



Lập trình Backend

Spring Hibernate part 2

Ths. Vũ Duy Khương

1

Giới thiệu JPA , EntityManager

2

Giới thiệu Relationship trong DB

3

Các annotation trong ORM

4

Ví dụ về Hibernate vs Relationship

5

Giới thiệu công cụ GIT,SVN

Giới thiệu JPA , EntityManager

- **JPA là gì:**

- JPA (Java Persistence API) là 1 giao diện lập trình ứng dụng Java, nó mô tả cách quản lý các mối quan hệ dữ liệu trong ứng dụng sử dụng Java Platform.
- JPA cung cấp một mô hình POJO persistence cho phép ánh xạ các table/các mối quan hệ giữa các table trong database sang các class/mối quan hệ giữa các object.

Giới thiệu JPA , EntityManager

- **Một số khái niệm trong JPA:**
- **Entity**: Entity là các đối tượng thể hiện tương ứng 1 table trong cơ sở dữ liệu. Khi lập trình, entity thường là các class POJO đơn giản, chỉ gồm các method getter, setter.



Giới thiệu JPA , EntityManager

- **Một số khái niệm trong JPA:**
- ***EntityManager***: EntityManager là một giao diện (interface) cung cấp các API cho việc tương tác với các Entity như **Persist** (lưu một đối tượng mới), **merge** (cập nhật một đối tượng), **remove** (xóa 1 đối tượng).



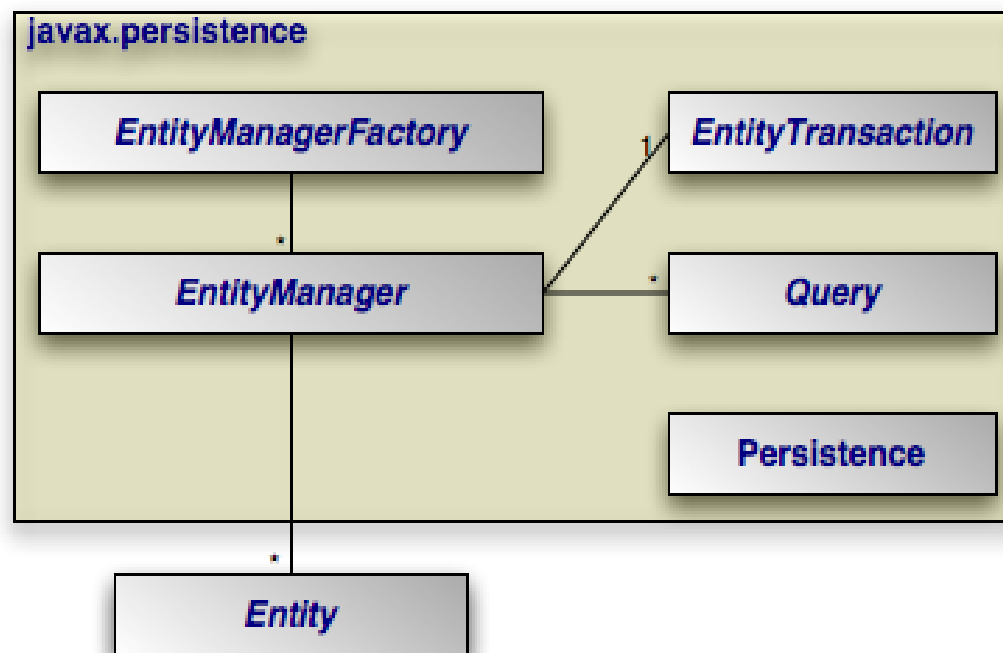
Giới thiệu JPA , EntityManager

- Một số khái niệm trong JPA:
- *EntityManagerFactory*: EntityManagerFactory được dùng để tạo ra một thể hiện của EntityManager.



Giới thiệu JPA , EntityManager

- Kiến trúc JPA:



Giới thiệu JPA , EntityManager

- So sánh Session và EntityManager:

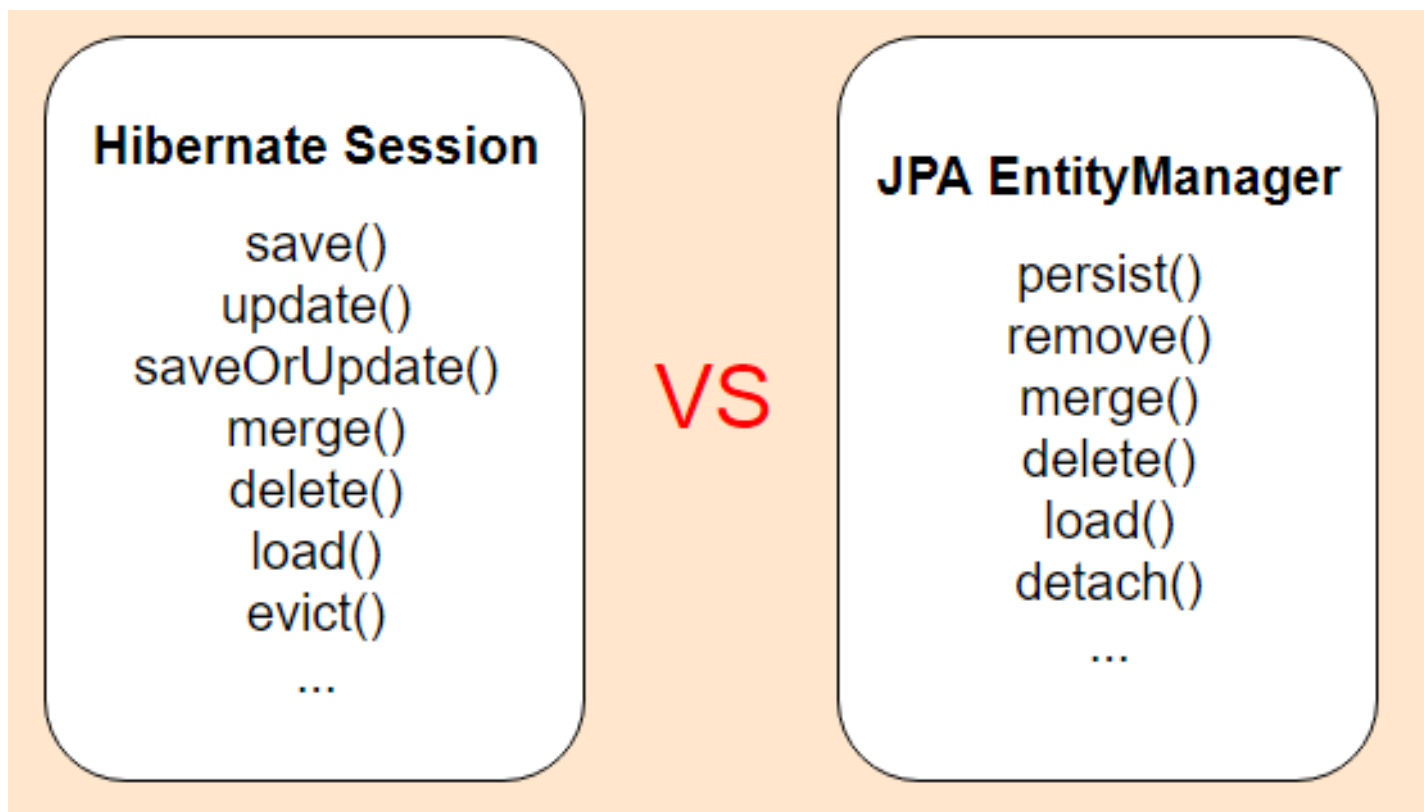
Hibernate là cài đặt của JPA (Hibernate thừa kế JPA).

EntityManager là chuẩn của JPA dùng để thực hiện truy vấn database (thêm, sửa, xóa...). Còn Session chỉ dùng riêng cho Hibernate.

Tất cả các framework ORM thừa kế từ JPA đều có thể sử dụng lại EntityManager (mỗi framework có một cách cài đặt lại khác nhau).

Giới thiệu JPA , EntityManager

- So sánh Session và EntityManager(tiếp):



Giới thiệu JPA , EntityManager

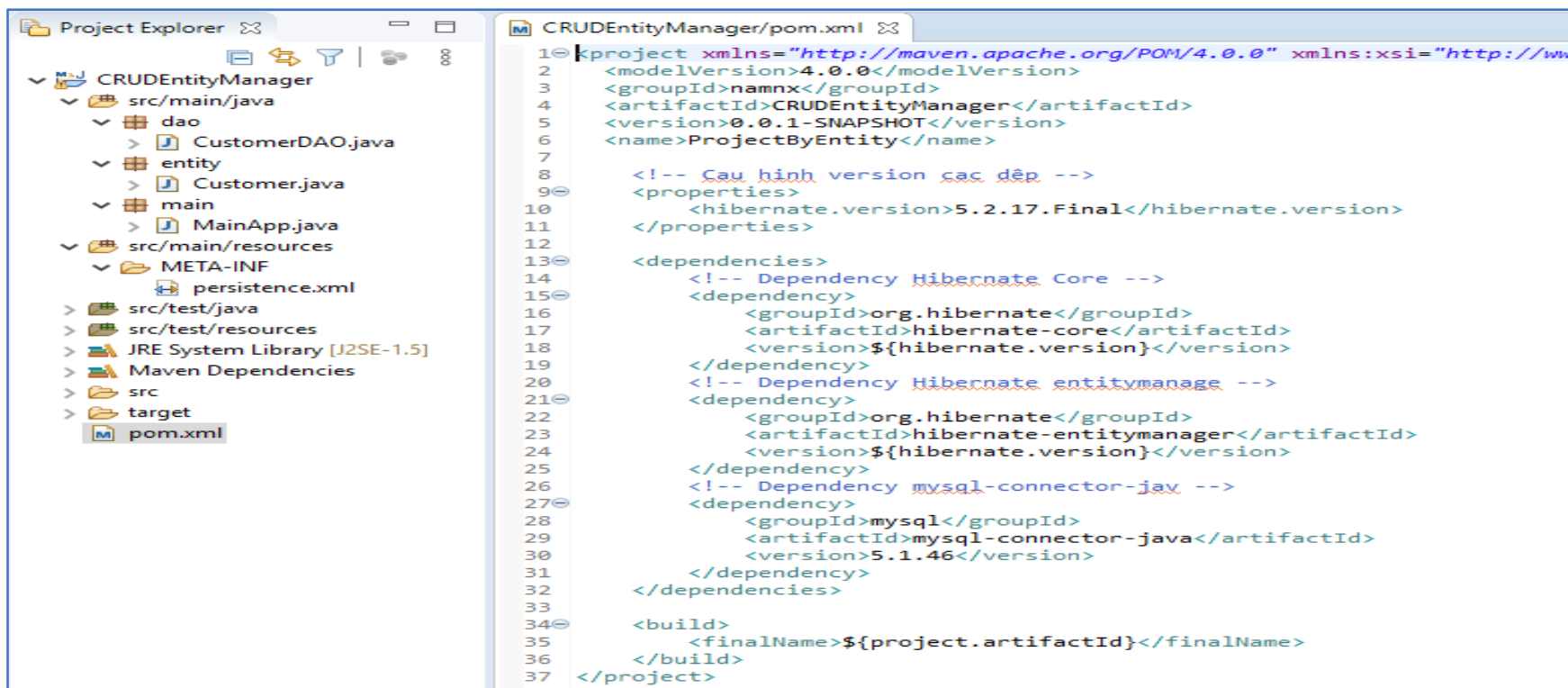
- **Suggest : Nên dùng API theo chuẩn JPA.**
- Các API của JPA có thể dùng lại ở nhiều project khác nhau, nhiều framework khác nhau.
- Các thư viện, kiến trúc đều tập trung vào JPA API
- Bạn vẫn có thể sử dụng Hibernate Session thông qua JPA EntityManager.

