02. 5주차. 10장 그루비 데이터베이스 프로그래밍

Table of Contents

그루비 데이터 베이스 프로그래밍343

자바에서는 JDBC(java database connectivity)통해서 접근해야한다.
그루비는 어떨까? JDBC에 groovy.sql패키지를 올려 놓은 곳이다.

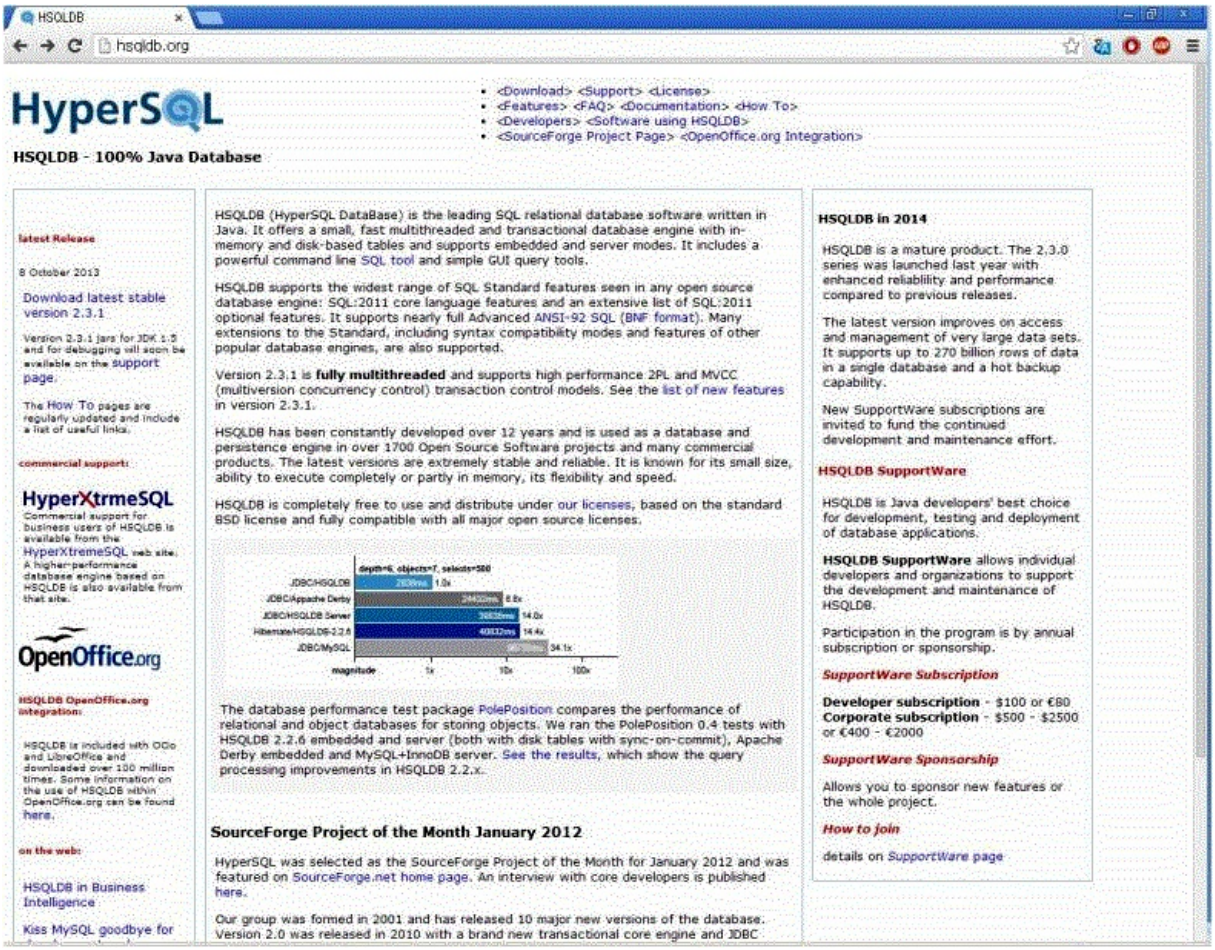
데이터베이스 기본조작

1. 데이터 베이스 연결
2. 데이터 베이스 스키마 작성
3. 샘플 데이터 처리

C,R,U,D를 해보자. 또한 메타데이터 가지고 유연하게 프로그래밍을 해보자.

데이터 베이스 연결

데이터베이스 설치 (Hypersonic데이터 베이스(HSQLDB)를 사용해보자
http://hsqldb.org 에서 받을수 있다. (HSQL은 파일베이스,메모리베이스 DBMS입니다)



에서 위측top메뉴에 Download latest stable version 2.3.1 받아보자
https://sourceforge.net/projects/hsqldb/files/

HSQLDB 실행

명령 프롬프트 창을 하나 띄우고 다음과 같이 입력합니다.

java -classpath ./hsqldb.jar org.hsqldb.Server -database.0 file:test -dbname.0 javaworld

"-database.0 file:data/test" 옵션은 Database가 사용할 경로와 파일명을 명시하고 있습니다. 예에서는 상대경로로 현재 폴더 아래의 data 폴더에 "test.*" 형식으로 관련 파일을 생성하라고 얘기하고 있습니다. 물론 절대경로로 지정해도 됩니다.
"-dbname.0 javaworld" 옵션은 Databse의 alias(별칭)을 지정하고 있습니다. 위의 경우는 'javaworld'라는 이름으로 Database의 이름을 지정하고 있는거죠. 이 alias는 어플리케이션에서 jdbc를 통해 hsqldb에 접근할 때 사용하게 됩니다. 위의 경우 jdbc connection URL은 "jdbc:hsqldb:hsql://localhost/javaworld"입니다.

