# Report of Project #1

Lab class 1

Reporter: 12012514李文锦，12013016焦傲

Contributions:

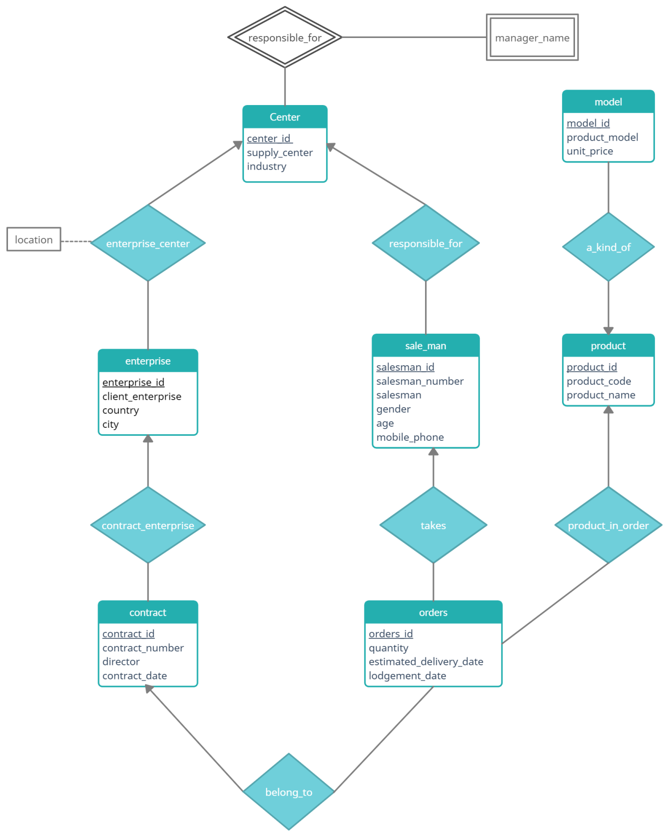
- Part one and part two: 李文锦
- Part three: 焦傲
- Part four: 焦傲，李文锦
- the percentages of contributions: 李文锦: 50%, 焦傲: 50%

## Part One: E-R Diagram

**Online service used to draw this E-R Diagram:** creately

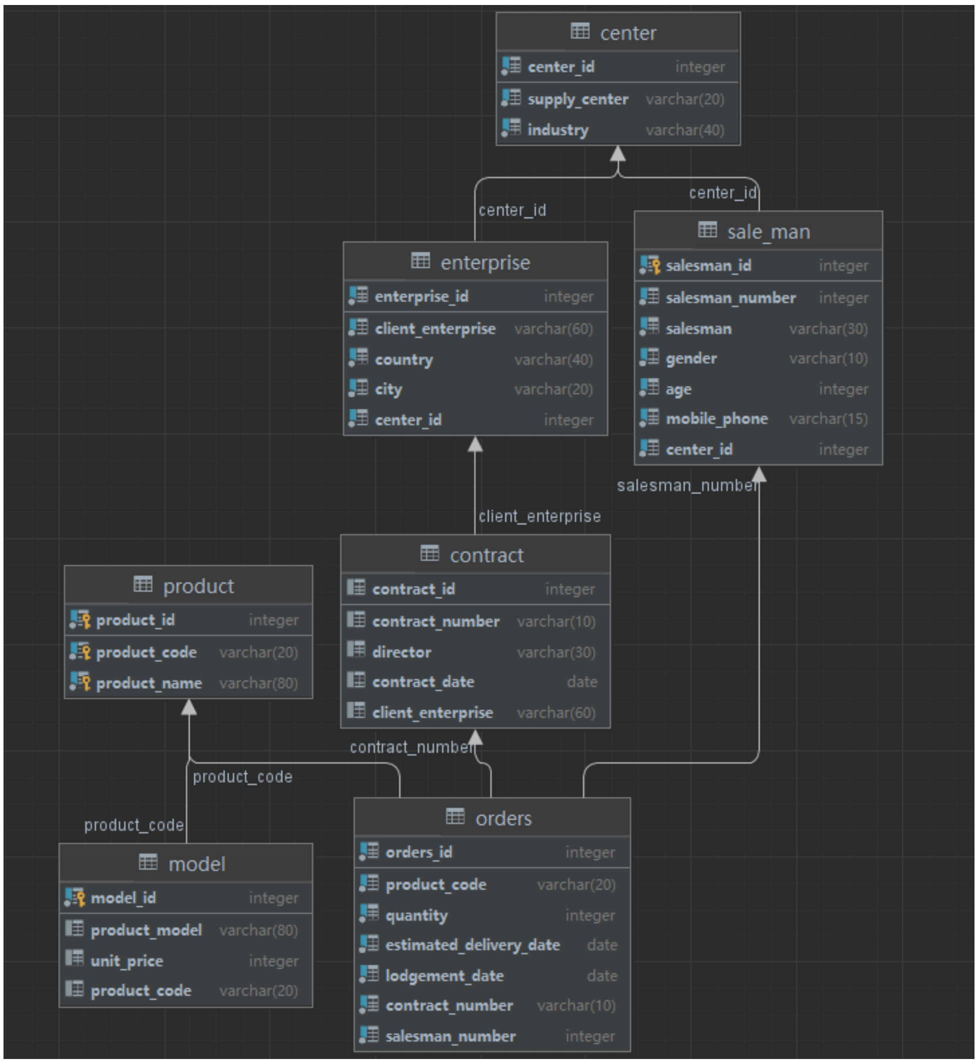
**Remind:** since the relationship between each entity is according to given data, I want to mention these:

- Every entity in the entity set participates in at least one relationship in the relationship set according to given data, which means they are all of total participaton.
- Weak entity manager_name contains every manager' s name responsible for corresponding supply center.
- Location is about where enterprises are located in, which determine they belong to which supply center.
- More details are in the diagram.



## Part Two: Database Design

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



| center | |
|---|---|
| center_id | integer |
| supply_center | varchar(20) |
| industry | varchar(40) |

| enterprise | |
|---|---|
| enterprise_id | integer |
| client_enterprise | varchar(60) |
| country | varchar(40) |
| city | varchar(20) |
| center_id | integer |

| sale_man | |
|---|---|
| salesman_id | integer |
| salesman_number | integer |
| salesman | varchar(30) |
| gender | varchar(10) |
| age | integer |
| mobile_phone | varchar(15) |
| center_id | integer |

| contract | |
|---|---|
| contract_id | integer |
| contract_number | varchar(10) |
| director | varchar(30) |
| contract_date | date |
| client_enterprise | varchar(60) |

| product | |
|---|---|
| product_id | integer |
| product_code | varchar(20) |
| product_name | varchar(80) |

| model | |
|---|---|
| model_id | integer |
| product_model | varchar(80) |
| unit_price | integer |
| product_code | varchar(20) |

| orders | |
|---|---|
| orders_id | integer |
| product_code | varchar(20) |
| quantity | integer |
| estimated_delivery_date | date |
| lodgement_date | date |
| contract_number | varchar(10) |
| salesman_number | integer |

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

- **How to design:**

  During the design of tables, I first list every column and consider the relationship between others. Then, using lines to connect two columns if there is a relationship between the two of them. In addition, writing down the mapping cardinality of every line and divide them into different relation set. After finish this, we can start to consider which column are supposed to be in certain tables, if "one-to-one", then it is just a column of the table, if "one-to-many" or "many-to-one", they belong to different tables and require foreign key to describe their relatinship, if "many-to-many", there should be an extra table to describe the relationship. What's more, in every relation set, there is at least one column that is unique and it can be the primary key of that relation set. Finally, you can get a clear relation diagram of these column which can guide you to create tables in datagrid.

