# CS307 Principle of Database System

# Project 1 Report

**Group Number: 212**

**Group Members:**
YUNWANG CHEN - Email: chenyw2021@mail.sustech.edu.cn - SN: 12110119-Contribution:50%
FEIYANG ZHENG - Email: zhengfy2021@mail.sustech.edu.cn - SN: 12110616-Contribution:50%
**Date of Submission:** April 29, 2024
**Total number of pages: 9+1 page of appendix**

In executing our project tasks, we've combined traditional database design principles with innovative approaches to data handling and system integration. Our contributions are detailed below:

- **ER Diagram (30%):** Chen contributed 20%; Zheng contributed 10%.
  - *Innovation:* Utilized advanced design principles like generalization and specialization, as recommended in our course textbook, enabling a more abstract and scalable design.
- **Relational Database Design (30%):** Chen contributed 15%; Zheng contributed 15%.
  - *Innovation:* Developed a design that follows strict normalization forms to ensure data integrity.
- **Data Import (10%):** Entirely managed by Zheng.
- **Data Accuracy Checking (15%):** Executed by Chen.
  - *Note:* Our meticulous data cleaning process, particularly our deduplication efforts on bus and station data, significantly enhances the quality of the dataset. We urge the teaching assistants to consider our proactive measures during evaluation to circumvent any potential deductions due to oversight.
- **Advanced Requirements (15%):** Primarily addressed by Zheng, with supplementary contributions from Chen.
  1. *Language Proficiency:* Implemented import scripts in both C++ and Python to leverage the strengths of each language.
  2. *Cross-Platform Capability:* Ensured script compatibility and performance across macOS, Windows, and Linux.
  3. *Optimization Techniques:* Employed C code and scripting automation to streamline the data import process. Benchmarks were established to gauge performance gains.
  4. *Database Diversity:* Explored the use of MySQL server, assessing its features against PostgreSQL.

Overall, we believe it is fair to summarize our contributions a 50%-50% division, reflecting an equitable distribution of workload and a collaborative spirit.

# 1 ER Diagram

For the ER diagram, we utilized Lucidchart as our tool to design it.

