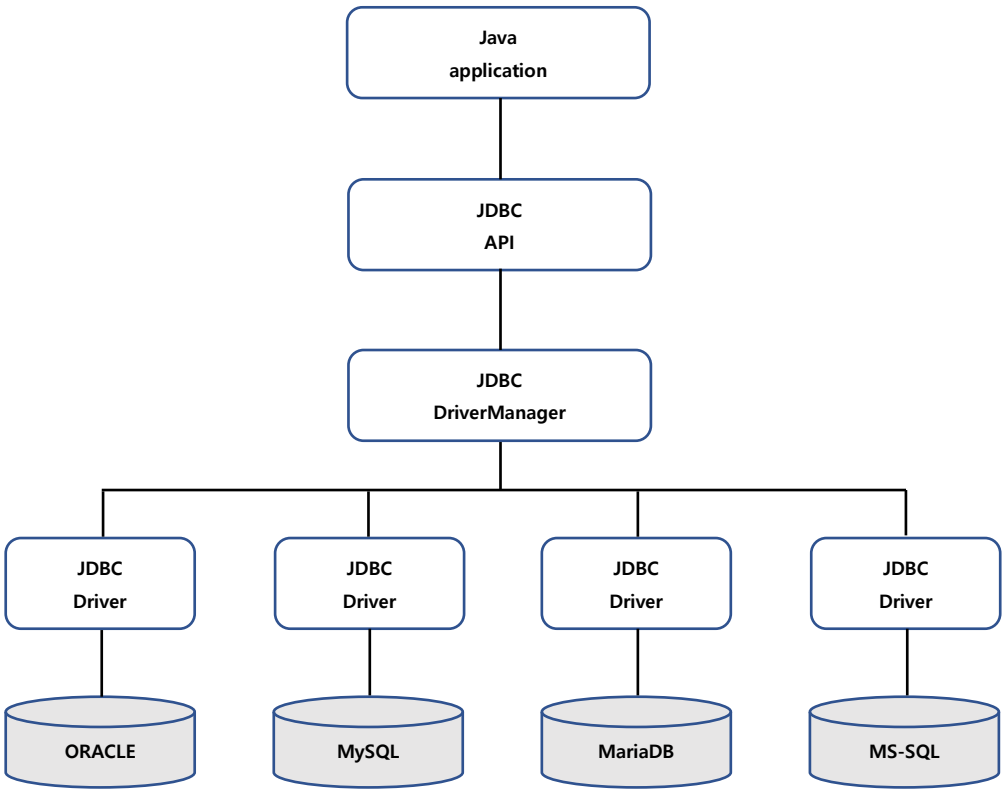
목 차

JDBC (Java Database Connectivity)

JDBC(Java Database Connectivity)는 자바에서 데이터베이스에 접속할 수 있도록 하는 자바 API이다. JDBC는 데이터베이스에서 자료를 쿼리하거나 업데이트하는 방법을 제공한다.



1. DriverManager 클래스

Drivermanager는 J2SE(Java 2 Plaform, Standard Edition) 및 JDK(Java SE Development Kit)의 정적 클래스이다. DriverManager는 사용할 어플리케이션에 대해 사용 가능한 JDBC(Java Database Connectivity)드라이버 세트를 관리한다.

어플리케이션은 필요한 경우 여러 JDBC 드라이버를 동시에 사용할 수 있다. 각 어플리케이션은 URL(Uniform Resource Locator)을 사용하여 JDBC드라이버를 지정한다. 어플리케이션은 DriverManager에 특정 JDBC 드라이버의 경로(FQDN)를 전달하여 DriverManager로 하여금 JDBC 연결 유형이 반환되어야 함을 알린다.

```
Class.forName(String FQDN_OF_DRIVER_CLASS);
```

JDBC 4.0 이전에는 DriverManager가 사용 가능한 JDBC 드라이버를 인식해야 하므로 Class.forName 메소드 호출을 통해 JVM으로 클래스를 로드 한다. 이때 Class.forName 메소드에 전달되는 인수는 문자열을 기반으로 하는 Driver 클래스의 FQDN이다.

Class.forName 메서드는 인수로 주어진 경로의 클래스를 찾아 로딩 하며 클래스 검색 실패 시 ClassNotFoundException 예외를 발생시킨다.

```
try {
      Class.forName("com.mysql.jdbc.Driver");
      System.out.println("JDBC 드라이버가 검색되었습니다.");
}
catch (ClassNotFoundException e) {
      System.err.println("JDBC 드라이버가 설치되지 않았습니다.");
}
```

※ JDBC 4.0 부터는 JDBC 드라이버는 자동으로 로드 되므로 Class.forName 메서드의 호출이 필요하지 않다. 하지만 드라이버 사용 이전에 드라이버 검색 절차를 수행하기 위한 방법으로 Class.forName 메서드의 사용을 권장한다.

JDBC 드라이버는 해당 드라이버 구현 클래스가 로딩될 때 자동으로 자체에 대해 DriverManager에 알리도록 설계 되어있다. 그리고 작업에 사용되는 DriverManager에 대해 getConnection 메서드 호출을 통해 Connection 객체를 획득한다.

```
DriverManager.getConnection(URL);
```

JDBC URL의 가장 간단한 양식은 콜론으로 분리되는 세 개의 리스트이다. 다음은 IP 주소가 192.168.0.254인 MySQL 데이터베이스 서버의 powerlinux DB에 kihee 계정과 iloveyou 비밀번호로 접속하기 위한 URL 형태이다.

```
"jdbc:mysql://192.168.0.254:3306/powerlinux", "kihee", "iloveyou"
```

물론 아래와 같이 하나의 문자열로 생성할 수도 있다. 아래와 같이 하나의 문자열로 URL을 구성하는 경우에는 user와 password 프로퍼티를 지정하여야 한다.

```
"jdbc:mysql://192.168.0.254:3306/powerlinux?user=kihee&password=ilo
veyou"
```

그 밖에도 Properties 오브젝트를 이용하는 방법도 있다.

```
java.util.Properties prop = new java.util.Properties();
prop.put("user", "...");
prop.put("password", "...");
...

"jdbc:mysql://192.168.0.254:3306/powerlinux", prop
```

URL format

```
jdbc:provider://[host][,failoverhost...][:port]/[database][?propert
yName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

Database 서버에 따른 Driver FQDN과 URL

- **Oracle**
  - oracle.jdbc.driver.OracleDriver
  - jdbc:oracle:thin:@[host]:[port=1521]:[instance]
- **MySQL & MariaDB**
  - com.mysql.jdbc.Driver
  - jdbc:mysql://[host]:[port=3306]/[instance]
- **Microsoft SQL Server**
  - com.microsoft.jdbc.sqlserver.SQLServerDriver
  - jdbc:microsoft:sqlserver://[host]:[port=1433];DatabaseName =[DatabaseName];SelectMethod=Cursor

메서드

Modifier and Type	Method and Description
static void	**deregisterDriver ()** 드라이버 매니저의 리스트에서 지정한 드라이버를 삭제한다.
static Connection	**getConnection (String url)** 지정한 데이터베이스 URL을 연결한다. 결과로 Connection 객체를 생성하고 반환한다.
static Connection	**getConnection (String url, Properties info)** 지정한 데이터베이스 URL을 연결한다. 결과로 Connection 객체를 생성하고 반환한다.
static Connection	**getConnection (String url, user, password)**

	지정한 데이터베이스 URL을 연결한다. 결과로 Connection 객체를 생성하고 반환한다.
static Driver	**getDriver (String url)** 지정한 JDBC URL에 해당하는 드라이버를 구한다.
static Enumeration <Driver>	**getDrivers ()** 현재 사용 가능한 모든 로드된 JDBC 드라이버를 열거형으로 반환한다.
static int	**getLoginTimeout ()** 데이터베이스 로그인 시 최대 대기시간을 반환한다.
static PrintWriter	**getLogStream ()** DriverManager와 모든 드라이버에 의해 사용된 기록에 사용되는 스트림을 반환한다.
static void	**println (String messgae)** JDBC 로그출력 스트림으로 메시지를 출력한다.
static void	**registerDriver (Driver driver)** 드라이버를 드라이버 목록에 추가한다. Class.forName() 메서드가 호출 되거나 Driver 클래스가 로드될 때 자동으로 호출되며 직접 호출도 가능하다.
static void	**setLoginTime (int seconds)** 데이터베이스에 로그인 시 최대 대기시간을 지정한다.
static void	**setLogStream (PrintStream out)** DriverManager와 모든 드라이버에 의해 사용된 내용을 기록하기 위한 출력 스트림을 설정한다.

JDBC 경고 및 에러 발생 시 프로퍼티 지정방법

SSL 경고 메시지 발생 시

