



02. 5주차. 10장 그루비 데이터베이스 프로그래밍

Table of Contents

- 02. 5주차. 10장 그루비 데이터베이스 프로그래밍
- 그루비 데이터 베이스 프로그래밍 343
- 데이터베이스 기본조작
- 데이터 베이스 연결
- HSQLDB 실행
- Database Manager
- HSQLDB 서브프로토콜
- 그루비에서 데이터 베이스 접속.
- 드라이버 매너저와 데이터 소스
- 데이터소스 사용하기
- SQL 실행하기.
- 데이터 베이스 스키마 생성하기
- 데이터 집어넣기
- 데이터 수정하고 삭제하기
- 데이터 조회
- eachRow 메서드와 query 메서드는 결과를 처리하기 위해 클로저를 사용한다
- 토달.
- DataSet로 SQL 없이 SQL 사용하기

그루비 데이터 베이스 프로그래밍 343

자바에서는 JDBC(java database connectivity) 통해서 접근해야한다.
그루비는 어떨까? JDBC에 groovy.sql 패키지를 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 (Hypersonic 데이터 베이스(HSQLDB)를 사용해보자
<http://hsqldb.org> 에서 받을 수 있다. (HSQL은 파일베이스, 메모리베이스 DBMS입니다)

HSQldb.org

HyperSQL

HSQldb - 100% Java Database

- <Download> <Support> <License>
- <Features> <FAQ> <Documentation> <How To>
- <Developers> <Software using HSQldb>
- <SourceForge Project Page> <OpenOffice.org Integration>

latest Release

8 October 2013

Download latest stable version 2.3.1

Version 2.3.1 jars for JDK 1.5 and for debugging will soon be available on the [Support page](#).

The [HOW TO](#) pages are regularly updated and include a list of useful links.

commercial support:

HyperXtremeSQL

Commercial support for business users of HSQldb is available from the [HyperXtremeSQL web site](#). A higher-performance database engine based on HSQldb is also available from that site.

OpenOffice.org

HSQldb OpenOffice.org integration:

HSQldb is included with OOo and LibreOffice and downloaded over 100 million times. Some information on the use of HSQldb within OpenOffice.org can be found [here](#).

on the web:

[HSQldb in Business Intelligence](#)

[Kiss MySQL goodbye for](#)

HSQldb (HyperSQL DataBase) is the leading SQL relational database software written in Java. It offers a small, fast multithreaded and transactional database engine with in-memory and disk-based tables and supports embedded and server modes. It includes a powerful command line SQL tool and simple GUI query tools.

HSQldb supports the widest range of SQL Standard features seen in any open source database engine: SQL:2011 core language features and an extensive list of SQL:2011 optional features. It supports nearly full Advanced ANSI-92 SQL (BNF format). Many extensions to the Standard, including syntax compatibility modes and features of other popular database engines, are also supported.

Version 2.3.1 is **fully multithreaded** and supports high performance 2PL and MVCC (multiversion concurrency control) transaction control models. See the [list of new features](#) in version 2.3.1.

HSQldb has been constantly developed over 12 years and is used as a database and persistence engine in over 1700 Open Source Software projects and many commercial products. The latest versions are extremely stable and reliable. It is known for its small size, ability to execute completely or partly in memory, its flexibility and speed.

HSQldb is completely free to use and distributes under our licenses, based on the standard BSD license and fully compatible with all major open source licenses.

The database performance test package [PolePosition](#) compares the performance of relational and object databases for storing objects. We ran the PolePosition 0.4 tests with HSQldb 2.2.6 embedded and server (both with disk tables with sync-on-commit), Apache Derby embedded and MySQL+InnoDB server. See the [results](#), which show the query processing improvements in HSQldb 2.2.x.

SourceForge Project of the Month January 2012

HyperSQL was selected as the SourceForge Project of the Month for January 2012 and was featured on [SourceForge.net](#) home page. An interview with core developers is published [here](#).

Our group was formed in 2001 and has released 10 major new versions of the database. Version 2.0 was released in 2010 with a brand new transactional core engine and JDBC.

HSQldb in 2014

HSQldb is a mature product. The 2.3.0 series was launched last year with enhanced reliability and performance compared to previous releases.

The latest version improves on access and management of very large data sets. It supports up to 270 billion rows of data in a single database and a hot backup capability.

New SupportWare subscriptions are invited to fund the continued development and maintenance effort.

HSQldb SupportWare

HSQldb is Java developers' best choice for development, testing and deployment of database applications.

HSQldb SupportWare allows individual developers and organizations to support the development and maintenance of HSQldb.

Participation in the program is by annual subscription or sponsorship.

SupportWare Subscription

Developer subscription - \$100 or €80
Corporate subscription - \$500 - \$2500 or €400 - €2000

SupportWare Sponsorship

Allows you to sponsor new features or the whole project.

How to join

[details on SupportWare page](#)

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 Database의 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:hsq://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.

