Module 02

JDBC를 이용한 데이터베이스 조회

NHN Academy

Overview

- JDBC 프로그래밍 모델
- JDBC 주요 객체

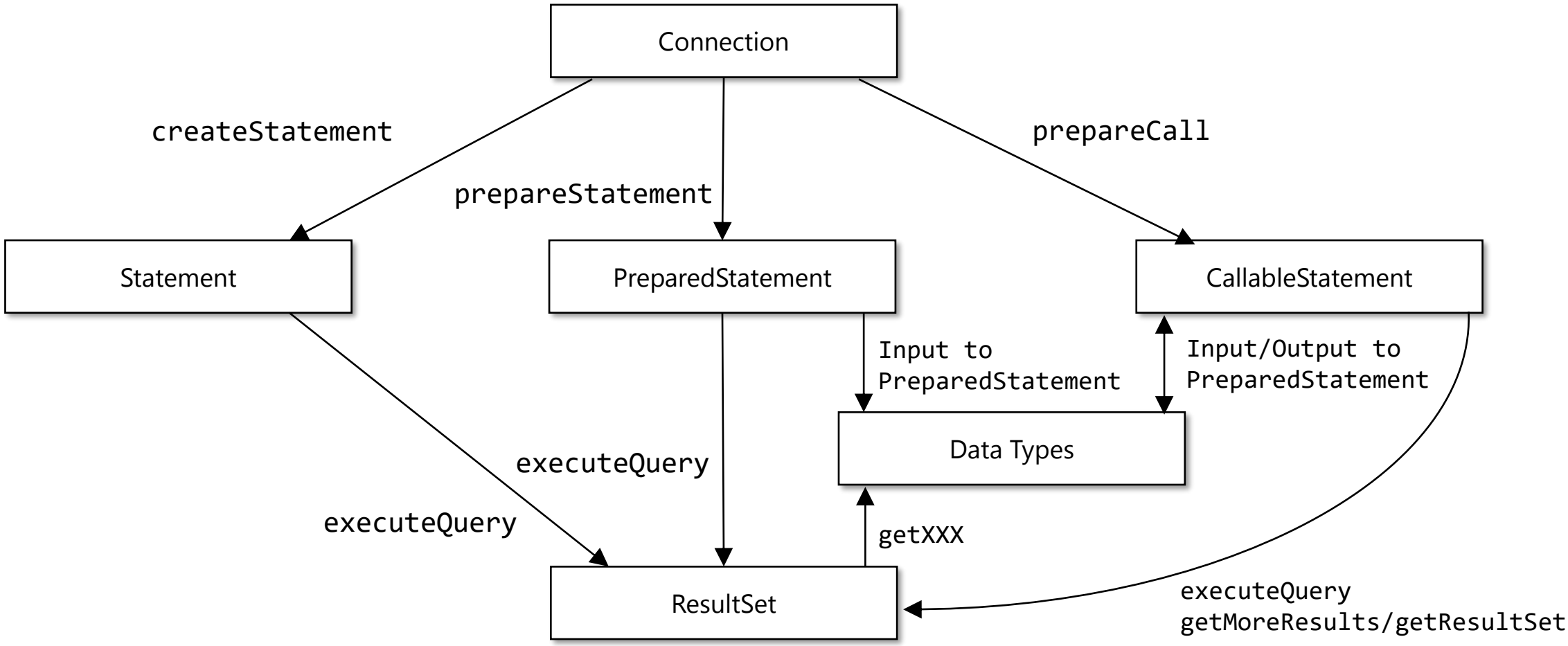
JDBC 프로그래밍 모델

- JDBC 프로그래밍 모델
- java.sql 패키지
- 드라이버 로드
- 데이터베이스에 연결
- 데이터베이스에서 명령 수행
- 명령 결과 반환

