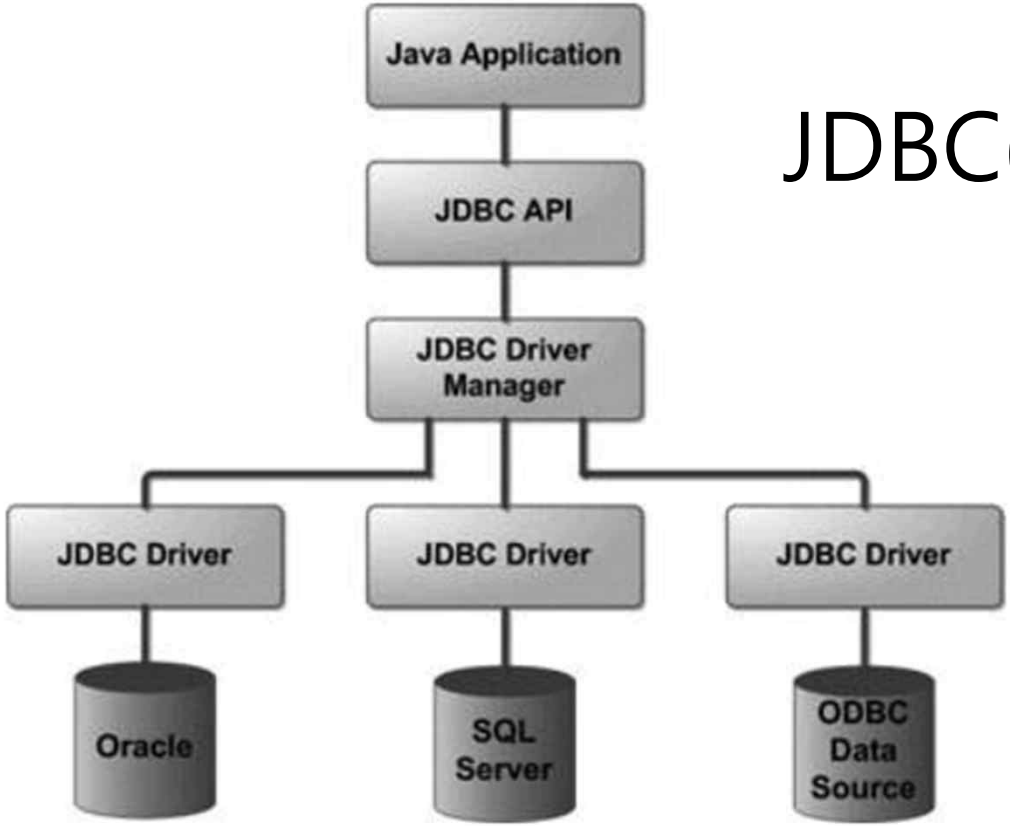


JDBC(Java Database Connectivity)

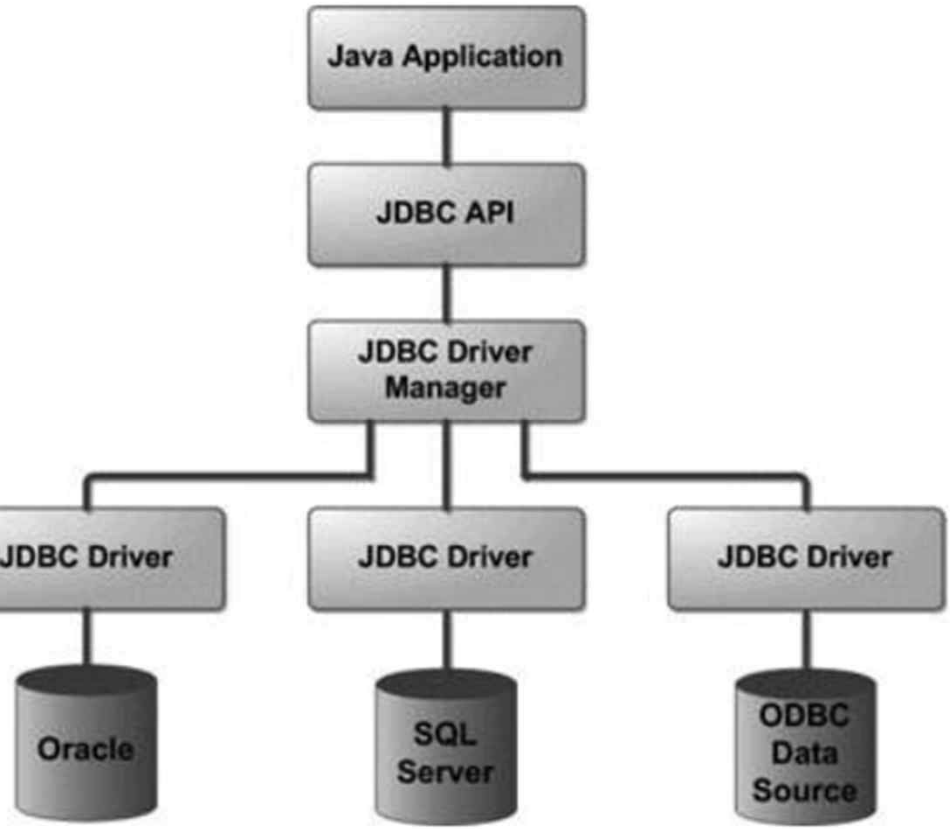
BC(Java Database Connectivity)

