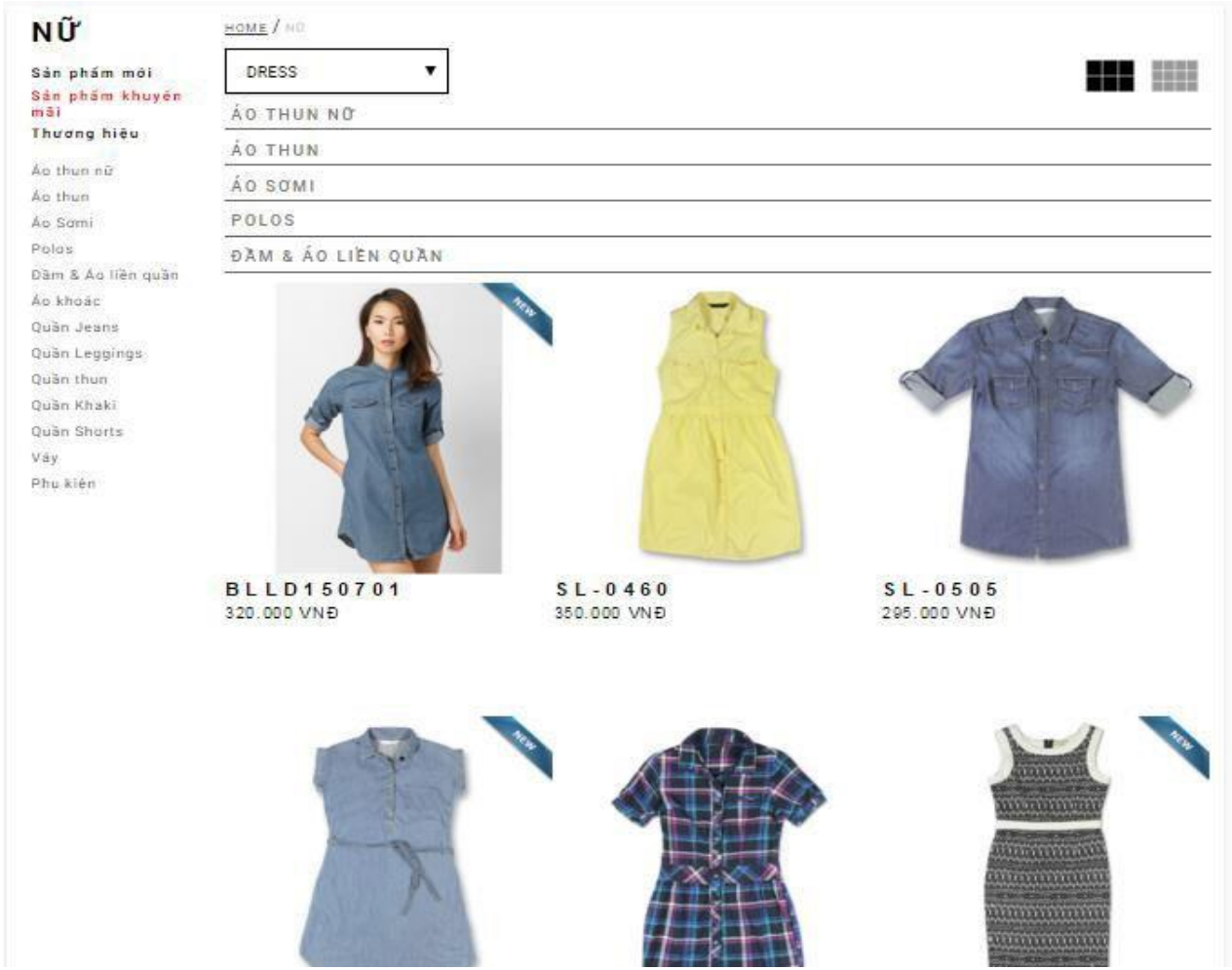
Bài 17

Database Connection

Kiến trúc ứng dụng

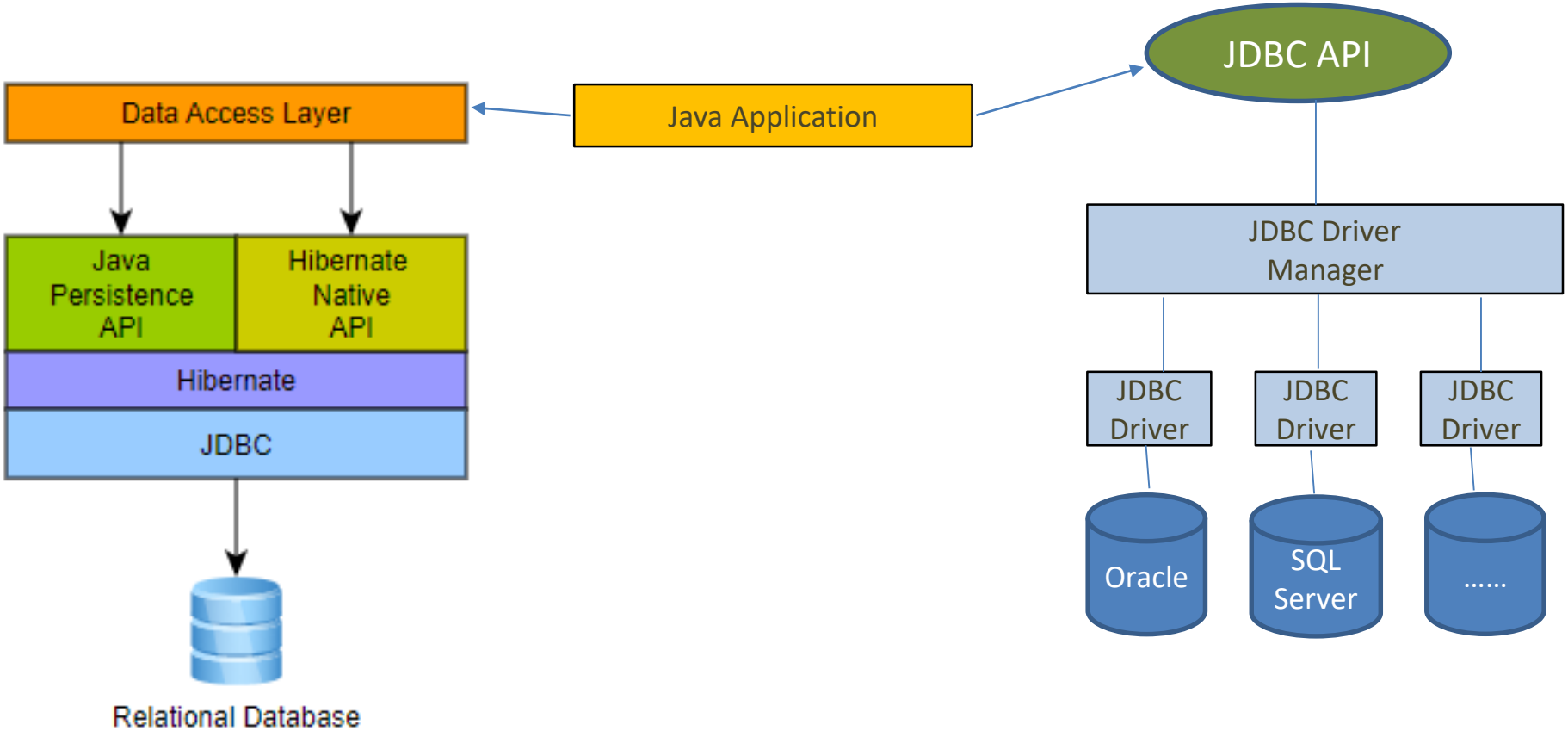
Java Application

❖ Overview



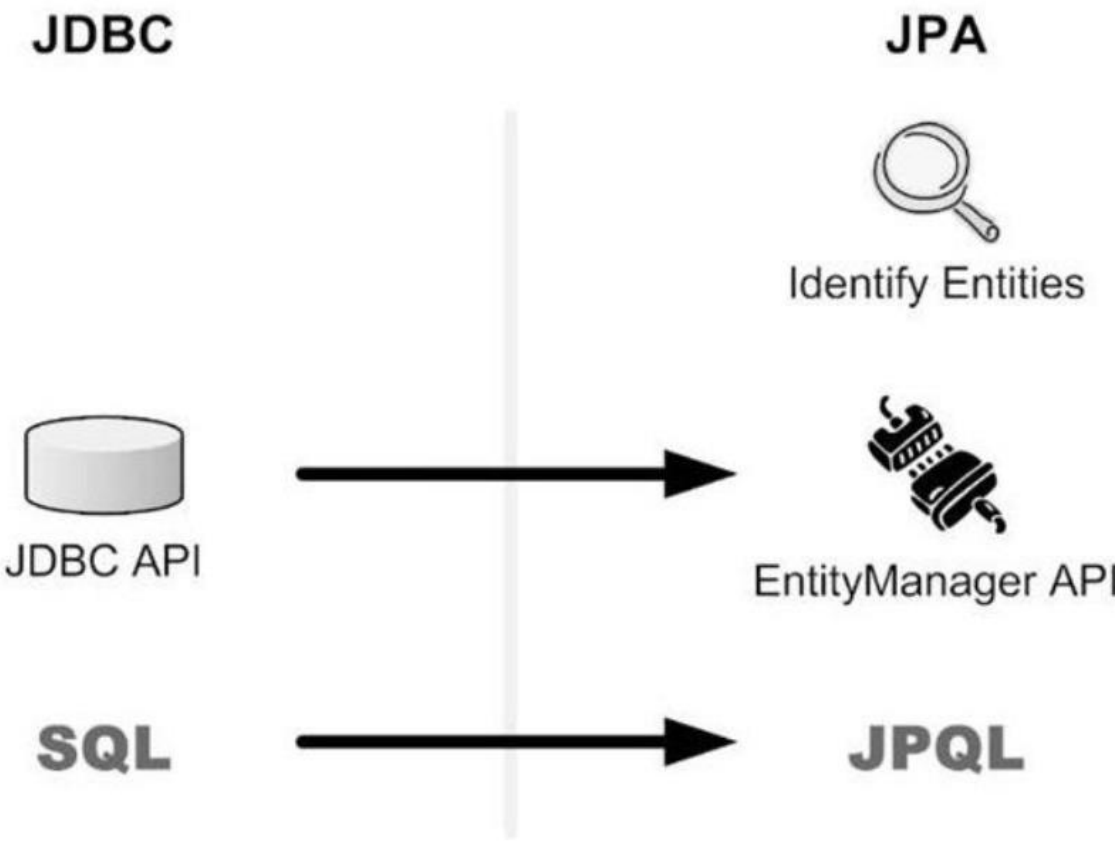
Kiến trúc ứng dụng

❖ Overview



Kiến trúc ứng dụng

❖ Overview



Kiến trúc ứng dụng

❖ Overview

JDBC in Java

Java Database Connectivity (**JDBC**) is an application programming interface (API) for the programming language **Java**, which defines how a client may access a database. It is a **Java**-based data access technology used for **Java** database connectivity. It is part of the **Java** Standard Edition platform, from Oracle Corporation.

Kiến trúc ứng dụng

❖ Overview

Java Persistence API (JPA) is ORM framework that helps you to manage the database transactions in much easier way compare to writing the plain SQL queries and invoking the JDBC statements.

Ultimately, JDBC is internally used. But, when you use JDBC, you have to manage multiple things and there is no direct mapping to the Java objects and Database tables. You are going to manually map every filed to tables. That is very time consuming and maintenance also difficult.

JPA does that in better way and it works good.

Kiến trúc ứng dụng

❖ Overview

What is the Java Persistence API?

The Java Persistence API which provides a specification for persisting, reading, managing data from your Java object to relational tables in the database.

Read more about JPA at **JPA Tutorial - Java Persistence API** (you will learn everything about JPA here)