- ## Meanings:
  - **center: multiple supply center with multiple industry**
    - **center_id:** a serial primary key, it describe the serial number of one center
    - **supply_center:** the name of a center
    - **industry:** the name of an industry
  - **enterprise: different enterprises**
    - **enterprise_id:** a serial primary key, it describe the serial number of one enterprise
    - **client_enterprise:** the name of a enterprise
    - **country:** the country where the enterprise located in
    - **city:** the city where the enterprise located in
    - **center_id:** foreign key, belongs to which center
  - **contract: different contracts**
    - **contract_id:** a serial primary key, it describe the serial number of one contract
    - **contract_number:** corresponding cnotract number
    - **director:** the person responsible for this contract
    - **contract_date:** the date of creating the contract
    - **client_enterprise:** foreign key, belongs to which center
  - **sale_man: different salesmen**
    - **salesman_id:** a serial primary key, it describe the serial number of one salesman
    - **salesman_number:** corresponding salesman number
    - **salesman:** the name of a salesman
    - **gender:** the gender of a salesman
    - **age:** the age of a salesman
    - **mobile_phone:** the phone number of a salesman
    - **center_id:** foreign key, responsible for which center
  - **product: different products**
    - **product_id:** a serial primary key, it describe the serial number of one product
    - **product_code:** the unique identifier of the product
    - **product_name:** the name of a product
  - **orders: different orders**
    - **orders_id:** corresponding order number
    - **product_code:** foreign key, which product in this order
    - **quantity:** the quantity of products ordered
    - **estimated_delivery_date:** the estimated date of the delivery of the product
    - **lodgement_date:** the actual date of the delivery of the product
    - **contract_number:** foreign key, belongs to which contract
    - **salesman_number:** foreign key, who responsible for this order
  - **model: different models**
    - **model_id:** a serial primary key, it describe the serial number of one model
    - **product_model:** the specific model of the product
    - **unit_price:** the unit price of a product_model
    - **product_code:** foreign key, belongs to which product

## Part Three: Data Import

- **Implementation Process:**

  1. Because the designed database has complex primary and foreign key constraints, the direct import efficiency is not high, so the script `center.py` is used to preprocess the original data `contract_info.csv`. The python Pandas library was used to read the data into the `dataframe`, select the corresponding columns, import the corresponding preprocessing files (src/splits), and read the corresponding tables from those files using Java scripts.
  2. Use `PreparedStatements`, batch commits, and turn off `AutoCommit` to provide preliminary optimizations for imports.
  3. The import process is divided into methods for multi-thread optimization. Specific optimization items: at the same time to establish the database connection and pretreatment, pretreatment using multi-thread synchronization generated pre-processing files, import process according to the topological order between the database tables for multi-thread optimization.

- **The Specific Implementation**

  1. Pretreatment

```python
import pandas as pd
import os
import threading

work_dir = '/Users/blank/IdeaProjects/DB_project_1/src'
splits_dir = f'{work_dir}/splits'

df = pd.read_csv(f'{work_dir}/contract_info.csv')
if not os.path.exists(splits_dir):
    os.mkdir(splits_dir)

# one of the method to read the source data and store in files
def create_center():
    center = df.loc[1:50001, ['supply center', 'industry']]
    center.drop_duplicates(subset=['supply center', 'industry'],
keep='first', inplace=True)
    file_name = f"{splits_dir}/center_sub.csv"
    center.to_csv(file_name, index=False)
...
# create threads
t1 = threading.Thread(target=create_center)
...
# run threads
t1.start()
...
```

  2. Connection

```java
// src.top.java
public static void main(String[] args) throws SQLException, IOException,
InterruptedException {
        System.out.println("currently running: (test computer 1,
PostgreSQL)");
        long start = System.currentTimeMillis();
        class Pre implements Runnable { // define the thread class and locks
            @Override
            public void run() {
                synchronized (lock1){
                    System.out.println("    Pretreatment time: 
"+pretreatment()+" ms");
                    t1 = true;
                    lock1.notify();
                }

            }
        }
        Thread thread1 = new Thread(new Pre());// pretreatment thread
        Thread thread2 = new Thread(new Con());// connection thread
        Thread thread3 = new Thread(new Imp());// import thread
        thread1.start();
        thread2.start();
        thread3.start();// start thread3 after thread1/2 executed
        thread3.join();
}
```

3. Concurrent

```
class Run1 implements Runnable{
    ...
    public Run1(int num, int cnt, Connection conn, PreparedStatement stmt){
        ...
    }
    @Override
    public void run() {
        synchronized (lock1) {
            DBConnect dbConnect = new DBConnect(1, cnt, conn, stmt);
            try {
                long time = dbConnect.connect(lookUp, count);
                t1 = true;
                lock1.notify();
                System.out.println("        table center has been imported, 
time: "+time+" ms");
            } catch (IOException | SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
...
// create threads
Thread thread1 = new Thread(new Run1(1, 1, top.conn, top.stmt1)); // center
Thread thread2 = new Thread(new Run2(2, 100001, top.conn, top.stmt2)); //
enterprise
...
thread1.start();
thread2.start();
...
thread6.join();// table "orders" takes the longest time
top.conn.commit();
top.conn.close();
```