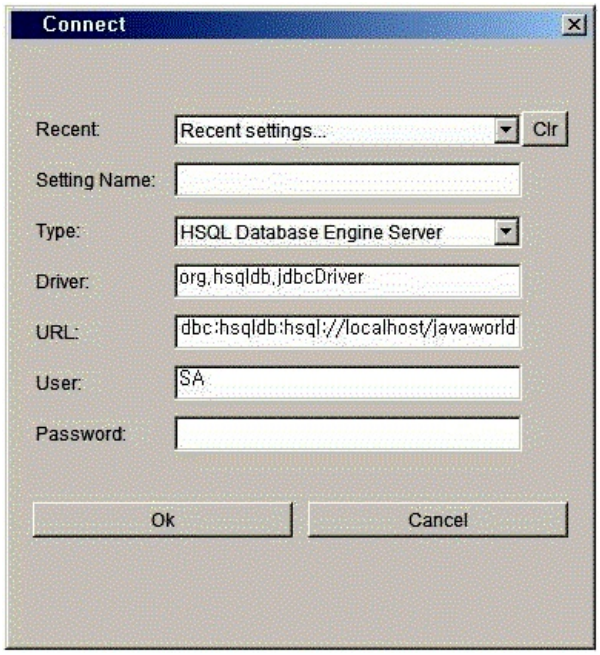
각 옵션명의 뒤쪽에 붙어있는 ".0" 이라는 문자는 database의 인덱스를 의미합니다. hsqldb는 0~9까지의 인덱스 값을 허용하므로, 총 10개의 Database를 동시에 띄울 수 있습니다.
즉 옵션 "-database.0 file:data/test -dbname.0 javaworld"은 인덱스 값 0에 할당된 Database가 현재 폴더 아래의 'data' 폴더에 'test.*' 형식의 관련 파일들을 사용하며, alias(dbname)은 javaworld라는 것을 의미합니다. 위의 실행 예에 녹색으로 표시된 부분을 보면 이해에 조금 더 도움이 될겁니다.

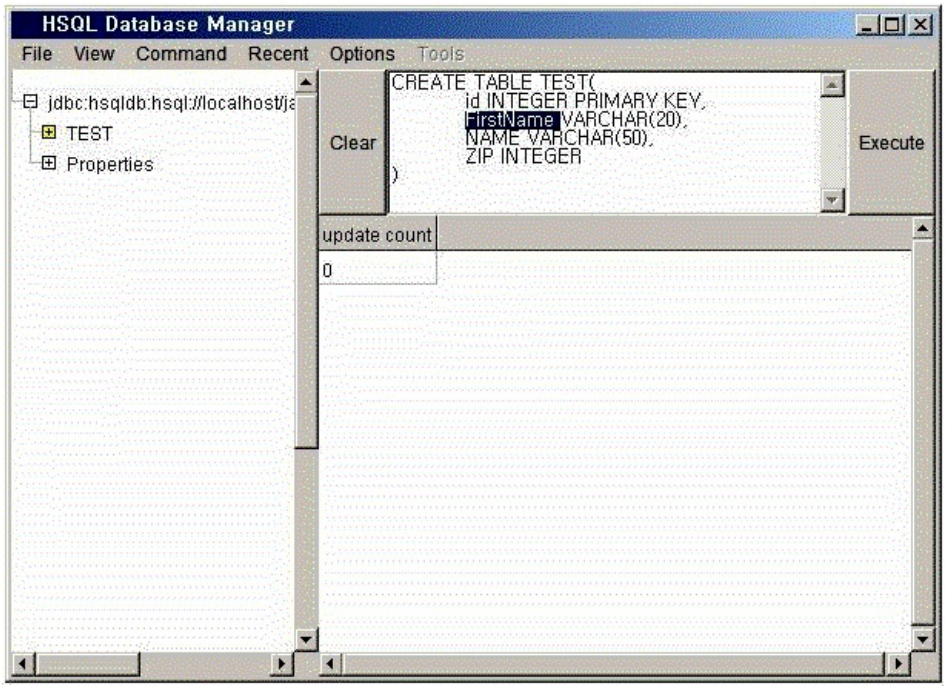
Database Manager

Database Manager는 HSQLDB Database에 질의를 하거나 테이블 등의 Database 객체를 볼 수 있게 해주는 GUI 툴입니다.
Database Manager는 hsqldb.jar 파일에 포함 되어 있기 때문에 아래와 같이 명령 프롬프트에서 입력하면 바로 사용이 가능합니다.

java -classpath lib/hsqldb.jar org.hsqldb.util.DatabaseManager

W:₩code.google₩groovy-visualkhh₩groovy_study₩db₩hsqldb-2.3.1₩hsqldb-2.3.1₩hsqldb₩lib>java -classpath ./hsqldb.jar org.hsqldb.util.DatabaseManager





