하고, 이후의 4바이트는 연속되는 주소값으로 설정한다.
- 예를 들어, 127.0.0.1의 경우에는 5바이트 배열 0x04, 0x7f, 0x00, 0x00, 0x01 의 순으로 들어가게 된다.

```
// 4개의 컬럼 정보를 가지는 테이블의 경우 (short(16), int(32), long(64), varchar)

testAppendIPFunc()
{
   short val1 = 0;
   int   val2 = 1;
   long long  val3 = 2;
   char *val4 = "my string";
   void *valueArray[4];
```

```
    valueArray[0] = (void *)&val1;
    valueArray[1] = (void *)&val2;
    valueArray[2] = (void *)&val3;
    valueArray[3] = (void *)val4;

    SQLAppendData(aStmt, valueArray);
}
```

데이터 타입에 따른 설정

datetime 형

- Machbase 는 내부적으로 나노 단위 시간 해상도 값을 가지기 때문에 클라이언트에서 시간을 설정할 때는 변환과정을 거쳐야 하며, 64비트 부호없는 정수형 값으로 표현된다.
  따라서 적절한 변환을 위해서는 유닉스 라이브러리인 mktime을 이용하여 초로 변환한 이후에 나노 값을 더해주어야 한다.
  ※ Machbase의 시간 = (1970년 1월 1일 이후로부터의 총 시간 (초)) * 1,000,000,000 + mili-second * 1,000,000 + micro-second * 1000 + nano-second;

```
// Date String이 "연도-월-일 시:분:초 밀리:마이크로:나노" 형태로 입력될 경우 코드

testAppendDateStrFunc(char *aDateString)
{
    int yy, int mm, int dd, int hh, int mi, int ss;
    unsigned long t1;
    void *valueArray[5];
    sscanf(aDateString, "%d-%d-%d %d:%d:%d %d:%d:%d",
        &yy, &mm, &dd, &hh, &mi, &ss, &mmm, &uuu, &nnn);
    sTm.tm_year = yy - 1900;
    sTm.tm_mon = mm - 1;
    sTm.tm_mday = dd;
    sTm.tm_hour = hh;
    sTm.tm_min = mi;
    sTm.tm_sec = ss;
    t1 = mktime(&sTm);
    t1 = t1 * 1000000000L;
    t1 = t1 + (mmm*1000000L) + (uuu*1000) + nnn;

    valueArray[4] = &t1;
    SQLAppendData(aStmt, valueArray);
}
```

SQLAppendDataByTime(deprecated)

```
SQLRETURN  SQLAppendDataByTime(SQLHSTMT StatementHandle, SQLBIGINT aTime, void *aData[]);
```

이 함수는 해당 채널에 대해 데이터를 입력하는 함수이며, DB에 저장되는 _arrival_time 값을 현재 시간이 아닌 특정 시간의 값으로 설정할 수 있다.

예를 들면, 1개월전 로그 파일에 있는 날짜를 그 당시의 날짜로 입력하고자 할때 사용된다.

- aTime은 _arrival_time으로 설정된 time 값이다.
- aData는 입력될 데이터의 포인터를 담고 있는 배열이다.
- 배열의 개수는 Open시에 지정한 테이블이 보유하고 있는 컬럼의 개수와 일치해야 한다.

나머지 사항은 SQLAppendData()함수를 참고하여 작성하면 된다.

```
// 4개의 컬럼 정보를 가지는 테이블의 경우  (short(16), int(32), long(64), varchar)

testAppendFuncWithTime()
{
   long long sTime = 1;
   short val1 = 0;
   int   val2 = 1;
   long long  val3 = 2;
   char *val4 = "my string";
   void *valueArray[4];

   valueArray[0] = (void *)&val1;
   valueArray[1] = (void *)&val2;
   valueArray[2] = (void *)&val3;
```

```
    valueArray[3] = (void *)val4;

    SQLAppendDataByTime(aStmt, sTime, valueArray);
}
```

SQLAppendDataV2

```
SQLRETURN  SQLAppendDataV2(SQLHSTMT StatementHandle, SQL_APPEND_PARAM *aData);
```

이 함수는 Machbase 2.0 부터 새로 도입된 Append 함수로서, 기존의 함수에서 불편했던 입력 방식을 편리하게 대폭 개선한 함수이다.

특히, 2.0에서 도입된 TEXT와 BINARY 타입의 경우는 SQLAppendDataV2() 함수에서만 입력이 가능하다.

- 각 타입에 맞는 NULL 입력 가능
- VARCHAR 입력시 스트링 길이 입력 가능
- IPv4, IPv6 입력시 바이너리 및 스트링 형태의 데이터 입력 가능
- TEXT, BINARY 타입에 대한 데이터 길이 지정 가능

함수 인자는 다음과 같이 구성된다.

- aData는 SQL_APPEND_PARAM 이라는 인자배열을 가리키는 포인터이다. 이 배열의 개수는 Open시에 지정한 테이블이 보유하고 있는 컬럼의 개수와 일치해야 한다.
- 리턴값은 SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR 가 가능하다. 특히, SQL_SUCCESS_WITH_INFO가 반환되었을 경우에는 입력된 특정 컬럼의 길이가 길어 잘리는 등의 오류가 있을 수 있으므로 결과를 다시 확인하여야 한다.

아래는 실제로 V2에서 사용될 `SQL_APPEND_PARAM` 의 정의이며, 이 내용은 `machbase_sqlcli.h` 에 포함되어 있다.

```
typedef struct machAppendVarStruct
{
    unsigned int mLength;
    void *mData;
} machAppendVarStruct;

/* for IPv4, IPv6 as bin or string representation */
typedef struct machbaseAppendIPStruct
{
    unsigned char   mLength; /* 0:null, 4:ipv4, 6:ipv6, 255:string representation */
    unsigned char   mAddr[16];
    char           *mAddrString;
} machbaseAppendIPStruct;

/* Date time*/
typedef struct machbaseAppendDateTimeStruct
{
    long long       mTime;
#if defined(SUPPORT_STRUCT_TM)
    struct tm       mTM;
#endif
    char           *mDateStr;
    char           *mFormatStr;
} machbaseAppendDateTimeStruct;

typedef union machbaseAppendParam
{
    short                       mShort;
    unsigned short              mUShort;
    int                         mInteger;
    unsigned int                mUInteger;
    long long                   mLong;
    unsigned long long          mULong;
    float                       mFloat;
    double                      mDouble;
    machbaseAppendIPStruct      mIP;
    machbaseAppendVarStruct     mVar;     /* for all varying type */
    machbaseAppendVarStruct     mVarchar; /* alias */
    machbaseAppendVarStruct     mText;    /* alias */
    machbaseAppendVarStruct     mBinary;  /* binary */
    machbaseAppendVarStruct     mBlob;    /* reserved alias */
    machbaseAppendVarStruct     mClob;    /* reserved alias */
    machbaseAppendDateTimeStruct mDateTime;
```

```
} machbaseAppendParam;

#define SQL_APPEND_PARAM machbaseAppendParam
```

위에서 볼 수 있듯이 내부적으로 machbaseAppendParam 이라는 공용 구조체가 하나의 인자를 담고 있는 구조이다. 각 데이터 타입에 대해 데이터 및 스트링에 대한 길이 및 값을 명시적으로 입력할 수 있도록 되어 있다. 실제 사용 예는 다음과 같다.

고정 길이 숫자형 타입의 입력

고정 길이 숫자형 타입이라 함은 short, ushort, integer, uinteger, long, ulong, float, double 을 말한다. 이 타입의 경우 SQL_APPEND_PARAM의 구조체 멤버에 직접 값을 대입함으로써 입력 가능하다.

데이터베이스 타입	NULL 매크로	SQL_APPEND_PARAM 멤버
SHORT	SQL_APPEND_SHORT_NULL	mShort
USHORT	SQL_APPEND_USHORT_NULL	mUShort
INTEGER	SQL_APPEND_INTEGER_NULL	mInteger
UINTEGER	SQL_APPEND_UINTEGER_NULL	mUInteger
LONG	SQL_APPEND_LONG_NULL	mLong
ULONG	SQL_APPEND_ULONG_NULL	mULong
FLOAT	SQL_APPEND_FLOAT_NULL	mFloat
DOUBLE	SQL_APPEND_DOUBLE_NULL	mDouble

다음은 실제 값을 입력하는 예제이다.

```
// Table Schema가 8개의 컬럼이고, 각각 SHORT, USHORT, INTEGER, UINTEGER, LONG, ULONG, FLOAT, DOUBLE로 이루어진 것으로 가

void testAppendExampleFunc()
{
    SQL_APPEND_PARAM sParam[8];

    /* fixed column */
    sParam[0].mShort = SQL_APPEND_SHORT_NULL;
    sParam[1].mUShort = SQL_APPEND_USHORT_NULL;
    sParam[2].mInteger = SQL_APPEND_INTEGER_NULL;
    sParam[3].mUInteger = SQL_APPEND_UINTEGER_NULL;
    sParam[4].mLong = SQL_APPEND_LONG_NULL;
    sParam[5].mULong = SQL_APPEND_ULONG_NULL;
    sParam[6].mFloat = SQL_APPEND_FLOAT_NULL;
    sParam[7].mDouble = SQL_APPEND_DOUBLE_NULL;

    SQLAppendDataV2(Stmt, sParam);

    /* FIXED COLUMN Value */
    sParam[0].mShort = 2;
    sParam[1].mUShort = 3;
    sParam[2].mInteger = 4;
    sParam[3].mUInteger = 5;
    sParam[4].mLong = 6;
    sParam[5].mULong = 7;
    sParam[6].mFloat = 8.4;
    sParam[7].mDouble = 10.9;

    SQLAppendDataV2(Stmt, sParam);
}
```

날짜형 타입의 입력

아래는 DATETIME형의 데이터를 입력하는 예이다. 편의를 위해 몇가지의 매크로가 준비되어 있다.

SQL_APPEND_PARAM에서 mDateTime 멤버에 대한 조작을 수행한다. 아래의 매크로는 mDateTime 구조체에서 mTime이라는 64비트 정수값에 대해 설정함으로써 날짜를 지정할 수 있다.

```
typedef struct machbaseAppendDateTimeStruct
{
```

```
    long long       mTime;
#if defined(SUPPORT_STRUCT_TM)
    struct tm       mTM;
#endif
    char           *mDateStr;
    char           *mFormatStr;
} machbaseAppendDateTimeStruct;
```

매크로	설명
SQL_APPEND_DATETIME_NOW	현재의 클라이언트 시간을 입력한다.
SQL_APPEND_DATETIME_STRUCT_TM	mDateTime의 struct tm 구조체인 mTM에 값을 설정하고, 그 값을 데이터베이스로 입력한다.
SQL_APPEND_DATETIME_STRING	mDateTime의 스트링형에 대한 값을 설정하고, 이를 데이터베이스로 입력한다. mDateStr : 실제 날짜 스트링 값이 할당 mFormatStr : 날짜 스트링에 대한 포맷 스트링 할당
SQL_APPEND_DATETIME_NULL	날짜 컬럼의 값을 NULL로 입력한다.
임의의 64비트 값	이 값이 실제 datetime으로 입력된다. 이 값을 1970년 1월 1일 이후로부터 나노세컨드 단위의 시간이 흐른 정수값을 나타낸다. 예를 들어, 만일 이 값이 10억 (1,000,000,000) 이라면, 1970년 1월 1일 0시 0분 1초를 나타낸다.(GMT)

```
// 다음은 각각의 경우에 대해 실제 값을 입력하는 예제이다. 하나의 DATETIME 컬럼이 존재한다고 가정한다.
void testAppendDateTimeFunc()
{
    SQL_mach_PARAM sParam[1];
    /* NULL 입력 */
    sParam[0].mDateTime.mTime   = SQL_APPEND_DATETIME_NULL;
    SQLAppendDataV2(Stmt, sParam);

    /* 현재 시간 입력 */
    sParam[0].mDateTime.mTime      = SQL_APPEND_DATETIME_NOW;
    SQLAppendDataV2(Stmt, sParam);

    /* 임의의 값 입력 :1970.1.1일 이후로부터의 현재까지 나노세컨드의 값 */
    sParam[0].mDateTime.mTime      = 1234;
    SQLAppendDataV2(Stmt, sParam);

    /*  스트링 포맷 기준 입력 */
    sParam[0].mDateTime.mTime      = SQL_APPEND_DATETIME_STRING;
    sParam[0].mDateTime.mDateStr   = "23/May/2014:17:41:28";
    sParam[0].mDateTime.mFormatStr = "DD/MON/YYYY:HH24:MI:SS";
    SQLAppendDataV2(Stmt, sParam);

    /*  struct tm의 값을 변경하여 입력 */
    sParam[0].mDateTime.mTime      = SQL_APPEND_DATETIME_STRUCT_TM;
    sParam[0].mDateTime.mTM.tm_year = 2000 - 1900;
    sParam[0].mDateTime.mTM.tm_mon  =  11;
    sParam[0].mDateTime.mTM.tm_mday  = 31;
    SQLAppendDataV2(Stmt, sParam);
}
```

인터넷 주소형 타입의 입력

아래는 IPv4와 IPv6 형의 데이터를 입력하는 예이다. 이 역시 편의를 위해 몇가지의 매크로가 준비되어 있다. SQL_APPEND_PARAM에서 mLength 멤버에 대한 조작을 수행한다.

```
/* for IPv4, IPv6 as bin or string representation */
typedef struct machbaseAppendIPStruct
{
    unsigned char   mLength; /* 0:null, 4:ipv4, 6:ipv6, 255:string representation */
    unsigned char   mAddr[16];
    char           *mAddrString;
} machbaseAppendIPStruct;
```

매크로 (mLength 에 설정)	설명
SQL_APPEND_IP_NULL	해당 컬럼에 NULL 값을 입력
SQL_APPEND_IP_IPV4	mAddr이 IPv4를 가지고 있음
SQL_APPEND_IP_IPV6	mAddr이 IPv6를 가지고 있음
SQL_APPEND_IP_STRING	mAddrString이 주소 문자열을 가지고 있음.

다음은 각각의 경우에 대해 실제 값을 입력하는 예제이다.

```
void testAppendIPFunc()
{
    SQL_APPEND_PARAM sParam[1];
    /* NULL */
    sParam[0].mIP.mLength  = SQL_APPEND_IP_NULL;
    SQLAppendDataV2(Stmt, sParam);

    /* 배열을 직접 수정 */
    sParam[0].mIP.mLength  = SQL_APPEND_IP_IPV4;
    sParam[0].mIP.mAddr[0] = 127;
    sParam[0].mIP.mAddr[1] = 0;
    sParam[0].mIP.mAddr[2] = 0;
    sParam[0].mIP.mAddr[3] = 1;
    SQLAppendDataV2(Stmt, sParam);

    /* IPv4 from binary */
    sParam[0].mIP.mLength  = SQL_APPEND_IP_IPV4;
    *(in_addr_t *)(sParam[0].mIP.mAddr) = inet_addr("192.168.0.1");
    SQLAppendDataV2(Stmt, sParam);

    /* IPv4 : ipv4 from string */
    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "203.212.222.111";
    SQLAppendDataV2(Stmt, sParam);

    /* IPv4 : ipv4 from invalid string */
    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "ip address is not valid";
    SQLAppendDataV2(Stmt, sParam);                          // invalid IP value

    /* IPv6 : ipv6 from binary bytes */
    sParam[0].mIP.mLength  = SQL_APPEND_IP_IPV6;
    sParam[0].mIP.mAddr[0]  = 127;
    sParam[0].mIP.mAddr[1]  = 127;
    sParam[0].mIP.mAddr[2]  = 127;
    sParam[0].mIP.mAddr[3]  = 127;
    sParam[0].mIP.mAddr[4]  = 127;
    sParam[0].mIP.mAddr[5]  = 127;
    sParam[0].mIP.mAddr[6]  = 127;
    sParam[0].mIP.mAddr[7]  = 127;
    sParam[0].mIP.mAddr[8]  = 127;
    sParam[0].mIP.mAddr[9]  = 127;
    sParam[0].mIP.mAddr[10] = 127;
    sParam[0].mIP.mAddr[11] = 127;
    sParam[0].mIP.mAddr[12] = 127;
    sParam[0].mIP.mAddr[13] = 127;
    sParam[0].mIP.mAddr[14] = 127;
    sParam[0].mIP.mAddr[15] = 127;
    SQLAppendDataV2(Stmt, sParam);

    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "::127.0.0.1";
    SQLAppendDataV2(Stmt, sParam);

    sParam[0].mIP.mLength     = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = "FFFF:FFFF:1111:2222:3333:4444:7733:2123";
    SQLAppendDataV2(Stmt, sParam);
```

```
}
```

IP 타입을 문자열 (STRING) 로 입력할경우 SQLAppendDataV2 이후에 각각 자료형에 맞게 mLength가 4 또는 6으로 바뀌게 된다.
따라서 반복문에서 코딩할 경우 매번 SQLAppendDataV2() 전에, mLength 를 `SQL_APPEND_IP_STRING` 으로 지정해주어야한다.

가변 데이터형(문자 및 이진 데이터) 입력

가변 데이터 형에는 VARCHAR 및 TEXT 그리고, BLOB과 CLOB이 포함된다. 기존함수에서는 VARCHAR 만이 지원되었고, 또한 스트링의 길이를 사용자가 입력할 수 있는 방법이 없었다. 그런 이유로 매번 strlen() 함수를 통해 길이를 얻어야 했지만, 함수 V2 부터는 사용자가 직접 가변 데이터형에 대한 길이를 지정할 수 있게 되었다. 따라서, 만일 사용자가 그 길이를 미리 알고 있다면, 더 빠르게 데이터를 입력할 수 있다. 내부적으로는 가변 데이터형이 하나의 구조체로 되어 있지만, 개발 편의를 위해 각 데이터타입에 따라 멤버를 별도로 만들어 놓았다.

```
typedef struct machAppendVarStruct
{
    unsigned int mLength;
    void *mData;
} machAppendVarStruct;
```

가변 데이터형의 입력시에는 데이터의 길이를 mLength에 설정하고, 원시 데이터 포인터를 mData로 설정하면 된다. 만일 mLength의 길이가 정의된 스키마보다 클 경우에는 자동으로 잘려서 입력된다. 이때 SQLAppendDataV2() 함수는 **SQL_SUCCESS_WITH_INFO**을 리턴하게 되고, 더불어 관련 경고 메시지를 내부 구조체에 채운다. 이 경고 메시지를 확인하기 위해서는 SQLError() 함수를 이용하면 된다.

데이터베이스 타입	NULL 매크로	SQL_APPEND_PARAM 멤버 (mVar를 사용해도 무방함)
VARCHAR	SQL_APPEND_VARCHAR_NULL	mVarchar
TEXT	SQL_APPEND_TEXT_NULL	mText
BINARY	SQL_APPEND_BINARY_NULL	mBinary
BLOB	SQL_APPEND_BLOB_NULL	mBlob
CLOB	SQL_APPEND_CLOB_NULL	mClob

다음은 각각의 환경에 대해 실제 값을 입력하는 예제이다. 하나의 VARCHAR 컬럼이 존재한다고 가정한다.

```
CREATE TABLE ttt (name VARCHAR(10));
```

```
void testAppendVarcharFunc()
{
    SQL_mach_PARAM sParam[1];

    /*  VARCHAR : NULL */
    sParam[0].mVarchar.mLength = SQL_APPEND_VARCHAR_NULL
    SQLAppendDataV2(Stmt, sParam); /* OK */

    /*  VARCHAR : string */
    strcpy(sVarchar, "MY VARCHAR");
    sParam[0].mVarchar.mLength = strlen(sVarchar);
    sParam[0].mVarchar.mData   = sVarchar;
    SQLAppendDataV2(Stmt, sParam); /* OK */

    /*  VARCHAR : Truncation! */
    strcpy(sVarchar, "MY VARCHAR9"); /* Truncation! */
    sParam[0].mVarchar.mLength = strlen(sVarchar);
    sParam[0].mVarchar.mData   = sVarchar;
    SQLAppendDataV2(Stmt, sParam);  /* SQL_SUCCESS_WITH_INFO */
}
```

다음은 Text 타입에 대한 입력 예제이다.

```
CREATE TABLE ttt (doc TEXT);
```

```
void testAppendFunc()
```

```
{
    SQL_mach_PARAM sParam[1];

    /*  VARCHAR : NULL */
    sParam[0].mText.mLength = SQL_APPEND_TEXT_NULL
    SQLAppendDataV2(Stmt, sParam); /* OK */

    /*  VARCHAR : string */
    strcpy(sText, "This is the sample document for tutorial.");
    sParam[0].mVar.mLength = strlen(sText);
    sParam[0].mVar.mData   = sText;
    SQLAppendDataV2(Stmt, sParam); /* OK */
}
```

SQLAppendDataByTimeV2

```
SQLRETURN  SQLAppendDataByTimeV2(SQLHSTMT StatementHandle, SQLBIGINT aTime, SQL_APPEND_PARAM  *aData);
```

이 함수는 해당 채널에 대해 데이터를 입력하는 함수이며, DB에 저장되는 _arrival_time 값을 현재 시간이 아닌 특정 시간의 값으로 설정할 수 있다. 예를 들면, 1개월전 로그 파일에 있는 날짜를 그 당시의 날짜로 입력하고자 할때 사용된다.

- aTime은 _arrival_time으로 설정될 time값이다. *1970년 1월 1일 이후로부터의 현재까지 nano second 값을 입력해야 한다. 또한 입력되는 값이 과거부터 현재순으로 순차적으로 정렬되어 있어야 한다.*
- aData는 입력될 데이터의 포인터를 담고 있는 배열이다. 배열의 개수는 Open시에 지정한 테이블이 보유하고 있는 컬럼의 개수와 일치해야 한다.

나머지 사항은 SQLAppendDataV2()함수를 참고하여 작성하면 된다.

SQLAppendFlush

```
SQLRETURN SQLAppendFlush(SQLHSTMT StatementHandle);
```

이 함수는 현재 채널 버퍼에 쌓여있는 데이터를 Machbase 서버로 즉시 전송한다.

SQLAppendClose

```
SQLRETURN SQLAppendClose(SQLHSTMT   aStmtHandle,
                         SQLBIGINT* aSuccessCount,
                         SQLBIGINT* aFailureCount);
```

이 함수는 현재 열린 채널을 닫는다. 만일 열려지지 않은 채널이 존재할 경우 에러가 발생한다.

- aSuccessCount : Append를 성공한 레코드 개수 값을 가진다.
- aFailureCount : Append를 실패한 레코드 개수 값을 가진다.

SQLAppendSetErrorCallback

```
SQLRETURN SQLAppendSetErrorCallback(SQLHSTMT aStmtHandle, SQLAppendErrorCallback aFunc);
```

이 함수는 SQLAppendOpen()이 성공한 다음 Append시 에러가 발생했을 때 호출되는 콜백 함수를 설정한다. 만일 이 함수를 설정하지 않을 경우에는 서버에 에러가 발생하더라도, 클라이언트에서는 무시하게 된다.

- aStmtHandle : 에러를 확인할 Statement를 지정한다.
- aFunc : Append 실패시 호출할 함수 포인터를 지정한다.

SQLAppendErrorCallback의 프로토타입은 다음과 같다.

```
typedef void (*SQLAppendErrorCallback)(SQLHSTMT aStmtHandle,
                                       SQLINTEGER aErrorCode,
                                       SQLPOINTER aErrorMessage,
                                           SQLLEN aErrorBufLen,
                                       SQLPOINTER aRowBuf,
                                           SQLLEN aRowBufLen);
```

- aStatementHandle : 에러를 발생한 Statement 핸들
- aErrorCode : 에러의 원인이 된 32비트 에러 코드
- aErrorMessage : 해당 에러코드에 대한 문자열
- aErrorBufLen : aErrorMessage의 길이
- aRowBuf : 에러를 발생시킨 레코드의 상세 명세가 담긴 문자열
- aRowBufLen : aRowBuf의 길이

에러 콜백(dumpError)의 사용 예

```
void dumpError(SQLHSTMT    aStmtHandle,
               SQLINTEGER  aErrorCode,
               SQLPOINTER  aErrorMessage,
               SQLLEN      aErrorBufLen,
               SQLPOINTER  aRowBuf,
               SQLLEN      aRowBufLen)
{
    char       sErrMsg[1024] = {0, };
    char       sRowMsg[32 * 1024] = {0, };

    if (aErrorMessage != NULL)
    {
        strncpy(sErrMsg, (char *)aErrorMessage, aErrorBufLen);
    }

    if (aRowBuf != NULL)
    {
        strncpy(sRowMsg, (char *)aRowBuf, aRowBufLen);
    }

    fprintf(stdout, "Append Error : [%d][%s]\n[%s]\n\n", aErrorCode, sErrMsg, sRowMsg);
}


......

    if( SQLAppendOpen(m_IStmt, TableName, aErrorCheckCount) != SQL_SUCCESS )
    {
        fprintf(stdout, "SQLAppendOpen error\n");
        exit(-1);
    }
    // 콜백을 설정한다.
    assert(SQLAppendSetErrorCallback(m_IStmt, dumpError) == SQL_SUCCESS);

    doAppend(sMaxAppend);

    if( SQLAppendClose(m_IStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        fprintf(stdout, "SQLAppendClose error\n");
        exit(-1);
    }
}
```

SQLSetConnectAppendFlush

```
SQLRETURN SQL_API SQLSetConnectAppendFlush(SQLHDBC hdbc, SQLINTEGER option)
```

Append에 의해서 입력된 데이터는 통신 버퍼에 기록되어 전송대기 상태에서 사용자가 SQLAppendFlush 함수를 호출하거나 통신 버퍼가 가득 차게 되면 서버로 전송된다. 사용자가 버퍼가 가득 차 있지 않아도 일정 주기로 서버에게 Append에 의한 데이터를 전송하게 하려면 이 함수를 이용하면 된다. 이 함수는 매 100ms 주기로 마지막으로 전송한 시간과 현재 시간의 차이를 계산하여 지정된 시간(설정하지 않은 경우에는 1초)가 지난 경우 통신 버퍼의 내용을 서버에 전달한다.

매개변수는 다음과 같다.

- hdbc : DB의 connection handle이다.
- option : 0이면 auto flush를 off, 0이 아닌 값이면 auto flush를 on으로 한다.

연결되지 않은 hdbc에 대해서 실행하면 오류로 처리된다.

SQLSetStmtAppendInterval

```
SQLRETURN SQL_API SQLSetStmtAppendInterval(SQLHSTMT hstmt, SQLINTEGER fValue)
```

SQLSetConnectAppendFlush를 이용해서 시간 단위 flush기능을 켰을 경우, 특정 statement에 대해서는 자동 flush를 끄거나 flush 주기를 조정하고 싶을 경우 이 함수를 사용한다.

매개변수는 다음과 같다.

- hstmt : flush주기를 조정하고자 하는 statement handle이다.
- fValue : flush주기를 조정하고자 하는 값이다. **0이면 flush를 하지 않으며 단위는 ms이다**. 100ms마다 flush할지를 결정하는 스레드가 실행되므로 100의 배수로 설정한다. 정확히 원하는 시점에 자동 flush가 실행되지는 않는다. **1000이 기본 값**이다.

시간 기반 flush가 실행중이지 않은 경우라도 이 함수의 실행은 성공한다.

Error 확인 및 설명

Append 관련 함수를 사용할때 에러를 확인하는 방법과 코드에 대한 설명이다. CLI 함수에서 return 값이 SQL_SUCCESS가 아닌 경우 아래 코드를 이용하여 에러 메시지를 확인할 수 있다.

```
SQLINTEGER errNo;
int msgLength;
char sqlState[6];
char errMsg[1024];

if (SQL_SUCCESS == SQLError ( env, con, stmt, (SQLCHAR *)sqlState, &errNo,
                              (SQLCHAR *)errMsg, 1024, &msgLength ))
{
    //error code값을 5자리 숫자로 지정한다.
    printf("ERROR-%05d: %s\n", errNo, errMsg);
}
```

Append관련 함수에서 리턴되는 에러 메시지는 아래와 같다.

