# SpringData JPA

段維瀚 老師

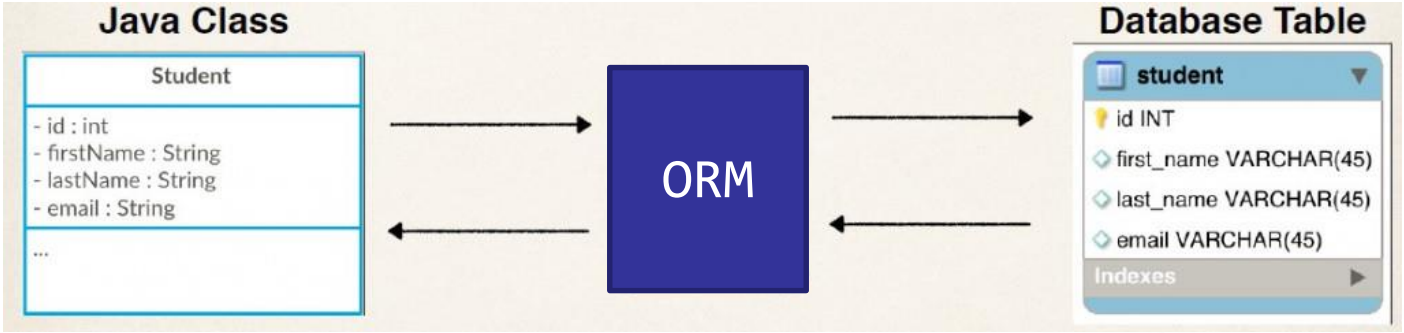
# 傳統資料庫應用開發上的問題 ！

- 實作一個具有基本資料庫(CRUD)存取功能的程式，需要寫大量且僵化的程式做資料庫的連接，甚至還需了解不同資料庫的語法，才能看到功能的雛形，而每多一個功能也需要做許多重複的事情。
- 如何可以有效地大幅降低資料庫存取功能的工作，讓開發人員更專注在商業邏輯上?

# Java與資料庫

- Java尋訪資料庫技術歷經幾個階段
  - JDBC
    - 優點：簡單
    - 缺點：程式碼量大，呆板
  - ORM 框架：Hibernate、MyBatis
    - 優點：程式碼少了
    - 缺點：需要進行二次封裝
  - SpringData JPA
    - 一次搞定
      - SpringData JPA 預設是通過 Hibernate 來實現

# ORM




- ORM（Object/Relation Mapping）
  - 物件/關聯資料庫映射
  - 讓操作資料庫/表如同操作物件一樣容易
- ORM 也是一種規範
  - 可以當成是應用程式與資料庫間的橋樑