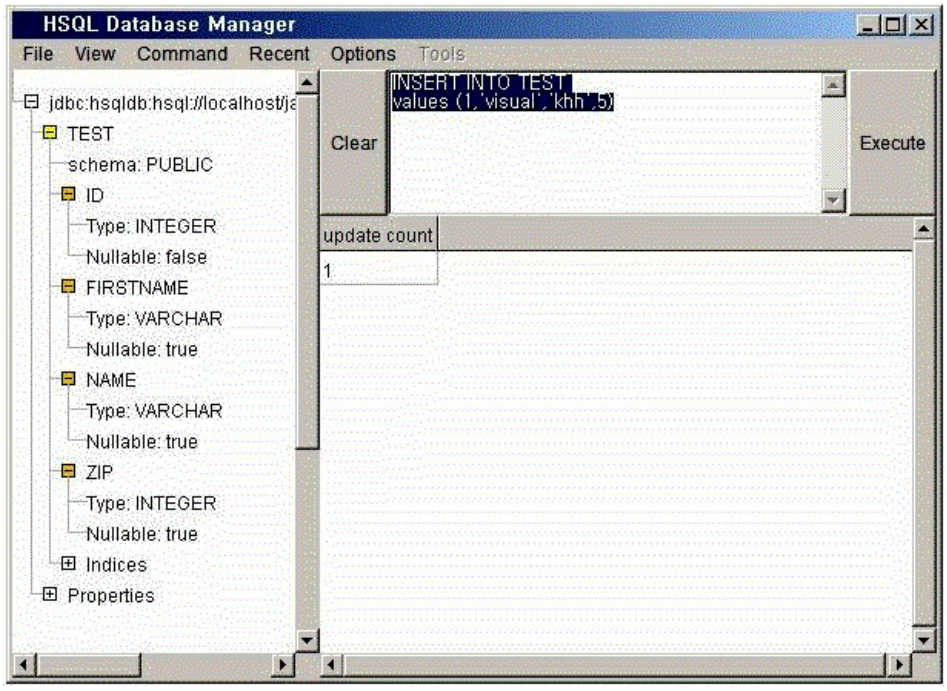

```
1  CREATE TABLE TEST(
2      id INTEGER PRIMARY KEY,
3      FirstName VARCHAR(20),
4      NAME VARCHAR(50),
5      ZIP INTEGER
6  )
```

DDL이나 DML을 입력한 후 Execute 버튼을 누르면(단축키 : Ctrl-Enter) 입력한 SQL문이 실행됩니다.
위와 같이 DDL 문을 실행했을 경우에는 좌측의 트리가 자동으로 갱신되지 않는데, 이럴 때는 View-Refresh Tree 메뉴를 한 번 클릭해 주면 갱신된 트리를 볼 수 있습니다.

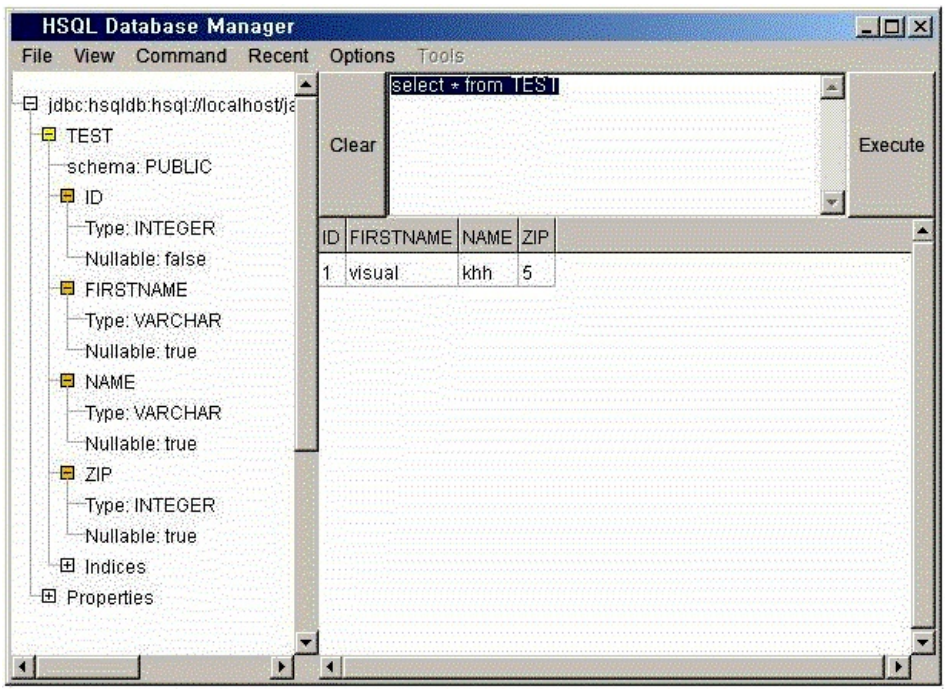
HSQLDB 서브프로토콜

URL 패턴	목적

jdbc:hsqldb:hsql://localhost/dbname	HSQLDB서버프로세스에 접속 여러 클라이언트나 프로세스에서 데이터 베이스 공유할때사용
jdbc:hsqldb:file:«database/path?»	단독 클라이언트로 HSQLDB 파일에 접속 데이터 베이스가 아직 없을 경우 dbname으로 시작하는 파일들이 여러 개 생성된다
jdbc:hsqldb:mem:dbname	메모리에만 생성되고 디스크에 저장되지 안흔 데이터 접속