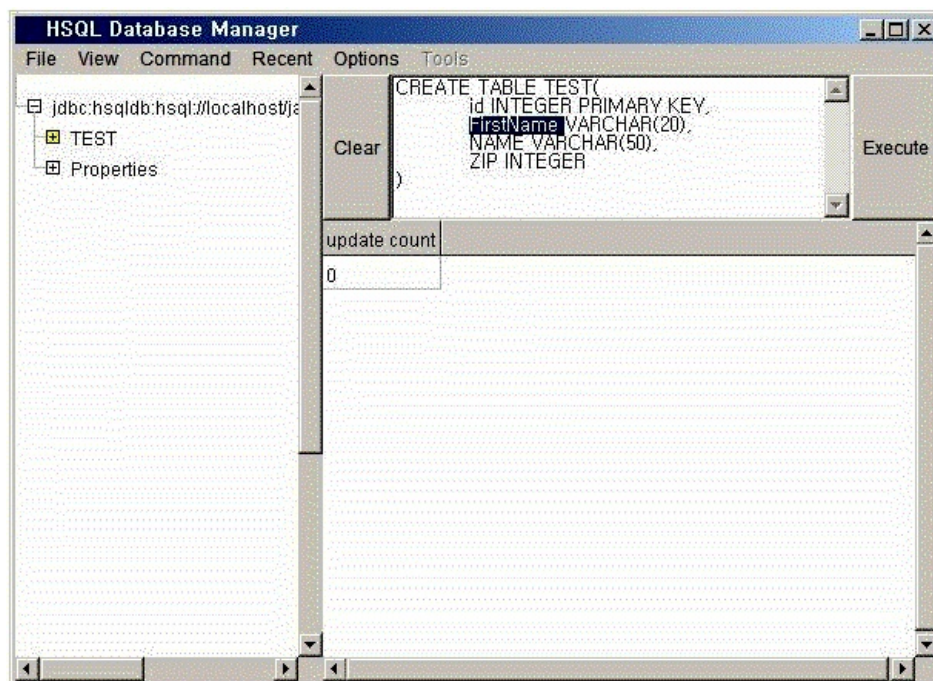
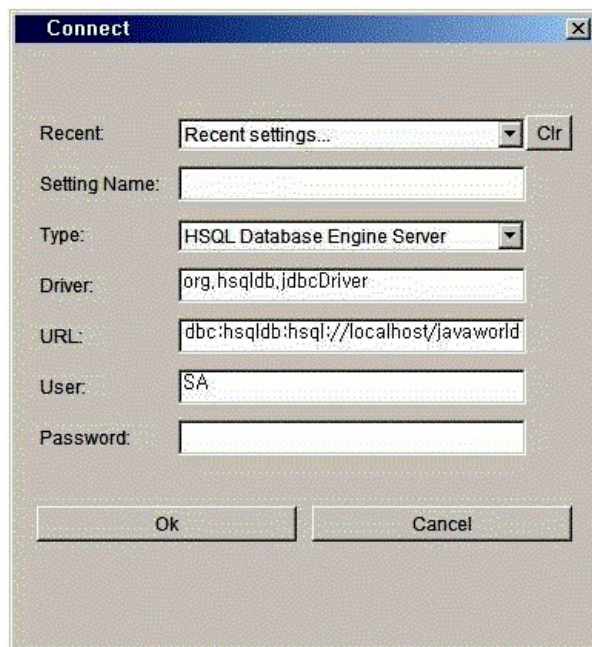
즉 옵션 "-database.0 file:data/test -dbname.0 javaworld"은 인덱스 값 0에 할당된 Database가 현재 폴더 아래의 'data' 폴더에 'test.*' 형식의 관련 파일들을 사용하며, alias(dbname)은 javaworld라는 것을 의미합니다. 위의 실행 예에 녹색으로 표시된 부분을 보면 이해에 조금 더 도움이 될겁니다.