함수	message	설명
SQLAppendOpen	statement is already opened.	중복으로 SQLAppendOpen을 하는 경우 발생함.
	Failed to close stream protocol.	스트림 프로토콜 종료에 실패함.
	Failed to read protocol.	네트워크 읽기 오류가 발생함.
	cannot read column meta.	column meta 정보 구조가 잘못됨
	cannot allocate memory.	내부 버퍼 메모리 할당 오류가 발생함.
	cannot allocate compress memory.	압축 버퍼 메모리 할당 오류가 발생함.
	invalid return after reading column meta.	return값에 오류가 있음.
SQLAppendData	statement is not opened.	AppendOpen을 하지 않고 AppendData를 call함.
	column() truncated :	varchar 타입 컬럼에 지정된 사이즈 보다 큰 데이터를 입력하는 경우 발생함.
	Failed to add binary.	통신버퍼에 쓰기 오류가 발생함.
SQLAppendClose	statement is not opened.	AppendOpen상태가 아님.
	Failed to close stream protocol.	스트림 프로토콜 종료에 실패함.
	Failed to close buffer protocol.	버퍼 프로토콜 종료에 실패함.
	cannot read column meta.	column meta정보 구조가 잘못됨.
	invalid return after reading column meta.	return값에 오류가 있음.
SQLAppendFlush	statement is not opened.	AppendOpen상태가 아님
	Failed to close stream protocol.	네트워크 쓰기 오류가 발생함.
SQLSetErrorCallback	statement is not opened.	AppendOpen상태가 아님.
	Protocol Error (not APPEND_DATA_PROTOCOL)	통신버퍼 읽기 결과 APPEND_DATA_PROTOCOL 값이 아님.
SQLAppendDataV2	Invalid date format or date string.	datetime 형식이 틀릴 경우 발생함.
	statement is not opened.	AppendOpen상태가 아님
	column() truncated :	binary 타입 컬럼에 지정된 사이즈 보다 큰 데이터를 입력하는 경우 발생함.
	column() truncated :	varchar, text 타입 컬럼에 지정된 사이즈 보다 큰 데이터를 입력하는 경우 발생함.

함수	message	설명
	Failed to add stream.	통신 버퍼에 쓰기 오류가 발생함.
	IP address length is invalid.	IPv4, IPv6 타입 구조체의 mLength 값이 잘못 지정됨.
	IP string is invalid.	IPv4, IPv6 형식의 데이터가 아님.
	Unknown data type has been specified.	Machbase에서 사용하는 data type이 아님.

열 형식 매개변수 바인딩

이를 위해서 Machbase 5.5 이후 버전에서는 열 형색 매개변수 바인딩을 지원한다. (행 형식 매개변수 바인딩은 아직 지원되지 않는다.)

함수 SQLSetStmtAttr()의 인자 Attribute에 SQL_ATTR_PARAM_BIND_TYPE을 설정하고 인자 param에 SQL_PARAM_BIND_BY_COLUMN을 설정한다. 바인드할 각 칼럼에 대해서 매개변수를 배열로 설정하고, 지시자 변수 또한 배열로 설정한다. 이후 SQLBindParameter()를 이 매개변수를 전달하여 호출한다.

아래 그림은 각 매개변수 배열에 대해 열 형식 바인딩이 동작하는 방식을 보여준다.

Column A (parameter A)		Column B (parameter B)		Column C (parameter C)	
Value_Array	Indicator/ length array	Value_Array	Indicator/ length array	Value_Array	Indicator/ length array

아래 예제는 열 형식 매개변수 바인딩을 이용하여 대량의 데이터를 삽입하는 예제이다.

```
#define DESC_LEN 51
#define ARRAY_SIZE 10
SQLCHAR * Statement = "INSERT INTO Parts (PartID, Description, Price) VALUES (?, ?, ?)";

/* 바인드할 매개변수 배열 */
SQLUINTEGER PartIDArray[ARRAY_SIZE];
SQLCHAR DescArray[ARRAY_SIZE][DESC_LEN];
SQLREAL PriceArray[ARRAY_SIZE];
/* 바인드할 지사자 변수 배열 */
SQLINTEGER PartIDIndArray[ARRAY_SIZE], DescLenOrIndArray[ARRAY_SIZE], PriceIndArray[ARRAY_SIZE];
SQLUSMALLINT i, ParamStatusArray[ARRAY_SIZE];
SQLUINTEGER ParamsProcessed;

// Set the SQL_ATTR_PARAM_BIND_TYPE statement attribute to use
// column-wise binding.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_BIND_TYPE, SQL_PARAM_BIND_BY_COLUMN, 0);
// Specify the number of elements in each parameter array.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);
// Specify an array in which to return the status of each set of
// parameters.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);
// Specify an SQLUINTEGER value in which to return the number of sets of
// parameters processed.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);
// Bind the parameters in column-wise fashion.
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG, SQL_INTEGER, 5, 0,
    PartIDArray, 0, PartIDIndArray);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, DESC_LEN - 1, 0,
    DescArray, DESC_LEN, DescLenOrIndArray);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_REAL, 7, 0,
    PriceArray, 0, PriceIndArray);
```

지원되는 문자열

마크베이스는 기본적으로 UTF-8 방식을 사용하여 문자열 데이터를 저장한다.

UTF-8 이외의 방식으로 문자열을 입/출력하는 Windows의 경우 ODBC에서 아래와 같이 변환한다.

OS	Unicode/Non-Unicode	문자열 변환	Note
Windows	Unicode (UTF-16)	UTF-16 ⟷ UTF-8	N/A
Windows	Non-Unicode (MBCS)	MBCS ⟷ UTF-8	Windows 설정의 Non-Unicode 어플리케이션의 기본 문자열을 사용함
Linux	UTF-8	N/A	UTF-8 만 지원됨

CLI/ODBC 예제

응용 프로그램 개발

목차

CLI 설치 확인

마크베이스가 설치된 디렉터리의 include 및 lib에 다음과 같은 파일이 있으면 응용 프로그램을 개발할 수 있는 환경이 완비된 것이다.

```
Mach@localhost:~/machbase_home$ ls -l include lib install/
include:
total 176
-rwxrwxr-x 1 mach mach 31449 Jun 18 19:26 machbase_sqlcli.h

install/:
total 12
-rw-rw-r-- 1 mach mach 1667 Jun 18 19:26 machbase_env.mk

lib:
total 16196
-rw-rw-r-- 1 mach mach  78603 Jun 18 19:26 machbase.jar
-rw-rw-r-- 1 mach mach 964290 Jun 18 19:26 libmachbasecli.a
```

Makefile 작성 가이드

```
mach@localhost:~/machbase_home$ cd sample/
mach@localhost:~/machbase_home/sample$ cd cli/
mach@localhost:~/machbase_home/sample/cli$ ls
Makefile sample1_connect.c
```

마크베이스 패키지를 설치했다면, 다음 경로에 샘플 프로그램이 설치되어 있을 것이다.

```
include $(MACHBASE_HOME)/install/machbase_env.mk
INCLUDES += $(LIBDIR_OPT)/$(MACHBASE_HOME)/include

all : sample1_connect

sample1_connect : sample1_connect.o
$(LD_CC) $(LD_FLAGS) $(LD_OUT_OPT)$@ $< $(LIB_OPT)machbasecli$(LIB_AFT) $(LIBDIR_OPT)$(MACHBASE_HOME)/lib $(LD_LIBS

sample1_connect.o : sample1_connect.c
$(COMPILE.cc) $(CC_FLAGS) $(INCLUDES) $(CC_OUT_OPT)$@ $<

clean :
rm -f sample1_connect
```

컴파일 및 링크

주어진 샘플에 대해 다음과 같이 수행하면 실행 파일이 만들어진다.

```
mach@localhost:~/machbase_home/sample/cli$ make
gcc -c -g -W -Wall -rdynamic -O3 -finline-functions -fno-omit-frame-pointer -fno-strict-aliasing -m64 -mtune=k8 -g
gcc -m64 -mtune=k8 -L/home/machbase/machbase_home/lib -osample1_connect sample1_connect.o -lmachbasecli -L/home/mac
mach@localhost:~/machbase_home/sample/cli$ ls -al
total 1196
drwxrwxr-x 2 mach mach 4096 Jun 18 20:15 .
drwxrwxr-x 4 mach mach 4096 Jun 18 19:26 ..
-rw-rw-r-- 1 mach mach 483 Jun 18 19:26 Makefile
```

```
-rwxrwxr-x 1 mach mach 1196943 Jun 18 20:15 sample1_connect
-rw-rw-r-- 1 mach mach 549 Jun 18 19:26 sample1_connect.c
-rw-rw-r-- 1 mach mach 8168 Jun 18 20:15 sample1_connect.o
```

ⓘ 필요에 따라 얼마든지 위의 샘플 Makefile을 수정하여 응용 프로그램을 작성할 수 있을 것이다.

샘플 프로그램

접속 예제

CLI를 이용하여 접속하는 예제 프로그램을 작성해 보기로 한다.

샘플 파일명은 sample1_connect.c 로 한다.

MACHBASE_PORT_NO는 $MACHBASE_HOME/conf/machbase.conf 파일에 있는 PORT_NO 값과 같아야 한다.

› sample1_connect.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <machbase_sqlcli.h>

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;

void connectDB()
{
    char connStr[1024];
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }
    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }
    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);
    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");
        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
                                      errMsg, 1024, &msgLength ))
        {
            printf("mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }
    printf("connected ... \n");
}

void disconnectDB()
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
```

```
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon))
    {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                     errMsg, 1024, &msgLength ))
        {
            printf("mach-%d : %s\n", errNo, errMsg);
        }
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

int main()
{
    connectDB();
    disconnectDB();
    return 0;
}
```

Makefile에 sample1_connect.c를 등록하고 컴파일하여 실행하면 다음과 같이 나온다.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample1_connect
connected ...
```

데이터 입력 및 출력 예제

아래의 예제 소스에서는 CREATE TABLE 구문을 이용하여 테이블을 생성하고, 간단한 데이터 값들을 임의로 생성해서 INSERT 구문을 사용해서 데이터를 입력하고, SELECT 구문을 이용하여 데이터를 출력한다. 이를 활용하여 직접 값을 입력하고 확인할 때 각 타입별로 어떻게 설정을 해야 하는지 알수 있을 것이다.

샘플 파일명은 sample2_insert.c 라고 한다.

› sampe2_insert.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <machbase_sqlcli.h>

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
    char connStr[1024];
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];
    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }
    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }
```

```
    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);
    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");
        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
                                      errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }
    printf("connected ... \n");
}

void disconnectDB()
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];
    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");
        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                     errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
    }
    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT stmt)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];
    printf("ERROR : (%s)\n", aMsg);
    if (SQL_SUCCESS == SQLError( gEnv, gCon, stmt, NULL, &errNo,
                                 errMsg, 1024, &msgLength ))
    {
        printf(" mach-%d : %s\n", errNo, errMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT stmt;
    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", stmt);
    }
    if (SQLExecDirect(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", stmt);
    }
    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", stmt);
    }
}
```

```
void prepareExecuteSQL(const char *aSQL)
{
    SQLHSTMT stmt;
    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        outError("AllocStmt error", stmt);
    }
    if (SQLPrepare(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        printf("Prepare error[%s]\n", aSQL);
        outError("Prepare error", stmt);
    }
    if (SQLExecute(stmt) == SQL_ERROR)
    {
        outError("prepared execute error", stmt);
    }
    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        outError("FreeStmt Error", stmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE1", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE1(seq short, score integer, total long, percentage float, ratio dou
}

void selectTable()
{
    SQLHSTMT stmt;
    const char *aSQL = "SELECT seq, score, total, percentage, ratio, id, srcip, dstip, reg_date, textlog, image
    int i=0;
    SQLLEN Len = 0;
    short seq;
    int score;
    long total;
    float percentage;
    double ratio;
    char id [11];
    char srcip[16];
    char dstip[40];
    SQL_TIMESTAMP_STRUCT regdate;
    char log [1024];
    char image[1024];
    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR) {
        outError("AllocStmt Error", stmt);
    }
    if (SQLPrepare(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR) {
        printf("Prepare error[%s] \n", aSQL);
        outError("Prepare error", stmt);
    }
    if (SQLExecute(stmt) == SQL_ERROR) {
        outError("prepared execute error", stmt);
    }
    SQLBindCol(stmt, 1, SQL_C_SHORT, &seq, 0, &Len);
    SQLBindCol(stmt, 2, SQL_C_LONG, &score, 0, &Len);
    SQLBindCol(stmt, 3, SQL_C_BIGINT, &total, 0, &Len);
    SQLBindCol(stmt, 4, SQL_C_FLOAT, &percentage, 0, &Len);
    SQLBindCol(stmt, 5, SQL_C_DOUBLE, &ratio, 0, &Len);
    SQLBindCol(stmt, 6, SQL_C_CHAR, id, sizeof(id), &Len);
    SQLBindCol(stmt, 7, SQL_C_CHAR, srcip, sizeof(srcip), &Len);
    SQLBindCol(stmt, 8, SQL_C_CHAR, dstip, sizeof(dstip), &Len);
    SQLBindCol(stmt, 9, SQL_C_TYPE_TIMESTAMP, &regdate, 0, &Len);
    SQLBindCol(stmt, 10, SQL_C_CHAR, log, sizeof(log), &Len);
    SQLBindCol(stmt, 11, SQL_C_CHAR, image, sizeof(image), &Len);
    while (SQLFetch(stmt) == SQL_SUCCESS)
    {
        printf("===== %d ========\n", i++);
```

```
        printf("seq = %d", seq);
        printf(", score = %d", score);
        printf(", total = %ld", total);
        printf(", percentage = %.2f", percentage);
        printf(", ratio = %g", ratio);
        printf(", id = %s", id);
        printf(", srcip = %s", srcip);
        printf(", dstip = %s", dstip);
        printf(", regdate = %d-%02d-%02d %02d:%02d:%02d",
               regdate.year, regdate.month, regdate.day,
               regdate.hour, regdate.minute, regdate.second);
        printf(", log = %s", log);
        printf(", image = %s\n", image);
    }
    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        outError("FreeStmt eror", stmt);
    }
}

void directInsert()
{
    int i;
    char query[2 * 1024];
    short seq;
    int score;
    long total;
    float percentage;
    double ratio;
    char id [11];
    char srcip [16];
    char dstip [40];
    char reg_date [40];
    char log [1024];
    char image [1024];
    for(i=1; i<10; i++)
    {
        seq = i;
        score = i+i;
        total = (seq + score) * 10000;
        percentage = (float)score/total;
        ratio = (double)seq/total;
        sprintf(id, "id-%d", i);
        sprintf(srcip, "192.168.0.%d", i);
        sprintf(dstip, "2001:0DB8:0000:0000:0000:0000:1428:%04d", i);
        sprintf(reg_date, "2015-03-31 15:26:%02d", i);
        sprintf(log, "text log-%d", i);
        sprintf(image, "binary image-%d", i);
        memset(query, 0x00, sizeof(query));
        sprintf(query, "INSERT INTO CLI_SAMPLE1 VALUES(%d, %d, %ld, %f, %f, '%s', '%s', '%s',TO_DATE('%s','YYYY-
                seq, score, total, percentage, ratio, id, srcip, dstip, reg_date, log, image);
        prepareExecuteSQL(query);
        printf("%d record inserted\n", i);
    }
}

int main()
{
    connectDB();
    createTable();
    directInsert();
    selectTable();
    disconnectDB();
    return 0;
}
```

Makefile에 sample2_insert.c를 등록하고 컴파일하여 실행하면 다음과 같이 나온다.

```
[mach@localhost cli]$ make
```

```
[mach@localhost cli]$ ./sample2_insert

connected ...
1 record inserted
2 record inserted
3 record inserted
4 record inserted
5 record inserted
6 record inserted
7 record inserted
8 record inserted
9 record inserted
===== 0 ========
seq = 9, score = 18, total = 270000, percentage = 0.00, ratio = 3.3e-05, id = id-9, srcip = 192.168.0.9, dstip = 20
===== 1 ========
seq = 8, score = 16, total = 240000, percentage = 0.00, ratio = 3.3e-05, id = id-8, srcip = 192.168.0.8, dstip = 20
===== 2 ========
seq = 7, score = 14, total = 210000, percentage = 0.00, ratio = 3.3e-05, id = id-7, srcip = 192.168.0.7, dstip = 20
===== 3 ========
seq = 6, score = 12, total = 180000, percentage = 0.00, ratio = 3.3e-05, id = id-6, srcip = 192.168.0.6, dstip = 20
===== 4 ========
seq = 5, score = 10, total = 150000, percentage = 0.00, ratio = 3.3e-05, id = id-5, srcip = 192.168.0.5, dstip = 20
===== 5 ========
seq = 4, score = 8, total = 120000, percentage = 0.00, ratio = 3.3e-05, id = id-4, srcip = 192.168.0.4, dstip = 200
===== 6 ========
seq = 3, score = 6, total = 90000, percentage = 0.00, ratio = 3.3e-05, id = id-3, srcip = 192.168.0.3, dstip = 2001
===== 7 ========
seq = 2, score = 4, total = 60000, percentage = 0.00, ratio = 3.3e-05, id = id-2, srcip = 192.168.0.2, dstip = 2001
===== 8 ========
seq = 1, score = 2, total = 30000, percentage = 0.00, ratio = 3.3e-05, id = id-1, srcip = 192.168.0.1, dstip = 2001
```

Prepare Execute 예제

데이터를 binding하여 INSERT하는 예제 프로그램을 작성해 보자.

마크베이스에서 데이터를 binding 하는 방식으로 값을 입력할수 있는데 이를 이용할시에는 데이터의 값들의 타입들을 명확히 지정해주고, 긴 문자열 타입들의 경우에는 길이 값을 반드시 지정해줘야 한다.

아래의 예제를 통해서 각 타입별로 데이터를 binding하는 방법을 알수 있다.

파일명은 sample3_prepare.c 라고 한다.

› sample3_prepare.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <machbase_sqlcli.h>
#include <time.h>

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
    char sConnStr[1024];

    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }
```

```
    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)sConnStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");

        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &sErrorNo,
                                      sErrorMsg, 1024, &sMsgLength ))
        {
            printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }

    printf("connected ... \n");

}

void disconnectDB()
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &sErrorNo,
                                     sErrorMsg, 1024, &sMsgLength ))
        {
            printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
        }
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT aStmt)
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, aStmt, NULL, &sErrorNo,
                                 sErrorMsg, 1024, &sMsgLength ))
    {
        printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt;
```

```
    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", sStmt);
    }

    if (SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", sStmt);
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", sStmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float, ratio doub
}

void selectTable()
{
    SQLHSTMT sStmt;
    const char *aSQL = "SELECT seq, score, total, percentage, ratio, id, srcip, dstip, reg_date, tlog, image FRO

    int i=0;
    short sSeq;
    int sScore;
    long sTotal;
    float sPercentage;
    double sRatio;
    char sId [20];
    char sSrcIp[20];
    char sDstIp[50];
    SQL_TIMESTAMP_STRUCT sRegDate;
    char sLog [1024];
    char sImage[1024];
    SQL_LEN sLen;

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR) {
        outError("AllocStmt Error", sStmt);
    }

    if (SQLPrepare(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR) {
        printf("Prepare error[%s] \n", aSQL);
        outError("Prepare error", sStmt);
    }

    if (SQLExecute(sStmt) == SQL_ERROR) {
        outError("prepared execute error", sStmt);
    }

    SQLBindCol(sStmt, 1, SQL_C_SSHORT, &sSeq, 0, &sLen);
    SQLBindCol(sStmt, 2, SQL_C_SLONG, &sScore, 0, &sLen);
    SQLBindCol(sStmt, 3, SQL_C_SBIGINT, &sTotal, 0, &sLen);
    SQLBindCol(sStmt, 4, SQL_C_FLOAT, &sPercentage, 0, &sLen);
    SQLBindCol(sStmt, 5, SQL_C_DOUBLE, &sRatio, 0, &sLen);
    SQLBindCol(sStmt, 6, SQL_C_CHAR, sId, sizeof(sId), &sLen);
    SQLBindCol(sStmt, 7, SQL_C_CHAR, sSrcIp, sizeof(sSrcIp), &sLen);
    SQLBindCol(sStmt, 8, SQL_C_CHAR, sDstIp, sizeof(sDstIp), &sLen);
    SQLBindCol(sStmt, 9, SQL_C_TYPE_TIMESTAMP, &sRegDate, 0, &sLen);
    SQLBindCol(sStmt, 10, SQL_C_CHAR, sLog, sizeof(sLog), &sLen);
    SQLBindCol(sStmt, 11, SQL_C_CHAR, sImage, sizeof(sImage), &sLen);
```

```
    while (SQLFetch(sStmt) == SQL_SUCCESS)
    {
        printf("===== %d ========\n", i++);
        printf("seq = %d", sSeq);
        printf(", score = %d", sScore);
        printf(", total = %ld", sTotal);
        printf(", percentage = %.2f", sPercentage);
        printf(", ratio = %g", sRatio);
        printf(", id = %s", sId);
        printf(", srcip = %s", sSrcIp);
        printf(", dstip = %s", sDstIp);
        printf(", regdate = %d-%02d-%02d %02d:%02d:%02d",
               sRegDate.year, sRegDate.month, sRegDate.day,
               sRegDate.hour, sRegDate.minute, sRegDate.second);
        printf(", log = %s", sLog);
        printf(", image = %s\n", sImage);
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
        outError("FreeStmt eror", sStmt);
    }
}

void prepareInsert()
{
    SQLHSTMT sStmt;
    int i;
    short sSeq;
    int sScore;
    long sTotal;
    float sPercentage;
    double sRatio;
    char sId [20];
    char sSrcIp [20];
    char sDstIp [50];
    long reg_date;
    char sLog [100];
    char sImage [100];
    int sLength[5];

    const char *sSQL = "INSERT INTO CLI_SAMPLE VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )";

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
        outError("AllocStmt error", sStmt);
    }

    if (SQLPrepare(sStmt, (SQLCHAR *)sSQL, SQL_NTS) == SQL_ERROR)
    {
        printf("Prepare error[%s]\n", sSQL);
        outError("Prepare error", sStmt);
    }

    for(i=1; i<10; i++)
    {
        sSeq = i;
        sScore = i+i;
        sTotal = (sSeq + sScore) * 10000;
        sPercentage = (float)(sScore+2)/sScore;
        sRatio = (double)(sSeq+1)/sTotal;
        sprintf(sId, "id-%d", i);
        sprintf(sSrcIp, "192.168.0.%d", i);
        sprintf(sDstIp, "2001:0DB8:0000:0000:0000:0000:1428:%04x", i);
        reg_date = i*10000;
        sprintf(sLog, "log-%d", i);
        sprintf(sImage, "image-%d", i);

        if (SQLBindParameter(sStmt,
                             1,
```

```
                    SQL_PARAM_INPUT,
                    SQL_C_SSHORT,
                    SQL_SMALLINT,
                    0,
                    0,
                    &sSeq,
                    0,
                    NULL) == SQL_ERROR)
{
    outError("BindParameter error 1", sStmt);
}

if (SQLBindParameter(sStmt,
                    2,
                    SQL_PARAM_INPUT,
                    SQL_C_SLONG,
                    SQL_INTEGER,
                    0,
                    0,
                    &sScore,
                    0,
                    NULL) == SQL_ERROR)
{
    outError("BindParameter error 2", sStmt);
}

if (SQLBindParameter(sStmt,
                    3,
                    SQL_PARAM_INPUT,
                    SQL_C_SBIGINT,
                    SQL_BIGINT,
                    0,
                    0,
                    &sTotal,
                    0,
                    NULL) == SQL_ERROR)
{
    outError("BindParameter error 3", sStmt);
}

if (SQLBindParameter(sStmt,
                    4,
                    SQL_PARAM_INPUT,
                    SQL_C_FLOAT,
                    SQL_FLOAT,
                    0,
                    0,
                    &sPercentage,
                    0,
                    NULL) == SQL_ERROR)
{
    outError("BindParameter error 4", sStmt);
}

if (SQLBindParameter(sStmt,
                    5,
                    SQL_PARAM_INPUT,
                    SQL_C_DOUBLE,
                    SQL_DOUBLE,
                    0,
                    0,
                    &sRatio,
                    0,
                    NULL) == SQL_ERROR)
{
    outError("BindParameter error 5", sStmt);
}

sLength[0] = strlen(sId);
if (SQLBindParameter(sStmt,
```

```
                    6,
                    SQL_PARAM_INPUT,
                    SQL_C_CHAR,
                    SQL_VARCHAR,
                    0,
                    0,
                    sId,
                    0,
                    (SQLLEN *)&sLength[0]) == SQL_ERROR)
{
    outError("BindParameter error 6", sStmt);
}

sLength[1] = strlen(sSrcIp);
if (SQLBindParameter(sStmt,
                    7,
                    SQL_PARAM_INPUT,
                    SQL_C_CHAR,
                    SQL_IPV4,
                    0,
                    0,
                    sSrcIp,
                    0,
                    (SQLLEN *)&sLength[1]) == SQL_ERROR)
{
    outError("BindParameter error 7", sStmt);
}

sLength[2] = strlen(sDstIp);
if (SQLBindParameter(sStmt,
                    8,
                    SQL_PARAM_INPUT,
                    SQL_C_CHAR,
                    SQL_IPV6,
                    0,
                    0,
                    sDstIp,
                    0,
                    (SQLLEN *)&sLength[2]) == SQL_ERROR)
{
    outError("BindParameter error 8", sStmt);
}

if (SQLBindParameter(sStmt,
                    9,
                    SQL_PARAM_INPUT,
                    SQL_C_SBIGINT,
                    SQL_DATE,
                    0,
                    0,
                    &reg_date,
                    0,
                    NULL) == SQL_ERROR)
{
    outError("BindParameter error 9", sStmt);
}

sLength[3] = strlen(sLog);
if (SQLBindParameter(sStmt,
                    10,
                    SQL_PARAM_INPUT,
                    SQL_C_CHAR,
                    SQL_VARCHAR,
                    0,
                    0,
                    sLog,
                    0,
                    (SQLLEN *)&sLength[3]) == SQL_ERROR)
{
    outError("BindParameter error 10", sStmt);
```

```
        }

        sLength[4] = strlen(sImage);
        if (SQLBindParameter(sStmt,
                             11,
                             SQL_PARAM_INPUT,
                             SQL_C_CHAR,
                             SQL_BINARY,
                             0,
                             0,
                             sImage,
                             0,
                             (SQLLEN *)&sLength[4]) == SQL_ERROR)
        {
            outError("BindParameter error 11", sStmt);
        }

        if( SQLExecute(sStmt) == SQL_ERROR) {
            outError("prepare execute error", sStmt);
        }

        printf("%d prepared record inserted\n", i);

    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP)) {
        outError("FreeStmt", sStmt);
    }
}

int main()
{
    connectDB();
    createTable();
    prepareInsert();
    selectTable();
    disconnectDB();

    return 0;
}
```

Makefile에 sample3_prepare.c를 등록하고 컴파일하여 실행하면 다음과 같이 나온다.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample3_prepare

connected ...
1 prepared record inserted
2 prepared record inserted
3 prepared record inserted
4 prepared record inserted
5 prepared record inserted
6 prepared record inserted
7 prepared record inserted
8 prepared record inserted
9 prepared record inserted
===== 0 ========
seq = 9, score = 18, total = 270000, percentage = 1.11, ratio = 3.7037e-05, id = id-9, srcip = 192.168.0.9, dstip =
===== 1 ========
seq = 8, score = 16, total = 240000, percentage = 1.12, ratio = 3.75e-05, id = id-8, srcip = 192.168.0.8, dstip = 2
===== 2 ========
seq = 7, score = 14, total = 210000, percentage = 1.14, ratio = 3.80952e-05, id = id-7, srcip = 192.168.0.7, dstip
===== 3 ========
seq = 6, score = 12, total = 180000, percentage = 1.17, ratio = 3.88889e-05, id = id-6, srcip = 192.168.0.6, dstip
===== 4 ========
seq = 5, score = 10, total = 150000, percentage = 1.20, ratio = 4e-05, id = id-5, srcip = 192.168.0.5, dstip = 200
===== 5 ========
seq = 4, score = 8, total = 120000, percentage = 1.25, ratio = 4.16667e-05, id = id-4, srcip = 192.168.0.4, dstip =
===== 6 ========
```