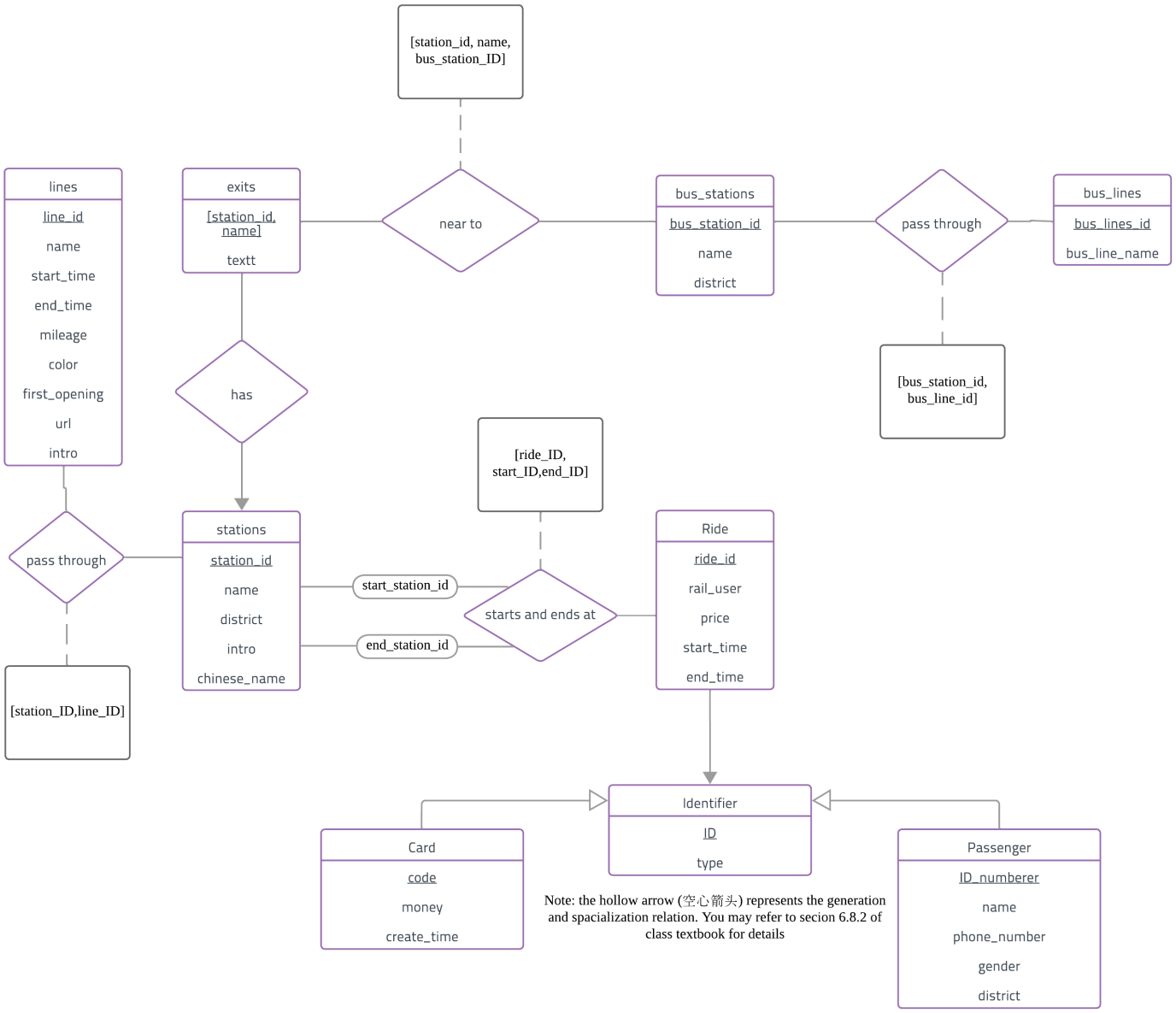


Figure 1: ER diagram reflecting the database design for the transportation management system.

The diagram adheres to standard ER diagram conventions, with entities represented as rectangles, relationships as diamonds, and attributes as ovals. Specialization is depicted with a hollow arrow, following the notation described in section 6.8.2 of our class textbook and allowing future expansion for other types of user like WeChat QR code user, Alipay user or bank card users. This structure of the ER diagram facilitates an understanding of the database's architecture and ensures proper normalization in the subsequent design phase.

# 2 Relational Database Design

Our database's table design adheres to the ER diagram and fulfills the project's requirements. Below is a snapshot of the ER diagram generated by DataGrip:

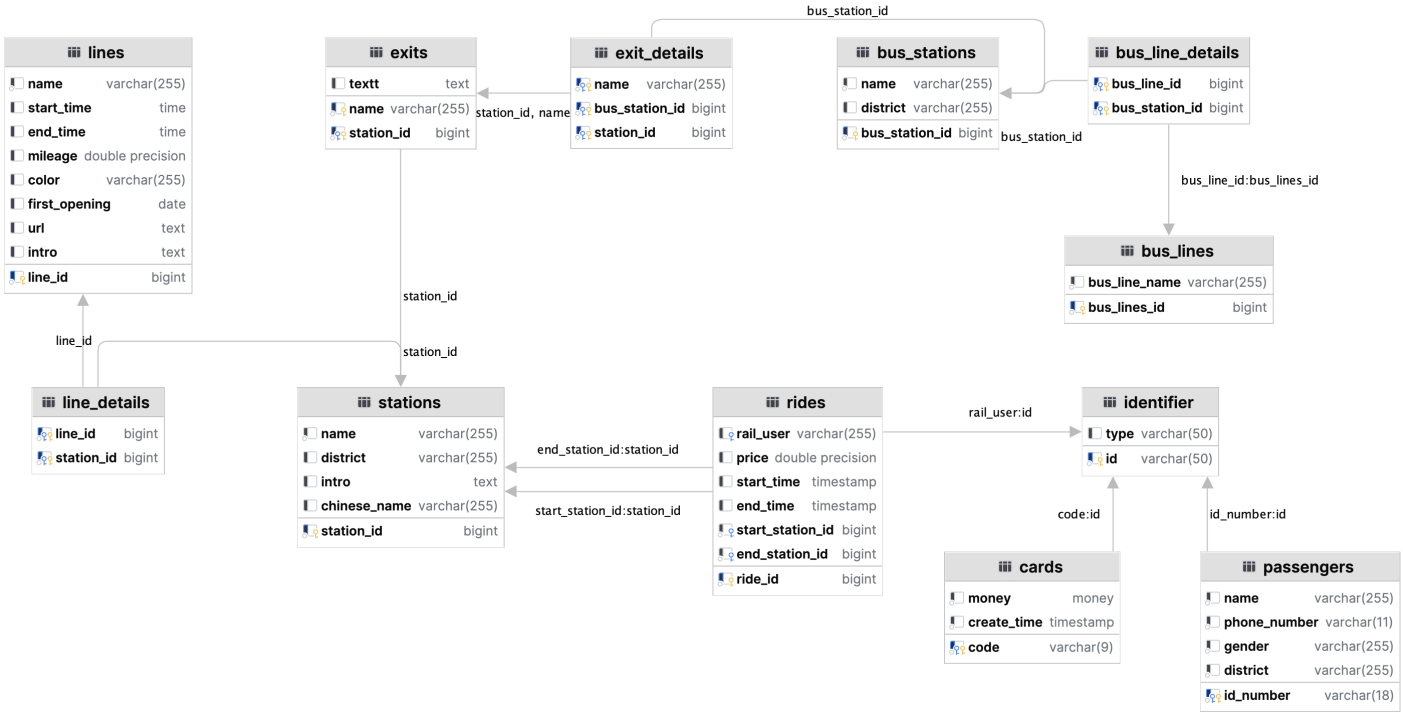


Figure 2: Database diagram generated by DataGrip.

## 2.1 Database Tables and Column Definitions

Our design comprises several interconnected tables, each serving a distinct aspect of the transportation network's data model:

**lines Table** stores details of transit lines, identified by a **line_id** (PK), along with **name**, **start_time**, **end_time**, **mileage**, **color**, **first_opening**, **url**, and **intro**. It has a foreign key (**line_id**) referenced in the **line_details** table.

**line_details Table** contains records linking lines with the stations they serve, with **line_id** (FK from lines) and **station_id** (FK from stations). It effectively joins the **lines** and **stations** tables.

**stations Table** lists transit stations, each with a unique **station_id** (PK), **name**, **district**, **intro**, and **chinese_name**. The **station_id** is referenced as a foreign key in the **rides**, **exit_details**, and **line_details** tables.

**rides Table** logs individual trips, captured by a **ride_id** (PK), with foreign keys for **start_station_id** and **end_station_id** linking to the **stations** table and **rail_user** linking to the **identifier** table. It also includes **price**, **start_time**, and **end_time**.

**bus_lines Table** details bus lines within the transit system, identified by a **bus_lines_id** (PK) and includes **bus_line_name**.

**bus_stations Table** contains details of bus stations, identified by a **bus_station_id** (PK), with **name** and **district**.

**bus_line_details Table** connects **bus_lines** with **bus_stations**, containing **bus_line_id** and **bus_stati** (both FKs).

**exits Table** (assuming a distinct table from **exit_details**) has a composite primary key of **name** and **station_id** and includes a column **textt**.

**exit_details Table** provides a mapping of exits within stations to locations and transport options. This table has a composite key made up of **name**, **station_id**, and **bus_station_id**. It references **station_id** (FK) and details nearby bus stations with **bus_station_id** (FK).

