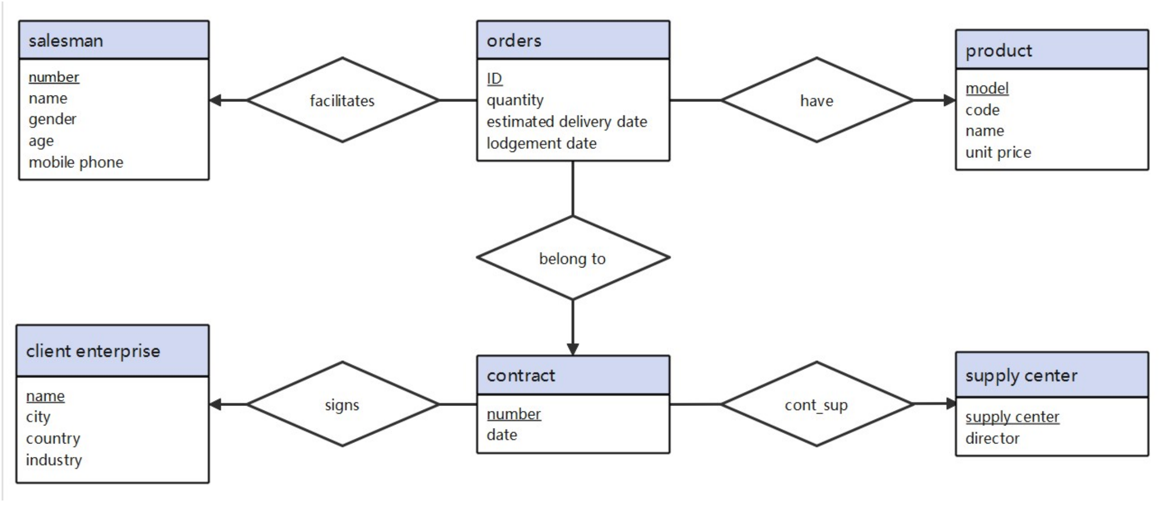
Spring 2022 CS307 Project1

Part1-OurTeam

王宇航 SID 12012208 Lab5	**李玉金 SID 12012225 Lab5**
Tasks:	Tasks:
Design the tables and columns	Design an E-R Diagram
Data Import	Data import
Compare DBMS with File I/O	Write paper
Percentage of contribution:50%	Percentage of contribution:50%

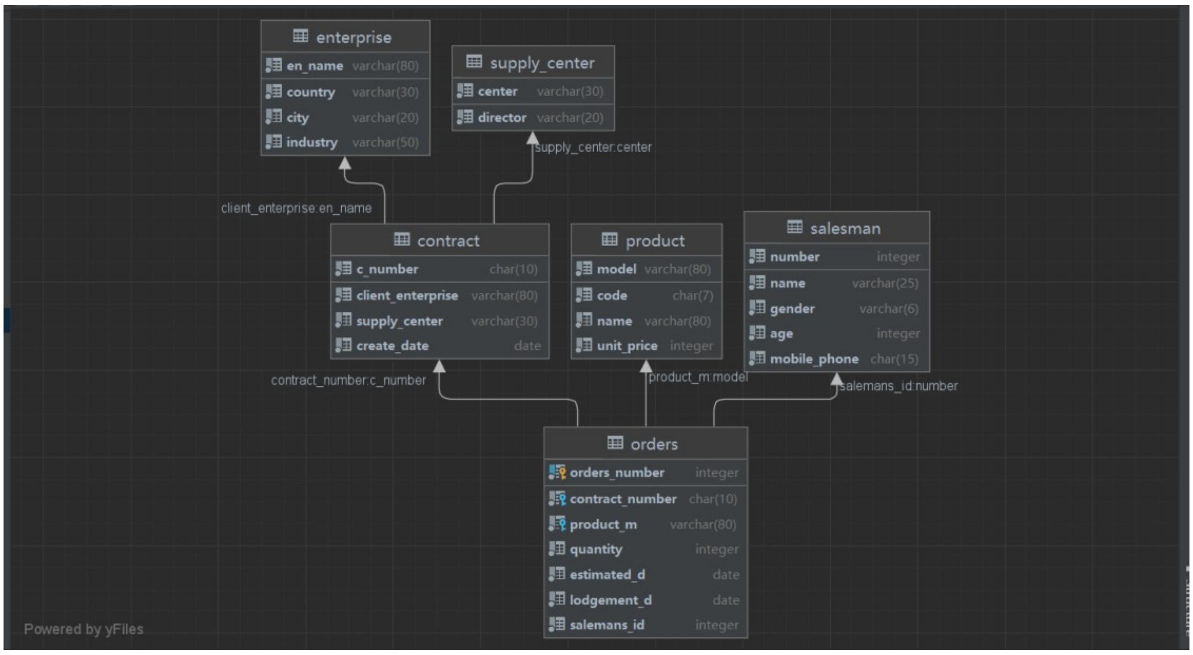
Part2-E-R Diagram



Software used for drawing the diagram：亿图图示

Part3-Database Design

3.1)E-R diagram generated by DataGrip



3.2)Description of the table "orders"

- Design of the table and column

The table "orders" can be perceived as a relational table that joins multiple other tables, with many foreign keys. This table represents the relationship between each order. The attributes of an order include the contract to which it belongs, the product ordered (down to the specific model), the quantity, estimated delivery date, lodgement date and finally the salesman responsible for the order. Due to the fact that there are multiple orders in one contract, the contract_number is used as the foreign key of the contract table, thus connecting the table "orders"with the table "contract". Similarly, product_m and salesmans_id are used as foreign keys for other tables.

- Details description

1）We set up an increment orders_number as the primary key, which can uniquely identify each of the 50000 orders.

2）For contract_Number, we need to check that it starts with "CSE" and has a length of 10.

3）The two attribute data types representing the time are set to "date"。For lodgement_d, it has two possibilities, one is empty, the other is there is a corresponding date, which we need to check if it is earlier than 2022-03-02.

4）The attributes "contract_number,product_m, and salemans_id" are used as foreign keys of other tables and need to be marked.

3.3)Description of the table "contract"

- Design of the table and column

We believe that the attributes of a contract should include contract number, client company, supply center and the date of signing. In the same vein as the table "orders", "client_enterprise" and "supply_center" can be used as foreign keys to the company and supplier center tables. The primary key of this table is pointed to by the attribute "contract_number" of the table "orders".

- Details description

Since the attribute "create_date" does not show missing values in the given table, I assume that a contract must have a date on which it was signed, so I set it to non-null. The constraints of the remaining attributes are relatively simple and will not be described again.

3.4)Description of the table "enterprise"

- Design of the table and column

We think that the attributes of a company have company name, country, city and industry. Each of the non-primary key attributes is uniquely dependent on the primary key "en_name".

The primary key of this table is pointed to by the attribute "client_enterprise" of the table "contract".

- Details description

Note that when the country is not China, the city column is null, so there is no null constraint for the city property.

3.5)Description of the table "supply center"

- Design of the table and column

We believe that the attributes of supply center include the name of supply center and the director, who is completely dependent on the primary key "center".The primary key of this table is pointed to by the attribute "supply_center" of the table "contract".

- Details description

Notice that the director and the supply center satisfy one-to-one correspondence, so the unique constraint is added to the attribute "director".

3.6)Description of the table "product"

- Design of the table and column

We believe that product attributes include product name, specific product model, product code and unit price. Product name, product code, and unit price are all attributes used to describe the product. Due to the fact that the product name and code satisfy one-to-one correspondence, they are used to describe which family the product model belongs to. For example, "TVBaseR1" and "TVBase65" belong to the product family "TV Base" which corresponds to the product code "T67P542". The primary key of this table is pointed to by the attribute "product" of the table "orders".

- Details description

Note that the maximum length of the data type "varchar" is set to 80 because some extreme data is 60-70 characters long .