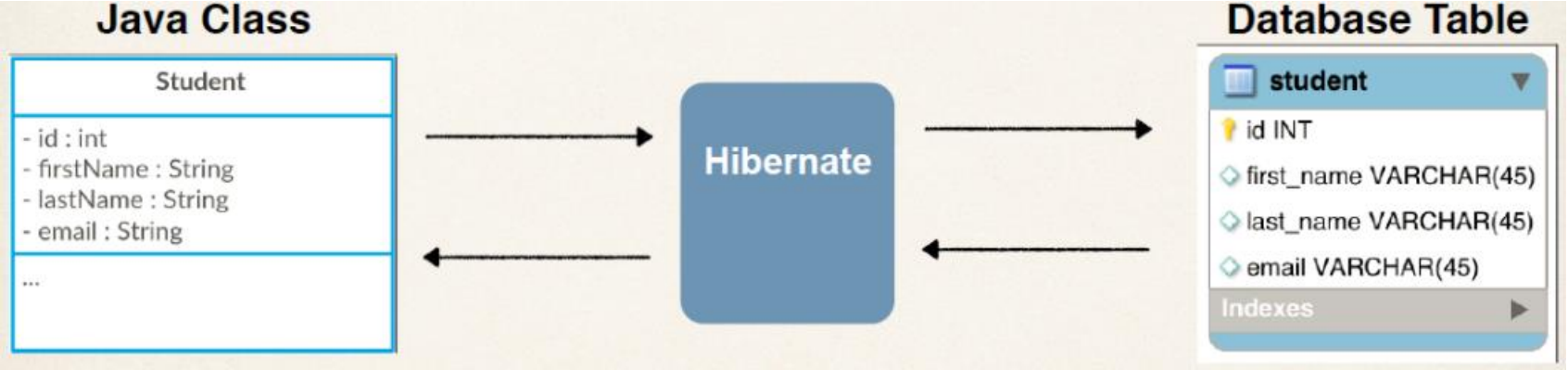
What is Hibernate Framework?

Hibernate is an `Object/Relational Mapping` solution for Java environments. Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables. Hibernate is a java based ORM tool that provides a framework for mapping application domain objects to the relational database tables and vice versa.

Hibernate provides a reference implementation of `Java Persistence API`, that makes it a great choice as an ORM tool with benefits of loose coupling.

Kiến trúc ứng dụng

❖ Overview

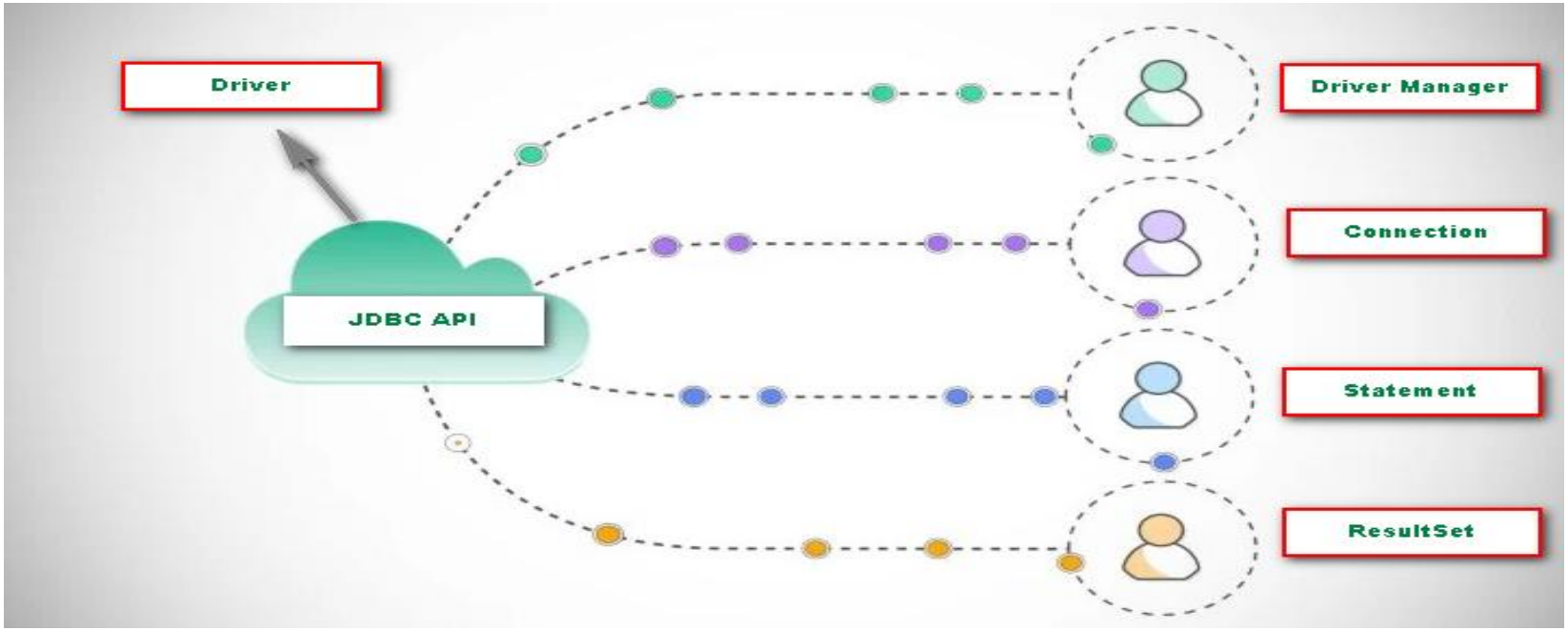



```java
@Entity
@Table(name = "department")
@Cache(usage = CacheConcurrencyStrat
public class Department {

    @Id
    @Column(name = "dept_id")
    private String id;

    @Column(name = "dept_name")
    private String name;
```

```java
/**
 * Returns the entity for the given id.
 *
 * @param id entity id
 * @return entity or <code>null</code>
 */
Entity get(Id id);

/**
 * Returns entities for given id collection.
 *
 * @param ids collection of ids
 * @param sortOrder sorting definition, may be {@code null}
 * @return list of entities, sorted if specified
 */
List<Entity> listByIds(Collection<Id> ids, SortOrder sortOrder);
```

