IRIS Programmer Guide

Mobigen, November 20, 2015

목차

1 지원 정보 **7**

1.1 문서 버전 7

1.2 IRIS 버전 7

1.3 사용 문의 및 기술 정보 7

2 IRIS Overview **8**

2.1 Overview 8

2.1.1 IRIS, 주요 지원 항목 9

2.2 IRIS의 특징 9

2.2.1 실시간 (Real-Time) 통계 / 분석 9

2.2.2 대용량 데이터의 처리 9

2.2.3 Scale-Out, 확장성 10

2.2.4 Fault-Tolerance, 무정지 서비스 10

2.2.5 분산 / 전역 테이블 간의 JOIN 지원 10

2.3 IRIS 구성 12

2.3.1 주요 프로세스 [노드 모니터링] 12

2.3.2 주요 프로세스 쿼리 실행 14

2.3.3 주요 프로세스 데이터 관리 15

2.3.4 IRIS 디렉토리 구성 16

2.4 IRIS 의 테이블 종류 19

2.4.1 System 테이블 20
2.4.2 Global 테이블 20
2.4.3 Local 테이블 20
2.5 KEY / PARTITION 20
2.5.1 PARTITIONKEY 22
2.5.2 PARTITIONDATE 22
2.5.3 KEY AND PARTITION 23

3 IRIS 사용하기 25

3.1 IPLUS 25
3.2 사용자 관리 27
3.2.1 나는 누구인가? 28
3.2.2 사용자 리스트 28
3.2.3 사용자 추가 29
3.2.4 사용자 패스워드 변경 31
3.2.5 사용자 삭제 31
3.3 테이블 관리 33
3.3.1 테이블 리스트 33
3.3.2 테이블 생성 34
3.3.3 테이블 정보 확인 36
3.3.4 테이블 삭제 37
3.3.5 테이블 삭제와 데이터 삭제 38

3.4 데이터 관리 39
3.4.1 DROP BACKEND 39
3.4.2 PL의 동작 방식 40
3.4.3 PR의 동작 방식 40
3.4.4 PL/PR의 룰 41
3.4.5 PL / PR 동작 옵션 관련 명령어 42
3.5 세션 48
3.5.1 세션 명령어 48
3.5.2 세션 리스트 49
3.5.3 세션 정보 51
3.5.4 세션 종료 52
3.6 휴지통 53
3.6.1 휴지통 관리 명령어 53
3.6.2 휴지통 기능 사용하기 56

4 API **71**

4.1 JDBC 71
4.1.1 사용 방법 72
4.2 Java 74
4.2.1 사용 방법 75
4.2.2 API 상세 77
4.3 Python 79

4.3.1 사용 방법 79
4.3.2 API 상세 81
4.4 C / C++ 82
4.4.1 사용 방법 82
4.4.2 API 상세 : c 88
4.4.3 API 상세 : cpp 89
4.5 C# 90
4.5.1 사용 방법 91
4.5.2 API 상세 93
4.6 ODBC 94
4.6.1 사용 방법 (Linux) 94
4.6.2 사용 방법 (Windows) 98
4.6.3 지원함수 목록 99
4.6.4 사용시 유의사항 100

5 쿼리 사용법 101

5.1 로딩 101
5.2 JOIN 101
5.2.1 INNER JOIN 102
5.2.2 LEFT OUTER JOIN 102
5.2.3 기타 : Oracle JOIN 과 비교 102
5.3 HINT 103

5.3.1 HINT 종류 104
5.4 지원 함수 목록 105
5.4.1 별첨항목 107
5.4.2 지원함수 명세 109

1 지원 정보

1.1 문서 버전

0.1

1.2 IRIS 버전

1.5.1

1.3 사용 문의 및 기술 정보

(주) 모비젠

본사

- 135-280 서울 강남구 대치동 967-3번지 KM빌딩 2층, (주) 모비젠
- T : 02 - 538 - 9360
- F : 02 - 538 - 9369

기술 연구소

- 135-280 서울 강남구 대치동 967-3번지 KM빌딩 5층, (주) 모비젠
- T : 02 - 538 - 9364
- F : 02 - 538 - 9368

모비젠 IRIS 기술 지원

- T : 02 - 538 - 9364
- M : iris@mobigen.com

2 IRIS Overview

본 항목에서는 IRIS 시스템의 기본 개념에 대하여 설명합니다.

2.1 Overview

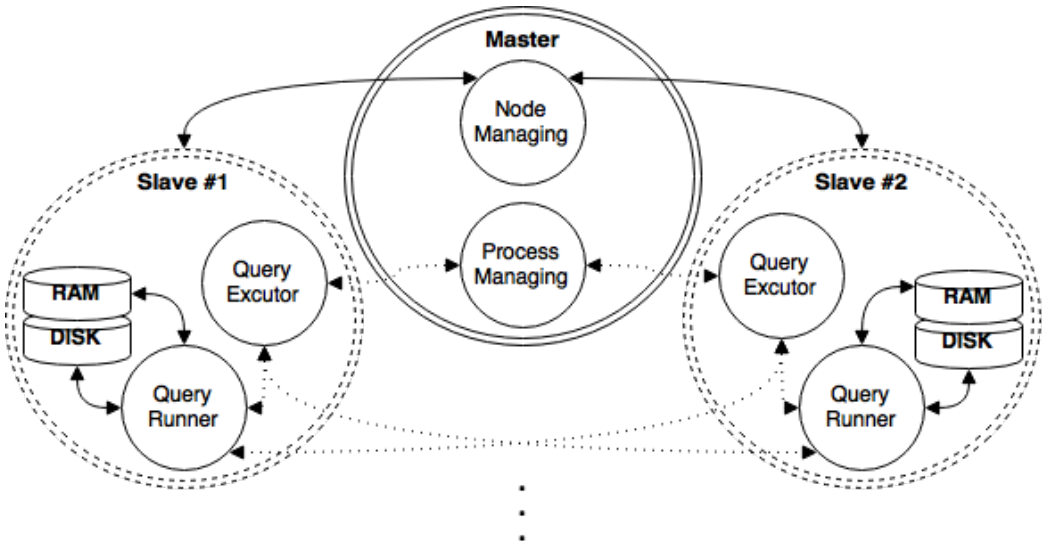

그림 1: IRIS Overview

IRIS는 SQL을 지원하는 분산 아키텍쳐 기반의 Big Data 데이터 베이스 솔루션 입니다. 단일 시스템으로 구성되어 있는 일반 DBMS와 다르게 IRIS는 쿼리에 대한 데이터 및 처리 프로세스의 분산을 관리하는 마스터 노드와 분산된 쿼리를 수행하는 슬레이브 노드로 구성됩니다. IRIS는 슬레이브 노드의 확장 및 증설이 용이하고, 데이터 저장 공간으로 메모리와 디스크를 모두 사용하는 하이브리드 (In-Memory, On-Disk) 방식의 데이터 베이스 입니다.

2.1.1 IRIS, 주요 지원 항목

- Linux 기반, IRIS의 모든 노드는 서버에서 널리 사용되고 있는 Linux를 최적화 커스터 마이징 하여 사용합니다.
- SQL 지원, IRIS는 자체적으로 SQL을 지원하여 SQL을 아는 모든 사용자 및 모든 개발자들이 간단한 교육만으로 쉽게 사용할 수 있는 환경을 제공합니다.
- ClientPackage, IRIS는 IRIS 전용 CLI와 데이터베이스 접근에 적용되는 JDBC를 비롯한 각종 API (JAVA, C, C#, Python)를 포함한 패키지를 제공합니다.

2.2 IRIS의 특징

2.2.1 실시간 (Real-Time) 통계 / 분석

IRIS의 슬레이브 노드에서는 최근 시점에 입력된 데이터들을 사용자가 정의한 일정 기간 동안 메모리상에서 관리합니다. 따라서 메모리상의 데이터들을 대상으로 고속의 조회/통계/분석 처리가 가능합니다.

2.2.2 대용량 데이터의 처리

IRIS는 분산 처리 구조를 채택하고 있으므로 PetaByte(1,000TB) 수준의 대용량 BIG Data의 관리가 가능합니다. 수십 TB 수준의 데이터를 관리하는 수십대의 슬레이브 노드를 하나의 시스템으로 구성했기 때문입니다. 따라서 성능, 가격 등의 문제로 기존에는 처리가 불가능했던 대용량의 Big Data들을 IRIS를 이용하여 분석이 가능합니다.

2.2.3 Scale-Out, 확장성

IRIS는 Scale-Out이 가능한 분산 구조로서 용량의 증설, 슬레이브 노드의 추가에 제한을 받지 않습니다. 모든 데이터의 다중화를 통하여 IRIS 확장 작업 중에도 지속적인 서비스가 가능합니다.

2.2.4 Fault-Tolerance, 무정지 서비스

IRIS로 저장되는 데이터는 최소 2개 이상 (기본 설정이 2개) 슬레이브 노드로 중복 저장됩니다. 이러한 데이터의 이중화를 통하여 일부 슬레이브 노드의 장애 상황에서도 서비스를 유지할 수 있으며, 데이터 손실 및 손상을 방지할 수 있습니다. 또한 슬레이브 노드의 부하를 균등하게 유지합니다. 마스터 노드는 Active / Standby 이중화를 통하여 무정지 서비스를 제공합니다.

2.2.5 분산 / 전역 테이블 간의 JOIN 지원

일반적인 Big Data 시스템은 노드간의 데이터 공유를 하지 않는 "Shared Nothing" 개념을 따릅니다. Shared Nothing 구조인 Big Data DB에서 Join 연산은 그것 자체로 매우 중요한 이슈입니다. IRIS에서는 분산 저장되어 있는 Big Data Table 간에 Join 연산을 지원하지는 않지만, 일반적인 관리 S/W 들이 필요로하는 Join을 지원하기 위해서 분산된 Big Data Table, 즉 Local 테이블과 Global 테이블이라는 개념을 도입하여 Local / Global 테이블 간의 Join을 지원합니다.

- Local 테이블 : 슬레이브 노드에 분산되어 관리되는 테이블입니다. 각 테이블의 데이터는 모든 슬레이브 노드에 분산되어 저장되므로 Local 테이블 간의 Join은 지원하지 않습니다.

- Global 테이블 : 마스터 노드, 슬레이브 노드에 모두 동일한 데이터가 유지되는 테이블 입니다. 이 테이블은 모든 노드에 동일한 데이터를 가지고 있으므로 Join을 지원합니다.

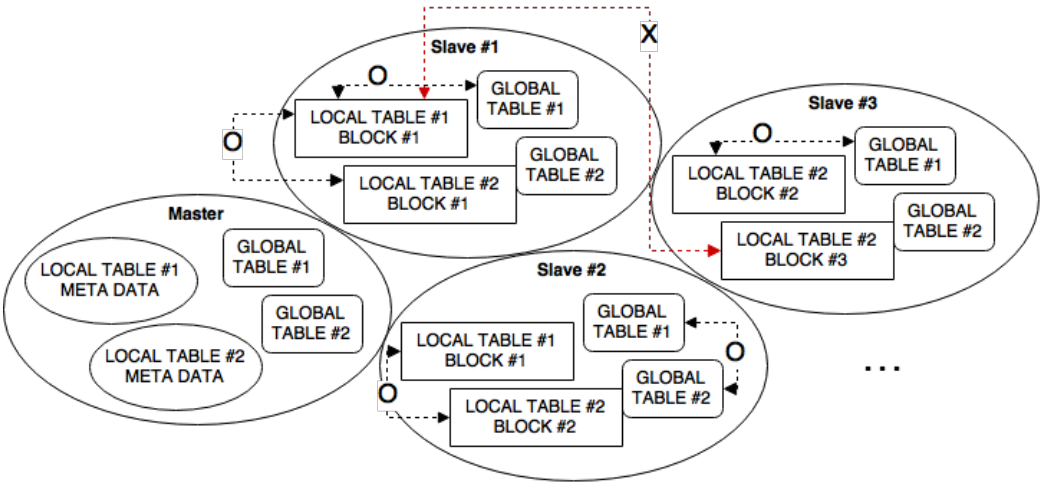

그림 2: Global & Local Table Join

위의 그림은 Global, Local 테이블간의 Join 가능성을 표현한 그림입니다. 먼저 Global 테이블간의 Join의 가능성입니다. 위의 그림을 참고하면 아래와 같은 결과가 나타납니다.

```
Global Table #1 & Global Table #2 : O
```

이므로 Global Table 간의 Join은 가능합니다. 다음은 Global Table 1과 Local Table 1 간의 Join 입니다.

```
Global Table #1 & Local Table #1 Block #1 : O
Global Table #1 & Local Table #1 Block #2 : O
```

다음은 Local Table 1과 Local Table 1간의 Join 입니다.

```
Local Table #1 Block #1 & Local Table #2 Block #1 : O
Local Table #1 Block #1 & Local Table #2 Block #2 : O
Local Table #1 Block #1 & Local Table #2 Block #3 : X
```

이므로 Join 불가입니다.

2.3 IRIS 구성

IRIS 시스템은 마스터 / 슬레이브 구조를 가지고 있으며 데이터 처리를 위한 다양한 프로세스들이 서로 연결되어 동작하고 있습니다. 다음 그림은 IRIS 시스템에 대한 프로세스 구성 요소를 표현한 그림입니다.

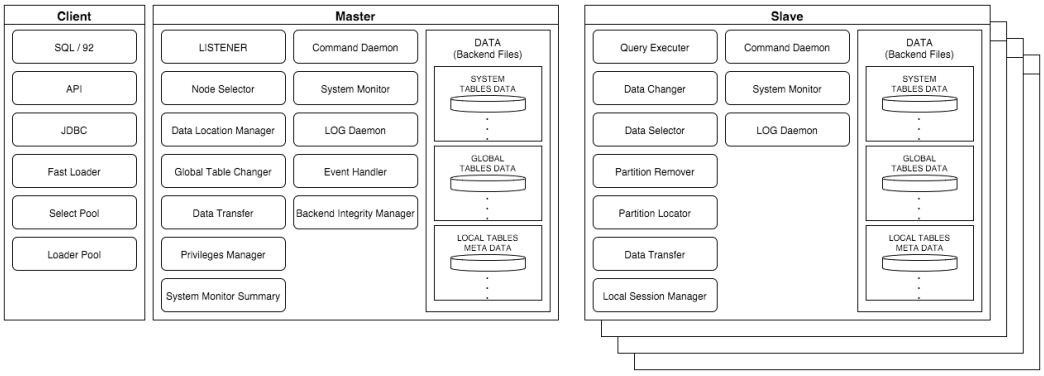

그림 3: IRIS S/W Architecture

2.3.1 주요 프로세스 [노드 모니터링]

2.3.1.1 Event Handler [EHD] 각각의 노드에서 발생하는 이벤트와 각 노드의 System Monitor 로부터 발생하는 상태 정보를 바탕으로 노드 상태를 확인 / 관리하고 장애 발생시 노드의 상태를 변경합니다. 또한 사용자가 확인할 수 있는 메시지를 생성합니다.

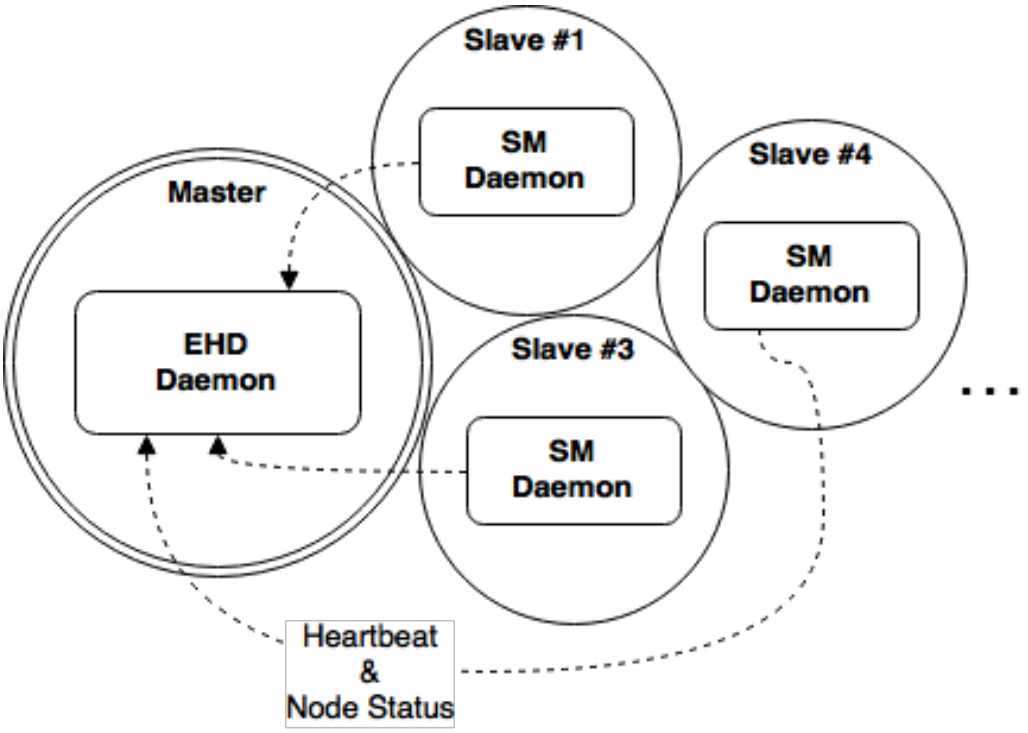

그림 4: IRIS Node Monitoring

2.3.1.2 System Monitor [SM] 실행된 각 노드의 시스템 정보, 테이블 정보, 상태 정보를 주기적으로 Event Handler로 전송합니다.

2.3.2 주요 프로세스 쿼리 실행

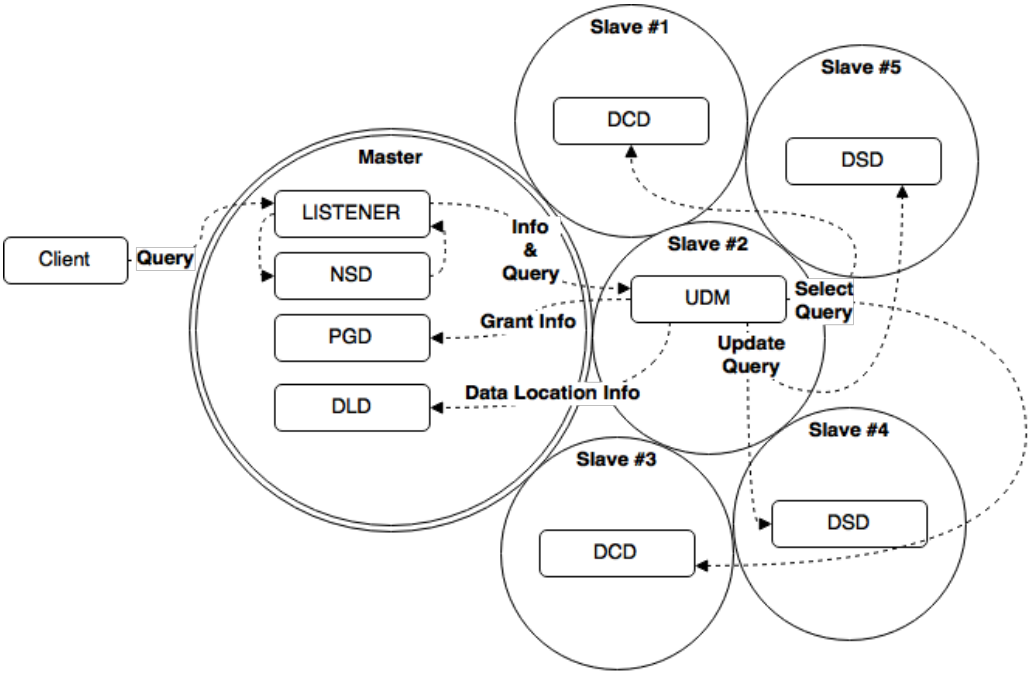

그림 5: IRIS Run Query

2.3.2.1 LISTENER [LISTENER] 클라이언트의 접속 요청을 받아들이며, Node Selector와 내부 통신하여 클라이언트의 쿼리 요청을 처리하기 위한 Query Executer를 할당합니다. 그 후 클라이언트와 Query Executer 간의 데이터 전달 역할을 수행합니다.

