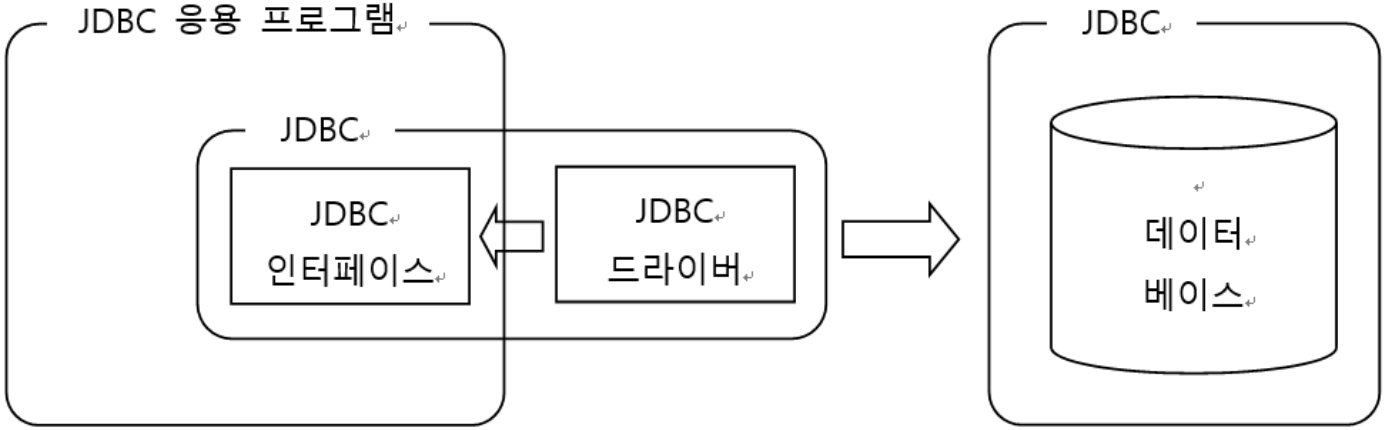

JAVA
데이터베이스 프
로그래밍

데이터베이스 프로그래밍

1. 자바와 데이터베이스
2. 데이터베이스의 기초
3. SQL
4. JDBC를 이용한 프로그래밍
5. 변경가능한 결과 집합

JDBC 이해

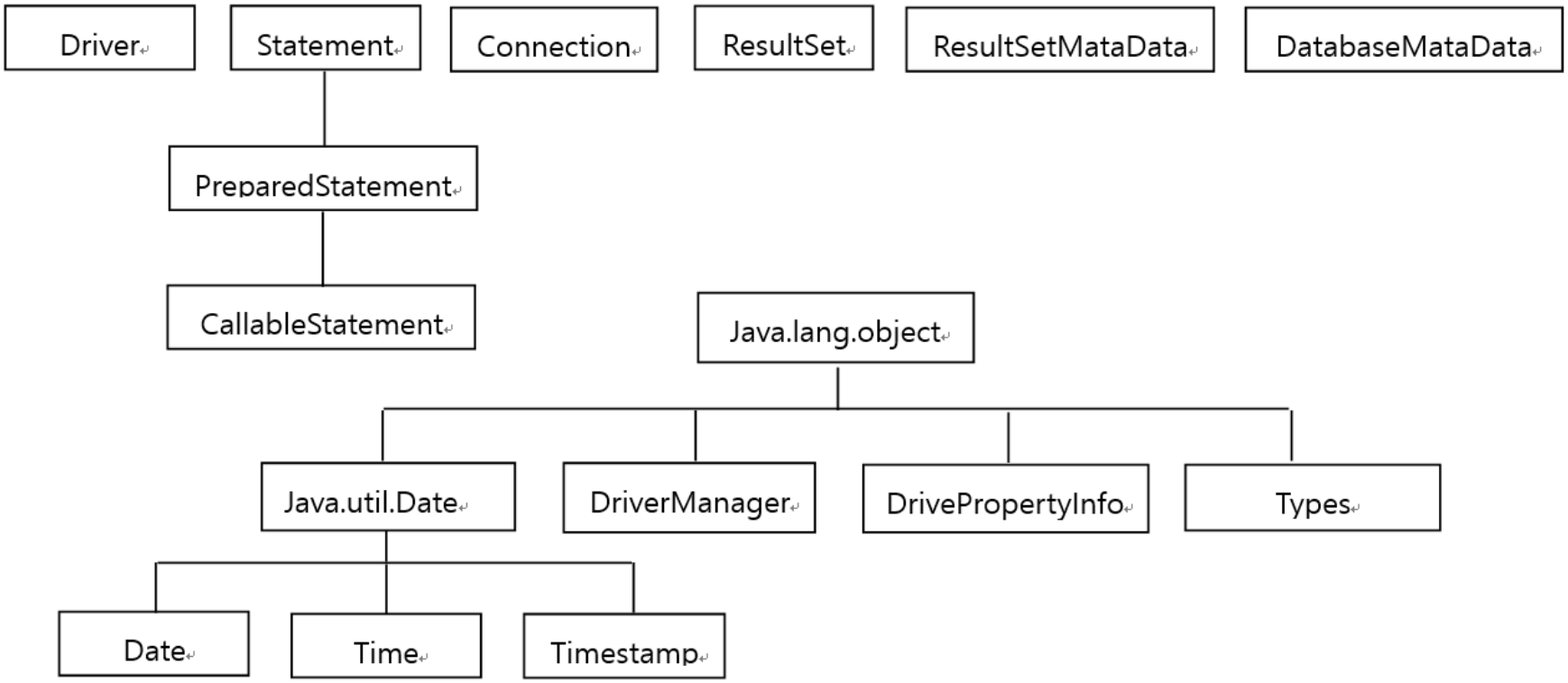
- JDBC(Java DataBase Connectivity)
- 자바 프로그램에서 데이터베이스와 연결하여 데이터베이스 관련 작업을 할 수 있도록 해주는 API, 다양한 종류의 관계형 데이터베이스에 접근할 때 사용되는 자바 표준 SQL 인터페이스이다.



- JDBC는 DataSource, DriverManager, Connection, Statement, PreparedStatement, CallableStatement, ResultSet 등 여러 개의 클래스와 인터페이스로 구성된 패키지 java.sql와 javax.sql로 구성되어 있다.

JDBC 이해

- 즉 JDBC는 다음과 같은 데이터베이스 기능을 지원하기 위한 표준 API를 제공하고 있다.
  - 데이터베이스를 연결하여 테이블 형태의 자료를 참조
  - SQL 문을 질의
  - SQL 문의 결과를 처리



자바와 데이터베이스

각 DBMS와 관련하여 회사도 많고 그로 인해 사용하는 방법이 달라진다면

이러한 문제를 해결하기 위해 SUN은 표준이 되는 인터페이스를 제공하고 있는 것이다.