3.7)Description of the table "salesman"

- Design of the table and column

We think that the attributes of the salesman are id, name, gender, age,and phone number. Each of the non-primary key attributes is uniquely dependent on the primary key "number". The primary key of this table is pointed to by the attribute "salemans_id" of the table "orders".

- Details description

Note that the gender can only be male or female and is case insensitive, and the validity of the number and age should be simply determined.

Part4-Import data

4.1)Overview of data

Here we use python's library "pandas" to help us with the overview of the data. Here is the code and the results.

```
#Firstly we load the csv file here. As shown in the figure, which is utf-8
encoding, it is found that #there are many empty values in the two attributes of
"city" and "lodgement date". 134 rows of #lodgement data are empty, so it needs
to be careful when injecting data. Meanwhile, each of several #time-related
attributes in the table is of the type "object" instead of type "datatime".

import pandas as pd
import chardet as cd
'''Open the file in binary mode and read the first 100000 characters to guess the
encoding  '''
with open("contract_info.csv",'rb') as temp:
    encodingType=cd.detect(temp.read(100000))
    print(encodingType)
hw=pd.read_csv("contract_info.csv",encoding=encodingType['encoding'])
print('overview')
hw.info()
```

```
{'encoding': 'utf-8', 'confidence': 0.99, 'language': ''}
overview
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 21 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   contract number          50000 non-null  object
 1   client enterprise        50000 non-null  object
 2   supply center            50000 non-null  object
 3   country                  50000 non-null  object
 4   city                     7007 non-null   object
 5   industry                 50000 non-null  object
 6   product code             50000 non-null  object
 7   product name             50000 non-null  object
 8   product model            50000 non-null  object
 9   unit price               50000 non-null  int64
 10  quantity                 50000 non-null  int64
 11  contract date            50000 non-null  object
 12  estimated delivery date  50000 non-null  object
 13  lodgement date           49866 non-null  object '''134 rows of data are
empty'''
 14  director                 50000 non-null  object
 15  salesman                 50000 non-null  object
 16  salesman number          50000 non-null  int64
 17  gender                   50000 non-null  object
 18  age                      50000 non-null  int64
 19  mobile phone             50000 non-null  int64
 20  Unnamed: 20              0 non-null      float64
dtypes: float64(1), int64(5), object(15)
```

```
memory usage: 8.0+ MB
```

4.2)Setting up the operating environment

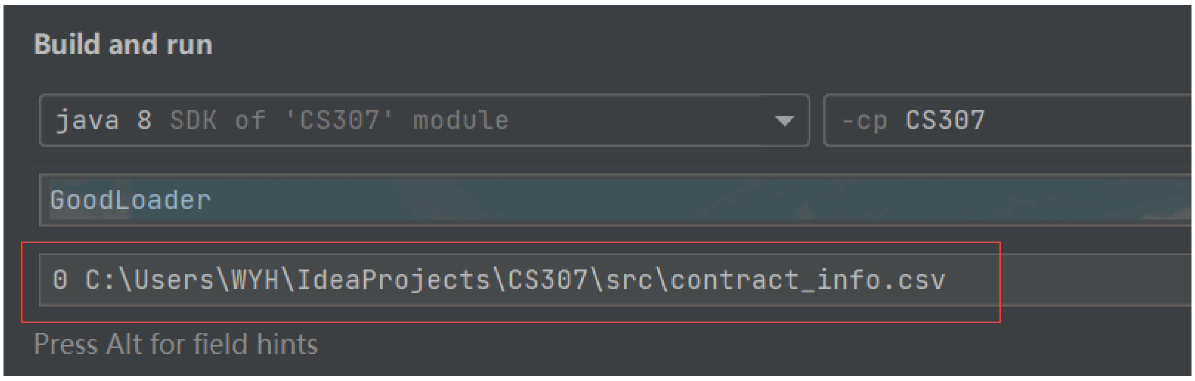
Here, we import data in the way of Java script. We have two scripts, Averageloader and GoodLoader, and the difference between them will be introduced later. The next part mainly explains how to set up the script running environment.

- Put drive postgresql-42.2.5.jar, AverageLoader.java,Goodloader.java, contract_info.csv, loader.cnf, and datainput.iml in the same src directory. Then put datainput.iml and src in the same module in order to use relative path to lookup contract_info.csv and loader.cnf.
- Add the postgresql-42.2.5.jar path to the Project Structure Libraries.
- Empty the tables in Datagrip.

4.3)Setting running Parameters

Since there are dependencies between tables, there are also certain orders when importing data. Before running, we should set relevant parameters.

As shown, the first parameter represents which table to import, and the second parameter is the path to the original csv file.



```
0 C:\Users\WYH\IdeaProjects\CS307\src\contract_info.csv
```

We need to import 6 times in total, and the first parameter should be from 5 to 0 in order. The value of the first parameter and the imported table are shown below.

5: supply center   4:salesman   3:product 2:enterprise 1:contract 0:orders

4.4)Explanation for important code

- Here we use AverageLoader as an example to explain important code.
- Global variable

```
//Stmt[] corresponds to the injection statements of six kinds of tables. Since
precompiled statements are used, this method only passes parameters and executes
in compiled SQL statements, avoiding repetitive operations generated by dynamic
compilation and avoiding SQL injection. We use this approach to improve
efficiency in compiling SQL statements.
private static PreparedStatement[]  stmt = new PreparedStatement[6];
```

```
//Load variables for insert statements. We opening arrays to load data of a
particular length for a particular table.
private static SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");

//To find whether dates in the later section are valid, set a critical date value
in advance.
private static Date flagDateformat.parse("2022-03-02");
static {
    try {
        flagDate = format.parse("2022-03-02");
    } catch (ParseException e) {
        e.printStackTrace();
    }
}
```

- openDB

```
// The main function of this function is to link the database according to the
parameters filled by the user, and convert the statements in stmt [] into SQL
statements to execute in the database.
// This function is called in the main function, where we must fill in the
corresponding database parameters.
```

- method loadData

```
//The main function of the loadData function is to load the dataList parsed from
the file into stmt[] and convert it into SQL statement for execution. Run data
injection statements for different tables based on different index values, with
additional constraints. The following is an example of loading the orders table.
(Here index=0)
private static void loadData(List<String> dataList,int index)

// We should be careful that the dataList is a character linked list type data.
So for the quantity of goods, we need to convert it to type "Int", and for the
date of order signing, we need to convert it to type "date", and we need to
determine whether the lodgement date is less than the limit of "2022-03-02".
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
java.sql.Date d;
stmt[index].setString(1, dataList.get(0));
stmt[index].setString(2, dataList.get(1));
stmt[index].setInt(3, Integer.parseInt(dataList.get(2)));//Converts a string to
an integer.
if (Objects.equals(dataList.get(3), "")) {
    //If the date is empty, it is set to a null value of the date type.
    stmt[index].setNull(4, Types.DATE);
} else {
    //Otherwise, convert the string to date type.
    Date date = simpleDateFormat.parse(dataList.get(3));
    d = new java.sql.Date(date.getTime());
    stmt[index].setDate(4, d);
}
if (Objects.equals(dataList.get(4), "")) {
    stmt[index].setNull(5, Types.DATE);
}else {
    Date date = simpleDateFormat.parse(dataList.get(4));
    d = new java.sql.Date(date.getTime());
    if(d.compareTo(flagDate) > 0){
        //If the date does not meet the limit, it is set to a null value of the
date type.
        stmt[index].setNull(5, Types.DATE);
    }
    else {
        stmt[index].setDate(5, d);
    }
}
stmt[index].setInt(6, Integer.parseInt(dataList.get(5)));
```

