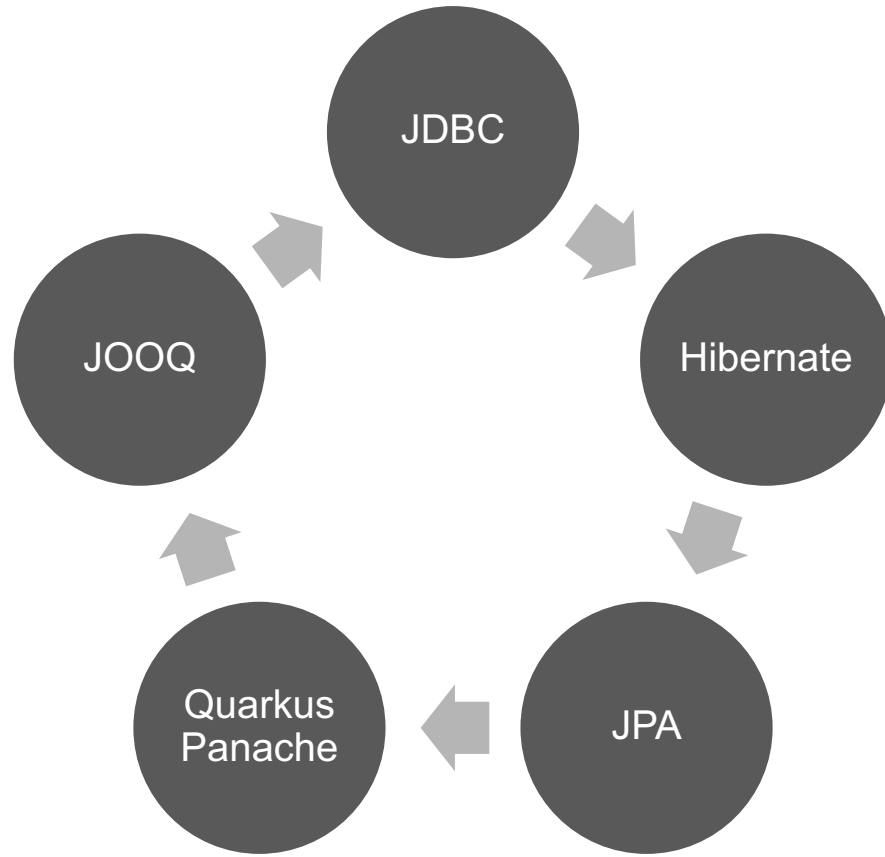




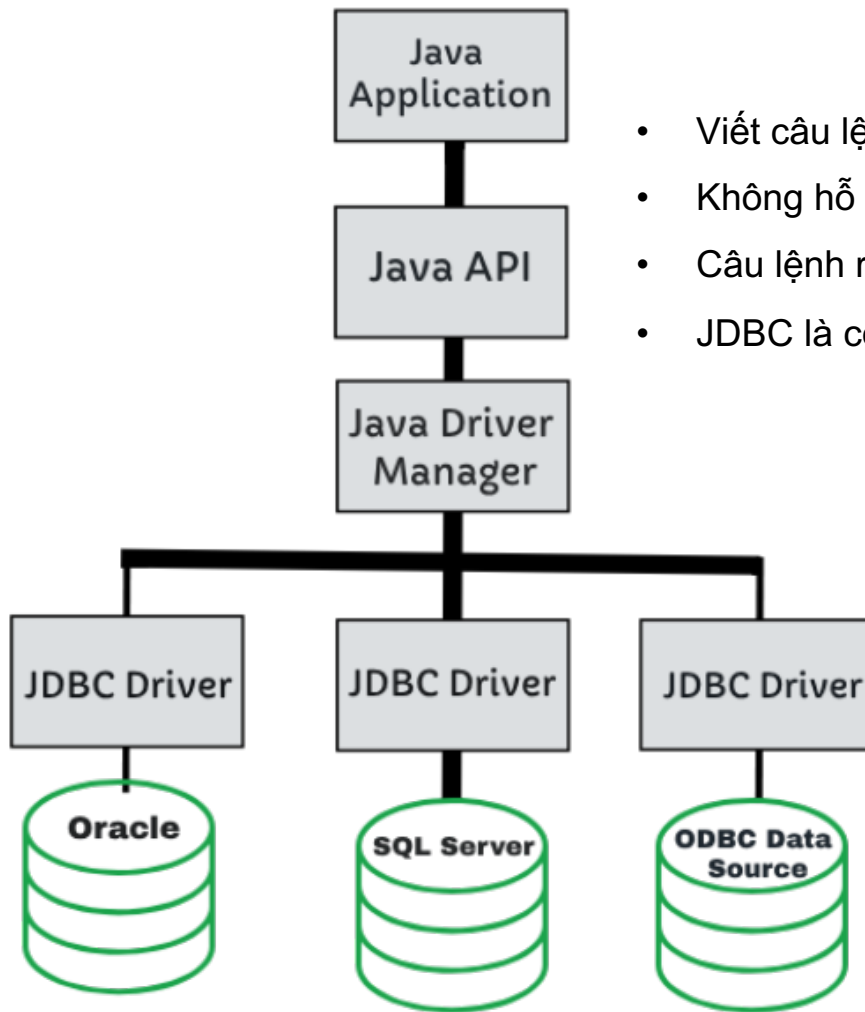
Lập trình JPA

cuong@techmaster.vn

JPA là gì?



JDBC Architecture



- Viết câu lệnh SQL trực tiếp.
- Không hỗ trợ ánh xạ giữa Java Object và Table RDBMS
- Câu lệnh rất dài
- JDBC là công nghệ. Còn Hibernate, JPA là framework.

```

Connection conn = null;
Statement stmt = null;
try {
    // STEP 1: Register JDBC driver
    Class.forName(JDBC_DRIVER);

    // STEP 2: Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");

    // STEP 3: Execute a query
    stmt = conn.createStatement();
    String sql = "INSERT INTO Registration " + "VALUES (100, 'Zara', 'Ali', 18)";

    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration " + "VALUES (101, 'Mahnaz', 'Fatma', 25)";

    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration " + "VALUES (102, 'Zaid', 'Khan', 30)";

    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration " + "VALUES(103, 'Sumit', 'Mittal', 28)";

    stmt.executeUpdate(sql);
    System.out.println("Inserted records into the table...");

    // STEP 4: Clean-up environment
    stmt.close();
    conn.close();
} catch (SQLException se) {
    // Handle errors for JDBC
    se.printStackTrace();
} catch (Exception e) {
    // Handle errors for Class.forName
    e.printStackTrace();
} finally {
    // finally block used to close resources
    try {
        if (stmt != null)
            stmt.close();
    } catch (SQLException se2) {
    } // nothing we can do
    try {
        if (conn != null)
            conn.close();
    } catch (SQLException se) {
        se.printStackTrace();
    } // end finally try
} // end try

