

# CS307-spring22-project1-Group(307)

Class: Lab class 3

Name: 陈茜

SID: 12011136

Contribution ratio: 50%

Name: 加泽瑄

SID: 12011126

Contribution ratio: 50%

Task1,2: together

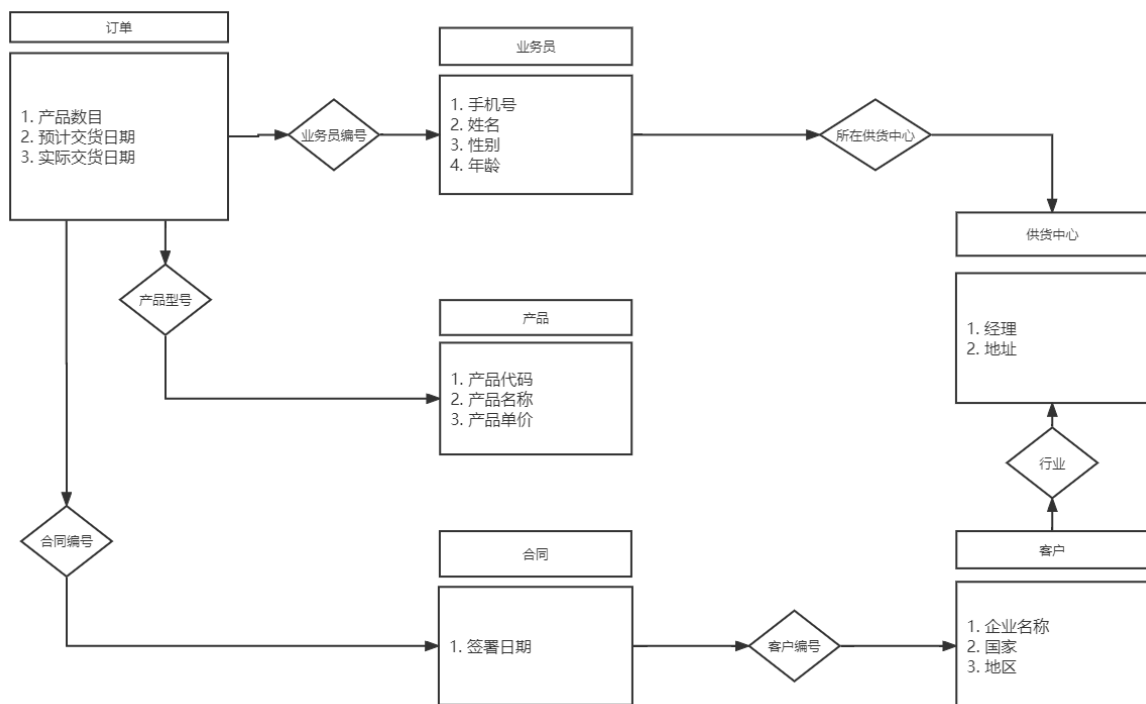
Task3: 陈茜

Task4: 加泽瑄

## Task1-E-R Diagram

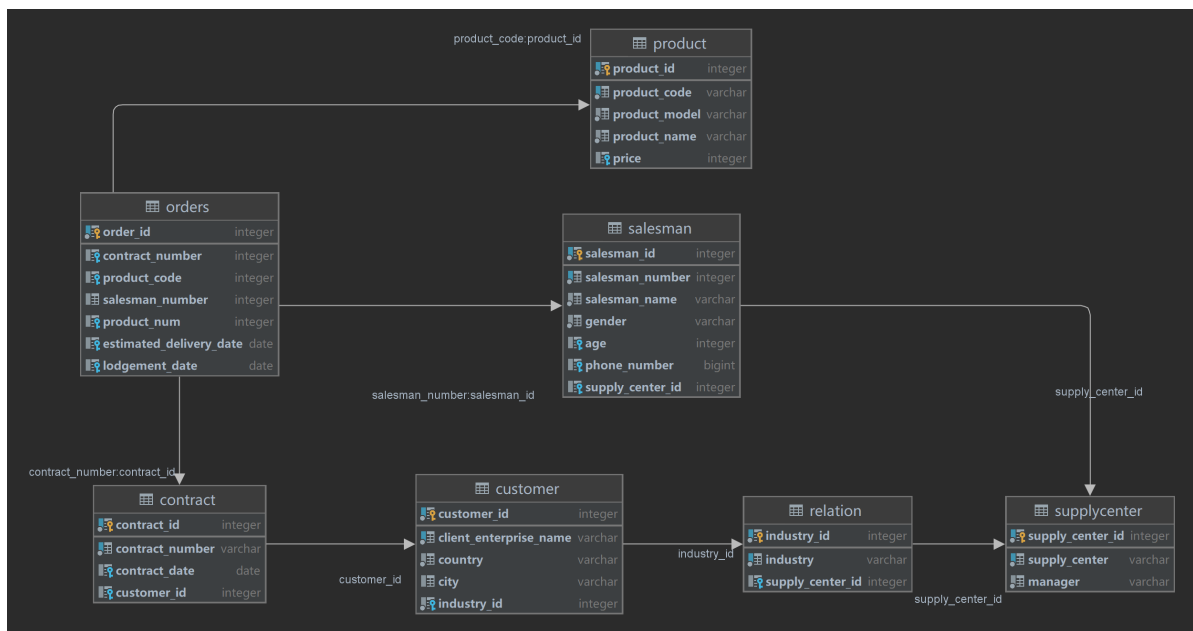
Snapshot:

The name of the software/online service you use for drawing the diagram :



## Task2——Database Design

## 1 The snapshot of the E-R diagram generated by DataGrip:



## 2 Briefly describe the table designs and the meanings of each table and column:

### Overview:

Using **orders** as an index, many-to-one connection of **product**, **contract** and **salesman** means that each **product**, **contract** and **salesman** corresponds to multiple **orders**;

Under a **contract**, many-to-one connection of **customers** (enterprise) means that each **customer** corresponds to multiple **contracts**;

The primary key of the **relation** table is **industry**, and the **customer** many-to-one connection **relation** table means that each **industry** corresponds to multiple **customer**;

Finally, the **supply center** connects **relation** and **salesman** one-to-many, which means that each **salesman** has a fixed **supply center**, and each **customer** also has a fixed **supply center**, and they are all in a many-to-one relationship.

### Form analysis:

#### orders

Primary key: order\_id, an integer of serial auto-increment type, representing the serial number of the order;

Foreign key: The next three integers: contract\_number, product\_code, salesman\_number are the foreign keys of the three tables contract, product, and salesman, respectively, representing the respective serial numbers of the three tables;

product\_num, type integer: the number of products in the order;

The last two dates: estimated\_delivery\_date and lodgement\_date, represent the estimated delivery time and actual delivery time of the order (null if not).

#### contract:

Primary key: contract\_id, an integer of serial auto-increment type, representing the serial number of the contract;

Foreign key: type integer, variable name customer\_id, is the foreign key of the table customer, representing the serial number of the customer;

contract\_number, type varchar: unique identifier for each contract;  
contract\_date, type date: contract creation date.

#### **product:**

Primary key: product\_id, which is a serial self-incremented integer, representing the serial number of the product;

product\_code, type varchar: unique identifier for each product;  
product\_model, type varchar: the specific model of the product;  
product\_name, type varchar: product name;  
price, type integer: product unit price.

#### **salesman:**

Primary key: salesman\_id, which is a serial auto-increment type integer, representing the serial number of salesman;

Foreign key: type integer, variable name supply\_center\_id, is the foreign key of the table supplycenter, representing the serial number of supplycenter;

contract\_number, type varchar: unique identifier for each contract;  
contract\_date, type date: contract creation date.

salesman\_number, type integer: salesman number.

salesman\_name, type varchar: Salesman name.

gender, type varchar: Salesperson gender.

age, type integer: salesperson age.

phone\_number, type bigint: salesperson phone number.

#### **customer:**

Primary key: customer\_id, which is an integer of serial auto-increment type, representing the serial number of the customer;

Foreign key: industry\_id, type integer: the index of the customer's enterprise industry, which is the foreign key of the table relation.

client\_enterprise\_name, type varchar: the name of the client enterprise;

country, type varchar: the country of the client enterprise;

city, type varchar: the city of the client enterprise;

#### **relation:**

Primary key: industry\_id, which is an integer of serial auto-increment type, representing the serial number of the industry;

Foreign key: supply\_center\_id, type integer, is the foreign key of the table supplyCenter, representing the serial number of supplyCenter;

industry, type varchar: industry name;

#### **supplyCenter:**

Primary key: supply\_center\_id, which is a serial auto-incremented integer, representing the serial number of supplyCenter;

supply\_center, type varchar: supply center name;

manager, type varchar: The name of the fulfillment center manager.

## **Task3——Data Import**

---

## 1 script code

Please see attachment for details.(Load.java, Load2.java)

## 2 Script introduction and analysis

**Method1** :Continuously import data through JDBC insert statements.(Load.java)

1 Download the JDBC driver for PostgreSQL, and import the jar package into the project through project structure-dependencies; prepare the pre-import file.

2 Prepare the host, database, user, and password needed to connect to the database, and pass in the openDB method.

3 openDB method: The purpose is to connect to the database and determine whether it can be successfully connected.

First, try to load the Driver class to check whether it can be successfully registered in the DriverManager class;

If successful, continue to create the url according to the jdbc protocol ("jdbc:postgresql://" + host + "/" + dbname);

If successful, continue to execute Connection con = DriverManager.getConnection(url, props), this step is trying to connect to the database;

If all the executions are successful, the database connection is successful. Subsequent interaction with the database can be performed using the Connection object.

4 Test the openDB method once and execute some simple static Statement objects, that is, execute some SQL statements that need to be executed.

5 Execute openDB again, and use PreparedStatement to preload the statements that need to be called multiple times to facilitate subsequent processing.

6 Read the csv data line by line, and perform logical sorting in jdbc, execute the statement through the PreparedStatement object prepared in advance, and import different tables; at the same time, count the time.

7 Close the database.

**Method2** :Run the copy command through JDBC's CopyManager.(Load2.java)

1 Same as method1, download the JDBC driver for PostgreSQL, and then import; prepare the host, database, user, and password, create the connection: Connection con = DriverManager.getConnection(url, props).

2 Get the object of copyManager through the obtained connection(CopyManager copyManager = new CopyManager((BaseConnection) connection);)

3 Split the files you need to import and match them to the prepared table format.

4 Execute copyIn method to import data.

## 3 Script optimization

### Optimization point1:

Execute the delete foreign key statement first, import data, and create a foreign key after the import is completed.

#### Reason:

In the process of importing data, jdbc will check whether the foreign key of each inserted data has a corresponding primary key; thus increasing consumption.

