

JPA

Jakarta Persistence API

SC363204

Java Web Application Development
การพัฒนาโปรแกรมประยุกต์บนเว็บด้วยภาษาจาวา

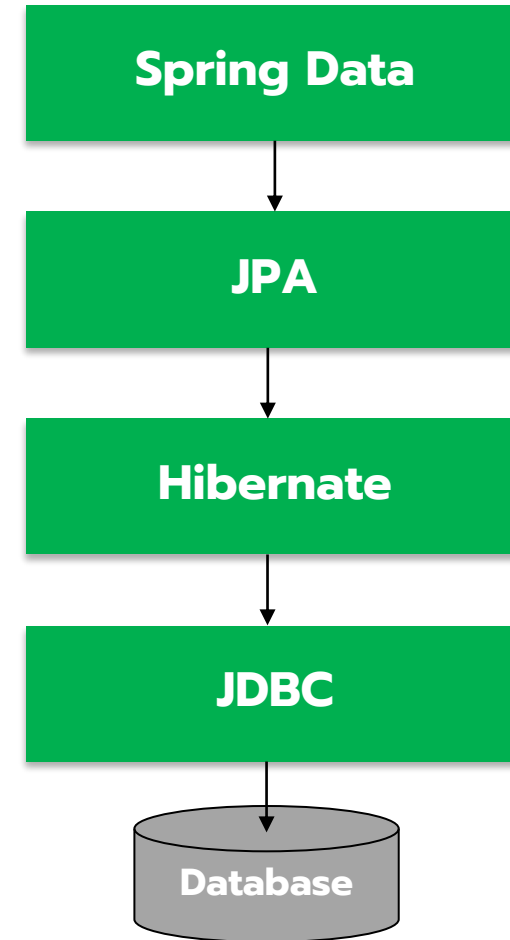


- JPA เป็นมาตรฐานใน Java EE ที่กำหนด (specification) **วิธีการ mapping** ระหว่าง object และ ข้อมูลในฐานข้อมูล (ORM: Object/Relation Mapping) **แบบอัตโนมัติ**
- JPA ช่วยให้นักพัฒนาสามารถจัดการกับข้อมูลใน object **เพียงอย่างเดียว** โดย**ไม่ต้อง**ใช้คำสั่ง SQL ช่วยให้นักพัฒนาเขียนโค้ดน้อยลง
- ไบรารีที่ implement ตามมาตรฐาน JPA มีชื่อว่า **"Hibernate"**

- **Spring Data** คือ Framework สำหรับเข้าถึงข้อมูลที่ครอบคลุมทั้งฐานข้อมูลแบบ relational และ non-relational, map-reduce frameworks และบริการบน cloud-based
- **คุณสมบัติของ Spring Data**
 - การจับคู่ Java Object กับโครงสร้างในฐานข้อมูลแบบอัตโนมัติ
 - สร้างและประมวลผล SQL แบบอัตโนมัติตามที่นักพัฒนากำหนด
- **โมดูลหลักของ Spring Data**
 - Spring Data Commons ส่วนแกนหลักของ Spring Data ทุกชนิด
 - Spring Data JPA ใช้สร้างโมดูลจัดการฐานข้อมูลแบบ relational database
 - Spring Data KeyValue ใช้สร้างโมดูลจัดการฐานข้อมูลแบบ Map-base
 - Spring Data LDAP ใช้สร้างโมดูลจัดการฐานข้อมูลผู้ใช้
 - Spring Data MongoDB ใช้สร้างโมดูลจัดการฐานข้อมูลแบบ non-relational ชื่อ MongoDB

Spring Data JPA

- **Spring Data JPA** เป็นชุด Library ที่ช่วยอำนวยความสะดวกในการใช้งาน JPA มีลักษณะเป็น Interface-based programming model กล่าวคือ ไม่ต้อง implement โค้ด แต่ใช้การกำหนด Query จากชื่อ method ใน interface แทน



Entity Class (หรือ JavaBean)

