

# Hibernate Mapping

Design by: DieuNT1



# Lesson Objectives

1

- Understand the **basic annotations** be used in hibernate.

2

- Understand the **type of hibernate relationship**.

3

- Able to use **annotation in hibernate mapping**.

4

- Understand the **Composite key** and how to implement it.

5

- Able to distinguish **Lazy loading** and **Eager loading**.

- ❖ Basic Annotations
- ❖ Hibernate Relationships
- ❖ Collection Mapping
- ❖ Lazy loading and Eager loading

Lecture 01

# BASIC ANNOTATIONS

- ❖ **@Entity**: marks a class as an entity bean.
- ❖ **@Table**: allows you to specify the details of the table that will be used to persist the entity in the database.
  - ✓ **catalog, schema**: `catalogs` and `schemas` are "namespaces" that you define on the server side of the database. Some databases contains schemas, some contains catalogs, and some contains both.
  - ✓ `indexes = { @Index(name = "IDX_MYIDX1", columnList = "id, name,surname") }`
  - ✓ `uniqueConstraints = { @UniqueConstraint(columnNames = "stock_name"),  
@UniqueConstraint(columnNames = "stock_code") }`
- ❖ **@Id**: each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation.

- ❖ **@GeneratedValue**: Let database generate (auto-increment) the id column.
  - ✓ **strategy** = GenerationType.*IDENTITY*
  - ✓ **strategy** = GenerationType.**AUTO**
  - ✓ **generator** uses **sequences object** if they're supported by our database, and switches to table generation if they aren't.
- ❖ **@Column**: used to specify the details of the column to which a field or property will be mapped.
  - ✓ **name** attribute permits the name of the column to be explicitly specified.
  - ✓ **length** attribute permits the size of the column used to map a value particularly for a String value.
  - ✓ **columnDefinition**: defines a column include data type, null or not.

```
Ex: @Column(name = "price_open", columnDefinition = "NUMERIC(12,2) NULL",  
         unique = false, nullable = true)
```

- ❖ **@Temporal**: This annotation must be specified for persistent fields or properties of type **java.util.Date** and **java.util.Calendar**. It may only be specified for fields or properties of these types.

Ex: `@Temporal(TemporalType.DATE)`

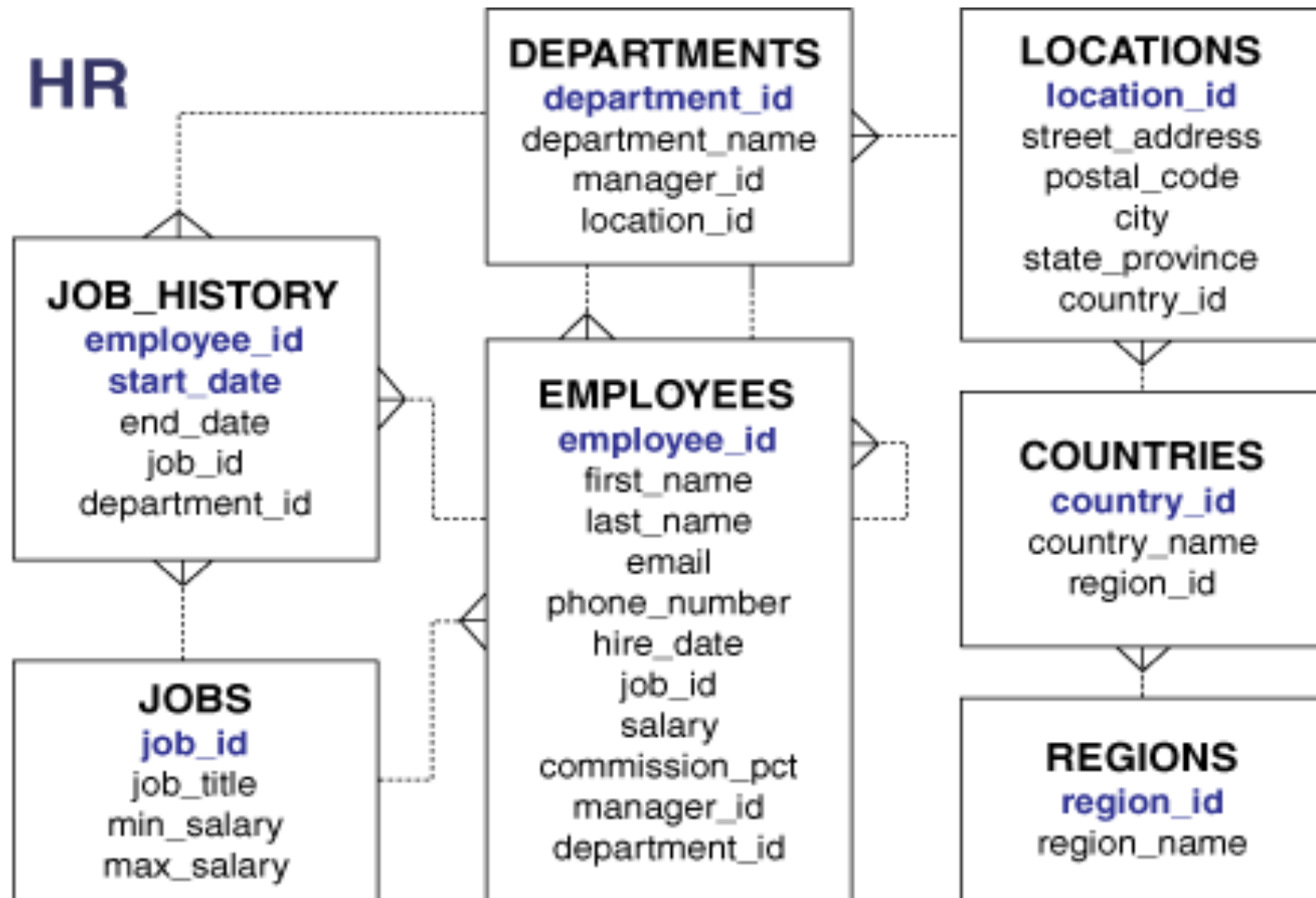
```
private java.util.Date creationDate;
```