```
WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
```

useSSL 프로퍼티 지정

```
useSSL=false
```

SSL 미사용 에러 발생 시

```
Error updating database. Cause: java.sql.SQLNonTransientConnectionException: CLIENT_PLUGIN_AUTH is required
```

useSSL 프로퍼티 지정

```
useSSL=false
```

MySQL JDBC 5.1.X 이후 KST 타임존 인식 불가 시

```
The server time zone value 'KST' is unrecognized or represents more than one time zone. You must configure either the server or JDBC driver (via the serverTimezone configuration property) to use a more specifc time zone value if you want to utilize time zone support.
```

serverTimezone 프로퍼티 지정

```
serverTimezone=UTC
```

프로퍼티 구분자 & 인식 오류 발생 시

```
The reference to entity "serverTimezone" must end with the ';' delimiter.
```

프로퍼티 구분자 & 를 사용

```
jdbc:mysql://ip:port/DB?characterEncoding=UTF-8&amp;serverTimezone=UTC
```

Driver Class Deprecaded 메시지 출력 시

```
LOADING CLASS `COM.MYSQL.JDBC.DRIVER'. THIS IS DEPRECATED. THE NEW DRIVER CLASS IS `COM.MYSQL.CJ.JDBC.DRIVER'. THE DRIVER IS AUTOMATICALLY REGISTERED VIA THE SPI AND MANUAL LOADING OF THE DRIVER CLASS IS GENERALLY UNNECESSARY.
```

Driver Class 변경

```
com.mysql.jdbc.Driver -> com.mysql.cj.jdbc.Driver
```

아래의 예제는 현재 사용 가능한 로딩되어 있는 JDBC 드라이버 목록을 출력하는 예제이다.

```
Enumeration<Driver> drivers = DriverManager.getDrivers();
while (drivers.hasMoreElements()) {
	System.out.println(drivers.nextElement());
}
```

결과는 아래와 같으며 MySQL JDBC 드라이버를 사용할 수 있음을 나타낸다.

```
com.mysql.jdbc.Driver@677327b6
com.mysql.fabric.jdbc.FabricMySQLDriver@6d6f6e28
```

JDBC 드라이버가 존재하는가를 판단하기 위해서라면 위의 예제를 활용하기 보다는 Class.forName 메서드를 이용하는 것이 보다 편리하다. Class.forName 메서드는 인수로 주어진 FQDN에 해당하는 클래스를 찾지 못할 경우 ClassNotFoundException 예외를 발생시킨다.

```
try {
	Class.forName("com.mysql.jdbc.Driver");
	System.out.println("JDBC 드라이버 검색에 성공하였습니다.");
}
catch (ClassNotFoundException e) {
	System.err.println("JDBC 드라이버 검색에 실패하였습니다.");
}
```

어떠한 방법을 사용할 것인지 강요하지는 않지만 JDBC 드라이버를 사용하기 이전에 반드시 JDBC 드라이버의 사용가능여부, 즉 JDBC 드라이버의 설치여부를 판단하도록 하여야 한다.

데이터베이스를 사용하는 어플리케이션은 데이터베이스의 접속 시 정보(로그)를 남기는 것이 데이터베이스 관련 문제가 발생하였을 때 문제를 분석하고 해결하기가 쉽다. DriverManager의 setLogWriter 메서드를 이용하여 로그 기록을 위한 출력 스트림을 지정해 두면 데이터베이스의 접속 시 사용된 URL이나 JDBC Driver 정보 등이 지정된 출력 스트림으로 출력한다.

```
PrintWriter out = null;