**identifier Table** centralizes identification for passengers and payment cards, with an **id** (PK) and **type**, linking to the **rides** and **cards** tables via foreign keys.

**cards Table** oversees people travelled via cards, identified by a **code** (PK), including columns for **money** and **create_time**. The **code** is also a foreign key in the **identifier** table.

**passengers Table** contains details of passengers using the transit system, with **id_number** (PK), **name**, **phone_number**, **gender**, and **district**.

### 2.2 Database Rationality and Project Compliance

The database design is rigorously structured to meet the three normal forms:

- All entities are broken down to their simplest form with no repeating groups, achieving the **First Normal Form (1NF)**.
- Every non-primary key attribute is fully functionally dependent on the primary key, fulfilling the **Second Normal Form (2NF)**.
- There are no transitive dependencies for non-primary key attributes, ensuring the **Third Normal Form (3NF)**.

Furthermore, the design adheres to additional detailed requirements set forth by the project mandate, ensuring no loss of data integrity and enabling effective data management within the transportation network. The relational model is also poised for adaptability, ready to accommodate future enhancements or modifications to the system with minimal rework.

Please refer to the attached SQL file named `ReformatDatabase.sql` for the DDL (Data Definition Language) statements for all the tables created. This file also includes data cleaning processes and database building process

## 3 Data Import

We used various scripts for data importation, each fulfilling specific tasks. Detailed descriptions of these scripts, along with their authors and functionalities, can be found in the project documentation.

### 3.1 Basic Requirements

#### 3.1.1 Folder structure of source code

```
root
|-- CS307 Project 1.pdf # project requirement from blackboard
|-- Cpp
|   |-- CMakeLists.txt
|   |-- data.sh # Only turn original json to CSV
|   |-- load.sh # Load original file to datagrip as tables (need to run reformulate later)
|   |-- mysql.sh # Call MysQL to parse json to database
|   |-- mysqlclient.sh # Call mysql client to parse json to database
|   |-- postgresclient.sh # Call Postgres client to parse json to database
```

```
|   |-- remote.sh # deprecated
|   |-- setup.sh # Used to set up Postgres
|   |-- speed.sh # Call speed test for fake rides and parse json to database
|   `-- src
|       |-- copy.cpp # Postgres version data loader, old version, can work without json2csv.
|       |-- dataonly.cpp # Data converter, doing nothing more
|       |-- fakedata.cpp # Postgres version speed test, only loads fakerides.json, works onl
|       |-- json.hpp # json parser
|       |-- json2csv.hpp # File for loading json into csv,
will be used by both postgres and mysql
|       |-- mysql.cpp # MySQL version data loader, works both locally and remotely
|       |-- postgres.cpp # Postgres version data loader,  works only locally
|       |-- priliminary.cpp # Oldest version, use insert to add lines, has been deprecated
|       `-- speed.cpp # Postgres version speed test, only loads fakerides.json, works only l
|-- Data # The provided data with some manual modification
|   |-- cards.json
|   |-- lines.json
|   |-- passengers.json
|   |-- rides.json
|   `-- stations.json
|-- Diagram
|   |-- Database Project 1 ER Diagram (1).png # Diagram generated by Datagrip
|   `-- er_diagram.png # ER diagram drawn by ourseleves
|-- Python
|   |-- fakeData.py # Generate fake data to meet 1,000,000 data volume
|   |-- loadByPython.ipynb # Load data (please run data.cpp first, also need to run reformul
|   |-- project.md # Explain why we choose to use psycopg package in C and Python.
|   |-- requirements.txt # Requirements for python packages
|   `-- test.ipynb # test
|-- README.md # Please always read readme.md first
|-- SQL
|   |-- AccuracyCheckForRawData.sql # Check accuracy of data for task 3.2 without building d
|   |-- AccuracyCheckForRevisedDatabase.sql # Check accuracy of data for task 3.2 after buil
|   |-- Database System Concepts.pdf # Our official textbook
|   |-- ReformatDatabase.sql # Reformulate the loaded raw table into database in Postgresql
|   |-- mysql.sql # sql command used in mysql when connected to remote server
|   `-- raw.sql # sql commnad used in load.sh
`-- screenshotInDataGrip.png
```

#### 3.1.2 JSON Parsing

The JSON parsing process is shown in figure 3, `load.sh` will read the five JSON files provided by the guideline, parsing it into 10 csv files, and copy them into the database.