Khái niệm

- JDBC là một Java API (Application programming interface) tiêu chuẩn dùng để kết nối và truy vấn với các loại cơ sở dữ liệu quan hệ.
- JDBC có một tập hợp các class và các Interface dùng cho ứng dụng Java có thể nói chuyện với các cơ sở dữ liệu.



Các thành phần chính

1. **DriverManager**:
   - Là class, được dùng để quản lý danh sách các **Driver** (database drivers).
2. ***Driver***:
   - Là một Interface, nó dùng để liên kết các liên lạc với cơ sở dữ liệu, điều khiển các liên lạc với database. Một khi Driver được tải lên, lập trình viên không cần phải gọi nó một cách cụ thể.
3. ***Connection*** :
   - Là một Interface với tất cả các method cho việc liên lạc với database. Nó mô tả nội dung liên lạc. tất cả các thông tin liên lạc với cơ sở dữ liệu là thông qua chỉ có đối tượng ***Connection***.
4. ***Statement/PrepareStatement***:
   - Là một Interface, **gói gọn một câu lệnh SQL** gửi tới cơ sở dữ liệu được phân tích, tổng hợp, lập kế hoạch và thực hiện.
5. ***ResultSet***:
   - ***ResultSet*** đại diện cho tập hợp các bản ghi lấy do thực hiện truy vấn.

Thư viện

Home » mysql » mysql-connector-java » 8.0.27

MySQL Connector/J » 8.0.27

JDBC Type 4 driver for MySQL

License	GPL 2.0
Categories	MySQL Drivers
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	(Oct 18, 2021)
Files	pom (2 KB) jar (2.4 MB) View All
Repositories	Central
Used By	5,690 artifacts

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.27</version>
</dependency>
```

Database connection

- B1: Add dependency

```
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.18</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.10</version>
    </dependency>
    <dependency>
        <groupId>commons-codec</groupId>
        <artifactId>commons-codec</artifactId>
        <version>1.10</version>
    </dependency>
</dependencies>
```

Database connection

- B2: Configuration and Connection

```java
public static Connection getConnection() {
    Properties props = DBProvider.getProperties();
    if (connection == null) {
        try {
            Class.forName(props.getProperty("DRIVER"));
            connection = DriverManager.getConnection(
                    props.getProperty("URL"),
                    props.getProperty("USER"),
                    props.getProperty("PASSWORD")
            );
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return connection;
}
```

