Spring 2022 CS307 Project Report

Group information

Name	Student ID	Teacher	Lab session	Contributions
睢和	12012929	Zhu Yueming	Tuesday-78	50%
徐剑	12010923	Zhu Yueming	Tuesday-78	50%

Contribution:

Xu Jian : import data to file

Draw ER model

Participate in database design

Compare File I/O with DBMS

Generate test data

Compare index

Sui He: participate in ER model

import data to database

Participate in database design

Optimize code

High concurrency and transaction management

User privileges management

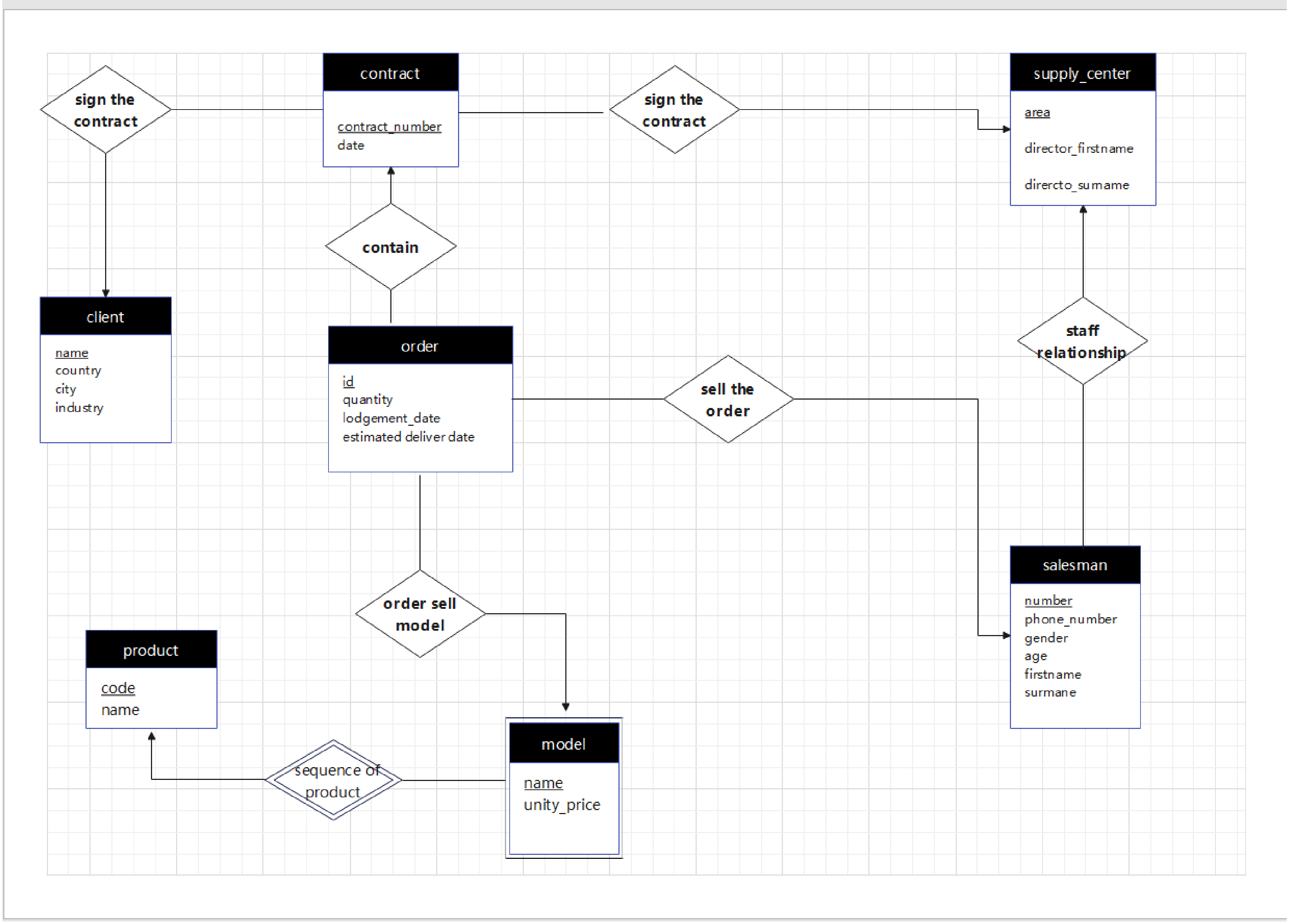
Database index and file IO

Test Environment

CPU model: Intel i7-10710U, Intel i5-10210U	java version: 8
Size of memory: 16.0 GB	compiler version : jdk1.8.0_241
SSD: no	jar: postgresql-42.2.5
HDD: no	Software: IDEA
DataGrip version: 2021.2.4	Programming language: java
Operation system: win11	

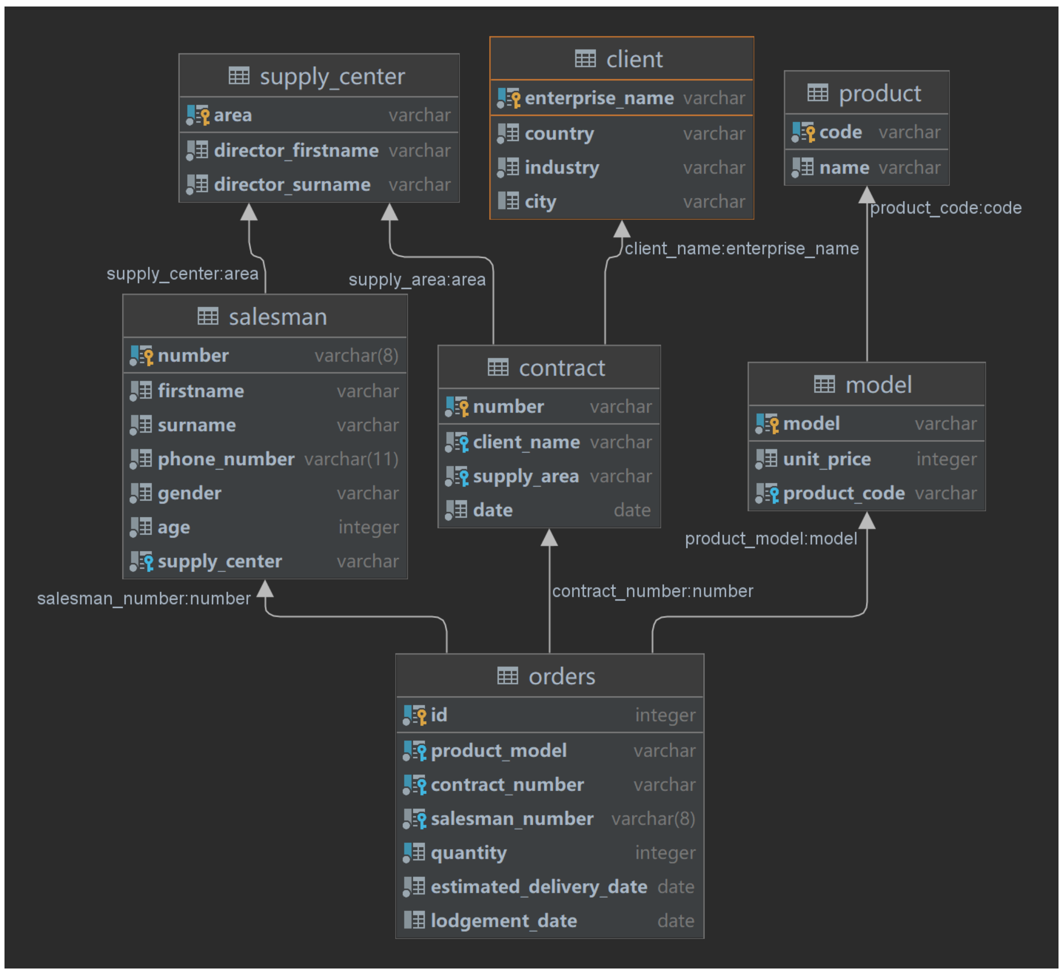
Task 1 E-R Diagram

E-R Model (draw by EDraw Max)



Task 2-------Database Design

E-R Diagram Generated by Data Grip.



Explanation for E-R model

1. There are seven Tables in total.

(1) Table client is used to store information of companies cooperating with SUSTC, it has the name of enterprise, the country and city that enterprise located. Since different enterprise have different name, enterprise_name is the primary key.

(2) Table contract is used to store information of contract. It has column contract_number, which stores the ID of the contract, client name, supply canter and the date that contract was signed. Different contract has its unique contract number, and contract_number is primary key.

(3) Table supply_center is used to store information of seven main supply center of SUSTC, It stores the information of the responsible area, and the director name, which is separated into first name and surname. From data that provides, different supply center has unique name therefore it is set to be primary key.

(4) Table orders is used to store information of the order. It has a self_increment ID as primary key. It has contract_number which refer to the contract belongs to. The column salesman_number refer to the salesman who sell this order. The column product model refers to the model of this order and column quantity refers to the amounts of the product. It also has column estimated_delivery_date and lodgement_date shows the date of delivery date and lodgement date.

(5) Table salesman is used to store information of salesman that participated in the order. It contains column salesman number (ID of salesman), phone number, first name, surname, age and gender where salesman number is the primary key of salesman.

(6) Table model, it has column model name, price and product code refer to the product belongs to. The model name is the primary key.

(7) Table product has column code and name, where product code is the primary key.

2. As we can see from the E-R model, it does not contain circular foreign-key links.

3. Normalization

For the data in csv file, we can easily find that it violates 1NF and 3NF. For 1NF, we know that one contract can have many orders which means that one contract can have many models and salesman. Therefore, we just divide it into contract table and orders table. The unity_price only depend on the model rather than contract or orders id, which violates 3NF, therefore we create a table called model. The product name also depends on product code that violates 3NF, therefore we create table product. For enterprise in contract, the industry, country and city only depend on enterprise rather than contract_number which violate 3NF. Therefore, we create table client. For salesman name, age, gender, phone_number only depend on salesman_number rather than contract_number or orders id which violates 3NF, therefore we create table salesman. For all the tables, they all have one primary key and don't have any composite primary key therefore when they follow 1NF, they must follow 2NF.

4. As we can see from the E-R model, there is no table isolated and every table has primary key.

5. Only table orders have self-increment ID column but it has (contract_number, model) as unique. Since each contract can buy two same model. Therefore (contract_number, model) it is unique.

6. All the attributes are not null expect city and lodgement_date, therefore all the tables have one mandatory ("Not Null") column.