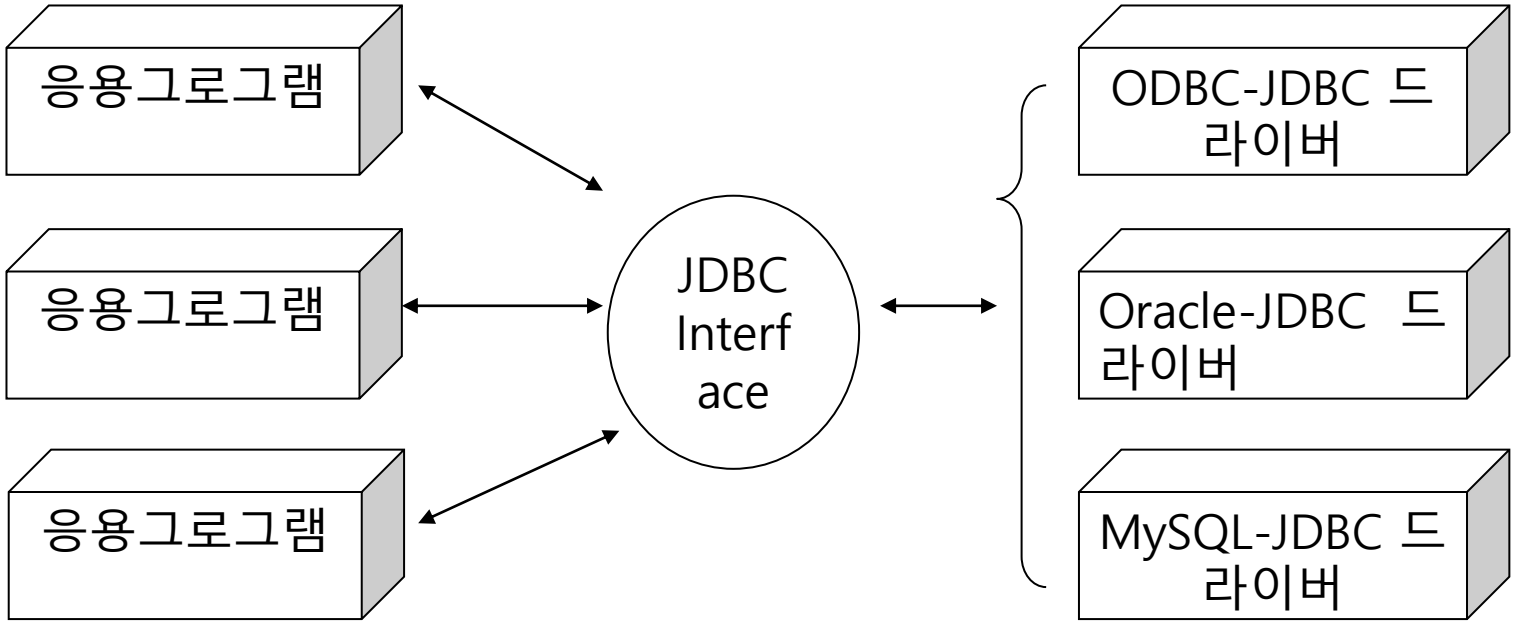
데이터베이스를 만드는 회사는 SUN사에서 JDBC 인터페이스를 통해서 자신들의 데이터베이스에 맞게 기능들을 구현하는 것이다. 그래서 인터페이스만 알면 데이터베이스를 조작할 수 있는 것이다.

이렇게 데이터베이스 회사들이 SUN사의 표준이 되는 인터페이스를 상속 받아 구현해 놓은 것이 바로 JDBC 드라이버이다.

※ JDBC 드라이버
해당 DBMS에서 JDBC 관련 API 호출이 가능하도록 관련 인터페이스와 클래스를 구현한 클래스 라이브러리

자바와 데이터베이스

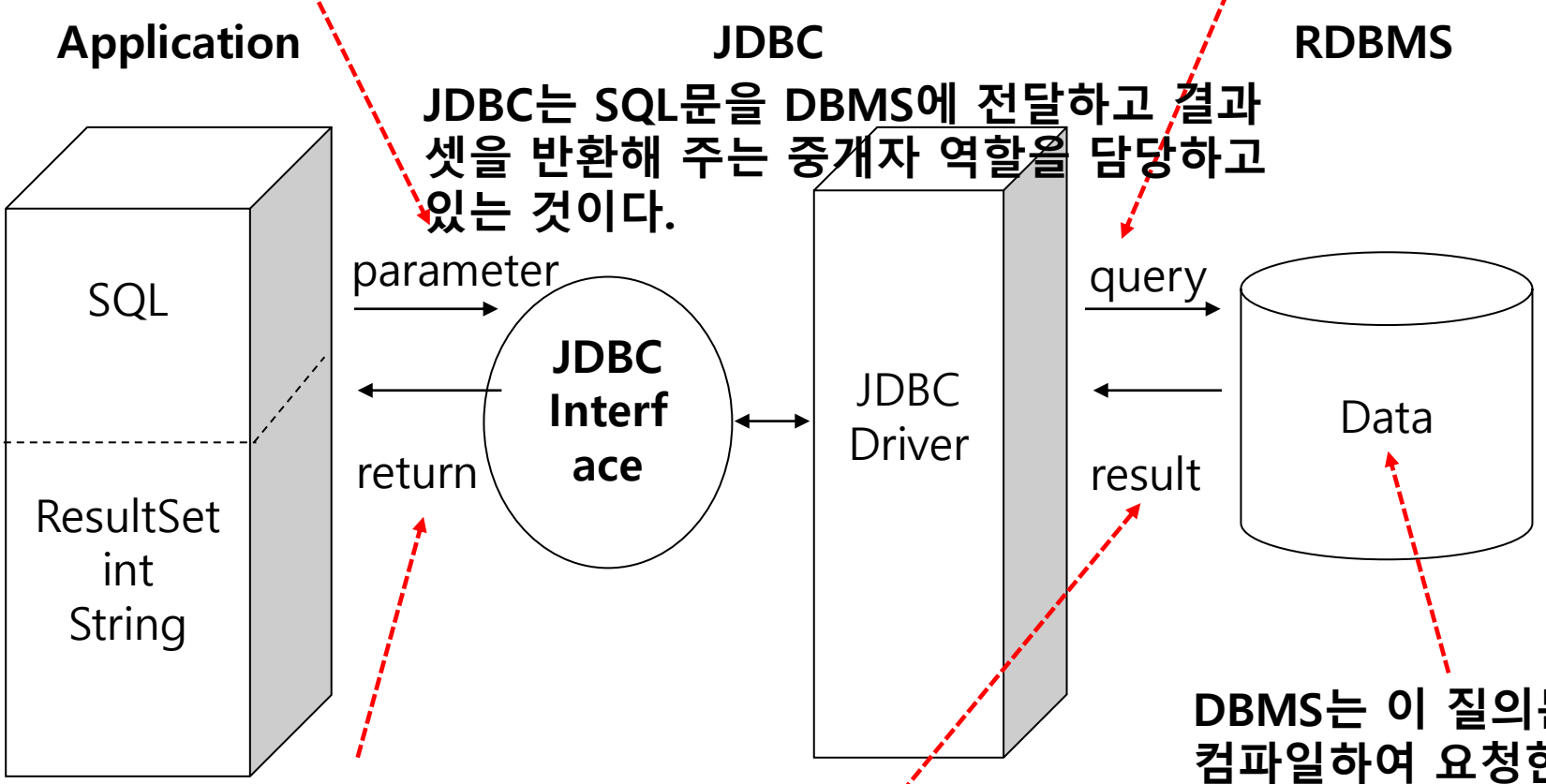
Oracle이나 MS, IBM 등의 DBMS(DataBase Management System)제품 회사들이 자신들의 DBMS에 알맞은 기능들을를 일반적으로 JDBC Driver로 구현하여 제공해 주기 때문에 사용자들은 인터페이스만 이해하면 데이터베이스를 조작할 수 있다. JDBC는 이런 드라이버들의 공통된 인터페이스인 것이다.