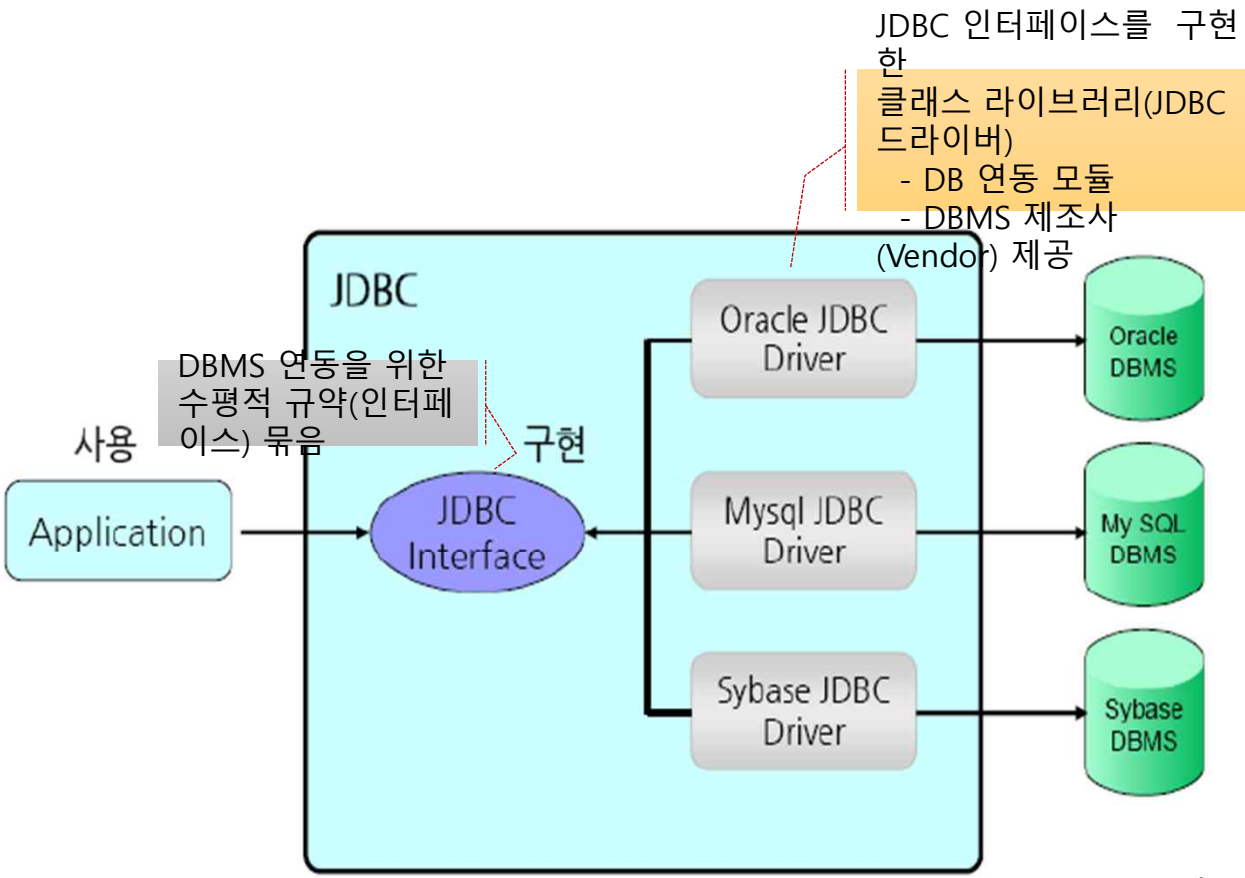
VM의 시스템과 DB 시스템을 연결하고 통신하기 위한 JAVA의 표준 스펙

atabase 연동을 위한 인터페이스와 클래스들의 집합(클래스 라이브러리)

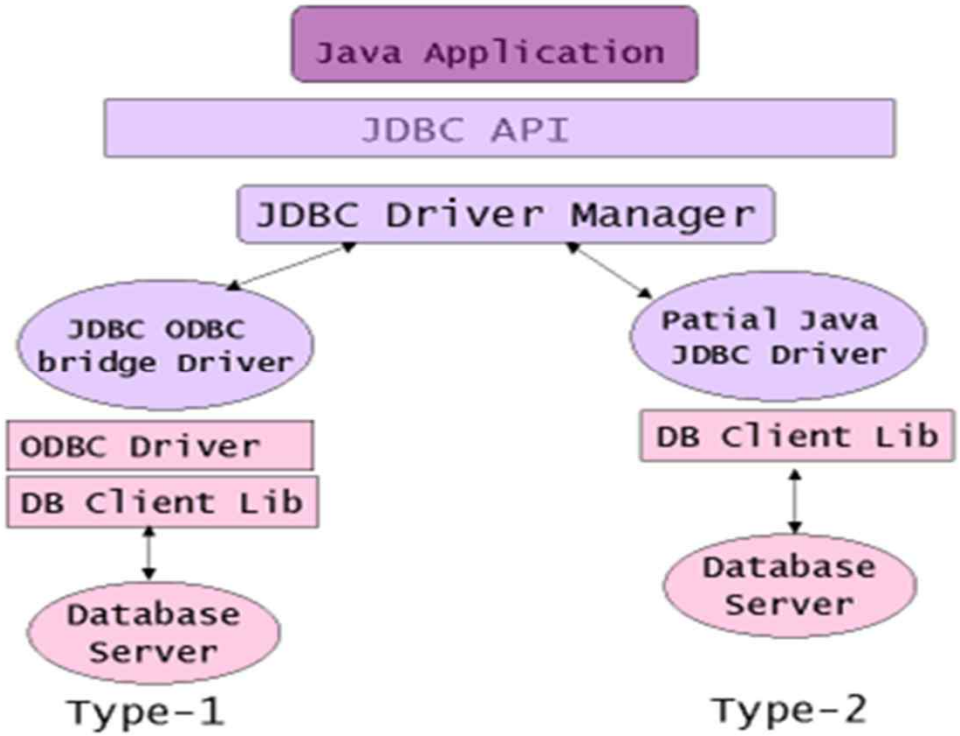
DBC를 이용하면 RDBMS에 상관없이 일관된 방법으로 데이터베이스에 연동할 수 있다.