7. It is easy to expand when the requirement change.

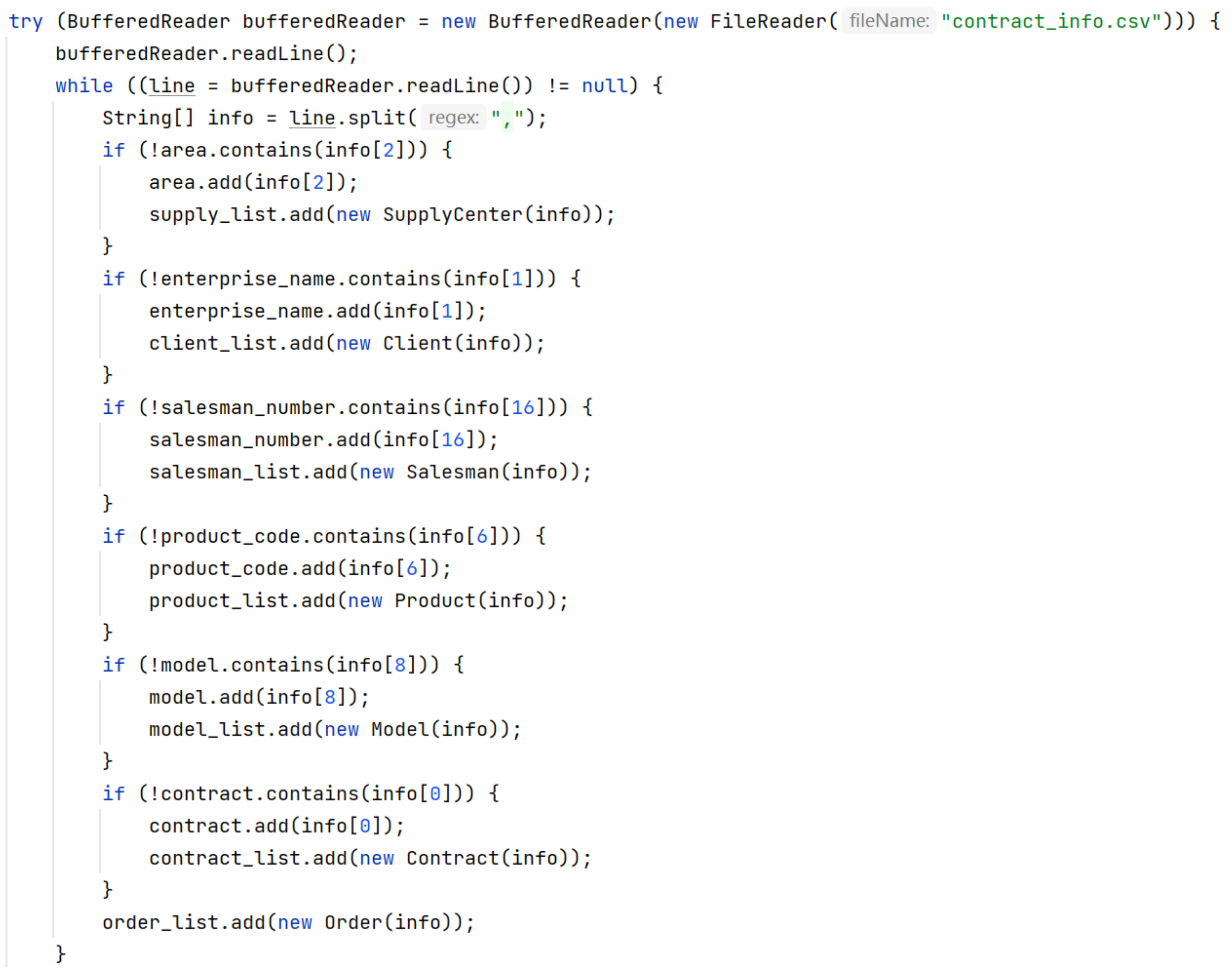
Conclusion: for the explanation above, we can know that database design meets all the demands.

Task 3: Data Import

1. Data preprocessing

The initial data is in .csv format, we cannot get the information directly. So, we need to do some preprocessing on it. Therefore, the script first split the line by "," and store the information in different class (Client, SupplyCenter, Salesman, Product, Model, Contract, Order).

Since there is many duplicated information, if we did not check them, it will create many duplicated information. Therefore, we use sets to record the primary key of each table. Only if the set does not contain the key will create an object and add it to list.

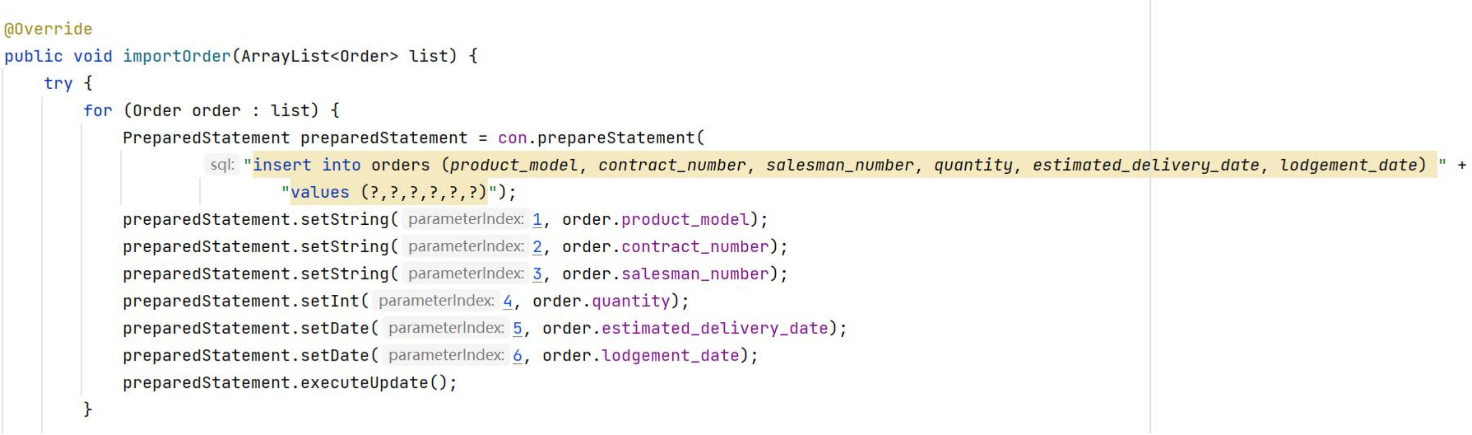


```
try (BufferedReader bufferedReader = new BufferedReader(new FileReader( fileName: "contract_info.csv"))) {
    bufferedReader.readLine();
    while ((line = bufferedReader.readLine()) != null) {
        String[] info = line.split( regex: ",");
        if (!area.contains(info[2])) {
            area.add(info[2]);
            supply_list.add(new SupplyCenter(info));
        }
        if (!enterprise_name.contains(info[1])) {
            enterprise_name.add(info[1]);
            client_list.add(new Client(info));
        }
        if (!salesman_number.contains(info[16])) {
            salesman_number.add(info[16]);
            salesman_list.add(new Salesman(info));
        }
        if (!product_code.contains(info[6])) {
            product_code.add(info[6]);
            product_list.add(new Product(info));
        }
        if (!model.contains(info[8])) {
            model.add(info[8]);
            model_list.add(new Model(info));
        }
        if (!contract.contains(info[0])) {
            contract.add(info[0]);
            contract_list.add(new Contract(info));
        }
        order_list.add(new Order(info));
    }
```

2. Import data to the database

Optimization strategy analysis

(1) Use PreparedStatement to import data



```java
@Override
public void importOrder(ArrayList<Order> list) {
    try {
        for (Order order : list) {
            PreparedStatement preparedStatement = con.prepareStatement(
                    sql: "insert into orders (product_model, contract_number, salesman_number, quantity, estimated_delivery_date, lodgement_date) " +
                            "values (?,?,?,?,?,?)");
            preparedStatement.setString( parameterIndex: 1, order.product_model);
            preparedStatement.setString( parameterIndex: 2, order.contract_number);
            preparedStatement.setString( parameterIndex: 3, order.salesman_number);
            preparedStatement.setInt( parameterIndex: 4, order.quantity);
            preparedStatement.setDate( parameterIndex: 5, order.estimated_delivery_date);
            preparedStatement.setDate( parameterIndex: 6, order.lodgement_date);
            preparedStatement.executeUpdate();
        }
```

Total import time: 12422 ms

Analysis: Using PreparedStatement can precompile the statement and reduce the running time on database.

(2) Put the declaration of the PreparedStatement outside of the loop

Total import time: 9517 ms

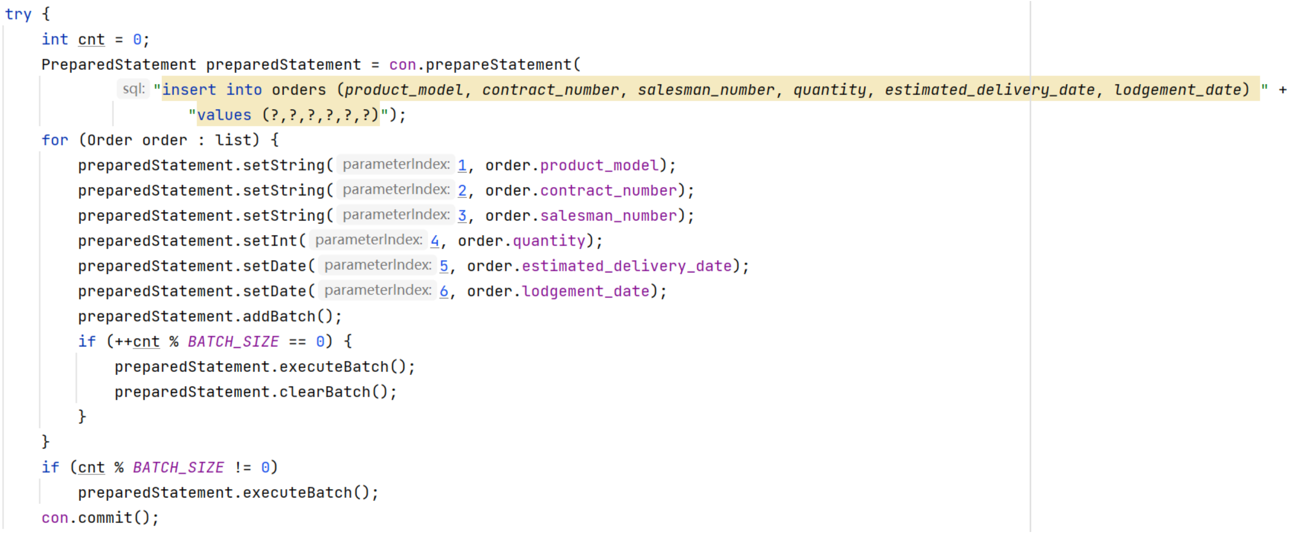
Analysis: Put the PreparedStatement outside the loop will only precompile the statement once, it will decrease the connection time with database.

(3) Execute multiple insert statements in batches

In order to use batch to insert, we need to set auto commit false:

con.setAutoCommit(false);

When the number of statements in batch is equal to BATCH_SIZE, then executeBatch.



```java
try {
    int cnt = 0;
    PreparedStatement preparedStatement = con.prepareStatement(
            sql: "insert into orders (product_model, contract_number, salesman_number, quantity, estimated_delivery_date, lodgement_date) " +
                    "values (?,?,?,?,?,?)");
    for (Order order : list) {
        preparedStatement.setString( parameterIndex: 1, order.product_model);
        preparedStatement.setString( parameterIndex: 2, order.contract_number);
        preparedStatement.setString( parameterIndex: 3, order.salesman_number);
        preparedStatement.setInt( parameterIndex: 4, order.quantity);
        preparedStatement.setDate( parameterIndex: 5, order.estimated_delivery_date);
        preparedStatement.setDate( parameterIndex: 6, order.lodgement_date);
        preparedStatement.addBatch();
        if (++cnt % BATCH_SIZE == 0) {
            preparedStatement.executeBatch();
            preparedStatement.clearBatch();
        }
    }
    if (cnt % BATCH_SIZE != 0)
        preparedStatement.executeBatch();
    con.commit();
```

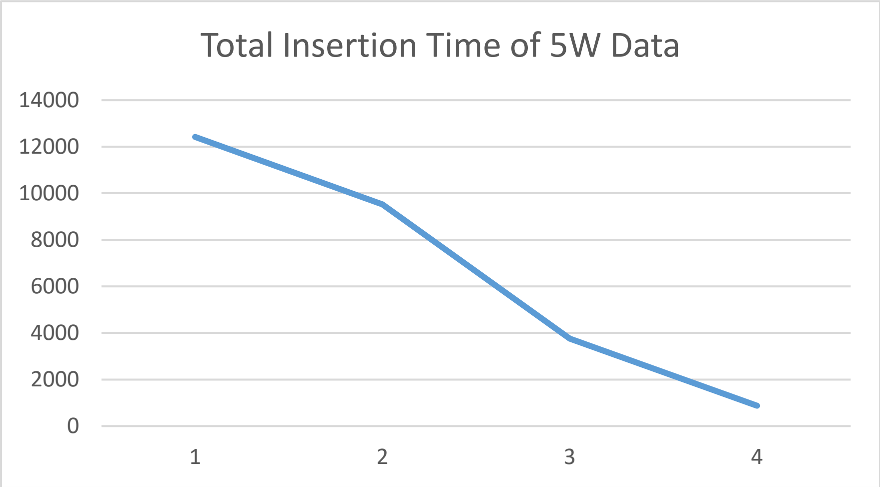
Total import time: 3751ms

Analysis: Using will execute statements at a time, reduce the time on connection with the database. Therefore, has a lower time consuming.

(4) Remove all the **constraints** and **index** before import data and restore the constraints and index after import.

Total import time: 874ms

Analysis: If we can ensure the correctness of the data, and won't have duplicated data. We can remove all the constraints and index. If the constraints exist, database would check whether it meets the requirement and it would take a large amount of time. And we can restore the constraints and index after insertion.



The final time consumption was reduced to 93% of the original.