- Entity Class คือ คลาส JPA ใช้เป็นตัวแทนตารางหนึ่งในฐานข้อมูล
 - `@Id` ใช้ระบุว่าเป็น Primary Key
 - `@GeneratedValue` ระบุว่าให้รับค่า Primary คีย์แบบอัตโนมัติ
- Entity Class เป็นคลาสที่มีคุณสมบัติเช่นเดียวกับ JavaBeans แตกต่างกันที่ จะต้องมีการกำหนด Annotation
- การตั้งชื่อตัวแปรจะต้องอยู่ในรูปแบบ **camelCase** ส่วนชื่อคอลัมน์จะอยู่ในรูปแบบ **snake_case**
- ชนิดข้อมูลของตัวแปรในคลาสควรเป็นคลาส เช่น ใช้ **Integer** แทน **int**

```
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String firstName;
    private String lastName;

    // Getter and Setter Method
}
```

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI
💡 id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
🔍 first_name	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
🔍 last_name	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

ตาราง Customer
ที่จับคู่กับคลาส
Customer

การกำหนดชื่อตารางและคอลัมน์ด้วยตนเอง

- หากต้องการกำหนดชื่อตาราง ที่แตกต่างจากชื่อคลาส จะใช้

@Entity (name="ชื่อตาราง")




- กำหนดชื่อคอลัมน์ที่ไม่ใช่ชื่อเดียวกับ attribute

@Column(name="ชื่อคอลัมน์")

```
@Data
@Entity (name="tbl_customer")
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    @Column(name="fname")
    private String firstName;

    @Column(name="lname")
    private String lastName;
}
```

Column Name	Datatype	PK	NN
 id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 fname	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>
 lname	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>

Repository Class (หรือ DAO)

```
@Repository
public class CustomerRepository {

    @PersistenceContext
    private EntityManager entityManager; // ใช้เรียกเมธอดจัดการฐานข้อมูล ที่สร้างมาให้แล้ว

    public List<Customer> findAll() {
        Query query = entityManager.createQuery("from Customer"); // สร้างคำสั่ง SELECT ข้อมูลจากตาราง customer
        return query.getResultList(); // ดึงรายการผลลัพธ์จากการ Query ส่งกลับ
    }

    public Customer findById(Integer id) {
        return entityManager.find(Customer.class, id); // ค้นหา Customer ตาม id
    }

    @Transactional
    public Customer save(Customer customer) {
        entityManager.persist(customer); // insert กรณีไม่มีค่า id ใน object หรือ update กรณีมีค่า id ใน object
        return customer;
    }

    @Transactional
    public void delete(Integer id) {
        Customer customer = entityManager.find(Customer.class, id); // ค้นหาตาม id ที่ต้องการลบ
        entityManager.remove(customer); // เริ่มลบจริง
    }
}
```

- การนิยามคำสั่ง SQL ที่นอกเหนือจากเมธอดที่ JPA ได้ให้มา ใช้รูปแบบดังตัวอย่าง ซึ่งเป็นการกำหนดเงื่อนไข โดยไม่ระบุว่าการคอลัมน์ใด ซึ่งจะได้ข้อมูลทุกคอลัมน์เก็บลงใน object

```
public Employee findByLastName(String lastName) {  
    Query query = entityManager.createQuery("from Employee e where e.lastName = :LAST_NAME");  
    query.setParameter("LAST_NAME", lastName);  
    List resultList = query.getResultList();  
  
    return resultList.isEmpty() ? null : (Employee) resultList.get(0);  
}
```


- หากต้องการนิยามคำสั่ง SQL เองทั้งหมด ที่นอกเหนือจากเมธอดที่ JPA กำหนด ใช้รูปแบบดังตัวอย่าง

```
public List<Employee> findAllByNativeQuery() {  
    Query nativeQuery = entityManager.createNativeQuery("select id, first_name, last_name  
                                                         from employee", Employee.class);  
  
    return nativeQuery.getResultList();  
}
```

ตัวอย่างการใช้ count(*)

```
public BigInteger count() {  
    Query nativeQuery = entityManager.createNativeQuery("select count(*) from customer");  
    return (BigInteger)nativeQuery.getSingleResult();  
}
```

คำสั่ง SQL ของแต่ละ method

- save(S entity)

- กรณีที่ไม่มีค่า ID ใน object

INSERT INTO tablename VALUES (value1,value2,value3,...)