JDBC 역할 & Driver 정보

응용프로그램에서는 SQL문을 String으로 작성하여 이를 JDBC 클래스의 매개변수로 넘겨주면

JDBC는 이 매개변수로 넘어온 SQL문을 DBMS에 전달하고,

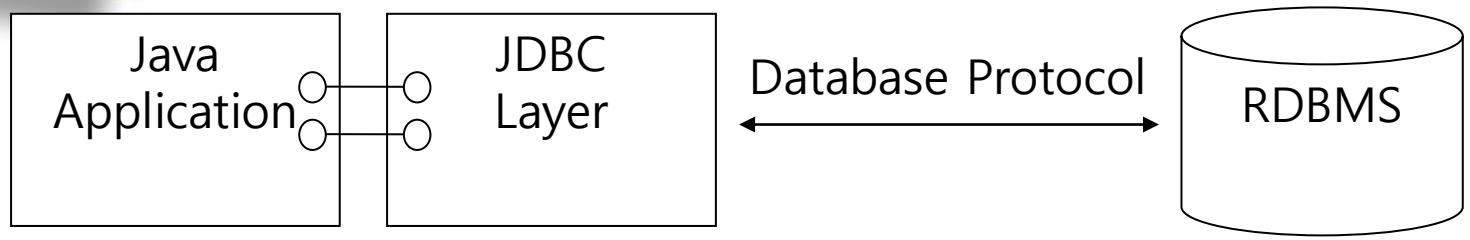


JDBC는 결과로 받은 데이터를 자바에서 사용할 수 있는 형태로 변환하여 프로그램에 반환한다.

작업 결과가 있으면

DBMS는 이 질의문을 컴파일하여 요청한 작업을 처리한다.

DBMS 프로토콜 드라이버



순수 자바로 만들어졌으며, DBMS를 직접 호출하는 드라이버로 JDBC 드라이버와 데이터베이스간의 1:1 관계를 가진다. 현재 가장 많이 쓰이는 드라이버이다

확장, 배포가 용이하고, Web 환경에 적합하다. 대표적으로 Oracle의 Thin Driver가 여기에 해당 된다.

JDBC 드라이버 & 클래스

JDBC interface를 각 회사가 특성에 맞게 구현하여 jar 또는 zip 등의 형태로 제공하고 있다.

JDBC 드라이버	드라이버 클래스
JDK 기본(JDBC-ODBC 브리지)	sun.jdbc.odbc.JdbcOdbcDriver
Oracle Thin	oracle.jdbc.driver.OracleDriver
MySQL	org.git.mm.mysql.Driver
DB2	com.ibm.db2.jdbc.app.DB2Driver
MS-SQL	com.microsoft.jdbc.sqlserver.SQLServerDriver
Sybase	com.sybase.jdbc.jdbc.SybDriver
Informix	com.informix.jdbc.IfxDriver

사용자 계정 만들기

```
create user javauser
identified by java1234
default tablespace USERS
temporary tablespace TEMP;
```
사용자가 생성되었습니다

```
grant connect, resource to javauser;
```
권한이 부여되었습니다.

데이터베이스 프로그램 개발 절차

① DBMS(DataBase Management System)를 설치

② 자신이 설치한 DBMS에 필요한 JDBC 드라이버를 설치 한다.

오라클 서버 설치 폴더로 이동하여
C:₩app₩사용자명₩product₩11.2.0₩dbhome_1₩jdbc₩lib₩ojdbc6.jar
C:₩app₩사용자명₩product₩12.1.0₩dbhome_1₩jdbc₩lib₩ojdbc7.jar

데이터베이스 프로그램 개발 절차

③ JDBC가 제공하는 기능을 이용하여 데이터베이스 응용 프로그램을 개발한다.

- 자바 어플리케이션에 추가하는 방법(두가지 방법)

1. 자바 프로젝트 생성시마다 추가하는 방법
   [프로젝트]-[Properties]-[Java Build Path]-[Libraries] 탭에서
   Add External JARs... 선택하여 추가한다

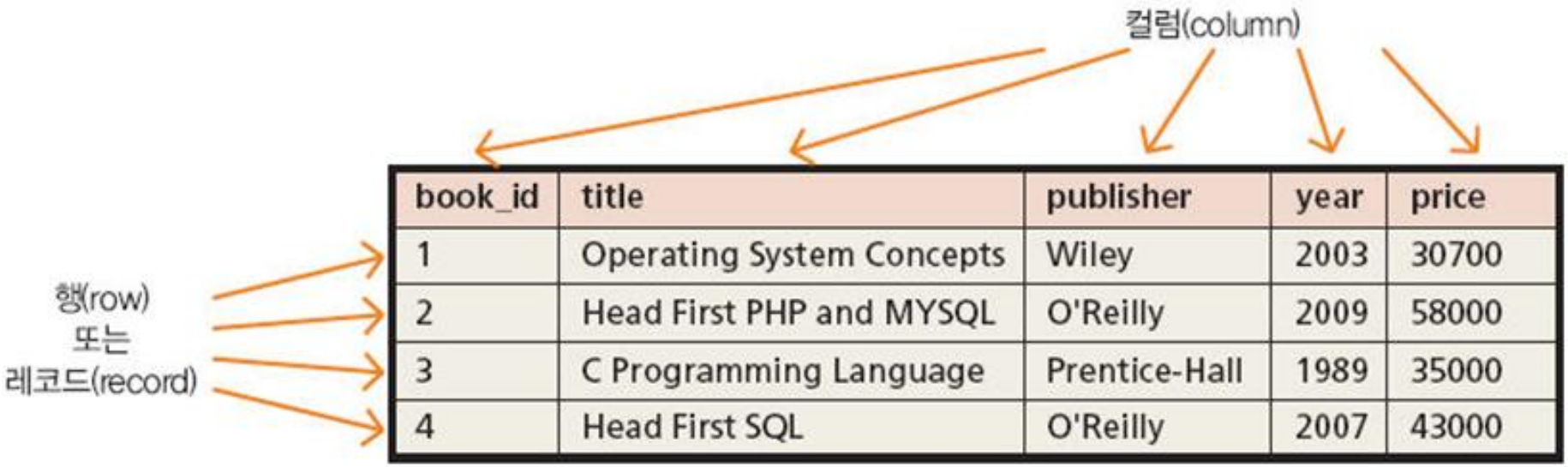
2. 자바 설치 파일에 추가하는 방법
   C:₩Java₩jdk1.8₩jre₩lib₩ext₩ojdbc6.jar // 오라클 11g
   C:₩Java₩jdk1.8₩jre₩lib₩ext₩ojdbc7.jar // 오라클 12c