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-visual\hh\groovy_study\Wdb\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 )

```

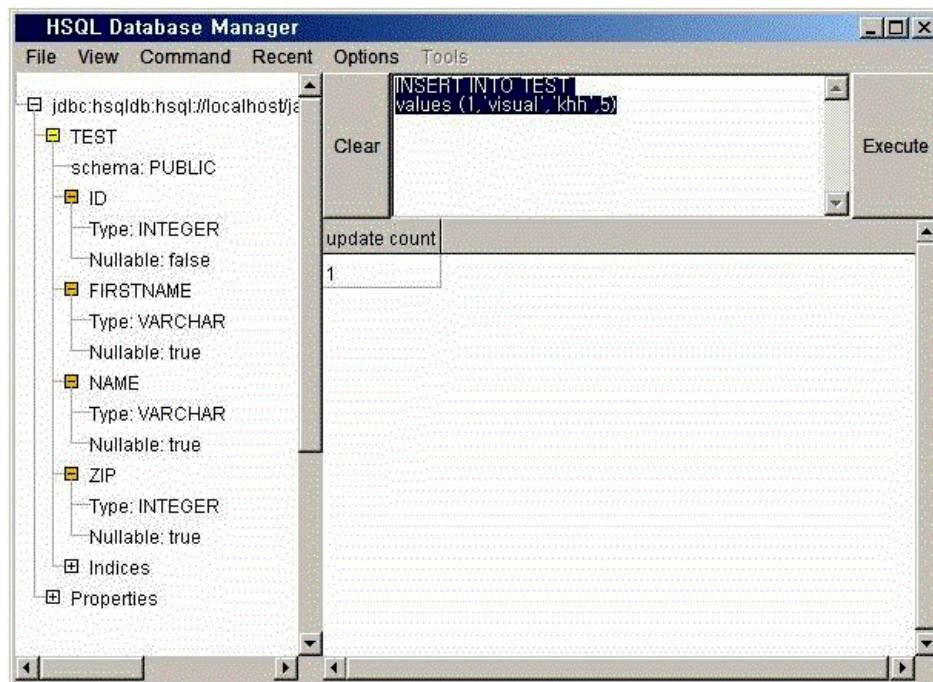


DDL이나 DML을 입력한 후 Execute 버튼을 누르면(단축키 : Ctrl-Enter) 입력한 SQL문이 실행됩니다.

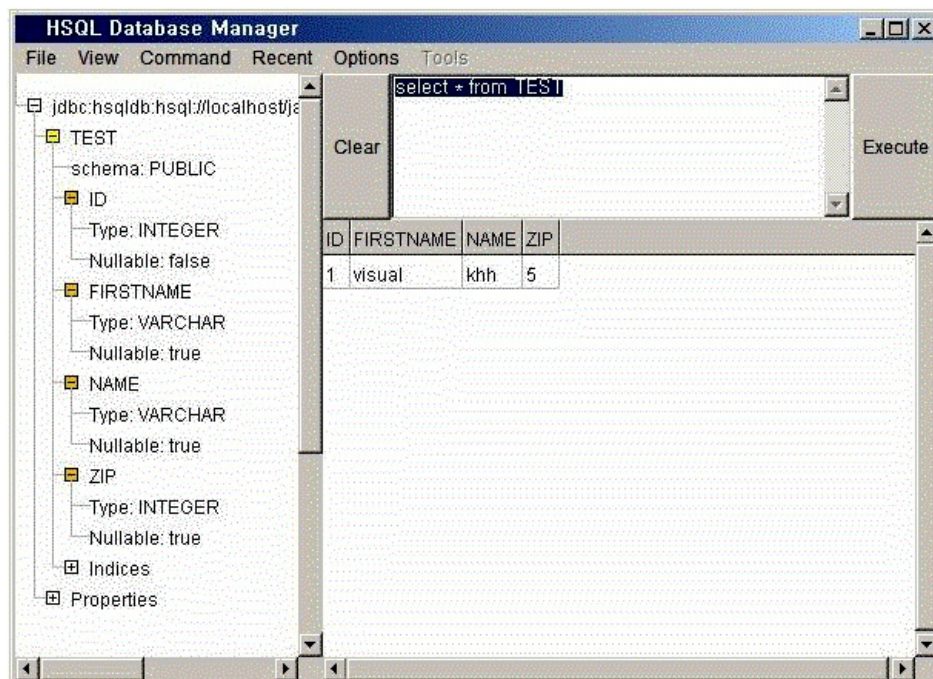
위와 같이 DDL 문을 실행했을 경우에는 좌측의 트리가 자동으로 갱신되지 않는데, 이럴 때는 View-Refresh Tree 메뉴를 한 번 클릭해 주면 갱신된 트리를 볼 수 있습니다.

HSQLDB 서브프로토콜

jdbc:hsqldb:hsqldb://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 않은 데이터 접속



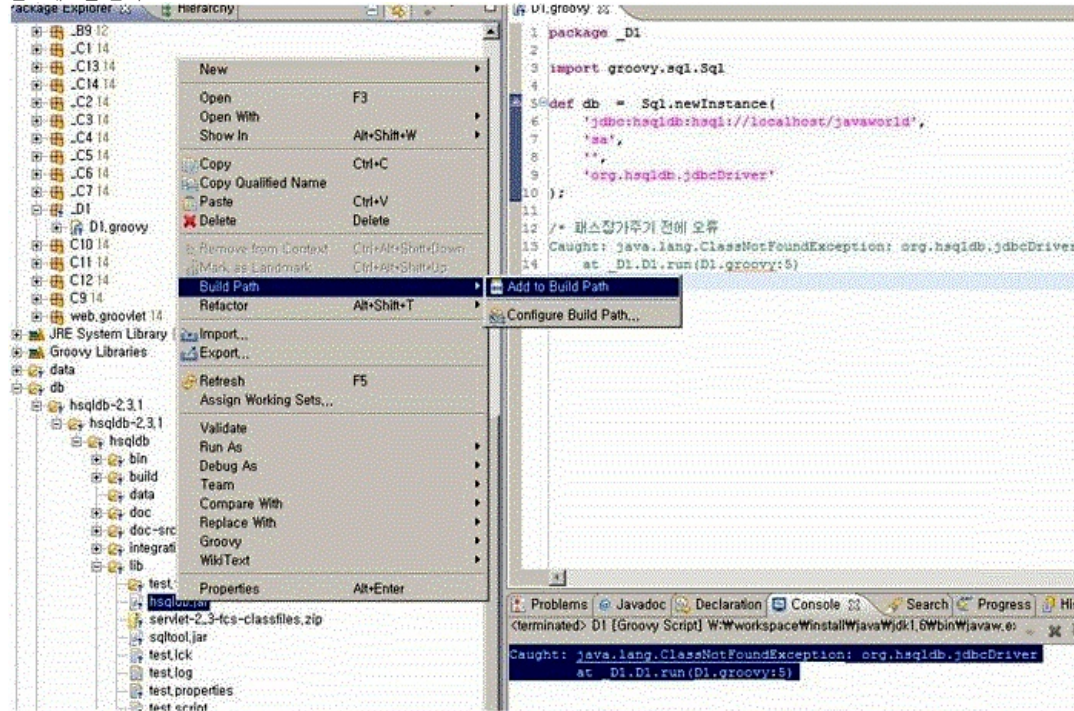
```
1 | INSERT INTO TEST
2 | values (1,'visual','khh',5)
```



