# ORM

LESSON 18

# ORM Framework
# Java Persistence API
# Hibernate

# ORM

- ORM Object Relational Mapping is the programing technique to map application objects to relational database tables
- ORM is concerned with helping your application to achieve persistence. Persistence simply means that we would like our application's data to outlive the applications process
- ORM is a task–one that developers have good reason to avoid doing manually
- Finally, application can focus on objects

# ORM – Solve the JDBC problems first

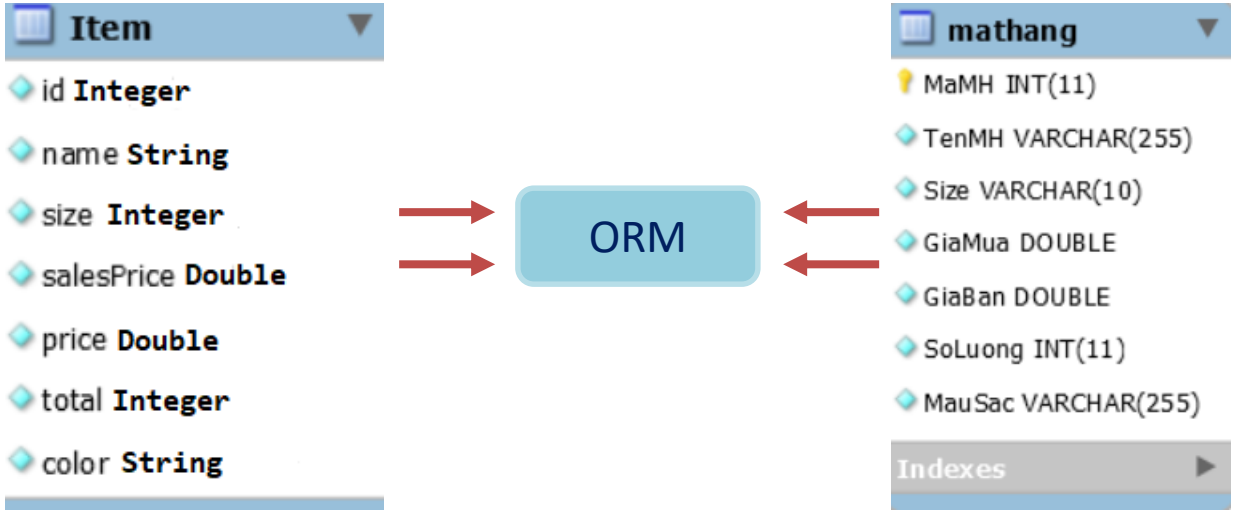
```java
@Override
public List<Item> getItems(LocalDate salesDate) {
    List<Item> result = new ArrayList<>();
    final String query = "SELECT mh.MaMH     AS " + Item.ID         + ",\n"
                       + "       mh.TenMH    AS " + Item.NAME       + ",\n"
                       + "       mh.GiaBan   AS " + Item.SALES_OUT  + ",\n"
                       + "       mh.SoLuong  AS " + Item.QUANTITY   + " \n"
                       + "FROM MatHang mh\n"
                       + "JOIN ChiTietDonHang ctdh\n"
                       + "  ON mh.MaMH = ctdh.MaMH\n"
                       + "JOIN DonHang dh\n"
                       + "  ON dh.MaDH = ctdh.MaDH\n"
                       + "WHERE cast(dh.NgayTao as Date) = ?";
    try {
        pst = conn.prepareStatement(query);
        pst.setDate(1, java.sql.Date.valueOf(salesDate));
        rs = pst.executeQuery();
        while(rs.next()) {
            Item item = new Item();
            transformer(item);
            result.add(item);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        SqlUtils.close(rs, pst);
    }
    return result;
}
```

```java
private void transformer(Item item) throws SQLException{
    item.setId(rs.getInt(Item.ID));
    item.setName(rs.getString(Item.NAME));
    item.setSalesOut(rs.getDouble(Item.SALES_OUT));
    item.setQuantity(rs.getInt(Item.QUANTITY));
}
```

ResultSet mapping manually

```java
public List<Item> getAll() {
    return openSession()
            .createNativeQuery("SELECT * FROM MatHang", Item.class)
            .getResultList();
}
```

Automatic mapping by ORM



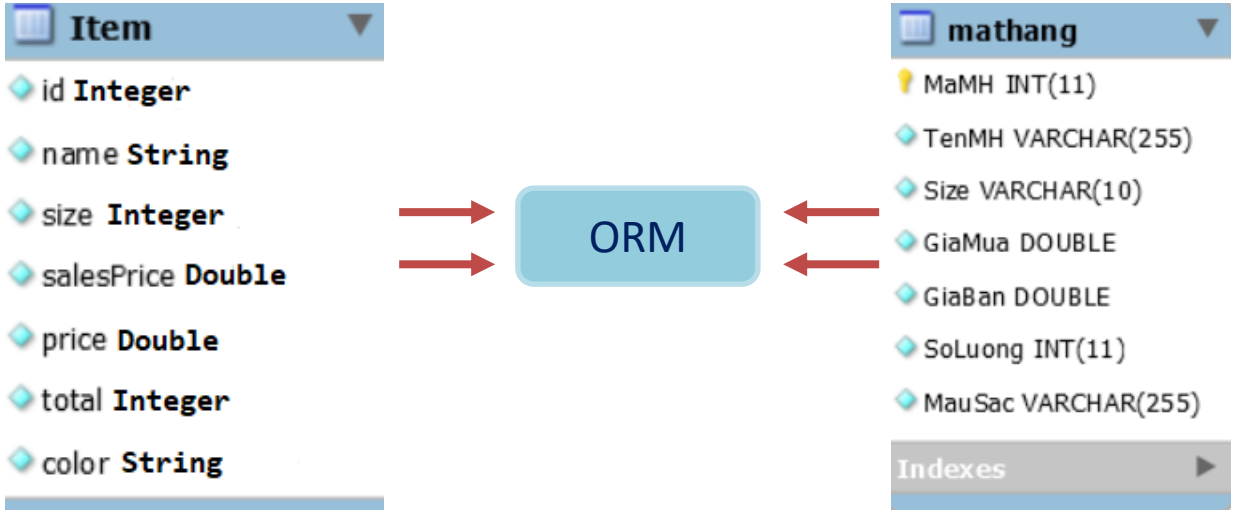
# ORM – Solve the JDBC problems first

```java
public boolean save(Item item) {
    boolean isSuccess = false;
    String sql = "INSERT INTO MatHang(MaMH, TenMH, Size, "
                                + "GiaBan, GiaMua, SoLuong, MauSac)\n"
            + "VALUES(?, ?, ?, ?, ?, ?, ?)";
    try {
        pst = conn.prepareStatement(sql);

        pst.setInt(1, item.getId());
        pst.setString(2, item.getName());
        pst.setString(3, item.getSize());
        pst.setDouble(4, item.getSalesPrice());
        pst.setDouble(5, item.getPrice());
        pst.setInt(6, item.getQuantity());
        pst.setString(7, item.getColor());

        isSuccess = pst.executeUpdate() > 0;

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        SqlUtils.close( pst);
    }

    return isSuccess;
}
```

