

JPA-Oracle

안 화 수

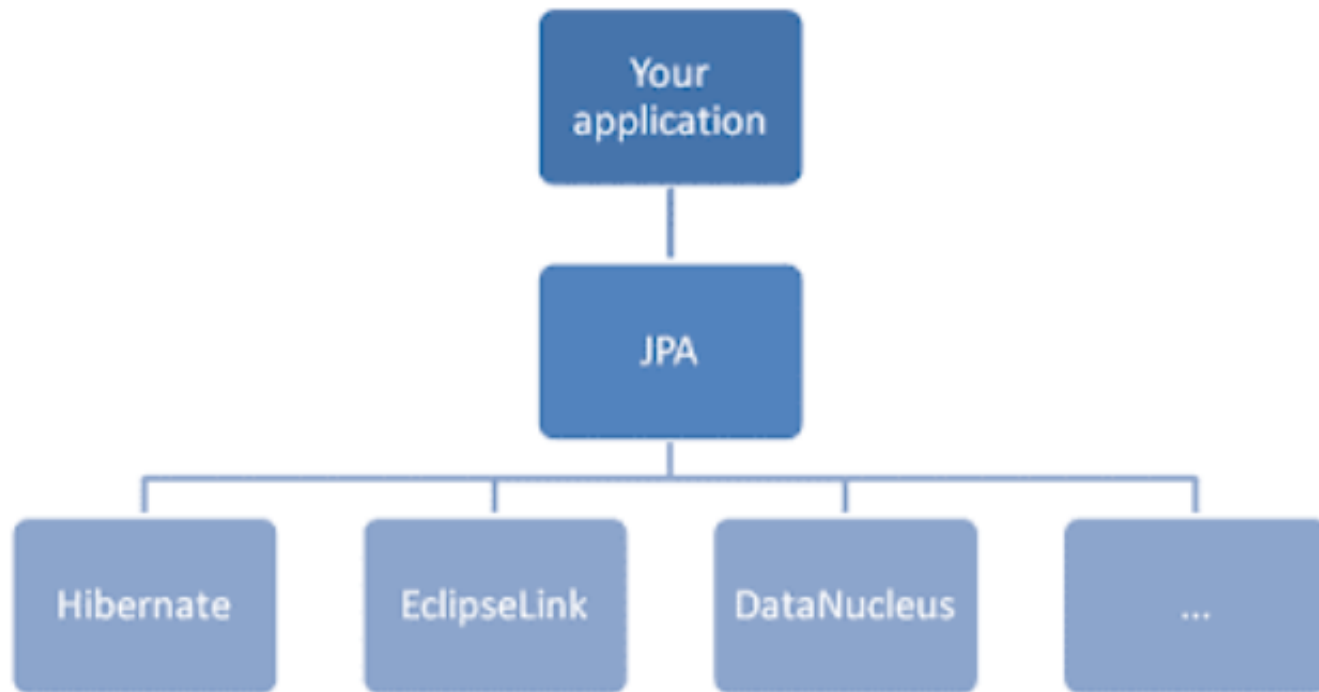
❖ JPA(Java Persistence API)

1. JPA는 java ORM에 대한 API 표준 명세이며 인터페이스의 모음
 - 구현된 클래스와 매핑을 해주기 위해 사용되는 프레임워크이다.
 - ORM (Object Relational Mapping) 프레임워크
Entity Class와 RDB(Relational DataBase)의 테이블을 매핑한다는 의미
객체를 RDB 테이블에 자동으로 영속화를 해주는 것
2. 구현체가 따로 없으며 ORM 프레임워크를 선택하여 사용
3. 구현체로는 Hibernate, EclipseLink, Data Nucleus가 있으며 Hibernate가 가장 대중적이다.

JPA

❖ JPA 구현체

- JPA 구현체 중에는 Hibernate, EclipseLink, DataNucleus 등이 있다.



JPA 장단점

❖ JPA 장점

1. 개발이 편리하다.

웹 애플리케이션에서 반복적으로 작성하는 기본적인 CRUD(Create, Read, Update, Delete)용 SQL을 직접 작성하지 않아도 된다.

2. 데이터베이스에 독립적인 개발이 가능하다.

JPA는 특정 데이터베이스에 종속적이지 않기 때문에 데이터베이스와 관계없이 개발할 수 있다. 데이터베이스가 변경 되더라도 JPA가 해당 데이터베이스에 맞는 쿼리를 생성해 준다.

3. 유지보수 및 리팩토링이 용이하다.