#### Code:

After the first openDB, insert the statement:

```
Statement statement =connection.createStatement();
statement.execute("alter table relation drop constraint
relation_supply_center_id_fkey cascade ;\n" +
                "alter table salesman drop constraint
salesman_supply_center_id_fkey cascade ;\n" +
                "alter table customer drop constraint customer_industry_id_fkey
cascade ;\n" +
                "alter table contract drop constraint contract_customer_id_fkey
cascade ;\n" +
                "alter table orders drop constraint orders_contract_number_fkey
cascade ;\n" +
                "alter table orders drop constraint orders_product_code_fkey
cascade ;\n" +
                "alter table orders drop constraint orders_salesman_number_fkey
cascade ;\n");
```

Thus deleting the foreign key; after the last openDB, insert the statement:

```
Statement statement2 =connection.createStatement();
statement2.execute("alter table relation add foreign key
(supply_center_id) references supplyCenter(supply_center_id);\n" +
                "alter table salesman add foreign key (supply_center_id)
references supplyCenter(supply_center_id);\n" +
                "alter table customer add foreign key (industry_id) references
relation(industry_id);\n" +
                "alter table contract add foreign key (customer_id) references
customer(customer_id);\n" +
                "alter table orders add foreign key (salesman_number) references
salesman(salesman_id);\n" +
                "alter table orders add foreign key (contract_number) references
contract(contract_id);\n" +
                "alter table orders add foreign key (product_code) references
product(product_id);");
```

Reinsert the foreign key.

### Optimization point2:

Run the drop index statement, import data, and then create indexe.

#### Reason:

Index helps greatly increase query speed, but affects insert speed. Because in the process of adding data, the corresponding index entries need to be updated in real time. Therefore, deleting indexes before inserting data can improve the speed.

#### Code:

After the first openDB, insert the statement:

```
Statement stmt0 = con.createStatement();
stmt0.execute("drop index product_index;\n" +
              "drop index supplyCenter_index;\n" +
              "drop index relation_index;\n" +
              "drop index customer_index;\n" +
              "drop index salesman_index;\n" +
              "drop index contract_index;\n" +
              "drop index orders_index;");
con.commit();
```

Thus dropping the index; after the last openDB, insert the statement:

```
stmt0.execute("create index product_index on
product(product_id);\n" +
              "create index supplyCenter_index on
supplyCenter(supply_center_id);\n" +
              "create index relation_index on
relation(industry_id);\n" +
              "create index customer_index on
customer(customer_id);\n" +
              "create index salesman_index on
salesman(salesman_id);\n" +
              "create index contract_index on
contract(contract_id);\n" +
              "create index orders_index on orders(order_id);");
con.commit();
```

Reinsert the index.

## 4 Test

Test with computer with:

Processor: AMD Ryzen 7 4800H with Radeon Graphics, 2.90 GHz

Memory:16.0 GB(16384MB) RAM

For **method 1**, Without using index, using foreign key -> delete foreign key before inserting and add foreign key after inserting. Speed improvement: 20202 pieces/second -> 43327 pieces/second, more than double the speed.

```
50000 records successfully loaded
Loading speed : 20202 records/s
```

```
50000 records successfully loaded
Loading speed : 43327 records/s
```

With deleting foreign key, use index when inserting -> dropping index before inserting and add index after inserting. Speed improvement: 32341 pieces/second -> 42808 pieces/second

```
50000 records successfully loaded
Loading speed : 32341 records/s
```

```
50000 records successfully loaded
Loading speed : 42808 records/s
```

For **method 2**, Without using index, using foreign key -> delete foreign key before inserting and add foreign key after inserting. Speed improvement: 41220 pieces/second -> 137741 pieces/second, more than three times faster.

```
50000 records successfully loaded
Loading speed : 41220 records/s
```

```
50000 records successfully loaded
Loading speed : 137741 records/s
```

With deleting foreign key, use index when inserting -> dropping index before inserting and add index after inserting. Speed improvement: 90909 pieces/second -> 135869 pieces/second.

```
50000 records successfully loaded
Loading speed : 90909 records/s
```

```
50000 records successfully loaded
Loading speed : 135869 records/s
```

We can see that the first method is two to three times faster than the second method;

After the optimization of foreign keys, the speed of the two methods is increased by 2-3 times.

After the optimization of index, the speed of the two methods is increased by about 1.5 times.

## Task4-Compare DBMS with File I/O

### 1. Speed comparison:

#### 1. Search

##### BDMS

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
1
From 20000
to 30000
```

```
29998 3015 141 240 700 2015-11-12 null
29999 3015 175 144 500 2015-12-23 null
30000 3015 300 22 310 2015-11-03 null
程序运行时间: 24ms
共执行 10000 条SQL查询语句, 平均每秒执行 416666 条语句
```

##### Java program

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
1
From 20000
to 30000|
```

```

29999 3015 175 144 500 2015/12/23
30000 3015 300 22 310 2015/11/3
Time: 134ms
Totally 10000 Java deleting statement.
Loading speed is 74626 records/s

```

## 2. Delete

### DBMS delete continuously

```

Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
2
From 12345
to 43210
程序运行时间: 286ms
共执行 30865 条SQL删除语句, 平均每秒执行 107919 条语句
进程已结束,退出代码0

```

### Java program delete continuously

```

Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
2
From 12345
to 43210
Time: 76ms
Totally 30865 Java deleting statement.
Loading speed is 406118 records/s
进程已结束,退出代码0

```

### DBMS non-consecutive deletion

```

Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
2
Input 1 to batch delete ,2 to select delete
2
Input deleting lines:
10000
36921 48353 18446 7289 19492 16361 24474 2552 32153 36463 26976 48881 1530 12650 39
程序运行时间: 120ms
共执行 10000 条SQL删除语句, 平均每秒执行 83333 条语句