```
1 | select * from TEST
```



빌드패스를 잡자



hsqldb의 드라이버 패스는 org.hsqldb.jdbcDriver다.
그루비에서 데이터 베이스 접속은 groovy.sql.Sql 객체를 이용한다.
이객체는 Sql의 팩토리 메서드인 newInstance에 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:hsqldb://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 매니저와 데이터 소스

구분	내용
드라이버 매니저	Sql.newInstance메서드는 드라이버 매니저를 이용해서 동작하는데 고전적인 저수준 연결 방식이라고 생각할수도 있다
데이터 소스	1.4부터는 데이터 소스 라는 새로운 개념을 이용하는 다른 연결 방식을 사용할수 있다.데이터 소스는 커넥션풀, 분산트랜잭션도 지원한다. JNDI(Java Naming and Directory Interface)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.

```
1 package _D1
2
3 import groovy.sql.Sql
4 //데이터 소스 이용 커넥션
5 def source = new org.hsqldb.jdbc.JDBCDataSource();
6 source.database = 'jdbc:hsqldb:hsq://localhost/javaworld';
7 source.user = 'sa';
8 source.password = '';
9 def db = new groovy.sql.Sql(source);
10 def db2 = new Sql(db); //또하나의 커넥션이 만들어진다.
```

SQL실행하기.

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

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

데이터 베이스 스키마 생성하기

```
1 package _D1
2
3 import groovy.sql.Sql
4 //데이터 소스 이용 커넥션
5 def source = new org.hsqldb.jdbc.JDBCDataSource();
6 source.database = 'jdbc:hsqldb:hsq://localhost/javaworld';
7 source.user = 'sa';
8 source.password = '';
9 def db = new groovy.sql.Sql(source);
10 //db.execute ""
11 //CREATE TABLE LOGIN_LOG (
12 //NAME VARCHAR(64),
13 //IP VARCHAR(64),
14 //LOGIN_DATE DATE
15 //);
16 //"";
17
18 //db.execute ""
19 //CREATE TABLE LOGIN_LOG(
20 //SEQ INTEGER GENERATED BY DEFAULT AS IDENTITY,
21 //NAME VARCHAR(64),
22 //IP VARCHAR(64),
23 //LOGIN_DATE DATE
24 //);
25 //";
26 //CREATE INDEX SEQIdx ON LOGIN_LOG (SEQ);
27 //"";
28 /* 이미 TABLE있어서 오류.
29 CREATE INDEX SEQIDX ON LOGIN_LOG (SEQ);
30 because: user lacks privilege or object not found: SEQ
31 Caught: java.sql.SQLException: user lacks privilege or object not found: SEQ
32 at _D1.D1_2.run(D1_2.groovy:22)
33 */
34
35
36 db.execute ""
37 DROP INDEX SEQIDX IF EXISTS;
38 DROP TABLE LOGIN_LOG IF EXISTS;
39 CREATE TABLE LOGIN_LOG(
40 SEQ INTEGER GENERATED BY DEFAULT AS IDENTITY,
41 NAME VARCHAR(64),
42 IP VARCHAR(64),
43 LOGIN_DATE DATE
44 );
45 CREATE INDEX SEQIdx ON LOGIN_LOG (SEQ);
46 "";
```

데이터 집어넣기

```
1 package _D1
2
3
4 import java.util.logging.Logger;
5
6 import groovy.sql.Sql
```

