UniSQL KOREA COMPUTER COMMUNICATIONS, LTD.
기술부

JDBC Program Guide

Version
1.0

한국컴퓨터통신(주)
135-080 서울시 강남구 역삼동 679-4 로담코타워 13층. ☏ 02-3430-1200 http://www.unisql.com

목 차

1. 개요

본 문서는 JDBC 인터페이스를 통해 UniSQL DBMS(이하 UniSQL)를 사용하는 방법을 설명한다. 일반적인 JDBC API 및 UniSQL에 대해서는 본 문서에서는 별도로 설명하지 않는다. 따라서 본 문서의 샘플코드들을 이해하려면 사전에 JDBC 및 UniSQL에 대한 지식이 있어야 한다. 특별히 UniSQL을 이해하기 위해서는 본사 홈페이지의 매뉴얼이나 교육시 제공된 기본교육교재를 활용할 수 있다.

1.1 동작환경

UniSQL에서 JDBC를 사용하려면 다음과 같은 사항을 확인해야 한다.

1.1.1 JDBC 1.0

- UniSQL/X Realese 4.1 이상
- UniCAS 4.1 이상
- JDK 1.2 이상

1.1.2 JDBC 2.0

- UniSQL/X Realese 4.3 이상
- UniCAS 4.2 이상
- JDK 1.2 이상

1.2 환경설정

UniSQL에서 JDBC를 사용하려면 UniCAS 및 CLASSPATH가 세팅되어야 한다. 특히 UniCAS는 UniSQL 데이터베이스에 접속하기 위해 반드시 거쳐야 하는 미들웨어이며 Pooling의 기능을 수행한다. 따라서 UniSQL에서 JDBC를 이용려면 UniCAS의 환경 설정에 대해 알아야 한다.

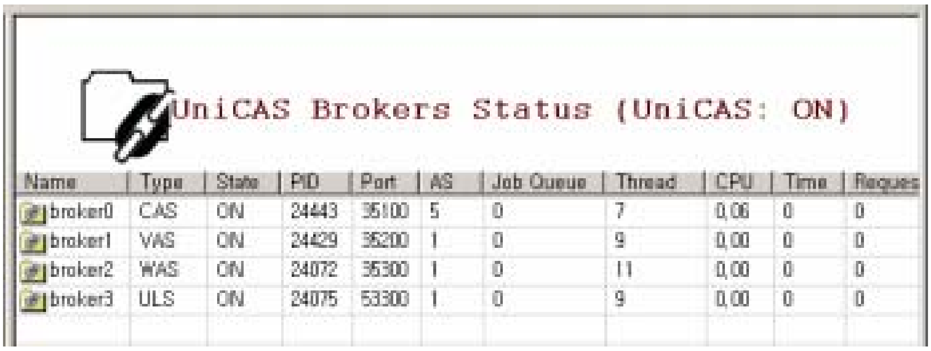
1.2.1 UniCAS 환경 설정

본 절에서는 UniCAS의 환경설정방법을 Easy-Manager를 이용한 방법과 Terminal 편집기를 이용한 방법으로 나누어 설명한다.

1.2.1.1. Easy-Manager를 이용한 UniCAS 환경설정

JDBC 요구사항 처리를 위한 UniCAS 환경은 그림1과 같이 CAS Type의 브로커가 ON 상태로 정상 서비스 중에 있어야 하고, 그림2와 같이 CAS Type 브로커에 속한

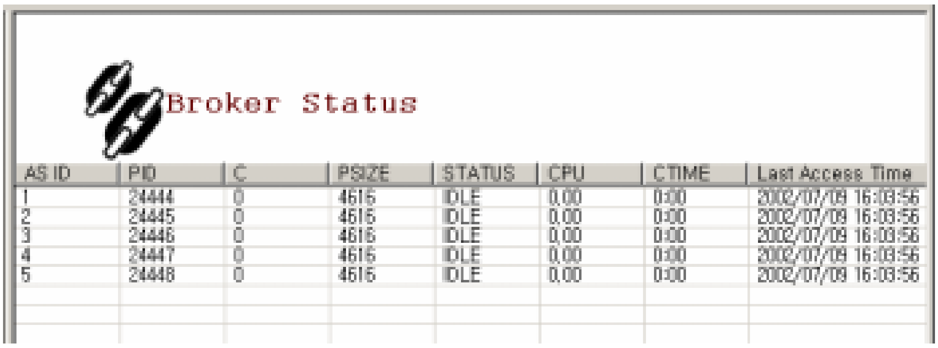
Application Server가 적어도 한 개 이상 존재해야 한다. Easy-Manager를 이용하여 CAS Type의 브로커를 추가방법은 다음과 같다.



UniCAS Brokers Status (UniCAS: ON)

Name	Type	State	PID	Port	AS	Job Queue	Thread	CPU	Time	Reques
broker0	CAS	ON	24443	35100	5	0	7	0.06	0	0
broker1	VAS	ON	24429	35200	1	0	9	0.00	0	0
broker2	WAS	ON	24072	35300	1	0	11	0.00	0	0
broker3	ULS	ON	24075	53300	1	0	9	0.00	0	0

그림 1. EasyManager에서 브로커 정보 확인



Broker Status

AS ID	PID	C	PSIZE	STATUS	CPU	CTIME	Last Access Time
1	24444	0	4616	IDLE	0.00	0:00	2002/07/09 16:03:56
2	24445	0	4616	IDLE	0.00	0:00	2002/07/09 16:03:56
3	24446	0	4616	IDLE	0.00	0:00	2002/07/09 16:03:56
4	24447	0	4616	IDLE	0.00	0:00	2002/07/09 16:03:56
5	24448	0	4616	IDLE	0.00	0:00	2002/07/09 16:03:56

그림 2. EasyManager에서 Application Server 정보확인

새로운 브로커를 추가하기 위해서 Broker Level 팝업 메뉴에서 “Broker Add”를 선택하고 아래의 단계1 ~ 단계3을 수행한다.