```

### Java program delete continuously

```

Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
2
Input 1 to batch delete ,2 to select delete
2
Input deleting lines:
10000
36921 48353 18446 7289 19492 16361 24474 2552 32153 36463 26976 48881 1530
Time: 341ms
Totally 10000 Java deleting statement.
Loading speed is 29325 records/s

```

**Conclusion:** database operations have a huge advantage over java when dealing with discontinuous data, and the speed is relatively stable with little fluctuation



### 3. Update

#### DBMS Wide range of test sample

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.  
3  
From 23456  
to 34567  
程序运行时间: 509ms  
共执行 11111 条SQL更新语句, 平均每秒执行 21829 条语句  
进程已结束, 退出代码0
```

#### Java program Wide range of test sample

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.  
3  
From 23456  
to 34567  
Time: 132ms  
Totally 11111 Java deleting statement.  
Loading speed is 84174 records/s  
进程已结束, 退出代码0
```

#### DBMS Small scale test example

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.  
3  
From 123  
to 1234  
程序运行时间: 104ms  
共执行 1111 条SQL更新语句, 平均每秒执行 10682 条语句  
进程已结束, 退出代码0
```

#### Java program Small scale test example

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.  
3  
From 123  
to 1234  
Time: 113ms  
Totally 1111 Java deleting statement.  
Loading speed is 9831 records/s  
进程已结束, 退出代码0
```

It can be found that no matter the size of the sample range, the java program operation time basically does not change, because the java program operation is a global operation on the file, which is not greatly affected by the sample range.

Expand the total number of samples to 10 million pieces of data, and then conduct a large-scale test sample.

#### DBMS with 10 million pieces of data

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
3
From 1919810
to 1929810
程序运行时间: 584ms
共执行 10000 条SQL更新语句, 平均每秒执行 17123 条语句
进程已结束,退出代码0
```

#### Java program with 10 million pieces of data

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
3
From 1919810
to 1929810
Time: 4672ms
Totally 10000 Java deleting statement.
Loading speed is 2140 records/s
进程已结束,退出代码0
```

**Conclusion:** In the face of massive data, database operations have a huge advantage over java, and the speed is relatively stable and the fluctuation is not large

#### 4. Insert

**remake:** Here, only randomly generated data is added for comparison. When inserting external file data, the read operation is the same, and the impact on time is the same. Modify 10,000 pieces of data each time.

##### DBMS first time

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
4
Count 10000
程序运行时间: 519ms
共执行 10000 条SQL增添语句, 平均每秒执行 19267 条语句
进程已结束,退出代码0
```

##### DBMS tenth time

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
4
Count 10000
程序运行时间: 513ms
共执行 10000 条SQL增添语句, 平均每秒执行 19493 条语句
进程已结束,退出代码0
```

##### DBMS After the data volume reaches 1,000,000

```
Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
4
Count 10000
程序运行时间: 521ms
共执行 10000 条SQL增添语句, 平均每秒执行 19193 条语句
进程已结束,退出代码0
```

##### Java program first time

```

Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
4
Count 10000
Time: 127ms
Totally 10000 Java deleting statement.
Loading speed is 78740 records/s
进程已结束,退出代码0

```

### Java program tenth time

```

Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
4
Count 10000
Time: 281ms
Totally 10000 Java deleting statement.
Loading speed is 35587 records/s
进程已结束,退出代码0

```

### Java program After the data volume reaches 1,000,000

```

Input 1 to select, 2 to delete, 3 to update, 4 to insert, -1 to quit.
4
Count 10000
Time: 742ms
Totally 10000 Java deleting statement.
Loading speed is 13477 records/s
进程已结束,退出代码0

```

The java program operation is for the whole world. With the increase of the number of operations, the java operation speed will seriously decrease, and the gap between the operation of the database and the database operation is obvious in the face of massive data.

**Conclusion:** When the amount of data is large, DBMS has obvious advantages over java in all operations

## 2. Statement complexity comparison

### 1. Search

```

//DBMS
select * from orders where order_id > ? and order_id <= ?;

```

**Implementation principle:** operate directly on the disk, record a hash value in the disk for each piece of data, and use the hash value to match.

```

//JAVA PROGRAM
while ((line = infile.readLine()) != null) {
    parts = line.split(",");
    if (parts.length > 1) {
        if (Integer.parseInt(parts[0]) > over) break;
        if (Integer.parseInt(parts[0]) > begin) {
            //Get the required data here
        }
    }
}
}

```

**Implementation principle:** query one by one and operate on the disk. You can also use hash value matching, but the object directly manipulated by this method is memory, which is inconvenient for comparison

## 2. Delete

```
//DBMS
delete from orders where order_id = ?;
```

**Implementation principle:** search first, and then delete the data directly from the disk.

```
//JAVA PROGRAM
//Generate temporary files, omitted here
//line is an array of ids that store deleted data
while ((line = infile.readLine()) != null) {
    parts = line.split(",");
    if (parts.length > 1) {
        if (!lines.contains(Integer.parseInt(parts[0]))) {
            tempwriter.write(line);
            tempwriter.newLine();
            cnt++;
            //The reverse selection is here, and the data that is not in the
            array is written to the temporary file
        }
    }
}

//Write the temporary file to the original file, omitted here
```

