



08.

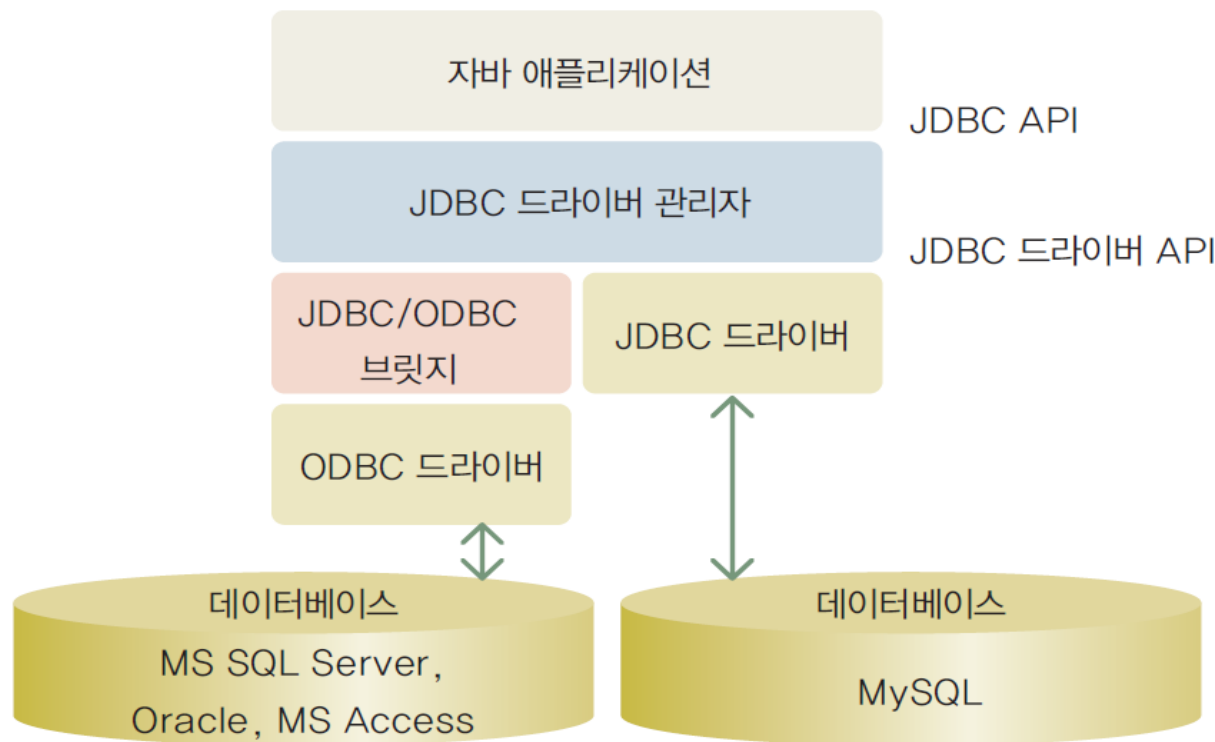
JDBC 프로그래밍

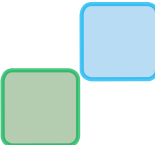
- ✓ JDBC 개념
- ✓ JDBC Programming 단계
 - Statement, PreparedStatement, CallableStatement
 - ResultSet, ResultSetMetaData
 - JDBC 예외처리
 - 샘플예제(테이블 스키마 활용, Date 타입 활용)
- ✓ 자바 GUI for JDBC Programming

JDBC

□ JDBC(Java Database Connectivity) :

- ▶ 데이터베이스에 연결하여 검색하고 변경할 수 있게 하는 **표준 API**
- ▶ 자바 프로그램에서 서로다른 DBMS라도 **같은 방법으로 접근 가능**





<https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>

Java JDBC API

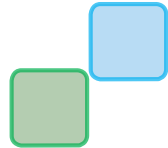
The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which tools and alternate interfaces can be built.

The JDBC API is comprised of two packages:

- `java.sql`
- `javax.sql`

You automatically get both packages when you download the Java Platform Standard Edition (Java SE) 8.

To use the JDBC API with a particular database management system, you need a JDBC technology-based driver to mediate between JDBC technology and the database. Depending on various factors, a driver might be written purely in the Java programming language or in a mixture of the Java programming language and Java Native Interface (JNI) native methods. To obtain a JDBC driver for a particular database management system, see [JDBC Data Access API](#).



java.security.spec
java.sql
java.text
java.text.spi
java.time
java.time.chrono
java.time.format
java.time.temporal
java.time.zone
java.util
java.util.concurrent
java.util.concurrent.atomic
java.util.concurrent.locks
java.util.function
java.util.jar
java.util.logging

java.sql

Interfaces

Array
Blob
CallableStatement
Clob
Connection
DatabaseMetaData
Driver
DriverAction
NClob
ParameterMetaData
PreparedStatement
Ref
ResultSet
ResultSetMetaData
RowId
Savepoint
SQLData
SQLInput
SQLOutput
SQLType
SQLXML
Statement
Struct
Wrapper

Classes

Date
DriverManager
DriverPropertyInfo
SQLPermission

Package java.sql

Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java™ programming language.