```

7 import groovy.sql.*
8
9 //데이터 소스 이용 커넥션
10 def source = new org.hsqldb.jdbc.JDBCDataSource();
11 source.database = 'jdbc:hsqldb:hsqldb://localhost/javaworld';
12 source.user = 'sa';
13 source.password = '';
14 def db = new groovy.sql.Sql(source);
15
16
17 db.execute """
18 INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
19 VALUES('AAA','123.123.123','2014-02-04');
20 INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
21 VALUES('AAA','223.223.223','2014-02-05');
22 INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
23 VALUES('AAA','201.203.204','2014-02-06');
24 """
25
26
27 ///////////////prepared statement 제공/////////////////
28 /*
29 주의 prepared statement 에서는 '(작은따옴표)' 쓰지말아야 한다.
30 명령문 쪽에서도 없어야하고 명령에 전달되는 문자열 내부에도 없어야 한다
31 작은따옴표는 그루비를 위한것이지 SQL을 위한것이 아니다.
32 */
33 String sql = """
34 INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
35 VALUES(?,?,?);
36 """
37 db.execute (sql,['BBB','10.10.10','1954-02-01']);
38 db.execute (sql,['BBB','11.11.11','1954-02-02']);
39 db.execute (sql,['B'B'B " B','13.13.13','1954-02-03']);
40
41 ///////////////
42 sql = """
43 INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
44 VALUES('CCC',?,?);
45 """
46 db.execute (sql,['10.10.10','1954-02-01']);
47 db.execute (sql,['11.11.11','1954-02-02']);
48 db.execute (sql,['12.12.12','1954-02-03']);
49
50 ///////////////
51
52 def loglist = [
53 [NAME:'DDD', IP:'1.1.1.1', LOGIN_DATE:'1234-11-11'],
54 [NAME:'DDD', IP:'2.2.2.2', LOGIN_DATE:'1234-11-12'],
55 [NAME:'DDD', IP:'3.3.3.3', LOGIN_DATE:'1234-11-13']
56 ];
57
58 //로그를 찍어보장~
59 //Logger.getLogger ('groovy.sql').level = Level.FINE;
60
61 loglist.each { inRow ->
62 db.execute """
63 INSERT INTO LOGIN_LOG (NAME,IP,LOGIN_DATE)
64 VALUES(${inRow.NAME},${inRow.IP},${inRow.LOGIN_DATE});
65 """
66 }
67 }

```

HSQL Database Manager

File View Command Recent Options Tools

jdbc:hsqldb:hsqldb://localhost/javaworld

LOGIN_LOG

schema: PUBLIC

SEQ

NAME

IP

LOGIN_DATE

Indices

TEST

Properties

SELECT * FROM LOGIN_LOG

Clear

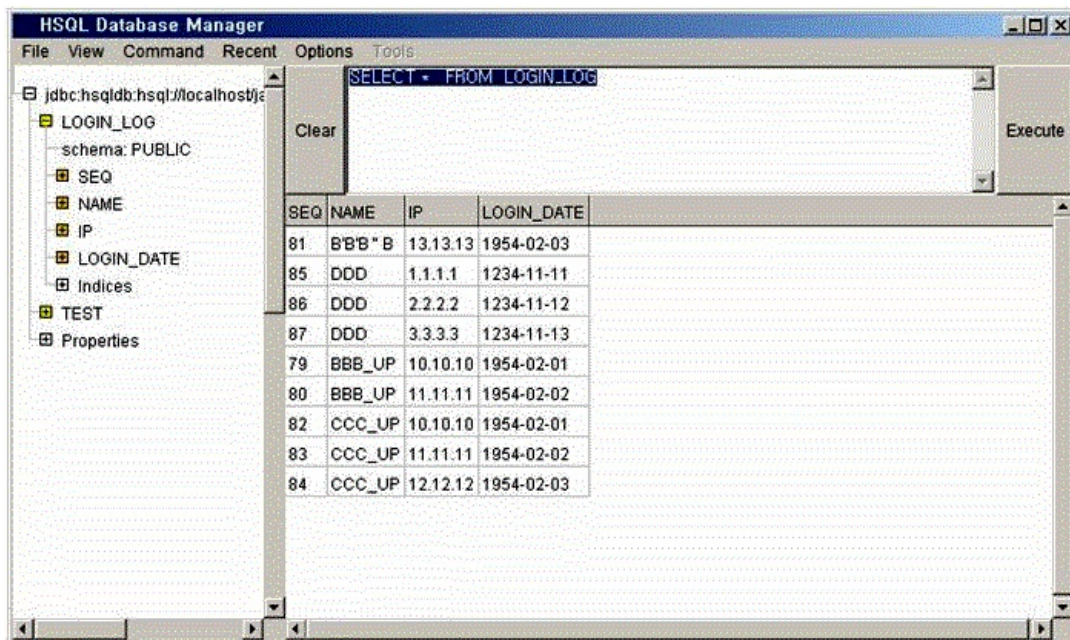
Execute

SEQ	NAME	IP	LOGIN_DATE
40	AAA	123.123.123	2014-02-04
41	AAA	223.223.223	2014-02-05
42	AAA	201.203.204	2014-02-06
43	BBB	10.10.10	1954-02-01
44	BBB	11.11.11	1954-02-02
45	B'B'B " B	13.13.13	1954-02-03
46	CCC	10.10.10	1954-02-01
47	CCC	11.11.11	1954-02-02
48	CCC	12.12.12	1954-02-03
49	DDD	1.1.1.1	1234-11-11
50	DDD	2.2.2.2	1234-11-12
51	DDD	3.3.3.3	1234-11-13

데이터 수정하고 삭제하기

수정과 삭제도 execute메서드로 실행한다