- main method

```
//In the main method, we first read the configuration file, then define the
parameters to access the database, then we read the file with the following code.
We first read data into memory, then generate SQL statements from memory to
execute, and finally write to the database.
//Set the start time.
          start = System.currentTimeMillis();
//Open DB before the loop starts and close it after the loop ends, thus avoiding
repeated operations of opening and closing.
           openDB(prop.getProperty("host"), prop.getProperty("database"),
                  prop.getProperty("user"),
prop.getProperty("password"),index+1);
            infile.readLine();
//It is read line by line into the buffer and loaded into stmt[]PreparedStatement
by loadData function. Here are a series of judgments. Take an example of reading
data from a Supply Center table. The data structure of hashset is adopted here,
for example, the supplyMap below can be determined whether the element is
contained in the set by O(1), effectively achieving the purpose of de-
duplication.
           while ((line = infile.readLine()) != null)
               if(index==5){
                if(supplyMap.contains(parts[2])){continue;}
                    dataList.add(parts[2]);dataList.add(parts[14]);
                    supplyMap.add(parts[2]);
                    loadData(dataList,index);
                }
//Set the end time.
           end = System.currentTimeMillis();
```

4.6)Improved script

- Train of thought

In the optimization of the code, we try to use batch processing to save time in the process of data transmission. The specific method is that in the process of executing SQL statements, we process multiple statements at a time, and send multiple statements to the database server, and finally execute multiple statements at a time in the server.

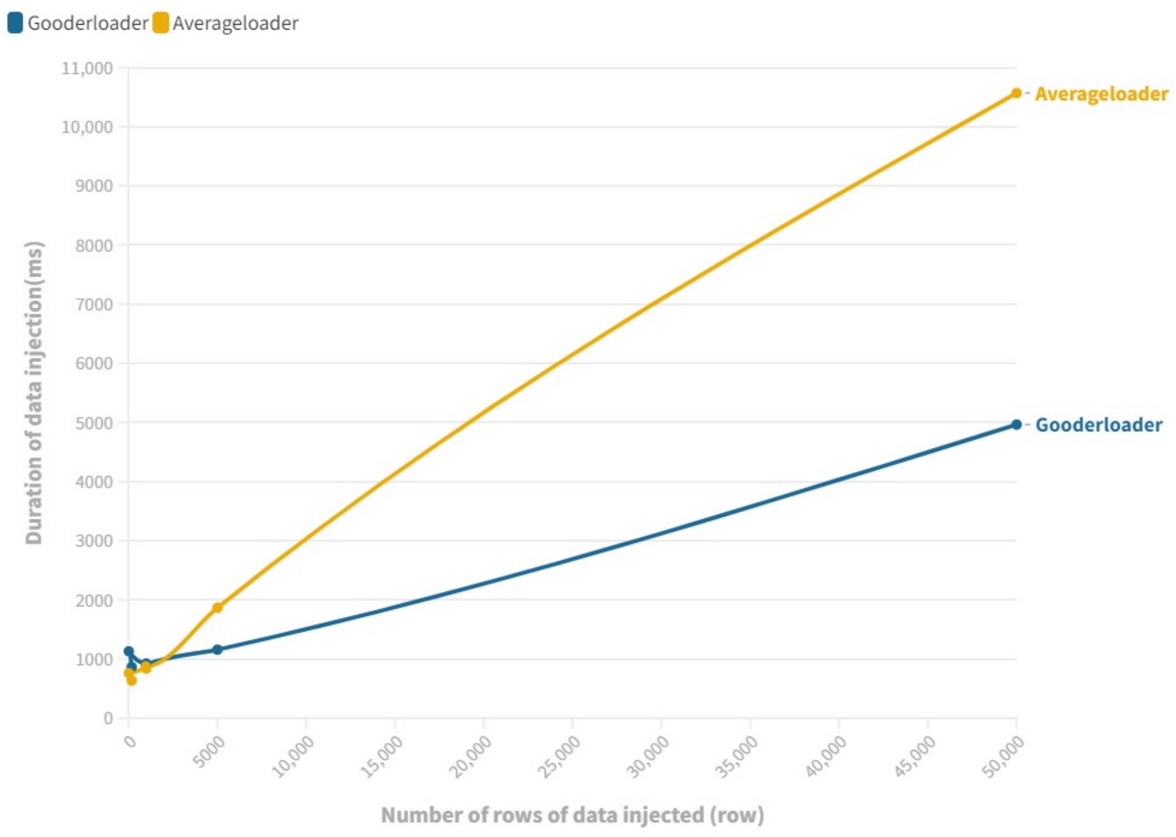
- Concrete operation

We first initialize a "BATCH_SIZE". The SQL statement is then added to batch each time it is executed, and a counter is incremented each time when the sql statement is inserted. When the number of times in loadData reaches a certain value, the loadData is executed uniformly. Finally, empty Batch to continue receiving data with Batch in the next loop.

- Code

```
//Here is a global variable definition. After many experiments, we found that
BATCH_SIZE set to 1000 works best for the current database import.
private static final int  BATCH_SIZE = 1000;

// The SQL statement is added to batch each time it is executed, and a counter is
incremented each time when the sql statement is inserted. When the number of
times in loadData reaches a certain value, the loadData is executed uniformly.
Finally, empty Batch to continue receiving data with Batch in the next loop.
stmt[index].addBatch();
if (cnt % BATCH_SIZE == 0) {
     stmt[index].executeBatch();
     stmt[index].clearBatch();
}
```

- Comparison of running time (the following data were preheated and averaged repeatedly)



Analysis：At around 2500 rows or less, GoodLoader is close to AverageLoader, and AverageLoader is even more efficient. However, after that, the efficiency of GoodLoader is significantly higher than that of Averageloader, and with the increase of the number of statement lines, the gap between these two scripts rapidly expands. When loading the same 50000 lines of data, Averageloader takes twice the time of GoodLoader. Moreover, the time gap can be reliably predicted to increase if the number of inserted data rows continues to increase.

4.7)Data injection with python pandas

We also use Python's pandas library to perform data injection. Pandas is a good data analysis tool that inherits a large number of APIs to simplify our work.