- ❖ **@OrderBy**: Sort your data using **@OrderBy** annotation. In example below, it will sort all contacts in a company by their firstname in ascending order.

Ex: `@OrderBy("first_name ASC")`

```
private Set<Contact> contacts; // first_name
```

# Database sample





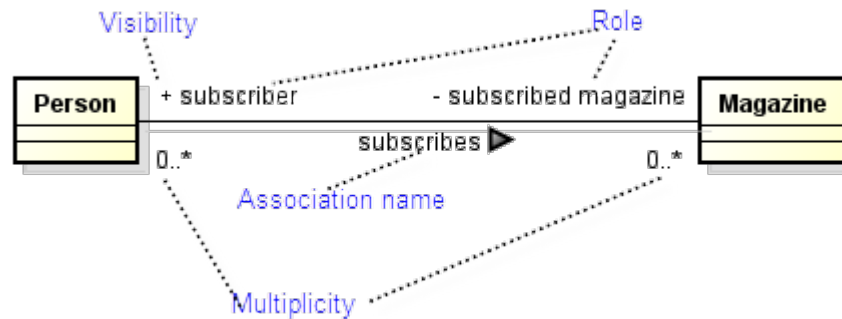
## Section 02

# HIBERNATE RELATIONSHIPS

❖ Represents the static **relationship** shared among the objects of two classes.

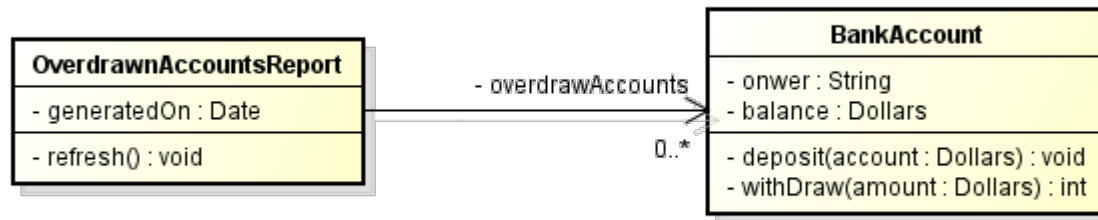
## ❖ Bi-directional:

✓ both classes are aware of each other and their relationship



## ❖ Uni-directional:

✓ two classes are related, but only one class knows that the relationship exists (overdrawn: thấu chi, counselor: nhân viên)



- ❖ **Hibernate mapping** is one of the essential features of Hibernate: “*how the objects of persistent classes are associated with each other*”. Hibernate provides four types of association mapping:

- ✓ One To One



- ✓ One To Many



- ✓ Many To One



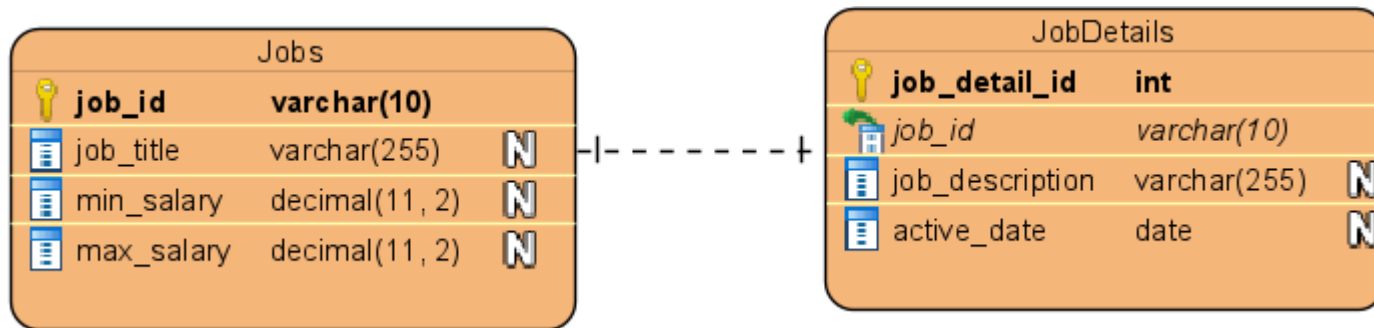
- ✓ Many To Many



# One-to-One Mapping Annotation

## @JoinColumn

- ❖ The most widely used and uses a **foreign key column in one of the tables.**
- ❖ Use **@OneToOne** **mappedBy** and **@JoinColumn** & attribute when foreign key is held by one of the entities.
- ❖ **Example:**



```
@OneToOne(mappedBy = "job")  
private JobDetails jobDetail;
```

```
@OneToOne  
@JoinColumn(name = "job_id")  
private Jobs job;
```