MyBatis와 같은 매퍼 프레임워크를 사용해서 데이터베이스 중심의 개발을 하면 테이블이 변경될 경우에 관련된 코드를 모두 변경해야 한다. 그렇지만 JPA를 사용하면 JPA의 Entity만 수정하면 된다.

JPA 장단점

❖ JPA 단점

1. 학습곡선(learning curve)이 크다.

기존의 데이터베이스 위주의 개발방식에 비해서 배워야 할 것들이 많다. 또한 SQL을 직접적으로 작성하지 않기 때문에 튜닝 등을 할 때도 어려움을 겪을 수 있다.

2. 특정 데이터베이스의 기능을 사용할 수 없다.

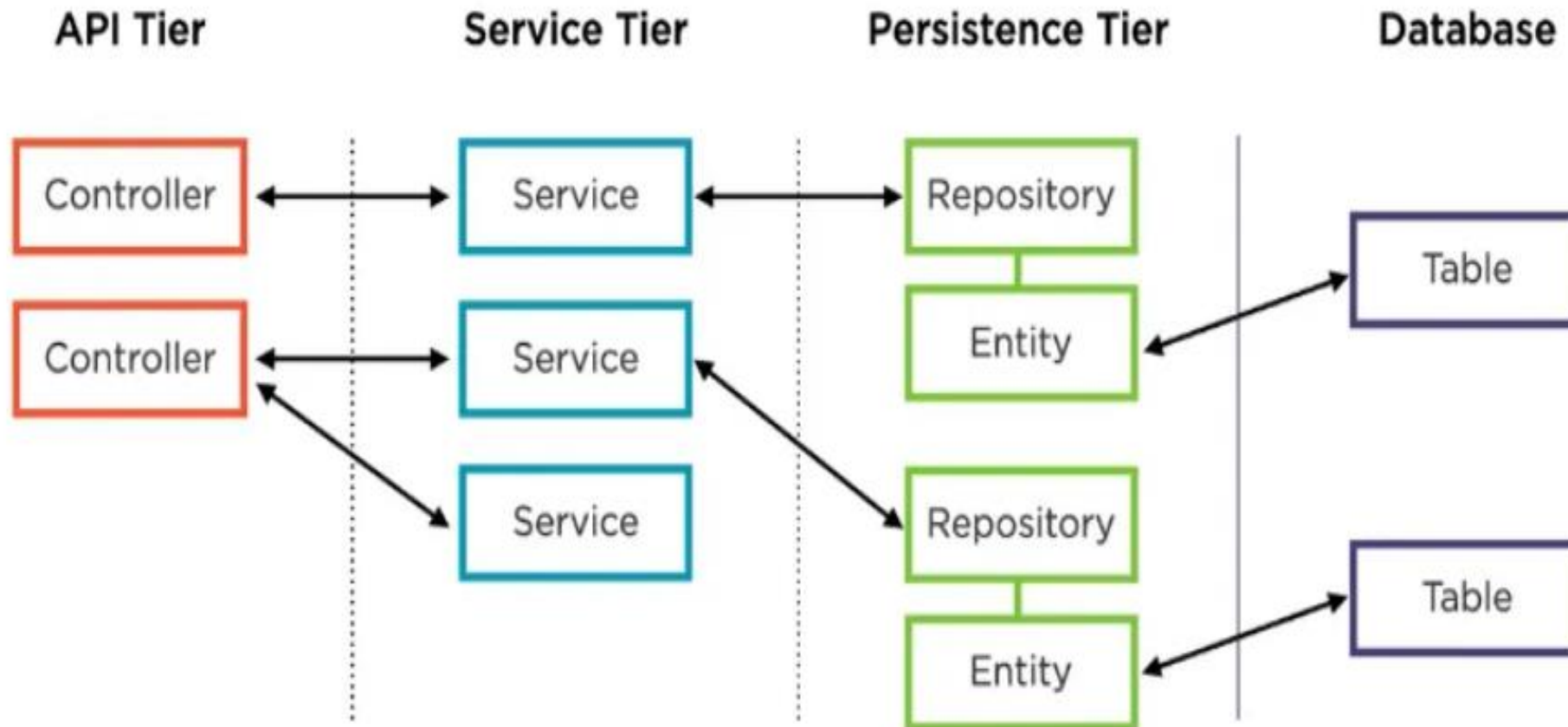
JPA를 사용할 때 특정 데이터베이스 종속적인 기능은 사용하지 않는 것이 좋다. 특히 오라클은 강력한 함수를 많이 제공하는데, 이와 같은 데이터베이스에 종속적인 기능을 사용할 경우 데이터베이스에 독립적인 개발이 불가능 해진다. 이 경우 데이터베이스에 독립적인 개발이 가능한 JPA의 장점을 잃게 된다.