ResultSet mapping manually

```java
public void save(Item item) {
    Session session = openSession();
    session.save(item);
}
```

Automatic mapping by ORM



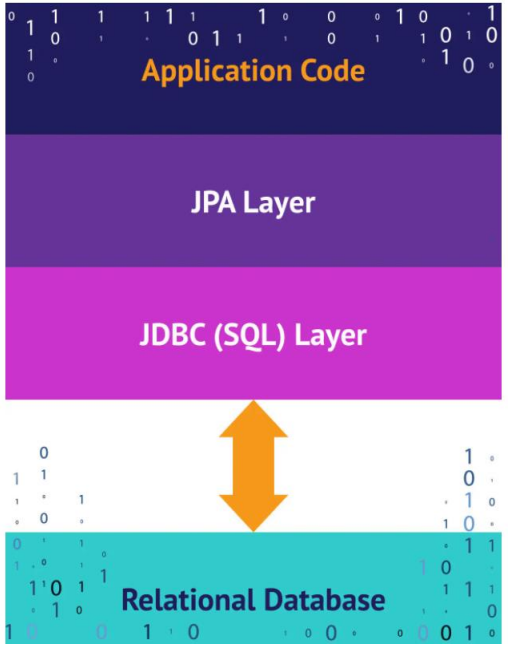
# Java Persistence API

- Java Persistence is java based ORM tool that provides framework for mapping application domain objects to relational database tables and vice versa

- JPA is a specification, it defines a set of concepts that can be implemented by any tool or framework

- Several implementations are available such as Hibernate, EclipseLink

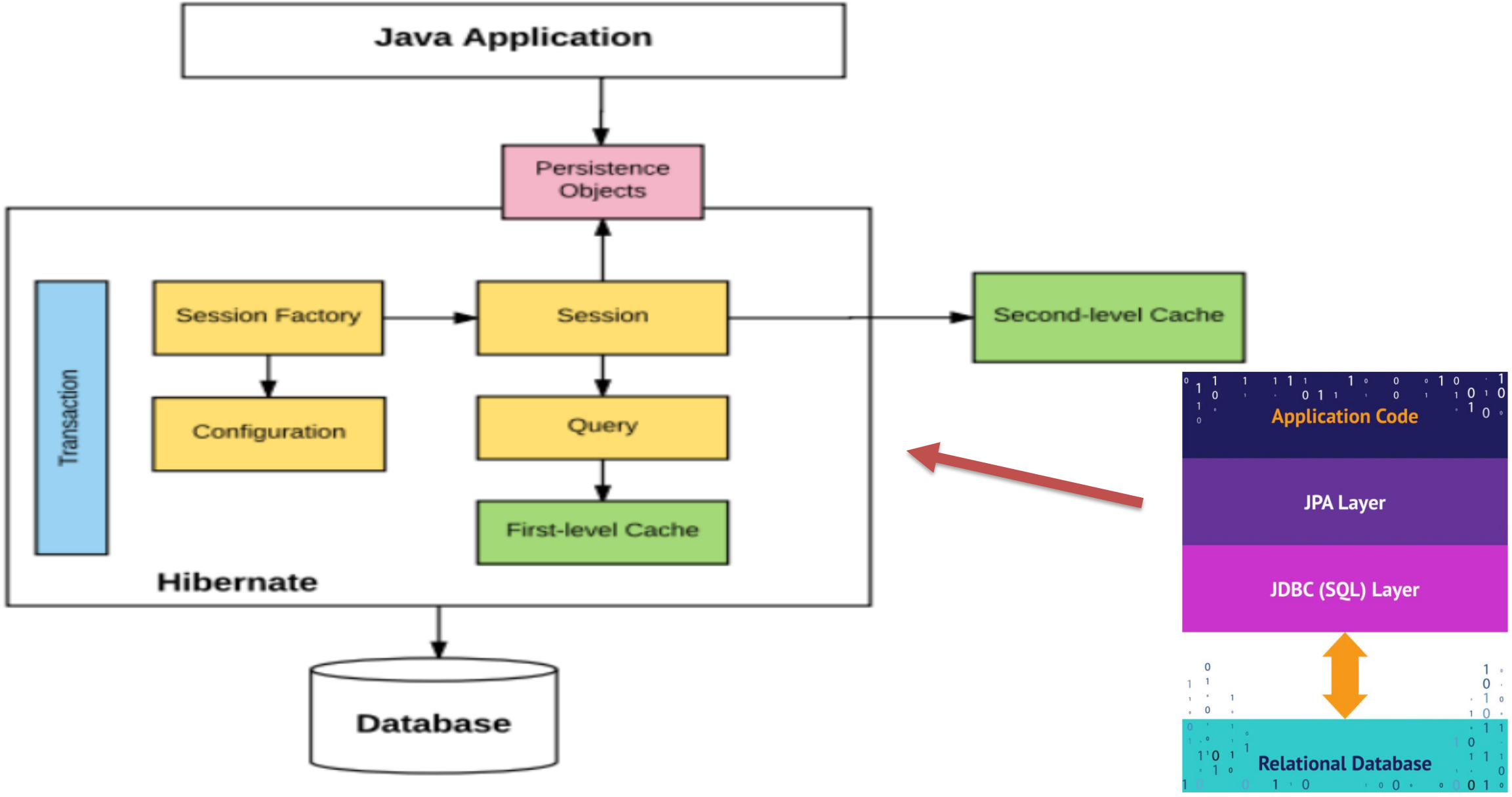
- JPA specifications are defined with annotation in java.persistence p annotation helps us in writing implementation independent code



# Hibernate

- Hibernate ORM is one of the most mature JPA implementations, and still a popular option for ORM in Java.
- Hibernate ORM 5.3.8 (the current version as of this writing) implements JPA 2.2.
- Additionally, Hibernate's family of tools has expanded to include popular tools like Hibernate Search, Hibernate Validator, and Hibernate OGM, which supports domain-model persistence for NoSQL.
- Because of their intertwined history, Hibernate and JPA are frequently conflated