  try {
        Class.forName("com.mysql.jdbc.Driver");
        System.out.println("JDBC 드라이버 검색에 성공하였습니다.");
        out = new PrintWriter(new BufferedWriter(new FileWriter("log.txt",
  true)), true);
        DriverManager.setLogWriter(out);
  }
  catch (ClassNotFoundException e) {
        System.err.println("JDBC 드라이버 검색에 실패하였습니다.");
  }
  catch (IOException e) {
        System.err.println("JDBC 로그 출력 오류가 발생하였습니다.");
  }
```

2. Connection 인터페이스

Connection은 특정 데이타베이스와의 접속 (세션)을 표현한다. 접속의 컨텍스트 내에서 SQL 문이 실행되고 결과가 리턴 된다.

Connection 객체 중의 데이타베이스는 테이블, 지원하고 있는 SQL 문법, 스토어드 프로시저 및 접속의 기능 등에 대한 정보를 제공한다. 이 정보는 getMetaData 메서드로 취득할 수 있다.

※ Connection 객체는 SQL 문 실행 시 autocommit 모드로 작동된다. 만약 autocommit 모드가 아닐 경우에는 SQL 문 실행 후 commit을 명시적으로 호출하여야 한다.

필드

Modifier and Type	Field and Description
static int	**TRANSACTION_NONE** 트랜잭션이 지원되지 않음을 나타내는 상수
static int	**TRANSACTION_READ_COMMITTED** dirty read, non-repeatable read와 phantom read가 발생할 수 있음을 나타내는 상수
static int	**TRANSACTION_READ_UNCOMMITTED** dirty read는 금지되고 non-repeatable read와 phantom read는 발생할 수 있음을 나타내는 상수
static int	**TRANSACTION_REPEATABLE_READ** dirty read와 non-repeatable read는 금지되고 phantom read가 발생할 수 있음을 나타내는 상수
static int	**TRANSACTION_SERIALIZABLE** dirty read, non-repeatable read와 phantom read가 모두 금지됨을 나타내는 상수

*** Dirty read**

한 트랜잭션 T1 이 테이블의 행을 갱신하고 아직 commit 이 수행되지 않은 상태에서, 다른 트랜잭션 T2 가 그 테이블의 행을 읽으려고 하는 경우, T1 에서 commit() 메소드가 수행되지 않더라도 T2 의 질문에서 T1 의 갱신된 행의 데이터 값으로 읽혀 지는 것을 말한다.

T1 -> A 테이블 갱신
T2 -> A 테이블 조회
T1 -> commit

이런 순서로 트랜잭션이 진행될 경우 Dirty read 가 허용이 되어있으면 commit 전이지만 T2 에는 T1 의 수행결과가 포함된 정보가 포함되고, 허용되지 않을 경우에는 T1 의 수행결과는 포함되지 않는다.

*** non-repeatable read**

트랜잭션 T1 이 행의 값을 읽어오고, 다른 트랜잭션 T2 가 그 행의 값을 변경하고 commit 한 후 트랜잭션 T1 이 다시 그 행의 값을 읽을 때 처음 검색결과와 다른 검색결과가 나오는 것을 말한다.

*** phantom read**

트랜잭션 T1 이 동일한 조건으로 여러 번 읽기를 하는 경우, 중간에 다른 트랜잭션 T2 에 의해 동일 조건으로 추가한 값을 읽을 수 있는 경우를 말한다. 즉 트랜잭션 T1 이 특정조건으로 행의 정보를 검색하고, 중간에 다른 트랜잭션 T2 가 동일 조건의 행을 추가하고 commit 을 수행한 다음, 트랜잭션 T1 이 동일한 조건으로 다시 행의 정보를 검색한 경우 중간에 추가된 행의 정보를 읽을 수 있는 것을 의미한다.

메서드

void	**clearWarnings ()** Connection 객체에 관해서 통지된 모든 경고를 클리어 한다.
void	**close ()** 자동적인 해제를 기다리지 않고, 즉시 Connection 객체의 데이타베이스와 JDBC 자원을 해제한다.
void	**commit ()** 직전의 위탁/롤백(rollback) 이후에 행해진 변경을 모두 유효로 해, Connection 객체가 현재 보관 유지하는 데이타베이스 락을 모두 해제한다.
Statement	**createStatement ()** SQL 문을 데이타베이스에 보내기 위한 Statement 객체를 생성한다.
Statement	**createStatement (int resultSetType, nt re-sultSetConcurrency)** 지정된 형태와 concurrent rocessing 로 ResultSet 객체를 생성하는 Statement 객체를 생성한다.
Statement	**createStatement (int resultSetType, int re-sultSetConcurrency, int resultSetHoldability)** 지정된 형태, concurrent processing 및 holdability 로 ResultSet 객체를 생성하는 Statement 객체를 생성한다

boolean	**getAutoCommit ()** Connection 객체의 현재의 자동 위탁 모드를 취득한다.
String	**getCatalog ()** Connection 객체의 현재의 카탈로그명을 취득한다.
int	**getHoldability ()** Connection 객체를 사용해 생성된 ResultSet 객체의 현재의 보관 유지 기능을 취득한다.
DatabaseMetaData	**getMetaData ()** Connection 객체가 접속을 나타내는 데이타베이스에 관한 메타데이타를 포함하는 DatabaseMetaData 객체를 취득한다.
int	**getTransactionIsolation ()** Connection 객체의 현재의 트랜잭션(transaction) 차단 레벨을 취득한다.
Map <String,Class <?>>	**getTypeMap ()** Connection 에 관련한 Map 객체를 취득한다.
SQLWarning	**getWarnings ()** Connection 객체에 관한 호출에 의해 보고되는 최초의 경고를 취득한다.
boolean	**isClosed ()** Connection 객체가 클로우즈 되고 있는지를 취득한다.
boolean	**isReadOnly ()** Connection 객체가 읽기 전용 모드인가 어떤가를 취득한다
String	**nativeSQL (String sql)** 지정된 SQL 문을 시스템의 본래의 SQL 문법으로 변환한다.
CallableStatement	**prepareCall (String sql)** 데이타베이스의 스토어드 프로시저를 호출하기 위한 CallableStatement 객체를 생성한다.
CallableStatement	**prepareCall (String sql, int resultSetType, int resultSetConcurrency)** 지정된 형태와 concurrent processing 로 ResultSet 객체를 생성하는 CallableStatement 객체를 생성한다.
CallableStatement	**prepareCall (String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)** 지정된 형태와 concurrent processing 로 ResultSet 객체를 생성하는 CallableStatement 객체를 생성한다.

PreparedStatement	**prepareStatement (String sql)** 파라미터 첨부 SQL 문을 데이타베이스에 보내기 위한 PreparedStatement 객체를 작성한다.
PreparedStatement	**prepareStatement (String sql, int autoGeneratedKeys)** 자동 생성 키를 얻는 기능을 가지는 디폴트의 PreparedStatement 객체를 생성한다.
PreparedStatement	**prepareStatement (String sql, int[] columnIndexes)** 지정된 배열에 의해 지정된 자동 생성 키를 돌려주는 기능을 가지는 디폴트의 PreparedStatement 객체를 생성한다.
PreparedStatement	**prepareStatement (String sql, int resultSetType, int resultSetConcurrency)** 지정된 형태와 concurrent processing 로 ResultSet 객체를 생성하는 PreparedStatement 객체를 생성한다.
PreparedStatement	**prepareStatement (String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)** 지정된 형태, concurrent processing 및 보관 유지 기능으로 ResultSet 객체를 생성하는 PreparedStatement 객체를 생성한다.
PreparedStatement	**prepareStatement (String sql, String [] columnNames)** 지정된 배열에 의해 지정된 자동 생성 키를 돌려주는 기능을 가지는 디폴트의 PreparedStatement 객체를 생성한다.
void	**releaseSavepoint (Savepoint savepoint)** 현재의 트랜잭션(transaction)로부터 지정된 Savepoint 객체를 삭제한다.
void	**rollback ()** 현재의 트랜잭션(transaction)로 행해진 변경을 모두 바탕으로 되돌려, Connection 객체가 현재 보관 유지하는 데이타베이스 락을 모두 해제한다.
void	**rollback (Savepoint savepoint** 지정된 Savepoint 객체가 설정된 뒤에 행해진 모든 변경을 바탕으로 되돌린다.
void	**setAutoCommit (boolean autoCommit)** 접속의 자동 위탁 모드가 지정된 상태로 설정한다.
void	**setCatalog (String catalog)**

	Connection 객체의 데이타베이스에 작업을 위한 서브 스페이스를 선택하기 위해서 카탈로그명을 설정한다.
void	**setHoldability (int holdability)** Connection 객체를 사용해 생성된 ResultSet 객체의 보관 유지 기능이 지정된 보관 유지 기능에 변경한다.
void	**setReadOnly (boolean readOnly)** Connection 를 읽기 전용 모드로 설정한다.
Savepoint	**setSavepoint ()** 현재의 트랜잭션(transaction)로 이름이 없는 세이브 포인트를 작성해, 그것을 나타내는 새로운 Savepoint 객체를 리턴한다.
Savepoint	**setSavepoint (String name)** 현재의 트랜잭션(transaction)로 지정된 이름의 세이브 포인트를 작성해, 그것을 나타내는 새로운 Savepoint 객체를 리턴한다.
void	**setTransactionIsolation (int level)** Connection 트랜잭션(transaction) 차단 레벨이 지정된 것으로 변경하는 것을 시도한다.
void	**setTypeMap (Map <String, Class <?>> map)** Connection 객체의 형태 맵으로서 지정된 TypeMap 객체를 인스톨 한다.

아래의 예제는 호스트명이 powerlinux.co.kr 인 데이터베이스 서버의 project 라는 데이터베이스에 유저명 project와 비밀번호 iloveyou7으로 접속하는 예제이다.

```
try {
	Class.forName("com.mysql.jdbc.Driver");
	System.out.println("JDBC 드라이버 검색에 성공하였습니다.");
	out = new PrintWriter(new BufferedWriter(new FileWriter("log.txt",
true)), true);
	DriverManager.setLogWriter(out);
	con = DriverManager.getConnection("jdbc:mysql://power-
linux.co.kr:3306/project?useSSL=false", "project", "iloveyou7");
	System.out.println("데이터베이스 서버 접속에 성공하였습니다.");
}
catch (ClassNotFoundException e) {
	System.err.println("JDBC 드라이버 검색에 실패하였습니다.");
}
catch (SQLException e) {
	System.err.println("데이터베이스 서버 접속에 실패하였습니다.");
}
catch (IOException e) {
	System.err.println("출력 스트림 생성에 실패하였습니다. ");
}
```

```
finally {
	if (con != null) {
		try {
			con.close();
		}
		catch (SQLException e) {}
	}
}
```

위의 예제를 실행한 후 DriverManager.setLogWriter 메서드에 지정된 출력 스트림과 연결된 파일을 오픈하면 아래와 같이 접속과 Connection 반환에 대한 기록 저장되어 있는 것을 확인할 수 있다.