- กรณีที่มีค่า ID ใน object

UPDATE tablename SET column1=value1, column2=value2, ... WHERE
some_column=some_value

- findOne(ID primaryKey)

SELECT * from table where id = ?

คำสั่ง SQL ของแต่ละ method

- `Iterable<T> findAll();`

`SELECT * from tablename`

- `Long count();`

`SELECT count(*) from tablename`

- `void delete(T entity);`

`DELETE from table WHERE id = some_column=some_value`

ทดสอบ Repository

```
package com.example.demo.controller;
```

Class: HelloController

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

```
@RestController
```

```
public class HelloController {
```

```
@Autowired
```

```
JokerRepo repo;
```

```
@GetMapping("/hello")
```

```
//ทดสอบ insert
```

```
public void add() {
```

```
    Joker joker = new Joker();
```

```
    joker.setName("Somsak");
```

```
    joker.setWeight((float) 75);
```

```
    repo.save(joker);}
```

```
//ทดสอบ Select all
```

```
public String sayHello() {
```

```
    List<Joker> jokerList = repo.findAll();
```

```
    for(Joker j: jokerList) {
```

```
        System.out.println(j.getName());
```

```
    }
```

```
    return "Hello From Spring";
```

```
}
```

```
}
```

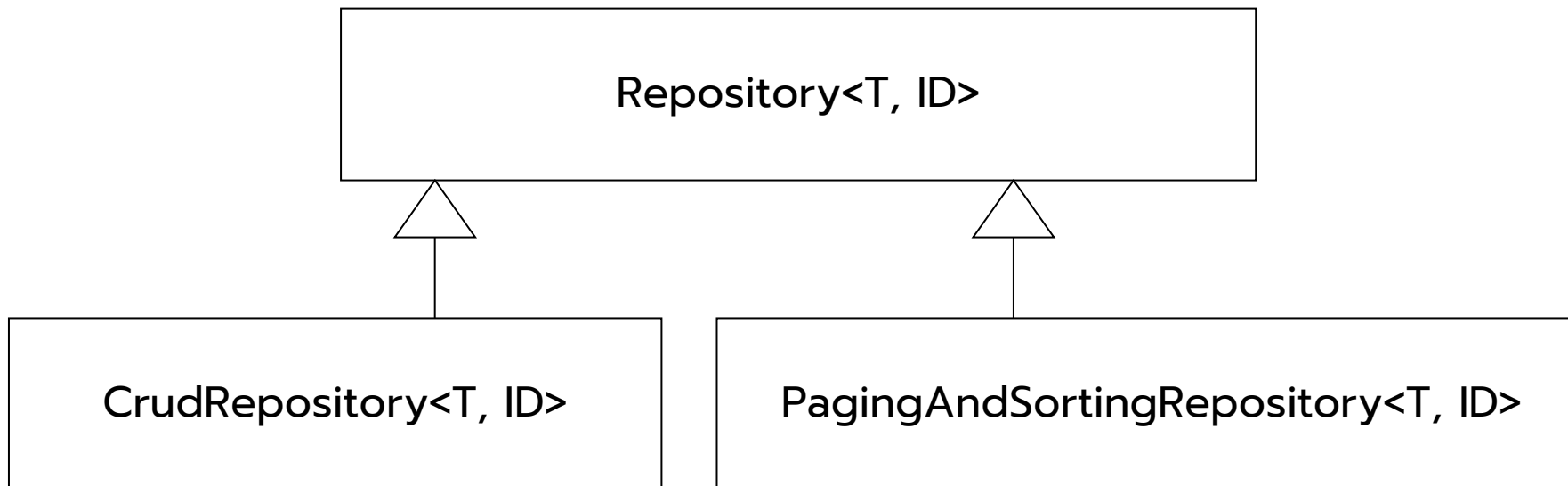
Annotation khác

- @Entity
- @Id
- @GeneratedValue
- @Column
- @JoinColumn
- @OneToMany
- @ManyToOne
- @ManyToMany