# Hibernate Architecture



# Hibernate Architecture

Configuration

Session Factory

- Contains the configuration properties of JDBC background and Hibernate

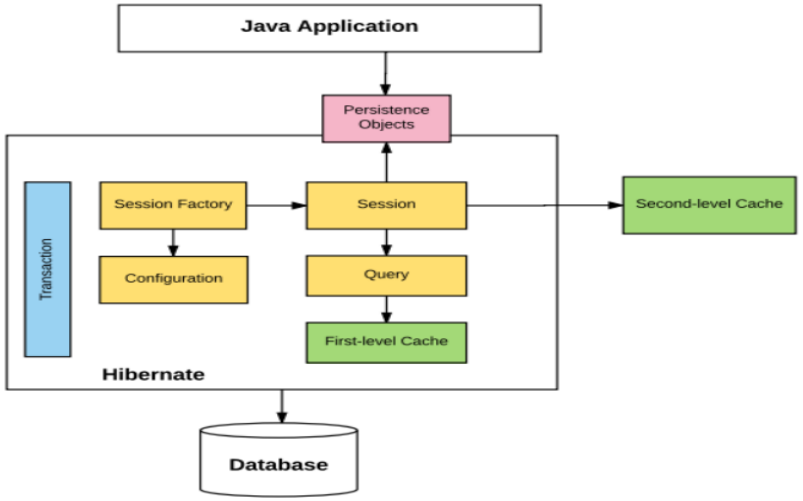
```java
private static SessionFactory sessionFactory;

public static SessionFactory getSessionFactory() {
    if (sessionFactory == null) {
        Configuration configuration = new Configuration();
        sessionFactory = configuration.configure("hibernate.cfg.xml")
                                      .buildSessionFactory();
    }
    return sessionFactory;
}
```

XML

```java
public static SessionFactory getSessionFactory() {
    if (sessionFactory == null) {
        Configuration configuration = new Configuration();
        configuration.addAnnotatedClass(ItemGroup.class);
        sessionFactory = configuration.setProperties(getHibernateProps())
                                      .buildSessionFactory();
    }
    return sessionFactory;
}
```

JAVA Annotation

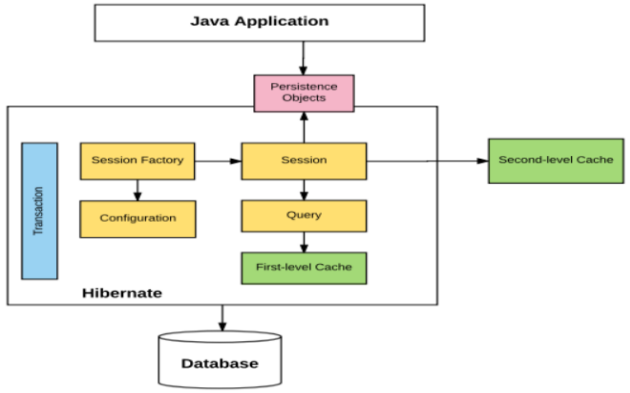
```xml
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/java10_shopping</property>
        <property name="connection.username">root</property>
        <property name="connection.password">1234</property>

        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>

        <property name="show_sql">true</property>
        <property name="format_sql">true</property>

        <!-- Set the current session context -->
        <property name="current_session_context_class">thread</property>

        <!-- Scan Entities -->
        <mapping class="persistence.ItemGroup" />
        <mapping class="persistence.Item" />
    </session-factory>
</hibernate-configuration>
```



# Hibernate Architecture

Session Factory | Session | Transaction

- Session Factory responsible for the creation of Session objects
- Session provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection.
- It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data.
- Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria
- Query could be Native Query or Hibernate Query Language
- Transaction manages a transaction of Sessions with dataset



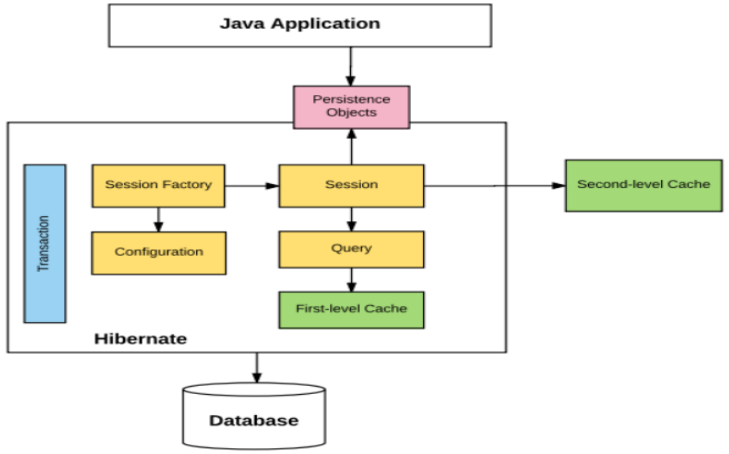
# Hibernate Architecture

Session Factory | Session | Transaction

```
private static SessionFactory sessionFactory;

static {
    sessionFactory = HibernateUtils.getSessionFactory();
}

Session openSession() {
    // response EACH session for each execution of query
    return sessionFactory.openSession();
}

Session getCurrentSession() {
    // response ONE unique session for ONE Session Factory
    // <property name="current_session_context_class">thread</property>
    return sessionFactory.getCurrentSession();
}
```