```
DriverManager.getConnection("jdbc:mysql://powerlinux.co.kr:3306/project?us-
eSSL=false")
    trying com.mysql.fabric.jdbc.FabricMySQLDriver
    trying com.mysql.jdbc.Driver
getConnection returning com.mysql.jdbc.Driver
```

DriverManager.getConnection 메서드를 통해 연결객체를 획득하여 데이터베이스에 대한 작업이 완료되면 반드시 연결객체의 close 메서드를 호출함으로써 데이터베이스와의 연결을 종료한다.

3. Statement 인터페이스

Statement는 정적 SQL 문을 실행하고 작성된 결과를 돌려주기 위해서 사용되는 객체이다.

디폴트에서는 Statement 객체 마다 1개의 ResultSet 객체만이 동시에 오픈할 수 있다. Statement인터페이스의 모든 execution 메서드는 문장의 현재의 ResultSet 객체로 오픈 되고 있는 것이 존재할 경우 그것을 묵시적으로 닫는다.

필드

Modifier and Type	Field and Description
static int	**CLOSE_ALL_RESULTS** getMoreResults가 호출될 때 직전까지 오픈되어 있던 모든 ResultSet 객체가 클로우즈 되는 것을 나타내는 상수이다.
static int	**CLOSE_CURRENT_RESULT** getMoreResults가 호출될 때 직전까지 오픈되어 있던 현재 ResultSet 객체가 클로우즈 되는 것을 나타내는 상수이다.
static int	**EXECUTE_FAILED** 배치문의 실행 중에 에러가 발생하였음을 나타내는 상수이다.
static int	**KEEP_CURRENT_RESULT** getMoreResults가 호출될 때 현재의 ResultSet 객체가 클로우즈 되지 않음을 나타내는 상수이다.
static int	**NO_GENERATED_KEYS** 생성된 키의 검색이 불가능 함을 나타내는 상수이다.
static int	**RETURN_GENERATED_KEYS** 생성된 키를 검색 가능함을 나타내는 상수이다.
static int	**SUCCESS_NO_INFO** 배치문이 정상적으로 실행되었지만 영향을 받은 행의 수를 알 수 없음을 나타내는 상수이다.

메서드

void	**addBatch** (String sql) Statement 객체의 현재의 커멘드의 리스트로 지정된 SQL 커멘드를 추가한다.
void	**cancel** ()

	DBMS 및 드라이버의 양쪽 모두가 SQL 문의 종료를 지원하는인 경우에 Statement 객체를 취소한다.
void	**clearBatch** () Statement 객체의 현재의 SQL 커멘드 리스트를 비웁니다.
void	**clearWarnings** () Statement 객체에 관해서 보고된 모든 경고를 클리어 한다.
void	**close** () 자동적으로 클로우즈 될 때 Statement 객체의 데이타베이스와 JDBC 자원이 해방되는 것을 기다리는 것이 아니라, 즉시 그것들을 해방한다.
boolean	**execute** (String sql) 복수의 결과를 돌려줄 가능성이 있는 지정된 SQL 문을 실행한다.
boolean	**execute** (String sql, int autoGeneratedKeys) 복수의 결과를 돌려줄 가능성이 있는 지정된 SQL 문을 실행해, 모든 자동 생성 키를 검색 가능하게 할지 어떨지에 임해서 드라이버에 통지한다.
boolean	**execute** (String sql, int[] columnIndexes) 복수의 결과를 돌려줄 가능성이 있는 지정된 SQL 문을 실행해, 지정된 배열로 나타난 자동 생성 키를 검색 가능하게 할지 어떨지에 임해서 드라이버에 통지한다.
boolean	**execute** (String sql, String [] columnNames) 복수의 결과를 돌려줄 가능성이 있는 지정된 SQL 문을 실행해, 지정된 배열로 나타난 자동 생성 키를 검색 가능하게 할지 어떨지에 임해서 드라이버에 통지한다.
int[]	**executeBatch** () 커멘드의 배치를 데이타베이스에 송신해 실행해, 모든 커멘드가 정상적으로 실행되면, 갱신 카운트의 배열을 리턴한다.
ResultSet	**executeQuery** (String sql) 단일의 ResultSet 객체를 돌려주는 지정된 SQL 문을 실행한다.
int	**executeUpdate** (String sql) 지정된 SQL 문을 실행한다.
int	**executeUpdate** (String sql, int autoGeneratedKeys) 지정된 SQL 문을 실행해, Statement 객체에 의해 생성된 자동 생성 키를 검색 가능하게 할지 어떨지에 임해서 지정된 플래그로 드라이버에 통지한다.

int	**executeUpdate** (String sql, int[] columnIndexes) 지정된 SQL 문을 실행해, 지정된 배열로 나타난 자동 생성 키를 검색 가능하게 할지 어떨지에 임해서 드라이버에 통지한다.
int	**executeUpdate** (String sql, String [] columnNames) 지정된 SQL 문을 실행해, 지정된 배열로 나타난 자동 생성 키를 검색 가능하게 할지 어떨지에 임해서 드라이버에 통지한다.
Connection	**getConnection** () Statement 객체를 생성한 Connection 객체를 취득한다.
int	**getFetchDirection** () Statement 객체로부터 생성된 결과 세트의 디폴트인, 데이타베이스 테이블에서 행을 페치 할 방향을 취득한다.
int	**getFetchSize** () Statement 객체로부터 생성된 ResultSet 객체의 디폴트의 페치 사이즈인, 결과 세트의 행수를 취득한다.
ResultSet	**getGeneratedKeys** () Statement 객체를 실행한 결과적으로 작성된 자동 생성 키를 취득한다.
int	**getMaxFieldSize** () Statement 객체에 의해 생성되는 ResultSet 객체의 문자 및 바이너리의 각 렬치에 대해 리턴된 최대 바이트수를 취득한다.
int	**getMaxRows** () Statement 객체에 의해 생성되는 ResultSet 객체가 포함할 수 있는 최대의 행수를 취득한다.
boolean	**getMoreResults** () Statement 객체의 다음의 결과로 이동한다.
boolean	**getMoreResults** (int current) Statement 객체의 다음의 결과로 이동한다.
int	**getQueryTimeout** () 드라이버가 Statement 객체의 실행을 기다리는 초수를 취득한다.
ResultSet	**getResultSet** () ResultSet 객체로서 현재의 결과를 취득한다.
int	**getResultSetConcurrency** ()

	Statement 객체로부터 생성된 ResultSet 객체의 결과 세트의 병행성을 취득한다.
int	**getResultSetHoldability** () Statement 객체로부터 생성된 ResultSet 객체의 결과 세트의 보관 유지 기능을 취득한다.
int	**getResultSetType** () Statement 객체로부터 생성된 ResultSet 객체의 결과 세트의 형태를 취득한다.
int	**getUpdateCount** () 갱신 카운트로서 현재의 결과를 취득한다.
SQLWarning	**getWarnings** () Statement 객체에 관한 호출에 의해 보고되는 최초의 경고를 취득한다.
void	**setCursorName** (String name) 후속의 Statement 객체의 execute 메서드에 의해 사용되는 SQL 커서명이 지정된 String 로 설정한다.
void	**setEscapeProcessing** (boolean enable) 이스케이프의 처리를 온 또는 오프로 설정한다.
void	**setFetchDirection** (int direction) Statement 객체를 사용해 작성된 ResultSet 객체의 행이 처리될 방향에 대한 힌트를 드라이버에 제공한다.
void	**setFetchSize** (int rows) 보다 많은 행이 필요한 때에 데이타베이스로부터 꺼낼 필요가 있는 행수에 대한 힌트를 JDBC 드라이버에 제공한다.
void	**setMaxFieldSize** (int max) 문자 또는 바이너리의 값을 포함하는 ResultSet 열에 대한 최대 바이트수의 제한치를, 지정된 바이트수로 설정한다.
void	**setMaxRows** (int max) 임의의 ResultSet 객체가 포함할 수 있는 최대행수의 제한치를, 지정된 수로 설정한다.
void	**setQueryTimeout** (int seconds) 드라이버가 Statement 객체의 실행을 기다리는 초수를, 지정된 초수로 설정한다.

Statement 객체는 연결객체의 createStatement 메서드 호출을 통해 획득한다. 아래의 예제는 연결객체의 createStatement 메서드 호출로 얻어진 Statement 객체의 queryUpdate 메서드를 호출하여 테이블에 데이터를 입력하는 예제이다.