JDBC 프로그래밍 모델

- 단순한 프로그래밍 모델
  1. 데이터베이스에 연결한다
  2. 데이터베이스에 명령을 내린다
     I. 명령 수행으로 반환되는 결과가 있으면
     II. 사용한다
  3. 연결을 닫는다
- 모든 형태의 데이터 소스에 대해 "완벽히 동일한" 프로그래밍 모델 적용

java.sql 패키지


Connection
createStatement
prepareStatement
prepareCall
Statement
PreparedStatement
CallableStatement
Input to
PreparedStatement
Input/Output to
PreparedStatement
Data Types
executeQuery
executeQuery
getXXX
ResultSet
executeQuery
getMoreResults/getResultSet

Java 프로그래밍 모델

- 데이터베이스 연결과 연결 해제
  - 드라이버를 로드한다 (드라이버 로드는 1회 만)
  1. Connection 객체를 통해 연결한다.
  2. Statement/PreparedStatement/CallableStatement 객체를 통해 명령을 내린다
     1. 명령 수행으로 반환되는 결과가 있으면 ResultSet 객체로 반환 받는다
     2. ResultSet 객체를 사용하여 반환 받은 결과를 이용한다
  3. 모든 연결을 닫는다
- 데이터 소스의 형태와 관계없이 동일한 방식으로 프로그래밍

드라이버 로드

- JDBC 드라이버는 jar 형태로 데이터베이스 벤더에서 제공
- Class.forName 메소드를 사용하여 드라이버 로드
  - Java의 Reflection 기능을 사용하여 Java Virtual Machine으로 JDBC 드라이버를 로드
  - DriverManager를 사용할 수 있도록 처리
  - ClassNotFoundException에 대한 처리 필수

```
String driverName = "com.mysql.cj.jdbc.Driver";

try {
    Class.forName(driverName);
}
catch (Exception e) {
    e.printStackTrace();
}
```

데이터베이스에 연결

- Connection 객체를 통해 연결
- DriverManager 객체를 사용해 JDBC 드라이버로 부터 연결을 획득

```
try  {
    Connection connection = DriverManager.getConnection(databaseURL, userName, password);
    Statement state = connection.createStatement();
    result = state.executeQuery(sqlQuery);
}
catch (SQLException e) {
    e.printStackTrace();
}
```

데이터베이스에서 명령 수행

- Statement 객체를 통해 명령 수행
  - **executeQuery()** 메소드
    - SELECT 쿼리 수행
    - ResultSet 객체 반환
  - executeUpdate() 메소드
    - INSERT / UPDATE / DELETE 쿼리 수행
    - 성공 여부나 영향 받은 투플의 수를 int 타입으로 반환

```
try  {
    Connection connection = DriverManager.getConnection(databaseURL, userName, password);
    Statement state = connection.createStatement();
    result = state.executeQuery(sqlQuery);
}
catch (SQLException e) {
    e.printStackTrace();
}
```

명령 수행 결과 반환

- ResultSet 객체를 사용하여 SQL 질의의 결과를 반환 받음
- 현재 데이터의 투플을 가리키는 커서(Cursor)를 가리키는 구조
- 커서의 위치와 관련된 메소드와 레코드를 가져오는 메소드 제공

```
ResultSet result = SQLHelper.executeResult(“...");
try {
    for(;result.next();) {
        System.out.print(result.getString(1) + " ");
        System.out.print(result.getString(2) + " ");
        System.out.println(result.getString(3));
    }
    result.close();
}
catch (SQLException e) {
    e.printStackTrace();
}
```

연결 닫기

- 데이터베이스 작업이 완료되면, 연결을 닫아야 함
- Connection을 **`Close()`** 하지 않으면 메모리 누수 발생
- Close()된 Connection에서 메소드를 호출하면 SQLException 발생

```
try {
    if (connection != null && !connection.isClosed()) {connection.close();}
    if (state != null && !state.isClosed()) {state.close();}
    if (result != null && !result.isClosed()) {result.close();}
}
catch(SQLException e) {
    e.printStackTrace();
}
```