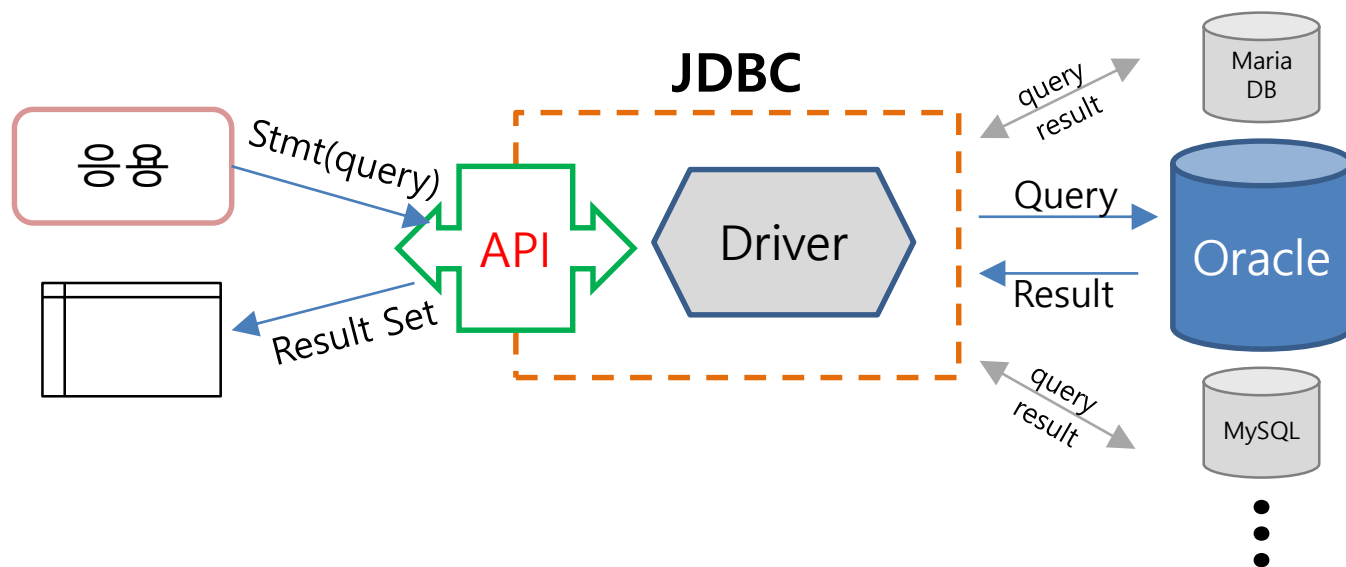
See: Description

Interface Summary	
Interface	Description
Array	The mapping in the Java programming language for the SQL type ARRAY.
Blob	The representation (mapping) in the Java™ programming language of an SQL BLOB value.
CallableStatement	The interface used to execute SQL stored procedures.
Clob	The mapping in the Java™ programming language for the SQL CLOB type.
Connection	A connection (session) with a specific database.
DatabaseMetaData	Comprehensive information about the database as a whole.
Driver	The interface that every driver class must implement.
DriverAction	An interface that must be implemented when a Driver wants to be notified by DriverManager.
NClob	The mapping in the Java™ programming language for the SQL NCLOB type.
ParameterMetaData	An object that can be used to get information about the types and properties for each parameter marker in a PreparedStatement object.
PreparedStatement	An object that represents a precompiled SQL statement.
Ref	The mapping in the Java programming language of an SQL REF value, which is a reference to an SQL structured type value in the database.
ResultSet	A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
ResultSetMetaData	An object that can be used to get information about the types and properties of the columns in a ResultSet object.
RowId	The representation (mapping) in the Java programming language of an SQL ROWID value.
Savepoint	The representation of a savepoint, which is a point within the current transaction that can be referenced from the Connection.rollback method.



데이터베이스 프로그램 개발 절차

- ① DBMS(DataBase Management System)를 설치
- ② 자신이 설치한 DBMS에 필요한 JDBC 드라이버를 설치한다.
- ③ JDBC가 제공하는 기능을 이용해 DB 응용 프로그램을 개발한다.




JDBC 드라이버 다운로드

□ JDBC 드라이버

- ▶ 각 DBMS 제조업체 홈페이지에서 제공
- ▶ 오라클의 JDBC 드라이버
 - <https://www.oracle.com/database/technologies/jdbcdriver-ucp-downloads.html>
 - 11g R2의 드라이버 다운로드 : ojdbc6.jar

Oracle Database 11g Release 2 (11.2.0.4) JDBC Drivers & UCP Downloads


Zipped JDBC Driver and Companion JARs

Download	Release Notes
 ojdbc-full.tar.gz	(6,761,477 bytes) - (SHA1 Checksum: 1ce3d1055b94ee1c6148d74a440c937d0a2df30e)

The TAR archive contains the latest 11.2.0.4 JDBC Thin driver (**ojdbc6.jar** and **ojdbc5.jar**), Universal Connection Pool (**ucp.jar**), other companion jars, and README that has more information about the contents of the tar file.

Unzipped JDBC Driver and Companion JARs

The JARs included in the **ojdbc-full.tar.gz** are also available as individual downloads in this section.

Download	Release Notes
 ojdbc6.jar	(2,739,670 bytes) - (SHA1 Checksum: a483a046eee2f404d864a6ff5b09dc0e1be3fe6c) Certified with JDK8, JDK7, and JDK6;