```

Hãy chạy các hàm test trong file JDBCTest.java

```

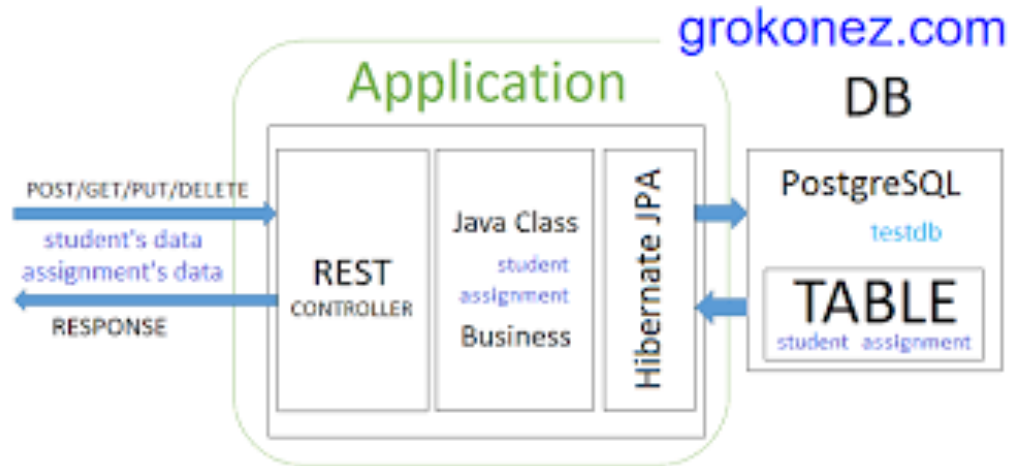
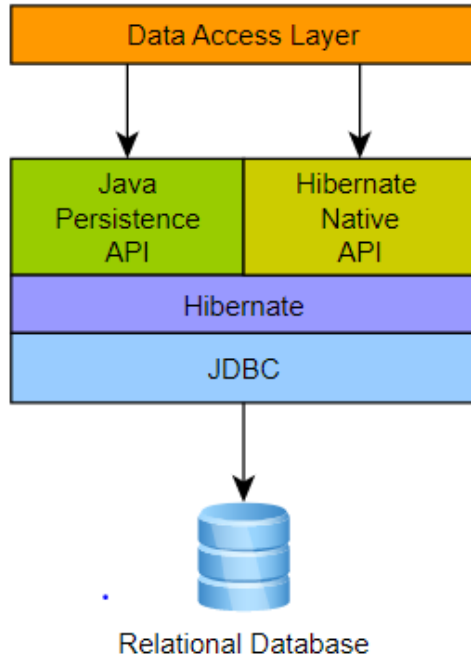
✓ JDBCTest 1.0s
  ✓ create_table() 322ms
  ✓ insert_data() 247ms
  ✓ query_data() 245ms
  ✓ drop_table() 211ms

```

Cảm giác khi lập trình JDBC, code rất dài dòng, không có nhiều hàm hỗ trợ. Thật là khâm phục thể hệ lập trình Java cách đây 15-20 năm trước

JPA – Java Persistence API

- Cung cấp các interface, các hàm tiện ích để thao tác CSDL quan hệ dễ dàng hơn
- JPA sẽ gọi xuống thư viện ORM Hibernate



Clean Code khi lập trình database

- Code ngắn, dễ hiểu
- Tránh viết lặp lại quá nhiều
- Sử dụng tối đa hiệu quả của ORM
- Ưu tiên thực hiện nghiệp vụ ở tầng Java business hơn là viết stored procedure SQL

Xa CSDL dữ liệu hơn. Tốc độ truy vấn chậm hơn
Nhưng có thể sử dụng caching

Chỉ cần vững Java là code được, logic dễ
đọc, dễ bảo trì

Dễ viết unit test

Exception handling, regular express, Java stream

Rất gần CSDL. Tốc độ truy vấn câu lệnh phức tạp
là tối ưu nhất

Phải nắm SQL và cú pháp stored procedure

Khó viết unit test

Bổ xung JPA vào maven

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

JPA

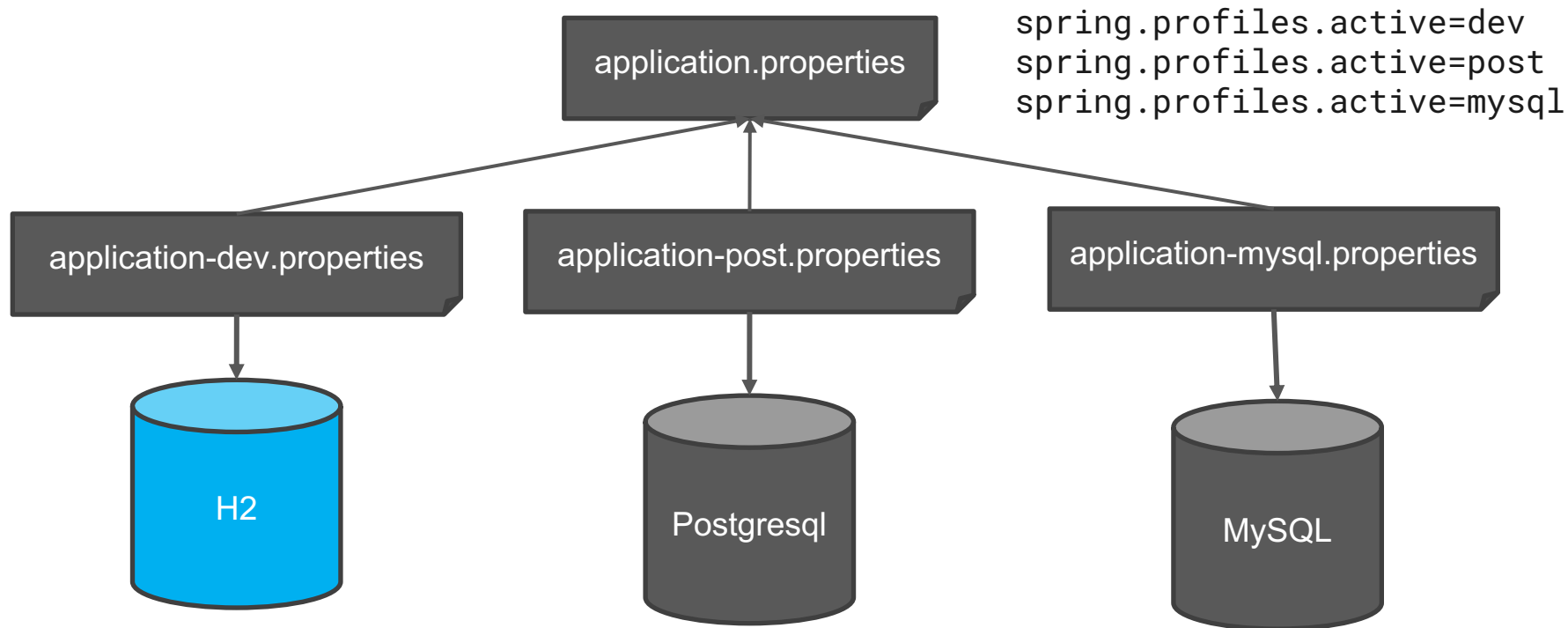
```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

H2 in memory database

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

Postgresql

Cấu hình kết nối CSDL



Chạy thử nghiệm
Unit Test

application.properties chỉ nên lưu những cấu hình chung có thể áp dụng cho các loại CSDL khác nhau

```
spring.jpa.properties.hibernate.hbm2ddl.import_files=f_post.sql,naturalperson.sql, person.sql //chèn dữ liệu ban đầu vào bảng
spring.jpa.hibernate.ddl-auto=create //tự động tạo bảng
spring.jpa.show-sql=false //hiển thị câu lệnh SQL trong console
spring.jpa.properties.hibernate.format_sql=false
```

application-dev.properties cấu hình kết nối H2

```
spring.datasource.url=jdbc:h2:mem:test
```

```
spring.datasource.driverClassName=org.h2.Driver
```

```
spring.datasource.username=sa
```

```
spring.datasource.password=123
```

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

```
spring.h2.console.enabled=true //bật web console để thao tác H2
```

```
spring.h2.console.enabled=true //bật web console để thao tác H2
```

← → ↻ 🏠 ⓘ localhost:8080/h2-console/login.jsp?jsessionid=l

English ▼

[Preferences](#) [Tools](#) [Help](#)

Login

Saved Settings:

H2 ▼

Setting Name:

H2

Save

Remove

Driver Class:

org.h2.Driver

JDBC URL:

jdbc:h2:mem:test

User Name:

sa

Password:

...

Connect

Test Connection

☒ Auto commit

 Max rows:

 Auto complete
 Auto select

jdbc:h2:mem:test

- ☒ AUTOID
- ☒ BAR
- ☒ CAR
- ☒ DEMOTABLEID
- ☒ E_ORDER
- ☒ E_POST
- ☒ FOO
- ☒ F_POST
- ☒ IDENTITYID
- ☒ NATURALPERSON
- ☒ PERSON
- ☒ SEQUENCEID
- ☒ TABLE_ID
- ☐ USERS
 - ☒ ID
 - ☒ DATE
 - ☒ EMAIL
 - ☒ MOBILE
 - ☒ FULLNAME
 - ☒ Indexes
- ☒ INFORMATION_SCHEMA
- ☒ Sequences
- ☒ Users
- H2 1.4.200 (2019-10-14)

SQL statement:

SELECT * FROM PERSON

SELECT * FROM PERSON;

ID	BIRTHDAY	CITY	FULLNAME	GENDER	JOB	SALARY
1	1970-02-19	Berlin	Riobard Folli	Male	Project Manager	10022
2	1963-02-09	Stockholm	Harlin McAuslan	Male	Personal Trainer	7094
3	1973-08-01	Wellington	Courtney Willson	Male	Photographer	17726
4	1983-05-16	London	Aleece Colquyte	Female	Accountant	19776
5	1967-07-18	Washington DC	Gustaf Allenby	Male	Pole Dancer	12541
6	1946-03-28	Helsinki	Hyacinthie Gayler	Female	Soldier	1836
7	1969-04-18	Hanoi	Brena Barradell	Female	Soldier	1599
8	1967-12-08	Canberra	Babara Speenden	Female	Film Maker	8511
9	1978-02-26	Wellington	Linnie Labitt	Female	Bartender	17581
10	1997-08-01	Canberra	Marcello Gofford	Male	Taxi Driver	4388
11	1969-04-21	Jakarta	Christopher Bohlens	Male	Soldier	908
12	1962-12-01	Cairo	Lavena Wattam	Female	Personal Trainer	9415
13	1975-11-04	Berlin	Cassandra Skurm	Female	Film Maker	18315
14	1946-12-21	Copenhagen	Vern Deniso	Male	Developer	19457

`application-mysql.properties` cấu hình kết nối MySQL