3. The rows of each table after import

Table	client	contract	model	orders	product	salesman	supply_center
Rows	167	5000	961	50000	325	990	7

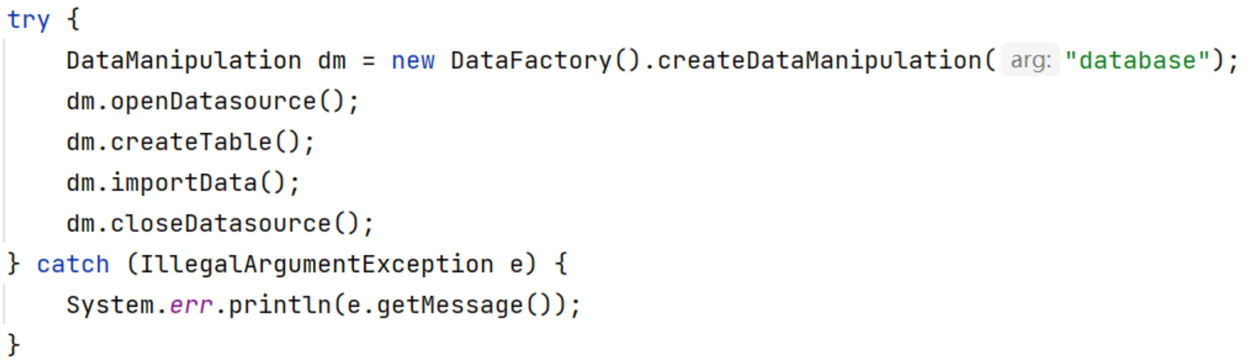
4. How to Use the script

(1) First fill in the host, database name, user, password and port of the database you want to import in DatabaseManipulation.java. For example,

```
private String host = "localhost";
private String dbname = "contract";
private String user = "checker";
private String pwd = "123456";
private String port = "5432";
```

(2) Put the **contract_info.csv** under the working directory.

(3) Execute the following statement in the main method in **Client.java**



```
try {
    DataManipulation dm = new DataFactory().createDataManipulation( arg: "database");
    dm.openDatasource();
    dm.createTable();
    dm.importData();
    dm.closeDatasource();
} catch (IllegalArgumentException e) {
    System.err.println(e.getMessage());
}
```

Task 4

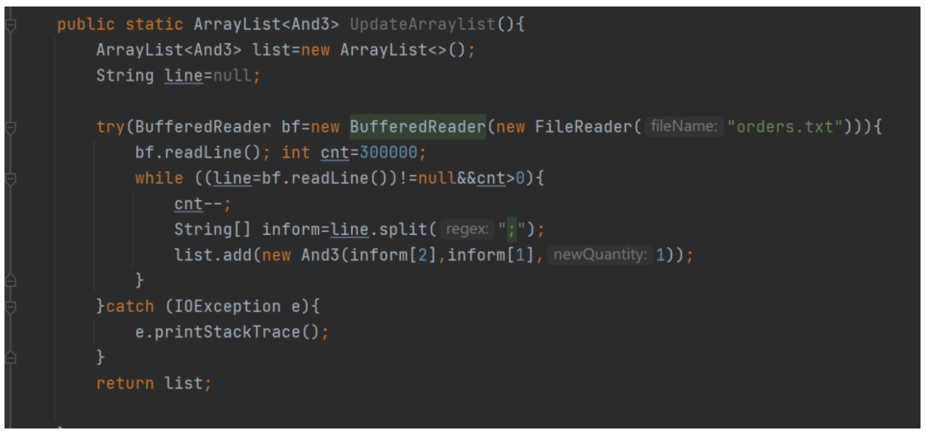
Compare DBMS with file I/0

Suggestion:

When you are going to replicate. First of all, you need to ensure that the computer is connected to the power supply, which can improve the running performance. Second, speed up your computer by not running unnecessary software. Finally, you need to conduct multiple experiments to obtain the average value, because sometimes there will be relatively large data fluctuations, so you need to conduct multiple experiments to obtain the average value.

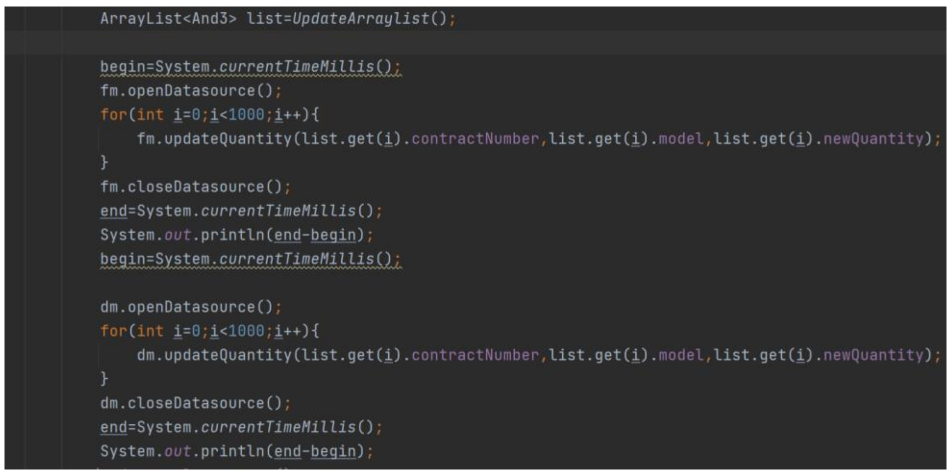
Declaration

We use System.currentTimeMillis() to count time. All the all the update operation, select operation and delete operations base on orders table which has 1,100,000 data. On how to organize test data, we just create a list which stores the corresponding information and then transfer the parameters in list to the function that we construct. And for the data that we import into orders and contract, we just replicate the information in .csv and change part of information (contract number) and keep others the same. Therefore, we use the data in csv and generate 1,050,000 data for test.



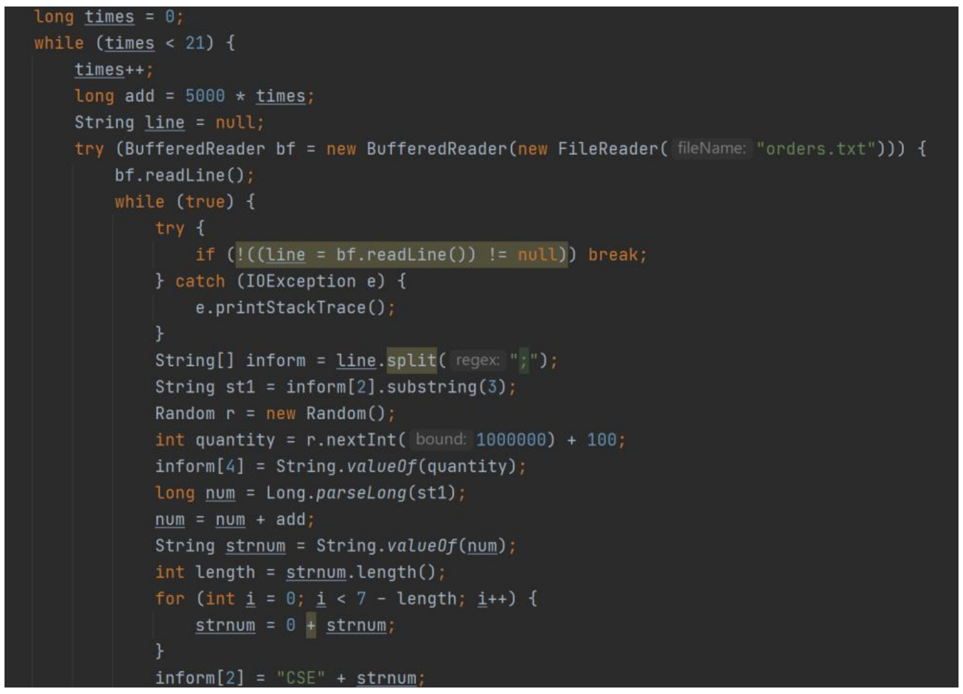
```
public static ArrayList<And3> UpdateArraylist(){
    ArrayList<And3> list=new ArrayList<>();
    String line=null;

    try(BufferedReader bf=new BufferedReader(new FileReader( fileName: "orders.txt"))){
        bf.readLine(); int cnt=300000;
        while ((line=bf.readLine())!=null&&cnt>0){
            cnt--;
            String[] inform=line.split( regex: ";");
            list.add(new And3(inform[2],inform[1], newQuantity: 1));
        }
    }catch (IOException e){
        e.printStackTrace();
    }
    return list;
```

(Generate data and store in list)



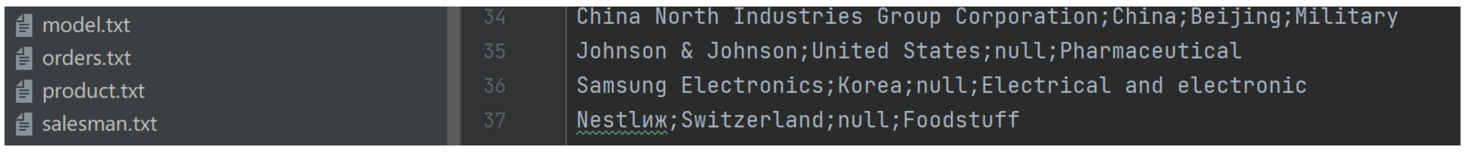
```
ArrayList<And3> list=UpdateArraylist();

begin=System.currentTimeMillis();
fm.openDatasource();
for(int i=0;i<1000;i++){
    fm.updateQuantity(list.get(i).contractNumber,list.get(i).model,list.get(i).newQuantity);
}
fm.closeDatasource();
end=System.currentTimeMillis();
System.out.println(end-begin);
begin=System.currentTimeMillis();

dm.openDatasource();
for(int i=0;i<1000;i++){
    dm.updateQuantity(list.get(i).contractNumber,list.get(i).model,list.get(i).newQuantity);
}
dm.closeDatasource();
end=System.currentTimeMillis();
System.out.println(end-begin);
```

(Transfer parameter into function)



```
long times = 0;
while (times < 21) {
    times++;
    long add = 5000 * times;
    String line = null;
    try (BufferedReader bf = new BufferedReader(new FileReader( fileName: "orders.txt"))) {
        bf.readLine();
        while (true) {
            try {
                if (!((line = bf.readLine()) != null)) break;
            } catch (IOException e) {
                e.printStackTrace();
            }
            String[] inform = line.split( regex: ";");
            String st1 = inform[2].substring(3);
            Random r = new Random();
            int quantity = r.nextInt( bound: 1000000) + 100;
            inform[4] = String.valueOf(quantity);
            long num = Long.parseLong(st1);
            num = num + add;
            String strnum = String.valueOf(num);
            int length = strnum.length();
            for (int i = 0; i < 7 - length; i++) {
                strnum = 0 + strnum;
            }
            inform[2] = "CSE" + strnum;
```

(Generate import data which has 1,050,000 data)



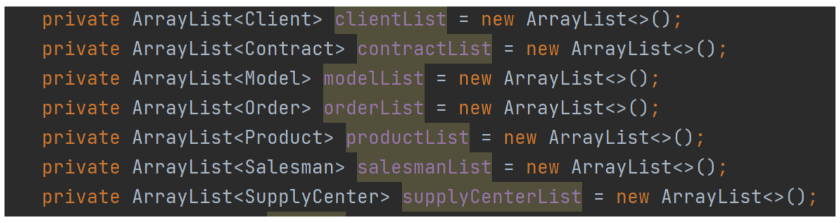
```
China North Industries Group Corporation;China;Beijing;Military
Johnson & Johnson;United States;null;Pharmaceutical
Samsung Electronics;Korea;null;Electrical and electronic
Nestlиж;Switzerland;null;Foodstuff
```

(our file use line format and data is stored in Strings)

File IO Optimize

We have found that for select, update or delete, we need to read all the file first and then do the operations and then write back to file. If there are multiple operations to be executed, the file may be read and write several times.