Optional transaction in GET method

Always need a transaction behind



# Hibernate Architecture

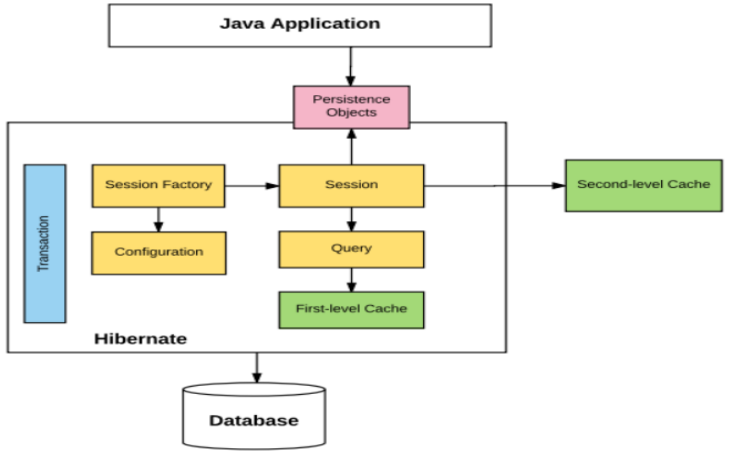
Session Factory | Session | Transaction

```java
@Override
public List<Item> getAll() {
    return openSession()
            .createNativeQuery("SELECT * FROM MatHang", Item.class)
            .getResultList();
}

@Override
public Item get(int id) {
    Item item = null;
    Session session = getCurrentSession();

    Transaction transaction = session.beginTransaction();
    try {
        item = session.get(Item.class, id);

        transaction.commit();
    } catch (Exception e) {
        transaction.rollback();
    }

    return item;
}


@Override
public boolean save(Item item) {
    Session session = openSession();
    Transaction transaction = session.beginTransaction();
    try {
        session.saveOrUpdate(item);

        transaction.commit();
    } catch (Exception e) {
        transaction.rollback();
    }

    return true;
}
```

```java
@Override
public List<ItemGroupDto> getItemGroupDtos() {
    NativeQuery<?> query = openSession().createNativeQuery(GET_ITEM_DTOS);

    query.addScalar(ItemGroupDto.ID, StandardBasicTypes.INTEGER)
         .addScalar(ItemGroupDto.NAME, StandardBasicTypes.STRING)
         .addScalar(ItemGroupDto.TOTAL_AMOUNT, StandardBasicTypes.INTEGER)
         .setResultTransformer(Transformers.aliasToBean(ItemGroupDto.class));

    return safeList(query);
}
```

Using @Transaction in some of back end framework to reduce boilerplate code

```java
@Transactional
public void save(User user) {
    Session currentSession = sessionFactory.getCurrentSession();
    currentSession.saveOrUpdate(user);
}
```



# Hibernate Architecture

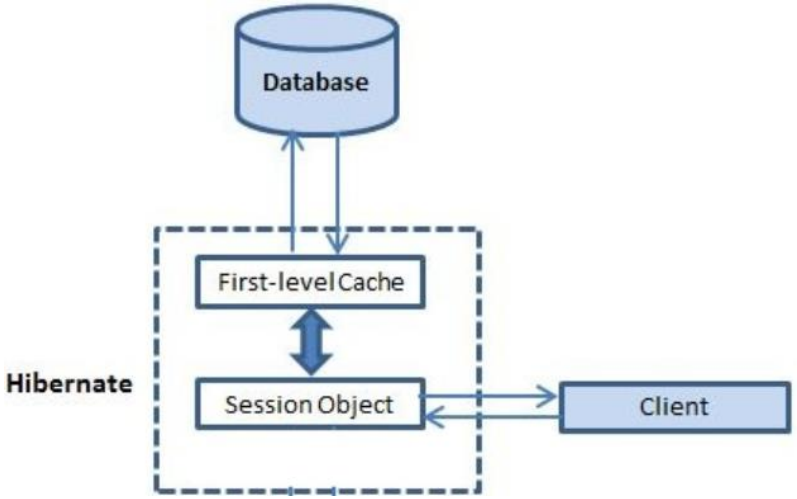
Cache

First Level Cache

- Caching is a facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction.

- Hibernate achieves the second goal by implementing first level cache

- Fist level cache is enabled by default and you do not need to do anything to get this functionality working. In fact, you can not disable it even forcefully.



# Hibernate Architecture

Cache

First Level Cache

- First level cache is associated with "session" object and other session objects in application can not see it.
- The scope of cache objects is of session. Once session is closed, cached objects are gone forever.
- First level cache is enabled by default and you can not disable it.
- When we query an entity first time, it is retrieved from database and stored in first level cache associated with hibernate session.
- If we query same object again with same session object, it will be loaded from cache and no sql query will be executed.
- The loaded entity can be removed from session using evict() method. The next loading of this entity will again make a database call if it has been removed using evict() method.
- The whole session cache can be removed using clear() method. It will remove all the entities stored in cache

# Hibernate Architecture

Cache

First Level Cache

```java
public void firstLevelCache() {
    Session session1 = openSession();
    Session session2 = openSession();

    Department d1 = session1.get(Department.class, "mgm-dn"); // YES
    System.out.println("d1: " + d1);
    System.out.println("d1s1 contains: " + session1.contains(d1)); // T
    System.out.println("d1s2 contains: " + session2.contains(d1)); // F

    System.out.println("======= clear/evict =======");
    session1.clear();

    System.out.println("d1s1 contains: " + session1.contains(d1)); // F
    System.out.println("d1s2 contains: " + session2.contains(d1)); // F

    Department d2 = session1.get(Department.class, "mgm-dn"); // YES
    System.out.println("d2: " + d2);

    Department d3 = session1.get(Department.class, "mgm-mu"); // YES
    System.out.println("d3: " + d3);

    Department d4 = session2.get(Department.class, "mgm-mu"); // YES
    System.out.println("d4: " + d4);
}
```

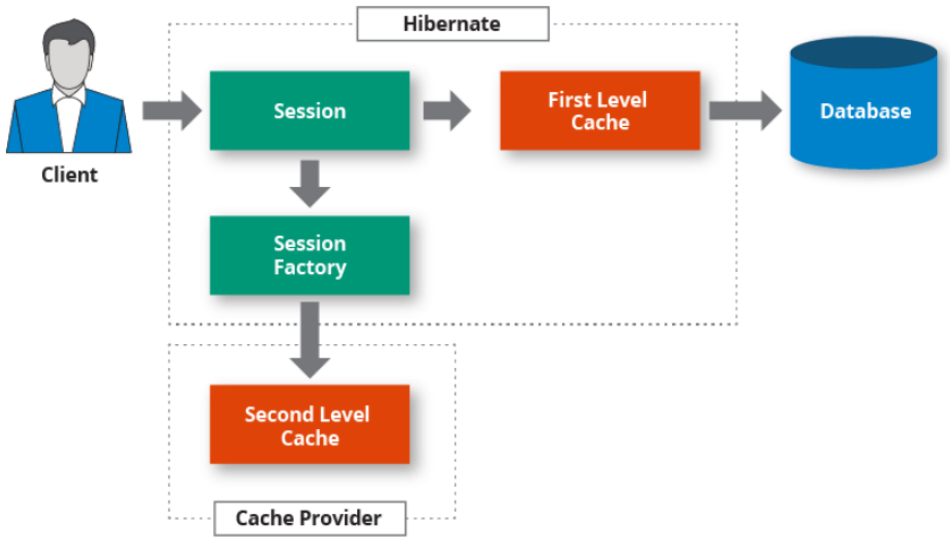
```
Hibernate:
    select
        department0_.dept_id as dept_id1_0_0_,
        department0_.dept_name as dept_nam2_0_0_
    from
        department department0_
    where
        department0_.dept_id=?
d1: Department [id=mgm-dn, name=mgm da nang]
d1s1 contains: true
d1s2 contains: false
======= clear/evict =======
d1s1 contains: false
d1s2 contains: false
Hibernate:
    select
        department0_.dept_id as dept_id1_0_0_,
        department0_.dept_name as dept_nam2_0_0_
    from
        department department0_
    where
        department0_.dept_id=?
d2: Department [id=mgm-dn, name=mgm da nang]
Hibernate:
    select
        department0_.dept_id as dept_id1_0_0_,
        department0_.dept_name as dept_nam2_0_0_
    from
        department department0_
    where
        department0_.dept_id=?
d3: Department [id=mgm-mu, name=mgm-munich]
d4: Department [id=mgm-mu, name=mgm-munich]
```