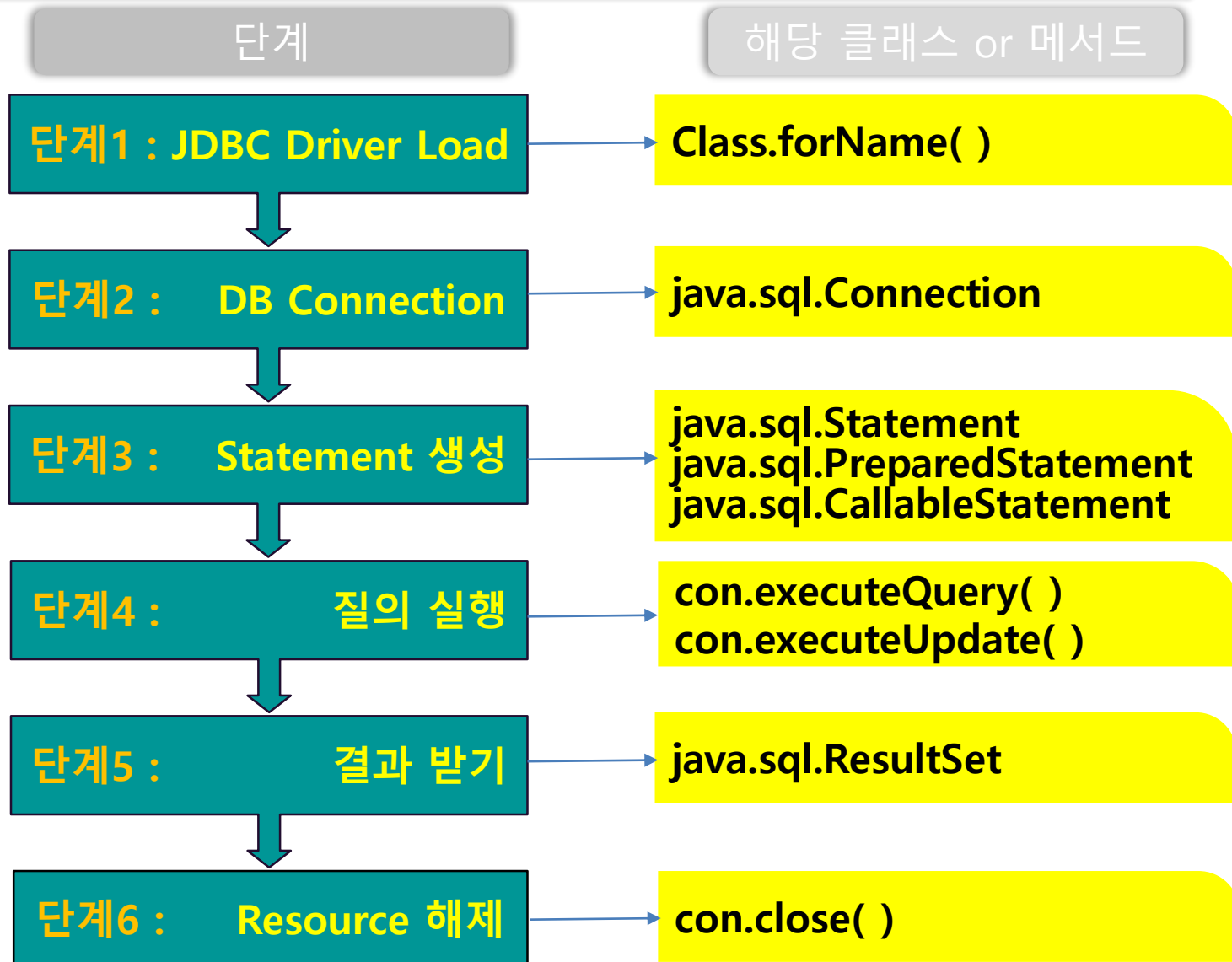


JDBC 드라이버 설치

□ 3가지 방법이 있음

- ❶ [java 설치 디렉터리\jre\lib\ext]에 복사하는 방법
 - ❷ 시스템 변수에 CLASS_PATH를 생성. 드라이버가 있는 경로를 CLASS_PATH의 값으로 지정
 - ❸ 프로젝트이름에서 팝업:[File|Properties|Java Build Path|Libraries|Add Ext. Jars] 설정
- ▶ 방법1을 많이 사용함

JDBC 프로그래밍 단계





단계 1 : JDBC 드라이버 적재

□ jdbc.drivers 환경변수 이용하는 방법

ORACLE : `System.setProperty("jdbc.drivers", "oracle.jdbc.OracleDriver");`

MariaDB : `System.setProperty("jdbc.drivers", "com.mariadb.jdbcDriver");`

□ Class.forName() 메서드 이용하는 방법

▶ 프로그램 내에서 처리하기 용이한 방법

ORACLE : `Class.forName("oracle.jdbc.OracleDriver");`

MariaDB : `Class.forName("com.mariadb.jdbcDriver");`

단계2 : 데이터베이스 연결

□ DriverManager클래스의 getConnection()메서드 이용

- ▶ 메서드 인자 : 접속url, id, password
 - 접속은 thin Client 방식을 이용함
 - DBMS별로 url이 다름

```
String url = "jdbc:oracle:thin:@localhost:1521:XE";  
String id = "hmart";  
String password = "1234";  
con = DriverManager.getConnection(url, id, password);
```

```
String url = "jdbc:mysql://IP주소:3306/db이름";  
String id = "hmart";  
String password = "1234";  
con = DriverManager.getConnection(url, id, password);
```

예외 처리를 고려한 단계1, 2의 통합(오라클)

```
public class DB_Conn_Query {
    Connection con = null;
    public DB_Conn_Query() {
        String url = "jdbc:oracle:thin:@localhost:1521:XE";
        String id = "hmart";
        String password = "1234";
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("드라이버 적재 성공");
            con = DriverManager.getConnection(url, id, password);
            System.out.println("데이터베이스 연결 성공");
        } catch (ClassNotFoundException e) {
            System.out.println("드라이버를 찾을 수 없습니다.");
        } catch (SQLException e) {
            System.out.println("연결에 실패하였습니다.");
        }
    }
}
```

