Database Project2

Tianao Wang,Yancheng Mo

June 2, 2022

Abstract

In order to help SUSTC, we optimized the database and designed a seres of APIs on the database.Firstly, We design an Import API that import the data given by teacher into databse.And then, we implement 13 API functions.

About the the implement of the real back-end server, we use the springboot to implement MVC model with the mapper/dao/service/model level and provide HTTP web services. In the springboot, we use the Hikari as the database connection pools. Also, we use the html and css to make a usable and beautiful GUI for the users. What's more, we use user privileges, indexing, and views to improve user experience. Finally, we use redis database as the distributed access to support the high-concurrent in the snap up scene and pass the proper pressure tests by using apache-jmeter-5.4.

We deployed our web on the Linux server.You can visit our web by http://106.55.104.82:8080/

1 Import original Data

We didn't use JDBC to import the origanal data like we used to do in project1.Instead, We used the Dao functions that Springboot supplies.Then we Encapsulated these functions that are used to import different csv files into one API named ImportService.

```
public interface ImportService {
    void importStaff();
    void importCenter();
    void importModel();
    void importEnterPrise();
}
```

2 Basic Requirements

2.1 API Specification

2.1.1 APIs for manipulating the original data:

There are some API operations that can access the original data, including insert, select, delete and update.Take staff as example:

```
public interface IDSUService {
 boolean InsertStaffInfo(String ageS,String gender,String phoneS,String
     staff_name,String numberS,String type,String supply_center);
 boolean DeleteStaffInfo(String ageS,String gender,String phoneS,String
     staff_name,String numberS,String type,String supply_center);
 boolean UpdateStaffInfo(String ageS,String gender,String phoneS,String
     staff_name,String numberS,String type,String supply_center);
 ArrayList<Staff> SelectStaffInfo(String ageS,String gender,String phoneS
     ,String staff_name,String numberS,String type,String supply_center);
```

What's more, we implement operations for the list based on multiple parameters.We will introduce in the next part in detail.

2.1.2 stockIn:

```
1 public void stockIn()
```

input: void
output: void
todo: increase the product inventory and merge inventory of the same product

2.1.3 placeOrder:

```
1 public void placeOrder()
```

input: void
output: void
todo: use the data in csv to place the orders

2.1.4 updateOrder:

```
1 public void updateOrder()
```

input: void
output: void
todo: use the data in csv to update order

2.1.5 deleteOrder:

```
1 public void deleteOrder()
```

input: void
output: void
todo: use the data in csv to delete order

2.1.6 getAllStaffCount:

```
1 ArrayList<String> findNumberGroupByType();
```

input: void
output: the type and the corresponding number
todo: count the corresponding number for each type

2.1.7 getContractCount:

```
1 int countContract();
```

input: void
output: the number of existing contracts
todo: count of the existing contracts

2.1.8 getOrderCount:

```
1 int countOrder();
```

input: void
output: the number of existing orders
todo: count of the existing orders

2.1.9 getNeverSoldProductCount:

```
public long getNeverSoldProductCount();
```

input: void
output: count of the never-sold products
todo: find the product model that no order needs

2.1.10 getFavoriteProductModel:

```
List<String> getFavoriteProductModel();
```

input: void
output: the favorite product and corresponding number
todo: Find the models with the highest sold quantity, and the number of sales

2.1.11 getAvgStockByCenter:

```
List<String> countAvgInventory();
```

input: void
output: the center and and corresponding average quantity of the remaining product models
todo: For each supply center, calculate the average quantity of the remaining product models. The results should be ordered by the name of the supply centers and rounded to one decimal place

2.1.12 getProductByNumber:

```
List<ModelMoreInfo> getProductByNumber(String productNumber);
```

input: the product code.
output: the information of all the model that belongs to the product.
todo: Find a product according to the product number and return some information of each product model in each supply center

2.1.13 getContractInfo:

```
Contract findByNum(String num);
```

input: num for the contract we want to query.
output: the corresponding entiey class object which contains all the infomation we want.
todo: Find a contract with a contract number and return the content of the contract.

2.2 Functional Requirements

Main code language: Java

Independent testing: Webpage-based testing interface to test all the APIs with given input data.

One-step input/export: Finished

3 Advanced Requirements

3.1 usability enhanced APIs

3.1.1 IDSU based on multiple parameters

We design the IDSU APIs that can using more than one parameters.The parameters are in an And relation to operate(delete/update/query) the list(take staff as example).

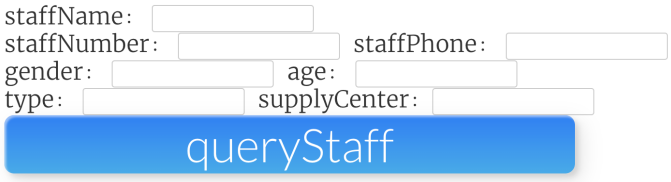

Figure 1: query staff based on multiple parameters

3.1.2 Some new APIs

Take the example UpdateOrderState API that is a mechanism to change order status according to time and date. We can according to compare the order's lodgement date with the present time to change the order state into 'Finished' or 'Not Finished' and then show the order whose state has been changed.

```
1 public List<ContractOrder> UpdateOrderState();
```

3.2 the implement of the real back-end server:springboot

We use the springboot to implement MVC model with the mapper/dao/service/model level and provide HTTP web services. The structure for our project is showed here.

bean → all the entity classes used in the project
controller→ use the method from the service and correspond to the front end
dao → manipulate the data in the database as the corresponding object in the bean package
redis → manipulate the data about the database redis
service → use the dao object to implement the business logic

