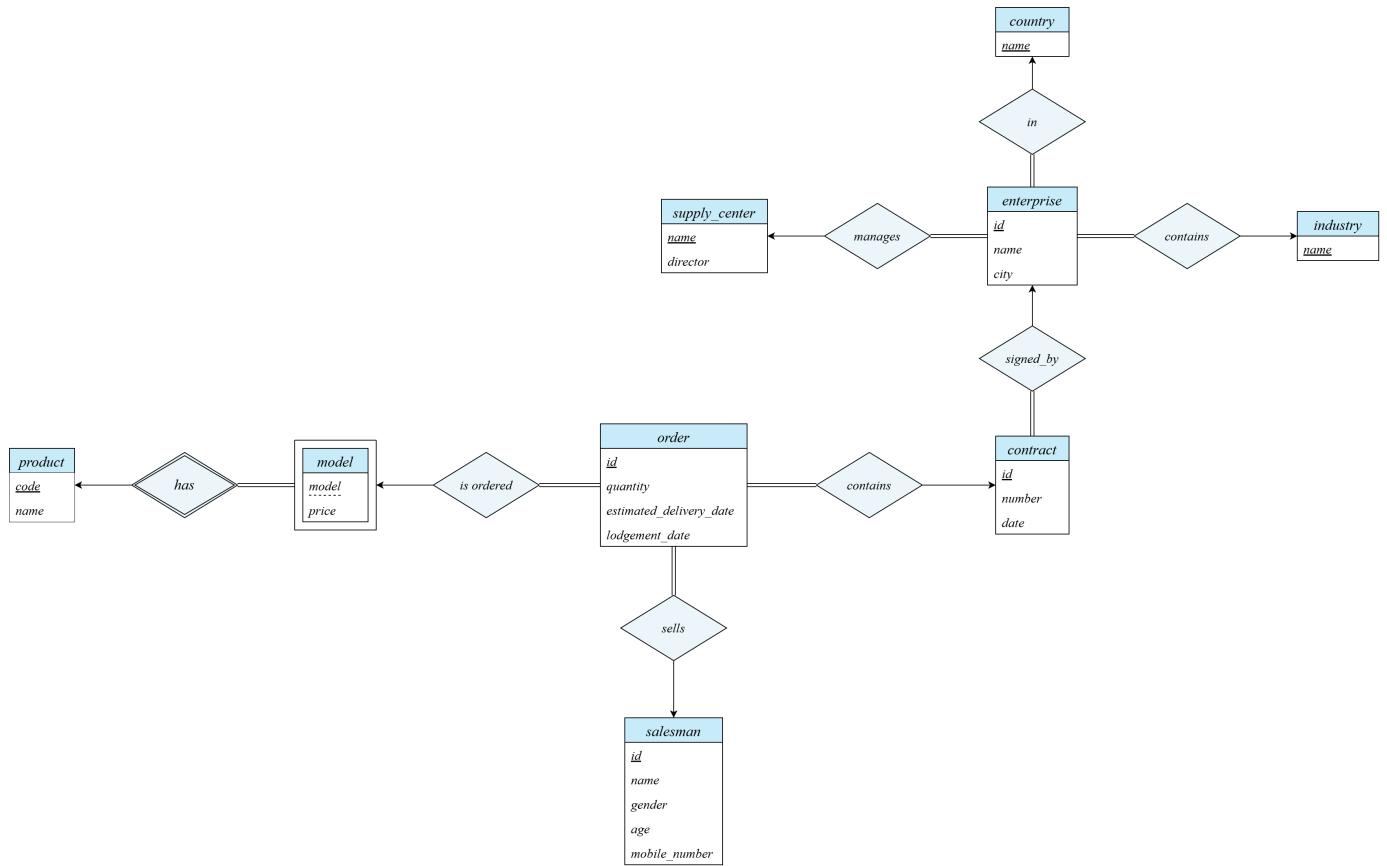


# CS307 2025 Spring Project1 Report

## Basic Information

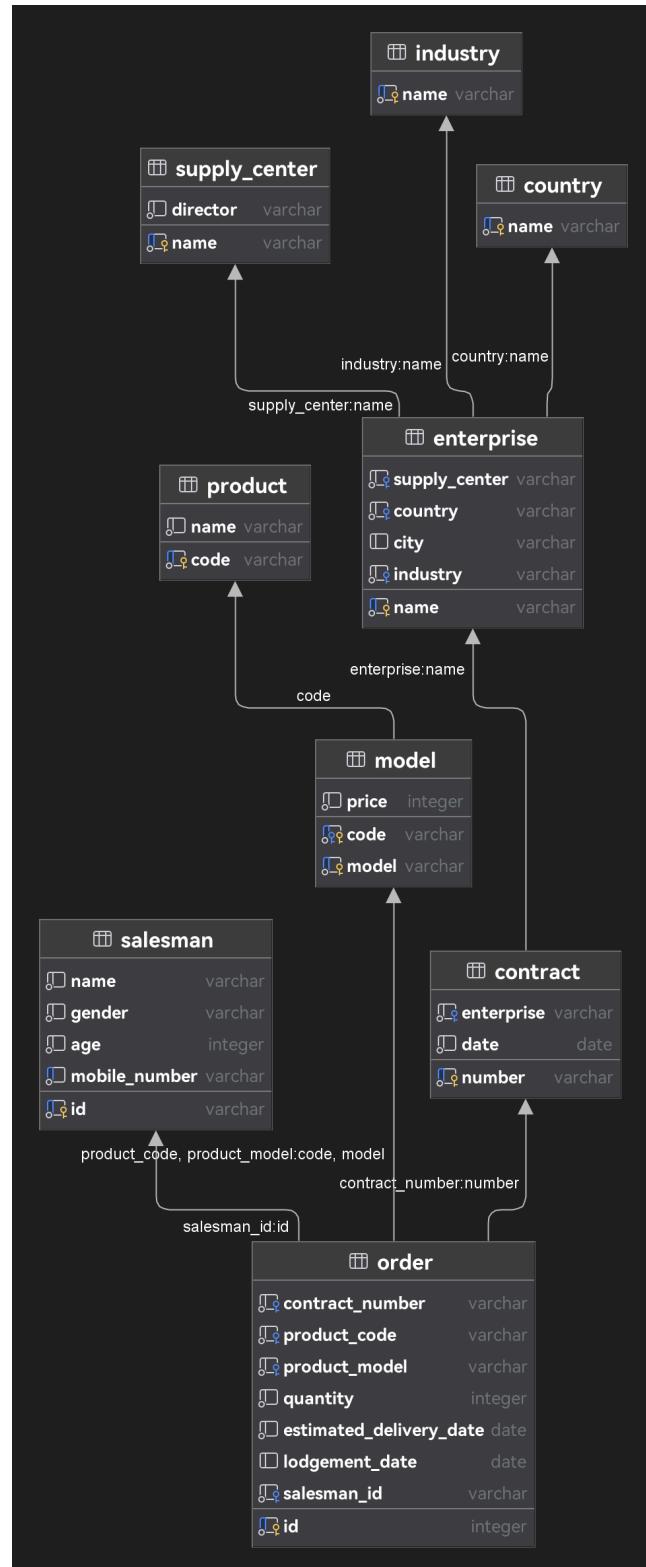
Name	Student ID	Lab Session	Contribution	Contribution Pct.
Zhu Mingyu	12312125	Wednesday 5-6	Data Import	40%
Zhou Liangyu	12312706	Wednesday 5-6	E-R Diagram, Database Design	60%

## E-R Diagram (drawn using draw.io)



## Database Design

E-R Diagram generated by DataGrip



## Description

Table `country`: Store the information of different countries

`name`: Primary key of varchar type. Country name.

Table `supply_center`: Store the information of supply centers.

`name`: Primary key of varchar type. Supply center name.

`director`: Varchar type. The name of the director who manages the supply center.

Table `industry`: Store the information of different industries.

`name`: Primary key of varchar type. Industry name.

Table `enterprise`: Store the information of enterprises.

`name`: Primary key of varchar type. Enterprise name.

`supply_center`: Foreign key of varchar type constrained by `not null`, which references `name` in `supply_center`. The supply center that manages the enterprise.

`country`: Foreign key of varchar type constrained by `not null`, which references `name` in `country`. The country in which the enterprise locates.

`city`: Varchar type. The city in which the enterprise locates. `null` if `country` is not `china`.

`industry`: Foreign key of varchar type constrained by `not null`, which references `name` in `industry`. The industry to which the enterprise belongs.

Table `product`: Store the information of products.

`code`: Primary key of varchar type. The product code.

`name`: Varchar type constrained by `not null`. The product name.

Table `model`: Store the information of models under certain products.

`code`: Foreign key of varchar type constrained by `not null`, which references `code` in `product`. The product code of the model.

