CODING 101

WEB PROGRAMMING

02/12 14:00 PM

SQL (Structured Query Language)

- **SQL (Structured Query Language) : 데이터베이스의 데이터에 대해 작업을 수행할 때 사용하는 언어**
  - ✓ 세미콜론 (;) 사용

➢ DDL (Data Definition Language)

- ✓ CREATE TABLE : 테이블이나 인덱스, 뷰 등 데이터베이스 객체 생성
- ✓ DROP TABLE: 생성된 데이터베이스 객체 삭제
- ✓ ALTER TABLE: 이미 생성된 데이터베이스 객체 수정
- ✓ TRUNCATE : 테이블이나 클러스터의 데이터를 통째로 삭제

➢ DML (Data Manipulation Language)

- ✓ SELECT : 테이블이나 뷰에 있는 데이터 조회
- ✓ INSERT : 데이터를 신규로 생성
- ✓ UPDATE : 이미 생성된 데이터를 수정
- ✓ DELETE : 데이터를 삭제
- ✓ COMMIT : 트랜잭션 처리, 변경된 데이터를 최종 적용
- ✓ ROLLBACK : 트랜잭션 처리, 변경된 데이터를 적용하지 않고 이전으로 되돌림

➢ DCL (Data Control Language)

- ✓ GRANT : 사용자에게 특정 권한을 부여
- ✓ REVOKE : 사용자에게 부여된 권한을 회수

➢ 데이터베이스 생성

- ✓ CREATE DATABASE db이름 DEFAULT CHARACTER SET UTF8;

➢ 사용자 계정 생성 및 권한 부여

- ✓ CREATE USER '계정명' IDENTIFIED BY '암호';
- ✓ GRANT 권한 목록 ON db이름.대상 TO '계정명';

➢ 테이블 생성

- ✓ CREATE TABLE 테이블 이름 (

  컬럼명 컬럼 데이터 타입(byte),

  …

  );
- ✓ 데이터 타입 : 문자형 컬럼 - VARCHAR2, 정수형 컬럼 NUMBER, 날짜형 컬럼 DATE
- ✓ Primary Key : 테이블의 고유 튜플을 식별할 수 있게 해 주는 컬럼

  ⇒ 빠르게 레코드 검색 가능

- **INSERT**
  - ✓ 데이터 저장
  - ✓ INSERT INTO 테이블명 (컬럼1, 컬럼2, …) VALUES (값1, 값2, …);
  - ✓ 테이블명 뒤에 컬럼명 명시하지 않으면 전체 컬럼들에 대해 값 넣어 줘야 함
- **UPDATE**
  - ✓ 데이터 수정
  - ✓ UPDATE 테이블명 SET 수정할 컬럼 = 수정할 값 WHERE 조건;
- **DELETE**
  - ✓ 데이터 삭제
  - ✓ DELETE FROM 테이블명 WHERE 조건;
- **SELECT**
  - ✓ 데이터 조회
  - ✓ SELECT 컬럼1, 컬럼2, …/* FROM 테이블명 WHERE 조건;

- **WHERE**
  - ✓ 조건 붙일 때 사용
- **ORDER BY**
  - ✓ WHERE 조건절 뒤에 ORDER BY절 사용하여 데이터 정렬
  - ✓ ASC - 오름차순, DESC - 내림차
- **DROP**
  - ✓ 테이블 삭제
  - ✓ DROP TABLE 테이블명;
- **COMMIT (DML)**
  - ✓ 쿼리문들의 상태 지점을 저장
- **ROLLBACK (DML)**
  - ✓ COMMIT된 위치를 기준으로, 그 이후에 작성한 쿼리문을 실행 취소시킴;

- **transaction : 두 개 이상의 쿼리를 한 개의 쿼리처럼 처리하는 것**
  - 시작과 종료를 가짐
  - 트랜잭션 시작되면 이후로 실행되는 쿼리는 임시로 보관됨
  - 이후 COMMIT하면 임시 보관한 쿼리 결과를 실제 데이터베이스에 반영
  - COMMIT 전에 에러가 발생하면 임시로 보관한 모든 쿼리 결과를 실제 데이터에 반영하지 않고 ROLLBACK함

JDBC (Java Database Connectivity)

➢ JDBC : Java 프로그램에서 SQL문을 실행하여 데이터를 관리하기 위한 JAVA API

✓ 해당 데이터베이스의 JDBC를 사용하면, 다양한 데이터베이스에 대해 별도의 프로그램을 만들 필요 없이, 하나의 프로그램으로 데이터베이스 관리할 수 있음