```
seq = 3, score = 6, total = 90000, percentage = 1.33, ratio = 4.44444e-05, id = id-3, srcip = 192.168.0.3, dstip =
===== 7 ========
seq = 2, score = 4, total = 60000, percentage = 1.50, ratio = 5e-05, id = id-2, srcip = 192.168.0.2, dstip = 2001:0
===== 8 ========
seq = 1, score = 2, total = 30000, percentage = 2.00, ratio = 6.66667e-05, id = id-1, srcip = 192.168.0.1, dstip =
```

확장 함수 Append 예제

마크베이스에서는 대량의 데이터를 파일로부터 읽어서 고속으로 입력하는 방법으로 Append 프로토콜을 제공하고 있다. 이 Append 프로토콜을 이용하는 예제 프로그램을 작성해 보자.

먼저 마크베이스에서 제공하는 다양한 타입별로 append 하는 방식의 예제를 살펴보자. Append 방식은 각 타입별로 편리하게 입력해 줄 수 있도록 각각의 설정값들이 정해져있다. 그러므로 모든 방법별로 사용하는 입력하는 방식에 대한 숙지를 한다면 더욱더 효율적으로 프로그램을 작성할 수 있을 것이다. 아래쪽에 있는 예제 코드에 그 방법들이 모두 나와있다.

파일명은 sample4_append1.c 라고 한다.

› sample4_append1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <machbase_sqlcli.h>
#include <arpa/inet.h>

#if __linux__
#include <sys/time.h>
#endif

#if defined(SUPPORT_STRUCT_TM)
# include <time.h>
#endif

#define MACHBASE_PORT_NO 5656
#define MAX_APPEND_COUNT 0xFFFFFFFF
#define ERROR_CHECK_COUNT 100

#define ERROR -1
#define SUCCESS 0

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB();
void disconnectDB();
void outError(const char *aMsg);
void executeDirectSQL(const char *aSQL, int aErrIgnore);
void createTable();
void appendOpen();
void appendData();
int appendClose();
time_t getTimeStamp();

int main()
{
    unsigned int sCount=0;
    time_t sStartTime, sEndTime;

    connectDB();
    createTable();

    appendOpen();
    sStartTime = getTimeStamp();
    appendData();
    sEndTime = getTimeStamp();
    appendClose();

    printf("timegap = %ld microseconds for %d records\n", sEndTime - sStartTime, sCount);
```

```
    printf("%.2f records/second\n", ((double)sCount/(double)(sEndTime - sStartTime))*1000000);

    disconnectDB();
    return SUCCESS;
}

void connectDB()
{
    char sConnStr[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        outError("SQLAllocEnv error!!");
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        outError("SQLAllocConnect error!!");
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)sConnStr, SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        outError("connection error\n");
    }

    if (SQL_ERROR == SQLAllocStmt(gCon, &gStmt) )
    {
        outError("AllocStmt error");
    }

    printf("connected ... \n");
}

void disconnectDB()
{
    if( SQL_ERROR == SQLFreeStmt(gStmt, SQL_DROP) )
    {
        outError("SQLFreeStmt error");
    }

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        outError("disconnect error");
    }
    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg)
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    printf("ERROR : (%s)\n", aMsg);
    if (SQL_SUCCESS == SQLError( gEnv, gCon, gStmt, NULL, &sErrorNo,
                                 sErrorMsg, 1024, &sMsgLength ))
    {
        printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
    }

    if( gStmt )
    {
        SQLFreeStmt(gStmt, SQL_DROP);
    }

    if( gCon )
    {
```

```
        SQLFreeConnect( gCon );
    }

    if( gEnv )
    {
        SQLFreeEnv( gEnv );
    }
    exit(ERROR);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt;

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error");
    }

    if (SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error");
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error");
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(short1 short, integer1 integer, long1 long, float1 float, double1
}

void appendOpen()
{
    const char *sTableName = "CLI_SAMPLE";

    if( SQLAppendOpen(gStmt, (SQLCHAR *)sTableName, ERROR_CHECK_COUNT) != SQL_SUCCESS )
    {
        outError("SQLAppendOpen error");
    }

    printf("append open ok\n");
}

void appendData()
{
    SQL_APPEND_PARAM sParam[11];
    char sVarchar[10] = {0, };
    char sText[100] = {0, };
    char sBinary[100] = {0, };

    memset(sParam, 0, sizeof(sParam));

    /* NULL FOR ALL*/
    /* fixed column */
    sParam[0].mShort = SQL_APPEND_SHORT_NULL;
    sParam[1].mInteger = SQL_APPEND_INTEGER_NULL;
    sParam[2].mLong = SQL_APPEND_LONG_NULL;
    sParam[3].mFloat = SQL_APPEND_FLOAT_NULL;
    sParam[4].mDouble = SQL_APPEND_DOUBLE_NULL;
    /* datetime */
    sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_NULL;
```

```
/* varchar */
sParam[6].mVarchar.mLength = SQL_APPEND_VARCHAR_NULL;
/* ipv4 */
sParam[7].mIP.mLength = SQL_APPEND_IP_NULL;
/* ipv6 */
sParam[8].mIP.mLength = SQL_APPEND_IP_NULL;
/* text */
sParam[9].mText.mLength = SQL_APPEND_TEXT_NULL;
/* binary */
sParam[10].mBinary.mLength = SQL_APPEND_BINARY_NULL;
SQLAppendDataV2(gStmt, sParam);

/* FIXED COLUMN Value */
sParam[0].mShort = 2;
sParam[1].mInteger = 4;
sParam[2].mLong = 6;
sParam[3].mFloat = 8.4;
sParam[4].mDouble = 10.9;
SQLAppendDataV2(gStmt, sParam);

/* DATETIME : absolute value */
sParam[5].mDateTime.mTime = MACH_UINT64_LITERAL(1000000000);
SQLAppendDataV2(gStmt, sParam);

/* DATETIME : current */
sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_NOW;
SQLAppendDataV2(gStmt, sParam);

/* DATETIME : string format*/
sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_STRING;
sParam[5].mDateTime.mDateStr = "23/May/2014:17:41:28";
sParam[5].mDateTime.mFormatStr = "DD/MON/YYYY:HH24:MI:SS";
SQLAppendDataV2(gStmt, sParam);

/* DATETIME : struct tm format*/
sParam[5].mDateTime.mTime = SQL_APPEND_DATETIME_STRUCT_TM;
sParam[5].mDateTime.mTM.tm_year = 2000 - 1900;
sParam[5].mDateTime.mTM.tm_mon = 11;
sParam[5].mDateTime.mTM.tm_mday = 31;
SQLAppendDataV2(gStmt, sParam);

/* VARCHAR : string */
strcpy(sVarchar, "MY VARCHAR");
sParam[6].mVar.mLength = strlen(sVarchar);
sParam[6].mVar.mData = sVarchar;
SQLAppendDataV2(gStmt, sParam);

/* IPv4 : ipv4 from binary bytes */
sParam[7].mIP.mLength = SQL_APPEND_IP_IPV4;
sParam[7].mIP.mAddr[0] = 127;
sParam[7].mIP.mAddr[1] = 0;
sParam[7].mIP.mAddr[2] = 0;
sParam[7].mIP.mAddr[3] = 1;
SQLAppendDataV2(gStmt, sParam);

/* IPv4 : ipv4 from binary */
sParam[7].mIP.mLength = SQL_APPEND_IP_IPV4;
*(in_addr_t *)(sParam[7].mIP.mAddr) = inet_addr("192.168.0.1");
SQLAppendDataV2(gStmt, sParam);

/* IPv4 : ipv4 from string */
sParam[7].mIP.mLength = SQL_APPEND_IP_STRING;
sParam[7].mIP.mAddrString = "203.212.222.111";
SQLAppendDataV2(gStmt, sParam);

/* IPv6 : ipv6 from binary bytes */
sParam[8].mIP.mLength = SQL_APPEND_IP_IPV6;
sParam[8].mIP.mAddr[0] = 127;
sParam[8].mIP.mAddr[1] = 127;
sParam[8].mIP.mAddr[2] = 127;
```

```
    sParam[8].mIP.mAddr[3] = 127;
    sParam[8].mIP.mAddr[4] = 127;
    sParam[8].mIP.mAddr[5] = 127;
    sParam[8].mIP.mAddr[6] = 127;
    sParam[8].mIP.mAddr[7] = 127;
    sParam[8].mIP.mAddr[8] = 127;
    sParam[8].mIP.mAddr[9] = 127;
    sParam[8].mIP.mAddr[10] = 127;
    sParam[8].mIP.mAddr[11] = 127;
    sParam[8].mIP.mAddr[12] = 127;
    sParam[8].mIP.mAddr[13] = 127;
    sParam[8].mIP.mAddr[14] = 127;
    sParam[8].mIP.mAddr[15] = 127;
    SQLAppendDataV2(gStmt, sParam);
    sParam[8].mIP.mLength = SQL_APPEND_IP_NULL; /* recover */

    /* TEXT : string */
    memset(sText, 'X', sizeof(sText));
    sParam[9].mVar.mLength = 100;
    sParam[9].mVar.mData = sText;
    SQLAppendDataV2(gStmt, sParam);

    /* BINARY : datas */
    memset(sBinary, 0xFA, sizeof(sBinary));
    sParam[10].mVar.mLength = 100;
    sParam[10].mVar.mData = sBinary;
    SQLAppendDataV2(gStmt, sParam);
}

int appendClose()
{
    int sSuccessCount = 0;
    int sFailureCount = 0;

    if( SQLAppendClose(gStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        outError("SQLAppendClose error");
    }

    printf("append close ok\n");
    printf("success : %d, failure : %d\n", sSuccessCount, sFailureCount);
    return sSuccessCount;
}

time_t getTimeStamp()
{
#if _WIN32 || _WIN64

#if defined(_MSC_VER) || defined(_MSC_EXTENSIONS)
#define DELTA_EPOCH_IN_MICROSECS 11644473600000000Ui64
#else
#define DELTA_EPOCH_IN_MICROSECS 11644473600000000ULL
#endif
    FILETIME sFT;
    unsigned __int64 sTempResult = 0;

    GetSystemTimeAsFileTime(&sFT);

    sTempResult |= sFT.dwHighDateTime;
    sTempResult <<= 32;
    sTempResult |= sFT.dwLowDateTime;

    sTempResult -= DELTA_EPOCH_IN_MICROSECS;
    sTempResult /= 10;

    return sTempResult;
#else
    struct timeval sTimeVal;
    int sRet;
```

```
    sRet = gettimeofday(&sTimeVal, NULL);

    if (sRet == 0)
    {
        return (time_t)(sTimeVal.tv_sec * 1000000 + sTimeVal.tv_usec);
    }
    else
    {
        return 0;
    }
#endif
}
```

Makefile에 sample4_append1.c를 등록하고 컴파일하여 실행하면 다음과 같이 나온다.

```
[mach@localhost cli]$ make sample4_append1
gcc -c -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -I/home/ma
gcc -m64 -mtune=k8 -L/home/mach/machbase_home/lib -osample4_append1 sample4_append1.o -lmachcli -L/home/mach/machba
[mach@localhost cli]$ ./sample4_append1
connected ...
append open ok
append close ok
success : 13, failure : 0
timegap = 48 microseconds for 13 records
270833.33 records/second
[mach@localhost cli]$

You can check what is inserted after MACH_SQL.

Mach> select * from CLI_SAMPLE;
SHORT1 INTEGER1 LONG1 FLOAT1 DOUBLE1
---------------------------------------------------------------------------------------------------------
DATETIME1 VARCHAR1 IP IP2
-----------------------------------------------------------------------------------------------------------------------
TEXT1
-------------------------------------------------------------------------------
BIN1
-------------------------------------------------------------------------------
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 NULL
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
FAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFA
FAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFA
FAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFAFA
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 NULL
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F:7F7F
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 203.212.222.111 NULL
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 192.168.0.1 NULL
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR 127.0.0.1 NULL
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 MY VARCHAR NULL NULL
```

```
NULL
NULL
2 4 6 8.4 10.9
2000-12-31 00:00:00 000:000:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
2014-05-23 17:41:28 000:000:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
2015-04-09 16:44:11 134:256:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
1970-01-01 09:00:01 000:000:000 NULL NULL NULL
NULL
NULL
2 4 6 8.4 10.9
1970-01-01 09:00:00 000:000:000 NULL NULL NULL
NULL
NULL
[12] row(s) selected.
```

이제 파일을 이용해서 고속으로 append하는 방식을 사용해 보자. 실제로 업무에서 사용되는 많은 양의 로그, 패킷등의 값들을 고속으로 입력하는 데 유용한 예제이다. 파일명은 sample4_append2.c 라고 한다.

미리 입력할 데이터를 data.txt에 저장해 두어야 한다.

```
./make_data
```

미리 주어진 make_data.c 를 수정하면 상황에 맞게 data.txt 파일을 생성할 수 있다.

› make_data.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <machbase_sqlcli.h>

#define MACHBASE_PORT_NO 5656
#define MAX_APPEND_COUNT 0xFFFFFFFF
#define ERROR_CHECK_COUNT 100

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB();
void disconnectDB();
void outError(const char *aMsg);
void executeDirectSQL(const char *aSQL, int aErrIgnore);
void createTable();
void appendOpen();
int appendData();
void appendClose();
time_t getTimeStamp();

int main()
{
    unsigned int sCount=0;
    time_t sStartTime, sEndTime;

    connectDB();
    createTable();
```

```
    appendOpen();
    sStartTime = getTimeStamp();
    sCount = appendData();
    sEndTime = getTimeStamp();

    appendClose();

    printf("timegap = %ld microseconds for %d records\n", sEndTime - sStartTime, sCount);
    printf("%.2f records/second\n", ((double)sCount/(double)(sEndTime - sStartTime))*1000000);

    disconnectDB();

    return 0;
}

void connectDB()
{
    char sConnStr[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        outError("SQLAllocEnv error!!");
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        outError("SQLAllocConnect error!!");
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)sConnStr, SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        outError("connection error!!");
    }

    if( SQL_ERROR == SQLAllocStmt(gCon, &gStmt) )
    {
        outError("SQLAllocStmt error!!");
    }

    printf("connected ... \n");
}

void disconnectDB()
{
    if( SQL_ERROR == SQLFreeStmt(gStmt, SQL_DROP) )
    {
        outError("SQLFreeStmt error");
    }

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        outError("disconnect error");
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg)
{
    SQLINTEGER sErrorNo;
    SQLSMALLINT sMsgLength;
    SQLCHAR sErrorMsg[1024];

    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, gStmt, NULL, &sErrorNo,
```

```
                                sErrorMsg, 1024, &sMsgLength ))
    {
        printf(" mach-%d : %s\n", sErrorNo, sErrorMsg);
    }

    if( gStmt )
    {
        SQLFreeStmt( gStmt, SQL_DROP );
    }
    if( gCon )
    {
        SQLFreeConnect( gCon );
    }
    if( gEnv )
    {
        SQLFreeEnv( gEnv );
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt;

    if (SQLAllocStmt(gCon, &sStmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error");
    }

    if (SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("sql_exec_direct error");
    }

    if (SQL_ERROR == SQLFreeStmt(sStmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error");
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float, ratio doub

    printf("table created\n");
}

void appendOpen()
{
    const char *sTableName = "CLI_SAMPLE";

    if( SQLAppendOpen(gStmt, (SQLCHAR *)sTableName, ERROR_CHECK_COUNT) != SQL_SUCCESS )
    {
        outError("SQLAppendOpen error!!");
    }

    printf("append open ok\n");
}

int appendData()
{
    FILE *sFp;
    char sBuf[1024];
    int j;
    char *sToken;
    unsigned int sCount=0;
```

```
SQL_APPEND_PARAM sParam[11];

sFp = fopen("data.txt", "r");
if( !sFp )
{
    printf("file open error\n");
    exit(-1);
}

printf("append data start\n");

memset(sBuf, 0, sizeof(sBuf));

while( fgets(sBuf, 1024, sFp ) != NULL )
{
    if( strlen(sBuf) < 1)
    {
        break;
    }

    j=0;
    sToken = strtok(sBuf,",");

    while( sToken != NULL )
    {
        memset(sParam+j, 0, sizeof(sParam));
        switch(j){
            case 0 : sParam[j].mShort = atoi(sToken); break; //short
            case 1 : sParam[j].mInteger = atoi(sToken); break; //int
            case 2 : sParam[j].mLong = atol(sToken); break; //long
            case 3 : sParam[j].mFloat = atof(sToken); break; //float
            case 4 : sParam[j].mDouble = atof(sToken); break; //double
            case 5 : //string
            case 9 : //text
            case 10 : //binary
                    sParam[j].mVar.mLength = strlen(sToken);
                    strcpy(sParam[j].mVar.mData, sToken);
                    break;
            case 6 : //ipv4
            case 7 : //ipv6
                    sParam[j].mIP.mLength = SQL_APPEND_IP_STRING;
                    strcpy(sParam[j].mIP.mAddrString, sToken);
                    break;
            case 8 : //datetime
                    sParam[j].mDateTime.mTime = SQL_APPEND_DATETIME_STRING;
                    strcpy(sParam[j].mDateTime.mDateStr, sToken);
                    sParam[j].mDateTime.mFormatStr = "DD/MON/YYYY:HH24:MI:SS";
                    break;
        }

        sToken = strtok(NULL, ",");

        j++;
    }
    if( SQLAppendDataV2(gStmt, sParam) != SQL_SUCCESS )
    {
        printf("SQLAppendData error\n");
        return 0;
    }
    if ( ((sCount++) % 10000) == 0)
    {
        printf(".");
    }

    if( ((sCount) % 100) == 0 )
    {
        if( SQLAppendFlush( gStmt ) != SQL_SUCCESS )
        {
            outError("SQLAppendFlush error");
        }
```

```
        }
        if (sCount == MAX_APPEND_COUNT)
        {
            break;
        }
    }

    printf("\nappend data end\n");

    fclose(sFp);

    return sCount;
}

void appendClose()
{
    int sSuccessCount = 0;
    int sFailureCount = 0;

    if( SQLAppendClose(gStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        outError("SQLAppendClose error");
    }

    printf("append close ok\n");
    printf("success : %d, failure : %d\n", sSuccessCount, sFailureCount);
}

time_t getTimeStamp()
{
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec*1000000+tv.tv_usec;
}
```

Makefile에 sample4_append2.c를 등록하고 컴파일하여 실행하면 다음과 같이 나온다.

```
[mach@localhost cli]$ make
gcc -c -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -I/home/ma
gcc -m64 -mtune=k8 -L/home/mach/machbase_home/lib -osingle_append2 single_append2.o -lmachcli -L/home/mach/machbase
[mach@localhost cli]$ ./single_append2
connected ...
table created
append open ok
append data start
.....................................................................................................
append data end
append close ok
success : 1000000, failure : 0
timegap = 1641503 microseconds for 1000000 records
609197.79 records/second
```

테이블 열 정보 획득 예제

테이블 열 정보를 획득하는 방법은 다양하지만 그중에 SQLDescribeCol과 SQLColumns를 이용한 방법을 살펴본다.

SQLDescribeCol

SQLDescribeCol은 테이블 열의 번호, 이름, 버퍼 크기, 길이, 타입 등을 가져오는 함수로 이를 이용해서 데이터베이스 내부에서 원하는 내용을 손쉽게 가져올수 있다.
예제 파일명은 sample5_describe.c 라고 한다.

› sample5_describe.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <machbase_sqlcli.h>
#include <time.h>

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
    char connStr[1024];

    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }

    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");

        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
                                      errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }

    if (SQLAllocStmt(gCon, &gStmt) == SQL_ERROR)
    {
        outError("AllocStmt error", gStmt);
    }

    printf("connected ... \n");

}

void disconnectDB()
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                     errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
```

```
        }
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT stmt)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, stmt, NULL, &errNo,
                                 errMsg, 1024, &msgLength ))
    {
        printf(" mach-%d : %s\n", errNo, errMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT stmt;

    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", stmt);
    }

    if (SQLExecDirect(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", stmt);
    }

    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", stmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float, ratio doub

}

int main()
{
    char sSqlStr[] = "select * from cli_sample";
    SQLCHAR sColName[32];
    SQLSMALLINT sColType;
    SQLSMALLINT sColNameLen;
    SQLSMALLINT sNullable;
    SQLULEN sColLen;
    SQLSMALLINT sDecimalDigits;
    SQLLEN sOutlen;
    SQLCHAR* sData;
    SQLLEN sDisplaySize;
    int i;

    SQLSMALLINT sColumns;
```

```
    connectDB();

    createTable();

    if(SQLPrepare(gStmt, (SQLCHAR*)sSqlStr, SQL_NTS))
    {
        outError("sql prepare fail", gStmt);
        return -1;
    }

    if(SQLNumResultCols(gStmt, &sColumns) != SQL_SUCCESS )
    {
        printf("get col length error \n");
        return -1;
    }

    printf("----------------------------------------------------------------\n");
    printf("%32s%16s%10s\n","Name","Type","Length");
    printf("----------------------------------------------------------------\n");

    for(i = 0; i < sColumns; i++)
    {
        SQLDescribeCol(gStmt,
                       (SQLUSMALLINT)(i + 1),
                       sColName,
                       sizeof(sColName),
                       &sColNameLen,
                       &sColType,
                       (SQLULEN *)&sColLen,
                       &sDecimalDigits,
                       (SQLSMALLINT *)&sNullable);

        printf("%32s%16d%10d\n",sColName, sColType, sColLen);
    }

    printf("----------------------------------------------------------------\n");

    disconnectDB();

    return 0;
}
```

위의 파일을 추가하고 make를 실행하면 아래와 같이 원하는 열의 내용들이 나타나는 것을 볼 수 있다.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample5_describe
connected ...
----------------------------------------------------------------
Name Type Length
----------------------------------------------------------------
SEQ 5 5
SCORE 4 10
TOTAL -5 19
PERCENTAGE 6 27
RATIO 8 27
ID 12 10
SRCIP 2104 15
DSTIP 2106 60
REG_DATE 9 31
TLOG 2100 67108864
IMAGE -2 67108864
----------------------------------------------------------------
[mach@localhost cli]$
```

SQLColumns

SQLColumns은 현재 테이블 내에 존재하는 컬럼들의 정보를 알아낼 수 있는 함수이다. 마크베이스에서도 위와 같은 함수를 지원하고 있으며 이를 이용하여 컬럼 각각의 정보들을 알아낼 수 있다.

파일이름은 sample6_columns.c라고 한다.

› sample6_columns.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <machbase_sqlcli.h>

#include <time.h>

#define MACHBASE_PORT_NO 5656

SQLHENV gEnv;
SQLHDBC gCon;
SQLHSTMT gStmt;
SQLCHAR gErrorState[6];

void connectDB()
{
    char connStr[1024];

    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLAllocEnv(&gEnv)) {
        printf("SQLAllocEnv error!!\n");
        exit(1);
    }

    if (SQL_ERROR == SQLAllocConnect(gEnv, &gCon)) {
        printf("SQLAllocConnect error!!\n");
        SQLFreeEnv(gEnv);
        exit(1);
    }

    sprintf(connStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if (SQL_ERROR == SQLDriverConnect( gCon, NULL,
                                       (SQLCHAR *)connStr,
                                       SQL_NTS,
                                       NULL, 0, NULL,
                                       SQL_DRIVER_NOPROMPT ))
    {
        printf("connection error\n");

        if (SQL_SUCCESS == SQLError ( gEnv, gCon, NULL, NULL, &errNo,
                                      errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
        SQLFreeEnv(gEnv);
        exit(1);
    }

    if (SQLAllocStmt(gCon, &gStmt) == SQL_ERROR)
    {
        outError("AllocStmt error", gStmt);
    }

    printf("connected ... \n");

}

void disconnectDB()
```

```
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    if (SQL_ERROR == SQLDisconnect(gCon)) {
        printf("disconnect error\n");

        if( SQL_SUCCESS == SQLError( gEnv, gCon, NULL, NULL, &errNo,
                                     errMsg, 1024, &msgLength ))
        {
            printf(" mach-%d : %s\n", errNo, errMsg);
        }
    }

    SQLFreeConnect(gCon);
    SQLFreeEnv(gEnv);
}

void outError(const char *aMsg, SQLHSTMT stmt)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[1024];

    printf("ERROR : (%s)\n", aMsg);

    if (SQL_SUCCESS == SQLError( gEnv, gCon, stmt, NULL, &errNo,
                                 errMsg, 1024, &msgLength ))
    {
        printf(" mach-%d : %s\n", errNo, errMsg);
    }
    exit(-1);
}

void executeDirectSQL(const char *aSQL, int aErrIgnore)
{
    SQLHSTMT stmt;

    if (SQLAllocStmt(gCon, &stmt) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        outError("AllocStmt error", stmt);
    }

    if (SQLExecDirect(stmt, (SQLCHAR *)aSQL, SQL_NTS) == SQL_ERROR)
    {
        if (aErrIgnore != 0) return;
        printf("sql_exec_direct error[%s] \n", aSQL);
        outError("sql_exec_direct error", stmt);
    }

    if (SQL_ERROR == SQLFreeStmt(stmt, SQL_DROP))
    {
        if (aErrIgnore != 0) return;
        outError("FreeStmt Error", stmt);
    }
}

void createTable()
{
    executeDirectSQL("DROP TABLE CLI_SAMPLE", 1);
    executeDirectSQL("CREATE TABLE CLI_SAMPLE(seq short, score integer, total long, percentage float, ratio doub
}

int main()
{
    SQLCHAR sColName[32];
    SQLSMALLINT sColType;
    SQLCHAR sColTypeName[16];
```

```
    SQLSMALLINT sColNameLen;
    SQLSMALLINT sColTypeLen;
    SQLSMALLINT sNullable;
    SQLULEN sColLen;
    SQLSMALLINT sDecimalDigits;
    SQLLEN sOutlen;
    SQLCHAR* sData;
    SQLLEN sDisplaySize;
    int i;

    SQLSMALLINT sColumns;

    connectDB();

    createTable();

    if(SQLColumns(gStmt, NULL, 0, NULL, 0, "cli_sample", SQL_NTS, NULL, 0) != SQL_SUCCESS)
    {
        printf("sql columns error!\n");
        return -1;
    }

    SQLBindCol(gStmt, 4, SQL_C_CHAR, sColName, sizeof(sColName), &sColNameLen);
    SQLBindCol(gStmt, 5, SQL_C_SSHORT, &sColType, 0, &sColTypeLen);
    SQLBindCol(gStmt, 6, SQL_C_CHAR, sColTypeName, sizeof(sColTypeName), NULL);
    SQLBindCol(gStmt, 7, SQL_C_SLONG, &sColLen, 0, NULL);

    printf("--------------------------------------------------------------------------------\n");
    printf("%32s%16s%16s%10s\n","Name","Type","TypeName","Length");
    printf("--------------------------------------------------------------------------------\n");

    while( SQLFetch(gStmt) != SQL_NO_DATA )
    {
        printf("%32s%16d%16s%10d\n",sColName, sColType, sColTypeName, sColLen);
    }
    printf("--------------------------------------------------------------------------------\n");

    disconnectDB();

    return 0;
}
```

위의 파일을 추가하고 make를 실행한다. 결과는 다음과 같다.

```
[mach@localhost cli]$ make

[mach@localhost cli]$ ./sample6_columns
connected ...
--------------------------------------------------------------------------------
Name Type TypeName Length
--------------------------------------------------------------------------------
_ARRIVAL_TIME 93 DATE 31
SEQ 5 SMALLINT 5
SCORE 4 INTEGER 10
TOTAL -5 BIGINT 19
PERCENTAGE 6 FLOAT 27
RATIO 8 DOUBLE 27
ID 12 VARCHAR 10
SRCIP 2104 IPV4 15
DSTIP 2106 IPV6 60
REG_DATE 93 DATE 31
TLOG 2100 TEXT 67108864
IMAGE -2 BINARY 67108864
--------------------------------------------------------------------------------
```

멀티 쓰레드 append 예제

하나의 프로그램에서 여러 스레드를 이용해 여러 테이블에 append하는 예제이다.

파일 이름은 sample8_multi_session_multi_table.c로 한다.