- Entity class ต้องระบุ @Entity เสมอ
- ภายในคลาสต้องมี attribute ที่ระบุ @Id เพื่อบ่งบอกว่า เป็น Primary Key
- ความสัมพันธ์ระหว่างตารางจะไม่ถูกสร้าง หากไม่ใส่ annotation
 - @OneToMany
 - @ManyToOne
 - @ManyToMany

Repository Interface

- Interface ภายใต Spring Data สร้างไว้คือ Repository ซึ่งเป็น interface หลักในการจัดการชนิดของคีย์หลัก และชนิดของคลาส Entity
- **CrudRepository** คือ interface ลูกสำหรับฟังก์ชัน CRUD พื้นฐาน
- **PagingAndSortingRepository** คือ interface สำหรับการทำ Paging และการ Sort



CrudRepository Interface

T, ID, S เรียกว่า Generic Type

```
public interface CrudRepository<T, ID extends Serializable>  
    extends Repository<T, ID> {
```

```
    <S extends T> S save(S entity);
```

บันทึกข้อมูลตาม Entity Object ที่กำหนด

```
    T findOne(ID primaryKey);
```

ค้นหาตามคีย์หลักและส่ง Entity Object กลับ

```
    Iterable<T> findAll();
```

ดึงข้อมูลทั้งหมดส่งกลับในรูปแบบอาร์เรย์ของ Entity Object

```
    Long count();
```

นับจำนวน Record ทั้งหมด

```
    void delete(T entity);
```

ลบข้อมูลตาม Entity Object ที่กำหนด

```
    boolean exists(ID primaryKey);
```

ตรวจสอบว่ามีข้อมูลตาม Entity Object ที่กำหนดหรือไม่ถ้ามีจะส่งค่าจริง ไม่มีส่งค่าเท็จ

```
    // ... more functionality omitted.
```

```
}
```

- Generic Type คือ การกำหนดอักษรใดๆขึ้นมา เพื่อเป็นตัวแทนของชนิดข้อมูลใดๆ
 - E – Element (ตัวมาตรฐานที่ใช้ใน Java Collections Framework)
 - K – Key
 - N – Number
 - T – Type
 - V – Value
 - S, U, V, ID, ... – ชนิดข้อมูลตัวที่ 2, 3, 4, 5, ...
- เรียนรู้ Generic Type เพิ่มเติมได้จาก

<http://www.tamemo.com/post/101/java-generic>

การสร้างเมธอดที่นอกเหนือจากเมธอดพื้นฐาน

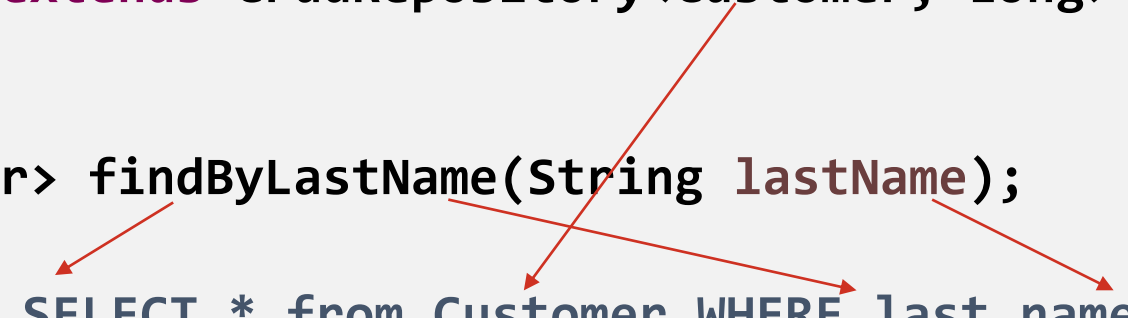
- หากต้องการสร้างเมธอดจัดการฐานข้อมูลที่นอกเหนือจากเมธอดพื้นฐานที่ interface CrudRepository เตรียมมาให้สามารถทำได้ โดยการระบุ **"Query keywords"** ให้กับชื่อเมธอด เช่น
 - find...By ใช้ในการ SELECT
 - count...By ใช้ในการนับ SELECT count()
- Hibernate จะสร้าง Query แบบอัตโนมัติจากชื่อของเมธอดที่นักพัฒนากำหนด เช่น
 - findAllByState(String state)
 - คำสั่ง SQL: SELECT * FROM [ชื่อ Entity ของ interface] WHERE state = ?
 - หมายถึงการค้นหาทุก record ตามฟิลด์ชื่อ state ที่อยู่ในฐานข้อมูล

```
public interface CustomerRepository
    extends CrudRepository<Customer, Long> {

    List<Customer> findByLastName(String lastName);

    // แทนคำสั่ง SELECT * from Customer WHERE last_name = ?

}
```