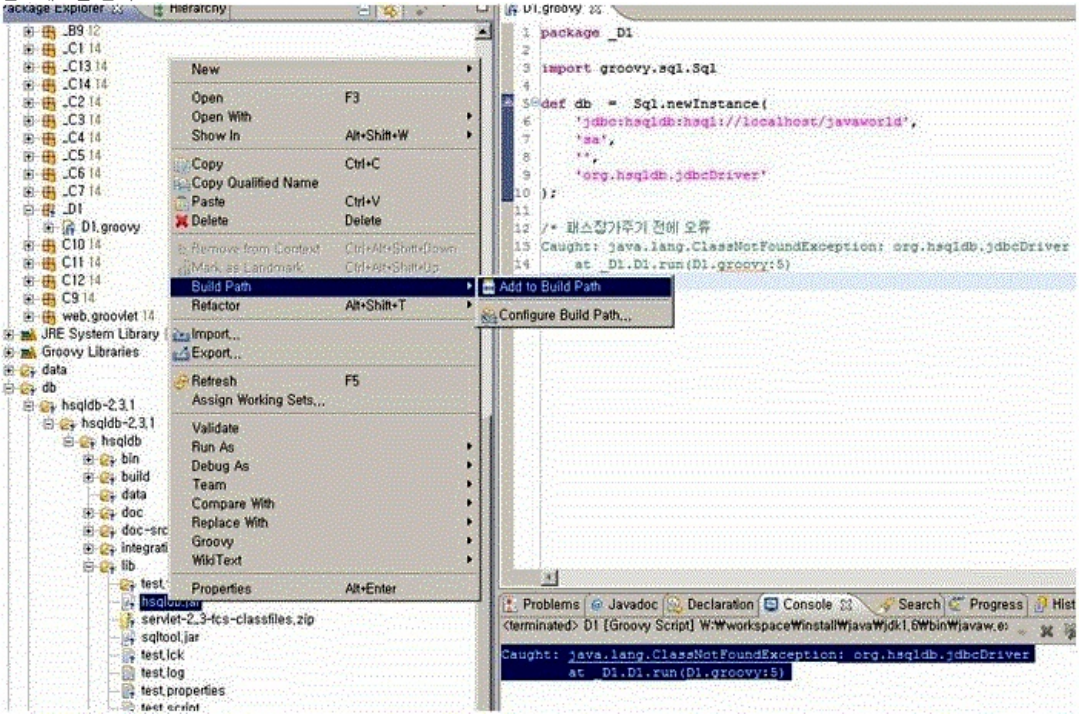

```
1 INSERT INTO TEST
2 values (1,'visual','khh',5)
```




```
1 select * from TEST
```

그루비에서 데이터 베이스접속.

빌드패스를 잡자



hsqldb의 드라이브 패스는 org.hsqldb.jdbcDriver다.
그루비에서 데이터 베이스 접속은 groovy.sql.Sql 객체를 이용한다.
이객체는 Sql의 팩토리 메서드인 newIntance에 앞의 정보를 인자로 전달받는다.

```
1  package _D1
2
3  import groovy.sql.Sql
4
5  //드라이버 매니저 이용
6  def db  =  Sql.newInstance(
7      'jdbc:hsqldb:hsql://localhost/javaworld',
8      'sa',
9      '',
10     'org.hsqldb.jdbcDriver'
11 );
12
13 /* 패스잡가주기 전에 오류
14 Caught: java.lang.ClassNotFoundException: org.hsqldb.jdbcDriver
15     at _D1.D1.run(D1.groovy:5)
16 */
```

드라이버 매너저와 데이터 소스

구분	내용
드라이버 매니저	Sql.newIntance메서드는 드라이버 매니저를 이용해서 동작하는데 고전적인 저수진 연결 방식이라고 생각할수도 있다
데이터 소스	1.4부터는 데이터 소스 라는 새로운 개념을 이용하는 다른 연결 방식을 사용할수 있다.데이터 소스는 커넥션풀, 분선트랜잭션도 지원한다. JNDI(Java Naming and Directory Insterface)를 통해 데이터소스 를 얻을수도있다.

데이터소스 사용하기

데이터베이트 공급사 에서 javax.sql.DataSource인터페이스를 구현한 클래스를 제공한다

디비	데이터소스
HSQLDB	org.hsqldb.jdbc.jdbcDataSource

데이터 소스 객체를 생성후 프로퍼티를 설정한다음 Sql클래스의 생성자에 전달하면된다.

```
package _D1

import groovy.sql.Sql
//데이터 소스 이용 커넥션
def source = new org.hsqldb.jdbc.JDBCDataSource();
source.database = 'jdbc:hsqldb:hsql://localhost/javaworld';
source.user = 'sa';
source.password = '';
def db = new groovy.sql.Sql(source);
def db2 = new Sql(db); //또하나의 커넥션이 만들어진다.
```

SQL실행하기.

그루비에서는 커넥션을 얻고, 명령문(statement)을 생성,설정, 전송하고, 오류를 로깅하고, 올바르게 명령문이나 연결등을 리소스를 닫는 작업을 해준다.

```
db.execute(statement)
```

