17장 JDBC

학습 목표

- DBMS와 SQL에 대해 알아본다
- 2tier와 3tier의 구조에 대해 알아본다.
- JDBC 필요성과 JDBC 코딩에 대해 알아본다
- PreparedStatement의 특징에 대해 알아본다.
- CallableStatement의 특징에 대해 알아본다.
- Transaction 처리에 대해 알아본다.
- JDBC에서 Properties 클래스의 사용 방법에 대해 알아본다.
- ResultSetMetaData의 사용 방법에 대해 알아본다.
- JDBC2.0 과 JDBC3.0에 대해 알아본다.
- Connection Pool에 대해 알아본다.

데이터 베이스

데이터 베이스

- 지속적으로 저장되는 연관된 정보의 모음이다.
- 즉, 특정 관심의 데이터를 수집하여 그 데이터의 성격에 맞도록 잘 설계하여 저장하고 관리함으써 필요한 데이터를 효율적으로 사용할 수 있는 자원이다.

DBMS(Database Management System)

- 데이터를 효율적으로 관리할 수 있는 시스템을 말한다.
- 이런 데이터를 효율적으로 관리하기 위해서는 데이터 베이스에 추가, 삭제, 변경, 검색을 할 수 있는 기능이 있어야 한다.

DBMS 종류

- 계층형, 네트워크용, 릴레이션형으로 구분된다.
- 최근에는 릴레이션형 DBMS가 주류를 이루고 있다.
- 릴레이션형 DBMS를 RDBMS라 하고, 이런 제품으로 Oracle, DB2, MS-SQL, Infomix 등이 있다.

02 관계형 데이터 베이스

기본 키(Primary Key)

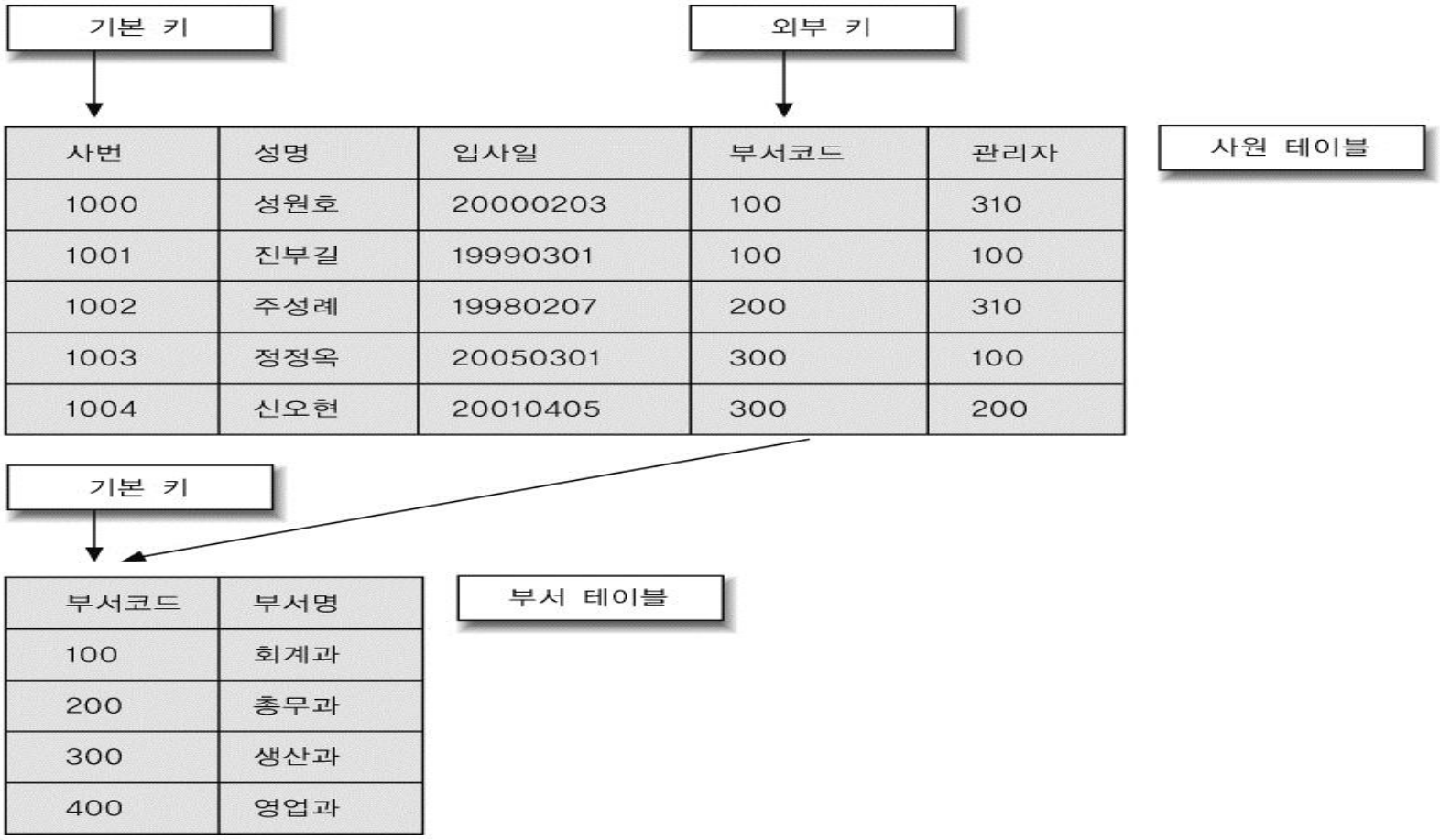
- 기본 키는 테이블의 각 행을 다른 행과 구분해주는 역할을 하는 필드를 말한다.
- 기본 키는 반드시 '유일함' 이라는 조건을 만족해야 하며, '값' 이 있어야 한다.

  즉, NOT NULL 제약 조건과, UNIQUE 제약 조건을 포함해야 한다.

외부 키(Foreign Key)

- 한 테이블의 기본 키에 기반한 관계를 가지 두 개의 테이블이 있는 경우를 위한 것이다.
- 외부 키는 테이블 내의 한 열의 필드인 동시에 다른 테이블의 기본 키인 열의 필드를 말한다.

02 관계형 데이터 베이스



사번	성명	입사일	부서코드	관리자
1000	성원호	20000203	100	310
1001	진부길	19990301	100	100
1002	주성례	19980207	200	310
1003	정정옥	20050301	300	100
1004	신오현	20010405	300	200

부서코드	부서명
100	회계과
200	총무과
300	생산과
400	영업과

[그림 17-1] 기본 키와 외부 키의 관계

03 SQL

SQL(Standard Query Language)

- SQL은 RDBMS의 표준 언어이다.
- SQL문을 이용해서 단순한 쿼리뿐만 아니라 데이터 베이스 객체를 만들거나, 제거하고, 데이터를 삽입, 갱신, 삭제하거나 다양한 운영 작업을 할 수 있다.
- SQL문이 첫선을 보인 것은 1970년대 IBM에 의해서이며, 이후 ANSI/ISO 표준으로 편입되어 여러 차례의 개량과 개발을 거쳤다.
- SQL의 종류는 크게 데이터와 구조를 정의하는 DDL, 데이터의 검색과 수정을 위한 DML, 데이터 베이스의 권한을 정의하는 DCL로 구분할 수 있다.