```
spring.jpa.hibernate.ddl-auto=update
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/demo
```

```
spring.datasource.username=demo
```

```
spring.datasource.password=toiyeuhanoi123-
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

application-pos.properties cấu hình kết nối Postgresql

```
## default connection pool

spring.datasource.hikari.connectionTimeout=20000

spring.datasource.hikari.maximumPoolSize=5

## PostgreSQL

spring.datasource.url=jdbc:postgresql://localhost:5432/demo

spring.datasource.username=demo

spring.datasource.password=toiyeuhanoi123-
```

Lệnh để khởi động Postgresql container

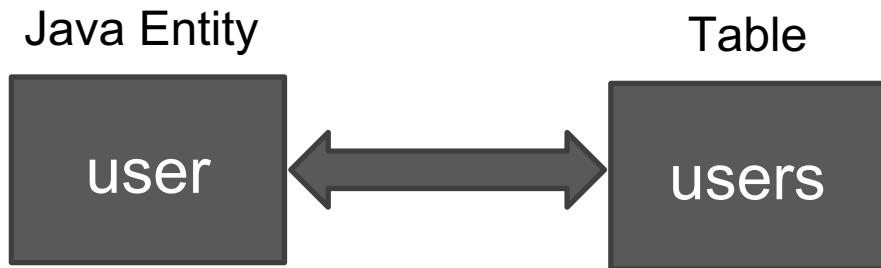
```
docker run --name pg -p 5432:5432 -e POSTGRES_USER=demo -e
POSTGRES_PASSWORD=toiyeuhanoi123- -d postgres:14.1-alpine
```

Định nghĩa Entity

@Entity, @Table

- @Entity cấu hình tên entity. Các lệnh JPQL sử dụng tên entity
- @Table cấu hình tên table trong CSDL. Các lệnh native SQL

```
@Entity(name="user")  
@Table(name="users")  
public class User
```



Quy ước đặt tên bảng @Table(name="")

- Luôn dùng danh từ tiếng Anh, số ít, đơn giản, phổ biến, dễ nhớ. "job" sẽ dễ đánh vần hơn "occupation"
- Không cần bổ xung prefix kiểu như "tbl", "list"....
- Tránh dùng lại từ khoá của CSDL. Ví dụ với Postgresql hãy tránh đặt tên bảng tên user, select, order, except, default

Các loại thuộc tính khi định nghĩa Entity

- Primary Key: duy nhất (trong 1 bảng, trong 1 hệ CSDL, trong cả 1 hệ mặt trời), không đổi, khó đoán hoặc tự tăng hoặc có thể sắp xếp.
- Trường căn bản ánh xạ xuống cột trong bảng
- Các thuộc tính cấu hình trường khi ánh xạ vào cột ở CSDL
- Trường dạng @Formula sinh dữ liệu lúc truy vấn SQL
- Trường dạng @Transient sinh dữ liệu khi gọi phương thức getter
- Định nghĩa các quan hệ thông qua foreign key
- Các luật để validate dữ liệu của trường

Trong phần tiếp theo sẽ đề cập từng loại trường một

Ví dụ #1

```
@Data
@Entity(name="user")
@Table(name="users")
public class User {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name="fullname", nullable=false, length=50)
    private String name;

    private String date;

    private String mobile;

    private String email;

    public User(String name, String date, String mobile, String email) {
        this.name = name;
        this.date = date;
        this.mobile = mobile;
        this.email = email;
    }
}
```

Ví dụ #2

```
@Entity(name = "person") @Table(name = "person") @Data
public class Person {
    @Id Long id; //primary key
    private String fullname;
    private String job;
    private String gender;
    private String city;
    private int salary;

    @Column(name="birthday") @Temporal(TemporalType.DATE) //Trường kiểu Date.
    private Date birthday;

    @Column(name="sex") @Formula(value = "case when gender='Male' then true else false end")
    private Boolean sex; //Trường dạng SQL formula sinh động khi truy vấn, không lưu xuống bảng.

    @Transient //Trường tạm thời sinh động lúc gọi Getter
    private int age;
    public int getAge(){
        Date safeDate = new Date(birthday.getTime());
        LocalDate birthDayInLocalDate = safeDate.toInstant().atZone(ZoneId.systemDefault())
            .toLocalDate();
        return Period.between(birthDayInLocalDate, LocalDate.now()).getYears();
    }
}
```

[-]	[-]	USERS
[-]	[-]	ID
	•	BIGINT DEFAULT NEXT VALUE FOR "PUBLIC"."SYSTEM_SEQUEN
[-]	[-]	DATE
	•	VARCHAR(255)
[-]	[-]	EMAIL
	•	VARCHAR(255)
[-]	[-]	MOBILE
	•	VARCHAR(255)
[-]	[-]	FULLNAME
	•	VARCHAR(50) NOT NULL
+	↓ a z	Indexes

Cấu trúc bảng xem trong H2 console

DDL Script khi sinh bảng vào Postgresql

```
CREATE TABLE public.users (
  id int8 NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  "date" varchar(255) NULL,
  email varchar(255) NULL,
  mobile varchar(255) NULL,
  fullname varchar(50) NOT NULL,
  CONSTRAINT users_pkey PRIMARY KEY (id)
);
```

Định nghĩa primary key

Mỗi Entity phải định nghĩa tối thiểu một primary key @Id

- @Id xác định trường nào là primary key
- Nếu không có @Id, khi biên dịch sẽ báo lỗi
- Nếu có nhiều hơn một @Id thì không định nghĩa như thế này

```
@Data
@Entity(name = "compound")
@Table(name = "compound")
public class Compound {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id1;

    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id2;
}
```

Lỗi biên dịch

@GeneratedValue sinh giá trị cho primary key

- @GeneratedValue(strategy = GenerationType.AUTO) chế độ mặc định, không tự sinh giá trị cho primary. Dev phải tự sinh, đảm bảo các tính chất của primary key
- @GeneratedValue(strategy = GenerationType.IDENTITY)
id **int8 NOT NULL GENERATED BY DEFAULT AS IDENTITY**
- @GeneratedValue(strategy = GenerationType.SEQUENCE)
Tạo ra hibernate sequence, mỗi lần insert bản ghi thì lấy ra giá trị tiếp theo
- @GeneratedValue(strategy = GenerationType.TABLE)
Tạo ra bảng để lưu giá trị primary key

@GeneratedValue(strategy = GenerationType.SEQUENCE)

```
1 Hibernate: call next value for hibernate_sequence
2 Hibernate: call next value for hibernate_sequence
3 Hibernate: call next value for hibernate_sequence
4 Hibernate: call next value for hibernate_sequence
5 Hibernate: call next value for hibernate_sequence
6 Hibernate: call next value for hibernate_sequence
```

Màn hình console in ra lệnh lấy giá trị tiếp theo của sequence

hibernate_sequence
trong CSDL Postgresql

The screenshot shows a database management tool interface. On the left, the 'Database Navigator' pane displays a tree structure of a PostgreSQL database. The 'Sequences' folder is expanded, and 'hibernate_sequence' is selected. On the right, the 'Properties' pane shows the configuration for the 'hibernate_sequence' table. The 'Last Value' field is empty, and the 'Start value' is 1. The 'Max Value' is 9223372036854775807. The 'Increment By' is 1, and the 'Cache' is 1. The 'Comment' field is empty. Below the 'Properties' pane, the 'Permissions' pane shows the permissions for the 'demo' user, including 'database owner', 'execute server program', 'monitor', 'read all data', 'read all settings', 'read all stats', 'read server files', 'signal backend', 'stat scan tables', 'write all data', and 'write server files'.