### 3.2 Data Cleaning

Data cleaning was a crucial step in ensuring the integrity and accuracy of the database. We encountered several types of inconsistencies and errors across various JSON files, which necessitated a systematic approach to cleaning. Here, we outline a few typical examples:

- **Whitespace and Formatting Errors:** Fields such as ‘busName‘ and ‘start_station‘ had extra-

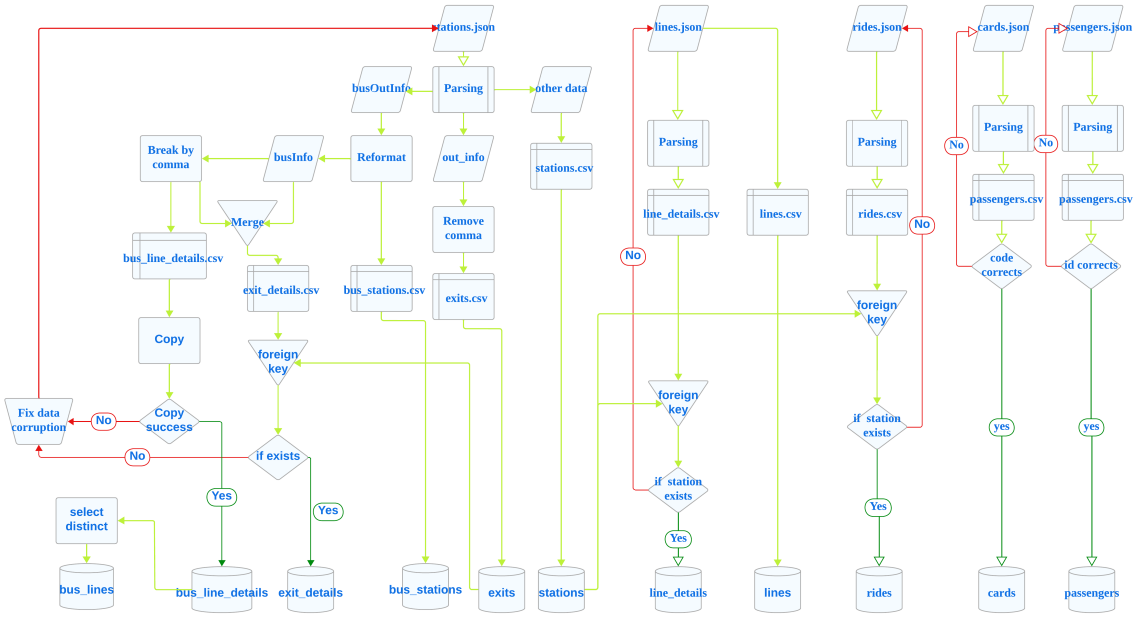


Figure 3: Parsing Process

neous whitespace and new line characters (e.g., ‘”Huilongpu station \n”‘), which were trimmed for uniformity.

- **Incorrect Data:** Some station names had incorrect district assignments or used inconsistent naming conventions (e.g., ”\西丽线” corrected to ” 福田区”).
- **Textual Inconsistencies:** Various naming inconsistencies, such as ”K578(右环)” corrected to “K578（右环）” to match the more commonly used format.
- **Unnecessary Information:** Certain ‘busInfo’ fields contained irrelevant or placeholder text, such as ‘” 此站暂无数据”‘ (meaning ”No data for this station”), which we removed.
- **Redundant Keywords:** Entries containing phrases like ‘”B 出入口”‘ (exit names suffixed with ”entry/exit”) were standardized by removing such suffixes.