[표 17-1] DDL과 관련된 SQL문

SQL문	설명
CREATE	데이터베이스 객체를 생성한다.
DROP	데이터베이스의 객체를 삭제한다.
ALTER	기존에 존재하는 데이터베이스의 객체를 다시 정의하는 역할을 한다.

03 SQL

[표 17-2] DML과 관련된 SQL문

SQL문	설명
INSERT	데이터베이스 객체에 데이터를 입력한다.
UPDATE	데이터베이스 객체에 데이터를 갱신한다.
DELETE	데이터베이스 객체에 데이터를 삭제한다.
SELECT	데이터베이스 객체에 데이터로부터 데이터를 검색한다.
COMMIT	커밋 구문 전에 발생한 데이터베이스 액션을 영구히 저장한다.
ROLLBACK	마지막으로 발생한 커밋 후의 데이터베이스 액션들을 원시 데이터로 복구한다.

[표 17-3] DCL과 관련된 SQL문

SQL문	설명
GRANT	데이터베이스 객체에 권한을 부여한다.
REVOKE	이미 부여된 데이터베이스 객체의 권한을 취소한다.

03 SQL

SELECT문

SELECT 문은 데이터 베이스로부터 저장되어 있는 데이터를 검색하는 방법이다.

```
SELECT [ALL | DISTINCT] {*|컬럼,...}
FROM 테이블 명
[WHERE 조건]
[GROUP BY {컬럼,...}]
[HAVING 조건]
[ORDER BY {컬럼,...} [ASC, DESC]]
```

INSERT문

테이블을 사용하여 새로운 행을 삽입하기 위해서 INSERT문을 사용한다.

```
INSERT INTO 테이블명[(컬럼1[, 컬럼2, ... , 컬럼N ])]
VALUES(값1[, 값2, ... , 값N]);
또는
INSERT INTO 테이블명
VALUES(값1[, 값2, ... , 값N]);
```

UPDATE문

테이블을 사용하여 기존 행을 변경하기 위해서 UPDATE문을 사용한다.

```
UPDATE 테이블 명
SET 컬럼1 = 값1 [ , 컬럼2 = 값2 , ... , 컬럼N = 값N]
[WHERE 조건];
```

DELETE 문

테이블을 사용하여 기존 행을 삭제하기 위해서 DELETE문을 사용한다.

```
DELETE
FROM 테이블 명
[WHERE 조건];
```

04 JDBC의 탄생 배경과 구조

JDBC(Java Database Connectivity)

JDBC란 자바를 이용하여 데이터베이스에 접근하여 각종 SQL문을 수행할 수 있도록 제공하는 API를 말한다.

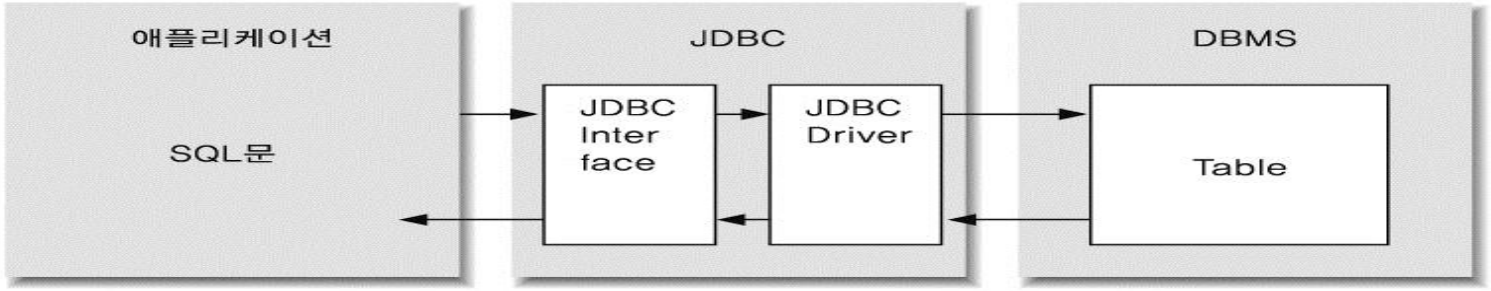
각종 DBMS를 통합한 라이브러리 필요성

- 자바가 데이터베이스에 접근하는 프로그램을 시도할때 한 가지 문제점이 있었다.
- 문제점 : DBMS의 종류가 다양하고, 구조와 특징이 다름.
- 자바는 모든 DBMS에서 공통적으로 사용할 수 있는 인터페이스와 클래스로 구성하는 JDBC를 개발하게 되었고, 실제 구현은 DBMS의 밴더에게 구현하도록 했다.
- 각 DBMS의 벤더에서 제공하는 구현 클래스를 JDBC 드라이버라고 한다.
- JDBC로 코딩하기 위해서는 DBMS를 선택하고, DBMS에서 제공하는 JDBC 드라이버가 반드시 필요하다.

04 JDBC의 탄생 배경과 구조

JDBC의 구조와 역활

JDBC는 크게 JDBC 인터페이스와 JDBC 드라이버로 구성되어 있다.



[그림 17-12] 애플리케이션과 JDBC, DBMS의 관계

- 응용프로그램에서는 SQL문 만들어 JDBC Interface를 통해 전송하면 실제 구현 클래스인 JDBC 드라이버에서 DBMS에 접속을 시도하여 SQL문을 전송하게 된다.
- DBMS의 결과를 JDBC Driver와 JDBC Interface에게 전달되고 이를 다시 응용프로그램으로 전달 되어 SQL문의 결과를 볼 수 있다.
- JDBC의 역할은 Application과 DBMS의 Bridge 역할을 하게 된다.

04 JDBC의 탄생 배경과 구조

JDBC 드라이버의 종류

JDBC 드라이버는 DBMS의 벤더나 다른 연구 단체들에서 만들어진다. JDBC 드라이버는 크게 네 가지로 분류된다.

JDBC-ODBC 드라이버

JDBC API로 작성된 프로그램이 JDBC-ODBC 브리지를 통해 ODBC 드라이버를 JDBC 드라이버로 여기고 동작하도록 한다. 반드시 운영체제 내에 ODBC 드라이버가 존재해야 한다.

데이터 베이스 API 드라이버

JDBC API 호출을 특정 데이터 베이스의 클라이언트 호출 API로 바꿔주는 드라이버다. 오라클 OCI 드라이버가 여기에 속한다.