Giới thiệu JPA , EntityManager

- **Suggest : Nên dùng API theo chuẩn JPA(tiếp).**
- EntityManager cho phép sử dụng các annotation callback như **@PrePersist, @PostPersist, @PreUpdate**
- Các annotation của JPA có thể làm việc được với Hibernate Session API.

Giới thiệu JPA , EntityManager

- Tạo project sử dụng JPA: Cấu trúc vs pom.xml



The screenshot displays the Project Explorer on the left and the pom.xml file on the right.

Project Explorer Structure:

- CRUDEntityManager
 - src/main/java
 - dao
 - CustomerDAO.java
 - entity
 - Customer.java
 - main
 - MainApp.java
 - src/main/resources
 - META-INF
 - persistence.xml
 - src/test/java
 - src/test/resources
 - JRE System Library [J2SE-1.5]
 - Maven Dependencies
 - src
 - target
 - pom.xml

CRUDEntityManager/pom.xml Content:

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <groupId>namnx</groupId>
5   <artifactId>CRUDEntityManager</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>ProjectByEntity</name>
8
9   <!-- Cấu hình version các dep -->
10  <properties>
11    <hibernate.version>5.2.17.Final</hibernate.version>
12  </properties>
13
14  <dependencies>
15    <!-- Dependency Hibernate Core -->
16    <dependency>
17      <groupId>org.hibernate</groupId>
18      <artifactId>hibernate-core</artifactId>
19      <version>${hibernate.version}</version>
20    </dependency>
21    <!-- Dependency Hibernate entitymanager -->
22    <dependency>
23      <groupId>org.hibernate</groupId>
24      <artifactId>hibernate-entitymanager</artifactId>
25      <version>${hibernate.version}</version>
26    </dependency>
27    <!-- Dependency mysql-connector-java -->
28    <dependency>
29      <groupId>mysql</groupId>
30      <artifactId>mysql-connector-java</artifactId>
31      <version>5.1.46</version>
32    </dependency>
33  </dependencies>
34
35  <build>
36    <finalName>${project.artifactId}</finalName>
37  </build>
38 </project>
  
```

Giới thiệu JPA , EntityManager

- Tạo project sử dụng JPA: persistence.xml

```

1 persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
4     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd" version="2.1"
5 <persistence-unit name="persistence">
6   <description>Demo Hibernate Entity Manager</description>
7   <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
8   <properties>
9     <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
10    <property name="javax.persistence.jdbc.url"
11      value="jdbc:mysql://localhost:3306/hibernatedb" />
12    <property name="javax.persistence.jdbc.user" value="root" />
13    <property name="javax.persistence.jdbc.password" value="root@123" />
14    <property name="hibernate.show_sql" value="true" />
15    <property name="hibernate.hbm2ddl.auto" value="update"/>
16  </properties>
17 </persistence-unit>
18 </persistence>
  
```

Giới thiệu JPA , EntityManager

- Tạo project sử dụng JPA: Entity

```

Customer.java
6+ import java.io.Serializable;
14
15 /**
16  * @author NAM
17  *
18  */
19 @Entity
20 @Table(name = "Customer")
21 public class Customer implements Serializable {
22     private static final long serialVersionUID = 1L;
23     @Id
24     @Column(name = "id")
25     @GeneratedValue(strategy = GenerationType.IDENTITY)
26     private int id;
27     @Column(name = "name")
28     private String name;
29     @Column(name = "address")
30     private String address;
31
32     // Constructor
33     public Customer(String name, String address) {
34         this.name = name;
35         this.address = address;
36     }
37
38     // getter - setter

```

Giới thiệu JPA , EntityManager

- Tạo project sử dụng JPA: DAO file

```

CustomerDAO.java
14 /**
15  * @author NAM
16  *
17  */
18 public class CustomerDAO {
19     //Lấy thông tin từ file META-INF/persistence.xml để tạo đối tượng EntityManagerFactory
20     EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("persistence");
21     //Được dùng để tạo 1 EntityManager
22     EntityManager entityManager = entityManagerFactory.createEntityManager();
23
24     public void save(Customer customer) {
25         entityManager.getTransaction().begin();
26         entityManager.persist(customer); // Lưu object tới đối tượng persist
27         entityManager.getTransaction().commit();
28     }
29
30     public Customer findById(int id) {
31         Customer customer = entityManager.find(Customer.class, id);
32         return customer;
33     }
34
35     public List<Customer> findAll() {
36         // lấy ra 1 list record
37         return entityManager.createQuery("SELECT c FROM Customer c", Customer.class).getResultList();
38     }
39
40     public void delete(Customer customer) {
41         entityManager.getTransaction().begin();
42         entityManager.remove(customer); //Xóa 1 record
43         entityManager.getTransaction().commit();
44     }
45
46     public void close() {
47         entityManager.close();
48         entityManagerFactory.close();
49     }
50 }
51
52

```

Giới thiệu JPA , EntityManager

- Tạo project sử dụng JPA: Main Class