We just list some major issues with the given data, and we have tried our best effort to handle both major and minor issues with a combination of manual review and automated scripts.

For a comprehensive review of all the data errors identified and the corresponding cleaning procedures, please refer to **readme.md**, which records all the error we found in raw data.

## 3.3 Data accuracy checking

Please see the **Appendix** at the end. And please refer to `AccuracyCheckForRevisedDatabase.sql` for detailed SQL command.

## 3.4 Advanced Requirements

We optimized our scripts and explored various data import methods to provide a comparative analysis of computational efficiencies. Our experiments included data imports across multiple systems using different programming languages and various databases, such as MySQL and PostgreSQL. Additionally, we dealt with different data volumes, from hundreds of thousands to millions of records.

### 3.4.1 Data Import Implements

There are mainly 3 means to load data into our database: shell scripts with C/C++ programs, offical clients and Python API. For PostgreSQL, we can use `load.sh`, which will set up environment variables and execute `C++` executable to parse json into csv, then call `COPY` in `<pqxx>` to copy the whole csv files into the database; or use `Data`, a C++ binary to only parse json into csv, then upload the csv using `COPY` from Python using `loadByPython.ipynb`. However, the `\copy` function is not available in the provided C++ API, and can only be called from `psql` offical client, thus the aforementioned method can only be used locally, although we could use `scp` or other data transfer to upload csv to the `\tmp` directory and load it into database from there, such an approach will make our server vulnerable to potential attacks. On the other hand, `<mysql.h>`, the C API provided by MySQL contains a functioin called `LOAD DATA LOCAL INFILE`, which allows the program to directly interact with the server and upload the whole csv file to the server.

| Script Name | Author | Description |
|---|---|---|
| postgres.sh | Zheng | Import into PostgreSQL database locally using C++ API. |
| mysql.sh | Zheng | Import into MySQL database both remotely and locally using C API. |
| postgresclient.sh | Zheng | Import into PostgreSQL database both remotely and locally using `psql` client. |
| mysqlclient.sh | Zheng | Import into MySQL database both remotely and locally using `mysql` client. |
| speed.sh | Zheng | Import only `fakerides.json` (1,000,000 records) into PostgreSQL database locally using C++ API. |
| loadByPython.ipynb | Chen | Load raw data via Python. |

Table 1: Scripts for achieving advanced requirements

### 3.4.2 Script Optimization and Data Import Methods

- Explored efficiencies between `COPY` and `INSERT` commands.
- Integrated automation scripts in both C++ and Python for diverse loading strategies.

### 3.4.3 Multiple Systems

**Test Devices:**

- **Mac:** MacBook Pro 14-inch with M1 Pro, 32GB RAM, 4TB ROM.
- **Windows:** WSL with i9-12900HX, 16GB RAM, 1TB ROM.
- **Linux:** J1900, 4GB RAM, 512GB ROM.

#### 3.4.4 Database Server

For Linux system, we managed to deploy an Ubuntu server on J1900, a 4 core small form factor pc shown in figure 4. Using net port in the dormitory, this server can be accessed from any devices in campus. Both `PostgreSAL` and `MySQL` services are available. With this set up in hand, we can further explore data import from remote clients, which will be closer to actual scenarios. A detailed manual is provided in the `README.md` in the project.

Figure 4: Not a Real Apple

#### 3.4.5 Programming Languages Utilized

- Python
- C++
- Shell(zsh on Mac, bash on Ubuntu)

#### 3.4.6 Comparative Database Analysis

- PostgreSQL
- MySQL (used on the remote server and wsl)

#### 3.4.7 Data Volume Variations

- 100,000 rides data-set.
- Combined data-set of 100,000 rides with additional data.
- 1,000,000 synthetic rides data-set (`fakerides.json`).

#### 3.4.8 Results from Optimized Scripts

**Note:** Detailed results are consolidated in the table above, reflecting the performance metrics in seconds for the respective data import operations.