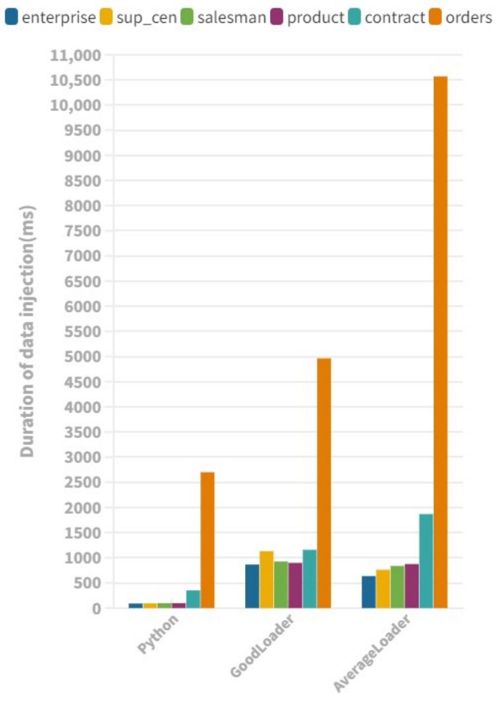
- Explanation for important code

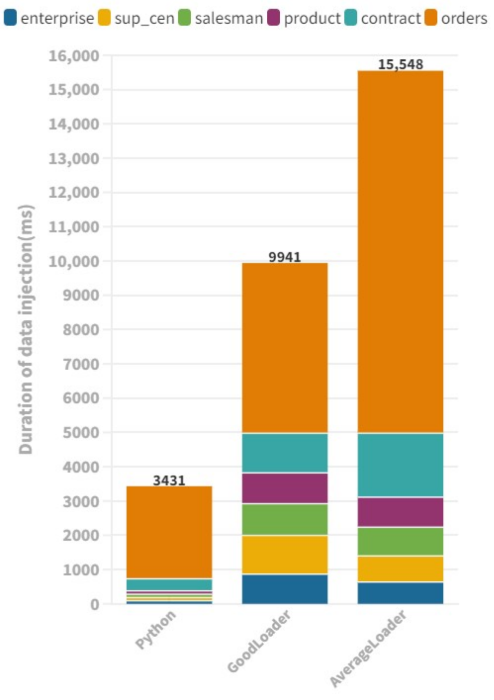
```
#We first define a function, df is the data we need to import, and it is in
dataframe format. Table_name is the name of the table in the database we want to
import.
def write_to_table(df, table_name):
    # Connect the database.
    db_engine  = create_engine('postgresql+psycopg2://' + 'username' + ':' +
'password' +
                               '@localhost' + ':' + str('port') + '/' +
'database name')
    string_data_io = io.StringIO()

    # First change the DataFrame format into the csv format.
    df.to_csv(string_data_io, sep='|', index=False)
    # Change data into the format of table which is in sql by sql_builder of
pandas.
    pd_sql_engine = pd.io.sql.pandasSQL_builder(db_engine)
    table = pd.io.sql.SQLTable(table_name, pd_sql_engine, frame=df,
                               index=False, if_exists='append',schema =
'public')
    #If there is data in the table, add more data below. The corresponding schema
is public.
    table.create()
    string_data_io.seek(0)

    # Copy the table to the database using the SQL COPY command.
    with db_engine.connect() as connection:
        with connection.connection.cursor() as cursor:
            copy_cmd = "COPY public.%s FROM STDIN HEADER DELIMITER '|' CSV"
%table_name
            cursor.copy_expert(copy_cmd, string_data_io)
        connection.connection.commit()
```

```
#Only the import of the orders table, which is the most complex, is described
here. The remaining code can be seen in attachments.
import numpy as np
time_start=time.time()
orders_f=hw[['contract number','product model','quantity','estimated delivery
date','lodgement date','salesman number']].copy()

#Convert the lodgement date to the time type, and set the value to null if the
value does not meet the requirements.
orders_f['lodgement date']=pd.to_datetime(hw['lodgement
date'],format='%Y/%m/%d')
orders_f.loc[orders_f['lodgement date']>pd.to_datetime('2022-03-02'),['lodgement
date']]=None
```

```
#Generate an increasing matrix and change it to the shape of N x 1 for our orders
number (1-50000).
data=np.arange(1,len(orders_f)+1).reshape(len(orders_f),-1)
data
data_f=pd.DataFrame(data)

#Merge the two dataframes and write them to the database.
orders=pd.concat([data_f,orders_f],axis=1)
orders.to_csv('orders.csv',index=False)
write_to_table(orders,'orders')
time_end=time.time()
print('shape of data: ',orders.shape[0], ' rows')
print('totally cost',time_end-time_start, 's')
```

- Comparison of running time (the following data were preheated and averaged repeatedly)





Analysis: Both the data injection time of each table and the total data injection time of python program written by us are much faster than Averageloader and Goodloader. This is due to the fact that many of the functions in pandas are extremely optimized , and they are specifically aimed at the field of data analysis, so when used correctly, results will generally be faster than individual implementations.

4.8)Test environment

- Hardware specification

Computer mainboard：Lenovo LNVNB161216

CPU：intel(R)Core(TM)i7_8750HCPU@2.20GHz。

Memory：8GB

Primary hard drive：512GB/SSD

GPU：6GB/Lenovo

- Software specification

Operating system： Windows 10 家庭版64位

DBMS：postgreSQL

Programming language：java and python

Part5-Compare DBMS with FileI/O

5.1)Test environment

- Hardware specification

CPU：AMD Ryzen 5 4500U with Radeon Graphics   2.38 GHz

Memory:16.0 GB (15.2 GB available)

Primary hard drive：512GB/SSD

- Software specification

Operation System：Window11，x64

Python： Python 3.9.10

java： java version "1.8.0_221"

Pandas : pandas 1.4.1

DBMS：postgreSQL

5.2)Organization the test data in the DBMS and the data file

5.2.1)DBMS

We first use DBMS to operate the database. Here I use Python as a programming language to link the database and execute the corresponding statements. The code is relatively simple, so just an example of a query is illustrated.

```
#Connect the database.
db_engine  = create_engine('postgresql+psycopg2://' + 'username' + ':' +
'password' + 'IP' + ':' +
                           str('port') + '/' + 'databasename')

#Execute the corresponding statement.
time_start=time.time()

#We can execute sql statements directly.
databaseData = pd.read_sql("select distinct(salemans_id) from
orders",con=db_engine)
time_end=time.time()
print('DBMS totally cost',(time_end-time_start)*1000, ' ms')
```

5.2.2)FIle I/O

Here we use the Java language for file IO processing

- Required documents and specific parameter settings

  File: FileIO.java DataManipulation.java Clinet.java DataFactory.java

  File relationship: Clinet.java is the program entry, and it contains main function. FileIO.java is a subclass of DataManipulation.java, and DataManipulation.java acts as an interface to Clinet.java.DataFactory is the implementation of the interface.

  Parameter setting: Since we are operating on files, the operation parameter is set to file .

- Operational approach

  Since it is difficult to compare time if the data set is too small, the objects of single-table operations on this task are all the rders table (50,000 rows of data). The Clinet.java file can be manipulated to perform different functions. All the output of this program is output to a new csv file in the current directory, and the time to write the file is also counted as the running time. Here we define seven functions that we can use to perform the purpose of a particular function by commenting out the rest of the functions, and then check its running time separately. The functions of each function will be explained below.

```
public static void main(String[] args) {

long start,end;
start = System.currentTimeMillis();
DataManipulation dm = new DataFactory().createDataManipulation(args[0]);
//dm.getColumnMap(6);
//Get a list which contains every salemansID that appear in orders and print
it to Map.csv.  We can also select the value of the other de-duplicated
column by changing the method parameters.

//dm.getALLInformationByRule();
//Filter all rows in the orders table with quantity 480.
//Print to infByrule-csv.

//dm.groupByMax();
//Count the quantity of the highest volume orders in each saleman's orders
in orders table.
//Print to groupByMax.csv.

//dm.addOneData(",CSE0004999,TvBaseR1,1,2022-01-01,2022-01-
01,11211429",10000,50000);
//Add 10000 rows of data to Orders and check if the restrictions are met,
and the result is printed to orders.csv.

//dm.join(3);
//Add join operations to the orders table and other tables.The parameters
can be 3 or 4, 3 representing join Product table and 4 representing join
Saleman table. The results are printed to join.csv.

//dm.reSetData("CSE0004999");
//The quantity of all contracts with the contract No. CSE0004999 is doubled.
The result is printed to reorder.csv.

//dm.deleteData("CSE0004999",1);
```

```
//Delete the contract whose contract number is CSE0004999 and output the
result to delorder.csv.
end=System.currentTimeMillis();
System.out.println("The statement you choose to run" + (end - start) + "
ms");
```

- Specific function description

We take a concrete example to illustrate.This function is groupByMax. Other functions will not be described, details can be seen in the code attachment.