```
try {
	Class.forName("com.mysql.jdbc.Driver");
	out = new PrintWriter(new BufferedWriter(new FileWriter("log.txt",
true)), true);
	DriverManager.setLogWriter(out);
	con = DriverManager.getConnection("jdbc:mysql://power-
linux.co.kr:3306/project?useSSL=false", "project", "iloveyou7");
}
catch (ClassNotFoundException e) {
	System.err.println("JDBC 드라이버 검색에 실패하였습니다.");
}
catch (SQLException e) {
	System.err.println("데이터베이스 서버 접속에 실패하였습니다.");
}
catch (IOException e) {
	System.err.println("출력 스트림 생성에 실패하였습니다.");
}

try {
	stmt = con.createStatement();
	result = stmt.executeUpdate("INSERT INTO parts VALUES (4, 'CM',
'코믹')");
	DriverManager.getLogWriter().println(stmt);
	System.out.println(result + "건이 입력되었습니다.");
}
catch (SQLException e) {
	System.err.println("데이터 입력에 실패하였습니다.");
}
finally {
	if (stmt != null) {
		try {
			stmt.close();
		} catch (SQLException e) {}
	}
	if (con != null) {
		try {
			con.close();
		} catch (SQLException e) {}
	}
}
```

Statement 객체의 executeUpdate 메서드는 인수로 주어진 SQL구문을 실행하고 SQL 구문을 실행함으로써 영향을 받은 행의 수를 반환한다. 일반적으로 입력이나 삭제, 갱신이 executeUpdate 메서드를 사용하는 반면 데이터 검색 구문의 경우 executeQuery 메소드를 사용 한다.

executeQuery 메서드는 executeUpdate 와 마찬가지로 인수로 주어진 SQL 문을 수행하지만 반환되는 결과는 검색된 행의 수가 아닌 검색된 결과를 ResultSet 으로 반환한다.

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

```
rs = stmt.executeQuery("SELECT * FROM parts");
DriverManager.getLogWriter().println(stmt);
rs.last();
System.out.println(rs.getRow());
```

ResultSet 에 대한 내용은 이후 자세히 설명하도록 할 것이다. 위의 예제에서는 parts 테이블로 부터 검색된 결과가 몇 건인가를 출력한다.

위에서도 언급하였지만 입력이나 삭제 또는 갱신은 일반적으로 executeUpdate 메서드를 활용한다. 이것은 executeQuery 메서드를 통해 얻어진 ResultSet 객체를 이용하여 데이터를 삭제하거나 입력하는 것이 가능함을 말한다.

executeQuery 메서드를 통해 얻어진 ResultSet 객체를 통해 데이터를 조작하기 위해서는 Statement 생성 시 ResultSet에 대한 타입과 Concurrency 타입을 지정해 주어야 하며 ResultSet 타입과 Concurrency 타입은 아래와 같다. 참고로 아래의 타입은 ResultSet의 맴버이다.

ResultSet type

TYPE_FORWARD_ONLY	: 스크롤이 불가능하며 forward only
TYPE_SCROLL_INSENSITIVE	: scroll은 가능하지만 변경사항은 적용 안됨
TYPE_SCROLL_SENSITIVE	: scroll이 기능하며 변경사항도 적용 됨

Concurrency type

CONCUR_READ_ONLY	: ResultSet 객체에 대한 변경 불가
CONCUR_UPDATABLE	: ResultSet 객체에 대한 변경 가능

아래의 예제는 ResultSet 객체를 통해 데이터를 입력하는 예제이다.

```
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CON-
CUR_UPDATABLE);
rs = stmt.executeQuery("SELECT * FROM parts");
DriverManager.getLogWriter().println(stmt);
rs.afterLast();
rs.moveToInsertRow();  // 삽입할 행을 작성한다. 이때 ResultSet 외부 버퍼에 작성
rs.updateInt(1, 5);    // 각 컬럼의 값을 지정
rs.updateString(2, "EB");
rs.updateString(3, "전자도서");
rs.insertRow();                // ResultSet에 작성시도
rs.moveToCurrentRow(); // ResultSet내의 커서 재배치
```

Statement 객체나 ResultSet 객체는 사용이 완료되고 난 후 반드시 close 메서드를 호출하도록 한다.

4. PreparedStatement 인터페이스

PreparedStatement는 프리 컴파일 된 SQL 문을 나타낸다. SQL 문은 프리 컴파일(PreCompile)되어 PreparedStatement 객체에 포함된다. 이때 이 객체를 이용하면 문장을 여러 차례 효율적으로 실행하는 목적으로 사용할 수 있다.

※ preparedStatement 객체의 IN 파라미터값를 설정하는 메서드 (setShort, setString 등)는 입력 파라미터의 정의된 SQL 형과 호환이 있는 형태를 지정하여야 한다.

메서드

Modifier and Type	Field and Description
void	**addBatch** () 이 PreparedStatement 객체의 커멘드의 배치에 파라미터세트를 추가한다.
void	**clearParameters** () 현재의 파라미터값를 곧바로 클리어 한다.
boolean	**execute** () 이 PreparedStatement 객체의, 모든 종류의 SQL 문을 실행한다.
ResultSet	**executeQuery** () 이 PreparedStatement 객체의 SQL 쿼리를 실행해, 그 쿼리에 의해 생성된 ResultSet 객체를 리턴한다.
int	**executeUpdate** () 이 PreparedStatement 객체의 SQL INSERT 문, UPDATE 문, 또는 DELETE 문을 실행한다.
ResultSetMetaData	**getMetaData** () 이 PreparedStatement이 실행될 때 리턴된 ResultSet 객체의 열에 관한 정보를 포함하는 ResultSetMetaData 객체를 취득한다.
ParameterMetaData	**getParameterMetaData** () 이 PreparedStatement 객체의 파라미터의 수, 형태 및 프로퍼티를 취득한다.
void	**setArray** (int i, Array x) 지정된 파라미터가 지정된 Array 객체로 설정한다.
void	**setAsciiStream** (int parameterIndex, InputStream x, int length)

	지정된 파라미터를, 지정된 바이트수를 가지는 지정된 입력 스트림로 설정한다.
void	**setBigDecimal** (int parameterIndex, BigDecimal x) 지정된 파라미터가 지정된 java.math.BigDecimal값으로 설정한다.
void	**setBinaryStream** (int parameterIndex, InputStream x, int length) 지정된 파라미터를, 지정된 바이트수를 가지는 지정된 입력 스트림로 설정한다.
void	**setBlob** (int i, Blob x) 지정된 파라미터가 지정된 Blob 객체로 설정한다.
void	**setBoolean** (int parameterIndex, boolean x) 지정된 파라미터가 지정된 Java boolean값으로 설정한다.
void	**setByte** (int parameterIndex, byte x) 지정된 파라미터가 지정된 Java byte값으로 설정한다.
void	**setBytes** (int parameterIndex, byte[] x) 지정된 파라미터가 지정된 Java 바이트 배열로 설정한다.
void	**setCharacterStream** (int parameterIndex, Reader reader, int length) 지정된 파라미터를, 지정된 문자수인 지정된 Reader 객체로 설정한다.
void	**setClob** (int i, Clob x) 지정된 파라미터가 지정된 Clob 객체로 설정한다.
void	**setDate** (int parameterIndex, Date x) 지정된 파라미터가 지정된 java.sql.Date값으로 설정한다.
void	**setDate** (int parameterIndex, Date x, Calendar cal) 지정된 Calendar 객체를 사용해, 지정된 파라미터가 지정된 java.sql.Date값으로 설정한다.
void	**setDouble** (int parameterIndex, double x) 지정된 파라미터가 지정된 Java double값으로 설정한다.
void	**setFloat** (int parameterIndex, float x) 지정된 파라미터가 지정된 Java float값으로 설정한다.
void	**setInt** (int parameterIndex, int x) 지정된 파라미터가 지정된 Java int값으로 설정한다.

void	**setLong** (int parameterIndex, long x) 지정된 파라미터가 지정된 Java long값으로 설정한다.
void	**setNull** (int parameterIndex, int sqlType) 지정된 파라미터를 SQL NULL로 설정한다.
void	**setNull** (int paramIndex, int sqlType, String typeName) 지정된 파라미터를 SQL NULL로 설정한다.
void	**setObject** (int parameterIndex, Object x) 지정된 파라미터의 값을 지정된 객체를 사용해 설정한다.
void	**setObject** (int parameterIndex, Object x, int targetSqlType) 지정된 파라미터의 값을 지정된 객체로 설정한다.
void	**setObject** (int parameterIndex, Object x, int targetSqlType, int scale) 지정된 파라미터의 값을 지정된 객체를 사용해 설정한다.
void	**setRef** (int i, Ref x) 지정된 파라미터가 지정된 REF(<structured-type>)값으로 설정한다.
void	**setShort** (int parameterIndex, short x) 지정된 파라미터가 지정된 Java short값으로 설정한다.
void	**setString** (int parameterIndex, String x) 지정된 파라미터가 지정된 Java String값으로 설정한다.
void	**setTime** (int parameterIndex, Time x) 지정된 파라미터가 지정된 java.sql.Time값으로 설정한다.
void	**setTime** (int parameterIndex, Time x, Calendar cal) 지정된 Calendar 객체를 사용해, 지정된 파라미터가 지정된 java.sql.Time값으로 설정한다.
void	**setTimestamp** (int parameterIndex, Timestamp x) 지정된 파라미터가 지정된 java.sql.Timestamp값으로 설정한다.
void	**setTimestamp** (int parameterIndex, Timestamp x, Calendar cal) 지정된 Calendar 객체를 사용해, 지정된 파라미터가 지정된 java.sql.Timestamp값으로 설정한다.
void	**setUnicodeStream** (int parameterIndex, InputStream x, int length)

	사용을 추천하지 않음.
void	**setURL** (int parameterIndex, URL x) 지정된 파라미터가 지정된 java.net.URL값으로 설정한다.

PreparedStatement와 Statement의 차이점은 캐시(Cache)의 사용 여부이다. 예를 들어 Statement 를 통해 SQL 구문을 수행하는 경우 매번 아래와 같은 단계를 수행하게 된다.

① SQL 구문 분석
② 컴파일
③ 실행

그러나 PreparedStatement는 처음 한번 만 위의 세 단계를 거친 후 캐시에 담아 재사용을 한다. 그러므로 동일한 쿼리를 반복적으로 수행할 경우 PreparedStatement는 Statement에 비해 DB 부하기 적다.

그렇다면 언제 PreparedStatement를 사용하여야 할까? 예를 들어 아래와 같이 사용자의 입력 값이나 변수 값으로 SQL 구문을 생성하고 수행할 경우이다.