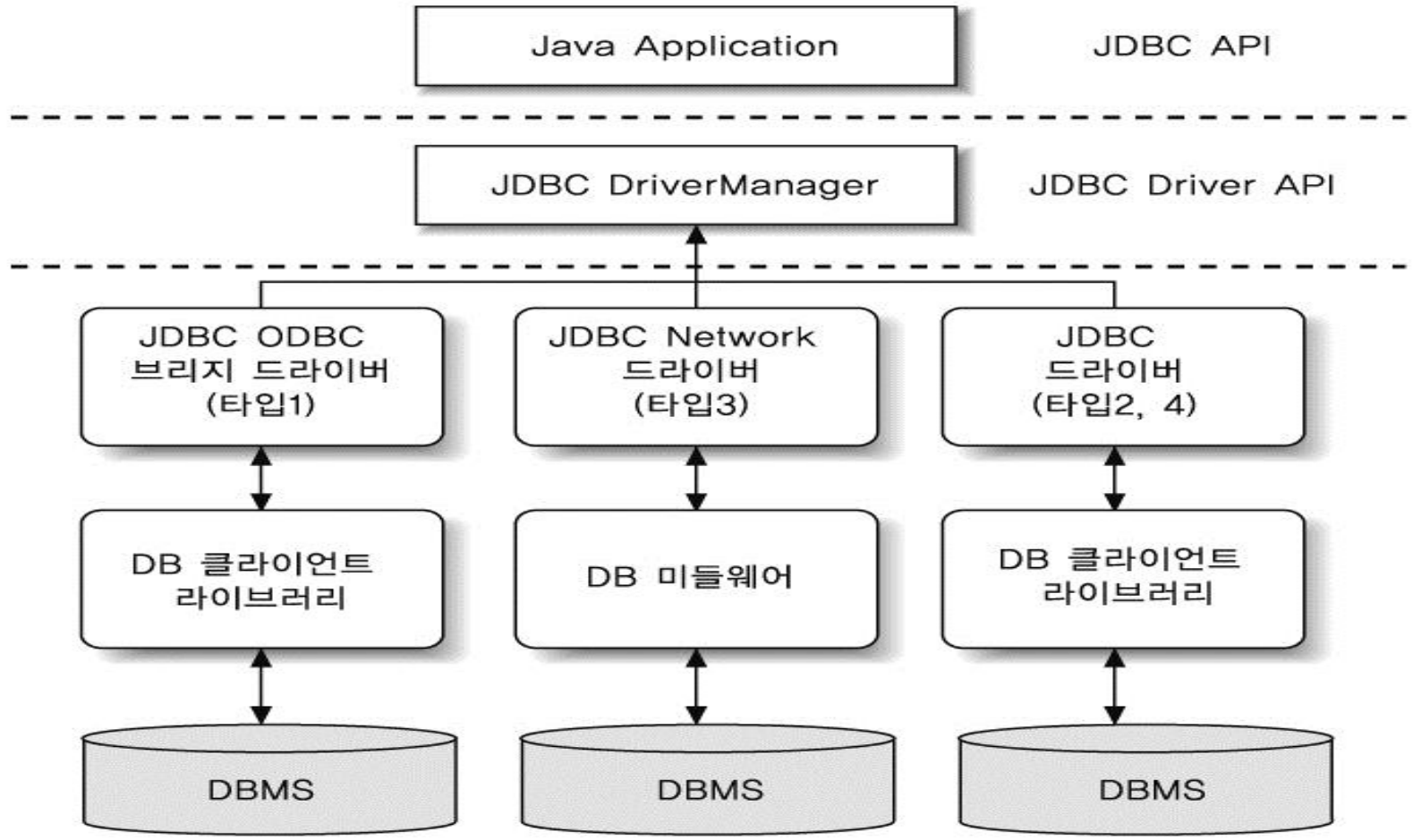
네트워크 프로토콜 드라이버

클라이언트의 JDBC API 호출을 특정 데이터 베이스의 프로토콜과 전혀 상관없는 독자적인 방식의 프로토콜로 바꾸어 서버로 전송한다. 서버에는 미들웨어가 프로토콜을 특정 데이터베이스 API로 바꾸어 처리한다.

데이터 베이스 프로토콜 드라이버

JDBC API 호출을 서버의 특정 데이터 베이스에 맞는 프로토콜로 변환시켜 서버로 전송하는 드라이버(Java Thin Driver) 이다.

04 JDBC의 탄생 배경과 구조



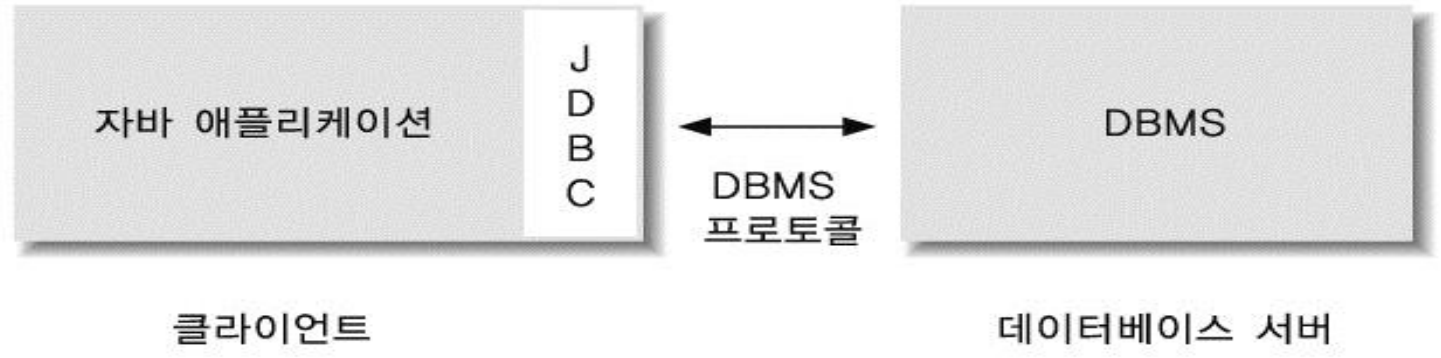
[그림 17-13] JDBC Driver의 타입

04 JDBC의 탄생 배경과 구조

2tier

자바 애플리케이션이 JDBC 드라이버를 통해서 직접 데이터베이스를 접근하는 형식이다.

이 모델에서 JDBC 드라이버는 JDBC API 호출을 통해 특정 DBMS에 직접 전달해 주는 역할을 한다.