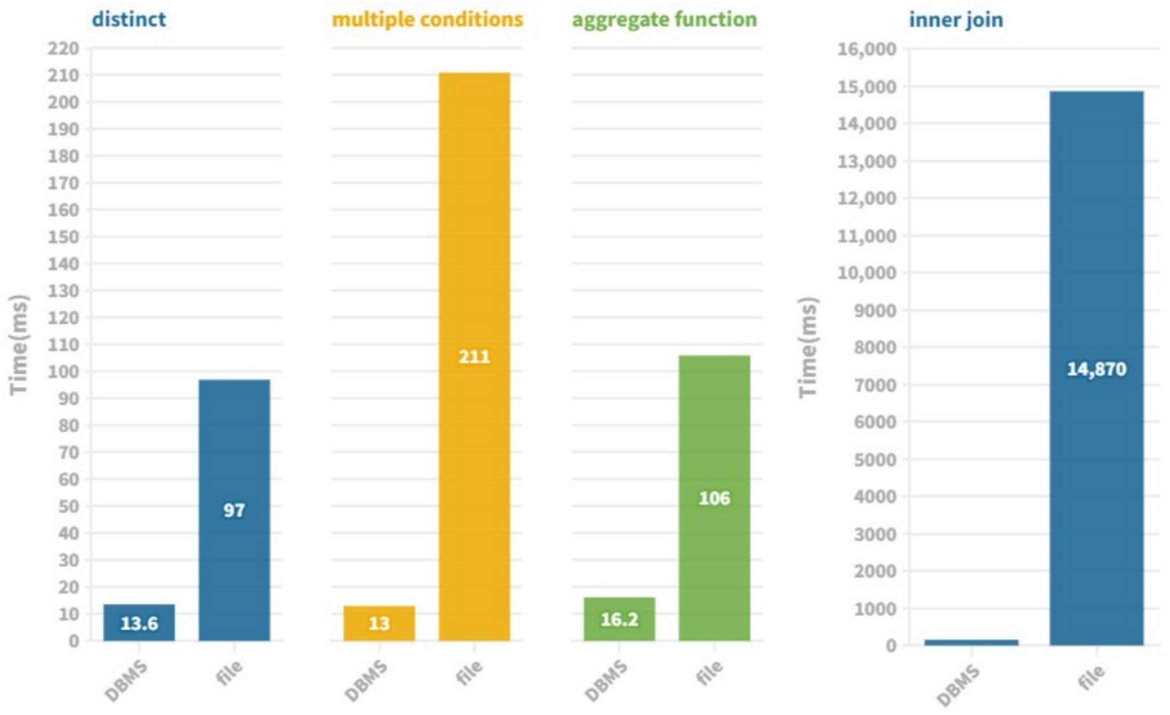
```
public void groupByMax() throws IOException, ParseException {
        String line;
        String[] splitArray;
        //The object is groupByMax.csv.
        File writeFile = new File("groupByMax.csv");
        HashMap<String,Integer> group=new HashMap<>();//Carries the current
quantity maximum for each saleman
        BufferedWriter writeText = new BufferedWriter(new
FileWriter(writeFile));
        int counter=0;
        //Writes the attributes of the table.
        writeText.write("salemans,quantity");
        writeText.newLine();
        try (BufferedReader bufferedReader = new BufferedReader(new
FileReader("path"))){
            bufferedReader.readLine();
            String str;
            while ((line = bufferedReader.readLine()) != null) {
                splitArray = line.split(",");
                str=splitArray[6];
                if(group.containsKey(str)){
                    //If it is greater than the current maximum value, it is
updated.
                    if(Integer.parseInt(splitArray[3])>group.get(str)){
                        group.put(str,Integer.parseInt(splitArray[3]));
                    }
                }
                else{
                    group.put(str,Integer.parseInt(splitArray[3]));
                }
            }
            for (Map.Entry < String, Integer > entry: group.entrySet()) {
                //Iterate over the HashMap, and write the data to the buffer.
                writeText.write(entry.getKey()+","+entry.getValue());
                writeText.newLine();
                counter++;
            }
            if(counter%500==0){
                //If 500 pieces of data have been written, the buffer is cleared
and the data is written to the file.
                writeText.flush();
            }
            writeText.flush();
            writeText.close();
        }
    }
```

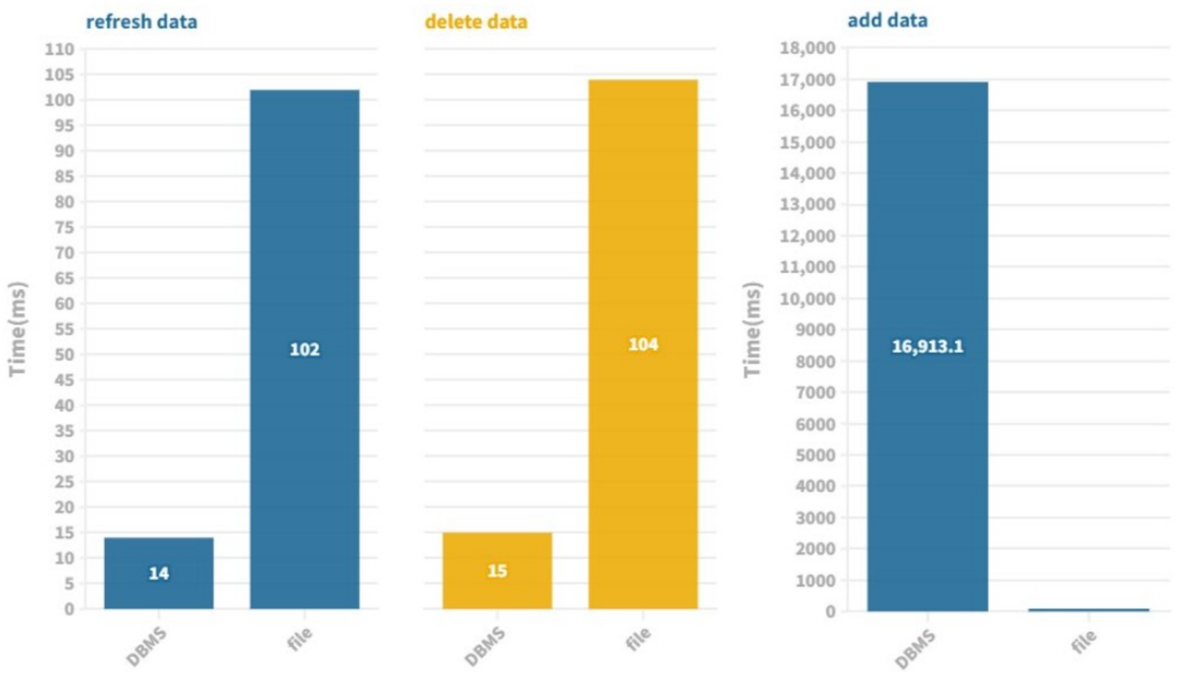
5.2.3)Run time comparison of DBMS and the fileI/O

In order to ensure the correctness of the experiment, we executed the statement with the same purpose in both operations so that we could compare the difference in running time between the two operations

The following figure compares the speed of the two operations in the query statement. All experiments were preheated and the average value was selected repeatedly. The data size of each operation is given in the table below

Function	Volume of operatied data
distinct	50000
mulitiple conditions	50000
arrregate function	50000
inner join(orders and product)	50000x961
reSetData	50000
deleteData	50000
addData	50000+10000





We can see that whether single condition, multi-condition query, aggregate function, update operation, or delete operation, DBMS performance are far better than the file operation, which shows that the database language has a huge advantage over the general file IO operation in the data operation. However, we can see that the DBMS is much slower than fileIO for adding data for the reason that I didn't do any optimization to the DBMS, just executing a thousand statements sequentially , where each one is committed as a transaction . And the file operation saves such trouble. It can batch write data through the buffer, and the examination of foreign keys and other constraints on the table is not as meticulous as DBMS. Thus this phenomenon, which is the opposite of the previous results, appears.