데이터베이스란?

관계형 데이터베이스(database)는 데이터를 여러 개의 테이블에 나누어서 저장한다.

- 가장 많이 사용되는 DBMS
  - 오라클, 마이크로소프트의 SQL Server, 사이베이스, MySQL

테이블



book_id	title	publisher	year	price
1	Operating System Concepts	Wiley	2003	30700
2	Head First PHP and MYSQL	O'Reilly	2009	58000
3	C Programming Language	Prentice-Hall	1989	35000
4	Head First SQL	O'Reilly	2007	43000

테이블 :

행(레코드):

열(컬럼):

SQL이란?

관계형 데이터베이스에서 사용하기 위하여 설계된 언어

구분	명령어	설명
데이터 정의 명령어 (Data Definition Language)	create	사용자가 지정한 컬럼명을 가지고 테이블을 생성한다.
	alter	테이블에서 컬럼을 추가하거나 삭제한다
	drop	테이블의 모든 레코드를 제거하고 테이블의 정의 자체를 데이터베이스로부터 삭제하는 명령어이다.
	use	어떤 데이터베이스를 사용하는지를 지정한다

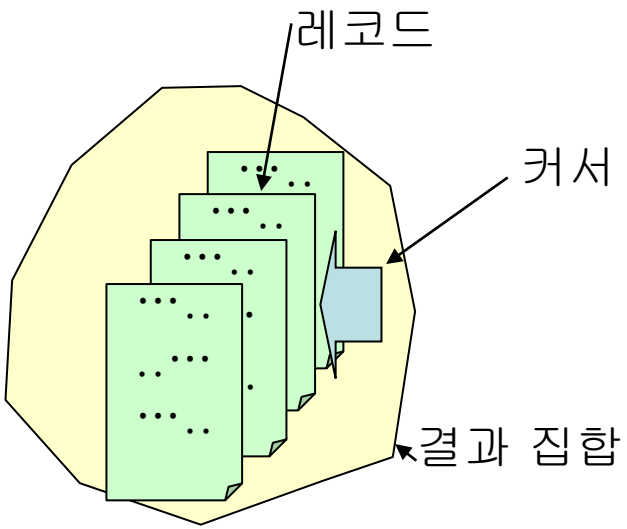
SQL이란?

관계형 데이터베이스에서 사용하기 위하여 설계된 언어

구분	명령어	설명
데이터 조작 명령어 (Data Manipulation Language)	select	테이블에 저장된 데이터를 검색한다. select 명령어들은 결과 집합에 포함시킬 컬럼을 지정한다.
	insert	새로운 레코드를 테이블에 추가한다.
	update	테이블에서 레코드에 존재하는 값을 변경한다.
	delete	지정된 레코드를 테이블로부터 삭제한다.

결과 집합과 커서

- 쿼리의 조건을 만족하는 레코드들의 집합이 결과 집합(ResultSet)이다.
- 커서(cursor)는 커서란 특정 SQL 문장을 처리한 결과를 담고 있는 영역을 가리키는 일종의 포인터이다.



기본 구현 순서

JDBC를 이용하여 데이터베이스에 데이터를 액세스하기 위해서는 기본적인 순서가 있다.

▣ java.sql.* ;

JDBC API를 사용하기 위해서는 반드시 선언해야 한다.

▣ JDBC 드라이버 로딩 및 레지스터 등록

Class.forName() 사용

▣ 데이터 베이스 연결(Connection 객체 얻는다.)

DriverManager.getConnection() 사용

기본 구현 순서

- SQL을 위한 Statement 객체 생성
  Connection.createStatement() 사용
- SQL 실행
  executeQuery()/ executeUpdate()/ execute() 사용
- ResultSet 결과셋 얻기
  SQL문의 실행 결과셋 얻기
  ResultSet에 저장된 데이터 값 출력
- 객체 자원 반납(연결 해제)
  close() 사용

JDBC 프로그래밍 절차

JDBC 프로그래밍 절차 6단계

1. JDBC 드라이버 로드

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. 데이터베이스 연결

```
String URL = "jdbc:oracle:thin:@localhost:1521:orcl";
Connection con = DriverManager.getConnection(URL,
                                "scott", "tiger");
```

3. SQL을 위한 Statement 객체 생성

```
Statement stmt = con.createStatement();
```

4. SQL 문장 실행

```
StringBuffer sb = new StringBuffer();
sb.append("select sd_num, sd_name from test ");
ResultSet rs = stmt.executeQuery(sb.toString());
```

JDBC 프로그래밍 절차

JDBC 프로그래밍 절차 6단계

5. 질의 결과 ResultSet 처리

```
while(rs.next()){
        String sd_num = rs.getString(1);
        String sd_name = rs.getString(2);
}
```

6. JDBC 객체 연결 해제

```
rs.close();
stmt.close();
con.close();
```

JDBC 관련 기본 클래스

- JDBC 프로그래밍에 이용되는 클래스로는 가장 먼저 JDBC 드라이버를 로드하는 클래스 java.lang.Class가 있으며, 패키지 java.sql에 소속된 클래스 DriverManager, 인터페이스 Connection, Statement, ResultSet 등이 있다. 이러한 클래스의 용도와 이용 메소드를 정리하면 다음과 같다.

패키지	인터페이스(클래스)	클래스 용도	이용 메소드
java.lang	클래스 Class	지정된 JDBC 드라이버를 실행시간동안 메모리에 로드	forName();
java.sql	클래스 DriverManager	여러 JDBC 드라이버를 관리하는 클래스로 데이터베이스를 접속하여 연결 객체 반환	getConnection();
	인터페이스 Connection	특정한 데이터베이스 연결 상태를 표현하는 클래스로 질의할 문장 객체 반환	createStatement(); close();
	인터페이스 Statement	데이터베이스에 SQL 질의 문장을 질의하여 그 결과인 결과집합 객체를 반환	executeQuery(); close();
	인터페이스 ResultSet	질의 결과의 자료를 저장하며 테이블 구조	next(); getString(); getInt(); close();

1단계 : JDBC 드라이버 로드

데이터베이스에 연결하기 위해서는 2가지 과정이 필요하다. 가장 먼저 드라이버 클래스를 로딩을 해야 하고 그 다음은 DB와 접속을 시도하는 것이다.