```

MainApp.java
1  /**
4  package main;
5
6  import java.util.List;
10
11 /**
12  * @author NAM
13  *
14  */
15 public class MainApp {
16
17     /**
18     * @param args
19     */
20     public static void main(String[] args) {
21         // TODO Auto-generated method stub
22         CustomerDAO customerDAO = new CustomerDAO();
23         customerDAO.save(new Customer("Kai", "Viet Nam"));
24         customerDAO.save(new Customer("Thanos", "Viet Nam"));
25         customerDAO.save(new Customer("Thor", "Asgard"));
26         customerDAO.save(new Customer("Hulk", "USA"));
27         customerDAO.save(new Customer("Iron Man", "USA"));
28
29         System.out.println("all customer after insert:");
30         List<Customer> listCustomer = customerDAO.findAll();
31         for (Customer customer : listCustomer) {
32             System.out.println(customer.getName());
33         }
34         customerDAO.close();
35     }
36 }
37
38
39

```

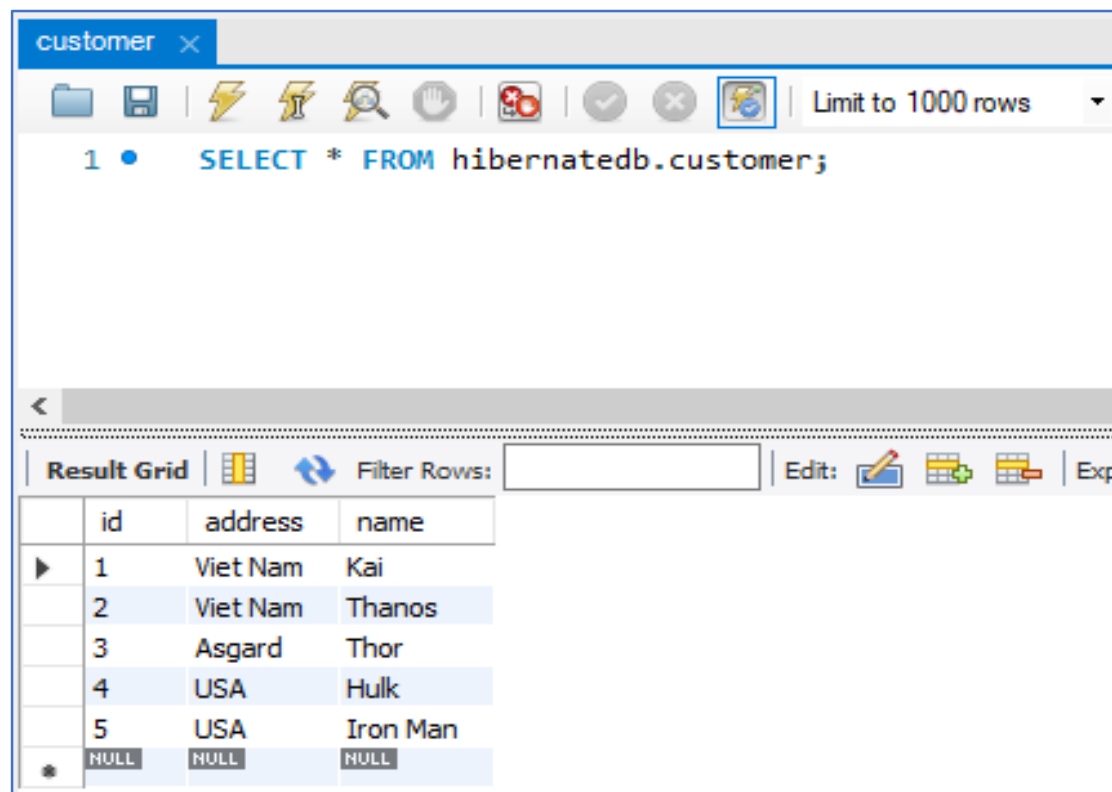

Giới thiệu JPA , EntityManager

- Tạo project sử dụng JPA: Log console