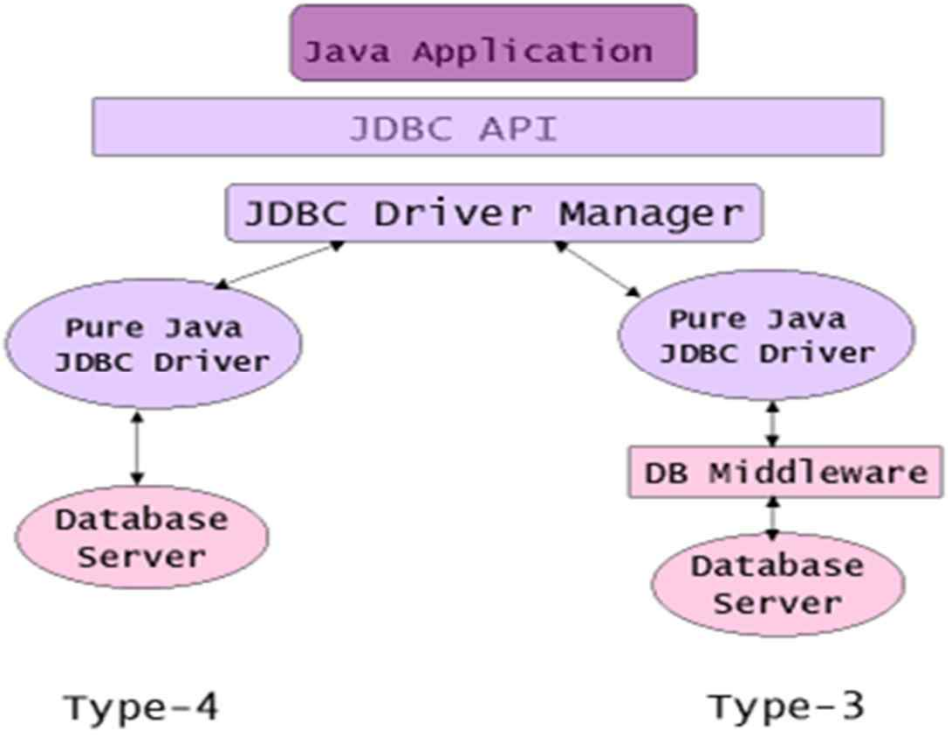
racle, mySQL, DB2, Sybase, Informix, MS-SQL 등의 DBMS와 연동하는 별도의 프로그램을 작성 할 필요가 없다.