DriverManager 클래스는 Class.forName() 메소드를 통해서 생성된다. Class.forName()는 드라이버 로딩시 DriverManager.registerDriver() 메소드를 자동으로 호출하여 DriverManager에 등록된다. 드라이버 클래스들은 로드될 때 자신의 인스턴스(객체)를 생성하고, 자동적으로 DriverManager 클래스는 메소드를 호출하여 그 인스턴스를 등록한다.

드라이버 클래스를 찾지 못할 경우 forName() 메소드는 ClassNotFoundException 예외를 발생시키므로 반드시 예외를 처리를 해야 한다.

2단계 : 데이터베이스 연결

DriverManager 클래스의 Connection 인터페이스의 구현 객체를 생성하는데 getConnection() 메소드를 사용한다.

Connection객체는 특정 데이터원본과 연결된 커넥션을 나타내고 특정한 SQL문을 정의하고 실행시킬 수 있는 Statement 객체를 생성할 때도 Connection 객체를 사용한다.

JDBC 드라이버 로딩&Connection 객체 생성

[형식 1]

1-1. Class.forName("드라이버 클래스명")

Class.forName("oracle.jdbc.driver.OracleDriver")

1-2. DriverManager.registerDriver(

new oracle.jdbc.driver.OracleDriver());

1.3 Connection conn = DriverManager.getConnection("연결 url", "id", "password");

드라이브별 데이터베이스 연결 정보

드라이버	드라이버 클래스	연결 URL
JDBC-ODBC	sun.jdbc.odbc.JdbcOdbcDriver	"jdbc:odbc:<dbname>"
Oracle Thin	**oracle.jdbc.driver.OracleDriver**	**"jdbc:oracle:thin:@<serverip>:<port(1521)>:SID"**
MS-SQL	com.microsoft.jdbc.sqlserver.SQLServerDriver	"jdbc:microsoft:sqlserver://<serverip>:<port(1433)>;Database=<db_name>","id","pwd"

드라이브별 데이터베이스 연결 정보

드라이버	드라이버 클래스	연결 URL
MYSQL	org.gjt.mm.mysql.Driver	"jdbc.mysql://<serverip>:<port>/<dbname>"
DB2	com.ibm.db2.jdbc.app.DB2Driver	"jdbc:db2://<host 주소>:<port>/<db_name>"
Sybase	com.sybase.jdbc.jdbc.SybDriver	"jdbc:sybase.Tds//<serverip>:<port>"
Informix	com.informix.jdbc.IfxDriver	"jdbc:Informix-sqli://<serverip>:<port>/<db_name>:INFORMIXSERVER"

JDBC 드라이버 로딩&Connection 객체 생성

[형식 1]

```
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@127.0.
0.1:1521:orcl11“, "javauser", "java1234");
```

Connection 객체 생성

- Properties 특징
  - 키와 값을 String 타입으로 제한한 Map 컬렉션
  - Properties는 프로퍼티(~.properties) 파일을 읽어 들일 때 주로 사용

- 프로퍼티(~.properties) 파일
  - 옵션 정보, 데이터베이스 연결 정보, 애플리케이션에서 주로 변경이 잦은 문자열을 저장
    → 유지 보수를 편리하게 만들어 줌
  - 키와 값이 = 기호로 연결되어 있는 텍스트 파일

JDBC 드라이버 로딩&Connection 객체 생성

[형식 2]

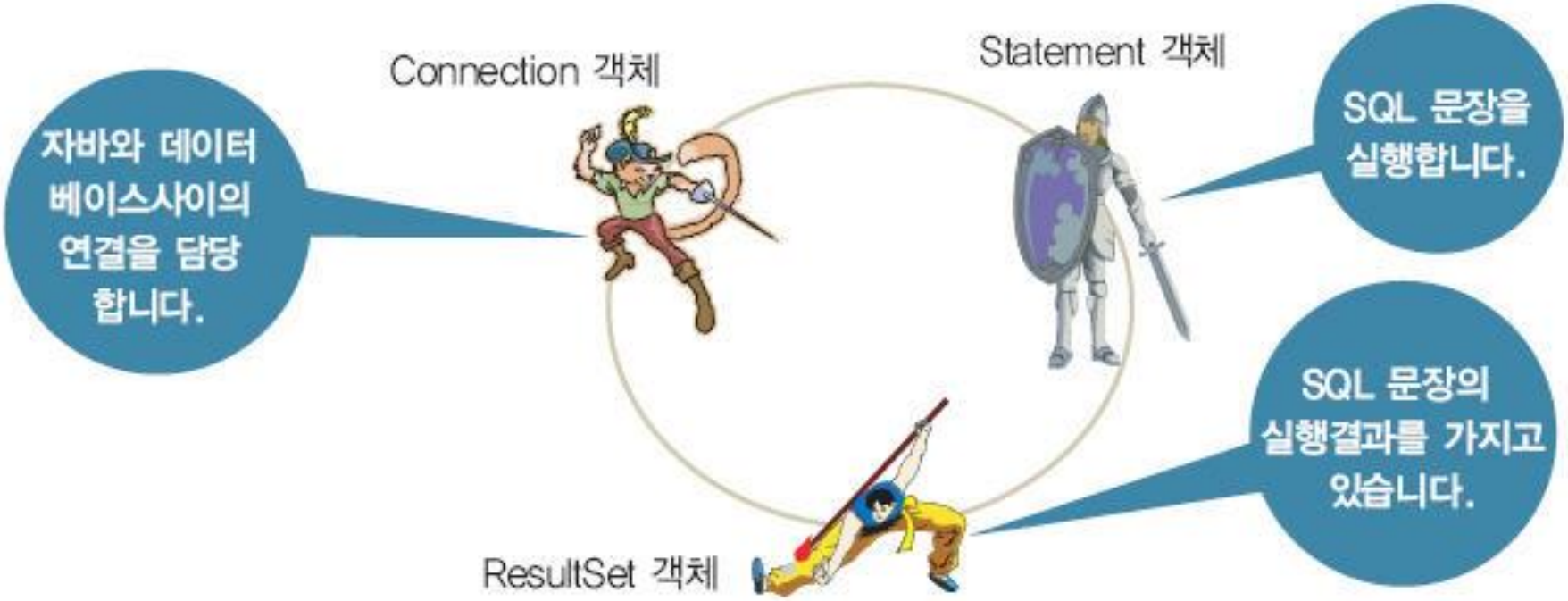
1.1 String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";

```
Properties type= new Properties();
type.put("user", "javauser");
type.put("password","java1234");
type.put("charSet", "utf-8");
```