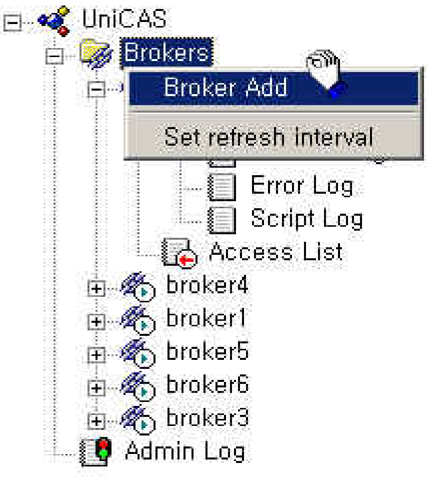
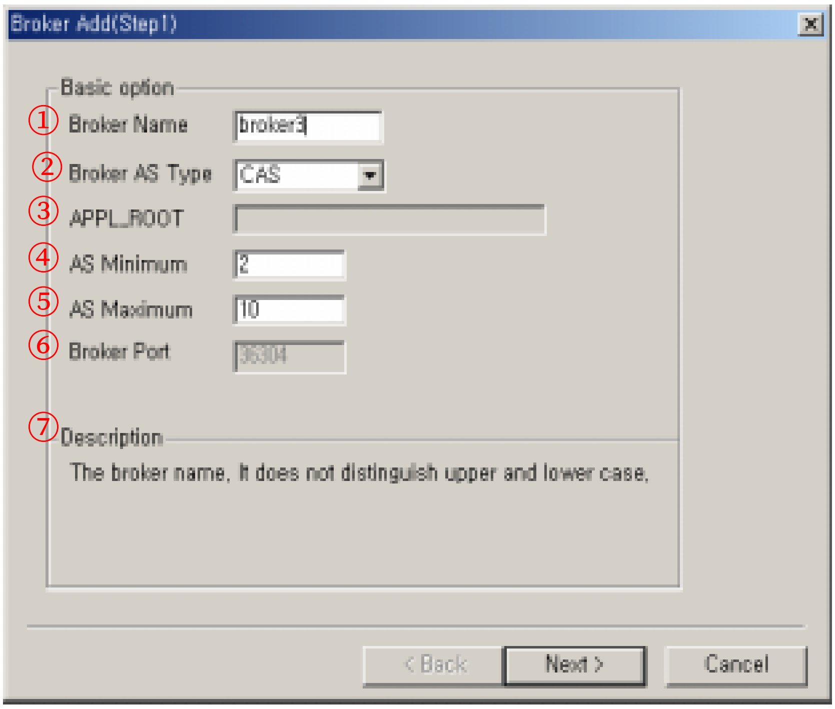


그림3. 브로커 팝업 메뉴

단계. 1



Broker Add(Step1)

Basic option

① Broker Name: broker3

② Broker AS Type: CAS

③ APPL_ROOT:

④ AS Minimum: 2

⑤ AS Maximum: 10

⑥ Broker Port: 36304

⑦ Description

The broker name. It does not distinguish upper and lower case.

그림4. Broker Add – Step1

① Broker Name : 브로커 이름을 입력한다.

② Broker AS Type : 어플리케이션 서버 타입을 CAS로 선택한다.

③ APPL_ROOT : 어플리케이션 서버 루트 디렉토리를 정한다. CAS에서는 필요 없는 부분이므로 비활성화 된다.

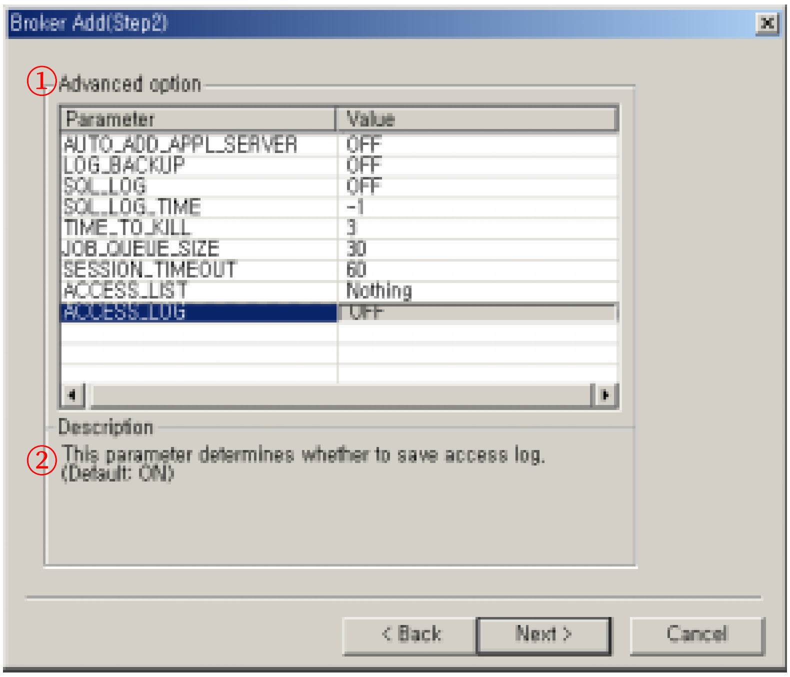
④ AS Minimum : 어플리케이션 서버의 최소 개수를 정한다.

⑤ AS Maximum : 어플리케이션 서버의 최대 개수를 정한다.

⑥ Broker Port : 브로커 포트를 설정한다. 여기에서 설정된 포트가 JDBC Connection시에 사용되는 포트가 된다.

⑦ Description : 각 항목에 대한 자세한 설명이 기술된다.

단계. 2



Parameter	Value
AUTO_ADD_APPL_SERVER	OFF
LOG_BACKUP	OFF
SQL_LOG	OFF
SQL_LOG_TIME	-1
TIME_TO_KILL	3
JOB_QUEUE_SIZE	30
SESSION_TIMEOUT	60
ACCESS_LIST	Nothing
ACCESS_LOG	OFF

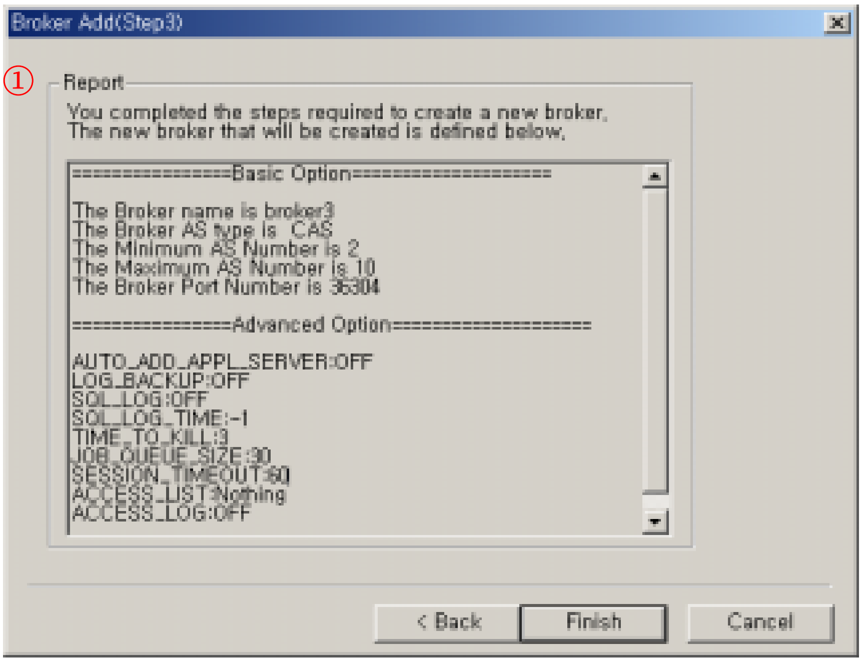
This parameter determines whether to save access log.
(Default: ON)

그림5. Broker Add – Step2

① Advanced option : 단계 1. 에서 설정하지 못했던 고급 옵션들에 대하여 설정할 수 있는 목록이 나오며 직접 입력할 수 있다.

② Description : 목록에 표시되고있는 파라미터에 대한 자세한 설명이 기술된다.

단계. 3



You completed the steps required to create a new broker.
The new broker that will be created is defined below.