# Hibernate Architecture

Cache

Second Level Cache

- Fist level cache: This is enabled by default and works in session scope. Read more about hibernate first level cache.
- Second level cache: This is apart from first level cache which is available to be used globally in session factory scope. Available for all sessions



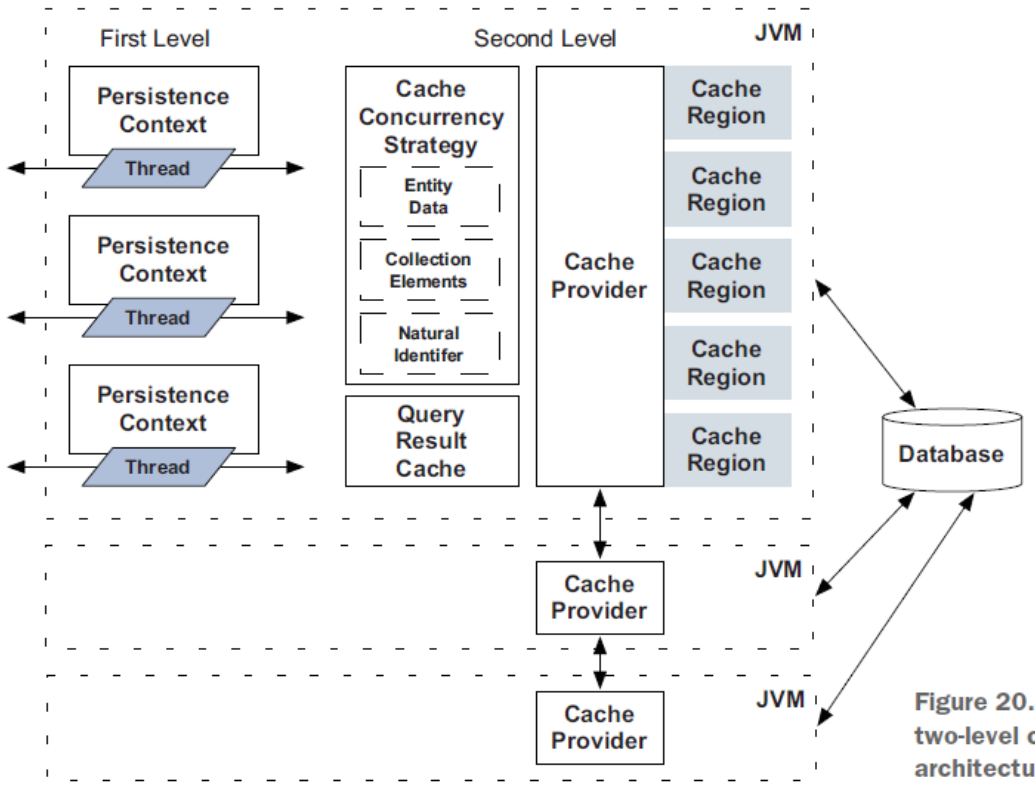




Figure 20.1 Hibernate's two-level cache architecture

## 1. Add ehcache dependency as cache provider

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
    <version>5.4.21.Final</version>
</dependency>
```

## 2. Add cache region to hibernate.cfg.xml

```
<property name="hibernate.cache.use_second_level_cache">true</property>
<property name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
<property name="hibernate.cache.provider_configuration_file_resource_path">ehcache.xml</property>
```

## 3. Add Cache Strategy to expected cached Entity

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
@Table (name = "department")
public class Department {
    @Id
    @Column(name = "dept_id")
    private String deptId;
```

```
encoding="UTF-8"?>
<ehcache>
    <cache name="persistence.Department"
        maxElementsInMemory="1000"
        timeToIdleSeconds="3"
        timeToLiveSeconds="20"
    />
    <!--
        maxElementsInMemory: elements limit in memory
        timeToIdleSeconds: out of memory after the last time access with duration
        timeToLiveSeconds: out of memory after the initial time access with duration
     -->

</ehcache>
```

# Hibernate Architecture

Configuration | Second Level Cache | EhCache

```java
public void secondLevelCache() {
    Session session1 = openSession();
    Session session2 = openSession();

    // db
    Department d1 = session1.get(Department.class, "mgm-dn");
    System.out.println("d1: " + d1);

    // 1st
    Department d2 = session1.get(Department.class, "mgm-dn");
    System.out.println("d2: " + d2);

    // database
    Department d3 = session1.createQuery("SELECT dp FROM Department dp WHERE dp.id =
            .setParameter("id", "mgm-mu")
            .getSingleResult();
    System.out.println("d3: " + d3);

    // native query, hibernate query: works with first level cache
    sleep(2);

    // 2nd
    Department d4 = session2.get(Department.class, "mgm-mu");
    System.out.println("d4: " + d4);
}
```

```
Hibernate:
    select
        department0_.dept_id as dept_id1_0_0_,
        department0_.dept_name as dept_nam2_0_0_
    from
        department department0_
    where
        department0_.dept_id=?
d1: Department [id=mgm-dn, name=mgm da nang]
d2: Department [id=mgm-dn, name=mgm da nang]
Hibernate:
    select
        department0_.dept_id as dept_id1_0_,
        department0_.dept_name as dept_nam2_0_
    from
        department department0_
    where
        department0_.dept_id=?
d3: Department [id=mgm-mu, name=mgm-munich]
d4: Department [id=mgm-mu, name=mgm-munich]
```

# Hibernate Proxy

```java
@Entity
@Table(name = "MatHang")
public class Item {
    @Id
    @Column(name = "MaMH")
    private Integer itemId;

    @Column(name = "TenMH")
    private String itemName;

    // nameValue: FK_ColumnName SubTable
    // referencedColumnNameValue: PK_ColumnName ParentTable
    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "MaLoai", referencedColumnName = "MaLoai")
    private ItemGroup itemGroup;

    @OneToOne(mappedBy = "item", fetch = FetchType.LAZY)
    private ItemDetail itemDetail;
```