- **Final Results：(The test environment are shown below)**

1. **(test computer 1, PostgreSQL)**
   Connect time: 196 ms
   Pretreatment time: 644 ms
   table center has been imported, time: 8 ms
   table product has been imported, time: 16 ms
   table enterprise has been imported, time: 19 ms
   table model has been imported, time: 43 ms
   table contract has been imported, time: 86 ms
   table salesman has been imported, time: 97 ms
   table orders has been imported, time: 839 ms
   Import time: 968 ms
   Total time: 1613 ms
2. **(test computer 1, MySQL)**
   Pretreatment time : 431 ms
   Connect time : 242ms
   table center has been imported, time: 18 ms
   table enterprise has been imported, time: 33 ms
   table contract has been imported, time: 321 ms
   table salesman has been imported, time: 54 ms
   table product has been imported, time: 14 ms
   table order has been imported, time: 2630 ms
   table model has been imported, time: 41 ms
   Import time : 3127 ms
   Total time : 3800 ms
3. **(test computer 2, PostgreSQL)**
   Connect time: 381 ms
   Pretreatment time: 1038 ms
   table product has been imported, time: 18 ms
   table center has been imported, time: 20 ms
   table enterprise has been imported, time: 9 ms
   table salesman has been imported, time: 37 ms
   table model has been imported, time: 45 ms
   table contract has been imported, time: 212 ms
   table orders has been imported, time: 2694 ms
   Import time: 2972 ms
   Total time: 4014 ms
4. **(test computer 2, MySQL)**
   Pretreatment time : 886 ms
   Connect time : 309ms
   table center has been imported, time: 23 ms
   table enterprise has been imported, time: 46 ms
   table contract has been imported, time: 895 ms
   table salesman has been imported, time: 131 ms
   table product has been imported, time: 40 ms
   table order has been imported, time: 7868 ms
   table model has been imported, time: 117 ms
   Import time : 9150 ms
   Total time : 10345 ms
5. **(test computer 3, PostgreSQL)**
   Connect time: 432 ms

```
    Pretreatment time: 1063 ms
        table product has been imported, time: 15 ms
        table center has been imported, time: 17 ms
        table model has been imported, time: 37 ms
        table salesman has been imported, time: 66 ms
        table enterprise has been imported, time: 8 ms
        table contract has been imported, time: 220 ms
        table orders has been imported, time: 2742 ms
    Import time: 3063 ms
Total time: 4129 ms
```

6. **(test computer 3, MySQL)**

```
    Pretreatment time : 832 ms
    Connect time : 362ms
        table center has been imported, time: 28 ms
        table enterprise has been imported, time: 57 ms
        table contract has been imported, time: 1135 ms
        table salesman has been imported, time: 183 ms
        table product has been imported, time: 51 ms
        table order has been imported, time: 8234 ms
        table model has been imported, time: 155 ms
Import time : 9882 ms
Total time : 11076 ms
```

- ## Comparative Study

1. Postgresql and MySQL are very different in import speed. Postgresql and MySQL databases take about the same amount of time to establish a connection with Java, so the difference is mainly due to the execution speed of specific database import instructions.

   The test shows that the import time of each table is from the start of the import to the end of the import. PostgreSQL is more advantageous in larger table import processes. It may be the caching mechanism and memory call strategy of different databases that cause the big difference in import speed.

2. The speed at which different operating systems import databases still varies greatly, and the differences are maintained across different databases. Referring to the data processing results of other groups, the differences between Linux and Windows among different distributions are not as significant as the results of this experiment. In addition to the differences caused by the different memory calls of different operating systems, consider the impact of computer performance on the speed of data import.

   In the Cinebench R23 benchmark, the single-core scores of the three participating machines were almost identical, so consider the impact of memory and storage on database import rates.

   The preprocessing time of test computer 1 is obviously shorter. Compared with PCI 3.0, the faster read and write speed of PCI 4.0 May be of great help. Test computer 1 uses LPDDR5 memory and has 2-3 times of read and write speed, which may be the reason for the faster import speed.

3. Due to limited conditions, there is no opportunity to test on more computers. Further exploration can be conducted on COMPUTERS using DDR5 (4800MHz-6400MHz) import test to confirm the specific relationship between import speed and memory.