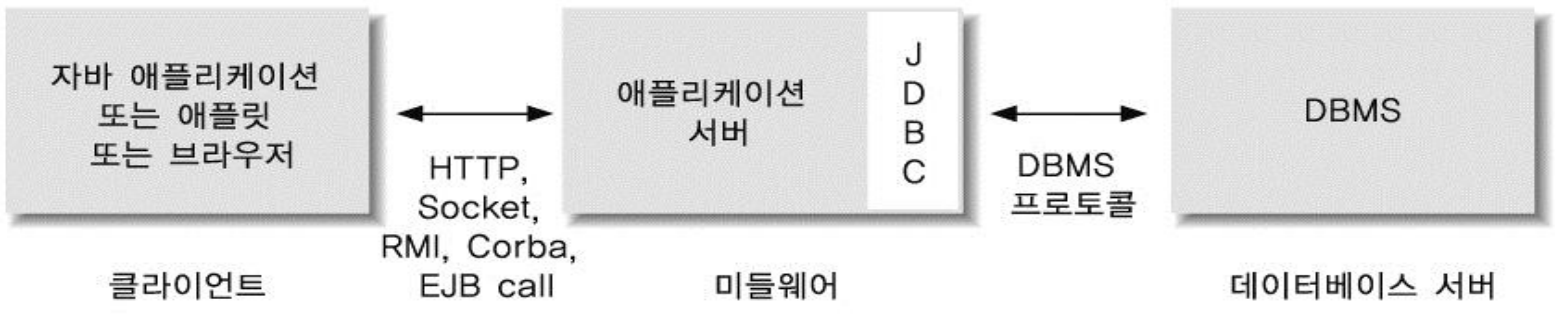


[그림 17-14] 2tier

04 JDBC의 탄생 배경과 구조

3tier

3tier 모델은 2tier 모델에 미들웨어 계층이 추가된 형태이며, 미들웨어에서 DBMS에 직접 질의하게 된다.



[그림 17-15] 3tier

- 3tier 모델은 2tier 모델에 보다 유지보수(Maintenance)에 비용을 절약할 수 있다.
- 단점은 2tier에 비해 속도는 다소 느리다.

04 JDBC를 이용한 데이터베이스 연결방법

JDBC를 이용한 데이터베이스 연결 방법

1 단계 : import java.sql.*;
2 단계 : 드라이버를 로드 한다.
3 단계 : Connection 객체를 생성한다.
4 단계 : Statement 객체를 생성한다.
5 단계 : SQL문에 결과물이 있다면 ResultSet 객체를 생성한다.
6 단계 : 모든 객체를 닫는다.

드라이버 다운 받기와 설정

- JDBC API를 이용해서 DBMS에 접근하기 위해서는 DBMS에서 제공되는 드라이버를 내려 받아야 한다.

  만약, 오라클이 설치되어 있다면 아래의 경로에서 드라이버를 제공하고 있다.

  %ORACLE_HOME%₩ora92₩jdbc₩lib₩classes12.zip

  오라클이 설치되어 있지 않다면 아래의 주소에서 내려 받으면 된다.

  http://download.oracle.com/otn/utilities_drivers/jdbc/101020/ojdbc14.jar

04 JDBC를 이용한 데이터베이스 연결방법

- ojdbc14.jar 파일이 C 드라이브에 있다는 가정에서 JDBC 드라이버를 설정한다.
- 에디트 플러스 : c:\ojdbc14.jar를 CLASSPATH에 추가한다.
- 이클립스 : ‘project에서 오른쪽 클릭 => properties선택 => libraries(탭) 선택 =>
- Add External JARs. 선택 => ojdbc14.jar 를 찾아서 <열기> 버튼 클릭’

JDBC API import

JDBC 코딩하기 위한 첫 번째 단계로 JDBC에서 사용되는 클래스와 인터페이스가 있는 패키지를 import 해야 한다.

```
import java.sql.*;
public class JdbcEx{
}
```

04 JDBC를 이용한 데이터베이스 연결방법

드라이버를 로드한다

- JDBC Driver를 로드해야 한다.2 단계 : 드라이버를 로드 한다.

```
try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
}catch(ClassNotFoundException cnfe){
        cnfe.printStackTrace();
}
```

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

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

04 JDBC를 이용한 데이터베이스 연결방법

- 실제로 driver 객체는 JDBC 프로그램에서 더 이상 사용하지 않기 때문에 위와 같은 코딩은 잘 사용하지 않는다. 2 단계 : 드라이버를 로드 한다.
- Class.forName(~)을 이용하면 OracleDriver클래스가 런타임 메모리에 로딩 되고 DriverManager 클래스의 static 멤버 변수로 저장된다.

```
만약 이 부분에서 java.lang.ClassNotFoundException
에러가 발생 했다면 다음 2가지 사항을 확인해 보도록 한다.
1. oracle.jdbc.driver.OracleDriver의 철자가 정확한지
   확인해 본다.
2. 이클립스에 ojdbc14.jar를 등록 했는지 확인해 본다.
```

04 JDBC를 이용한 데이터베이스 연결방법

Connection 객체를 생성한다

- DBMS와 연결을 담당하는 Connection 객체를 생성한다.

```
try{
        Connection con = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:orcl",
                                "scott",
                                "tiger");
}catch(SQLException sqle){
        sqle.printStackTrace();
}
```

- Connection 객체를 얻어 왔다면 DBMS와 접속이 성공적으로 이루어진 것이다.
- getConnection() 메서드에 들어가는 url, user, password에 대해서 알아보자

  IP - oracle이 설치된 ip를 작성한다.

  PORT - oracle의 포트를 작성하는데 일반적으로 설치 하였다면 1521 이다.

  ORACLE_SID – oracle를 설치할 때 설정하는 것인데 대부분 일정하지 않다. 따라서 ORACLE_SID를 정확히 찾아 확인한 후에 값을 정하도록 하자.(일반적으로 – ORCL)

  user - oracle의 user를 말한다.

  password – oracle user에 대한 password를 말한다.

04 JDBC를 이용한 데이터베이스 연결방법

[표 17-5] Connection의 주요 메서드

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.

04 JDBC를 이용한 데이터베이스 연결방법

Statement 객체를 생성한다

- 오라클과 연결되었다면 SQL문을 전송할 수 있는 Statement 객체를 생성해야 한다.

```
try{
        Statement stmt = con.createStatement();
}catch(SQLException sqle){
        sqle.printStackTrace();
}
```

- Statement 객체는 Connection 인터페이스의 createStatement() 메서드를 사용하여 얻어 올 수 있다.
- Statement 객체를 생성했다면 SQL를 전송할 수 있는데 Statement 인터페이스에는 SQL문을 전송할 수 있는 여러 가지 메서드 중에 3가지에 대해 살펴 보도록 하자.

