**1. Cấu trúc file cấu hình Json (mappings.json)**

**1.1 Cấu hình dữ liệu nguồn và đích**

"sqlServer": {

        "url": "jdbc:sqlserver://HoangTam\\TAMHOANG;encrypt=true;trustServerCertificate=true;databaseName=NhanVienDB",

        "username": "sa",

        "password": "123456"

},

"mysql": {

      "url": "jdbc:mysql://localhost:3306/QLNhanVienDB",

        "username": "root",

        "password": "123456"

}

**-** Dữ liệu nguồn được cung cấp “*sqlserver*”và dữ liệu đích là *“mysql”* bao gồm các key : *url, username, password* dùng cho mục đích kết nối đến các nguồn dữ liệu.

**1.2 Cấu hình cơ chế mapping table**

{

    "tables": {

        "tableName": {

            "sqlServerTable": "sqlServerTableName",

            "mySqlTable": "mySqlTableName"

        },

        "tableName": {

            "sqlServerTable": "sqlServerTableName",

            "mySqlTable": "mySqlTableName"

        },

        "tableName": {

            "sqlServerTable": "sqlServerTableName",

            "mySqlTable": "mySqlTableName"

        }, ....

    }

}

Cấu trúc JSON ở trên là một mapping table, được sử dụng để ánh xạ các bảng trong một hệ thống cơ sở dữ liệu sang các bảng trong một hệ thống cơ sở dữ liệu khác. Trong ví dụ này, các bảng được ánh xạ từ SQL Server sang MySQL.

Mỗi bảng trong mapping table có một khóa chính, là tên của bảng. Các giá trị khác của bảng là tên của các bảng trong hệ thống cơ sở dữ liệu đích. Ví dụ, bảng "*tableName*" được ánh xạ sang bảng "*sqlServerTableName*" trong SQL Server và bảng "*mySqlTableName*" trong MySQL.

Mapping table có thể được sử dụng để di chuyển dữ liệu từ một hệ thống cơ sở dữ liệu sang một hệ thống cơ sở dữ liệu khác. Nó cũng có thể được sử dụng để tạo các truy vấn chéo giữa các hệ thống cơ sở dữ liệu.

**1.3 Cấu hình cơ chế mappings fields**

{

    "fields": {

        "ChucVuFields": {

            "sqlServerFields": {

                "id": "idChucVu",

                "name": "tenChucVu"

            },

            "mySqlFields": {

                "id": "idChucVu",

                "name": "tenChucVu"

            }

        }

    }

}

Cấu trúc JSON ở trên là một mapping table, được sử dụng để ánh xạ các trường trong một bảng SQL Server sang các trường trong một bảng MySQL. Trong ví dụ này, các trường được ánh xạ từ bảng "ChucVu" trong SQL Server sang bảng "ChucVu" trong MySQL.

Mỗi trường trong mapping table có một khóa chính, là tên của trường. Các giá trị khác của trường là tên của các trường trong hệ thống cơ sở dữ liệu đích. Ví dụ, trường "id" trong SQL Server được ánh xạ sang trường "idChucVu" trong MySQL và trường "name" trong SQL Server được ánh xạ sang trường "tenChucVu" trong MySQL.

Mapping table có thể được sử dụng để di chuyển dữ liệu từ một hệ thống cơ sở dữ liệu sang một hệ thống cơ sở dữ liệu khác. Nó cũng có thể được sử dụng để tạo các truy vấn chéo giữa các hệ thống cơ sở dữ liệu.

Dưới đây là cách giải thích cơ chế mapping field ở trên:

* Mỗi trường trong mapping table có một khóa chính, là tên của trường. Điều này cho phép bạn dễ dàng tìm thấy trường trong mapping table.
* Các giá trị khác của trường là tên của các trường trong hệ thống cơ sở dữ liệu đích. Điều này cho phép bạn ánh xạ các trường trong một hệ thống cơ sở dữ liệu sang các trường trong một hệ thống cơ sở dữ liệu khác.
* Mapping table có thể được sử dụng để di chuyển dữ liệu từ một hệ thống cơ sở dữ liệu sang một hệ thống cơ sở dữ liệu khác. Điều này có thể được thực hiện bằng cách sử dụng các công cụ di chuyển dữ liệu hoặc bằng cách viết các truy vấn SQL.
* Mapping table cũng có thể được sử dụng để tạo các truy vấn chéo giữa các hệ thống cơ sở dữ liệu. Điều này cho phép bạn truy vấn dữ liệu từ nhiều hệ thống cơ sở dữ liệu cùng một lúc.

