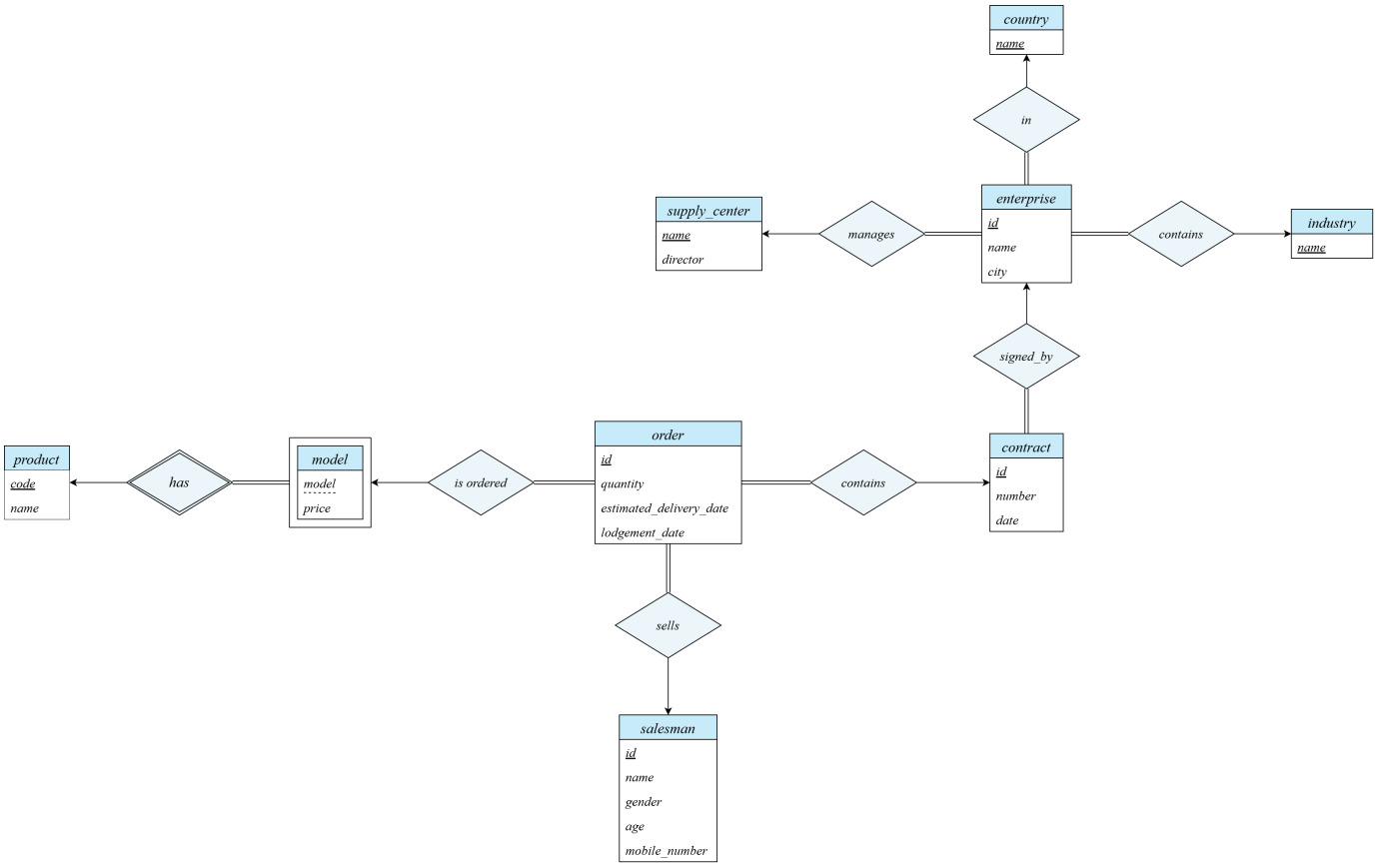
CS307 2025 Spring Project1 Report

Basic Information

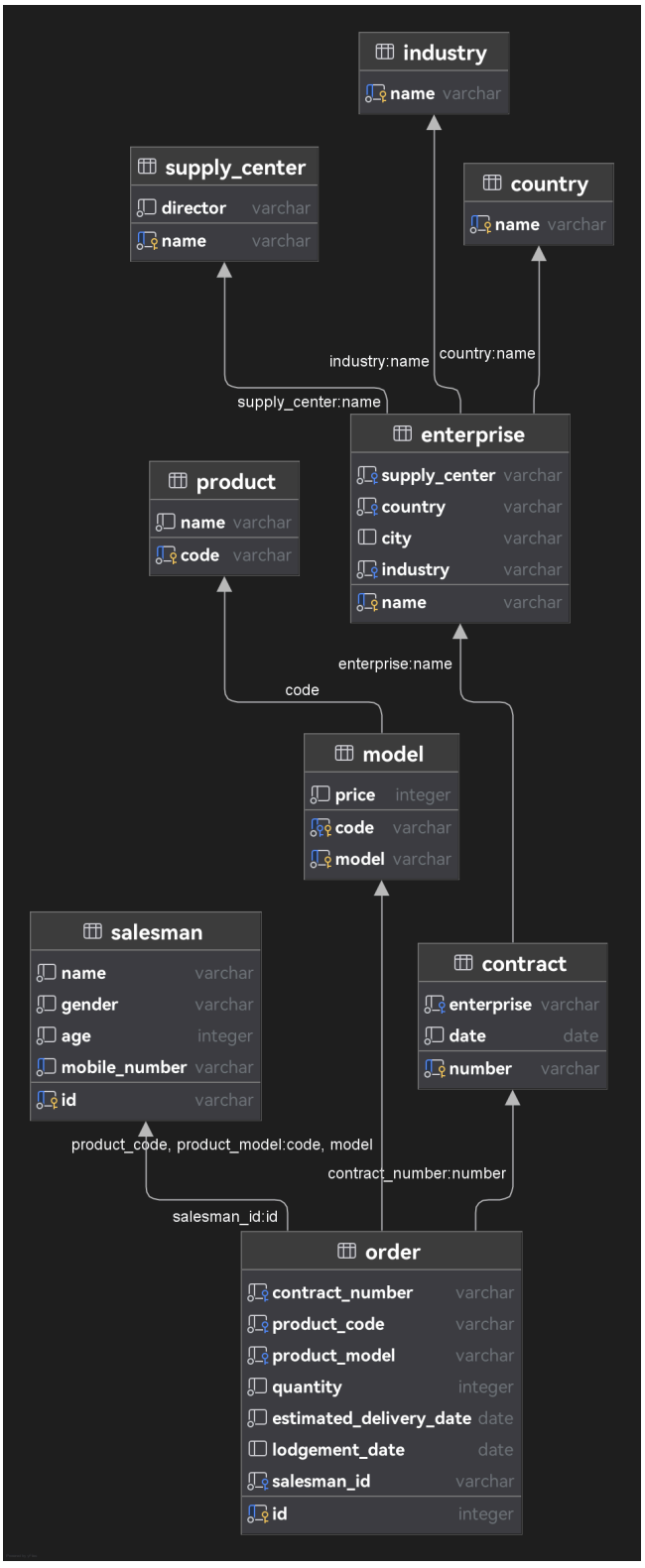
Name	Student ID	Lab Session	Contribution	Contribution Pct.