```
1DRIVER = com.mysql.cj.jdbc.Driver
2URL = jdbc:mysql://localhost:3306/java10_shopping
3USER = root
4PASS = 1234
```

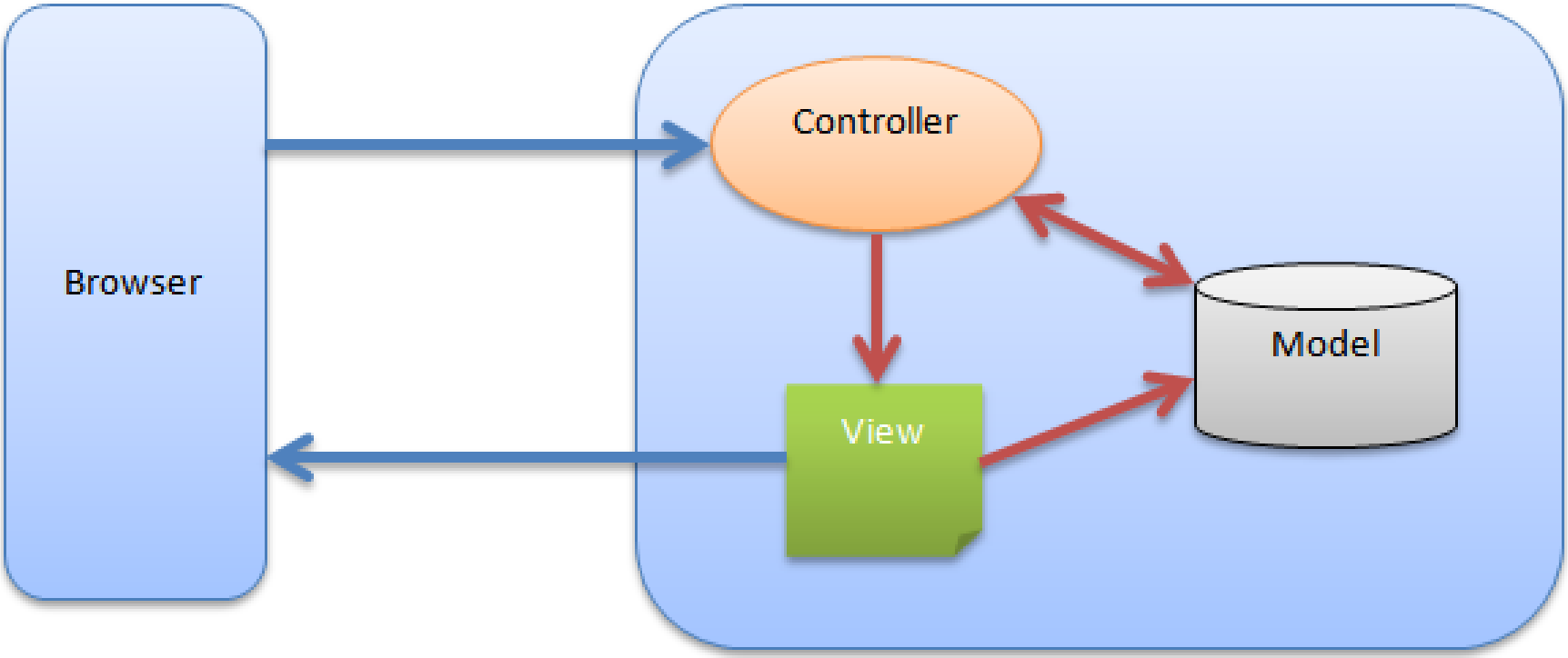
Database connection

- B3: Query data with Statement/PrepareStatement

```
B1: Write down a native query
B2: IF
    + UPDATE: => executeUpdate => TRUE|FALSE
    + SELECT: => executeQuery  => ResultSet (columns/rows)
                                => List<T> : T > Entity
```

```java
public List<ItemGroup> getAll() {
    List<ItemGroup> result = new ArrayList<>();
    String sql = "SELECT * FROM LoaiHang";
    try {
        st = conn.createStatement();
        rs = st.executeQuery(sql);
        while (rs.next()) {
            ItemGroup ig = new ItemGroup(rs.getInt("MaLoai"), rs.getString("TenLoai"));
            result.add(ig);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        // close connection
    }

    return result;
}
```

MÔ HÌNH MVC



MD5 – Message Digest

- MD5(message digest) là thuật toán mã hóa theo chuẩn RFC 1321. Các chương trình mã hóa của MD5 thường được gọi là MD5Sum

- Được dùng để tạo ra chuỗi 128 bit duy nhất từ một chuỗi dữ liệu nhập bất kì, được dùng để kiểm tra tính toàn vẹn dữ liệu của một tập tin.

- Một bảng băm MD5 thường biểu diễn bằng một hệ số thập lục phân 32 kí tự

- Năm 1996 người ta phát hiện ra một lỗ hỏng trong MD5. Và có thể được sử dụng để thay thể bởi các giải thuật khác như SHA-1, Bcrypt, RSA

RETURN_GENERATE_KEYS