**2. Xây dựng các lớp trừu tượng hóa đối tượng file cấu hình**

**2.1 Lớp DatabaseConfig**

package tamit0.mappingdata.Configuration;

public class DatabaseConfig {

    private String url;

    private String username;

    private String password;

    public String getUrl() {

        return url;

    }

    public void setUrl(String url) {

        this.url = url;

    }

    public String getUsername() {

        return username;

    }

    public void setUsername(String username) {

        this.username = username;

    }

    public String getPassword() {

        return password;

    }

    public void setPassword(String password) {

        this.password = password;

    }

}

Lớp ***DatabaseConfig*** được sử dụng để lưu cấu hình của một cơ sở dữ liệu.

**2.2 Lớp Table Config**

package tamit0.mappingdata.Configuration;

import java.util.Map;

import com.google.gson.annotations.SerializedName;

public class TableConfig {

    @SerializedName("sqlServerTable")

    private String sqlServerTableName;

    @SerializedName("mySqlTable")

    private String mySqlTableName;

    @SerializedName("fields")

    private Map<String, FieldConfig> fields;

    @SerializedName("model")

    private String model;

    @SerializedName("rule")

    private String rule;

    public String getSqlServerTableName() {

        return sqlServerTableName;

    }

    public void setSqlServerTableName(String sqlServerTableName) {

        this.sqlServerTableName = sqlServerTableName;

    }

    public String getMySqlTableName() {

        return mySqlTableName;

    }

    public void setMySqlTableName(String mySqlTableName) {

        this.mySqlTableName = mySqlTableName;

    }

    public Map<String, FieldConfig> getFields() {

        return fields;

    }

    public void setFields(Map<String, FieldConfig> fields) {

        this.fields = fields;

    }

    public String getModel() {

        return model;

    }

    public void setModel(String model) {

        this.model = model;

    }

    public String getRule() {

        return rule;

    }

    public void setRule(String rule) {

        this.rule = rule;

    }

}

Lớp ***TableConfig*** được sử dụng để lưu cấu hình của một bảng trong tệp JSON.

**2.3 Lớp FieldConfig**

package tamit0.mappingdata.Configuration;

import java.util.Map;

import com.google.gson.annotations.SerializedName;

public class FieldConfig {

    @SerializedName("sqlServerFields")

    private Map<String, String> sqlServerFields;

    @SerializedName("mySqlFields")

    private Map<String, String> mySqlFields;

    public Map<String, String> getSqlServerFields() {

        return sqlServerFields;

    }

    public void setSqlServerFields(Map<String, String> sqlServerFields) {

        this.sqlServerFields = sqlServerFields;

    }

    public Map<String, String> getMySqlFields() {

        return mySqlFields;

    }

    public void setMySqlFields(Map<String, String> mySqlFields) {

        this.mySqlFields = mySqlFields;

    }

}

Lớp ***FieldConfig*** được sử dụng để lưu cấu hình của một trường trong bảng. Cấu hình bao gồm các thông tin sau:

* Các trường trong SQL Server
* Các trường trong MySQL

Thông tin này được sử dụng để ánh xạ các trường giữa SQL Server và MySQL. Ví dụ, nếu trường id trong SQL Server được ánh xạ sang trường idNhanVien trong MySQL, thì các giá trị của trường id trong SQL Server sẽ được sử dụng để điền vào trường idNhanVien trong MySQL.