Zhu Mingyu	12312125	Wednesday 5-6	Data Import	40%
Zhou Liangyu	12312706	Wednesday 5-6	E-R Diagram, Database Design	60%

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 firstargument 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.psq' 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.

```sql
-- How many salesmen are there for each gender?
select gender, count(gender)
from salesman
group by gender;

-- How many companies are there in each supply center?
select supply_center, count(supply_center)
from enterprise
group by supply_center;

-- How many salesmen are there in each supply center?
select supply_center, count(distinct salesman_id)
from "order" o
         join contract c on c.number = o.contract_number
         join enterprise e on e.name = c.enterprise
group by supply_center;

-- How many salesmen are there within a given age range (lower and upper bounds)?
select count(age)
from salesman
where age > 40
  and age < 45;

-- How many countries are there in each supply center?
select supply_center, count(distinct country)
from enterprise
group by supply_center;

-- Given a country name, list all company names and their respective industries.
```

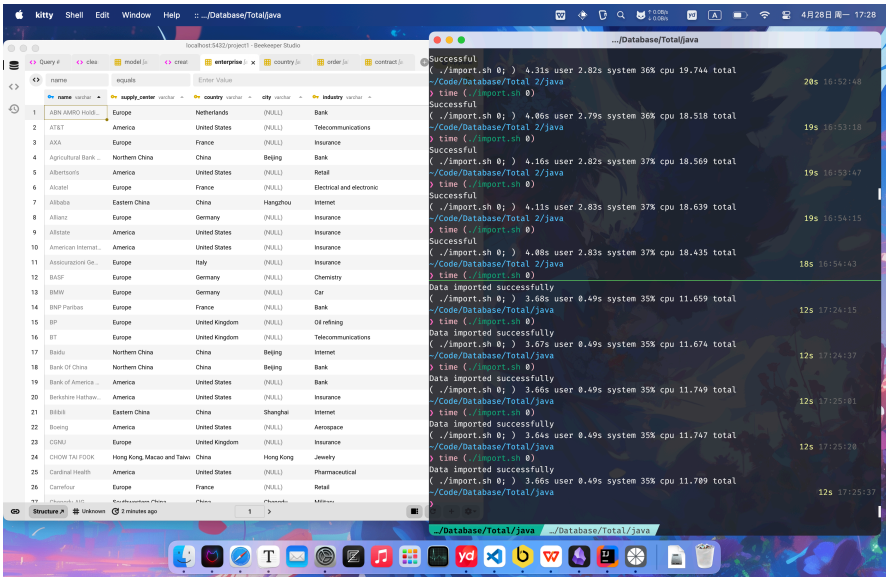
```sql
select name, industry
from enterprise
where country = 'United Kingdom';

-- Given a product code, list all product models and their corresponding unit prices.
select model, price
from model
where code = 'H7Y6824';

-- Given a contract number, list all order details in the contract, including (product model, order
-- quantity, salesmen name, and lodgement_date).
select product_model, quantity, s.name as "salesman name", lodgement_date
from "order" o
         join contract c on c.number = o.contract_number
         join salesman s on s.id = o.salesman_id
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

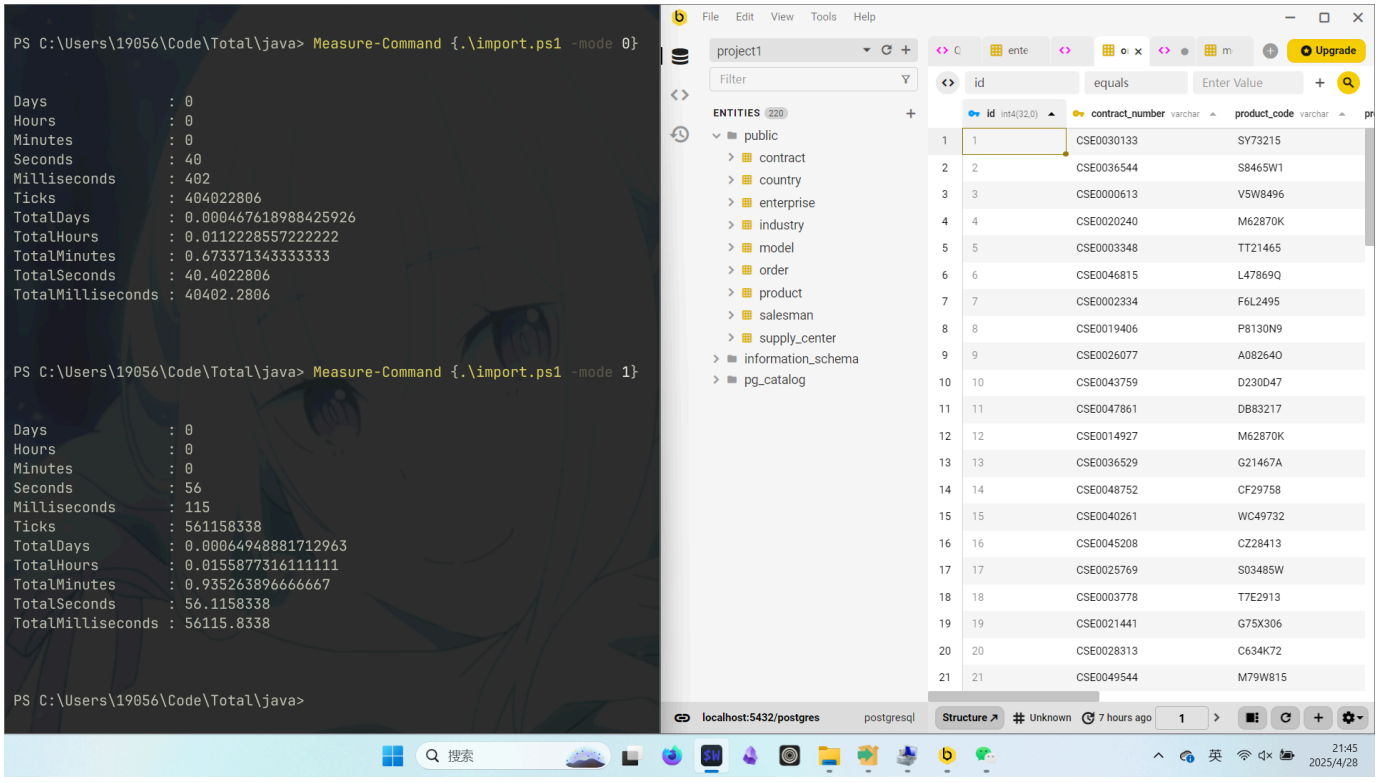
- Different ways to import data
  - import data from csv file to database directly(way 0)
    - Java: CsvToDB.java
    - Python: csv_to_DB.py
  - create sql file by proccssing the csv file, and then import data from sql file to database(way 1)
    - Java: CsvToSql.java, SqlToDB.java
    - Python: csv_to_sql.py, 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 proccessing, but way 1 need to create a new sql file, so it takes more time.