# One-to-One Mapping Annotation

## @JoinColumn

```
@Entity
@Table(schema = "dbo", name = "Jobs",
        indexes = {@Index(columnList = "job_id, job_title", name = "IDX_ID_TITLE") })
public class Jobs {

    @Id
    @Column(name = "job_id", length = 10)
    private String jobId;

    @Column(name = "job_title", length = 255, nullable = false, unique = true)
    private String jobTitle;

    @Column(name = "min_salary", precision = 11, scale = 2)
    private double minSalary;

    @Column(name = "max_salary", precision = 11, scale = 2)
    private double maxSalary;

    @OneToOne(mappedBy = "job")
    private JobDetails jobDetail;

    public Jobs() {
    }

    // Constructors with params
    // getter and setter methods
    public JobDetails getJobDetail() {
        return jobDetail;
    }

    public void setJobDetails(JobDetails jobDetail) {
        this.jobDetail = jobDetail;
    }
}
```

# One-to-One Mapping Annotation

## @JoinColumn

```
@Entity
@Table(name = "JobDetails", schema = "dbo")
public class JobDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "job_detail_id")
    private int jobDetailId;

    @Column(name = "job_description", length = 255)
    private String jobDescription;

    @Column(name = "active_date")
    private LocalDate activeDate;

    @OneToOne
    @JoinColumn(name = "job_id", referencedColumnName = "job_id")
    private Jobs job;

    public JobDetails() {
    }

    // Constructors with params
    // getter and setter methods

    public Jobs getJob() {
        return job;
    }

    public void setJob(Jobs job) {
        this.job = job;
    }
}
```

- ❖ “**cascade**” attribute: An entity defines “**cascade=CascadeType.ALL**” and it essentially means that **any change happened on Jobs must cascade to JobDetails as well**.
  - ✓ If you save a Job, then a JobDetail will also be saved into database.
  - ✓ If you delete a Job then a JobDetail associated with that Job also be deleted.
- ❖ If we only want to cascade **one of operations**, then we can use one of cascade types as below:
  1. **CascadeType.PERSIST** : cascade type `persist` means that `save()` or `persist()` operations cascade to related entities.
  2. **CascadeType.MERGE** : cascade type `merge` means that related entities are merged when the owning entity is merged.
  3. **CascadeType.REFRESH** : cascade type `refresh` does the same thing for the `refresh()` operation.
  4. **CascadeType.REMOVE** : cascade type `remove` removes all related entities association with this setting when the owning entity is deleted.
  5. **CascadeType.DETACH** : cascade type `detach` detaches all related entities if a “manual detach” occurs.
  6. **CascadeType.ALL** : cascade type `all` is shorthand for all of the above cascade operations.

### ❖ Update **Jobs** class:

```
@Entity
@Table(schema = "dbo", name = "Jobs",
    indexes = {@Index(columnList = "job_id, job_title", name = "IDX_ID_TITLE") })
public class Jobs {

    // ...

    @OneToOne(cascade = CascadeType.ALL, mappedBy = "job")
    private JobDetails jobDetail;

    public Jobs() {

    }

    // Constructors with params
    // getter and setter methods

    public JobDetails getJobDetail() {
        return jobDetail;
    }

    public void setJobDetails(JobDetails jobDetail) {
        this.jobDetail = jobDetail;
    }

}
```



### ❖ Create a **JobDao** class:

```
public class JobDaoImpl implements JobDao {

    @Override
    public boolean save(Jobs job) throws Exception {
        Session session = null;
        Transaction transaction = null;

        try {
            session = HibernateUtils.getSessionFactory().openSession();
            transaction = session.beginTransaction();

            Serializable result = session.save(job);

            transaction.commit();

            return (result != null);

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
}
```

# One-to-One Mapping Annotation

## @JoinColumn

```
class JobDaoTest {
    static JobDao jobDao;

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        jobDao = new JobDaoImpl();
    }

    @Test
    void testSave1() throws Exception {
        JobDetails jobDetail = new JobDetails("Java Developer Level 1", LocalDate.of(2020, 9, 1));
        Jobs job = new Jobs("J01", "Java Dev1", 1000, 2000);

        job.setJobDetails(jobDetail);
        jobDetail.setJob(job);

        assertEquals(true, jobDao.save(job));
    }
}
```

Oct 05, 2020 10:16:35 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService  
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]  
Hibernate: select next value for hibernate\_sequence  
Hibernate: insert into dbo.Jobs (job\_title, max\_salary, min\_salary, job\_id) values (?, ?, ?, ?)  
Hibernate: insert into dbo.JobDetails (active\_date, job\_id, job\_description, job\_detail\_id) values (?, ?, ?, ?)