예외 처리를 고려한 단계1, 2의 통합(MySQL)

```
01 import java.sql.*;
02 public class ConnectDatabase {
03     public static Connection makeConnection()
04     {
05         String url = "jdbc:mysql://localhost/book_db";
06         String id = "root";
07         String password = "password";
08         Connection con = null;
09         try {
10             Class.forName("com.mysql.jdbc.Driver");
11             System.out.println("드라이버 적재 성공");
12             con = DriverManager.getConnection(url, id, password);
13             System.out.println("데이터베이스 연결 성공");
14         } catch (ClassNotFoundException e) {
15             System.out.println("드라이버를 찾을 수 없습니다.");
16         } catch (SQLException e) {
17             System.out.println("연결에 실패하였습니다.");
18         }
19     }
20 }
```

jdbc 드라이버 클래스를
찾는다

단계 3, 4 : Statement 생성 + SQL 전송

□ Statement 클래스 : SQL문을 수행할 수 있도록 해주는 클래스

▶ 주요메서드

▣ executeQuery() : SELECT문사용. 반환값이 ResultSet

```
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("select * from 고객");
```

▣ executeUpdate() : 변경연산에 사용. 반환값은 int(처리된 데이터 수)

```
String query = "update 고객 set 적립금 = 6000 where 고객이름 = '정소화' ";  
Statement stmt = con.createStatement();  
int count = stmt.executeUpdate(query);
```

▶ 쿼리를 문자열로 조합해야 하므로 소스가 복잡하고 오류가 발생

레코드 수정, 삭제의 반환값 확인

```
18      System.out.println(s);
19      int i = stmt.executeUpdate(s);
20      if (i == 1)
21          System.out.println("레코드 추가 성공");
22      else
23          System.out.println("레코드 추가 실패");
24      } catch (SQLException e) {
25          System.out.println(e.getMessage());
26          System.exit(0);
27      }
28  }
29  }
```

→ 레코드를 수정할때 사용.

Statement의 메서드

□ SQL문 종류에 따라 사용되는 메서드

메서드 Signature	기능
ResultSet executeQuery (String sql)	Select SQL문에 사용. ResultSet반환
int executeUpdate (String sql)	create, insert, delete, update 등의 갱신문에 사용. 적용된 행의 갯수 반환
Boolean execute (String sql)	모든 SQL문 사용가능. 첫번째 결과가 ResultSet객체이면 반환값이 true
void close ()	리소스 해제



Statement의 단점

- 질의 구성이 복잡
- 컴파일 타임에는 오류를 발견할 수 없음
 - ▶ 오류 수정이 어려움
- SQL을 실행할 때마다 질의를 해석하는 오버헤드가 있음

⇒⇒ 이런 단점을 극복하기 위해 새로운 방법 도입

- ▶ **PreparedStatement** (Statement로 부터 상속)
 - 파라미터 개념 도입
 - 질의문은 미리 컴파일. 속도 향상
 - 질의문 구성이 쉬움
- ▶ **CallableStatement** (PreparedStatement로 부터 상속)
 - 저장 프로시저 실행 가능(속도, 소스코드의 독립성, 보안성)

단계 3, 4 : PreparedStatement + SQL 전송

- 질의에 필요한 변수 데이터를 “?”로 표시하고 메서드를 통해 설정
 - ▶ Statement보다 구조적이고, 편리

```
Statement stmt = con.createStatement();  
stmt.executeUpdate( "insert into 고객 values (" +  
    textbox1.getText( ) + "," + textbox2.getText( ) + "," +  
    textbox3.getText( ) + "," + textbox4.getText( ) + ")" );
```

```
PreparedStatement pstmt =  
    conn.prepareStatement("insert into test values(?,?,?,?)");  
pstmt.setString(1, textbox1.getText() );  
pstmt.setString(2, textbox2.getText() );  
pstmt.setInt(3, Integer.parseInt(textbox3.getText()) );  
pstmt.setString(4, textbox4.getText() );  
pstmt.executeUpdate();
```

PreparedStatement의 메서드

메서드 Signature	기능
ResultSet executeQuery (String sql)	Select SQL문에 사용. ResultSet반환
int executeUpdate (String sql)	갱신문에사용. 적용된 행의 갯수 반환
Boolean execute (String sql)	모든 SQL문 가능.
void close ()	리소스 해제
void setXxx (int 인자번호, Type a)	a값을 인자 번호 위치의 값으로 설정 Xxx는 Type(DB타입)에 대응되는 자바 자료형
void setObject (int 인자번호, Object o)	객체 o를 번호에 인자 번호 위치의 값으로 설정
void setNull (int 인자번호, int sqlType)	인자 번호 위치에 NULL값을 설정
ResultSetMetaData getMetaData ()	반환되는 ResultSet의 메타데이터 정보 반환

CallableStatement 인터페이스

public interface **CallableStatement** extends [PreparedStatement](#)

PreparedStatement의 모든 메서드 사용 가능

□ SQL객체(함수, 저장프로시저, 패키지 등)를 자바 프로그램에서 실행시킬 수 있는 인터페이스 제공

▶ 장점 :

- SQL문이 DB서버에서 관리됨. 자바 프로그램이 간단해짐
- 미리 컴파일되어 있으므로 실행시간 단축. **주기적 실행 연산일 경우 효과적**
- 네트워크 트래픽 감소. 함수 호출구문만 전송됨
- **복잡한 검색 연산의 효율적인 표현**

□ SQL객체(함수, 저장 프로시저 등) 생성방법

- (1) 서버에 접속해 생성(이미 학습)
- (2) 자바 프로그래밍으로 Statement객체를 이용하여 생성

SQL객체 생성 및 실행(1) in 자바 프로그램

□ 저장 프로시저 생성 : without Parameter

```
String CP2 = "CREATE PROCEDURE SP_고객보기2( ) " +  
            "BEGIN select * from 고객; END;";  
Statement stmt = con.createStatement( );  
stmt.executeUpdate(CP);
```

