JPA & Hibernate

SenghuyJR11

Contents

- Introduce JPA
- JPA Architecture
- JPA Core Annotation
- JPA Project Setup
- Entity Relationship

1. ការណែនាំអំពី JPA

អ្វីជា ORM ?

- ORM = Object-relational mapping
- គេប្រើវាសម្រាប់ mapping រវាង java object និង database table ។

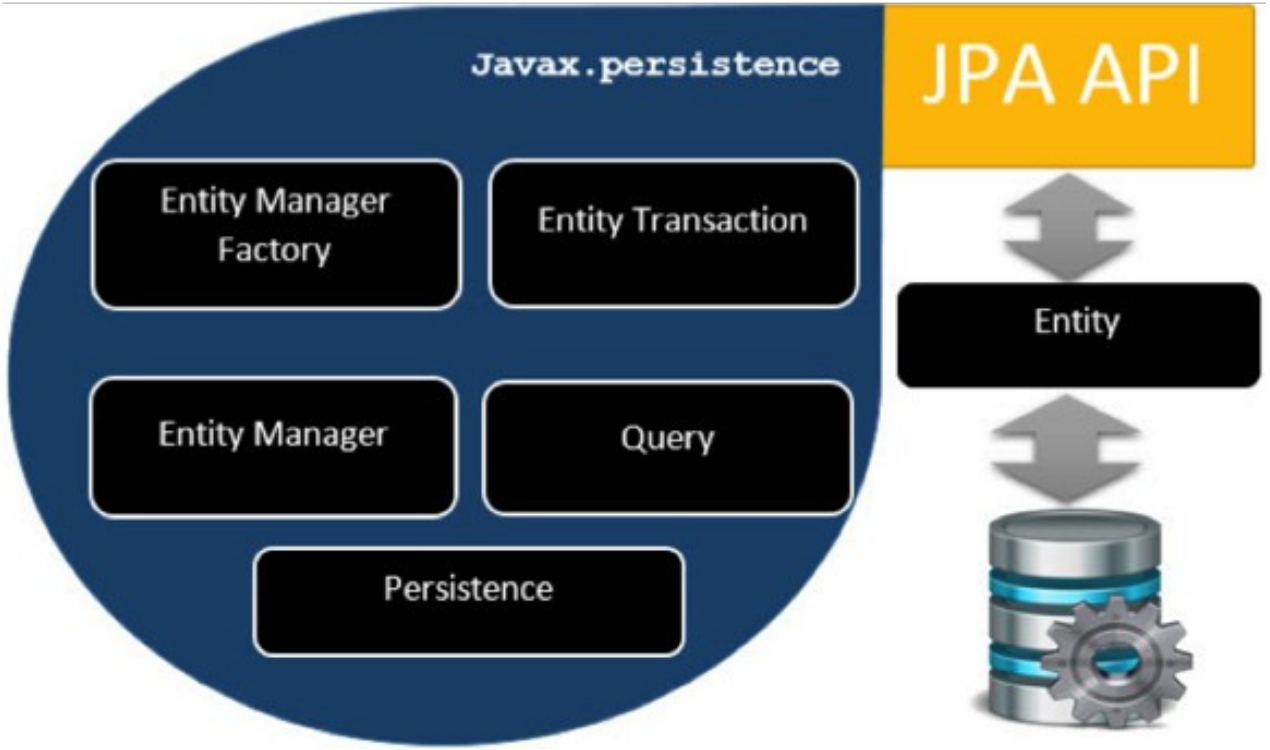
អ្វីជា JPA ?

- JPA = Java Persistence API
- វាជាបណ្តុំនៃ class និង method ប្រើសម្រាប់ store data ដែលច្រើននៅក្នុង database ។

អ្វីជា Hibernate ?

- វាជា Java framework ដែលប្រើសម្រាប់ store Java objects នៅក្នុង relational database system ។

2. JPA Achitecture



2. JPA Achitecture

- Entity ជា persistence object ដែល store record ក្នុង database ។
- EntityManager ជា Interface ដែល manages នៅ persistence operations លើ objects ហើយ ធ្វើជា factory ដើម្បីបង្កើត Query instance ។
- EntityManagerFactory ជាអ្នកបង្កើតនិង manages លើ multiple EntityManager instances ។
- EntityTransaction ជាអ្នករក្សាទុកនូវ operation របស់ EntityManager ។

2. JPA Achitecture

- Persistence ជា class ដែលចាំទទួល EntityManagerFactory instance ។
- Query ជា interface ដែល implement ចេញពី JPA ។