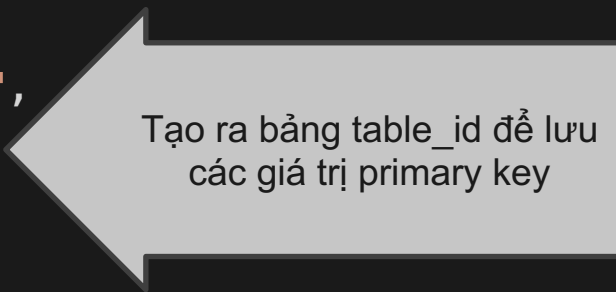
Table Name:	hibernate_sequence
Last Value:	
Start value:	1
Min Value:	1
Max Value:	9223372036854775807
Increment By:	1
Cache:	1
<input type="checkbox"/> Cycled	
Comment:	

demo

- pa database owner
- pa execute server program
- pa monitor
- pa read all data
- pa read all settings
- pa read all stats
- pa read server files
- pa signal backend
- pa stat scan tables
- pa write all data
- pa write server files

@GeneratedValue(strategy = GenerationType.TABLE)

```
@Entity(name="demotableid")
@Table(name="demotableid")
public class TableID {
    @TableGenerator(name = "table_id_generator",
        table = "table_id",
        pkColumnName = "id",
        valueColumnName = "value",
        allocationSize = 10)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "table_id_generator")
    private Long id;
}
```

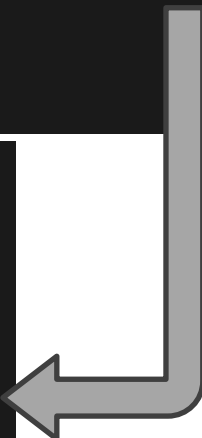


Tạo ra bảng table_id để lưu
các giá trị primary key

Custom ID generator

```
public class RandomIDGenerator implements IdentifierGenerator {  
    @Override  
    public Serializable generate(SharedSessionContractImplementor session, Object obj)  
        throws HibernateException {  
        RandomString randomString = new RandomString(10); // Sinh chuỗi ngẫu nhiên 10 ký tự  
        return randomString.nextString();  
    }  
}
```

```
@Data  
@Entity(name="bar")  
@Table(name="bar")  
public class Bar {  
    @GenericGenerator(name = "random_id", strategy =  
        "vn.techmaster.demojpa.model.id.RandomIDGenerator")  
    @Id @GeneratedValue(generator="random_id")  
    private String id;  
    private String name;  
}
```



Khi muốn gán vào primary một chuỗi
gồm 10 ký tự ngẫu nhiên a-z,A-Z,0-9

Kiểm thử RandomID generator

File IdTest.java. Khi chạy kiểm thử ở chế độ debug, chúng ta thấy chuỗi ID 10 ký tự ngẫu nhiên đã được sinh ra

```
✓ Local
> bar: Bar@52 "Bar(id=GLR6XjgD4K, name=Elephant)"
> id: "GLR6XjgD4K"
> this: IdTest@48

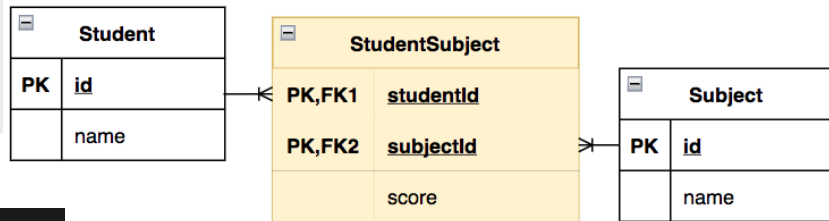
23
24
25
26
27
28
29

@Test
void randomIDGenerator() {
    Bar bar = new Bar(); bar = Bar@52 "Bar(id=GLR6XjgD4K, name=Elephant)"
    bar.setName("Elephant"); bar = Bar@52 "Bar(id=GLR6XjgD4K, name=Elephant)"
    String id = (String) testEntityManager.persistAndGetId(bar); id = "GLR6XjgD4K"
    assertThat(id).hasSizeGreaterThanOrEqualTo(10); id = "GLR6XjgD4K"
}
```

Composite Primary Key

1. Định nghĩa composite key

```
@Data
public class StudentSubjectId implements Serializable
{
    private String studentId;
    private String subjectId;
}
```



2. Định nghĩa Entity cho bảng trung gian

```
@Entity
@Data
@AllArgsConstructor
@IdClass(StudentSubjectId.class)
public class StudentSubject {
    @Id private String studentId;

    @Id private String subjectId;

    private int score;
}
```

Kiểm thử composite key

File idtest.java

```
@Test
@Transactional
void testCompositeKey(){
    StudentSubject ss1 = new StudentSubject("OX-11", "Math", 5);
    StudentSubject ss2 = new StudentSubject("OX-11", "English", 10);
    StudentSubject ss3 = new StudentSubject("OX-13", "Physics", 8);
    em.persist(ss1);
    em.persist(ss2);
    em.persist(ss3);
    em.flush();
    var query = em.createQuery("SELECT ss FROM StudentSubject ss",
StudentSubject.class);
    List<StudentSubject> result = query.getResultList();
    assertThat(result).hasSize(3);
}
```


@NaturalId

- @NaturalID tạo unique constrain lên một trường không phải PrimaryKey
- Dùng cho những dữ liệu bản chất đã là unique mà không cần hệ thống sinh ví dụ như email, di động, mã căn cước, ISBN
- @Id, primary cần giữ nguyên không đổi, nhưng @NaturalId có thể được phép thay đổi, miễn đảm bảo duy nhất

```
@Data
@Table(name="naturalperson")
@Entity(name="naturalperson")
@NoArgsConstructor
public class Person {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NaturalId
    private String email;
}
```

```
CREATE TABLE public.naturalperson (
  id int8 NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  email varchar(255) NULL,
  CONSTRAINT naturalperson_pkey PRIMARY KEY (id),
  CONSTRAINT uk_pc25mowwpr9v02qedmhxcffet UNIQUE
  (email)
);
```



Unique constraint do @NaturalId tạo ra

Tìm kiếm sử dụng NaturalId

```
Person p1 = new Person();  
p1.setEmail("cuong@techmaster.vn");  
em.persist(p1);  
  
Session session = em.unwrap(Session.class);  
Person p2 = session.byNaturalId(Person.class).using("email",  
"cuong@techmaster.vn").load();  
  
assertThat(p1).isEqualTo(p2);
```

Định nghĩa trường trong Entity

Các annotation định nghĩa Entity

- @Column và các thuộc tính
- @Transient tạo computed property hoặc sẽ không xuống CSDL
- @Formula gán biểu thức SQL
- @Temporal kiểu date/calendar
- @Embeddable, @Embedded
- @Pattern
- @Validation

Hãy làm quen với bảng Person

```
insert into person (id, fullname, job, gender, city, salary, birthday) values (1, 'Riobard Folli', 'Project Manager', 'Male', 'Berlin', 10022, '1970-02-19');
```

```
insert into person (id, fullname, job, gender, city, salary, birthday) values (2, 'Harlin McAuslan', 'Personal Trainer', 'Male', 'Stockholm', 7094, '1963-02-09');
```

```
{
  "id": 1,
  "fullname": "Riobard Folli",
  "job": "Project Manager",
  "gender": "Male",
  "city": "Berlin",
  "salary": 10022,
  "birthday": "1970-02-19",
  "sex": true,
  "age": 51
}
```

```
{
  "id": 2,
  "fullname": "Harlin McAuslan",
  "job": "Personal Trainer",
  "gender": "Male",
  "city": "Stockholm",
  "salary": 7094,
  "birthday": "1963-02-09",
  "sex": true,
  "age": 58
}
```

```
@Entity(name = "person") @Table(name = "person") @Data
public class Person {
    @Id Long id; //primary key
    private String fullname;
    private String job;
    private String gender;
    private String city;
    private int salary;

    @Column(name="birthday") @Temporal(TemporalType.DATE) //Trường kiểu Date.
    private Date birthday;

    @Column(name="sex") @Formula(value = "case when gender='Male' then true else false end")
    private Boolean sex; //Trường dạng SQL formula sinh động khi truy vấn, không lưu xuống bảng.

    @Transient //Trường tạm thời sinh động lúc gọi Getter
    private int age;
    public int getAge(){
        Date safeDate = new Date(birthday.getTime());
        LocalDate birthDayInLocalDate = safeDate.toInstant().atZone(ZoneId.systemDefault())
            .toLocalDate();
        return Period.between(birthDayInLocalDate, LocalDate.now()).getYears();
    }
}
```