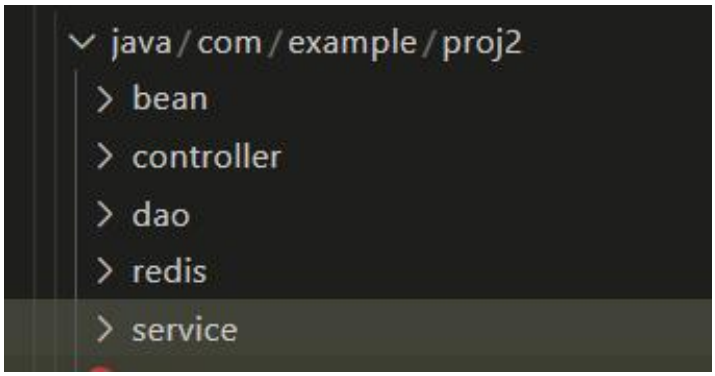

Figure 2: structure

3.3 database connection pools

We use the Hikari as the database connection pools. The database connection pools configs is listed here.

configs for Hikari

```
# Hikari will use the above plus the following to setup connection
    pooling
# type
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
# name
spring.datasource.hikari.pool-name=MdmApiHikariPool
# maximum-pool-size
spring.datasource.hikari.maximum-pool-size=12
#connection-timeout
spring.datasource.hikari.connection-timeout=60000
# minimum-idle
spring.datasource.hikari.minimum-idle=10
# idle-timeout
spring.datasource.hikari.idle-timeout=500000
# max-lifetime
spring.datasource.hikari.max-lifetime=540000
#test
spring.datasource.hikari.connection-test-query=SELECT 1
```

3.4 GUI

We use the html and css to make a beautiful and useful GUI for the web. The screenshot of the GUI is listed here:

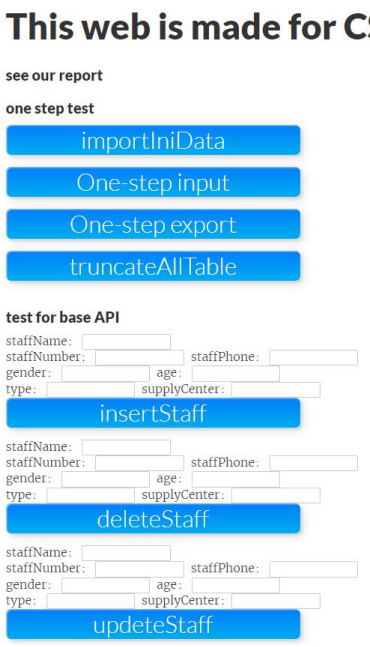

Figure 3: GUI

The web GUI of Show result:


Q1:

supply_center	average
America	313.8
Asia	201.7
Eastern China	145.9
Europe	310.9
Hong Kong Macao and Taiwan regions of China	305.4
Northern China	260.7
Southern China	221.1
Southwestern China	315.7

Q2:

supply_center	product_model	quantity

Q3(1):

num	CSE0000216
direct	Brian Brown
enterpriseName	Ping An Insurance (Group) Company Of China
supplyCenter	Southern China

model	salesman_name	quantity	unit_price	estimate_delivery_date	lodgement_date
Servers40	Sun Yankun	304	220	2022-01-12	2022-01-12
Casette(Library1)	Qi Yinxiang	390	480	2022-01-11	2022-01-11
BarcodeAndCardReadingEquipment34	Chu Ouqin	491	790	2022-01-10	2022-01-12

Q3(2):

num	CSE0000209
direct	Joseph Davies
enterpriseName	Sinopec
supplyCenter	Northern China

Figure 4: GUI-show-data

3.5 user privileges, procedures, indexing, and views

We make two roles for the database: root for the web and order-reader for other user.

For the root, it has all the privileges. And for the order-reader, it only has the privileges to query the infomation from the view all-order.

As for the all-order view, it contains all the information we want to show for the unroot user.

The sql for the all-order view is list here:

sql for the all-order view

```
create view all_order as
select c.num,s2.name as contracts_manager,s.name as salesman,m.model_name
    ,quantity,
ce.name as client_enterprise,sc.name as supply_center,
lodgement_date,estimated_delivery_date,state
from contract_order
join model m on m.id = contract_order.model_id
join staff s on s.id=contract_order.salesman_id
join contract c on contract_order.contract_id = c.id
join staff s2 on c.contracts_manager_id =s2.id
join client_enterprise ce on c.client_enterprise_id = ce.id
join supply_center sc on ce.supply_center_id = sc.id;
```

What's more, we have used the index for the unique column and auto increment primary key to increase the speed for query.

3.6 high-concurrent and distributed access

In this part, we use a new distributed database redis to support the high-concurrent.

We simulate the snap up scene which we suppose a lots of request come in at the same time to buy the same good. If we do nothing, the oversold situation will appear and the number of unsold goods will be incorrect.

In this situation, we use the database redis as the distributed data access which has wonderful performance in the high-concurrent scene. We first move our inventory infomation from prosgre to the redis and make each good as a independent key-value in the redis. When the request come in, we just need to update the key-value data stored in the redis and update the same data to the prosgre regular. By using this method, we pass the pressure tests without error which will have 5000 threads and each threads will make 10 requests one by one.

About the pressure tests, we use the tool apache-jmeter-5.4

The config for the redis is listed here:

```
spring.redis.host=106.55.104.82
spring.redis.port=6379
spring.redis.database=0
spring.redis.jedis.pool.max-active=50
spring.redis.jedis.pool.max-wait=3000
spring.redis.jedis.pool.max-idle=20
spring.redis.jedis.pool.min-idle=2
spring.redis.timeout=5000
```