JPA

JPA(Java Persistence API)

자바 애플리케이션에서 관계형 데이터베이스를 쉽게 사용할 수 있도록 도와주는 표준 API

자바 객체를 데이터베이스 테이블에 매핑하고, 객체의 상태를 데이터베이스에 반영하거나 데이터베이스에서 객체를 가져오는 작업을 처리한다.

JPA 탄생 배경

1. 객체-관계 불일치 문제 : 객체 지향 프로그래밍과 데이터베이스의 패러다임이 다르므로, 객체와 테이블 사이의 매핑의 필요성이 생겼다.
2. 표준화의 필요성 : 특정 프레임워크에 종속되거나, 다른 프레임워크로의 전환이 어렵다.
3. EJB의 복잡성 : EJB의 복잡한 설정과 사용 방법 때문에 효율적인 개발이 어렵다.

JPA 장점

객체 지향 프로그래밍과 데이터베이스 사이의 불일치를 해결한다.

JPA 기능

1. 객체와 관계형 데이터베이스의 매핑
2. CRUD 작업 지원
3. 쿼리 언어 지원
4. 트랜잭션 관리

JDBC vs JPA

JDBC

1. 데이터베이스와 직접적인 상호작용을 하여, 개발자가 직접 SQL 쿼리를 작성하고 결과를 수동으로 처리해야 한다.
2. 저수준 API로, 데이터베이스와의 연결 및 SQL 실행, 결과 처리 등을 직접 관리해야 한다.
3. connection, statement, resultSet 등을 명시적으로 열고 닫아야 한다.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:h2:mem:testdb";
        String user = "sa";
        String password = "";

        try (Connection conn = DriverManager.getConnection(ur
            // Create table
            String createTableSQL = "CREATE TABLE users (id I
            try (PreparedStatement stmt = conn.prepareStateme
                stmt.executeUpdate();
            }

            // Insert data
            String insertSQL = "INSERT INTO users (name, emai
            try (PreparedStatement stmt = conn.prepareStateme
                stmt.setString(1, "우리");
                stmt.setString(2, "woorifisa@example.com");
                stmt.executeUpdate();
            }

            // Query data
            String selectSQL = "SELECT id, name, email FROM u
            try (PreparedStatement stmt = conn.prepareStateme
                 ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    long id = rs.getLong("id");
                    String name = rs.getString("name");
                    String email = rs.getString("email");
                    System.out.println("User ID: " + id + ", 
                }
            }
```

```
            // Delete data
            String deleteSQL = "DELETE FROM users WHERE name =
            try (PreparedStatement stmt = conn.prepareStateme
                stmt.setString(1, "우리");
                int rowsAffected = stmt.executeUpdate();
                System.out.println("Rows deleted: " + rowsAff
            }

            // Verify deletion
            try (PreparedStatement stmt = conn.prepareStateme
                 ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    long id = rs.getLong("id");
                    String name = rs.getString("name");
                    String email = rs.getString("email");
                    System.out.println("Remaining User ID: " +
                }
            }

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

JPA

1. 데이터베이스의 테이블과 자바 객체 간의 매핑을 자동으로 처리하여, 개발자가 직접 SQL 쿼리를 작성하지 않고 객체를 통해 데이터베이스 작업을 수행한다.
2. 고수준 API로, 엔티티 객체와 매핑된 데이터베이스 테이블 간의 상호작용을 간편하게 관리한다.
3. 데이터베이스 연결 및 트랜잭션 관리를 자동으로 처리한다.

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
```

```
public class JpaExample {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityMa
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();

        try {
            tx.begin();

            // Create and save entity
            User user = new User("우리", "woorifisa@example.co
            em.persist(user);

            // select entity
            User selectUser = em.find(User.class, user.getId(
            System.out.println("User: " + selectUser.getName(

                        // delete entity
            em.remove(foundUser);
            System.out.println("User deleted");

            // Verify deletion
            User deletedUser = em.find(User.class, user.getId
            if (deletedUser == null) {
                System.out.println("User not found after dele
            } else {
                System.out.println("User still exists: " + de
            }

            tx.commit();
        } catch (Exception e) {
            if (tx.isActive()) {
                tx.rollback();
            }
            e.printStackTrace();
        } finally {
            em.close();
```

```
            emf.close();
        }
    }
}
```