```
try {
    int autoIdkey = -1;
    PreparedStatement pst = conn.prepareStatement(sql, PreparedStatement.RETURN_GENERATED_KEYS);
    pst.setString(1, "Lê Na");
    int result = pst.executeUpdate();
    if(result > 0){
        ResultSet rs = pst.getGeneratedKeys();
        while(rs.next()){
            autoIdkey = rs.getInt(1);
        }
    }
    System.out.println("Lastest ID: " + autoIdkey);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Khi bạn thêm một dòng dữ liệu với ID được generate AI
Sau khi Thêm xong
=> Làm sao mình in ra dữ liệu mà mình vừa thêm vào database ???

BATCH UPDATE

```
public boolean addBatch(List<ItemGroup> itemGroups) {
    boolean isSaved = false;
    String sql = "INSERT INTO LoaiHang(MaLoai, TenLoai)\n"
               + "VALUES(?, ?)";
    try {
        pst = conn.prepareStatement(sql);

        for (ItemGroup itemGroup: itemGroups) {
            pst.setInt(1, itemGroup.getId());
            pst.setString(2, itemGroup.getName());
            pst.addBatch();
        }
        isSaved = Arrays.stream(pst.executeBatch()).sum() > 0;
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        SqlUtils.close(rs, pst);
    }
    return isSaved;
}
```

“A batch update is a batch of updates grouped together, and sent to the database in one "batch", rather than sending the updates one by one”.

TRANSACTION MANAGEMENT

❖ Transactions Management

- Là một tập hợp các action được thực hiện lần lượt single, atomic action hoặc tất cả hoặc không một query nào được thực hiện

❖ Example

- Điển hình về giao dịch của các tài khoản ngân hàng. Bạn cần **chuyển 10$** từ một tài khoản này sang tài khoản kia. Bạn thực hiện bằng cách trừ 10$ từ tài khoản đầu tiên và thêm 10$ vào tài khoản thứ hai. Nếu quá trình này không thành công sau khi trừ 10$ từ tài khoản ngân hàng đầu tiên, 10$ sẽ không bao giờ được thêm vào tài khoản ngân hàng thứ hai. Số tiền bị mất sẽ đi vào **"secret space"**

❖ Khắc phục

- Để giải quyết vấn đề này, việc chuyển và nhận 10$ được nhóm vào một **transaction**. Nếu chuyển tiền thành công, nhưng nhận tiền không thành công, bạn có thể "**rollback**" trước trạng thái chuyển tiền. Bằng cách đó cơ sở dữ liệu được **để lại trong cùng một trạng thái như trước khi chuyển tiền** được thực hiện.

TRANSACTION MANAGEMENT

- Quá trình thực hiện

```
Connection connection = LibDBConnect.getConnectMySQL();
try {
    /** Auto Commit(false): Tất cả actions là một phần của transaction **/
    connection.setAutoCommit(false);
    /** Tạo và thực thi statements etc. **/

    /**
     *
     *
     */
    connection.commit();
} catch (Exception e) {
    try {
        connection.rollback();
    } catch (SQLException ex) {
        /** ignore **/
    }
} finally {
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            /** ignore **/
        }
    }
}
```

Thực hành

- Sử dụng cơ sở dữ liệu shopping đã học
  - Thực hiện chức năng mua hàng
    - Giảm số lượng của mặt hàng tương ứng
    - Cập nhật thông tin đơn hàng về số lượng mặt hàng
    - Thông tin
      - MaMH: 9003
      - SoDH: 100102
  - Rollback khi hệ thống xảy ra vấn đề