```
1 package _D1
2
3 //데이터 소스 이용 커넥션
4 def source = new org.hsqldb.jdbc.JDBCDataSource();
5 source.database = 'jdbc:hsqldb:hsqldb://localhost/javaworld';
6 source.user = 'sa';
7 source.password = '';
8 def db = new groovy.sql.Sql(source);
9
10
11 db.execute '''
12 DELETE FROM LOGIN_LOG WHERE NAME ='AAA'
13 '''
14
15 db.execute '''
16 UPDATE LOGIN_LOG SET NAME='BBB_UP' WHERE NAME ='BBB'
17 '''
18
19
20 def updateCnt = db.executeUpdate ('''
21 UPDATE LOGIN_LOG SET NAME='CCC_UP' WHERE NAME ='CCC'
22 ''')
23 println 'WHERE NAME ='CCC' updateCnt : '+updateCnt
24
25 /* 결과
26 WHERE NAME ='CCC' updateCnt : 3
27 */
```



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

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

데이터 조회

데이터베이스에서 데이터를 읽어 들이는 작업은 기존과 비슷하다 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.

```
1 package _D1
2
3 //데이터 소스 이용 커넥션
4 def source = new org.sql.db.jdbc.JDBCDataSource();
5 source.database = 'jdbc:hsqldb:hsqldb://localhost/javaworld';
6 source.user = 'sa';
7 source.password = '';
8 def db = new groovy.sql.Sql(source);
9
10 //Row 가져오기
11 db.eachRow('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG'){
12     atRow->
13     println "SEQ : ${atRow.SEQ}, NAME : ${atRow.NAME}, IP : ${atRow.IP}, LOGIN_DATE : ${atRow.LOGIN_DATE}";
14     println "SEQ : ${atRow[0]}, NAME : ${atRow[1]}, IP : ${atRow[2]}, LOGIN_DATE : ${atRow[3]}";
15     println '-'*25;
16 }
17
18 println '**'*25;
19
20 //ResultSet 가져오기
21 db.query('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG'){
22     resultSet->
23     if(resultSet.next()){
24         print resultSet.getString(1); //여기서 인덱스는 0부터 시작이 아니라 1부터 시작이다.
25         print
26         println resultSet.getString('NAME');
27     }
28 }
29
30 println '**'*25;
31 //rows메서드를 이용하면 모든 행을 하나의 긴 리스트에 담을 수 있다.
32 List list = db.rows('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG')
33 println list;
34
35 println '**'*25;
36 //메타데이터 가져오기.
37 db.query('SELECT SEQ, NAME, IP, LOGIN_DATE FROM LOGIN_LOG'){
38     resultSet->
39     def meta = resultSet.metaData;
40     if(meta.columnCount<=0){
41         return;
42     }
43     while(resultSet.next()){
44         for(i in 0..<meta.columnCount){
45             print "${i} : ${meta.getColumnLabel(i+1)}".padRight(20);
46             print resultSet.getObject(i+1);
47             print '\n'
48         }
49         println '-'*40;
50     }
51 }
52
53
54 /*
55 결과
56 SEQ : 81, NAME : B'B'B " B, IP : 13.13.13, LOGIN_DATE : 1954-02-03
57 SEQ : 81, NAME : B'B'B " B, IP : 13.13.13, LOGIN_DATE : 1954-02-03
58 -----
59 SEQ : 85, NAME : DDD, IP : 1.1.1.1, LOGIN_DATE : 1234-11-11
60 SEQ : 85, NAME : DDD, IP : 1.1.1.1, LOGIN_DATE : 1234-11-11
61 -----
62 SEQ : 86, NAME : DDD, IP : 2.2.2.2, LOGIN_DATE : 1234-11-12
63 SEQ : 86, NAME : DDD, IP : 2.2.2.2, LOGIN_DATE : 1234-11-12
64 -----
65 SEQ : 87, NAME : DDD, IP : 3.3.3.3, LOGIN_DATE : 1234-11-13
66 SEQ : 87, NAME : DDD, IP : 3.3.3.3, LOGIN_DATE : 1234-11-13
67 -----
68 */
```