Query keywords

Logical keyword	Keyword expressions
AND	And
OR	Or
AFTER	After, IsAfter
BEFORE	Before, IsBefore
CONTAINING	Containing, IsContaining, Contains
BETWEEN	Between, IsBetween
ENDING_WITH	EndingWith, IsEndingWith, EndsWith
EXISTS	Exists
FALSE	False, IsFalse
GREATER_THAN	GreaterThan, IsGreaterThan
GREATER_THAN_EQUALS	GreaterThanEqual, IsGreaterThanEqual
IN	In, IsIn
IS	Is, Equals, (or no keyword)
IS_EMPTY	IsEmpty, Empty
IS_NOT_EMPTY	IsNotEmpty, NotEmpty
IS_NOT_NULL	NotNull, IsNotNull
IS_NULL	Null, IsNull
LESS_THAN	LessThan, IsLessThan
LESS_THAN_EQUAL	LessThanEqual, IsLessThanEqual
LIKE	Like, IsLike
NEAR	Near, IsNear
NOT	Not, IsNot
NOT_IN	NotIn, IsNotIn
NOT_LIKE	NotLike, IsNotLike
REGEX	Regex, MatchesRegex, Matches
STARTING_WITH	StartingWith, IsStartingWith, StartsWith
TRUE	True, IsTrue
WITHIN	Within, IsWithin

ตัวอย่างการใช้ชื่อเมธอด

Keyword	Sample	SQL
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname,findByFirstnames,findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgesIsNull	... where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1

ตัวอย่างการใช้ชื่อเมธอด

Keyword	Sample	SQL
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 = false
IgnoreCase	findByFirstnameIgnoreCase	... where UPPER(x.firstname) = UPPER(?1)

เมธอดที่ทำงานตามคำสั่ง SQL ที่กำหนด

- หากต้องการสร้าง Query นอกเหนือจาก Query keywords จะเขียน annotation @Query เหนือเมธอด ส่วนชื่อเมธอดจะมีรูปแบบอย่างไรก็ได้

คำสั่ง SQL จะไม่ใช่ชื่อฟิลด์ตาม Entity Class แต่ใช้ตามชื่อที่ถูกแปลงไปแล้วในฐานข้อมูล

```
@Query(value = "SELECT * FROM department d, employee e WHERE d.dept_id =  
                e.dept_id AND d.dept_name like '%?1'", nativeQuery = true)  
List<Employee> getEmployeeByDept(String deptName);
```

ระบุลำดับให้ตรงกับ
parameter ของ
เมธอด

Repository Interface

- เมธอดพื้นฐานที่ทำงานเกี่ยวกับ CRUD จะถูกเตรียมไว้ให้ทั้งหมดแล้วใน Interface Repository และ CrudRepository ดังนั้นหากไม่มีการทำงานใดนอกเหนือจากนี้ นักพัฒนาจะสร้าง interface ที่มีการสืบทอดจาก CrudRepository โดยไม่ต้องระบุชื่อเมธอดก็ได้

```
public interface CustomerRepository
    extends CrudRepository<Customer, Long> {

    // หากไม่มีเมธอดใดที่ทำงานมากกว่าเมธอดพื้นฐานสามารถเขียน
    interface เปล่าๆ ได้

}
```

ระบุชื่อ Entity
Class ที่จะใช้

ระบุชนิดข้อมูลของ Primary
Key
ซึ่งต้องเป็นชนิดข้อมูลที่เป็น
คลาสเท่านั้น ไม่สามารถใช้
primitive type ได้ เช่น int,
long, char, float