› sample8_multi_session_multi_table.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <machbase_sqlcli.h>

#define MACHBASE_PORT_NO       5656
#define ERROR_CHECK_COUNT   100

#define LOG_FILE_CNT        3
#define MAX_THREAD_NUM      LOG_FILE_CNT

#define RC_FAILURE          -1
#define RC_SUCCESS          0

#define UNUSED(aVar) do { (void)(aVar); } while(0)

char *gTableName[LOG_FILE_CNT] = {"table_f1", "table_f2", "table_event"};
char *gFileName[LOG_FILE_CNT] =  {"suffle_data1.txt","suffle_data2.txt","suffle_data3.txt"};

void printError(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char *aMsg);
int connectDB(SQLHENV *aEnv, SQLHDBC *aCon);
void disconnectDB(SQLHENV aEnv, SQLHDBC aCon);
int executeDirectSQL(SQLHENV aEnv, SQLHDBC aCon, const char *aSQL, int aErrIgnore);
int appendOpen(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char* aTableName);
int appendClose(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt);
int createTables(SQLHENV aEnv, SQLHDBC aCon);

/*
 * error code returned from CLI lib
 */
void printError(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char *aMsg)
{
    SQLINTEGER      sNativeError;
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    if( aMsg != NULL )
    {
        printf("%s\n", aMsg);
    }

    if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                 sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) == SQL_SUCCESS )
    {
        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);
    }
}

/*
 * error code returned from Machbase server
 */

void appendDumpError(SQLHSTMT    aStmt,
                     SQLINTEGER  aErrorCode,
                     SQLPOINTER  aErrorMessage,
                     SQLLEN      aErrorBufLen,
                     SQLPOINTER  aRowBuf,
                     SQLLEN      aRowBufLen)
{
    char       sErrMsg[1024] = {0, };
    char       sRowMsg[32 * 1024] = {0, };
```

```
    UNUSED(aStmt);

    if (aErrorMessage != NULL)
    {
        strncpy(sErrMsg, (char *)aErrorMessage, aErrorBufLen);
    }

    if (aRowBuf != NULL)
    {
        strncpy(sRowMsg, (char *)aRowBuf, aRowBufLen);
    }

    fprintf(stdout, "Append Error : [%d][%s]\n[%s]\n\n", aErrorCode, sErrMsg, sRowMsg);
}


int connectDB(SQLHENV *aEnv, SQLHDBC *aCon)
{
    char sConnStr[1024];

    if( SQLAllocEnv(aEnv) != SQL_SUCCESS )
    {
        printf("SQLAllocEnv error\n");
        return RC_FAILURE;
    }

    if( SQLAllocConnect(*aEnv, aCon) != SQL_SUCCESS )
    {
        printf("SQLAllocConnect error\n");

        SQLFreeEnv(*aEnv);
        *aEnv = SQL_NULL_HENV;

        return RC_FAILURE;
    }

    sprintf(sConnStr,"SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;PORT_NO=%d", MACHBASE_PORT_NO);

    if( SQLDriverConnect( *aCon, NULL,
                          (SQLCHAR *)sConnStr,
                          SQL_NTS,
                          NULL, 0, NULL,
                          SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS
      )
    {

        printError(*aEnv, *aCon, NULL, "SQLDriverConnect error");

        SQLFreeConnect(*aCon);
        *aCon = SQL_NULL_HDBC;

        SQLFreeEnv(*aEnv);
        *aEnv = SQL_NULL_HENV;

        return RC_FAILURE;
    }

    return RC_SUCCESS;
}


void disconnectDB(SQLHENV aEnv, SQLHDBC aCon)
{
    if( SQLDisconnect(aCon) != SQL_SUCCESS )
    {
        printError(aEnv, aCon, NULL, "SQLDisconnect error");
    }

    SQLFreeConnect(aCon);
```

```
    aCon = SQL_NULL_HDBC;

    SQLFreeEnv(aEnv);
    aEnv = SQL_NULL_HENV;
}


int executeDirectSQL(SQLHENV aEnv, SQLHDBC aCon, const char *aSQL, int aErrIgnore)
{
    SQLHSTMT sStmt = SQL_NULL_HSTMT;

    if( SQLAllocStmt(aCon, &sStmt) != SQL_SUCCESS )
    {
        if( aErrIgnore == 0 )
        {
            printError(aEnv, aCon, sStmt, "SQLAllocStmt Error");
            return RC_FAILURE;
        }
    }

    if( SQLExecDirect(sStmt, (SQLCHAR *)aSQL, SQL_NTS) != SQL_SUCCESS )
    {

        if( aErrIgnore == 0 )
        {
            printError(aEnv, aCon, sStmt, "SQLExecDirect Error");

            SQLFreeStmt(sStmt,SQL_DROP);
            sStmt = SQL_NULL_HSTMT;
            return RC_FAILURE;
        }
    }

    if( SQLFreeStmt(sStmt, SQL_DROP) != SQL_SUCCESS )
    {
        if (aErrIgnore == 0)
        {
            printError(aEnv, aCon, sStmt, "SQLFreeStmt Error");
            sStmt = SQL_NULL_HSTMT;
            return RC_FAILURE;
        }
    }
    sStmt = SQL_NULL_HSTMT;

    return RC_SUCCESS;
}


int appendOpen(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, char* aTableName)
{
    if( aTableName == NULL )
    {
        printf("append open wrong table name");
        return RC_FAILURE;
    }

    if( SQLAppendOpen(aStmt, (SQLCHAR *)aTableName, ERROR_CHECK_COUNT) != SQL_SUCCESS )
    {
        printError(aEnv, aCon, aStmt, "SQLAppendOpen error");
        return RC_FAILURE;
    }
    return RC_SUCCESS;
}


int appendClose(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt)
{
    int sSuccessCount = 0;
    int sFailureCount = 0;
```

```
    if( SQLAppendClose(aStmt, &sSuccessCount, &sFailureCount) != SQL_SUCCESS )
    {
        printError(aEnv, aCon, aStmt, "SQLAppendClose error");
        return RC_FAILURE;
    }

    printf("append result success : %d, failure : %d\n", sSuccessCount, sFailureCount);

    return RC_SUCCESS;
}


int createTables(SQLHENV aEnv, SQLHDBC aCon)
{
    int      i;
    char    *sSchema[] = { "srcip1 ipv4, srcip2 ipv6, srcport short, dstip1 ipv4, dstip2 ipv6, dstport short, da
        "srcip1 ipv4, srcip2 ipv6, srcport short, dstip1 ipv4, dstip2 ipv6, dstport short, data1 long, data2 lon
        "machine ipv4, err integer, msg varchar(30)"
    };

    char sDropQuery[256];
    char sCreateQuery[256];

    for(i = 0; i < LOG_FILE_CNT; i++)
    {
        snprintf(sDropQuery, 256, "DROP TABLE %s", gTableName[i]);
        snprintf(sCreateQuery, 256, "CREATE TABLE %s ( %s )", gTableName[i], sSchema[i]);

        executeDirectSQL(aEnv, aCon, sDropQuery, 1);
        executeDirectSQL(aEnv, aCon, sCreateQuery, 0);
    }

    return RC_SUCCESS;
}


int appendF1(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, FILE *aFp)
{
    SQL_APPEND_PARAM sParam[8];
    SQLRETURN        sRC;

    SQLINTEGER      sNativeError;
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    char             sData[4][64];

    memset(sParam, 0, sizeof(sParam));

    fscanf(aFp, "%s %s %hd %s %s %hd %lld %lld\n",
           sData[0], sData[1], &sParam[2].mShort,
           sData[2], sData[3], &sParam[5].mShort,
           &sParam[6].mLong, &sParam[7].mLong);

    sParam[0].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = sData[0];

    sParam[1].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[1].mIP.mAddrString = sData[1];

    sParam[3].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[3].mIP.mAddrString = sData[2];

    sParam[4].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[4].mIP.mAddrString = sData[3];

    sRC = SQLAppendDataV2(aStmt, sParam);
    if( !SQL_SUCCEEDED(sRC) )
    {
```

```
        if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                     sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) != SQL_SUCCESS )
        {
            return RC_FAILURE;
        }

        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);

        if( sNativeError != 9604 &&
            sNativeError != 9605 &&
            sNativeError != 9606 )
        {
            return RC_FAILURE;
        }
        else
        {
            //data value error in one record, so return success to keep attending
        }
    }
    return RC_SUCCESS;
}


int appendF2(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, FILE* aFp)
{
    SQL_APPEND_PARAM sParam[8];
    SQLRETURN        sRC;

    SQLINTEGER      sNativeError;
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    char             sData[4][64];

    memset(sParam, 0, sizeof(sParam));

    fscanf(aFp, "%s %s %hd %s %s %hd %lld %lld\n",
           sData[0], sData[1], &sParam[2].mShort,
           sData[2], sData[3], &sParam[5].mShort,
           &sParam[6].mLong, &sParam[7].mLong);

    sParam[0].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = sData[0];

    sParam[1].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[1].mIP.mAddrString = sData[1];

    sParam[3].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[3].mIP.mAddrString = sData[2];

    sParam[4].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[4].mIP.mAddrString = sData[3];

    sRC = SQLAppendDataV2(aStmt, sParam);
    if( !SQL_SUCCEEDED(sRC) )
    {
        if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                     sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) != SQL_SUCCESS )
        {
            return RC_FAILURE;
        }

        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);

        if( sNativeError != 9604 &&
            sNativeError != 9605 &&
            sNativeError != 9606 )
        {
            return RC_FAILURE;
```

```
        }
        else
        {
            //data value error in one record, so return success to keep attending
        }
    }
    return RC_SUCCESS;
}


int appendEvent(SQLHENV aEnv, SQLHDBC aCon, SQLHSTMT aStmt, FILE* aFp)
{
    SQL_APPEND_PARAM sParam[3];
    SQLRETURN        sRC;

    SQLINTEGER      sNativeError;
    SQLCHAR         sErrorMsg[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR         sSqlState[SQL_SQLSTATE_SIZE + 1];
    SQLSMALLINT     sMsgLength;

    char             sData[2][20];

    memset(sParam, 0, sizeof(sParam));

    fscanf(aFp, "%s %d %s\n",sData[0], &sParam[1].mInteger, sData[1]);

    sParam[0].mIP.mLength = SQL_APPEND_IP_STRING;
    sParam[0].mIP.mAddrString = sData[0];

    sParam[2].mVarchar.mLength = strlen(sData[1]);
    sParam[2].mVarchar.mData = sData[1];

    sRC = SQLAppendDataV2(aStmt, sParam);
    if( !SQL_SUCCEEDED(sRC) )
    {
        if( SQLError(aEnv, aCon, aStmt, sSqlState, &sNativeError,
                     sErrorMsg, SQL_MAX_MESSAGE_LENGTH, &sMsgLength) != SQL_SUCCESS )
        {
            return RC_FAILURE;
        }

        printf("SQLSTATE-[%s], Machbase-[%d][%s]\n", sSqlState, sNativeError, sErrorMsg);

        if( sNativeError != 9604 &&
            sNativeError != 9605 &&
            sNativeError != 9606 )
        {
            return RC_FAILURE;
        }
        else
        {
            //data value error in one record, so return success to keep attending
        }
    }
    return RC_SUCCESS;
}


void *eachThread(void *aIdx)
{
    SQLHENV    sEnv = SQL_NULL_HENV;
    SQLHDBC    sCon = SQL_NULL_HDBC;
    SQLHSTMT   sStmt[LOG_FILE_CNT] = {SQL_NULL_HSTMT,};

    FILE*      sFp;
    int        i;
    int        sLogType;

    int        sThrNo = *(int *)aIdx;
```

```
// Alloc ENV and DBC
if( connectDB(&sEnv, &sCon) == RC_SUCCESS )
{
    printf("[%d]connectDB success.\n", sThrNo);
}
else
{
    printf("[%d]connectDB failure.\n", sThrNo);
    goto error;
}

// set timed flush true
if( SQLSetConnectAppendFlush(sCon, 1) != SQL_SUCCESS )
{
    printError(sEnv, sCon, NULL, "SQLSetConnectAppendFlush Error");
    goto error;
}

for( i = 0; i < LOG_FILE_CNT; i++ )
{
    // Alloc stmt
    if( SQLAllocStmt(sCon,&sStmt[i]) != SQL_SUCCESS )
    {
        printError(sEnv, sCon, sStmt[i], "SQLAllocStmt Error");
        goto error;
    }

    if( appendOpen(sEnv, sCon, sStmt[i], gTableName[i]) == RC_FAILURE )
    {
        printError(sEnv, sCon, sStmt[i], "SQLAppendOpen Error");
        goto error;
    }
    else
    {
        printf("[%d-%d]appendOpen success.\n", sThrNo, i);
    }

    if( SQLAppendSetErrorCallback(sStmt[i], appendDumpError) != SQL_SUCCESS )
    {
        printError(sEnv, sCon, sStmt[i], "SQLAppendSetErrorCallback Error");
        goto error;
    }

    // set timed flush interval as 2 seconds
    if( SQLSetStmtAppendInterval(sStmt[i], 2000) != SQL_SUCCESS )
    {
        printError(sEnv, sCon, sStmt[i], "SQLSetStmtAppendInterval Error");
        goto error;
    }
}

sFp = fopen((char*)gFileName[sThrNo], "rt");
if( sFp == NULL )
{
    printf("file open error - [%d][%s]\n", sThrNo, gFileName[sThrNo]);
}
else
{
    printf("file open success - [%d][%s]\n", sThrNo, gFileName[sThrNo]);

    for( i = 0; !feof(sFp); i++ )
    {
        fscanf(sFp, "%d ", &sLogType);
        switch(sLogType)
        {
            case 1://f1
                if( appendF1(sEnv, sCon, sStmt[0], sFp) == RC_FAILURE )
                {
                    goto error;
                }
```

```
                    break;
                case 2://f2
                    if( appendF2(sEnv, sCon, sStmt[1],sFp) == RC_FAILURE )
                    {
                        goto error;
                    }
                    break;
                case 3://event
                    if(appendEvent(sEnv, sCon, sStmt[2], sFp) == RC_FAILURE )
                    {
                        goto error;
                    }
                    break;
                default:
                    printf("unknown type error\n");
                    break;
            }

            if( (i%10000) == 0 )
            {
                fprintf(stdout, ".");
                fflush(stdout);
            }
        }
        printf("\n");

        fclose(sFp);
    }

    for( i = 0; i < LOG_FILE_CNT; i++)
    {
        printf("[%d-%d]appendClose start...\n", sThrNo, i);
        if( appendClose(sEnv, sCon, sStmt[i]) == RC_FAILURE )
        {
            printf("[%d-%d]appendClose failure\n", sThrNo, i);
        }
        else
        {
            printf("[%d-%d]appendClose success\n", sThrNo, i);
        }

        if( SQLFreeStmt(sStmt[i], SQL_DROP) != SQL_SUCCESS )
        {
            printError(sEnv, sCon, sStmt[i], "SQLFreeStmt Error");
        }
        sStmt[i] = SQL_NULL_HSTMT;
    }

    disconnectDB(sEnv, sCon);

    printf("[%d]disconnected.\n", sThrNo);

    pthread_exit(NULL);

error:
    for( i = 0; i < LOG_FILE_CNT; i++)
    {
        if( sStmt[i] != SQL_NULL_HSTMT )
        {
            appendClose(sEnv, sCon, sStmt[i]);

            if( SQLFreeStmt(sStmt[i], SQL_DROP) != SQL_SUCCESS )
            {
                printError(sEnv, sCon, sStmt[i], "SQLFreeStmt Error");
            }
            sStmt[i] = SQL_NULL_HSTMT;
        }
    }

    if( sCon != SQL_NULL_HDBC )
```

```
    {
        disconnectDB(sEnv, sCon);
    }

    pthread_exit(NULL);
}


int initTables()
{
    SQLHENV     sEnv  = SQL_NULL_HENV;
    SQLHDBC     sCon  = SQL_NULL_HDBC;

    if( connectDB(&sEnv, &sCon) == RC_SUCCESS )
    {
        printf("connectDB success.\n");
    }
    else
    {
        printf("connectDB failure.\n");
        goto error;
    }

    if( createTables(sEnv, sCon) == RC_SUCCESS )
    {
        printf("createTables success.\n");
    }
    else
    {
        printf("createTables failure.\n");
        goto error;
    }

    disconnectDB(sEnv, sCon);

    return RC_SUCCESS;

error:

    if( sCon != SQL_NULL_HDBC )
    {
        disconnectDB(sEnv, sCon);
    }

    return RC_FAILURE;
}


int main()
{
    pthread_t sThread[MAX_THREAD_NUM];
    int       sNum[MAX_THREAD_NUM];
    int       sRC;
    int       i;

    initTables();

    //
    //eachThread has own ENV,DBC and STMT
    //
    for(i = 0; i < MAX_THREAD_NUM; i++)
    {
        sNum[i] = i;

        sRC = pthread_create(&sThread[i], NULL, (void *)eachThread, (void*)&sNum[i]);
        if ( sRC != RC_SUCCESS )
        {
            printf("Error in Thread create[%d] : %d\n", i, sRC);
            return RC_FAILURE;
        }
```

```
    }

    for(i = 0; i < MAX_THREAD_NUM; i++)
    {
        sRC = pthread_join(sThread[i], NULL);
        if( sRC != RC_SUCCESS )
        {
            printf("Error in Thread[%d] : %d\n", i, sRC);
            return RC_FAILURE;
        }
        printf("%d thread join\n", i+1);
    }

    return RC_SUCCESS;
}
```

make 코드를 추가하고 실행 파일을 실행해본다. 쓰레드를 이용하므로 출력 순서가 다를 수 있다. 실행 결과는 다음과 같다.

```
[mach@localhost cli]$ make sample8_multi_session_multi_table
gcc -c -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -g -W -Wall -rdynamic -fno-inline -m64 -mtune=k8 -I/home/ma
gcc -m64 -mtune=k8 -L/home/mach/machbase_home/lib -osample8_multi_session_multi_table sample8_multi_session_multi_t
[mach@localhost cli]$ ./sample8_multi_session_multi_table
connectDB success.
createTables success.
[0]connectDB success.
[1]connectDB success.
[2]connectDB success.
[1-0]appendOpen success.
[0-0]appendOpen success.
[2-0]appendOpen success.
[1-1]appendOpen success.
[2-1]appendOpen success.
[0-1]appendOpen success.
[1-2]appendOpen success.
[2-2]appendOpen success.
file open success - [1][suffle_data2.txt]
file open success - [2][suffle_data3.txt]
[0-2]appendOpen success.
file open success - [0][suffle_data1.txt]
.....................................................................................

[1-0]appendClose start...
..
[0-0]appendClose start...
append result success : 100000, failure : 0
[1-0]appendClose success
[1-1]appendClose start...
append result success : 100000, failure : 0
[1-1]appendClose success
[1-2]appendClose start...
append result success : 100000, failure : 0
[1-2]appendClose success
append result success : 100000, failure : 0
[0-0]appendClose success
[0-1]appendClose start...
.append result success : 100000, failure : 0
[0-1]appendClose success
[0-2]appendClose start...
append result success : 100000, failure : 0
[0-2]appendClose success

[2-0]appendClose start...
append result success : 100000, failure : 0
[2-0]appendClose success
[2-1]appendClose start...
append result success : 100000, failure : 0
[2-1]appendClose success
[2-2]appendClose start...
```

```
append result success : 100000, failure : 0
[2-2]appendClose success
[1]disconnected.
[2]disconnected.
[0]disconnected.
1 thread join
2 thread join
3 thread join
```

machsql을 통해 아래와 같이 결과를 확인할 수 있다.

```
[mach@localhost cli]$ machsql


=================================================================
     Machbase Client Query Utility
     Release Version 3.5.0
     Copyright 2014, Machbase Inc. or its subsidiaries.
     All Rights Reserved.
=================================================================
Machbase Server Addr (Default:127.0.0.1) :
Machbase User ID  (Default:SYS)
Machbase User Password : manager
MACH_CONNECT_MODE=INET, PORT=5656
Mach> select count(*) from table_f1;
count(*)
-----------------------
300000
[1] Row Selected.
Mach> select count(*) from table_f2;
count(*)
-----------------------
300000
[1] row(s) selected.
Mach> select count(*) from table_event;
count(*)
-----------------------
300000
[1] row(s) selected.
```

JDBC

JDBC 개요

자바 프로그래밍 언어로 만들어진 데이터베이스 조작 인터페이스의 집합을 JDBC(Java DataBase Connectivity)라고 한다. 다양한 관계형 데이터베이스를 위해 일관된 인터페이스를 제공하는 API 집합으로서 프로그래머가 SQL 요구를 만드는데 사용할 일련의 객체지향 프로그램의 클래스들을 정의하고 있다. 즉, 어떤 데이터베이스를 사용하더라도 JDBC 드라이버만 제공된다면 코드 수정 없이 바로 적용 가능한 장점이 있다.

목차

표준 JDBC 함수

표준 함수 스펙 4.0

확장 JDBC 함수

setIPv4

```
void setIpv4(int ind, String ipString)
```

PrepareStatement에서 IPv4 주소 타입을 입력하기 위한 함수이다.

컬럼 인덱스와 IPv4 문자열을 인자로 받는다.

setIPv6

```
void setIpv6(int ind, String ipString)
```

PrepareStatement에서 IPv6 주소 타입을 입력하기 위한 함수이다.

컬럼 인덱스와 IPv6 문자열을 인자로 받는다.

executeAppendOpen

```
ResultSet executeAppendOpen(String aTableName, int aErrorCheckCount)
```

Statement에서 Append 프로토콜을 쓰기 위한 것으로 프로토콜을 오픈한다.

테이블 이름과 오류 검사 간격을 인자로 받는다. 결과값으로 ResultSet을 리턴한다.

executeAppendData

```
int executeAppendData(ResultSetMetaData rsmd, ArrayList aData)
```

Statement에서 Append 프로토콜을 위한 것으로 실제 데이터를 입력한다.

executeAppendOpen의 결과값인 ResultSet의 메타데이터와 입력하고자 하는 데이터를 인자로 받는다. 결과값이 전송 버퍼에 저장되면 1이 리턴되고, 전송 버퍼가 차서 마크베이스로 전송되면 2가 리턴된다. 따라서 1 또는 2가 리턴되면 성공으로 판단하면 된다.

executeAppendDataByTime

```
int executeAppendDataByTime(ResultSetMetaData rsmd, long aTime, ArrayList aData)
```

Statement에서 Append 프로토콜을 위한 것으로 실제 데이터를 시간 기준으로 입력한다.

executeAppendOpen의 결과값인 ResultSet의 메타데이터와 설정하고자 하는 특정 시간대의 시간 값, 입력하고자 하는 데이터를 인자로 받는다. 결과값이 전송 버퍼에 저장되면 1이 리턴된다.

executeAppendClose

```
int executeAppendClose()
```

Statement에서 Append 프로토콜을 위한 것으로 statement를 종료한다.
결과값으로 성공하면 1을 리턴한다.

executeSetAppendErrorCallback

```
int executeSetAppendErrorCallback(MachAppendCallback aCallback)
```

Append 수행하는 도중에 에러가 발생하는 경우 에러를 출력하는 콜백 함수를 설정한다.
에러 로그를 출력하는 콜백 함수를 인자로 받는다. 결과값으로 성공하면 1이 리턴된다.

getAppendSuccessCount

```
long getAppendSuccessCount()
```

Statement에서 Append 프로토콜을 위한 것으로 성공한 개수를 리턴한다.
결과값으로 성공한 개수를 리턴한다.

getAppendFailCount

```
long getAppendFailCount()
```

Statement에서 Append 프로토콜을 위한 것으로 실패한 개수를 리턴한다.
결과값으로 실패한 개수를 리턴한다.

응용 프로그램 개발

JDBC 라이브러리 설치 확인

$MACHBASE_HOME/lib 디렉터리에 machbase.jar 파일이 있는지 확인한다.

```
[mach@localhost ~]$ cd $MACHBASE_HOME/lib
[mach@localhost lib]$ ls -l machbase.jar
-rw-rw-r-- 1 mach mach 78599 Jun 18 10:00 machbase.jar
[mach@localhost lib]$
```

Makefile 작성 가이드

$(MACHBASE_HOME)/lib/machbase.jar를 classpath에 지정해주어야 한다. 다음은 Makefile 예시이다.

```
CLASSPATH=".:$(MACHBASE_HOME)/lib/machbase.jar"

SAMPLE_SRC = Sample1Connect.java Sample2Insert.java Sample3PrepareStmt.java Sample4Append.java

all: build

build:
    -@rm -rf *.class
    javac -classpath $(CLASSPATH) -d . $(SAMPLE_SRC)

create_table:
    machsql -s localhost -u sys -p manager -f createTable.sql

select_table:
    machsql -s localhost -u sys -p manager -f selectTable.sql
```

```
run_sample1:
    java -classpath $(CLASSPATH) Sample1Connect

run_sample2:
    java -classpath $(CLASSPATH) Sample2Insert

run_sample3:
    java -classpath $(CLASSPATH) Sample3PrepareStmt

run_sample4:
    java -classpath $(CLASSPATH) Sample4Append

clean:
    rm -rf *.class
```

컴파일 및 링크

다음과 같이 make 명령어를 수행하여 컴파일 및 링크를 수행한다.

```
[mach@localhost jdbc]$ make
javac -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" -d . Sample1Connect.java Sample2Insert.java Samp
[mach@localhost jdbc]$
```

JDBC 샘플

접속 예제

마크베이스 JDBC 드라이버를 이용하여 마크베이스 서버에 접속하는 예제 프로그램을 작성해 보기로 한다. 소스 파일명을 Sample1Connect.java로 한다.

> ⓘ _arrival_time 컬럼은 디폴트로 표시되지 않는다. 따라서 _arrival_time 컬럼을 표시하려면, 연결 문자열에 show_hidden_cols=1 을 추가하면 된다.
>
> 아래 예제 소스에서 접속 문자열을 다음과 같이 수정하면 된다.
>
> `String sURL = "jdbc:machbase://localhost:5656/mhdb?show_hidden_cols=1";`

```
import java.util.*;
import java.sql.*;
import com.machbase.jdbc.*;

public class Sample1Connect
{
    public static Connection connect()
    {
        Connection conn = null;
        try
        {
            String sURL = "jdbc:machbase://localhost:5656/mhdb";

            Properties sProps = new Properties();
            sProps.put("user", "sys");
            sProps.put("password", "manager");

            Class.forName("com.machbase.jdbc.driver");
            conn = DriverManager.getConnection(sURL, sProps);
        }
        catch ( ClassNotFoundException ex )
        {
            System.err.println("Exception : unable to load mach jdbc driver class");
        }
        catch ( Exception e )
        {
            System.err.println("Exception : " + e.getMessage());
        }
        return conn;
```

```
    }

    public static void main(String[] args) throws Exception
    {
        Connection conn = null;

        try
        {
            conn = connect();
            if( conn != null )
            {
                System.out.println("mach JDBC connected.");
            }
        }
        catch( Exception e )
        {
            System.err.println("Exception : " + e.getMessage());
        }
        finally
        {
            if( conn != null )
            {
                conn.close();
                conn = null;
            }
        }
    }
}
```

이제 소스 코드를 컴파일하고 실행한다. 이미 작성한 Makefile을 이용한다.

```
[mach@localhost jdbc]$ make
javac -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" -d . Sample1Connect.java Sample2Insert.java Samp
[mach@localhost jdbc]$ make run_sample1
java -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" Sample1Connect
mach JDBC connected.
```

데이터 입력 및 출력 예제 (1) 직접 입/출력

마크베이스 JDBC 드라이버를 이용하여 데이터를 입력하고 출력하는 예제를 작성하여 보기로 한다.

소스 파일명은 Sample2Insert.java 라고 한다.

먼저, machsql 프로그램을 이용하여 필요한 테이블을 생성하여야 한다.
예제에서는 sample_table이라는 테이블을 미리 생성한 뒤에 샘플 코드를 이용하는 방식을 사용했다.

```
[mach@localhost jdbc]$ machsql
=================================================================
     Machbase Client Query Utility
     Release Version 3.5.0.826b8f2.official
     Copyright 2014, Machbase Inc. or its subsidiaries.
     All Rights Reserved.
=================================================================
Machbase server address (Default:127.0.0.1):
Machbase rser ID  (Default:SYS)
Machbase user password: MANAGER
MACHBASE_CONNECT_MODE=INET, PORT=5656
mach> create table sample_table(d1 short, d2 integer, d3 long, f1 float, f2 double, name varchar(20), text text, bi
Created successfully.
mach> exit
[mach@localhost jdbc]$
```

```
import java.util.*;
import java.sql.*;
import com.machbase.jdbc.*;
```

```
public class Sample2Insert
{
    public static Connection connect()
    {
        Connection conn = null;
        try
        {

            String sURL = "jdbc:machbase://localhost:5656/mhdb";

            Properties sProps = new Properties();
            sProps.put("user", "sys");
            sProps.put("password", "manager");

            Class.forName("com.machbase.jdbc.driver");

            conn = DriverManager.getConnection(sURL, sProps);

        }
        catch ( ClassNotFoundException ex )
        {
            System.err.println("Exception : unable to load mach jdbc driver class");
        }
        catch ( Exception e )
        {
            System.err.println("Exception : " + e.getMessage());
        }

        return conn;
    }


    public static void main(String[] args) throws Exception
    {
        Connection conn = null;
        Statement stmt = null;
        String sql;

        try
        {
            conn = connect();
            if( conn != null )
            {
                System.out.println("mach JDBC connected.");

                stmt = conn.createStatement();

                for(int i=1; i<10; i++)
                {
                    sql = "INSERT INTO SAMPLE_TABLE VALUES (";
                    sql += (i - 5) * 6552;//short
                    sql += ","+ ((i - 5) * 429496728);//integer
                    sql += ","+ ((i - 5) * 922337203685477580L);//long
                    sql += ","+ 1.23456789+"e"+((i<=5)?"":"+")+((i-5)*7);//float
                    sql += ","+ 1.23456789+"e"+((i<=5)?"":"+")+((i-5)*61);//double
                    sql += ",'id-"+i+"'";//varchar
                    sql += ",'name-"+i+"'";//text
                    sql += ",'aabbccddeeff'";//binary
                    sql += ",'192.168.0."+i+"'";//ipv4
                    sql += ",'::192.168.0."+i+"'";
                    sql += ",TO_DATE('2014-08-0"+i+"','YYYY-MM-DD')";//dt
                    sql += ")";

                    stmt.execute(sql);

                    System.out.println( i+" record inserted.");
                }

                String query = "SELECT d1, d2, d3, f1, f2, name, text, bin, to_hex(bin), v4, v6, to_char(dt,'YYYY-M
                ResultSet rs = stmt.executeQuery(query);
```

```
                while( rs.next () )
                {
                    short d1 = rs.getShort("d1");
                    int d2 = rs.getInt("d2");
                    long d3 = rs.getLong("d3");
                    float f1 = rs.getFloat("f1");
                    double f2 = rs.getDouble("f2");
                    String name = rs.getString("name");
                    String text = rs.getString("text");
                    String bin = rs.getString("bin");
                    String hexbin = rs.getString("to_hex(bin)");
                    String v4 = rs.getString("v4");
                    String v6 = rs.getString("v6");
                    String dt = rs.getString("dt");

                    System.out.print("d1: " + d1);
                    System.out.print(", d2: " + d2);
                    System.out.print(", d3: " + d3);
                    System.out.print(", f1: " + f1);
                    System.out.print(", f2: " + f2);
                    System.out.print(", name: " + name);
                    System.out.print(", text: " + text);
                    System.out.print(", bin: " + bin);
                    System.out.print(", hexbin: "+hexbin);
                    System.out.print(", v4: " + v4);
                    System.out.print(", v6: " + v6);
                    System.out.println(", dt: " + dt);

                }
                rs.close();
            }
        }
        catch( SQLException se )
        {
            System.err.println("SQLException : " + se.getMessage());
        }
        catch( Exception e )
        {
            System.err.println("Exception : " + e.getMessage());
        }
        finally
        {
            if( stmt != null )
            {
                stmt.close();
                stmt = null;
            }
            if( conn != null )
            {
                conn.close();
                conn = null;
            }
        }
    }
}
```

이제 소스 코드를 컴파일하고 실행한다. 이미 작성한 Makefile을 이용한다.