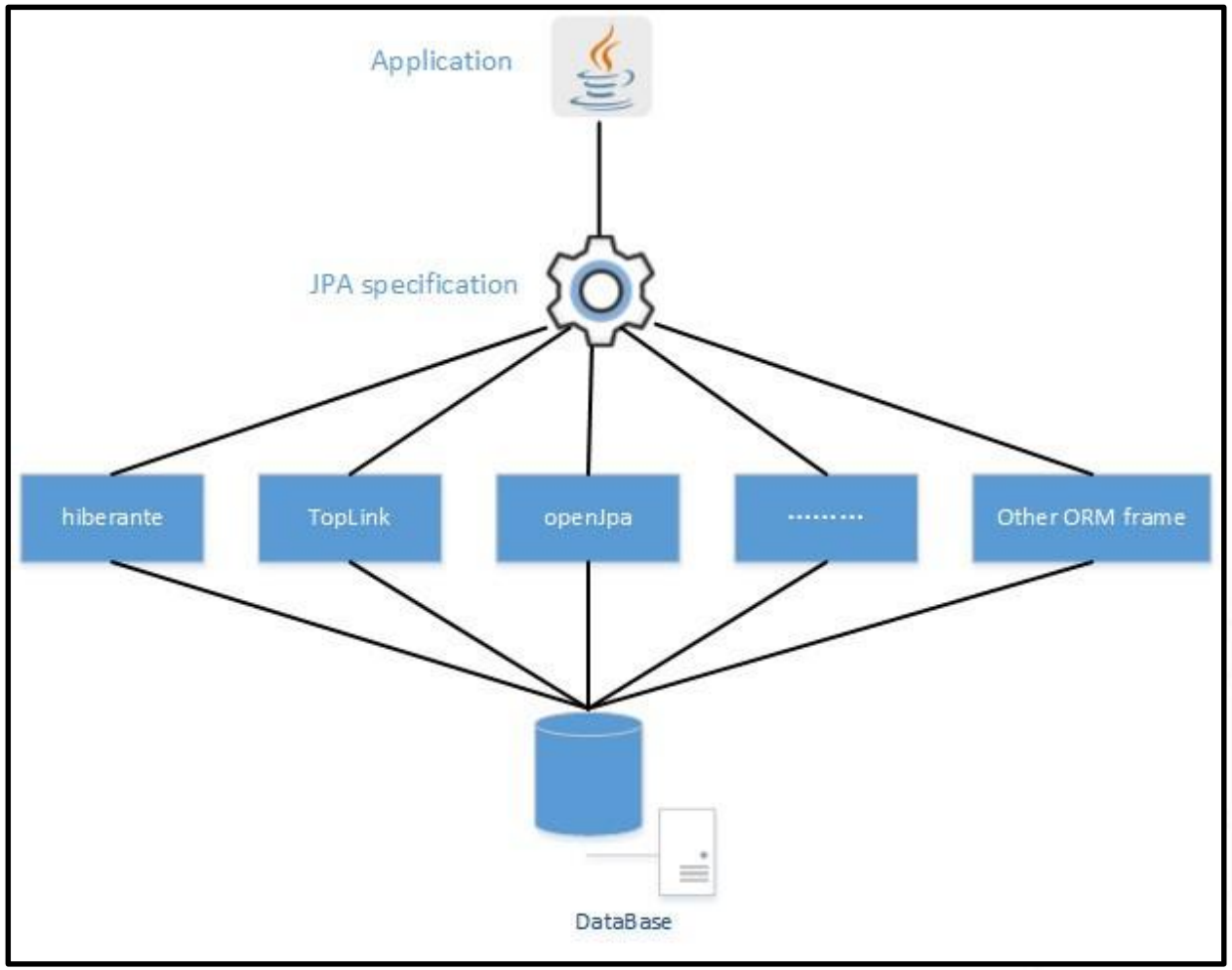
# ORM 的未來

- ORM 是物件導向程式設計與關聯式資料庫發展不同步的解決方案
  - 因著物件導向資料庫不斷發展，若關聯式資料庫有可能被物件導向資料庫取代時，如此就現今流行的ORM工具也將功成身退了。

# JPA(Java Persistence API)

- JPA 是什麼?
  - 透過 Java 將資料儲存到資料庫的 API 就叫做 JPA
  - JPA 是用於在關聯資料庫中存儲、查找(訪問)和管理 Java 物件的 Java ORM 標準
    - JPA 是 SUN 針對 ORM 技術提出的規範，目的為簡化持久化的開發工作以及整合各家 ORM 技術(Hibernate、TopLink、OpenJpa 等)。