3. 객체지향 설계가 어려울 수 있다.

객체 위주의 설계보다 데이터베이스의 테이블을 설계하고 그에 맞춰서 객체 및 비즈니스 로직이 설계, 개발 되기 때문에 객체지향적인 설계가 어려울 수 있다.

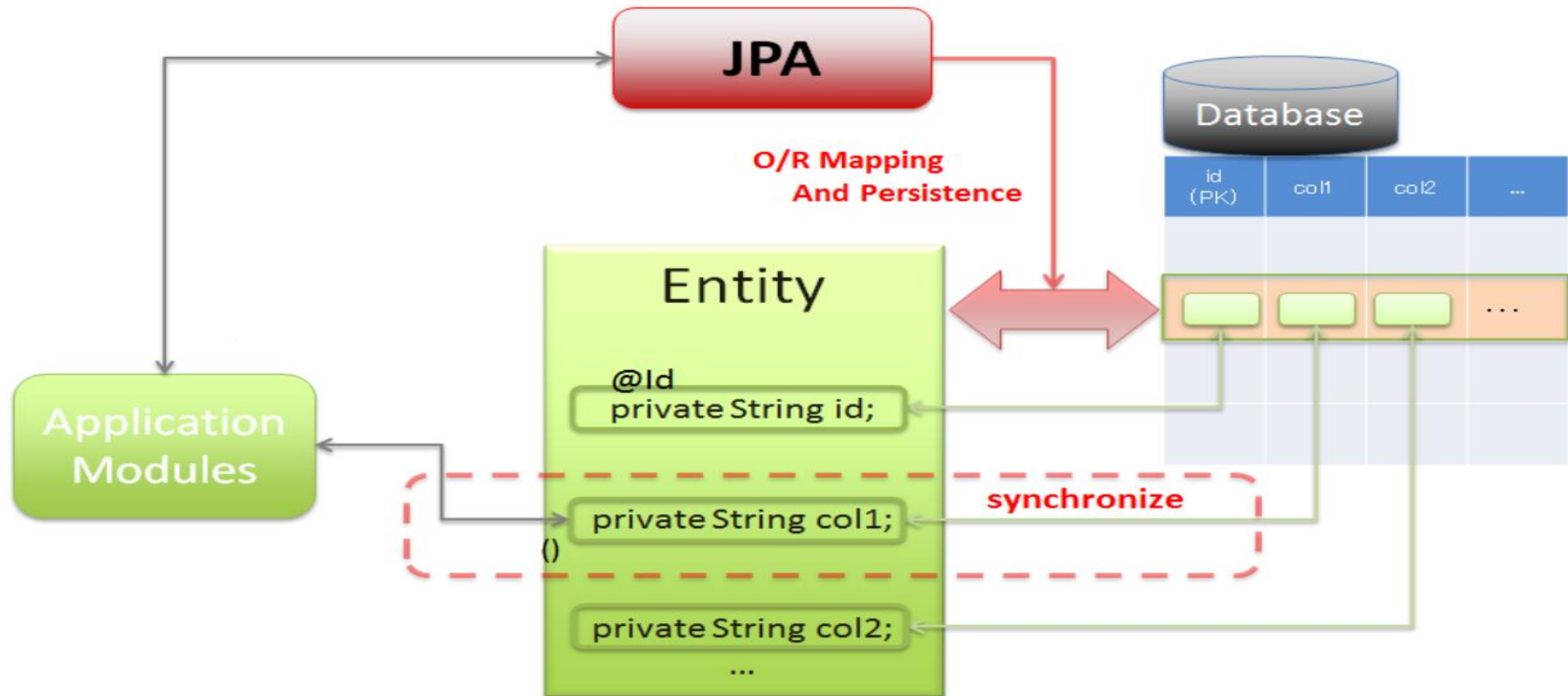
JPA Process

❖ JPA Process



JPA - ORM

❖ JPA – ORM(Object Relational Mapping)