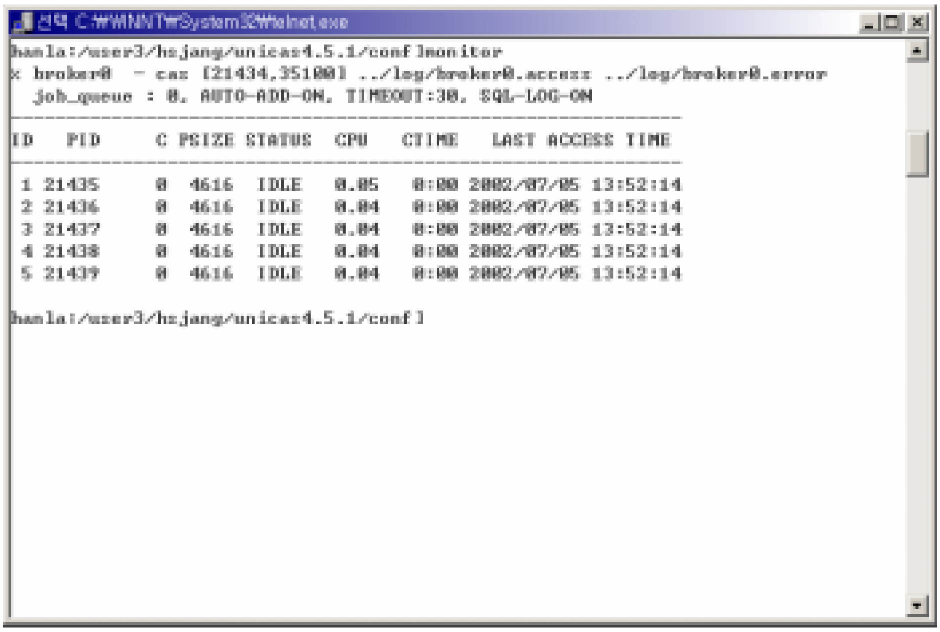
```
================Basic Option====================

The Broker name is broker3
The Broker AS type is  CAS
The Minimum AS Number is 2
The Maximum AS Number is 10
The Broker Port Number is 36304

================Advanced Option===================

AUTO_ADD_APPL_SERVER:OFF
LOG_BACKUP:OFF
SQL_LOG:OFF
SQL_LOG_TIME:-1
TIME_TO_KILL:3
JOB_QUEUE_SIZE:30
SESSION_TIMEOUT:60
ACCESS_LIST:Nothing
ACCESS_LOG:OFF
```

그림6. Broker Add – Step3

① Report : 지금까지 설정한 내용들을 요약하여 보여준다.

1.2.1.1 Terminal 편집기를 이용한 UniCAS 환경설정

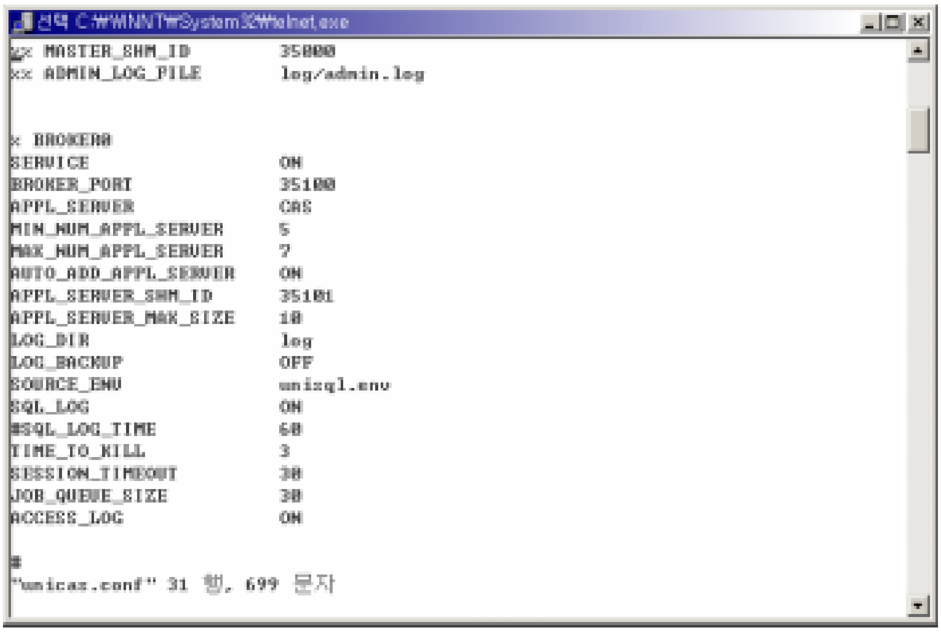
UniCAS의 상태 점검을 위해서는 monitor 유틸리티를 사용한다.(그림7. 참조) JDBC 요구사항 처리를 위해서는 CAS type으로 서비스 중인 broker가 적어도 한 개 이상 모니터 되어야 한다. 또한, CAS broker의 Application Server들이 적어도 한 개 이상 존재해야 한다. 다음은 Ternînal 편집기를 이용한 CAS type 브로커 추가 방법이다.



```
hanla:/user3/hsjang/unicas4.5.1/conf]monitor
% broker0   - cas [21434,35100] ../log/broker0.access ../log/broker0.error
  job_queue : 0, AUTO-ADD-ON, TIMEOUT:30, SQL-LOG-ON
----------------------------------------------------------------------------
ID    PID      C PSIZE STATUS   CPU    CTIME    LAST ACCESS TIME
----------------------------------------------------------------------------
 1 21435      0  4616  IDLE    0.05    0:00 2002/07/05 13:52:14
 2 21436      0  4616  IDLE    0.04    0:00 2002/07/05 13:52:14
 3 21437      0  4616  IDLE    0.04    0:00 2002/07/05 13:52:14
 4 21438      0  4616  IDLE    0.04    0:00 2002/07/05 13:52:14
 5 21439      0  4616  IDLE    0.04    0:00 2002/07/05 13:52:14

hanla:/user3/hsjang/unicas4.5.1/conf]
```

그림7. broker and current job information

CAS 형태 브로커를 설정하기 위해서 $UNICAS/conf/unicas.conf에서 broker설정을 한다.(그림8. 참조) broker를 서비스 상태로 만들기 위해 SERVICE값에 ON을, Application Server를 1개 이상 생성하기 위해 MIN_NUM_APPL_SERVER값을 1이상으로 설정한다.



```
xx MASTER_SHM_ID          35000
xx ADMIN_LOG_FILE         log/admin.log


x BROKER0
SERVICE                   ON
BROKER_PORT               35100
APPL_SERVER               CAS
MIN_NUM_APPL_SERVER       5
MAX_NUM_APPL_SERVER       7
AUTO_ADD_APPL_SERVER      ON
APPL_SERVER_SHM_ID        35101
APPL_SERVER_MAX_SIZE      10
LOG_DIR                   log
LOG_BACKUP                OFF
SOURCE_ENV                unisql.env
SQL_LOG                   ON
#SQL_LOG_TIME             60
TIME_TO_KILL              3
SESSION_TIMEOUT           30
JOB_QUEUE_SIZE            30
ACCESS_LOG                ON

#
"unicas.conf" 31 행, 699 문자
```

그림8. specify the parameters of unicas.conf file

자세한 사항은 본사 홈페이지의 UniCAS Installation Guide 5~13페이지를 참조 할 수 있다.