@Column bổ xung thuộc tính khi ánh xạ trường vào cột trong bảng

- **name**: đặt lại tên bảng, khác với tên entity
- **unique**: tạo unique constraint
- **nullable**: cho phép null hay không null
- insertable / **updatable** chống cập nhật trong audit_log, price_history
- **length**: số ký tự tối đa áp dụng cho kiểu chuỗi
- precision, scale: áp dụng số thập phân

Entity

```
@Column(name="fullname", nullable=false, length=50)  
private String name;
```




fullname **varchar**(50) **NOT NULL**

@Transient – trường tính toán

@Transient đánh dấu trường sẽ tính toán khi được truy cập chứ không thực sự lưu dữ liệu xuống cột trong bảng. Ví dụ bảng Person lưu trường birthday (ngày sinh) luôn cố định với từng người. Nhưng tuổi thì tăng dần theo từng năm

```
@Column(name="birthday")
@Temporal(TemporalType.DATE)
private Date birthday;

@Transient
private int age;
public int getAge(){
    Date safeDate = new Date(birthday.getTime());
    LocalDate birthDayInLocalDate = safeDate.toInstant()
        .atZone(ZoneId.systemDefault())
        .toLocalDate();
    return Period.between(birthDayInLocalDate, LocalDate.now()).getYears();
}
```



Chú ý không thể viết lệnh truy vấn trên trường transient



Giá trị trường transient chỉ được tính khi bản ghi truy vấn và đổ vào đối tượng Java Entity. Lúc này quá trình truy vấn đã hoàn tất do đó không thể dùng trong biểu thức truy vấn, join...

```
List<Person> findByAge(Long age);
```



Caused by:

```
org.springframework.data.repository.query.QueryCreationException: Could
not create query for public abstract java.util.List
vn.techmaster.demojpa.repository.PersonRepository.findByAge(java.lang.Long)
Reason: Failed to create query for method public abstract
java.util.List
vn.techmaster.demojpa.repository.PersonRepository.findByAge(java.lang.Long)
Unable to locate Attribute with the the given name [age] on this
ManagedType [vn.techmaster.demojpa.model.Person]
```


@Formula sinh cột giả

```
@Column(name="sex")  
@Formula(value = "case when gender='Male' then true else false end")  
private Boolean sex;
```

Hibernate: select person0_.id as id1_11_, person0_.birthday as birthday2_11_, person0_.city as city3_11_, person0_.fullname as fullname4_11_, person0_.gender as gender5_11_, person0_.job as job6_11_, person0_.salary as salary7_11_, case when person0_.gender='Male' then true else false end as formula1_ from person person0_ where upper(person0_.fullname) like upper(?) escape ?

Tốc độ khi số lượng bản ghi lớn sẽ không tốt đâu. Chỉ hữu ích với số lượng bản ghi ít, không muốn lưu cột xuống CSDL

@Embeddable và @Embedded

[-]	[Table Icon]	E_ORDER
[+]	[ID Icon]	ID
[+]	[Text Icon]	CREATED_BY
[+]	[Text Icon]	CREATED_ON
[+]	[Text Icon]	UPDATED_BY
[+]	[Text Icon]	UPDATED_ON
[+]	[Text Icon]	PRODUCT_ID
[+]	[Index Icon]	Indexes
[-]	[Table Icon]	E_POST
[+]	[ID Icon]	ID
[+]	[Text Icon]	CREATED_BY
[+]	[Text Icon]	CREATED_ON
[+]	[Text Icon]	UPDATED_BY
[+]	[Text Icon]	UPDATED_ON
[+]	[Text Icon]	TITLE
[+]	[Index Icon]	Indexes

- Khi có nhiều bảng chung nhau một cấu trúc, thì đánh dấu cấu trúc dùng chung bằng @Embeddable
- Những bảng dùng chung cấu trúc đánh dấu bằng @Embedded

```
@Embeddable
@Data
public class Audit {
}
```

```
@Table(name="e_order")
@Entity(name="e_order")
public class Order {
    @Embedded
    private Audit audit = new Audit();
}
```

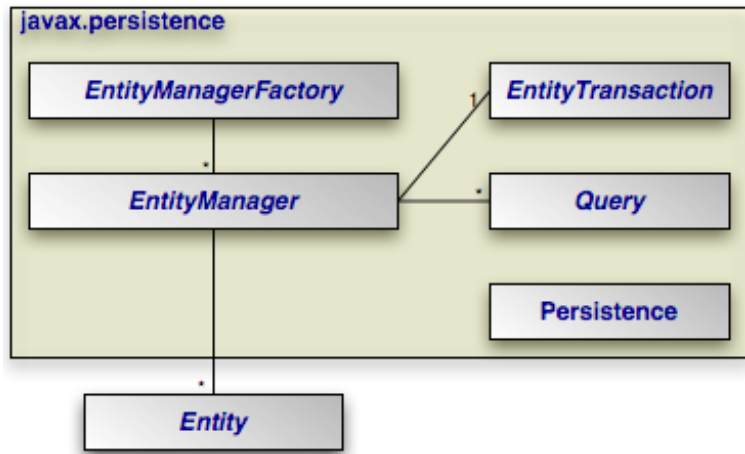
```
@Table(name="e_post")
@Entity(name="e_post")
public class Post {
    @Embedded
    private Audit audit = new Audit();
}
```

Entity Manager - TestingEntityManager

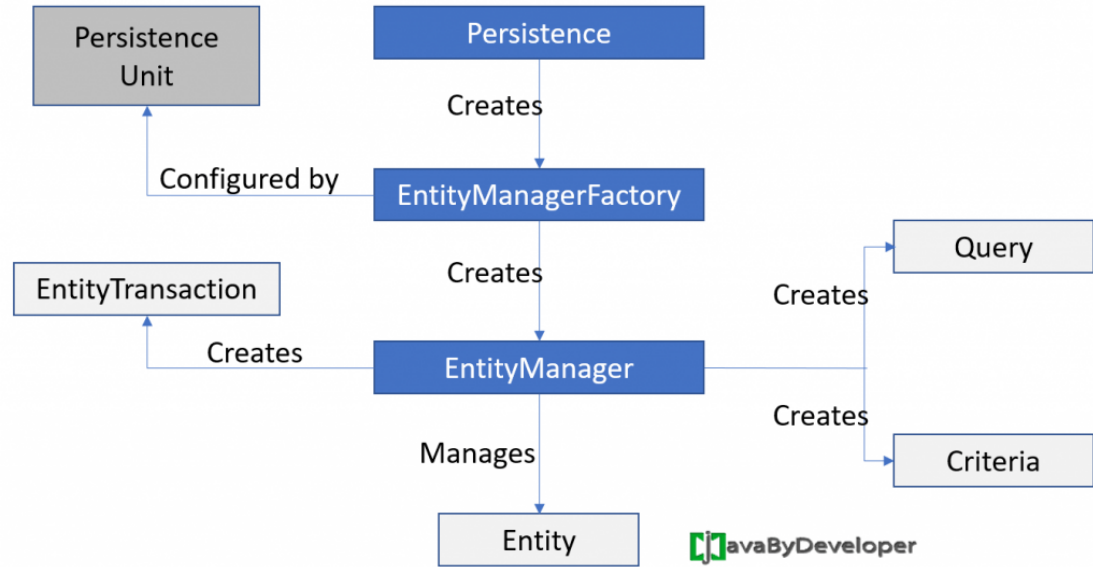
EntityManager là singleton object quản lý vòng đời các Java entities trong bộ nhớ.

