Objective

- Discuss HQL
- Describe Native SQL Query and Criteria Query
- Explain caching in Hibernate
- Explain Transaction management

Why HQL: Example

Analyze the following code to retrieving student record from student table:

```
// POJO class


class Student
{
private int id;
private String name;
// getter and setter
methods
}
```

hibernate.cfg.xml

```
public class Test
{
org.hibernate.cfg.Configuration cfg= new
org.hibernate.cfg.Configuration().config
ure("onetomany/hibernate.cfg.xml");


       StandardServiceRegistryBuilder
builder= new
StandardServiceRegistryBuilder().applySe
ttings(cfg.getProperties());
              SessionFactory
factory=cfg.buildSessionFactory(builder.
build());
              Session s=
factory.openSession();
              s.beginTransaction();
Person user=new Person();
Student s1=user.get(Person.class,1);

}
```

id	name
1	John

Why HQL?

To retrieve the student's data, we provide primary key value for that Student: user.get(Person.class,1);

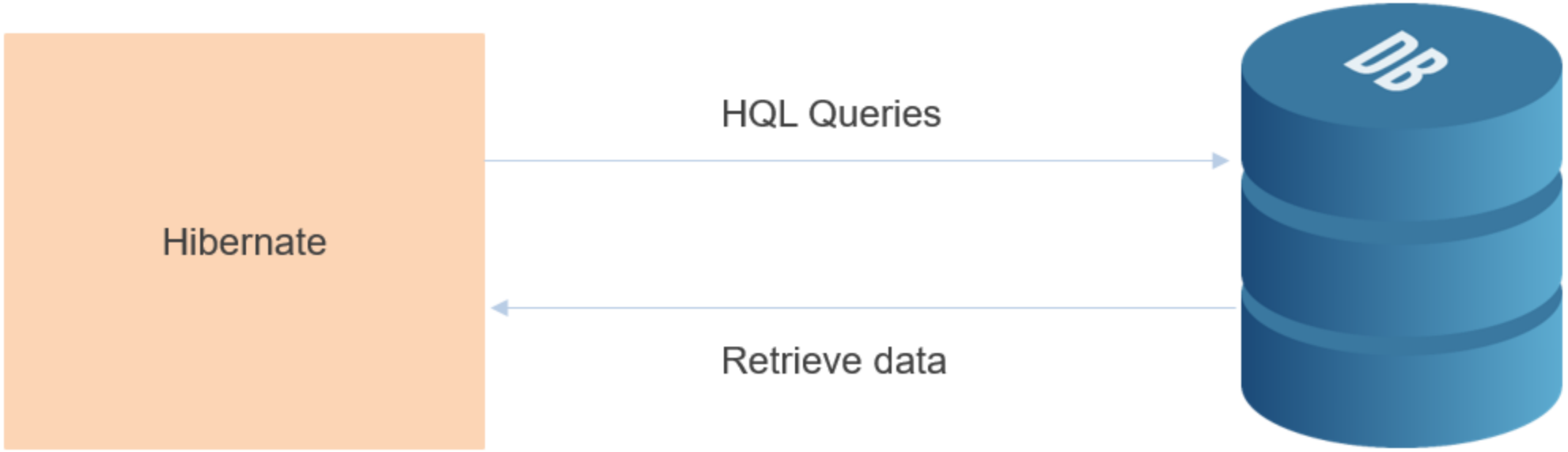
If one table doesn't have a primary key, the records cannot be retrieved.

Hibernate provides its own language known as Hibernate Query Language, to solve such problems. The syntax is quite similar to database SQL language,

Why HQL?

HQL is used to communicate with a database.

It is a Hibernate language for relational database management systems. HQL statements are used to perform tasks such as updating data on a database or retrieving data from a database.



HQL vs. SQL

<table>
<tr><th>HQL</th><th>SQL</th></tr>
<tr><td>It is related to hibernate framework</td><td>It is related to a specific database</td></tr>
<tr><td>HQL queries are object queries</td><td>SQL queries are table queries</td></tr>
<tr><td>Example:

<pre>class Student
{
String name;
int id;
}</pre>

<pre>HQL Query :- from Student
where Student is one class name</pre>

HQL Query: from Student, where Student is one class name</td><td>Example: Student table

<table>
<tr><td></td><td></td></tr>
<tr><td>Id</td><td>name</td></tr>
</table>

SQL Query: Select * from Student, where Student is one table name</td></tr>
</table>

Features of HQL

- HQL is object-oriented
- It understands notions like inheritance, polymorphism, and association
- Its queries are case-insensitive, unlike queries in Java classes.

Example: For HQL queries, SeLeCT is the same as sELEct
but for Java class org.hibernate.eg.STUDENT is not equal to org.hibernate.eg.Student.

Uses of HQL

Using HQL, we can create required CRUD operations:

- UPDATE
- DELETE
- SELECT

Using HQL, we can move table data to another table. However, we cannot insert data using HQL.

Creating HQL query

1. Create configuration object and hibernate.cfg.xml file

```
StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

2. Create SessionFactory object

```
Metadata metaData = new
MetadataSources(standardRegistry).getMetadataBuilder().build();
sessionFactory sf =
metaData.getSessionFactoryBuilder().build();
```

3. Open Session

```
Session sn = sf.openSession();
```

4. Create query

```
Query query = sn.createQuery("Here you can write HQL Query");
```

Methods of query interface

1. getReturnType() - Return the Hibernate types of the query result set
2. iterate() - Return the query results as an Iterator
3. list() - Return the query results as a List

Hibernate parameter building

Named parameters: This is the most common way followed by a parameter name (:example) to define a named parameter. Suppose we have Stock table with stockCode column.

setParameter: The setParameter discovers the parameter data type

```
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql) .setParameter("stockCode",
"1111") .list();
```

setString: This informs Hibernate that the parameter date type is String

```
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql) .setString("stockCode", "1111") .list();
```

setProperties: This can be used to pass an object to the parameter binding. Hibernate will automatically check the object's properties and match them with the colon parameter