```
pstmt = con.prepareStatement("INSERT INTO parts VALUES (?, ?, ?)");
pstmt.setInt(1, 6);
pstmt.setString(2, "SO");
pstmt.setString(3, "스페이스 오페라");
result = pstmt.executeUpdate();
System.out.println(result + "건이 입력되었습니다.");
```

위 예제에서 PreparedStatement의 setXXX 메서드는 prepareStatement 메서드 호출 시 주어진 SQL 구문의 파라미터 (?)에 대한 값이며 인덱스는 1부터 시작한다.

5. ResultSet 인터페이스

ResultSet 객체는 데이터베이스의 결과 세트를 나타내는 데이터의 테이블로 보통 데이타베이스를 조회 문장의 실행에 의해 생성된다.

ResultSet 객체는 커서가 데이터의 현재의 행을 지시하도록 유지한다. 초기 상태의 ResultSet 객체의 커서는 최초의 행의 선두에 배치된다. next 메서드에 의해 커서는 다음의 행으로 이동한다. next는 ResultSet 객체에 더 이상 행이 없는 경우 false 를 반환하므로 while 루프에 사용해 결과 세트를 반복 처리할 수 있다.

필드

Modifier and Type	Field and Description
static int	**CLOSE_CURSORS_AT_COMMIT** Connection.commit 메서드가 호출될 때 ResultSet 객체가 close 됨을 나타내는 상수
static int	**CONCUR_READ_ONLY** 갱신 불가능을 나타내는 상수
static int	**CONCUR_UPDATABLE** 갱신 가능함을 나타내는 상수
static int	**FETCH_FORWARD** ResultSet 객체의 행에 대한 처리가 처음에서 마지막 방향으로 처리됨을 나타내는 상수
static int	**FETCH_REVERSE** ResultSet 객체의 행에 대한 처리가 마지막에서 처음 방향으로 처리됨을 나타내는 상수
static int	**FETCH_UNKNOWN** ResultSet 객체의 행에 대한 처리 순서가 불분명함을 나타내는 상수
static int	**HOLD_CURSORS_OVER_COMMIT** Connection.commit 메서드가 호출될 때 ResultSet 객체가 close 되지 않음을 나타내는 상수
static int	**TYPE_FORWARD_ONLY** ResultSet 객체내의 커서가 처음에서 마지막 방향으로만 이동 가능함을 나타내는 상수
static int	**TYPE_SCROLL_INSENSITIVE** ResultSet 객체내의 커서가 이동(스크롤)은 가능하지만 변경을 반영하지 않음을 나타내는 상수

static int	**TYPE_SCROLL_SENSITIVE** ResultSet 객체내의 커서가 이동(스크롤)의 이동이 가능하며 변경을 반영함을 나타내는 상수

메서드

Modifier and Type	Field and Description
boolean	**absolute** (int row) 커서를 이 ResultSet 객체내의 지정된 행으로 이동한다.
void	**afterLast** () 커서를 이 ResultSet 객체의 종단, 즉 맨 마지막 줄의 직후로 이동한다.
void	**beforeFirst** () 커서를 이 ResultSet 객체의 첨단, 즉 선두행의 직전으로 이동한다.
void	**cancelRowUpdates** () 이 ResultSet 객체의 현재의 행에 대해서 간 갱신을 취소한다.
void	**clearWarnings** () 이 ResultSet 객체에 관해서 보고된 모든 경고를 클리어 한다.
void	**close** () 자동적으로 클로우즈 될 때, 이것을 기다리는 것이 아니라, 즉시 ResultSet 객체의 데이타베이스와 JDBC 자원을 해방한다.
void	**deleteRow** () 이 ResultSet 객체 및 기본으로 되는 데이타베이스로부터, 현재의 행을 삭제한다.
int	**findColumn** (String columnName) 지정된 ResultSet 렬명을 ResultSet 열인덱스에 매핑 한다.
boolean	**first** () 커서를 이 ResultSet 객체내의 선두행으로 이동한다.
Array	**getArray** (int i) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Array 객체로서 취득한다.
Array	**getArray** (String colName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Array 객체로서 취득한다.
InputStream	**getAsciiStream** (int columnIndex)

	이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 ASCII 문자의 스트림로서 취득한다.
InputStream	**getAsciiStream** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 ASCII 문자의 스트림로서 취득한다.
BigDecimal	**getBigDecimal** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 java.math.BigDecimal 객체로서 전정밀도로 취득한다.
BigDecimal	**getBigDecimal** (int columnIndex, int scale) **추천하지 않음.**
BigDecimal	**getBigDecimal** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 java.math.BigDecimal 객체로서 전정밀도로 취득한다.
BigDecimal	**getBigDecimal** (String columnName, int scale) **추천하지 않음.**
InputStream	**getBinaryStream** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 미해석의 바이트의 바이너리 스트림로서 취득한다.
InputStream	**getBinaryStream** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 미해석의 바이트 스트림로서 취득한다.
Blob	**getBlob** (int i) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Blob 객체로서 취득한다.
Blob	**getBlob** (String colName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Blob 객체로서 취득한다.
boolean	**getBoolean** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 boolean로 서 취득한다.
boolean	**getBoolean** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 boolean로 서 취득한다.
byte	**getByte** (int columnIndex)

	이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 byte로 서 취득한다.
byte	**getByte** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 byte로 서 취득한다.
byte[]	**getBytes** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 byte 배열로서 취득한다.
byte[]	**getBytes** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 byte 배열로서 취득한다.
Reader	**getCharacterStream** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 java.io.Reader 객체로서 취득한다.
Reader	**getCharacterStream** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 java.io.Reader 객체로서 취득한다.
Clob	**getClob** (int i) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Clob 객체로서 취득한다.
Clob	**getClob** (String colName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Clob 객체로서 취득한다.
int	**getConcurrency** () 이 ResultSet 객체의 concurrent processing 모드를 취득한다.
String	**getCursorName** () 이 ResultSet 객체가 사용하는 SQL 커서의 이름을 취득한다.
Date	**getDate** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Date 객체로서 취득한다.
Date	**getDate** (int columnIndex, Calendar cal) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Date 객체로서 취득한다.
Date	**getDate** (String columnName)

	이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Date 객체로서 취득한다.
Date	**getDate** (String columnName, Calendar cal) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Date 객체로서 취득한다.
double	**getDouble** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 double로 서 취득한다.
double	**getDouble** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 double로 서 취득한다.
int	**getFetchDirection** () 이 ResultSet 객체의 페치 방향을 취득한다.
int	**getFetchSize** () 이 ResultSet 객체의 페치 사이즈를 취득한다.
float	**getFloat** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 float로 서 취득한다.
float	**getFloat** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 float로 서 취득한다.
int	**getInt** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 int로 서 취득한다.
int	**getInt** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 int로 서 취득한다.
long	**getLong** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 long로 서 취득한다.
long	**getLong** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 long로 서 취득한다.
ResultSetMetaData	**getMetaData** ()

	이 ResultSet 객체의 열의 수, 형태 및 프로퍼티를 취득한다.
Object	**getObject** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Object로 서 취득한다.
Object	**getObject** (int i, Map <String, Class <? >> map) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Object로 서 취득한다.
Object	**getObject** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Object로 서 취득한다.
Object	**getObject** (String colName, Map <String, Class <? >> map) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Object로 서 취득한다.
Ref	**getRef** (int i) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Ref 객체로서 취득한다.
Ref	**getRef** (String colName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 Ref 객체로서 취득한다.
int	**getRow** () 현재의 행의 번호를 취득한다.
short	**getShort** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 short로 서 취득한다.
short	**getShort** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 short로 서 취득한다.
Statement	**getStatement** () 이 ResultSet 객체를 생성한 Statement 객체를 취득한다.
String	**getString** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 String로 서 취득한다.
String	**getString** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로

	그램 언어의 String로 서 취득한다.
Time	**getTime** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Time 객체로서 취득한다.
Time	**getTime** (int columnIndex, Calendar cal) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Time 객체로서 취득한다.
Time	**getTime** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Time 객체로서 취득한다.
Time	**getTime** (String columnName, Calendar cal) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Time 객체로서 취득한다.
Timestamp	**getTimestamp** (int columnIndex) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Timestamp 객체로서 취득한다.
Timestamp	**getTimestamp** (int columnIndex, Calendar cal) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Timestamp 객체로서 취득한다.
Timestamp	**getTimestamp** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 java.sql.Timestamp 객체로서 취득한다.
Timestamp	**getTimestamp** (String columnName, Calendar cal) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.sql.Timestamp 객체로서 취득한다.
int	**getType** () 이 ResultSet 객체의 형태를 리턴한다.
InputStream	**getUnicodeStream** (int columnIndex) **추천하지 않음.** *대신에 getCharacterStream를 사용*
InputStream	**getUnicodeStream** (String columnName) **추천하지 않음.** *대신에 getCharacterStream를 사용한다.*
URL	**getURL** (int columnIndex)

	이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.net.URL 객체로서 취득한다.
URL	**getURL** (String columnName) 이 ResultSet 객체의 현재행에 있는 지정된 열의 값을 Java 프로그램 언어의 java.net.URL 객체로서 취득한다.
SQLWarning	**getWarnings** () 이 ResultSet 객체에 관한 호출에 의해 보고되는 최초의 경고를 리턴한다.
void	**insertRow** () 삽입행의 내용을 이 ResultSet 객체 및 데이타베이스에 삽입한다.
boolean	**isAfterLast** () 커서가 이 ResultSet 객체내의 맨 마지막 줄보다 뒤에 있을지 어떨지를 취득한다.
boolean	**isBeforeFirst** () 커서가 이 ResultSet 객체내의 선두행보다 전에 있을지 어떨지를 취득한다.
boolean	**isFirst** () 커서가 이 ResultSet 객체내의 선두행에 있을지 어떨지를 취득한다.
boolean	**isLast** () 커서가 이 ResultSet 객체의 맨 마지막 줄에 있을지 어떨지를 취득한다.
boolean	**last** () 커서를 이 ResultSet 객체내의 맨 마지막 줄로 이동한다.
void	**moveToCurrentRow** () 커서를, 기억되고 있는 커서 위치 (일반적으로 현재의 행)로 이동한다.
void	**moveToInsertRow** () 커서를 삽입행으로 이동한다.
boolean	**next** () 커서를 현재의 위치로부터 1 행 하로 이동한다.
boolean	**previous** () 커서를 이 ResultSet 객체내의 앞의 행으로 이동한다.
void	**refreshRow** ()

	현재의 행을 데이타베이스내의 최신의 값으로 재표시한다.
boolean	**relative** (int rows) 커서를 정 또는 부의 상대행수만큼 이동한다.
boolean	**rowDeleted** () 행이 삭제되고 있는지를 취득한다.
boolean	**rowInserted** () 현재의 행에 삽입이 있었는지 어떠했는지를 취득한다.
boolean	**rowUpdated** () 현재의 행이 갱신되고 있는지를 취득한다.
void	**setFetchDirection** (int direction) 이 ResultSet 객체내의 행이 처리될 방향에 대한 힌트를 제공한다.
void	**setFetchSize** (int rows) 이 ResultSet 객체로 보다 많은 행이 필요한 때에 데이타베이스로부터 꺼낼 필요가 있는 행수에 대한 힌트를 JDBC 드라이버에 제공한다.
void	**updateArray** (int columnIndex, Array x) 지정된 열을 java.sql.Array값으로 갱신한다.
void	**updateArray** (String columnName, Array x) 지정된 열을 java.sql.Array값으로 갱신한다.
void	**updateAsciiStream** (int columnIndex, InputStream x, int length) 지정된 열을 ASCII 스트림치로 갱신한다.
void	**updateAsciiStream** (String columnName, InputStream x, int length) 지정된 열을 ASCII 스트림치로 갱신한다.
void	**updateBigDecimal** (int columnIndex, BigDecimal x) 지정된 열을 java.math.BigDecimal값으로 갱신한다.
void	**updateBigDecimal** (String columnName, BigDecimal x) 지정된 열을 java.sql.BigDecimal값으로 갱신한다.
void	**updateBinaryStream** (int columnIndex, InputStream x, int length) 지정된 열을 바이너리 스트림치로 갱신한다.
void	**updateBinaryStream** (String columnName, InputStream x, int length)

	지정된 열을 바이너리 스트림치로 갱신한다.
void	**updateBlob** (int columnIndex, Blob x) 지정된 열을 java.sql.Blob값으로 갱신한다.
void	**updateBlob** (String columnName, Blob x) 지정된 열을 java.sql.Blob값으로 갱신한다.
void	**updateBoolean** (int columnIndex, boolean x) 지정된 열을 boolean값으로 갱신한다.
void	**updateBoolean** (String columnName, boolean x) 지정된 열을 boolean값으로 갱신한다.
void	**updateByte** (int columnIndex, byte x) 지정된 열을 byte값으로 갱신한다.
void	**updateByte** (String columnName, byte x) 지정된 열을 byte값으로 갱신한다.
void	**updateBytes** (int columnIndex, byte[] x) 지정된 열을 byte 배열치로 갱신한다.
void	**updateBytes** (String columnName, byte[] x) 지정된 열을 byte 배열치로 갱신한다.
void	**updateCharacterStream** (int columnIndex, Reader x, int length) 지정된 열을 문자 스트림치로 갱신한다.
void	**updateCharacterStream** (String columnName, Reader reader, int length) 지정된 열을 문자 스트림치로 갱신한다.
void	**updateClob** (int columnIndex, Clob x) 지정된 열을 java.sql.Clob값으로 갱신한다.
void	**updateClob** (String columnName, Clob x) 지정된 열을 java.sql.Clob값으로 갱신한다.
void	**updateDate** (int columnIndex, Date x) 지정된 열을 java.sql.Date값으로 갱신한다.
void	**updateDate** (String columnName, Date x) 지정된 열을 java.sql.Date값으로 갱신한다.
void	**updateDouble** (int columnIndex, double x)

	지정된 열을 double값으로 갱신한다.
void	**updateDouble** (String columnName, double x) 지정된 열을 double값으로 갱신한다.
void	**updateFloat** (int columnIndex, float x) 지정된 열을 float값으로 갱신한다.
void	**updateFloat** (String columnName, float x) 지정된 열을 float값으로 갱신한다.
void	**updateInt** (int columnIndex, int x) 지정된 열을 int값으로 갱신한다.
void	**updateInt** (String columnName, int x) 지정된 열을 int값으로 갱신한다.
void	**updateLong** (int columnIndex, long x) 지정된 열을 long값으로 갱신한다.
void	**updateLong** (String columnName, long x) 지정된 열을 long값으로 갱신한다.
void	**updateNull** (int columnIndex) null을 설정 가능한 열에 null 치를 설정한다.
void	**updateNull** (String columnName) 지정된 열을 null값으로 갱신한다.
void	**updateObject** (int columnIndex, Object x) 지정된 열을 Object값으로 갱신한다.
void	**updateObject** (int columnIndex, Object x, int scale) 지정된 열을 Object값으로 갱신한다.
void	**updateObject** (String columnName, Object x) 지정된 열을 Object값으로 갱신한다.
void	**updateObject** (String columnName, Object x, int scale) 지정된 열을 Object값으로 갱신한다.
void	**updateRef** (int columnIndex, Ref x) 지정된 열을 java.sql.Ref값으로 갱신한다.
void	**updateRef** (String columnName, Ref x) 지정된 열을 java.sql.Ref값으로 갱신한다.
void	**updateRow** ()

	기본으로 되는 데이타베이스를, 이 ResultSet 객체의 현재의 행의 새로운 내용에 갱신한다.
void	**updateShort** (int columnIndex, short x) 지정된 열을 short값으로 갱신한다.
void	**updateShort** (String columnName, short x) 지정된 열을 short값으로 갱신한다.
void	**updateString** (int columnIndex, String x) 지정된 열을 String값으로 갱신한다.
void	**updateString** (String columnName, String x) 지정된 열을 String값으로 갱신한다.
void	**updateTime** (int columnIndex, Time x) 지정된 열을 java.sql.Time값으로 갱신한다.
void	**updateTime** (String columnName, Time x) 지정된 열을 java.sql.Time값으로 갱신한다.
void	**updateTimestamp** (int columnIndex, Timestamp x) 지정된 열을 java.sql.Timestamp값으로 갱신한다.
void	**updateTimestamp** (String columnName, Timestamp x) 지정된 열을 java.sql.Timestamp값으로 갱신한다.
boolean	**wasNull** () 마지막

디폴트의 ResultSet 객체는 갱신이 불가능하며 커서는 순차적으로 아래 방향으로만 이동이 가능하다. 따라서 이 객체는 최초의 행으로부터 마지막 행으로 향해 1회 만 실행할 수 있다. 스크롤 또는 갱신, 혹은 앞 뒤로 커서가 이동 가능한 ResultSet 객체를 생성할 수 있다.