```
[mach@localhost jdbc]$ make
javac -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" -d . Sample1Connect.java Sample2Insert.java Samp
[mach@localhost jdbc]$ make run_sample2
make run_sample2
java -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" Sample2Insert
mach JDBC connected.
1 record inserted.
2 record inserted.
3 record inserted.
4 record inserted.
5 record inserted.
6 record inserted.
```

```
7 record inserted.
8 record inserted.
9 record inserted.
d1: 26208, d2: 1717986912, d3: 3689348814741910320, f1: 1.2345679E28, f2: 1.23456789E244, name: id-9, text: name-9,
d1: 19656, d2: 1288490184, d3: 2767011611056432740, f1: 1.2345678E21, f2: 1.23456789E183, name: id-8, text: name-8,
d1: 13104, d2: 858993456, d3: 1844674407370955160, f1: 1.23456788E14, f2: 1.23456789E122, name: id-7, text: name-7,
d1: 6552, d2: 429496728, d3: 922337203685477580, f1: 1.2345679E7, f2: 1.23456789E61, name: id-6, text: name-6, bin
d1: 0, d2: 0, d3: 0, f1: 1.2345679, f2: 1.23456789, name: id-5, text: name-5, bin: aabbccddeeff, hexbin: 616162626
d1: -6552, d2: -429496728, d3: -922337203685477580, f1: 1.2345679E-7, f2: 1.23456789E-61, name: id-4, text: name-4,
d1: -13104, d2: -858993456, d3: -1844674407370955160, f1: 1.2345679E-14, f2: 1.23456789E-122, name: id-3, text: nam
d1: -19656, d2: -1288490184, d3: -2767011611056432740, f1: 1.2345679E-21, f2: 1.23456789E-183, name: id-2, text: na
d1: -26208, d2: -1717986912, d3: -3689348814741910320, f1: 1.2345679E-28, f2: 1.23456789E-244, name: id-1, text: na
```

데이터 입력 및 출력 예제 (2) PreparedStatement 이용한 입력

PreparedStatement를 이용하여 데이터를 입력하고 출력하는 예제를 작성하여 보기로 한다.

소스 파일명은 Sample3PrepareStmt.java 로 한다.

```java
import java.util.*;
import java.sql.*;
import java.text.SimpleDateFormat;
import com.machbase.jdbc.*;

public class Sample3PrepareStmt
{
    public static Connection connect()
    {
        Connection conn = null;
        try
        {
            String sURL = "jdbc:machbase://localhost:5656/mhdb";

            Properties sProps = new Properties();
            sProps.put("user", "sys");
            sProps.put("password", "manager");

            Class.forName("com.machbase.jdbc.driver");

            conn = DriverManager.getConnection(sURL, sProps);

        }
        catch ( ClassNotFoundException ex )
        {
            System.err.println("Exception : unable to load mach jdbc driver class");
        }
        catch ( Exception e )
        {
            System.err.println("Exception : " + e.getMessage());
        }
        return conn;
    }

    public static void main(String[] args) throws Exception
    {
        Connection conn = null;
        Statement stmt = null;
        machPreparedStatement preStmt = null;
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss SSS");

        try
        {
            conn = connect();
            if( conn != null )
            {
                System.out.println("mach JDBC connected.");

                stmt = conn.createStatement();
```

```
            preStmt = (machPreparedStatement)conn.prepareStatement("INSERT INTO SAMPLE_TABLE VALUES(?, ?, ?, ?,

            String ipStr = null;
            String dateStr = null;
            for(int i=1; i<10; i++)
            {
                ipStr = String.format("172.16.0.%d",i);
                dateStr = String.format("2014-08-09 12:23:34 %03d", i);
                byte[] bin = new byte[20];
                for(int j=0;j<20;j++){
                    bin[j]=(byte)(Math.random()*255);
                }
                java.util.Date day = sdf.parse(dateStr);
                java.sql.Date sqlDate = new java.sql.Date(day.getTime());

                preStmt.setShort(1, (i-5) * 3276 );
                preStmt.setInt(2, (i-5) * 214748364 );
                preStmt.setLong(3, (i-5) * 922337203685477580L );
                preStmt.setFloat(4, 1.23456789101112131415*Math.pow(10,i));
                preStmt.setDouble(5, 1.23456789101112131415*Math.pow(10,i*10));
                preStmt.setString(6, String.format("varchar-%d",i));
                preStmt.setString(7, String.format("text-%d",i));
                preStmt.setBytes(8, bin);
                preStmt.setIpv4(9, ipStr);
                preStmt.setIpv6(10, "::"+ipStr);
                preStmt.setDate(11, sqlDate);
                preStmt.executeUpdate();

                System.out.println( i+" record inserted.");
            }

            //date type format : YYYY-MM-DD HH24:MI:SS mmm:uuu:nnnn
            String query = "SELECT d1, d2, d3, f1, f2, name, text, bin, to_hex(bin), v4, v6, to_char(dt,'YYYY-M
            ResultSet rs = stmt.executeQuery(query);
            while( rs.next () )
            {
                short d1 = rs.getShort("d1");
                int d2 = rs.getInt("d2");
                long d3 = rs.getLong("d3");
                float f1 = rs.getFloat("f1");
                double f2 = rs.getDouble("f2");
                String name = rs.getString("name");
                String text = rs.getString("text");
                String bin = rs.getString("bin");
                String hexbin = rs.getString("to_hex(bin)");
                String v4 = rs.getString("v4");
                String v6 = rs.getString("v6");
                String dt = rs.getString("dt");

                System.out.print("d1: " + d1);
                System.out.print(", d2: " + d2);
                System.out.print(", d3: " + d3);
                System.out.print(", f1: " + f1);
                System.out.print(", f2: " + f2);
                System.out.print(", name: " + name);
                System.out.print(", text: " + text);
                System.out.print(", bin: " + bin);
                System.out.print(", hexbin: "+hexbin);
                System.out.print(", v4: " + v4);
                System.out.print(", v6: " + v6);
                System.out.println(", dt: " + dt);
            }
            rs.close();
        }
    }
    catch( SQLException se )
    {
        System.err.println("SQLException : " + se.getMessage());
    }
    catch( Exception e )
```

```
        {
            System.err.println("Exception : " + e.getMessage());
        }
        finally
        {
            if( stmt != null )
            {
                stmt.close();
                stmt = null;
            }
            if( conn != null )
            {
                conn.close();
                conn = null;
            }
        }
    }
}
```

이제 소스 코드를 컴파일하고 실행본다. 이미 작성한 Makefile을 이용한다.

Sample2Insert.java에서 입력한 데이터가 함께 출력되고 있다는 점에 유의해야 한다.

```
[mach@localhost jdbc]$ make
javac -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" -d . Sample1Connect.java
Sample2Insert.java Sample3PrepareStmt.java Sample4Append.java
[mach@localhost jdbc]$ make run_sample3
make run_sample3
java -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" Sample3PrepareStmt
Mach JDBC connected.
1 record inserted.
2 record inserted.
3 record inserted.
4 record inserted.
5 record inserted.
6 record inserted.
7 record inserted.
8 record inserted.
9 record inserted.
d1: 13104, d2: 858993456, d3: 3689348814741910320, f1: 754454.6, f2: 453821.380752063, name:
varchar-9, text: text-9, bin: ?+,??r?J?????S)n?, hexbin:
A4C9A8D491D6728B4AACB39EE5FC5300296EFA9F, v4: 172.16.0.9, v6: 0:0:0:0:0:0:ac10:9, dt:
2014-08-09 12:23:34 009:000:000
?h???a?, hexbin: 6C20F09329ABBA3E7DE501C30DA368D6EFC961EF, v4: 172.16.0.8, v6:
0:0:0:0:0:0:ac10:8, dt: 2014-08-09 12:23:34 008:000:000
d1: 6552, d2: 429496728, d3: 1844674407370955160, f1: 2664182.0, f2: 1357910.1926900472, name:
varchar-7, text: text-7, bin: ????Uls?q?H?I?&(?, hexbin:
B5A0A2EFA185556C73BF719448BD49C92628F8C6, v4: 172.16.0.7, v6: 0:0:0:0:0:0:ac10:7, dt:
2014-08-09 12:23:34 007:000:000
d1: 3276, d2: 214748364, d3: 922337203685477580, f1: 443847.1, f2: 9342855.256576871, name:
varchar-6, text: text-6, bin: ??>x??Eu?? ?Iw??+n, hexbin:
BC973E78F5B44575D6CC15F94977DAE62B6E1D0E, v4: 172.16.0.6, v6: 0:0:0:0:0:0:ac10:6, dt:
2014-08-09 12:23:34 006:000:000
d1: 0, d2: 0, d3: 0, f1: 1283723.1, f2: 1771261.2019240903, name: varchar-5, text: text-5,
bin: &== j?j3?? T??y?
??, hexbin: 263D3D1C6AF56A33F79D0C54A5C479A4030AFE8B, v4: 172.16.0.5, v6: 0:0:0:0:0:0:ac10:5,
dt: 2014-08-09 12:23:34 005:000:000
d1: -3276, d2: -214748364, d3: -922337203685477580, f1: 9447498.0, f2: 7529392.937964935,
name: varchar-4, text: text-4, bin: ?Sw ??)? ?h2?E??/?, hexbin:
C653771DD2DF29CDB30ED96832E745D3D7A52FD2, v4: 172.16.0.4, v6: 0:0:0:0:0:0:ac10:4, dt:
2014-08-09 12:23:34 004:000:000
d1: -6552, d2: -429496728, d3: -1844674407370955160, f1: 9589634.0, f2: 5994172.201347323,
name: varchar-3, text: text-3, bin: 9aB,.????L/?=3,?`?f, hexbin:
3961422C2EA39BE6F2964C2FCD3D332C8960A466, v4: 172.16.0.3, v6: 0:0:0:0:0:0:ac10:3, dt:
2014-08-09 12:23:34 003:000:000
d1: -9828, d2: -644245092, d3: -2767011611056432740, f1: 7409537.5, f2: 2313739.6613546023,
name: varchar-2, text: text-2, bin: _? N?3 ?? ??~H ??= 8, hexbin:
5F84144EF63320F3C718B0FD7E4809A4CB3D1838, v4: 172.16.0.2, v6: 0:0:0:0:0:0:ac10:2, dt:
2014-08-09 12:23:34 002:000:000
d1: -13104, d2: -858993456, d3: -3689348814741910320, f1: 596626.75, f2: 2649492.1936065694,
```

```
name: varchar-1, text: text-1, bin: ???d??Wu$v? 7m?-, hexbin:
E8D0C564B4EB57E59B08752476FC07376DBF2D14, v4: 172.16.0.1, v6: 0:0:0:0:0:0:ac10:1, dt:
2014-08-09 12:23:34 001:000:000
d1: 26208, d2: 1717986912, d3: 3689348814741910320, f1: 1.2345679E28, f2: 1.23456789E244,
name: id-9, text: name-9, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4:
192.168.0.9, v6: 0:0:0:0:0:0:c0a8:9, dt: 2014-08-09 00:00:00 000:000:000
d1: 19656, d2: 1288490184, d3: 2767011611056432740, f1: 1.2345678E21, f2: 1.23456789E183,
name: id-8, text: name-8, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4:
192.168.0.8, v6: 0:0:0:0:0:0:c0a8:8, dt: 2014-08-08 00:00:00 000:000:000
d1: 13104, d2: 858993456, d3: 1844674407370955160, f1: 1.23456788E14, f2: 1.23456789E122,
name: id-7, text: name-7, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4:
192.168.0.7, v6: 0:0:0:0:0:0:c0a8:7, dt: 2014-08-07 00:00:00 000:000:000
d1: 6552, d2: 429496728, d3: 922337203685477580, f1: 1.2345679E7, f2: 1.23456789E61, name:
id-6, text: name-6, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4: 192.168.0.6, v6:
0:0:0:0:0:0:c0a8:6, dt: 2014-08-06 00:00:00 000:000:000
d1: 0, d2: 0, d3: 0, f1: 1.2345679, f2: 1.23456789, name: id-5, text: name-5, bin:
aabbccddeeff, hexbin: 616162626363646465656666, v4: 192.168.0.5, v6: 0:0:0:0:0:0:c0a8:5, dt:
2014-08-05 00:00:00 000:000:000
d1: -6552, d2: -429496728, d3: -922337203685477580, f1: 1.2345679E-7, f2: 1.23456789E-61,
name: id-4, text: name-4, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4:
192.168.0.4, v6: 0:0:0:0:0:0:c0a8:4, dt: 2014-08-04 00:00:00 000:000:000
d1: -13104, d2: -858993456, d3: -1844674407370955160, f1: 1.2345679E-14, f2: 1.23456789E-122,
name: id-3, text: name-3, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4:
192.168.0.3, v6: 0:0:0:0:0:0:c0a8:3, dt: 2014-08-03 00:00:00 000:000:000
d1: -19656, d2: -1288490184, d3: -2767011611056432740, f1: 1.2345679E-21, f2: 1.23456789E-183,
name: id-2, text: name-2, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4:
192.168.0.2, v6: 0:0:0:0:0:0:c0a8:2, dt: 2014-08-02 00:00:00 000:000:000
d1: -26208, d2: -1717986912, d3: -3689348814741910320, f1: 1.2345679E-28, f2: 1.23456789E-244,
name: id-1, text: name-1, bin: aabbccddeeff, hexbin: 616162626363646465656666, v4:
192.168.0.1, v6: 0:0:0:0:0:0:c0a8:1, dt: 2014-08-01 00:00:00 000:000:000
```

확장 함수 Append 예제

마크베이스 JDBC 드라이버는 많은 건수의 데이터를 빠르게 업로드하기 위한 Append 프로토콜을 지원한다.

다음은 Append 프로토콜 사용 예제이다.
이전 예제에 사용된 sample_table을 그대로 이용한다.

소스 파일명은 Sample4Append.java 라고 한다.
data.txt에 있는 내용을 sample_table에 입력한다.
CLI append 예제에 이용한 data.txt 파일을 복사하여 사용하기로 한다.

```java
import java.util.*;
import java.sql.*;
import java.io.*;
import java.text.SimpleDateFormat;
import java.math.BigDecimal;
import com.machbase.jdbc.*;


public class Sample4Append
{
    protected static final String sTableName = "sample_table";
    protected static final int sErrorCheckCount = 100;

    public static Connection connect()
    {
        Connection conn = null;
        try
        {
            String sURL = "jdbc:machbase://localhost:5656/mhdb";

            Properties sProps = new Properties();
            sProps.put("user", "sys");
            sProps.put("password", "manager");

            Class.forName("com.machbase.jdbc.driver");

            conn = DriverManager.getConnection(sURL, sProps);
```

```
        }
        catch ( ClassNotFoundException ex )
        {
            System.err.println("Exception : unable to load mach jdbc driver class");
        }
        catch ( Exception e )
        {
            System.err.println("Exception : " + e.getMessage());
        }
        return conn;
    }

    public static void main(String[] args) throws Exception
    {
        Connection conn = null;
        MachStatement stmt = null;
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Calendar cal = Calendar.getInstance();
        String filename = "data.txt";

        try
        {
            conn = connect();
            if( conn != null )
            {
                System.out.println("Mach JDBC connected.");

                stmt = (MachStatement)conn.createStatement();

                ResultSet rs = stmt.executeAppendOpen(sTableName, sErrorCheckCount);
                ResultSetMetaData rsmd = rs.getMetaData();

                System.out.println("append open ok");

                MachAppendCallback cb = new MachAppendCallback() {
                        @Override
                        public void onAppendError(long aErrNo, String aErrMsg, String aRowMsg) {
                             System.out.format("Append Error : [%05d - %s]\n%s\n", aErrNo, aErrMsg, aRowMsg);
                        }
                    };

                stmt.executeSetAppendErrorCallback(cb);

                System.out.println("append data start");
                BufferedReader in = new BufferedReader(new FileReader(filename));
                String buf = null;
                int cnt = 0;
                long dt;

                long startTime = System.nanoTime();

                while( (buf = in.readLine()) != null )
                {
                    ArrayList<Object> sBuf = new ArrayList<Object>();
                    StringTokenizer st = new StringTokenizer(buf,",");
                    for(int i=0; st.hasMoreTokens() ;i++ )
                    {
                        switch(i){
                            case 7://binary case
                                sBuf.add(new ByteArrayInputStream(st.nextToken().getBytes())); break;
                            case 10://date case
                                java.util.Date day = sdf.parse(st.nextToken());
                                cal.setTime(day);
                                dt = cal.getTimeInMillis()*1000000; //make nanotime
                                sBuf.add(dt);
                                break;
                            default:
                                sBuf.add(st.nextToken()); break;
                        }
                    }
```

```
                    if( stmt.executeAppendData(rsmd, sBuf) != 1 )
                    {
                        System.err.println("Error : AppendData error");
                        break;
                    }

                    if( (cnt++%10000) == 0 )
                    {
                        System.out.print(".");
                    }
                    sBuf = null;

                }
                System.out.println("\nappend data end");

                long endTime = System.nanoTime();
                stmt.executeAppendClose();
                System.out.println("append close ok");
                System.out.println("Append Result : success = "+stmt.getAppendSuccessCount()+", failure = "+stmt.ge
                System.out.println("timegap " + ((endTime - startTime)/1000) + " in microseconds, " + cnt + " recor

                try {
                    BigDecimal records = new BigDecimal( cnt );
                    BigDecimal gap = new BigDecimal( (double)(endTime - startTime)/1000000000 );
                    BigDecimal rps = records.divide(gap, 2, BigDecimal.ROUND_UP );

                    System.out.println( rps + " records/second" );
                } catch(ArithmeticException ae) {
                    System.out.println( cnt + " records/second");
                }

                rs.close();
            }
        }
        catch( SQLException se )
        {
            System.err.println("SQLException : " + se.getMessage());
        }
        catch( Exception e )
        {
            System.err.println("Exception : " + e.getMessage());
        }
        finally
        {
            if( stmt != null )
            {
                stmt.close();
                stmt = null;
            }
            if( conn != null )
            {
                conn.close();
                conn = null;
            }
        }
    }
}
```

Append를 할 때 date 타입 데이터는 반드시 long 타입의 나노초 단위 시간으로 변환하여 전송하여야 한다.

```
[mach@localhost jdbc]$ make run_sample4
make run_sample4
java -classpath ".:/home/machbase/machbase_home/lib/machbase.jar" Sample4Append;
Mach JDBC connected.
append open ok
append data start
......
append data end
append close ok
```

```
Append Result : success = 60000, failure = 0
timegap 6905594 in microseconds, 60000 records
8688.61 records/second
```

10,000건마다 점(.)을 표시하고 있으며, 입력 소요 시간을 알 수 있다.

```
# machsql을 이용하여 실제 입력된 건수를 확인해보자.
# Sample2Insert,Sample3PrepareStmt에서 입력한 건수와 함께 60018건이 입력된 것을 확인한다.


[mach@localhost jdbc]$ machsql
=================================================================
     Machbase Client Query Utility
     Release Version 3.0.0
     Copyright 2014, Machbase Inc. or its subsidiaries.
     All Rights Reserved.
=================================================================
Machbase server address (Default:127.0.0.1):
Machbase user ID  (Default:SYS)
Machbase user password: MANAGER
MACH_CONNECT_MODE=INET, PORT=5656
mach> select count(*) from sample_table;
count(*)
-----------------------
60018
[1] row(s) selected.
```

Python

목차

Python 모듈 사용 개요

마크베이스에서는 Python 모듈을 지원한다. 모듈을 설치함으로써 마크베이스 서버와 CLI 방식으로 값을 주고 받을 수 있는 클래스를 제공한다. 이를 활용해서 Python에서 쉽게 마크베이스에 질의 형태로 값을 입력하거나 삭제, 테이블 생성, 삭제 등 다양한 명령어를 사용할 수 있다.

사용 환경 설정

리눅스

이를 사용하기 위해서는 간단한 환경 설정과 라이브러리 파일들이 필요하다. 먼저 `$LD_LIBRARY_PATH`에 `$MACHBASE_HOME/lib` 디렉터리가 등록되어있는지 확인한다. `CLI`를 이용해서 마크베이스에 접속하기 때문에 `libmachbasecli_dll.so` 파일이 라이브러리 폴더에 존재해야 한다. 그리고 `$MACHBASE_HOME/3rd-party/python3-module` 폴더에 있는 `machbaseAPI-1.0.tar.gz`를 압축 해제후 `python setup.py install` 명령을 통해 사용하려는 Python에 모듈을 설치한다.

윈도우

먼저 `%MACHBASE_HOME%/3rd-party/python3-module` 폴더에 있는 `machbaseAPI-1.0.zip`을 압축 해제 후 `python setup.py install` 명령을 통해 사용하려는 Python에 모듈을 설치한다.

Class 생성

마크베이스 Python 모듈을 사용하기 위해서 해당 클래스를 선언해야 한다.

해당 클래스 명은 machbase이다.

```
from machbaseAPI import machbase
```

접속과 접속해제

machbase.open(aHost, aUser, aPw, aPort)

마크베이스에 접속하는 함수이다. 알맞은 파라미터 값을 입력했을시에 DB에 접속 성공했는지 실패했는지를 반환한다. 정상 종료시 1, 실패시 0을 반환한다.

machbase.close()

마크베이스 접속을 해제하는 함수이다. 정상 종료시 1, 실패시 0을 반환한다.

machbase.isConnected()

선언한 클래스가 해당 서버에 접속중인지 아닌지를 판별하는 함수이다. 접속 중일 때 1, 접속 중이 아닐 때 0을 반환한다.

명령어 실행 및 사용자 편의 함수

machbase.execute(aSql)

서버에 접속되어 있을 때 해당 서버에 질의문을 전송하는 명령어이다. 정상적으로 실행되었을 때 1, 실패하거나 에러가 발생했을 시에는 0을 반환한다.

마크베이스에서 지원하지 않는 UPDATE를 제외하고 모든 명령어들을 사용할 수 있다.

결과를 한꺼번에 return하는 구조이기 때문에 SELECT 구문을 사용하는 경우 메모리 부족 오류가 발생할 수 있다. 따라서 SELECT 구문은 machbase.select() 함수를 사용한다.

machbase.append(aTableName, aTypes, aValues, aFormat)

마크베이스에서 지원하는 Append 프로토콜을 사용할 수 있는 함수이다.

데이터를 입력하게 될 테이블명, 각 컬럼들의 타입의 딕셔너리, 그리고 입력할 값들을 JSON 형태로 입력하고 dateformat을 지정해 주면 Append를 실행할 수 있다.

정상적으로 실행하였다면 1, 실패시에는 0을 반환한다.

타입명	값
short	4
ushort	104

타입명	값
integer	8
uinteger	108
long	12
ulong	112
float	16
double	20
datetime	6
varchar	5
ipv4	32
ipv6	36
text	49
binary	97

machbase.tables()

접속한 서버에 있는 모든 테이블들의 정보를 반환한다. 정상적으로 실행되었다면 1, 실패했을 시에는 0을 반환한다.

machbase.columns(aTableName)

접속한 서버에 있는 해당 테이블 내의 컬럼들의 정보를 반환한다. 정상적으로 실행되었다면 1, 실패했을 시에는 0을 반환한다.

결과 확인

마크베이스 Python 모듈에서 모든 결과값은 JSON으로 반환된다.

다양한 환경에서 활용하기 쉬운 형태의 결과를 반환하는 것으로 채택하였다.

machbase.result()

위쪽에서 설명하였던 함수들은 실행 결과들을 해당 함수의 반환값으로 나타내지 않고 성공, 실패 여부만 반환한다. 함수들의 결과값은 이 함수의 반환값으로만 얻을 수 있다.

SELECT 결과 확인

machbase.execute() 함수로 SELECT 결과를 읽어오면 모든 결과를 한번에 읽기때문에 많은 메모리가 필요하며 경우에 따라서 메모리 부족 오류가 발생할 수 있다.

따라서 SELECT 구문의 경우에는 한 레코드씩 결과를 얻을 수 있는 함수를 사용해야 한다.

machbase.select(aSql)

서버에 접속되어 있을 때 해당 서버에 SELECT 질의문을 전송하는 명령어이다. 정상적으로 실행되었을 때 1, 실패하거나 에러가 발생했을 시에는 0을 반환한다.

SELECT 구문만 사용이 가능하며, 모든 결과를 한번에 가져오는 machbase.execute() 함수와 달리 machbase.fetch()함수를 사용해서 1 레코드 씩 결과를 얻을 수 있다.

machbase.fetch()

machbase.select(aSql) 함수의 결과를 한 레코드 씩 읽어온다.

return 값은 성공여부와 결과값으로 성공여부는 1과 0으로 표시되고, 결과값은 한 레코드의 결과가 json 형식으로 return 된다.

machbase.selectClose()

한번 선택된 machbase.select()의 결과는 다른 machbase.select()나 machbase.execute() 함수가 호출될 때까지 open된 상태로 존재한다.

이 것을 명시적으로 닫아주는 것이 machbase.selectClose() 함수이다.

machbase.select() 예제

```
db = machbase()
if db.open('127.0.0.1','SYS','MANAGER', 5656) is 0 :
    return db.result()

query = 'SELECT * from sample_table'
while True:
```

```
    is_success, result = db.fetch()
    if is_success is 0:
        break;

    res = json.loads(result)

db.selectClose()
if db.close() is 0 :
    return db.result()
```

예제

간단한 예제들을 통해서 마크베이스 Python 모듈을 사용하는 방법을 알아보자

$MACHBASE_HOME/sample/python 파일들을 이용해서 확인할 수 있다. 해당 디렉터리에는 간편하게 테스트를 해 볼 수 있게 해주는 Makefile과 데이터를 만들어주는 MakeData.py 파일이 있다. Makefile 내부 변수 중 PYPATH의 값을 마크베이스 Python 모듈이 설치된 파이썬으로 지정해야 정상 작동한다. 기본값은 마크베이스 패키지에 설치된 파이썬으로 지정되어 있다. 또한 파이썬에서 모듈을 독자적으로 실행하기 위해서는 __init__.py 파일이 필요하므로 해당 디렉터리에 파일이 존재하는지 확인하도록 하다.

```
[mach@localhost]$ cd $MACHBASE_HOME/sample/python
[mach@localhost python]$ ls -l
total 20
-rw-rw-r-- 1 mach mach    0 Oct  7 14:37 __init__.py
-rw-rw-r-- 1 mach mach  764 Oct  7 14:37 MakeData.py
-rw-rw-r-- 1 mach mach  593 Oct  7 14:58 Makefile
-rw-rw-r-- 1 mach mach  664 Oct  7 14:37 Sample1Connect.py
-rw-rw-r-- 1 mach mach 2401 Oct  7 14:37 Sample2Simple.py
-rw-rw-r-- 1 mach mach 1997 Oct  7 14:37 Sample3Append.py
```

Connect

아래의 예제는 서버에 접속해서 질의를 실행하고 결과값을 반환하는 단순한 함수이다. 각각의 함수들이 실패(0)를 반환했을 때에 결과값을 반환하는 경우는 에러 결과를 반환하기 위함이다. 정상적으로 실행된다면 m$tables 테이블의 값들 개수가 반환 된다.

파일 이름은 Sample1Connect.py이다.