Lớp FieldConfig có thể được sử dụng để đọc và ghi cấu hình của một trường trong bảng. Điều này có thể được thực hiện bằng cách sử dụng các phương thức getSqlServerFields() và getMySqlFields()

**2.4 Lớp AppConfig**

package tamit0.mappingdata.Configuration;

import java.util.Map;

import com.google.gson.annotations.SerializedName;

public class AppConfig {

    private DatabaseConfig sqlServer;

    private DatabaseConfig mysql;

    @SerializedName("tables")

    private Map<String, TableConfig> tableConfigs;

    public DatabaseConfig getSqlServer() {

        return sqlServer;

    }

    public DatabaseConfig getMysql() {

        return mysql;

    }

    public Map<String, TableConfig> getTables() {

        return tableConfigs;

    }

}

Lớp ***AppConfig*** được sử dụng để lưu cấu hình của ứng dụng. Cấu hình bao gồm các thông tin sau:

* Cấu hình của cơ sở dữ liệu SQL Server
* Cấu hình của cơ sở dữ liệu MySQL
* Cấu hình của các bảng

Thông tin này được sử dụng để kết nối với các cơ sở dữ liệu và ánh xạ các bảng giữa các cơ sở dữ liệu.

Lớp ***AppConfig*** có thể được sử dụng để đọc và ghi cấu hình của ứng dụng. Điều này có thể được thực hiện bằng cách sử dụng các phương thức *getSqlServer(), getMysql()* và *getTables()*.

Trong trường hợp của tệp JSON mà bạn đã cung cấp ban đầu, lớp AppConfig có thể được sử dụng để lưu cấu hình của hai cơ sở dữ liệu: SQL Server và MySQL. Cấu hình của cơ sở dữ liệu SQL Server được lưu trong biến sqlServer và cấu hình của cơ sở dữ liệu MySQL được lưu trong biến mysql. Cấu hình của các bảng được lưu trong biến *tableConfigs*.

Mỗi bảng trong cấu hình được ánh xạ sang một lớp *TableConfig*. Lớp *TableConfig* lưu trữ thông tin về tên của bảng, tên của các trường trong bảng và ánh xạ giữa các trường trong SQL Server và MySQL.

**3. Xây dựng API nguồn**

- Để xây dựng API lấy toàn bộ dữ liệu có trong dữ liệu nguồn, cụ thể ở đây là SQL Server. Đầu tiền, chúng ta cần xây dựng lớp kết nối đến dữ liệu. Dưới đây là lớp kết nối:

**3.1 Lớp ConnectionService**

package tamit0.mappingdata.Services;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import com.google.gson.Gson;

import com.google.gson.JsonIOException;

import com.google.gson.JsonSyntaxException;

import tamit0.mappingdata.Configuration.AppConfig;

public class ConnectionService {

    public static AppConfig getJsonMapper() {

        String jsonFilePath = "E:\\JavaCS445\\mappingdata\\src\\main\\resources\\test.json";

        // Tạo một đối tượng Gson

        Gson gson = new Gson();

        // Đọc JSON từ file và chuyển thành đối tượng AppConfig

        try {

            return gson.fromJson(new FileReader(jsonFilePath), AppConfig.class);

        } catch (JsonSyntaxException e) {

            e.printStackTrace();

        } catch (JsonIOException e) {

            e.printStackTrace();

        } catch (FileNotFoundException e) {

            e.printStackTrace();

        }

        return null;

    }

    public static Connection getConnection(String databaseType) throws SQLException {

        if (databaseType.equals("sqlserver")) {

            return DriverManager.getConnection(getJsonMapper().getSqlServer().getUrl(), getJsonMapper().getSqlServer().getUsername(), getJsonMapper().getSqlServer().getPassword());

        } else if (databaseType.equals("mysql")) {

            return DriverManager.getConnection(getJsonMapper().getMysql().getUrl(), getJsonMapper().getMysql().getUsername(), getJsonMapper().getMysql().getPassword());

        } else {

            throw new SQLException("Invalid database type: " + databaseType);

        }

    }

}

Trong lớp này có hai phương thức :

* *getJsonMapper()* : phương thức này dùng để đọc tất cả các cấu hình ở file cấu hình “*mappings.json*” và lưu vào đối tượng *AppConfig* đã được định nghĩa ở trên.
* *getConnection()* :sau khi có dữ liệu từ phương thức *getJsonMapper()*, phương thức này có nhiệm vụ sử dụng các thông số đã được cấu hình để tiến hành kết nối đến các nguồn dữ liệu, cụ thể là *sqlserver* và *mysql.*

- Ở đây để làm việc với Json, chúng tôi đã chọn thư viện Gson của Google. Dưới đây là cách import thư viện trên vào dự án Spring boot:

<dependency>

            <groupId>com.google.code.gson</groupId>

            <artifactId>gson</artifactId>

            <version>2.10.1</version>

</dependency>

- Chúng ta có thể import đoạn code ở trên vào file *pom.xml* trong dự án.