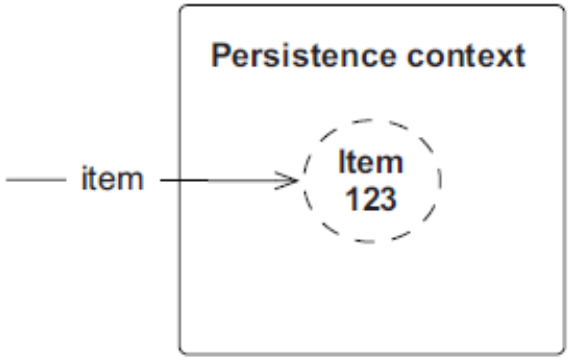
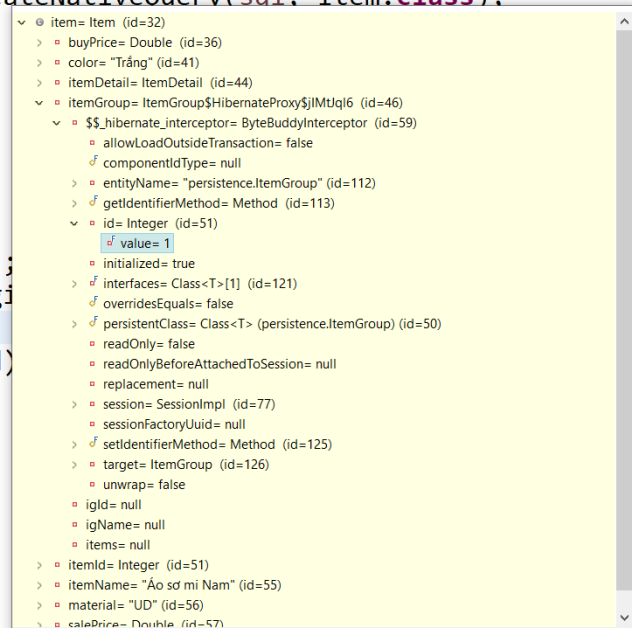



Figure 12.1 The persistence context contains an Item proxy.

```java
@Override
public List<Item> getAll() {
    Session session = openSession();
    String sql = "SELECT * FROM MatHang";
    NativeQuery<Item> query = session.createNativeQuery(sql, Item.class);
    return query.getResultList();
}

@Override
public Item get(int id) {
    // demo proxy
    Item item = null;
    Session session = getCurrentSession();
    Transaction transaction = session.begi
    try {
        item = session.get(Item.class, id)
        transaction.commit();
    } catch (Exception e) {
        e.printStackTrace();
        transaction.rollback();
    }
    return item;
}
```



## Configuration

Câu 1: Liệt kê tất cả các loại hàng

Câu 2: Liệt kê tất cả các mặt hàng

a. Chứa thông tin loại hàng

b. Lấy tất cả các mặt hàng của loại hàng đó → Câu 1

Câu 3: Liệt kê tất các mặt hàng theo MaMH theo 2 cách

a. Using Session – 1st level cache

b. Using native query return entity with no cache

## Configuration

Câu 4: Liệt kê các mặt hàng theo MaLoai

a. Giới thiệu Single Entity mapping với LoaiHang, MatHang

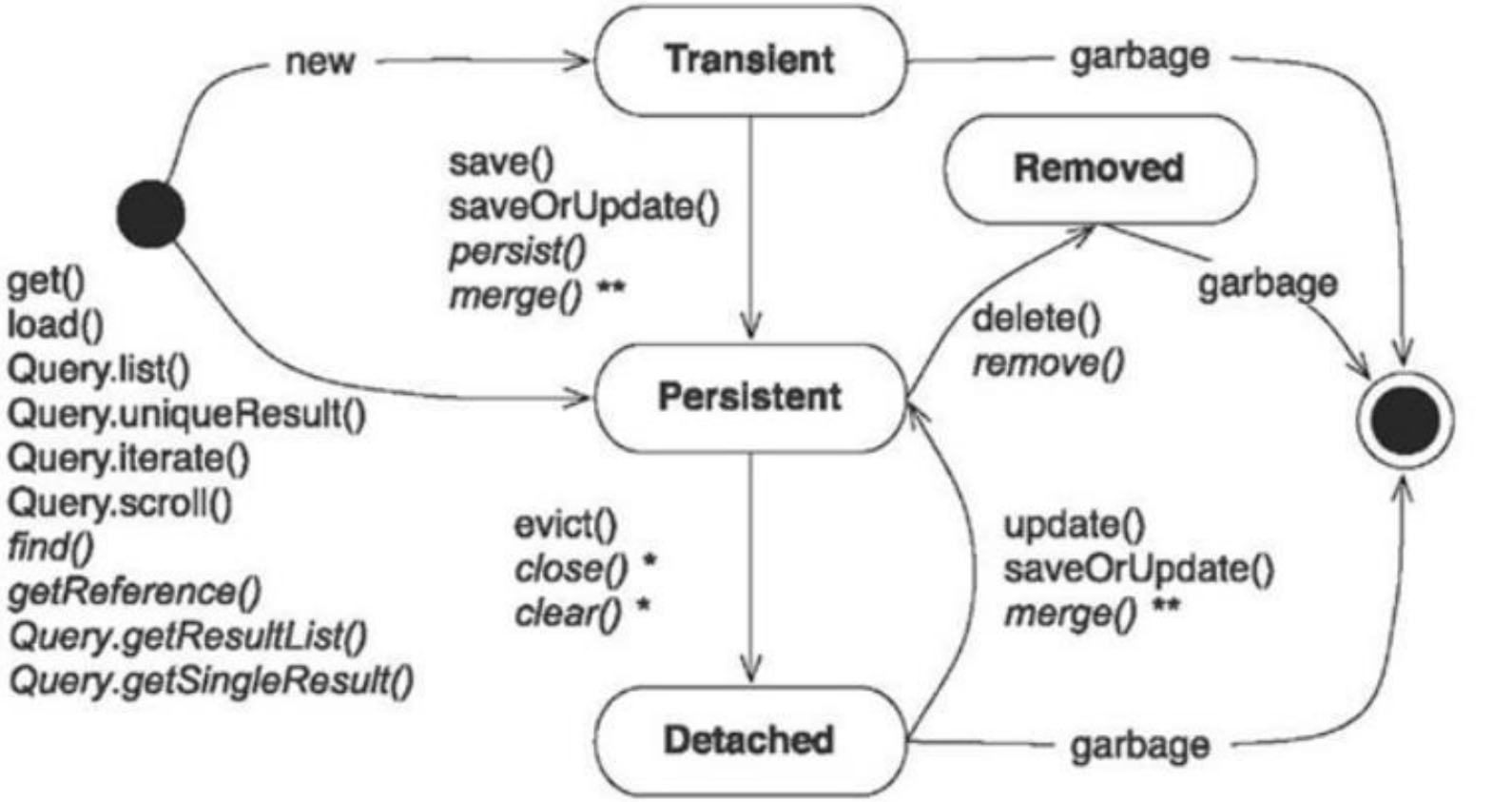
b. Giới thiệu OneToMany, ManyToOne với bidirectional