```
Sep 18, 2020 7:52:40 AM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionPro
Hibernate: create table Customer (id integer not null auto_increment, address varchar(255), name varchar(255), primary key (id)) engine=InnoDB
Hibernate: insert into Customer (address, name) values (?, ?)
Hibernate: insert into Customer (address, name) values (?, ?)
Hibernate: insert into Customer (address, name) values (?, ?)
Hibernate: insert into Customer (address, name) values (?, ?)
Hibernate: insert into Customer (address, name) values (?, ?)
all customer after insert:
Sep 18, 2020 7:52:41 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select customer0_.id as id1_0_, customer0_.address as address2_0_, customer0_.name as name3_0_ from Customer customer0_
Kai
Thanos
Thor
Hulk
Iron Man
Sep 18, 2020 7:52:41 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/hibernatedb]
```

Giới thiệu JPA , EntityManager

- Tạo project sử dụng JPA: Kết quả trong DB



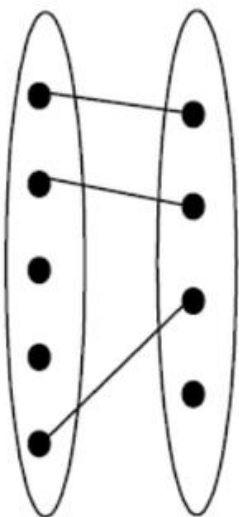
The screenshot shows a database client window titled 'customer'. The SQL query entered is `SELECT * FROM hibernatedb.customer;`. The result is displayed in a table with columns 'id', 'address', and 'name'.

	id	address	name
1	1	Viet Nam	Kai
2	2	Viet Nam	Thanos
3	3	Asgard	Thor
4	4	USA	Hulk
5	5	USA	Iron Man
*	NULL	NULL	NULL

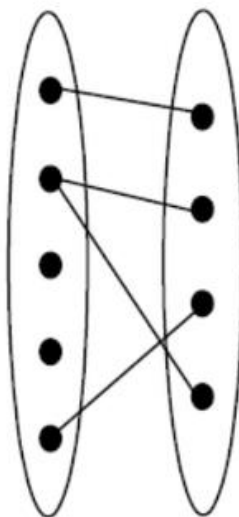
Relationship trong Database

Relationship:

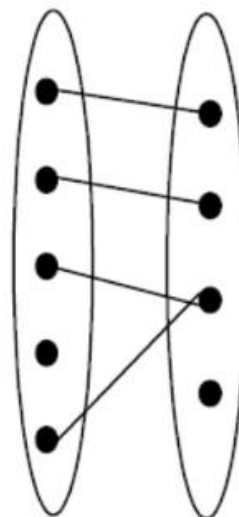
Mối quan hệ, sự liên kết giữa hai hay nhiều table với nhau.



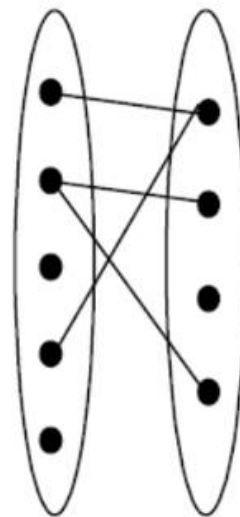
1-to-1



1-to Many



Many-to-1

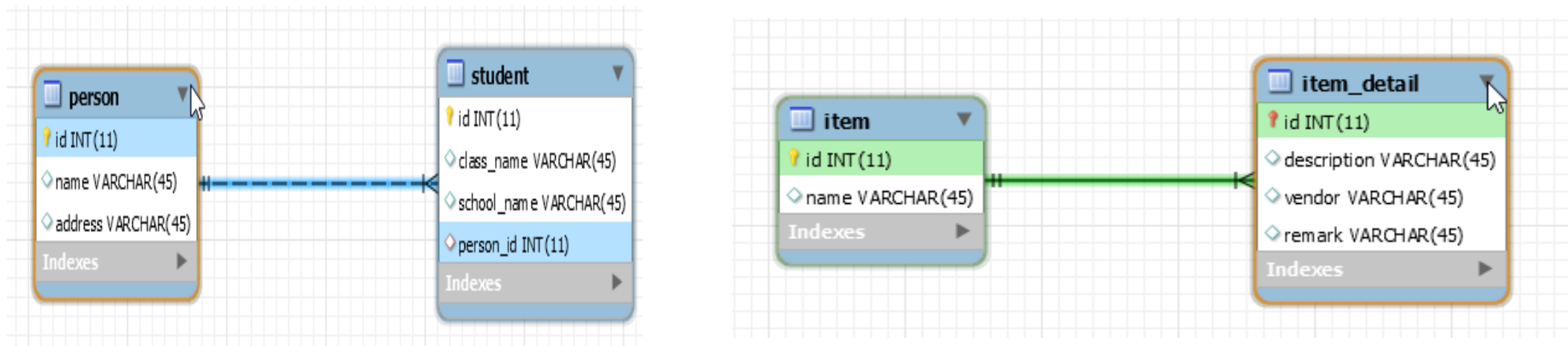


Many-to-Many

Relationship trong Database

Relationship:

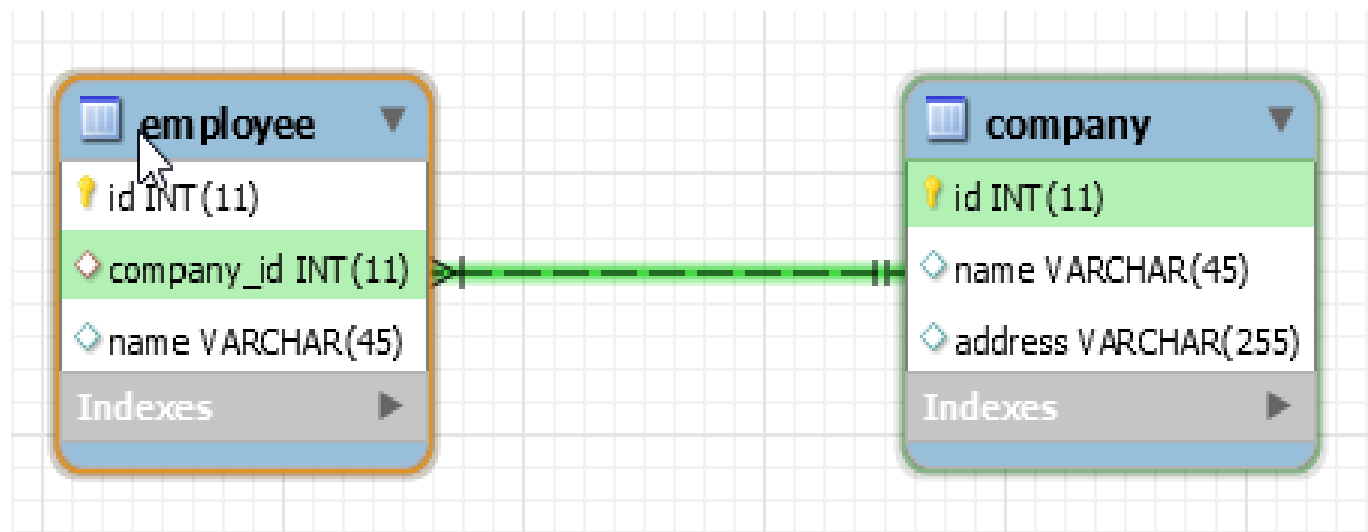
Mối quan hệ 1 – 1: được dùng cho những trường hợp một bản ghi chỉ cho phép duy nhất một bản ghi khác tham chiếu tới nó.



Relationship trong Database

Relationship:

Mối quan hệ 1 – n: được dùng cho những trường hợp một bản ghi của table nguồn được tham chiếu bởi n bản ghi tạo table đích



Relationship trong Database

Relationship:

Mối quan hệ n – 1: được dùng cho trường hợp một đối tượng ở table A có thể liên kết với nhiều đối tượng table B