04 JDBC를 이용한 데이터베이스 연결방법

- **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());
```

- **execute(String sql) – SQL문을 알지 못하는 경우**

04 JDBC를 이용한 데이터베이스 연결방법

- **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);
}
```

04 JDBC를 이용한 데이터베이스 연결방법

[표 17-6] Statement의 주요 메서드

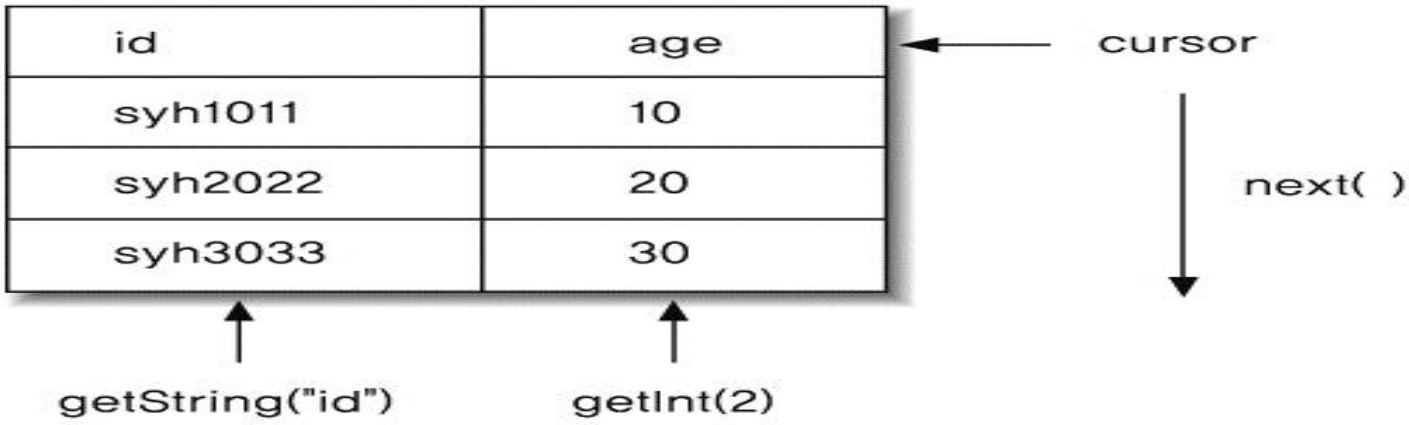
반환형	메서드	설명
void	addBatch(String sql)	Statement 객체에 SQL문을 추가한다. 이 메서드를 이용해서 SQL의 일괄처리를 할 수 있다.
	clearBatch( )	Statement 객체에 모든 SQL문을 비운다.
	close( )	Statement 객체를 해제한다.
boolean	execute(String sql)	매개변수인 SQL문을 수행한다. 만약, 수행한 결과가 ResultSet 객체를 반환하면 true, 어떠한 결과도 없거나, 갱신된 숫자를 반환하면 false를 반환한다.
int[ ]	executeBatch( )	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.

04 JDBC를 이용한 데이터베이스 연결방법

ResultSet 객체를 생성한다

- ResultSet은 SQL문에 대한 결과를 처리할 수 있는 객체이다.
- Statement 인터페이스의 executeQuery() 메서드를 실행한 결과로 ResultSet 객체를 리턴 받는다.

```
select id , age from test
```



[그림 17-27] ResultSet 구조

04 JDBC를 이용한 데이터베이스 연결방법

- 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.

04 JDBC를 이용한 데이터베이스 연결방법

- 예를 들어 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

모든 객체를 닫는다

Connection, Statement, ResultSet 객체는 사용이 끝난 후에는 종료를 해 줘야 한다. 종료 해주는 메서드는 모든 객체에 close() 메서드로 정의 되어 있다

■ 예제[17-1] JdbcEx.java

04 JDBC를 이용한 데이터베이스 연결방법

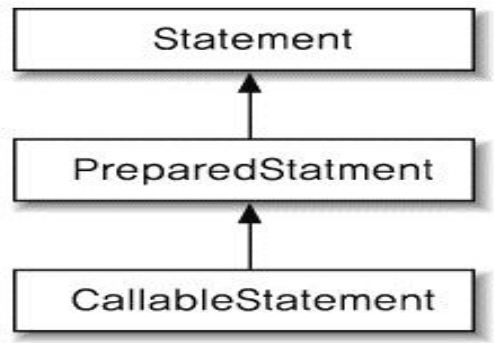
[표 17-7] ResultSet의 주요 메서드

반환형	메서드	설명
boolean	absolute(int row)	ResultSet 객체에서 매개변수 row로 커서를 이동한다. 만약, 매개변수 row로 커서를 이동할 수 있으면 true, 그렇지 않으면 false를 반환한다.
void	afterLast( )	ResultSet 객체에서 커서를 마지막 로우 다음으로 이동한다.
	beforeFirst( )	ResultSet 객체에서 커서를 처음 로우 이전으로 이동한다.
boolean	last( )	ResultSet 객체에서 커서를 마지막 로우로 이동한다. 만약, ResultSet에 row가 있다면 true, 그렇지 않으면 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 이전 row가 있다면 true, 그렇지 않으면 false를 반환한다.
void	close( )	ResultSet 객체를 해제한다.
boolean	first( )	ResultSet 객체에서 커서를 처음 로우로 이동한다. 만약, ResultSet에 row가 있다면 true, 그렇지 않으면 false를 반환한다.
InputStream	getBinaryStream (int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 InputStream으로 반환한다.
	getBinaryStream (String columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 InputStream으로 반환한다.
Blob	getBlob (int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 Blob으로 반환한다.
	getBlob (String columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 Blob으로 반환한다.
byte	getByte (int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 byte로 반환한다.
	getByte (String columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 byte로 반환한다.
Clob	getClob (int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 Clob으로 반환한다.
	getClob (String columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 Clob으로 반환한다.
double	getDouble (int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 double로 반환한다.
	getDouble (String columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 double로 반환한다.
int	getInt (int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 int로 반환한다.
	getInt (String columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 int로 반환한다.
String	getString (int columnIndex)	ResultSet 객체의 현재 로우에 있는 columnIndex의 값을 String으로 반환한다.
	getString (String columnName)	ResultSet 객체의 현재 로우에 있는 columnName의 값을 String으로 반환한다.

05 Statement 상속관계

Statement 상속관계

- Statement 인터페이스는 SQL문을 전송할 수 있는 객체이다.
- 상속관계를 보면 아래와 같다.