2.3.2.2 Node Selector [NSD] 클라이언트 작업 요청을 처리하기 위한 Query Executer를 할당합니다.

2.3.2.3 Data Location Manager [DLD] KEY / PARTITION 에 의해 분배되어 있는 데이터의 위치 정보, 데이터 상태 정보를 생성/조회/수정/관리하는 역할을 수행합니다.

2.3.2.4 Privileges Manager [PGD] 데이터의 접근 권한을 관리합니다.

2.3.2.5 Query Executer [UDM] 사용자 요청에 대한 작업을 수행하는 데몬으로, 사용자의 Query를 분석하여 작업 과정을 생성하고 작업 종류에 따라 Data Changer 혹은 Data Selector 에 데이터 처리 명령을 내리며, 그 결과들을 수집하여 최종 결과를 생성하는 여할을 담당합니다.

2.3.2.6 Data Changer [DCD] 실제 데이터가 저장된 Backend의 정보를 변경하기 위해 사용되는 데몬입니다.

2.3.2.7 Data Selector [DSD] Backend의 정보를 수집하기 위해 사용되는 데몬입니다.

2.3.3 주요 프로세스 데이터 관리

2.3.3.1 Partition Remover [PR] 데이터 삭제를 담당하는 프로세스입니다. Disk 보관주기가 지난 파일을 삭제하며, Lock, TMP 파일 등을 삭제하는 데몬입니다. 또한 깨진 파일을 복구합니다.

2.3.3.2 Partition Locator [PL] 데이터 위치를 관리하는 프로세스입니다. Ram, SSD 보관 주기에 따라 데이터 위치를 이동 시킵니다.

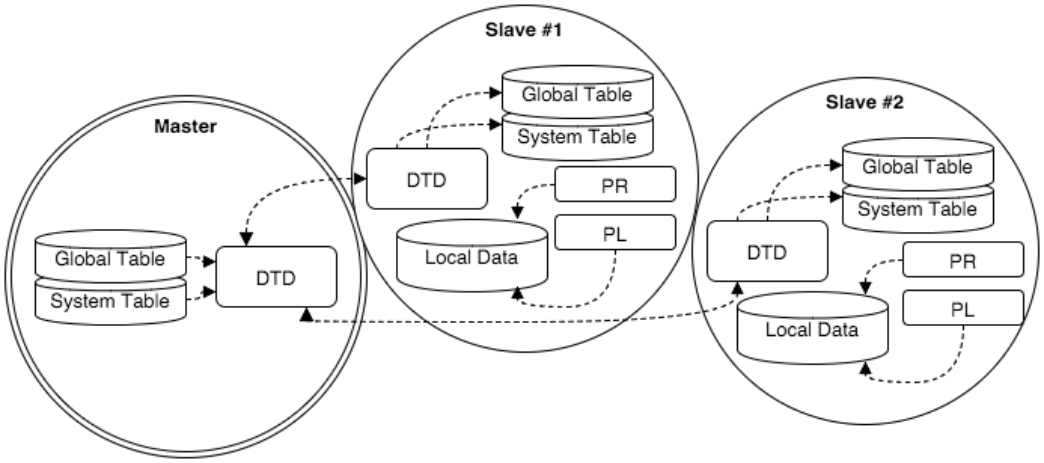

그림 6: Data Managing

2.3.3.3 Data Transfer [DTD] Global / System 테이블의 동기화를 위한 데몬입니다.

2.3.4 IRIS 디렉토리 구성

본 챕터에서는 IRIS 디렉토리 구성에 대해서 설명합니다. 아래 그림은 IRIS의 주요 디렉토리 구조입니다.

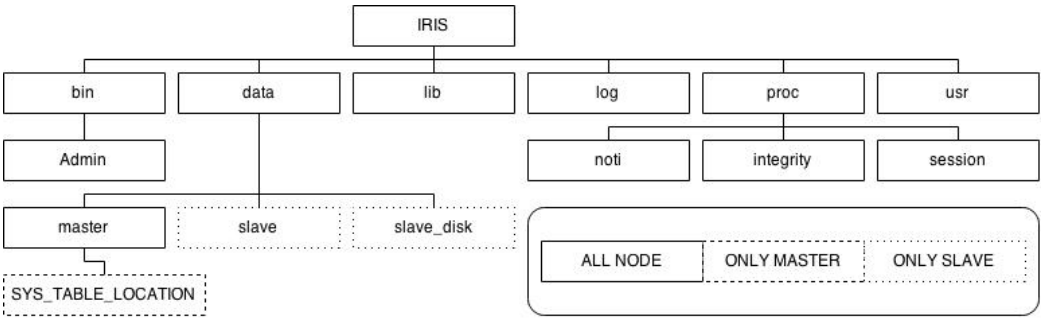

그림 7: IRIS Directory Architecture

가장 상위 폴더는 IRIS며, 그 아래에 bin, data, lib, log, proc, usr 폴더가

존재합니다. 각각의 폴더에 대한 설명은 다음과 같습니다.

2.3.4.1 bin 데몬 실행에 필요한 실행 파일들이 있는 디렉토리입니다.

2.3.4.1.1 Admin bin 디렉토리 밑에 있는 디렉토리로 IRIS를 관리하기 위한 실행파일들이 존재합니다. 대표적인 것으로 IRIS-Startup, IRIS-Shutdown, NodeEnable, NodeAdd 등이 존재합니다.

2.3.4.2 data IRIS에서 사용하는 데이터가 저장되는 디렉토리입니다. 하위 디렉토리로는 master, slave, slave_disk 등이 있으며 master 디렉토리는 마스터, 슬레이브 노드에 모두 존재하며 slave, slave_disk 디렉토리는 슬레이브 노드에만 존재합니다.

2.3.4.2.1 master master 디렉토리에는 System 테이블과 Global 테이블이 존재합니다. 하위 디렉토리에는 SYS_TABLE_LOCATION이 있습니다. SYS_TABLE_LOCATION 디렉토리는 마스터 노드에만 있으며 Local 테이블의 메타 데이터가 저장됩니다.

2.3.4.2.2 slave slave 디렉토리는 ramfs, 램 파일 시스템으로 램에 데이터를 저장하기 위한 공간입니다. IRIS에 처음 들어온 데이터는 이 곳에 저장됩니다. 이 디렉토리는 슬레이브 노드에만 존재합니다.

2.3.4.2.3 slave_disk slave_disk는 IRIS 슬레이브 노드에만 존재하는 디렉토리로 실제 데이터를 저장하는 디스크의 정보를 담고 있는 디렉토리입니다. 실제 사용하는 폴더의 링크가 저장되며 일반적으로 part00, part01, part02 … 형태로 링크가 생성되어 있습니다.

2.3.4.3 lib lib 디렉토리는 IRIS 코어 라이브러리를 포함하는 디렉토리입니다.

2.3.4.4 log log 디렉토리는 IRIS에서 발생하는 로그를 저장하는 디렉토리입니다.

2.3.4.5 proc proc 디렉토리에서는 IRIS에서 발생하는 이벤트를 저장합니다. 해당 디렉토리 아래에는 noti, session, integrity 정보를 저장하는 디렉토리가 존재합니다.

2.3.4.5.1 noti 특정 노드에서 발생한 이벤트를 저장합니다. 저장되는 파일명은 다음과 같습니다.

```
[이벤트 발생 시간]_[이벤트 발생 노드 IP].[이벤트 발생 등급]
```

마지막 확장자는 이벤트 발생 등급으로 아래와 같이 등급을 구분합니다.

- INFO : 단순 정보
- WARN : 주의 정보
- BUSY : 노드 상태가 BUSY로 변함을 알림
- FATAL : 위험 정보

2.3.4.5.2 session session 관련된 정보가 저장되는 디렉토리입니다.

2.3.4.5.3 integrity 데이터에 손상이 갔을 경우 손상된 데이터를 정상 데이터로 저장하기 위해 사용되는 정보를 저장하는 디렉토리입니다. 데이터 복제 수가 1일 경우에는 동작하지 않으며 데이터 복제 수가 2 이상일 경우 목록을 생성 저장하여 정상적인 데이터를 유지합니다.

2.3.4.6 usr IRIS Core 라이브러리를 동작시키는데 필요한 라이브러리를 저장한 디렉토리입니다.

2.4 IRIS 의 테이블 종류

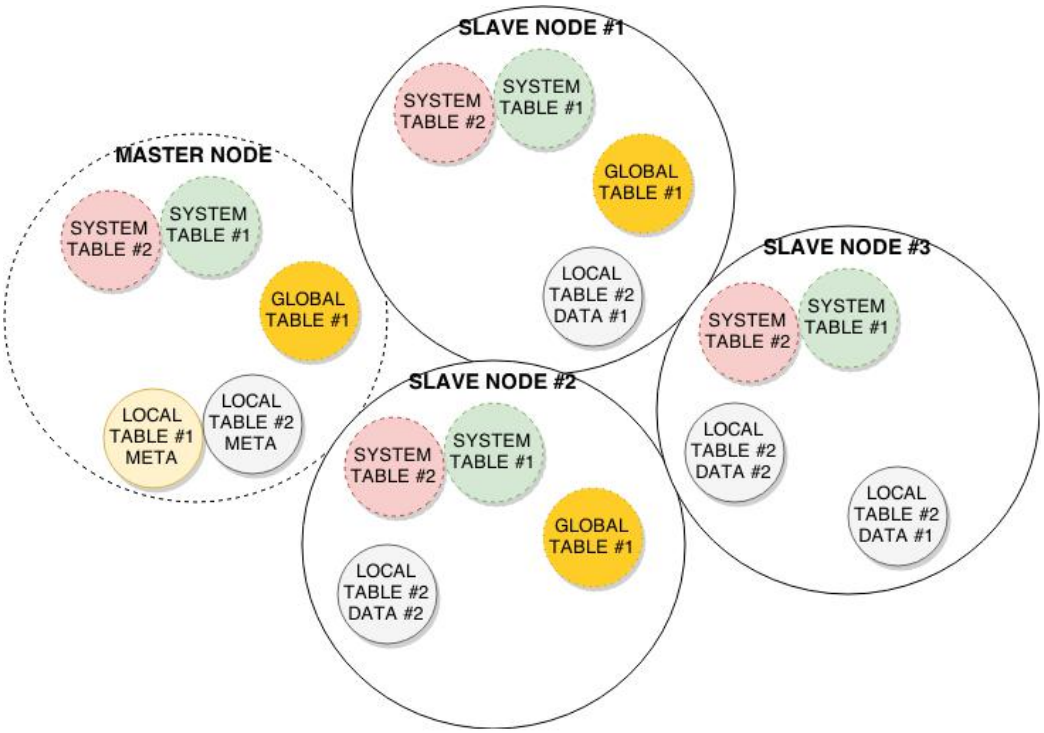

그림 8: 테이블 종류

IRIS의 테이블은 세 가지 종류로 나눌 수 있습니다. SYSTEM / GLOBAL / LOCAL 테이블이 그 종류입니다.

2.4.1 System 테이블

IRIS 시스템에서 사용되는 테이블로 IRIS를 구성하고 있는 정보를 담고 있는 테이블 입니다. 대표적인 예로는 노드 정보를 담고 있는 SYS_NODE_INFO, 사용자 정보를 담고 있는 SYS_USER_INFO 등이 있습니다. 모든 노드에 동일한 데이터가 항상 유지되는 테이블입니다.

2.4.2 Global 테이블

각각의 슬레이브 노드에서 동일한 데이터 접근이 가능한 테이블 입니다. 이 테이블은 주로 데이터 규모가 작고, 변경이 적게 일어나는 데이터에 한 하여 사용하기를 제안하고 있습니다. 데이터 조회는 슬레이브 노드에서 동작하며 데이터 수정은 마스터에서 수행된 후에 각각의 슬레이브 노드와 동기화 됩니다.

2.4.3 Local 테이블

대용량 데이터를 구성하는 경우에 사용되는 테이블입니다. 단일 파일로 기록되지 않고 다수의 슬레이브 노드에 KEY/PARTITION으로 구분된 여러 조각으로 나뉘어 저장됩니다. 마스터 노드에서는 해당 데이터의 위치 정보만 가지고 있으며 실제 데이터는 슬레이브 노드에 분산, 관리됩니다.

2.5 KEY / PARTITION

Key, Partition 은 IRIS 내부적으로 데이터를 분산 저장하기 위한 개념입니다. 즉, 분산되어 저장되는 Local 테이블에 적용되는 개념입니다.

KEY, 데이터베이스에서의 Key 개념과 다름

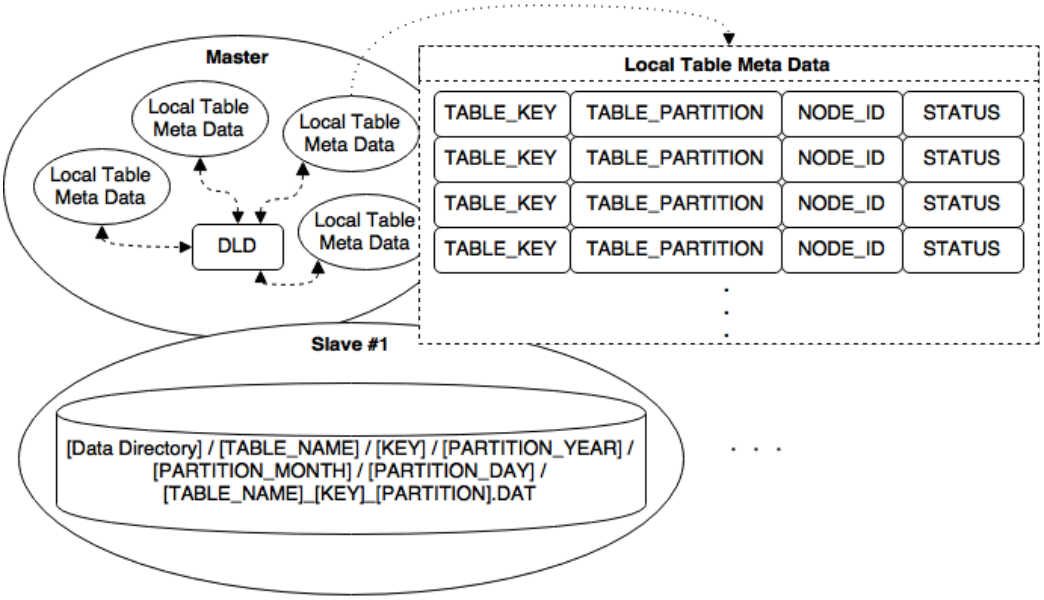

그림 9: Table Key Partition

Key, Partition 은 테이블 생성시 정의한 컬럼의 값을 따릅니다.

```
CREATE TABLE TEST_TABLE (
    A TEXT,
    B TEXT,
    C TEXT
)
datascope LOCAL
ramexpire 60
diskexpire 1440
partitionkey A
partitiondate B
partitionrange 10
;
```

위의 구조를 가지는 테이블 있다고 가정합니다. 먼저 “CREATE TABLE ···. C TEXT)” 까지는 일반적인 CREATE TABLE 구문과 동일합니다. 하지만

하위에 옵션 값이 다른 것을 확인할 수 있습니다. KEY, PARTITION에 대해서 이해하고자 할 때 중요한 부분은 바로 partition으로 시작하는 옵션 값들입니다.

2.5.1 PARTITIONKEY

```
partitionkey A
```

테이블 생성시에 옵션으로 준 partitionkey A 라는 옵션이 바로 Key 컬럼을 설정하는 부분입니다. 특정 데이터의 Key 값은 바로 A 컬럼에 들어있는 값이 됩니다. 예를 들어

```
CAR,20150101000000,KIA
```

라는 데이터가 들어왔을 때 이 값은 Key가 CAR 인 곳에 저장됩니다.

2.5.2 PARTITIONDATE

```
partitiondate B
partitionrange 10
```

테이블 생성시에 옵션으로 준 partitiondate B 라는 옵션이 바로 Partition 컬럼을 설정하는 부분입니다. 특정 데이터의 Partition 값은 바로 B 컬럼에 들어 있는 값이 됩니다. 예를 들어

```
CAR,20150101000000,KIA
CAR,20150101000900,HYUNDAI
CAR,20150101001000,BENZ
```

라는 데이터가 차례대로 들어왔습니다. 먼저 첫 번째 데이터

```
CAR,20150101000000,KIA
```

값의 PARTITION 값은 20150101000000 입니다. 다음 데이터를 보겠습니다.

```
CAR,20150101000900,HYUNDAI
```

입니다. PARTITION 값을 확인하면 20150101000900 임을 확인할 수 있습니다. 하지만 partitionrange 10이라는 값을 확인할 수 있습니다. partitionrange 옵션은 몇분 단위로 PARTITION 값을 나눌것인가에 대한 옵션입니다. 즉, 20150101000900 는 partitionrange 10 옵션에 의해 최종적으로 PARTITION 값 20150101000900 이 됩니다.

```
partitionrange 10
20150101000000 - 20150101000959 : 20150101000000
20150101001000 - 20150101001959 : 20150101001000
20150101002000 - 20150101002959 : 20150101002000
.
.
.
```

위의 예제는 partitionrange 옵션이 10분일 경우에 어떻게 데이터가 저장되는지 볼 수 있는 리스트입니다.