5.3)File I/O with pandas

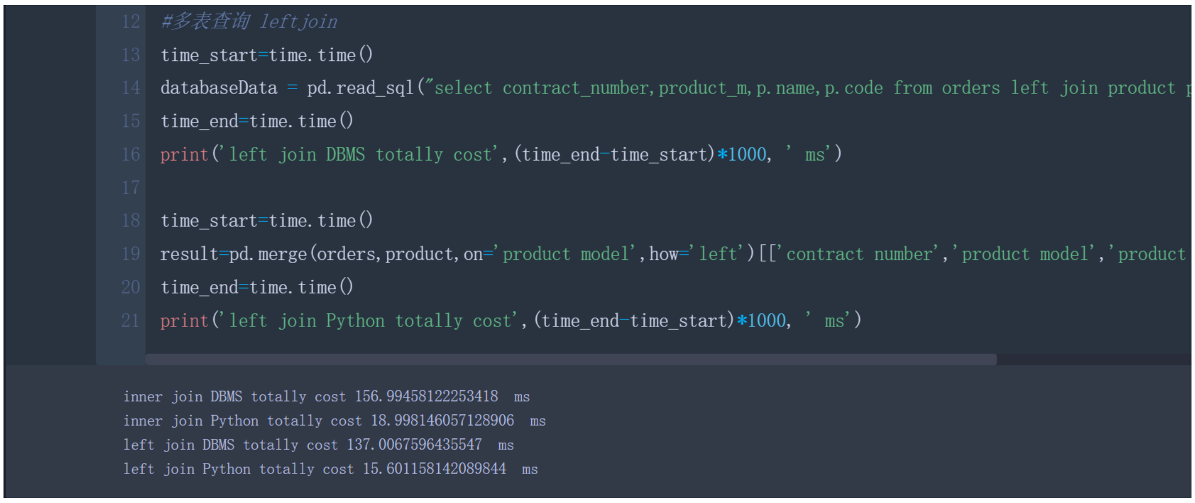
We then use pandas to process the csv file. We correspond the tables in the database to the Dataframe in pandas, and then use the pandas API for data analysis.

5.3.1)New test function of pandas

During using pandas, we fist reproduce all of the functions of IO, and then implement several data-heavy manipulations to test performance, which will be introduced detailedly below.

- left join

The following is the implementation of leftJoin: the first section is the operation by the DBMS, and the second section is the operation by the pandas on the file. It can be seen that pandas has a simple syntax for joining operations so that we can complete join operation with simple grammar.

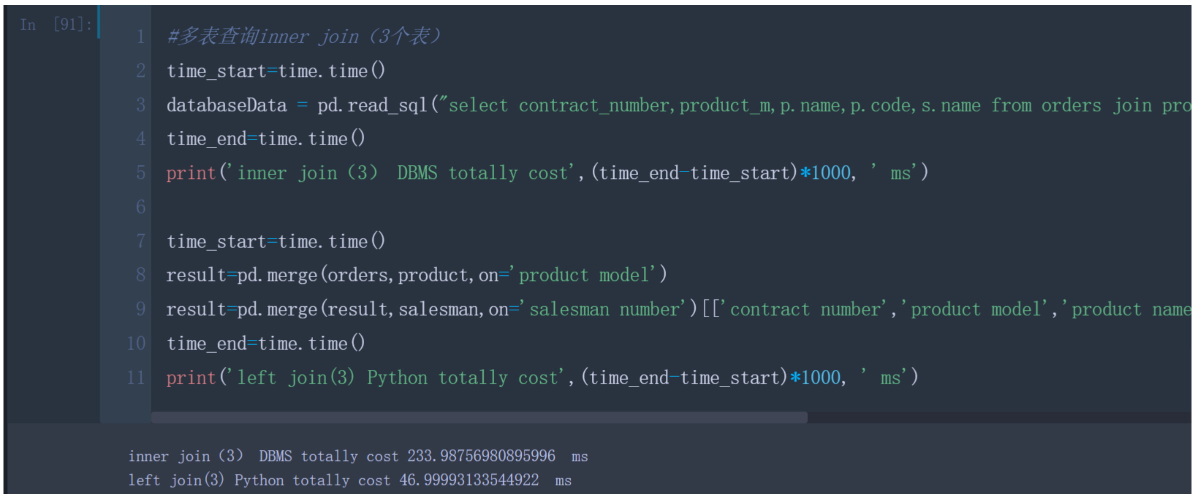


```
#多表查询 leftjoin
time_start=time.time()
databaseData = pd.read_sql("select contract_number,product_m,p.name,p.code from orders left join product p
time_end=time.time()
print('left join DBMS totally cost',(time_end-time_start)*1000, ' ms')

time_start=time.time()
result=pd.merge(orders,product,on='product model',how='left')[['contract number','product model','product
time_end=time.time()
print('left join Python totally cost',(time_end-time_start)*1000, ' ms')
```

```
inner join DBMS totally cost 156.99458122253418  ms
inner join Python totally cost 18.998146057128906  ms
left join DBMS totally cost 137.0067596435547  ms
left join Python totally cost 15.601158142089844  ms
```

- inner join (three tables)

The following is the inner join operations on the three tables. We join the orders, salemans, and product tables by the foreign key of orders and select the columns we are interested in.



```
In [91]:
#多表查询inner join (3个表)
time_start=time.time()
databaseData = pd.read_sql("select contract_number,product_m,p.name,p.code,s.name from orders join pro
time_end=time.time()
print('inner join (3) DBMS totally cost',(time_end-time_start)*1000, ' ms')

time_start=time.time()
result=pd.merge(orders,product,on='product model')
result=pd.merge(result,salesman,on='salesman number')[['contract number','product model','product name
time_end=time.time()
print('left join(3) Python totally cost',(time_end-time_start)*1000, ' ms')
```

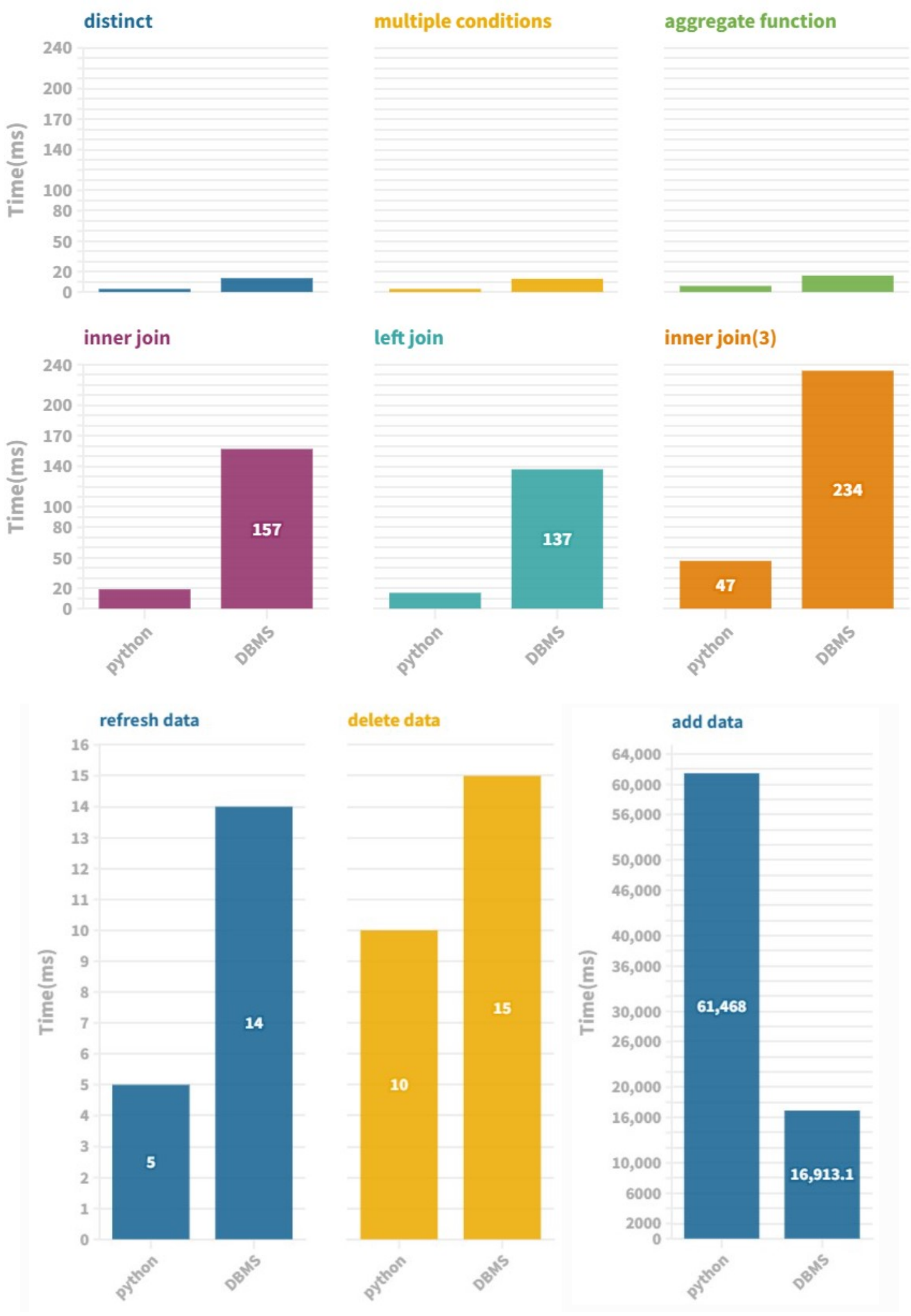
```
inner join (3) DBMS totally cost 233.98756980895996  ms
left join(3) Python totally cost 46.99993133544922  ms
```