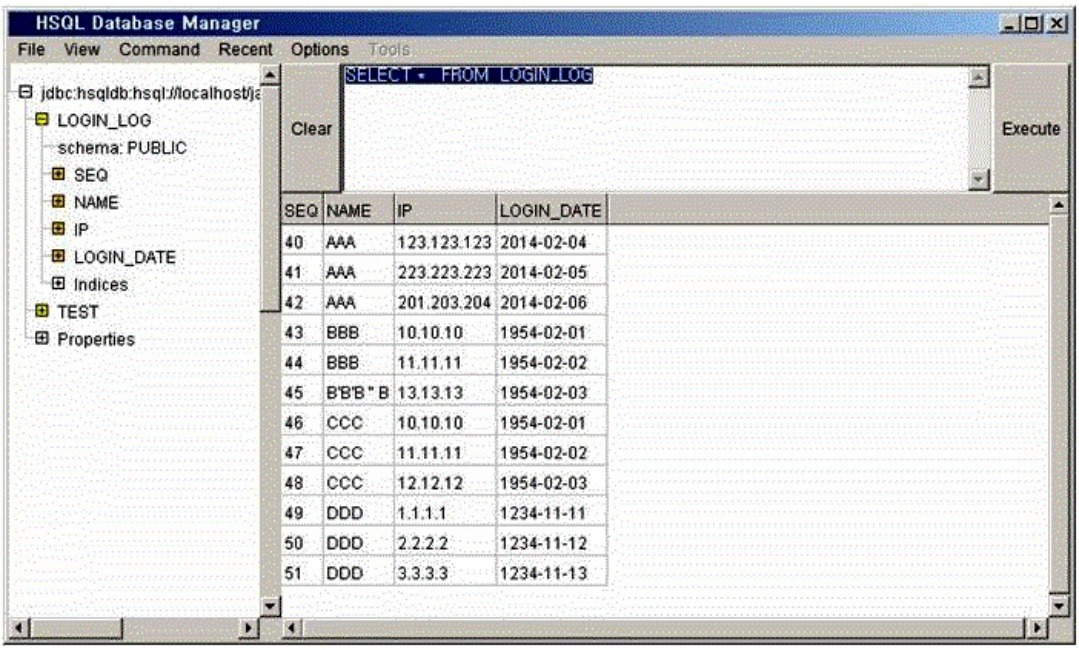
데이터 베이스 스키마 생성하기

```
package _D1

import groovy.sql.Sql
//데이터 소스 이용 커넥션
def source = new org.hsqldb.jdbc.JDBCDataSource();
source.database = 'jdbc:hsqldb:hsql://localhost/javaworld';
source.user = 'sa';
source.password = '';
def db = new groovy.sql.Sql(source);
//db.execute '''
//CREATE TABLE LOGIN_LOG (
//NAME VARCHAR(64),
//IP VARCHAR(64),
//LOGIN_DATE DATE
//);
//''';


//db.execute '''
//CREATE TABLE LOGIN_LOG(
//SEQ INTEGER GENERATED BY DEFAULT AS IDENTITY,
//NAME VARCHAR(64),
//IP VARCHAR(64),
//LOGIN_DATE DATE
//);
//CREATE INDEX SEQIdx ON LOGIN_LOG (SEQ);
//''';
/* 이미  TABLE있어서 오류.
CREATE INDEX SEQIDX ON LOGIN_LOG (SEQ);
 because: user lacks privilege or object not found: SEQ
Caught: java.sql.SQLSyntaxErrorException: user lacks privilege or object not found: SEQ
    at _D1.D1_2.run(D1_2.groovy:22)
*/


db.execute '''
    DROP INDEX SEQIDX IF EXISTS;
    DROP TABLE LOGIN_LOG IF EXISTS;
    CREATE TABLE LOGIN_LOG(
    SEQ INTEGER GENERATED BY DEFAULT AS IDENTITY,
    NAME VARCHAR(64),
    IP VARCHAR(64),
    LOGIN_DATE DATE
    );
    CREATE INDEX SEQIdx ON LOGIN_LOG (SEQ);
''';
```

데이터 집어넣기

```
package _D1


import java.util.logging.Logger;

import groovy.sql.Sql
```

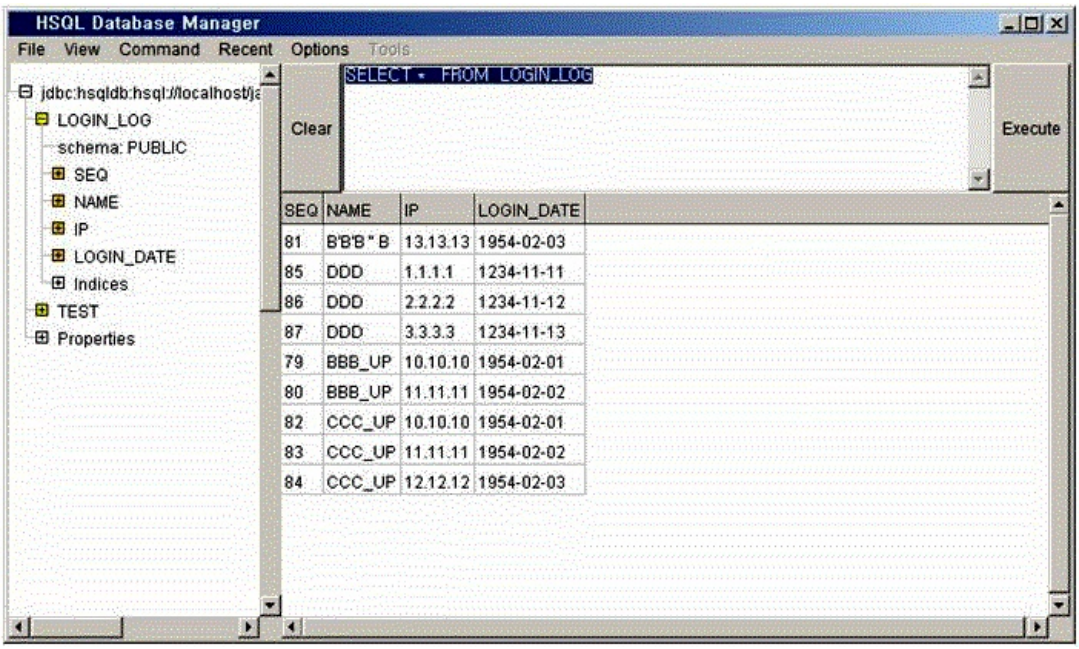
```groovy
import groovy.sql.Sql;


//데이터 소스 이용 커넥션
def source = new org.hsqldb.jdbc.JDBCDataSource();
source.database = 'jdbc:hsqldb:hsql://localhost/javaworld';
source.user = 'sa';
source.password = '';
def db = new groovy.sql.Sql(source);



db.execute '''
INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
    VALUES('AAA','123.123.123','2014-02-04');
INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
    VALUES('AAA','223.223.223','2014-02-05');
INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
    VALUES('AAA','201.203.204','2014-02-06');
'''

/////////prepared statement 제공////////
/*
 주의 prepared statement  에서는 '(작은따옴표) 쓰지말아야 한다.
명령문 쪽에서도 없어야하고 명령에 전달되는 문자열 내부에도 없어야 한다
작은따옴표는 그루비를 위한것이지 SQL을 위한것이 아니다.
*/
String sql = '''
INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
    VALUES(?,?,?);
    '''
db.execute (sql,['BBB','10.10.10','1954-02-01']);
db.execute (sql,['BBB','11.11.11','1954-02-02']);
db.execute (sql,['B'B'B " B','13.13.13','1954-02-03']);

////////////////////////////////////////
sql = '''
INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
    VALUES('CCC',?,?);
    '''
db.execute (sql,['10.10.10','1954-02-01']);
db.execute (sql,['11.11.11','1954-02-02']);
db.execute (sql,['12.12.12','1954-02-03']);


////////////////////////////////////

def loglist = [
    [NAME:'DDD',    IP:'1.1.1.1',    LOGIN_DATE:'1234-11-11'],
    [NAME:'DDD',    IP:'2.2.2.2',    LOGIN_DATE:'1234-11-12'],
    [NAME:'DDD',    IP:'3.3.3.3',    LOGIN_DATE:'1234-11-13']
];

//로그를 찍어보장~
//Logger.getLogger ('groovy.sql').level = Level.FINE;

loglist.each { inRow ->
    db.execute """
    INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
        VALUES(${inRow.NAME},${inRow.IP},${inRow.LOGIN_DATE});
    """
}
```