```
Statement stmt = con.createStatement(
                            ResultSet.TYPE_SCROLL_INSENSITIVE,
                            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
```

ResultSet 인터페이스는 현재의 행으로부터 열 값을 얻는 메서드 (getBoolean, getLong 등)를 제공한다. 값은 열의 인덱스 번호나 컬럼명을 사용해 취득할 수 있다. 일반적으로 컬럼 인덱스를 사용하는 편이 효과적이다. 컬럼은 1 부터 순차적으로 번호가 부여된다.

getter 메서드에서는 JDBC 드라이버가 기본적인 데이터를 getter 메서드로 지정된 Java 형으로 변환해, 적절한 Java 값을 리턴한다. JDBC 사양에는 ResultSet getter

메서드로 가능한 SQL 형으로부터 Java 형에의 매핑을 나타내는 테이블이 있다.

getter 메서드 입력에서 사용되는 컬럼명은 대문자와 소문자를 구분하지 않는다. 컬럼명으로 getter 메서드를 호출할 때 동일한 이름의 컬럼이 여러 개 존재하는 경우 최초로 일치하는 컬럼의 값이 리턴 된다. 컬럼명의 옵션은 결과 세트로 생성되는 SQL 쿼리로 컬럼명이 사용되는 경우에 사용되도록 설계되어 있다. 쿼리에 명시적으로 명명되지 않는 컬럼의 경우에는 컬럼 인덱스를 사용하는 것이 효율적인 방법이다. 컬럼명을 사용할 경우, 실제로 목적의 컬럼을 가리키는 것을 보증할 방법이 프로그래머에게는 없기 때문이다.

JDBC 2.0 API (JDK 1.2)로 updater 메서드 세트가 추가되었다. getter 메서드의 파라미터에 관한 내용은 updater 메서드의 파라미터에도 적용된다.

updater 메서드는 다음의 2개의 방법으로 사용할 수 있다.

현재의 행의 컬럼 값을 갱신한다. 스크롤 가능한 ResultSet 객체에서는 커서를 순서 방향 및 역방향으로 작동시켜 절대 위치 또는 현재의 행과의 상대적인 위치에 가지고 갈 수가 있다.

아래의 코드는 ResultSet 객체 rs의 5행 째에 있는 NAME 열을 갱신하고, 계속하여 메서드 updateRow를 사용함으로써 rs를 취득한 데이터 소스 테이블을 갱신한다.