Note: persistence object ជា instance របស់ POJO class ដែលតំណាងឲ table row ក្នុង Database ។

3. JPA Core Annotation

JPA annotation សំខាន់ៗមានដូចជា៖

- @Entity វាបញ្ជាក់ថ class ថាជា table ដែល store ក្នុង database ។ instance របស់ Entity និងតំណាងថ table row ។name attribute ប្រើសម្រាប់ដាក់ឈ្មោះ class ។

```
import javax.persistence.Entity;

@Entity(name = "employee")
public class Employee implements Serializable {

}
```

3. JPA Core Annotation

- @Table វាជាអ្នកបញ្ជាក់ទៅកាន់ database table ណាដែលបាន map ជាមួយនិង Entity របស់វានិង។name attribute នៃ @Table annotation សម្រាប់ដាក់ឈ្មោះ database table ។

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

@Entity
@Table(name = "employee")
public class Employee implements Serializable {

}
```

3. JPA Core Annotation

- @Column វាប្រើសម្រាប់ map ទៅកាន់ database column ។ name attribute របស់វា សម្រាប់ដាក់ឈ្មោះ table column ។

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {

  @Column(name = "employee_name")
  private String employeeName;
}
```

3. JPA Core Annotation

- @Id វាប្រើសម្រាប់បញ្ជាក់ primary key នៃ Entity ។

```
import javax.persistence.*;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {
  @Id
  @Column(name = "id")
  private int id;
}
```

3. JPA Core Annotation

- @GeneratedValue វាសម្រាប់ generate strategies សម្រាប់ value នៃ primary key ។

```
import javax.persistence.*;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {

  @Id
  @Column(name = "id")
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
}
```

4. JPA Project Setup

- JPA Project Setup steps with postgresql
  - Add dependency
  - Config properties
  - Create JPA Entity
  - Extends JPA Repository

4. JPA Project Setup

Gradle dependencies

```
dependencies {
     implementation('org.springframework.boot:spring-boot-starter-data-jpa')
     implementation('org.springframework.boot:spring-boot-starter-web')
     implementation('org.postgresql:postgresql')
     testImplementation('org.springframework.boot:spring-boot-starter-test')
}
```

Configuration for postgresql

```
spring.jpa.database=POSTGRESQL
spring.datasource.platform=postgres
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=root
spring.jpa.show-sql=true
```

4. JPA Project Setup

- Create JPA entity

Example:

```
@Entity
@Table(name = "employee")
public class Employee{

  @Id
  @Column(name = "id")
  @GeneratedValue(strategy=SEQUENCE, generator="ID_SEQ")
  private int id;
}
```

4. JPA Project Setup

- Create Repository Interface extend jpaRepository

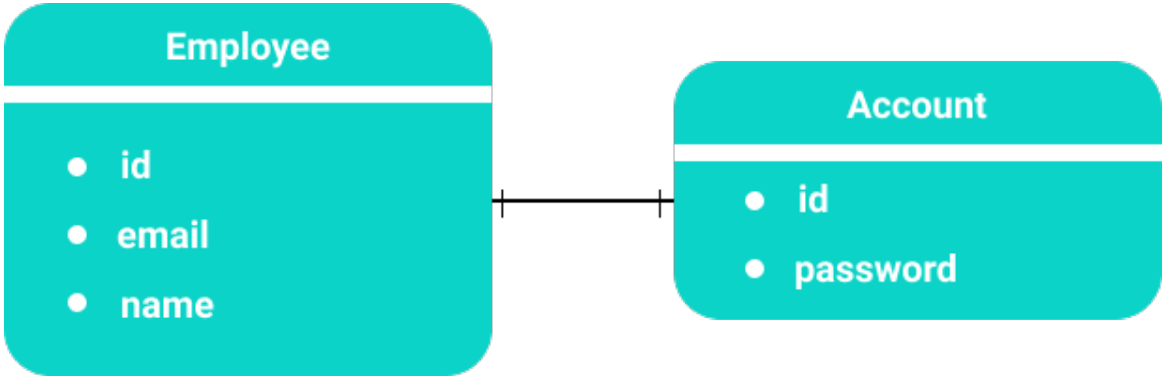
```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>{

  //List<Employee> findByFirstName(String FirstName);
  //List<Employee> findAll();
  //...your works...

}
```

5. Entity Relationship

- Entity Relationship: គឺជាទំនាក់ទំនងរវាង Entity មួយទៅផ្ទងៗទៀតដែលមានដូចជា:
  - One To One
  - One To Many
  - Many To One
  - Many To Many