## Results:

Results		Messages		
	job_id	job_title	max_salary	min_salary
1	J01	Java Dev1	2000	1000

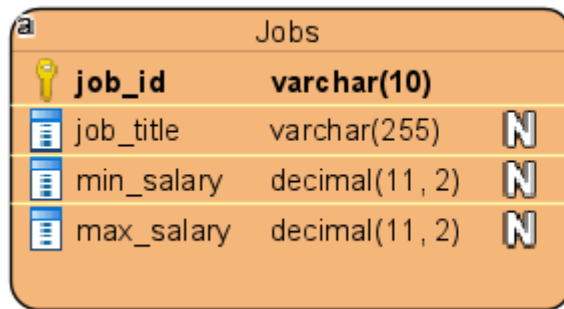
  

	job_detail_id	active_date	job_description	job_id
1	14	2020-09-01	Java Developer Level 1	J01

# One-to-One Mapping Annotation

@PrimaryKeyJoinColumn

- ❖ A technique is something new which uses a **common primary key value in both the tables.**
- ❖ Use **@OneToOne mappedBy** and **@PrimaryKeyJoinColumn** for associated entities sharing the same primary key.
- ❖ **Example:**



@OneToOne

@PrimaryKeyJoinColumn

```
private JobDetails jobDetail;
```

@OneToOne(mappedBy = "userDetails")

```
private Jobs job;
```

# One-to-One Mapping Annotation

## @PrimaryKeyJoinColumn

```
@Entity
@Table(schema = "dbo", name = "Jobs", indexes = {@Index(columnList = "job_id, job_title", name = "IDX_ID_TITLE") })
public class Jobs {

    @Id
    @Column(name = "job_id", length = 10)
    private String jobId;

    @Column(name = "job_title", length = 255, nullable = false, unique = true)
    private String jobTitle;

    @Column(name = "min_salary", precision = 11, scale = 2)
    private double minSalary;

    @Column(name = "max_salary", precision = 11, scale = 2)
    private double maxSalary;

    @OneToOne (cascade = CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private JobDetails jobDetail;

    public Jobs() {

    }
    // Constructors with params
    // getter and setter methods
    public JobDetails getJobDetail() {
        return jobDetail;
    }

    public void setJobDetails(JobDetails jobDetail) {
        this.jobDetail = jobDetail;
    }
}
```

# One-to-One Mapping Annotation

## @PrimaryKeyJoinColumn

```
@Entity
@Table(name = "JobDetails", schema = "dbo")
public class JobDetails {

    @Id
    @GeneratedValue(generator = "foreignngen")
    @GenericGenerator(parameters = {
        @Parameter(name = "property", value = "job"),
    }, strategy = "foreign", name = "foreignngen")
    @Column(name = "job_id")
    private String jobDetailId;

    @Column(name = "job_description", length = 255)
    private String jobDescription;

    @Column(name = "active_date")
    private LocalDate activeDate;

    @OneToOne(mappedBy = "jobDetail")
    private Jobs job;

    public JobDetails() {
    }
    // Constructors with params
    // getter and setter methods

    public Jobs getJob() {
        return job;
    }

    public void setJob(Jobs job) {
        this.job = job;
    }
}
```

# One-to-One Mapping Annotation

@PrimaryKeyJoinColumn

- ❖ Re-run the above test case script, we get the following result:

Results		Messages		
	job_id	job_title	max_salary	min_salary
1	J01	Java Dev1	2000	1000

	job_id	active_date	job_description
1	J01	2020-09-01	Java Developer Level 1

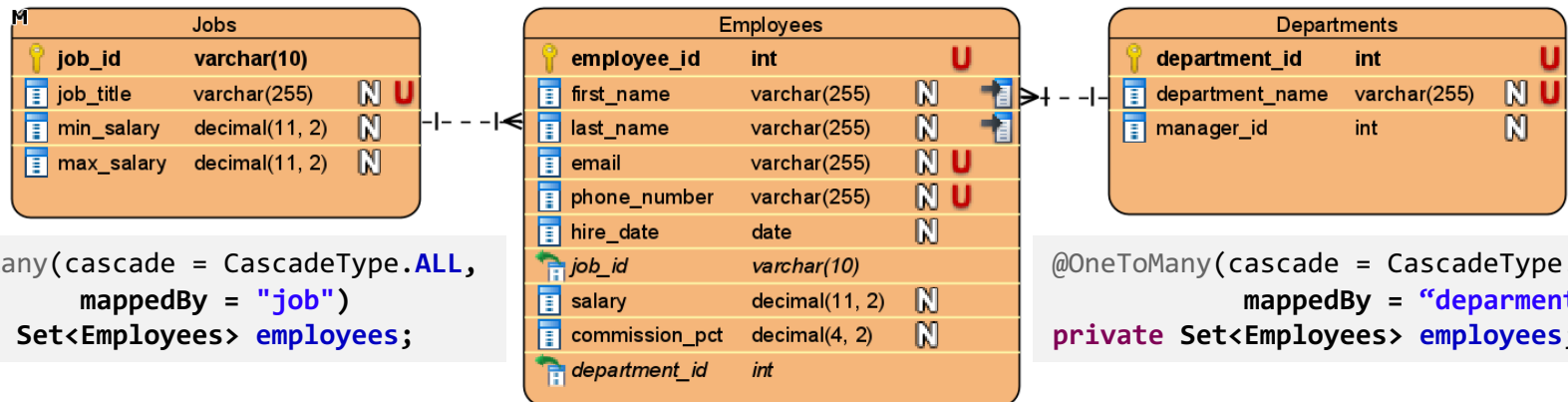
## Section 03

# COLLECTION MAPPING

# One-to-Many Mapping Annotation

## @JoinColumn

- ❖ The most widely used and uses a **foreign key column** in one of the tables.
- ❖ Use **@OneToOne** **mappedBy** and **@JoinColumn** & attribute when foreign key is held by one of the entities.
- ❖ **Example:**



```
@OneToMany(cascade = CascadeType.ALL,  
          mappedBy = "job")  
private Set<Employees> employees;
```

```
@OneToMany(cascade = CascadeType.ALL,  
          mappedBy = "department")  
private Set<Employees> employees;
```

```
@ManyToOne  
@JoinColumn(name = "job_id", columnDefinition = "job_id",  
            referencedColumnName = "job_id")  
private Jobs job;
```