JPA Project 생성

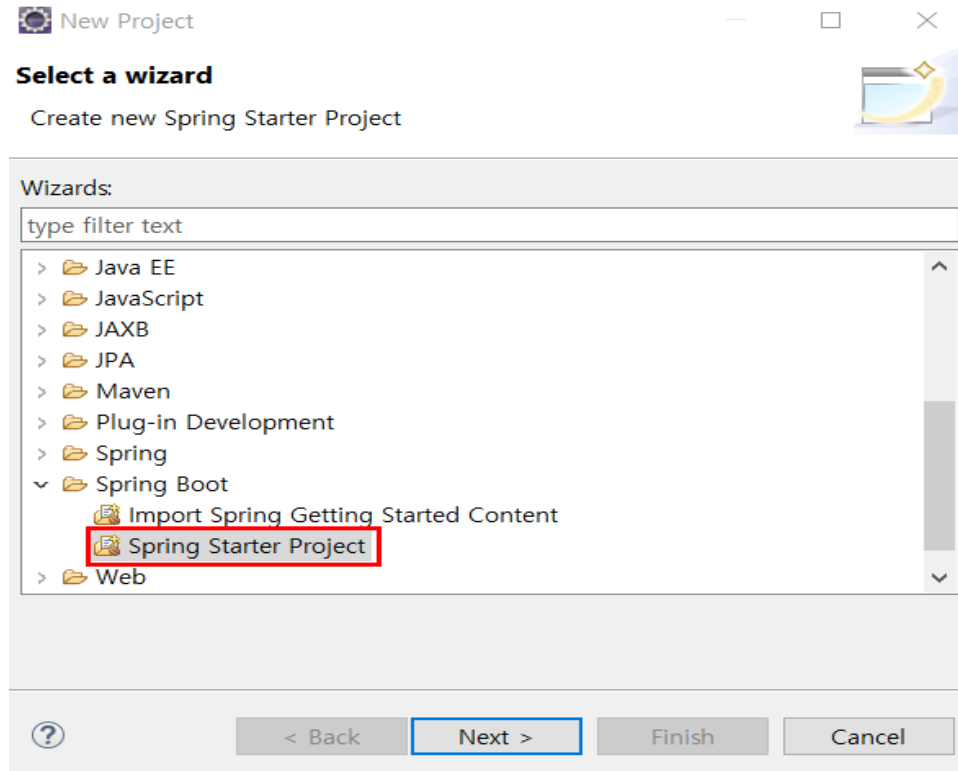
❖ JPA Oracle 연동 프로젝트 : 회원관리

jpaoracle01 프로젝트 생성

JPA Project 생성

❖ jpaoracle01 project 생성

[File] – New - Project



JPA Project 생성

❖ jpaoracle01 project 생성

- Name : **jpaoracle01**
- Type : **Maven**, Gradle
- Packaging : **War**, Jar

New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

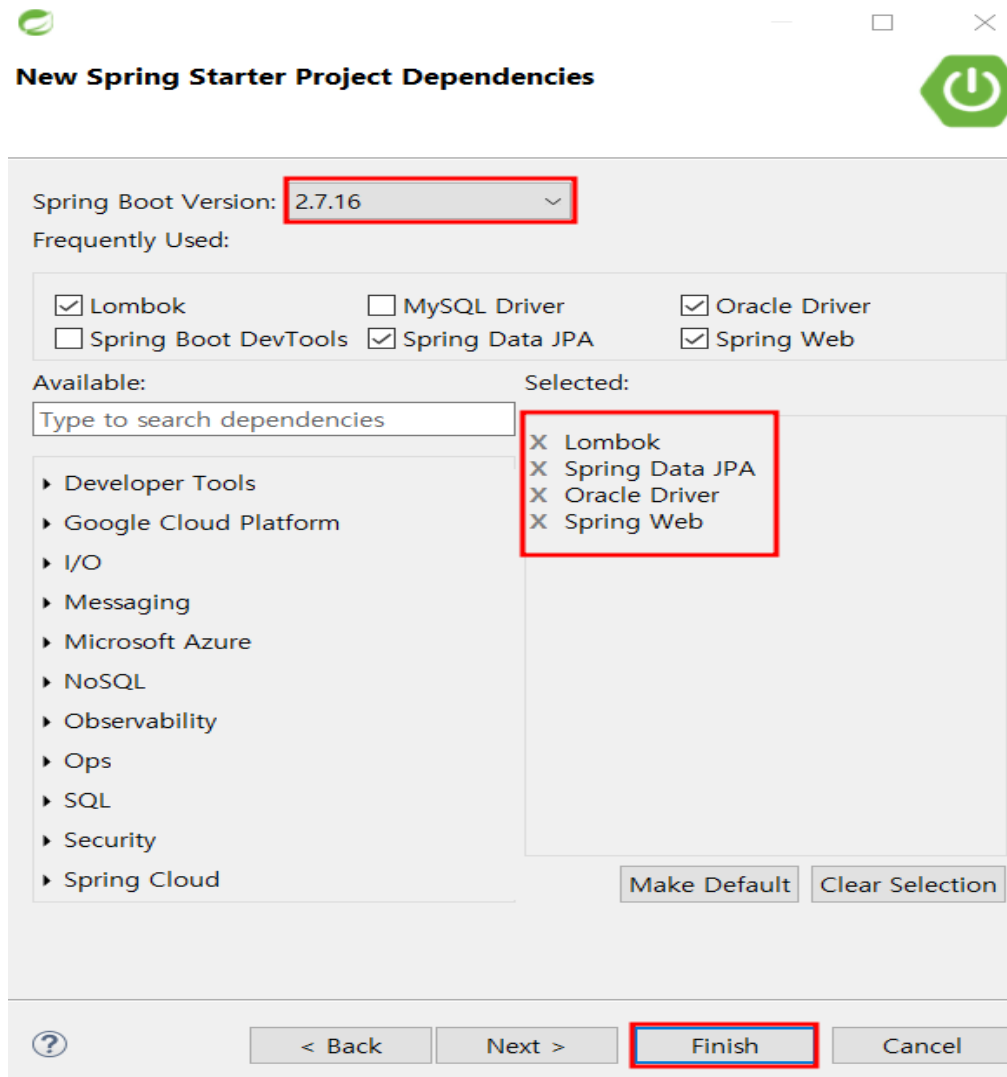
☐ Add project to working sets

Working sets:

JPA Project 생성

❖ jpaoracle01 project 생성

Spring Web
Spring Data JPA
Oracle Driver
Lombok



The image shows a 'New Spring Starter Project Dependencies' dialog box. At the top, the title bar includes a green icon, a minimize button, a maximize button, and a close button. Below the title bar is a green power button icon. The main content area has a 'Spring Boot Version:' dropdown menu set to '2.7.16', which is highlighted with a red rectangle. Below this is a 'Frequently Used:' section with a grid of checkboxes: 'Lombok' (checked), 'MySQL Driver' (unchecked), 'Oracle Driver' (checked), 'Spring Boot DevTools' (unchecked), 'Spring Data JPA' (checked), and 'Spring Web' (checked). Below the checkboxes are two columns: 'Available:' and 'Selected:'. The 'Available:' column has a search bar labeled 'Type to search dependencies' and a list of categories: 'Developer Tools', 'Google Cloud Platform', 'I/O', 'Messaging', 'Microsoft Azure', 'NoSQL', 'Observability', 'Ops', 'SQL', 'Security', and 'Spring Cloud'. The 'Selected:' column has a red rectangle around it containing four items: 'X Lombok', 'X Spring Data JPA', 'X Oracle Driver', and 'X Spring Web'. At the bottom right of the 'Available:' section are 'Make Default' and 'Clear Selection' buttons. At the very bottom of the dialog are four buttons: a help button (question mark icon), '< Back', 'Next >', and 'Finish' (highlighted with a red rectangle), followed by a 'Cancel' button.