5.3.2)Comparison of running time of pandas and DBMS

The single-table operation objects are all orders tables. The join operation involves three tables: orders, product and saleman. The data volume of the new operation is given in the form of the table below. To ensure the correctness of the experiment, the purpose of the statement is the same in each of the three different operations. All experiments were preheated and the average value was selected repeatedly.

Function Volume of operatied data	
left join(orders and product)----The standard file I/O is not implemented	50000x961
inner join(3)(orders and product and saleman)----The standard file I/O is not implemented	5000x961x990

- The following figure compares pandas with the DBMS on each operation.



We can find that pandas performs better than DBMS in all operations except addData. The reason why addData Python is slower is unknown to me due to limited time. It may be related to the dataframe storage structure and my failure to gracefully implement the function. This problem can be used as an area for improvement in the future.

Part6-A preliminary exploration of the DataGrip index

6.1)Overview

In PostgreSQL, indexe is a special kind of database object that are used to speed up data access. Currently, PostgreSQL provides B-tree, Hash, GiST, SP-gist, GIN, RUM, BRIN, and Bloom indexes. The two most commonly used indexes are B-Tree and GiST, and we mainly explore B-tree indexes. Although there are some differences between index types, each of them will eventually associate a key with the row containing the key. Each line of the index is identified by a tuple id, which consists of the block number in the file and the position of the line inside the block. That is, when there is a large amount of data in a table, after creating an index on a conditional field of the table, we can quickly read rows that might contain information of interest to us without scanning the entire table by using a known key or some information about it.

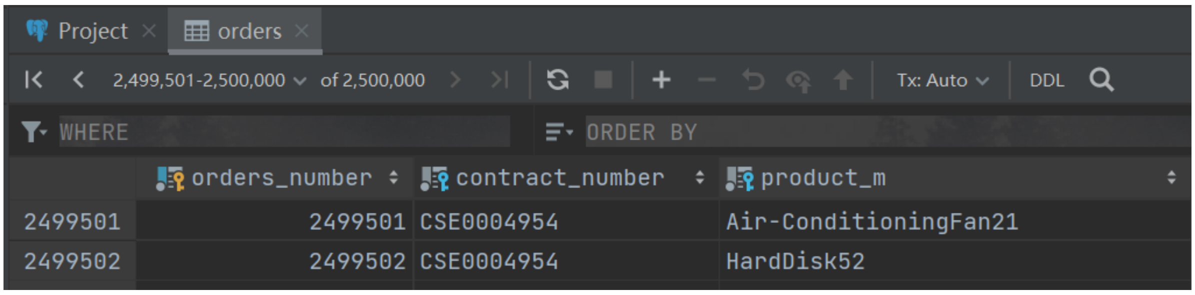
6.2)The advandages and dis advantages of index

The biggest advantage of index creation is that it can accelerate the speed of query and retrieval, which is mainly used to speed up conditional query, delete, update, join operation, foreign key constraint update and delete operation, etc., which is also the most important feature of index. Yet the speed brought by indexing does not come without cost. For every operation on indexed data, whether rows are inserted, deleted, or updated, the index of the table also needs to be updated. There is also an additional time and space overhead associated with creating an index.

6.3)B-tree index

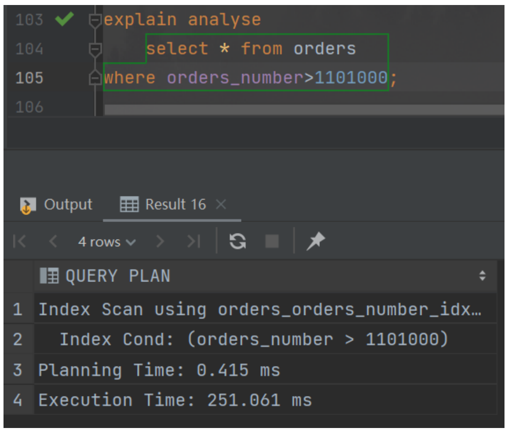
The B-tree index is the default and most commonly used index type in PostgreSQL. It is suitable for handling “=”, “<”, “>”, “<=”, “>=”, which are sequentially stored data, and other equivalent operations such as BETWEEN, IN, IS NULL, and fuzzy queries "LIKE '%s%'" on strings.

To be able to significantly differentiate the speed of the different approaches, I scaled up the Orders table to 2.5 million rows by copying the original Orders table 50 times and then resetting the increment sequence. Orders after capacity expansion is shown below.



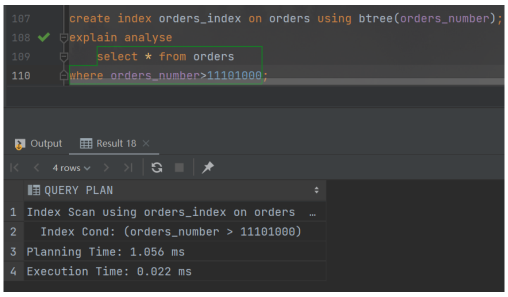
	orders_number	contract_number	product_m
2499501	2499501	CSE0004954	Air-ConditioningFan21
2499502	2499502	CSE0004954	HardDisk52

- We first analyze the case of a single index



```
explain analyse
    select * from orders
where orders_number>1101000;
```

```
Output    Result 16
4 rows
  QUERY PLAN
1 Index Scan using orders_orders_number_idx…
2   Index Cond: (orders_number > 1101000)
3 Planning Time: 0.415 ms
4 Execution Time: 251.061 ms
```