```
Stock stock = new Stock(); stock.setStockCode("1111");
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql) .setProperties(stock) .list();
```

HQL Syntax

1. from clause
2. Aliasing
3. Aggregate functions and select clause
4. where clause
5. Expressions
6. order by clause
7. Associations and Joins

HQL Syntax

from clause

- from clause
- Aliasing
- Aggregate functions and select clause
- where clause
- Expressions
- order by clause
- Associations and Joins

from packagename.class_name returns all instances of the class.

Suppose there is a Student table with rollNo and name member variable corresponding to Student class.

To fetch the data from Student table, use the following syntax:

```
String hql = "FROM Student";
 Query query = session.createQuery(hql);
List results = query.list();
```

HQL Syntax

Aliasing

- from clause
- Aliasing
- Aggregate functions and select clause
- where clause
- Expressions
- order by clause
- Associations and Joins

You can assign an alias that is a different name to the class. Suppose alias name is als. The syntax can be:

```
from ClassName as als
```

This query assigns the alias als to ClassName instances, so that you can use that alias later.

HQL Syntax

Aggregate functions and select clause

- from clause
- Aliasing
- Aggregate functions and select clause
- where clause
- Expressions
- order by clause
- Associations and Joins

To get results of aggregate functions on the properties select avg(std.marks), sum(std.marks), max(cat.marks), and count(std)from Student std, the supported aggregate functions are:

- avg(...), sum(...), min(...), max(...)
- count(*)
- count(...), count(distinct ...), count(all...)

You can use arithmetic operators and concatenation

HQL Syntax

where clause

- from clause
- Aliasing
- Aggregate functions and select clause
- where clause
- Expressions
- order by clause
- Associations and Joins

The where clause allows you to refine the list of instances.

- from ClassName where column_name='Ram‘: If there is an alias, use a qualified property name
- from ClassName as als where als.column_name='Ram‘: This returns instances of ClassName named 'Ram'

HQL Syntax

Expressions

- from clause
- Aliasing
- Aggregate functions and select clause
- where clause
- Expressions
- order by clause
- Associations and Joins

Expressions include:

- mathematical operators: +, -, *, /
- binary comparison operators: =, >=, <=, <>, !=, like
- logical operations: and, or, not
- Parentheses ( ) that indicate grouping
- in, not in, between, is null, is not null, is empty, is not empty, member of, and not member of
- "Simple" case, case ... when ... then ... else ... end, and "searched" case, case when ... then ... else ... end
- string concatenation ...||... or concat(...,...)
- current_date(), current_time(), and current_timestamp()
- second(...), minute(...), hour(...), day(...), month(...), and year(...)
- Any function or operator defined by EJB-QL 3.0: substring(), trim(), lower(), upper(), length(), locate(), abs(), sqrt(), bit_length(), mod()
- coalesce() and nullif()
- str() for converting numeric or temporal values to a readable string

HQL Syntax

order by clause

- from clause
- Aliasing
- Aggregate functions and select clause
- where clause
- Expressions
- order by clause
- Associations and Joins

This clause uses a ordered list by any property that is either ascending (ASC) or descending (DESC).

Example:

Suppose there is a list of employees and you need to find their salaries in decreasing order. The following syntax can be used:

```
"FROM Employee E  ORDER BY E.salary DESC";
```

HQL Syntax

Associations and Joins

Join is used to associated entities or elements of a collection of values.

The join types are as follow:

- inner join
- left outer join
- right outer join
- full join

The inner join, left outer join ,and right outer join constructs may be abbreviated.
You may supply extra join conditions using the HQL with keyword

Native SQL queries

1. createSQLQuery() method is used to create native SQL Query for the session.
2. You can create SQL, including stored procedure, for all create, update, delete operations using Hibernate 3.x.
3. The result can be added to Hibernate entity.

How to write SQL Query :

```
Session s;
SQLQuery query = s.createSQLQuery(" Here you can write native SQL Queries"
);
```

Hibernate provides an option to execute native SQL queries through the use of **SQLQuery** object.

Native SQL: Example

Consider a table that has the following data:

Person

PERSONID	FIRSTNAME	LASTNAME
1	Marry	SMITH
1	Carl	SMITH
2	JOHN	SMITH

Name the database as record. POJO class for creating above table:

```
Class Person
{
private int id;
private String firstNmae;
private String lastName;
// public  getter and setter methods
}
```

Lets create a list of all persons record using Native SQL Query.

Native SQL: Example

```
/* Code  to  READ all the Person record using Entity Query */
      Session session; // Let assume We have already created Session object by previous discussed steps.
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         String sql = "SELECT * FROM PERSON";
         SQLQuery query = session.createSQLQuery(sql);
         query.addEntity(Person.class);
         List person = query.list();

         for (Iterator iterator =  person.iterator(); iterator.hasNext();)
{
            Person person = (Person) iterator.next();
            System.out.print("First Name: " + person.getFirstName());
            System.out.print("  Last Name: " + person.getLastName());
}
         tx.commit();
      }
catch (HibernateException e)
 {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
 }
finally
 {
         session.close();
  }
```

What is Criteria Query?

Criteria Query allows you to apply filtration rule and logical condition with query.

org.hibernate.Criteria Query is a method provided by Criteria API that helps in creating criteria query programmatically.

Restriction Class

The criterion package may be used by applications as a framework for building new kinds of criteria.

However, it is intended that most applications will simply use the built-in criterion types via the static factory methods of Restrictions class:

Restriction class	Description
`static SimpleExpression eq(String propertyName, Object value)`	Applies an "equal" constraint to the named property
`static SimpleExpression gt(String propertyName, Object value)`	Applies a "greater than" constraint to the named property
`static PropertyExpression neProperty(String propertyName, String otherPropertyName)`	Applies a "not equal" constraint to two
`static SimpleExpression like(String propertyName, Object value)`	Applies a "like" constraint to the named property
`static Criterion between(String propertyName, Object lo, Object hi)`	Applies a "between" constraint to the named property

Criteria Query: Example

Let's assume we have one table named student, and one class named Student mapped to student table, which has following structure:

Roll no	Name	Age	Marks

Structure for Student class:

```
class Student
{
private int rollno;
private String name;
private int age;
private int marks;
//getter and setter methods
}
```

Applying Criteria Query

Criteria Query in given table can be applied in two ways:

1. Criteria Query without restrictions

```
Session s;  s.createCriteria(Student.class).list();
```

This will show data of student table in the form of a list.

2. Criteria Query with restrictions