`model`: Varchar type constrained by `not null`. The product model.

`price`: Integer type constrained by `not null`. The unit price of the model.

Primary key: `(code, model)`.

Table `salesman`: Store the information of salesmen.

`id`: Primary key of varchar type. The ID number of the salesman.

`name`: Varchar type constrained by `not null`. The number of the salesman.

`gender`: Varchar type constrained by `not null`. The gender of the salesman.

`age`: Integer type constrained by `not null`. The age of the salesman.

`mobile_number`: varchar type constrained by `unique not null`. The phone number of the salesman.

Table `contract`: Store the information of contracts.

`number`: Primary key of varchar type. The number of the contract.

`enterprise`: Foreign key of varchar type constrained by `not null`, which references `name` in `enterprise`. The enterprise that signed the contract.

`date`: Date type constrained by `not null`. The date that the contract was signed.

Table `order`: Store the information of different orders.

`id`: Serial primary key of integer type. The ID auto generated by system of the order.

`contract_number`: Foreign key of varchar type constrained by `not null`, which references `number` in `contract`. The contract number to which this order belongs.

`product_code`: Foreign key of varchar type constrained by `not null`, which references `code` in `model`. The code of the product that the order orders.

`product_model`: Foreign key of varchar type constrained by `not null`, which references `model` in `model`. The model of the product that the order orders.

`quantity`: Integer type constrained by `not null`. The quantity of the products that the order orders.

`estimated_delivery_date`: Date type constrained by `not null`. Estimated product delivery date.

`lodgement_date`: Date type. Actual product arrival date. ( `null` if it's later than `2025-3-24` )

`salesman_id`: Foreign key of varchar type constrained by `not null`, which references `id` in `salesman`. The ID of the salesman who sells the order.

## Data Import

### Basic Data Import by Scripts

By java

Script name	Author	Description
CsvToDB.java	Zhu Mingyu	Java file that provides method to process "file.csv" and import data into the database directly
CsvToSql.java	Zhu Mingyu	Java file that provides method to process "file.csv" and create a sql file "total.sql" which contains all the insert sentences needed to import data into the database
SqlToDB.java	Zhu Mingyu	Java file that provides method to connect the database and import the sql file into the database
Import.java	Zhu Mingyu	Run this java file with the arguments (0/1, file_name) to choose the different import way and import_file. If the first argument is 0, will run CsvToDB who will import data into the database directly If the first argument is 1, will run CsvToSql to create a sql file "total.sql" and then run SqlToDB to import data by sql
import.sh	Zhu Mingyu	Run this script with arguments (0 or 1, file_name) will compile the needed java files and then run Import with the same argument to import data from csv to database (If the file_name is empty, will import "file.csv")
Import.ps1	Zhu Mingyu	Script runned on Windows Powershell having same function with import.sh. Before running it, run "Set-ExecutionPolicy RemoteSigned -Scope CurrentUser" to let 'import.ps1' can be runned.

Besides these java file and shell scripts, there are three jar file(openCSV-5.7.1.jar, commons-lang3-3.12.0.jar, postgresql-42.2.5.jar).

openCSV-5.7.1.jar and commons-lang3-3.12.0.jar are used to process csv file to let the content be String[].

postgresql-42.2.5.jar is used to let the java program can connect with psql database directly.

## By python

Script name	Author	Description
csv_to_sql.py	Zhu Mingyu	Convert csv file(file.csv) to sql file("total.sql")
sql_to_DB.py	Zhu Mingyu	Import data into the database by sql file
csv_to_DB.py	Zhu Mingyu	Import data into the database by python file directly
import.py	Zhu Mingyu	Run this script with argument 0 or 1, If 0, run csv_to_DB, import directly If 1, run csv_to_sql and sql_to_DB to import data through a sql file
venv.sh	Zhu Mingyu	Create new python environment to import "psycopg2-binary" to connect with psql

## Data Accuracy checking

SQL queries for the requirements of the report are listed below.

```
1  -- How many salesmen are there for each gender?
2  select gender, count(gender)
3  from salesman
4  group by gender;
5
6  -- How many companies are there in each supply center?
7  select supply_center, count(supply_center)
8  from enterprise
9  group by supply_center;
10
11 -- How many salesmen are there in each supply center?
12 select supply_center, count(distinct salesman_id)
13 from "order" o
14     join contract c on c.number = o.contract_number
15     join enterprise e on e.name = c.enterprise
16 group by supply_center;
17
18 -- How many salesmen are there within a given age range (lower and upper bounds)?
19 select count(age)
20 from salesman
21 where age > 40
22 and age < 45;
23
24 -- How many countries are there in each supply center?
25 select supply_center, count(distinct country)
26 from enterprise
27 group by supply_center;
28
29 -- Given a country name, list all company names and their respective industries.
```

```

30 select name, industry
31 from enterprise
32 where country = 'United Kingdom';
33
34 -- Given a product code, list all product models and their corresponding unit prices.
35 select model, price
36 from model
37 where code = 'H7Y6824';
38
39 -- Given a contract number, list all order details in the contract, including (product
40 model, order
41 -- quantity, salesmen name, and lodgement_date).
42 select product_model, quantity, s.name as "salesman name", lodgement_date
43 from "order" o
44     join contract c on c.number = o.contract_number
45     join salesman s on s.id = o.salesman_id
46 where contract_number = 'CSE0020543';

```

## Advanced Requirements

### Script Optimization and Performance Comparison

- Script Optimization

- use import.sh to compile java files and run the proper java program with argument.

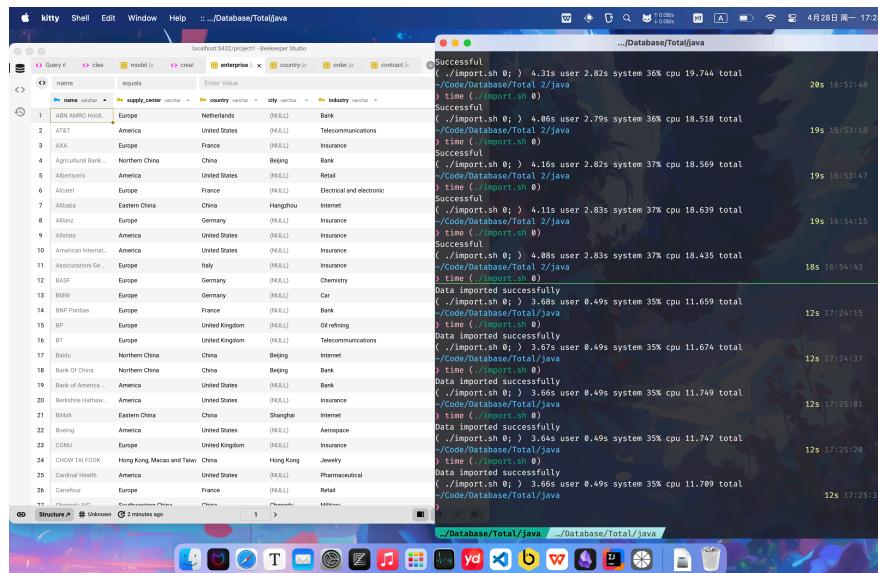
It saves time for people to compile files. And the argument are available for people to test different way to import data

- use batch processing

Not import data line by line, but use batch processing.

Here I set the size of batch 5000, it can be changed for different size of csv file

Test	1	2	3	4	5
Line by line	19.744s	18.518s	18.569s	18.639s	18.435s
Batch processing	11.659s	11.674s	11.749s	11.747s	11.789s



when I import data line by line, the script costs 19s, but when I use batch processing, it costs 12s