5. Entity Relationship

- One To One Relationship: គឺជាលក្ខណៈដែល data មួយនៅក្នុង table មួយមានទំនាក់ទំនងជាមួយ data តែមួយប៉ុណ្ណោះនៅក្នុង table មួយផ្សេងទៀត



5. Entity Relationship

- One To One joining table:

Employee

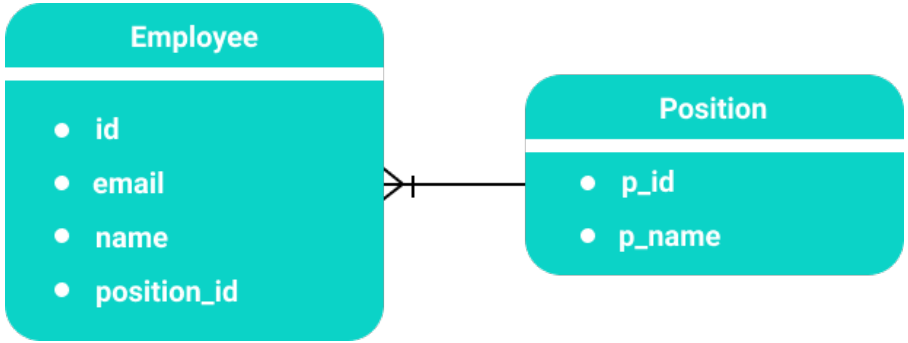
```
@Entity
@Table(name = "employee")
public class Employee {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private int id;
    private String name;
    private String email;
  }
```

Account

```
@Entity
@Table(name = "account")
public class Account {

    @Id
    private int id;
    private String password;
    @OneToOne
    @JoinColumn(name = "id",
referencedColumnName = "id")
    private Employee employee;
}
```

5. Entity Relationship

- One To Many **និង** Many To One Relationship: គឺជាលក្ខណៈដែល data នៅក្នុង table មួយមានទំនាក់ទំនងជាមួយ data ចាប់ពី២ឡើងទៅនៅក្នុង table មួយផ្ទុងទៀត ឬ data ជាច្រើននៅក្នុង table មួយមានទំនាក់ទំនងជាមួយ data តែមួយប៉ុណ្ណោះនៅក្នុង table មួយផ្ទុងទៀត។



5. Entity Relationship

- One To Many joining table:

Employee

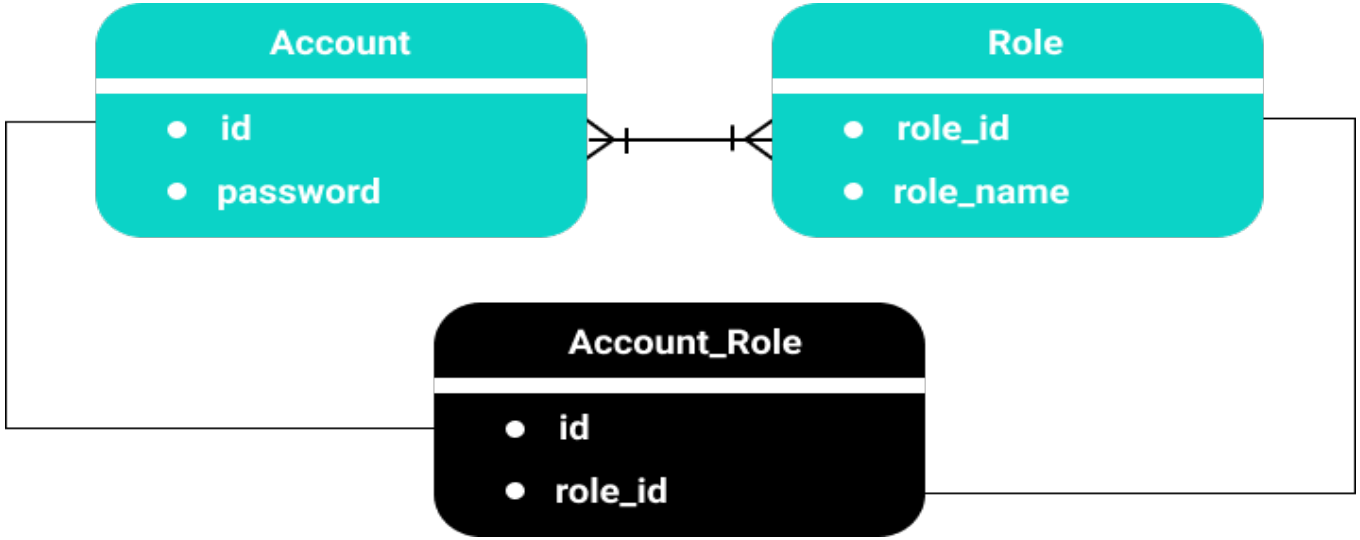
```
@Entity
@Table(name = "employee")
public class Employee {
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private int id;
    private String name;
    private String email;
    @ManyToOne(optional = false)
    @JoinColumn(name = "position_id", nullable =
false,
        referencedColumnName = "p_id")
    private Position position;
}
```