**Implementation principle:** more information in reamke.

## 3. Update

```
//DBMS
update orders set contract_number = ?,product_code = ?,salesman_number =
?,product_num = ?,estimated_delivery_date=? where order_id = ?;

for (int i = begin; i < over; i++) {
    //Using the Random tool to generate modified data
    con_id = random.nextInt((contract_id - 1) + 1) + 1;
    pro_id = random.nextInt((product_id - 1) + 1) + 1;
    sal_id = random.nextInt((salesman_id - 1) + 1) + 1;
    product_num = random.nextInt((10000 - 1) + 1) + 1;
    date = String.valueOf(random.nextInt((2021 - 2000) + 1) + 2000) + "-" +
    (random.nextInt((12 - 1) + 1) + 1) + "-" + (random.nextInt((28 - 1) + 1) + 1);

    stm.setInt(1, con_id);
    stm.setInt(2, pro_id);
    stm.setInt(3, sal_id);
    stm.setInt(4, product_num);
    stm.setDate(5, Date.valueOf(date));
    stm.setInt(6, i);

    stm.addBatch();
}
```

```

        if (cnt++ % SIZE == 0) {
            stm.executeBatch();
            stm.clearBatch();
            //batch processing
        }
    }
}

```

**Implementation principle:** search first, and then modify the data directly on the disk

```

//JAVA PROGRAM
//Generate temporary files, omitted here

//line is an array of ids that store deleted data

while ((line = infile.readLine()) != null) {
    parts = line.split(",");
    if (parts.length > 1) {
        if (!lines.contains(Integer.parseInt(parts[0]))) {
            tempwriter.write(line);
            tempwriter.newLine();
            //No need to modify the data, write directly to the temporary file
        } else {
            con_id = random.nextInt((contract_id - 1) + 1) + 1;
            pro_id = random.nextInt((product_id - 1) + 1) + 1;
            sal_id = random.nextInt((salesman_id - 1) + 1) + 1;
            product_num = random.nextInt((10000 - 1) + 1) + 1;
            date = String.valueOf(random.nextInt((2021 - 2000) + 1) + 2000) + "-"
            + (random.nextInt((12 - 1) + 1) + 1) + "-" + (random.nextInt((28 - 1) + 1) +
            1) + "c";

            tempwriter.write(parts[0] + "," + con_id + "," + pro_id + "," +
            sal_id + "," + pro_id + "," + product_num + "," + date);
            tempwriter.newLine();
            //write the modified data to a temporary file
        }
        cnt++;
    }
}

//write the temporary file to the original file, omitted here

```

**Implementation principle:** more information in reamke.

#### 4. Insert

```

//DBMS
for (int i = 0; i < count; i++) {
    con_id = random.nextInt((contract_id - 1) + 1) + 1;
    pro_id = random.nextInt((product_id - 1) + 1) + 1;
    sal_id = random.nextInt((salesman_id - 1) + 1) + 1;
    product_num = random.nextInt((10000 - 1) + 1) + 1;
    date = String.valueOf(random.nextInt((2021 - 2000) + 1) + 2000) + "-" +
    (random.nextInt((12 - 1) + 1) + 1) + "-" + (random.nextInt((28 - 1) + 1) + 1);

    stm.setInt(1, con_id);
}

```

```

        stm.setInt(2, pro_id);
        stm.setInt(3, sal_id);
        stm.setInt(4, product_num);
        stm.setDate(5, Date.valueOf(date));
        stm.setDate(6, Date.valueOf(date));

        stm.addBatch();
        if (cnt++ % SIZE == 0) {
            stm.executeBatch();
            stm.clearBatch();
        }
    }
}

```

**Implementation principle:** operate at the end of the data, and use the linkedlist structure to add data.

```

//JAVA PROGRAM
//Generate temporary files, omitted here
while ((line = infile.readLine()) != null) {
    parts = line.split(",");
    if (parts.length > 1) {
        tempwriter.write(line);
        tempwriter.newLine();
        cnt++;
        //Write all original file data to temporary file
    }
}

tempwriter.flush();
infile.close();

for (int i = 0; i <= count; i++) {
    con_id = random.nextInt((contract_id - 1) + 1) + 1;
    pro_id = random.nextInt((product_id - 1) + 1) + 1;
    sal_id = random.nextInt((salesman_id - 1) + 1) + 1;
    product_num = random.nextInt((10000 - 1) + 1) + 1;
    date = String.valueOf(random.nextInt((2021 - 2000) + 1) + 2000) + "-" +
    (random.nextInt((12 - 1) + 1) + 1) + "-" + (random.nextInt((28 - 1) + 1) + 1);
    tempwriter.write(cnt++ + "," + con_id + "," + pro_id + "," + sal_id + "," +
    product_num + "," + date);
    tempwriter.newLine();
    //Write the newly added data to a temporary file
}
tempwriter.flush();
tempwriter.close();

//Write the temporary file to the original file, omitted here

```

**Implementation principle:** more information in reamke.

### Remark:

**Note on addition, deletion and modification operations:** Since there is no non-overwriting write function in the standard io stream library provided in java, each addition, deletion and modification operation can only be performed on the entire data. The specific implementation method is to create a new temporary file, write the processed data to the temporary file, and then write the temporary file to the original file.