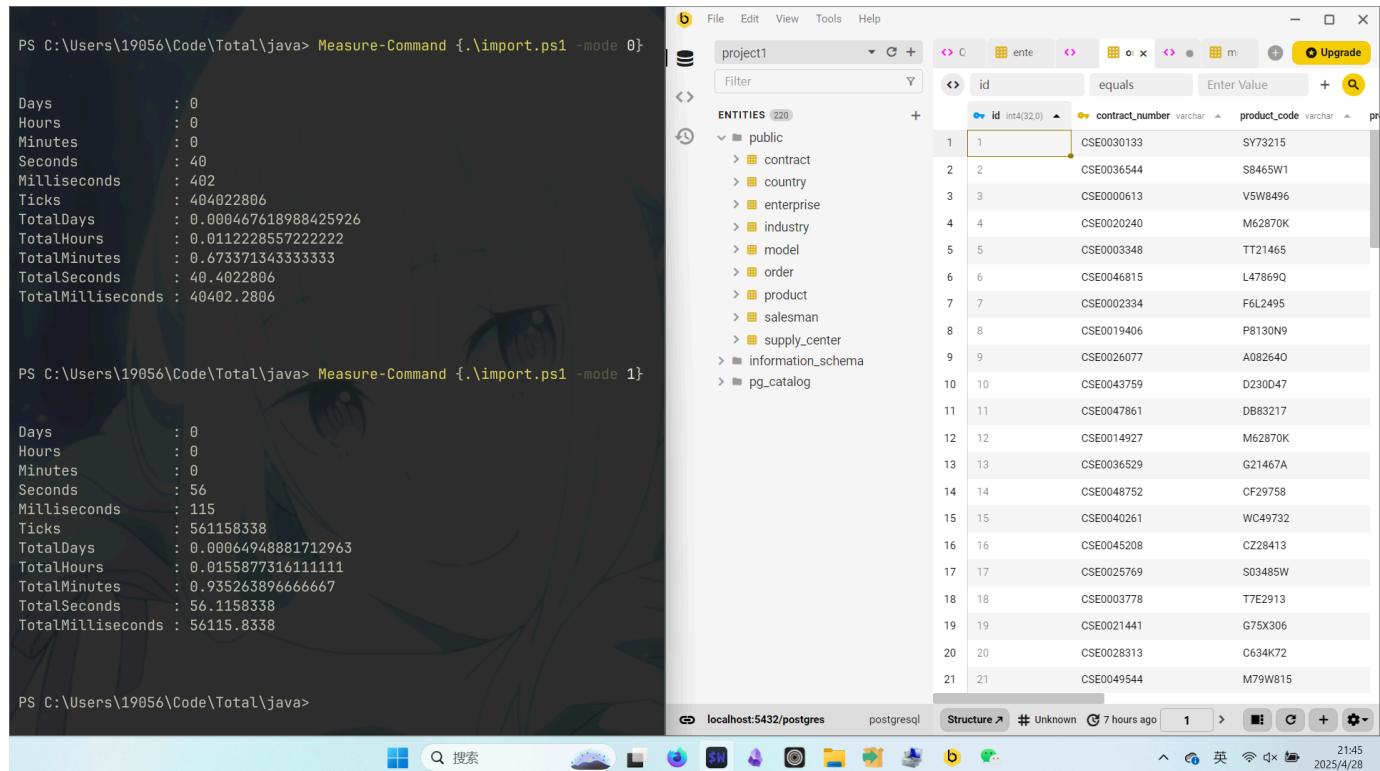
- Different ways to import data
  - import data from csv file to database directly(way 0)
    - Java: CsvToDB.java
    - Python: csv\_to\_DB.py
  - create sql file by proccsing the csv file, and then import data from sql file to database(way 1)
    - Java: CsvToSql.java, SqlToDB.java
    - Python: csv\_to\_sql.py, sql\_to\_DB.py
- Comparative analysis of the computational efficiencies between these two ways:

Test	1	2	3	4	5
Way 0	11.765s	11.711s	11.682s	16.341s	11.660s
Way 1	15.482s	11.386s	15.568s	15.466s	15.361s

Both way use batch proccsing, but way 1 need to create a new sql file, so it takes more time.

## Data import across multiple systems: Windows, macOS, Linux

### Windows:



```

PS C:\Users\19056\Code\Total\java> Measure-Command {.\import.ps1 -mode 0}
Days : 0
Hours : 0
Minutes : 0
Seconds : 40
Milliseconds : 402
Ticks : 404022806
TotalDays : 0.000467618988425926
TotalHours : 0.0112228557222222
TotalMinutes : 0.673371343333333
TotalSeconds : 40.4022806
TotalMilliseconds : 40402.2806

PS C:\Users\19056\Code\Total\java> Measure-Command {.\import.ps1 -mode 1}
Days : 0
Hours : 0
Minutes : 0
Seconds : 56
Milliseconds : 115
Ticks : 561158338
TotalDays : 0.00064948881712963
TotalHours : 0.015877314111111
TotalMinutes : 0.9352638966666667
TotalSeconds : 56.1158338
TotalMilliseconds : 56115.8338
  
```

Test environment: Windows 11, Powershell

### Procedures:

- Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

Powershell defaultly not allow user to run script, this command gives the current user permission to run script .

- Measure-Command {.\import.ps1 -mode 0}

Use Measure-Command to test the time of script when the argument is 0 which means java program will import data from csv to database directly.

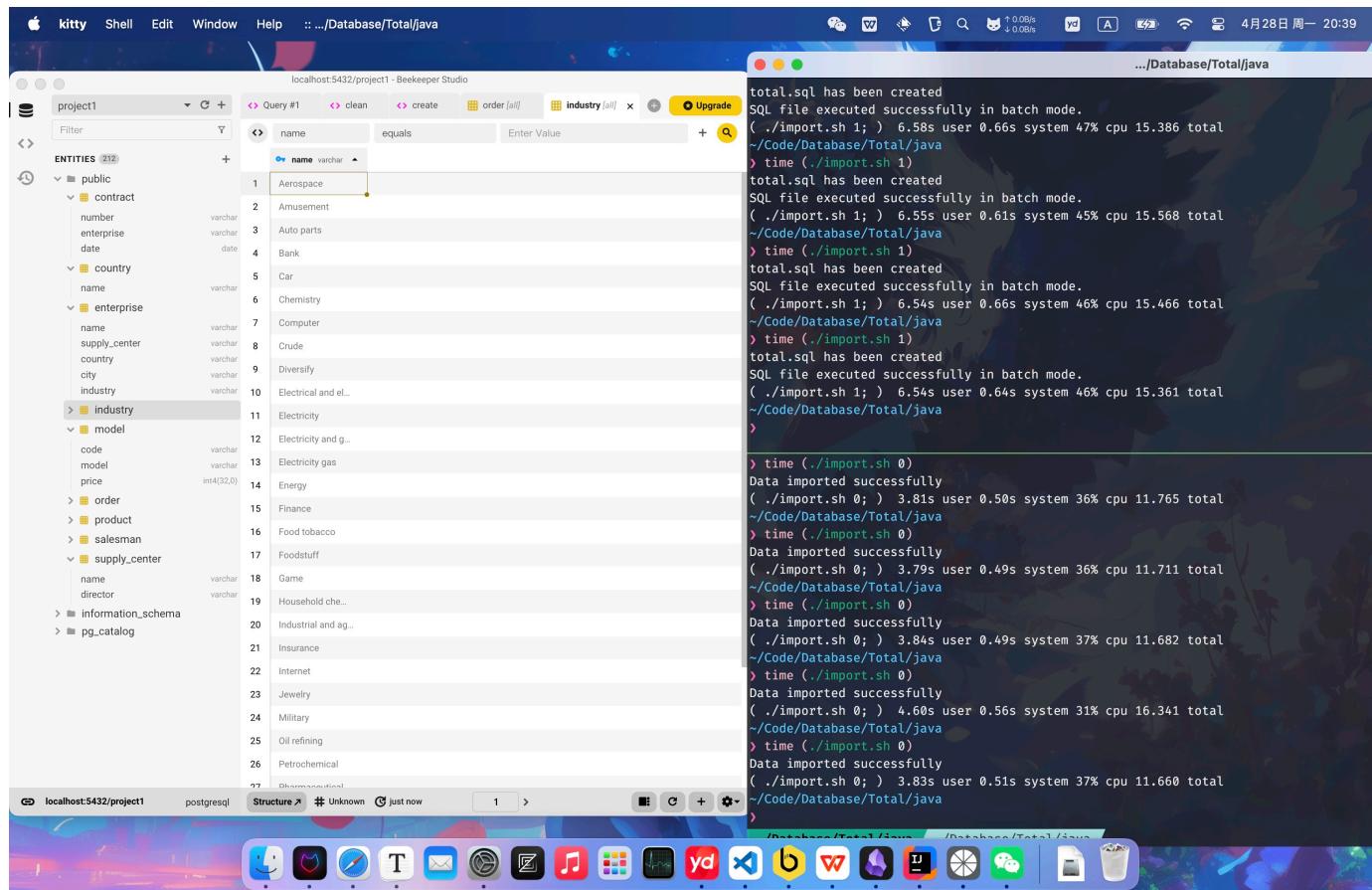
- Measure-Command {.\import.ps1 -mode 1}

Use Measure-Command to test the time of script when the argument is 1 which means java program will import data from the csv file to sql file and then import data from sql file to database.