```
Session s;
Criteria crt = s.createCtriteria(Student.class);
crt.add(Restrictions.gt("age",23));
List <Student> list = crt.list();
```

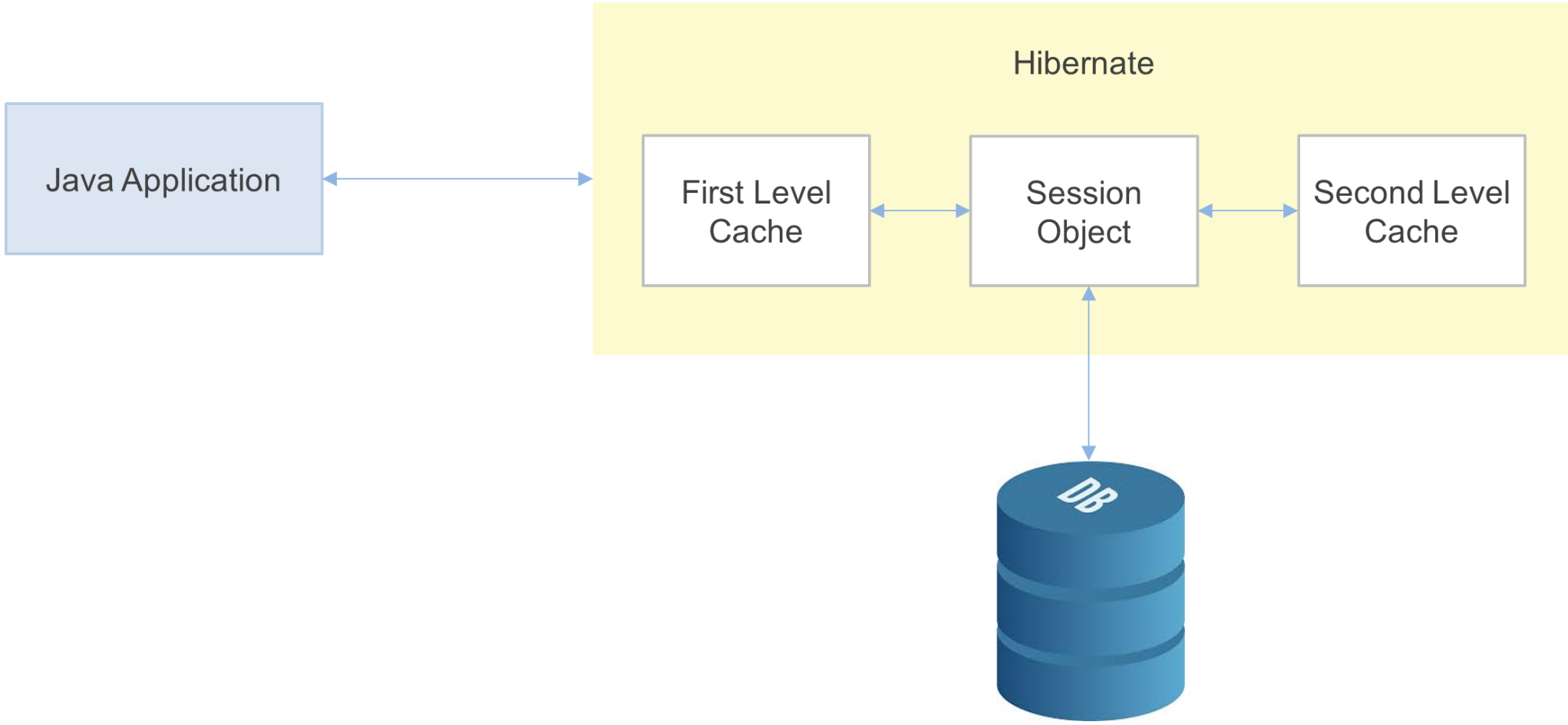
This will show list of students with age greater than 23.

Cache and Caching

- Cache is a component that stores data temporarily so future requests for that data can be served faster
- Cache is a temporary storage area
- Caching is a process to store data in cache
- Caching in Hibernate avoids database hits and increases performance of critical applications

Cache Hierarchy in Hibernate

Hibernate follows multilevel cache schema, as shown:



Types of cache in hibernate

First - Level Cache:

1. All caches must pass through First-Level Cache.
2. When you close the Session object, all objects being cached are lost and persist or are updated in database.

Second - Level Cache:

1. It is an optional cache
2. It uses a common cache for all the session objects of a session factory.

Example:

JBoss cache as a Hibernate Second - Level Cache. It is tree-structured, clustered, transactional cache.

Why mapping?

Hibernate mapping is used for the following:

- To map inheritance hierarchy classes with the table of the database
- To meet the need of mapping collection elements of Persistent class in Hibernate
- To map dependent object as a component
- Hibernate mapping is used to map Java data type to SQL data type and vice versa. Hibernate uses Hibernate Type, which is different from Java data type and SQL data type

Hibernate types

1. Value type has the following types:
    - Basic value Type: Hibernate provides a number of built-in basic types
    - Composite type: Components represent aggregation of value into a single Java type
    - Collection type: To map collection, Hibernate provides collection type
2. Entity type: Entities are classes that correlate with row in table
3. Custom type: Hibernate allows you to create your own value types

Types of mapping in hibernate

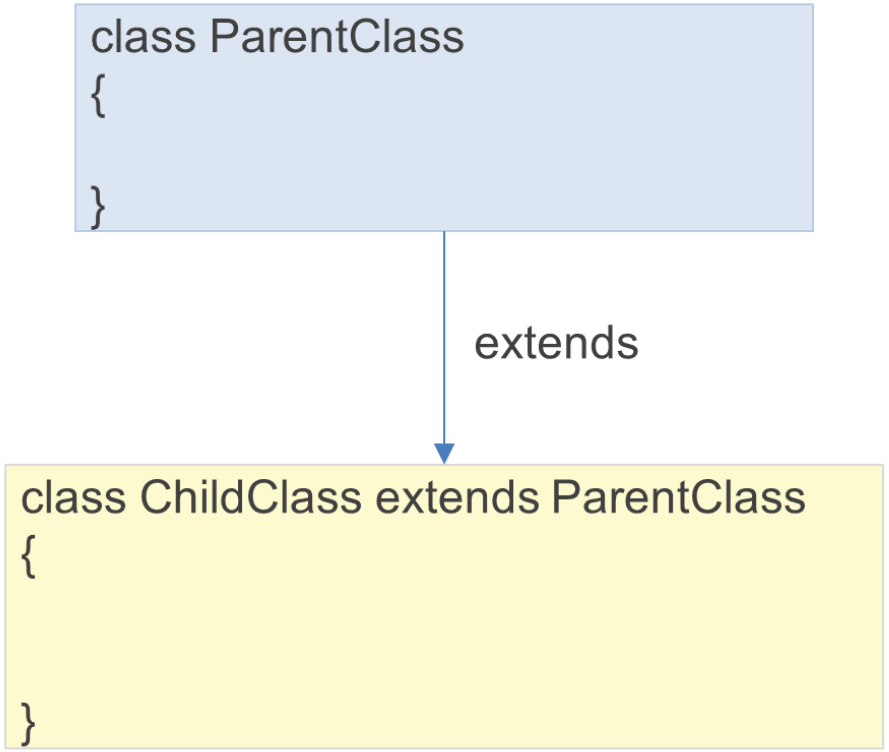
1. Inheritance Mapping
2. Association Mapping
3. Collection Mapping
4. Component Mapping

Types of mapping in hibernate

INHERITANCE MAPPING

Java inheritance relationship between classes is represented as follows:

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping



We can map the inheritance hierarchy classes with the table of the database using inheritance mapping.

Types of mapping in hibernate

INHERITANCE MAPPING: TYPES

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

There are three types of inheritance mapping:

- Single Table per Class Hierarchy Strategy: the <subclass> element in Hibernate
- Table per class: the <union-class> element in Hibernate
- Joined Subclass Strategy: the <joined-subclass> element in Hibernate

Types of mapping in hibernate

INHERITANCE MAPPING: SINGLE TABLE PER CLASS HIERARCHY STRATEGY

It is also known as Table per class. It uses one table. To distinguish classes, it uses discriminator column. Lets take an example. Consider two classes as shown:

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Person class