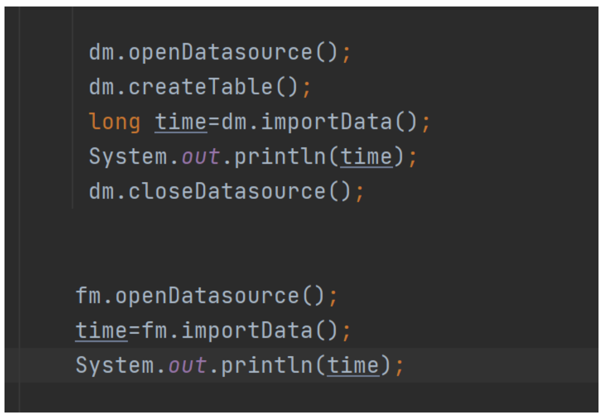
Since File IO is a time-consuming task. We optimize the program which only read all the file once when open the data source, and save the information in the memory. When handling the operations, we can directly deal with them from the memory. When close the data source or invoke the method commit(), we will write back to the file. This will greatly decrease the time consumed on FILE IO and increase the efficiency on file management.



```
    private ArrayList<Client> clientList = new ArrayList<>();
    private ArrayList<Contract> contractList = new ArrayList<>();
    private ArrayList<Model> modelList = new ArrayList<>();
    private ArrayList<Order> orderList = new ArrayList<>();
    private ArrayList<Product> productList = new ArrayList<>();
    private ArrayList<Salesman> salesmanList = new ArrayList<>();
    private ArrayList<SupplyCenter> supplyCenterList = new ArrayList<>();
```

Insert

import data from csv which has 50,000 levels data



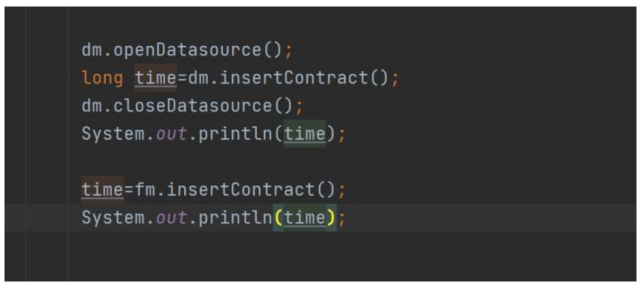
```
        dm.openDatasource();
        dm.createTable();
        long time=dm.importData();
        System.out.println(time);
        dm.closeDatasource();


       fm.openDatasource();
       time=fm.importData();
       System.out.println(time);
```

PSQL running time: 3171

File I/O running time: 297

Insert into PSQL without any constraint and foreign key running time :874

Insert 10w5k data into table contract and contract.txt

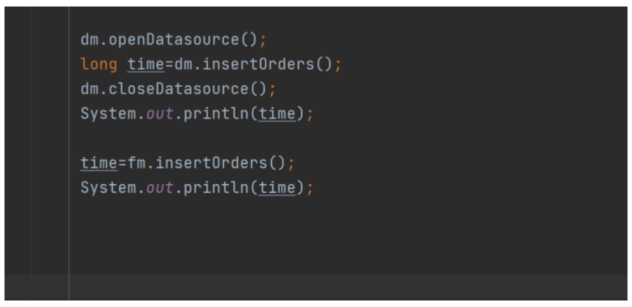


```
       dm.openDatasource();
       long time=dm.insertContract();
       dm.closeDatasource();
       System.out.println(time);

       time=fm.insertContract();
       System.out.println(time);
```

PSQL running time: 4342

File I/O running time: 156

Insert 105w data into table orders and orders.txt



```java
dm.openDatasource();
long time=dm.insertOrders();
dm.closeDatasource();
System.out.println(time);

time=fm.insertOrders();
System.out.println(time);
```

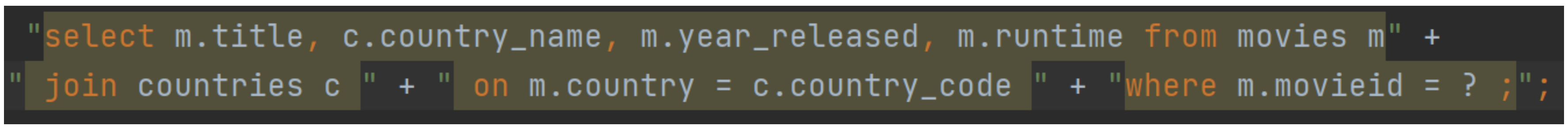
PSQL running time: 78497

File I/O running time: 3151

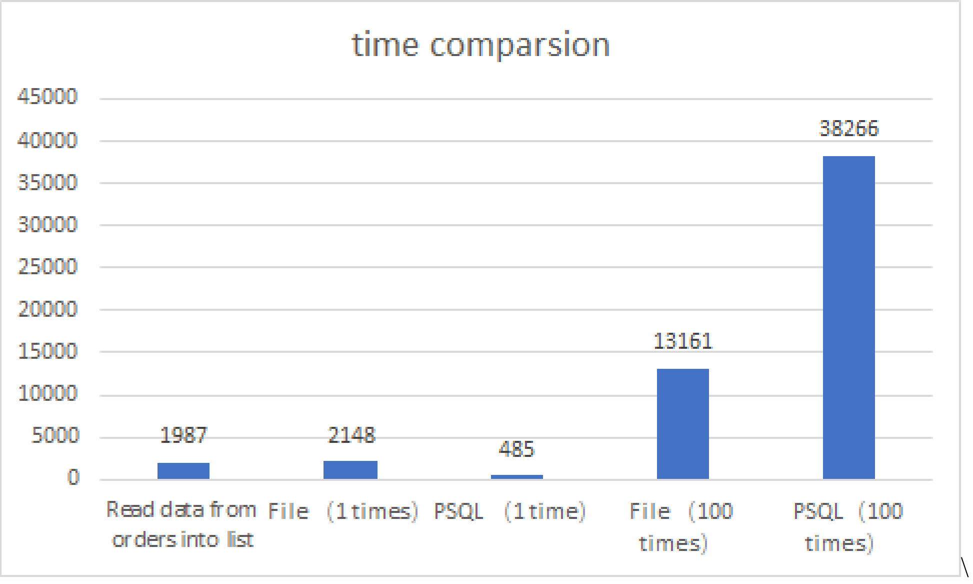
Explanation: For the data above, we clearly found that inserting data into the database is slower than a file. This might because importing the database needs to consider foreign keys and constraints, and whether the imported data is legal or not. File import does not consider these relationships and directly writes the data into txt, so the time of file I/O is less. At the same time, we also found that inserting 105,000 pieces of data into the contract is faster than importing 50,000 pieces of data into the document.

Multi-table query

Select from table orders which has 105w data

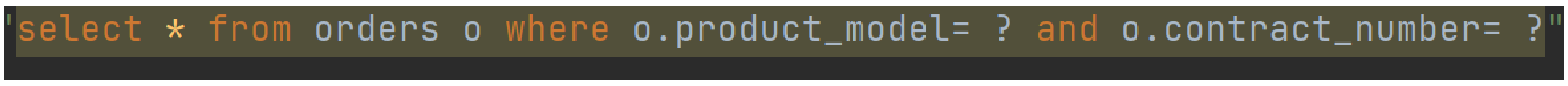


```
 "select m.title, c.country_name, m.year_released, m.runtime from movies m" +
" join countries c " + " on m.country = c.country_code " + "where m.movieid = ? ;";
```

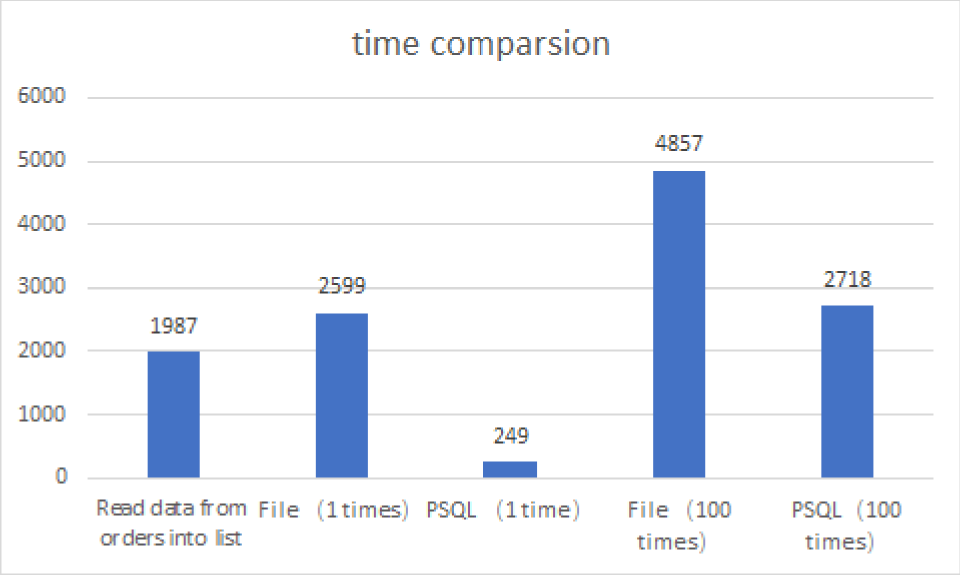


Explanation: When selecting only 1 time, the database efficiency is obviously better than the file I/O, but 100 times the database is slower. This is because our file I/O query is designed according to the actual content, and the file I/O query is optimized. When performing 100 select operations, we just need read data from txt file into list once. and the time consumed by the file read once is 1987

Single table query

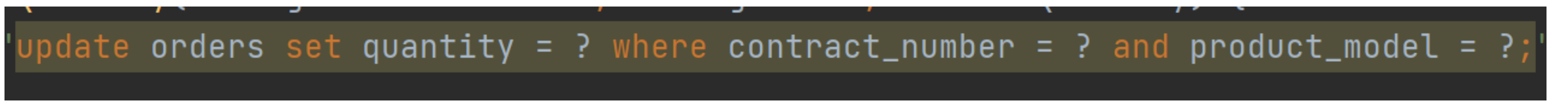


```
"select * from orders o where o.product_model= ? and o.contract_number= ?"
```

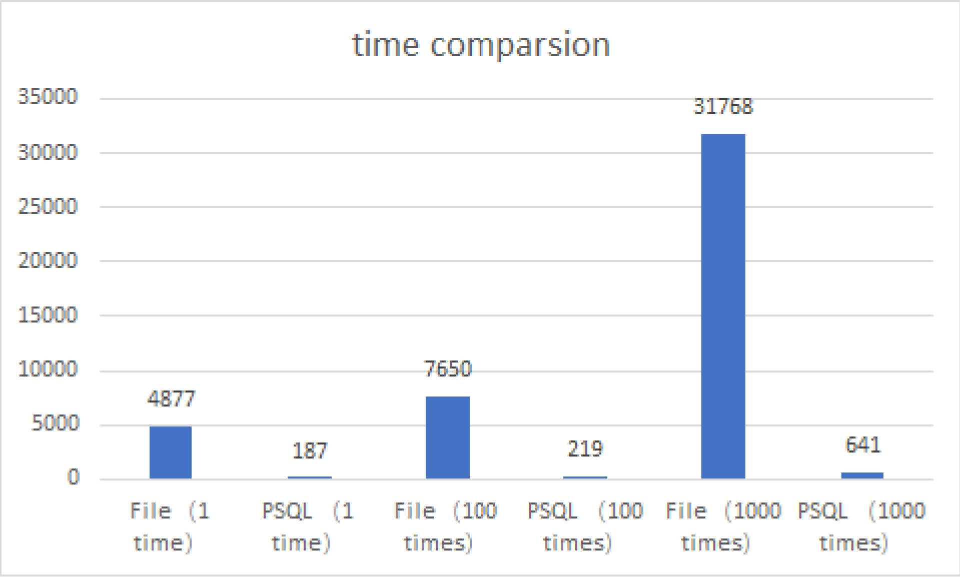


Explanation: Database single table query efficiency is better than file. The reason why file I/O grows slower than database is that we just only read data from txt file into list once

Update

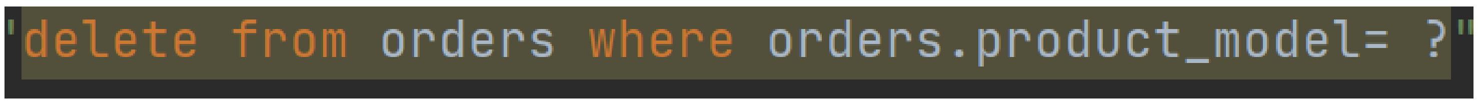


```
"update orders set quantity = ? where contract_number = ? and product_model = ?;"
```

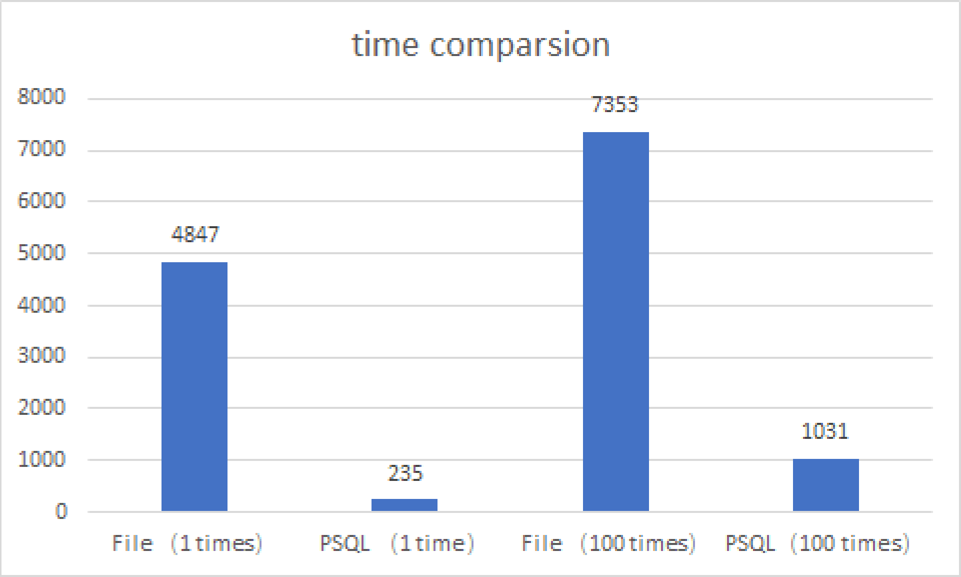


Explanation: As we can see from figure, when we do update operation, database is much faster than file I/O. This is because when we do update operation in File, we need to read the data in txt file and store them in list and traverse the list to update data which we want to change, after that we need write the data stored in list into txt file.