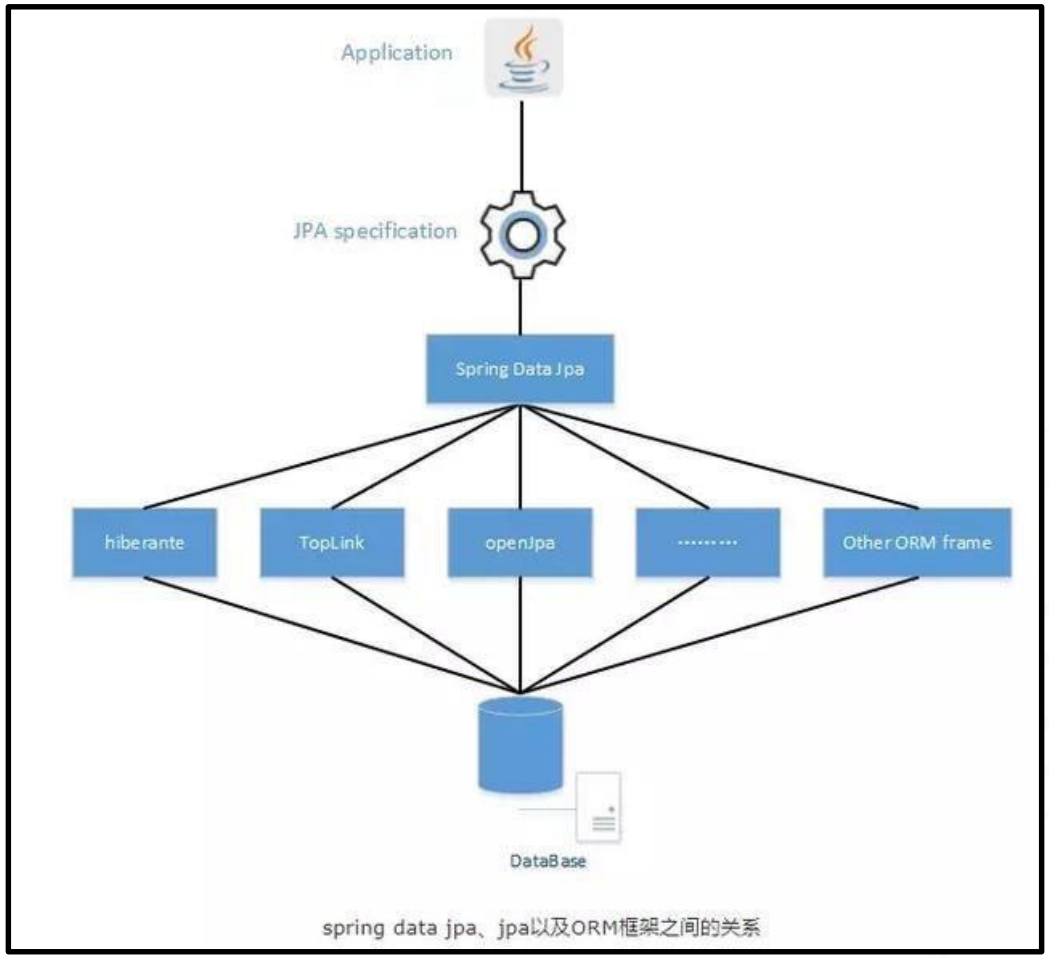
# JPA(Java Persistence API)