Time cost:

- Import directly(way 0): 40.402s
- From sql file(way 1): 56.115s

MacOS:



Test environment: macOS Sequoia 15.4.1, zsh

Procedures:

- time(./import.sh 0)

Use time to test the time of the script when the argument is 0 which means the java program will import data from csv to database directly

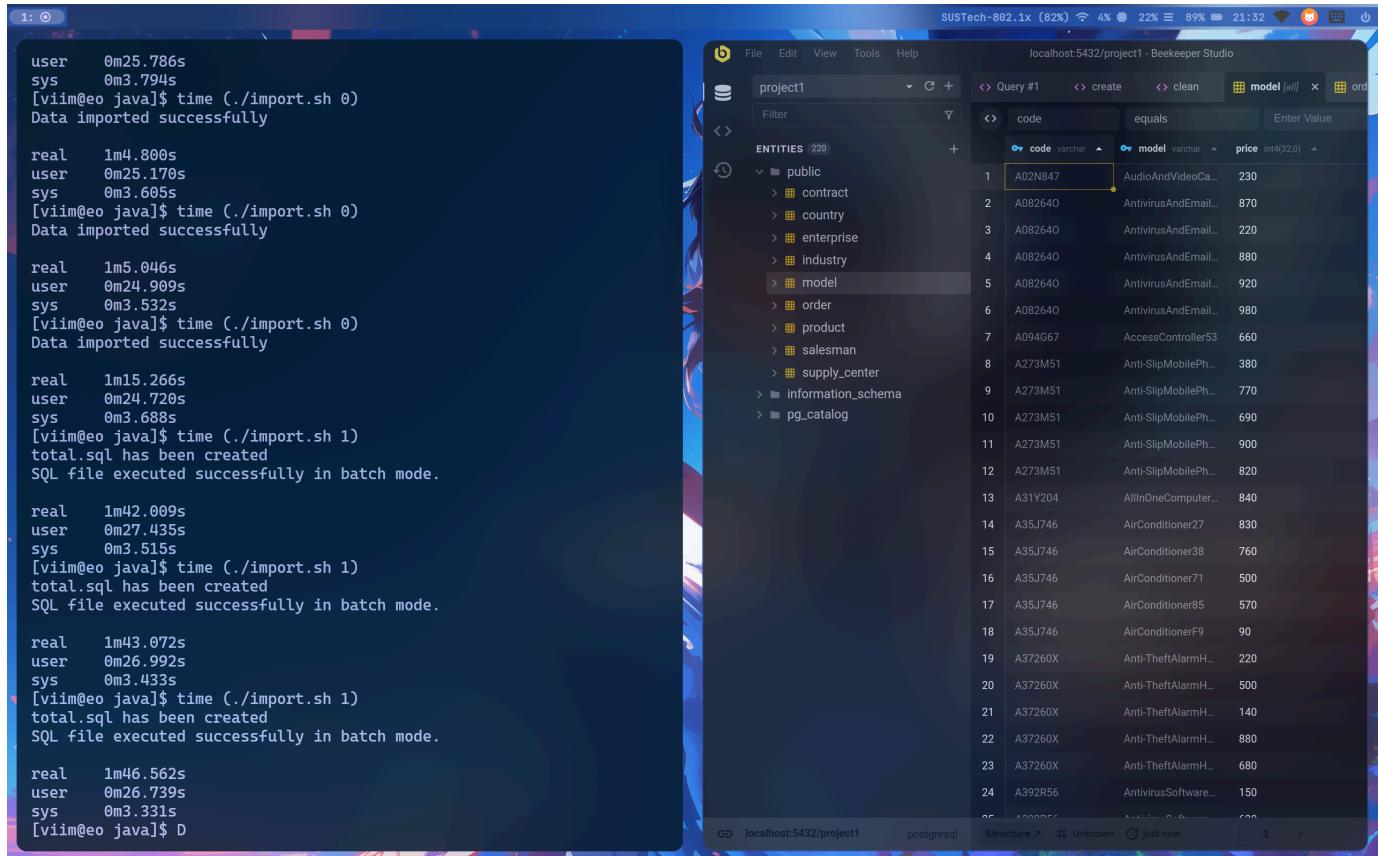
- Time(./import.sh 1)

Use time to test the time of the script when the argument is 1 which means the java program will import data from the csv file to a sql file and then import data from the sql file to the database

## Time cost

Test	1	2	3	4	5
Way 0	11.765s	11.711s	11.682s	16.341s	11.660s
Way 1	15.482s	11.386s	15.568s	15.466s	15.361s

## Linux:



The terminal window on the left shows the execution of the script with arguments 0 and 1, and the Beekeeper Studio application on the right shows a database table with 25 rows of data.

```

user 0m25.786s
sys 0m3.794s
[vim@eo java]$ time ./import.sh 0
Data imported successfully

real 1m4.800s
user 0m25.170s
sys 0m3.605s
[vim@eo java]$ time ./import.sh 0
Data imported successfully

real 1m5.046s
user 0m24.909s
sys 0m3.532s
[vim@eo java]$ time ./import.sh 0
Data imported successfully

real 1m15.266s
user 0m24.720s
sys 0m3.688s
[vim@eo java]$ time ./import.sh 1
total.sql has been created
SQL file executed successfully in batch mode.

real 1m42.009s
user 0m27.435s
sys 0m3.515s
[vim@eo java]$ time ./import.sh 1
total.sql has been created
SQL file executed successfully in batch mode.

real 1m43.072s
user 0m26.992s
sys 0m3.433s
[vim@eo java]$ time ./import.sh 1
total.sql has been created
SQL file executed successfully in batch mode.

real 1m46.562s
user 0m26.739s
sys 0m3.331s
[vim@eo java]$ D
  
```

Test environment: EndeavourOS(kernel: Linux 6.14.2-arch1-1)

## Procedures:

- time(./import.sh 0)

Use time to test the time of the script when the argument is 0 which means the java program will import data from csv to database directly

- Time(./import.sh 1)

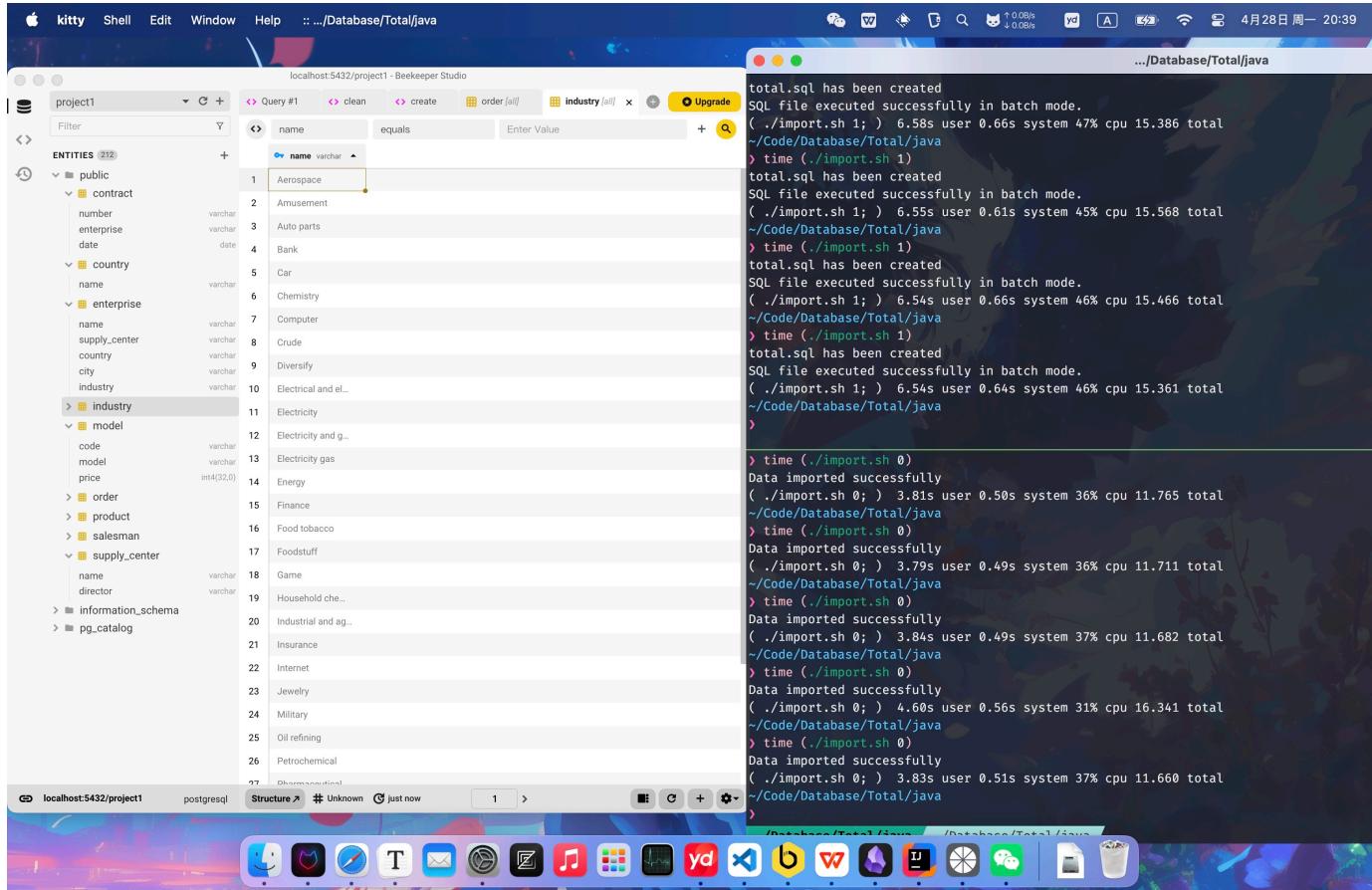
Use time to test the time of the script when the argument is 1 which means the java program will import data from the csv file to a sql file and then import data from the sql file to the database