BC Driver 유형





MicroSoft 표준인 ODBC 드라이버를 연결하고 이를 통하여 MS-SQL 서버를 연결하는 드라이버

DB 벤더들이 자체적으로 제공하는 DB클라이언트 라이브러리를 통하여 DB를 연결하는 드라이버

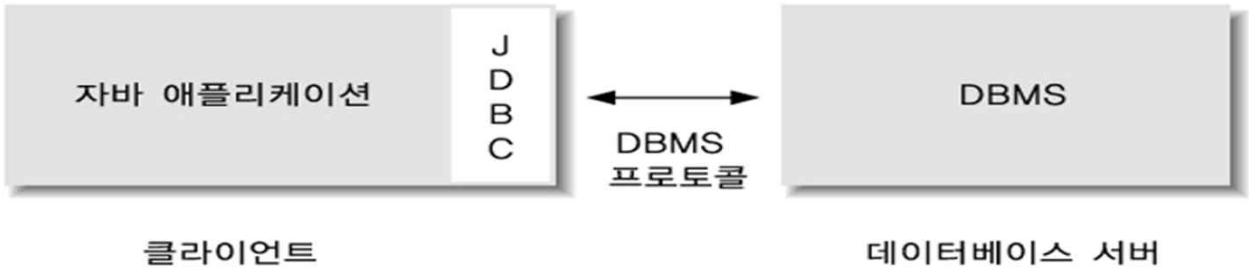
JDBC 표준에 의하여 만들어진 가장 일반적인 순수 자바드라이버

JDBC 표준을 기준으로 기능을 추가 또는 삭제하여 제공하는 드라이버

BC Programming

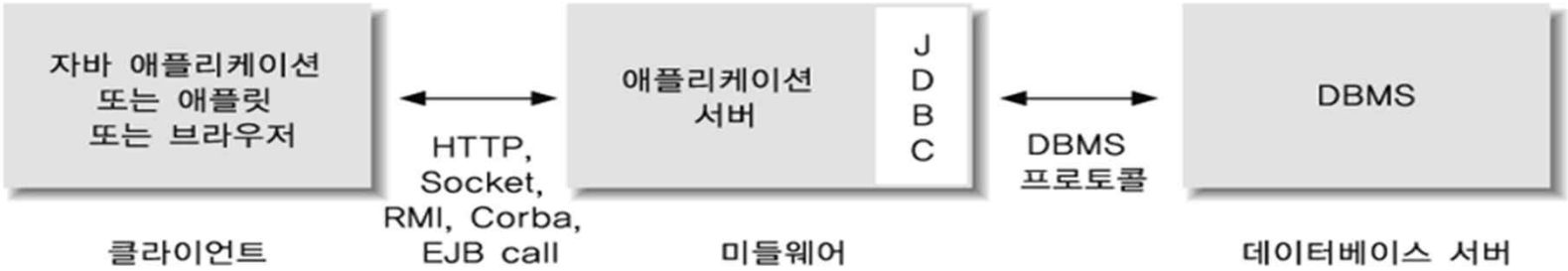
ier

- 자바 애플리케이션이 JDBC 드라이버를 통해서 직접 데이터베이스를 접근하는 형식이다.
- 이 모델에서 JDBC 드라이버는 JDBC API 호출을 통해 특정 DBMS에 직접 전달해 주는 역할을 한다.



3tier

- 3tier 모델은 2tier 모델에 미들웨어 계층이 추가된 형태이며, 미들웨어에서 DBMS에 직접 질의하게 된다.
- 3tier 모델은 2tier 모델에 보다 유지보수(Maintenance)에 비용을 절약할 수 있다.
- 단점은 2tier에 비해 속도는 다소 느리다.



BC Programming

DBC 프로그래밍 순서

1. JDBC API import
2. JDBC 드라이버 로딩
3. Connection 맺기
4. SQL 실행
5. [SQL문이 select문이었다면 ResultSet을 이용한 처리]
6. 자원 반환
7. 예외처리
8. Resource close

B Vendor가 제공하는 JDBC Driver를 다운받아 설치한다

1. JDBC Driver는 자바 클래스로 이루어져 있으며 zip 또는 jar 파일로 제공된다
2. JDBC Driver를 개발툴(이클립스) 프로젝트에서 인식하도록 클래스패스를 설정하거나 JAVA_HOME₩jre₩lib₩ext에 가한다.

DBC Programming

단계 : JDBC Driver 로드

- Class.forName(~)은 동적으로 JDBC 드라이브 클래스를 로딩하는 것이다.
- forName(~) 메서드에 매개변수로 오는 OracleDriver 클래스의 객체를 만들어 런타심 메모리에 로딩시켜 주는 메서드이다.

```
// Oracle JDBC Driver 로드(인스턴스 생성)
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
```

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

단계 : Database 접속

- Class.forName(~)을 이용하면 OracleDriver클래스가 런타임 메모리에 로딩 되고 DriverManager 클래스의 static 멤
  변수로 저장된다.

```
// Oracle 연결(JDBC URL 설정)
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:SID", "UserId", "UserPw");
```

단계 : Connection 인스턴스로부터 Statement 인스턴스 획득

```
// Statement는 SQL 문장을 실행(전송)하며, 반환된 결과를 반환
String sql = "SELEC * FROM employees";
Statement stmt = con.createStatement();
```

onnection의 주요 메소드

반환형	메서드	설명
void	close( )	Connection 객체를 해제한다.
	commit( )	트랜잭션으로 설정된 모든 자원을 커밋한다.
Statement	createStatement( )	SQL문을 전송할 수 있는 Statement 객체를 생성한다.
	createStatement (int resultSetType, int resultSetConcurrency)	매개변수로 SQL문을 전송할 수 있는 Statement 객체를 생성한다. 매개변수값을 어떻게 설정하느냐 따라 Statement 객체의 기능이 달라진다.
boolean	getAutoCommit( )	Connection 객체의 현재 auto-commit 상태를 반환한다.
CallableStatement	prepareCall(String sql)	SQL문 전송과 Store Procedure를 호출할 수 있는 CallableStatement 객체를 생성한다.
	prepareCall(String sql, int resultSetType, int resultSetConcurrency)	매개변수로 CallableStatement 객체를 생성한다. 매개변수값을 어떻게 설정하느냐 따라 CallableStatement 객체의 기능이 달라진다.
PreparedStatement	prepareStatement (String sql)	SQL문을 전송할 수 있는 PreparedStatement 객체를 생성한다.
	prepareStatement (String sql, int resultSetType, int resultSetConcurrency)	매개변수로 PreparedStatement 객체를 생성한다. 매개변수값을 어떻게 설정하느냐 따라 CallableStatement 객체의 기능이 달라진다.
void	rollback( )	현재 트랜잭션에 설정된 모든 변화를 되돌린다.
	rollback(Savepoint savepoint)	Savepoint로 설정된 이후의 모든 변화를 되돌린다.
Savepoint	setSavepoint(String name)	현재 트랜잭션에서 name으로 Savepoint를 설정한다.