# One-to-Many Mapping Annotation

## @JoinColumn

- ❖ Update `Jobs` class:

```
@Entity
@Table(schema = "dbo", name = "Jobs", indexes = {
    @Index(columnList = "job_id, job_title", name = "IDX_ID_TITLE") })
public class Jobs {
    // ...

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "job")
    private Set<Employees> employees;

    public Set<Employees> getEmployees() {
        return employees;
    }

    public void setEmployees(Set<Employees> employees) {
        this.employees = employees;
    }
}
```

# One-to-Many Mapping Annotation

## @JoinColumn

```
@Entity
@Table(name = "Employees", schema = "dbo", indexes =
    {@Index(columnList = "first_name, last_name",
        name = "IDX_EMP_NAME") })
public class Employees {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "employee_id")
    private int employeeId;

    @Column(name = "first_name", length = 255, nullable = false)
    private String first_name;

    @Column(name = "last_name", length = 255, nullable = false)
    private String last_name;

    @Column(name = "email", length = 255, unique = true)
    private String email;

    @Column(name = "phone_number", length = 255, unique = true)
    private String phoneNumber;

    @Column(name = "hire_date")
    private LocalDate hireDate;

    private double salary;

    @Column(name = "commission_pct")
    private double commissionPct;

    @ManyToOne
    @JoinColumn(name = "job_id", columnDefinition = "job_id",
        referencedColumnName = "job_id")
    private Jobs job;

    public Employees() {

    }
}
```

```
public Jobs getJob() {
    return job;
}

public void setJob(Jobs job) {
    this.job = job;
}
}
```

❖ Create a **EmployeeDaoImpl** class:

```
public class EmployeeDaoImpl implements EmployeeDao{

    @Override
    public boolean save(Employees employee) throws Exception {
        Session session = null;
        Transaction transaction = null;

        try {
            session = HibernateUtils.getSessionFactory().openSession();
            transaction = session.beginTransaction();

            Serializable result = session.save(employee);

            transaction.commit();

            return (result != null);

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
}
```

- ❖ Create a Unit Test Script to test the above `EmployeeDaoImpl` class:

```
class EmployeeDaoTest {
    static EmployeeDao employeeDao;

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        employeeDao = new EmployeeDaoImpl();
    }

    @Test
    void testSave() throws Exception {
        Employees employee = new Employees("Nguyen", "Quang Anh",
            "anhnd22@fsoft.com.vn", "0988777666", LocalDate.of(2019, 1, 1), 1000, 1.1);

        employee.setJob(job);

        assertTrue(employeeDao.save(employee));
    }
}
```

# One-to-Many Mapping Annotation

@JoinColumn

## ❖ Results:

Results		Messages	
job_id	job_title	max_salary	min_salary
1	J01	2000	1000

employee_id	commission_pct	email	first_name	hire_date	last_name	phone_number	salary	job_id
1	1.1	anhnd22@fsoft.com.vn	Nguyen	2019-01-01	Quang Anh	0988777666	1000	J01