□ 저장 프로시저 실행

```
CallableStatement cstmt = con.prepareCall( "{call SP_고객보기}" );  
ResultSet rs = cstmt.executeQuery( );
```

[복잡한 검색 연산의 효율적인 표현]

조인, 뷰, 부질의 등 어렵고 복잡한 검색 연산을 저장 프로시저로 저장하고,
CallableStatement의 executeQuery()로 실행하여 결과를 ResultSet으로 받아 처리

SQL객체 생성 및 실행(2) in 자바 프로그램

□ 저장 프로시저 생성 : with Parameter

```
String CP = "CREATE OR REPLACE PROCEDURE SP_In_Java(" +  
    "Pi_ID IN CHAR," +  
    "Pi_Name IN CHAR)" +  
    "BEGIN" +  
    "INSERT INTO 고객(고객아이디, 고객이름, 등급) VALUES(Pi_ID, Pi_Name, 'Silver');" +  
    "END ;"  
Statement stmt = con.createStatement( );  
stmt.executeUpdate(CP);
```

□ 저장 프로시저 실행

```
CallableStatement cstmt = con.prepareCall( "{call SP_In_Java(?,?) }" );  
cstmt.setString(1, "tomato");  
cstmt.setString(2, "홍안");  
cstmt.executeUpdate( );
```

일괄 갱신 in JDBC

- 트랜잭션에서 다룸

[Statement 일괄 갱신]

```
con.setAutoCommit(false);  
Statement stmt = con.createStatement();  
stmt.addBatch("insert into R1 values(1,'kim')");  
stmt.addBatch("update R1 set ID=100 where name = 'kim' ");  
int[ ] cnt = stmt.executeBatch();  
con.commit( );
```

[PreparedStatement 일괄 갱신]

```
con.setAutoCommit(false);  
PreparedStatement pstmt =  
    con.prepareStatement("update R1 set ID=? where name=?");  
pstmt.setInt(1,200); pstmt.setString(2,"kim");  
pstmt.addBatch();  
pstmt.setInt(1,300); pstmt.setString(2,"lee");  
pstmt.addBatch( );  
int[ ] cnt = pstmt.executeBatch( );  
con.commit( );
```

단계 5 : 검색 결과 받기

❑ executeQuery의 결과 : ResultSet rs

- ▶ ResultSet : 튜플의 집합체
- ▶ rs : 커서(포인터)
 - rs.next()를 수행할 때마다 rs는 다음 튜플을 가리킴
 - rs.getXxx()를 수행하여 데이터를 Read

getString()
getInt()
getDouble()
getDate()
...

1 rs

2 rs.next()
rs.next()



	고객아이디	고객이름	나이	등급	직업	적립금
1	apple	정소화	20	gold	학생	1000
2	banana	김선우	25	vip	간호사	2500
3	carrot	고명석	28	gold	교사	4500
4	orange	김용욱	22	silver	학생	0
5	melon	성원웅	35	gold	회사원	5000
6	peach	오형준	(null)	silver	의사	300
7	pear	채광주	31	silver	회사원	500
8	star	홍길동	30	Silver	학생	4500

ResultSet의 메서드

<JDT> : java data type

메서드 Signature	기능
<JDT> getXxx (int 인자번호)	인자번호에 해당하는 컬럼의 값을 반환 Xxx는 JDT에 대응하는 자료형
<JDT> getXxx (String 컬럼명)	컬럼명에 해당하는 컬럼의 값을 반환
object getObject (int 인자번호)	현재 행의 컬럼번호에 해당하는 값을 객체로 반환
object getObject (String 컬럼명)	현재 행의 컬럼명에 해당하는 값을 객체로 반환
void close ()	리소스 해제
int getRow ()	현재 행의 번호를 반환
Boolean first () / Boolean isFirst ()	커서를 첫번째행으로/ 첫번째 행 참조 여부 반환
Boolean last () / Boolean isLast ()	커서를 마지막행으로/ 마지막행 참조여부 반환
Boolean next () / Boolean previous ()	커서를 다음행(이전)으로(존재하지 않으면 false)
void updateXxx (int 인자번호, <JDT> x) void updateXxx (string 컬럼명, <JDT> x)	인자번호(컬럼명)에 해당하는 값을 x로 변경
void updateRow ()	참조하고 있는 행을 새로운 값으로 변경
ResultSetMetaData getMetaData ()	ResultSet의 컬럼유형, 속성, 컬럼 수 등을 반환



단계 6 : 리소스 해제

- ❑ `stmt.close();`
- ❑ `pstmt.close();`
- ❑ `rs.close();`
- ❑ `con.close();`

Simple JDBC Program

```
public class DB_Conn_Query {
    Connection con = null;
    public DB_Conn_Query() {
        String url = "jdbc:oracle:thin:@localhost:1521:XE";
        String id = "hmart";    String password = "1234";
        try { Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("드라이버 적재 성공");
            con = DriverManager.getConnection(url, id, password);
            System.out.println("DB 연결 성공");
        } catch (ClassNotFoundException e) {      System.out.println("No Driver.");    }
        catch (SQLException e) {      System.out.println("Connection Fail");    }
    }

    private void sqlRun() {
        String query = "select 고객아이디, 고객이름, 적립금 from 고객";
        try { Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
                System.out.print("\t" + rs.getString("고객아이디"));
                System.out.print("\t" + rs.getString("고객이름"));
                System.out.print("\t" + rs.getInt(3) + "\n");
            }
            stmt.close();  rs.close();  con.close();
        } catch (SQLException e) { e.printStackTrace(); }
    }

    public static void main(String arg[]) throws SQLException {
        DB_Conn_Query dbconquery = new DB_Conn_Query();
        dbconquery.sqlRun();
    }
}
```