[그림 17-29] Statement의 상속관계

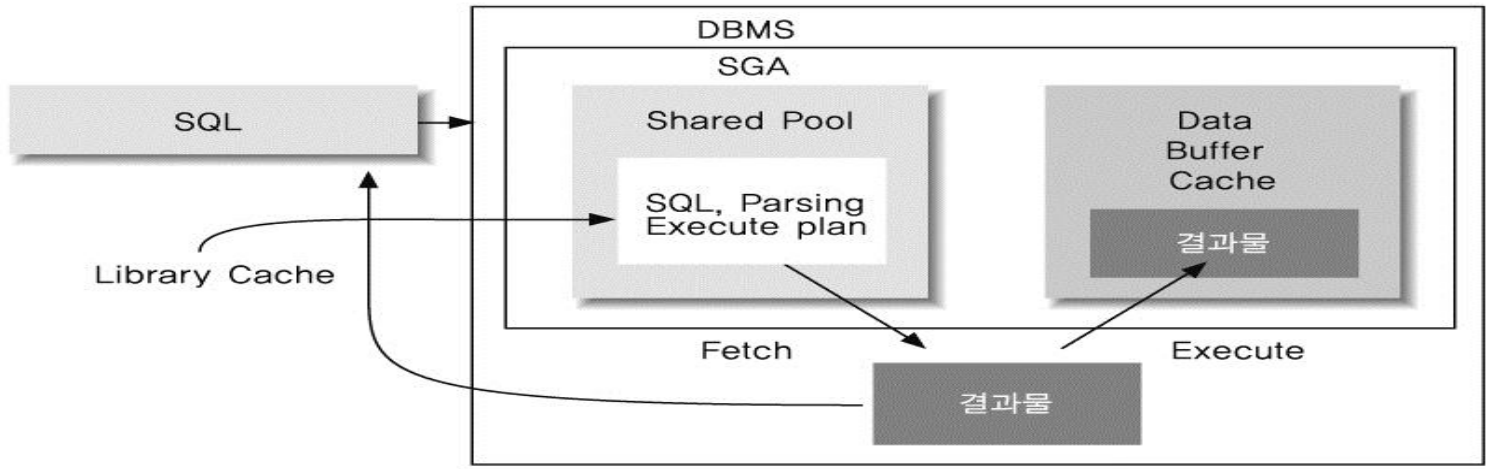
- 서브 인터페이스로 갈 수록 향상된 기능을 제공한다.

05 Statement 상속관계

PreparedStatement 기능

- PreparedStement는 SQL문의 구조는 동일하나 조건이 다른 문장을 변수 처리함으로써 항상 SQL문을 동일하게 처리할 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.

05 Statement 상속관계



[그림 17-30] 오라클에서 SQL문 수행 시 SGA 영역의 메모리 관리

- 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 사용할 수 없고 다시 PARSING => EXECUTE PLAN => FETCH 작업을 수행하게 된다.

05 Statement 상속관계

PreparedStatement 사용방법

- PreparedStatement의 객체 생성은 아래와 같다.

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

- 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 바인딩 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() 메서드와 유사하게 PreparedStatement 인터페이스에는 바인딩
- 변수에 값을 저장하는 setXXX() 메서드를 제공하고 있다.

■ 예제[17-2] PreparedStatementEx.java

05 Statement 상속관계

CallableStatement 기능

- 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.

05 Statement 상속관계

CallableStatement 사용 방법

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

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

- 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.

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

■ 예제[17-4] CallableStatementEx.java

06 JDBC를 이용한 Transaction

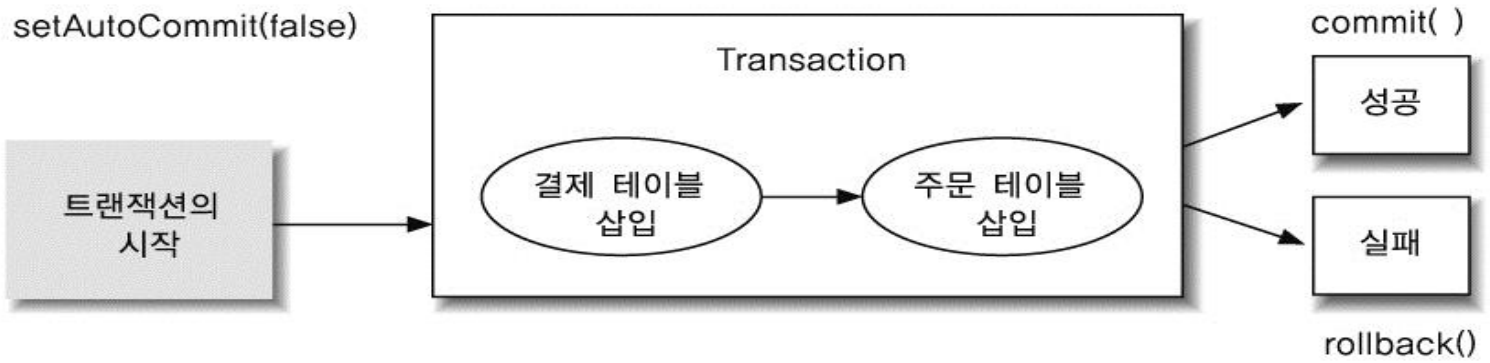
트랜잭션(Transcation)

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

[표 17-8] Transaction의 4가지 특성(ACID)

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.

06 JDBC를 이용한 Transaction



[그림 17-34] 응용프로그램에서 Transaction 처리

- **Connection 인터페이스에서 Transaction과 관련된 메서드를 정리하면 다음과 같다.**

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

- **예제[17-5] TransactionEx.java**

07 Properties 를 이용한 JDBC 연결과 ResultSetMetaData

Properties

- Properites 클래스는 properties파일의 집합을 추상화 클래스이다.
- 이 클래스는 Stream를 로드하여 저장할 수 있고, 이를 다시 Map 형태로 관리하여 Key를 알면 Key에 대한 Value을 얻어올 수 있는 클래스이다.

  C:\jdbc.properties 파일을 아래와 같이 만들어보자.

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

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

07 Properties 를 이용한 JDBC 연결과 ResultSetMetaData

```
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();
}
```

- **Properties 클래스를 이용하여 JDBC를 연결하게 되면 데이터 베이스가 바뀌는 경우, 또는 데이터 베이스의 IP가 바뀌는 경우 , 사용자가 바뀌는 경우에 jdbc.properties 파일만 수정하여 재 컴파일 방지하고, 유지보수에 용이하게 함이다.**

■ 예제[17-7] PropertiesEx.java

07 Properties 를 이용한 JDBC 연결과 ResultSetMetaData

ResultSetMetaData

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

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

예제[17-8] ResultSetMetaDataEx.java

08 JDBC2.0

BatchQuery

- 여러 개의 SQL문을 한꺼번에 전송하는 일괄 처리 방식이다.
- JDBC1.0에서는 executeUpdate() 메서드만을 제공하기 때문에 하나의 SQL문만 처리 가능했다.
- JDBC2.0에서는 executeBatch() 메서드를 제공해서 SQL문을 일괄 처리할 수 있다.
- 모든 SQL문을 처리하지 못한다. 즉 INSERT, UPDATE, DELETE, CREATE , DROP , ALTER문 등에서만 사용할 수 있다.