# SpringData JPA

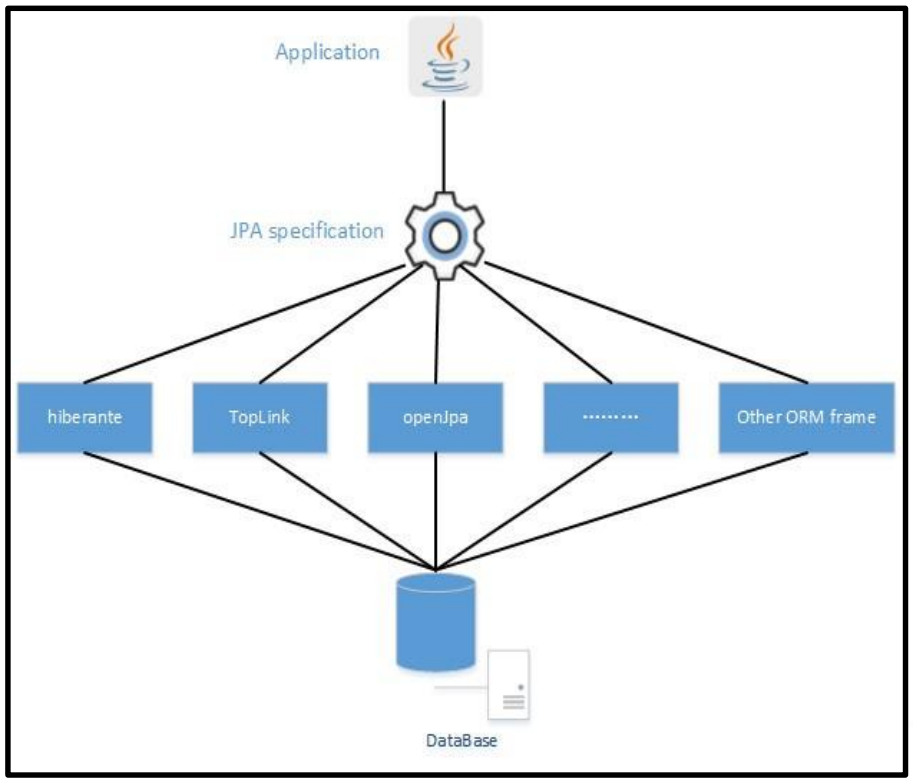
- 是一種實現JPA的技術與模組
  - 根據 ORM 框架和 JPA 規範而封裝的 JPA 應用框架，以精簡及減少編寫SQL代碼來改善開發者的開發效率
  - 目的
    - 降低存取資料層的工作量，讓開發人員只需寫出 repository 的介面，而 Spring 自動幫你實作其功能。

# SpringData JPA

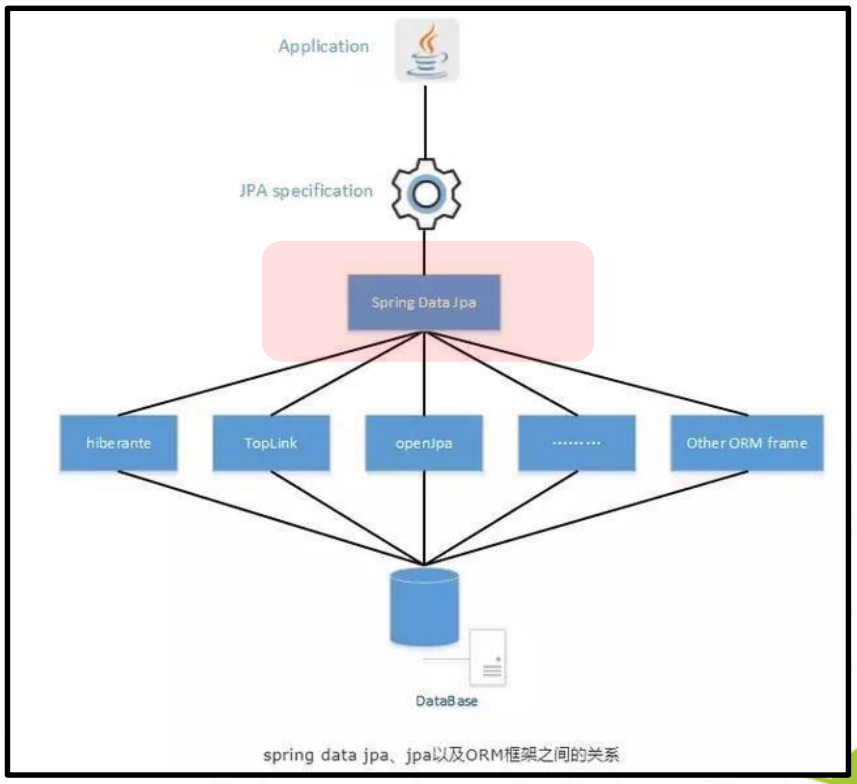



spring data jpa、jpa以及ORM框架之间的关系

# JPA




# SpringData JPA




spring data jpa、jpal以及ORM框架之间的关系

# SpringData JPA

- SpringData JPA 是 Spring 的一個子項目
  - 用於簡化資料庫CRUD
  - 支援關聯式資料庫技術
    - JDBC、JPA
  - 支援NoSQL資料庫
    - MongoDB、Neo4j、Redis、Hbase