Data import across multiple systems: Windows, macOS, Linux

Windows:



Test environment: Windows 11, Powershell

Procedures:

- Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

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

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

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

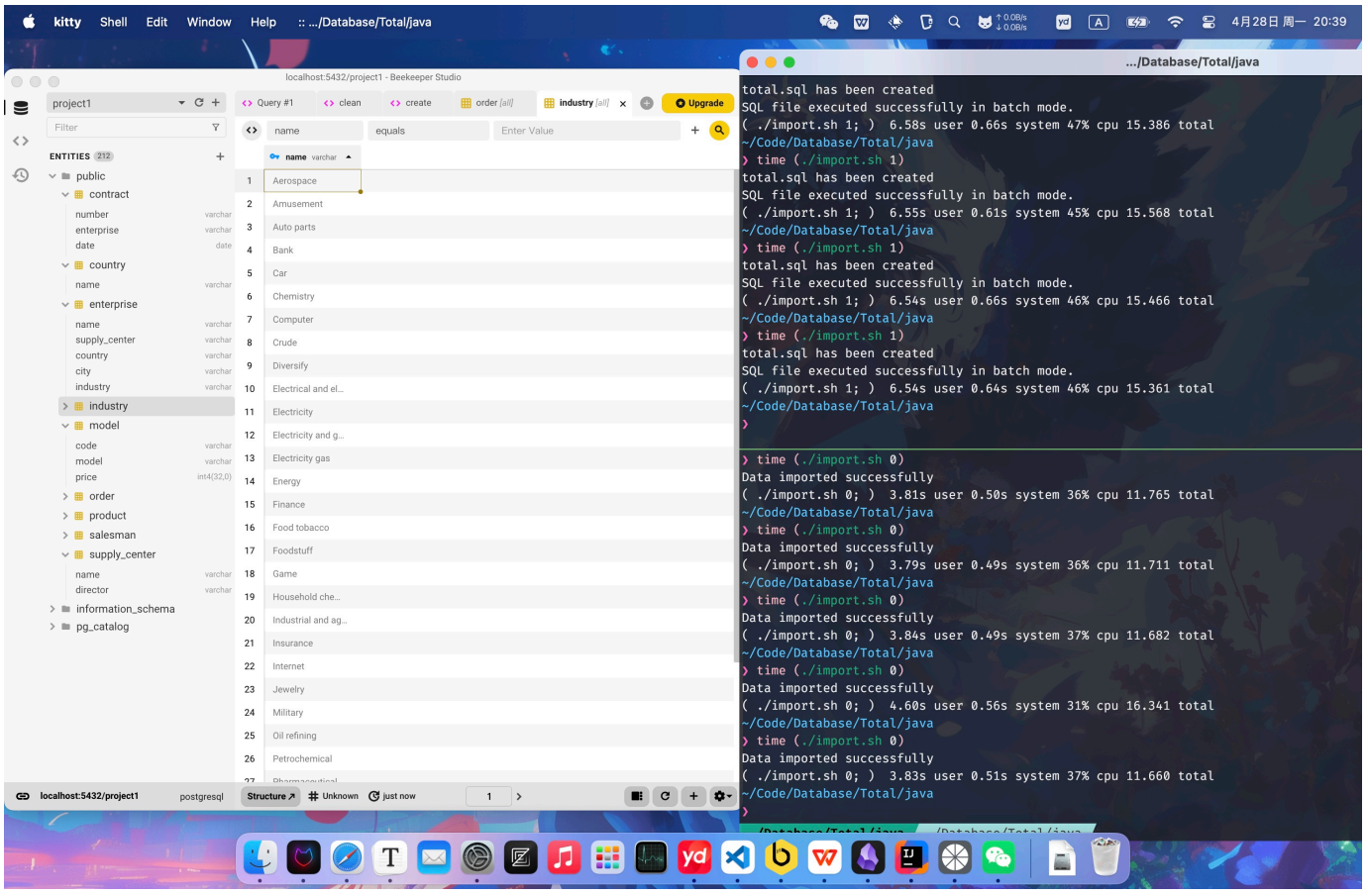