Scrollable

- JDBC2.0의 가장 큰 변화 중에 하나는 ResultSet의 커서가 양방향으로 움직(Scrollable)인다는 것이다.
- ResultSet의 메서드중에 커서를 내리기위한 메서드로 next() 메서드가 있었다.
- 이것은 커서를 forward 방향으로 움직이는 것인데, JDBC2.0 에서는 backward 방향으로 움직이는 메서드를 제공하고 있다.
- 양방향 커서를 코딩하기 위해서는 createStatement()의 두개의 매개변수 갖는 메서드를 제공하고 있다.
- JDBC2.0 에서는 createStatement(int resultSetType, int resultSetConcurrency) 메서드를 제공하고 있다.

■ 예제[17-9] BatchInsertEx.java

08 JDBC2.0

- 매개변수 인자로 들어가는 2가지 int 값에 대해 알아보자.
- 2가지 인자 값은 ResultSet 의 상수 값으로 존재한다.

- 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.
- resultSetConcurrency => CONCUR_XXX 형태
CONCUR_READ_ONLY : 읽기만 된다.
CONCUR_UPDATABLE : 데이터를 동적으로 갱신할 수 있다.

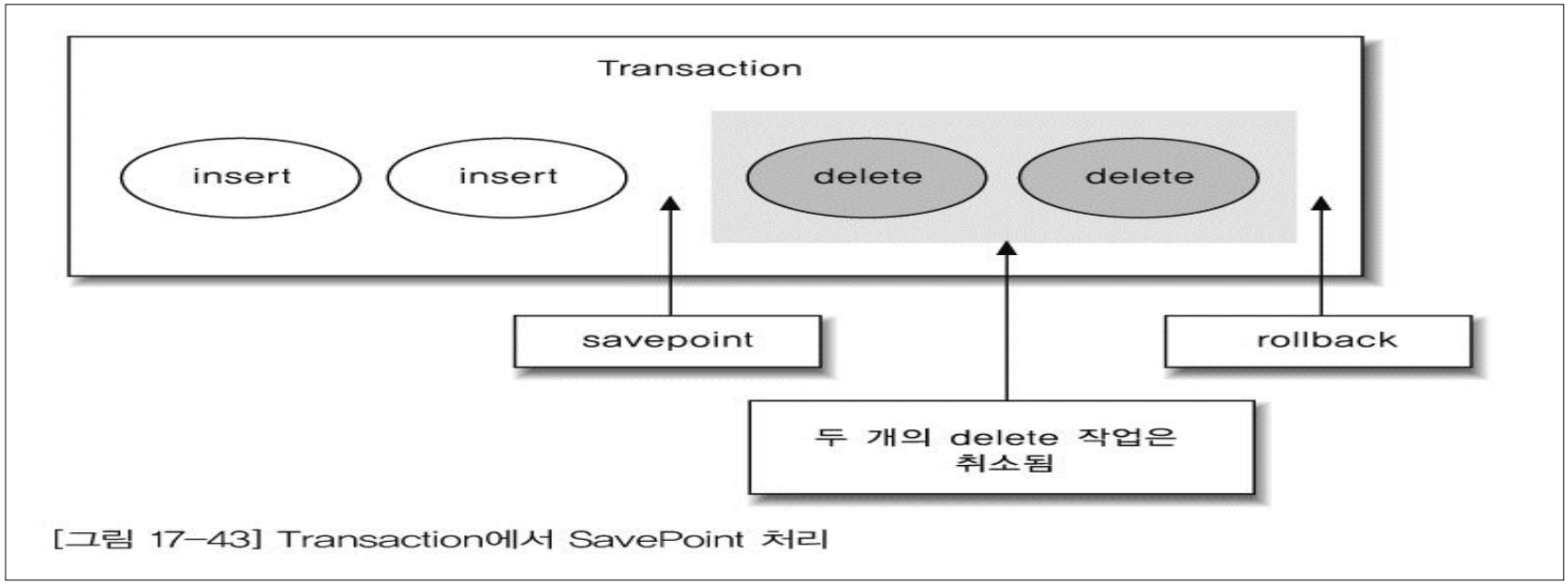
- 양방향 가능한 Statement 객체를 생성하는 방법은 아래와 같다

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

예제[17-10] ScrollableEx.java

BatchQuerySavePoint

- SAVEPOINT는 트랜잭션 내에 세이브 포인트를 만들 수 있게 해준다.
- 하나의 트랜잭션에 내에 여러 개의 세이브 포인트를 지정할 수 있다.
- 트랜잭션 내에 설정 세이브 포인트를 이용하면 세이브 포인트 이전의 작업은 그대로 두고, 세이브 포인트 이후의 작업만 선택적으로 롤백 할 수 있다.



[그림 17-43] Transaction에서 SavePoint 처리

- 예제[17-11] SavepointEx.java

09 JDBC3.0

JDBC3.0

- JDBC RowSet 객체는 ResultSet 객체보다 사용법이 편하고, 더 유연한 방법을 제공한다.
- 썬 마이크로 시스템에서는 RowSet에서 사용 빈도가 높은 5개의 인터페이스를 정의했고,
- JCP와 DB 벤더에서는 5개의 인터페이스를 구현하였다.
- 5개의 인터페이스는 JDK 5.0의 한 부분으로 추가 되었다.
- 5개의 인터페이스에 대한 각각의 구현 클래스는 아래와 같다.

[표 17-9] 각종 RowSet과 구현 클래스의 관계

인터페이스	인터페이스를 구현한 클래스	인터페이스	인터페이스를 구현한 클래스
JdbcRowSet	JdbcRowSetImpl	CachedRowSet	CachedRowSetImpl
JoinRowSet	JoinRowSetImpl	FilteredRowSet	FilteredRowSetImpl
WebRowSet	WebRowSetImpl		

JdbcRowSet

- JdbcRowSet은 기본적으로 ResultSet 객체의 기능을 향상 시켰다.
- 첫 번째 url,user,password, sql문을 설정할 수 있는 메서드를 제공한다. 즉, Connection,
- Statement, ResultSet 객체를 생성하지 않고, JdbcRowSet 객체로 바로 DBMS를 접근 할 수 있다는 뜻이다.

```
JdbcRowSetImpl jdbcRs = new JdbcRowSetImpl();
jdbcRs.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
jdbcRs.setUsername("scott");
jdbcRs.setPassword("tiger");
jdbcRs.setCommand("select * from coffees");
jdbcRs.execute();
```

- 두 번째는 JdbcRowSet은 기본적으로 ResultSet scrollable, update, delete, insert 기능을 가지고 있다.