## Time cost

Test	1	2	3
way 0	64.800s	65.046s	75.266s
way 1	102.009s	103.072s	106.562s

## Data import using various programming languages: Java, Python

Java:



localhost:5432/project1 - Beekeeper Studio

project1

ENTITIES (21)

- public
- contract
- country
- enterprise
- model
- order
- product
- salesman
- supply\_center
- information\_schema
- pg\_catalog

Filter: name equals Enter Value: Aerospace

1 Aerospace

2 Amusement

3 Auto parts

4 Bank

5 Car

6 Chemistry

7 Computer

8 Crude

9 Diversify

10 Electrical and el...

11 Electricity

12 Electricity and g...

13 Electricity ga...

14 Energy

15 Finance

16 Food tobacco

17 Foodstuff

18 Game

19 Household che...

20 Industrial and ag...

21 Insurance

22 Internet

23 Jewelry

24 Military

25 Oil refining

26 Petrochemical

27 Pharmaceutical

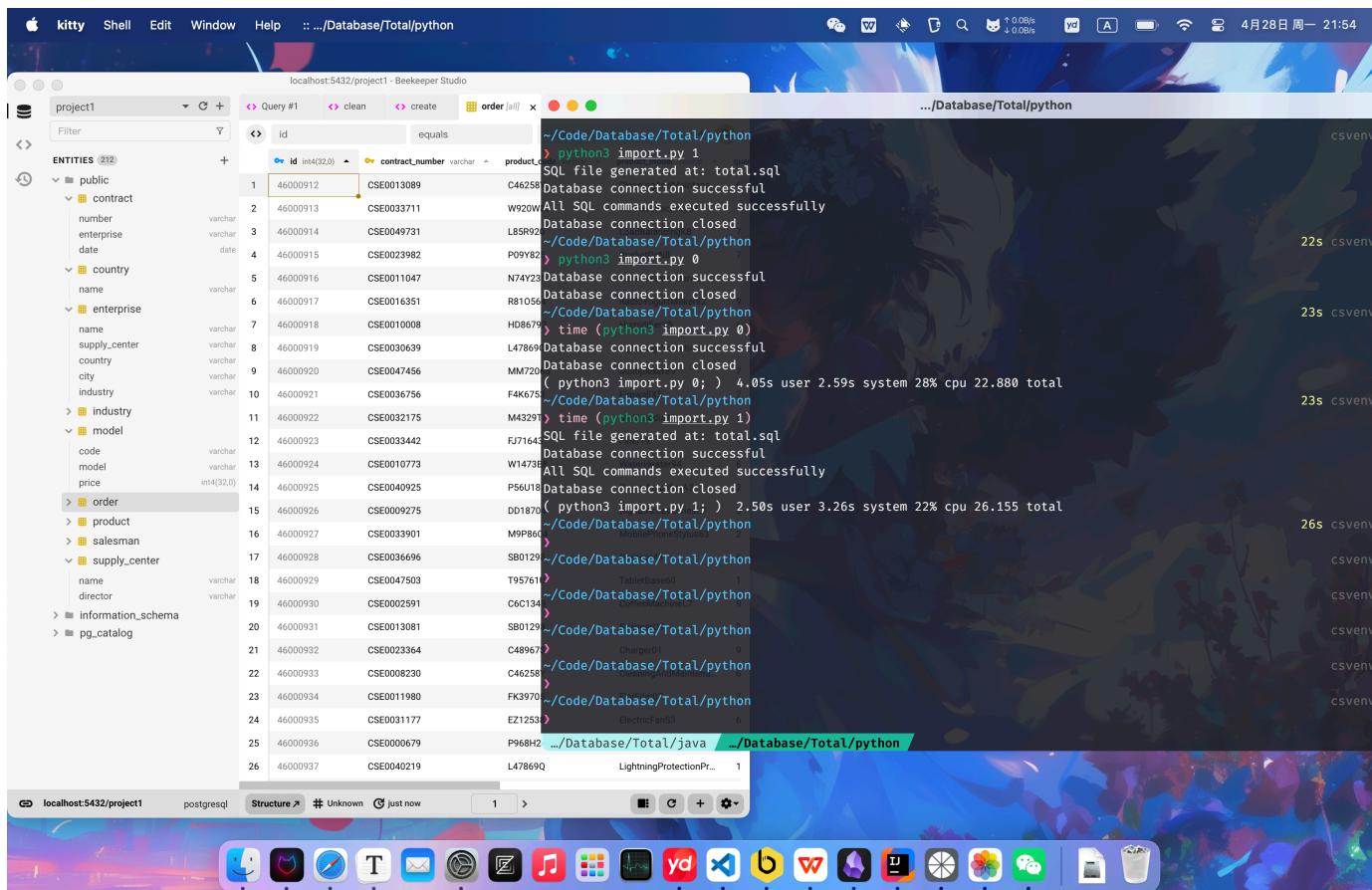
localhost:5432/project1 postgresql Structure # Unknown just now 1 > +

.../Database/Total/java

```
total.sql has been created
SQL file executed successfully in batch mode.
( ./import.sh 1; ) 6.58s user 0.66s system 47% cpu 15.386 total
~/Code/Database/Total/java
> time ./import.sh 1
total.sql has been created
SQL file executed successfully in batch mode.
( ./import.sh 1; ) 6.55s user 0.61s system 45% cpu 15.568 total
~/Code/Database/Total/java
> time ./import.sh 1
total.sql has been created
SQL file executed successfully in batch mode.
( ./import.sh 1; ) 6.54s user 0.66s system 46% cpu 15.466 total
~/Code/Database/Total/java
> time ./import.sh 1
total.sql has been created
SQL file executed successfully in batch mode.
( ./import.sh 1; ) 6.54s user 0.64s system 46% cpu 15.361 total
~/Code/Database/Total/java
>

> time ./import.sh 0
Data imported successfully
( ./import.sh 0; ) 3.81s user 0.50s system 36% cpu 11.765 total
~/Code/Database/Total/java
> time ./import.sh 0
Data imported successfully
( ./import.sh 0; ) 3.79s user 0.49s system 36% cpu 11.711 total
~/Code/Database/Total/java
> time ./import.sh 0
Data imported successfully
( ./import.sh 0; ) 3.84s user 0.49s system 37% cpu 11.682 total
~/Code/Database/Total/java
> time ./import.sh 0
Data imported successfully
( ./import.sh 0; ) 4.60s user 0.56s system 31% cpu 16.341 total
~/Code/Database/Total/java
> time ./import.sh 0
Data imported successfully
( ./import.sh 0; ) 3.83s user 0.51s system 37% cpu 11.660 total
~/Code/Database/Total/java
>
```

Python:



localhost:5432/project1 - Beekeeper Studio

project1

ENTITIES (21)

- public
- contract
- country
- enterprise
- model
- order
- product
- salesman
- supply\_center
- information\_schema
- pg\_catalog