```
create table EMPLOYEE (  
    id INT NOT NULL auto_increment,  
    first_name VARCHAR(20) default NULL,  
    last_name VARCHAR(20) default NULL,  
    salary INT default NULL,  
    address INT NOT NULL,  
    PRIMARY KEY (id)  
);
```

```
create table ADDRESS (  
    id INT NOT NULL auto_increment,  
    street_name VARCHAR(40) default NULL,  
    city_name VARCHAR(40) default NULL,  
    state_name VARCHAR(40) default NULL,  
    zipcode VARCHAR(10) default NULL,  
    PRIMARY KEY (id)  
);
```

Relationship trong Database

Relationship:

Mối quan hệ $n - n$:

mối quan hệ một nhân viên(employee) có một hoặc nhiều chứng chỉ(certificate). Và ngược lại một chứng chỉ(certificate) có một hoặc nhiều nhân viên(employee).

Relationship trong Database

Relationship:

Mối quan hệ n – n (tiếp):

```
create table EMPLOYEE (  
    id INT NOT NULL auto_increment,  
    first_name VARCHAR(20) default NULL,  
    last_name  VARCHAR(20) default NULL,  
    salary     INT default NULL,  
    PRIMARY KEY (id)  
);
```

```
create table CERTIFICATE (  
    id INT NOT NULL auto_increment,  
    certificate_name VARCHAR(30) default NULL,  
    PRIMARY KEY (id)  
);
```


Relationship trong Database

Relationship:

Mối quan hệ n – n (tiếp):

Chúng ta sẽ phải tạo một bảng trung gian chứa ID của Employee và ID của Certificate

```
create table EMP_CERT (  
    employee_id INT NOT NULL,  
    certificate_id INT NOT NULL,  
    PRIMARY KEY (employee_id,certificate_id)  
);
```

Relationship trong Database

Relationship:

Mối quan hệ $n - n$ (tiếp):

VD2 :

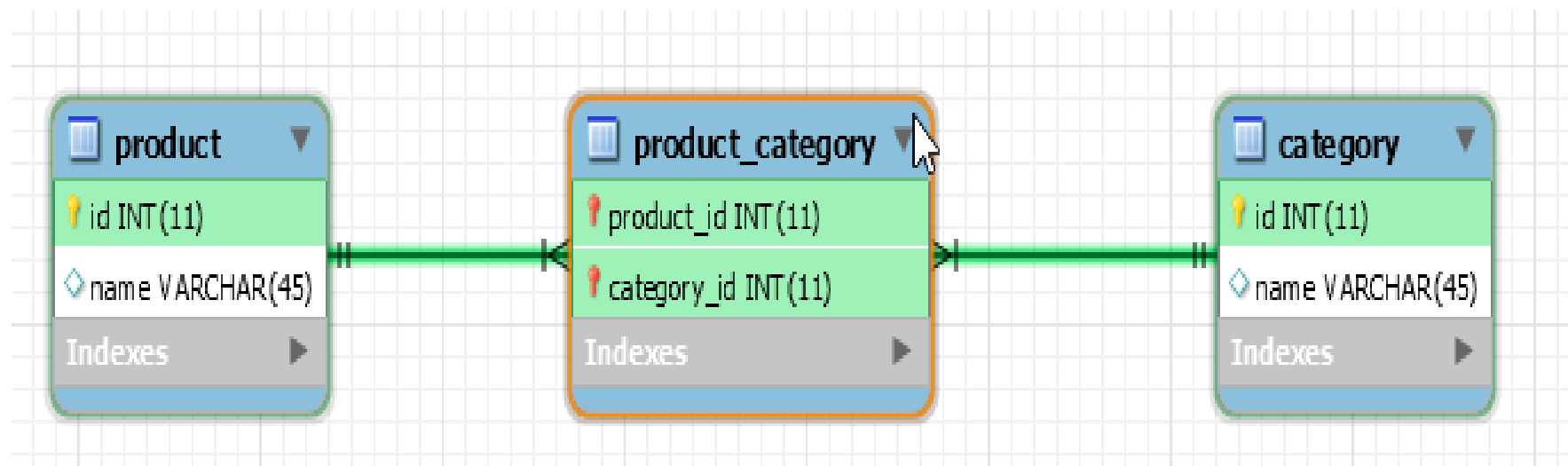
Ví dụ mình có 1 bảng product (sản phẩm) và 1 bảng category (danh mục). Quan hệ giữa hai bảng này là nhiều – nhiều vì 1 sản phẩm có thể thuộc nhiều danh mục khác nhau, một danh mục cũng có thể chứa nhiều sản phẩm khác nhau.

Relationship trong Database

Relationship:

Mối quan hệ n – n (tiếp):

VD2 :



Các Annotation trong Hibernate

Class Employee

```
import javax.persistence.*;

@Entity
@Table(name = "EMPLOYEE")
public class Employee {
    @Id @GeneratedValue
    @Column(name = "id")
    private int id;