- Measure-Commond {.\import.ps1 -mode 1}

  > Use Measure-Commond 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

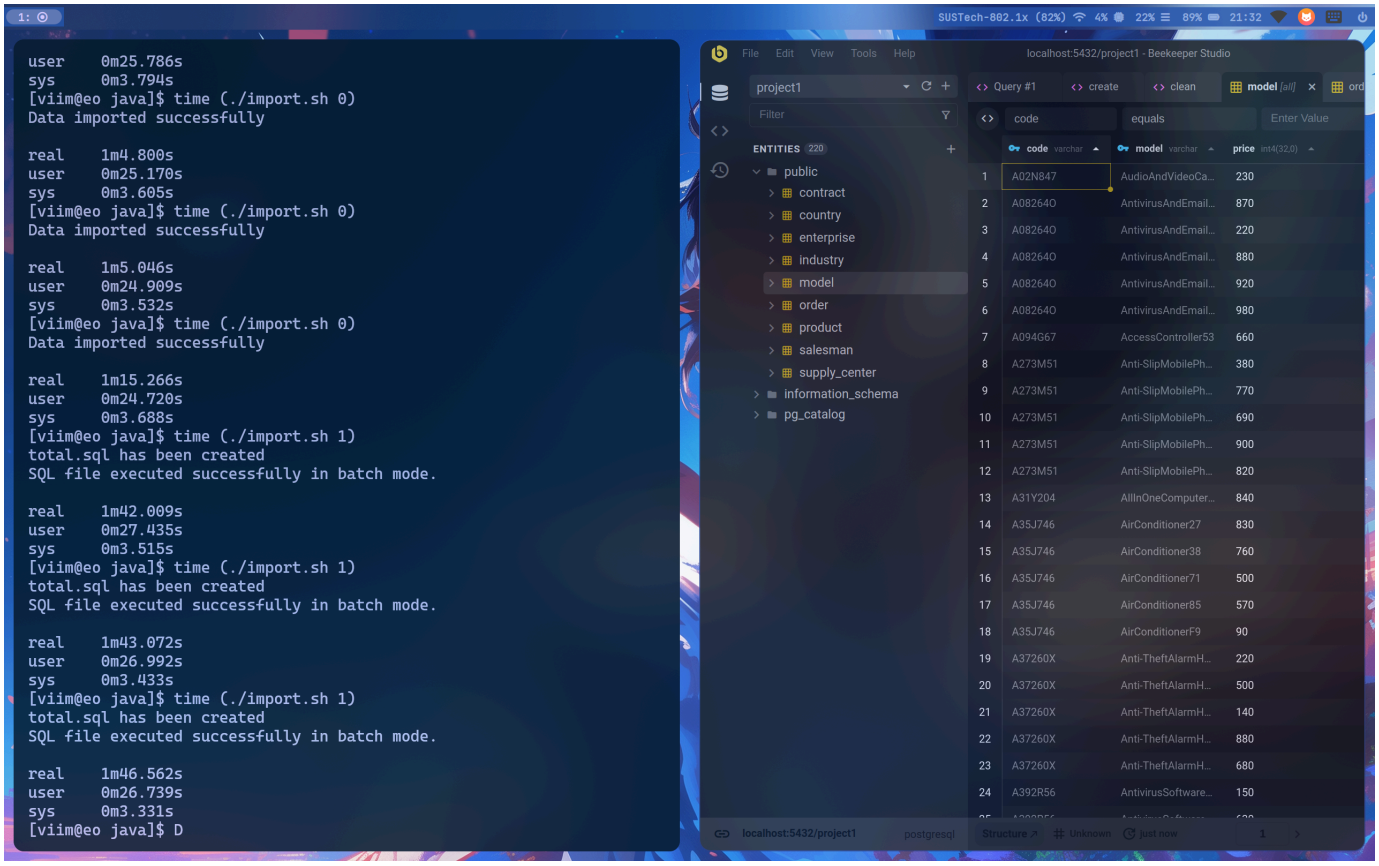
- Time(./import.sh 1)

  > 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:



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)

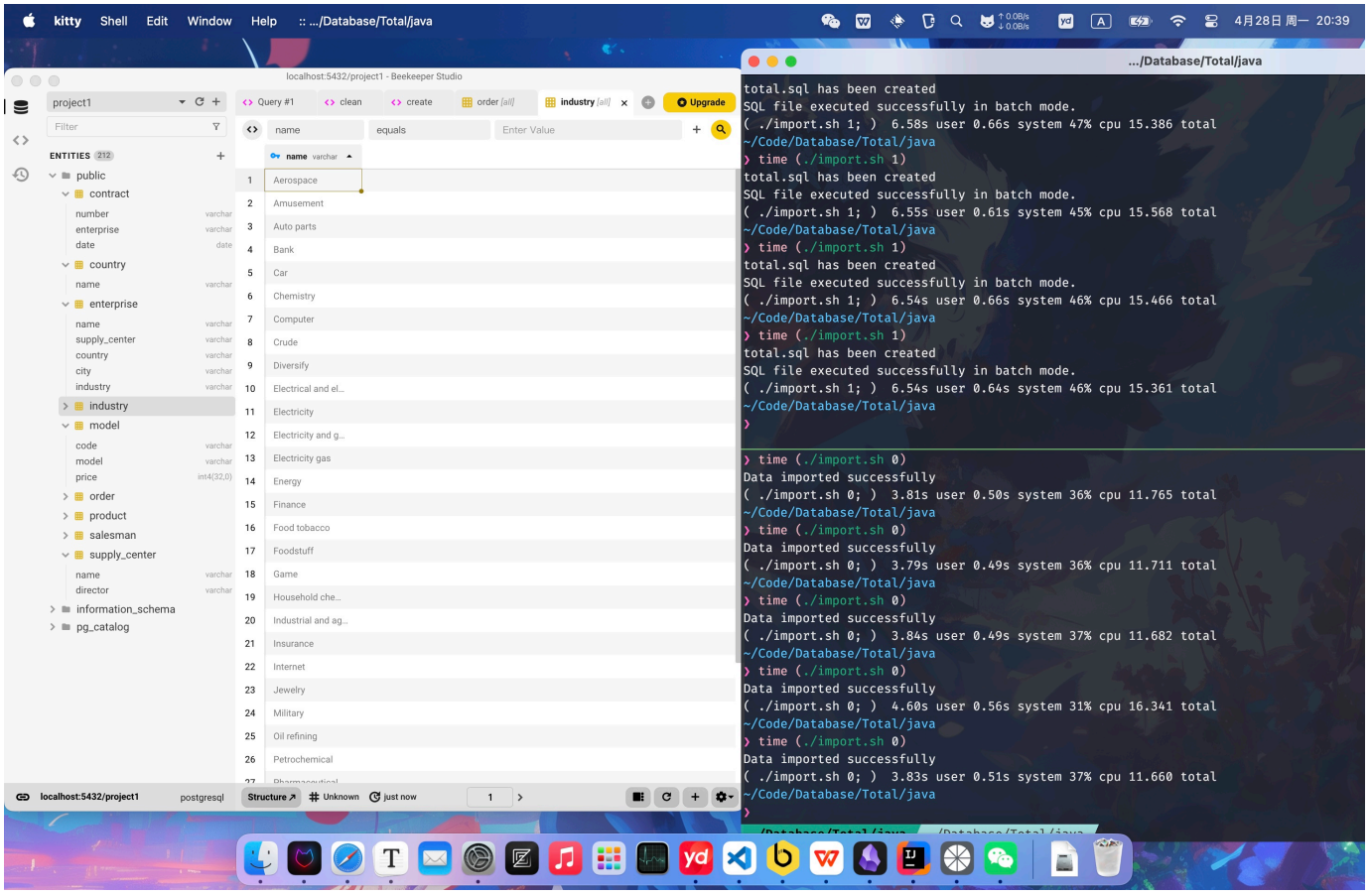