New Spring Starter Project Dependencies

Spring Boot Version: 2.7.16

Frequently Used:

<input checked="" type="checkbox"/> Lombok	<input type="checkbox"/> MySQL Driver	<input checked="" type="checkbox"/> Oracle Driver
<input type="checkbox"/> Spring Boot DevTools	<input checked="" type="checkbox"/> Spring Data JPA	<input checked="" type="checkbox"/> Spring Web

Available:

Type to search dependencies

- ▶ Developer Tools
- ▶ Google Cloud Platform
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud

Selected:

- X Lombok
- X Spring Data JPA
- X Oracle Driver
- X Spring Web

Make Default Clear Selection

< Back Next > Finish Cancel

pom.xml

❖ pom.xml (1/2)

<dependencies>

```
<!-- jstl -->
```

```
<dependency>
```

```
    <groupId>javax.servlet</groupId>
```

```
    <artifactId>jstl</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>com.oracle.database.jdbc</groupId>
```

```
    <artifactId>ojdbc8</artifactId>
```

```
    <scope>runtime</scope>
```

```
</dependency>
```

pom.xml

❖ pom.xml (2/2)

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>
```

DataBase 연결 설정

❖ application.properties (1/2)

port

server.port=80

view resolver

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

oracle

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521/x

spring.datasource.username=spring

spring.datasource.password=spring123

DataBase 연결 설정

❖ application.properties (2/2)

JPA Setting

spring.jpa.hibernate.ddl-auto=update

spring.jpa.generate-ddl=false

spring.jpa.show-sql=true

spring.jpa.database=oracle

spring.jpa.database-platform=org.hibernate.dialect.OracleDialect

Logging Setting

logging.level.org.hibernate=info

Oracle + hikari CP

❖ application.properties

oracle