```
from machbaseAPI import machbase
def connect():
    db = machbase()
    if db.open('127.0.0.1','SYS','MANAGER',5656) is 0 :
        return db.result()
    if db.execute('select count(*) from m$tables') is 0 :
        return db.result()
    result = db.result()
    if db.close() is 0 :
        return db.result()
    return result
if __name__=="__main__":
    print connect()
```

```
[mach@localhost python]$ make run_sample1
/home/machbase/machbase_home/webadmin/flask/Python/bin/python Sample1Connect.py
{"count(*)":"13"}
```

Simple

아래 예제를 이용해서 단순하게 마크베이스에 파이썬을 이용해서 테이블을 만들고 값을 입력하고 입력된 값을 추출해서 확인하는 예제이다. 파일 이름은 Sample2Simple.py이다.

```
import re
import json
from machbaseAPI import machbase
def insert():
    db = machbase()
    if db.open('127.0.0.1','SYS','MANAGER',5656) is 0 :
```

```
        return db.result()
    db.execute('drop table sample_table')
    db.result()
    if db.execute('create table sample_table(d1 short, d2 integer, d3 long, f1 float, f2 double, name varchar(20),
        return db.result()
    db.result()
    for i in range(1,10):
        sql = "INSERT INTO SAMPLE_TABLE VALUES ("
        sql += str((i - 5) * 6552) #short
        sql += ","+ str((i - 5) * 42949672) #integer
        sql += ","+ str((i - 5) * 92233720368547758L) #long
        sql += ","+ "1.234"+str((i-5)*7) #float
        sql += ","+ "1.234"+str((i-5)*61) #double
        sql += ",'id-"+str(i)+"'" #varchar
        sql += ",'name-"+str(i)+"'" #text
        sql += ",'aabbccddeeff'" #binary
        sql += ",'192.168.0."+str(i)+"'" #ipv4
        sql += ",'::192.168.0."+str(i)+"'" #ipv6
        sql += ",TO_DATE('2015-08-0"+str(i)+"','YYYY-MM-DD')" #date
        sql += ")";
        if db.execute(sql) is 0 :
            return db.result()
        else:
            print db.result()
        print str(i)+" record inserted."
    query = "SELECT d1, d2, d3, f1, f2, name, text, bin, to_hex(bin), v4, v6, to_char(dt,'YYYY-MM-DD') as dt from S
    if db.execute(query) is 0 :
        return db.result()
    result = db.result()
    for item in re.findall('{[^}]+}',result):
        res = json.loads(item)
        print "d1 : "+res.get('d1')
        print "d2 : "+res.get('d2')
        print "d3 : "+res.get('d3')
        print "f1 : "+res.get('f1')
        print "f2 : "+res.get('f2')
        print "name : "+res.get('name')
        print "text : "+res.get('text')
        print "bin : "+res.get('bin')
        print "to_hex(bin) : "+res.get('to_hex(bin)')
        print "v4 : "+res.get('v4')
        print "v6 : "+res.get('v6')
        print "dt : "+res.get('dt')
    if db.close() is 0 :
        return db.result()
    return result
if __name__=="__main__":
    print insert()
```

```
[mach@loclahost python]$ make run_sample2
/home/machbase/machbase_home/webadmin/flask/Python/bin/python Sample2Simple.py
{"EXECUTE RESULT":"Execute Success"}
1 record inserted.
{"EXECUTE RESULT":"Execute Success"}
2 record inserted.
{"EXECUTE RESULT":"Execute Success"}
3 record inserted.
{"EXECUTE RESULT":"Execute Success"}
4 record inserted.
{"EXECUTE RESULT":"Execute Success"}
5 record inserted.
{"EXECUTE RESULT":"Execute Success"}
6 record inserted.
{"EXECUTE RESULT":"Execute Success"}
7 record inserted.
{"EXECUTE RESULT":"Execute Success"}
8 record inserted.
{"EXECUTE RESULT":"Execute Success"}
9 record inserted.
```

```
d1 : 26208
d2 : 171798688
d3 : 368934881474191032
f1 : 1.23428
f2 : 1.23424
name : id-9
text : name-9
bin : 616162626363646465656666
to_hex(bin) : 6161626263636464656566666
v4 : 192.168.0.9
v6 : ::192.168.0.9
...
```

Append

마크베이스에서 고속으로 데이터를 입력할 수 있는 Append 방식 또한 파이썬 모듈을 활용해서 사용할 수 있다. 아래의 예제는 고속으로 데이터를 입력하는 예제이다. 컬럼 정보 및 초기화를 위한 접속 클래스 db, Append를 하기 위한 접속용 클래스 db2를 선언하여 각각의 함수를 활용하는 방식을 사용했다. 파일 이름은 Sample3Append.py이다.

```
import re
import json
from machbaseAPI import machbase
def append():
#init,columns start
    db = machbase()
    if db.open('127.0.0.1','SYS','MANAGER',5656) is 0 :
        return db.result()
    db.execute('drop table sample_table')
    db.result()
    if db.execute('create table sample_table(d1 short, d2 integer, d3 long, f1 float, f2 double, name varchar(20),
        return db.result()
    db.result()
    tableName = 'sample_table'
    db.columns(tableName)
    result = db.result()
    if db.close() is 0 :
        return db.result()
#init, colums end
#append start
    db2 = machbase()
    if db2.open('127.0.0.1','SYS','MANAGER',5656) is 0 :
        return db2.result()
    types = []
    for item in re.findall('{[^}]+}',result):
        types.append(json.loads(item).get('type'))
    values = []
    with open('data.txt','r') as f:
        for line in f.readlines():
            v = []
            i = 0
            for l in line[:-1].split(','):
                t = int(types[i])
                if t == 4 or t == 8 or t == 12 or t == 104 or t == 108 or t == 112:
                    #short   integer    long       ushort      uinteger     ulong
                    v.append(int(l))
                elif t == 16 or t == 20:
                    #float      double
                    v.append(float(l))
                else:
                    v.append(l)
                i+=1
            values.append(v)
    db2.append(tableName, types, values, 'YYYY-MM-DD HH24:MI:SS')
    result = db2.result()
    if db2.close() is 0 :
        return db2.result()
#append end
    return result
```

```
if __name__=="__main__":
    print append()
```

```
[mach@localhost python]$ make run_sample3
/home/machbase/machbase_home/webadmin/flask/Python/bin/python Sample3Append.py
{"EXECUTE RESULT":"Append success"}
```

RESTful API

RESTful API 개요

Representational State Transfer (REST) 는 소프트웨어 구조 스타일 의 일종으로, 확장 가능한 웹 서비스에서 제공하는 인터페이스의 가이드라인과 모범적인 규범들로 구성되어 있다.

HTTP protocol에 정의된 4개의 Method 들이 Resource에 대한 CRUD를 정의한다.

HTTP Method	의미
POST	Create
GET	Select
PUT	Update
DELETE	Delete

마크베이스는 표준 RESTful API 방식이 아니라, POST와 GET method만을 이용하여 CRUD를 처리하는 방식으로 RESTful API라고 할 수 있다.

즉, 데이터 입력에는 POST method를 사용하고 나머지는 SQL query를 GET Method parameter로 전달하여 모든 작업을 할 수 있도록 구성되어 있다.

목차

Machbase RestAPI

Machbase에 웹서버를 내장하여 별도의 서버 구동 없이 Machbase에 직접 RestAPI를 수행하는 기능이다.

RestAPI 지원 Machbase Edition

Edge / Fog / Cluster

RestAPI 지원 Table 종류

Tag Table / Log Table / Lookup Table / Volatile Table

Configuration 설정

machbase.conf 와 http.conf 두 가지 설정 파일이 존재한다.

- machbase.conf : Rest API를 사용하기 위해 HTTP 서버를 사용할 지와 최대 메모리 사용량 등 HTTP Server에 대한 설정을 저장
- http.conf : HTTP Web Server 자체에 대한 설정을 저장

설정 파일을 수정 하면 machbase 서버를 다시 시작해야 변경 내용이 적용된다.

버전별 .conf 파일의 위치

Edge / Fog 버전

$MACHBASE_HOME/conf/machbase.conf

$MACHBASE_HOME/http/conf/http.conf

Cluster 버전

EACH_BROKER_HOME/conf/machbase.conf (Broker 별로 모두 수정)

EACH_BROKER_HOME/http/conf/http.conf (Broker 별로 모두 수정)

각 .conf 파일 Property 설명

machbase.conf (PROPERTY = VALUE 형태로 설정)

Property	설명
HTTP_ENABLE	내장 웹 서버를 구동할 지 여부 0 : 구동 안함, 1 : 구동
HTTP_PORT_NO	내장 웹 서버 접속 Port 번호 Port 범위 : 0 ~ 65535 Default : 5657
HTTP_MAX_MEM	하나의 Web Session에서 사용할 최대 메모리 Min : 1048576 (1MB) Default : 536870912 (512MB)
HTTP_AUTH	내장 웹 서버 사용 시 기본 인증을 사용할 지 여부 0 : 인증 사용 안함, 1 : 인증 사용함

http.conf (JSON 형식으로 설정)

Property	설명
document_root	$MACHBASE_HOME 기준의 html 파일 위치 Default : http/html ($MACHBASE_HOME/http/html)
max_request_size	1회 요청의 최대 요청 byte 크기 제한
request_timeout_ms	1회 요청의 최대 응답 대기 시간 (millisecond)
enable_auth_domain_check	도메인 인증을 활성화 할지 여부 "yes" or "no" 값으로 설정 Default : "no"
reverse_proxy	요청 URL을 특정 URL로 변경 참고: https://brainbackdoor.tistory.com/113

각 .conf 파일 Sample

machbase.conf

```
##############################################################################
# Rest-API port
##############################################################################
HTTP_PORT_NO = 5657

##############################################################################
# Maximum memory per web session.
# Default Value: 536870912 (512MB)
##############################################################################
HTTP_MAX_MEM = 536870912

##############################################################################
# Min Value:     0
# Max Value:     1
# Default Value: 0
#
# Enable REST-API service.
##############################################################################
HTTP_ENABLE = 0

##############################################################################
# Min Value:     0
# Max Value:     1
# Default Value: 0
#
```

```
# Enable Basic Authentication for Rest-API service
###############################################################################
HTTP_AUTH = 0
```

http.conf

```
{
    "document_root":"http/html/",
    "max_request_size": "100000",
    "request_timeout_ms": "10000",
    "enable_auth_domain_check": "no",
    "reverse_proxy" : [["/machbase/tables", "http://127.0.0.1:55657/machbase"],
        ["/self_machbase_proxy", "http://127.0.0.1:55657/machbase"],
        ["/dead_proxy", "http://127.0.0.0/machbase"]]
}
```

RestAPI 사용

DDL / DML / Append 수행 가능

```
기본 요청 형식

http://addr:port/machbase?q=query&f=dateformat

응답 형식 DDL / Append / DML (except Select)

{"error_code":0, "error_message" :"Message", "data":[]}

응답 형식 DML (Select)

{"error_code":0, "error_message" :"Message", "columns":[Columns], "data":[Data]}
```

DDL Sample

```
## 테이블 생성 요청
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=create table test_table (name varchar(20), time datet

## 정상 응답
{"error_code":0, "error_message" :"No Error", "data":[]}

## 테이블 삭제 요청
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=drop table test_table'

## 정상 응답
{"error_code":0, "error_message" :"No Error", "data":[]}
```

DML Sample

```
## 로그 테이블 입력(Insert) 요청
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=insert into test_table values ("test", "1999-01-01 00

## 정상 응답
{"error_code":0, "error_message" :"No Error", "data":[]}

## 로그 테이블 조회 요청
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=select * from test_table'

## 정상 응답
{"error_code":0, "error_message": "", "columns" : [{"name":"NAME", "type":5, "length":20},{"name":"TIME", "type":6,
```

APPEND Sample

```
## 로그 테이블 입력(Append) 요청
curl -X POST -H "Content-Type: application/json" "http://127.0.0.1:5657/machbase" -d '{"name":"test_table", "date_t

## 정상 응답
{"error_code":0, "error_message" :"No Error", "data":[], "append_success":3, "append_failure":0}
```

Binary Append

Binary Append의 경우 Binary 데이터를 Base64로 인코딩 후 전송하면 서버에서 디코딩 후 저장하게 된다.

출력 시에는 바이너리 데이터가 Base64로 인코딩되어 반환된다.

입력 : Binary Data >> Base64 Encoding >> HTTP(POST) >> Base64 Decoding >> Append(BLOB Binary)

출력 : BLOB Binary >> Base64 Encoding >> HTTP (GET) >> Base64 Decoding >> Save or View Binary

Binary Append Sample

```
## 00 ~ FF까지의 256바이트 바이너리 데이터를 Base64로 인코딩 후 입력 / 출력 예

## 로그 테이블 입력(Append) 요청
curl  -X POST -H "Content-Type: application/json" "http://127.0.0.1:5657/machbase" -d '{"name":"test_table", "date_

## 정상 응답
{"error_code":0, "error_message" :"No Error", "data":[], "append_success":1, "append_failure":0}

## 로그 테이블 출력 요청
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=select * from test_table';

## Base64 데이터 출력
{"error_code" :0, "error_message": "No Error", "columns" : [{"name":"V1", "type":57, "length":67108864}],"data" :[

## machsql을 통한 HEX Dump 값 확인
select to_hex(v1) from test_table;
to_hex(v1)
--------------------------------------------------------------------------------
000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20212223242526 27
28292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F404142434445464748494A4B4C4D4E4F
505152535455565758595A5B5C5D5E5F606162636465666768696A6B6C6D6E6F7071727374757677
78797A7B7C7D7E7F808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F
A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBFC0C1C2C3C4C5C6C7
C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDFE0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF
F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
[1] row(s) selected.
```

HTTP_AUTH Property 사용

Request Header에 'Authorization: Basic Base64String' 문자열을 포함하여 정상 유저임을 인증하도록 설정하는 옵션이다.

Base64 문자열은 ID@Host:Password 구조로 작성한다. (단, Host명은 정확하지 않아도 된다. ID, Password는 Machbase의 유저 정보를 입력해야 한다.)

인증을 위한 Basic Base64String 생성 방법

```
## ID: sys, Password: manager 일 경우의 생성 예
echo -n "sys@localhost:manager" | base64

## 생성된 Base64String
c3lzQGxvY2FsaG9zdDptYW5hZ2Vy
```

Base64String 사용 Sample (HTTP_AUTH = 1 인 경우)

```
## Authorization을 넣지 않은 요청의 예
```

```
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=select * from v$stmt'

## 에러 발생
{"error_code":3118, "error_message" :"There is No Authorization Header.", "data":[]}

## Request Header에 'Authorization:Base64String' 추가한 요청의 예
curl -H "Authorization: Basic c3lzQGxvY2FsaG9zdDptYW5hZ2Vy"  -G "http://127.0.0.1:5657/machbase" --data-urlencode

## 정상 응답
{"error_code":0, "error_message": "No Error", "columns" : [{"name":"ID", "type":8, "length":4},{"name":"SESS_ID", "
```

출력 소수점 Scale 지정 (s 옵션)

응답 데이터의 소수점을 몇자리까지 출력할 지 지정

0 ~ 9 값으로 설정 (범위 값이 아닐 경우 3으로 동작)

소수점 5자리까지 출력 Sample (s=5)

```
## 소수점 5자리까지 출력
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=select * from test_table' --data-urlencode 's=5';

## 정상 응답
{"error_code" :0, "error_message": "", "columns" : [{"name":"C1", "type":16, "length":4},{"name":"C2", "type":20, "
```

데이터 Fetch 모드 변경 (m 옵션)

응답 data에 column 이름을 항상 붙여서 표시할 지 결정 (0 : 표시, 1 : 표시 안함)

Default Fetch mode Sample (m=0)

```
## fetch mode (m=0) 요청
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=select * from tag limit 2' --data-urlencode 'm=0';

## 정상 응답 (Column Name을 포함한 결과 출력)
{"error_code" :0, "error_message": "", "columns" : [{"name":"NAME", "type":5, "length":20},{"name":"TIME", "type":6
"data" :[{"NAME":"tag1", "TIME":"2001-09-09 10:46:40 000:000:000", "VALUE":1000000000.000}, {"NAME":"tag1", "TIME"
```

Advanced Fetch mode Sample (m=1)

```
## fetch mode (m=1) 요청
curl -G "http://127.0.0.1:5657/machbase" --data-urlencode 'q=select * from tag limit 2' --data-urlencode 'm=1';

## 정상 응답 (Column Name이 생략된 결과 출력)
{"error_code" :0, "error_message": "", "columns" : [{"name":"NAME", "type":5, "length":20},{"name":"TIME", "type":6
"data" :[["tag1", "2001-09-09 10:46:40 000:000:000", 1000000000.000], ["tag1", "2001-09-09 10:46:41 000:000:000", 1
```

NULL 값의 처리

DML의 처리 중 Insert나 Append 시 NULL 값은 그대로 null로 입력하면 된다.

Append 시 NULL 값을 포함한 JSON Sample

```
[["data1", "data2", "data3"],["data11", "data12", "data13"],["data21", "data22", "data23"],[null,null,null]]
```

Select 시의 NULL 값 포함 결과 Sample

```
[{"C1":null, "C2":null, "C3":null, "C4":null, "C5":null, "C6":null, "C7":null, "C8":null, "C9":null, "C10":null, "C
```

RestAPI for Tag Table 사용

Tag table에 접근할 수 있는 Historian like한 RestAPI를 제공한다.

기본 요청 형식으로 http://ipaddr:port/machiot/ 또는 http://ipaddr:port/machiot-rest-api/ 를 사용한다.

또한 URL에 아래와 같은 parameter를 넘길 수 있다.

Parameter	설명	Sample
f 또는 DateFormat	date format 지정	XXX?f=YYYY/MM/DD XXX?DateFormat=YYYY/MM/DD
s 또는 Scale	Scale 지정	XXX?s=12 XXX?Scale=12
m 또는 FetchMode	fetch 모드 지정	XXX?m=1 XXX?FetchMode=1

Raw 데이터 처리 함수

Raw Value 입력 API

이 API는 주어진 테이블에 데이터를 대량으로 입력하는 함수이다.

URL

http://ipaddr:port/machiot/datapoints/raw/{Table}

http://ipaddr:port/machiot/v1/datapoints/raw/{Table}

- HTTP method : POST
- Table : 입력할 대상 태그 테이블

사용법

```
요청
curl  -X POST -H "Content-Type: application/json" "http://127.0.0.1:${ITF_HTTP_PORT}/machiot-rest-api/datapoints/ra
-d '{"date_format":"YYYY-MM-DD HH24:MI:SS",
     "values":
       [
          ["tag1", "1999-01-01 00:00:00", 0],
          ["tag1", "1999-01-01 00:00:01", 1],
          ["tag1", "1999-01-01 00:00:02", 2]
        ]
     }';
응답
{
   "error_code":0,
   "error_message" :"No Error",
   "timezone":"+0900",
   "data":[],
   "append_success":3,
   "append_failure":0
}
```

Raw Value 추출 API

이 API는 주어진 테이블의 데이터를 얻는 함수이다.

직접 URL을 모두 지정하는 방법을 기본으로 지원하고, GET method의 인자로 넘기는 방법도 지원한다.

아래의 URL에서 각각의 디렉토리명을 인자로 지정할 수도 있다.

URL

http://ipaddr:port/machiot/datapoints/raw/{Table}/{TagNames}/{Start}/{End}/{Direction}/{Count}/{Offset}

http://ipaddr:port/machiot/v1/datapoints/raw/{Table}/{TagNames}/{Start}/{End}/{Direction}/{Count}/{Offset}

- HTTP method : GET
- Table : 데이터를 가져올 대상 테이블명
- TagNames : 데이터를 가져올 대상 태그명
  - 이 태그명은 ,(콤마)로 구분해 복수의 Tag 결과를 하나의 Series로 얻을 수 있음.

- Start : 데이터를 추출할 시작 시간값을 나타냄.
- End : 데이터를 추출할 마지막 시간값을 나타냄
  - 시간 포맷 아래와 같이 스페이스가 없는 형태와 있는 형태 둘다 지원한다. (curl로 테스트할 경우 부가형태를 활용할 수 있다)
    - 기본 형태 (스페이스 지원, 나노초까지 지원)
      - 연-월-일 시:분:초,밀리초
      - 연-월-일 시:분:초 밀리:마이크로:나노
    - 부가 형태 (스페이스 없음, 스페이스 대신 대문자 **T**를 사용하며, 밀리초까지 지원)
      - 연-월-일**T**시:분:초,밀리초
  - 사용예)
    - "2020-12-12"
    - "2020-12-12 03:22:22"
    - "2020-12-12 03:22:22 222:333:444"
    - "2020-12-12**T**03:22:22"
    - "2020-12-12**T**03:22:22**,**234"
- Direction (생략 가능)
  - 0 (디폴트): 디폴트 값으로서 입력된 순서대로 출력
  - 1 : 시간이 감소하는 방향으로 값을 출력
  - 2 : 시간이 증가하는 방향으로 값을 출력
- Count (생략 가능)
  - 0 (디폴트): 전체 데이터를 모두 출력
  - 기타 값 : 주어진 갯수 만큼 레코드를 출력
- Offset (생략 가능)
  - 0 (디폴트) : 건너뛰지 않는다.
  - 기타 값 : 주어진 값 만큼 값을 건너 뜀

사용법

```
요청 (URL을 모두 지정하는 방법)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/v1/datapoints/raw/tag/tag-1/2001-09-09T00:00:00,000/2001-09
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "NAME",
      "type": 5,
      "length": 20
    },
    {
      "name": "TIME",
      "type": 6,
      "length": 31
    },
    {
      "name": "VALUE",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:00:01 000:000:000",
      "VALUE": 8001
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:01:41 000:000:000",
      "VALUE": 8101
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:03:21 000:000:000",
      "VALUE": 8201
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:05:01 000:000:000",
      "VALUE": 8301
    },
```

```
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:06:41 000:000:000",
      "VALUE": 8401
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:08:21 000:000:000",
      "VALUE": 8501
    }
  ]
}
```


요청(인자를 넘기는 방법)


```
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/v1/datapoints/raw?Table=tag&amp;TagNames=tag-1&amp;Start=20
```


응답


```
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "NAME",
      "type": 5,
      "length": 20
    },
    {
      "name": "TIME",
      "type": 6,
      "length": 31
    },
    {
      "name": "VALUE",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:00:01 000:000:000",
      "VALUE": 8001
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:01:41 000:000:000",
      "VALUE": 8101
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:03:21 000:000:000",
      "VALUE": 8201
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:05:01 000:000:000",
      "VALUE": 8301
    },
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:06:41 000:000:000",
      "VALUE": 8401
    }
  ]
}
```


요청 (다수의 태그(tag-1, tag-2, tag-3)를 지정하는 경우)


```
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/v1/datapoints/raw/tag/tag-1,tag-2,tag-3/2001-09-09T00:00:00
```


응답


```
[
```

```
  {
    "NAME": "tag-1",
    "TIME": "2001-09-09 01:00:01 000:000:000",
    "VALUE": 8001
  },
  {
    "NAME": "tag-1",
    "TIME": "2001-09-09 01:01:41 000:000:000",
    "VALUE": 8101
  },
  {
    "NAME": "tag-2",
    "TIME": "2001-09-09 01:00:02 000:000:000",
    "VALUE": 8002
  },
  {
    "NAME": "tag-2",
    "TIME": "2001-09-09 01:01:42 000:000:000",
    "VALUE": 8102
  },
  {
    "NAME": "tag-3",
    "TIME": "2001-09-09 01:00:03 000:000:000",
    "VALUE": 8003
  },
  {
    "NAME": "tag-3",
    "TIME": "2001-09-09 01:01:43 000:000:000",
    "VALUE": 8103
  },
  {
    "NAME": "tag-3",
    "TIME": "2001-09-09 01:03:23 000:000:000",
    "VALUE": 8203
  }
]
```

전체 Tag 기준 Raw Value 삭제 API

이 API는 입력된 모든 태그에 대해 특정 시간 이전의 데이터를 모두 삭제한다.

이 함수는 디스크의 용량이 부족하거나, 백업이 완료된 후 더 이상 필요하지 않은 데이터를 제거하는 데 유용하게 사용할 수 있다.

URL

http://ipaddr:port/machiot/datapoints/raw/{Table}/{BeforeTime}

http://ipaddr:port/machiot/v1/datapoints/raw/{Table}/{BeforeTime}

- HTTP method : **DELETE**
- Table : 삭제할 데이터가 저장된 테이블명
- BeforeTime : 삭제할 이전 시간의 데이터 범위

사용법

```
요청
curl -X DELETE  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/v1/datapoints/raw/tag/2001-09-09T01:20:00,000";
응답
{
  "error_code": 0,
  "error_message": "No Error",
  "timezone": "+0900",
  "effect_rows": "1201",
  "data": []
}
```

특정 Tag 기준 Raw Value 삭제 API

이 API는 특정 태그에 대해 특정 시간 이전의 데이터를 모두 삭제한다.

이 함수는 디스크의 용량이 부족하거나, 백업이 완료된 후 더 이상 필요하지 않은 데이터를 제거하는 데 유용하게 사용할 수 있다.

URL

http://ipaddr:port/machiot/datapoints/raw/{Table}/{TagNames}/{BeforeTime}

http://ipaddr:port/machiot/v1/datapoints/raw/{Table}/{TagNames}/{BeforeTime}

- HTTP method : **DELETE**
- Table : 삭제할 데이터가 저장된 테이블명
- TagNames : 삭제할 대상 태그명들. ,(콤마)로 구분된 다수의 태그를 지정할 수 있음
- BeforeTime : 삭제할 이전 시간의 데이터 범위

사용법

```
요청
curl -X DELETE  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/v1/datapoints/raw/tag/tag-2,tag-3/2001-09-09T01:20:00,0(
응답
{
  "error_code": 0,
  "error_message": "No Error",
  "timezone": "+0900",
  "effect_rows": "32",
  "data": []
}
```

통계 데이터 처리 함수

통계 추출 API

이 API는 저장된 데이터에 대한 통계 결과를 빠르게 얻는 함수이다.

URL

http://ipaddr:port/machiot/datapoints/calculated/{Table}/{TagNames}/{Start}/{End}/{CalculationMode}/{Count}/{IntervalType}/{IntervalValue}

http://ipaddr:port/machiot/v1/datapoints/calculated/{Table}/{TagNames}/{Start}/{End}/{CalculationMode}/{Count}/{IntervalType}/{IntervalValue}

- HTTP method : **GET**
- Table : 데이터를 추출할 대상 테이블명
- TagNames : 대상 태그명
  - 만일 다수의 태그를 지정할 경우에는 그 태그들에 대한 총 연산 결과가 출력됨. (각각의 태그에 대한 통계 결과를 얻고 싶을 경우 반복 호출해야 함)
- Start, End : 데이터를 얻고자 하는 시간 범위 지정 (Raw 데이터 추출 API 참조)
- Count : 데이터의 출력 갯수
- CalculationMode : 얻고자 하는 통계 함수를 지정하며, ,(콤마)를 통해 다수의 통계 함수를 지정할 수 있다. (지정되는 함수명은 아래와 동일해야 한다)
  - min : 최소값
  - max : 최대값
  - sum : 값의 총합
  - count : 값의 갯수
  - avg : 평균 값
- IntervalType : 얻고자 하는 시간 종류 (시, 분, 초)
  - sec : 초 단위
  - min : 분 단위
  - hour : 시간 단위
- IntervalValue : 얻고자 하는 시간 유닛 단위
  - 0보다 큰 값으로서 60의 약수로 지정한다.
  - 주로 5, 10, 15, 30 등이 지정된다.

사용법

```
요청 (단일 통계함수)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/v1/datapoints/calculated/tag/tag-1/2001-09-09T00:00:00,000/
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "time",
      "type": 6,
      "length": 31
    },
    {
      "name": "sum",
      "type": 20,
      "length": 17
```

```
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "time": "2001-09-09 01:00:00 000:000:000",
      "sum": 24303
    },
    {
      "time": "2001-09-09 01:05:00 000:000:000",
      "sum": 25203
    },
    {
      "time": "2001-09-09 01:10:00 000:000:000",
      "sum": 26103
    },
    {
      "time": "2001-09-09 01:15:00 000:000:000",
      "sum": 27003
    },
    {
      "time": "2001-09-09 01:20:00 000:000:000",
      "sum": 9201
    }
  ]
}
요청(다중 통계함수)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/v1/datapoints/calculated/tag/tag-1/2001-09-09T00:00:00,000/
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "time",
      "type": 6,
      "length": 31
    },
    {
      "name": "sum",
      "type": 20,
      "length": 17
    },
    {
      "name": "min",
      "type": 20,
      "length": 17
    },
    {
      "name": "max",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "time": "2001-09-09 01:00:00 000:000:000",
      "sum": 24303,
      "min": 8001,
      "max": 8201
    },
    {
      "time": "2001-09-09 01:05:00 000:000:000",
      "sum": 25203,
      "min": 8301,
      "max": 8501
    },
    {
      "time": "2001-09-09 01:10:00 000:000:000",
      "sum": 26103,
```

```
      "min": 8601,
      "max": 8801
    },
    {
      "time": "2001-09-09 01:15:00 000:000:000",
      "sum": 27003,
      "min": 8901,
      "max": 9101
    },
    {
      "time": "2001-09-09 01:20:00 000:000:000",
      "sum": 9201,
      "min": 9201,
      "max": 9201
    }
  ]
}
```

태그 메타 데이터 처리 함수

이 섹션에서 사용되는 테이블의 구조는 아래와 같이 생성되었다.