1.2.2 CLASSPATH 설정

CLASSPATH에 UniSQL JDBC 드라이버를 추가한다. (자세한 사항은 본사 홈페이지의 UniCAS Installation Guide 17~18 페이지 참조)

Ex) csh - **setenv CLASSPATH ${CLASSPATH}:${HOME}/jdbc/unisqljdbc202.jar**

2. 일반적인 사용

본 장에서는 UniSQL의 특징적인 기능들을 사용하지 않고 표준SQL과 같은 방법으로 UniSQL을 사용할 때 쓸 수 있는 코드들을 소개한다. 이는 다른 RDBMS에서 JDBC를 사용하는 방법과 다르지 않으므로 UniSQL 데이터베이스에 Connection하는 방법만 유의하면 된다.

2.1 Connection

Connection은 다음과 같은 형식으로 이루어진다.

```
Connection conn =
DriverManager.getConnection( “jdbc : unisql : IP : PORT :::” , ”ID” , ”PASSWORD” );
```

- IP : 데이터베이스가 있는 서버 IP
- PORT : UniCAS에서 CAS Type 설정된 브로커의 포트
- ID : 데이터베이스에 접속할 아이디. 기본적으로 데이터베이스에는 DBA와 PUBLIC 2개의 아이디가 존재한다. ID 파라미터에 아무 것도 입력하지 않고 “”를 입력하면 기본적으로 PUBLIC 아이디로 데이터베이스에 접속한다.
- PASSWORD : 앞에서 사용한 아이디에 암호가 설정되어 있다면 PASSWORD 파라미터를 입력한다. 없다면 그냥 “”를 입력한다.

2.2 UniSQLConnection

UniSQL JDBC에는 일반 Connection외에도 UniSQLConnection이 있다. GLO를 사용하거나 getNewInstance API를 사용하려면 반드시 UniSQLConnection을 해야 한다.

```
UniSQLConnection conn =
(UniSQLConnection)DriverManager.getConnection(“jdbc:unisql:IP : PORT :::” , ”ID” , ”PASSWORD” );
```

2.3 데이터 조회

일반적인 데이터 조회 방법이다. 방법은 기존 RDBMS와 동일하다.

```
import java.sql.*;

public class selectData {
   public static void main(String[] args) throws Exception {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
      // UniSQL DB에 Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");              // ①
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");   // ②

      String sql = "select name, country, cost, checkout_time, number_of_restaurants from hotel";
      stmt = conn.createStatement();
      rs = stmt.executeQuery(sql);

      while(rs.next()) {
        String name = rs.getString("name");
        String country = rs.getString("country");
        String cost = rs.getString("cost");
        String checkout_time = rs.getString("checkout_time");
        String number_of_restaurants = rs.getString("number_of_restaurants");
        System.out.println("name ==> " + name);
        System.out.println("country ==> " + country);
        System.out.println("cost ==> " + cost);
        System.out.println("checkout_time ==> " + checkout_time);
        System.out.println("number_of_restaurants ==> " + number_of_restaurants);
        System.out.println("\n========================================\n");
      }
      rs.close();
      stmt.close();
      conn.close();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- ①: UniSQL JDBC 드라이버 클래스를 로드한다.
- ②: 데이터베이스와 Connection한다. 이 sample코드에서와 같이 ID파라미터 값이 주어지지 않으면 PUBLIC user로 데이터베이스에 Connection된다. 특정 user로 접속한다거나 암호가 설정되어 있다면 파라미터 값을 입력하면 된다.
- ②: Connection하기 위해 브로커 포트인 43300번을 사용하였다.

2.4 데이터 입력/삭제/갱신

일반적으로 사용되는 데이터 입력, 삭제, 갱신 방법이다. 기존 RDBMS와 사용방법이 같다.

```
import java.sql.*;

public class insertData {
   public static void main(String[] args) throws Exception {

      Connection conn = null;
      Statement stmt = null;

      try {

            // UniSQL DB에 Connect
            Class.forName("unisql.jdbc.driver.UniSQLDriver");
            conn =
                DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

            // 현재 connection을 autocommit하지 않음. (default는 autocommit임.)
            conn.setAutoCommit(false);          // ①

            String sql = "insert into test_class(cur_date) values (SYS_DATE)";
            stmt = conn.createStatement();
            stmt.executeUpdate(sql);
            System.out.println("데이터가 입력되었습니다.");
            stmt.close();
            conn.commit();

      } catch ( SQLException e ) {
            System.err.println(e.getMessage());
      } catch ( Exception e ) {
            System.err.println(e.getMessage());
      } finally {
            if ( conn != null ) conn.close();
      }
   }
}
```

- ①: 기본값은 true. 따라서 autoCommit되지 않도록 하려면 false로 설정해야 한다. (3.2.1.2 참조)

- 데이터 삭제 및 갱신방법은 입력방법과 수행방법이 같으므로 이 코드를 그대로 사용하면 된다.

3. UniSQL의 특징적 기능 사용

3.1 OID의 사용

본 장에서는 JDBC에서 OID를 사용하는 코드에 대해 설명한다. 코드를 보기에 앞서 OID가 무엇인지, 어떻게 질의하는지 사전에 알고 있어야 한다. OID를 잘 사용하면 질의의 수를 줄일 수 있어 많은 속도향상을 이끌어낼 수도 있다. 아래는 이 장에서 사용될 DB의 스키마이다. 특히 user_info class의 car attribute가 car class를 가리키고 있는데, 이러한 관계를 **composition**이라 하며 이 장에서 주로 사용하게 될 부분이다.

```
CREATE CLASS car (
   name char(40),
   spec  string,
   color  char(20)
)
CREATE CLASS user_info (
   name char(20),
   addr  string,
   car    car
)
```

3.1.1 데이터 조회

OID를 이용하면 별도로 질의를 수행하지 않고도 Connection만으로 데이터를 조회할 수 있다. 본 절에서는 OID를 이용한 데이터 조회 방법과 이를 위해서 필요한 string의 UniSQLOID 변환 방법등을 알아본다.

3.1.1.1 OID를 사용한 데이터 조회

attribute를 지정하면 OID를 이용하여 데이터를 조회할 수 있다. 본 예제는 user_info 클래스에서 car의 OID를 얻어와서 그 OID의 데이터 중 name, spec, color 어트리뷰트의 값을 얻어온다.