Filter: id equals Enter Value: 46000912

1 46000912 CSE0013089

2 46000913 CSE0033711

3 46000914 CSE0049731

4 46000915 CSE0023982

5 46000916 CSE0011047

6 46000917 CSE0016351

7 46000918 CSE0010008

8 46000919 CSE0030639

9 46000920 CSE0047456

10 46000921 CSE0036756

11 46000922 CSE0032175

12 46000923 CSE0033442

13 46000924 CSE0010773

14 46000925 CSE0040925

15 46000926 CSE0009275

16 46000927 CSE0033901

17 46000928 CSE0036696

18 46000929 CSE0047503

19 46000930 CSE0002591

20 46000931 CSE0013081

21 46000932 CSE0023364

22 46000933 CSE0008230

23 46000934 CSE0011980

24 46000935 CSE0031177

25 46000936 CSE0000679

26 46000937 CSE0040219

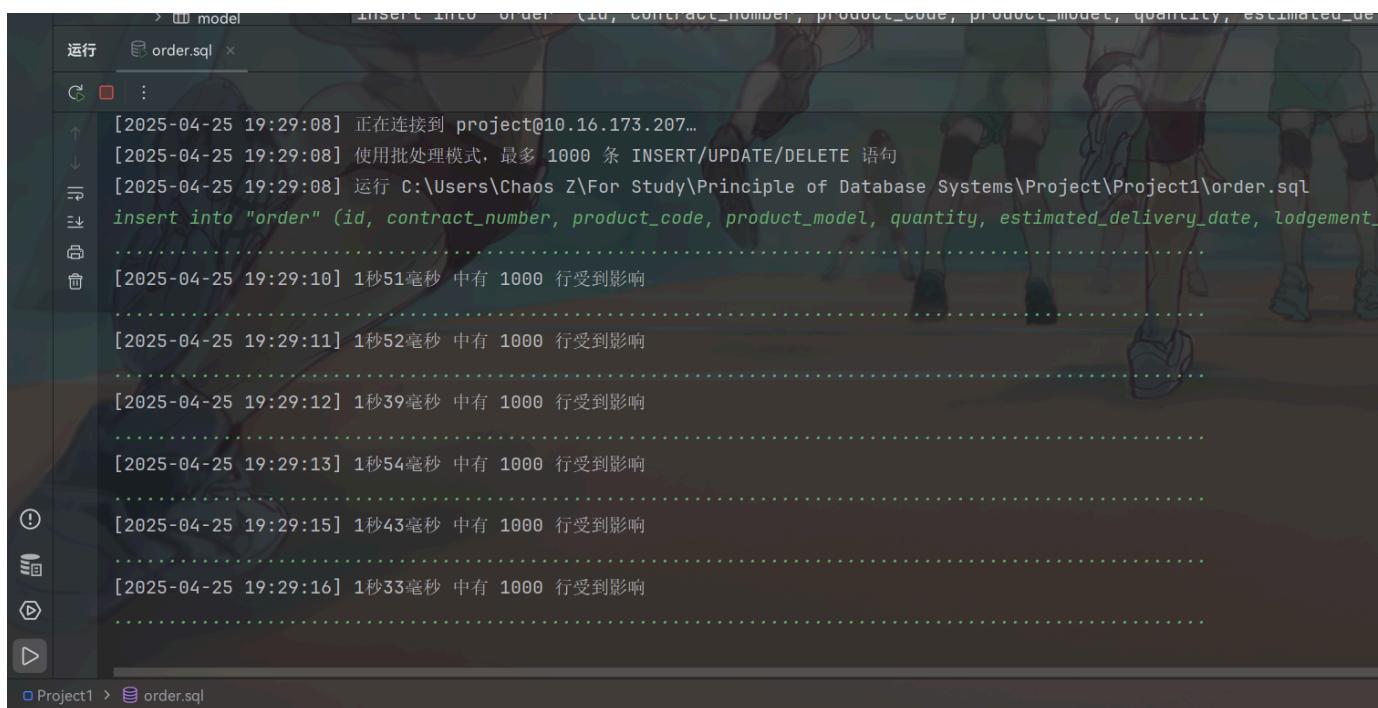
~/Code/Database/Total/python
> Python3 import.py 1
SQL file generated at: total.sql
Database connection successful
W920W All SQL commands executed successfully
L85R92 Database connection closed
~/Code/Database/Total/python
P09V82 > python3 import.py 0
N7V22 Database connection successful
R81054 Database connection closed
~/Code/Database/Total/python
H8676 > time (python3 import.py 0)
L47869 Database connection successful
MM726 Database connection closed
( python3 import.py 0; ) 4.05s user 2.59s system 28% cpu 22.880 total
F4K675 ~/Code/Database/Total/python
M43297 > time (python3 import.py 1)
F71644 SQL file generated at: total.sql
Database connection successful
W14737 All SQL commands executed successfully
P56U18 Database connection closed
( python3 import.py 1; ) 2.50s user 3.26s system 22% cpu 26.155 total
D01870 ~/Code/Database/Total/python
M9P866 > time (python3 import.py 1)
SB0129 ~/Code/Database/Total/python
T95761 TabletBase60
C46258 > ~/Code/Database/Total/python
C6C134 > ~/Code/Database/Total/python
SB0129 > ~/Code/Database/Total/python
T95761 > ~/Code/Database/Total/python
C48967 > ~/Code/Database/Total/python
C46258 > ~/Code/Database/Total/python
FK3970 > ~/Code/Database/Total/python
EZ1253 > ElectricFanSS
P968H2 .../Database/Total/java .../Database/Total/python
L47869Q LightningProtectionPr...

## Experiment with other databases: SQLite, Apache Derby, H2 Database

The following data is based on the `project` database with an identical structure, executing `insert` statements on the `contract` table. The databases are deployed on an ARM-based Ubuntu server with Hi3798MV100(Cortex-A7 @ 1.5GHz ×4) CPU and 512MB RAM.

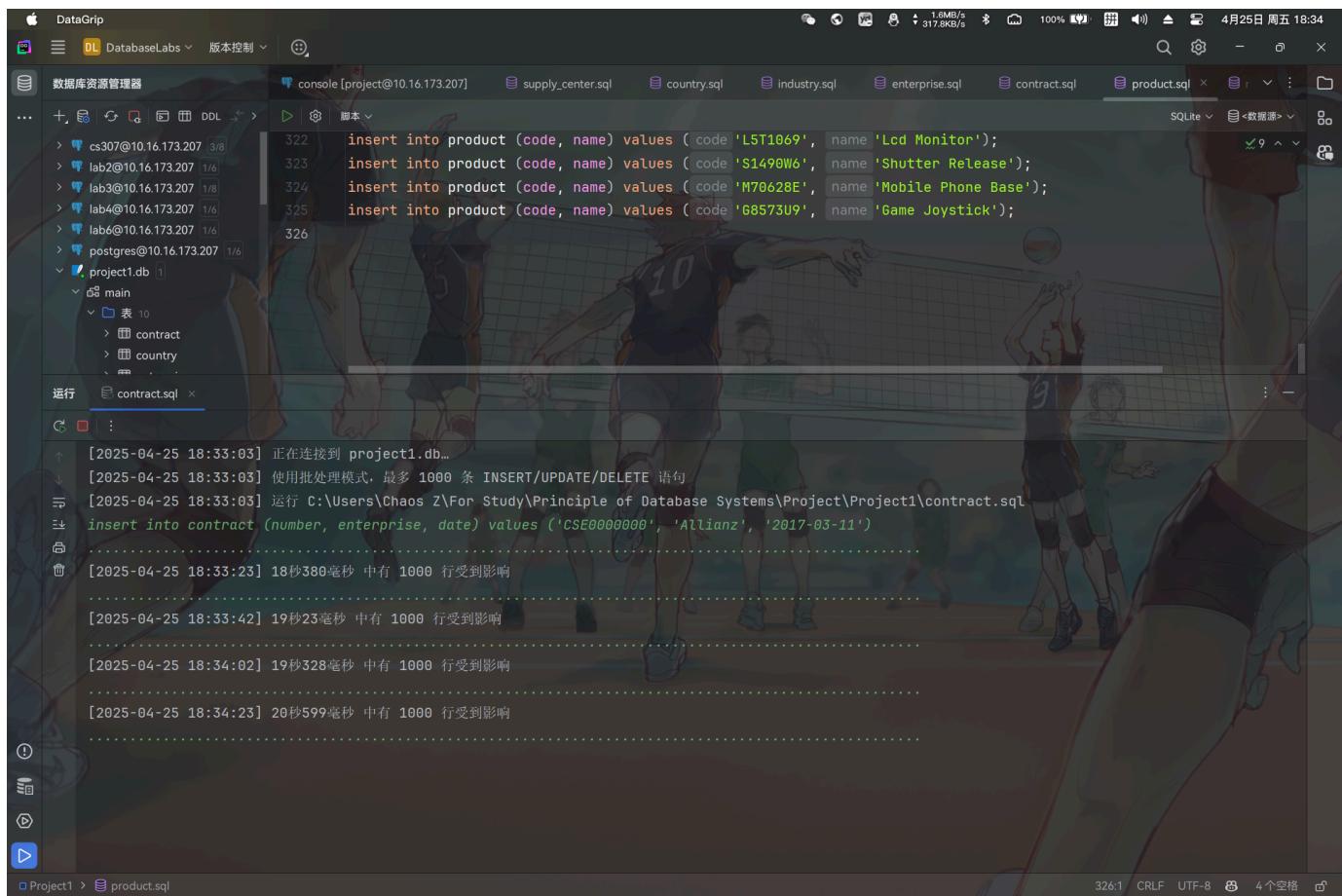
Name	Feature	Storage	Standards	Performance(avg. time per 1000 <code>insert</code> )
PostgreSQL	Comprehensive, high scalability and reliability.	Disk	SQL 92, SQL 99, SQL 2003, SQL 2008, SQL/JSON	1.05s
SQLite	Lightweight, cross-platform	Single File	SQL 92, basic ANSI SQL	19s
Apache Derby	Embedded database, Java integration.	Disk	SQL 92, basic ANSI SQL	12.5s
H2	In-memory storage, efficient optimization	Memory/Disk	SQL 92, SQL 2003, ANSI SQL	Decreases from 3.5s to 1.5s with more executed <code>insert</code>