2.5.3 KEY AND PARTITION

IRIS Local 테이블에 저장되는 모든 데이터는 Key, Partition 값을 참조하여 저장됩니다. 예를 들어

```
KIA,20150101000000,K5
HYUNDAI,20150101000000,SONATA
KIA,20150101000901,K3
HYUNDAI,20150101001002,AVANTE
```

라는 값이 있을 때

```
KIA,20150101000000,K5
KIA,20150101000901,K3
    => KEY : KIA, PARTITION : 20150101000000

HYUNDAI,20150101000000,SONATA
    => KEY : HYUNDAI, PARTITION : 20150101000000

HYUNDAI,20150101001002,AVANTE
    => KEY : HYUNDAI, PARTITION : 20150101001000
```

라는 결론이 만들어 집니다. 즉, 위의 두 개의 값은 같은 곳에 저장되고 아래의 값 두 개는 각각 다른 곳에 저장되어 짐을 알 수 있습니다.

다음은 KEY, PARTITION의 특징입니다.

- KEY 컬럼과 PARTITION 컬럼은 변경할 수 없습니다.
- Local 테이블에만 KEY/PARTITION 컬럼을 설정하며 Global 테이블에는 설정하지 않습니다. (설정은 가능하나 동작하지 않습니다. 일반적으로 None을 기입합니다.)
- PARTITION 컬럼의 값은 14자리 숫자(YYYYMMDDhhmmss)로 이루어져 있어야 합니다.
- 기본적으로 데이터는 PARTITION을 통해 나누고, 데이터 분산을 더 시키기 위해서 KEY 값을 사용합니다.
- KEY 값이 지나치게 늘어나거나, PARTITION RANGE 가 지나치게 작을 경우 분산된 Backend 수가 많아져서 성능 저하를 야기할 수 있습니다.

3 IRIS 사용하기

본 챕터에서는 IRIS의 기본적인 사용방법에 대해서 설명합니다. IRIS CLI 사용법, 사용자 관리, 권한 관리, 테이블 관리, 데이터 관리, 세션 관리, 백업 관리 등에 대해서 설명합니다.

3.1 IPLUS

IPLUS 는 IRIS에서 제공하는 CLI 툴입니다. 사용 방법은 아래와 같습니다.

```
[iris@master ~]$ iplus [ID]@[HOST]:[PORT]
Password:[PASSWORD]
```

IRIS 설치 후 기본적으로 제공하는 계정은 test 계정입니다. 패스워드 역시 test 입니다. IRIS의 마스터 노드에서 접속을 시도한다면 다음과 같은 명령어가 됩니다.

```
[iris@master ~]$ iplus test
Password:test
Connecting to IRIS(test@127.0.0.1:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus>
```

기본적으로 HOST와 PORT 값을 입력하지 않으면 설정 값을 참고하여 마스터 노드의 IP, LISTENER의 포트로 접속하게 됩니다. 위와 같이 iplus> 콘솔이 실행되면 정상적인 상태입니다.

이 상태에서 간단하게 테이블 리스트를 출력해보도록 하겠습니다. 먼저 iplus에 접속합니다. 그 후에 .table list 라는 명령어를 사용하여 테이블 리스트를 조회합니다.

```
[iris@master ~]$ iplus test
Password:test
Connecting to IRIS(test@127.0.0.1:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .table list
Ret : +OK Success
 TABLE_NAME    SCOPE         RAM_EXP_TIME  DSK_EXP_TIME  KEY_STRING    ...
========================================================
========================================================
0 row in set
0.3649 sec

iplus>
```

현재는 테이블이 생성된게 없으므로 0개의 결과를 리턴합니다. .table list 와 같이 .으로 시작하는 명령어를 닷-커맨드라고 부릅니다. IRIS에서는 이러한 닷-커맨드를 다수 지원하고 있습니다. iplus에서 사용할 수 있는 명령을 보기 위해서는 .h 혹은 .help를 입력하면 됩니다.

```
iplus> .h
HELP
========================================================
.h / .help                      show this message
.v / .version                   show IRIS version
.i / .info                      show IRIS connection information
.i / .info                      show IRIS connection information
.q / .quit                      quit iplus console
.history                        print command(query) history
.sep <f sep> <r sep>            change seperator
                                <f sep> : field seperator
                                <r sep> : record seperator
.field_sep <f sep>              change field seperator
.record_sep <r sep>             change record seperator
.width <num> <num> ...          set column widths
.load <t> <k> <p> <c> <d>       load data
                                <t> : table
```

```
                                <k> : key
                                <p> : partition
                                <c> : control file path
                                <d> : data file path
.spool [filename]               output to file
                                if filename is empty, output to console
.table                          table command
                                more infomation with -h
.session                        session command
                                more infomation with -h
.error                          error command
                                more infomation with -h
.node                           node command
                                more infomation with -h
.sm                             system monitor command
                                more infomation with -h
.system                         system command
                                more infomation with -h
!<shell command>                execute shell command
@<sequence file>                run sequence file
I
iplus >
```

3.2 사용자 관리

본 장에서는 사용자 관리에 사용되는 몇 가지 명령어들을 소개합니다. 사용자 추가를 위해서는 관리자 계정이 필요합니다. IRIS에서 관리자 계정은 하나만 존재하며 'root' 라는 이름으로 존재합니다.

```
ID   : root
PASS : m6administrator
```

3.2.1 나는 누구인가?

IRIS에서는 현재 로그인한 계정을 확인하는 명령어를 제공합니다. 명령어는 다음과 같습니다.

```
iplus> .whoami
```

각각의 계정에서 로그인한 후에 .whoami 명령을 내린 결과입니다.

```
[iris@master ~]$ iplus test
Password: test
Connecting to IRIS(test@192.168.111.100:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .whoami
Ret : +OK test

0.0036 sec

iplus> .q
Goodbye
[iris@master ~]$ iplus root
Password: m6administrator
Connecting to IRIS(root@192.168.111.100:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .whoami
Ret : +OK root

0.0033 sec

iplus>
```

3.2.2 사용자 리스트

사용자 리스트를 출력하는 명령어는 다음과 같습니다.

```
iplus> .user list
```

root 계정에서만 실행 가능합니다. 다른 계정으로 실행할 경우 권한이 없다는 에러가 발생하게 됩니다.

```
[iris@master ~]$ iplus root
Password: m6administrator
Connecting to IRIS(root@192.168.111.100:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .user list
Ret : +OK Success

 HOST          USER          CREATE_PRIV   DROP_PRIV     ALTER_PRIV  GRANT_PRIV  ALL_PRIV
=========================================================
 *             root          0             0             0             0             1
 *             test          1             1             0             0             0
=========================================================
2 row in set
0.0845 sec

iplus>
```

3.2.3 사용자 추가

다른 사용자 명령어와 동일하게 root 권한으로만 가능합니다. 다음과 같은 포맷을 지켜서 실행합니다.

```
iplus> .user add [USER_ID]:[USER_PASS]@[HOST]
```

다음은 실제 사용자를 추가하는 모습입니다.

```
[iris@master ~]$ iplus root
Password:m6administrator
Connecting to IRIS(root@192.168.111.100:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .user list
Ret : +OK Success

 HOST          USER          CREATE_PRIV   DROP_PRIV     ALTER_PRIV  GRANT_PRIV  ALL_PRIV
=========================================================
 *             root          0             0             0             0             1
 *             test          1             1             0             0             0
=========================================================
2 row in set
0.0845 sec

iplus> .user add iris_test:iris_test@*
Ret : +OK ADDUSER iris_test@* has added

0.0720 sec

iplus> .user list
Ret : +OK Success

 HOST          USER          CREATE_PRIV   DROP_PRIV     ALTER_PRIV  GRANT_PRIV  ALL_PRIV
=========================================================
 *             root          0             0             0             0             1
 *             test          1             1             0             0             0
 *             iris_test     1             1             0             0             0
=========================================================
3 row in set
0.0395 sec

iplus   >
```

3.2.4 사용자 패스워드 변경

다른 사용자 명령어와 다르게 모든 권한에서 사용이 가능합니다. 단, root 계정이 아닌 경우 자기자신의 패스워드만 변경할 수 있습니다.

```
iplus> .user password [USER_ID]:[NEW_PASSWORD]@[HOST]
```

다음은 사용자 패스워드를 변경하는 모습입니다.

```
[iris@master ~]$ iplus test
Password: test
Connecting to IRIS(test@192.168.111.100:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .user password root:test2@*
Error : -ERR Access denied

After the error occured, session recreated.
iplus> .user password test:test2@*
Ret : +OK PASSWORD test:test2@* password(s) has modified

0.0954 sec

iplus>
```

첫번째 명령어에서 에러가 발생한 이유는 test 사용자가 root 사용자의 패스워드를 변경하려 했기 때문에 발생한 문제입니다. test 사용자는 test 사용자 계정의 패스워드만 변경가능한 것을 확인할 수 있습니다.

3.2.5 사용자 삭제

사용자 삭제는 root 계정에서만 실행 가능합니다. 계정 삭제를 위해서는 USER_ID, HOST 를 입력해야 합니다. 다음은 사용자 삭제 명령어 입니다.

```
iplus> .user del [USER_ID]@[HOST]
```

다음은 실제 사용자를 삭제하는 과정입니다.

```
[iris@master ~]$ iplus root
Password:
Connecting to IRIS(root@192.168.111.100:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .user list
Ret : +OK Success

 HOST          USER          CREATE_PRIV   DROP_PRIV     ALTER_PRIV  GRANT_PRIV  ALL_PRIV
=========================================================
 *             root          0             0             0             0             1
 *             test          1             1             0             0             0
 *             iris_test     1             1             0             0             0
=========================================================
3 row in set
0.0825 sec

iplus> .user del iris_test@*
Ret : +OK DELUSER iris_test@{ALL_REGISTERED_HOSTS} has deleted

0.0783 sec

iplus> .user list
Ret : +OK Success

 HOST          USER          CREATE_PRIV   DROP_PRIV     ALTER_PRIV  GRANT_PRIV  ALL_PRIV
=========================================================
 *             root          0             0             0             0             1
 *             test          1             1             0             0             0
=========================================================
2 row in set
0.0430 sec

iplus>
```

3.3 테이블 관리

본 장에서는 테이블을 관리하는 방법을 소개합니다.

3.3.1 테이블 리스트

현재 생성되어 있는 테이블 리스트를 출력합니다.

```
iplus> .table list
```

아래에서 그 결과를 볼 수 있습니다.

```
[iris@Master1 ~]$ iplus test
Password:
Connecting to IRIS(test@192.168.111.100:5050).
Connected to IRIS.
Enter ".help" for instructions
iplus> .table list
Ret : +OK Success

 TABLE_NAME    SCOPE         RAM_EXP_TIME  DSK_EXP_TIME  KEY_STRING    ...
=======================================================
=======================================================
0 row in set
0.2884 sec

iplus>
```

.table list는 해당 계정에서 생성한 테이블만 볼 수 있습니다. 만약 전체 테이블을 보기 위해서는 옵션 값을 줘야합니다. 옵션을 확인하기 위해서는 아래와 같이 -h 옵션을 사용하면 됩니다.

```
iplus> .table list -h
Ret : +OK Success

 HELP
=========================================================
 table list option help
     -a, --all : show all table list include system table
                 default is False
     -s, --sys : show system table list
                 default is False
=========================================================
5 row in set
0.1020 sec

iplus>
```

3.3.2 테이블 생성

IRIS에서 테이블을 생성하는 방법은 일반적인 데이터베이스에서 테이블을 생성하는 방법과 유사합니다. 쿼리를 이용해서 테이블을 생성하게 됩니다. 아래는 일반적인 데이터베이스에서 사용하는 테이블 생성 쿼리입니다.

```
CREATE TABLE [TABLE_NAME] (
    [COLUMN1_NAME] [COLUMN1_TYPE] [COLUMN OPTION],
    [COLUMN2_NAME] [COLUMN2_TYPE] [COLUMN OPTION],
    [COLUMN3_NAME] [COLUMN3_TYPE] [COLUMN OPTION],
    .
    .
    .
);
```

IRIS에서는 위의 쿼리에 추가 옵션이 들어가게 됩니다. 그럼 옵션을 설명하기에 앞서 간단한 테이블을 먼저 만들어봅니다. 아래는 그 예제입니다.

```
CREATE TABLE LOCAL_TEST_TABLE (
    K TEXT,
    P TEXT,
    A TEXT
)
datascope       LOCAL
ramexpire       30
diskexpire      34200
partitionkey        K
partitiondate       P
partitionrange  10
;
```

기본 테이블 생성 쿼리뒤에 값이 옵션 값이 추가 되어 있는 것을 알 수 있습니다. 각각의 옵션값은 다음과 같은 의미를 가집니다.

- datascope : 데이터 저장 방식 설정 [LOCAL OR GLOBAL]
- ramexpire : 램에 저장하는 시간 [GLOBAL 테이블일 경우 0]
- diskexpire : 디스크에 저장하는 시간 [GLOBAL 테이블 경우 0]
- partitionkey : KEY 로 사용할 값이 저장되는 컬럼 [GLOBAL 테이블 경우 None]
- partitiondate : PARTITION으로 사용할 값이 저장되는 컬럼 [GLOBAL 테이블 경우 None]
- partitionragne : 하나의 PARTITION 이 가지는 범이 [GLOBAL 테이블 경우 0]

옵션값은 테이블의 저장 방식이 GLOBAL, LOCAL에 따라 달라집니다.

3.3.2.1 datascope 데이터 저장 방식을 설정하는 부분입니다. GLOBAL, LOCAL 두가지 타입이 있습니다.

3.3.2.2 ramexpire 램에 저장하는 시간을 설정하는 부분입니다. 값은 분 단위로 설정할 수 있습니다. GLOBAL 테이블에서는 영향을 끼치지 않는 값입니다.

3.3.2.3 diskexpire 디스크에 저장되는 시간을 설정하는 부분입니다. 값은 분 단위로 설정할 수 있습니다. GLOBAL 테이블에서느 영향을 끼치지 않는 값입니다.

3.3.2.4 partitionkey KEY로 설정될 값을 가진 컬럼의 이름을 설정하는 부분입니다. 위의 테이블에서는 K의 값에 따라 KEY 값이 정해집니다.

3.3.2.5 partitiondate PARTITION으로 설정될 값을 가진 컬럼의 이름을 설정하는 부분입니다. 이 때 데이터는 14자리의 숫자로 이루어진 값이어야 합니다. 위의 테이블에서는 P 컬럼의 값에 따라 PARTITION 값이 정해집니다.

3.3.2.6 partitionrange PARTITION을 몇분 단위로 나누는지 결정하는 옵션입니다. 분 단위로 설정가능하며 위의 테이블에서는 10분 단위로 PARTITION을 나누어 사용합니다.

3.3.3 테이블 정보 확인

위의 테이블 리스트 조회 방법에서 테이블 옵션 정보는 부가적으로 확인할 수 있는 방법입니다. 하지만 스키마 정보를 확인하기 위해서는 다음과 같은 명령어를 사용해야 합니다.

```
iplus> .table schema [TABLE_NAME]
```

위의 명령어를 사용하여 테이블 생성에서 사용한 LOCAL_TEST_TABLE을 확인하면 아래와 같습니다.

```
iplus> .table schema LOCAL_TEST_TABLE
Ret : +OK Success

 SQL_SCRIPT
=========================================================
 CREATE TABLE LOCAL_TEST_TABLE ( K TEXT , P TEXT , A TEXT );
=========================================================
1 row in set
0.0607 sec

iplus>
```

3.3.4 테이블 삭제

테이블 삭제는 IRIS가 아닌 다른 SQL을 사용하는 데이터베이스와 동일합니다.

```
iplus> drop table [TABLE_NAME]
```

실제 실행을 하면 다음과 같습니다.

```
iplus> .table list
Ret : +OK Success

 TABLE_NAME    SCOPE         RAM_EXP_TIME  DSK_EXP_TIME  KEY_STRING    ...
=========================================================
 LOCAL_TEST_TABLE LOCAL         30            34200         K             P             10
=========================================================
1 row in set
0.2839 sec

iplus> drop table LOCAL_TEST_TABLE;
```

```
Ret : +OK Drop Table

0.4401 sec

iplus> .table list
Ret : +OK Success

 TABLE_NAME    SCOPE         RAM_EXP_TIME  DSK_EXP_TIME  KEY_STRING    ...
=========================================================
=========================================================
0 row in set
0.2821 sec

iplus>
```

3.3.5 테이블 삭제와 데이터 삭제

IRIS에서 테이블 삭제시 가장 많이 들어오는 문의는 다음과 같습니다.

```
테이블을 삭제했는데 디스크 사용량이 줄어들지 않습니다.
```

위와 같은 문의 사항은 IRIS에서 테이블 삭제 후 데이터 삭제 방식이 다음과 같기 때문입니다.

```
1. 테이블 삭제
2. 데이터 위치 메타 데이터에 삭제 표기
3. 다른 데몬에서 이를 감지하고 데이터 삭제 시작
```

즉, 테이블 삭제 직후에는 데이터가 남아있습니다. 일정 시간 이후에 천천히 삭제가 되니 디스크 사용량이 급격히 줄어들지는 않습니다.

3.4 데이터 관리

IRIS는 데이터를 관리함에 있어서 Backend 기준으로 관리가 됩니다. Backend는 테이블, KEY, PARTITION으로 구성되는 데이터를 저장하는 최소 단위 입니다. 본 항목에서는 Backend를 관리하기 위한 명령어, Backend를 관리하는 PR / PL의 동작 옵션에 대해서 설명합니다.

3.4.1 DROP BACKEND

IRIS에서는 Backend의 생성은 데이터 입력시에 자동으로 이루어집니다. 삭제 역시 delete 쿼리를 사용해 삭제가 가능하지만 특정 테이블, KEY, PARTITION의 데이터를 제거하기 위해서 drop backend 쿼리를 제공하고 있습니다.

```
iplus> drop backend [TABLE_NAME];
```

위의 쿼리는 테이블의 스키마만 남기고 모든 데이터를 삭제합니다.

```
iplus> drop backend [TABLE_NAME] ( KEY = '[KEY_VALUE]' );
iplus> drop backend [TABLE_NAME] ( KEY in ('[KEY_VALUE]', '[KEY_VALUE]', ... ));
```

위의 쿼리는 특정 테이블의 KEY 하나 혹은 하나 이상에 해당하는 데이터를 삭제합니다.

```
iplus> drop backend [TABLE_NAME] ( PARTITION = '20130101000000' );
iplus> drop backend [TABLE_NAME] ( PARTITION < '20130101000000');
```

위의 쿼리는 특정 테이블에서 PARTITION 하나 혹은 범위에 해당하는 데이터를 삭제합니다.