Delete



```
"delete from orders where orders.product_model= ?"
```



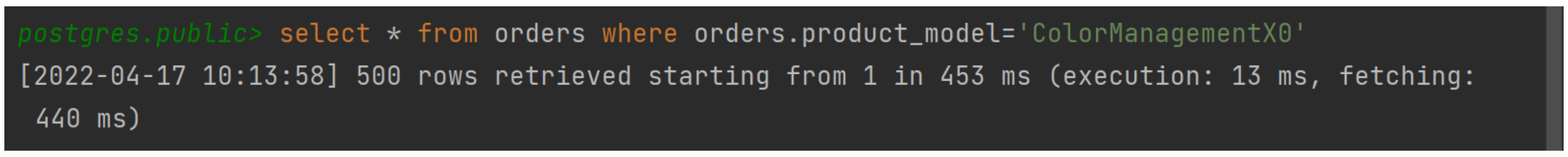
Explanation: As we can see from figure, when we do delete operation, database is faster than file I/O. This is because when we do delete operation in File, we need to read the data in txt file and store them in list and traverse the list to delete data which we want to delete, after that we need write the data stored in list into txt file.

Database Index

Select

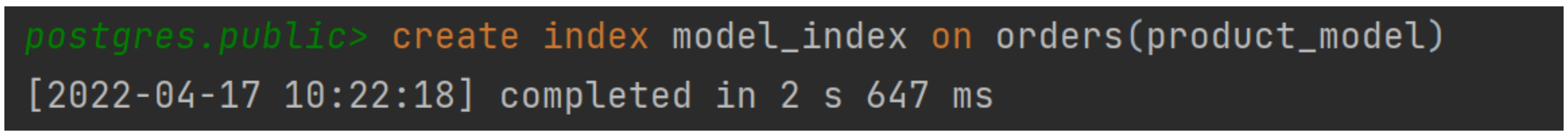
Table: orders which has 1100,000 data

Select directly form orders which product model is ColorManagementX0 which uses 453ms



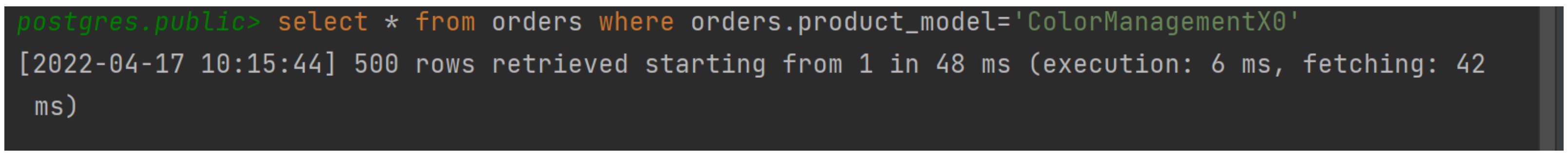
```
postgres.public> select * from orders where orders.product_model='ColorManagementX0'
[2022-04-17 10:13:58] 500 rows retrieved starting from 1 in 453 ms (execution: 13 ms, fetching:
 440 ms)
```

Add the model_index in orders(product_model) which uses 2647ms



```
postgres.public> create index model_index on orders(product_model)
[2022-04-17 10:22:18] completed in 2 s 647 ms
```

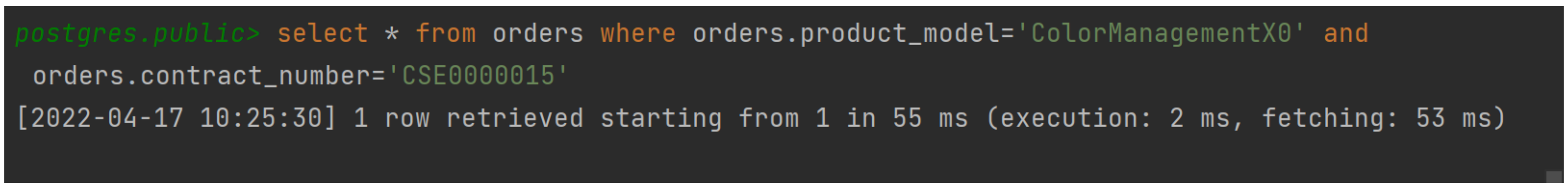
Select again with index which uses 48ms



```
postgres.public> select * from orders where orders.product_model='ColorManagementX0'
[2022-04-17 10:15:44] 500 rows retrieved starting from 1 in 48 ms (execution: 6 ms, fetching: 42
 ms)
```

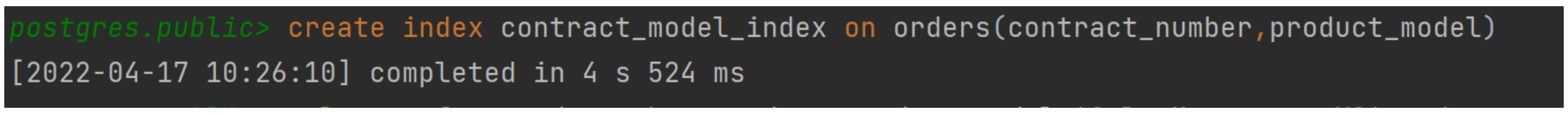
Compare with no index which increase by 843%

Select directly from orders which product model is ColorManagementX0 and contract_number is CSE0000015 without index which uses 55ms



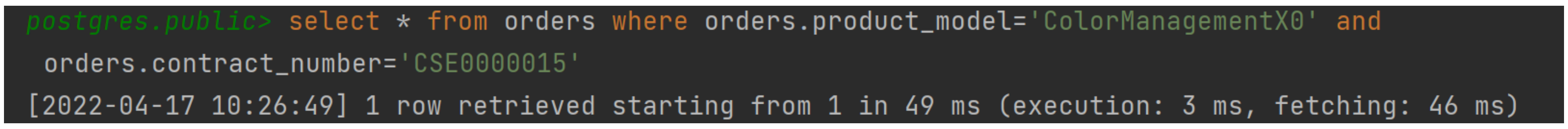
```
postgres.public> select * from orders where orders.product_model='ColorManagementX0' and
 orders.contract_number='CSE0000015'
[2022-04-17 10:25:30] 1 row retrieved starting from 1 in 55 ms (execution: 2 ms, fetching: 53 ms)
```

Add the contract_model_index on orders(contract_number,product_model) which uses 4524ms



```
postgres.public> create index contract_model_index on orders(contract_number,product_model)
[2022-04-17 10:26:10] completed in 4 s 524 ms
```

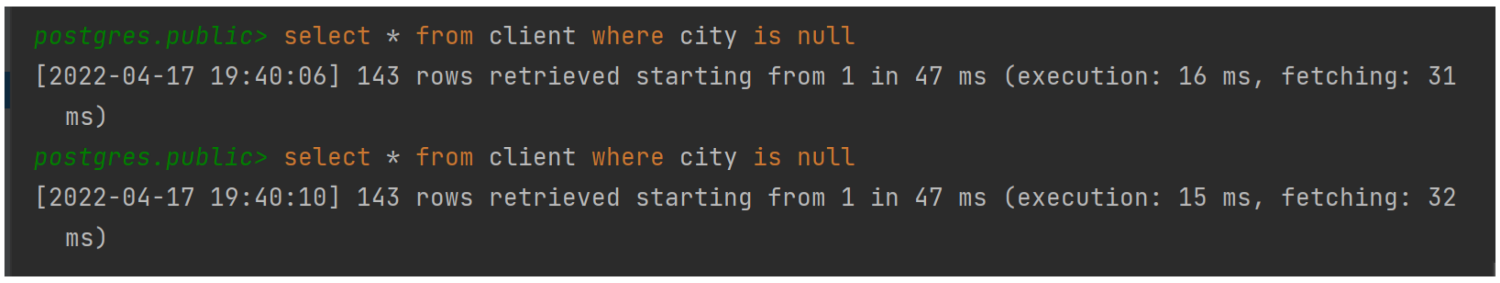
Select from orders which product model is ColorManagementX0 and contract_number is CSE0000015 with index which uses 49ms



```
postgres.public> select * from orders where orders.product_model='ColorManagementX0' and
 orders.contract_number='CSE0000015'
[2022-04-17 10:26:49] 1 row retrieved starting from 1 in 49 ms (execution: 3 ms, fetching: 46 ms)
```

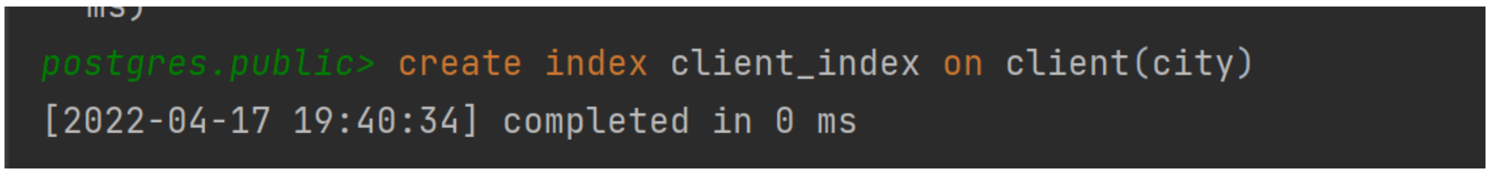
Compare with no index which increase by 12%

Select company with empty city in client without index which uses 47 ms



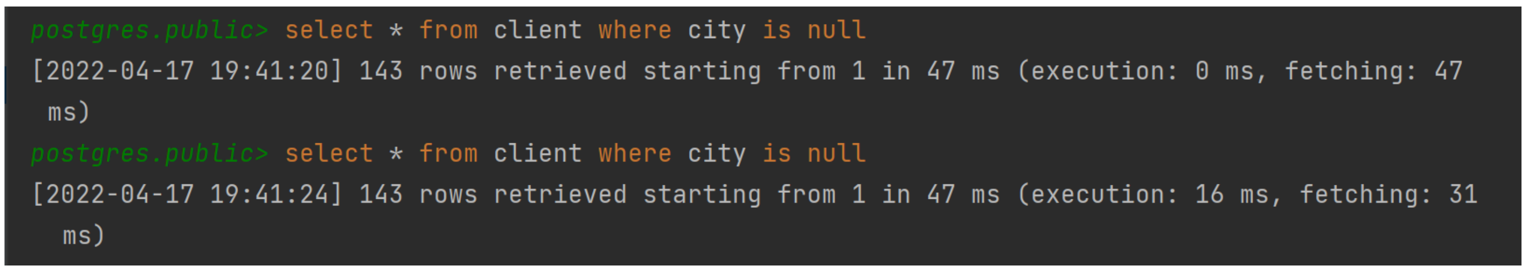
```
postgres.public> select * from client where city is null
[2022-04-17 19:40:06] 143 rows retrieved starting from 1 in 47 ms (execution: 16 ms, fetching: 31
  ms)
postgres.public> select * from client where city is null
[2022-04-17 19:40:10] 143 rows retrieved starting from 1 in 47 ms (execution: 15 ms, fetching: 32
  ms)
```