단계 : Statement 메서드를 이용하여 SQL 실행(전송)

```
// SQL 실행 후 결과집합(SELECT) 또는 숫자(DML)를 받아 처리
ResultSet rs = stmt.executeQuery(sql);
//int updateCount = stmt.executeUpdate(sql);
```

단계: 결과집합(ResultSet) 처리 및 사용한 자원 반납(close())

```
// ResultSet에서 한행씩 이동하면서 getXXX()메소드를 이용해서
// 원하는 컬럼값을 패치(인출)한다
while(rs.next()){
    int employeeId = rs.getInt("employee_id"); // rs.getInt(1)
    String firstName = rs.getString("first_name");
}
```

Statement 인터페이스의 SQL를 전송 메소드

executeQuery(String sql) – SQL문이 select 일 경우

```
Statement stmt = con.createStatement();
StringBuffer sb = new StringBuffer();
sb.append("select id from test ");
ResultSet rs = stmt.executeQuery(sb.toString());
```

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

```
Statement stmt = con.createStatement();
StringBuffer sb = new StringBuffer();
sb.append("update test set id='syh1011' ");
int updateCount = stmt.executeUpdate(sb.toString());
```

Statement 인터페이스의 SQL를 전송 메소드

execute(String sql) – SQL문을 알지 못하는 경우

```
Statement stmt = con.createStatement();
StringBuffer sb = new StringBuffer();
sb.append("update test set id='syh5055'");
boolean isResult = stmt.execute(sb.toString());
if(isResult){
        ResultSet rs = stmt.getResultSet();
        while(rs.next()){
                System.out.println("id : "+rs.getString(1));
        }
}else{
        int rowCount = stmt.getUpdateCount();
        System.out.println("rowCount : "+rowCount);
}
```

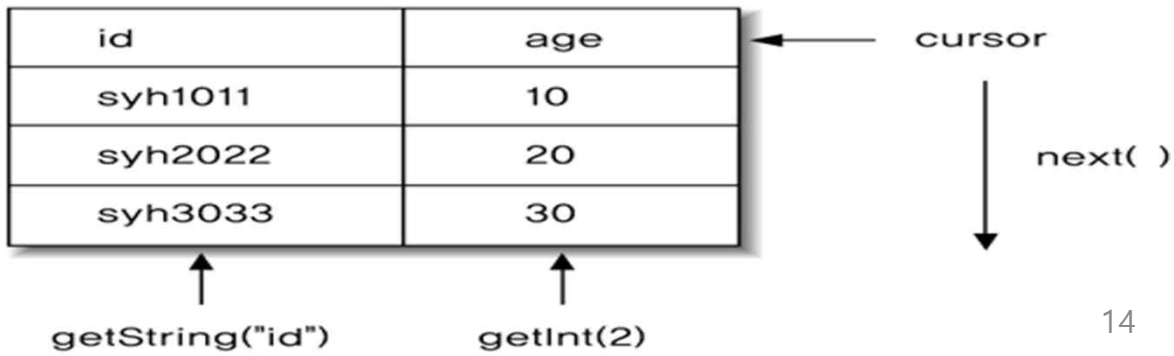
tatement 주요 메소드

반환형	메서드	설명
void	addBatch(String sql)	Statement 객체에 SQL문을 추가한다. 이 메서드를 이용해서 SQL의 일괄처리를 할 수 있다.
	clearBatch( )	Statement 객체에 모든 SQL문을 비운다.
	close( )	Statement 객체를 해제한다.
boolean	execute(String sql)	매개변수인 SQL문을 수행한다. 만약, 수행한 결과가 ResultSet 객체를 반환하면 true, 어떠한 결과도 없거나, 갱신된 숫자를 반환하면 false를 반환한다.
int[ ]	executeBatch( )	Statement 객체에 추가된 모든 SQL문을 일괄처리한다. 일괄처리된 각각의 SQL문에 대한 결과값을 int[ ]로 반환한다.
ResultSet	exectueQuery(String sql)	매개변수인 SQL문을 수행하고 ResultSet 객체를 반환한다.
int	executeUpdate(String sql)	매개변수인 SQL문을 수행한다. SQL문은 INSERT문, UPDATE문, CREATE문, DROP문 등을 사용한다.
ResultSet	getResultSet( )	ResultSet 객체를 반환한다.

esultSet 객체

- 모든 데이터를 한번에 가져올 수 없기 때문에 cursor의 개념을 가지고 있다.
- cursor란 ResultSet 객체가 가져올 수 있는 행을 지정해 준다.
- 처음 커서의 위치는 결과물(필드)에 위치하지 않기 때문에 cusror를 이동해야 한다.
- 커서를 이동하는 메서드가 ResultSet 의 next() 메서드이다.
- next() 메서드의 리턴 타입은 boolean 인데 이는 다음 행의 결과물(필드)이 있으면 true, 없으면 false를 리턴 한다.
- ResultSet 객체가 결과물(필드)을 가져올 수 있는 행으로 이동이 되었다면 이제는 실제 결과물(필드)을 가져와야 한다.
- ResultSet 인터페이스에는 결과물(필드)을 가져오는 수많은 메서드(getXXX())를 제공한다.
- getXXX() 메서드는 oracle의 자료형 타입에 따라 달라지게 된다.
- 예) id 컬럼이 varchar2 타입이라면 getString(~) 메서드를 사용해야 하고, age 컬럼이 number 타입이라면 getInt(~) 메서드를 사용해야 한다.
- getXXX() 메서드는 두개씩 오버로드 되어 정의 되어 있는데 하나는 정수를 인자로 받는 것과 String 타입으로 인자를 받는 메서드를 제공하고 있다. 커서를 이동하는 메서드가 ResultSet 의 next() 메서드이다.
- 첫번째 정수를 받는 타입은 select 문 다음에 쓰는 컬럼명의 인덱스를 지정하는데 인덱스의 처음번호는 1부터 시작하게 된다.
- 두번째 String 타입은 select 문의 다음에 오는 컬럼명으로 지정해야 한다