데이터 수정하고 삭제하기

수정과 삭제도 execute메서드로 실행한다

```
package _D1

//데이터 소스 이용 커넥션
def source = new org.hsqldb.jdbc.JDBCDataSource();
source.database = 'jdbc:hsqldb:hsql://localhost/javaworld';
source.user = 'sa';
source.password = '';
def db = new groovy.sql.Sql(source);


db.execute '''
DELETE FROM LOGIN_LOG WHERE NAME ='AAA'
'''

db.execute '''
UPDATE LOGIN_LOG SET NAME='BBB_UP' WHERE NAME ='BBB'
'''


def updateCnt = db.executeUpdate ('''
UPDATE LOGIN_LOG SET NAME='CCC_UP' WHERE NAME ='CCC'
''')
println 'WHERE NAME ='CCC'  updateCnt : '+updateCnt

/* 결과
WHERE NAME ='CCC'  updateCnt : 3
*/
```



execute 메서드의 다른 버전으로 executeUpdate 메서드가 있다 이 메서드는 동작하는 방식은 같고 리턴 값만 틀리다 executeUpdate는 명령에 의해 변경된 행의 수를 리턴한다 java.sqlPreparedStatement의 API문서에 상세한 설명이 있다

리턴	메서드이름	인자
boolean	execute	String statement
boolean	execute	String prepStmt,List values
boolaen	execute	GString prepStmt
int	exeucteUpdate	String statement
int	exeucteUpdate	String prepStmt,List values
int	exeucteUpdate	GString prepStmt

데이터 조회

데이터베이스에서 데이터를 읽어 들이는 작업은 기존과비슷하다 Sql객체의 메서드로 수행한다

리턴	메서드이름	인자
void	eachRow	String statement {row->code}
void	eachRow	String prepStmt, List values {row->code}
void	eachRow	GString prepStmt {row->code}
void	query	String statement{resultSet->code}
void	query	String prepStmt, List values{resultSet->code}
void	query	GString prepStmt {resultSet->code}
List	rows	String statement
List	rows	String prepStmt,List values
Object	firstRow	String statement
Object	firstRow	String prepStmt,List values