Create index on city which uses 0ms



```
  ms)
postgres.public> create index client_index on client(city)
[2022-04-17 19:40:34] completed in 0 ms
```

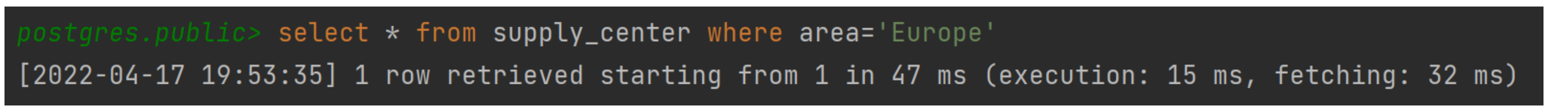
Do the same operation with index which uses 47ms



```
postgres.public> select * from client where city is null
[2022-04-17 19:41:20] 143 rows retrieved starting from 1 in 47 ms (execution: 0 ms, fetching: 47
  ms)
postgres.public> select * from client where city is null
[2022-04-17 19:41:24] 143 rows retrieved starting from 1 in 47 ms (execution: 16 ms, fetching: 31
  ms)
```

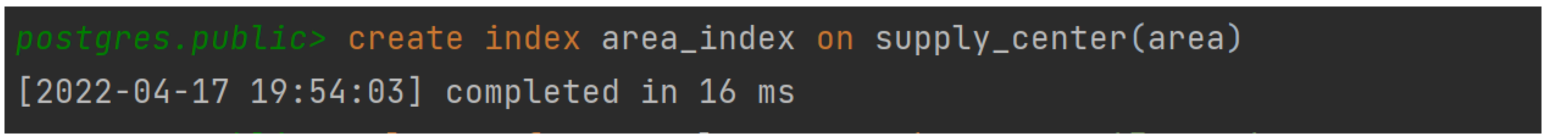
No performance improvement

Select supply_center in Europe which uses 47ms



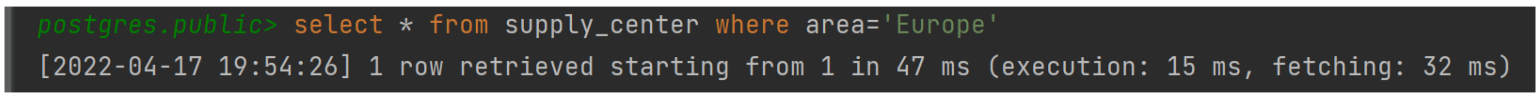
```
postgres.public> select * from supply_center where area='Europe'
[2022-04-17 19:53:35] 1 row retrieved starting from 1 in 47 ms (execution: 15 ms, fetching: 32 ms)
```

Create index on area which uses 16ms



```
postgres.public> create index area_index on supply_center(area)
[2022-04-17 19:54:03] completed in 16 ms
```

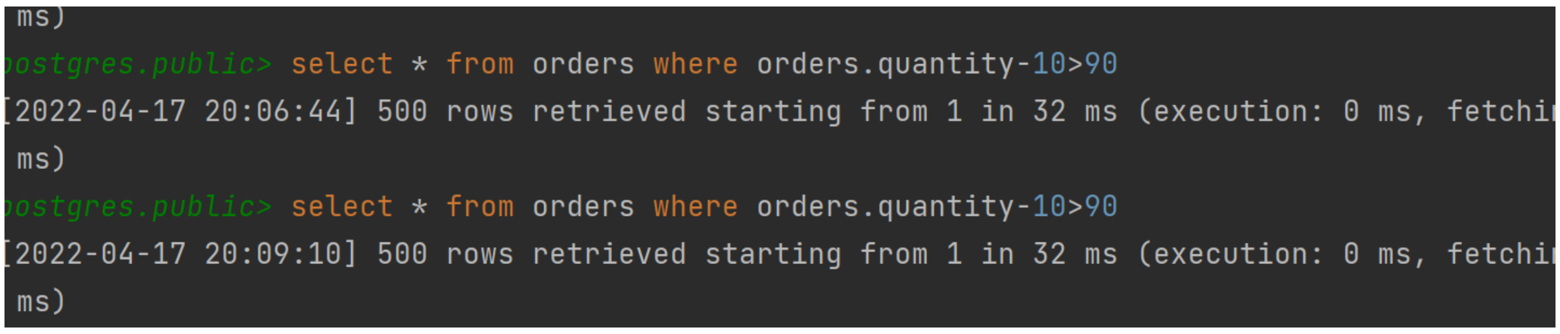
Do the same operation with index which uses 47ms



```
postgres.public> select * from supply_center where area='Europe'
[2022-04-17 19:54:26] 1 row retrieved starting from 1 in 47 ms (execution: 15 ms, fetching: 32 ms)
```

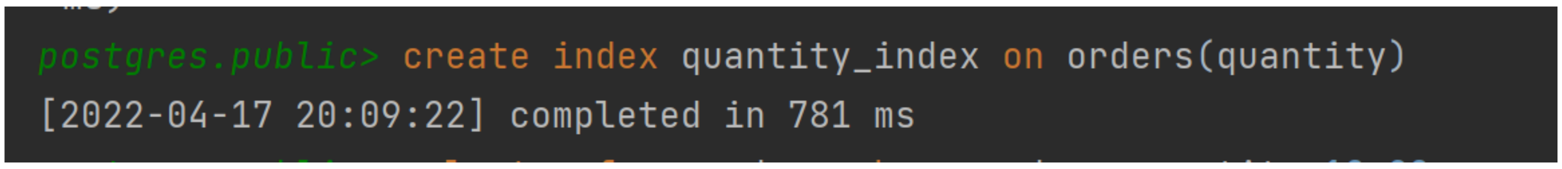
No performance improvement

Select orders with quantity greater than 100 which uses 32ms



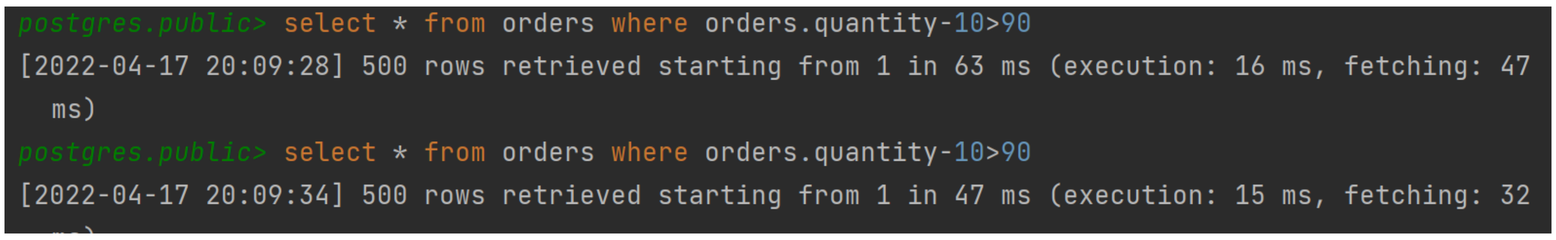
```
 ms)
ostgres.public> select * from orders where orders.quantity-10>90
[2022-04-17 20:06:44] 500 rows retrieved starting from 1 in 32 ms (execution: 0 ms, fetchi
 ms)
ostgres.public> select * from orders where orders.quantity-10>90
[2022-04-17 20:09:10] 500 rows retrieved starting from 1 in 32 ms (execution: 0 ms, fetchi
 ms)
```

Create index on quantity which uses 781ms



```
postgres.public> create index quantity_index on orders(quantity)
[2022-04-17 20:09:22] completed in 781 ms
```

Do the same operation with index which uses 47ms

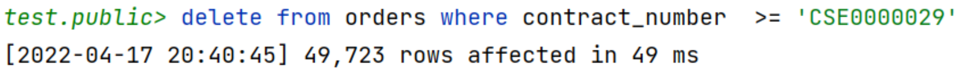


```
postgres.public> select * from orders where orders.quantity-10>90
[2022-04-17 20:09:28] 500 rows retrieved starting from 1 in 63 ms (execution: 16 ms, fetching: 47
  ms)
postgres.public> select * from orders where orders.quantity-10>90
[2022-04-17 20:09:34] 500 rows retrieved starting from 1 in 47 ms (execution: 15 ms, fetching: 32
```

No performance improvement

Delete

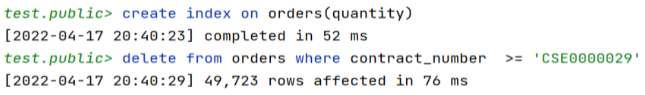
Delete orders with orders contract_number greater or equal to ‘CSE0000029’ which uses 49ms



```
test.public> delete from orders where contract_number  >= 'CSE0000029'
[2022-04-17 20:40:45] 49,723 rows affected in 49 ms
```

Create index on quantity which uses 52ms

Do the same operation with index which uses 76ms



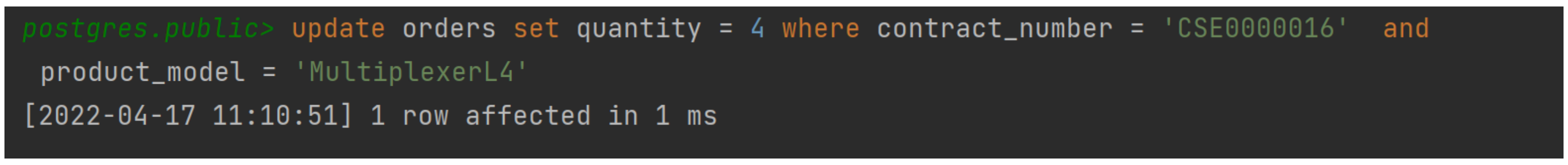
```
test.public> create index on orders(quantity)
[2022-04-17 20:40:23] completed in 52 ms
test.public> delete from orders where contract_number  >= 'CSE0000029'
[2022-04-17 20:40:29] 49,723 rows affected in 76 ms
```

Decrease by 55%

Update

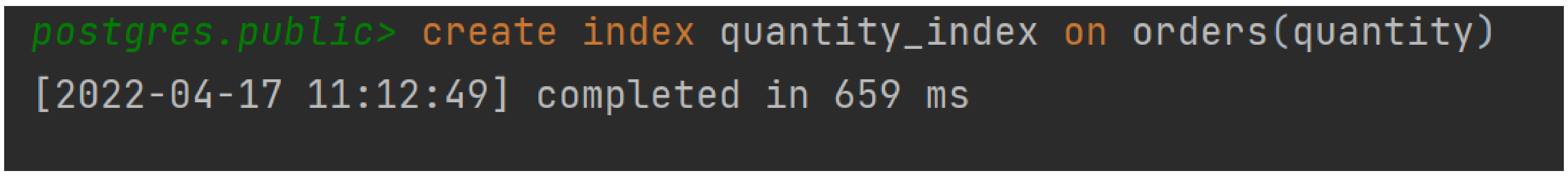
Table: orders which has 1100,000 data

Update orders’ quantity into 4 whose product model is MultiplexerL4 and contract_number is CSE0000016 without index



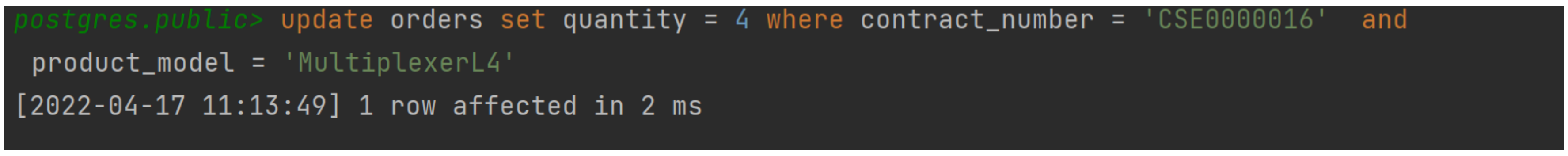
```
postgres.public> update orders set quantity = 4 where contract_number = 'CSE0000016'  and
 product_model = 'MultiplexerL4'
[2022-04-17 11:10:51] 1 row affected in 1 ms
```

Add index quantity in table orders which uses 659ms



```
postgres.public> create index quantity_index on orders(quantity)
[2022-04-17 11:12:49] completed in 659 ms
```

Update orders’ quantity into 4 whose product model is MultiplexerL4 and contract_number is CSE0000016with index which uses 2ms.



```
postgres.public> update orders set quantity = 4 where contract_number = 'CSE0000016'  and
 product_model = 'MultiplexerL4'
[2022-04-17 11:13:49] 1 row affected in 2 ms
```

Compare with no index decrease by 100%

Conclusion:

(1) Index can speed up selection operation

(2) Index will not work if index contains empty column

(3) The effect of index is not obvious if the table is small

(4) In fact, the nature of index is to build B+ tree. Therefore, when we update or delete data from table or insert data into table , it need to maintain B+ tree, so it will lower speed.