# SpringData - JPA

- interface Repository
  - interface CrudRepository
    - interface PagingAndSortingRepository
      - interface JpaRepository

Spring Data JPA - Reference Documentation
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference

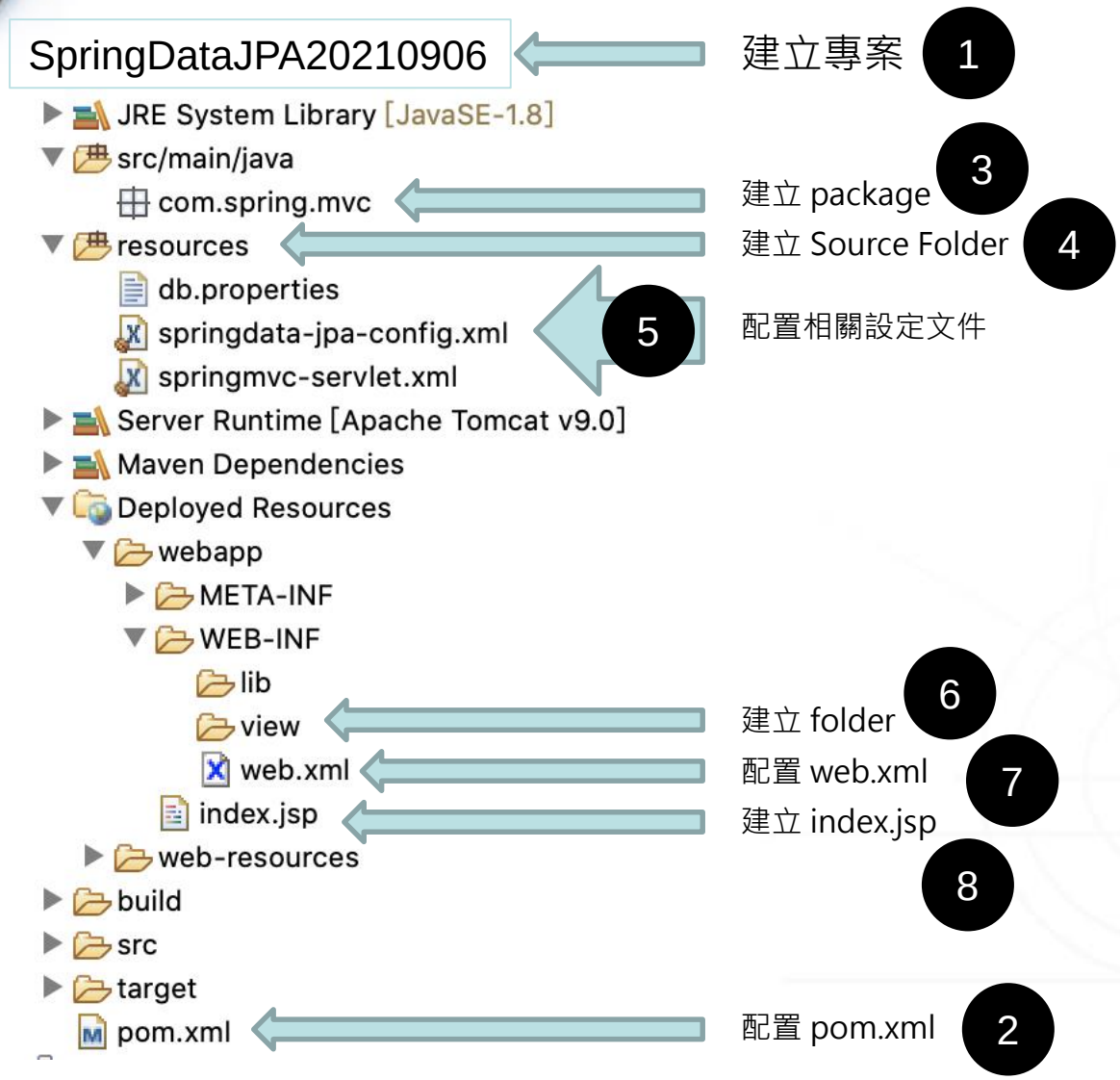
# SpringData - JPA

- 在開發上唯一要做的就是
  - 繼承持久層的介面
    - `extends JpaRepository<User, Long>`
      - SQL 業務邏輯上分為：
      - 不用撰寫 SQL
        - » 只要直接聲明方法名稱即可
          - » 例如：根據 name 來取得 User
            - » User getByName(String name);
            - » 當然方法名稱的設計需配合命名規範與原則
      - 撰寫 SQL
        - » @Query(“你的SQL指令”, nativeQuery = true或false)

pom.xml

```
<!-- SpringData -->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-commons</artifactId>
    <version>1.13.23.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.11.23.RELEASE</version>
</dependency>

<!--
Simple Logging Facade for Java
Java的簡單日誌記錄外觀（SLF4J）可作為各種日誌記錄框架（例如
java.util.logging，logback，log4j）的簡單外觀，允許最終用戶在部署 時插入
所需的日誌記錄框架。
-->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.30</version>
</dependency>
```


SpringDataJPA20210906
建立專案
1
JRE System Library [JavaSE-1.8]
src/main/java
com.spring.mvc
建立 package
3
resources
建立 Source Folder
4
db.properties
springdata-jpa-config.xml
springmvc-servlet.xml
5