3.4.2 PL의 동작 방식

PL은 각각의 슬레이브 노드에서 동작하는 Backend의 이동을 관리하는 데몬입니다. 테이블 생성시에 입력한 ramexpire, ssdexpire 를 기준으로 Backend 파일을 램에서 SSD 혹은 디스크로, SSD에서 디스크로 이동을 관리합니다.

3.4.2.1 이동 방법 PL 동작 중 램에서 디스크로 이동하는 동작은 다음과 같은 룰로 동작합니다.

동작 옵션	동작 방식
OFF	이동하지 않음
LOCAL_TIME_BASE	슬레이브 시스템 시간을 참조하여 해당 시간이 지난 파일을 디스크로 이동
PARTITION_BASE	가장 최근에 들어온 데이터의 파티션 시간을 참조하여 해당 시간이 지난 파일을 이동

위의 방식을 제공하고 있습니다. 기본 값은 PARTITION_BASE로 되어 있습니다.

3.4.3 PR의 동작 방식

PR은 각각의 슬레이브 노드에서 동작하는 Backend의 삭제, 파일 복구, 기타 파일을 관리하는 데몬입니다. 그 중에서 Backend 삭제의 경우 diskexpire를 기준으로 디스크에서 삭제를 수행합니다.

3.4.3.1 Backend 삭제 PR 동작 중에 디스크에서 삭제를 수행하는 동작은 다음과 같은 룰로 동작합니다.

동작 옵션	동작 방식
OFF	삭제 하지 않음

동작 옵션	동작 방식
LOCAL_TIME_BASE	슬레이브 시스템 시간을 참조하여 해당 시간이 지난 파일을 삭제
PARTITION_BASE	가장 최근에 들어온 데이터의 파티션 시간을 참조하여 해당 시간이 지난 파일을 삭제
RECYCLEBIN	사용자 명령을 받기 전까지는 삭제하지 않습니다. 휴지통 항목 참조

위의 방식을 제공하고 있습니다. 기본 값은 LOCAL_TIME_BASE로 되어 있습니다.

3.4.4 PL/PR의 룰

3.4.4.1 OFF 동작 옵션이 OFF로 되어 있는 경우 PL / PR은 아무런 동작도 하지 않습니다.

3.4.4.2 LOCAL_TIME_BASE 동작 옵션이 LOCAL_TIME_BASE인 경우 다음과 같이 동작합니다.

```
ram expire time : 30 min
dst file name : TEST_TABLE_K1_20130101000000.DAT
dst file's info :
        table name : TEST_TABLE
        key : K1
        partition : 20130101000000
해당 파일이 있는 슬레이브 노드의 시스템 시간이 20130101003000를 지나면
파일은 디스크로 이동
```

3.4.4.3 PARTITION_BASE 동작 옵션이 PARTITION_BASE 인 경우 다음과 같이 동작합니다.

```
ram expire time : 30 min
dst file name : TEST_TABLE_K1_20130101000000.DAT
dst file's info :
        table name : TEST_TABLE
        key : K1
        partition : 20130101000000
해당 파일이 있는 슬레이브 노드에 파티션 값이 20130101000000 인 파일이 생성되면
파일은 디스크로 이동 [ TEST_TABLE_K1_20130101003000.DAT 파일 생성시 ]
```

3.4.4.4 RECYCLEBIN 동작 옵션이 RECYCLEBIN 인 경우 사용자가 purge 명령을 내리지 않을 경우 OFF와 동일하게 동작합니다. 자세한 내용은 휴지통 항목을 참조하시면 됩니다.

3.4.5 PL / PR 동작 옵션 관련 명령어

본 항목에서는 PL / PR 동작 옵션을 제어하기 위해 IRIS에서 제공하는 명령어를 설명합니다. 다음과 같이 실행하면 도움말을 확인할 수 있습니다.

```
iplus> .pm
Ret : +OK Success

 HELP
=======================================================
 pm command help
     list         : show pm option configuration
                    ex) .pm list
     default      : change default pm configuration
                    you can see more help ex) .pm default -h
                    ex) .pm default [ram|disk] [ram_option|disk_option]
     ram          : change pm ram option configuration
                    you can see more help ex) .pm ram -h
                    ex) .pm ram [table_name] [ram_option]
     disk         : change pm disk option configuration
                    you can see more help ex) .pm disk -h
```

```
                    ex) .pm disk [table_name] [disk_option]
     del           : remove table pm option
                     ex) .pm del [table_name]
     [table_name] : initializing pm conf each table
                     you can see more help ex) .pm [table_name] -h
                     ex) .pm [table_name] [ram_option] [disk_option]
=========================================================
17 row in set
0.2214 sec
```

3.4.5.1 PM LIST PL / PR 옵션값의 리스트를 출력합니다.

```
iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=========================================================
 $DEFAULT      PARTITION_BASE LOCAL_TIME_BASE
=========================================================
1 row in set
0.2495 sec

iplus>
```

아무것도 설정하지 않은 상태라면 다음과 같이 $DEFAULT 항목만 보이게 됩니다. 이 $DEFAULT 항목은 따로 설정하지 않은 테이블을 위한 PR / PR 옵션입니다. 즉, 기본 상태의 테이블은 RAM은 PARTITION_BASE로 DISK는 LOCAL_TIME_BASE로 동작하게 됩니다.

3.4.5.2 PM DEFAULT PM의 $DEFAULT, 즉 기본값을 변경하는 명령어로 아래와 같은 형태를 지닙니다.

```
iplus> .pm default [ram|disk] [ram_option|disk_option]
```

예를 들어 램 옵션을 OFF로 바꾸는 경우에는 다음과 같이 실행할 수 있습니다.

```
iplus> .pm default ram OFF
Ret : +OK

0.3987 sec

iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=========================================================
 $DEFAULT      OFF           LOCAL_TIME_BASE
=========================================================
1 row in set
0.2063 sec

iplus>
```

3.4.5.3 PM [TABLE_NAME] 이 명령어는 특정 테이블이 처음으로 독자적인 PL / PR OPTION을 가지기 위해 실행되어야 하는 명령어 입니다. 예를 들어 "LOCAL_TEST_TABLE"의 PL / PR OPTION 을 모두 OFF 하고자 할 때 처음 실행되는 명령어 입니다. 이 명령어가 실행 된 이후에는 다른 명령어로 개별적 옵션을 추가로 변경할 수 있습니다.

```
iplus> .pm [TABLE_NAME] [RAM_OPTION] [DISK_OPTION]
```

예를 들어 LOCAL_TEST_TABLE 이라는 테이블에 램은 PARTITION_BASE, 디스크는 OFF로 설정하고자 한다면 다음과 같이 수행합니다.

```
iplus> .pm LOCAL_TEST_TABLE PARTITION_BASE OFF
Ret : +OK
```

```
0.4363 sec

iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=======================================================
 $DEFAULT      OFF           LOCAL_TIME_BASE
 LOCAL_TEST_TABLE PARTITION_BASE OFF
=======================================================
2 row in set
0.2107 sec

iplus>
```

한 번 이렇게 실행된 이후에는 램, 디스크에 대한 옵션을 바꾸기 위해서는 RAM, DISK 명령어를 사용하면 됩니다. 자세한 내용은 PM RAM, PM DISK 항목을 참조하시면 됩니다.

3.4.5.4 PM RAM 특정 테이블의 RAM 동작 옵션을 설정하는 명령어 입니다.

```
iplus> .pm ram [TABLE_NAME] [RAM_OPTION]
```

위의 명령어가 실행되기 이전에 반드시 PM [TABLE_NAME]이 실행되어야 합니다. 아래는 실제 실행 결과 입니다.

```
iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=======================================================
 $DEFAULT      OFF           LOCAL_TIME_BASE
```

```
 LOCAL_TEST_TABLE PARTITION_BASE OFF
=====================================================
2 row in set
0.2107 sec

iplus> .pm ram LOCAL_TEST_TABLE LOCAL_TIME_BASE
Ret : +OK

0.4042 sec

iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=====================================================
 $DEFAULT      OFF           LOCAL_TIME_BASE
 LOCAL_TEST_TABLE LOCAL_TIME_BASE OFF
=====================================================
2 row in set
0.2059 sec

iplus>
```

3.4.5.5 PM DISK 특정 테이블의 DISK 동작 옵션을 설정하는 명령어입니다.

```
iplus> .pm disk [TABLE_NAME] [DISK_OPTION]
```

위의 명령어가 실행되기 이전에 반드시 PM [TABLE_NAME]이 실행되어야 합니다. 아래는 실제 실행 결과 입니다.

```
iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=====================================================
```

```
 $DEFAULT      OFF           LOCAL_TIME_BASE
 LOCAL_TEST_TABLE PARTITION_BASE OFF
=========================================================
2 row in set
0.3107 sec

iplus> .pm disk LOCAL_TEST_TABLE LOCAL_TIME_BASE
Ret : +OK

0.2042 sec

iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=========================================================
 $DEFAULT      OFF           LOCAL_TIME_BASE
 LOCAL_TEST_TABLE LOCAL_TIME_BASE LOCAL_TIME_BASE
=========================================================
2 row in set
0.3059 sec

iplus>
```

3.4.5.6 PM DEL 특정 테이블의 옵션을 삭제하고 기본 설정으로 돌아가게 하기 위한 명령어 입니다. 이 명령어를 수행하게 되면 해당 테이블의 RAM 동작 옵션, DISK 동작 옵션은 모두 삭제되고 기본값으로 동작하게 됩니다.

```
iplus> .pm del [TABLE_NAME]
```

아래는 실제 적용 결과 입니다.

```
iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
```

```
=======================================================
 $DEFAULT      OFF           LOCAL_TIME_BASE
 LOCAL_TEST_TABLE LOCAL_TIME_BASE OFF
=======================================================
2 row in set
0.2059 sec

iplus> .pm del LOCAL_TEST_TABLE
Ret : +OK

0.2535 sec

iplus> .pm list
Ret : +OK Success

 TABLE_NAME    RAM_OPTION    DISK_OPTION
=======================================================
 $DEFAULT      OFF           LOCAL_TIME_BASE
=======================================================
1 row in set
0.2064 sec

iplus>
```

3.5 세션

IRIS에서 세션은 하나의 쿼리가 하나의 세션으로 정의됩니다. 즉, 하나의 커넥션을 생성하여 쿼리를 다수개 실행하였다면 하나의 세션이 아니라 다수개의 세션이 생성됩니다. 본 장에서는 이러한 세션을 관리하는 방법에 대해서 설명을 드립니다.

3.5.1 세션 명령어

IRIS에서는 세션을 관리하기 위한 명령어를 제공하고 있습니다.

```
iplus> .session
Ret : +OK Success

 HELP
=======================================================
 session command help
     list : show session list
            ex) .session list [options]
            you needs option help, .session list -h
     info : show session detail info
            ex) .session info [session id]
     term : term targeted session
            ex) .session term [session id]
=======================================================
8 row in set
0.0438 sec

iplus>
```

3.5.2 세션 리스트

다음 명령어는 세션 리스트를 출력하는 명령어입니다.

```
iplus> .session list
Ret : +OK Success

 SID           QUERY_STRING  START_TIME    END_TIME      TYPE          NODE
PID           RESULT
=======================================================
 20150615003238_1_18861_11 ls            20150615003238 20150615003239 ABN          ...
 20150615003243_1_16519_0 session       20150615003243 20150615003243 END          ...
 20150615003249_1_16519_1 session       20150615003249 20150615003249 END          ...
 20150615003255_1_16562_0 CREATE        20150615003255 20150615003256 ABN          ...
 20150615003257_1_16569_0 CREATE        20150615003257 20150615003257 END          ...
 20150615003259_1_16575_1 INSERT        20150615003259 20150615003259 END          ...
 20150615003259_1_16575_2 INSERT        20150615003259 20150615003259 END          ...
 20150615003259_1_16575_3 INSERT        20150615003259 20150615003259 END          ...
```

```
 20150615003259_1_16575_0 INSERT        20150615003259 20150615003300 ABN          ...
 20150615003259_1_16575_4 INSERT        20150615003259 20150615003300 END          ...
=========================================================
10 row in set
0.0642 sec

iplus>
```

세션 리스트는 다음과 같은 정보를 제공합니다.

- Session ID : 세션에 할당된 고유 ID 입니다.
- Query Type : 해당 세션에 실행된 쿼리 타입입니다.
- Start Time : 쿼리의 시작 시간입니다.
- End Time : 쿼리의 종료 시간입니다.
- Type : 현재 상태를 의미합니다. [START | END | ABN] 으로 상태를 구분합니다.
- Node : 쿼리 실행이 어떤 노드에서 실행되었는지에 관한 정보입니다.
- Process ID : 쿼리 실행을 담당하는 프로세스의 ID 값입니다.
- Result : 쿼리 실행 결과입니다.

기본적으로 세션 리스트는 최근 10개의 값을 출력하도록 설정되어 있습니다. 해당 값 보다 더 많은 값들을 출력해서 보고 싶다면 옵션을 사용하면 됩니다. 옵션 값은 아래 HELP 화면을 통해서 확인할 수 있습니다.

```
iplus> .session list -h
Ret : +OK Success

 HELP
=========================================================
 session list option help
     -s, --stime : start time 14 digit
                   default is onehour before time
```

```
                    ex) 20140101000000
     -e, --etime : end time 14 digit
                    default is 99999999999999
                    ex) 20140101000000
     -t, --type  : session type
                    [ START | END | ABN ]
     -h, --help  : show this message
=========================================================
10 row in set
0.0429 sec

iplus>
```

3.5.3 세션 정보

세션 리스트를 보면서 세션 세부 정보를 확인할 때 사용하는 명령어 입니다.

```
.session info [session id]
```

세션 리스트를 통해서 세션 ID 값을 구한 후 위의 명령어를 사용하면 자세한 정보를 얻을 수 있습니다.

```
iplus> .session list
Ret : +OK Success

 SID            QUERY_STRING  START_TIME    END_TIME      TYPE          NODE
PID           RESULT
=========================================================
 20150615003259_1_16575_2 INSERT        20150615003259 20150615003259 END           ...
=========================================================
10 row in set
0.0884 sec

iplus> .session info 20150615003259_1_16575_1
Ret : +OK Success
```

```
 SID            QUERY_STRING  START_TIME    END_TIME      TYPE          NODE
PID           RESULT
=======================================================
 20150615003259_1_16575_1 INSERT INTO LOCAL_TEST_TABLE (k, p, a) VALUES ...
=======================================================
1 row in set
0.1234 sec

iplus>
```

세션 정보에서 출력하는 정보는 세션 리스트에서 출력하는 정보와 유사합니다. 단, Query Type 대신에 쿼리문 전체를, RESULT 부분에서는 결과 전체를 출력합니다.

3.5.4 세션 종료

세션 ID기반으로 현재 동작중인 세션을 종료시키는 기능입니다. 사용법은 아래와 같습니다.

```
.session term [session id]
```

아래는 실제 테스트 상황입니다. 우선 .session list 로 종료시킬 세션을 찾습니다. 그 후에 찾은 세션 ID를 이용하여 세션을 종료시킵니다.

```
iplus> .session list --type START
Ret : +OK Success

 SID            QUERY_STRING  START_TIME    END_TIME      STYPE         NODE
PID           RESULT
=======================================================
 20150615043250_2_29137_0 session       20150615043250 20150615043250 START         ...
=======================================================
1 row in set
```

```
0.0955 sec

iplus> .session term 20150615043250_2_29137_0
Ret : +OK kill 20150615043250_2_29137_0 session.

0.2268 sec

iplus>
```

3.6 휴지통

IRIS에서는 데이터 관리에 필요한 휴지통 기능을 제공하고 있습니다. 휴지통 기능은 기본적으로 꺼진 상태이며 테이블 별로 휴지통 기능을 켤 수 있습니다.

```
* 휴지통 기능 주의 사항
휴지통 기능은 Replication 개수의 2배 이상의 노드에서 실행하기를 권장합니다.
IRIS 삭제 로직의 경우 바로 삭제를 수행하지 않기 때문에,
데이터 삭제 직후, 데이터 로딩시 에러가 발생할 수 있습니다.
```

3.6.1 휴지통 관리 명령어

휴지통 관리 명령어는 테이블 별로 현재 휴지통 기능이 사용중인지 확인할 수 있는 LIST, 휴지통 기능을 켤 수 있는 ADD, 휴지통 기능을 끌 수 있는 OFF, 현재 전체 휴지통 상태를 확인하는 STATUS 명령어를 제공합니다. 해당 명령어는 아래 명령을 입력하여 자세한 내용을 확인할 수 있습니다.

```
iplus> .recyclebin
Ret : +OK Success

 HELP
=======================================================
 recyclebin command help
     add  : set recyclebin option to table
```

```
     list : print table list which set recyclebin
     del  : unset recyclebin option from table
=========================================================
4 row in set
0.0973 sec

iplus>
```

3.6.1.1 LIST 현재 휴지통 기능이 켜져 있는 테이블 리스트를 출력합니다.

```
iplus> .recyclebin list
Ret : +OK Success

 TABLE_NAME
=========================================================
=========================================================
0 row in set
0.0545 sec

iplus>
```

3.6.1.2 ADD 선택한 테이블의 휴지통 기능을 사용하도록 설정합니다.

```
iplus> .recyclebin add LOCAL_TEST_TABLE_NOT_EXIST
Error : -ERR table [LOCAL_TEST_TABLE_NOT_EXIST] does not exist.

After the error occured, session recreated.

iplus> .recyclebin add LOCAL_TEST_TABLE
Ret : +OK

0.1381 sec

iplus> .recyclebin list
Ret : +OK Success
```

```
 TABLE_NAME
=======================================================
 LOCAL_TEST_TABLE
=======================================================
1 row in set
0.0415 sec

iplus>
```

현재 생성되지 않은 테이블에 대해서 추가 명령을 내릴 경우 에러 메시지를 출력합니다. 반드시 테이블 생성 후에 실행하여야합니다.