ResultSetMetaData 클래스

□ 테이블의 스키마 정보를 검색할 때 사용하는 클래스

- ▶ 테이블의 컬럼의 갯수, 컬럼명, 데이터 타입 등의 정보를 가짐
- ▶ 다양한 메서드 이용
 - getColumnCount() : 컬럼의 갯수
 - getColumnName(columnIndex) : 컬럼명
 - getColumnTypeName(columnIndex) : 컬럼의 데이터 타입
 - getPrecision(columnIndex) : 컬럼의 크기

□ ResultSetMetaData 객체 획득 방법

```
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("select * from 고객");  
ResultSetMetaData schema = rs.getMetaData() ;
```

ResultSetMetaData API

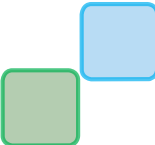
<JDT> : java data type

메서드 Signature	기능
int getColumnCount()	컬럼의 갯수
String getColumnName(int 인자번호)	인자번호에 해당하는 컬럼의 이름을 반환
String getColumnTypeNames(int 인자번호)	인자번호에 해당하는 컬럼의 DB 데이터 타입 이름
<JDT> getColumnClassName(int 인자번호)	인자번호에 해당하는 컬럼에 대응하는 자바 클래스 이름
int getPrecision(int 인자번호)	컬럼의 크기. string이면 문자열의 길이. 숫자이면 전체 자리수
int Scale(int 인자번호)	인자번호에 해당하는 컬럼의 소수점이하 자리수

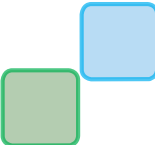
Example : 테이블 스키마 메타 정보 활용

❖ 'A' 테이블로부터 'B'테이블을 만들고 내용을 복사함

```
public class TableInfo {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        if ( args.length < 2)  
        {  
            System.out.println("Usage : java TableInfo source dest");  
            System.exit(0);  
        }  
        Connection con = null;  
        String url = "jdbc:oracle:thin:@localhost:1521:XE";  
        try {  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
            con = DriverManager.getConnection(url, "hmart", "1234");  
        } catch (ClassNotFoundException e) {  
            System.out.println("드라이버를 찾을 수 없습니다.");  
        } catch (SQLException e) {  
            System.out.println("연결에 실패하였습니다.");  
        }  
        if( con == null ) return;  
    }  
}
```



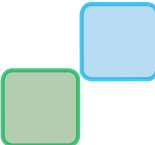
```
try {
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from " + args[0]);
    Copy_Table copytable = new Copy_Table( );
    copytable.create(con, args[1], rs);
    copytable.insert(con, args[1], rs);
    rs.close();
    stmt.close();
} catch(SQLException e) {
    System.out.println(e.getMessage());
} finally {
    try {
        con.close();
    } catch(SQLException e) { }
}
}
```



```
public class Copy_Table {
    private Statement stmt = null;
    private PreparedStatement pstmt = null;

    public void create(Connection con, String tableName, ResultSet rs) {
        try {
            ResultSetMetaData schema = rs.getMetaData();
            String query = "create table " + tableName + " (";

            for (int i = 1; i <= schema.getColumnCount(); i++) {
                query += schema.getColumnName(i) + " " + schema.getColumnTypeName(i);
                query += "(" + schema.getPrecision(i) + ")";
                if (i != schema.getColumnCount()) query += ",";
            }
            query += ")";
            stmt = con.createStatement();
            stmt.executeUpdate(query);
            stmt.close();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



```
public void insert(Connection con, String tableName, ResultSet rs) {
    try {
        int numCol = rs.getMetaData().getColumnCount();
        String query = "insert into " + tableName + " Values (";
        for ( int i = 1; i < numCol; i++) {
            query += "? ,";
        }
        query += "? )";
        pstmt = con.prepareStatement(query);
        while(rs.next()) {
            for ( int i = 1; i <= numCol; i++)
                pstmt.setObject(i, rs.getObject(i));
            pstmt.executeUpdate();
        }
        System.out.println("복사 완료");
        pstmt.close();
    } catch(SQLException e) {
        System.out.println(e.getMessage());
    }
}
```


SQL 자료형과 자바 자료형의 매핑

□ JDBC프로그래밍에서는

▶ DB자료형을 자바 자료형에 맞게 변환해줌

▣ 단, 프로그래머는 적절한 자바 자료형을 선택해야 변환해 줌

- 객체테이블의 '고객이름'을 검색했을 때는 getString() 메서드 사용해야 함

DB : 고객이름 - VARCHAR2(10)
JAVA : getString(1);

✓ 이 때 getInt()를 사용하면 안 됨

Oracle 주요 데이터 타입 매핑 테이블

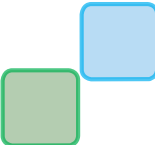
Oracle Type	getXxx 메서드	자바 Type
number(p,s)	getDouble()	double
number(p)	getInt()	int
char, varchar2	getString()	String
nchar, nvarchar	getString()	String
date	getDate()	String, Date(java.sql.Date)
blob	getBlob()	Blob
timestamp	getTimestamp()	Timestamp(java.sql.Timestamp)
clob, nclob	getClob()	Clob

Date, TimeStamp 사용 예제

+ Blob 예제