```
curl -X GET "http://127.0.0.1:${ITF_HTTP_PORT}/machbase" --data-urlencode 'q=create tagdata table TAG (name_multi v
```

태그 정보 INSERT API

이 API는 사용할 태그를 등록할 때 사용한다. Tag ID와 함께 테이블 생성 시 추가했던 metadata 컬럼의 갯수만큼 데이터를 입력한다.

URL

http://ipaddr:port/machiot/tags/list/{TableName}

http://ipaddr:port/machiot/v1/tags/list/{TableName}

- HTTP method : **POST**
- TableName : 입력할 대상 태그 테이블명을 지정한다.

사용법

```
요청
curl -X POST -H "Content-Type: application/json" "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/list/tag" -d
       '{
        "values":[
            ["tag3", 0, "127.0.0.0"],
            ["tag4", 1, "127.0.0.1"],
            ["tag4", 1, "127.0.0.1"],
            ["tag5", 2, "127.0.0.2"]
         ]
        }';
응답
{
  "error_code": 0,
  "error_message": "No Error",
  "timezone": "+0900",
  "data": [],
  "append_success": 3,
  "append_failure": 1
}
#tag4의 중복 입력으로 에러 1건, 나머지 3건 성공
```

태그 정보 SELECT API

URL

http://ipaddr:port/machiot/tags/list/{Table}/{TagName}

http://ipaddr:port/machiot/v1/tags/list/{Table}/{TagName}

- HTTP method : **GET**
- Table : 추출 대상 태그 테이블
  - 만일 테이블명만 지정될 경우 모든 태그 이름의 리스트를 출력
- TagName : 추출 대상 태그명

- 해당 태그의 상세 정보 출력

사용법

```
요청(전체 태그명 얻기)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/list/tag"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "name_multi",
      "type": 5,
      "length": 20
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "name_multi": "tag0"
    },
    {
      "name_multi": "tag1"
    },
    {
      "name_multi": "tag3"
    },
    {
      "name_multi": "tag4"
    },
    {
      "name_multi": "tag5"
    }
  ]
}
```

태그 정보 UPDATE API

이 API는 부가 태그 정보에 대한 수정을 지원한다.

PUT 혹은 PATCH 모두 지원되며, 입력시 사용되는 JSON 포맷의 값은 해당 테이블의 컬럼명과 동일해야 한다.

또한, 다수의 컬럼명을 지원하기 때문에 한번에 두개 이상의 컬럼의 값을 변경할 수 있다.

URL

http://ipaddr:port/machiot/tags/list/{Table}/{TagName}

http://ipaddr:port/machiot/v1/tags/list/{Table}/{TagName}

- HTTP method : **PUT / PATCH**
- Table : 수정 대상 태그 테이블명
- TagName : 수정 대상 태그명

사용법

```
요청(PUT 사용시)
curl -X PUT -H "Content-Type: application/json" "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/list/tag/tag0" -d
응답
{
   "error_code":0,
   "error_message" :"No Error",
   "timezone":"+0900",
   "effect_rows":"1",
   "data":[]
}

요청 (PATCH 사용시)
curl -X PATCH -H "Content-Type: application/json" "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/list/tag/tag3" -d
응답
{
   "error_code":0,
   "error_message" :"No Error",
```

```
   "timezone":"+0900",
   "effect_rows":"1",
   "data":[]
}
```

태그 정보 삭제 API

이 API는 지정된 태그를 삭제한다. 하지만 해당 태그에 raw 데이터가 존재할 경우 삭제에 실패한다.

데이터가 존재하는 태그를 삭제하기 위해서는 해당 태그의 raw 데이터에 대한 삭제를 먼저 수행하고 난 뒤에 이 함수를 호출해야 한다.

URL

http://ipaddr:port/machiot/tags/list/{Table}/{TagNames}

http://ipaddr:port/machiot/v1/tags/list/{Table}/{TagNames}

- HTTP method : **DELETE**
- Table: 삭제 대상 태그 테이블명
- TagName: 삭제할 대상 태그명

사용법

```
요청 (데이터가 있는 태그의 경우)
curl -X DELETE -H "Content-Type: application/json" "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/list/tag/tag0";
응답 (에러)
{
   "error_code":2324,
   "error_message" :"Cannot delete tagmeta. there exist data with deleted_tag key.",
   "timezone":"+0900",
   "data":[]
}


요청 (데이터가 모두 삭제된 태그의 경우)
curl -X DELETE -H "Content-Type: application/json" "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/list/tag/tag4";
응답 (성공)
{
   "error_code":0,
   "error_message" :"No Error",
   "timezone":"+0900",
   "effect_rows":"1",
   "data":[]
}
```

기타 함수

시간 범위 얻기 API

이 API는 지정된 테이블 및 태그의 데이터에 대한 전체의 시간 범위(최소, 최대)를 얻는다.

URL

http://ipaddr:port/machiot/tags/range/{Table}/{TagName}

http://ipaddr:port/machiot/v1/tags/range/{Table}/{TagName}

- HTTP method : **GET**
- Table : 추출한 대상 테이블명
- TagName : 추출할 태그명
    - 지정하지 않았을 경우 전체 시간 범위 반환(이때 태그명은 ALL로 되돌아온다)

사용법

```
요청 (전체 테이블 범위)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/range/tag"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "name",
      "type": 5,
```

```
      "length": 3
    },
    {
      "name": "min",
      "type": 6,
      "length": 31
    },
    {
      "name": "max",
      "type": 6,
      "length": 31
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "name": "ALL",
      "min": "2001-09-09 01:00:00 000:000:000",
      "max": "2032-09-09 10:46:49 000:000:000"
    }
  ]
}

요청(특정 태그 tag-1, tag-2에 대한 시간 범위)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/range/tag/tag-1,tag-2"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "name",
      "type": 5,
      "length": 20
    },
    {
      "name": "min",
      "type": 6,
      "length": 31
    },
    {
      "name": "max",
      "type": 6,
      "length": 31
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "name": "tag-1",
      "min": "2001-09-09 01:00:01 000:000:000",
      "max": "2001-09-21 12:31:41 000:000:000"
    },
    {
      "name": "tag-2",
      "min": "2001-09-09 01:00:02 000:000:000",
      "max": "2001-09-21 12:31:42 000:000:000"
    }
  ]
}
```


최소 value 얻기 API

이 API는 지정된 테이블 혹은 태그에 존재하는 최소 Value를 얻는다.

URL

http://ipaddr:port/machiot/tags/min/{Table}/{TagName}

http://ipaddr:port/machiot/v1/tags/min/{Table}/{TagName}

- HTTP method : **GET**

- Table : 추출한 대상 테이블명
- TagName : 추출할 태그명
  - 지정하지 않았을 경우 해당 테이블의 최소 value만을 출력

사용법

```
요청 (전체 테이블)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/min/tag"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "min",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "min": 0.0
    }
  ]
}


요청 (특정 태그 tag-1, tag-2)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/min/tag/tag-1,tag-2";
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "name",
      "type": 5,
      "length": 100
    },
    {
      "name": "time",
      "type": 6,
      "length": 31
    },
    {
      "name": "min",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "name": "tag-1",
      "time": "2001-09-09 10:46:42 000:000:000",
      "min": 10001.0
    },
    {
      "name": "tag-2",
      "time": "2001-09-09 10:46:43 000:000:000",
      "min": 10002.0
    }
  ]
}
```

최대 value 얻기 API

이 API는 지정된 테이블 혹은 태그에 존재하는 최대 Value를 얻는다.

URL

http://ipaddr:port/machiot/tags/max/{Table}/{TagName}

http://ipaddr:port/machiot/v1/tags/max/{Table}/{TagName}

- HTTP method : **GET**
- Table : 추출한 대상 테이블명
- TagName : 추출할 태그명
  - 지정하지 않았을 경우 해당 테이블의 최대 value만을 출력

사용법

```
요청 (전체 테이블)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/max/tag"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "max",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "max": 100000000000.0
    }
  ]
}


요청 (특정 태그 tag-1, tag-2)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/max/tag/tag-1,tag-2";
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "name",
      "type": 5,
      "length": 100
    },
    {
      "name": "time",
      "type": 6,
      "length": 31
    },
    {
      "name": "max",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "name": "tag-1",
      "time": "2001-09-09 13:12:12 000:000:000",
      "max": 9999999991.0
    },
    {
      "name": "tag-2",
      "time": "2001-09-09 13:12:13 000:000:000",
      "max": 9999999992.0
    }
  ]
}
```

최초 row 얻기 API

이 API는 지정된 테이블 혹은 태그에 존재하는 가장 작은 time값의 row를 얻는다.

URL

http://ipaddr:port/machiot/tags/first/{Table}/{TagName}

http://ipaddr:port/machiot/v1/tags/first/{Table}/{TagName}

- HTTP method : **GET**
- Table : 추출한 대상 테이블명
- TagName : 추출할 태그명
    - 지정하지 않았을 경우 해당 테이블의 최초 row를 출력

사용법

```
요청 (전체 테이블)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/first/tag"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "NAME",
      "type": 5,
      "length": 20
    },
    {
      "name": "TIME",
      "type": 6,
      "length": 31
    },
    {
      "name": "VALUE",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "NAME": "tag-0",
      "TIME": "2001-09-09 01:00:00 000:000:000",
      "VALUE": 8000.0
    }
  ]
}


요청 (특정 태그 tag-1, tag-2)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/first/tag/tag-1,tag-2";
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "NAME",
      "type": 5,
      "length": 20
    },
    {
      "name": "TIME",
      "type": 6,
      "length": 31
    },
    {
      "name": "VALUE",
      "type": 20,
      "length": 17
    }
  ],
```

```
  "timezone": "+0900",
  "data": [
    {
      "NAME": "tag-1",
      "TIME": "2001-09-09 01:00:01 000:000:000",
      "VALUE": 8001.0
    },
    {
      "NAME": "tag-2",
      "TIME": "2001-09-09 01:00:02 000:000:000",
      "VALUE": 8002.0
    }
  ]
}
```

최후 row 얻기 API

이 API는 지정된 테이블 혹은 태그에 존재하는 가장 큰 time값의 row를 얻는다.

URL

http://ipaddr:port/machiot/tags/last/{Table}/{TagName}

http://ipaddr:port/machiot/v1/tags/last/{Table}/{TagName}

- HTTP method : **GET**
- Table : 추출한 대상 테이블명
- TagName : 추출할 태그명
    - 지정하지 않았을 경우 해당 테이블의 최후 row를 출력

사용법

```
요청 (전체 테이블)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/last/tag"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "NAME",
      "type": 5,
      "length": 20
    },
    {
      "name": "TIME",
      "type": 6,
      "length": 31
    },
    {
      "name": "VALUE",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "NAME": "dummy",
      "TIME": "2032-09-09 10:46:49 000:000:000",
      "VALUE": 1000000009.0
    }
  ]
}


요청 (특정 태그 tag-1, tag-2)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/last/tag/tag-1,tag-2";
{
  "error_code": 0,
  "error_message": "",
  "columns": [
```

```
    {
      "name": "NAME",
      "type": 5,
      "length": 20
    },
    {
      "name": "TIME",
      "type": 6,
      "length": 31
    },
    {
      "name": "VALUE",
      "type": 20,
      "length": 17
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "NAME": "tag-1",
      "TIME": "2001-09-21 12:31:41 000:000:000",
      "VALUE": 999901.0
    },
    {
      "NAME": "tag-2",
      "TIME": "2001-09-21 12:31:42 000:000:000",
      "VALUE": 999902.0
    }
  ]
}
```

레코드 갯수 얻기 API

이 API는 지정된 테이블 혹은 태그에 존재하는 레코드의 갯수를 얻는다.

URL

http://ipaddr:port/machiot/tags/count/{Table}/{TagNames}

혹은

http://ipaddr:port/machiot/tags/cnt/{Table}/{TagNames}

http://ipaddr:port/machiot/v1/tags/count/{Table}/{TagNames}

혹은

http://ipaddr:port/machiot/v1/tags/cnt/{Table}/{TagNames}

- HTTP method : **GET**
- Table : 추출한 대상 테이블명
- TagName : 추출할 태그명
    - 지정하지 않았을 경우 해당 테이블의 전체 레코드 갯수 출력

사용법

```
요청 (전체 테이블)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/count/tag"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "count",
      "type": 12,
      "length": 20
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "count": 1000001
    }
  ]
```

```
}


요청 (특정 태그 tag-1, tag-2)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/count/tag/tag-1,tag-2";
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "name",
      "type": 5,
      "length": 100
    },
    {
      "name": "count",
      "type": 12,
      "length": 20
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "name": "tag-1",
      "count": 10000
    },
    {
      "name": "tag-2",
      "count": 10000
    }
  ]
}
```

디스크 사용량 얻기 API

이 API는 지정된 테이블 혹은 태그가 사용중인 디스크 사용량의 근사치를 얻는다.

URL

http://ipaddr:port/machiot/tags/disksize/{Table}/{TagNames}

http://ipaddr:port/machiot/v1/tags/disksize/{Table}/{TagNames}

- HTTP method : **GET**
- Table : 추출한 대상 테이블명
- TagName : 추출할 태그명
    - 지정하지 않았을 경우 해당 테이블의 전체 디스크 사용량 출력

사용법

```
요청 (전체 테이블)
curl -X GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/disksize/tag/"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "disksize",
      "type": 12,
      "length": 20
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "disksize": 276904448
    }
  ]
}

요청 (특정 태그 tag-1, tag-2)
```

```
curl -X  GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/disksize/tag/tag-1,tag-2"
응답
{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "name",
      "type": 5,
      "length": 100
    },
    {
      "name": "disksize",
      "type": 12,
      "length": 20
    }
  ],
  "timezone": "+0900",
  "data": [
    {
      "name": "tag-1",
      "disksize": 240000
    },
    {
      "name": "tag-2",
      "disksize": 240000
    }
  ]
}
```

롤업 요청 API

이 API는 특정 롤업 테이블에 대한 강제적인 수행을 요청한다. 이를 통해 아직 계산되지 않은 통계 값을 계산하도록 강제한다.

이 API를 호출하면 상황에 따라 몇초에서 몇분까지 대기할 수 있으므로 신중하게 사용해야 한다.

URL

http://ipaddr:port/machiot/rollup/{Table}

http://ipaddr:port/machiot/v1/rollup/{Table}

- HTTP method : **GET**
- Table : 강제로 롤업을 수행할 태그 테이블명

사용법

```
요청
curl -X HTTP GET  "http://127.0.0.1:${ITF_HTTP_PORT}/machiot/tags/rollup/tag"
응답
{
  "error_code": 0,
  "error_message": "No Error",
  "timezone": "+0900",
  "data": []
}
```

.NET Connector

ADO.NET 드라이버 일부 기능을 지원하는 .NET (C#) Connector 라이브러리를 제공하고 있다.

라이브러리 위치는 `$MACHBASE_HOME/lib/` 에서 DLL 형태로 제공하고 있으며, .NET 버전에 따라 다른 DLL 을 제공한다.

- > .NET Framework 4.0 : `machNetConnector.dll`
- > .NET Core 2.0 : `machNetConnectorCore.dll`

목차

클래스

> 아래 소개된 기능 외의 것은 아직 구현되어 있지 않거나, 올바르게 작동되지 않을 수 있다.
> 미구현으로 명시된 메서드나 필드를 부르는 경우, NotImplementedException 또는 NotSupportedException 을 발생시킨다.

마크베이스와의 연결을 담당하는 클래스이다. DbConnection 과 같이 IDisposable 을 상속받기 때문에, Dispose() 를 통한 객체 해제나 using() 문을 이용한 객체의 자동 Dispose를 지원한다.MachConnection : DbConnection

생성자	설명
MachConnection(string aConnectionString)	Connection String 을 입력으로, MachConnection 을 생성한다.

메서드	설명
Open()	입력받은 Connection String 으로 실제 연결을 시도한다.
Close()	연결 중이라면, 해당 연결을 종료한다.
BeginDbTransaction(IsolationLevel isolationLevel)	(미구현) MACHBASE 는 특별한 Transaction 이 없으므로 해당 객체 역시 지원하지 않는다.
CreateDbCommand()	(미구현) 아직은, 명시적으로 MachCommand 를 만들도록 유도한다.
ChangeDatabase(string databaseName)	(미구현) MACHBASE 는 DATABASE 구분이 없다.

필드	설명
State	System.Data.ConnectionState 값을 나타낸다.
StatusString	연결된 MachCommand 로 수행하는 상태를 나타낸다. Error Message 를 꾸미는 용도로 내부에서 사용되며, 작업이 시작된 상태를 나타내기 때문에 이 값으로 쿼리 상태를 체크하는 것은 적절하지 않다.
Database	(미구현)
DataSource	(미구현)
ServerVersion	(미구현)

> **Connection String**
>
> 각 항목은 semicolon (;) 으로 구분되며, 다음을 사용할 수 있다.
> 동일 항목에 있는 여러 Keyword 는, 모두 같은 의미이다.

Keyword	설명	예제	기본값
SERVER	Hostname	SERVER=192.168.0.1	
PORT PORT_NO	Port No.	PORT=5656	5656
USERID USERNAME USER UID	사용자 ID	USER=SYS	SYS

Keyword	설명	예제	기본값
PASSWORD PWD	사용자 패스워드	PWD=manager	
CONNECT_TIMEOUT ConnectionTimeout connectTimeout	연결 최대 시간	CONNECT_TIMEOUT	60초
COMMAND_TIMEOUT commandTimeout	각 명령 수행 최대 시간	COMMAND_TIMEOUT	60초

예제로, 아래와 같은 문자열을 준비해 둘 수 있다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;CONNECT_TIMEOUT=10000;COMMAND_TIM
```

MachCommand : DbCommand

MachConnection 을 이용해 **SQL 명령 또는 APPEND 를** 수행하는 클래스이다.
DbCommand 와 같이 IDisposable 을 상속받기 때문에, Dispose() 를 통한 객체 해제나 using() 문을 이용한 객체의 자동 Dispose를 지원한다.

생성자	설명
MachCommand(string aQueryString, MachConnection)	연결할 MachConnection 객체와 함께, 수행할 쿼리를 입력해서 생성한다.
MachCommand(MachConnection)	연결할 MachConnection 객체를 입력해서 생성한다. 수행할 쿼리가 없는 경우 (e.g. APPEND) 에만 사용한다.

메서드	설명
void CreateParameter() / void CreateDbParameter()	새로운 MachParameter 를 생성한다.
void Cancel()	(미구현)
void Prepare()	(미구현)

MachAppendWriter
AppendOpen(aTableName, aErrorCheckCount = 0,
MachAppendOption = None)
	APPEND 를 시작한다. MachAppendWriter 객체를 반환한다.
	• aTableName : 대상 테이블 이름
	• aErrorCheckCount : APPEND-DATA 로 입력한 레코드 누적 개수가 일치할 때 마다, 서버로 보내 실패 여부를 확인한다.
	말하자면, 자동 APPEND-FLUSH 지점을 정하는 셈이다.
	• MachAppendOption : 현재는 하나의 옵션만 제공하고 있다.
	  • MachAppendOption.None : 아무런 옵션도 붙지 않는다.
	  • MachAppendOption.MicroSecTruncated : DateTime 객체의 값 입력 시, microsecond 까지만 표현된 값을 입력한다.
	(DateTime 객체의 Ticks 값은 100 nanosecond 까지 표현된다.)

void AppendData(MachAppendWriter aWriter, List<object> aDataList)	MachAppendWriter 객체를 통해, 데이터가 들어있는 리스트를 받아 데이터베이스에 입력한다. • List 에 들어간 데이터 순서대로, 각각의 자료형은 테이블에 표현된 컬럼의 자료형과 일치해야 한다. • List 에 들어있는 데이터가 모자라거나 넘치면, 에러를 발생시킨다. ⓘ ulong 객체로 시간 값을 표현할 때, 단순히 DateTime 객체의 Tick 값을 입력하면 안 된다. 그 값에서, 1970-01-01 을 나타내는 DateTime Tick 값을 제외한 값을 입력해야 한다.
void AppendDataWithTime(MachAppendWriter aWriter, List<object> aDataList, DateTime aArrivalTime)	AppendData() 에서, _arrival_time 값을 DateTime 객체로 명시적으로 넣을 수 있는 메서드이다.
void AppendDataWithTime(MachAppendWriter aWriter, List<object> aDataList, ulong aArrivalTimeLong)	AppendData() 에서, _arrival_time 값을 ulong 객체로 명시적으로 넣을 수 있는 메서드이다. ulong 값을 _arrival_time 값으로 입력할 때 발생할 수 있는 문제는 위의 AppendData() 를 참고한다.
void AppendFlush(MachAppendWriter aWriter)	AppendData() 로 입력한 데이터들을 즉시 서버로 보내, 데이터 입력을 강제한다. 호출 빈도가 잦을 수록, 성능은 떨어지지만 시스템 오류로 인한 데이터 유실율을 낮출 수 있고 에러 검사를 빠르게 할 수 있다.

메서드	설명
	호출 빈도가 뜸할 수록, 데이터 유실이 발생할 가능성이 크고 에러 검사가 지연되지만 성능은 크게 올라간다.
void AppendClose(MachAppendWriter aWriter)	APPEND 를 마친다. 내부적으로 AppendFlush() 를 호출한 뒤에 실제 프로토콜을 마친다.
int ExecuteNonQuery()	입력받았던 쿼리를 수행한다. 쿼리가 영향을 미친 레코드 개수를 반환한다. 보통 SELECT 를 제외한 쿼리를 수행할 때 사용한다.
object ExecuteScalar()	입력받았던 쿼리를 수행한다. 쿼리 Targetlist 의 첫 번째 값을 객체로 반환한다. 보통 SELECT 쿼리, 그 중에서도 결과가 1건만 나오는 SELECT (Scalar Query) 를 수행해 DbDataReader 없이 결과를 받고자 하는 경우 사용한다.
DbDataReader ExecuteDbDataReader(CommandBehavior aBehavior)	입력받았던 쿼리를 수행해, 해당 쿼리의 결과를 읽어 올 수 있는 DbDataReader 를 생성해 반환한다.

필드	설명
Connection / DbConnection	연결된 MachConnection.
ParameterCollection / DbParameterCollection	Binding 목적으로 사용할 MachParameterCollection.
CommandText	쿼리 문자열.
CommandTimeout	특정 작업 수행 중, 서버로부터 응답을 기다리기까지의 시간. MachConnection 에 설정된 값을 따르며, 여기서는 값 참조만 할 수 있다.
FetchSize	한번에 서버로부터 Fetch 할 레코드 개수. 기본값은 3000 이다.
IsAppendOpened	APPEND 작업 중인 경우, Append 가 이미 열려있는지 아닌지를 판단한다.
CommandType	(미구현)
DesignTimeVisible	(미구현)
UpdatedRowSource	(미구현)

MachDataReader : DbDataReader

Fetch 한 결과를 읽어들이는 클래스이다. 명시적으로 생성이 불가능하고 MachCommand.ExecuteDbDataReader() 로 생성된 객체만 사용이 가능하다.

메서드	설명
string GetName(int ordinal)	ordinal 번째 컬럼 이름을 반환한다.
string GetDataTypeName(int ordinal)	ordinal 번째 컬럼의 자료형 이름을 반환한다.
Type GetFieldType(int ordinal)	ordinal 번째 컬럼의 자료형을 반환한다.
int GetOrdinal(string name)	컬럼 이름이 위치한 인덱스를 반환한다.
object GetValue(int ordinal)	현재 위치한 레코드의 ordinal 번째 값을 반환한다.
bool IsDBNull(int ordinal)	현재 위치한 레코드의 ordinal 번째 값이 NULL 인지 여부를 반환한다.
int GetValues(object[] values)	현재 위치한 레코드의 모든 값들을 전부 설정하고, 그 개수를 반환한다.
xxxx GetXXXX(int ordinal)	ordinal 번째 컬럼 값을, 자료형 (XXXX) 에 맞춰 반환한다. • Boolean • Byte • Char • Int16/32/64 • DateTime • String • Decimal • Double • Float

메서드	설명
bool Read()	다음 레코드를 읽는다. 결과가 존재하지 않으면 False 를 반환한다.
DataTable GetSchemaTable()	(미지원)
bool NextResult()	(미지원)

필드	설명
FetchSize	한번에 서버로부터 Fetch 할 레코드 개수. 기본값은 3000 이며 여기서는 수정할 수 없다.
FieldCount	결과 컬럼 개수.
this[int ordinal]	object GetValue(int ordinal) 와 동일하다.
this[string name]	object GetValue(GetOrdinal(name)) 와 동일하다.
HasRows	결과가 존재하는지 여부를 나타낸다.
RecordsAffected	MachCommand 의 것과 달리, 여기서는 Fetch Count 를 나타낸다.

MachParameterCollection : DbParameterCollection

MachCommand 에 필요한 파라메터를 바인딩하는 클래스이다.

바인딩한 이후에 수행하게 되면, 해당 값이 같이 수행된다.

> Prepared Statement 개념이 구현되어 있지 않아, Binding 이후 Execute 를 해도 수행 성능은 최초 수행한 것과 같다.

메서드	설명
MachParameter Add(string parameterName, DbType dbType)	파라메터 이름과 타입을 지정해, MachParameter 를 추가한다. 추가된 MachParameter 객체를 반환한다.
int Add(object value)	값을 추가한다. 추가된 인덱스를 반환한다.
void AddRange(Array values)	단순 값의 배열을 모두 추가한다.
MachParameter AddWithValue(string parameterName, object value)	파라메터 이름과 그 값을 추가한다. 추가된 MachParameter 객체를 반환한다.
bool Contains(object value)	해당 값이 추가되었는지 여부를 판단한다.
bool Contains(string value)	해당 파라메터 이름이 추가되었는지 여부를 판단한다.
void Clear()	파라메터들을 모두 삭제한다.
int IndexOf(object value)	해당 값의 인덱스를 반환한다.
int IndexOf(string parameterName)	해당 파라메터 이름의 인덱스를 반환한다.
void Insert(int index, object value)	특정 인덱스에, 해당 값을 추가한다.
void Remove(object value)	해당 값을 포함한 파라메터를 삭제한다.
void RemoveAt(int index)	인덱스에 위치한 파라메터를 삭제한다.
void RemoveAt(string parameterName)	해당 이름을 가진 파라메터를 삭제한다.

필드	설명
Count	파라메터 개수

필드	설명
this[int index]	index 번째의 MachParameter 를 나타낸다.
this[string name]	파라메터 이름과 일치하는 순서의 MachParameter 를 나타낸다.

MachParameter : DbParameter

MachCommand 에 필요한 파라메터를 각각 바인딩한 정보를 담는 클래스이다.

특별히 메서드는 지원하지 않는다.

필드	설명
ParameterName	파라메터 이름
Value	값
Size	값의 크기
Direction	ParameterDirection (Input / Output / InputOutput / ReturnValue) 기본값은 Input 이다.
DbType	DB Type
MachDbType	MACHBASE DB Type DB Type 과 다를 수 있다.
IsNullable	NULL 가능 여부
HasSetDbType	DB Type 이 지정되었는지 여부

MachException : DbException

마크베이스에서 나타나는 에러를 표시하는 클래스이다.

에러 메시지가 설정되어 있는데, 모든 에러 메시지는 *MachErrorMsg* 에서 확인할 수 있다.

필드	설명
int MachErrorCode	MACHBASE 에서 제공하는 에러 코드

MachAppendWriter

MachCommand 를 사용하는 별도의 클래스로 APPEND 를 지원한다.
ADO.NET 표준이 아닌, MACHBASE 의 Append Protocol 을 지원하기 위한 클래스이다.