3.6.1.3 DEL

선택한 테이블의 휴지통 기능을 사용하지 않도록 설정합니다.

```
iplus> .recyclebin del LOCAL_TEST_TABLE
Ret : +OK

0.0902 sec

iplus> .recyclebin list
Ret : +OK Success

 TABLE_NAME
=======================================================
=======================================================
0 row in set
0.0385 sec

iplus>
```

휴지통 기능을 끄게 된다고 해서 데이터가 삭제 되지는 않습니다.

3.6.2 휴지통 기능 사용하기

휴지통 기능 관리를 위한 명령어는 위에서 설명했습니다. 휴지통 기능을 사용하게 되면 기존 명령어와 다르게 동작하는 명령어가 몇가지 있습니다.

명령어	휴지통 기능을 사용하지 않는 경우	휴지통 기능을 사용하는 경우
drop table	테이블을 삭제	테이블을 삭제 하지 않고 휴지통에 기록
drop backend	해당 backend를 삭제	해당 backend를 삭제하지 않고 휴지통에 기록

또한 추가로 사용 가능한 명령어가 있습니다.

명령어	동작 내용
flashback table	해당 테이블을 복원
flashback backend	해당 backend를 복원

3.6.2.1 TABLE

3.6.2.1.1 DROP TABLE 테이블을 삭제하기 위해서 drop table 명령어를 사용합니다. 아래는 drop table을 휴지통 기능을 사용하는 테이블을 대상으로 실행한 결과 입니다.

```
iplus> .table list
Ret : +OK Success

 TABLE_NAME    SCOPE         RAM_EXP_TIME  DSK_EXP_TIME  KEY_STRING    PARTITION_STRING PARTITION_RANGE
=======================================================
 LOCAL_TEST_TABLE LOCAL         30            34200         k             p              10
=======================================================
1 row in set
0.2878 sec

iplus> drop table LOCAL_TEST_TABLE;
Ret : +OK Drop Table
```

```
0.1994 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=======================================================
 $DT1434436184.9897079 LOCAL_TEST_TABLE 2015/06/16 06:29:45 test          test
=======================================================
1 row in set
0.0413 sec

iplus> .table list
Ret : +OK Success

 TABLE_NAME    SCOPE         RAM_EXP_TIME  DSK_EXP_TIME  KEY_STRING    PARTITION_STRING PARTITION_RANGE
=======================================================
=======================================================
0 row in set
0.2805 sec

iplus>
```

먼저 drop table 실행 후 휴지통에 정상적으로 들어가 있는 것을 확인할 수 있습니다. 휴지통에서 보여주는 값을 먼저 보도록하겠습니다.

정보 명칭	설명
JOB_ID	삭제 작업에 설정된 ID 값입니다. 각각의 값은 고유한 값을 가지고 있습니다.이 값을 이용하여 복원이 가능합니다.
TABLE_NAME	삭제 작업이 수행된 테이블 이름입니다.
DATE	실행된 시간입니다.
QUERIED_BY	삭제 작업을 수행한 사용자 이름입니다.
TABLE_OWNER	테이블의 실제 소유자 이름입니다.

drop table에 의해서 휴지통에 들어가 있는 테이블은 .table list 에서 더 이상 보이지 않습니다. 물론 단순히 테이블 리스트에서 빠진게 아니라 동일한

이름으로 테이블을 생성해도 생성이 됩니다.

3.6.2.1.2 PURGE TABLE 휴지통에 들어가 있는 테이블의 완전 삭제는 다음과 같이 이루어집니다.

```
iplus> purge table LOCAL_TEST_TABLE;
Ret : +OK Purge Table

0.1605 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=========================================================
=========================================================
0 row in set
0.0415 sec

iplus>
```

위에서 사용한 명령어는 purge table 입니다. 휴지통에 들어가 있는 테이블을 완전히 지우는 작업을 수행합니다. purge table의 사용법은 다음과 같습니다.

```
iplus> purge table [TABLE_NAME];
```

또는

```
iplus> purge table [JOB_ID];
```

예제에서는 테이블의 이름을 사용하여 삭제를 수행하였지만 휴지통 상태 정보에서 볼 수 있는 JOB ID로도 삭제가 가능합니다.

3.6.2.1.3 DROP TABLE WITH PURGE 만약 drop table [TABLE_NAME] 을 통해서 삭제를 할 때 바로 삭제를 해야 하는 경우에는 아래와 같은 명령을 사용합니다.

```
iplus> drop table [TABLE_NAME] purge;
```

drop table 시에 뒤에 purge 옵션을 주게 되면 휴지통을 거치지 않고 바로 삭제 됩니다.

3.6.2.1.4 FLASHBACK TABLE 휴지통에 들어가 있는 테이블을 다시 복구 하기 위해서 flashback table 명령어를 사용하면 됩니다.

```
iplus> flashback table [TABLE_NAME];
```

또는

```
iplus> flashback table [JOB_ID];
```

를 사용하면 복구가 가능합니다. 아래는 LOCAL_TEST_TABLE을 생성하고 휴지통 기능을 켜고 테이블을 삭제한 후 복원하는 예제입니다.

```
iplus> .recyclebin list
Ret : +OK Success

 TABLE_NAME
=========================================================
 LOCAL_TEST_TABLE
=========================================================
1 row in set
0.0406 sec
```

```
iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k6            20110616000000 None
 k7            20110616000000
 k4            20110616000000 0
 k5            20110616000000 0.1
 k3            20110616000000 1.2
 k2            20110616000000 1
=======================================================
6 row in set
0.0526 sec
```

휴지통 기능은 켜져 있는 상태고 데이터는 6개가 들어가 있는 상태입니다.

```
iplus> drop table LOCAL_TEST_TABLE;
Ret : +OK Drop Table

0.4469 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=========================================================
 $DT1434440305.1507170 LOCAL_TEST_TABLE 2015/06/16 07:38:25 test          test
=========================================================
1 row in set
0.0432 sec

iplus> select * from LOCAL_TEST_TABLE;
Error : -ERR Table does not exists. [select * from LOCAL_TEST_TABLE;]

After the error occured, session recreated.
```

테이블을 삭제하고 휴지통에 들어간 것을 확인한 후에 조회를 시도합니다. 조회 시도시 없는 테이블이라고 나오며 에러가 납니다.

```
iplus> flashback table LOCAL_TEST_TABLE;
Ret : +OK Flashback Table

0.2861 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k6            20110616000000 None
 k7            20110616000000
 k4            20110616000000 0
 k5            20110616000000 0.1
 k3            20110616000000 1.2
 k2            20110616000000 1
=======================================================
6 row in set
0.0530 sec

iplus>
```

복원 명령 수행 후에 다시 한번 조회 쿼리를 시도하면 기존과 동일한 데이터가 나옴을 알 수 있습니다.

다음은 조금 다른 상황입니다.

```
테이블에 데이터 입력 => 테이블 삭제 (휴지통으로 이동) =>
동일한 이름의 테이블을 생성 => 데이터 입력 => 복원
```

위와 같은 상황을 테스트 하기 위해서 테이블 하나를 만들고 데이터를 입력합니다.

```
iplus> .recyclebin list
Ret : +OK Success
```

```
 TABLE_NAME
=======================================================
 LOCAL_TEST_TABLE
=======================================================
1 row in set
0.0386 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k6            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k4            20110616000000 BEFORE
 k5            20110616000000 BEFORE
 k3            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
6 row in set
0.0580 sec
```

먼저 휴지통에 등록된 것을 확인하고 내부의 데이터를 확인합니다. 데이터가 있음을 확인합니다.

```
iplus> drop table LOCAL_TEST_TABLE;
Ret : +OK Drop Table

0.2073 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=======================================================
 $DT1434440816.6835511 LOCAL_TEST_TABLE 2015/06/16 07:46:56 test          test
=======================================================
```

```
1 row in set
0.0437 sec
```

성공적으로 테이블 삭제가 이루어졌습니다. 휴지통에 들어간 것 역시 확인 가능합니다.

```
iplus> CREATE TABLE LOCAL_TEST_TABLE (
     >    k         TEXT,
     >    p         TEXT,
     >    a         TEXT
     > )
     > datascope       LOCAL
     > ramexpire       30
     > diskexpire      34200
     > partitionkey    k
     > partitiondate   p
     > partitionrange  10
     > ;
Ret : +OK Create table

0.2042 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=========================================================
=========================================================
0 row in set
0.0874 sec
```

동일한 테이블을 다시 생성합니다. 조회 결과 새로 만든 테이블이므로 데이터는 비워져 있습니다.

```
iplus> flashback table LOCAL_TEST_TABLE;
Error : -ERR table name (LOCAL_TEST_TABLE} already exists
```

```
After the error occured, session recreated.
iplus>
```

flashback table을 수행했지만 복원에 실패했습니다. 복원 실패 이유는 바로 동일한 이름의 테이블이 이미 존재하기 때문입니다. 이러한 상황에서 강제 복원을 위한 하나의 옵션을 지원합니다.

```
iplus> flashback table LOCAL_TEST_TABLE OVERWRITE;
Ret : +OK Flashback Table

0.3363 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P              A
=========================================================
 k6            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k4            20110616000000 BEFORE
 k5            20110616000000 BEFORE
 k3            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=========================================================
6 row in set
0.0524 sec

iplus>
```

뒤에 OVERWRITE 옵션을 사용하면 기존 테이블을 덮어서 복원합니다. 기존의 데이터는 삭제되니 주의해야 합니다.

3.6.2.2 BACKEND

3.6.2.2.1 DROP BACKEND DROP BACKEND 는 DROP TABLE 과 유사하게 동작합니다. 단, TABLE의 경우에는 전체 테이블을 휴지통으로 이동하는 경우인 반면에 BACKEND는 KEY, PARTITION 기준으로 일정한 부분만 휴지통으로 보내는 명령어 입니다. 다음과 같이 명령어를 입력할 수 있습니다.

```
iplus> drop backend [TABLE_NAME] ( [OPTION] );
```

아래는 예제입니다.

```
iplus> drop backend LOCAL_TEST_TABLE ( KEY = 'k2' );
    LOCAL_TEST_TABLE에서 KEY가 'k2'인 데이터를 휴지통으로 이동
iplus> drop backend LOCAL_TEST_TABLE ( PARTITION > '20140101000000' );
    LOCAL_TEST_TABLE에서 PARTITION 이 2014년 01월 01일 00시 00분 00초 보다 큰 데이터를 휴지통으로 이동
```

아래는 실제 수행 결과 입니다.

```
iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k6            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k4            20110616000000 BEFORE
 k5            20110616000000 BEFORE
 k3            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
6 row in set
0.0522 sec

iplus> drop backend LOCAL_TEST_TABLE ( KEY = 'k3' );
Ret : +OK SUCCESS
```

```
0.1210 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k5            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k6            20110616000000 BEFORE
 k4            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
5 row in set
0.0515 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=======================================================
 $DB1434442140.5975571 LOCAL_TEST_TABLE 2015/06/16 08:09:00 test          test
=======================================================
1 row in set
0.0438 sec

iplus>
```

LOCAL_TEST_TABLE 에서 KEY가 'k3' 인 것을 삭제 시도 했고 삭제 결과 6개의 결과에서 5개의 결과로 줄어든 것을 확인할 수 있습니다. 그와 더불어 휴지통으로 이동되었다는 것 역시 확인할 수 있습니다.

3.6.2.2.2 PURGE BACKEND 휴지통으로 이동된 BACKEND 완전 삭제를 위해서 purge backend 명령어를 사용합니다. purge table과 다른점은 purge backend는 오로지 JOB_ID로만 실행된다는 점입니다.

```
iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=======================================================
 $DB1434442140.5975571 LOCAL_TEST_TABLE 2015/06/16 08:09:00 test          test
=======================================================
1 row in set
0.0438 sec

iplus> purge backend $DB1434442140.5975571
Ret : +OK SUCCESS

0.1372 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=======================================================
=======================================================
0 row in set
0.0408 sec

iplus>
```

purge backend [JOB_ID]를 실행했고 그 결과 휴지통이 비워진 것을 확인할 수 있습니다.

3.6.2.2.3 DROP BACKEND WITH PURGE DROP TABLE과 동일하게 DROP BACKEND 도 옵션 값으로 PURGE를 받아 휴지통을 거치지 않고 삭제가 가능합니다.

```
iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success
```

```
 K             P             A
=======================================================
 k5            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k6            20110616000000 BEFORE
 k4            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
5 row in set
0.0746 sec

iplus> drop backend LOCAL_TEST_TABLE ( KEY = 'k4' ) PURGE;
Ret : +OK SUCCESS

0.0731 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k7            20110616000000 BEFORE
 k5            20110616000000 BEFORE
 k6            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
4 row in set
0.0488 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=======================================================
=======================================================
0 row in set
0.0403 sec

iplus>
```

PURGE 옵션을 주게 되면 휴지통을 거치지 않고 바로 삭제됨을 알 수 있습니다.

3.6.2.2.4 FLASHBACK BACKEND FLASHBACK TABLE과 유사하게 BACKEND에 작용하는 명령어 입니다. 단, PURGE BACKEND와 마찬가지로 [JOB_ID] 만으로 복원이 가능합니다.

```
iplus> flashback backend [JOB_ID]
```

다음은 실제 수행 결과 입니다.

```
iplus> select * from LOCAL_TEST_TABLE
Ret : +OK Success

 K             P             A
=======================================================
 k5            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k6            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
4 row in set
0.0507 sec

iplus> drop backend LOCAL_TEST_TABLE ( KEY = 'k6' );
Ret : +OK SUCCESS

0.1229 sec

iplus> .recyclebin status
Ret : +OK Success

 JOB_ID        TABLE_NAME    DATE          QUERIED_BY    TABLE_OWNER
=======================================================
 $DB1434442891.1222861 LOCAL_TEST_TABLE 2015/06/16 08:21:31 test          test
=======================================================
```

```
1 row in set
0.0416 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k5            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
3 row in set
0.0485 sec
```

KEY 값이 'k4' 인 데이터를 삭제했고 그 결과 휴지통으로 이동한 것을 확인할 수 있습니다.

```
iplus> flashback backend $DB1434442891.1222861;
Ret : +OK Flashback Backend

0.1144 sec

iplus> select * from LOCAL_TEST_TABLE;
Ret : +OK Success

 K             P             A
=======================================================
 k5            20110616000000 BEFORE
 k7            20110616000000 BEFORE
 k6            20110616000000 BEFORE
 k2            20110616000000 BEFORE
=======================================================
4 row in set
0.0479 sec

iplus>
```

flashback backend 명령어를 통하여 복원된 것을 확인할 수 있습니다.

3.6.2.2.5 FLASHBACK BACKEND OPTION FLASHBACK TABLE의 경우 OVERWRITE 라는 옵션이 있었습니다. FLASHBACK BACKEND 역시 유사한 옵션이 있습니다. 단, 차이가 있습니다. FLASHBACK TABLE의 경우 테이블을 통채로 덮어 쓰기 때문에 기존 데이터는 지워지지만 FLASHBACK BACKEND의 경우 우선 순위를 두어 두개의 데이터를 적절하게 섞어서 반영합니다.

```
iplus> flashback backend [JOB_ID] TABLEBASE;
```

위의 경우는 TABLEBASE라는 이름에서 알 수 있듯이 KEY, PARTITION 기준으로 테이블에 새로운 데이터가 있을 경우 휴지통의 데이터는 무시, 새로운 데이터가 없을 경우에는 휴지통의 데이터로 복원됩니다.

```
iplus> flashback backend [JOB_ID] RECYCLEBINBASE;
```

이 경우는 반대입니다. 휴지통에 있는 데이터를 위주로 복원을 하고 겹치는 데이터만 삭제합니다.

4 API

본 챕터에서는 IRIS 에 접속할 수 있는 API 에 대해 설명합니다.

IRIS 는 Java, Python, C/C++, C# 언어의 API 를 지원합니다.

4.1 JDBC

IRIS 는 Java JDBC 드라이버를 제공합니다. 이 드라이버를 사용해서 Java 프로그램으로 IRIS DB 에 접속할 수 있습니다. 현재 iBatis 적용가능 수준의 구현이 되어있으며, 추가 구현 진행중입니다.

Java 1.4, 1.6 두 버전을 지원합니다.

4.1.1 사용 방법

IRIS JDBC 드라이버파일(jar 파일) 을 받습니다. 실행할 Java 버전에 맞는 코어파일을 받아야 합니다.

IRIS JDBC 드라이버파일을 CLASSPATH 에 추가합니다.

IRIS JDBC 를 사용할 프로그램을 작성합니다. IRIS JDBC 를 import 해서 일반적인 JDBC 프로그래밍과 동일하게 프로그램을 작성합니다.

```
package com.mobigen.iris.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
~~~

String url = "jdbc:iris://127.0.0.1:5050";

try {
    Class.forName("com.mobigen.iris.jdbc.IRISDriver");
} catch (java.lang.ClassNotFoundException e) {
    e.printStackTrace();
    return;
}

Connection conn = null;
try {
    conn = DriverManager.getConnection(url, "test", "test");
    Statement stmt = conn.createStatement();
    String query = "INSERT INTO TEST_TABLE_1 (k, p, a) \
VALUES ('k2', '20110616000000', '1');"

    int ret = stmt.executeUpdate(query);
    stmt.close();
```

```
    conn.close();
} catch (SQLException e) {
    System.out.println("code : " + e.getErrorCode() + ", msg : " + e.getMessage());
} catch (java.io.UnsupportedEncodingException e) {
    System.out.println(e);
}
```

4.1.1.1 쿼리 실행 stmt.executeQuery(query) 함수를 실행해서 쿼리를 실행합니다. 쿼리 실행중 오류가 발생하면 Exception 을 발생시킵니다.

4.1.1.2 쿼리 실행 (검색) 실행은 위의 '쿼리 실행' 과 동일합니다. 검색결과데이터는 fetch 함수로 한번에 가져오거나, iteration 을 사용할 수 있습니다.

```
String query = "SELECT * FROM TEST_TABLE_1;";

Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(query);

for(int i=1; i<=rs.getRow(); i++)
{
    rs.next();
    String val = rs.getString(1) + " " + rs.getString(2) + " " + rs.getString(3);
    System.out.println("select returned '" + val + "'");
}
```

4.1.1.3 데이터 로딩 한번에 대량의 데이터를 파일에 저장한 후 한번에 INSERT 할 때 사용합니다. 이 기능을 사용하면 동일한 양의 데이터를 저장하는 경우 INSERT 쿼리에 비해 저장속도가 매우 빠릅니다. 로딩한 데이터는 같은 Backend 파일에 저장됩니다. (key, partition 값이 같습니다.)

- 데이터파일의 구분자(Separator)를 확인합니다.

  - Record, Field 구분자를 확인합니다. 문자열도 가능합니다.
  - csv 파일은 RecordSep : \n, FieldSep : , 입니다.

- 컬럼 위치정보를 저장한 ctl 파일을 만듭니다.
  - Record 구분자로 나눠서 컬럼이름을 나열합니다.
  - csv 파일이면 ctl 파일에는 한줄에 한개 컬럼씩 넣습니다.

- 데이터를 저장한 dat 파일을 만듭니다.

위에서 만든 ctl, dat 파일경로를 Load, LoadGlobal 함수에 인자로 넣어서 데이터를 로딩합니다.