JDBC 주요 객체

- 드라이버 관리 – DriverManager 클래스
- 연결 - Connection 인터페이스
- 명령 – Statement 인터페이스
- 명령 – PreparedStatement 인터페이스
- 명령 – CallableStatement 인터페이스
- 결과 집합 – ResultSet 인터페이스

DriverManager 클래스

- JDBC 드라이버 세트를 관리하는 정적 클래스
- 드라이버를 로딩하고, 데이터베이스 연결을 관리
- 사용할 드라이버를 등록하여 연결 등을 반환

```
try {
        Class.forName(mysqlDriver);
        Class.forName(mssqlDriver);

        mysqlConnection = DriverManager.getConnection(mysqlURL, "root", “pwd");
        mssqlConnection = DriverManager.getConnection(mssqlURL, "sa", “pwd");

        ...
}
catch (Exception e) {
        e.printStackTrace();
}
```

Connection 인터페이스

- 특정 데이터 원본과의 연결을 나타냄
- 데이터베이스 질의를 정의하고 실행할 수 있는 Statement 객체를 생성
- https://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html

```
try {
	Class.forName(mssqlDriver);
	Connection mysqlConnection = DriverManager.getConnection(mysqlURL, "root", “pwd");

	...
}
catch (Exception e) {
	e.printStackTrace();
}
```

Statement 인터페이스

- Connection 객체를 통해 반환되는 객체에 의해 구현되는 메소드 집합 정의
- Connection 인터페이스의 **`CreateStatement()`** 메소드로 반환

```
Statement statement = myConnection.createStatement();
```

- Connection 객체로 연결된 데이터베이스에서 질의를 수행
  - executeQuery 메소드
    명령을 수행하고 결과로 ResultSet을 반환 받음
  - executeUpdate 메소드
    명령을 수행하고 결과로 영향 받은 행에 숫자를 int로 반환 받음

Statement 인터페이스

```
Statement statement = myConnection.createStatement();

String sqlQuery = "SELECT PassengerName, Grade, Age FROM Passenger";
ResultSet result = statement.executeQuery(sqlQuery);

for(;result.next();) {
    System.out.print(result.getString(1) + " ");
    System.out.print(result.getString(2) + " ");
    System.out.println(result.getString(3));
}

sqlQuery = "UPDATE Passenger SET Age = 30 WHERE PassengerNo = 3";
int effectedRow = statement.executeUpdate(sqlQuery);

System.out.println(effectedRow + " row(s) were updated");
```

PreparedStatement 인터페이스

- Statement 인터페이스와 동일한 역할 수행
- SQL 문장이 미리 컴파일 됨
  - 동일한 질의를 특정 값만 바꾸어 여러 번 실행할 때 유리함
  - 데이터베이스 서버에서 질의가 한번 분석되면 재 분석할 필요없이 재 사용
- 파라미터 인자 사용
  - 각각의 파라미터 인수에 대해 Placeholder를 사용하여 SQL 질의 정의
  - PlaceHolder는 ? 로 표시, SQL문이 실행되기 전에 실제 값으로 대체 됨
- 보안상 이점
  - SQL Injection 공격 방지