  > Use time to test the time of the script when the argument is 1 which means the java program will import data from the csv file to a sql file and then import data from the sql file to the database

Time cost

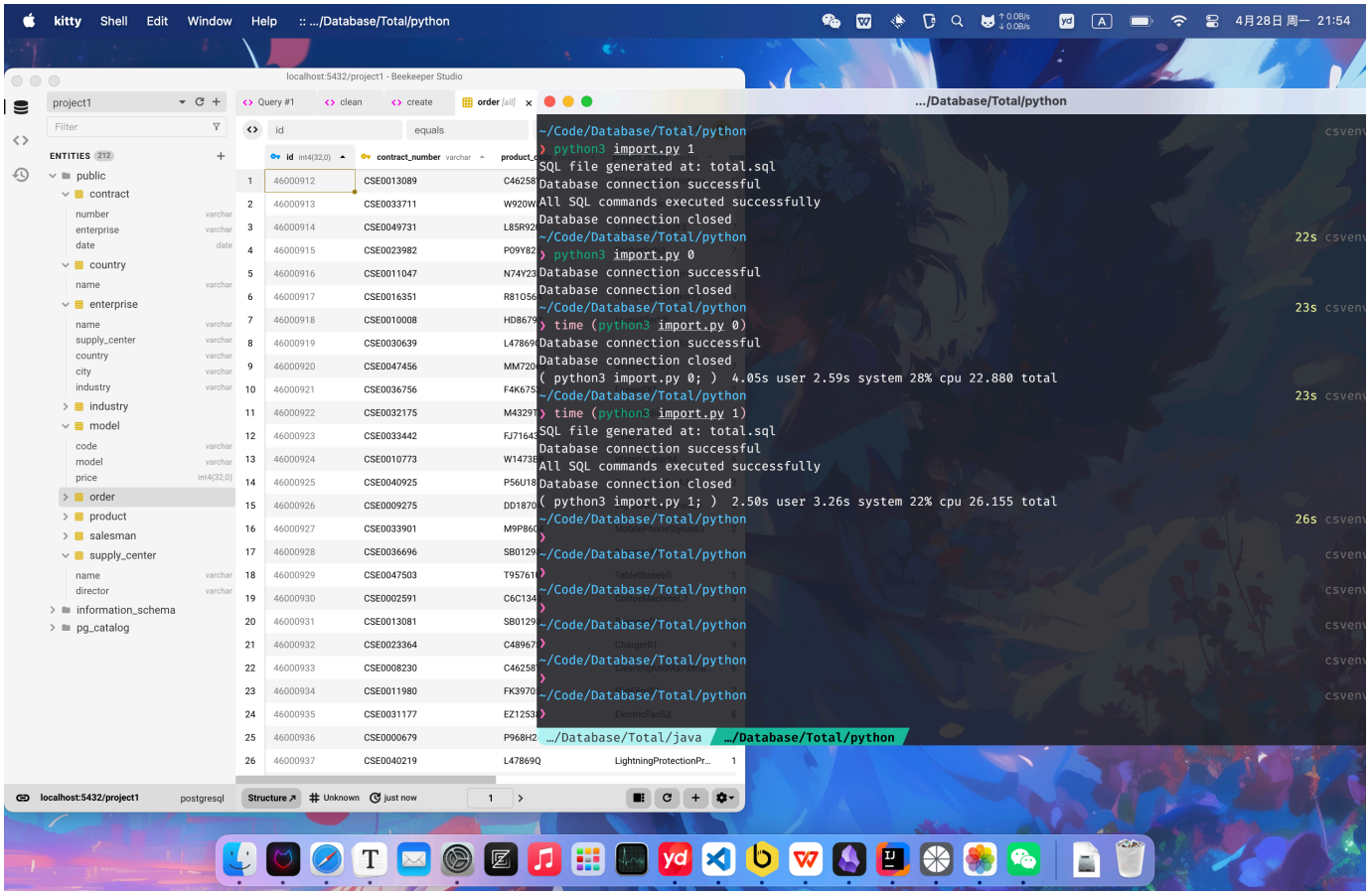
Test	1	2	3
way 0	64.800s	65.046s	75.266s
way 1	102.009s	103.072s	106.562s

Data import using various programming languages: Java, Python

Java:



Python:

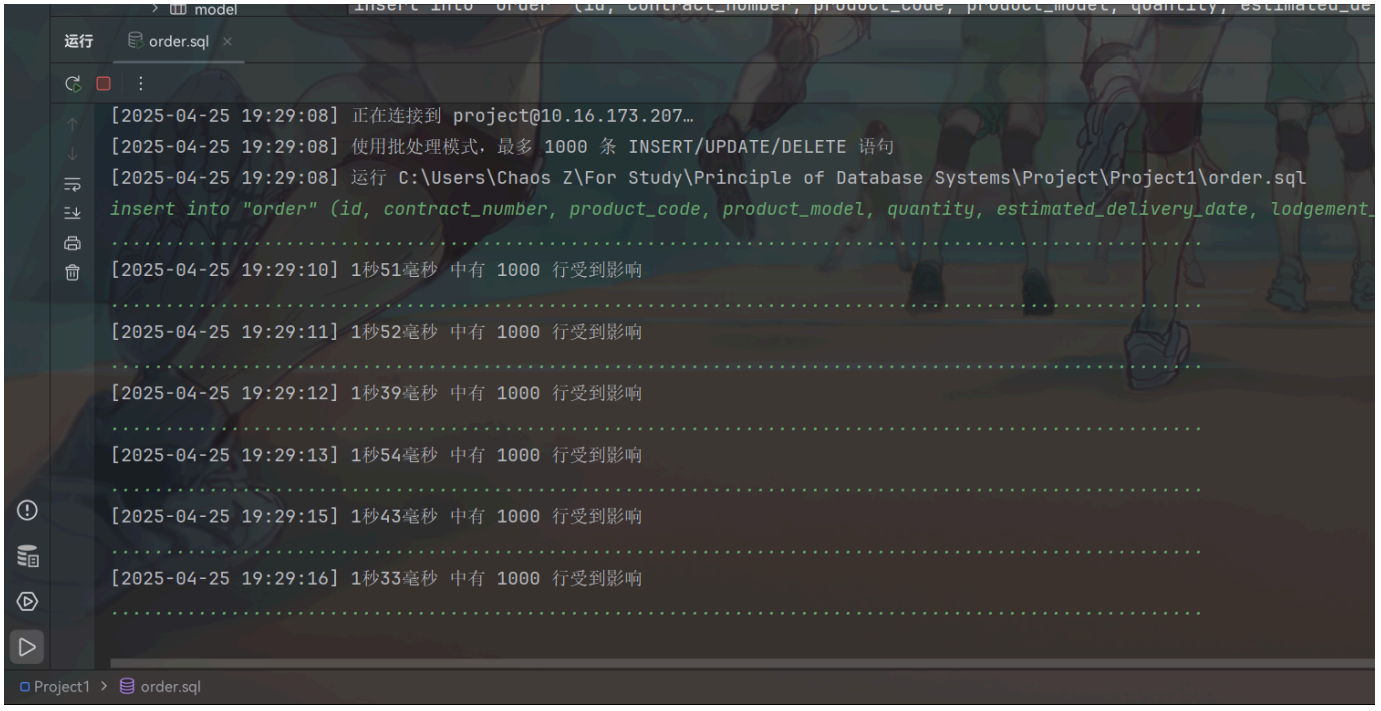


Experiment with other databases: SQLite, Apache Derby, H2 Database

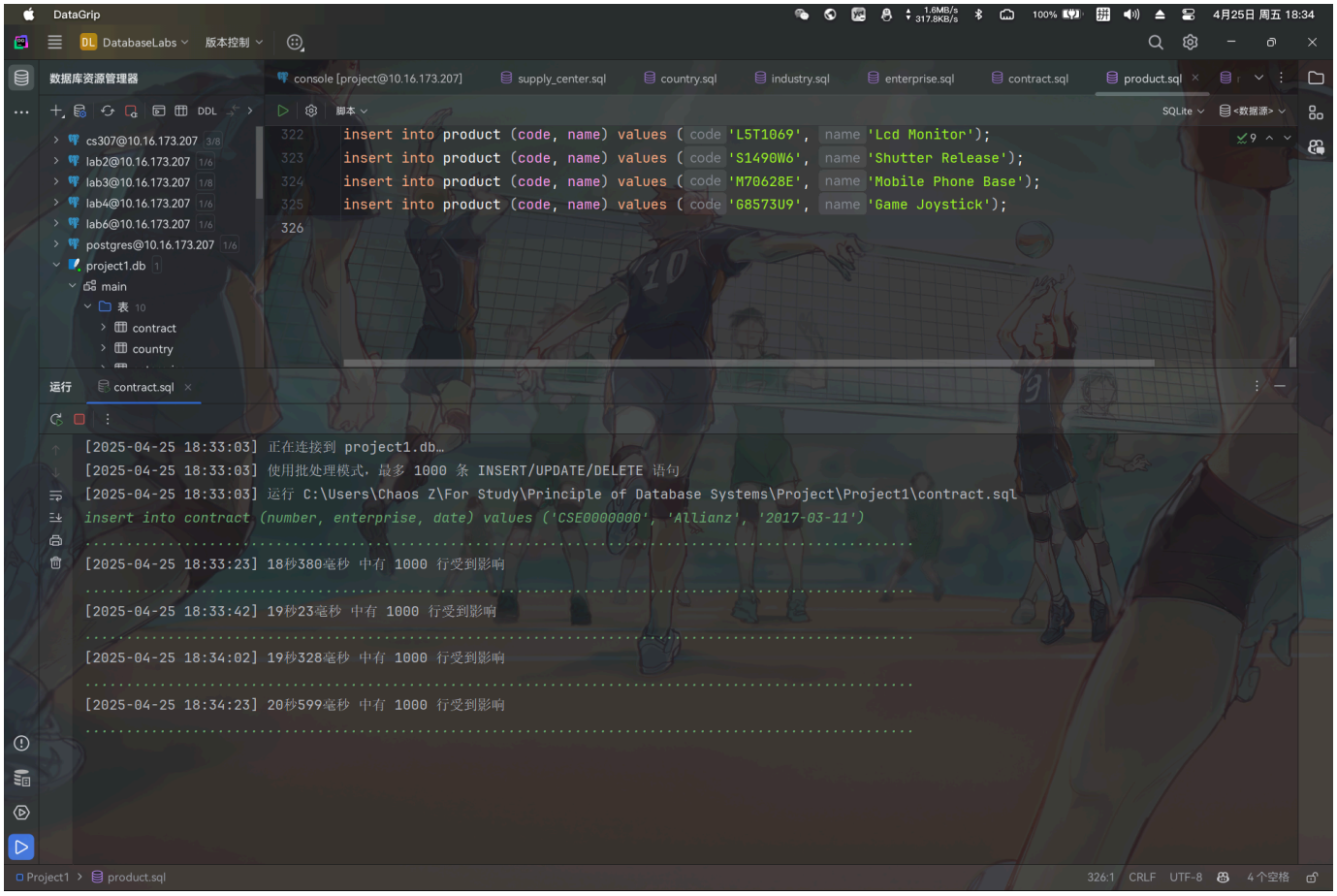
The following data is based on the `project` database with an identical structure, executing `insert` statements on the `contract` table. The databases are deployed on an ARM-based Ubuntu server with Hi3798MV100(Cortex-A7 @ 1.5GHz ×4) CPU and 512MB RAM.