1.2 Class.forName("드라이버 클래스명");

1.3 Connection conn = DriverManager.getConnection(url, type);

SQL 문장 수행


Connection 객체
Statement 객체
자바와 데이터 베이스사이의 연결을 담당 합니다.
SQL 문장을 실행합니다.
SQL 문장의 실행결과를 가지고 있습니다.
ResultSet 객체

Statement 객체 생성

```
String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn = DriverManager.getConnection(url,
    "javauser", "java1234");

Statement stmt = conn.createStatement();
```

SQL 문장 수행

- executeUpdate(String sql) – SQL문이 insert, update, delete문 등일 경우

```
Statement stmt = conn.createStatement();
StringBuffer sb = new StringBuffer();
sb.append("update emp set hiredate=sysdate where
empno=7788");
int updateCount = stmt.executeUpdate(sb.toString());
```

SQL 문장 수행

- executeUpdate(String sql) 사용시

- executeUpdate()메소드의 매개변수의 자료형은 String이다.
- 문자열, 날짜형 데이터 싱클쿼테이션(')으로 표현해야 한다.
- 데이터는 대소문자를 구분한다.

SQL 문장 수행

- executeQuery(String sql) – SQL문이 select 일 경우

```
Statement s = conn.createStatement();
String select = "select empno, ename from emp order
by empno";
ResultSet rs= s.executeQuery(select);

Statement stmt = conn.createStatement();
StringBuffer sb = new StringBuffer();
sb.append("select empno, ename from emp");
ResultSet rs = stmt.executeQuery(sb.toString());
```

결과 집합에서 이동

결과 집합(ResultSet)에서 레코드에 하나씩 접근하여서 작업을 하여야 한다.

executeQuery()메소드에 의하여 반환된 ResultSet 객체에는 SELECT문장에 의하여 추출된 모든 레코드가 들어 있다. 하지만 한 번에 하나의 레코드만 접근할 수 있다.

이것을 위하여 커서라는 포인터가 제공되는데 이 커서를 움직이는 다양한 메소드가 제공된다.

결과 집합에서 이동

메소드	설명
close()	결과 집합을 닫는다
first()	커서를 처음 레코드로 옮긴다.
last()	커서를 마지막 레코드로 옮긴다.
getRow()	현재 레코드 번호를 얻는다.
next()	다음 레코드로 이동한다.
previous()	이전 레코드로 이동한다.
absolute(int row)	지정된 row로 커서를 이동한다.
isFirst	첫 레코드이면 true 반환한다.
isLast()	마지막 레코드이면 true 반환한다.

결과 집합 처리

메소드의 매개변수로 컬럼명으로 접근할 수 있고, 쿼리문 작성시 컬럼 순번으로 접근할 수 있다.

```
int empno = rs.getInt("empno");
String ename = rs.getString("ename");

int empno = rs.getInt(1);
String ename = rs.getString(2);
```

결과 집합 처리

자바에서는 인덱스 번호가 0부터 시작하지만 SQL에서는 1부터 시작한다. 레코드에서 컬럼값을 추출하는 메소드는 getXXX()와 같은 형태를 가진다.

메소드	설명
Date getDate(String columnName)	지정된 컬럼의 값을 Date 타입으로 반환한다.
Date getDate(int columnIndex)	지정된 컬럼의 값을 Date 타입으로 반환한다.
int getInt(String columnName)	지정된 컬럼의 값을 int 타입으로 반환한다.
int getInt(int columnIndex)	지정된 컬럼의 값을 int 타입으로 반환한다.
String getString(String columnName)	지정된 컬럼의 값을 String 타입으로 반환한다.
String getString(int columnIndex)	지정된 컬럼의 값을 String 타입으로 반환한다.

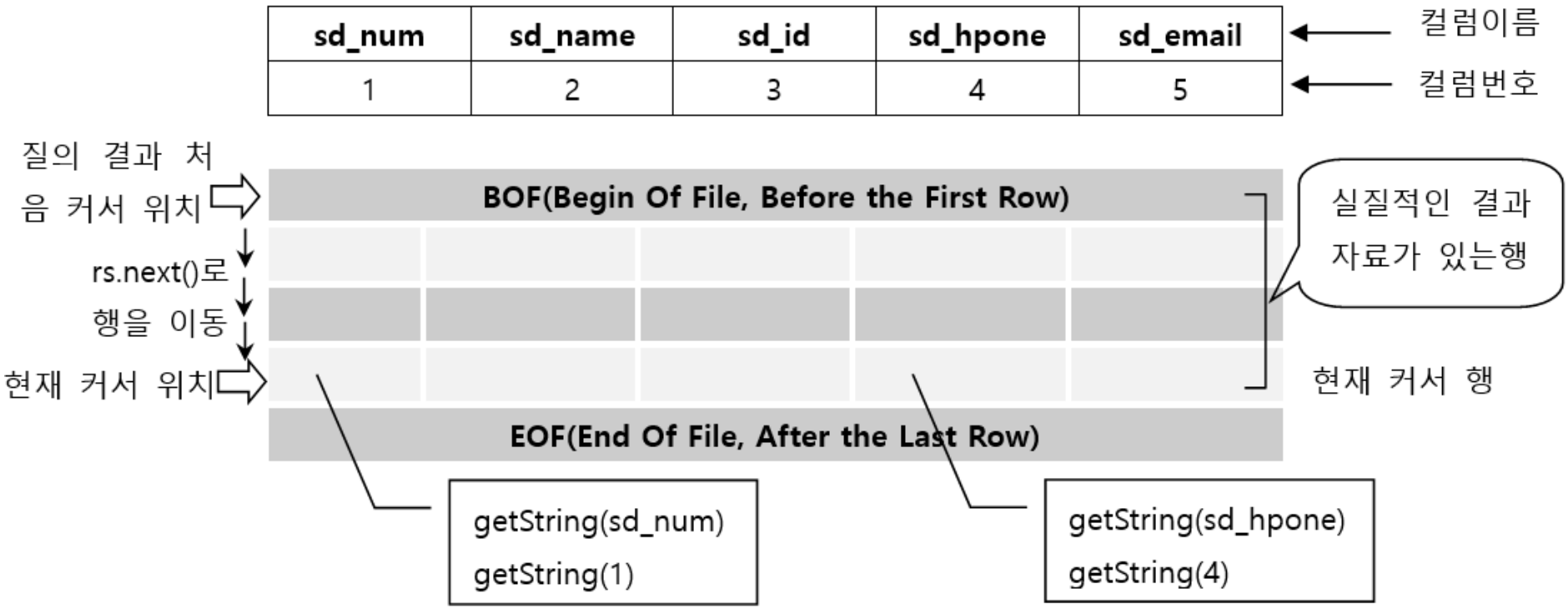
결과 집합 처리 시 체크 사항

- rs.next()를 맨 처음 한번 반드시 실행하여야 한다. 결과를 얻은 객체의 가장 처음 로우에 커서를 옮기는 작업으로 생각하면 된다. 만약 결과가 없다면 false를 리턴하게 된다.
- ResultSet 객체에 next()를 호출함으로써 해당 객체는 getXXX() 메소드를 호출할 준비를 하게 된다.
- getInt(), getString(), getDate() 등 여러 가지 데이터 형에 대한 메소드가 존재한다.
- 자바의 인덱스의 시작은 0부터 이지만, 데이터베이스는 1부터 시작한다.