➢ JDBC Driver : DBMS와의 통신을 담당하는 Java 클래스

✓ MySQL: "com.mysql.jdbc.Driver" , ORACLE: "oracle.jdbc.driver.OracleDriver"

✓ 데이터베이스와 JAVA 연결 순서

① JDBC 드라이버 로드
② 데이터베이스 Connection 객체 생성
③ 쿼리문 실행 위한 Statement 객체 생성
④ 쿼리문 실행
⑤ ResultSet 객체 통해 쿼리문 실행 결과값을 소비
⑥ Statement 객체 종료
⑦ Connection 객체 종료

➢ JDBC URL : 데이터베이스를 구분

- ✓ MySQL: "jdbc:mysql://호스트이름:포트번호/DB이름"
- ✓ ORACLE: "jdbc:oracle:thin:호스트이름:포트번호:DB이름"

➢ Connection 객체

- ✓ JDBC를 이용해서 데이터베이스를 사용하려면 데이터베이스와 연결된 커넥션을 구해야 함
- ✓ java.sql 패키지에 있는 Connection 클래스가 데이터베이스 커넥션을 지원
- ✓ DriverManager 클래스가 제공하는 getConnection() 메서드를 사용하여 커넥션을 구할 수 있음
- ✓ getConnection()
  - 파라미터 값으로 JDBC URL, DB 사용자 계정명, DB 사용자 암호를 전달
    → DB와 연결된 커넥션 객체를 리턴
  - 제대로 객체를 생성하지 못하면 SQLException이 발생
    ⇒ 반드시 try-catch 구문으로 예외 처리를 해줘야 함
- ✓ Connection 객체를 다 사용한 뒤에는 반드시 close() 메서드를 호출
  → Connection 객체가 사용한 시스템 자원을 반환

(그렇지 않으면 시스템 자원이 불필요하게 소모되어 커넥션을 구할 수 없는 상황 발생할 수도 있음)

➢ Statement 객체

- ✓ Connection 객체를 생성 후, Connection 객체로부터 Statement를 생성하고 쿼리문을 실행
- ✓ Statement 객체는 Connection 객체의 createStatement()를 이용하여 생성
- ✓ 쿼리문을 실행시킴
  - executeQuery(String query):ResultSet - Select 쿼리문을 실행합니다.
  - executeUpdate(String query):int - Insert, Update, Delete 쿼리문을 실행

✓ ResultSet 객체

- ✓ Statement 객체의 executeQuery()는 Select 쿼리문의 결과를 ResultSet 객체에 담아서 리턴
  ⇒ 데이터 조회의 결과값을 ResultSet이 제공하는 메소드를 통해 읽어올 수 있음
  - getString(String name):String - 지정한 컬럼 값을 String으로 읽어 옴
    - 파라미터 변수 name에는 DB 테이블의 컬럼 이름을 적음
  - getInt(String name):int - 지정한 컬럼 값을 int 타입으로 읽어 옴
  - getDouble(String name):double - 지정한 컬럼 값을 double 타입으로 읽어 옴

➢ PreparedStatement 객체

➢ Statement 객체와 PreparedStatement객체는 쿼리문을 실행하는 동일한 기능을 제공

✓ 값 변환을 자동으로 해주는 기능을 제공

✓ 간결한 코드를 만들 수 있음

✓ Statement 객체는 지정할 값이 많아질 경우 따옴표가 복잡하게 얽히기 때문에 코드 작성에서 오류가 발생할 수도 있고, 코드 수정 시에도 어려움이 발생

✓ PreparedStatement 객체는 값을 지정할 때 값 부분을 물음표(?)로 처리하여 간단히 값을 지정할 수 있음. 이때 첫번째 물음표의 인덱스는 1이며, 이후 물음표의 인덱스는 나오는 순서대로 인덱스 값이 1씩 증가함.