EntityManager có API thực hiện CRUD, truy vấn, transaction xuống CSDL

Repository ứng với mỗi Entity có thể có nhiều. Nhưng EntityManager chỉ có 1 !



Minh hoạ 1

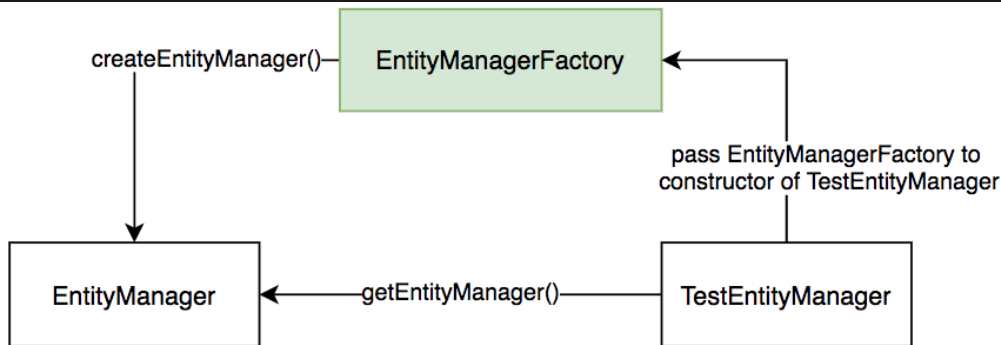


Minh hoạ 2

EntityManager vs TestEntityManager

- Cả EntityManager và TestEntityManager đều được tạo ra bởi EntityManagerFactory
- Cả hai đều có thể sử dụng trong Unit Test
- TestEntityManager có phương thức persistAndGetId
- Ngược lại EntityManager lại có vài phương thức để tạo Query.
- Từ đối tượng TestEntityManager có thể lấy đối tượng EntityManager bằng

```
EntityManager em = testEntityManager.getEntityManager();
```



2 cách khởi tạo EntityManager

Cả 2 cách khai báo biến EntityManager đều trở về một đối tượng duy nhất, điều đó chứng tỏ EntityManager là một singleton object

```
public class EMTest {  
    @Autowired private EntityManager em;  
    @PersistenceContext private EntityManager em2;  
  
    @Test  
    void testEntityManagerSingleton() {  
        assertThat(em).isEqualTo(em2); //Giống nhau  
    }  
}
```

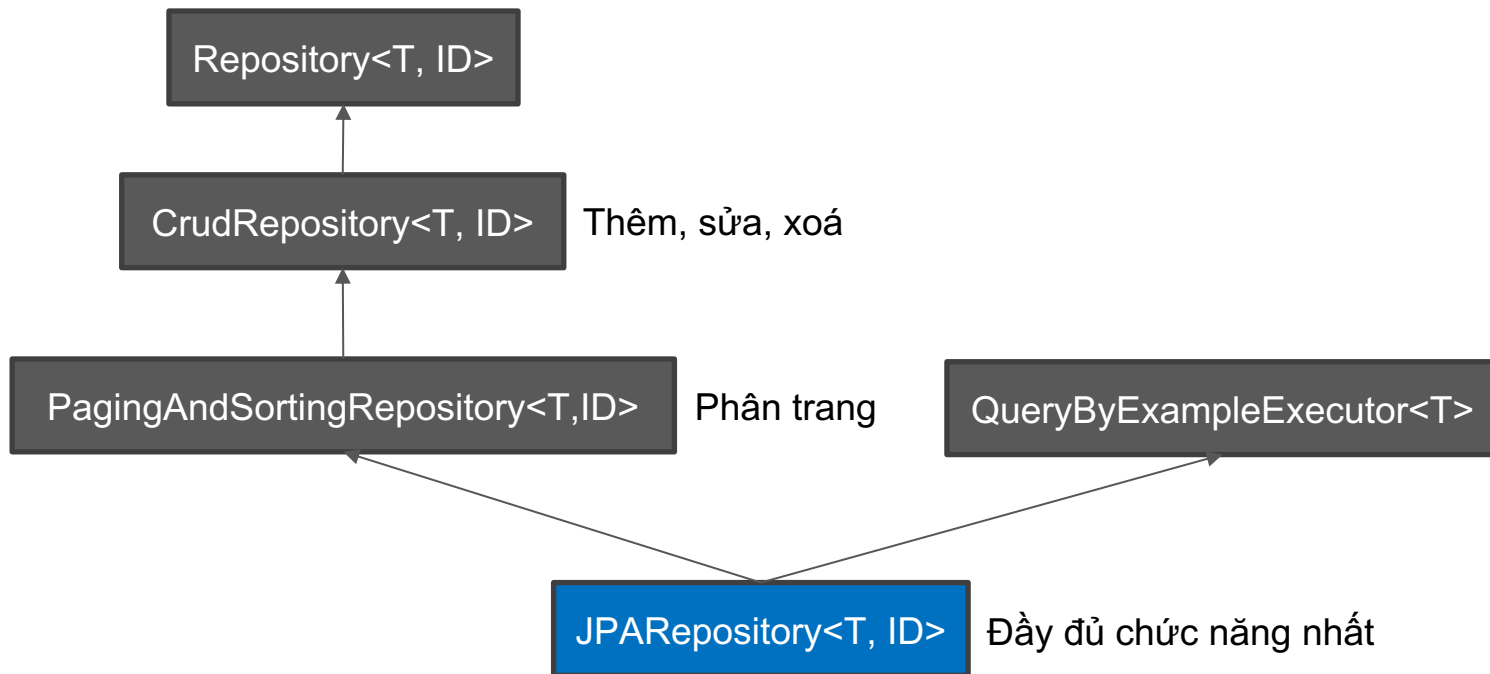
Entity Manager

- Dạng Interface
- CRUD các kiểu Entity
- Tạo được JPQL và native query
- Không hỗ trợ Derived Query
- Duy nhất - Singleton

Repository

- Dạng Interface có tham số kiểu Entity và Primary Key
- CRUD cho một kiểu Entity
- Bổ xung Query, Typed Query, Native Query
- Derived Query dễ viết, dịch sang SQL

Repository Interface



CRUD với Entity Manager

```
@RestController
@RequestMapping("/api/demo")
public class CRUDController {
    @Autowired private EntityManager em;
    @GetMapping("/crudbar")
    @Transactional
```

@Transaction buộc phải có nếu thêm, sửa, xóa dữ liệu

```
public void crudBar() {
    Bar bar = new Bar();
    bar.setName("Foo");
    em.persist(bar); //Create
    em.flush();
```

Hibernate: insert into bar (name, id) values (?, ?)

```
String id = bar.getId();
Bar barInDB = em.find(Bar.class, id); //Query
barInDB.setName("New Foo"); //Update
```

Context có sẵn entity nên không select nữa

```
em.flush();
em.merge(bar);
```

Hibernate: update bar set name=? where id=?

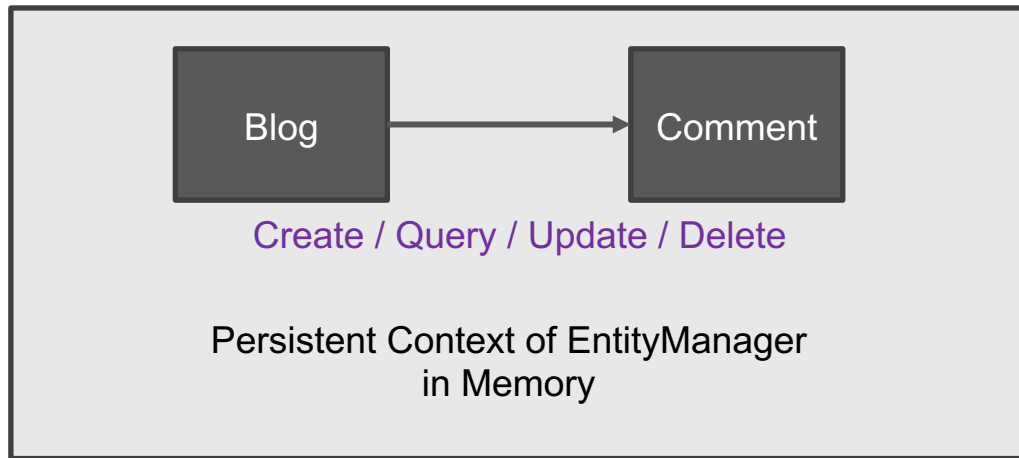
```
/* Cập nhật thay đổi từ database, nếu được sửa đổi từ một
thread khác, session khác. Còn trong điều kiện unit test, chạy
trong cùng một thread, thì không cần lệnh merge */
```

```
em.remove(bar); //Delete
em.flush();
```

Hibernate: delete from bar where id=?