```
spring.datasource.hikari.driver-class-name=oracle.jdbc.OracleDriver  
spring.datasource.hikari.jdbc-url=jdbc:oracle:thin:@localhost:1521:xe  
spring.datasource.hikari.username=scott  
spring.datasource.hikari.password=tiger
```

application.properties 파일에 Oracle + Hikari CP(Connection Pool)를 이용해서 DataBase 연동을 할 경우에는 DatabaseConfiguration.java 파일에 추가적인 설정을 해 주어야 된다.

Oracle + hikari CP

❖ application.properties (1/2)

port

server.port=80

view resolver

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

oracle

spring.datasource.hikari.driver-class-name=oracle.jdbc.OracleDriver

spring.datasource.hikari.jdbc-url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.hikari.username=scott

spring.datasource.hikari.password=tiger

Oracle + hikari CP

❖ application.properties (2/2)

JPA Setting

spring.jpa.hibernate.ddl-auto=update

spring.jpa.generate-ddl=false

spring.jpa.show-sql=true

spring.jpa.database=oracle

spring.jpa.database-platform=org.hibernate.dialect.OracleDialect

Logging Setting

logging.level.org.hibernate=info

Oracle + hikari CP

❖ `main/java/com.example.demo/configuration-DatabaseConfiguration.java`

`@Configuration`

`@PropertySource("classpath:/application.properties")`

`public class DatabaseConfiguration {`

`@Bean`

`@ConfigurationProperties(prefix = "spring.datasource.hikari")`

`public HikariConfig hikariConfig() {`

`return new HikariConfig();`

`}`

`@Bean`

`public DataSource dataSource() {`

`DataSource dataSource = new HikariDataSource(hikariConfig());`

`return dataSource;`

`}`

`}`

Entity 클래스

❖ Entity 클래스의 어노테이션

@NoArgsConstructor	: 파라미터가 없는 기본 생성자를 만들어준다.
@AllArgsConstructor	: 모든 필드를 파라미터로 가진 생성자만 만들어준다.
@Setter	: setter 메소드 생성
@Getter	: getter 메소드 생성
@Data	: getter, setter 메소드 생성
@Entity	: 해당 클래스를 Entity 클래스로 설정
@Table(name="members")	: members 테이블 생성
@Id	: 기본키 설정

@GeneratedValue(strategy = GenerationType.AUTO)	: JPA구현체(Hibernate)가 자동으로 생성 방식을 결정 hibernate_sequence가 자동으로 생성된다.
@Column(length = 500, nullable = false)	: 크기 500Byte, not null제약조건

Entity 클래스

❖ **main/java/com.example.demo/domain-Member.java (1/2)**

```
package com.example.demo.domain;
```

```
import java.sql.Timestamp;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.Table;
```

```
import org.hibernate.annotations.CreationTimestamp;
```

```
import org.hibernate.annotations.UpdateTimestamp;
```

```
import lombok.Data;
```

Entity 클래스

❖ main/java/com.example.demo/domain-Member.java (2/2)

```
@Data
@Entity
@Table(name = "members")
public class Member {
    @Id
    private String id;
    private String passwd;
    private String name;
    private String email;

    @CreationTimestamp
    private Timestamp regdate;