```

67 -----
68 SEQ : 79, NAME : BBB_UP, IP : 10.10.10, LOGIN_DATE : 1954-02-01
69 SEQ : 79, NAME : BBB_UP, IP : 10.10.10, LOGIN_DATE : 1954-02-01
70 -----
71 SEQ : 80, NAME : BBB_UP, IP : 11.11.11, LOGIN_DATE : 1954-02-02
72 SEQ : 80, NAME : BBB_UP, IP : 11.11.11, LOGIN_DATE : 1954-02-02
73 -----
74 SEQ : 82, NAME : CCC_UP, IP : 10.10.10, LOGIN_DATE : 1954-02-01
75 SEQ : 82, NAME : CCC_UP, IP : 10.10.10, LOGIN_DATE : 1954-02-01
76 -----
77 SEQ : 83, NAME : CCC_UP, IP : 11.11.11, LOGIN_DATE : 1954-02-02
78 SEQ : 83, NAME : CCC_UP, IP : 11.11.11, LOGIN_DATE : 1954-02-02
79 -----
80 SEQ : 84, NAME : CCC_UP, IP : 12.12.12, LOGIN_DATE : 1954-02-03
81 SEQ : 84, NAME : CCC_UP, IP : 12.12.12, LOGIN_DATE : 1954-02-03
82 -----
83 *****
84 81  B'B'B " B
85 *****
86 [[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
87 *****
88 0 : SEQ      81
89 1 : NAME      B'B'B " B
90 2 : IP        13.13.13
91 3 : LOGIN_DATE 1954-02-03
92 -----
93 0 : SEQ      85
94 1 : NAME      DDD
95 2 : IP        1.1.1.1
96 3 : LOGIN_DATE 1234-11-11
97 -----
98 0 : SEQ      86
99 1 : NAME      DDD
100 2 : IP        2.2.2.2
101 3 : LOGIN_DATE 1234-11-12
102 -----
103 0 : SEQ      87
104 1 : NAME      DDD
105 2 : IP        3.3.3.3
106 3 : LOGIN_DATE 1234-11-13
107 -----
108 0 : SEQ      79
109 1 : NAME      BBB_UP
110 2 : IP        10.10.10
111 3 : LOGIN_DATE 1954-02-01
112 -----
113 0 : SEQ      80
114 1 : NAME      BBB_UP
115 2 : IP        11.11.11
116 3 : LOGIN_DATE 1954-02-02
117 -----
118 0 : SEQ      82
119 1 : NAME      CCC_UP
120 2 : IP        10.10.10
121 3 : LOGIN_DATE 1954-02-01
122 -----
123 0 : SEQ      83
124 1 : NAME      CCC_UP
125 2 : IP        11.11.11
126 3 : LOGIN_DATE 1954-02-02
127 -----
128 0 : SEQ      84
129 1 : NAME      CCC_UP
130 2 : IP        12.12.12
131 3 : LOGIN_DATE 1954-02-03
132 -----
133 */

```

토탈.

```

1 package _D1
2
3 import groovy.sql.Sql
4
5
6 dbHandle = null // XXX todo explain why Sql is missing
7
8 def getDb(){
9     if (dbHandle) return dbHandle // #1
10    def source = new org.hsqldb.jdbc.jdbcDataSource()
11    source.database = 'jdbc:hsqldb:mem:GIA'
12    source.user = 'sa'
13    source.password = ''
14    dbHandle = new Sql(source)
15    return dbHandle
16 }
17 def reset() {
18     db.execute """
19         DROP INDEX athleteidx IF EXISTS;
20         DROP TABLE Athlete IF EXISTS;
21         CREATE TABLE Athlete (
22             athleteid INTEGER GENERATED BY DEFAULT AS IDENTITY,
23             firstname VARCHAR(64),
24             lastname VARCHAR(64).

```

```

25         dateOfBirth DATE
26     );
27     CREATE INDEX athleteIdx ON Athlete (athleteId);
28
29 }
30 def create(firstname, lastname, dateOfBirth) {           //#3
31     db.execute """
32         INSERT INTO Athlete ( firstname, lastname, dateOfBirth)
33             VALUES ($firstname,$lastname,$dateOfBirth);
34     """
35 }
36 def findAll() {                                           //#4
37     db.rows 'SELECT * FROM Athlete'
38 }
39 def updateFirstName(wrong, right) {                       //#5
40     db.execute """
41         UPDATE Athlete
42         SET firstname = $right WHERE firstname = $wrong;
43     """
44 }
45 def delete(firstname) {                                   //#6
46     db.execute "DELETE FROM Athlete WHERE firstname = $firstname;"
47 }
48
49 reset()
50 assert ! findAll(), 'we are initially empty'
51 create 'Dirk', 'Koenig', '1968-04-19'
52 assert 'Dirk' == findAll()[0].firstname                 //#7
53 updateFirstName 'Dirk', 'Dierk'
54 assert 'Dierk' == findAll()[0].firstname
55 delete 'Dierk'
56 assert ! findAll(), 'after delete, we are empty again'

```

DataSet로 SQL 없이 SQL사용하기

그루비는 sql을 사용하지 않으면서 데이터 베이스를 처리할 수 있는 방법이었다.

DataSet이라는 개념이다.

1. 테이블 행추가
2. 테이블이나 뷰 모든행 처리하기
3. 단순한 표현식으로 테이블이나 뷰 조회하기