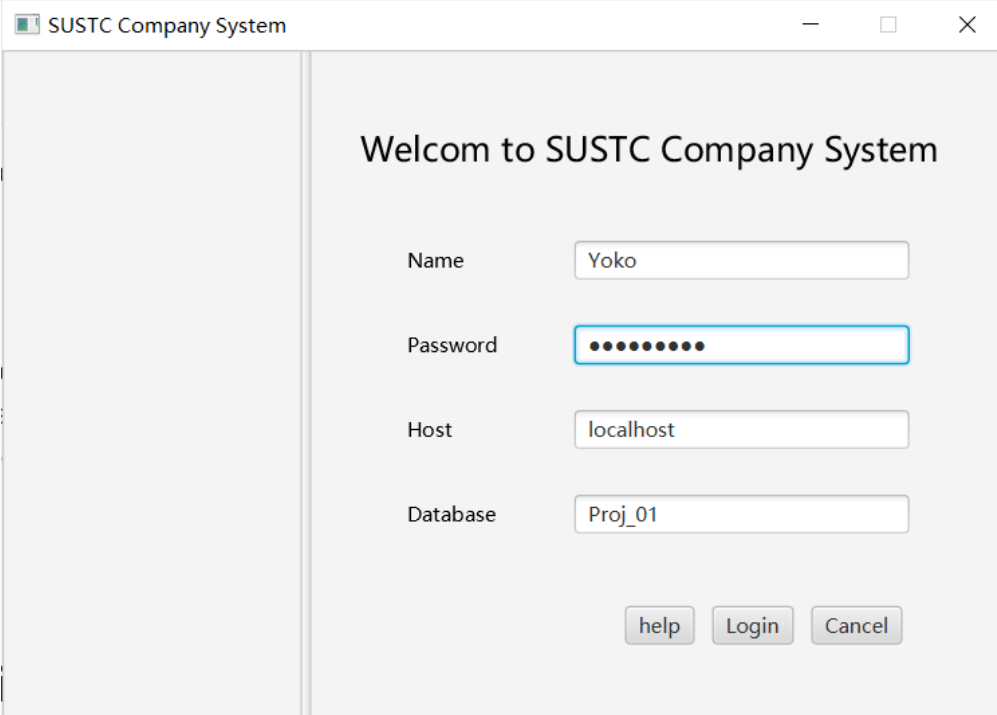
## Bonus

---

Two parts.

### 1 Visual operation management system

#### 1. Login frame



The screenshot shows a Java Swing window titled "SUSTC Company System". The window has a light gray background and a vertical separator line on the left. The main content area is titled "Welcom to SUSTC Company System" (note the typo). Below the title, there are four labeled text input fields: "Name" with the value "Yoko", "Password" with masked characters "••••••••", "Host" with the value "localhost", and "Database" with the value "Proj\_01". At the bottom right, there are three buttons: "help", "Login", and "Cancel".

**Explanation:** The user level mechanism is implemented. After successful login, different rights management will be carried out according to different user levels.

In order to prevent malicious users from bypassing the system and modifying the database files by other means, the system adopts two sets of account systems. The account of this system cannot log in and modify the database in any other way.

#### 2. Main frame

SUSTC Company System

Normal User: Yoko

Searching Information

Select table: product

Select row: product\_id

Searching way: Scope

Keywords: 1-500

<

>

GO!

Operation

GO!

Delete Update Insert

product_id	product_code	product_model	product_name	price
1	T67P542	TvBaseR1	Tv Base	680
2	E6N2308	ElectronicDictionary67	Electronic Dictionary	970
3	E9G8372	ExhaustFanD8	Exhaust Fan	320
4	M421X86	MultifunctionalT4	Multifunctional	90
5	SE17564	ServerBarebonesH4	Server Barebones	70
6	L17546O	LaptopDockingStation94	Laptop Docking Station	620
7	D8150P6	DotMatrixPrinterK1	Dot Matrix Printer	690
8	HD86793	HumidifierY5	Humidifier	470
9	P41857Y	PhysicalSecurityIsolation35	Physical Security Isolation	620
10	SY73215	SteeringWheel71	Steering Wheel	830
11	H7143H9	HotPotJ2	Hot Pot	230
12	B239N58	BuildingIntercom75	Building Intercom	80
13	S2180L4	SerialCommunicationServerE9	Serial Communication Server	170
14	C95F078	CasetteLibrary26	Casette Library	60
15	MR92570	MicrophoneDigitalCompanionX4	Microphone Digital Companion	730
16	M9J6453	Mp437	Mp4	600
17	S5291W4	ServerPowerO3	Server Power	860
18	T35X601	ToiletSeatO7	Toilet Seat	810
19	D7903R1	DiskArray57	Disk Array	530
20	S86042Z	SecurityCameraY2	Security Camera	230
21	C3752Y9	CameraBattery26	Camera Battery	340
22	H8713D4	HardDiskRecording82	Hard Disk Recording	450

**Explanation:** The upper left corner of the interface will display the current user level. Ordinary users can only perform query operations, and administrator users can perform operations such as inquiries, additions, deletions, and changes. Add, modify, and delete operations are based on query operations, and the object of the operation is the result set of the previous query. When the user is a normal user, the operation area below is unavailable.

## 1.) Select

SUSTC Company System

Normal User: Yoko

Searching Information

Select table: product

Select row: product\_id

Searching way: Scope

Keywords:

<

>

GO!

Operation

GO!