```
//DateDriver.java
class DBConn {
    public Connection connect() {
        Connection con = null;
        String url = "jdbc:oracle:thin:@localhost:1521:XE";
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con = DriverManager.getConnection(url, "hmart", "1234");
        } catch (ClassNotFoundException e) { }
        catch (SQLException e) { }
        return con;
    }
}

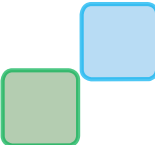
public class DateDriver {
    public static void main(String[] args) throws SQLException {
        // TODO Auto-generated method stub
        Connection con = null;
        DBConn db = new DBConn();
        con = db.connect();
        if( con == null ) return;
        DateType dateType = new DateType(con);
        dateType.create();
        dateType.insert("apple", "p06");
        dateType.insert("banana", "p01");
        dateType.select();
        con.close();
    }
}
```



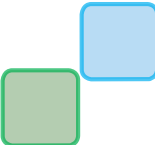
```
// DataType.java
public class DataType {
    Connection con = null;
    Statement stmt = null;
    PreparedStatement pstmt = null;

    public DataType(Connection con) {
        this.con = con;
    }

    public void create() {
        String query = "drop table 임시주문";
        try {
            stmt = con.createStatement();
            stmt.executeUpdate(query);
        } catch (SQLException e) {
            System.out.println("테이블삭제오류 : "+e.getMessage());
        }
        query = "drop sequence idSEQ";
        try {
            stmt.executeUpdate(query);
        } catch (SQLException e) {
            System.out.println("시퀀스삭제오류 : "+e.getMessage());
        }
        // Oracle : create or replace table 없음.
        // Oracle : create table if not exists 없음.
    }
}
```

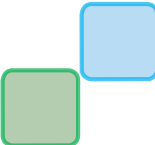


```
query = "create sequence idSEQ";
try {
    stmt.executeUpdate(query);
} catch(SQLException e) {
    System.out.println("시퀀스생성오류 : "+e.getMessage());
}
query = "create table 임시주문 (" +
    "주문번호 NUMBER(10)," +
    "주문자 VARCHAR2(20)," +
    "구매상품 VARCHAR2(20)," +
    "기록시작일 DATE," + //오라클은 Time type이 없음
    "주문일시 TIMESTAMP )";
try {
    stmt.executeUpdate(query);
    stmt.close();
} catch(SQLException e) {
    System.out.println("테이블생성오류 : "+e.getMessage());
}
}
```



```
public void insert(String 주문자, String 주문상품) {
    long milli = System.currentTimeMillis();
    java.util.Date date1 = new java.util.Date(milli);
    //System.out.println(date1); // sql.Date는 util.Date의 sub

    String query = "insert into 임시주문 values(idSeq.NEXTVAL,?,?,?,?)";
    try {
        pstmt = con.prepareStatement(query);
        pstmt.setString(1, 주문자);
        pstmt.setString(2, 주문상품);
        //pstmt.setDate(3, new java.sql.Date(milli)); //현재날짜
        pstmt.setDate(3, java.sql.Date.valueOf("2019-08-20"));
        // 특정 날짜 입력하고 싶을 때 valueOf( ) : static function
        pstmt.setTimestamp(4, new java.sql.Timestamp(milli));
        pstmt.executeUpdate();
        pstmt.close();
    } catch (SQLException e) {
        System.out.println("데이터삽입오류 : "+e.getMessage());
    }
}
```



```
public void select() {
    String query = "SELECT * from 임시주문";
    try {
        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while( rs.next( )) {
            System.out.print("\t" + rs.getInt(1) );
            System.out.print("\t" + rs.getString(2) );
            System.out.print("\t" + rs.getString(3) );
            System.out.print("\t" + rs.getDate(4) );
            System.out.print("\t" + rs.getTime(5) );
            System.out.print("\t" + rs.getTimestamp(5) + "\n" );
            System.out.print("\t요일 : " + rs.getTimestamp(5).getDay()); //월요일이 1
            int year = rs.getDate(4).getYear() + 1900;
            System.out.print("\t시작년도 : " + year + "\n");
        }
        stmt.close();
    } catch(SQLException e) {
        System.out.println("데이터검색오류 : "+e.getMessage());
    }
}
```



예외 처리 방식

- JDBC API의 거의 모든 메서드들의 예외가 SQLException임
- 처리 방식 : 기존 자바와 동일
 - ▶ try .. catch
 - 예외가 발생하면 바로 처리
 - ▶ throws
 - 예외가 발생하면 상위 메서드로 전송
 - ▶ finally
 - 자원 관리



try... catch

```
try {  
    Class.forName("oracle.jdbc.driver.OracleDriver");  
    System.out.println("드라이버 적재 성공");  
    con = DriverManager.getConnection(url, id, password);  
    System.out.println("데이터베이스 연결 성공");  
} catch (ClassNotFoundException e) {  
    System.out.println("드라이버를 찾을 수 없습니다.");  
} catch (SQLException e) {  
    System.out.println("연결에 실패하였습니다.");  
    System.err.println("Error Message : " + e.getMessage() );  
    System.err.println("SQL State : " + e.getSQLState() );  
    System.err.println("Error Code : " + e.getErrorCode() );  
}
```

throws

```
private void sqlRun() {  
    String query = "select 고객아이디, 고객이름, 적립금 from 고객";  
    try { Statement stmt = con.createStatement();  
        ResultSet rs = stmt.executeQuery(query);  
        while (rs.next()) {  
            System.out.print("₩t" + rs.getString("고객아이디"));  
            System.out.print("₩t" + rs.getString("고객이름"));  
            System.out.print("₩t" + rs.getInt(3) + "₩n");  
        }  
    } catch (SQLException e)  
    { e.printStackTrace(); }  
}
```

// try ... catch 방식