PreparedStatement 인터페이스

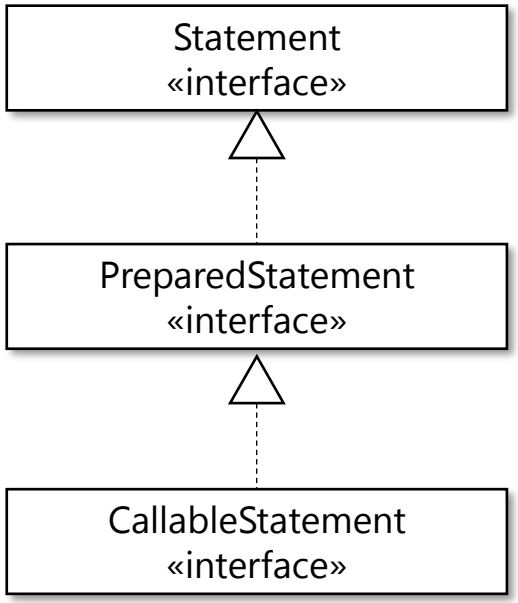
```
try {
    myConnection = DriverManager.getConnection(databaseURL, "root", "kogpsd");
    String sqlQuery = "SELECT PassengerName, Grade, Age FROM Passenger
                                                    WHERE PassengerNo = ?";
    state = myConnection.prepareStatement(sqlQuery);
    state.setString(1, "2");

    result = state.executeQuery();

    for(;result.next();) {
        System.out.print(result.getString(1) + " ");
        System.out.print(result.getString(2) + " ");
        System.out.println(result.getString(3));
    }
}
catch (SQLException e) {
    e.printStackTrace();
}
```

CallableStatement 인터페이스

- 저장 프로시저를 호출하기 위해 사용
- 저장 프로시저의 in/out/inout 파라미터를 사용할 수 있음
- 변수 처리를 위한 구문 실행
- 보다 빠른 성능의 질의 결과 반환이 가능



CallableStatement 인터페이스

```
# Stored Procedure in Database
DELIMITER $$
CREATE PROCEDURE GetPassengerCount (
    OUT count int
)
BEGIN
    SELECT COUNT(*) INTO count
    FROM Passenger;END $$
DELIMITER ;
```

CallableStatement 인터페이스

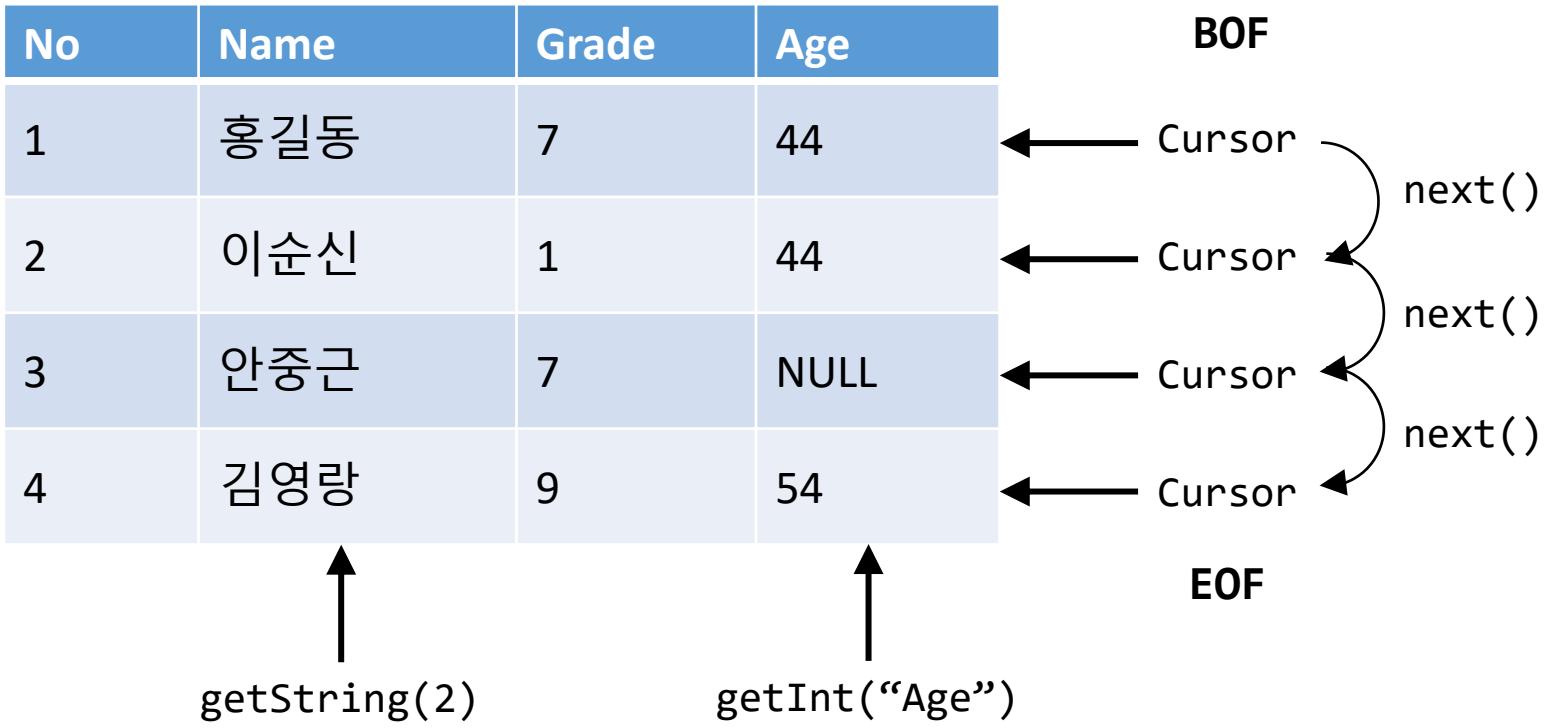
```
try {
    myConnection = DriverManager.getConnection(databaseURL, "root", "kogpsd");
    String sqlQuery = "CALL GetPassengerCount(?)";
    state = myConnection.prepareCall(sqlQuery);
    state.registerOutParameter(1, java.sql.Types.INTEGER);
    state.execute();

    int result = state.getInt(1);

    System.out.println("Passenger Count: " + result);
}
    catch (SQLException e) {
        e.printStackTrace();
}
```