配置相關設定文件
Server Runtime [Apache Tomcat v9.0]
Maven Dependencies
Deployed Resources
webapp
META-INF
WEB-INF
lib
view
建立 folder
6
web.xml
配置 web.xml
7
index.jsp
建立 index.jsp
web-resources
8
build
src
target
pom.xml
配置 pom.xml
2

# SpringData

- 配置
  1. 配置 數據源
  2. 配置 JPA 的 EntityManagerFactory
  3. 配置 事務管理器
  4. 配置 支援註解@的事務
  5. 配置 jpa:repositories

# JPA 常用註解 @

- @Entity 映射實體
- @Table 映射表
- @Column 映射欄位
- @Id 映射主鍵
  - @GeneratedValue 標示符自動生成
- @JoinColumn 關聯欄位
- @PrimaryKeyJoinColumn 主鍵關聯
- @JoinTable 關聯表的建立
- @OrderBy 排序

# JPA 操作設定

- @OneToOne
- @OneToMany
- @ManyToOne
- @ManyToMany
- @Transaction 交易進行
- @JsonIgnoreProperties
  - 防止 json 生成遞迴資料

# Lab: SpringData-JPA

```
@Entity
@Table(name = "Users")
public class Users {
    @Id
    @GeneratedValue
    private Long id;
    @Column
    private String name;
    @Column
    private Integer age;
    @Column
    private Boolean active;
    @Temporal(TemporalType.DATE)
    private Date birth;
    // getter, setter
}
```

Entity

Repository

```
@Repository
public interface UsersRepository extends JpaRepository<Users, Long> {

}
```

# 簡單查詢條件

- SpringData 規範
  - 查詢方法需以 find | read | get 開頭
  - 範例：
    - findByLastName(String lastName)
    - findByLastNameAndFirstName(String lastName,
      String firstName)