```
import java.sql.*;
import unisql.sql.*;            // ①

public class selectOID2 {
   public static void main(String[] args) throws Exception {

      Connection conn = null;
      Statement stmt = null;
      ResultSet rs = null;
      ResultSet urs = null;

      try {
         // UniSQL DB Connect
         Class.forName("unisql.jdbc.driver.UniSQLDriver");
         conn =
            DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");
      conn.setAutoCommit(false);          // ②

      // car라는 attribute는 다른 클래스의 인스턴스를 가리키고 있다.  따라서 이 값을 select하면
      // OID을 가져오게 된다.
      String sql = "select name, addr, car from user_info";
      stmt = conn.createStatement();
      rs = stmt.executeQuery(sql);

      // OID를 사용해서 실제로 가져오고 싶은 attribute들을 아래와 같이 array에 저장한다.
      String [] attr = {"name", "spec", "color"};          // ③

      while(rs.next()) {
        String user_name = rs.getString("name");
        String user_addr = rs.getString("addr");

        // select한 값이 OID일 경우 아래와 같이 UniSQLOID 타입으로 받아야 한다.
        UniSQLOID user_car = (UniSQLOID)rs.getObject("car");          // ④

        // 위에서 받아온 OID를 사용하여 실제 원하는 값들을 가지고 온다.  가지고 오고 싶은 attribute
        // 들은 앞에서 이미 array에 저장한 상태이고, 이렇게 해서 가져온 값은 ResultSet에 저장된다.
        urs = user_car.getValues(attr);          // ⑤
        urs.next();
        String car_name = urs.getString("name");
        String car_spec = urs.getString("spec");
        String car_color = urs.getString("color");

        System.out.println("이름 : " + user_name);
        System.out.println("주소 : " + user_addr);
        System.out.println("소유자동차 : " + car_name);
        System.out.println("자동차 사양 : " + car_spec);
        System.out.println("자동차 색깔 : " + car_color);
        System.out.println("\n==================================================\n");
      }
      urs.close();
      rs.close();
      stmt.close();
      conn.commit();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 이 예제는 user와 user의 자동차 정보를 보여준다.

- ①: OID를 사용해야하므로 unisql.sql.*를 import한다.

- ②: 한 개의 ResultSet이 close()되기 전에 다른 ResultSet을 생성할 경우 앞의 ResultSet은 autoCommit되어 close()된다. 따라서 이 코드와 같이 동시에 여러 개의 ResultSet이 필요한 경우엔 반드시 autoCommit을 false로 설정해야 한다.

- ③: attr은 OID를 통해 가져올 attribute들을 array로 저장한다.

- ④: user_info class에서 이름, 주소 및 user가 소유한 자동차의 OID를 얻어온다. OID는 UniSQLOID 타입을 사용하며 getObject를 통하여 값을 얻는다.

- ⑤: getObject를 통해 얻은 OID값을 이용하여 실제 원하는 데이터를 조회한다. 이때 사용하는 API가 getValues이며 결과는 ResultSet에 저장된다.

- getValues(attr) : attr은 String array 값으로서 값을 가져올 attribute들의 이름을 저장한다. 가져올 attribute들의 이름을 지정하는 대신에 null값을 입력해도 되며 이 경우엔 모든 attribute 값을 가져온다. (ex. getValues(null). 3.2.1.2 참조)

3.1.1.2 String 값을 OID 값으로 변환

웹 환경에서는 파라미터들을 String타입로 넘기는데, OID의 경우 이러한 String타입일 경우 그대로 사용하는 것이 불가능하다. 본 절에서는 String타입의 OID를 사용하기 위해 UniSQLOID로 변환하는 방법을 알아본다.

```
import java.sql.*;
import unisql.sql.*;
import unisql.jdbc.driver.*;                // ①

public class selectOID3 {
  public static void main(String[] args) throws Exception {

    UniSQLConnection conn =  null;
    ResultSet urs = null;

    try {

      // 아래의 OID는 웹 환경을 고려하여 임의로 String값으로 설정.
      String Car_OID_S = "@320|8|0";              // ②

      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
    (UniSQLConnection)DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","",
"");                    // ③

      // OID string을 UniSQLOID 타입으로 변환한다.
      UniSQLOID  Car_OID = UniSQLOID.getNewInstance(conn, Car_OID_S);          // ④
      String [] attr = {"name", "spec", "color"};

      // OID를 가지고 ResultSet을 구성
      urs = Car_OID.getValues(attr);
      urs.next();
      String name = urs.getString("name");
      String spec = urs.getString("spec");
      String color = urs.getString("color");
      System.out.println("car_name ==> " + name);
      System.out.println("car_spec ==> " + spec);
      System.out.println("car_color ==> " + color);
      System.out.println("\n=========================================\n");
      urs.close();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());

    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 일반적으로 웹 환경(servlet, JSP등)에서는 OID값을 쿼리스트링이나 hidden 값으로 다음 페이지에 넘겨준다. 이런 방법으로 넘어온 OID값은 string값의 형태를 가지므로 이것을 사용하기 위해서는 다시 **UniSQLOID** 타입으로 바꿔줄 필요가 있다. 이 예제는 그 변환방법에 대해 설명하고 있다.

- ② : 이 예제에서는 OID값이 string형태로 넘어왔다는 가정 하에 OID를 스트링화하여 Car_OID_S에 저장하였다. (실제로는 이와 같이 사용하지 않는다.)

- ④ : string형태의 OID값을 **UniSQLOID** 타입으로 바꾸려면 **getNewInstance**를 사용한다.

- ③ : **getNewInstance**는 connection 핸들을 파라미터로 사용하는데, 여기에서 사용되는 connection이 **UniSQLConnection**임에 주의해야 한다.

- ①: **UniSQLConnection**을 사용하려면 **unisql.jdbc.driver.***를 import해야 한다.

3.1.2 데이터 입력

OID는 데이터를 입력해야 존재한다. OID를 입력하는 것이 아니다. 따라서 본 절에서는 입력한 데이터의 OID를 얻어와서 그 OID를 이용해서 다른 작업을 하는 방법을 알아본다.

3.1.2.1 데이터 입력 후 OID 리턴

본 예제는 car 클래스에 데이터를 입력한 후, 그 OID를 얻어와서 user_info 클래스에 user 정보를 입력할 때 함께 입력하는 예제이다.

```
import java.sql.*;
import unisql.sql.*;
import unisql.jdbc.driver.*;    // ①

public class insertOID {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    UniSQLStatement ustmt = null;
    UniSQLPreparedStatement upstmt = null;

    try {

      // UniSQL DB Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

      conn.setAutoCommit(false);

      String sql = "insert into car values ('매그너스','2000 cc','흰색')";

      // executeInsert 명령을 사용하기 위해서는 UniSQLStatement를 사용해야한다.
      ustmt = (UniSQLStatement)conn.createStatement();        // ②
      // executeInsert를 통해 insert를 할 경우 insert함과 동시에 insert된 instance의 OID를 리턴
      UniSQLOID car_oid = ustmt.executeInsert(sql);          // ③
      System.out.println("자동차 데이터가 입력되었습니다.");

      // 위에서 리턴된 OID를 다른 class에 저장할 할 때에는 반드시 UniSQLPreparedStatement를
      // 사용해야 한다.
      // 아래의 질의에서 ? 부분이 OID가 입력될 부분
      sql = "insert into user_info(name, addr, car) values ('김삿갓', '주소 모름',?)";      // ④
      upstmt =
        (UniSQLPreparedStatement)conn.prepareStatement(sql);          // ⑤
      upstmt.setOID(1,car_oid);                    // ⑥
      upstmt.executeUpdate();
      System.out.println("user정보를 입력하였습니다.");
      ustmt.close();
      upstmt.close();
      conn.commit();

    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 한 class에 데이터를 입력한 후, 입력된 데이터의 OID를 다른 class에 입력해야 하는 경우가 있다. 이 코드에서는 먼저 car class에 데이터를 입력한 후, 그 OID를 얻어와서 user_info class에 user정보와 함께 OID를 입력한다.