c. Sử dụng NativeQuery, HQL, NamedQuery

Câu 5: Đếm số lượng các mặt hàng theo từng loại hàng

# Persistence objects state

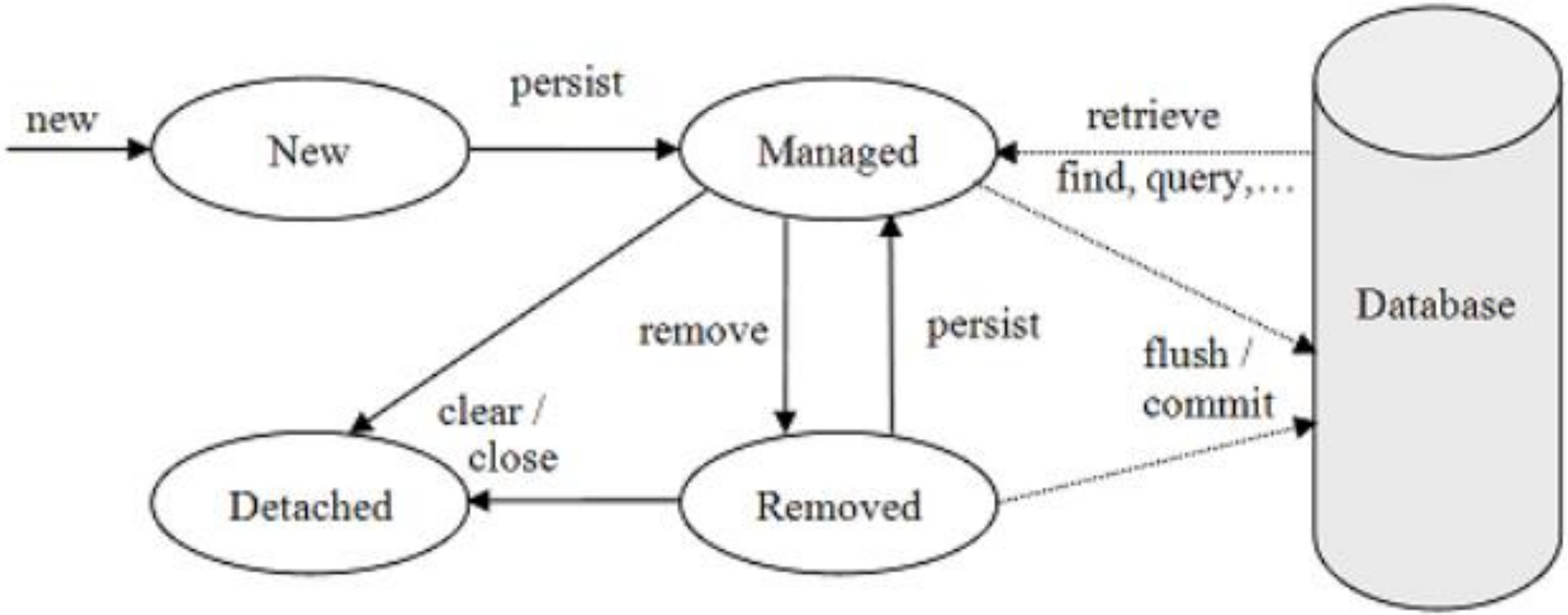
❖ State




* Hibernate & JPA, affects all instances in the persistence context
** Merging returns a persistent instance, original doesn't change state