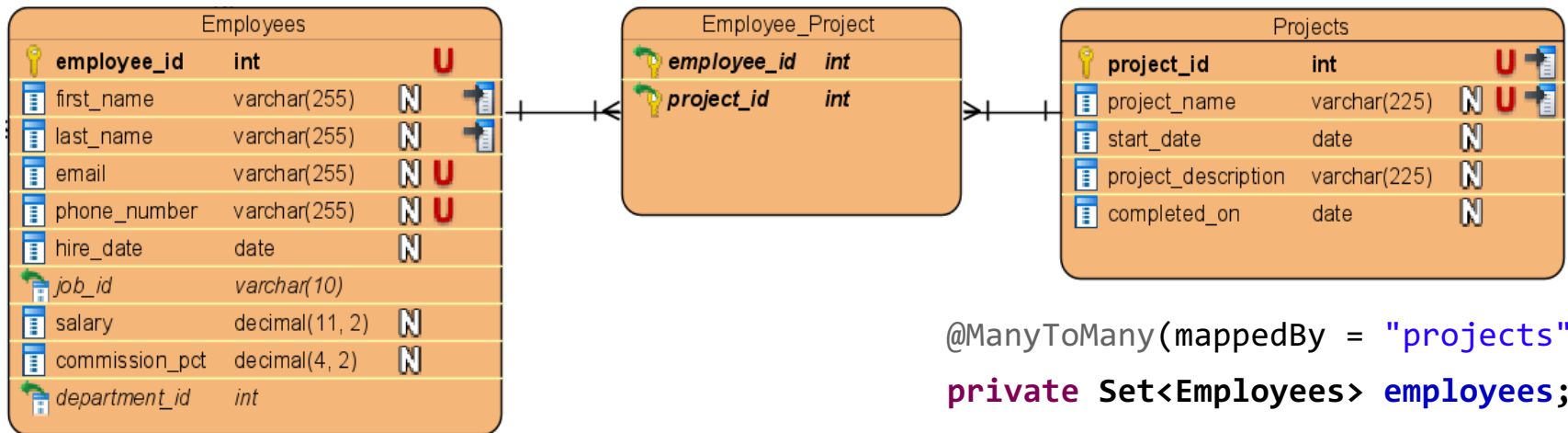
## ❖ Console:

```
Hibernate: create table dbo.Employees (employee_id int identity not null, commission_pct double precision, email varchar(255), first_name varchar(255) not null, last_name varchar(255) not null, phone_number varchar(255), salary double precision not null, job_id varchar(10), primary key (employee_id))
Hibernate: create index IDX_EMP_NAME on dbo.Employees (first_name, last_name)
Hibernate: alter table dbo.Employees drop constraint UK_76snkombmttoxvdljjqo42mmc
Hibernate: alter table dbo.Employees add constraint UK_76snkombmttoxvdljjqo42mmc unique (email)
Hibernate: alter table dbo.Employees drop constraint UK_ivjoyqecd8hc1w4411q70eqri
Hibernate: alter table dbo.Employees add constraint UK_ivjoyqecd8hc1w4411q70eqri unique (phone_number)
Hibernate: alter table dbo.Employees add constraint FKsrmrlbhpjhvfet64uvyt0j7cw foreign key (job_id) references dbo.Jobs
Oct 05, 2020 11:34:00 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate: select jobs_.job_id, jobs_.job_title as job_titl2_2_, jobs_.max_salary as max_sala3_2_, jobs_.min_salary as min_sala4_2_
from dbo.Jobs jobs_ where jobs_.job_id=?
Hibernate: insert into dbo.Employees (commission_pct, email, first_name, hire_date, job_id, last_name, phone_number, salary)
values (?, ?, ?, ?, ?, ?, ?, ?)
```

# Many-to-Many Mapping Annotation

## @JoinTable

- ❖ To map a many-to-many association, we use the `@ManyToMany`, `@JoinTable` and `@JoinColumn` annotations.



```
@ManyToMany(mappedBy = "projects")  
private Set<Employees> employees;
```

```
@ManyToMany(cascade = CascadeType.ALL)  
@JoinTable(name = "Employee_Project", schema = "dbo",  
    joinColumns = {@JoinColumn(referencedColumnName = "employee_id") },  
    inverseJoinColumns = { @JoinColumn(  
        referencedColumnName = "project_id") })  
private Set<Projects> projects;
```

### ❖ Update **Employees** class:

```
@Entity
@Table(name = "Employees", schema = "dbo", indexes = {
    @Index(columnList = "first_name, last_name", name = "IDX_EMP_NAME") })
public class Employees {
    // ...

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "employee_id")
    private int employeeId;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "Employee_Project", schema = "dbo",
        joinColumns = { @JoinColumn(referencedColumnName = "employee_id") },
        inverseJoinColumns = { @JoinColumn(referencedColumnName = "project_id") })
    private Set<Projects> projects;

    public Set<Projects> getProjects() {
        return projects;
    }

    public void setProjects(Set<Projects> projects) {
        this.projects = projects;
    }
    // ...
    // Constructors with params (if need)
    // getter and setter methods
}
```

# Many-to-Many Mapping Annotation

## @JoinTable

### ❖ Create **Projects** class:

```
@Entity
@Table(name = "Projects", schema = "dbo")
public class Projects {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "project_id")
    private int projectId;

    @Column(name = "project_name", length = 255,
            nullable = false, unique = true)
    private String projectName;

    @Column(name = "start_date")
    private LocalDate startDate;

    @Column(name = "project_description")
    private String projectDescription;

    @Column(name = "completed_on")
    private LocalDate completedOn;

    @ManyToMany(cascade = CascadeType.ALL, mappedBy = "projects")
    private Set<Employees> employees;
```

```
public Projects() {
}

// Constructors with params (if need)
// getter and setter methods

public Set<Employees> getEmployees() {
    return employees;
}

public void setEmployees(Set<Employees> employees) {
    this.employees = employees;
}
}
```



### ❖ Create class **ProjectDaoImpl**:

```
public class ProjectDaoImpl implements ProjectDao {

    @Override
    public boolean save(Projects project) {
        Session session = null;
        Transaction transaction = null;

        try {
            session = HibernateUtils.getSessionFactory().openSession();
            transaction = session.beginTransaction();
            Serializable result = session.save(project);
            transaction.commit();

            return (result != null);

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
}
```

- ❖ Create a Unit Test Script to test the `ProjectDaoImpl` above class:

```
class ProjectDaoTest {
    static ProjectDao projectDao;

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        projectDao = new ProjectDaoImpl();
    }

    @Test
    void testSave() throws Exception {
        Employees employee = new Employees("Nguyen", "Dang Khoa",
            "khoadk@fsoft.com.vn", "0988777665", LocalDate.of(2019, 1, 1), 1000, 1.1);

        Projects project = new Projects("IT Fundamental", LocalDate.of(2020, 10, 1),
            "Fsoft Academey It Fundamental Training Program", LocalDate.of(2020, 12, 31));

        Set<Employees> employees = new HashSet<>();
        employees.add(employee);

        Set<Projects> projects = new HashSet<>();
        projects.add(project);

        project.setEmployees(employees);
        employee.setProjects(projects);

        assertTrue(projectDao.save(project));
    }
}
```