- ③: car class의 데이터를 입력하면서 동시에 입력한 값의 OID를 얻어오려면 **executeInsert**를 사용해야 한다. **executeInsert**를 사용하면 insert쿼리수행 후, 입력된 OID를 리턴한다.

- ② : **executeInsert**를 사용하기 위해서는 Statement대신 **UniSQLStatement**를 사용해야 하며, 이를 위해서는 **unisql.jdbc.driver.***를 import해야 한다.(①)

- ④: 리턴된 OID 값을 user_info class에 입력하려면 **UniSQLPreparedStatement**를 사용해야한다.(⑤) OID를 입력할 자리에는 ?를 넣고 **setOID**를 사용하여 값을 입력한다.(⑥) **UniSQLPreparedStatement**를 사용하려면 **unisql.jdbc.driver.***를 import해야 한다.(①)

3.1.3 데이터 삭제

RDBMS에서는 일반적으로 조건을 이용하여 데이터를 삭제한다. 그러나 UniSQL에서는 삭제할 데이터의 OID를 알고 있을 경우 이를 이용하여 질의를 수행하지 않고 데이터를 삭제할 수 있다.

3.1.3.1 OID를 이용한 데이터 삭제

아래의 예제는 user_info class에서 car OID를 얻어와서 그 OID를 이용하여 car class에서 직접 해당 데이터를 삭제한다.

```
import java.sql.*;
import unisql.sql.*;
import unisql.jdbc.driver.*;

public class deleteOID {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
      // UniSQL DB Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

      String sql = "select car from user_info where name = '홍길동'";
      stmt = conn.createStatement();
      rs = stmt.executeQuery(sql);
      rs.next();
      // OID 값을 받아 온다.
      UniSQLOID user_car = (UniSQLOID)rs.getObject("car");

      // 받아온 OID를 가지고 질의 없이 직접 OID가 가리키는 데이터를 삭제.
      user_car.remove();            // ①
      System.out.println("삭제 되었습니다.");
      rs.close();
      stmt.close();
      conn.commit();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());

    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 이 코드는 '홍길동'이란 user의 car OID를 가지고 와서 OID에 해당되는 데이터를 삭제하는 코드이다.
- ①: 데이터 삭제는 **remove()** 를 이용한다.
- 이와 같이 car의 데이터를 삭제하면 데이터를 가리키고 있던 user_info class의 car attribute는 null이 된다. (함께 삭제되지 않음에 주의.)

3.1.4 데이터 갱신

OID를 이용하면 해당 데이터에 직접 접근해서 데이터를 갱신할 수 있다. 따라서 속도가 빠르다. OID를 이용하여 데이터를 갱신하는 방법에는 2가지가 있다. 하나는 질의를 이용하는 방법이고 다른 하나는 JDBC API를 이용하는 방법이다.

3.1.4.1 UPDATE OBJECT를 이용한 데이터 갱신

이 예제는 UPDATE OBJECT 질의에서 OID를 이용하여 데이터를 갱신한다. 질의를 이용하지만 데이터에 직접 접근하므로 속도가 빠르다.

```
import java.sql.*;
import unisql.sql.*;
import unisql.jdbc.driver.*;

public class updateOID {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    Statement stmt = null;
    UniSQLPreparedStatement upstmt = null;
    ResultSet rs = null;

    try {
      // UniSQL DB Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

      conn.setAutoCommit(false);

      String sql = "select car from user_info where name = '홍길동'";
      stmt = conn.createStatement();
      rs = stmt.executeQuery(sql);
      UniSQLOID user_car = null;

      // '홍길동'이란 이름을 가진 사람이 가진 자동차의 OID를 가지고 온다.
      rs.next();
      user_car = (UniSQLOID)rs.getObject("car");
      // OID를 통해 직접 UPDATE.
      sql = "update object ? set name = 'SM5', spec ='2000cc, ABS, DUAL AIRBAG', color = '검정색'";
// ①
      upstmt = (UniSQLPreparedStatement)conn.prepareStatement(sql);          // ②
      upstmt.setOID(1,user_car);
      upstmt.executeUpdate();

      rs.close();
      upstmt.close();
      stmt.close();
      conn.commit();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 이 코드는 홍길동의 자동차 정보를 갱신한다.
- ①: 먼저 홍길동의 자동차 OID를 가져오고 **update object OID**를 통해 정보를 갱신한다.
- ② : update object 질의에는 OID가 들어가므로 3.1.2.1과 같이 **UniSQLPreparedStatement**를 사용한다.

3.1.4.2 OID를 이용한 데이터 갱신

이 예제는 3.1.4.1처럼 질의를 이용하지 않고 UniSQL JDBC API를 이용하여 데이터를 갱신한다.

```
import java.sql.*;
import unisql.sql.*;

public class updateOID2 {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
      // UniSQL DB Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

      // '홍길동'이란 이름을 가진 사람이 가진 자동차의 OID를 가지고 온다.
      String sql = "select car from user_info where name = '홍길동'";
      stmt = conn.createStatement();
      rs = stmt.executeQuery(sql);
      UniSQLOID user_car = null;
      rs.next();
      user_car = (UniSQLOID)rs.getObject("car");

      // OID를 통해 질의를 수행하지 않고 데이터를 UPDATE.
      String [] attr  = { "name", "spec", "color"};                    // ①
      Object [] val = { "SM525V", "2500cc LSD ABS", "검정색"};              // ②
      user_car.setValues(attr, val);                 // ③
      rs.close();
      stmt.close();
      conn.commit();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 이 코드는 3.1.4.1과 같은 일을 수행한다.
- ①: update할 attribute들을 array에 저장한다.
- ②: attr array의 순서에 맞춰서 값을 array에 저장한다.
- ③: **setValues**를 통해 테이터를 갱신한다.

3.2 SET 사용

본 장에서는 JDBC에서 SET을 사용하는 코드에 대해 설명한다. 코드를 보기에 앞서 SET이 무엇인지 사전에 알고 있어야 한다. 아래는 이 장에서 사용될 DB의 스키마이다. user_info2 class의 car attribute는 다수의 car2 class들을 가리키고 있다. 또 car2 class의 spec attribute는 string의 집합으로 되어 있다.

```
CREATE CLASS car2 (
name char(40),
spec  set_of(string)
)
CREATE CLASS user_info2 (
name char(20),
addr  string,
car    set_of(car2)
)
```

3.2.1 데이터 조회

SET 데이터를 조회하는 방법에는 질의상에서 derived table을 이용하는 방법과 Collection객체를 사용하는 방법이 있다. 두 가지 모두 SET 데이터를 조회하는 방법이자만 결과가 조금 다름에 유의해야 한다.

3.2.1.1 derived table을 사용한 SET 타입의 데이터 조회

질의 상에서 derived table을 통해 SET 데이터를 조회한다.