ResultSet 인터페이스

No	Name	Grade	Age
1	홍길동	7	44
2	이순신	1	44
3	안중근	7	NULL
4	김영랑	9	54



ResultSet 인터페이스 주요 메소드

메소드	설명
`boolean first()`	커서를 첫 번째 행으로 이동합니다.
`boolean last()`	커서를 마지막 행으로 이동합니다.
`boolean next()`	커서를 다음 행으로 이동하고, 더 읽을 것이 없으면 false를 반환합니다.
`boolean previous()`	커서를 이전 행으로 이동하고, 더 읽을 것이 없으면 false를 반환합니다.
`boolean absolute(int row)`	커서를 지정한 행으로 이동하고, 데이터가 없으면 false를 반환합니다.
`boolean isFirst()`	현재 커서가 첫 번째 행에 있는지 여부를 반환합니다.
`boolean isLast()`	현재 커서가 마지막 행에 있는지 여부를 반환합니다.
`void close()`	데이터베이스 연결을 닫고 ResultSet 객체 리소스를 즉시 반환합니다.
`<type> get<Type>(String ColumnLabel)`	<type>은 해당 데이터 타입을 나타내며 현재 행에서 지정된 열 이름(ColumnLabel)에 해당하는 데이터를 반환합니다. 예를 들어, int getInt("Age")은 이름이 Age인 필드의 값을 정수형으로 반환합니다.
`<type> get<Type>(String ColumnIndex)`	<type>은 해당 데이터 타입을 나타내며 현재 행에서 지정된 열 번호(ColumnIndex)에 해당하는 데이터를 반환합니다. 예를 들어, int getInt(0)은 이름이 첫 번째 필드의 값을 정수형으로 반환합니다.

ResultSet을 이용한 데이터 조회

- ColumnIndex 사용

```
while(resultSet.next()) {
    System.out.println(resultSet.getLong(1));
    System.out.println(resultSet.getString(2));
    System.out.println(resultSet.getTimestamp(3));
}
```

- ColumnLabel 사용

```
while(resultSet.next()) {
    System.out.println(resultSet.getLong("id"));
    System.out.println(resultSet.getString("name"));
    System.out.println(resultSet.getTimestamp("created_at"));
}
```

O-R Mapping: Passenger Relation -> Passenger Class

Lab: 예약 정보 조회