Position

```
@Entity
@Table(name = "position")
public class Position {
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private int p_id;
    private String p_name;
    @OneToMany(mappedBy = "position",
cascade = CascadeType.ALL)
    private List<Employee> employee;
}
```

5. Entity Relationship

- Many To Many Relationship: គឺជាលក្ខណៈដែល data ជាច្រើននៅក្នុង table មួយមានទំនាក់ទំនងជាមួយ dataចាប់ពី២ឡើងទៅនៅក្នុង table មួយផ្សេងទៀត។



5. Entity Relationship

- Many To Many joining table:

Account

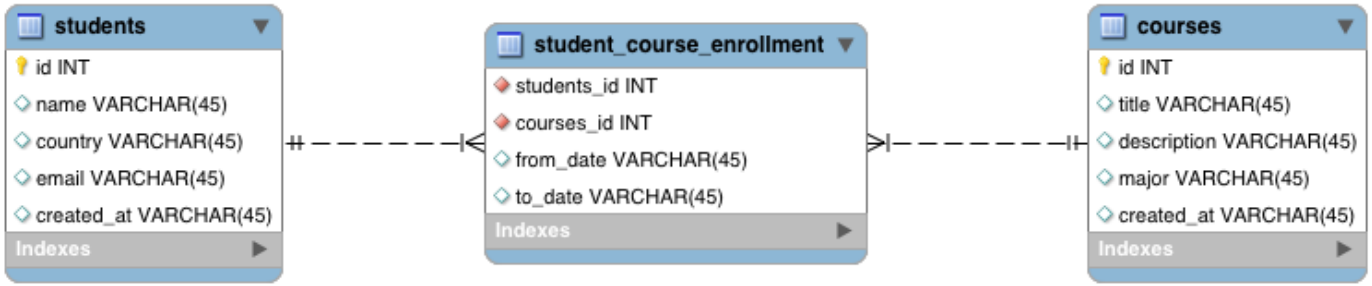
```
@Entity
@Table(name = "account")
public class Account {
    @Id
    private int id; private String password;
    @ManyToMany(cascade = CascadeType.ALL, fetch =
FetchType.EAGER)
    @JoinTable(name = "account_role",
        joinColumns = @JoinColumn(name = "id",
referencedColumnName        = "id"),
        inverseJoinColumns = @JoinColumn(name =
"role_id", referencedColumnName = "role_id"))
    private List<Role> role;
}
```

Role

```
@Entity
@Table(name = "role")
public class Role {
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private int role_id;

    private String role_name;
    @ManyToMany(mappedBy = "role")
    private List<Account> account;
// constructors, getters, setters
}
```

5. Entity Relationship

- Many To Many Extra Column Relationship : គឺជាការបន្ថែម column ទៅលើតារាងដែលជាចំណុចកណ្តាលសម្រាប់ភ្ជាប់តារាងដែលមានទំនាក់ទំនង Many To Many។



5. Entity Relationship

Students

```
@Entity
@Table(name = "students")
public class Student{

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name, country, email;
    @Column(name = "created_at")
    private String createdAt;
    @OneToMany(mappedBy = "student")
    private List <StudentCourseEnrollment>
studentCourseEnrollments;
}
```

Course

```
@Entity
@Table(name = "courses")
public class Course {
   @Id
   @GeneratedValue(strategy =
GenerationType.AUTO)
   private Long id;
   private String title,major,description;
   @Column(name = "created_at")
   private String createdAt;
   @OneToMany(mappedBy = "course")
   private List<StudentCourseEnrollment>
courseEnrollments;
}
```

5. Entity Relationship

StudentCourseEnrollment

```
@Entity
@Table(name = "student_courses")
@IdClass(StudentCourseId.class)
public class StudentCourseEnrollment implements Serializable{
    @Id
    @ManyToOne
    @JoinColumn(name = "course_id", referencedColumnName = "id")
    private Course course;
    @Id
    @ManyToOne
    @JoinColumn(name = "student_id", referencedColumnName = "id")
    private Student student;
    @JoinColumn(name = "from_date")
    private String fromDate;
    @JoinColumn(name = "to_date")
    private String toDate;
}
```