User privileges management:

In this part, we create three types of users: salesman, manager and admin.

salesman privilege:

	client	contract	model	orders	product	salesman	supply_center
Select	T	F	T	F	T	T	T
Insert	F	F	F	F	F	T	F
Update	F	F	F	F	F	T	F
Delete	F	F	F	F	F	T	F

Salesman can’t have access to trading secrets. Therefore, they do not have the privilege to select in contract and orders. And they can only select in client, model, product, supply_center and cannot insert, update or delete. They only have full access on the table salesman.

manager privilege:

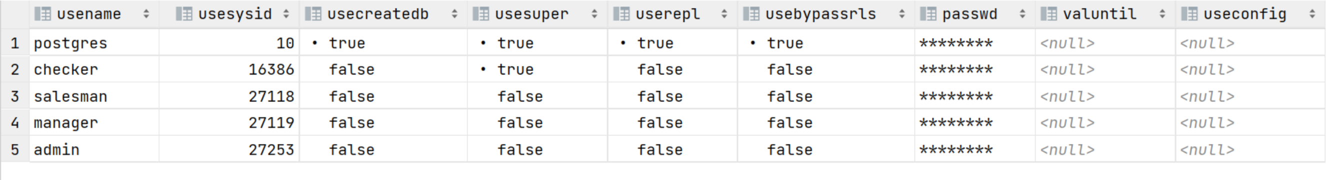
	client	contract	model	orders	product	salesman	supply_center
Select	T	T	T	T	T	T	T
Insert	T	T	T	T	T	T	F
Update	T	T	T	T	T	T	T
Delete	T	T	T	T	T	T	F

Manager should responsible to transactions. Therefore, they had full access to client, contract, model, orders, product and salesman. Adding or removing supply Center requires a higher-level decision. They can only select or update on supply center.

admin privilege:

admin have can select, insert, update and delete in all the table. Besides, it can create other users.

Besides, all three kinds of user cannot access on other database, and cannot modify the structure of the current database.



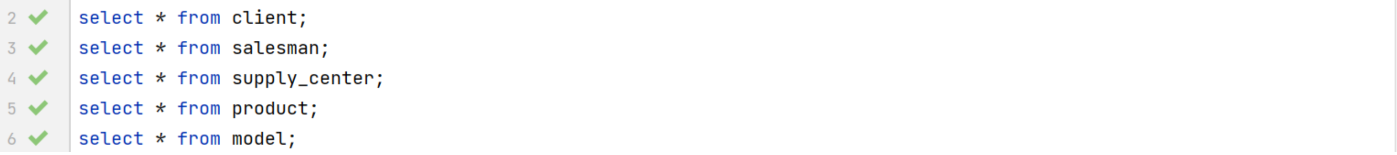
	usename	usesysid	usecreatedb	usesuper	userepl	usebypassrls	passwd	valuntil	useconfig
1	postgres	10	• true	• true	• true	• true	********	<null>	<null>
2	checker	16386	false	• true	false	false	********	<null>	<null>
3	salesman	27118	false	false	false	false	********	<null>	<null>
4	manager	27119	false	false	false	false	********	<null>	<null>
5	admin	27253	false	false	false	false	********	<null>	<null>

Privilege Test:

[salesman]

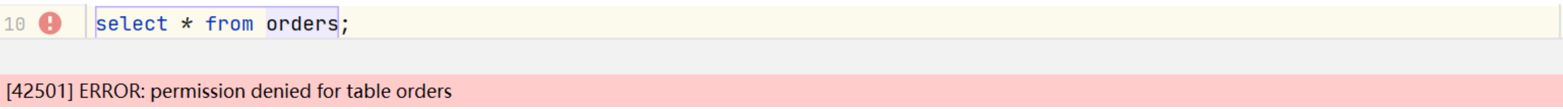
Log in with user salesman

Can select on client, salesman, supply_center, product and model.



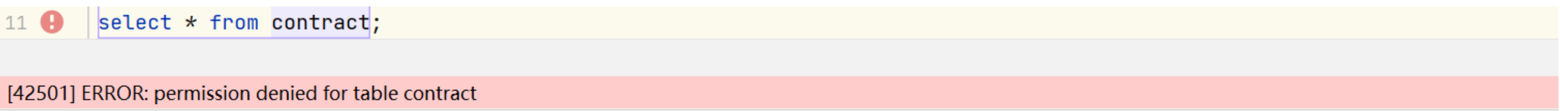
```sql
select * from client;
select * from salesman;
select * from supply_center;
select * from product;
select * from model;
```

Can not access on orders and contract.



```sql
select * from orders;
```

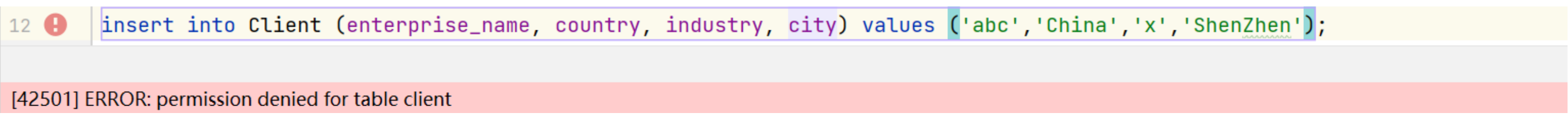
[42501] ERROR: permission denied for table orders



```sql
select * from contract;
```

[42501] ERROR: permission denied for table contract

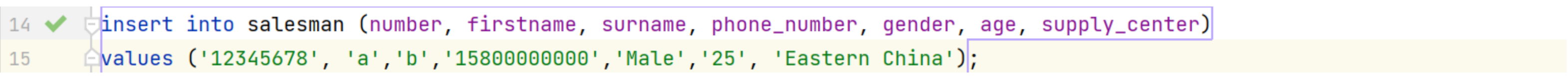
Can not insert, update or delete on table except salesman for example:



```sql
insert into Client (enterprise_name, country, industry, city) values ('abc','China','x','ShenZhen');
```

[42501] ERROR: permission denied for table client

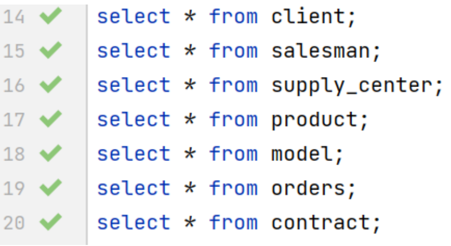
Can insert, update, delete on salesman:



```sql
insert into salesman (number, firstname, surname, phone_number, gender, age, supply_center)
values ('12345678', 'a','b','15800000000','Male','25', 'Eastern China');
```

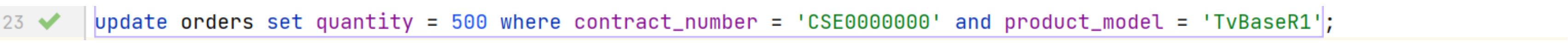
[manager]

Can select on all tables



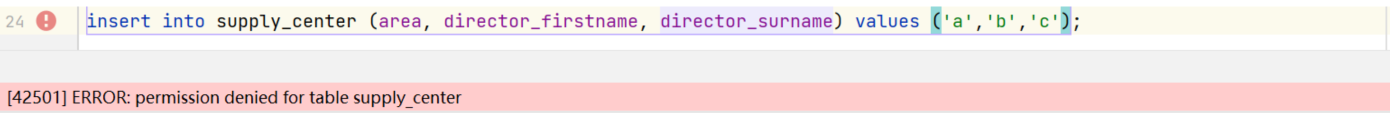
```sql
select * from client;
select * from salesman;
select * from supply_center;
select * from product;
select * from model;
select * from orders;
select * from contract;
```

Can insert, update or delete on client, contract, model, orders, product, salesman. For example:



```sql
update orders set quantity = 500 where contract_number = 'CSE0000000' and product_model = 'TvBaseR1';
```

Can not insert or delete on suppy_center.

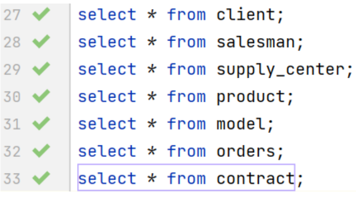


```sql
insert into supply_center (area, director_firstname, director_surname) values ('a','b','c');
```

[42501] ERROR: permission denied for table supply_center

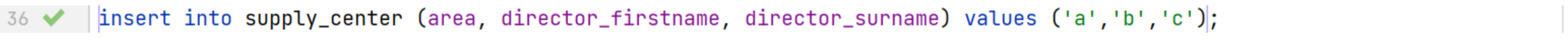
[admin]

Can select on all tables



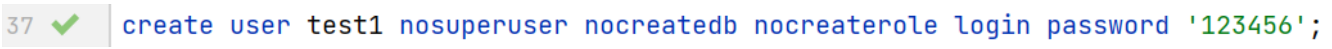
```sql
select * from client;
select * from salesman;
select * from supply_center;
select * from product;
select * from model;
select * from orders;
select * from contract;
```

Can insert, update or delete on all tables. For example:



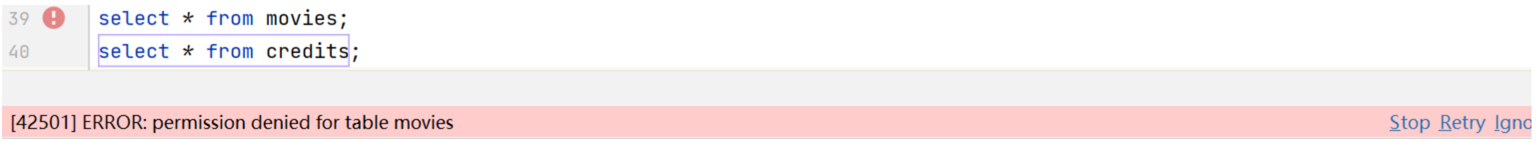
```sql
insert into supply_center (area, director_firstname, director_surname) values ('a','b','c');
```

Can create new user



```
create user test1 nosuperuser nocreatedb nocreaterole login password '123456';
```

Can not access other database.



```
select * from movies;
select * from credits;
```

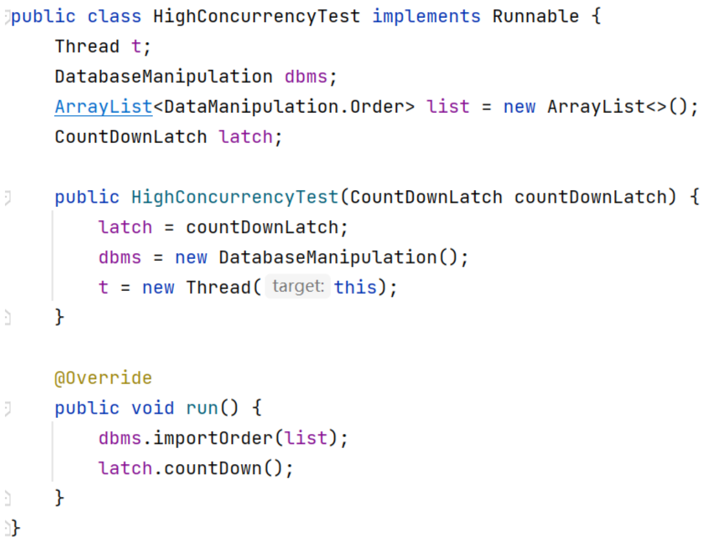
[42501] ERROR: permission denied for table movies

High Concurrency

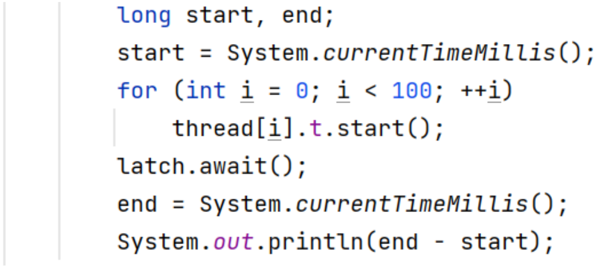
In this project, we test for the high concurrency insertion.