PostgreSQL:



```
运行  order.sql >
[2025-04-25 19:29:08] 正在连接到 project@10.16.173.207...
[2025-04-25 19:29:08] 使用批处理模式, 最多 1000 条 INSERT/UPDATE/DELETE 语句
[2025-04-25 19:29:08] 运行 C:\Users\Chaos Z\For Study\Principle of Database Systems\Project\Project1\order.sql
[2025-04-25 19:29:08] insert into "order" (id, contract_number, product_code, product_model, quantity, estimated_delivery_date, lodgement
[2025-04-25 19:29:10] 1秒51毫秒 中有 1000 行受到影响
[2025-04-25 19:29:11] 1秒52毫秒 中有 1000 行受到影响
[2025-04-25 19:29:12] 1秒39毫秒 中有 1000 行受到影响
[2025-04-25 19:29:13] 1秒54毫秒 中有 1000 行受到影响
[2025-04-25 19:29:15] 1秒43毫秒 中有 1000 行受到影响
[2025-04-25 19:29:16] 1秒33毫秒 中有 1000 行受到影响
```

## SQLite:

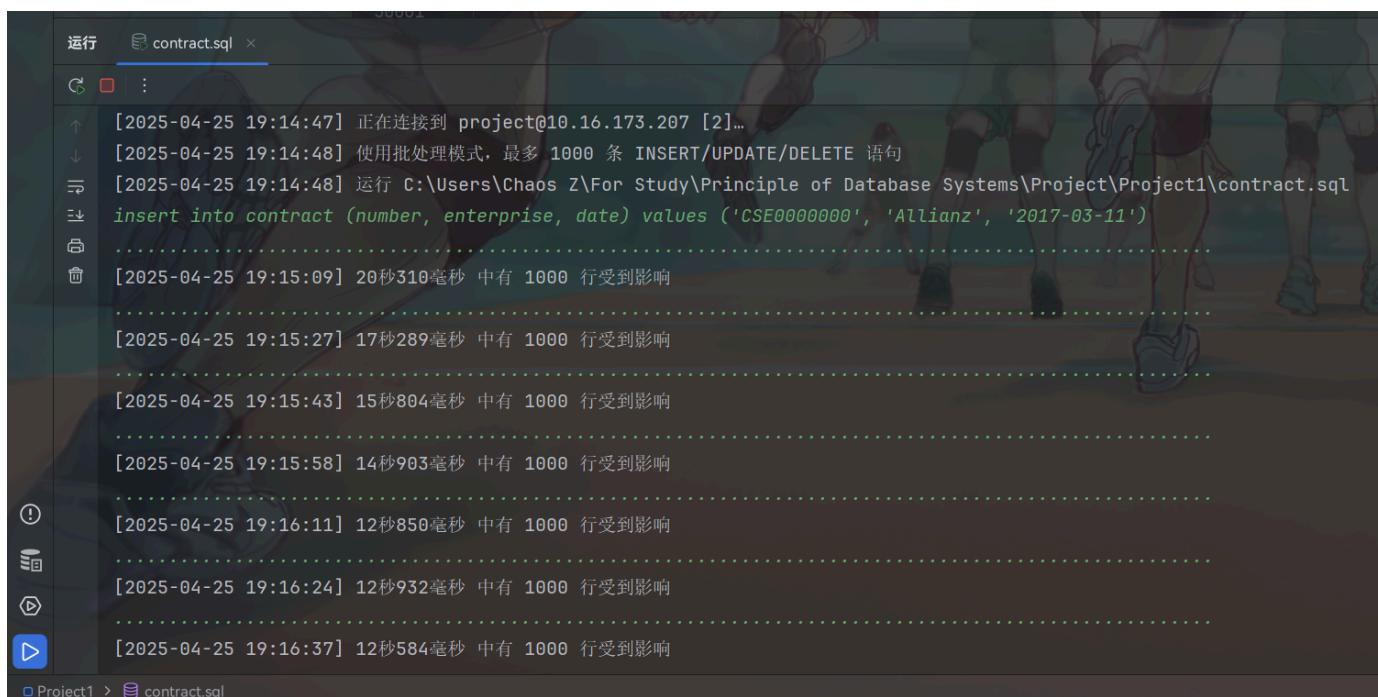


The screenshot shows the DataGrip interface for SQLite. The left sidebar displays the database structure of 'project1.db', including the 'main' schema and its 'contract', 'country', and 'product' tables. The main panel shows the 'contract.sql' script being run. The output window displays the execution log, showing the connection to 'project1.db', the use of batch processing, the execution of the 'contract.sql' script, and the insertion of a single row into the 'contract' table. The log also includes performance metrics for each step, such as execution times and the number of rows affected.

```
insert into product (code, name) values ('LST1069', 'Lcd Monitor');
insert into product (code, name) values ('S1490W6', 'Shutter Release');
insert into product (code, name) values ('M70628E', 'Mobile Phone Base');
insert into product (code, name) values ('G8573U9', 'Game Joystick');
```

```
[2025-04-25 18:33:03] 正在连接到 project1.db...
[2025-04-25 18:33:03] 使用批处理模式, 最多 1000 条 INSERT/UPDATE/DELETE 语句
[2025-04-25 18:33:03] 运行 C:\Users\Chaos Z\For Study\Principle of Database Systems\Project\Project1\contract.sql
[2025-04-25 18:33:03] insert into contract (number, enterprise, date) values ('CSE00000000', 'Allianz', '2017-03-11')
[2025-04-25 18:33:23] 18秒380毫秒 中有 1000 行受到影响
[2025-04-25 18:33:42] 19秒23毫秒 中有 1000 行受到影响
[2025-04-25 18:34:02] 19秒328毫秒 中有 1000 行受到影响
[2025-04-25 18:34:23] 20秒599毫秒 中有 1000 行受到影响
```