    @UpdateTimestamp
    private Timestamp updatedate;
}
```

// members 테이블 생성

// 기본키 설정

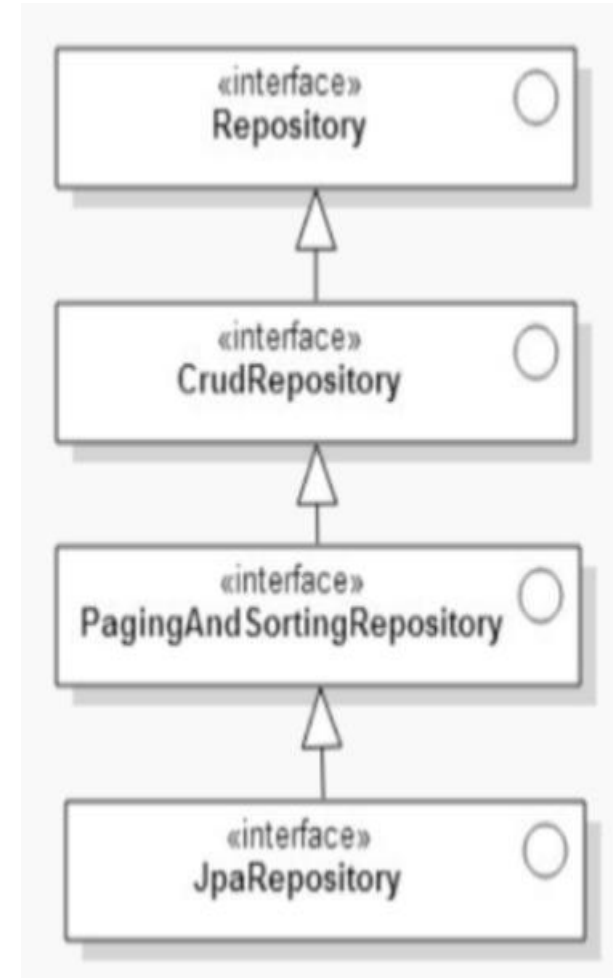
// 회원가입 시간

// 회원수정 시간

Repository

❖ Repository 구조

- **Repository** : 최상위 Repository로서 아무런 기능이 없기 때문에 잘 사용하지 않는다.
- **CrudRepository** : 기본적인 CRUD 기능을 제공하는 Repository이다.
- **PagingAndSortingRepository** : CrudRepository 인터페이스 기능에 페이징 및 정렬 기능이 추가된 인터페이스 이다.
- **JpaRepository** : PagingAndSortingRepository 인터페이스 기능뿐만 아니라 JPA에 특화된 기능까지 추가된 인터페이스 이다.



Repository Interface

❖ `main/java/com.example.demo/repository-MemberRepository.java`

```
package com.example.demo.repository;
```

```
import java.util.Optional;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.example.demo.domain.Member;
```

```
@Repository
```

```
public interface MemberRepository extends CrudRepository<Member, String>{
```

```
//      public Member save(Member member);           // 회원가입, 정보수정
```

```
//      public Optional<Member> findById(String id);   // 로그인, 수정폼
```

```
//      public void delete(Member member);           // 회원탈퇴
```

```
}
```


JPA Oracle 연동

❖ JPA Oracle 연동 프로젝트 : 게시판

jpaoracle02 프로젝트 생성

pom.xml

❖ pom.xml (1/2)

<dependencies>

```
<!-- jstl -->
```

```
<dependency>
```

```
    <groupId>javax.servlet</groupId>
```

```
    <artifactId>jstl</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>com.oracle.database.jdbc</groupId>
```

```
    <artifactId>ojdbc8</artifactId>
```

```
    <scope>runtime</scope>
```

```
</dependency>
```

pom.xml

❖ pom.xml (2/2)

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>
```

DataBase 연결 설정

❖ application.properties (1/2)

port

server.port=80

view resolver

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

oracle

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521/x

spring.datasource.username=spring

spring.datasource.password=spring123

DataBase 연결 설정

❖ application.properties (2/2)

JPA Setting

spring.jpa.hibernate.ddl-auto=update

spring.jpa.generate-ddl=false

spring.jpa.show-sql=true

spring.jpa.database=oracle

spring.jpa.database-platform=org.hibernate.dialect.OracleDialect

Logging Setting

logging.level.org.hibernate=info

Entity 클래스

❖ Entity 클래스의 어노테이션

@NoArgsConstructor	: 파라미터가 없는 기본 생성자를 만들어준다.
@AllArgsConstructor	: 모든 필드를 파라미터로 가진 생성자만 만들어준다.
@Setter	: setter 메소드 생성
@Getter	: getter 메소드 생성
@Data	: getter, setter 메소드 생성
@Entity	: 해당 클래스를 Entity 클래스로 설정
@Table(name="boards")	: boards 테이블 생성
@Id	: 기본키 설정

@GeneratedValue(strategy = GenerationType.AUTO)	: JPA구현체(Hibernate)가 자동으로 생성 방식을 결정 hibernate_sequence가 자동으로 생성된다.
@Column(length = 500, nullable = false)	: 크기 500Byte, not null제약조건

Entity 클래스

❖ 수동으로 시퀀스 설정

```
@Data
@Entity
@Table(name="boards")
@SequenceGenerator(name="boards_seq_gen",
    sequenceName="boards_seq",
    initialValue=1,
    allocationSize=1 )
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
        generator="boards_seq_gen" )
    private int no;
    private String writer;
    private String passwd;
    private String subject;
    private String content;
    @CreationTimestamp
    private Timestamp regdate;
    @UpdateTimestamp
    private Timestamp updatedate;
}
```