Delete Update Insert

product_id	product_code	product_model	product_name	price
1	T67P542	TvBaseR1	Tv Base	680
2	E6N2308	ElectronicDictionary67	Electronic Dictionary	970
3	E9G8372	ExhaustFanD8	Exhaust Fan	320
4	M421X86	MultifunctionalT4	Multifunctional	90
5	SE17564	ServerBarebonesH4	Server Barebones	70
6	L17546O	LaptopDockingStation94	Laptop Docking Station	620
7	D8150P6	DotMatrixPrinterK1	Dot Matrix Printer	690
8	HD86793	HumidifierY5	Humidifier	470
9	P41857Y	PhysicalSecurityIsolation35	Physical Security Isolation	620
10	SY73215	SteeringWheel71	Steering Wheel	830
11	H7143H9	HotPotJ2	Hot Pot	230
12	B239N58	BuildingIntercom75	Building Intercom	80
13	S2180L4	SerialCommunicationServerE9	Serial Communication Server	170
14	C95F078	CasetteLibrary26	Casette Library	60
15	MR92570	MicrophoneDigitalCompanionX4	Microphone Digital Companion	730
16	M9J6453	Mp437	Mp4	600
17	S5291W4	ServerPowerO3	Server Power	860
18	T35X601	ToiletSeatO7	Toilet Seat	810
19	D7903R1	DiskArray57	Disk Array	530
20	S86042Z	SecurityCameraY2	Security Camera	230
21	C3752Y9	CameraBattery26	Camera Battery	340
22	H8713D4	HardDiskRecording82	Hard Disk Recording	450

**Explanation:** The system supports three query search - accurate search, blurry search and range search.



SUSTC Company System

Normal User: Yoko

Searching Information

Select table: customer

Select row: client\_enterprise

Searching way: Blurry

Keywords: %e%

<

>

GO!

Operation

GO!

Delete Update Insert

customer_id	client_enterprise_name	country	city
2	Unilever	Netherlands/United Kingdom	NULL
3	Deutsche Bank	Germany	NULL
4	Albertson's	United States	NULL
5	Santander Central Hispano Group	Spain	NULL
6	Koninklijke Ahold	Netherlands	NULL
7	Kroger	United States	NULL
8	Tencent	China	Shenzhen
12	Delphi Automotive Systems	United States	NULL
14	Munich Re Group	Germany	NULL
15	Chengdu AIG	China	Chengdu
16	Home Depot	United States	NULL
17	Robert Bosch	Germany	NULL
18	Cardinal Health	United States	NULL
20	Meituan	China	Beijing
21	Target	United States	NULL
25	Tesco	United Kingdom	NULL
26	Industrial and Commercial Bank of China	China	Beijing
27	Peugeot	France	NULL
28	Alcatel	France	NULL
29	Dell Computer	United States	NULL
30	Safeway	United States	NULL
32	Royal Philips Electronics	Netherlands	NULL

## 2.) Delete

SUSTC Company System

Super User: Ghosh

Searching Information

Select table: orders

Select row: product\_num

Searching way: Accurate

Keywords: 750

<

>

GO!

Operation

sure

GO!

Delete Update Insert

order_id	contract_number	product_code	salesman_number	product_num	estimated_delive...
2	1	2	2	750	2018-03-12
3	1	3	3	750	2018-02-05
26	4	23	24	750	2018-05-31
67	10	60	60	750	2003-05-03
122	17	72	110	750	2013-10-27
191	20	114	156	750	2004-04-01
192	20	143	157	750	2004-03-22
349	34	203	76	750	2018-03-31
350	34	204	49	750	2018-02-23
525	56	164	49	750	2020-12-31
649	67	1	263	750	2013-12-15
757	76	54	385	750	2010-10-15
791	83	280	390	750	2003-07-22
822	85	132	405	750	2011-09-08
985	107	217	17	750	2019-06-12
1037	112	252	281	750	2009-03-06
1043	112	14	422	750	2009-03-02
1097	115	173	359	750	2000-10-13
1197	120	205	477	750	2016-07-25
1268	130	259	82	750	2002-04-28
1273	130	77	313	750	2002-05-06
1375	136	185	446	750	2020-09-13

**Explanation:** To prevent misoperation, after selecting the delete function, you must enter "sure" to confirm the operation.

## 3.) Update

SUSTC Company System

Super User: Ghosn

Searching Information

Select table: 

orders

Select row: 

order\_id

Searching way: 

Scope

Keywords: 

1-100

<

>

GO!

Operation

;514,1919-8-11,1919-8-19

GO!

Delete

Update

Insert

order_id	contract_number	product_code	salesman_number	product_num	estimated_delive...
1	1	1	1	480	2018-04-03
4	1	4	4	740	2018-02-23
5	1	5	1	500	2018-02-19
6	2	6	5	310	2017-07-25
7	2	7	6	610	2017-07-22
8	2	8	5	160	2017-08-10
9	2	9	7	770	2017-08-07
10	2	8	8	430	2017-07-13
11	2	10	9	1000	2017-07-23
12	2	11	10	430	2017-07-18
13	2	12	11	130	2017-08-19
14	2	13	12	1000	2017-09-01
15	2	14	13	170	2017-08-25
16	2	15	14	600	2017-07-13
17	2	16	15	300	2017-08-13
18	2	17	16	770	2017-08-17
19	2	18	17	690	2017-08-02
20	2	19	18	780	2017-08-12
21	2	20	19	820	2017-08-24
22	2	20	20	180	2017-09-03
23	3	21	21	150	2004-01-20
24	3	8	22	780	2003-12-13

**Explanation:** The delete operation is completed and the selected data is successfully deleted.

SUSTC Company System

Super User: Ghosn

Searching Information

Select table: 

orders

Select row: 

order\_id

Searching way: 

Scope

Keywords: 

1-100

<

>

GO!

Operation

1,1,4,514,1919-8-11,1919-8

GO!