```
private void sqlRun() throws SQLException {  
    String query = "select 고객아이디, 고객이름, 적립금 from 고객";  
    Statement stmt = con.createStatement();  
    ResultSet rs = stmt.executeQuery(query);  
    while (rs.next()) {  
        System.out.print("₩t" + rs.getString("고객아이디"));  
        System.out.print("₩t" + rs.getString("고객이름"));  
        System.out.print("₩t" + rs.getInt(3) + "₩n");  
    }  
}
```

// throws 방식

finally

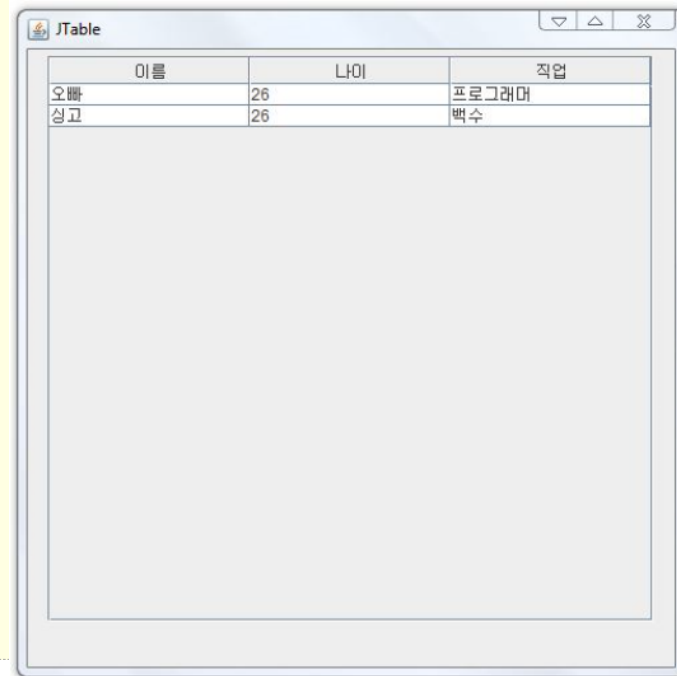
□ 예외 처리의 중요한 역할 : Resource 관리

- ▶ 프로그램이 비정상적으로 종료된 경우 resource를 안전하게 반환해야함
 - finally를 이용

```
private void sqlRun() {  
    String query = "select 고객아이디, 고객이름, 적립금 from 고객";  
    try { Statement stmt = con.createStatement();  
        ResultSet rs = stmt.executeQuery(query);  
        while (rs.next()) {  
            System.out.print("\t" + rs.getString("고객아이디"));  
            System.out.print("\t" + rs.getString("고객이름"));  
            System.out.print("\t" + rs.getInt(3) + "\n");  
        }  
    } catch (SQLException e)  
    { e.printStackTrace();  
    } finally {  
        re.close( );  
        stmt.close( );  
        con.close( );  
    }  
}
```

JAVA GUI : Jtable 활용

```
public class JTableExample extends JFrame {
    // Table 생성
    private String colName[] = { "이름", "나이", "직업" };
    private DefaultTableModel model = new DefaultTableModel(colName, 0);
    // Table에 들어갈 데이터 목록들 (헤더정보, 추가 될 row 개수)
    private JTable table = new JTable(model);
    private JPanel pan = new JPanel();
    public JTableExample() {
        /*== 기본 설정(title, size...)이 추가해야 함==*/
        // JTable에 삽입될 데이터 생성. while(rs.next()) {           } 도 가능
        String row1[] = new String[3];
        row1[0] = "오빠";
        row1[1] = "26";
        row1[2] = "프로그래머";
        String row2[] = new String[3];
        row2[0] = "싱고";
        row2[1] = "26";
        row2[2] = "백수";
        model.addRow(row1);    model.addRow(row2);
        // pan에 JScrollPane으로 JTable추가
        pan.add(new JScrollPane(table));
        add(pan);
        setVisible(true);
    }
}
```



이름	나이	직업
오빠	26	프로그래머
싱고	26	백수

GUI : JComboBox

□ item을 DB로 부터 받아서 동적으로 생성할 수도 있음

```
String[] petStrings = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };
```

```
//Create the combo box, select item at index 4.  
//Indices start at 0, so 4 specifies the pig.  
JComboBox petList = new JComboBox(petStrings);  
petList.setSelectedIndex(4);  
petList.addActionListener(this);
```

```
public class ComboBoxDemo ... implements ActionListener {  
    ...  
    petList.addActionListener(this) {  
        ...  
        public void actionPerformed(ActionEvent e) {  
            JComboBox cb = (JComboBox)e.getSource();  
            String petName = (String)cb.getSelectedItem();  
            updateLabel(petName);  
        }  
        ...  
    }  
}
```

```
public class ComboBoxEx extends JFrame {
```

```
    Container c = getContentPane();  
    String[] fruits = {"사과", "딸기", "귤", "바나나"};  
    JComboBox<String> combo1 = new JComboBox<>(fruits);  
    JComboBox<String> combo2 = new JComboBox<>();
```

```
    public ComboBoxEx() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setTitle("콤보박스 예제");  
        c.setLayout(new FlowLayout());  
        c.add(combo1);  
  
        combo2.addItem("BMW");  
        combo2.addItem("Tico");  
        combo2.addItem("Sonata");  
        combo2.addItem("Benz");  
  
        combo2.setSelectedIndex(3); //Benz가 selected. index는 0부터  
  
        c.add(combo2);  
  
        setSize(500,500);  
        setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new ComboBoxEx();  
    }  
}
```