# JPQL方法的定義與規範

| Keyword | Sample | JPQL snippet |
|---|---|---|
| And | findByLastnameAndFirstname | … where x.lastname = ?1 and x.firstname = ?2 |
| Or | findByLastnameOrFirstname | … where x.lastname = ?1 or x.firstname = ?2 |
| Between | findByStartDateBetween | … where x.startDate between 1? and ?2 |
| LessThan | findByAgeLessThan | … where x.age < ?1 |
| GreaterThan | findByAgeGreaterThan | … where x.age > ?1 |
| After | findByStartDateAfter | … where x.startDate > ?1 |
| Before | findByStartDateBefore | … where x.startDate < ?1 |
| IsNull | findByAgeIsNull | … where x.age is null |
| IsNotNull,NotNull | findByAge(Is)NotNull | … where x.age not null |

JPQL: Java Persistence Query Language

```
@Table(name = "Users")
@Entity
public class Users {
    @Id
    @GeneratedValue
    private Integer id;
    @Column
    private String firstname;
    @Column
    private String lastname;
    @Column
    private Date startDate;
    @Column
    private Integer age;
    @Column
    private Boolean active;
    // getter / setter
}
```

# JPQL方法的定義與規範

| Keyword | Sample | JPQL snippet |
|---|---|---|
| Like | findByFirstnameLike | … where x.firstname like ?1 |
| NotLike | findByFirstnameNotLike | … where x.firstname not like ?1 |
| StartingWith | findByFirstnameStartingWith | … where x.firstname like ?1 (parameter bound with appended %) |
| EndingWith | findByFirstnameEndingWith | … where x.firstname like ?1 (parameter bound with prepended %) |
| Containing | findByFirstnameContaining | … where x.firstname like ?1 (parameter bound wrapped in %) |
| OrderBy | findByAgeOrderByLastnameDesc | … where x.age = ?1 order by x.lastname desc |
| Not | findByLastnameNot | … where x.lastname <> ?1 |
| In | findByAgeIn(Collection<Age> ages) | … where x.age in ?1 |
| NotIn | findByAgeNotIn(Collection<Age> age) | … where x.age not in ?1 |
| TRUE | findByActiveTrue() | … where x.active = true |
| FALSE | findByActiveFalse() | … where x.active = fals |

```java
@Table(name = "Users")
@Entity
public class Users {
    @Id
    @GeneratedValue
    private Integer id;
    @Column
    private String firstname;
    @Column
    private String lastname;
    @Column
    private Date startDate;
    @Column
    private Integer age;
    @Column
    private Boolean active;
    // getter / setter
}
```

# @Subselect(sql_statement)

- 子查詢
  - org.hibernate.annotations.Subselect
  - 將不可變且只能讀的實體映射到給定的SQL select 語句，相當於資料庫 View 的概念
  - 一般來說多會配合
    - @Immutable
      - 只能查詢註釋
    - @Synchronize({"table_name"})
      - 資料與指定資料表同步資訊

# @Subselect(sql_statement)

```
@Entity
@Immutable // 只能查詢
@Synchronize({"Users"}) // 資料與 Users 同步
@Subselect("SELECT u.id, u.name, u.password, u.birth, "
        + "(YEAR(CURRENT_DATE)-YEAR(u.birth)) as age "
        + "FROM Users u "
        + "ORDER BY age DESC")

public class UserView {
    @Id
    private Long id;
    @Column
    private String name;
    @Column
    private String password;
    @Temporal(TemporalType.DATE) // 得到 yyyy/MM/dd 格式
    private Date birth;
    @Column
    private Integer age;
    // getter, setter
}
```

# @Modifying

不支援
INSERT

- JPQL 支援自定義 UPDATE、DELETE
  - 操作時需要另外定義 Service 層，並在 Service 層加入事務操作

```
@Modifying
@Query("UPDATE Users u SET age = :age WHERE id = :id")
void updateUsersAge(@Param("id") Integer id, @Param("age") Integer age
```

```
@Service
public class UsersService {
  @Autowired
  private UsersRepository usersRepository;
  @Transaction
  public void updateUsersAge(Integer id, Integer age) {
     usersRepository.updateUsersAge(id, age);
  }
}
```