# Part Four: Compare DBMS with File I/O

## 1. Description of test environment:

- **Test computer 1:**

  **SOC:** Apple M1 Pro (8 performance and 2 efficiency cores)

  **Memory:** 16GB LPDDR5 Universal Memory

  **Hard Drive:** 1TB SSD

- **Test computer 2:**

  **CPU:** AMD Ryzen 7 4800HS

  **Memory:** 16GB DDR4 3200MHz

  **Hard Drive:** Western Digital SN750 1TB

- **Test computer 3:**

  **CPU:** Intel Core i7-10750H

  **Memory:** 16GB DDR4 2666Mhz

  **Hard Drive1:** Samsung PM981a 256GB

  **Hard Drive2:** Seagate BarraCuda 1TB (SATA3)

- **JDK:** version 17.0.1

  **Python:** Python 3.10

  **PostgreSQL version:** 14.2

  **MySQL version:** 8.0.28

  Databases keep default setting

## 2. Specification of data organization:

We test four different manipulatioins: select, update, delete and insert, and in select part we test simple query and nested query. By including these test we can know the perforence of common manipulation. What's more this table is one of our designed table

about specific order and the form of file is .csv. Both of them are common inour daily life. That is also why we choose them.

## 3. Description of test SQL script and source code:

### Description of test SQL script and source code:

```
import java.io.*;
import java.sql.*;
import java.util.HashMap;
import java.util.Map;

public class DatabaseIO {
    static String url = "jdbc:postgresql://localhost:5432/project_1";
    static String username = "***";
    static String password = "***";

    static String url_my = "jdbc:mysql://localhost:3306/project_1";
    static String username_my = "***";
    static String password_my = "***";
    public static void main(String[] args) throws SQLException {
        Connection conn = DriverManager.getConnection(url, username, password);
        Statement statement = conn.createStatement();
        String command1 = "select product_code from orders where salesman_number
= 11110111;";
        String command2 = "select distinct t1.salesman_number, t2.quantity from
(select salesman_number, quantity from orders where lodgement_date <
to_date('2000-02-01', 'yyyy-mm-dd')) t1, (select salesman_number, quantity from
orders where quantity > 999) t2 where t1.salesman_number = t2.salesman_number;";
        String command4 = "update orders set salesman_number = 300991 where
salesman_number = 11110111;";
        String command5 = "delete from orders where salesman_number = 300991;";
        String command6 = "insert into orders (orders_id, product_code,
quantity, estimated_delivery_date, lodgement_date, contract_number,
salesman_number) values (550000, 'I36Z941', 800, '2010/4/12', '2010/4/21',
'CSE0004999', 12111601);";

        long time1 = System.nanoTime();
        statement.execute(command1);
        long time2 = System.nanoTime();
        System.out.println("Task1 takes : "+((double)(time2-time1)/1000000)+" ms
(Postgresql)");
        ...omit

        ////////////////
        Connection conn_my = DriverManager.getConnection(url_my, username_my,
password_my);
        statement = conn_my.createStatement();

        String command1_my = "select product_code from orders where
salesman_number = 11110111;";
        String command2_my = "select distinct t1.salesman_number, t2.quantity
from (select salesman_number, quantity from orders where lodgement_date <
date('2000-02-01')) t1, (select salesman_number, quantity from orders where
quantity > 999) t2 where t1.salesman_number = t2.salesman_number;";
        String command4_my = "update orders set salesman_number = 300991 where
salesman_number = 11110111;";
        String command5_my = "delete from orders where salesman_number =
300991;";
        String command6_my = "insert into orders (orders_id, product_code,
quantity, estimated_delivery_date, lodgement_date, contract_number,
salesman_number) values (550000, 'I36Z941', 800, '2010/4/12', '2010/4/21',
'CSE0004999', 12111601);";

        time1 = System.nanoTime();
        statement.execute(command1_my);
        time2 = System.nanoTime();
        System.out.println("Task1 takes : "+((double)(time2-time1)/1000000)+" ms
(MySQL)");
        ...omit
    }
}
```

## Benchmarking with Database APIs:

- **Use the database connection method in Task3 to test the sample separately, using 'system.nanotime ()' to improve the accuracy of obtaining the time**

## Benchmarking with file APIs:

- **Java part:**
    - In provided java program, we provide four data manipulations including insert, delete, update and select. What's more there are two different kinds of update manipulations and three different kinds of select manipulations.
    - For each different manipulation, if your input data is illegal, it will return zero and send you an error message which can tell you why running with error and do nothing to your file. And if your input data is legal, the program will return one and display how much time it takes and do corresponding manipulation.