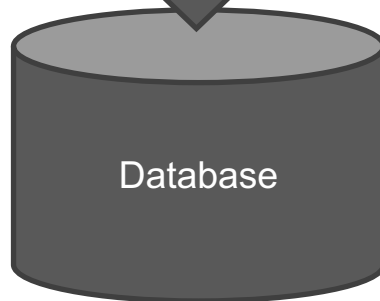
```
}
```

```
}
```



EntityManager.flush()

insert into XXX (...) values (...)
select ... from XXX
update XXX set ... = ...
delete XXX where



Khi chạy Unit Test, transaction sẽ không commit xuống CSDL thực sự

Trong ví dụ dưới đây, mọi thứ chạy ổn, đúng theo dự kiến. Tuy nhiên JPA ở trong unit test không thực sự lưu xuống CSDL mà chỉ lưu tạm trong bộ nhớ context do EntityManager quản lý

```
@Test
@Transactional
void tableIdGenerator() {
    TableID r1 = new TableID();
    r1.setName("Titok");
    em.persist(r1);
    var id1 = r1.getId();

    TableID r2 = new TableID();
    r2.setName("Bilibli");
    em.persist(r2);
    var id2 = r2.getId();

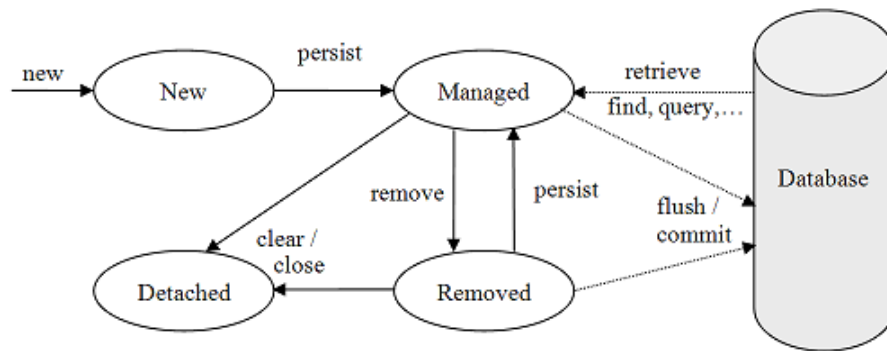
    em.flush();

    assertThat(em.find(TableID.class, id1)).isEqualTo(r1);
    assertThat(em.find(TableID.class, id2)).isEqualTo(r2);
}
```

Mở bảng ra không thấy có
bản ghi nào được insert

Một số chú ý

- Phải dùng `@Transactional` trong method của `@Controller` nếu sửa đổi dữ liệu
- Phải dùng `@DataJpaTest` trong testing class thì mới khởi tạo `EntityManager` hoặc `Repository`
- Khi chạy unit test, `EntityManager` không commit dữ liệu xuống database
- Muốn thực sự cập nhật thay đổi xuống CSDL hãy dùng `EntityManager.flush()`
- Trong vòng đời Entity có một số trạng thái



CRUD với Repository

CRUD Repository

```
@Test
void testRepositoryCRUD() {
    Bar bar = new Bar();
    bar.setName("Foo");

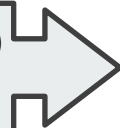
    barRepository.save(bar);
    String id = bar.getId();
    var foundBar = barRepository.findById(id).orElseThrow(() -> {
        return new RuntimeException("Bar is not found");
    });

    assertThat(foundBar).isEqualTo(bar);
    barRepository.delete(foundBar);
    assertThat(barRepository.existsById(id)).isFalse();
}
```

Repository không có hàm flush như EntityManager. Các thao tác dữ liệu thực hiện trong vùng nhớ persistence context trước. Trước khi hàm thoát, JPA thực hiện cập nhật xuống CSDL

```
@GetMapping("/crudbar2")
@Transactional
public void crudBar2() {
    Bar bar = new Bar();
    bar.setName("Foo");
    barRepository.save(bar);
    String id = bar.getId();
    var foundBar = barRepository.findById(id).orElseThrow(() -> {
        return new RuntimeException("Bar is not found");
    });
    barRepository.delete(foundBar);
}
```

Hibernate: insert into bar (name, id) values (?, ?)
Hibernate: delete from bar where id=?



Chạy trước khi hàm thoát

Query

JPA cung cấp các loại query sau đây

- Named Query
- Derived Query: dùng biểu thức hàm để sinh câu lệnh SQL
- Untyped JPQL Query / Typed JPQL Query
- Native Query
- Query By Example (không bàn tới trong slide này vì nó hoàn toàn có thể thay thế bằng các loại query bên trên /

JPQL - Java Persistence Query Language

- JPQL khá giống với SQL nhưng nó thực hiện lệnh truy vấn đối với Java Entity. Dùng EntityManager tạo JPQL hoặc khai báo JPQL trong repository interface
- Định nghĩa kiểu để nhận dữ liệu trả về

```
@Query("SELECT new vn.techmaster.demojpa.repository.MakerCount(c.maker, COUNT(*)) FROM oto AS c GROUP BY c.maker ORDER BY c.maker ASC")
```

- Hỗ trợ inner join, left outer join, right outer join...
- Hỗ trợ group by
- Hỗ trợ một số hàm upper, lower, current_date, abs...

@NamedQuery

Chú ý tên của Entity **oto** khác với tên của Table **car**. Tên của Entity sẽ được sử dụng trong câu lệnh JPQL

```
@Entity(name = "oto") //tên entity sẽ sử dụng trong câu lệnh JPQL
@Table(name = "car") //tên table sẽ sử dụng để lưu xuống bảng vật lý trong CSDL
@Data //annotation của Lombok
@NamedQuery(name = "Car.findById", query = "SELECT c FROM oto c WHERE c.id=:id")
public class Car {
    @Id private long id;
    private String model;
    private String maker;
    private int year;
}
```

NamedQuery gắn liền với Entity

Nếu bạn không sử dụng repository interface mà chỉ dùng EntityManager để thao tác dữ liệu. Query đó sử dụng ở nhiều nơi khác nhau thì có thể dùng @NamedQuery.

Về mặt clean code, thì không khuyến cáo sử dụng @NamedQuery

Gọi @NamedQuery

Thực tế giá trị @NamedQuery mang lại không nhiều. Do đó hãy ưu tiên khai báo query trong Repository

```
@NamedQuery(name = "Car.findById", query = "SELECT c FROM oto c WHERE c.id=:id")
```



```
Query namedQuery = em.createNamedQuery("Car.findById");  
namedQuery.setParameter("id", 1L);  
Car car = (Car) namedQuery.getSingleResult();  
System.out.println(car);
```

Untyped Query vs Typed Query

```
Query jpqlQuery = em.createQuery("SELECT o FROM oto o WHERE o.id=:id");  
jpqlQuery.setParameter("id", 1L);
```

```
Car car = Ép kiểu (Car) jpqlQuery.getSingleResult();  
System.out.println(car);
```

```
TypedQuery<Car> typedQuery = em.createQuery("SELECT c FROM oto c WHERE  
c.id=:id", Car.class); Truyền kiểu vào  
typedQuery.setParameter("id", 1L);  
Car car = typedQuery.getSingleResult(); // Khi dùng TypedQuery thì không  
cần ép kiểu  
System.out.println(car);
```

Annotation `@Query` để định nghĩa JPQL trong repository

```
@Query("SELECT o FROM oto AS o WHERE o.year=:year")
List<Car> listCarInYear(@Param("year") int year);

// Phải ghi rõ domain, package của kiểu trả về
vn.techmaster.demojpa.model.mapping.MakerCount
@Query("SELECT new vn.techmaster.demojpa.repository.MakerCount(c.make, COUNT(*))
" +
"FROM oto AS c GROUP BY c.make ORDER BY c.make ASC")
List<MakerCount> countByMaker();

@Query("SELECT new vn.techmaster.demojpa.repository.MakerCount(c.make, COUNT(*))
" +
"FROM oto AS c GROUP BY c.make ORDER BY COUNT(*) DESC")
List<MakerCount> topCarMaker(Pageable pageable);
//Chú ý PSQL không hỗ trợ cú pháp SELECT TOP hay LIMIT, thay vào đó phải truyền
vào Pageable pageable
```