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

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "salary")
    private int salary;

    public Employee() {}
}
```

Các Annotation trong Hibernate

@Entity Annotation:

Chúng ta đã sử dụng chú thích Entity vào lớp Employee đánh dấu lớp này như một Entity Bean, do đó nó phải có một constructor không có đối số mà có kiểu là public.

Các Annotation trong Hibernate

@Table Annotation:

Chú thích @Table cho phép bạn chỉ định các chi tiết của bảng sẽ được sử dụng để lưu trữ thực thể trong cơ sở dữ liệu.

Các Annotation trong Hibernate

@Id and @GeneratedValue Annotation:

Mỗi entity bean sẽ có một khóa chính, mà bạn chú thích trên lớp với chú thích @Id. Khóa chính có thể là một trường duy nhất hoặc kết hợp nhiều trường tùy thuộc vào cấu trúc bảng của bạn.

Các Annotation trong Hibernate

@Id and @GeneratedValue Annotation(tiếp):

@Id sẽ tự động xác định chiến lược tạo primary key, nhưng bạn có thể ghi đè bằng cách áp dụng chú thích @GeneratedValue có hai tham số **strategy** và **generator**. Chúng ta chỉ sử dụng chiến lược tạo key mặc định. Cho phép Hibernate xác định loại kiểu tạo nào để sử dụng làm cho mã di chuyển giữa các cơ sở dữ liệu khác nhau.

Các Annotation trong Hibernate

@Column Annotation:

Chú thích @Column được sử dụng để chỉ định chi tiết của cột mà trường hoặc thuộc tính sẽ được ánh xạ

Các Annotation trong Hibernate

@Column Annotation(tiếp):

Bạn có thể sử dụng chú thích cột với các thuộc tính được sử dụng phổ biến nhất sau đây:

1. Thuộc tính **name** được sử dụng để chỉ định tên cột nào trong db map với tên trường được chú thích.
2. Thuộc tính **unique** cho phép cột được đánh dấu chỉ chứa các giá trị duy nhất.

Các Annotation trong Hibernate

@Column Annotation(tiếp):

Bạn có thể sử dụng chú thích cột với các thuộc tính được sử dụng phổ biến nhất sau đây:

1. Thuộc tính **length** cho phép kích thước của cột được sử dụng để ánh xạ một giá trị đặc biệt cho một giá trị String.
2. Thuộc tính **nullable** cho phép cột được đánh dấu KHÔNG NULL khi schema được tạo ra.

Ví dụ về Hibernate vs Relationship

Ví dụ quan hệ 1 - 1 :

- Cho 2 table quan hệ với nhau 1 – 1:

```
CREATE TABLE `hibernatedb`.`person` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(45) NULL,  
  `address` VARCHAR(45) NULL,  
  PRIMARY KEY (`id`));  
  
CREATE TABLE `hibernatedb`.`student` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `class_name` VARCHAR(45) NULL,  
  `school_name` VARCHAR(45) NULL,  
  `person_id` INT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE INDEX `person_id_UNIQUE` (`person_id` ASC),  
  CONSTRAINT `student-person`  
    FOREIGN KEY (`person_id`)  
    REFERENCES `hibernatedb`.`person` (`id`));
```

Ví dụ về Hibernate vs Relationship

Ví dụ quan hệ 1 -1 :

- Yêu cầu 1 :
Insert 1 đối tượng person và 1 đối tượng student.
- Yêu cầu 2 :
Select đối tượng 1-1

Ví dụ về Hibernate vs Relationship

Ví dụ quan hệ 1 - n :

- Cho 2 table quan hệ với nhau 1 – n:

```
CREATE TABLE `hibernatedb`.`person` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(45) NULL,  
  `address` VARCHAR(45) NULL,  
  PRIMARY KEY (`id`));  
  
CREATE TABLE `hibernatedb`.`student` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `class_name` VARCHAR(45) NULL,  
  `school_name` VARCHAR(45) NULL,  
  `person_id` INT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE INDEX `person_id_UNIQUE` (`person_id` ASC),  
  CONSTRAINT `student-person`  
    FOREIGN KEY (`person_id`)  
    REFERENCES `hibernatedb`.`person` (`id`));
```

Ví dụ về Hibernate vs Relationship

Ví dụ quan hệ 1 - n :

- Yêu cầu :

Thực hiện insert 1 đối tượng Company và insert 2 đối tượng Customer cho đối tượng Company đó.

Ví dụ về Hibernate vs Relationship

Ví dụ quan hệ n - n :

- Cho 2 table quan hệ với nhau n – n:

```
# nhiều nhiều (n - n)
CREATE TABLE `hibernatedb`.`product` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NULL,
  PRIMARY KEY (`id`));

CREATE TABLE `hibernatedb`.`category` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NULL,
  PRIMARY KEY (`id`));