**3.2 Lớp MappingController**

Để xây dựng API ở trên, chúng ta sử dụng *@RestController* và xây dựng API *getAllData()* trong controller này.

- Việc lấy toàn bộ dữ liệu từ dữ liệu nguồn thông qua các phương thức dưới đây :

public List<Object> getResultSet(String tableName, List<Object> jsonObjectList) {

        String sql = getStrSqlSelect(tableName);

        try (PreparedStatement statement = ConnectionService.getConnection("sqlserver").prepareStatement(sql)) {

            ResultSet rs = statement.executeQuery();

            // Iterate over the result set and add the rows to the list of JSON objects.

            while (rs.next()) {

                JsonObject jsonObject = new JsonObject();

                for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {

                    jsonObject.addProperty(rs.getMetaData().getColumnName(i), rs.getObject(i).toString());

                }

                jsonObjectList.add(jsonObject);

            }

            return jsonObjectList;

        } catch (SQLException | JSONException e) {

            e.printStackTrace();

            return null;

        }

    }

* Phương thức : *getResultSet()* dùng để đọc tất cả các dữ liệu có trong dữ liệu nguồn và lưu vào đối tượng Resultset sau đó sử dụng thư viện Gson để thực hiện việc chuyển dữ liệu trong Resultset sang List<JsonObject> để có cấu trúc object tương ứng model.

public static String getStrSqlSelect(String tableName)

    {

        String sql = "SELECT ";

        int count = 0;

        for (var strField : getFieldOfTable(tableName))

        {

            if (++count == getFieldOfTable(tableName).size())

            {

                sql += strField + "";

            }

            else

            {

                sql += strField + ", ";

            }

        }

        sql += " FROM "  + tableName;

        return sql;

    }

* Phương thức : *getStrSqlSelect() :* dùng để trả về câu lệnh Select Sql với tableName được cung cấp và các trường tương ứng được cấu hình trong file “mapping.json”.

public static List<Object> getFieldOfTable(String tableName)

    {

        List<Object> listFieldOfTable = new ArrayList<>();

        TableConfig tableConfig = ConnectionService.getJsonMapper().getTables().get(tableName);

        if (tableConfig != null)

        {

            for (String fieldName : tableConfig.getFields().keySet())

            {

                for (String fieldMySqlTable : tableConfig.getFields().get(fieldName).getMySqlFields().values())

                {

                    listFieldOfTable.add(fieldMySqlTable);

                }

            }

        }

        return listFieldOfTable;

    }

* Phương thức : *listFieldOfTable()* dùng để đọc tất cả các field có trong table và lưu vào List<Object> dùng để sinh ra câu lệnh Sql ở trên.

- Sau khi hoàn thành các phương thức ở trên chúng ta đi vào xây dựng API trả về tất cả dữ liệu của dữ liệu nguồn.

@GetMapping("/getAllData")

    public String getAllData() throws Exception {

        var listJsonObject = new ArrayList<>();

        // Duyệt qua tất cả các table được cấu hình trong file mappings.json

        for (TableConfig tableConfig : ConnectionService.getJsonMapper().getTables().values()) {

            var listObject = new ArrayList<>();

            listObject.clear();

            listJsonObject.addAll(getResultSet(tableConfig.getSqlServerTableName(), listObject));

        }

        return (String) (new Gson().toJson(listJsonObject));

    }

- Ở đây chúng ta sử dụng thư viện Gson để thực hiện việc chuyển List<object> trả về chuỗi Json.

- Như vậy chúng ta đã hoàn thành việc cấu hình cũng như xây dựng các lớp trừu tượng dữ liệu cấu hình và xây dựng API để lấy dữ liệu nguồn.