```
@Entity
@Inheritance
@DiscriminatorColumn(
  name="discriminator",
discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("P")
public class Person
{
@Id
private int personId;
private String fName;
private String lName;
// setter and getter method;
}
```

Employee class

```
@Entity
@DiscriminatorValue("E")
public class Employee extends Person {
private String departmentName;
private Date joiningDate;
//setter and getter methods

}
```

Single Table per Class Hierarchy Strategy	Table per class	Joined Subclass Strategy

Types of mapping in hibernate

INHERITANCE MAPPING: SINGLE TABLE PER CLASS HIERARCHY STRATEGY

Inheritance Mapping
Association Mapping
Collection Mapping
Component Mapping

Let's create object for Employee class

```
Person person = new Person();
              person.setFirstname("john");
              person.setLastname("sully");
              person.setPersonId(1);


              Employee employee = new Employee("linda", "watson", "Tech", new Date());
              s.save(person);
              s.save(employee);
```

discriminator	id	fName	lName	departmentName	joiningDtae
P	101	John	Sully	null	null
E	102	Linda	Watson	tech	2017-08-08 17:53:58

Columns declared by the subclasses may not have NOT NULL constraints.

Single Table per Class Hierarchy Strategy	Table per class	Joined Subclass Strategy

Types of mapping in hibernate

INHERITANCE MAPPING: TABLE PER CLASS

Inheritance Mapping
Association Mapping
Collection Mapping
Component Mapping

Table Per class contains three tables in the database that are not related to each other. There are two ways to map the table with table per concrete class strategy.

- By union-subclass element
- By Self creating the table for each class

```
Syntax:

@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Flight implements Serializable
 {
 ...
}
```

Single Table per Class Hierarchy Strategy	Table per class	Joined Subclass Strategy

Types of mapping in hibernate

INHERITANCE MAPPING: PROPERTIES OF TABLE PER CLASS

Inheritance Mapping
Association Mapping
Collection Mapping
Component Mapping

1. Each sub-table has a column for all properties, including inherited properties.
2. Column name is the same in all sub tables.
3. Primary key seed is shared in all sub tables.

Single Table per Class Hierarchy Strategy	Table per class	Joined Subclass Strategy

Types of mapping in hibernate

INHERITANCE MAPPING: JOINED SUBCLASS STRATEGY

Inheritance Mapping
Association Mapping
Collection Mapping
Component Mapping

- It is also known as One Table Per Subclass
- It uses n number of tables, where n = total number of classes, including sub and super classes
- It uses one-to-one association
- All subtables use primary key association with super table
- The @PrimaryKeyJoinColumn and @PrimaryKeyJoinColumns annotations define the primary key(s) of the joined subclass table

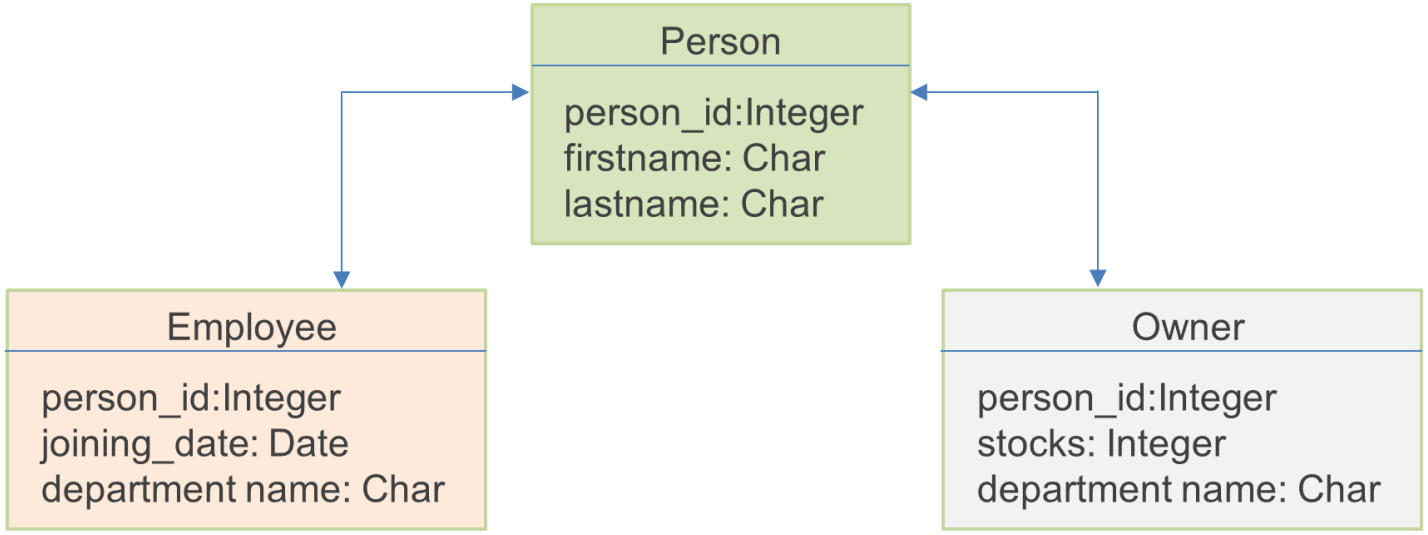
Single Table per Class Hierarchy Strategy	Table per class	Joined Subclass Strategy

Types of mapping in hibernate

INHERITANCE MAPPING: JOINED SUBCLASS STRATEGY

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

The @PrimaryKeyJoinColumn and @PrimaryKeyJoinColumns annotations define the primary key(s) of the joined subclass table:



Single Table per Class Hierarchy Strategy	Table per class	Joined Subclass Strategy

Types of mapping in hibernate

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

INHERITANCE MAPPING: JOINED SUBCLASS STRATEGY

Person class