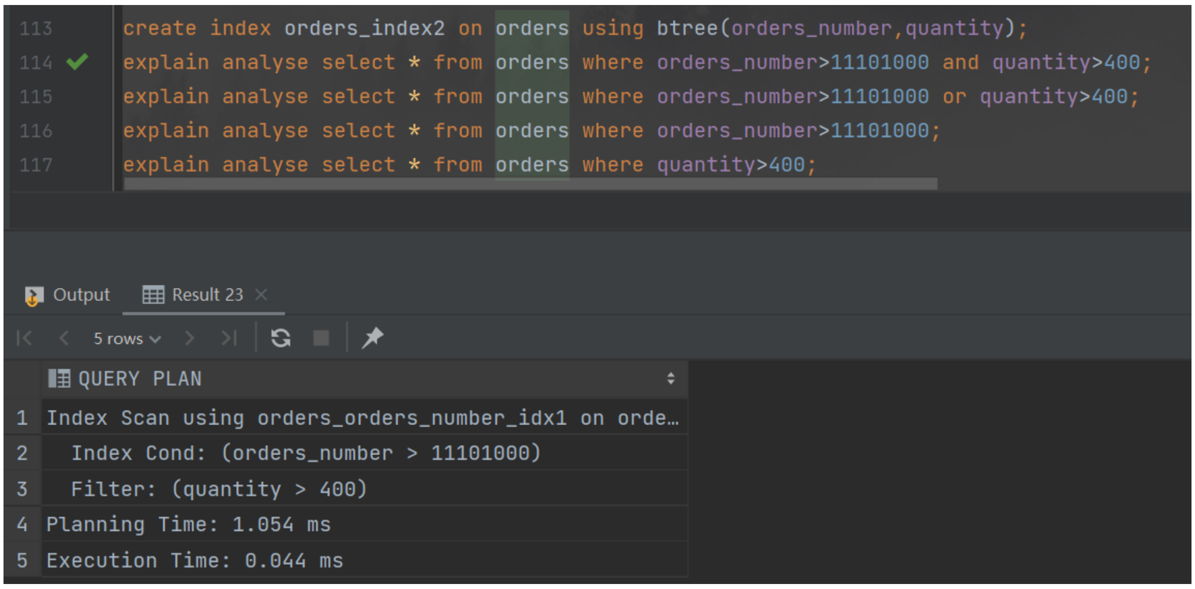
```
create index orders_index on orders using btree(orders_number);
explain analyse
    select * from orders
where orders_number>11101000;
```

```
Output    Result 18
4 rows
  QUERY PLAN
1 Index Scan using orders_index on orders  …
2   Index Cond: (orders_number > 11101000)
3 Planning Time: 1.056 ms
4 Execution Time: 0.022 ms
```

As shown in the figure above, after adding an index, "where" is followed by index-related criteria, resulting in a qualitative leap in query speed

- Then we delete the single index and add the federated index, as shown below. We execute ioint index in which different indexes are working. There are four cases.

  Only the results of the “and” query are shown here



```
create index orders_index2 on orders using btree(orders_number,quantity);
explain analyse select * from orders where orders_number>11101000 and quantity>400;
explain analyse select * from orders where orders_number>11101000 or quantity>400;
explain analyse select * from orders where orders_number>11101000;
explain analyse select * from orders where quantity>400;
```

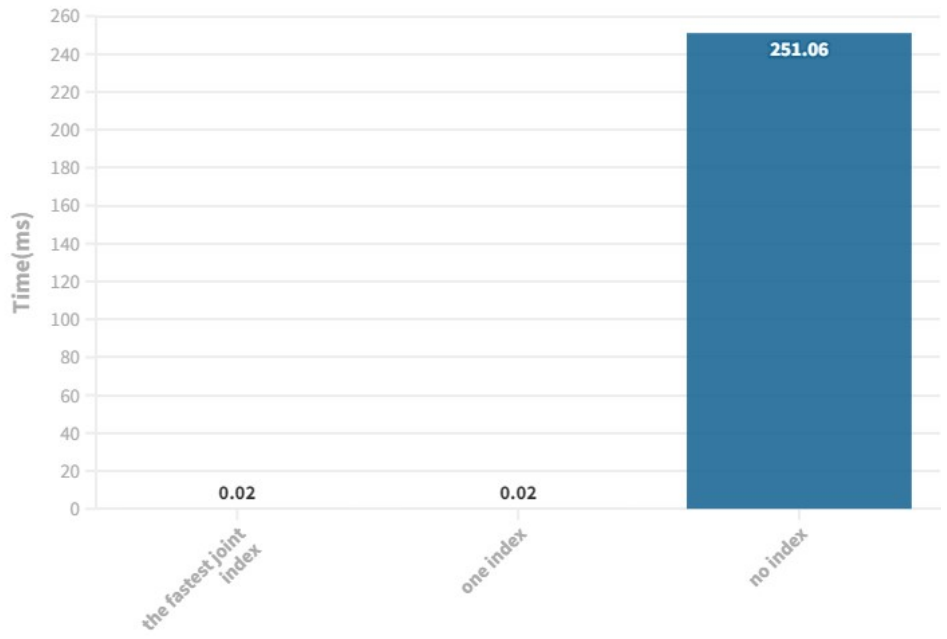
```
Output    Result 23
5 rows
  QUERY PLAN
1 Index Scan using orders_orders_number_idx1 on orde…
2   Index Cond: (orders_number > 11101000)
3   Filter: (quantity > 400)
4 Planning Time: 1.054 ms
5 Execution Time: 0.044 ms
```

Based on the results of the four queries, we can see that for the joint index, only if search criteria is the first index or "the first index and the second index", the query will only perform an index scan, otherwise it will still pass the full table scan.

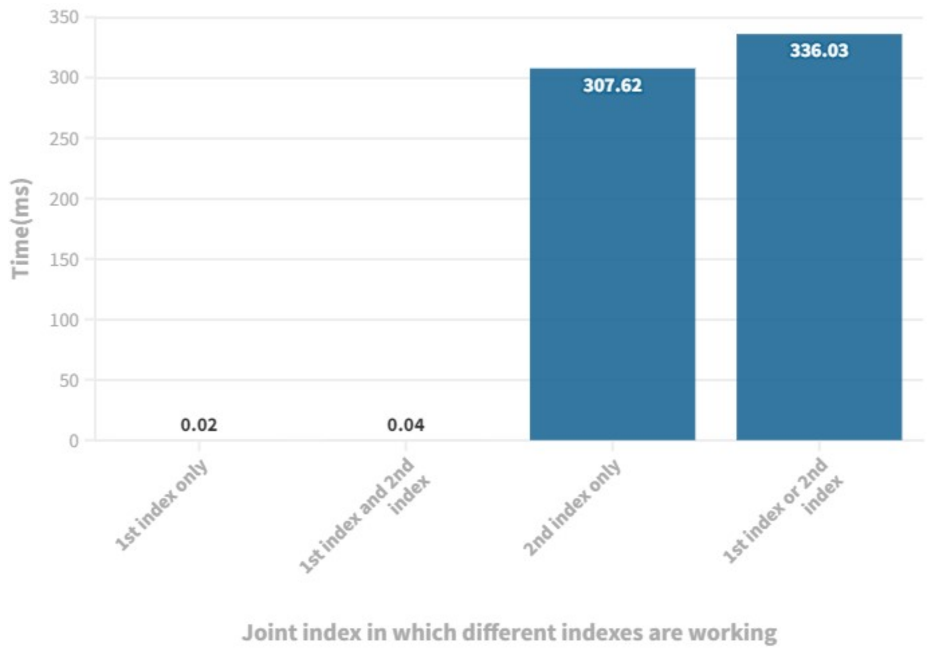
- Finally, we delete the joint index and create two single indexes to query "and" and "or" statements again. (Running result picture omitted)

  We can find that for two single indexes, the "and" result is similar to the joint index, while the "or" result does not change much compared to the no index.

Here we use a bar chart to better represent our results.



The fastest joint means that only the first index in a joint index is queried, and its speed is similar to that of a single index.



This figure shows the query of the joint index at a glance.

6.4)Index failure condition

Index failure refers to the fact that in some cases, when the where constraint is set improperly, even if the index exists and the constraint contains index columns, the query speed is almost as fast as no index. We found some common index failures through research and our own experiments. Constraints contain calculations or functions

1. "NOT" exists in the restriction condition .
2. Fuzzy query wildcard at the beginning of the character .
3. The queried fields occupy a relatively high proportion of data in the table .

We need to consider how to avoid the failure of the index, but due to time reasons, we failed to further explore, which is also where the project needs to be improved.

Part7-Conclusion

In this project, we first established a relational database according to the data information stored in csv files, and then successfully imported the data into it. In the process of import, we further deepen our understanding and application of data import knowledge by comparing the efficiency differences between different import methods. In addition, we analyzed the data in different languages and compared DML to Python and Java for different operations, which gave us an insight into the advantages of DML and pandas in manipulating data. In the continuous experiment and comparison, our understanding of database index is also deepening.