```
String url = "jdbc:iris://127.0.0.1:5050";

String table = "LOCAL_TT";
String key = "c";
String partition = "20150312100000";
String control_file_path = "/home/iris/IRIS/test/loadtest/LOADTEST.ctl";
String data_file_path = "/home/iris/IRIS/test/loadtest/LOADTEST.dat";

conn = DriverManager.getConnection(url, "test", "test");
IRISStatement stmt = (IRISStatement)conn.createStatement();
stmt.SetFieldSep(",");
stmt.SetRecordSep("\n");

String result = stmt.Load(table, key, partition, control_file_path, data_file_path);
System.out.println( result );
```

4.2 Java

IRIS Java API 는 Query 실행, 데이터 로딩 기능을 지원합니다.

4.2.1 사용 방법

IRIS mammoth API 코어파일(jar 파일) 을 받습니다. 실행할 Java 버전에 맞는 코어파일을 받아야 합니다. 그리고 IRIS API 코어파일을 CLASSPATH 에 추가합니다.

IRIS API 를 사용할 프로그램을 작성합니다. IRIS API 를 import 하고 IRIS 접속정보를 이용해서 Connection 객체를 만든 후, 이를 이용해서 Cursor 객체를 만들어서 쿼리를 실행합니다.

```
package com.mobigen.iris.mammoth;
import com.mobigen.iris.mammoth.*;

~~~

Connect c = new Connect("localhost:5050", "test", "test");

Cursor cur = c.cursor();
cur.SetFieldSep(",");
cur.SetRecordSep("\n");

String query = "INSERT INTO TEST_TABLE_1 (k, p, a) VALUES ('k2', '20110616000000', '1');"
cur.Execute(query);

String message = cur.readline();

c.close();
```

4.2.1.1 쿼리 실행

cursor.Execute(query) 함수를 실행해서 쿼리를 실행합니다. 쿼리 실행중 오류가 발생하면 Exception 을 발생시킵니다.

4.2.1.2 쿼리 실행 (검색)

실행은 위의 '쿼리 실행' 과 동일합니다. 검색결과데이터는 fetch 함수로 한번에 가져오거나, iteration 을 사용할 수 있습니다.

```
cur.Execute("SELECT * FROM TEST_TABLE_1;")

Iterator<String[]> i = (Iterator<String[]>) cur.iterator();
while (i.hasNext() ) {
    String[] record = (String[]) i.next();

    for(int j=0; j<record.length;j++) {
        System.out.print(record[j] + ", ");
    }
    System.out.println();
}
```

4.2.1.3 데이터 로딩 한번에 대량의 데이터를 파일에 저장한 후 한번에 INSERT 할 때 사용합니다. 이 기능을 사용하면 동일한 양의 데이터를 저장하는 경우 INSERT 쿼리에 비해 저장속도가 매우 빠릅니다. 로딩한 데이터는 같은 Backend 파일에 저장됩니다. (key, partition 값이 같습니다.)

- 데이터파일의 구분자(Separator)를 확인합니다.
  - Record, Field 구분자를 확인합니다. 문자열도 가능합니다.
  - csv 파일은 RecordSep : \n, FieldSep : , 입니다.
- 컬럼 위치정보를 저장한 ctl 파일을 만듭니다.
  - Record 구분자로 나눠서 컬럼이름을 나열합니다.
  - csv 파일이면 ctl 파일에는 한줄에 한개 컬럼씩 넣습니다.
- 데이터를 저장한 dat 파일을 만듭니다.

위에서 만든 ctl, dat 파일경로를 Load 함수에 인자로 넣어서 데이터를 로딩합니다.

```
String table = "LOCAL_TT";
String key = "c";
String partition = "20150312100000";
String ctlPath = "/home/iris/IRIS/test/loadtest/LOADTEST.ctl";
String dataPath = "/home/iris/IRIS/test/loadtest/LOADTEST.dat";

Connect c = new Connect("localhost:5050", "test", "test");
Cursor cur = c.cursor();
cur.SetFieldSep(',')
cur.SetRecordSep('\n')

String result = cur.Load(table, key, partition, ctlPath, dataPath);
System.out.println( result );
```

4.2.2 API 상세

4.2.2.1 Create Connection IRIS Connection 객체를 생성합니다.

```
conn = Connection({ADDRESS}, {ID}, {PASSWD})
conn = Connection({ADDRESS}, {ID}, {PASSWD}, {Use-Direct})

conn = Connection('127.0.0.1:5050', 'test', 'test')
conn = Connection('127.0.0.1:5000', 'test', 'test', true)
```

- 주소는 ip:port 로 구성된 문자열입니다. ex) 192.168.0.1:5050
  - IRIS 기본 설치시 port : 5050 입니다.
- id, password 는 각각 IRIS DB 계정의 ID, PW 문자열 입니다.
- Direct 옵션 : 4번째 인자를 true 로 설정시 Direct 모드 Connection 을 생성합니다.
  - 일반 모드는 Connection 이 Master 노드를 거쳐서 Slave 노드로 이동합니다.

- Direct 옵션을 사용하면 Master 노드로부터 접속할 Slave 노드를 할당받은 후, Slave 노드로 바로 연결됩니다.
- API 사용 프로그램에서 Slave 노드로 바로 접속이 가능해야 사용할 수 있습니다.
- Direct 모드 사용시, ip 는 Master 노드의 IP 를 사용하며 port 를 변경해야 합니다.

  * IRIS 기본 설치시 port : 5000 입니다.

4.2.2.2 Connection Class 사용함수 목록

함수명	설명
Cursor cursor()	Cursor 객체를 생성합니다.
void close()	Connection 객체를 닫습니다.

4.2.2.3 Cursor Class 사용함수 목록

함수명	설명
void SetTimeout(int sec)	소켓 Timeout 을 설정합니다.
boolean SetRecordSep(String sep)	row 구분자를 설정합니다.
boolean SetFieldSep(String sep)	column 구분자를 설정합니다.
void Execute(String sql)	쿼리를 실행하고 메세지를 리턴합니다.
ArrayList fetch()	실행한 검색쿼리의 결과데이터 전체를 리턴합니다.
List fetchdata()	실행한 검색쿼리의 결과데이터 한 row 를 리턴합니다.
String Load(String table, String key, String partition, String control_file_path, String data_file_path)	데이터파일을 IRIS 의 LOCAL 테이블에 로딩합니다.
String LoadEx(String table, String key, String partition, String control_file_path, byte[] data)	Load 함수와 동일하나, data 파일경로 대신 파일내용을 담은 byte array 를 인자로 받습니다.
boolean hasNext()	다음 row 가 존재하면 true 를 리턴합니다.
(str list) next()	다음 row 를 리턴합니다. (iteration 인터페이스를 위한 함수입니다.)
void close()	Cursor 객체를 닫습니다.

4.3 Python

IRIS Python API 는 Query 실행, 데이터 로딩 기능을 지원합니다.

4.3.1 사용 방법

IRIS API 코어파일을 받습니다. 실행할 Python 버전에 맞는 코어파일을 받아야 합니다. IRIS API 코어파일을 PYTHONPATH 에 추가합니다.

IRIS API 를 사용할 파일을 작성합니다. IRIS API 를 import 하고 IRIS 접속정보를 이용해서 Connection 객체를 만든 후, 이를 이용해서 Cursor 객체를 만들어서 쿼리를 실행합니다.

```
import API.M6 as M6

conn = M6.Connection('127.0.0.1:5050', 'test', 'test')
c = conn.Cursor()
c.SetFieldSep('|^|')
c.SetRecordSep('|^-^|')

query = "INSERT INTO TEST_TABLE_1 (k, p, a) VALUES ('k2', '20110616000000', '1');"
exec_message =  c.Execute2(query)
print exec_message

c.Close()
conn.close()
```

4.3.1.1 쿼리 실행 cursor.Execute2(query) 함수를 실행해서 쿼리를 실행합니다.

리턴값은 쿼리 동작 결과메세지이며 “+OK” 로 시작하는 경우 정상실행입니다. “-ERR” 로 시작하는 경우는 쿼리실행이 실패한 경우이며, 자세한 내용이 메세지 뒷부분에 포함되어 있습니다.

4.3.1.2 쿼리 실행 (검색)

실행은 위의 '쿼리 실행' 과 동일합니다. 메타데이터는 cursor.Metadata() 로 가져옵니다. 검색결과데이터는 fetchone, fetchall 함수로 한번에 가져오거나, python iteration 을 사용할 수 있습니다.

```
exec_message = cursor.Execute2("SELECT * FROM TEST_TABLE_1;")
print exec_message

for row in cursor:
    print row[0], row[1]
```

4.3.1.3 데이터 로딩

한번에 대량의 데이터를 파일에 저장한 후 한번에 INSERT 할 때 사용합니다. 이 기능을 사용하면 동일한 양의 데이터를 저장하는 경우 INSERT 쿼리에 비해 저장속도가 매우 빠릅니다. 로딩한 데이터는 같은 Backend 파일에 저장됩니다. (key, partition 값이 같습니다.)

- 데이터파일의 구분자(Separator)를 확인합니다.
  - Record, Field 구분자를 확인합니다. 문자열도 가능합니다.
  - csv 파일은 RecordSep : \n, FieldSep : , 입니다.
- 컬럼 위치정보를 저장한 ctl 파일을 만듭니다.
  - Record 구분자로 나눠서 컬럼이름을 나열합니다.
  - csv 파일이면 ctl 파일에는 한줄에 한개 컬럼씩 넣습니다.
- 데이터를 저장한 dat 파일을 만듭니다.

위에서 만든 ctl, dat 파일경로를 Load, LoadGlobal 함수에 인자로 넣어서 데이터를 로딩합니다.

```
cursor.SetFieldSep(',')
cursor.SetRecordSep('\n')
```

```
msg = cursor.Load('TEST_TABLE_1', 'k1', '20110616000000', \
'TEST_TABLE_1.ctl', 'TEST_TABLE_1.dat')

msg = cursor.LoadGlobal('TEST_TABLE_2', 'TEST_TABLE_1.ctl', 'TEST_TABLE_1.dat')
```

4.3.2 API 상세

4.3.2.1 Create Connection IRIS Connection 객체를 생성합니다.

```
Connection M6.Connection({ADDRESS}, {ID}, {PASSWD}, Direct={Use-Direct})

conn = M6.Connection('127.0.0.1:5050', 'test', 'test')
conn = M6.Connection('127.0.0.1:5000', 'test', 'test', Direct=True)
```

- addr_info : ip:port 로 구성된 문자열입니다. ex) 192.168.0.1:5050
  - IRIS 기본 설치시 port : 5050 입니다.
- id, password 는 각각 IRIS DB 계정의 ID, PW 문자열 입니다.
- Direct 옵션은 True 로 설정시 Direct 모드 Connection 을 생성합니다.
  - 일반 모드는 Connection 이 Master 노드를 거쳐서 Slave 노드로 이동합니다.
  - Direct 옵션을 사용하면 Master 노드로부터 접속할 Slave 노드를 할당받은 후, Slave 노드로 바로 연결됩니다.
  - API 사용 프로그램에서 Slave 노드로 바로 접속이 가능해야 사용할 수 있습니다.
  - Direct 모드 사용시, ip 는 Master 노드의 IP 를 사용하며 port 를 변경해야 합니다.
    * IRIS 기본 설치시 port : 5000 입니다.

4.3.2.2 Connection Class 사용함수 목록

함수명	설명
Cursor Cursor()	Cursor 객체를 생성합니다.
void close()	Connection 객체를 닫습니다.

4.3.2.3 Cursor Class 사용함수 목록

함수명	설명
bool SetRecordSep(sep)	row 구분자를 설정합니다.
bool SetFieldSep(sep)	column 구분자를 설정합니다.
str Execute2(sql)	쿼리를 실행하고 메세지를 리턴합니다.
(str list) Metadata()	실행한 검색쿼리의 메타데이터를 리턴합니다.
((str list) list) Fetchall()	실행한 검색쿼리의 결과데이터 전체를 리턴합니다.
(str list) Fetchone()	실행한 검색쿼리의 결과데이터 한 row 를 리턴합니다.
str Load(table, key, partition, control_file_path, data_file)	데이터파일을 IRIS 의 LOCAL 테이블에 로딩합니다.
str LoadGlobal(table, control_file_path, data_file)	데이터파일을 IRIS 의 GLOBAL 테이블에 로딩합니다.
(str list) next()	다음 row 를 리턴합니다. (iteration 인터페이스를 위한 함수입니다.)

4.4 C / C++

IRIS C / C++ API 는 Query 실행, 데이터 로딩 기능을 지원합니다.

4.4.1 사용 방법

IRIS API 코어파일 (base64.h, socket.h, iris.h, libiris.so) 을 받아서 PATH 에 추가합니다.

IRIS API 를 사용할 파일을 작성합니다. include “iris.h” 를 한 후 컴파일 시 libiris.so 파일을 참조하도록 옵션을 추가합니다.

```
gcc test.c -L. -liris
g++ test.cpp -L. -liris
```

c 예제)

```
#include <stdio.h>
#include <stdlib.h>
#include "iris.h"

int main()
{
    IRIS_CONNECTION conn;
    IRIS_CURSOR     cursor;
    IRIS_RESULT     result;
    IRIS_ROW        row;

    IRIS_CODE       code;

    int idx;

    char ip[]   = "127.0.0.1";
    int  port   = 5050;
    char user[] = "test";
    char pass[] = "test";
    char fieldSep[]  = ",";
    char recordSep[] = "\n";
    char query[] = "INSERT INTO TEST_TABLE_1 (k, p, a) \
VALUES ('k2', '20110616000000', '1');";

    EXIT_WITH_ERROR(iris_connection_open(&conn, ip, port, user, pass));
    EXIT_WITH_ERROR(iris_cursor_open(conn, &cursor));
    EXIT_WITH_ERROR(iris_cursor_set_field_sep(cursor, fieldSep));
    EXIT_WITH_ERROR(iris_cursor_set_record_sep(cursor, recordSep));

    EXIT_WITH_ERROR(iris_execute(cursor, query));

    EXIT_WITH_ERROR(iris_cursor_close(cursor));
    EXIT_WITH_ERROR(iris_connection_close(conn));
```

```
    return 0;
}
```

cpp 예제)

```
#include "iris.h"

int main()
{
    Connection *conn;
    Cursor     *cursor;
    Result     result;
    ResultIter resultIter;
    Row        row;
    RowIter    rowIter;
    Column     column;

    string table, key, partition, ctlFilePath, dataFilePath;

    string ip   = "127.0.0.1";
    int    port = 5050;
    string user = "test";
    string pass = "test";
    string fieldSep  = ",";
    string recordSep = "\n";
    string query = "INSERT INTO TEST_TABLE_1 (k, p, a) \
VALUES ('k2', '20110616000000', '1');";

    try {
        conn = new Connection(ip, port, user, pass);
        cursor = conn->cursor();
        cursor->setFieldSep(fieldSep);
        cursor->setRecordSep(recordSep);

        cursor->execute(query);

        cursor->close();
        conn->close();
```

```
        delete cursor;
        delete conn;
    } catch(IRISException &e) {
        PRINT_EXCEPTION(e);
    } catch(SocketException &e) {
        PRINT_EXCEPTION(e);
    }

    return 0;
}
```

4.4.1.1 쿼리 실행 cursor.Execute(query) 함수를 실행해서 쿼리를 실행합니다.

c API 는 IRIS_CODE 를 리턴하며, EXIT_WITH_ERROR 함수로 IRIS_CODE 를 체크해서 예외처리를 합니다.

cpp API 는 쿼리실패시 오류메세지에 따라서 Exception 을 발생시킵니다.

4.4.1.2 쿼리 실행 (검색) 실행은 위의 '쿼리 실행' 과 동일합니다. 결과데이터는 내부적으로 fatchall 처럼 데이터를 다 가져온 후, 사용자 요청시마다 1 row 씩 리턴합니다.

c 의 경우 iris_store_result 함수로 한번에 다 가져온 후, iris_fetch_row 함수로 한 줄씩 리턴합니다.

cpp 의 경우 cursor.fetch 함수로 한번에 다 가져온 후, iteration 을 사용해서 한 줄씩 리턴합니다.

c 예제)

```
char query[] = "SELECT * FROM TEST_TABLE_1;";

EXIT_WITH_ERROR(iris_execute(cursor, query));
```

```
EXIT_WITH_ERROR(iris_store_result(cursor, &result));

while( iris_fetch_row(result, &row) != IRIS_END )
{
    for( idx = 0 ; idx < 3 ; idx++ )
    {
        if( idx != 0 )
        {
            printf(",");
        }
        printf("[%s]", row[idx]);
    }
    printf("\n");
}
EXIT_WITH_ERROR(iris_free_result(result));
```

cpp 예제)

```
string query = "SELECT * FROM TEST_TABLE_1;";

cursor->execute(query);
cursor->fetch(result);

for( resultIter = result.begin() ; resultIter != result.end() ;  ++resultIter )
{
    row = *resultIter;
    for( rowIter = row.begin() ; rowIter != row.end() ;  ++rowIter )
    {
        if( rowIter != row.begin() ) {
            cout << ",";
        }
        column = *rowIter;
        cout << "[" << column << "]";
    }
    cout << endl;
}
```

4.4.1.3 데이터 로딩 한번에 대량의 데이터를 파일에 저장한 후 한번에 INSERT 할 때 사용합니다. 이 기능을 사용하면 동일한 양의 데이터를 저장하는 경우 INSERT 쿼리에 비해 저장속도가 매우 빠릅니다. 로딩한 데이터는 같은 Backend 파일에 저장됩니다. (key, partition 값이 같습니다.)

- 데이터파일의 구분자(Separator)를 확인합니다.
  - Record, Field 구분자를 확인합니다. 문자열도 가능합니다.
  - csv 파일은 RecordSep : \n, FieldSep : , 입니다.
- 컬럼 위치정보를 저장한 ctl 파일을 만듭니다.
  - Record 구분자로 나눠서 컬럼이름을 나열합니다.
  - csv 파일이면 ctl 파일에는 한줄에 한개 컬럼씩 넣습니다.
- 데이터를 저장한 dat 파일을 만듭니다.