Name	Feature	Storage	Standards	Performance(avg. time per 1000 `insert`)
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 `insert`

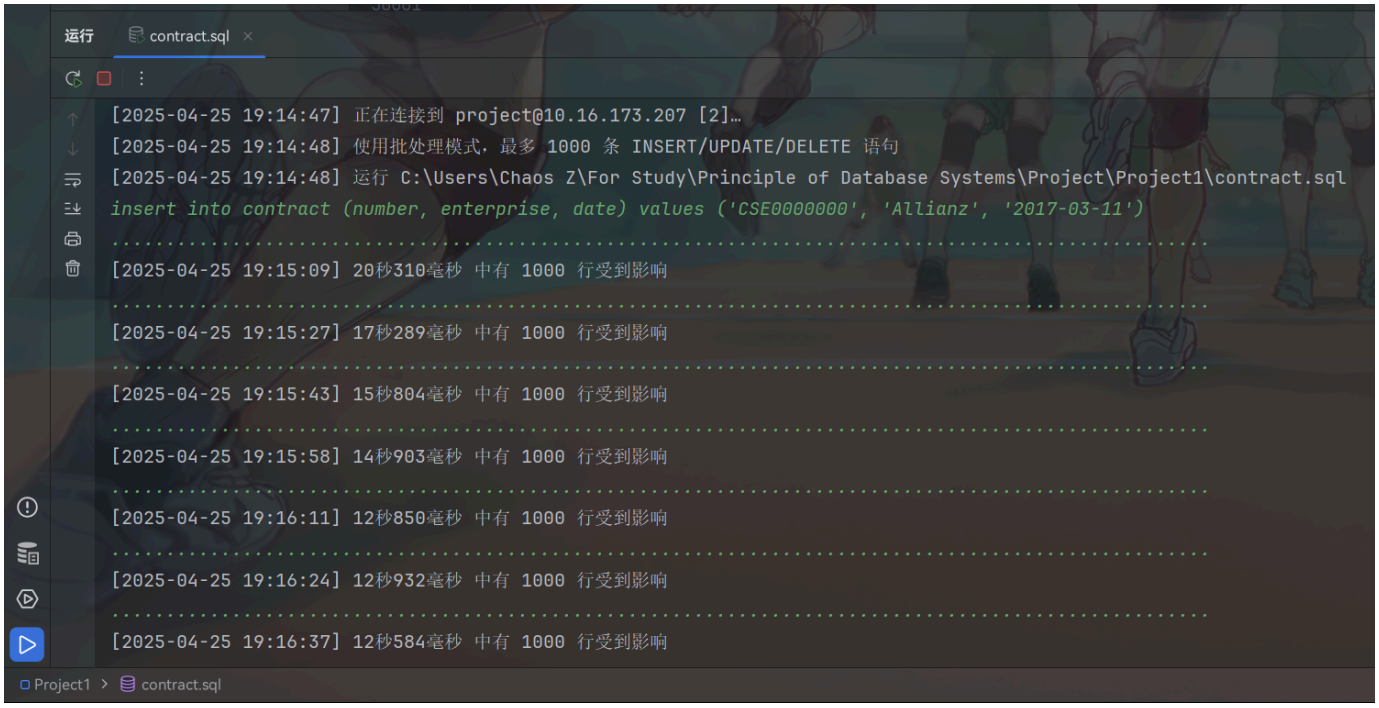
PostgreSQL:



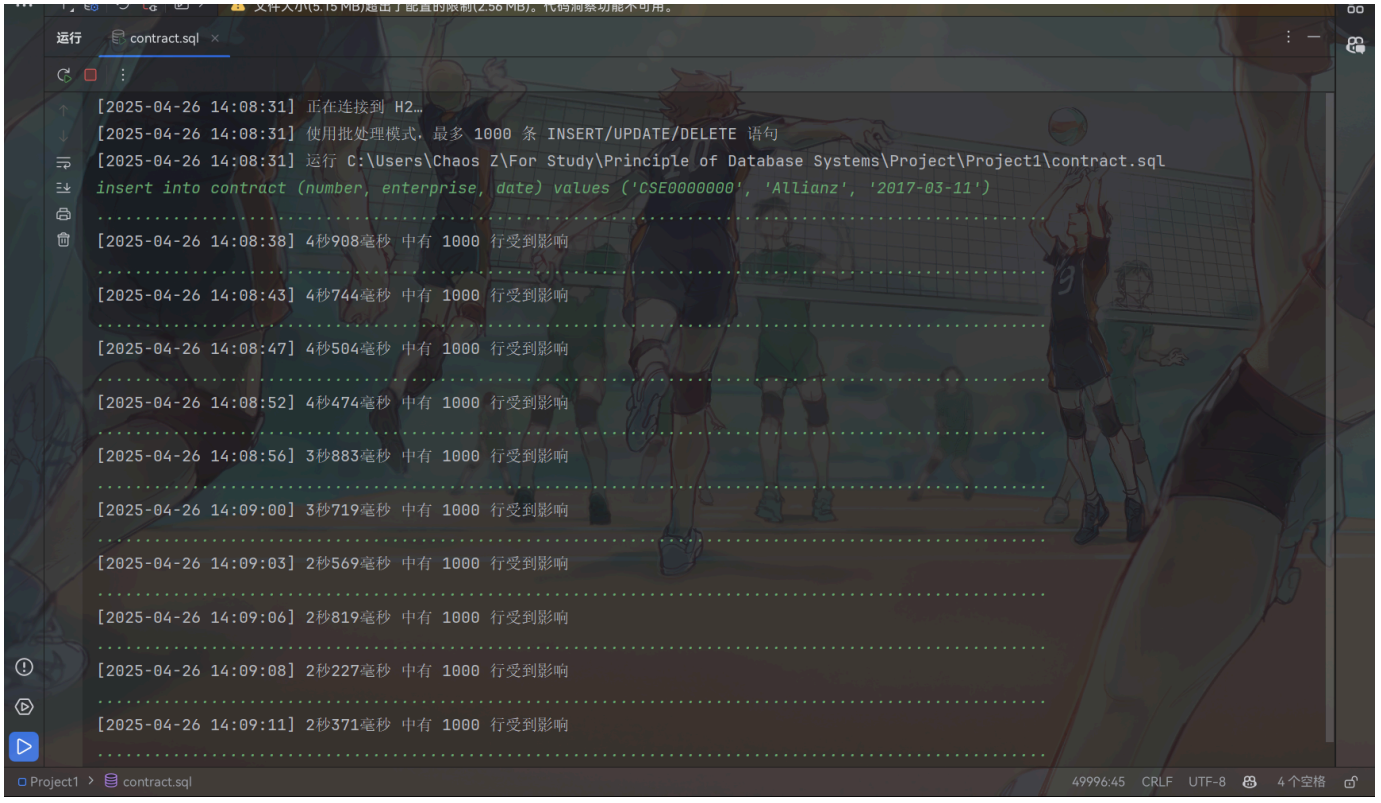