eachRow 메서드와 query 메서드는 결과를 처리하기 위해 클로저를 사용한다

1. query 메서드는 클로저를 한번만 호출 하면서 전체 ResultSet을 전달
2. eachRow 메서드는 조회 결과를 한행씩 클로저에 전달 반복 작업한다.

```
package _D1

//데이터 소스 이용 커넥션
def source = new org.hsqldb.jdbc.JDBCDataSource();
source.database = 'jdbc:hsqldb:hsql://localhost/javaworld';
source.user = 'sa';
source.password = '';
def db = new groovy.sql.Sql(source);

//Row 가져오기
db.eachRow('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG'){
    atRow->
    println "SEQ : ${atRow.SEQ}, NAME : ${atRow.NAME}, IP : ${atRow.IP},  LOGIN_DATE : ${atRow.LOGIN_DATE}";
    println "SEQ : ${atRow[0]}, NAME : ${atRow[1]}, IP : ${atRow[2]},  LOGIN_DATE : ${atRow[3]}";
    println '--'*25;
}

println '**'*25;

//ResultSet 가져오기
db.query('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG'){
    resultSet->
    if(resultSet.next()){
        print resultSet.getString(1); //여기서 인덱스는 0부터 시작이 아니라 1부터 시작이다.
        print '    '
        println resultSet.getString('NAME');
    }
}

println '**'*25;
//rows메서드를 이용하면 모든 행을 하나의 긴 리스트에 담을수 있다.
List list = db.rows('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG')
println list;

println '**'*25;
//메타데이터 가져오기.
db.query('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG'){
    resultSet->
        def meta = resultSet.metaData;
        if(meta.columnCount<=0){
            return;
        }
        while(resultSet.next()){
            for(i in 0..<meta.columnCount){
                print "${i} : ${meta.getColumnLabel(i+1)}".padRight(20);
                print resultSet.getObject(i+1);
                print 'n'
            }
            println '-'*40;
        }
}


/*
결과
SEQ : 81, NAME : B'B'B " B, IP : 13.13.13,  LOGIN_DATE : 1954-02-03
SEQ : 81, NAME : B'B'B " B, IP : 13.13.13,  LOGIN_DATE : 1954-02-03
--------------------------------------------------
SEQ : 85, NAME : DDD, IP : 1.1.1.1,  LOGIN_DATE : 1234-11-11
SEQ : 85, NAME : DDD, IP : 1.1.1.1,  LOGIN_DATE : 1234-11-11
--------------------------------------------------
SEQ : 86, NAME : DDD, IP : 2.2.2.2,  LOGIN_DATE : 1234-11-12
SEQ : 86, NAME : DDD, IP : 2.2.2.2,  LOGIN_DATE : 1234-11-12
--------------------------------------------------
SEQ : 87, NAME : DDD, IP : 3.3.3.3,  LOGIN_DATE : 1234-11-13
SEQ : 87, NAME : DDD, IP : 3.3.3.3,  LOGIN_DATE : 1234-11-13
```

```
----------------------------------------------
SEQ : 79, NAME : BBB_UP, IP : 10.10.10,  LOGIN_DATE : 1954-02-01
SEQ : 79, NAME : BBB_UP, IP : 10.10.10,  LOGIN_DATE : 1954-02-01
----------------------------------------------
SEQ : 80, NAME : BBB_UP, IP : 11.11.11,  LOGIN_DATE : 1954-02-02
SEQ : 80, NAME : BBB_UP, IP : 11.11.11,  LOGIN_DATE : 1954-02-02
----------------------------------------------
SEQ : 82, NAME : CCC_UP, IP : 10.10.10,  LOGIN_DATE : 1954-02-01
SEQ : 82, NAME : CCC_UP, IP : 10.10.10,  LOGIN_DATE : 1954-02-01
----------------------------------------------
SEQ : 83, NAME : CCC_UP, IP : 11.11.11,  LOGIN_DATE : 1954-02-02
SEQ : 83, NAME : CCC_UP, IP : 11.11.11,  LOGIN_DATE : 1954-02-02
----------------------------------------------
SEQ : 84, NAME : CCC_UP, IP : 12.12.12,  LOGIN_DATE : 1954-02-03
SEQ : 84, NAME : CCC_UP, IP : 12.12.12,  LOGIN_DATE : 1954-02-03
----------------------------------------------
**************************************************
81    B'B'B " B
**************************************************
[[SEQ:81, NAME:B'B'B " B, IP:13.13.13, LOGIN_DATE:1954-02-03], [SEQ:85, NAME:DDD, IP:1.1.1.1, LOGIN_DATE:1234-11-11], [SEQ:8
**************************************************
0 : SEQ            81
1 : NAME           B'B'B " B
2 : IP             13.13.13
3 : LOGIN_DATE     1954-02-03
----------------------------------------
0 : SEQ            85
1 : NAME           DDD
2 : IP             1.1.1.1
3 : LOGIN_DATE     1234-11-11
----------------------------------------
0 : SEQ            86
1 : NAME           DDD
2 : IP             2.2.2.2
3 : LOGIN_DATE     1234-11-12
----------------------------------------
0 : SEQ            87
1 : NAME           DDD
2 : IP             3.3.3.3
3 : LOGIN_DATE     1234-11-13
----------------------------------------
0 : SEQ            79
1 : NAME           BBB_UP
2 : IP             10.10.10
3 : LOGIN_DATE     1954-02-01
----------------------------------------
0 : SEQ            80
1 : NAME           BBB_UP
2 : IP             11.11.11
3 : LOGIN_DATE     1954-02-02
----------------------------------------
0 : SEQ            82
1 : NAME           CCC_UP
2 : IP             10.10.10
3 : LOGIN_DATE     1954-02-01
----------------------------------------
0 : SEQ            83
1 : NAME           CCC_UP
2 : IP             11.11.11
3 : LOGIN_DATE     1954-02-02
----------------------------------------
0 : SEQ            84
1 : NAME           CCC_UP
2 : IP             12.12.12
3 : LOGIN_DATE     1954-02-03
----------------------------------------
*/
```

토탈.

```groovy
package _D1

import groovy.sql.Sql


dbHandle = null // XXX todo explain why Sql is missing

def getDb(){                                        //#1
    if (dbHandle) return dbHandle
    def source = new org.hsqldb.jdbc.jdbcDataSource()
    source.database = 'jdbc:hsqldb:mem:GIA'
    source.user = 'sa'
    source.password = ''
    dbHandle = new Sql(source)
    return dbHandle
}
def reset() {                                       //#2
    db.execute '''
        DROP   INDEX athleteIdx IF EXISTS;
        DROP   TABLE Athlete IF EXISTS;
        CREATE TABLE Athlete (
            athleteId   INTEGER GENERATED BY DEFAULT AS IDENTITY,
            firstname   VARCHAR(64),
            lastname    VARCHAR(64),
```