위에서 만든 ctl, dat 파일경로를 iris_load / cursor->load 함수에 인자로 넣어서 데이터를 로딩합니다.

c 예제)

```
char fieldSep[]  = ",";
char recordSep[] = "\n";
char table[]        = "TEST_TABLE_1";
char key[]          = "k1";
char partition[]    = "20150606000000";
char ctlFilePath[]  = "TEST_TABLE_1.ctl";
char dataFilePath[] = "TEST_TABLE_1.dat";

EXIT_WITH_ERROR(iris_cursor_set_field_sep(cursor, fieldSep));
EXIT_WITH_ERROR(iris_cursor_set_record_sep(cursor, recordSep));

EXIT_WITH_ERROR(iris_load(cursor, table, key, partition, ctlFilePath, dataFilePath));
```

cpp 예제)

```
string fieldSep  = ",";
string recordSep = "\n";
string table        = "TEST_TABLE_1";
string key          = "k1";
string partition    = "20150606000000";
string ctlFilePath  = "TEST_TABLE_1.ctl";
string dataFilePath = "TEST_TABLE_1.dat";

cursor->setFieldSep(fieldSep);
cursor->setRecordSep(recordSep);

cursor->load(table, key, partition, ctlFilePath, dataFilePath);
```

4.4.2 API 상세 : c

4.4.2.1 Create Connection IRIS Connection 객체를 생성합니다.

```
iris_connection_open(&conn, {IP}, {PORT}, {ID}, {PASSWD});

IRIS_CONNECTION conn;
EXIT_WITH_ERROR(iris_connection_open(&conn, "127.0.0.1", 5050, "test", "test"));
```

- addr_info : ip:port 로 구성된 문자열입니다. ex) 192.168.0.1:5050
  - IRIS 기본 설치시 port : 5050 입니다.
- id, password 는 각각 IRIS DB 계정의 ID, PW 문자열 입니다.

4.4.2.2 Connection 관련 함수 사용함수 목록

함수명	설명
IRIS_CODE iris_cursor_open (IRIS_CONNECTION conn, IRIS_CURSOR *cursor)	Cursor 객체를 생성합니다.
IRIS_CODE iris_connection_close (IRIS_CONNECTION conn)	Connection 객체를 닫습니다.

4.4.2.3 Cursor 관련 함수 사용함수 목록

함수명	설명
void SetTimeout(int sec)	소켓 Timeout 을 설정합니다.
IRIS_CODE iris_cursor_set_record_sep (IRIS_CURSOR cursor, char *recordSep)	row 구분자를 설정합니다.
IRIS_CODE iris_cursor_set_field_sep (IRIS_CURSOR cursor, char *fieldSep)	column 구분자를 설정합니다.
IRIS_CODE iris_execute (IRIS_CURSOR cursor, char *query)	쿼리를 실행하고 메세지를 리턴합니다.
IRIS_CODE iris_store_result (IRIS_CURSOR cursor, IRIS_RESULT *result)	쿼리 실행결과를 담은 RESULT 객체를 생성합니다.
IRIS_CODE iris_fetch_row (IRIS_RESULT result, IRIS_ROW *row)	실행한 검색쿼리의 결과데이터 한 row 를 리턴합니다.
IRIS_CODE iris_free_row(IRIS_RESULT result)	RESULT 객체를 메모리해제합니다.
IRIS_CODE iris_load (IRIS_CURSOR cursor, char *table, char *key, char *partition, char *ctlFilePath, char *dataFilePath)	데이터파일을 IRIS 의 LOCAL 테이블에 로딩합니다.
IRIS_CODE iris_load_global (IRIS_CURSOR cursor, char *table, char *ctlFilePath, char *dataFilePath)	데이터파일을 IRIS 의 GLOBAL 테이블에 로딩합니다.
IRIS_CODE iris_load_ex (IRIS_CURSOR cursor, char *table, char *key, char *partition, char *ctlFilePath, int dataSize, char *data)	Load 함수와 동일하나, data 파일경로 대신 파일내용을 담은 char array 를 인자로 받습니다.
IRIS_CODE iris_cursor_close (IRIS_CURSOR cursor)	Cursor 객체를 닫습니다.

4.4.3 API 상세 : cpp

4.4.3.1 Create Connection IRIS Connection 객체를 생성합니다.

```
Connection({IP}, {PORT}, {ID}, {PASSWD});


conn = new Connection("127.0.0.1", 5050, "test", "test");
```

- addr_info : ip:port 로 구성된 문자열입니다. ex) 192.168.0.1:5050

  - IRIS 기본 설치시 port : 5050 입니다.

- id, password 는 각각 IRIS DB 계정의 ID, PW 문자열 입니다.
- Direct 옵션을 지원하지 않습니다.

4.4.3.2 Connection Class 사용함수 목록

함수명	설명
Cursor* cursor()	Cursor 객체를 생성합니다.
void close()	Connection 객체를 닫습니다.

4.4.3.3 Cursor Class 사용함수 목록

함수명	설명
void setRecordSep(string str)	row 구분자를 설정합니다.
void setFieldSep(string str)	column 구분자를 설정합니다.
void execute(string query)	쿼리를 실행하고 메세지를 리턴합니다.
void fetch(Result &result)	쿼리 실행결과를 담은 Result 객체를 생성합니다.
void load(string table, string key, string partition, string ctlFilePath, string dataFilePath)	데이터파일을 IRIS 의 LOCAL 테이블에 로딩합니다.
void loadGlobal(string table, string ctlFilePath, string dataFilePath)	데이터파일을 IRIS 의 GLOBAL 테이블에 로딩합니다.
void loadEx(string table, string key, string partition, string ctlFilePath, int dataSize, char *data)	Load 함수와 동일하나, data 파일경로 대신 파일내용을 담은 char array 를 인자로 받습니다.

4.5 C#

IRIS C# API 는 Query 실행, 데이터 로딩 기능을 지원합니다.

4.5.1 사용 방법

IRIS API 코어파일(dll) 을 받아서 C# 프로젝트에 추가합니다.

IRIS API 를 사용할 프로그램을 작성합니다. IRIS API 를 import 하고 IRIS 접속정보를 이용해서 Connection 객체를 만든 후, 이를 이용해서 Cursor 객체를 만들어서 쿼리를 실행합니다.

```
using com.mobigen.iris.csharp;

string ip = "127.0.0.1:5050;"
int port = 5050;
string user = "test";
string passwd = "test";
string fieldSep = ",";
string recordSep = "\n";

Connection conn = new Connection();
conn.Connect(ip, port, user, passwd);
Cursor cursor = conn.Cursor();
cursor.SetFieldSep(fieldSep);
cursor.SetRecordSep(recordSep);

string query = "INSERT INTO TEST_TABLE_1 (k, p, a) VALUES ('k2', '20110616000000', '1');"
string msg = cursor.Execute2(sql);

cursor.Close();
conn.Close();
```

4.5.1.1 쿼리 실행 cursor.Execute2(query) 함수를 실행해서 쿼리를 실행합니다. 리턴값은 쿼리 동작 결과메세지이며 “+OK” 로 시작하는 경우 정상실행입니다. “-ERR” 로 시작하는 경우는 쿼리실행이 실패한 경우이며, 자세한 내용이 메세지 뒷부분에 포함되어 있습니다.

4.5.1.2 쿼리 실행 (검색)

실행은 위의 '쿼리 실행' 과 동일합니다. 메타데이터는 cursor.Metadata() 로 가져옵니다. 검색결과데이터는 Fetch, FetchAll 함수로 한번에 가져오거나, iteration 을 사용할 수 있습니다.

```
string sql = "SELECT * FROM TEST_TABLE_1;";
string exec_message = cursor.Execute2(sql);

Metadata metadata = cursor.Metadata();
foreach (Row row in cursor)
{
    for (int i = 0; i < metadata.columnCount; i++)
    {
        Console.Write(row.getColumn(i) + ", ");
    }
    Console.WriteLine();
}
```

4.5.1.3 데이터 로딩

한번에 대량의 데이터를 파일에 저장한 후 한번에 INSERT 할 때 사용합니다. 이 기능을 사용하면 동일한 양의 데이터를 저장하는 경우 INSERT 쿼리에 비해 저장속도가 매우 빠릅니다. 로딩한 데이터는 같은 Backend 파일에 저장됩니다. (key, partition 값이 같습니다.)

- 데이터파일의 구분자(Separator)를 확인합니다.
  - Record, Field 구분자를 확인합니다. 문자열도 가능합니다.
  - csv 파일은 RecordSep : \n, FieldSep : , 입니다.
- 컬럼 위치정보를 저장한 ctl 파일을 만듭니다.
  - Record 구분자로 나눠서 컬럼이름을 나열합니다.
  - csv 파일이면 ctl 파일에는 한줄에 한개 컬럼씩 넣습니다.
- 데이터를 저장한 dat 파일을 만듭니다.

위에서 만든 ctl, dat 파일경로를 Load 함수에 인자로 넣어서 데이터를 로딩합니다.

```
cursor.SetFieldSep(",")
cursor.SetRecordSep("\n")

msg = cursor.Load("TEST_TABLE_1", "k1", "20110616000000", \
"TEST_TABLE_1.ctl", "TEST_TABLE_1.dat")
```

4.5.2 API 상세

4.5.2.1 Create Connection IRIS Connection 객체를 생성합니다.

```
conn.Connect({IP}, {PORT}, {ID}, {PASSWD});

Connection conn = new Connection();
conn.Connect("127.0.0.1", 5050, "test", "test");
```

- addr_info : ip:port 로 구성된 문자열입니다. ex) 192.168.0.1:5050
  - IRIS 기본 설치시 port : 5050 입니다.
- id, password 는 각각 IRIS DB 계정의 ID, PW 문자열 입니다.

4.5.2.2 Connection Class 사용함수 목록

함수명	설명
Cursor Cursor()	Cursor 객체를 생성합니다.
void Close()	Connection 객체를 닫습니다.

4.5.2.3 Cursor Class 사용함수 목록

함수명	설명
string SetRecordSep(string sep)	row 구분자를 설정합니다.
string SetFieldSep(string sep)	column 구분자를 설정합니다.
string Execute2(string sql)	쿼리를 실행하고 메세지를 리턴합니다.
Metadata Metadata()	실행한 검색쿼리의 메타데이터를 리턴합니다.
int Fetch(Boolean bAllFlag, ref ArrayList AL)	실행한 검색쿼리의 결과데이터를 인자로 받은 ArrayList 에 저장합니다.
int FetchAll(ref ArrayList AL)	실행한 검색쿼리의 결과데이터 전체를 인자로 받은 ArrayList 에 저장합니다.
string Load(string table, string key, string partition, string controlFilePath, string dataFilePath)	데이터파일을 IRIS 의 LOCAL 테이블에 로딩합니다.
string LoadEx(string table, string key, string partition, string controlFilePath, string data)	Load 함수와 동일하나, data 파일경로 대신 파일내용을 담은 string 문자열을 인자로 받습니다.

4.6 ODBC

IRIS 는 Linux(unixODBC) 와 Windows 에서 사용가능한 ODBC 드라이버를 제공합니다. 이 드라이버를 사용해서 IRIS DB 에 접속할 수 있습니다.

4.6.1 사용 방법 (Linux)

unixODBC 를 설치합니다. CentOS 에서는 yum 으로 설치가능합니다.

```
# yum install unixODBC
```

IRIS ODBC 드라이버파일(so 파일) 을 받습니다. 실행할 OS 버전에 맞게 컴파일된 코어파일을 받아야 합니다.

IRIS ODBC 드라이버파일을 라이브러리 디렉토리에 저장합니다. (/usr/lib/ 등)

IRIS ODBC 드라이버파일을 unixODBC 에 등록합니다.

1. 드라이버 등록

   - 드라이버 등록파일 내용

```
[irisodbc]
Description     = irisodbc driver
Driver          = /usr/lib/irisodbc.so
```

   - 등록 명령어

```
odbcinst -i -d -f 드라이버등록파일
```

2. DSN(Data Source Name) 등록

   - DSN 등록파일 내용

```
[IRISTEST]
Description     = iris test server
Driver          = irisodbc
host            = 192.168.131.10
port            = 5050
id              = test
pass            = test
fieldsep        = |^|
recordsep       = |^-^|
direct          = 0
logfile         = /home/iris/irisodbc.log
debug_level     = 1
timeout         = 60
```

   - 등록 명령어

```
odbcinst -i -s -f DSN등록파일
```

4.6.1.1 쿼리 실행 c 예제

```
#include <stdio.h>
#include <sqlext.h>
```

```
int main() {
    SQLHENV env;
    SQLHDBC dbc;
    SQLHSTMT stmt;
    SQLRETURN ret; /* ODBC API return status */
    char errmsg[500];
    int err_size;

    char * query = "SELECT * FROM LOCAL_TEST_TABLE_ODBC";

    char key_buf[100];
    char partition_buf[100];
    char data_buf[100];

    /* Allocate an environment handle */
    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);

    /* We want ODBC 3 support */
    SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3, 0);

    /* Allocate a connection handle */
    SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);

    /* Connect to the DSN mydsn */
    ret = SQLConnect( dbc, (SQLCHAR*) "IRISTEST", SQL_NTS, (SQLCHAR*) "kddi", SQL_NTS, \
(SQLCHAR*) "kddi", SQL_NTS);
    if( !SQL_SUCCEEDED(ret) ) {
        //ret = SQLError( env, dbc, stmt, NULL, NULL, errmsg, sizeof(errmsg), NULL );
        ret = SQLError( env, dbc, stmt, NULL, NULL,(SQLCHAR*) errmsg, sizeof(errmsg), \
(SQLSMALLINT*)(&err_size) );
        printf( "Connection Failed : %d %s end\n", ret, errmsg );
        return 1;
    }

    /* Allocate a statement handle */
    SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if( !SQL_SUCCEEDED(ret) ) {
        ret = SQLError( env, dbc, stmt, NULL, NULL, errmsg, sizeof(errmsg), NULL );
        printf( "Alloc STMT Failed : %d %s end\n", ret, errmsg );
```

```
        return 1;
    }

    /* Retrieve a list of tables */
    ret = SQLExecDirect(stmt, query, SQL_NTS);
    if( !SQL_SUCCEEDED(ret) ) {
        ret = SQLError( env, dbc, stmt, NULL, NULL, errmsg, sizeof(errmsg), NULL );
        printf( "Exec Query Failed : %d %s end\n", ret, errmsg );
        return 1;
    }

    SQLBindCol( stmt, 1, SQL_C_CHAR, key_buf, sizeof(key_buf) , NULL );
    SQLBindCol( stmt, 2, SQL_C_CHAR, partition_buf, sizeof( partition_buf ), NULL );
    SQLBindCol( stmt, 3, SQL_C_CHAR, data_buf, sizeof(data_buf) , NULL );
    int row_cnt = 0;
    while (1) {
        ret = SQLFetch(stmt);
        if (!SQL_SUCCEEDED(ret))
        {
            if (ret != SQL_NO_DATA_FOUND)
            {
                ret = SQLError( env, dbc, stmt, NULL, NULL, errmsg, sizeof(errmsg), NULL );
                printf( "Fetch Failed : %d %s end\n", ret, errmsg );
            }
            break;
        }
        printf("%s %s %s \n", key_buf, partition_buf, data_buf );
        row_cnt++;
    }
    printf("row count : %d\n", row_cnt);

    /* Disconnect and free all the handles */
    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    SQLDisconnect(dbc);
    SQLFreeHandle(SQL_HANDLE_DBC, dbc);
    SQLFreeHandle(SQL_HANDLE_ENV, env);

    return 0;
}
```

4.6.1.2 compile 방법

```
gcc TEST_CODE.c -l odbc
```

4.6.2 사용 방법 (Windows)

IRIS ODBC 드라이버파일(dll)을 라이브러리 디렉토리에 저장합니다. (기본 경로 : c:\irisodbc\)

reg 파일을 실행해서 IRIS ODBC 드라이버파일을 ODBC관리자에 등록합니다.

- OS 버전에 따라 irisodbc_{os_bit_ver}.reg 을 실행하세요.
  - 실행 전에 해당 파일을 텍스트에디터로 열어서 설정값을 변경할 수 있습니다.
- 윈도우2008 이후 버전의 OS 는 기본적으로 관리자계정에서 실행한 프로그램만 ODBC관리자에 접근할 수 있습니다. 일반계정으로 실행한 프로그램도 ODBC관리자를 접근하려면 Registry 권한을 수정해야 합니다.
  - 각 OS 별로 아래의 레지스트리의 권한설정에서 “Users” 가 “모든 권한” 을 허용해야 합니다.
  - 32bit : HKEY_LOCAL_MACHINE\SOFTWARE\ODBC
  - 64bit : HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC
- 드라이버 정보를 삭제할 경우 irisodbc_{os_bit_ver}_remove.reg 파일을 실행하세요.

4.6.2.1 쿼리 실행 c# 예제

```
string strDbConnectInfo = "DRIVER={iris ODBC};HOST=1.221.187.110;PORT=55555;" +
"ID=kddi;PASS=kddi;RECORDSEP=|^-^|;FIELDSEP=|^|;TIMEOUT=60";

OdbcConnection sqlCon = new OdbcConnection(strDbConnectInfo);
sqlCon.Open();
if (sqlCon.State == System.Data.ConnectionState.Open)
{
    OdbcCommand sqlCommand = new OdbcCommand();
    sqlCommand.Connection = sqlCon;
    //sqlCommand.CommandTimeout = 60000;
    OdbcDataAdapter dataAdpter = new OdbcDataAdapter();

    string strQuery = "select * from dummy_table";

    sqlCommand.CommandText = strQuery;
    dataAdpter.SelectCommand = sqlCommand;

    DataSet ds = new DataSet();
    dataAdpter.Fill(ds);

    DataTable dt = new DataTable();
    dt = ds.Tables[0];

    foreach (DataRow dr in dt.Rows)
    {
        Console.WriteLine(dr[0].ToString());
    }

    sqlCon.Close();
}
```

4.6.3 지원함수 목록

지원함수	설명
SQLAllocHandle	ODBC 핸들 객체를 할당합니다.
SQLSetEnvAttr	SQLHENV 객체를 초기화합니다.