```
import java.sql.*;

public class selectSET1 {
  public static void main(String[] args) throws Exception {

      Connection conn = null;
      Statement stmt = null;
      ResultSet rs = null;

      try {
        // UniSQL DB Connect
        Class.forName("unisql.jdbc.driver.UniSQLDriver");
        conn =
          DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

        // 아래의 질의는 SET 타입의 데이터를 derived table로 나누어서 값을 가지고 온다.
        String sql = "select name, addr, car2.name from user_info2, table(car) as t(car2)";    // ①
        stmt = conn.createStatement();
        rs = stmt.executeQuery(sql);
        while(rs.next()) {
          String user_name = rs.getString("name");
          String user_addr = rs.getString("addr");
          String car_name = rs.getString("car2.name");

          System.out.println("이름 : " + user_name);
          System.out.println("주소 : " + user_addr);
          System.out.println("자동차 : " + car_name);
          System.out.println("\n================================\n");
        }
        rs.close();
        stmt.close();
      } catch ( SQLException e ) {
        System.err.println(e.getMessage());
      } catch ( Exception e ) {
        System.err.println(e.getMessage());
      } finally {
        if ( conn != null ) conn.close();
```

```
    }
  }
}
```

- **derived table**은 질의 수준에서 SET값을 조회하는 방법이다. 이 방법은 car attribute를 **derived table**로 만들어 조인형태의 값을 얻어온다.

- ①: 이 질의는 car attribute를 t라는 **derived table**의 car2라는 attribute로 만들어서 값을 얻어오는데 다음과 같이 조인형태의 값을 가져오게 된다. (3.2.1.2와 비교)

홍길동	SM5
홍길동	레죠
홍길동	그랜져XG
김삿갓	쏘렌토
김삿갓	매그너스

3.2.1.2 Collection을 사용한 SET 타입의 데이터 조회

Collection 객체를 이용하면 3.2.1.1과 마찬가지로 SET 타입의 데이터를 조회할 수 있다. 하지만 결과가 조인 형태인 3.2.1.1과는 조금 다르다.

```
import java.sql.*;
import unisql.jdbc.driver.*;
import unisql.sql.*;

public class selectSET2 {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    Statement stmt = null;
    UniSQLResultSet rs = null;
    UniSQLResultSet urs = null;

    try {
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");
      conn.setAutoCommit(false);          // ①
      String sql = "select name, addr, car from user_info2";
      stmt = conn.createStatement();
      rs = (UniSQLResultSet)stmt.executeQuery(sql);

      while(rs.next()) {
        String user_name = rs.getString("name");
        String user_addr = rs.getString("addr");
        // 데이터 타입이 SET인 경우 아래와 같이 getCollection을 사용.
        // 아래는 OID의 SET이므로 UniSQLOID 의 array에 값을 저장.
        UniSQLOID[] car = (UniSQLOID[])rs.getCollection("car");         // ②
        System.out.println("이름 : " + user_name);
        System.out.println("주소 : " + user_addr);
        // array에 저장한 값을 가지고 실제 값을 가지고 온다.
        for(int i=0; i<car.length; i++) {
          // getValues의 파라미터로 null을 주면 전체 attribute를 가지고 온다.
          urs = (UniSQLResultSet)car[i].getValues(null);             // ③
          urs.next();
          String car_name = urs.getString("name");
                   System.out.println("소유자동차 : " + car_name);
        }
        System.out.println("\n================================\n");
      }
      urs.close();
      rs.close();
      stmt.close();
      conn.commit();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 이 코드는 derived table을 사용하지 않고 Collection을 이용하여 SET값을 조회하는 방법이다.

- ①: 한 개의 ResultSet이 close()되기 전에 다른 ResultSet을 생성할 경우 앞의 ResultSet은 autoCommit되어 close()된다. 따라서 이 코드와 같이 동시에 여러 개의 ResultSet이 필요한 경우엔 반드시 autoCommit을 false로 설정해야 한다.

- ②: 데이터 조회한 결과값이 SET 타입인 경우, **getCollection**을 이용하여 값을 얻는다. 이 결과는 array 형태로 저장된다.

- ③: array에 저장된 값을 for문을 통해 1개씩 불러온다. 여기에서는 저장된 값이 **UniSQLOID** 이므로 **getValues**를 이용하여 실제 필요한 값을 얻을 수 있다. **getValues**의 파라미터는 값을 가져올 attribute들의 이름이지만 위의 예제처럼 null을 줄 수도 있다. null을 입력한 경우에는 그 class의 모든 attribute들의 값을 얻는다. (3.1.1.1 참조)

- 이와 같은 방법으로 값을 가져오면 다음과 같은 형태로 값을 얻을 수 있다. (3.2.2.1 비교)

홍길동	SM5
	레조
	그랜져XG

김삿갓	쏘렌토
	매그너스

3.2.2 데이터 입력

SET 타입의 데이터는 직접 질의로 입력하는 방법과 API를 이용하는 방법이 있다. API를 이용하는 경우에는 이미 입력된 OID를 사용하므로 정확히 말하면 데이터의 갱신에 속한다. 따라서 본 절에서는 다루지 않는다. 이 부분은 3.3.2에서 다루도록 한다.

3.2.2.1 SET 타입의 데이터 입력

본 예제는 insert 후에 리턴된 OID를 직접 { }의 SET 형태로 입력한다.

```
import java.sql.*;

public class insertSET {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    Statement stmt = null;

    try {
      // UniSQL DB Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

      // 데이터를 SET형식으로 직접 입력하려면 { }의 집합형태로 입력해야 한다.
      // 아래의 질의는 SET인 데이터들이 다른 클래스를 가리키고 있는데 아래와 같이 집합기호 안에
      // car2에 데이터를 입력하면서 동시에 그것들의 OID를 user_info2에 SET형식으로 입력한다.
      String sql = "insert into user_info2 values (" +
                  "'홍길동', '서울 강남구 역삼동 679-4', { insert into car2 values
('레죠',{'2000cc','은색'})," +
                                                  "insert into car2 values
('SM5',{'2500cc','노랑'})})";          // ①
      stmt = conn.createStatement();
      stmt.executeUpdate(sql);
      System.out.println("데이터가 입력되었습니다.");
      conn.commit();
      stmt.close();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- 일반적으로 SET은 { }를 통해서 입력한다. 예를 들면 *insert into car2(name, spec) values('SM5', { 'V6 2500cc', 'DUAL AIRBACK', 'ABS'})* 와 같이 입력한다. 그런데 이 코드에서는 데이터 타입이 set_of(string)이 아니라 set_of(car2)이다. 따라서 입력할 때 { OID, OID, OID } 형태가 되어야 한다. 이 코드는 OID를 직접입력하지 않고 car2 class에 데이터를 입력하면서 동시에 리턴된 OID를 이용하여 insert하는 방법을 보여준다.

3.2.3 데이터 갱신

SET 타입의 데이터를 갱신하는 방법에는 질의를 이용하는 방법과 API를 사용하는 방법이 있다. SET 자체로 볼 땐 데이터의 추가/삭제처럼 보이자만 전체 인스턴스로 볼 땐 데이터의 갱신이다.

3.2.3.1 질의를 이용한 SET 타입의 데이터 갱신

본 예제는 SET 타입의 데이터 리스트 중에서 한가지를 삭제하고 다른 한가지를 추가하는 작업을 질의를 통해 수행한다.