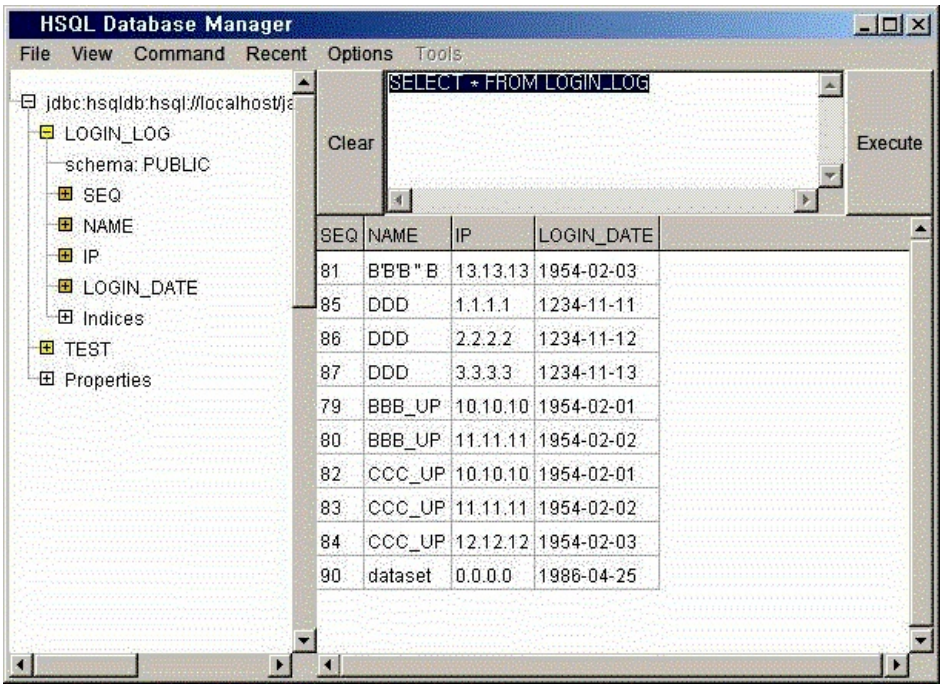
```
            dateOfBirth DATE
        );
        CREATE INDEX athleteIdx ON Athlete (athleteId);
    '''
}
def create(firstname, lastname, dateOfBirth) {          //#3
    db.execute """
        INSERT INTO Athlete ( firstname, lastname, dateOfBirth)
                VALUES ($firstname,$lastname,$dateOfBirth);
    """
}
def findAll() {                                         //#4
    db.rows 'SELECT * FROM Athlete'
}
def updateFirstName(wrong, right) {                     //#5
    db.execute """
        UPDATE Athlete
          SET firstname = $right WHERE firstname = $wrong;
    """
}
def delete(firstname) {                                 //#6
    db.execute "DELETE FROM Athlete WHERE firstname = $firstname;"
}

reset()
assert ! findAll(), 'we are intially empty'
create 'Dirk', 'Koenig', '1968-04-19'
assert 'Dirk'  == findAll()[0].firstname                //#7
updateFirstName  'Dirk', 'Dierk'
assert 'Dierk' == findAll()[0].firstname
delete 'Dierk'
assert ! findAll(), 'after delete, we are empty again'
```

DataSet로 SQL 없이 SQL사용하기

그루비는 sql을 사용하지 않으면서 데이터 베이스를 처리할 수 있는 방법이있다.
DataSet이라는 개념이다.
1. 테이블 행추가
2. 테이블이나 뷰 모든행 처리하기
3. 단순한 표현식으로 테이블이나 뷰 조회하기

```
package _D2

import groovy.sql.DataSet

def source = new org.hsqldb.jdbc.JDBCDataSource();
source.database = 'jdbc:hsqldb:hsql://localhost/javaworld';
source.user = 'sa';
source.password = '';
def db = new groovy.sql.Sql(source);


// 테이블에 새로운 행을 추가할수있다.
DataSet login_log = db.dataSet("LOGIN_LOG");
login_log.add(
    name:"dataset",
    ip:"0.0.0.0",
    login_date:"1986-04-25",
);

println '---each----'
login_log.each {
    println it.name+'   '+it.ip+'    '+it.login_date;
}

println '---findAll--where name=dataset--'
//특정 조건으로찾은 행들  DataSet를 리턴한다
//여기서 데이터를 다가져온후 클로저로 일일이 걸러내는것같이 할것같지만
//사실은 findAll은  클로저에 제시된 표현식으로 sql을 만들어서 실행한다.
//each할때 쿼리실행이된다.
DataSet login_log_where1 = login_log.findAll {
     it.name=='dataset';
}
login_log_where1.each {
    println it.name+'   '+it.ip+'    '+it.login_date;
}
println 'login_log_where1 sql : '+login_log_where1.sql;
println 'login_log_where1 parameters : '+login_log_where1.parameters;




DataSet login_log_where2 = login_log.findAll {
    it.name=='dataset' && it.ip=='0.0.0.0';
}
login_log_where2.each {
   println it.name+'   '+it.ip+'    '+it.login_date;
}
```

```
println 'login_log_where2 sql : '+login_log_where2.sql;
println 'login_log_where2 parameters : '+login_log_where2.parameters;

/*
 &&                ->       AND
 ||                ->       OR
 ==                ->       =
 다른연산자          ->       그대로적용
 it.propertyname    ->       프로퍼티 이름을 필드로 보고 그대로적용
 상수                ->       ? 표현식은 인자 리스트에 들어간다
 */

/* 결과
---each----
B'B'B " B   13.13.13    1954-02-03
DDD   1.1.1.1    1234-11-11
DDD   2.2.2.2    1234-11-12
DDD   3.3.3.3    1234-11-13
BBB_UP   10.10.10    1954-02-01
BBB_UP   11.11.11    1954-02-02
CCC_UP   10.10.10    1954-02-01
CCC_UP   11.11.11    1954-02-02
CCC_UP   12.12.12    1954-02-03
dataset   0.0.0.0    1986-04-25
---findAll--where name=dataset--
dataset   0.0.0.0    1986-04-25
login_log_where1 sql : select * from LOGIN_LOG where name = ?
login_log_where1 parameters : [dataset]
dataset   0.0.0.0    1986-04-25
login_log_where2 sql : select * from LOGIN_LOG where (name = ? and ip = ?)
login_log_where2 parameters : [dataset, 0.0.0.0]


 */
```



AST 노드	SQL
&&	AND
||	OR
==	=
다른연산자들	그대로 적용
it.propertname	프로퍼티 이름을 필드로 보고 그대로적용
tkdtn	? 표현식은 인자 리스트에 들어간다