```
@Entity
@Table(name = "PERSON")
@Inheritance(strategy=InheritanceType.JOINED)
public class Person {
    @Id
    @GeneratedValue
    @Column(name = "PERSON_ID")
    private Long personId;
    @Column(name = "FIRSTNAME")
    private String firstname;
    @Column(name = "LASTNAME")
    private String lastname;
    public Person() {
    }
    public Person(String firstname, String
lastname) {
            this.firstname = firstname;
            this.lastname = lastname;
    }
    // Getter and Setter methods,
}
```

Employee class

```
@Entity
@Table(name="EMPLOYEE")
@PrimaryKeyJoinColumn(name="PERSON_ID")
public class Employee extends Person {
       @Column(name="joining_date")
       private Date joiningDate;
       @Column(name="department_name")
       private String departmentName;
       public Employee() {
       }
       public Employee(String firstname,
String lastname, String departmentName, Date
joiningDate) {
              super(firstname, lastname);
              this.departmentName =
departmentName;
              this.joiningDate = joiningDate;
       }
       // Getter and Setter methods,
}
```

- Single Table per Class Hierarchy Strategy
- Table per class
- Joined Subclass Strategy

Types of mapping in hibernate

INHERITANCE MAPPING: JOINED SUBCLASS STRATEGY

Inheritance Mapping
Association Mapping
Collection Mapping
Component Mapping

Owner class

```
@Entity
@Table(name="OWNER")
@PrimaryKeyJoinColumn(name="PERSON_ID")
public class Owner extends Person {
      @Column(name="stocks")
      private Long stocks;
      @Column(name="partnership_stake")
      private Long partnershipStake;
      public Owner() {
      }
      public Owner(String firstname, String lastname, Long stocks, Long
partnershipStake) {
                        super(firstname, lastname);
            this.stocks = stocks;
            this.partnershipStake = partnershipStake;
      }
      // Getter and Setter methods,
}
```

Single Table per Class Hierarchy Strategy	Table per class	Joined Subclass Strategy

Types of mapping in hibernate

INHERITANCE MAPPING: JOINED SUBCLASS STRATEGY

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Database representation:

Person table

person_id	firstname	lastname
1	Steve	Balmer
2	James	Gosling
3	Bill	Gates

Employee table

person_id	joining_date	department_name
2	2011-12-23	Marketing

Owner table

person_id	stocks	partnership_stake
3	300	20

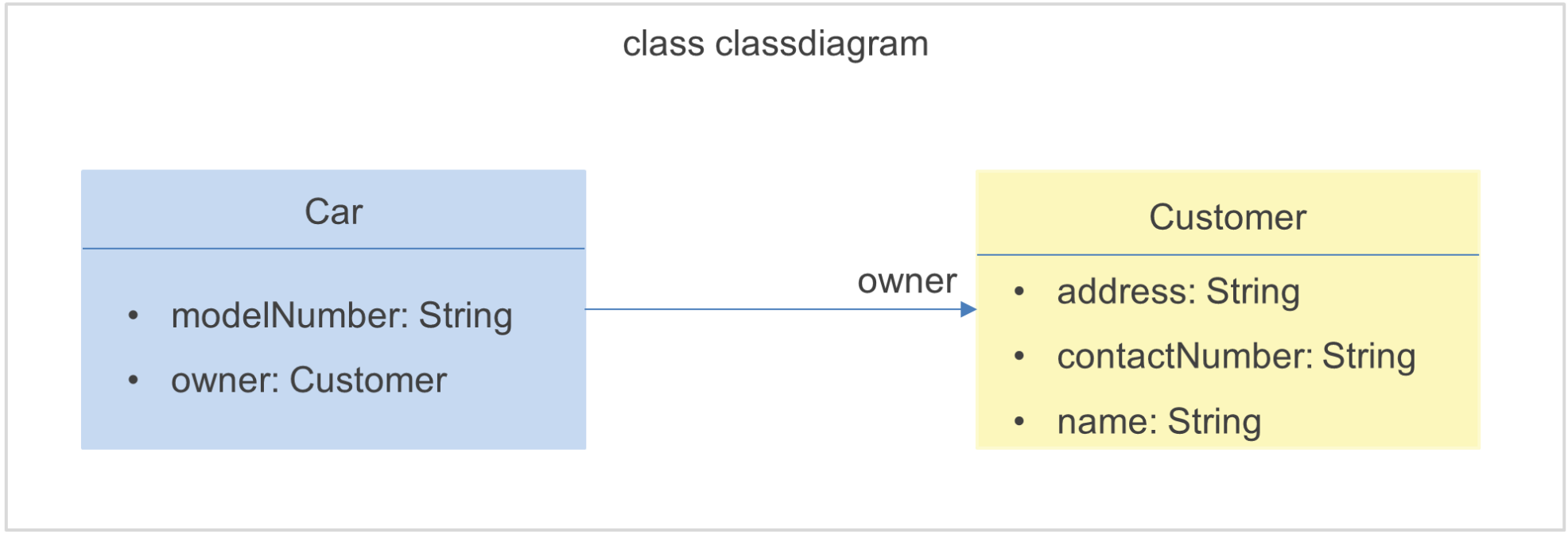
- Single Table per Class Hierarchy Strategy
- Table per class
- Joined Subclass Strategy

Types of mapping in hibernate

ASSOCIATION MAPPING

It is a mapping relationship between two objects in Java

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping



Types of mapping in hibernate

ASSOCIATION MAPPING: TYPES

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

1. One to One
2. One to Many
3. Many to One
4. Many to Many

Types of mapping in hibernate

Inheritance Mapping
Association Mapping
Collection Mapping
Component Mapping

ASSOCIATION MAPPING: ONE TO ONE

A one-to-one relationship occurs when one entity is related to exactly one occurrence in another entity.

Example:

One person is associated with one passport, and one passport belongs to one person. Here, Person class and Passport class have one-to-one relationship.