Delete

Update

Insert

order_id	con...A	product_code	salesman_number	product_num	estimated_delive...	lodgement_date
1	1	1	4	514	1919-08-11	1919-08-19
4	1	1	4	514	1919-08-11	1919-08-19
5	1	1	4	514	1919-08-11	1919-08-19
6	1	1	4	514	1919-08-11	1919-08-19
7	1	1	4	514	1919-08-11	1919-08-19
8	1	1	4	514	1919-08-11	1919-08-19
9	1	1	4	514	1919-08-11	1919-08-19
10	1	1	4	514	1919-08-11	1919-08-19
11	1	1	4	514	1919-08-11	1919-08-19
12	1	1	4	514	1919-08-11	1919-08-19
13	1	1	4	514	1919-08-11	1919-08-19
14	1	1	4	514	1919-08-11	1919-08-19
15	1	1	4	514	1919-08-11	1919-08-19
16	1	1	4	514	1919-08-11	1919-08-19
17	1	1	4	514	1919-08-11	1919-08-19
18	1	1	4	514	1919-08-11	1919-08-19
19	1	1	4	514	1919-08-11	1919-08-19
20	1	1	4	514	1919-08-11	1919-08-19
21	1	1	4	514	1919-08-11	1919-08-19
22	1	1	4	514	1919-08-11	1919-08-19
23	1	1	4	514	1919-08-11	1919-08-19
24	1	1	4	514	1919-08-11	1919-08-19

**Explanation:** To modify the data, you need to enter the modification result in the operation bar below, and the system will judge whether it is legal and modify it.

#### 4.) Insert

SUSTC Company System

Super User: Ghosn

Searching Information

Select table: orders

Select row: order\_id

Searching way: Scope

Keywords: 1-100

<

>

GO!

Operation

reset

GO!

Delete

Update

Insert

order_id	con...A	product_code	salesman_number	product_num	estimated_delive...	lodgement_date
1	1	1	1	480	2018-04-03	
2	1	2	2	750	2018-03-12	
3	1	3	3	750	2018-02-05	
4	1	4	4	740	2018-02-23	
5	1	5	1	500	2018-02-19	
6	2	6	5	310	2017-07-25	
7	2	7	6	610	2017-07-22	
8	2	8	5	160	2017-08-10	
9	2	9	7	770	2017-08-07	
10	2	8	8	430	2017-07-13	
11	2	10	9	1000	2017-07-23	
12	2	11	10	430	2017-07-18	
13	2	12	11	130	2017-08-19	
14	2	13	12	1000	2017-09-01	
15	2	14	13	170	2017-08-25	
16	2	15	14	600	2017-07-13	
17	2	16	15	300	2017-08-13	
18	2	17	16	770	2017-08-17	
19	2	18	17	690	2017-08-02	
20	2	19	18	780	2017-08-12	
21	2	20	19	820	2017-08-24	
22	2	20	20	180	2017-09-03	

**Explanation:** Enter "reset" to restore the data to the backup, enter "sure" to load the data of a local path (in order to avoid introducing other malicious files, the path has been set inside the program).

## 2 Database index

An index is a pointer to data in a table, it can help speed up SELECT queries and WHERE clauses. In our table design, every table has an index, which can greatly increase query speed.

```
29998 3015 141 240 700 2015-11-12 2015-11-21
29999 3015 175 144 500 2015-12-23 2015-12-03
30000 3015 300 22 310 2015-11-03 2015-11-07
程序运行时间: 17ms
共执行 10000 条SQL查询语句, 平均每秒执行 588235 条语句|
```

## 3 File IO optimization

java IO stream:  
"Stream" is a data source object which is capable of producing or receiving data.  
It can be divided into: character stream and byte stream.  
On the one hand, byte streams do not use buffers during operation, so byte streams read and write faster.  
But when reading a plain text file, the character stream processes 2 bytes of Unicode characters at a time, whereas the byte stream processes only one at a time, so the character stream reads and writes faster for this project.  
So what is faster? I designed a small test, simultaneously using byte stream and character stream to achieve the same operation (read and write the CSV provided by this project ten times to get the running time), and compared which is faster.

```

import java.io.*;

public class test {
    public static void main(String[] args) throws IOException {
        //character stream
        long start = System.currentTimeMillis();
        for (int i = 0; i < 10; i++) {
            BufferedReader infile = new BufferedReader(new
FileReader("javaPart/src/contract_info.csv"));
            String writeName = "javaPart/src/test.csv";
            FileWriter writer = new FileWriter(writeName);
            BufferedWriter out = new BufferedWriter(writer);
            String line;
            while ((line = infile.readLine()) != null) {
                out.write(line + "\n");
            }
            out.close();
        }
        long end = System.currentTimeMillis();
        System.out.println("character stream time: " + (end - start));

        //byte stream
        long start1 = System.currentTimeMillis();
        for (int i = 0; i < 10; i++) {
            FileInputStream fis = new
FileInputStream("javaPart/src/contract_info.csv");
            FileOutputStream fos = new
FileOutputStream("javaPart/src/test.csv");
            byte[] buffer = new byte[1024];
            int len = 0;
            while ((len = fis.read(buffer)) != -1) {
                fos.write(buffer, 0, len);
            }
            fis.close();
            fos.close();
        }
        long end1 = System.currentTimeMillis();
        System.out.println("byte stream time: " + (end1 - start1));
    }
}

```

```

character stream time: 537
byte stream time: 1097

```

As you can see from the output, the byte stream(1097) takes twice as long to read and write plain text files as the character stream(537), so the character stream is twice as fast. Therefore, this project adopts **character stream** for reading and writing.