# Many-to-Many Mapping Annotation

@JoinTable

## ❖ Results:

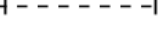
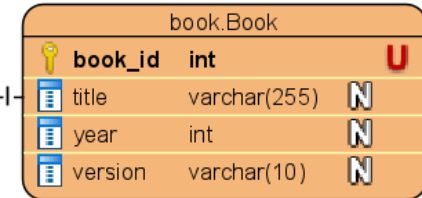
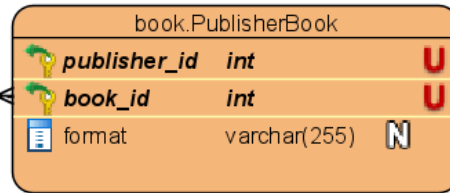
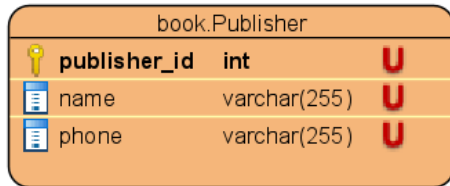
Results									
Messages									
	employee_id	commission_pct	email	first_name	hire_date	last_name	phone_number	salary	job_id
2	4	1.1	khoatk@fsoft.com.vn	Nguyen	2019-01-01	Dang Khoa	0988777665	1000	NULL
	employees_employee_id	projects_project_id							
1	4	1							
	project_id	completed_on	project_description		project_name	start_date			
1	1	2020-12-31	Fsoft Academey It Fundamental Training Program		IT Fundamental	2020-10-01			

## ❖ Console:

```
Oct 05, 2020 3:29:02 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate: insert into dbo.Projects (completed_on, project_description, project_name, start_date) values (?, ?, ?, ?)
Hibernate: insert into dbo.Employees (commission_pct, email, first_name, hire_date, job_id, last_name, phone_number)
values (?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into dbo.Employee_Project (employees_employee_id, projects_project_id) values (?, ?)
```

# Many-to-Many Mapping Annotation

- ❖ Map a **many-to-many** association with **extra columns**.



```
@OneToMany(cascade = CascadeType.ALL,  
           mappedBy = "publisher")  
private Set<PublisherBook> publisherBook;
```

```
@OneToMany(cascade = CascadeType.ALL,  
           mappedBy = "book")  
private Set<PublisherBook> publisherBook;
```

- ❖ **Many-to-Many Using a Composite Key**

- ❖ Note, that there're some **key requirements**, which a **composite key class** has to fulfill:

- ✓ We have to mark it with `@Embeddable`
- ✓ It has to implement `java.io.Serializable`
- ✓ We need to provide an implementation of the `hashCode()` and `equals()` methods
- ✓ None of the fields can be an entity themselves.

# Many-to-Many Mapping Annotation

```
@Entity
@Table(name = "Publisher_Book", schema = "book")
public class PublisherBook implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @ManyToOne
    @JoinColumn(name = "publisher_id")
    private Publisher publisher;

    @Id
    @ManyToOne
    @JoinColumn(name = "book_id")
    private Book book;
    // getter and setter methods

    @Override
    public int hashCode(){}

    @Override
    public boolean equals(Object obj)

}
```

## ❖ Create a test script to check:

```
class PublisherBookDaoTest {
    static PublisherBookDao publisherBookDao;
    static BookDao bookDao;
    static PublisherDao publisherDao;

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        publisherBookDao = new PublisherBookDao();
        bookDao = new BookDao();
        publisherDao = new PublisherDao();
    }

    @Test
    void testSave2() throws Exception {
        Book book = new Book(1, "Java SE", 2020, "1.0");
        assertTrue(bookDao.save(book));

        Publisher publisher = new Publisher(1, "NXB GD", "0979867234");
        assertTrue(publisherDao.save(publisher));

        PublisherBook publisherBook = new PublisherBook(publisher, book, "ABC");
        assertTrue(publisherBookDao.save(publisherBook));
    }
}
```