    -
      ```
      public int insertAnOrder(String insertData);
      ```

      This function implement insert manipulation, it allows you to insert a new order "insertData" into target file. Please use comma to separate every column in the "insertData".

    -
      ```
      public int deleteAnOrder(int choose, String target);
      ```

      This function implement delete manipulation, it can delete specific order by index. "choose" is the index of column you want to delete and "target" is the specific data you want to delete. After this manipulation, all line with data "target" in corresponding column will be deleted.

    -
      ```
      public int updateAnOrder(int Id, String updateOrder);
      ```

      This function implement one of update manipulation, it allows you update a line with specific "Id", this "Id" is order_id column. And String "updateOrder" is the new order you update.

    -
      ```
      public int updateAnData(int choose,String former, String target);
      ```

      This function implement the other update manipulation, it allows you update specific data in a line. "choose" is index of column you want to update, "former" is the data you want to update and "target" is the data after updating. After this manipulation, all line with data "former" in column index "choose" will update to "target".

    -
      ```
      public int selectMultiData(int[] choose, int[] relation, int[] display,
      String[] target);
      ```

      This function implement one of select manipulation, it allows you select multiple data from source and whether to display. "choose" is an array contains indexes of column, "relation" describe the relationship between source data and target, in every chosen column, if the data you want to select is larger than target, then the integer of relation will be 1, if the data you want to select is smaller than target, then the integer of relation will be -1 and if the data you want to select is equal to target, then the integer of relation will be 0, "display" is an array with one and zero, one for display and 0 for not display, "target" is corresponding data used to compare.

-
  ```java
  public int selectCount(int choose, String target);
  ```

  This function implement another select manipulation, it allows you count the number of specific data. "choose" is the index of column, "target" is the data in specific column you want to count. After this manipulation, it will print the number of lines with "target" data in specific column.

-
  ```java
  public int selectNested(int[] choose, int[] relation, int[] display, String[] target, File csv, File out,int[] choose1, int[] relation1, int[] display1, String[] target1, File out1,File out2)
  ```

  This function implement another select mani pulation, it allows you select columns you want from multiple tables. Every parameter is explained in "selectMultiData".

## 4. Comparative study:

By doing this task we find:

- When you deal with huge amout of data, database  always faster then Java IO. However, this Java API is coding by we student, it can be slower then master' s code.
- What' s more comparing to different databases, we found postgreAQL has a better perfermence than MySQL. Maybe that is why we use pg instead of mysql?
- Following is runing time
  - computer 1:

    Task1 takes : 19.548 ms (Postgresql)
    Task2 takes : 30.384708 ms (Postgresql)
    Task4 takes : 4.047667 ms (Postgresql)
    Task5 takes : 2.920041 ms (Postgresql)
    Task6 takes : 0.407125 ms (Postgresql)
    Task1 takes : 33.861375 ms (MySQL)
    Task2 takes : 19.856292 ms (MySQL)
    Task4 takes : 15.15225 ms (MySQL)
    Task5 takes : 14.866625 ms (MySQL)
    Task6 takes : 1.223375 ms (MySQL)

    Task1 takes : 66 ms (Java)
    Task2 takes : 271 ms (Java)
    Task4 takes : 116 ms (Java)
    Task5 takes : 98 ms (Java)
    Task6 takes : 64 ms (Java)
  - computer 2:

    Task1 takes : 8.4903 ms (Postgresql)
    Task2 takes : 2.4952 ms (Postgresql)
    Task4 takes : 1.0183 ms (Postgresql)
    Task5 takes : 0.3717 ms (Postgresql)
    Task6 takes : 1.7434 ms (Postgresql)
    Task1 takes : 18.6438 ms (MySQL)
    Task2 takes : 2.2889 ms (MySQL)
    Task4 takes : 2.1887 ms (MySQL)

Task5 takes : 1.9963 ms (MySQL)
Task6 takes : 3.3106 ms (MySQL)

Task1 takes : 65 ms (Java)
Task2 takes : 726 ms (Java)
Task4 takes : 114 ms (Java)
Task5 takes : 105 ms (Java)
Task6 takes : 68 ms (Java)