지원함수	설명
SQLConnect	입력받은 DSN 정보로 DB에 접속합니다.
SQLDriverConnect	입력받은 접속정보의 키워드들로 DB에 접속합니다.
SQLBindCol	인자로 받은 변수를 SELECT 쿼리의 ResultColumn 에 연결해서 SQLFetch 함수 호출시 변수에 값이 저장되도록 합니다. 지원타입 : SQL_C_CHAR, SQL_INTEGER, SQL_DOUBLE
SQLNumResultCols	실행한 SELECT 쿼리의 결과컬럼 갯수를 출력합니다.
SQLDisconnect	DB접속을 해제합니다.
SQLError	ODBC 함수 실행중 오류코드 리턴시, 세부 메세지를 가져옵니다.
SQLExecDirect	SQL 쿼리를 실행합니다.
SQLFetch	SELECT 쿼리 실행결과를 한줄씩 가져와서 SQLBindCol 함수로 설정한 변수에 저장합니다.
SQLFreeHandle	ODBC 핸들 객체를 해제합니다.

4.6.4 사용시 유의사항

- SQLConnect 사용시 ID, PASSWORD 는 입력값을 사용하며 DSN 의 ID, PASSWORD 는 사용하지 않습니다.
- SQLDirectConnect 사용시 DSN을 설정하면 그 외의 옵션을 무시하고 해당 DSN의 값으로 설정됩니다.
- SQLSTMT 는 1회의 쿼리실행만 가능합니다. 재사용할 수 없습니다.
- IRIS 오류메세지는 로그파일에 저장됩니다.
- timeout 단위는 sec 입니다.
- debug_level 값 의미
  - 0 : 로깅 미사용
  - 1 : 로깅 사용

5 쿼리 사용법

IRIS 는 일반적인 형태의 SQL 문을 지원합니다.

SQL 문을 API 나 IPlus(IRIS Colsole Interface) 를 이용해서 실행할 수 있습니다.

5.1 로딩

IRIS 는 대용량 데이터 로딩을 지원합니다. csv 파일처럼 여러개의 레코드를 단일파일에 저장한 후, Load 함수에 파일 경로를 인자로 넣어서 해당 파일내용을 전부 INSERT 합니다.

다수의 데이터를 저장할 때, INSERT 쿼리로는 매우 많은 수의 작업을 수행해야 하지만, Load 는 1회의 작업으로 수행합니다. 1회 Load 작업할 때 공통으로 사용하는 KEY, PARTITION 값을 인자로 넣습니다. KEY, PARTITION 값이 같은 데이터끼리 모아서 한번에 데이터를 넣습니다.

IPlus 또는 API 에 구현된 Load 함수를 사용해서 로딩할 수 있습니다. 자세한 설명 및 예제는 API 항목에 포함되어 있습니다.

5.2 JOIN

IRIS 는 JOIN 사용시 제약사항이 있습니다.

모든 SubQuery 를 포함해서, LOCAL 타입 테이블은 0개, 혹은 1개만 존재할 수 있으며, 테이블이름이 같더라도 여러번 사용했으면 여려개가 존재하는 것으로 간주합니다.

사용가능한 조인은 INNER JOIN, LEFT OUTER JOIN 두 가지 입니다.

5.2.1 INNER JOIN

```
FROM <TableName> INNER JOIN <TableName> [ON <expr>] [WHERE <expr>] ...
FROM <TableName>, <TableName> ... [WHERE <expr>] ...
```

일반적인 n * n JOIN 입니다.

데이터량이 많아질 수 있으므로 LOCATION HINT 와 WHERE 절을 잘 활용하세요. 중간데이터량이 매우 많은 경우 IRIS 에서 해당 작업을 강제종료할 수 있습니다.

5.2.2 LEFT OUTER JOIN

```
FROM <A> LEFT OUTER JOIN <B> [ON <expr>] [WHERE <expr>] ...
```

LOCAL 테이블은 B 의 위치에 사용합니다.

JOIN 연산은 각 분산저장된 파일 단위로 수행됩니다. 각 분산파일단위로 JOIN 수행시 일부 데이터가 누락될 수 있으므로 값이 없을 때 null 이 되는 B 의 위치에 LOCAL 테이블을 사용합니다.

A 는 GLOBAL 테이블을, B 에는 LOCAL 테이블또는 GLOBAL 테이블을 위치시켜서 사용합니다.

B(LOCAL TABLE) 만 대상으로 하는 WHERE 조건은 on 절에 넣습니다.

나머지 WHERE 조건은 WHERE 절에 넣습니다.

5.2.3 기타 : Oracle JOIN 과 비교

ORACLE 은 릴리즈 6부터 Oracle-specific한 문법으로 사용되는 left outer-join의 특별한 형식을 지원합니다.

ORACLE 의 left outer join 문법과 IRIS 의 문법은 동일하지 않습니다. 아래는 동일한 작업을 각각의 SQL 문법으로 표현한 것입니다.

```
오라클의 경우

SELECT T1.d, T2.c
FROM T1, T2
WHERE T1.x = T2.x (+);



IRIS 의 경우

SELECT T1.d, T2.c
FROM T1 LEFT OUTER JOIN T2
ON (T1.x = T2.x);
```

5.3 HINT

IRIS 는 쿼리실행시 IRIS DB 에서 제공하는 기능을 활용하기 위해 HINT 라 불리는 문법을 사용할 수 있습니다.

실행할 쿼리 앞부부분에 붙여서 사용합니다. 어려개의 HINT 를 사용할 경우, 콤마(,) 로 구분해서 여러개를 함께 사용할 수 있습니다.

주의 : LOCAL 테이블 사용시에는 반드시 LOCATION HINT 와 함께 사용하세요.

```
/*+ HINT-String */ Query-String


ex)
/*+ BYPASS, LOCATION (PARTITION >= '20160101000000'
AND PARTITION < '20160201000000')*/
Select * from Test_Table_1;
```

5.3.1 HINT 종류

HINT 는 UPDATE, DELETE, SELECT 쿼리 에서만 사용할 수 있습니다.

HINT 이름	사용가능한 쿼리	설명/사용법
LOCATION	전체	LOCAL 테이블 대상으로 쿼리실행시 참조할 데이터범위를 설정합니다. 사용법 : 아래 참조
FORMAT	SELECT	쿼리 실행결과의 출력 형태를 변경합니다. 사용법 : 아래 참조
THREAD_COUNT	SELECT	해당 쿼리의 동시접근파일수를 설정해서 (기본값 50) 작업 부하 및 속도를 조절합니다. 사용법 : THREAD_COUNT = {자연수}
BYPASS	SELECT	쿼리 실행결과를 summary 작업 없이 바로 출력합니다. 사용법 : BYPASS
FORCE	전체	일부 데이터에서 쿼리오류가 발생해도 무시하고 나머지 정상데이터를 출력합니다. 사용법 : FORCE

LOCATION HINT

이 HINT 는 LOCAL 테이블에서만 사용합니다.

IRIS 의 LOCAL 테이블은 KEY, PARTITION 값을 사용해서 데이터를 분산저장합니다. 쿼리 실행할 때, LOCATION HINT 에서 참조할 KEY, PARTITION 값의 범위를 설정해서 쿼리의 실행시 부하를 줄이고 속도를 높일 수 있습니다. 괄호 안에 SQL WHERE 절과 동일한 문법으로 조건을 설정할 수 있습니다. 사용가능한 컬럼은 KEY, PARTITION 두 개 입니다.

```
예시 : KEY 는 key3 이고, PARTITION 은 2015년 1월 1일 부터 2월 직전까지의 범위로 참조하는 경우
/*+ LOCATION (
    KEY = 'key3'
    AND PARTITION >= '20160101000000'
    AND PARTITION < '20160201000000'
) */
```

FORMAT HINT

출력결과의 데이터 포맷을 설정합니다. 중간 SubQuery 에는 사용할 수 없으며, 최종결과에만 사용할 수 있습니다. 각 Column 별로 데이터 출력포맷을 설정하며 '인덱스번호 = 포맷' 형태로 되어있습니다. 인덱스는 0 부터 시작합니다. 문자열 포맷 규칙은 Python 언어에서 사용하는 문자열 포맷과 같습니다.

데이터 타입	포맷 문자열	사용법
문자열	%s	%{최소길이}.{최대길이}s
정수	%d	%{최소길이}.{0으로 채우는 최소길이}d
실수	%f	%{최소길이(소숫점영역 포함)}.{소숫점자리 고정길이}f

예시 : 첫번째 컬럼은 3자리 숫자이며 앞은 0으로 채우고, 세번째 컬럼은 소수점 2번째까지 표시

```
/*+FORMAT(0='%03d',2='%.2f')*/ select id,name, point from ACCOUNT;
```

- 기본은 우측정렬이며, %음수s 처럼 가운데숫자를 음수로 설정하면 좌측정렬이 됩니다.
- 정수, 실수의 경우 가운데숫자가 0으로 시작하면 앞부분 빈공간을 0으로 채웁니다.
- 하나의 값이라도 형변환이 실패하면 쿼리작업 전체가 실패합니다.
- HINT 만으로는 문자열 컬럼을 정수 또는 실수로 형변환하지 못합니다.
- 정수, 실수는 문자열로 형변환할 수 있습니다.
- 실수 출력시 표현할 소숫점자리보다 실제 값의 소숫점자리가 길 경우, 자동으로 반올림됩니다.

5.4 지원 함수 목록

IRIS SQL 에서 지원하는 함수 목록입니다.

Group	Attribute	Global Table	Local Table
Aggregate Func	count()	O	O 별첨
	max()	O	O 별첨
	min()	O	O 별첨
	sum()	O	O 별첨
Func	concat()	O 별첨	O 별첨
	abs()	O	O
	char()	O	O
	coalesce()	O	O
	ifnull()	O	O
	hex()	O	O
	length()	O	O
	lower()	O	O
	ltrim()	O	O
	nullif()	O	O
	quote()	O	O
	randomblob()	O	O
	replace()	O	O
	round()	O	O
	rtrim()	O	O
	sqlite_compileoption_get()	O	O
	sqlite_compileoption_used()	O	O
	substr()	O	O
	trim()	O	O
	typeof()	O	O
	unicode()	O	O
	upper()	O	O
	zeroblob()	O	O
	date()	O	O
	time()	O	O
	datetime()	O	O
	julianday()	O	O
	strftime()	O	O
	to_number()	X 별첨	X 별첨
	to_date()	O 별첨	O 별첨
	to_char()	O	O
	power()	O	O
	instr()	O	O
	lpad()	O	O
	rpad()	O	O

Group	Attribute	Global Table	Local Table
	ifnull2()	O	O
	ceil()	O	O
	floor()	O	O
	sqrt()	O	O
	b64d()	O	O
	fstr()	O	O
	rmhint()	O	O
	rmcar()	O	O
	encrypt()	O	O
	decrypt()	O	O
	sha256()	O	O
	passwd()	O	O
Paragraph	group by	O	O 별첨
	order by[asc, desc]	O	O 별첨
	case when then end	O	O
	having	X	X
	limit	O 별첨	O 별첨
Operator	[=, >, <, >=, <=, <>]	O	O
	between and	O	O
	In	O	O
	like	O	O
	Is null, is not null	O	O
	and, or, not	O	O
	escape	O 별첨	O 별첨
	distinct	O	O

5.4.1 별첨항목

Aggregation 함수(sum, min, max, count), Group by, Order by 사용법

IRIS 는 LOCAL 테이블 사용시 최상위쿼리에서만 Aggregation 함수나나 Group by, Order by 연산이 정상적으로 동작합니다. 하위쿼리(SubQuery)에서도 사용은 가능하지만, 해당 작업은 개별 파일별로 실행되기 때문에 일부 데이터만으로 연산을 수행하며, 의미상 원하는 값이 나오지 않습니다. 사용법은

일반 함수와 동일하며, IRIS 내부에서 해당 함수를 Map/Reduce 작업에 맞게 쿼리를 변형해서 사용합니다.

ex) 정상실행되지 않는 경우

```
SELECT a_sum FROM (SELECT sum(a) AS a_sum FROM ONE_LOCAL_TABLE Group by a)
Order by a_sum;
```

ex) 정상실행되는 경우

```
SELECT sum(val), sum(val2), count(val) FROM ( SELECT val, val * val AS val2
FROM ONE_LOCAL_TABLE);
```

concat

- 사용법 : A||B

문자열을 더합니다. 한쪽이 NULL 일 경우 NULL 을 리턴합니다.

to_number

타입이 연산과정에서 자동으로 변환됩니다. 숫자로 변환가능한 문자열은 그 숫자로 변환합니다. 숫자로 변환불가능한 문자열은 0 으로 취급합니다.

limit

IRIS 에서는 limit 문법이 총 3가지 있습니다.

- limit
  - 일반 limit 문법과 동일한 결과가 나옵니다.
  - 전체 데이터를 가져와서 최종 summary 할 때 limit 를 적용합니다.

- dlimit
  - 분할된 데이터에 limit 를 적용해서 가져옵니다.
  - 최종 summary 할 때는 limit 가 적용되지 않습니다.
- mlimit
  - limit 와 dlimit 을 합친 것처럼 동작합니다.
  - 분할된 데이터에 limit 를 적용해서 가져온 후 최종 summary 할 때도 limit 를 적용합니다.

escape

like 와 함께 사용합니다. like 뒤에 위치하며, like 조건의 문자열에서 사용할 escape 문자를 설정합니다.

```
LIKE '<condition>' escape '<char>'

ex) % 를 와일드카드가 아니라 문자열로 사용하기 위해 \ 를 escape 문자열로 설정하고 \% 를 사용함.
    select * from TABLE_1 where a like '123\%' escape '\';
```

5.4.2 지원함수 명세

upper

- 사용법 : upper(A)

영문 소문자를 대문자로 변환합니다.

substr

- 사용법 : substr(X,Y,Z) , substr(X,Y)

X 문자열을 Y위치부터 Z길이만큼 부분문자열을 만듭니다. substr(X,Y) 으로 사용하면 X 문자열을 Y위치부터 X문자열 끝까지 부분문자열을 만듭니다.

Y위치는 양수이면 1이첫번째글자인덱스입니다. 커질수록끝으로이 동합니다. 0 역시 첫번째 글자를 가리키지만 인덱스연산시 혼동할 수 있습니다. 음수이면 -1이 마지막 글자 인덱스입니다. 작아질수록 처음 으로 이동합니다.

Z값은 항상 자연수여야하고, 음수일 경우 해당 값을 사용하지 않고 substr(X,Y) 형태로 실행됩니다.

Y, Z 값으로 만들어진 실제 인덱스범위가 첫번째 글자를 1로 기준으 로 해서 (1 뒤는 -1로 가정합니다.)

(음수, 음수) 인 경우(substr('123',-7,2) => -4,-2) , 실패로 간주 하고 전체 문자열을 반환합니다.

(음수, 양수) 인 경우(substr('123',-7,6) => -4,2) , 범위에 허용 되는 문자열을 생성합니다. ('12')

(양수, 양수) 인 경우(substr('123',2,6) => 2,8) , 범위에 허용되 는 문자열을 생성합니다. ('23')

시작위치가 글자길이보다 크면 빈문자열을 생성합니다. substr('123',5) : ''

length

- 사용법 : length(X)

문자열 X 의 길이를 출력합니다.

round

- 사용법 : round(X,Y), round(X)

숫자 X 를 소수점 Y+1번째 수에서 반올림합니다.

Y값은 0을 포함한 자연수만 정상작동하며, 나머지 값의 경우는 0으로 처리합니다. 소수점 이하에서만 반올림이 가능합니다.

round(X) 으로 사용하면 소수점 첫번째 수에서 반올림합니다. (Y값 이 0인 경우와 동일합니다.)

to_date

- 사용법 : to date(X, FORMAT)

X에 해당하는 값을 FORMAT 형태로 인식해서 Timestamp 값으로 변 환합니다. 1970/01/01 00:00:00 GMT 를 시작으로, 현재까지 경과한 초를 계산합니다.

SYSDATE 키워드를 사용하면 해당 값이 현재시간 문자열로 사용됩니다.

```
현재 시간이 2015년 1월 1일 13시 30분 30초 인 경우
insert into GLOBAL TEST TABLE (k,p,a) values ('a', SYSDATE, 1);
-> insert into GLOBAL TEST TABLE (k,p,a) values ('a', '20150101133030', 1);
```

FORMAT 키워드

Format	Output
%d	day of month: 00
%f	fractional seconds: SS.SSS
%H	hour: 00-24
%j	day of year: 001-366
%J	Julian day number
%m	month: 01-12
%M	minute: 00-59
%s	seconds since 1970-01-01
%S	seconds: 00-59
%w	day of week 0-6 with Sunday==0

Format	Output
%W	week of year: 00-53
%Y	year: 0000-9999
%%	%

```
ex)
SELECT to_date( p, '%Y%m%d%H%M%S' ) FROM LOCAL_TEST_TABLE;
TO_DATE(P, '%Y%m%d%H%M%S')
==========================================================
1355106008.0
1355106008.0
1355106008.0
```

to_char

- 사용법 : to char(X, FORMAT)

X의 날짜문자열(YYYYmmddHHMMSS 형태)을 FORMAT 형태로 출력합니다. (위의 to date 에서 사용하는 FORMAT 과 문법이 다릅니다.) X 가 날짜문자열이 아니면 오류를 출력합니다.

FORMAT 에서 아래의 키워드를 X 로부터 생성한 날짜정보로 변환합니다.

Format	Output
YYYY	년
MM	월
DD	일
HH24	시간
MI	분
SS	초
D	요일 ('1' : '일요일', '2' : '월요일', ⋯ , '7' : '토요일')

```
ex)
SELECT to_char( p, 'YYYY year MM month DD day' ) FROM LOCAL_TEST_TABLE;
```

```
TO_CHAR(P, 'YYYY year MM month DD day')
=======================================================
2011 year 06 month 16 day
2011 year 06 month 16 day
```

ifnull

- 사용법 : ifnull(A, B)

A 컬럼값이 NULL 인지 확인해서 NULL 이면 B 를 리턴하고 아니면 원래 값을 리턴합니다.

sum

- 사용법 : sum(X)

해당 컬럼의 값을 모두 더합니다. 정수나 실수로 변환되지 않는 값은 0 으로 인식합니다. 모든 값이 0을 제외한 정수인 경우에만 정수로 출력됩니다.

max

- 사용법 : max(X)

해당 컬럼에서 가장 큰 값을 반환합니다. 기본으로 문자열비교로 크고 작음을 판단하므로, FORMAT HINT 를 사용해서 형변환을 해야합니다.

min

- 사용법 : min(X)

해당 컬럼에서 가장 작은 값을 반환합니다. 기본으로 문자열비교로 크고 작음을 판단하므로, FORMAT HINT 를 사용해서 형변환을 해야합니다.

count

- 사용법 : count(X) , count(*)

테이블의 해당 컬럼의 NULL 이 아닌 레코드갯수를 반환합니다. count(*)은 테이블의 전체 레코드갯수를 반환합니다.