```
rs.absolute(5);
rs.updateString("NAME", "AINSWORTH");
rs.updateRow();
```

컬럼 값을 삽입 행에 삽입한다. 갱신 가능한 ResultSet 객체에는 이 객체에 관련한 특수한 행이 있다. 이 행이 삽입되는 행은 작성하는 집결지의 역할을 한다. 다음의 코드 fragment는 커서를 삽입 행에 이동시키고 3 열의 행을 작성한 후, 메서드 insertRow를 사용해 그 행을 rs 및 데이터 소스 테이블에 삽입한다.

```
rs.moveToInsertRow();
rs.updateString(1, "AINSWORTH");
rs.updateInt(2,35);
rs.updateBoolean(3, true);
rs.insertRow();
rs.moveToCurrentRow();
```

마지막 행의 moveToCurrentRow 메서드는 insertRow에 의해 추가된 행이 추가된 ResultSet 객체 내의 커서를 재배치 하기 위한 메서드로 moveToCurrentRow 이 외에 first나 last 또는 beforeFirst 나 afterLast, absolute 메서드 등을 이용할 수도 있다.

ResultSet객체는 이 객체를 생성한 Statement 객체가 닫혀지거나 재실행될 때, 또는 일련의 복수의 결과로부터 다음의 결과를 꺼내기 위해서 사용될 때, 자동적으로 닫

혀진다.

ResultSet 객체의 열의 수, 형태 및 프로퍼티는 ResultSet.getMetaData 메서드에 의해 리턴 되는 ResulSetMetaData 객체로 제공된다.

ResultSet 객체로부터 데이터를 추출(fetch)하기 위해서는 우선 커서를 데이터를 추출하기 위한 행으로 이동시킨 후 getXXX 메서드를 이용하여 현재 커서 행으로부터 컬럼 값을 추출한다.

```
pstmt = con.prepareStatement("SELECT * FROM parts");
rs = pstmt.executeQuery();
while (rs.next()) {
	int id = rs.getInt(1);
	String code = rs.getString(2);
	String name = rs.getString(3);
	System.out.println(String.format("%3d %-3s %s", id, code, name));
}
```

executeQuery 메서드를 통해 얻어진 ResultSet 객체내의 커서는 최소 첫 번째 행 이전에 위치하며 만약 next 메서드 호출 시 행이 하나도 존재하지 않을 경우 false를 반환한다. 즉 next 메서드 호출의 결과가 true 인 경우 커서가 이동 되었음을 의미하며 커서가 행을 가리키고 있음을 의미한다.

getXXX 메서드의 인수는 SQL 구문의 SELECT 절에 지정된 컬럼의 인덱스이며 setXXX 메서드와 마찬가지로 0이 아닌 1부터 시작된다는 것에 유의하여야 한다.

getXXX 메서드에서 인덱스 대신 컬럼명을 사용할 수 도 있다.

```
pstmt = con.prepareStatement("SELECT * FROM parts");
rs = pstmt.executeQuery();
while (rs.next()) {
	int id = rs.getInt("id");
	String code = rs.getString("p_code");
	String name = rs.getString("p_name");
	System.out.println(String.format("%3d %-3s %s", id, code, name));
}
```

마지막으로 PreparedStatement 은 캐시 기능을 사용하므로 executeQuery 메서드나 executeUpdate 메서드에 실행되는 SQL 구문을 확인할 수 있다.

```
System.out.println(pstmt);
```

만약 DriverManager.setLogWriter 메서드 호출을 통해 로그 출력 스트림을 지정하였다면 아래의 코드를 이용해 실행된 SQL 구문을 로그로 출력이 가능하다.

```
DriverManager.getLogWriter().println(pstmt);
```