ResultSet 구조도



BC Programming

esultSet의 주요 메소드

반환형	메서드	설명
olean	absolute(int row)	ResultSet객체서 매개변수 row로 커서를 이동한다. 만약, 매개변수 row커서를 이동할 수 있으며 true, 그지 않으면 false를 반환한다
id	afterLast()	Resultset 객체에서 커서를 마지막 로우 다음으로 이동한다.
	beforeFirst()	Resultset 객체에서 커서를 처음 로우 이전으로 이동한다.
olean	last()	Resultset 객체에서 커서를 마지막 로우로 이동한다. 만약, ResultSet에 row가 있다면 true, 그렇지 않으 false를 반환한다
	next()	Resultset 객체에서 현재 커서에서 다음 로우로 커서를 이동한다. 만약, ResultSet에 다음 row가 있다면 true, 그렇지 않으면 false를 반환한다
	previous()	Resultset 객체에서 현재 커서에서 이전 로우로 커서를 이동한다. 만약, ResultSet에 다음 row가 있다면 true, 그렇지 않으면 false를 반환한다
id	close()	ResultSet 객체를 해제한다.
olean	first()	ResultSet 객체에서 커서를 처음 로우로 이동한다. 만약, ResultSet에 row가 있다면 true, 그렇지 않으면 false를 반환한다
utStream	getBinaryStream(int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 InputStream으로 반환한다.
	getBinaryStream(String columnName)	ResultSet 객체의 현재 로우에 있는 columnname의 값을 InputStream으로 반환한다.
ob	getBlob(int columnIndex)	ResultSet객체의 현재 로우에 있는 columnIndex값의 값을 Blob로 반환한다
	getBlob(string columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 Blob으로 반환한다.
te	getByte(int columnIndex)	ResultSet객체의 현재 로우에 있는 columnIndex값의 값을 byte로 반환한다
	getByte(int columnName)	ResultSet객체의 현재 로우에 있는 olumnName값의 값을 byte로 반환한다

BC Programming

esultSet의 주요 메소드

반환형	메서드	설명
b	getClob(int columnIndex)	ResultSet객체의 현재 로우에 있는 columnIndex값의 값을 Clob로 반환한다
	getClob(string columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 Clob으로 반환한다.
uble	getDouble(int columnIndex)	ResultSet객체의 현재 로우에 있는 columnIndex값의 값을 double로 반환한다
	getDouble (string columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 double으로 반환한다.
	getInt(int columnIndex)	ResultSet객체의 현재 로우에 있는 columnIndex값의 값을 int로 반환한다
	getInt(int columnName)	ResultSet객체의 현재 로우에 있는 olumnName값의 값을 int로 반환한다
ng	getString(int columnIndex)	ResultSet객체의 현재 로우에 있는 columnIndex값의 값을 String로 반환한다
	getString(string columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 String으로 반환한다.

ResultSet의 주요 메소드

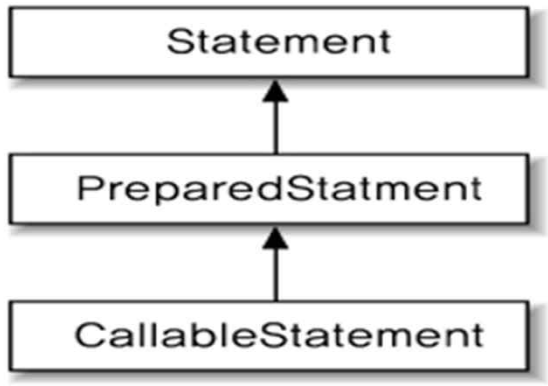
- TYPE_FORWARD_ONLY : scroll이 불가능한 forwad only 형
- TYPE_SCROLL_INSENSITIVE : scroll은 가능하나, 변경된 사항은 적용되지 않음
- TYPE_SCROLL_SENSITIVE : scroll은 가능하며, 변경된 사항이 적용됨
- CONCUR_READ_ONLY : resultset object의 변경이 불가능
- CONCUR_UPDATABLE : resultset object의 변경이 가능

```
Statement stmt = conn.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
```

BC Programming

atement 상속관계

- Statement 인터페이스는 SQL문을 전송할 수 있는 객체이다.
- 서브 인터페이스로 갈 수록 향상된 기능을 제공한다.