별도의 생성자 없이 MachCommand 의 AppendOpen() 으로 생성된다.

메서드	설명
void SetErrorDelegator(ErrorDelegateFuncType aFunc)	에러가 발생했을 때 호출할 ErrorDelegateFunc 을 지정한다.

필드	설명
SuccessCount	입력 성공한 레코드 개수. AppendClose() 이후 설정된다.
FailureCount	입력 실패한 레코드 개수. AppendClose() 이후 설정된다.
Option	AppendOpen() 때 입력받은 MachAppendOption

ErrorDelegateFuncType

```
public delegate void ErrorDelegateFuncType(MachAppendException e);
```

MachAppendWriter 에서, APPEND 도중 MACHBASE 서버 측에서 발생하는 Error 를 감지하기 위한 함수를 지정할 수 있다.

.NET 에서는 이 함수형을 Delegator Function 으로 지정한다.

MachAppendException : MachException

MachException 과 동일하지만, 다음 점이 다르다.

- 에러 메시지가 서버 측으로부터 수신된다.
- 에러가 발생한 데이터 버퍼를 획득할 수 있다. (comma-separated) 이 데이터를 가공해 다시 APPEND 하거나 기록하는 용도로 사용할 수 있다.

해당 예외는 ErrorDelegateFunc 내부에서만 획득이 가능하다.

메서드	설명
GetRowBuffer()	에러가 발생한 데이터 버퍼를 획득할 수 있다.

MachTransaction

지원하지 않는다.

샘플 코드

연결

MachConnection 을 만들어 Open() - Close() 하면 된다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;", SERVER_HOST, SERVER_PORT);
MachConnection sConn = new MachConnection(sConnString);
sConn.Open();
//... do something
sConn.Close();
```

using 구문을 사용하면, Connection 종료 작업인 Close() 를 호출하지 않아도 된다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;", SERVER_HOST, SERVER_PORT);
using (MachConnection sConn = new MachConnection(sConnString))
{
    sConn.Open();
    //... do something
} // you don't need to call sConn.Close();
```

쿼리 수행

MachCommand 를 만들어 쿼리를 수행하면 된다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;", SERVER_HOST, SERVER_PORT);
using (MachConnection sConn = new MachConnection(sConnString))
{
    String sQueryString = "CREATE TABLE tab1 ( col1 INTEGER, col2 VARCHAR(20) )";
    MachCommand sCommand = new MachCommand(sQueryString , sConn)
    try
    {
        sCommand.ExecuteNonQuery();
    }
    catch (MachException me)
    {
        throw me;
    }
}
```

이 역시 using 구문을 사용하면, MachCommand 해제 작업을 곧바로 진행할 수 있다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;", SERVER_HOST, SERVER_PORT);
using (MachConnection sConn = new MachConnection(sConnString))
{
    String sQueryString = "CREATE TABLE tab1 ( col1 INTEGER, col2 VARCHAR(20) )";
    using(MachCommand sCommand = new MachCommand(sQueryString , sConn))
    {
        try
        {
            sCommand.ExecuteNonQuery();
        }
        catch (MachException me)
        {
            throw me;
        }
    }
}
```

SELECT 수행

SELECT 쿼리를 가진 MachCommand 를 실행해 MachDataReader 를 얻을 수 있다.
MachDataReader 를 통해 레코드를 하나씩 Fetch 할 수 있다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;", SERVER_HOST, SERVER_PORT);
using (MachConnection sConn = new MachConnection(sConnString))
{
    String sQueryString = "SELECT * FROM tab1;";
    using(MachCommand sCommand = new MachCommand(sQueryString , sConn))
    {
        try
        {
            MachDataReader sDataReader = sCommand.ExecuteReader();
            while (sDataReader.Read())
            {
                for (int i = 0; i < sDataReader.FieldCount; i++)
                {
                    Console.WriteLine(String.Format("{0} : {1}",
                                                    sDataReader.GetName(i),
                                                    sDataReader.GetValue(i)));
                }
            }
        }
        catch (MachException me)
        {
            throw me;
        }
    }
}
```

파라메터 바인딩

MachParameterCollection 을 생성한 다음, MachCommand 에 연결해서 수행할 수 있다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;", SERVER_HOST, SERVER_PORT);
using (MachConnection sConn = new MachConnection(sConnString))
{
    string sSelectQuery = @"SELECT *
        FROM tab2
        WHERE CreatedDateTime < @CurrentTime
        AND CreatedDateTime >= @PastTime";

    using (MachCommand sCommand = new MachCommand(sSelectQuery, sConn))
    {
        DateTime sCurrtime = DateTime.Now;
        DateTime sPastTime = sCurrtime.AddMinutes(-1);
```

```
        try
        {
            sCommand.ParameterCollection.Add(new MachParameter { ParameterName = "@CurrentTime", Value = sCurrtime
            sCommand.ParameterCollection.Add(new MachParameter { ParameterName = "@PastTime", Value = sPastTime });

            MachDataReader sDataReader = sCommand.ExecuteReader();

            while (sDataReader.Read())
            {
                for (int i = 0; i < sDataReader.FieldCount; i++)
                {
                    Console.WriteLine(String.Format("{0} : {1}",
                                                    sDataReader.GetName(i),
                                                    sDataReader.GetValue(i)));
                }
            }
        }
        catch (MachException me)
        {
            throw me;
        }
    }
}
```

APPEND

MachCommand 에서 AppendOpen() 을 수행하면, MachAppendWriter 객체를 얻을 수 있다.

이 객체와 MachCommand 를 이용해, 입력 레코드 1건을 리스트로 준비해 AppendData() 를 수행하면 입력이 이뤄진다.

AppendFlush() 를 하면 모든 레코드의 입력이 반영되며, AppendClose() 를 통해 Append 전체 과정을 종료할 수 있다.

```
String sConnString = String.Format("SERVER={0};PORT_NO={1};UID=;PWD=MANAGER;", SERVER_HOST, SERVER_PORT);
using (MachConnection sConn = new MachConnection(sConnString))
{
    using (MachCommand sAppendCommand = new MachCommand(sConn))
    {
        MachAppendWriter sWriter = sAppendCommand.AppendOpen("tab2");
        sWriter.SetErrorDelegator(AppendErrorDelegator);

        var sList = new List<object>();
        for (int i = 1; i <= 100000; i++)
        {
            sList.Add(i);
            sList.Add(String.Format("NAME_{0}", i % 100));

            sAppendCommand.AppendData(sWriter, sList);

            sList.Clear();

            if (i % 1000 == 0)
            {
                sAppendCommand.AppendFlush();
            }
        }

        sAppendCommand.AppendClose(sWriter);
        Console.WriteLine(String.Format("Success Count : {0}", sWriter.SuccessCount));
        Console.WriteLine(String.Format("Failure Count : {0}", sWriter.FailureCount));
    }
}
```

```
private static void AppendErrorDelegator(MachAppendException e)
{
    Console.WriteLine("{0}", e.Message);
    Console.WriteLine("{0}", e.GetRowBuffer());
}
```

Timezone

마크베이스의 Timezone

마크베이스는 각 클라이언트의 타임존을 세션 단위에서 유효한 것으로 가정한다.

일반적으로는 특정한 시간을 나타내는 스트링으로 타임존을 지정하는 방식을 사용한다.

```
"YYYY-MM-DD HH24:MI:SS ZZZ(타임존스트링)"

예)
"12:06:56.568+01:00"
"2006.07.10 at 15:08:56 -05:00"
"09  AM, GMT+09:00"
```

그러나 위의 방식은 특정 시간을 매번 타임존을 기준으로 지정해야 하는 불편함이 있을 뿐만 아니라, 대량의 데이터에 대해서 모두 타임존의 값이 포함된 경우 데이터 전송량이 선형적으로 늘어나는 문제가 있다.

따라서 마크베이스에서는 클라이언트와 서버가 연결된 세션에 대해 타임존 속성을 지정하는 방식을 지원한다.

다음은, 마크베이스에서 제공하는 타임존이 동작에 대한 단계적 설명이다.

- 서버는 그 서버가 설치된 운영체제에서 제공하는 기본 타임존을 기준으로 동작한다.
  즉, 아무런 설정을 하지 않았을 경우 마크베이스는 해당 운영체제가 동작하는 타임존을 읽어서 활용한다.
- 클라이언트 프로그램에서 타임존 설정없이 서버로 접속하면 해당 클라이언트의 타임존의 서버의 타임존으로 설정된다.
  즉, 서버에서 설정된 TIMEZONE이 KST이라면, 클라이언트 역시 KST로 동작한다는 뜻이다.
- 클라이언트 프로그램에서 타임존을 명시적으로 설정한 경우에는 해당 서버의 해당 세션은 클라이언트에서 지정한 타임존으로 동작한다.
  즉, 서버에서 설정된 TIMEZONE이 KST이라 하더라도, 만일 클라이언트가 접속시 EDT로 타임존을 설정한 경우에는 해당 세션은 EDT로 동작한다.

마크베이스의 Timezone 지원 형식

마크베이스는 사용상의 편의성 증진과 복잡성을 제거하기 위해 5자리의 문자로 구성된 단 한가지 포맷만을 제공한다.

즉, 첫번째 문자는 + 혹은 -의 기호로 시간의 부호를 나타내고, 이어지는 두개의 문자는 00에서 23 사이의 값을 가진다. 그리고, 마지막 두개의 문자는 00에서 59까지의 시간을 가지는 것으로 한다.

아래는 마크베이스에서 지원하는 TIMEZONE의 형식을 나타낸 것이다.

```
ex)
TIMEZONE=+0900
TIMEZONE=-0900
```

machsql

machsql은 구동시 아래와 같은 옵션을 통해 동작할 타임존을 설정할 수 있다.

```
-z, --timezone=+-HHMM
```

SHOW TIMEZONE 명령을 통해 현재 자신이 설정된 타임존을 확인할 수 있다.

```
SHOW TIMEZONE;

```

```
Mach> show timezone;
Timezone : +0900
```

machloader

machloader는 구동시 아래와 같은 옵션을 통해 동작할 타임존을 설정할 수 있다.

```
-z, --timezone=+-HHMM
```

지정된 타임존으로 접속하고, 시간 연산도 해당 타임존을 기준으로 동작한다.

SDK

연결 스트링에 TIMEZONE이 추가되었으며, 해당 세션에 대한 타임존을 지정할 수 있다.

만일 연결 스트링에 TIMEZONE을 지정하지 않을 경우에는 서버의 타임존을 기준으로 동작한다.

이는 CLI, ODBC, JDBC, DOTNET 모두 동일한다.

연결 string 예제

```
SERVER=127.0.0.1;UID=SYS;PWD=MANAGER;CONNTYPE=1;NLS_USE=UTF8;PORT_NO=5656;TIMEZONE=+0300
```

Rest API

Rest API는 동작 요청시 HTTP 프로토콜의 HEADER에서 지정된 타임존을 기준으로 동작한다.

그 헤더의 이름은 **The-Timezone-Machbase으로 명명되었으며, 사용법은 아래와 같다.**

```
Authorization: Basic XXXXXXXXXXXXXXXXXX
...................
The-Timezone-Machbase: +0900
...............
```

앞에서 기술한 바와 같이 원하는 타임존 스트링을 지정하면 된다.

타임존을 지정하지 않았을 경우에는 서버의 타임존으로 동작한다.

요청 예제 : UTC로 설정

```
curl -H "The-Timezone-Machbase: +0000" -G "http://127.0.0.1:${ITF_HTTP_PORT}/machbase" --data-urlencode 'q=selec



{
  "error_code": 0,
  "error_message": "",
  "columns": [
    {
      "name": "C1",
      "type": 4,
      "length": 6
    },
    {
      "name": "C2",
      "type": 8,
      "length": 11
    },
    {
```

```
      "name": "C3",
      "type": 5,
      "length": 20
    },
    {
      "name": "C4",
      "type": 6,
      "length": 31
    },
    {
      "name": "C5",
      "type": 32,
      "length": 15
    }
  ],
  "timezone": "+0000",
  "data": [
    {
      "C1": 1,
      "C2": 2,
      "C3": "test1",
      "C4": "1999-09-09 00:09:09 000:000:000",
      "C5": "127.0.0.1"
    },
  ]
}
```

결과 JSON에 "timezone" 항목에 설정된 타임존 값이 되돌아온다.

Release Note

7.0 Release Note

7.0 변경 및 추가 기능

목차

Tag 테이블 Rollup의 시간단위 설정 기능 지원

기존 버전에서는 Tag 테이블에 대한 기본 Rollup 작업이 1초, 1분, 1시간 단위로 수행되었다.

하지만 Rollup 시간 단위를 변경할 수 없어서 데이터가 초 이상의 간격으로 들어올 경우 불필요한 자원을 사용하는 문제가 있었다.

Machbase 7.0 에서는 Rollup의 시간단위 설정 기능을 통해 사용자가 원하는 Rollup 만 생성할 수 있고 실행 주기도 원하는대로 설정할 수 있도록 하였다.

Rollup 생성 문법을 다음과 같다.

syntax

```
-- 생성
CREATE ROLLUP rollup_name FROM tag_table_name INTERVAL timesepc;

timespec : integer time_unit
time_unit : SEC, MIN, HOUR

-- 시작
EXEC ROLLUP_START(rollup_name);

-- 중지
EXEC ROLLUP_STOP(rollup_name);

-- 삭제
DROP ROLLUP rollup_name;
```

example

```
CREATE TAG TABLE tag (name VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, val

CREATE ROLLUP rollup_30_sec FROM tag INTERVAL 30 SEC;
CREATE ROLLUP rollup_10_min FROM rollup_30_sec INTERVAL 10 MIN;
CREATE ROLLUP rollup_1_hour FROM rollup_10_min INTERVAL 1 HOUR;

EXEC ROLLUP_START(rollup_30_sec);
EXEC ROLLUP_START(rollup_10_min);
EXEC ROLLUP_START(rollup_1_hour);

..
..

EXEC ROLLUP_STOP(rollup_1_hour);
EXEC ROLLUP_STOP(rollup_10_min);
EXEC ROLLUP_STOP(rollup_1_sec);

DROP ROLLUP rollup_1_hour;
DROP ROLLUP rollup_10_min;
DROP ROLLUP rollup_30_sec;
```

다중 Tag 테이블 지원

기존 버전에서는 한개의 Tag 테이블만 생성할 수 있는 문제를 해결하였다.

다중 Tag 테이블을 지원함으로써 다양한 스키마 형태로 들어오는 PLC 데이터를 효과적으로 저장할 수 있게 되었다.

테이블 이름 또한 자유롭게 지정할 수 있고 SYS가 아닌 일반 유저로 생성 가능하다.

example

```
CREATE TAG TABLE tag (tagid VARCHAR(50) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE SUMMARIZED);
CREATE TAG TABLE imotbl (tagid VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE SUMMARIZED, imo INTEGE
CREATE TAG TABLE shi985 (tagid VARCHAR(20) PRIMARY KEY, time DATETIME BASETIME, value DOUBLE SUMMARIZED, level INTE
```

```
Mach> show tables;
USER_NAME            DB_NAME                                                  TABLE_NAME
-------------------------------------------------------------------------------------------------------------------
SYS                  MACHBASEDB                                               IMOTBL
SYS                  MACHBASEDB                                               SHI985
SYS                  MACHBASEDB                                               TAG
SYS                  MACHBASEDB                                               _IMOTBL_DATA_0
SYS                  MACHBASEDB                                               _IMOTBL_META
SYS                  MACHBASEDB                                               _SHI985_DATA_0
SYS                  MACHBASEDB                                               _SHI985_DATA_1
SYS                  MACHBASEDB                                               _SHI985_META
SYS                  MACHBASEDB                                               _TAG_DATA_0
SYS                  MACHBASEDB                                               _TAG_DATA_1
SYS                  MACHBASEDB                                               _TAG_DATA_2
SYS                  MACHBASEDB                                               _TAG_DATA_3
SYS                  MACHBASEDB                                               _TAG_META
[13] row(s) selected.
```

TAG 데이터 저장 구조 변경

디스크 사용량 감소를 위해 데이터 압축방식(delta compress)을 추가하고 칼럼단위로 저장하도록 데이터 구조가 변경되었다.

기존 version 대비 데이터사용량 비교 테스트 결과 디스크 사용량이 최대 37% 감소하였다.

테스트환경	디스크사용량 감소	감소율
Tag 10,000건 RowCount: 10억건 Rollup(O)	11,105 MB -> 6,907 MB	37.80% 감소
Tag 10,000건 RowCount: 10억건 Rollup(O)	11,007 MB -> 8,987 MB	18.35% 감소
Tag 10,000건 RowCount: 10억건 Rollup(O)	14,836 MB -> 12,752 MB	14.05% 감소

Tag Table Index Memory 사용량 절감

TAG테이블 생성 시 Memory를 Index Build를 위한 Index Memory를 고정적으로 과도하게 할당하여 Memory를 효율적으로 사용할 수 없는 문제가 있어 Index Memory를 필요한 시점에 필요한 만큼의 Memory만 동적으로 할당하는 방식으로 변경하여 복수의 TAG테이블을 생성 및 동작이 가능하도록 하였다.

7.0에서는 아래와 같은 테스트 환경에서 기존 version에 비해 최대 Memory 사용량이 약 40%가량 감소 했다.

- 테스트환경: Tag 테이블 1개(partition 4개), Tag 10,000건 / 데이터 10억건 성능 테스트, PROCESS_MAX_SIZE 4GB

결과적으로 PROCESS_MAX_SIZE를 16GB로 설정 시 TAG 테이블(PARITITION 4개 기준)을 2개만 생성할 수 있었으나, 7.0에서는 동일설정 하에서 TAG테이블을 10개 이상 생성하여 사용할 수 있게 되었다.

운영환경에 따른 메모리 설정 가이드

초당 최대 입력 건수: append application은 TAG 테이블 수만큼 동작하는 상황에서 모든 append application의 초당 입력하는 데이터 합

(테이블당 append application 1개씩 동작)

운영환경				시스템 동작을 위한 최소 설정				시험장비사양		
TAG 테이블 수	테이블당 Tag 수	초당 최대 입력 건수	Session 수 (append 포함)	TAG_DATA_PART_SIZE	TAG_PARTITION_COUNT	PROCESS_MAX_SIZE	TAG_CACHE_MAX_MEMORY_SIZE	CPU	MEM	DISK
1	100,000	1,200,000 / sec	4	16 M	1	2 GB	128 MB	16 cores	64 GB	SSD 1 TB
1	100,000	1,000,000 / sec	4	4 M	1	1 GB	32 MB	16	64	SSD 1

								cores	GB	TB
1	100,000	800,000 / sec	4	1 M	1	512 MB	8 MB	16 cores	64 GB	SSD 1 TB
8	100,000	4,000,000 / sec	11	16 M	1	8 GB	256 MB	16 cores	64 GB	SSD 1 TB
32	40,000	6,000,000 / sec	35	16 M	1	16 GB	1024 MB	16 cores	64 GB	SSD 1 TB
32	40,000	4,000,000 / sec	35	4 M	1	8 GB	256 MB	16 cores	64 GB	SSD 1 TB
64	20,000	4,000,000 / sec	67	4 M	1	8 GB	512 MB	16 cores	64 GB	SSD 1 TB
64	20,000	1,000,000 / sec	67	1 M	1	4 GB	128 MB	16 cores	64 GB	SSD 1 TB
128	10,000	4,000,000 / sec	131	4 M	1	8 GB	1024 MB	16 cores	64 GB	SSD 1 TB
128	10,000	1,000,000 / sec	131	1 M	1	4 GB	256 MB	16 cores	64 GB	SSD 1 TB

TAG 테이블의 수에 비례하여 TAG_CACHE_MAX_MEMORY_SIZE설정 값이 높아져야하며, TAG 테이블의 생성시 TAG_CACHE_MAX_MEMORY_SIZE가 부족한 경우 테이블 생성이 실패할 수 있다.

적정 TAG_CACHE_MAX_MEMORY_SIZE 계산 방법 및 테이블 생성 성공 조건

t_i^{PC} : TAG Table i의 Partition Count

t_i^{PS} : TAG Table i의 Data Partition Size

m^T : TAG_CACHE_MAX_MEMORY_SIZE

n^T : Tag 테이블의 수

$$m^T > \sum_{i=1}^{n^T} (t_i^{PC} \times (t_i^{PS} \times 2))$$

TAG Parition이 4개이고, Partition Size가 16MB 인 경우

TAG_CACHE_MAX_MEMORY_SIZE는 적어도 128MB 보다

크게 설정 해야 한다.

128 MB = 4 * 16MB * 2

TAG ID기반 통계값 생성 / 조회

Tag별 통계값 조회시 응답시간이 너무 긴 문제가 있으며 이를 개선하기 위해 Tag별 통계를 사전에 구성해 놓고,
조회의 편의성을 위해 view와 같은 형태로 조회 기능을 제공한다.
기존 version에서는 TAG통계조회시 data scan으로 인해 응답시간이 길었으나, 7.0에서는 필요한 통계정보를 사전에 구성함으로써 응답시간이 크게 단축 되었다.

Tag 통계 정보

column	description
USER_ID	tag table user ID
TABLE_NAME	tag table name
TAG_ID	통계 정보의 Tag ID (사용자 관점에서는 Name이 출력된다)
MIN_TIME	현재까지 입력된 시간값의 최소치 (입력순서와 관계 없음)
MAX_TIME	현재까지 입력된 시간값의 최대치 (입력순서와 관계 없음)
MIN_VALUE	해당 tag의 Summarized 컬럼의 최소 값
MIN_VALUE_TIME	min_value와 같이 입력된 time

column	description
MAX_VALUE	해당 tag의 Summarized 컬럼의 최대 값
MAX_VALUE_TIME	max_value와 같이 입력된 time
ROW_COUNT	해당 Tag ID의 Row 개수
RECENT_ROW_TIME	마지막에 입력된 Row의 Basetime 컬럼값

- query

Tag 테이블을 생성할때마다 통계정보를 저장하는 view 를 같이 생성한다. 통계정보를 조회하고자 할때 view 로부터 조회한다.

view table name : "V$*name*_STAT" . (name 은 tag 테이블 이름)

```
-- ex)
SELECT min_time, max_time FROM v$tag_stat WHERE tagid = 'tag-01';

-- ex) 여러개의 tag
SELECT min_time, max_time FROM v$tag_stat WHERE tagid IN ('tag-01', 'tag-02', 'tag-03');
```

min_time (또는 max_time) 에 입력된 value 를 알고싶다면 아래와 같이 조회가 가능하다.

```
SELECT value FROM tag WHERE tagid = 'tag-01' AND time = (SELECT min_time FROM v$tag_stat WHERE tagid = 'tag-01')
```

JSON 타입 지원

지원 배경

최근 TEXT 타입의 데이터를 전송하는 장비도 사용되고 있다.

JSON 타입을 지원하면, TEXT 타입 뿐만 아니라 임의의 데이터 타입을 전송할 수 있다.

따라서, 칼럼의 갯수나 구조가 비정형으로 가능하기 때문에 확장성이 매우 높아지는 효과를 기대할 수 있다.

장점

- 별도의 확장 칼럼을 추가할 필요가 없기 때문에, 스키마 제약이 사라진다.
- 사용자가 임의의 데이터 타입을 마음대로 넣을 수 있다.
- JSON 데이터 특성으로 사용성 및 편의성이 높고, 데이터를 분리할 필요가 없다.

데이터 범위

- JSON data 길이 : 1 ~ 32768 (32K)
- JSON path 길이 : 1 ~ 512

데이터 생성

- JSON 키워드를 사용하여 JSON 타입의 칼럼을 지정할 수 있다.

```
CREATE TABLE jsontbl (name VARCHAR(20), jval JSON);
```

데이터 삽입

- JSON 포맷에 맞는 TEXT를 입력하여 데이터를 삽입할 수 있다.
- JSON 포맷에 맞지 않을 시, ERROR를 출력한다.

```
-- Single
INSERT INTO jsontbl VALUES("name1", '{"name":"test1"}');
-- Multi
INSERT INTO jsontbl VALUES("name2", '{"name":"test2", "value":123}');
-- Nested
INSERT INTO jsontbl VALUES("name3", '{"name":{"class1": "test3"}}');
-- Array
INSERT INTO jsontbl VALUES("name4", '{"myarray": [1, 2, 3, 4]}');
```

데이터 추출

- JSON 관련 함수를 사용하거나, Operator(->)를 사용하여 JSON 데이터를 부분적으로 추출할 수 있다.

```
-- 함수 사용
SELECT name, JSON_EXTRACT(jval, '$.name') FROM jsontbl;
SELECT name, JSON_EXTRACT_INTEGER(jval, '$.myarray') FROM jsontbl;
SELECT name, JSON_TYPEOF(jval, '$.name.class1') FROM jsontbl;

-- Operator(->) 사용
SELECT name, jval->'$.name' FROM jsontbl;
SELECT name, jval->'$.myarray' FROM jsontbl;
SELECT name, jval->'$.name.class1' FROM jsontbl;
```

JSON 관련 함수

함수명	설명	비고
JSON_EXTRACT(JSON 칼럼명, 'json path')	해당 값을 string type으로 출력한다. (해당 객체가 없을 경우 ERROR를 출력한다.)	• JSON 객체 혹은 배열형 : 포함된 모든 객체를 문자열로 변환해서 리턴 • 문자열 : 그대로 리턴 • 숫자형 :문자열로 변환하여 리턴 • boolean 형 : "True" or "False" 리턴
JSON_EXTRACT_STRING(JSON 칼럼명, 'json path')	해당 값을 string type으로 출력한다. (해당 객체가 없을 경우 NULL을 출력한다.) operator(->)와 같은 결과를 출력한다.	• JSON 객체 혹은 배열형 : 포함된 모든 객체를 문자열로 변환해서 리턴 • 문자열 : 그대로 리턴 • 숫자형 :문자열로 변환하여 리턴 • Boolean 형 : "True", "Fase" 리턴
JSON_EXTRACT_INTEGER(JSON 칼럼명, 'json path')	해당 값을 64비트 integer type으로 출력한다. (해당 객체가 없을 경우 NULL을 출력한다.)	• JSON 객체 혹은 배열형 : NULL 리턴 • 문자열 : 변환하여 리턴하고, 변환 실패시 NULL 리턴 • 숫자형 : 64비트 정수 리턴 • boolean 형 : "True"는 1 , "False"는 0 리턴
JSON_EXTRACT_DOUBLE(JSON 칼럼명, 'json path')	해당 값을 부동소수점 64비트 double type으로 출력한다. (해당 객체가 없을 경우 NULL을 출력한다.)	• JSON 객체 혹은 배열형 : NULL 리턴 • 문자열 : 변환하여 리턴하고, 변환 실패시 NULL 리턴 • 숫자형 : 64비트 실수 리턴 • boolean 형 : "True"는 1.0, "False"는 0.0 리턴
JSON_TYPEOF(JSON 칼럼명, 'json path')	해당 값의 타입을 반환한다.	• None : 해당 키가 존재하지 않음 • Object : 객체형 • Integer : 정수형 • Real : 실수형 • String : 문자형 • True/False : Boolean • Array :배열형 • Null : NULL
JSON_IS_VALID('json string')	json string이 json format에 유효한지 확인한다.	• 0 : False • 1 : True

자주 묻는 질문(FAQ)

쿼리 에러를 Property 수정하여 해결하는 방법

쿼리를 실행한 후 메모리 부족 에러가 발생하였을 때 Property를 수정하여 해결하는 방법을 설명한다.

목차

쿼리 실행 시 메모리가 부족하여 에러가 발생함

아래와 같은 이유로 쿼리를 실행하는데 필요한 메모리를 제한하고 있다.

- 특정 쿼리가 메모리를 너무 많이 사용하는 경우, 동시에 실행 중인 다른 쿼리가 메모리 부족으로 실행이 안되는 경우가 발생할 수 있다.

이를 방지하기 위해 하나의 쿼리가 사용가능한 메모리 최대 사이즈 Property 값을 증가시켜 에러를 해결할 수 있다.

MAX_QPX_MEM Property로 하나의 SQL에서 사용할 수 있는 최대 사용 가능 메모리를 관리한다.

실행 중 설정하는 방법및 메모리가 부족하여 발생하는 에러메세지및 TRC메세지도 SET MAX_QPX_MEM 페이지를 참고한다.

SET 명령어로 Property 값을 설정하면 마크베이스 재시작 시 설정 값이 적용되지 않으므로 machbase.conf 파일도 아래와 같이 함께 수정하여야 한다.

Fog Edition

machbase.conf의 MAX_QPX_MEM 를 보다 큰 값으로 수정한다.

Cluster Edition

Fog edtion과 동일하다. 단, 모든 클러스터 노드의 machbase.conf 를 수정해야 한다.