```

1 package _D2
2
3 import groovy.sql.DataSet
4
5 def source = new org.sql.db.jdbc.JDBCDataSource();
6 source.database = 'jdbc:hsqldb:hsqldb://localhost/javaworld';
7 source.user = 'sa';
8 source.password = '';
9 def db = new groovy.sql.Sql(source);
10
11
12 // 테이블에 새로운 행을 추가할수있다.
13 DataSet login_log = db.dataSet("LOGIN_LOG");
14 login_log.add(
15     name:"dataset",
16     ip:"0.0.0.0",
17     login_date:"1986-04-25",
18 );
19
20 println '---each---'
21 login_log.each {
22     println it.name+' '+it.ip+' '+it.login_date;
23 }
24
25 println '---findAll--where name=dataset--'
26 //특정 조건으로찾은 행들 DataSet를 리턴한다
27 //여기서 데이터를 다 가져온후 클로저로 일일이 걸러내는것같이 할것같지만
28 //사실은 findAll은 클로저에 제시된 표현식으로 sql을 만들어서 실행한다.
29 //each할때 쿼리실행이된다.
30 DataSet login_log_where1 = login_log.findAll {
31     it.name=='dataset';
32 }
33 login_log_where1.each {
34     println it.name+' '+it.ip+' '+it.login_date;
35 }
36 println 'login_log_where1 sql : '+login_log_where1.sql;
37 println 'login_log_where1 parameters : '+login_log_where1.parameters;
38
39
40
41
42 DataSet login_log_where2 = login_log.findAll {
43     it.name=='dataset' && it.ip=='0.0.0.0';
44 }
45 login_log_where2.each {
46     println it.name+' '+it.ip+' '+it.login_date;
47 }

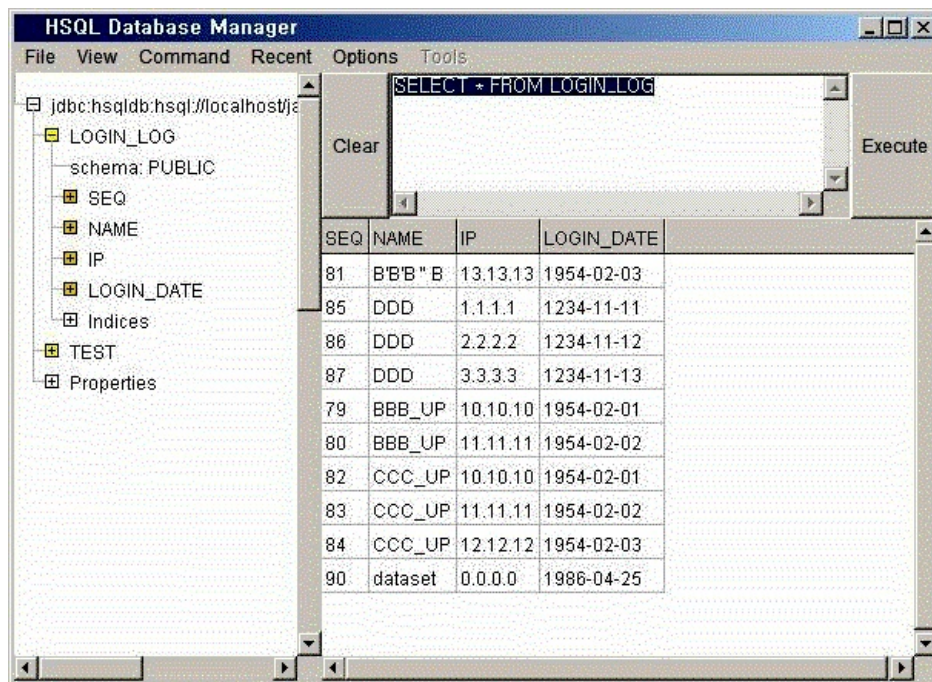
```



```

48 println 'login_log_where2 sql : '+login_log_where2.sql;
49 println 'login_log_where2 parameters : '+login_log_where2.parameters;
50
51 /*
52 &&      ->   AND
53 ||      ->   OR
54 ==      ->   =
55 다른연산자 ->   그대로적용
56 it.propertyname ->   프로퍼티 이름을 필드로 보고 그대로적용
57 상수      ->   ? 표현식은 인자 리스트에 들어간다
58 */
59
60 /* 결과
61 ---each---
62 B'B'B " B 13.13.13 1954-02-03
63 DDD 1.1.1.1 1234-11-11
64 DDD 2.2.2.2 1234-11-12
65 DDD 3.3.3.3 1234-11-13
66 BBB_UP 10.10.10 1954-02-01
67 BBB_UP 11.11.11 1954-02-02
68 CCC_UP 10.10.10 1954-02-01
69 CCC_UP 11.11.11 1954-02-02
70 CCC_UP 12.12.12 1954-02-03
71 dataset 0.0.0.0 1986-04-25
72 ---findAll--where name=dataset--
73 dataset 0.0.0.0 1986-04-25
74 login_log_where1 sql : select * from LOGIN_LOG where name = ?
75 login_log_where1 parameters : [dataset]
76 dataset 0.0.0.0 1986-04-25
77 login_log_where2 sql : select * from LOGIN_LOG where (name = ? and ip = ?)
78 login_log_where2 parameters : [dataset, 0.0.0.0]
79
80
81 */

```



AST 노트	SQL
&&	AND
	OR
==	=
다른 연산자들	그대로 적용
it.propertyname	프로퍼티 이름을 필드로 보고 그대로 적용
tkdtn	? 표현식은 인자 리스트에 들어간다