```
class Person{
private int id;
private String name;


}
```

```
class Passport
{

private int id;
private Date valid_date;
}
```

One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: ONE TO ONE

The annotation used to generate one to one mapping between person and passport table is @OneToOne .

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Person class

```
@Entity
@Table (name="PERSON")
class Person{
@Id
private int id;
private String name;
@OneToOne
private Passport passport;
}
```

Passport class

```
@Entity
@Table (name="PASSPORT")
class Passport
{
@Id
private int id;
private Date valid_date;
}
```

Hibernate creates one column in Person table corresponds to PASSPORT table id column.

When member variable is sent to Person, set Passport member of Person class can be avoided.

One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: ONE TO ONE

Let's create the object of Person class and Passport class:

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

```
Person p=new Person();
p.setId(1);
p.setName("John");
Passsport k=new Passport();
k.setId(101);
k.setValue("abc);
p.setPassport(k);
session.save(p);
session.save(k);
```

Person Table

id	name	passport_id
1	John	101

Passport Table

id	value
101	abc

If foreign key is added, annotation used should be @JoinColumn(name="passport_fk").

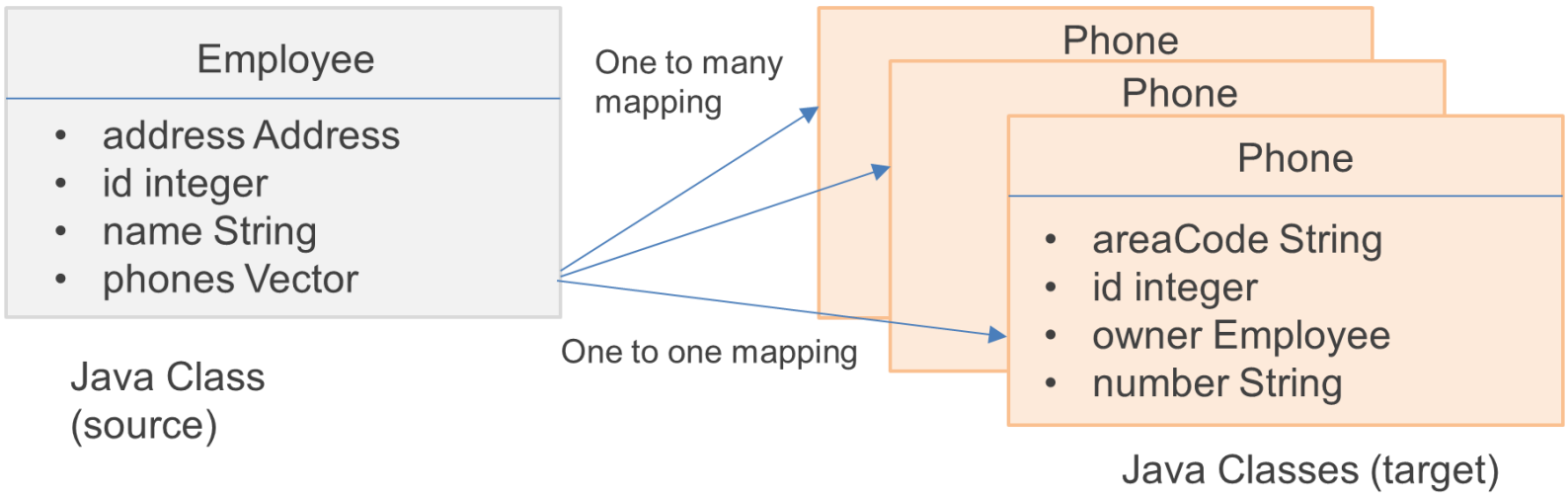
One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: ONE TO MANY

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Consider a scenario where one Employee has multiple phone numbers and one phone number belongs to one Employee.



One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: ONE TO MANY

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Creating classes:

```
class Person
{
int id;
String name;
}
```

```
class Phone
{
int id;
int phoneNumber;
}
```

Employee Class:

```
@Entity
@Table(name="EMPLOYEE")
public class Employee {
      @Id
            private int
employeeId;
            @OneToMany
private collection
<Phone>=new ArrayList <
phone>();
// setter getter methods
}
```

Phone Class:

```
@Entity
@Table(name="Phone")
public class Phone {
      @Id
            private int
PhoneId;
                  private int
PhoneNumber;
//setter and getter methods
}
```

Add user defined name using the following annotation:
@JoinTable(name="USER_VEHICLE", join Column=@Join Column(name="USER_ID")
inverseJoinColumns=@JoinColumn(name="VEHICLE_ID")

One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: ONE TO MANY

Creating objects for Person and Phone classes:

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