# Persistence objects state

❖ State

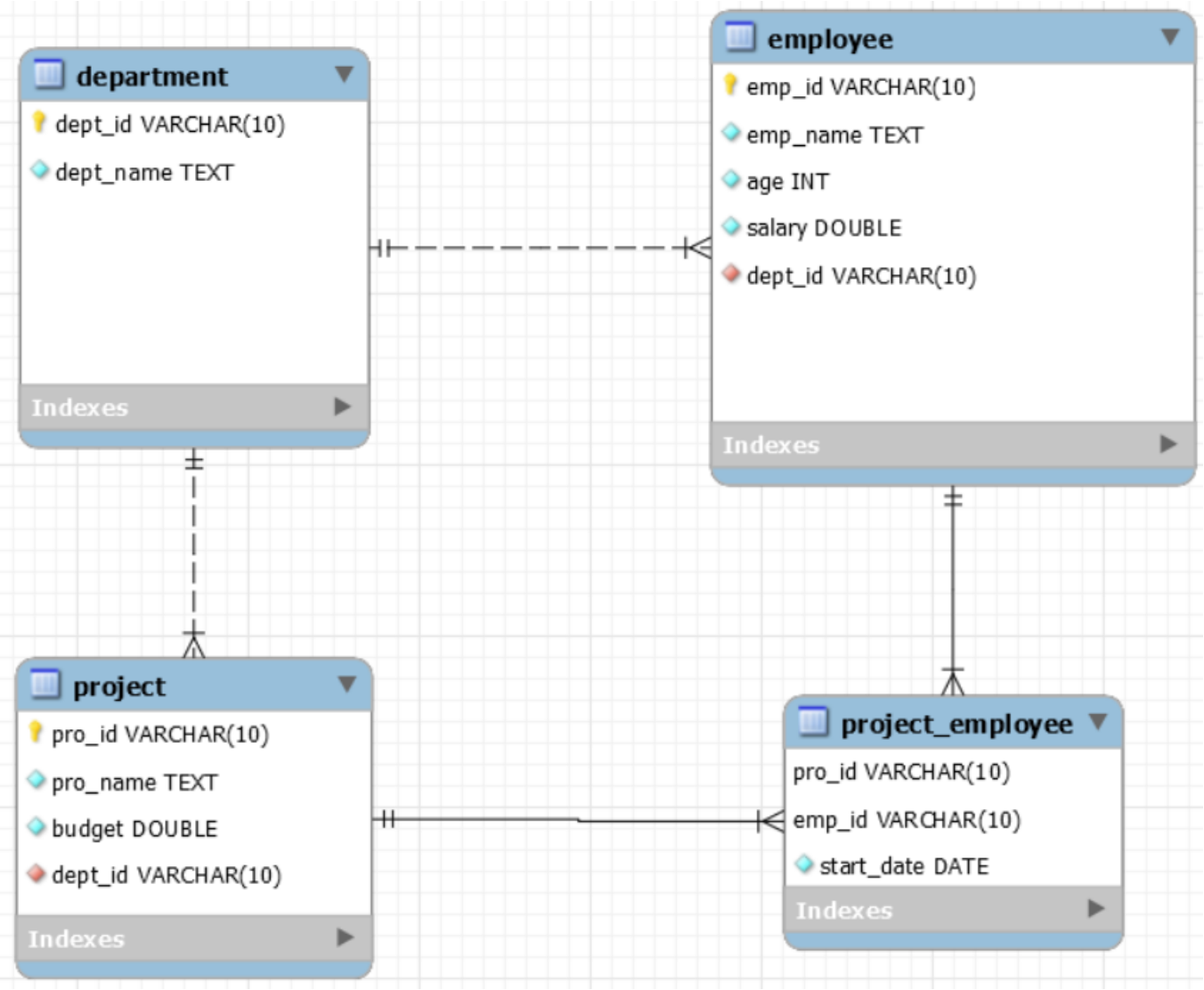


# Persistence objects state

- State
  - Persistence context: EntityManager.contains(entity) >> true
  - Detached:
  - Transient

# Review

❖ Provide a compman database



# Review

❖ 1. List all of employees by department id

Result Grid | Filter Rows: | Edi

| emp_id | emp_name | age | salary | dept_id |
|---|---|---|---|---|
| E1 | Quyen | 26 | 10 | mgm-dn |
| E2 | Hung | 27 | 20 | mgm-dn |
| E3 | Duy | 26 | 30 | mgm-dn |
| E4 | Nguyen | 30 | 20 | mgm-dn |
| E5 | Sa | 29 | 10 | mgm-dn |
| E6 | Hop | 28 | 20 | mgm-dn |
| NULL | NULL | NULL | NULL | NULL |

# Review

❖ 2. List all of employees from each department

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| dept_id | dept_name | emp_id | emp_name | age | salary | dept_id |
|---|---|---|---|---|---|---|
| mgm-by | mgm-bayern | E10 | Top | 28 | 30 | mgm-by |
| mgm-by | mgm-bayern | E9 | Cris | 26 | 20 | mgm-by |
| mgm-dn | mgm da nang | E1 | Quyen | 26 | 10 | mgm-dn |
| mgm-dn | mgm da nang | E2 | Hung | 27 | 20 | mgm-dn |
| mgm-dn | mgm da nang | E3 | Duy | 26 | 30 | mgm-dn |
| mgm-dn | mgm da nang | E4 | Nguyen | 30 | 20 | mgm-dn |
| mgm-dn | mgm da nang | E5 | Sa | 29 | 10 | mgm-dn |
| mgm-dn | mgm da nang | E6 | Hop | 28 | 20 | mgm-dn |
| mgm-mu | mgm-munich | E7 | Nadia | 30 | 10 | mgm-mu |
| mgm-mu | mgm-munich | E8 | Eric | 30 | 30 | mgm-mu |

## Review

❖ 3. List all projects of departments have budget greater than 400 main days

Result Grid | Filter Rows: | Expo

| dept_name | pro_name | budget |
|---|---|---|
| mgm-bayern | baking-schema | 420 |
| mgm-munich | allocation | 600 |
| mgm-munich | dispo-controlling | 800 |
| mgm-munich | dispol-list | 560 |

# Review

❖ 4. Show total of budget from each department in 2020

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

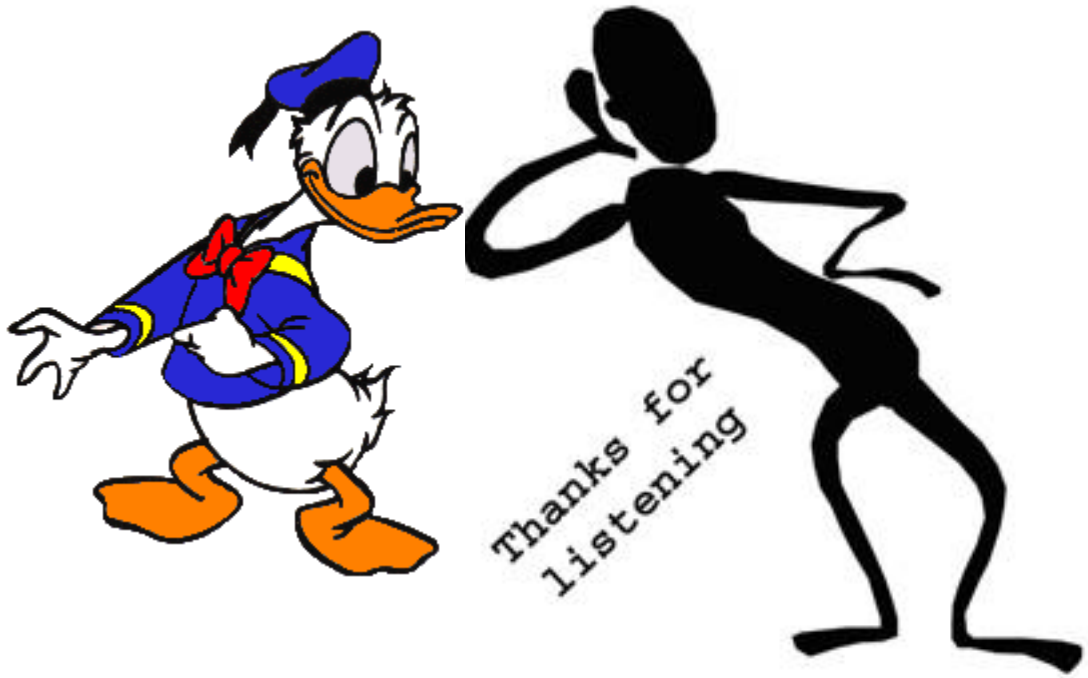
| dept_name | project_details | budget |
|---|---|---|
| mgm-bayern | accounting 120,baking-schema 420,master-data 360 | 900 |
| mgm da nang | A12 250 | 250 |
| mgm-munich | allocation 600,bake-off 260,dispo-controlling 800,dispol-list 560,volumn-plainning 400 | 2620 |

## Review

❖ 5. Show the department, project has highest budget in 2020

Result Grid | Filter Rows: | Export:

| | dept_name | pro_name | budget |
|---|---|---|---|
| ▶ | mgm-munich | dispo-controlling | 800 |




# END