➢ JDBC에서 트랜잭션 처리

✓ **transaction : 두 개 이상의 쿼리를 한 개의 쿼리처럼 처리하는 것**

- 시작과 종료를 가짐
- 트랜잭션 시작되면 이후로 실행되는 쿼리는 임시로 보관됨
- 이후 COMMIT하면 쿼리 결과를 실제 데이터베이스에 반영

Data Access Object (DAO) 클래스

➢ **데이터베이스에 접속해서 데이터의 추가, 삭제, 수정 등의 작업을 하는 클래스**

✓ 일반적으로 JSP 혹은 Servlet에서 위의 로직을 함께 기술할 수도 있지만, 유지 보수 및 코드의 모듈화를 위해 별도의 DAO 클래스를 만들어 사용

✓ front-end 개발과 back-end 개발을 나누어 할 수 있게 함

✓ 한 개의 테이블마다 한 개의 DAO 클래스를 작성

✓ 테이블로부터 데이터를 읽어 와서 자바 객체로 변환 / 자바 객체의 값을 테이블에 저장

✓ DAO를 구현하면 테이블의 컬럼과 매핑되는 값을 갖는 자바빈 클래스를 항상 작성해야 함
⇒ VO 클래스

Value Object (VO) 클래스

➢ **데이터베이스와 관련된 변수들의 모음 클래스**

✓ DAO 클래스를 이용하여 데이터베이스에서 데이터를 관리할 때 데이터를 일반적인 변수에 할당하여 작업할 수도 있지만, 별도의 VO 클래스를 작성

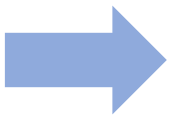
✓ VO클래스는 자바빈 클래스로 생성

Connection Pool (연결 풀)

➢ **데이터베이스 메모리 내에 있는 데이터베이스 커넥션들로 구성된 하나의 캐시**

- ✓ 데이터에 대한 요청이 발생하면 재사용됨
- ✓ 연결 풀에서 하나의 연결이 생성되어 풀에 배치되면 새로운 연결이 만들어지지 않도록 재사용하지만, 만약 모든 연결이 사용 중에 있으면 새로운 연결이 만들어져 풀에 추가됨
- ✓ 데이터베이스의 수행 능력을 향상시키기 위해 사용
- ✓ 데이터베이스 연결을 위해 기다리는 시간을 축소
- ✓ 주로 DAO 클래스 파일에 구현

```java
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

import org.apache.tomcat.jdbc.pool.DataSource;
import org.apache.tomcat.jdbc.pool.PoolProperties;

public class SimplePOJOExample {

    public static void main(String[] args) throws Exception {
        PoolProperties p = new PoolProperties();
        p.setUrl("jdbc:mysql://localhost:3306/mysql");
        p.setDriverClassName("com.mysql.jdbc.Driver");
        p.setUsername("root");
        p.setPassword("password");
        p.setJmxEnabled(true);
        p.setTestWhileIdle(false);
        p.setTestOnBorrow(true);
        p.setValidationQuery("SELECT 1");
        p.setTestOnReturn(false);
        p.setValidationInterval(30000);
        p.setTimeBetweenEvictionRunsMillis(30000);
        p.setMaxActive(100);
        p.setInitialSize(10);
        p.setMaxWait(10000);
        p.setRemoveAbandonedTimeout(60);
        p.setMinEvictableIdleTimeMillis(30000);
        p.setMinIdle(10);
        p.setLogAbandoned(true);
        p.setRemoveAbandoned(true);
        p.setJdbcInterceptors(
          "org.apache.tomcat.jdbc.pool.interceptor.ConnectionState;"+
          "org.apache.tomcat.jdbc.pool.interceptor.StatementFinalizer");
        DataSource datasource = new DataSource();
        datasource.setPoolProperties(p);

        Connection con = null;
        try {
          con = datasource.getConnection();
          Statement st = con.createStatement();
          ResultSet rs = st.executeQuery("select * from user");
          int cnt = 1;
          while (rs.next()) {
              System.out.println((cnt++)+". Host:" +rs.getString("Host")+
                " User:"+rs.getString("User")+" Password:"+rs.getString("Password"));
          }
          rs.close();
          st.close();
        } finally {
          if (con!=null) try {con.close();}catch (Exception ignore) {}
        }
    }

}
```



```xml
<Resource name="jdbc/TestDB"
          auth="Container"
          type="javax.sql.DataSource"
          factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
          testWhileIdle="true"
          testOnBorrow="true"
          testOnReturn="false"
          validationQuery="SELECT 1"
          validationInterval="30000"
          timeBetweenEvictionRunsMillis="30000"
          maxActive="100"
          minIdle="10"
          maxWait="10000"
          initialSize="10"
          removeAbandonedTimeout="60"
          removeAbandoned="true"
          logAbandoned="true"
          minEvictableIdleTimeMillis="30000"
          jmxEnabled="true"
          jdbcInterceptors="org.apache.tomcat.jdbc.pool.interceptor.ConnectionState;
            org.apache.tomcat.jdbc.pool.interceptor.StatementFinalizer"
          username="root"
          password="password"
          driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/mysql"/>
```

실습