TRANSACTION MANAGEMENT

```
String receiptSql = "UPDATE ttdonhang"
        + " SET SoLuong = (SoLuong + 7)"
        + " WHERE MaMH = 9003 AND SoDH = 100102";

String itemSql = "UPDATE mathang"
        + " SET SoLuong = (SoLuong - 7)"
        + " WHERE MaMH = 9003";
```

```
try {
    conn.setAutoCommit(false);
    pst = conn.prepareStatement(receiptSql);
    pst.executeUpdate();

    // system error
    System.out.println(5/2);

    pst = conn.prepareStatement(itemSql);
    pst.executeUpdate();

    conn.commit();
} catch (Exception e) {
    try {
        conn.rollback();
    } catch (SQLException e1) {}
}
return 0;
```

Call Stored Procedure

Calling Stored Procedures in Java DB and MySQL

The following excerpt from the method `runStoredProcedures`, calls the stored procedure `SHOW_SUPPLIERS` and prints the generated result set:

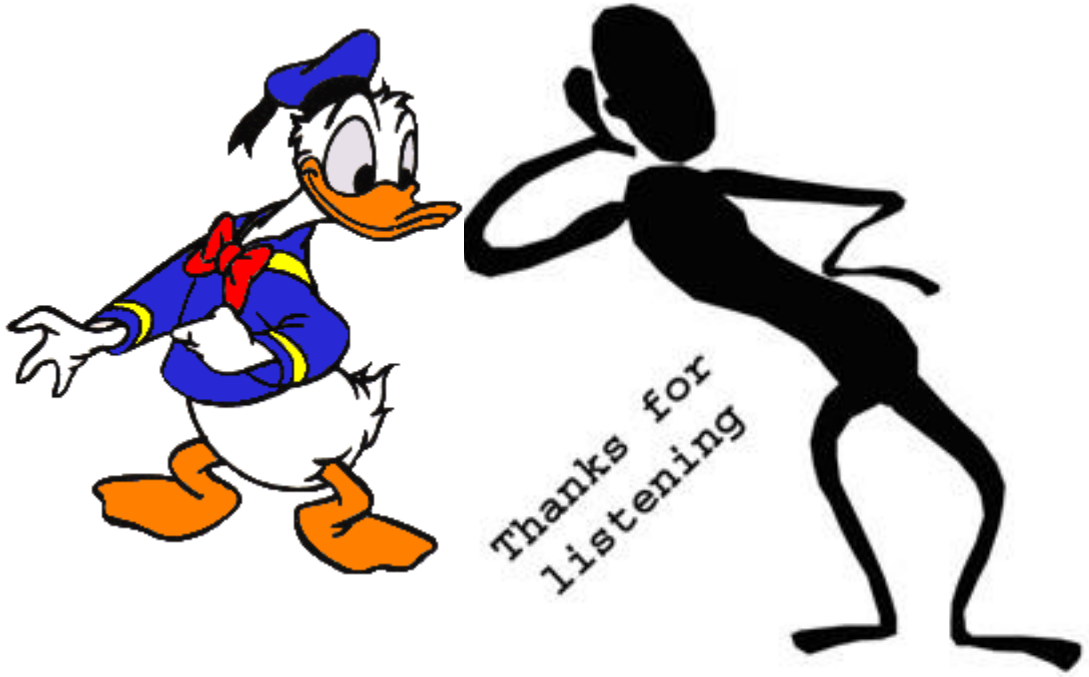
```
cs = this.con.prepareCall("{call SHOW_SUPPLIERS()}");
ResultSet rs = cs.executeQuery();

while (rs.next()) {
    String supplier = rs.getString("SUP_NAME");
    String coffee = rs.getString("COF_NAME");
    System.out.println(supplier + ": " + coffee);
}
```

The following excerpt from the method `runStoredProcedures`, calls the stored procedure `RAISE_PRICE`:

```
cs = this.con.prepareCall("{call RAISE_PRICE(?,?,?)}");
cs.setString(1, coffeeNameArg);
cs.setFloat(2, maximumPercentageArg);
cs.registerOutParameter(3, Types.NUMERIC);
cs.setFloat(3, newPriceArg);

cs.execute();
```

Call Stored Procedure

```
@Override
public Integer getResult(int p_max) {
    final String query = "{? = CALL F_SUM_OF_ODD_DIGITS(?)}";
    try {
        CallableStatement  cst = conn.prepareCall(query);
        cst.registerOutParameter(1, Types.INTEGER);

        cst.setInt(2, 10);
        cst.execute();

        return cst.getInt(1);

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```



END