## Apache Derby:

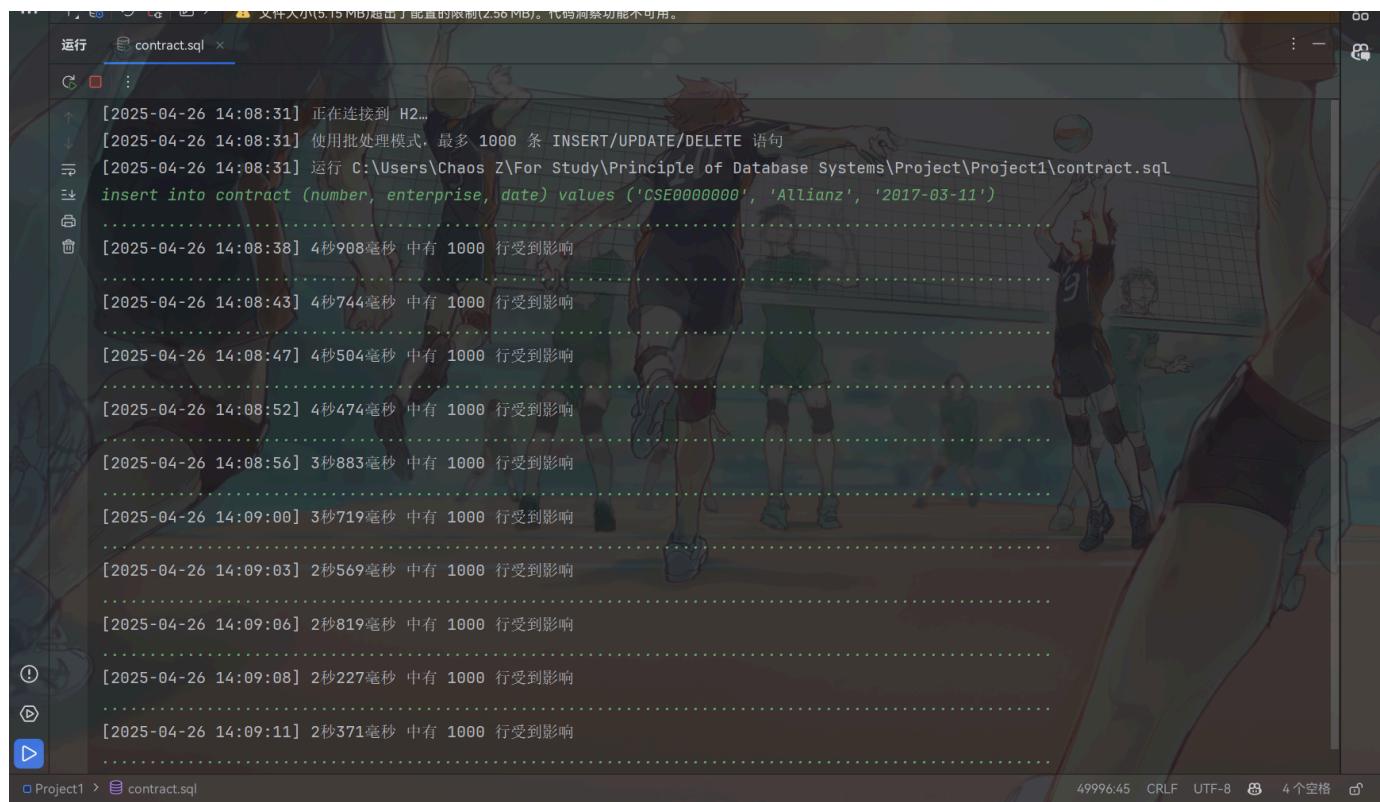


The screenshot shows the DataGrip interface for Apache Derby. The left sidebar displays the database structure of 'project1'. The main panel shows the 'contract.sql' script being run. The output window displays the execution log, showing the connection to 'project@10.16.173.207 [2]', the use of batch processing, the execution of the 'contract.sql' script, and the insertion of a single row into the 'contract' table. The log also includes performance metrics for each step, such as execution times and the number of rows affected.

```
insert into contract (number, enterprise, date) values ('CSE00000000', 'Allianz', '2017-03-11')
```

```
[2025-04-25 19:14:47] 正在连接到 project@10.16.173.207 [2]...
[2025-04-25 19:14:48] 使用批处理模式, 最多 1000 条 INSERT/UPDATE/DELETE 语句
[2025-04-25 19:14:48] 运行 C:\Users\Chaos Z\For Study\Principle of Database Systems\Project\Project1\contract.sql
[2025-04-25 19:14:48] insert into contract (number, enterprise, date) values ('CSE00000000', 'Allianz', '2017-03-11')
[2025-04-25 19:15:09] 20秒310毫秒 中有 1000 行受到影响
[2025-04-25 19:15:27] 17秒289毫秒 中有 1000 行受到影响
[2025-04-25 19:15:43] 15秒804毫秒 中有 1000 行受到影响
[2025-04-25 19:15:58] 14秒903毫秒 中有 1000 行受到影响
[2025-04-25 19:16:11] 12秒850毫秒 中有 1000 行受到影响
[2025-04-25 19:16:24] 12秒932毫秒 中有 1000 行受到影响
[2025-04-25 19:16:37] 12秒584毫秒 中有 1000 行受到影响
```

## H2 Database:



```
[2025-04-26 14:08:31] 正在连接到 H2...
[2025-04-26 14:08:31] 使用批处理模式, 最多 1000 条 INSERT/UPDATE/DELETE 语句
[2025-04-26 14:08:31] 运行 C:\Users\Chaos Z\For Study\Principle of Database Systems\Project\Project1\contract.sql
[2025-04-26 14:08:31] insert into contract (number, enterprise, date) values ('CSE0000000', 'Allianz', '2017-03-11')
[2025-04-26 14:08:38] 4秒908毫秒 中有 1000 行受到影响
[2025-04-26 14:08:43] 4秒744毫秒 中有 1000 行受到影响
[2025-04-26 14:08:47] 4秒504毫秒 中有 1000 行受到影响
[2025-04-26 14:08:52] 4秒474毫秒 中有 1000 行受到影响
[2025-04-26 14:08:56] 3秒883毫秒 中有 1000 行受到影响
[2025-04-26 14:09:00] 3秒719毫秒 中有 1000 行受到影响
[2025-04-26 14:09:03] 2秒569毫秒 中有 1000 行受到影响
[2025-04-26 14:09:06] 2秒819毫秒 中有 1000 行受到影响
[2025-04-26 14:09:08] 2秒227毫秒 中有 1000 行受到影响
[2025-04-26 14:09:11] 2秒371毫秒 中有 1000 行受到影响
```

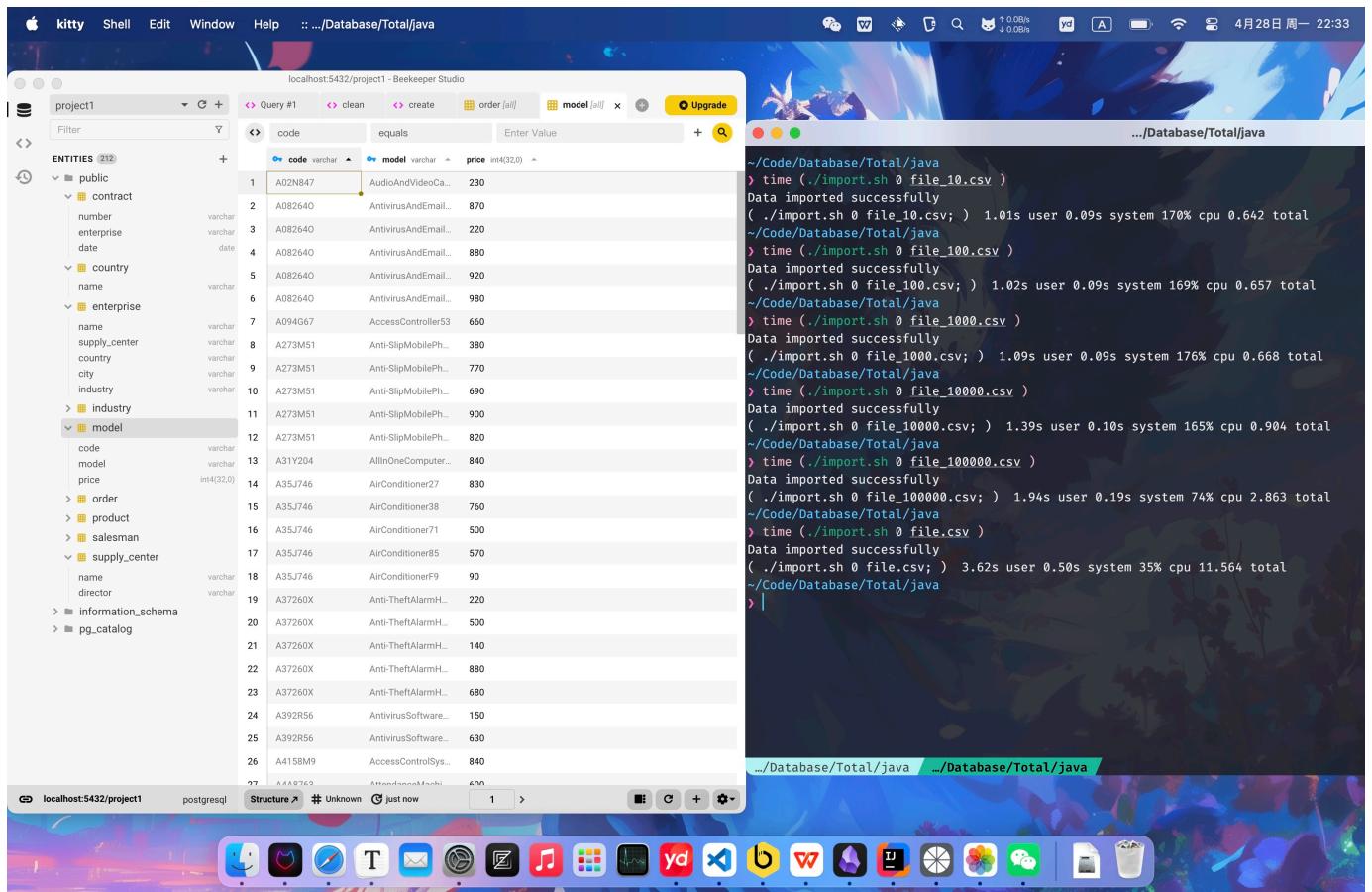
At the beginning of the import operation



```
[2025-04-26 14:10:00] 1秒513毫秒 中有 1000 行受到影响
[2025-04-26 14:10:08] 1秒513毫秒 中有 1000 行受到影响
[2025-04-26 14:10:09] 1秒587毫秒 中有 1000 行受到影响
[2025-04-26 14:10:11] 1秒711毫秒 中有 1000 行受到影响
[2025-04-26 14:10:13] 1秒596毫秒 中有 1000 行受到影响
[2025-04-26 14:10:14] 1秒475毫秒 中有 1000 行受到影响
[2025-04-26 14:10:16] 1秒496毫秒 中有 1000 行受到影响
[2025-04-26 14:10:18] 1秒615毫秒 中有 1000 行受到影响
[2025-04-26 14:10:19] 1秒522毫秒 中有 1000 行受到影响
[2025-04-26 14:10:19] 摘要: 在 1分钟48秒191毫秒中50,000/50,000 条语句已执行 (文件中有 4,998,090 个字符)
```

At the end of the import operation

## Import data with different data volumes



Size	10	100	1000	10000	100000	500000
Time	0.642s	0.657s	0.668s	0.904s	2.863s	11.564s