5. Entity Relationship

- Cascade: គឺជាវិធីសាស្ត្រក្នុងការ manage action ឬ operation ទៅលើ main-entity និង sub-entity របស់វា។
- JPA Cascade Type:
  - ALL: ប្រើសម្រាប់ manage លើ propagate នូវ operations (save, persist, merge, refresh, etc.) ក្នុង superclass ហើយ affect ទៅ subclass ។
  - PERSIST: manage លើ save operation ក្នុង superclass ។វាល់ពេលដែល save super-class object នោះ subclass object និង save auto ។

Note: propagate គឺ modifying a program and then sharing the modified code with sub-class

5. Entity Relationship

- MERGE: related entities និងត្រូវបាន merge ពេលដែល owning entity ត្រូវបាន merge ។

  វានិង update database ពេលដែលយើងកែអីមួយក្នុង persistent object ។

- REMOVE: លុប sub-entity ពេលដែល main-entity ត្រូវបានលុប។
- REFRESH: reload entity value ពី database ។ ពេល main-entity reload sub-entity និង auto reload ។
- DETACH: detaches all related entities if a “manual detach” occurs ។

  លុប sub-entity ពី persistent state ។

5. Entity Relationship

- Hibernate Cascade Type
  - REPLICATE: ប្រើសម្រាប់រៀបចំ data ជា sync ពេលដែលមាន data source ច្រើន ។
  - SAVE_UPDATE: propagates the same operation to the sub-entity. It's useful when we use Hibernate-specific operations like save, update and saveOrUpdate. It is used to save the entity into the database.
  - LOCK: set locking data When multiple users are working on same data

Note: When we don’t define any cascading type, then no operation in the superclass will affect the subclass.

5. Entity Relationship

- Cascade example:

```
@Entity
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String name;
    @OneToMany(mappedBy = "person", cascade = CascadeType.ALL)
    private List<Address> addresses;
}
```

5. Entity Relationship

- Fetch type: is the set of associated entity can be fetched lazily or eagerly. By default, @OneToMany and @ManyToMany associations use the FetchType.LAZY strategy while the @OneToOne and @ManyToOne use the FetchType.EAGER strategy instead.
  - LAZY loading: fetch when needed.

    ```
    fetch = FetchType.LAZY
    ```

  - EAGER Loading: fetch immediately.

    ```
    fetch = FetchType.EAGER
    ```

5. Entity Relationship

Fetch example:

```
@Entity
public class User{
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String name;
    @OneToMany(fetch = FetchType.LAZY)
    private Address address;
}
```

- ពេលយើង fetch User ផងហើយ fetch Address ផងគេហៅថា EAGER loading ។
- ពេលយើង fetch User តែយើង delay ការ fetch Address ហើយចាំ fetch ពេលត្រូវការ ហៅថា Lazy loading ។

5. Entity Relationship

- Recursive call in Bidirectional Relationship: StackOverflowError នឹងកើតឡើងនៅក្នុង Bidirectional Relationship នៅពេល Map ក្នុងទម្រង់ជា JSON ។ ដើម្បីការពារបញ្ហានេះ តម្រូវការប្រើប្រាស់ @JsonManagedReference និង @JsonBackReference annotation គឺចាំបាច់ក្នុងការ handle relation entities។
- @JsonManagedReference: បញ្ជូនផ្នែក reference ដែលត្រូវបាន serialized ។
- @JsonBackReference គឺជាមួយផ្នែកទៀតនៃ reference ដែលនឹងមិនរាប់បញ្ចូលក្នុង serialization ។

5. Entity Relationship

- Example:

```
public class User {
    public int id;
    public String name;

    @JsonManagedReference
    public List<Item>
userItems;
 }
```

```
public class Item {
    public int id;
    public String itemName;

    @JsonBackReference
    public User owner;
}
```

5. Entity Relationship

- Orphan Removal in Relationships: គឺជាវិធីសាស្ត្រក្នុងការ remove orphan entity ពី database ។ remove sub-entity ពេល main-entity បាន remove ។

```
@Entity
public class Book {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private int bookId;
	@Column(name = "book_name")
	private String bookName;

	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, orphanRemoval =
true)
	@JoinColumn(name = "book_id", referencedColumnName = "bookId")
	private List<Story> storyList = new ArrayList<>()

}
```

The end!

Everyone is endowed with different skills,
so... find yours and enjoy your part in this world...
Cheers!!

Thanks