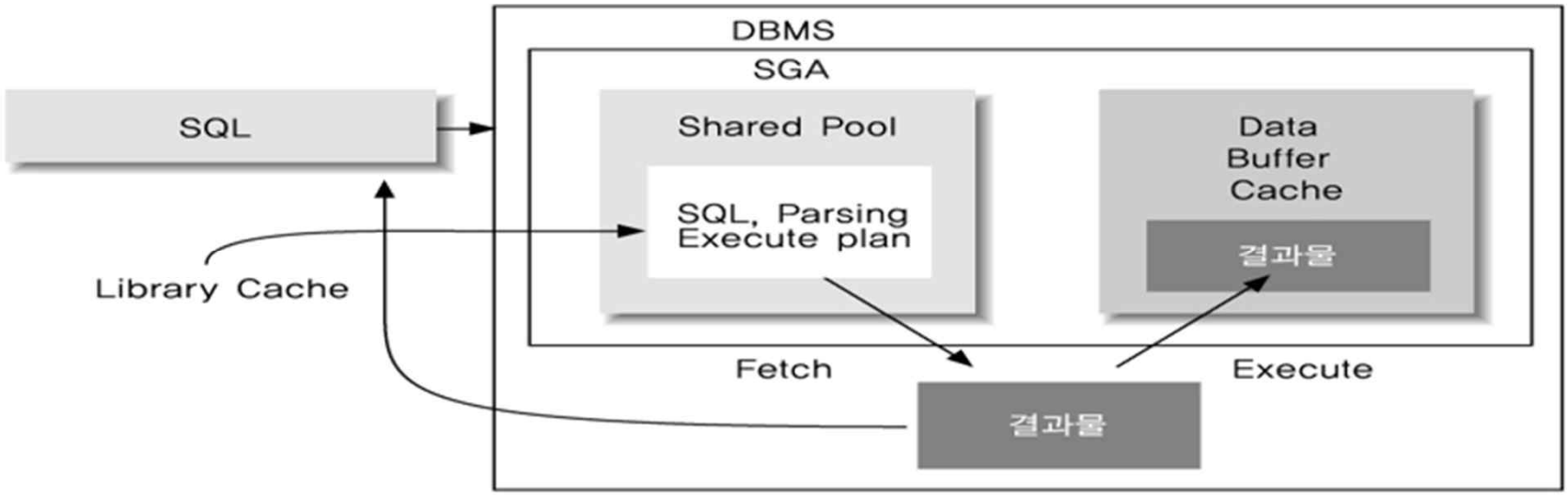
reparedStatement

- PreparedStement는 SQL문의 구조는 동일하나 조건이 다른 문장을 변수 처리함으로써 항상 SQL문을 동일하게 처리
  수 있는 인터페이스 이다.
- PreparedStement로 SQL문을 처리하게 되면 LIBRARY CACHE에 저장된 세 가지 작업을 재사용 함으로써 수행 속도를
  좀 더 향상시킬 수 있다.
- 이해를 돕기 위해 SQL문을 전송했을 경우 오라클은 내부적으로 어떻게 작동하는 보도록 하자.
- SQL문 전송하게 되면 오라클은 내부적으로 PARSING => EXECUTE PLAN => FETCH 작업을 한다.
- 이런 3가지 작업을 한 후에 검색한 결과를 SGA 영역 안에 Data Buffer Cache영역에 Block 단위로 저장하게 된다.
- SQL문과 PARSING한 결과와 실행계획을 SHARED POOL안에 LIBRARY CACHE에 저장하게 된다.

BC Programming

reparedStatement

- 똑같은 SQL문을 전송하면 LIBRARY CACHE에 저장된 SQL문과 PARSING한 결과와 실행계획을 그대로 사용하게 된다
- 똑 같은 SQL문이라도 대소문자가 하나라도 틀리거나 SQL문이 다르다면 LIBRARY CACHE에 저장된 3가지 작업을 재
  용할 수 없고 다시 PARSING => EXECUTE PLAN => FETCH 작업을 수행하게 된다.




```
String sql = "select age from test1 where id=?";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setString(1,"syh1011");
ResultSet rs = pstmt.executeUpdate();
```

BC Programming

reparedStatement

- PreparedStatement는 SQL문을 작성할 때 컬럼 값을 실제로 지정하지 않고, 변수 처리 함으로서 DBMS을 효율적으로 사용한다.
- PreparedStatement의 SQL문은 SQL문의 구조는 같은데 조건이 수시로 변할 때 조건의 변수처리를 "?" 하는데 이를 인딩 변수라 한다.
- 바인딩 변수는 반드시 컬럼 명이 아닌 컬럼 값이 와야 한다는 것이다.
- 바인딩 변수의 순서는 "?" 의 개수에 의해 결정이 되는데 시작 번호는 1 부터 시작하게 된다.
- 바인딩 변수에 값을 저장하는 메서드는 오라클의 컬럼 타입에 따라 지정해 주면 된다. 전에 ResultSet의 getXXX() 메드와 유사하게 PreparedStatement 인터페이스에는 바인딩 변수에 값을 저장하는 setXXX() 메서드를 제공하고 있다.

allableStatement

- CallableStatement는 DBMS의 저장 프로시저(Stored Procedure)를 호출할 수 있는 인터페이스이다.
- 저장 프로시저란 파라미터를 받을 수 있고, 다른 애플리케이션이나 PL/SQL 루틴에서 호출할 수 있는 이름을 가진 PL/SQL 블록이다. 즉, SQL문을 프로그램화 시켜 함수화 시킨 스크립트 언어이다.
- SQL문을 프로그램화 시켰기 때문에 조건문, 반복문, 변수처리 등을 사용하기 때문에 일괄처리 및 조건에 따라 틀려지는 SQL문을 작성할 때는 유리하다.
- 이런 Stored Procedure 호출을 가능 하게해 줄 수 있는 인터페이스가 CallableStatement 이다.

allableStatement

- CallableStatement는 Connection 인터페이스의 prepareCall(~)를 사용하면 된다.
- prepareCall(String procedure) 의 프로시저는 2가지 형태를 가지고 있다.

```
CallableStatement cstmt = con.prepareCall("{call adjust(?,?)}");
```