```
import java.sql.*;

public class updateSET {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    Statement stmt = null;

    try {
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");

      // SET 값 중에 한가지 값을 빼고 다른 값을 입력.
      String sql = "update car2 set spec = spec - {'은색'} + {'노란색'} where name = '누비라'";       // ①
      stmt = conn.createStatement();
      stmt.executeUpdate(sql);
      System.out.println("데이터가 수정되었습니다.");
      stmt.close();
      conn.commit();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
    }
  }
}
```

- ①: 이 질의는 spec attribute에서 '은색'을 지우고 '노란색'을 추가한다.

3.2.3.2 API를 이용한 SET 타입의 데이터 갱신

본 예제는 SET 타입의 데이터 리스트 중에서 한가지를 삭제하고 다른 한가지를 추가하는 작업을 API를 통해 수행한다.

```
import java.sql.*;
import unisql.jdbc.driver.*;
import unisql.sql.*;

public class updateSET2 {
  public static void main(String[] args) throws Exception {

    Connection conn = null;
    Statement stmt = null;
    UniSQLResultSet rs = null;

    try {
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
        DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");
      String sql = "select car from user_info2 where name = '홍길동'";
      stmt = conn.createStatement();
      rs = (UniSQLResultSet)stmt.executeQuery(sql);
      while(rs.next()) {
        UniSQLOID[] car = (UniSQLOID[])rs.getCollection("car");
        // API를 통해 SET 값에서 한가지 값을 지우고 다른 값을 입력.
        for(int i=0; i<car.length; i++) {
          car[i].addToSet("spec","노란색");
          car[i].removeFromSet("spec","검정색");           // ①
        }
      }
      rs.close();
      stmt.close();
      conn.commit();
    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      if ( conn != null ) conn.close();
     }
  }
}
```

- ①: 4.4처럼 SET값에서 '노란색'을 지우고 '검정색'을 입력하는 코드이다. OID를 알고 있을 경우엔 위의 코드처럼 질의를 수행하지 않고 **addToSet(attribute명, 데이터) / removeFromSet(attribute명, 데이터)** 을 이용하여 값을 갱신할 수 있다.

3.3 GLO 사용

GLO는 멀티미디어 데이터 파일을 데이터베이스에 입력할 수 있도록 한다. GLO에는 LO, FBO의 2가지 타입이 존재한다. 이 중 LO는 데이터 파일이 데이터베이스에 존재하며, FBO는 파일시스템에 존재한다.

GLO데이터는 GLO class에 입력한다. 본 장에서는 GLO class를 직접 이용하지 않고 상속 받아서 쓰도록 한다. 아래는 사용하게 될 class를 생성하는 구문이다.

```
Create class file_object under glo
```

본 장에서는 GLO 중 LO타입으로 데이터를 저장하는 방법 및 저장 데이터를 가져오는 방법을 알아본다.

3.3.1 GLO데이터 입력

본 절에서는 파일을 데이터베이스에 입력하는 방법을 설명한다. 아래의 예제는 시스템의 특정 경로에 있는 파일을 InputStream으로 만들어서 file_object class에 입력한다.

```
import java.sql.*;
import java.io.*;
import unisql.sql.*;
import unisql.jdbc.driver.*;

public class insertLO {
  public static void main(String[] args) throws Exception {

    UniSQLConnection conn = null;
    FileInputStream inputfile = null;

    try {

      // UniSQL DB에 Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
(UniSQLConnection)DriverManager.getConnection("jdbc:unisql:192.168.0.10:43300:demodb:::","","");
… ①
      // file을 InputStream으로 만든다.
      inputfile = new FileInputStream("/home/unisql/file/test.txt");          … ②
      // file_object class에 데이터를 입력한다.
      UniSQLOID glo_oid = conn.getNewGLO("file_object", inputfile);           … ③
      inputfile.close();
      conn.commit();
      conn.close();

    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      try {
       if ( conn != null ) conn.close();
      } catch ( SQLException e ) {
       System.err.println(e.getMessage());
      }
    }
  }
}
```

- ① GLO를 사용할 때에는 반드시 **UniSQLConnection**을 사용한다.
- ② 시스템 상의 파일을 데이터베이스에 GLO로 입력하기 위해 InputStream으로 만든다.
- ③ **getNewGLO()** API를 이용해 파일을 데이터베이스에 저장한다.
- getNewGLO("class명", InputStream) : InputStream화한 파일을 지정된 class에 저장시킨다. 저장 후 UniSQLOID를 리턴한다.

3.3.2 GLO 데이터 가져오기

본 절에서는 데이터베이스에 저장된 GLO 데이터를 가져오는 방법을 설명한다. 다음의 예제는 데이터베이스에 저장된 GLO 데이터를 시스템 상의 특정 경로에 파일로 만들어준다.

```
import java.sql.*;
import java.io.*;
import unisql.sql.*;
import unisql.jdbc.driver.*;

public class selectGLO {
  public static void main(String[] args) throws Exception {

    UniSQLConnection conn = null;
    FileOutputStream outputfile = null;

    try {

      // UniSQL DB에 Connect
      Class.forName("unisql.jdbc.driver.UniSQLDriver");
      conn =
(UniSQLConnection)DriverManager.getConnection("jdbc:unisql:210.216.33.250:52500:shindb:::","","");
… ①
      // GLO파일의 OID를 얻어온다.
      String sql = "select file_object from file_object";
      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery(sql);
      rs.next();
      UniSQLOID glo_oid = (UniSQLOID)rs.getObject("file_object");        … ②

      // loadGLO를 이용해 지정 경로에 GLO를 파일로 만들어 준다.
      outputfile = new FileOutputStream("/home/unisql/file/tmp000");          … ③
      glo_oid.loadGLO(outputfile);                              … ④

      outputfile.close();
      rs.close();
      conn.commit();
      conn.close();

    } catch ( SQLException e ) {
      System.err.println(e.getMessage());
    } catch ( Exception e ) {
      System.err.println(e.getMessage());
    } finally {
      try {
        if ( conn != null ) conn.close();
      } catch ( SQLException e ) {
        System.err.println(e.getMessage());
      }
    }
  }
}
```

- ■ ① UniSQLConnection을 사용한다.
- ■ ② 가져올 파일의 OID를 얻는다.
- ■ ③ OutPutStream을 생성한다. 이때 저장할 경로 및 파일명을 설정한다.
- ■ ④ loadGLO() API를 이용해 데이터베이스의 GLO 데이터를 파일로 만든다.
- ■ loadGLO(OutputStream) : 데이터베이스의 GLO데이터를 OutPutSteeam화 한다.