// boards 테이블 생성
// 시퀀스 제너레이터 이름
// 시퀀스 이름
// 시작값
// 증가값

// 기본키 설정
// 사용할 전략을 시퀀스로 선택
// 생성기를 boards_seq_gen으로 설정

Entity 클래스

❖ **main/java/com.example.demo/domain-Board.java (1/2)**

```
package com.example.demo.domain;
```

```
import java.sql.Timestamp;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.Table;
```

```
import org.hibernate.annotations.CreationTimestamp;
```

```
import org.hibernate.annotations.UpdateTimestamp;
```

```
import lombok.Data;
```


Entity 클래스

❖ main/java/com.example.demo/domain-Board.java (2/2)

@Data

@Entity

@Table(name="boards")

// boards 테이블 생성

public class Board {

 @Id

 @GeneratedValue(strategy = GenerationType.AUTO)

// 기본키 설정

// JPA 구현체(Hibernate)가 자동으로 생성 방식을 결정

// hibernate_sequence가 자동으로 생성된다.

 private int no;

 private String writer;

 private String passwd;

 private String subject;

 private String content;

 @CreationTimestamp

 private Timestamp regdate;

 @UpdateTimestamp

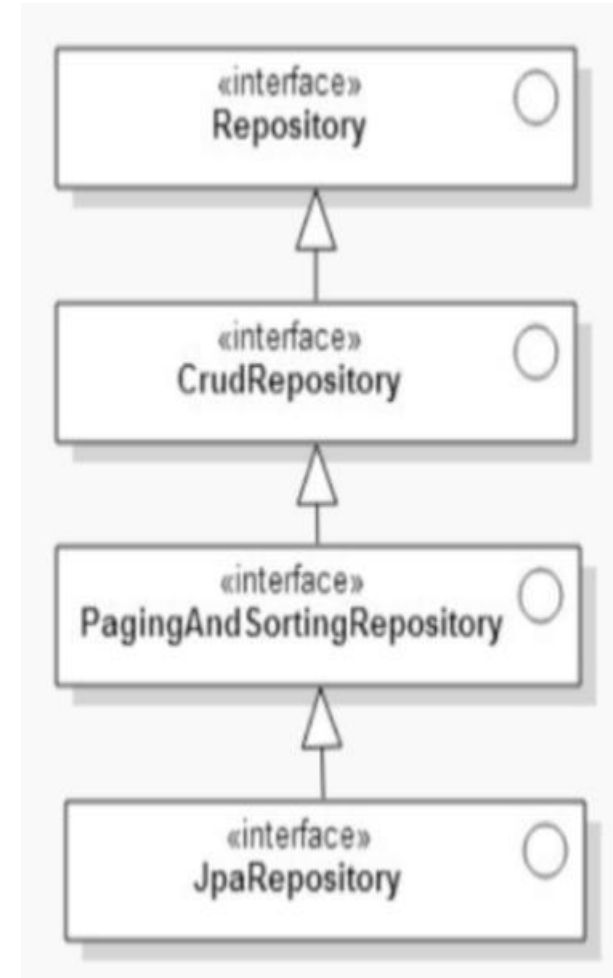
 private Timestamp updatedate;

}

Repository

❖ Repository 구조

- **Repository** : 최상위 Repository로서 아무런 기능이 없기 때문에 잘 사용하지 않는다.
- **CrudRepository** : 기본적인 CRUD 기능을 제공하는 Repository이다.
- **PagingAndSortingRepository** : CrudRepository 인터페이스 기능에 페이징 및 정렬 기능이 추가된 인터페이스 이다.
- **JpaRepository** : PagingAndSortingRepository 인터페이스 기능뿐만 아니라 JPA에 특화된 기능까지 추가된 인터페이스 이다.



Repository Interface

❖ `main/java/com.example.demo/repository-BoardRepository.java (1/2)`

```
package com.example.demo.repository;
```

```
import java.util.List;
```

```
import org.springframework.data.jpa.repository.Query;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import org.springframework.data.repository.query.Param;
```

```
import org.springframework.stereotype.Repository;
```

```
import com.example.demo.domain.Board;
```

Repository Interface

❖ main/java/com.example.demo/repository-BoardRepository.java (2/2)

@Repository

public interface BoardRepository extends CrudRepository<Board, Integer>{

//	public Board save(Board board);	// 글작성, 글수정
//	public long count();	// 글 갯수
//	public void delete(Board board);	// 글삭제
	public Board findByNo(int no);	// 상세 정보

// JPQL : 전체 목록 검색

@Query(value="select * from (select rownum rnum, board.* from " +
" (select * from boards order by no desc) board) " +
" where rnum >= (:page-1) * 10 + 1 and rnum <= (:page * 10)"

, nativeQuery = true)

public List<Board> findAll(@Param("page") int page);

}