예제[17-12] JdbcRowSetEx.java

CacheRowSet

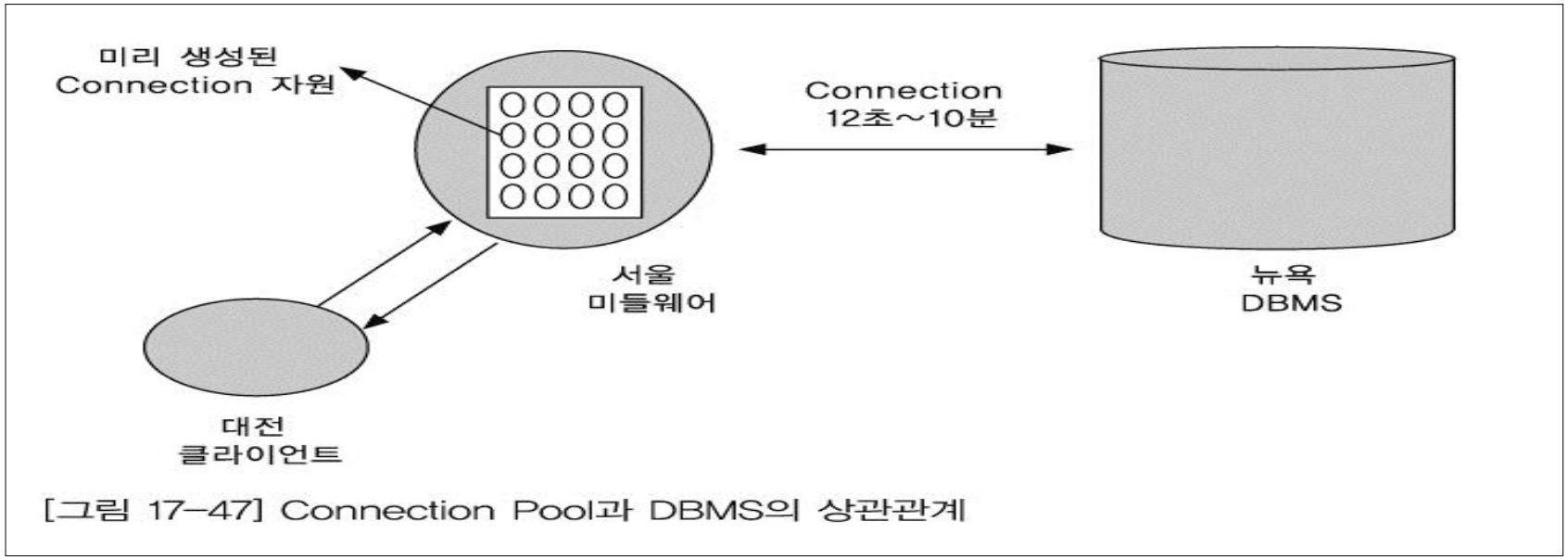
- CachedRowSet의 기능은 JdbcRowsSet의 기능을 모두 가지고 있고, 추가적으로 연결이 끊긴 RowSet 객체를 사용 할 수 있는 기능을 가지고 있다.
- JDBC 코딩에서는 Connection 객체를 close 하게 되면 ResultSet 객체를 역시 사용할 수 없게 되지만, CachedRowSet 객체는 Connection 객체가 close 하더라도 CachedRowSet 객체는 기존의 ResultSet 객체를 캐쉬하고 있게 된다.

■ 예제[17-13] CachedRowSetEx.java

10 Connection Pool

Connection Pool

- 우리는 앞서 3tier에 대한 대략적인 구조를 살펴봤다.
- 3tier구조에서 MIDDLEWARE 와 DBMS의 거리가 멀어진다면 JDBC API중에 Connection 객체를 만드는 비용은 상당할 것이다.
- 왜냐하면 MIDDLEWARE와 DBMS의 실제적인 접속시도는 Connection 객체를 생성할 때 이루어 지기 때문이다.
- JDBC API의 가장 코스트가 비싼 자원인 Connection 객체를 미리 생성하여 재 사용하는 메커니즘을 Connection Pool이라고 한다.

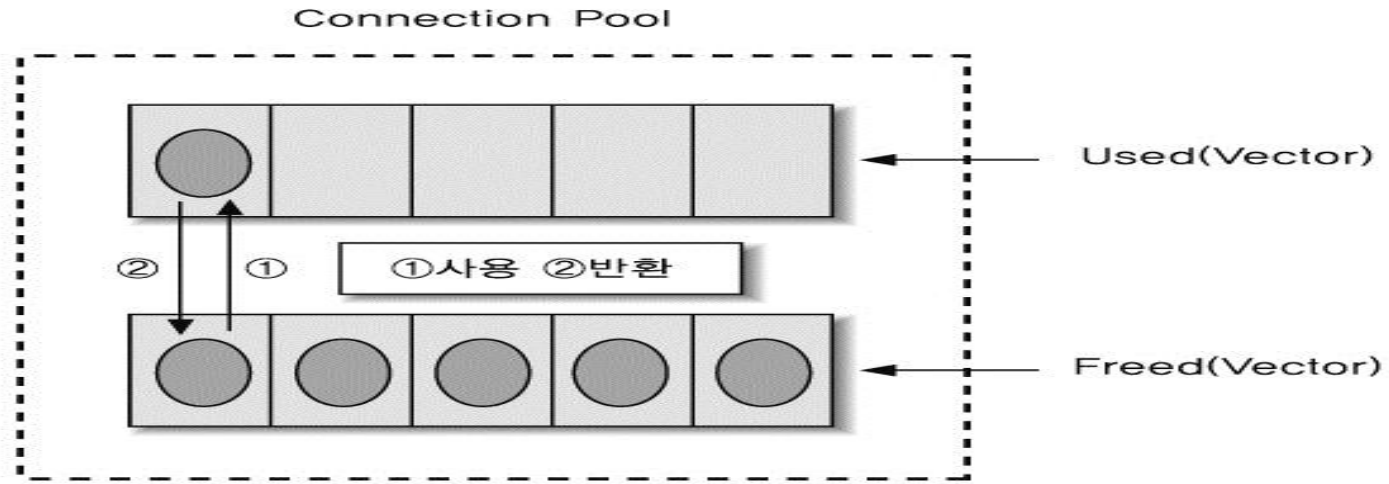


[그림 17-47] Connection Pool과 DBMS의 상관관계

10 Connection Pool

Connection Pool 만들기

- Connection를 저장할 수 있는 두개의 Vector를 생성한다.
- Freed(Vector)는 ConnectionPool 클래스의 객체가 생성될 때 미리 생성된 Connection 객체를 저장하는 장소이다.
- Used(Vector)는 실제 미들웨어에서 DBMS와 연결을 할 때 사용하는 Connection 공간이다.
- 이때 Freed(Vector)에서 Connection 객체를 꺼내와 Used(Vector)에 저장하고 Used(Vector)에 있는 Connection 객체를 실제 애플리케이션에 사용하는 것이다



[그림 17-48] Connection Pool의 프로그램 구조