CREATE TABLE `hibernatedb`.`product_category` (
  `product_id` INT NOT NULL,
  `category_id` INT NOT NULL,
  PRIMARY KEY (`product_id`, `category_id`),
  CONSTRAINT `fk-category`
    FOREIGN KEY (`category_id`)
      REFERENCES `hibernatedb`.`category` (`id`),
  CONSTRAINT `fk-product`
    FOREIGN KEY (`product_id`)
      REFERENCES `hibernatedb`.`product` (`id`));
```


Ví dụ về Hibernate vs Relationship

Ví dụ quan hệ n - n :

- Cho 2 table quan hệ với nhau n – n:
- Yêu cầu :
 Insert 1 category với nhiều product

Giới thiệu công cụ GIT,SVN

Giới thiệu GIT :

Git là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS), nó là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. Git cung cấp cho mỗi lập trình viên kho lưu trữ (repository) riêng chứa toàn bộ lịch sử thay đổi.

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

1. Branch

Các **Branch** (nhánh) đại diện cho các **phiên bản cụ thể** của một kho lưu trữ tách ra từ project chính của bạn.

Branch cho phép bạn theo dõi các thay đổi thử nghiệm bạn thực hiện đối với kho lưu trữ và có thể hoàn nguyên về các phiên bản cũ hơn.

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

2. Commit

Một commit đại diện cho một thời điểm cụ thể trong lịch sử dự án của bạn. Sử dụng lệnh commit kết hợp với lệnh **git add** để cho git biết những thay đổi bạn muốn lưu vào local repository.

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

3. Checkout

Sử dụng lệnh git checkout để chuyển giữa các branch. Chỉ cần nhập git checkout theo sau là tên của branch bạn muốn chuyển đến hoặc nhập git checkout master để trở về branch chính (master branch).

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

4. Fetch

Lệnh git fetch tìm nạp các bản sao và tải xuống tất cả các tệp branch vào máy tính của bạn. Sử dụng nó để lưu các thay đổi mới nhất vào kho lưu trữ của bạn. Nó có thể tìm nạp nhiều branch cùng một lúc.

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

4. Fetch

Lệnh git fetch tìm nạp các bản sao và tải xuống tất cả các tệp branch vào máy tính của bạn. Sử dụng nó để lưu các thay đổi mới nhất vào kho lưu trữ của bạn. Nó có thể tìm nạp nhiều branch cùng một lúc.

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

5. Head

Các commit ở đầu của một branch được gọi là head. Nó đại diện cho commit mới nhất của repository mà bạn hiện đang làm việc.

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

6. Index

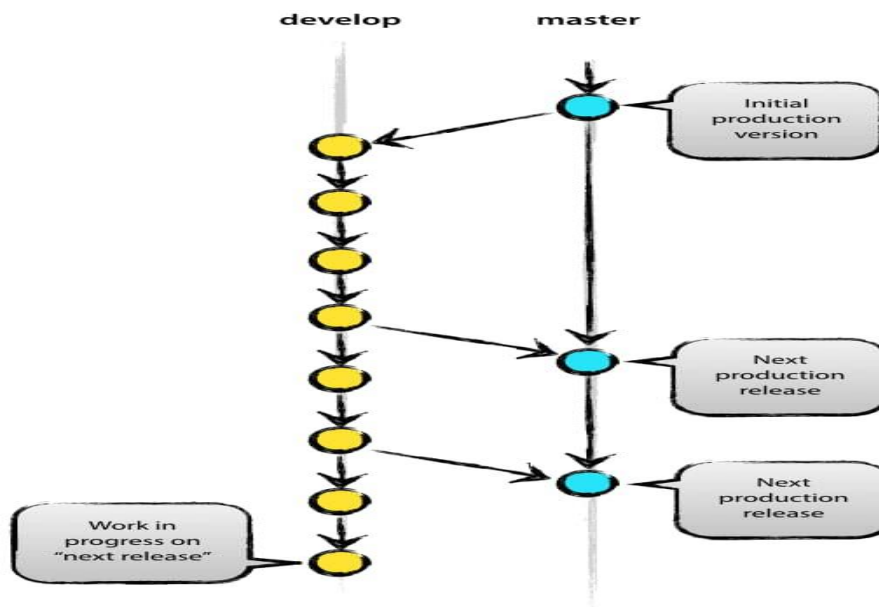
Bất cứ khi nào bạn thêm, xóa hoặc thay đổi một file, nó vẫn nằm trong chỉ mục cho đến khi bạn sẵn sàng commit các thay đổi. Nó như là khu vực tổ chức (staging area) cho Git. Sử dụng lệnh `git status` để xem nội dung của index của bạn.

Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

7. Master

Master là nhánh chính của tất cả các repository của bạn. Nó nên bao gồm những thay đổi và commit gần đây nhất.



Giới thiệu công cụ GIT,SVN

Các thuật ngữ Git quan trọng:

8. Merge

Lệnh git merge kết hợp với các yêu cầu kéo (pull requests) để thêm các thay đổi từ nhánh này sang nhánh khác.

Giới thiệu công cụ GIT,SVN

Các thao tác Git quan trọng:

git clone

Tác dụng: Copy 1 git repository từ remote source.

git clone <:clone git url:>

git status

Tác dụng: Để check trạng thái của những file bạn đã thay đổi trong thư mục làm việc. VD: Tất cả các thay đổi cuối cùng từ lần commit cuối cùng.

git status

Giới thiệu công cụ GIT,SVN

Các thao tác Git quan trọng:

git add

Tác dụng: Thêm thay đổi đến stage/index trong thư mục làm việc.

git add

Giới thiệu công cụ GIT,SVN

Các thao tác Git quan trọng:

git commit

Tác dụng: commit nghĩa là một action để Git lưu lại một snapshot của các sự thay đổi trong thư mục làm việc. Và các tập tin, thư mục được thay đổi đã phải nằm trong Staging Area. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của code kèm theo tên và địa chỉ email của người commit.

git commit -m "Đây là message, bạn dùng để note những thay đổi để sau này dễ dò lại"

Giới thiệu công cụ GIT,SVN

Các thao tác Git quan trọng:

git push/git pull

Tác dụng: Push hoặc Pull các thay đổi đến remote. Nếu bạn đã added và committed các thay đổi và bạn muốn đẩy nó lên hoặc remote của bạn đã update và bạn apply tất cả thay đổi đó trên code của mình.

git pull <:remote:> <:branch:> and git push <:remote:> <:branch:>

Giới thiệu công cụ GIT,SVN

Các thao tác Git quan trọng:

git checkout

Tác dụng: Chuyển sang branch khác

git checkout <: branch:>

git merge

Tác dụng: Merge 2 branch lại với nhau.

git merge <:branch_ban_muon_merge:>

Q&A





THANK YOU