Native Query

Khai báo native query trong Repository

```
@Query(value = "SELECT * FROM car WHERE id=:id", nativeQuery = true)  
List<Car> getCarById(@Param("id") long id);
```

Dùng EntityManager tạo native query

```
Query nativeQuery = em.createNativeQuery("SELECT * FROM car WHERE  
id=:id", Car.class); //Không dùng oto mà dùng car  
nativeQuery.setParameter("id", 1L);  
Car car = (Car) nativeQuery.getSingleResult();  
System.out.println(car);
```


Ưu và nhược điểm khi dùng Native Query mà không dùng JPQL

- Dùng các công cụ Dbeaver, MySQL WorkBench....viết câu lệnh native SQL chạy thử thành công rồi dùng trực tiếp trong code
- Nếu đã thạo SQL từ trước thì việc viết native query đôi khi dễ hơn JPQL
- Việc trả về dữ liệu hoặc ép kiểu từ native query cần khai báo trước cấu trúc dữ liệu phù hợp.

Derived Query

Derived Query là gì?

- Derived Query là cách đặt tên phương thức trong khai báo repository để JPA sinh ra câu lệnh SQL. Luôn gắn vào 1 Repository cụ thể
- Tên phương thức derived query gồm 2 phần: động từ và tham số trường
- Tiện khi truy vấn một bảng
- Không hỗ trợ các lệnh Group By hay Join

findBy

existsBy

countBy

deleteBy



Id(Long id)

FullnameLike(String fullName)

FullNameContainingIgnoreCase(String fullName)

JobAndCity(String job, String city)

Top5ByOrderBySalaryDesc()

```
public interface PersonRepository extends JpaRepository<Person, Long>{
    Optional<Person> findById(Long id);
    List<Person> findByFullnameContainingIgnoreCase(String fullName);
    List<Person> findByFullnameContainingIgnoreCaseAndCity(String fullName, String.
city);
    List<Person> findBySalaryBetweenOrderBySalaryAsc(int from, int to );
    List<Person> findByJobAndCity(String job, String city);
    List<Person> findByBirthdayBeforeOrderByBirthdayDesc(Date date);
    List<Person> findByBirthdayAfterOrderByBirthdayAsc(Date date);
    boolean existsByFullname(String fullname);
    boolean existsByFullnameLike(String fullname);
    List<Person> findTop5ByOrderBySalaryDesc();
    List<Person> findByJob(String job);
    long countByFullnameLikeAndJob(String fullname, String job);
}
```

Các biểu thức phổ biến trong Derived Query

_ là tên biến

- And/Or
- _Like
- _Containing / _ContainingIgnoreCase
- _StartingWith / _EndingWith
- _LessThan / _LessThanEqual
- _GreaterThan / _GreaterThanEqual
- _Between
- _In (trong một collection)
- OrderBy_Asc
- OrderBy_Desc
- _Before / _After
- _IsNull
- _IsNotNull
- _True / False

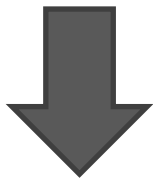
JPA dịch derived query ra SQL như thế nào

```
findByFullnameContainingIgnoreCase(fullname)
```



```
Hibernate: select person0_.id as id1_11_, person0_.birthday as  
birthday2_11_, person0_.city as city3_11_, person0_.fullname  
as fullname4_11_, person0_.gender as gender5_11_, person0_.job  
as job6_11_, person0_.salary as salary7_11_, case when  
person0_.gender='Male' then true else false end as formula1_  
from person person0_ where upper(person0_.fullname) like  
upper(?) escape ?
```

```
findByFullnameContainingIgnoreCaseAndCity(fullname, city)
```



Hibernate: select person0_.id as id1_11_, person0_.birthday as birthday2_11_, person0_.city as city3_11_, person0_.fullname as fullname4_11_, person0_.gender as gender5_11_, person0_.job as job6_11_, person0_.salary as salary7_11_, case when person0_.gender='Male' then true else false end as formula1_ from person person0_ **where**
(upper(person0_.fullname) like upper(?) escape ?) and
person0_.city=?

```
findBySalaryBetweenOrderBySalaryAsc(from, to)
```



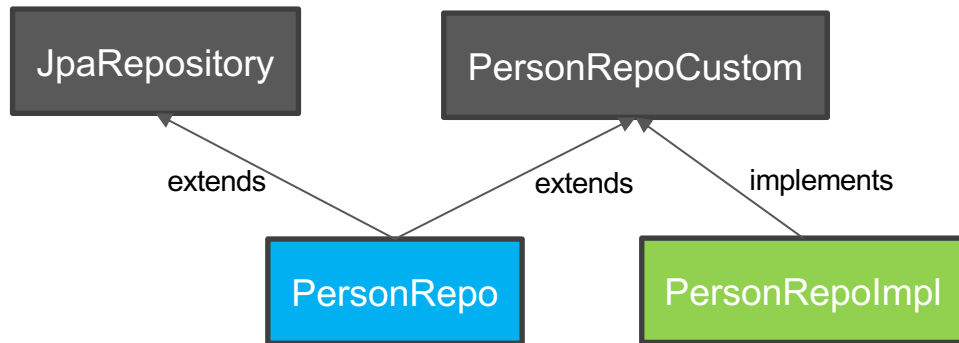
```
select person0_.id as id1_11_, person0_.birthday as birthday2_11_,  
person0_.city as city3_11_, person0_.fullname as fullname4_11_,  
person0_.gender as gender5_11_, person0_.job as job6_11_,  
person0_.salary as salary7_11_, case when person0_.gender='Male' then  
true else false end as formula1_ from person person0_ where  
person0_.salary between ? and ? order by person0_.salary asc
```

Tạo custom repository

Khi nào cần tạo custom repository?

Sau khi đã thử dùng Native Query, Derived Query, JPQL ...nhưng không thể giải quyết được một câu lệnh truy vấn phức tạp hoặc **cần trả về cấu trúc dữ liệu Set, Map, ...** Khi viết custom repository, bạn có thể tận dụng Java Stream, tất cả những sức mạnh cú pháp của Java để thể hiện ý tưởng, thậm chí caching

1. Tạo interface repository **PersonRepo** kế thừa JpaRepository và Custom Repository Interface. Chú ý quy tắc đặt tên **PersonRepoCustom**
2. Hiện thực hoá các phương thức khai báo ở Custom Repository Interface ở **PersonRepoImpl**



@Repository

```
public interface PersonRepo extends JpaRepository<Person, Long>,
    PersonRepoCustom
```

```
public interface PersonRepoCustom {
    TreeMap<String, List<Person>> groupPeopleByOrderCity();
}
```

Trả về kiểu dữ liệu khác với List

```
public class PersonRepoImpl implements PersonRepoCustom {
    @Autowired @Lazy PersonRepo personRepository;

    @PersistenceContext private EntityManager em;

    @Override
    public TreeMap<String, List<Person>> groupPeopleByOrderCity() {
        return personRepository.findAll().stream()
            .collect(Collectors.groupingBy(Person::getCity, TreeMap::new,
Collectors.toList()));
    }
```

Sử dụng Java Stream để biến hoá dữ liệu

Dùng Query gì trong trường hợp nào?

- @NameQuery hãy hạn chế dùng. Nó không tách biệt rõ ràng giữa định nghĩa Entity và Query
- Derived Query viết biểu thức hàm sinh ra lệnh SQL tiện lợi khi truy vấn trong một repository
- Ưu tiên dùng Typed Query hơn là Untyped Query. Đằng nào cũng phải ép kiểu dữ liệu trả về.
- Dùng JPQL query khai báo trong repository sẽ clean code hơn là dùng EntityManager.createQuery
- Nếu phải trả về cấu trúc dữ liệu dạng Set, Map và xử lý phức tạp hãy viết Custom Repository

Join – Primary Key / Foreign Key

Transaction