# Many-to-Many Mapping Annotation

## ❖ Results:

Results				
	book_id	title	version	year
1	1	Java SE	1.0	2020

	publisher_id	name	phone
1	1	NXB GD	0979867234

	format	publisher_id	book_id
1	ABC	1	1

## ❖ Console:

```
Hibernate: create table book2.Book (book_id int identity not null, title varchar(255), version varchar(10), year int not null, primary key (book_id))
Hibernate: create table book2.Publisher (publisher_id int identity not null, name varchar(255), phone varchar(255), primary key (publisher_id))
Hibernate: create table book2.Publisher_Book (format varchar(255), publisher_id int not null, book_id int not null, primary key (publisher_id, book_id))
Hibernate: alter table book2.Book drop constraint UK_odppys65lq7q1xbx8o6p6fgxj
Hibernate: alter table book2.Book add constraint UK_odppys65lq7q1xbx8o6p6fgxj unique (title)
Hibernate: alter table book2.Publisher drop constraint UK_era79tsdasvick3e38j0e9b6v
Hibernate: alter table book2.Publisher add constraint UK_era79tsdasvick3e38j0e9b6v unique (name)
Hibernate: alter table book2.Publisher drop constraint UK_lfeio9fee753ckef2tac2vfku
Hibernate: alter table book2.Publisher add constraint UK_lfeio9fee753ckef2tac2vfku unique (phone)
Hibernate: alter table book2.Publisher_Book add constraint FKt533ea1vy9qjr586kng5g2y0b foreign key (publisher_id) references book2.Publisher (publisher_id)
Hibernate: alter table book2.Publisher_Book add constraint FKKeylcdt22y4uw61t576re0ps8i foreign key (book_id) references book2.Book (book_id)
Oct 11, 2020 3:58:30 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate: insert into book2.Book (title, version, year) values (?, ?, ?)
Hibernate: insert into book2.Publisher (name, phone) values (?, ?)
Hibernate: insert into book2.Publisher_Book (format, publisher_id, book_id) values (?, ?, ?)
```

## Section 04

# LAZY LOADING AND EAGER LOADING



# Lazy loading and Eager loading

- ❖ **Eager Loading** is a design pattern in which data initialization occurs on the spot
- ❖ **Lazy Loading** is a design pattern which is used to defer initialization of an object as long as it's possible

	employee_id	commission_pct	email	first_name	hire_date	last_name	phone_number	salary	job_id
1	1	1.1	anhnd22@fsoft.com.vn	Nguyen	2019-01-01	Quang Anh	0988777666	1000	J01
2	4	1.1	khoatk@fsoft.com.vn	Nguyen	2019-01-01	Dang Khoa	0988777665	1000	NULL
3	5	1.1	thanh@fsoft.com.vn	Nguyen	1999-01-01	Minh Thanh	0988777111	1000	J01
4	7	1.1	Liem@fsoft.com.vn	Hoang	1999-01-01	Van Liem	0988777112	1000	J02

	job_id	job_title	max_salary	min_salary
1	J01	Java Dev1	2000	1000
2	J02	Java Dev2	2200	1200
3	J03	Java Dev3	3200	1400

## ❖ Differences

- ✓ One **Jobs** can have multiple **Employees**. In eager loading strategy, if we load the **Jobs** data, it will also load up all employees associated with it and will store it in a memory.
- ✓ When lazy loading is enabled, if we pull up a *JobsLazy*, *Employees* data won't be initialized and loaded into a memory until an explicit call is made to it.

## ❖ Loading Configuration

```
@Entity
@Table(schema = "dbo", name = "Jobs", indexes = {
    @Index(columnList = "job_id, job_title", name = "IDX_ID_TITLE") })
public class Jobs {

    // ...

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "job", fetch=FetchType.LAZY)
    private Set<Employees> employees;

}
```

# Working with lazy associations

- ❖ Does not actually load all the children when loading the parent.
- ❖ Load children when requested to do it
- ❖ Lazy loading can help improve the performance
- ❖ Create `findById()` method:

```
@Override
public Jobs findById(String jobId) throws Exception {
    Session session = null;

    try {
        session = HibernateUtils.getSessionFactory().openSession();

        Jobs job = session.get(Jobs.class, jobId);

        return job;
    } finally {
        if (session != null) {
            session.close();
        }
    }
}
```

## ❖ Use Eager loading:

**Hibernate:** `select jobs0_.job_id as job_id1_10_0_, jobs0_.job_title as job_titl2_10_0_, jobs0_.max_salary as max_sala3_10_0_, jobs0_.min_salary as min_sala4_10_0_, employees1_.job_id as job_id9_7_1_, employees1_.employee_id as employee1_7_1_, employees1_.employee_id as employee1_7_2_, employees1_.commission_pct as commisi2_7_2_, employees1_.email as email3_7_2_, employees1_.first_name as first_na4_7_2_, employees1_.hire_date as hire_dat5_7_2_, employees1_.job_id as job_id9_7_2_, employees1_.last_name as last_nam6_7_2_, employees1_.phone_number as phone_nu7_7_2_, employees1_.salary as salary8_7_2_ from dbo.Jobs jobs0_ left outer join dbo.Employees employees1_ on jobs0_.job_id=employees1_.job_id`

`Jobs [jobId=J01, jobTitle=Java Dev1, minSalary=1000.0, maxSalary=2000.0]`

## ❖ Use Lazy loading:

**Hibernate:** `select jobs0_.job_id as job_id1_10_0_, jobs0_.job_title as job_titl2_10_0_, jobs0_.max_salary as max_sala3_10_0_, jobs0_.min_salary as min_sala4_10_0_ from dbo.Jobs jobs0_ where jobs0_.job_id=?`

`Jobs [jobId=J01, jobTitle=Java Dev1, minSalary=1000.0, maxSalary=2000.0]`

## ❖ Hibernate reference documentation

- ✓ <http://docs.jboss.org/hibernate>
- ✓ [www.hibernate.org](http://www.hibernate.org)

## ❖ Hibernate example

- ✓ <http://www.mkyong.com/hibernate/>

## ❖ Hibernate course

- ✓ <http://courses.coreservlets.com/Course-Materials/hibernate.html>

# Thank you!

Q&A