Initial, there are 5W orders in the database. And then we allocate 80W orders to 100 threads and do the insertion at almost the same time.

Each thread needs to insert 8000 orders.



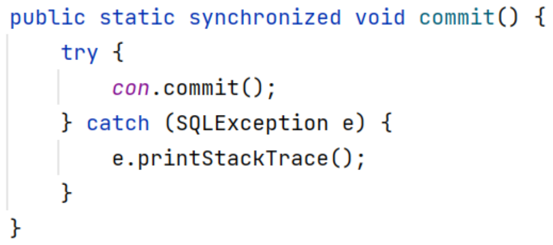
```
public class HighConcurrencyTest implements Runnable {
    Thread t;
    DatabaseManipulation dbms;
    ArrayList<DataManipulation.Order> list = new ArrayList<>();
    CountDownLatch latch;

    public HighConcurrencyTest(CountDownLatch countDownLatch) {
        latch = countDownLatch;
        dbms = new DatabaseManipulation();
        t = new Thread( target: this);
    }

    @Override
    public void run() {
        dbms.importOrder(list);
        latch.countDown();
    }
}
```



```
long start, end;
start = System.currentTimeMillis();
for (int i = 0; i < 100; ++i)
    thread[i].t.start();
latch.await();
end = System.currentTimeMillis();
System.out.println(end - start);
```

At first, after all the insertion, some thread throws exceptions and there are only **93%** data have inserted.

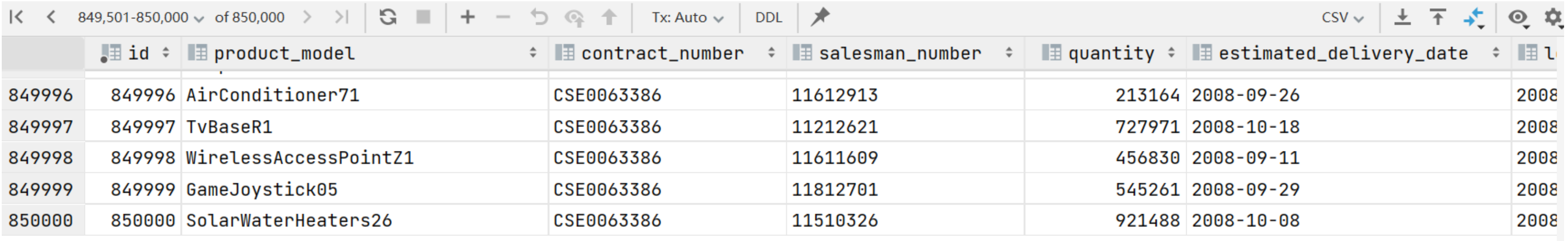
This might because several thread use the same connection with the database. Therefore, we can make commit method synchronized to avoid thread collisions.



```
public static synchronized void commit() {
    try {
        con.commit();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

After that, the insertion can proceed normally.

It takes only 21906 ms to insert 80W orders. And does not miss any data.



	id	product_model	contract_number	salesman_number	quantity	estimated_delivery_date	l
849996	849996	AirConditioner71	CSE0063386	11612913	213164	2008-09-26	2008
849997	849997	TvBaseR1	CSE0063386	11212621	727971	2008-10-18	2008
849998	849998	WirelessAccessPointZ1	CSE0063386	11611609	456830	2008-09-11	2008
849999	849999	GameJoystick05	CSE0063386	11812701	545261	2008-09-29	2008
850000	850000	SolarWaterHeaters26	CSE0063386	11510326	921488	2008-10-08	2008

Transcation Management

When the amount of concurrency is large, it is easy to have multiple transactions running at the same time. They both does not know the existence of the other transaction. If they does not protect the data manipulating, may cause some problem.

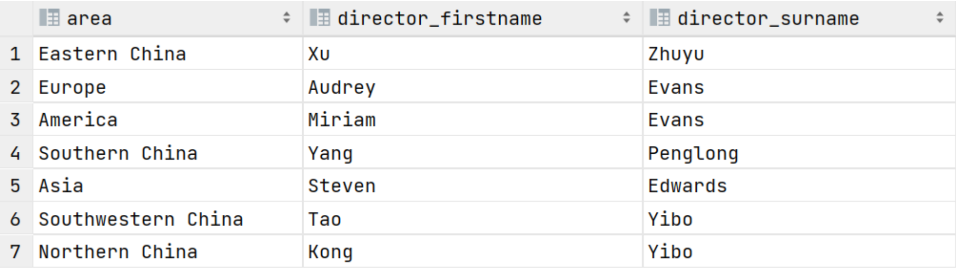
In this project, we test for some cases which may apear in transcation.

We open two terminals connect with the same database.

1. unrepeatable read

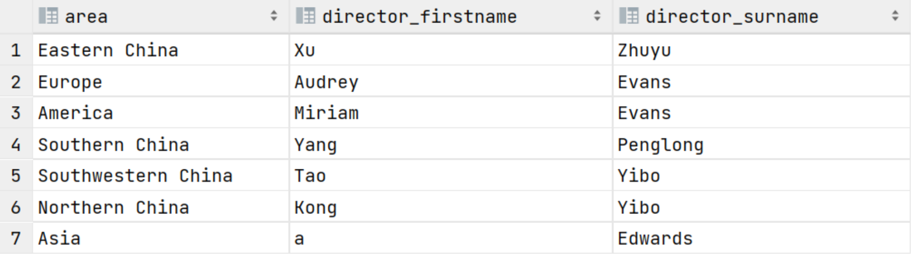
Step	Terminal1	Terminal 2
1	begin	begin
2	select * from supply_center	update supply_center set director_firstname = 'a' where area = 'Asia'
3		commit
4	select * from supply_center	

The selection in step 2:



	area	director_firstname	director_surname
1	Eastern China	Xu	Zhuyu
2	Europe	Audrey	Evans
3	America	Miriam	Evans
4	Southern China	Yang	Penglong
5	Asia	Steven	Edwards
6	Southwestern China	Tao	Yibo
7	Northern China	Kong	Yibo

The selection in step 4:



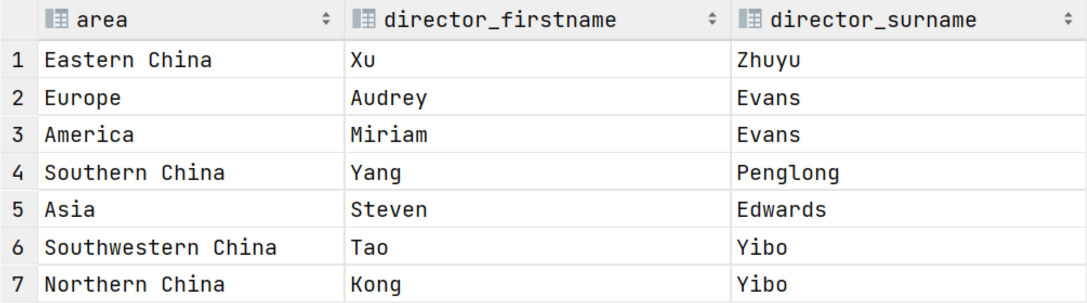
	area	director_firstname	director_surname
1	Eastern China	Xu	Zhuyu
2	Europe	Audrey	Evans
3	America	Miriam	Evans
4	Southern China	Yang	Penglong
5	Southwestern China	Tao	Yibo
6	Northern China	Kong	Yibo
7	Asia	a	Edwards

This can be solved by setting the isolation to REPEATABLE READ.

(adding “set session default_transaction_isolation = 'repeatable read';” before starting the transaction)

2. phantom problem

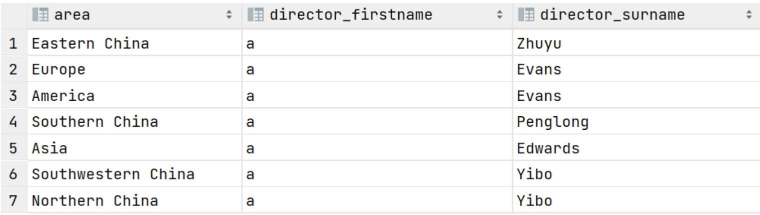
Initial table:



	area	director_firstname	director_surname
1	Eastern China	Xu	Zhuyu
2	Europe	Audrey	Evans
3	America	Miriam	Evans
4	Southern China	Yang	Penglong
5	Asia	Steven	Edwards
6	Southwestern China	Tao	Yibo
7	Northern China	Kong	Yibo

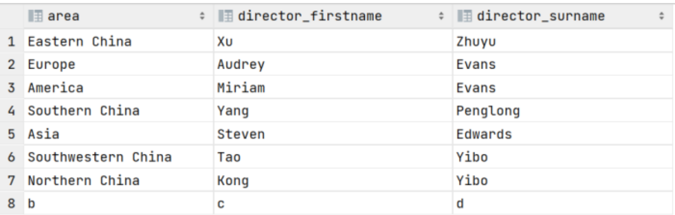
Step	Terminal1	Terminal 2
1	set session default_transaction_isolation = 'repeatable read'	set session default_transaction_isolation = 'repeatable read'
2	begin	begin
3	update supply_center set director_firstname = 'a'	insert into supply_center (area, director_firstname, director_surname) values ('b', 'c', 'd')
4	select * from supply_center;	select * from supply_center;
5	commit	commit

Step 3 in terminal 1:



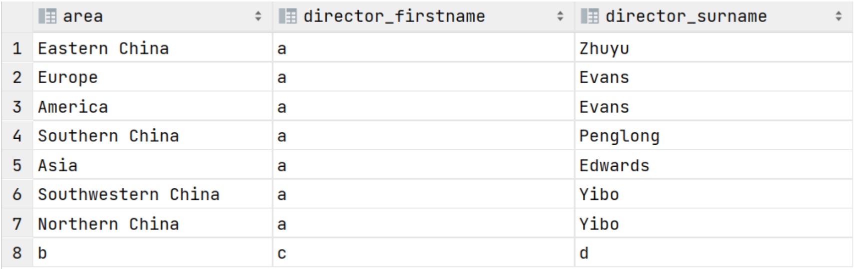
	area	director_firstname	director_surname
1	Eastern China	a	Zhuyu
2	Europe	a	Evans
3	America	a	Evans
4	Southern China	a	Penglong
5	Asia	a	Edwards
6	Southwestern China	a	Yibo
7	Northern China	a	Yibo

Step 3 in terminal 2:



	area	director_firstname	director_surname
1	Eastern China	Xu	Zhuyu
2	Europe	Audrey	Evans
3	America	Miriam	Evans
4	Southern China	Yang	Penglong
5	Asia	Steven	Edwards
6	Southwestern China	Tao	Yibo
7	Northern China	Kong	Yibo
8	b	c	d

After commiting:



	area	director_firstname	director_surname
1	Eastern China	a	Zhuyu
2	Europe	a	Evans
3	America	a	Evans
4	Southern China	a	Penglong
5	Asia	a	Edwards
6	Southwestern China	a	Yibo
7	Northern China	a	Yibo
8	b	c	d

Then, as if in an illusion, the user operating on the first transaction will discover that there are unmodified rows in the table.

This can solved by setting the isolation to Serializable. This may avoid all mistakes. But it may be low efficieny in high concurrency.

According to some sources, it can use Multiversion Concurrency Control to solve the problem:

1. Each piece of data adds two hidden columns (create and delete) so that each transaction starts with an incremented version
2. (1) insert: insert the data directly

   (2) delete: delete data and update delete_version to current to current transaction version.

   (3) update: first delete, then insert to update

   (4) select: To avoid selecting old data or data that has been changed by other transactions, it need to follow the constraints:

   a. The version number of the current transaction must be greater than or equal to the creation version number

   b. The version number of the current transaction must be smaller than the deleted version.

Using rules mentioned above, MCC can deal with these problems effectively.