```
Person user=new Person();
user.setId(1);
user.setName("john");
Phone p1= new Phone();
p1.(101);
p1.(123456789);
user.getPhone().add(p1);
Phone p2= new Phone();
p2.(102);
p2.(1112131415);
user.getPhone().add(p2);
session.save(user);
session.save(p1);session.save(p2);
```

Person table

id	name
1	john
2	Stella

Phone table

id	phoneNumber
101	123456789
102	1112131415

Person_Phone table

person_id	phone_id
1	101
1	102

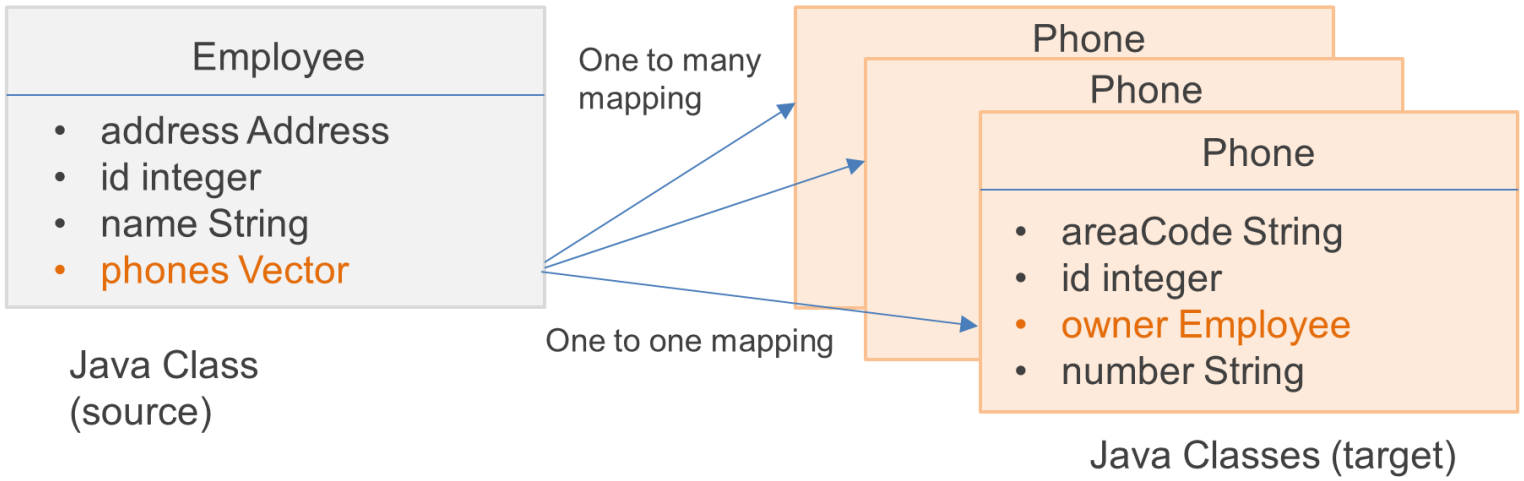
One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: MANY TO ONE

Let's consider the same scenario again. Many phone numbers can belong to one person.

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping



One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: MANY TO ONE

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Creating classes:

```
class Person
{
int id;
String name;
}
```

```
class Phone
{
int id;
int phoneNumber;
}
```

Person Class:

```
@Entity
@Table(name="PERSON")
public class Person {
      @Id
            private int id;
private String name;
// setter getter methods
}
```

Phone Class

```
@Entity
@Table(name="Phone")
public class Phone {
      @Id
            private int  Id;
                  private int
phoneNumber;
@ManytoOne
private Person person
//setter and getter methods
}
```

One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: MANY TO ONE

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Creating objects for Person and Phone class

```
Person user=new Person();
user.setId(1);
user.setName("john");
Phone p1= new Phone();
p1.(101);
p1.(123456789);
p1.setPerson(user);
Phone p2= new Phone();
p2.(102);
p2.(1112131415);
p2.setPerson(user);
session.save(user);
session.save(p1);session.save(p2);
```

Person table

id	name
1	john
2	Stella

Phone table

id	phoneNumber
101	123456789
102	1112131415

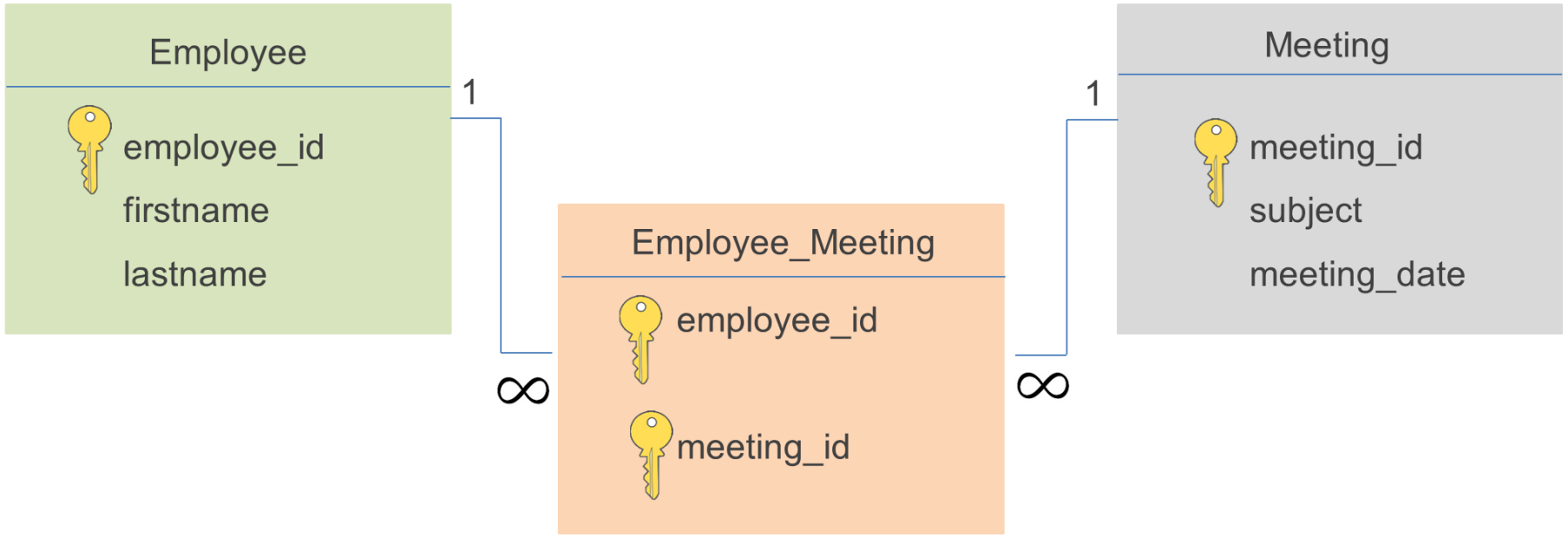
One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

ASSOCIATION MAPPING: MANY TO MANY

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

A many-to-many association is defined logically using the @ManyToMany annotation. You also have to describe the association table and the join conditions using the @JoinTable annotation.



One to One	One to Many	Many to One	Many to Many

Types of mapping in hibernate

COLLECTION MAPPING

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Hibernate allows you to map collection elements of Persistent class. It can be done by declaring the type of collection in Persistent class from one of the following types:

- java.util.List
- java.util.Set
- java.util.SortedSet
- java.util.Map
- java.util.SortedMap
- java.util.Collection
- or write the implementation of org.hibernate.usertype.UserCollectionType

Types of mapping in hibernate

COMPONENT MAPPING

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Component refers to the object-oriented notation.

For example: Student class has one class member: Name. Name is also one class that has two members, firstName and lastName.

Person Class