결과 집합에서 이동

한 레코드씩 처리되고,다음 행으로 이동시 next()메소드를 사용

ResultSet rs = stmt.executeQuery("select * from student");



PreparedStatement

PreparedStement는 SQL문의 구조는 동일하나 조건이 다른 문장을 변수 처리함으로써 항상 SQL문을 동일하게 처리할 수 있는 인터페이스이다.

SQL문 전송하게 되면 오라클은 내부적으로 PARSING → EXECUTE PLAN → FETCH 작업을 한다. 똑같은 SQL문을 전송하면 LIBRARY CACHE에 저장된 SQL문과 비교하여 SQL문이 동일하다면 파싱한 결과와 EXECUTE PLAN을 그대로 사용하게 되며, 그 결과 또한 디스크에서 I/O로 읽어오지 않지 않고 데이터 버퍼 캐시에 저장된 메모리에서 읽어 오기 때문에 결과를 가져오는 작업 또한 빠르다고 할 수 있다.

PreparedStement로 SQL문을 처리하게 되면 LIBRARY CACHE에 저장된 작업을 재사용 함으로써 수행 속도를 좀 더 향상시킬 수 있다.

PreparedStatement 사용방법

```
String sql = "select empno, ename from emp where
empno = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, 7369);
ResultSet rs = pstmt.executeQuery();
```

PreparedStatement는 SQL문을 작성할 때 컬럼값을 실제로 지정하지 않고, 변수 처리 함으로서 DBMS을 효율적으로 사용한다.

PreparedStatement의 SQL문은 SQL문의 구조는 같은데 조건이 수시로 변할 때 조건의 변수처리를 "?" 하는데 이를 바인딩 변수라 한다.

PreparedStatement 사용방법

바인딩 변수는 반드시 컬럼명이 아닌 **컬럼값**이 와야 한다는 것이다. 바인딩 변수의 순서는 "?" 의 개수에 의해 결정이 되는데 시작 번호는 1부터 시작하게 된다.

바인딩 변수에 값을 저장하는 메서드는 오라클의 컬럼 타입에 따라 지정해 주면 된다. **PreparedStatement 인터페이스에는 바인딩 변수에 값을 저장하는 setXXX() 메서드를 제공하고 있다.**

메소드	설명
setDate(int parameterIndex, Date x)	바인딩 변수에 Date 타입이 값을 설정한다.
setInt(int parameterIndex, int x)	바인딩 변수에 int 타입이 값을 설정한다.
setString(int parameterIndex, String x)	바인딩 변수에 String 타입이 값을 설정한다.
setDouble(int parameterIndex, double x)	바인딩 변수에 double 타입이 값을 설정한다.

Statement 사용방법

```
Statement stmt = conn.createStatement();

StringBuffer sb = new StringBuffer();
sb.append("INSERT INTO books (book_id,title,publisher,");
sb.append("year, price) values(books_seq.nextval, ");
sb.append("'" + title + "','" + publisher + "','");
sb.append(year + "',"+ price + ")");

int count = stmt.executeUpdate(sb.toString());
```

PreparedStatement 사용방법

```
StringBuffer sb = new StringBuffer();
sb.append("INSERT INTO books (book_id,title,publisher,");
sb.append("year, price) values(?,?,?,?,?)");

PreparedStatement pstmt = null;
pstmt = conn.prepareStatement(sb.toString());

pstmt.setString(1, 1);
pstmt.setString(2, "어서와 자바는 처음이지! " );
pstmt.setString(3,  " 인피니티북스");
pstmt.setString(4,  " 2016");
pstmt.setInt(5, 33000);

int count = pstmt.executeUpdate();
```

SQL을 위한 객체 생성

이제 반환된 Connection 객체를 이용해서 SQL문을 실행하고 그 결과를 반환 받을 수 있는 Statement, PreparedStatement, CallableStatement 객체를 생성하는 단계이다. 인터페이스인 Statement, PreparedStatement, CallableStatement는 질의 문장인 SQL문을 추상화시킨 인터페이스이다. 이들은 Connection 객체인 메소드 createStatement()를 호출하여 Statement 객체를 얻어온다.

```
Statement stmt = con.createStatement();
PreparedStatement pstmt = con.prepareStatement(SQL);
CallableStatement cstmt = con.prepareCall(SQL);
```

CallableStatement

CallableStatement는 DBMS의 저장 프로시저(Stored Procedure)를 호출할 수 있는 인터페이스이다.

저장 프로시저란 SQL문을 프로그램화 하여 함수화시킨 스크립트 언어이다.

SQL문을 프로그램화 시켜 조건문, 반복문, 변수처리 등을 사용하기 때문에 일괄처리 및 조건에 따라 달라지는 SQL문을 작성할 때는 유리하다.

CallableStatement

```
String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn = DriverManager.getConnection(url,
"javauser", "java1234");

CallableStatement cstmt = conn.prepareCall("{call 프로
시저명(?,?)}");
```

저장 프로시저(Stored Procedure)

```
create or replace procedure book_inproc
(p_book_id in book.book_id%TYPE,
 r_title out book.title%TYPE)
is
begin
    select title into r_title from book
    where book_id = p_book_id;
end book_inproc;
/
```

CallableStatement 사용방법

```
String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn = DriverManager.getConnection(url,
"javauser", "java1234");

CallableStatement cstmt
                = con.prepareCall("{call book_inproc(?,?)}");
cstmt.setInt(1, book_id);
cstmt.registerOutParameter(2, Types.VARCHAR);
cstmt.executeQuery();

System.out.println("₩n*** BOOK 테이블 검색 데이터 ***");
System.out.printf("책제목 : %s", cstmt.getString(2));
```

JDBC 객체 연결 해제

JDBC 프로그래밍의 마지막 단계는 이미 사용한 JDBC 객체의 연결을 해제하는 일이다. 특히 데이터베이스 연결을 의미하는 Connection 객체의 연결 해제는 메모리와 서버의 부하의 관점에서 중요하다.

데이터베이스 작업이 종료되었으면 Connection 객체 변수 con에서 메소드 close()를 이용하여 연결을 해제한다.