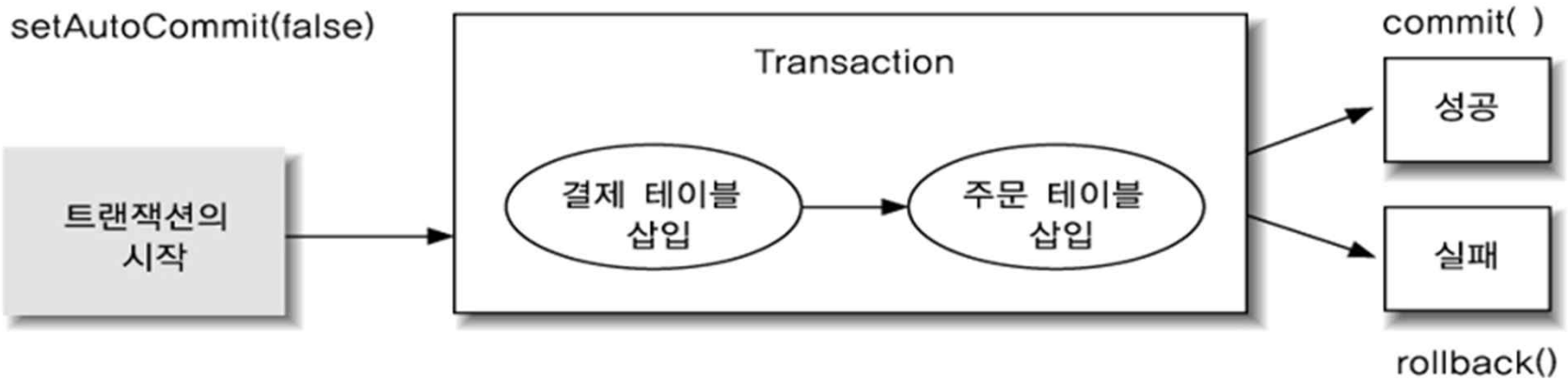
```
1. {?= call <procedure-name>[<arg1>,<arg2>, ...]}
2. {call <procedure-name>[<arg1>,<arg2>, ...]}
```

1번은 <arg1>,<arg2>,..은 PreparedStatement 처럼 바인딩 변수로 처리 하면 되고, ?은 registerOutParameter(~)계열의 메서드를 사용하면 된다.
2번은 <arg1>,<arg2>,..은 PreparedStatement 처럼 바인딩 변수로 처리 한다.

랜잭션(Transcation)

- 트랜잭션이란 여러 개의 오퍼레이션을 하나의 작업 단위로 묶어 주는 것을 말한다.
- 트랜잭션은 하나의 작업 단위의 일들은 전체 작업이 모두 올바르게 수행되거나 또는 전체 작업
- 이 모두 수행되지 않아야 한다.
- 트랜잭션은 네 가지 특성(ACID 특성)을 가지고 있다.

트랜잭션의 특성	설명
원자성(Automicity)	트랜잭션의 포함된 오퍼레이션(작업)들은 모두 수행되거나, 아니면 전혀 수행되지 않아야 한다.
일관성(Consistency)	트랜잭션이 성공적인 경우에는 일관성있는 상태에 있어야 한다.
고립성(Isolation)	각 트랜잭션은 다른 트랜잭션과 독립적으로 수행되는 것처럼 보여야 한다.
지속성(Durability)	성공적으로 수행된 트랜잭션의 결과는 지속성이 있어야 한다.



BC Programming

랜잭션(Transcation)

setAutoCommit(boolean autoCommit) – autoCommit이 true이면 트랜잭션을 시작하지 않겠다는 의미, false 이면 트랜잭션을 시작하겠다는 의미 이다.
commit() – setAutoCommit(false)와 commit() 사이에 있는 모든 operation를 수행하겠다는 의미이다.
rollback() - setAutoCommit(false)와 rollback() 사이에 있는 모든 operation를 수행하지 않겠다는 의미이다.

BC Programming

roperties

- Properites 클래스는 properties파일의 집합을 추상화 클래스이다.
- 이 클래스는 Stream를 로드하여 저장할 수 있고, 이를 다시 Map 형태로 관리하여 Key를 알면 Key에 대한 Value을 어올 수 있는 클래스이다.
- C:₩jdbc.properties 파일을 아래와 같이 만들어보자.
- jdbc.properties 파일을 읽기 위해서는 스트림을 생성해야 한다.
- 스트림을 Properties 클래스에 로드 시키고, Properties클래스의 getProperty(String key) 메서드를 이용해서 각각의 을 얻어 올 수 있다.

```
driver = oracle.jdbc.driver.OracleDriver
url = jdbc:oracle:thin:@localhost:1521:orcl
sser = scott
password = tiger
```

```
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 ee){
		ee.printStackTrace();
}
```

esultSetMetaData

- MetaData란 데이터의 구성요소를 의미한다.
- Table명은 알고 있지만 Table를 구성하는 컬럼명 이라든지 , 컬럼명의 자료형등을 알기 위해서는 ResultSetMeta-Data를 이용해야 한
- ResultSetMetaData는 ResultSet의 구성요소이다. 다시 말해서, SQL의 Table을 구성하는 모든 요소를 알아낼 수 있는 메서드를 제공하
있다.
- ResultSetMetaData 객체를 생성하는 방법은 아래와 같다.

```
rs = pstmt.executeQuery() ;
ResultSetMetaData rsmd = rs.getMetaData() ;
```