```
@Entity
@Table(name="PERSON")
public class Person {
      @Id
            private int id;
private String name;
@ManytoMany
private Collection<Meeting>=new
ArrayList();
// setter getter methods
}
```

Phone Class

```
@Entity
@Table(name="Phone")
public class Phone {
      @Id
            private int  Id;
                  private int
phoneNumber;
@ManytoOne
private Collection <Person > person=new
ArrayList();
//setter and getter methods
}
```

Types of mapping in hibernate

COMPONENT MAPPING

- Inheritance Mapping
- Association Mapping
- Collection Mapping
- Component Mapping

Creating objects for Person and Phone class:

```
Person user=new Person();
user.setId(1);
user.setName("john");
Meeting p1= new Meeting();
p1.(101);
p1.(123456789);
p1.getPerson().add(user);
Phone p2= new Phone();
p2.(102);
p2.(1112131415);
p2.getPerson().add(user);
user.getMeeting().add(p1);
user.getMeeting().add(p2);
session.save(user);
session.save(p1);session.save(p2);
```

Person table

id	name
1	john
2	Stella

Phone table

id	phoneNumber	person_id
101	123456789	1
102	1112131415	1

meeting_person TABLE

meeting_id	person_id
101	1
102	1

person_meeting TABLE

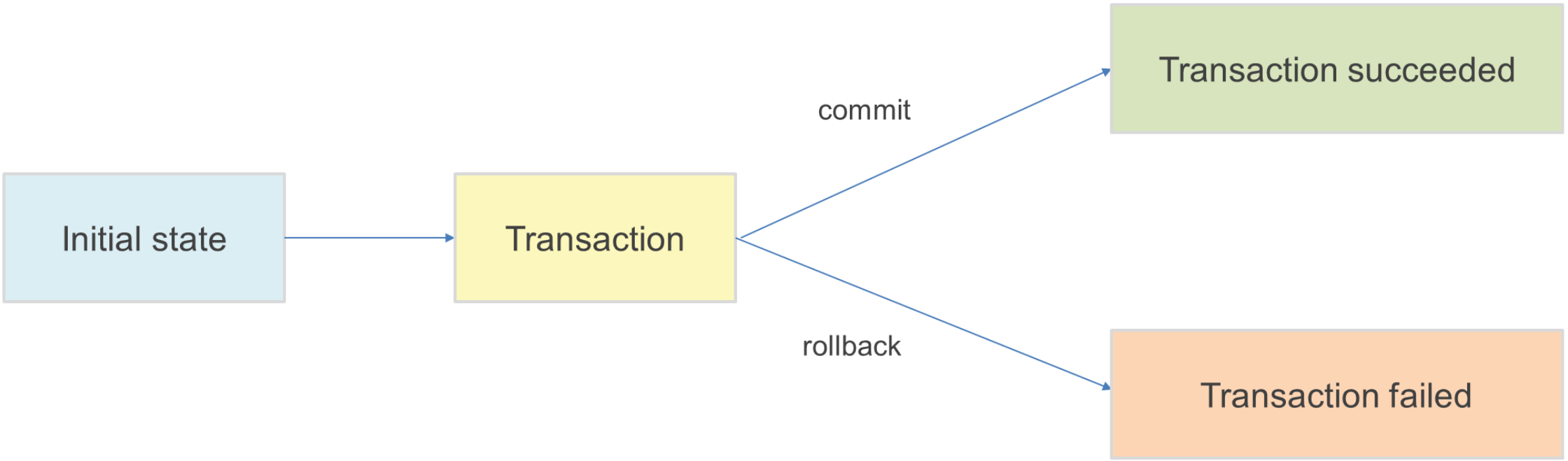
id	phoneNumber
1	101
1	102

In component mapping, the dependent object is mapped as a component. A component is an object that is stored as a value rather than entity reference. This is mainly used if the dependent object doesn't have primary key. It is used in case of composition (HAS-A relation). That is why it is termed component.

What is Transaction Management?

A transaction simply represents a unit of work if one step fails, the whole transaction fails.

It is described by ACID properties (Atomicity, Consistency, Isolation, and Durability).



Transaction Interface

In hibernate framework, we have transaction interface that defines the unit of work.

- A transaction is associated with Session and instantiated by calling session.beginTransaction().
- void begin() starts a new transaction
- void commit() ends the unit of work unless you are in FlushMode.NEVER.
- void rollback() forces this transaction to rollback.
- void setTimeout(int seconds) sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
- boolean isAlive() checks if the transaction is still alive.
- void registerSynchronization(Synchronizations) registers a user synchronization callback for this transaction.
- boolean wasCommited() checks if the transaction was committed successfully.
- boolean wasRolledBack() checks if the transaction was rolled back successfully.

Key Takeaways

- Hibernate provides its own language known as Hibernate Query Language. The syntax is quite similar to database SQL language.
- HQL is a Hibernate language for relational database management systems. HQL statements are used to perform tasks such as updating data on a database or retrieving data from a database.
- The criterion package may be used by applications as a framework for building new kinds of criterion.
- Hibernate provides the option to execute native SQL queries through the use of SQLQuery object.
- Criteria Query allows you to apply filtration rule and logical condition with query. org.hibernate.Criteria Query is a method provided by Criteria API that helps in creating criteria query programmatically.