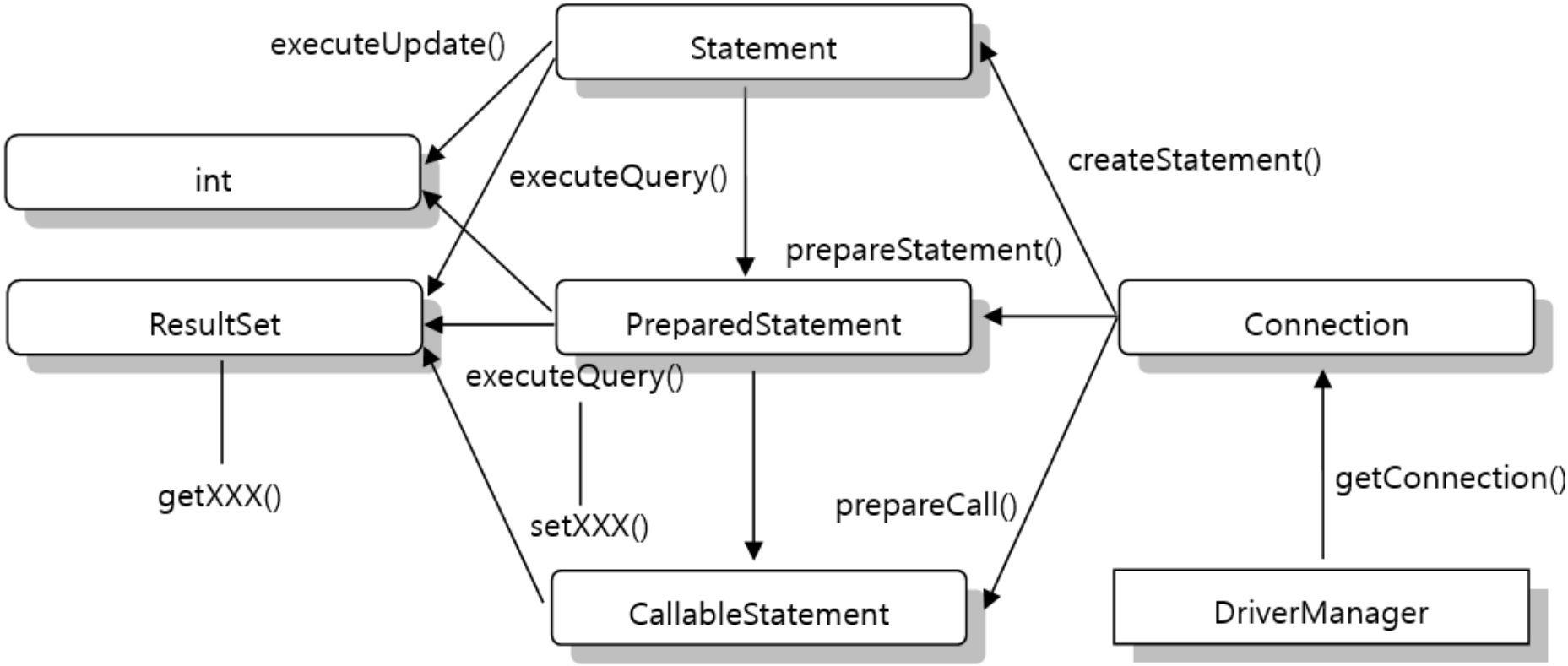
```
con.close();
```

Connection 객체의 연결을 해제하기 전에 객체 ResultSet과 Statement/PreparedStatement도 메소드 close()를 호출하여 명시적으로 연결을 해제함으로써 시스템 자원을 효율적으로 이용하도록 하는 것이 좋다.

```
rs.close();
stmt.close();
```

JDBC 관련 인터페이스와 클래스

JDBC 관련 클래스와 인터페이스 중에서 패키지 java.sql에 소속된 인터페이수와 클래스에서 메소드 호출과 상속 관계를 표현하면 다음과 같다.



Properties를 이용한 JDBC 연결

Properties클래스를 이용해서 특정 파일을 읽어온 후 드라이브 로드명, URL, USER, PASSWORD의 정보를 알아낸다. 이렇게 Properties 클래스를 이용하게 되면 DBMS가 바뀌거나 드라이브 로드명, URL, USER, PASSWORD가 변경되더라도 파일의 정보만 변경하고, 실제 프로그램을 수정하여 재컴파일을 할 필요가 없기 때문에 유지 보수에 상당히 좋다고 할 수 있다.

Properties를 이용한 JDBC 연결

C:₩jdbc.properties 파일을 아래와 같이 만들어보자.

```
driver = oracle.jdbc.driver.OracleDriver
url = jdbc:oracle:thin:@127.0.0.1:1521:orcl11
user = javauser
password = java1234
```

Properties를 이용한 JDBC 연결

jdbc.properties 파일을 읽기 위해서는 스트림을 생성해야 한다. 스트림을 Properties 클래스에 로드(load()) 시키고, Properties클래스의 getProperty(String key)메서드를 이용해서 각각의 값을 얻어 올 수 있다.

Properties를 이용한 JDBC 연결

```
try{
        FileInputStream fis
         = new FileInputStream("c:₩₩jdbc.properties");

        Properties pro = new Properties();
        pro.load(fis);

       String driver = pro.getProperty("driver");
        String url = pro.getProperty("url");
        String user = pro.getProperty("user");
        String password = pro.getProperty("password");
       …
}catch(IOException i){
        i.printStackTrace();
}
```

Scrollable

ResultSet의 커서가 양방향으로 움직(Scrollable)일 수 있다.

createStatement(int resultSetType, int resultSetConcurrency) 메서드를 제공하고 있다.

매개변수 인자로 들어가는 2가지 int 값(ResultSet의 상수 값으로 존재)에 대해 알아보자.

☞ resultSetType → TYPE_XXX 형태

TYPE_FORWARD_ONLY : 커서의 이동이 단 방향만 가능하다. 속도가 빠르다..

TYPE_SCROLL_INSENSITIVE : 커서의 이동이 양방향 가능하며, 갱신된 데이터를 반영하지 않는다.

TYPE_SCROLL_SENSITIVE : 커서의 이동이 양방향 가능하며, 갱신된 데이터를 반영한다.

Scrollable

ResultSet의 커서가 양방향으로 움직(Scrollable)일 수 있다.

createStatement(int resultSetType, int resultSetConcurrency) 메서드를 제공하고 있다.

☞ resultSetConcurrency → CONCUR_XXX 형태

CONCUR_READ_ONLY : 읽기만 된다.

CONCUR_UPDATABLE : 데이터를 동적으로 갱신할 수 있다.

Scrollable

양방향 가능한 Statement 객체를 생성하는 방법은 아래와 같다

```
Statement stmt = con.createStatement(
                    ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
```


Thank You