SQLite:



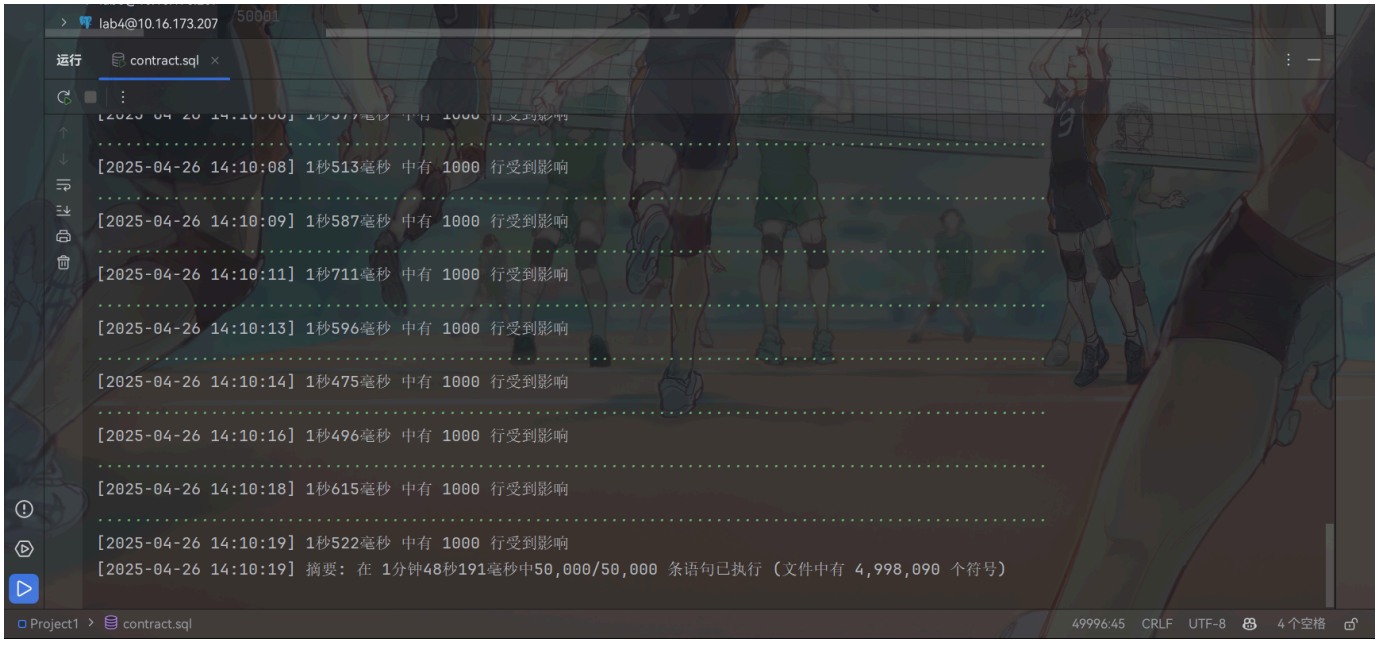
Apache Derby:



H2 Database:

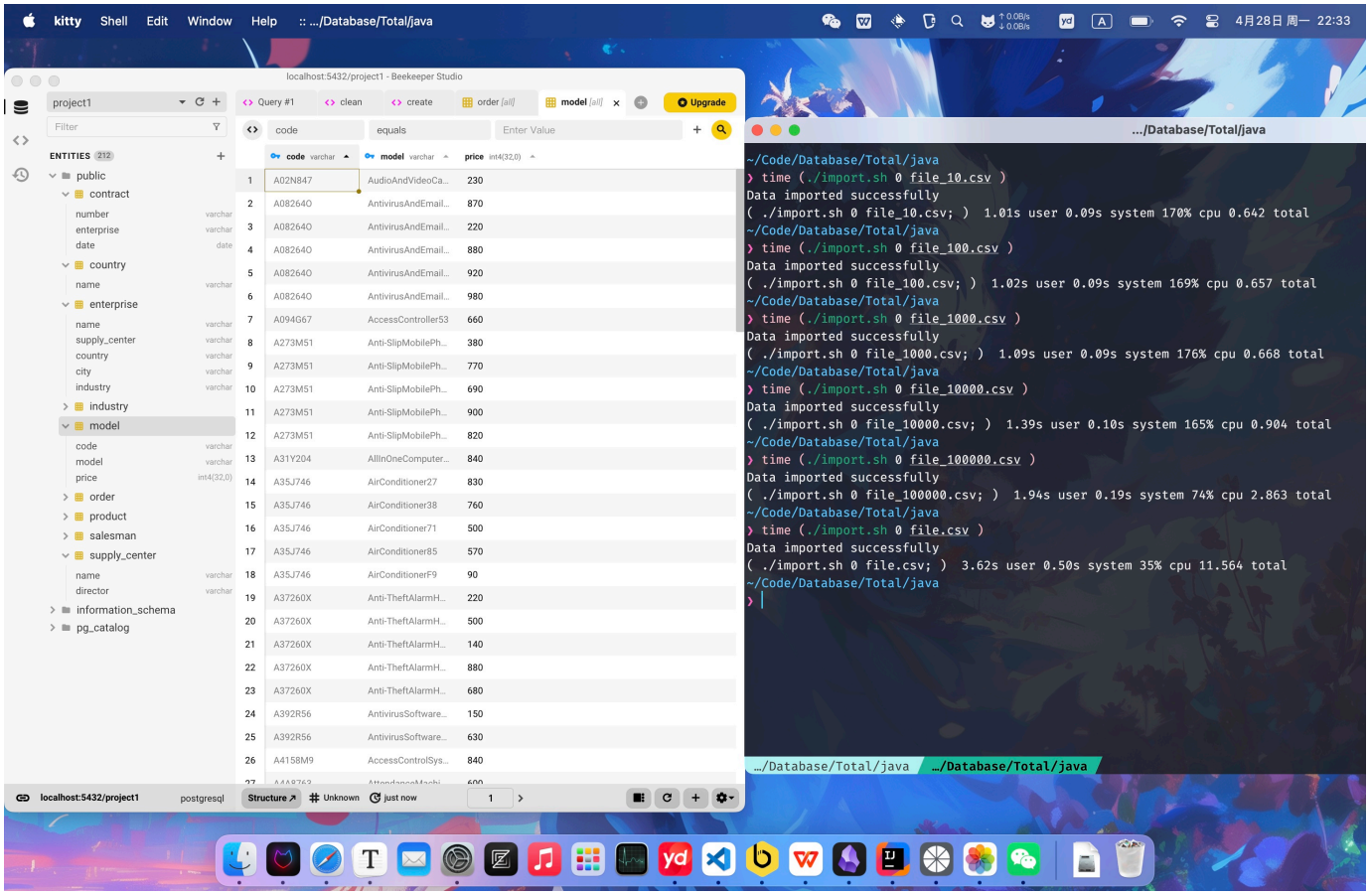


At the beginning of the import operation



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