The tables encapsulate our comprehensive testing strategy, revealing insights into the interplay between systems, programming languages, and databases on data import operations.

From table 2 and the experiences when doing experiments, we can observe the following phenomenon:

| Database | System | Language | Data Volume | Script Used | Performance (CSV to DB when applicable) |
|---|---|---|---|---|---|
| PostgreSQL | WSL | C | 100,000 rides | Awful | 9s |
| PostgreSQL | Mac | Python | All data | loadByPython.ipynb | 0.831s (CSV to DB) |
| PostgreSQL | Mac | C | All data | load.sh | 1.186s, 0.869s (CSV to DB) |
| PostgreSQL | WSL | Python | All data | loadByPython.ipynb | 0.826s (CSV to DB) |
| MySQL | WSL | C | All data | mysql.sh | 6.765s (CSV to DB) |
| MySQL | WSL to J1900 | C | All data | mysql.sh | 11.647s (CSV to DB) |
| PostgreSQL | WSL | C | All data | load.sh | 1.114s, 0.732s (CSV to DB) |
| PostgreSQL | WSL | C | 1,000,000 fakerides | speed.sh | 3.433s, 0.914s (CSV to DB) |
| PostgreSQL | Mac | C | 1,000,000 fakerides | speed.sh | 4.309s, 2.002s (CSV to DB) |
| MySQL | WSL | Shell | All data | mysqlclient.sh | 6.957s (CSV to DB) |
| MySQL | WSL to J1900 | Shell | All data | mysqlclient.sh | 11.107s (CSV to DB) |
| PostgreSQL | WSL | Shell | All data | postgresclient.sh | 1.179s (CSV to DB) |
| PostgreSQL | Mac | Shell | All data | postgresclient.sh | 0.924s (CSV to DB) |
| PostgreSQL | WSL to J1900 | Shell | All data | postgresclient.sh | 11.434s (CSV to DB) |
| PostgreSQL | J1900 | Python | All data | loadByPython.ipynb | 10.067s (CSV to DB) |
| PostgreSQL | J1900 | C | All data | load.sh | 13.106s, 10.554s (CSV to DB) |
| PostgreSQL | J1900 | Shell | All data | postgresclient.sh | 14.509s (CSV to DB) |
| MySQL | J1900 | C | All data | mysql.sh | 13.029s, 10.342s (CSV to DB) |
| MySQL | J1900 | Shell | All data | mysqlclient.sh | 10.667s (CSV to DB) |

Table 2: Comprehensive Performance Results of Data Import Scripts Including CSV to Database Times

- The performance of `COPY` command is significantly more efficient than `INSERT`, even if the query is prepared in the first place. This is due to the fact that `INSERT` queries are executed line by line, while `COPY` can insert the whole table with batch processing.
- The remote upload speed is not significantly longer than the local execution done on J1900 itself, indicating the fact that the major bottle neck is caused by the performance of the host rather than internet connection.
- `PostgreSQL` out performs `MySQL` in all scenarios, but the performance gap was minor on J1900, indicating that `PostgreSQL` can better utilize the extra cores on the host machine.
- Mac is better on all data loading, while WSL is perform better in speed test for loading 1,000,000 fake rides data. This is because the time consumption is mainly consists of three parts: reading JSON, reformatting to csv, and load csv into database. In all data tests, parsing complicated JSON file like `stations.json` will make the first two parts taking up considerable amount of time; while in speed test, loading csv into database takes up the majority of the time. We assumed that such performance differences is probably caused by job assigning strategy of WSL and Mac.
- Loading speed will be significantly faster if the foreign keys statements are commented out when creating the table, for the server will save the time for checking if the foreign keys are valid. However, this could potentially introduce flawed data into our database, causing more trouble later.
- The performance differences of uploading csv files to database is minor, we proposed that the major bottle necks are in `PostgreSQL` and `MySQL` back-ends, not our programs.

# 4 Future Works

For future works, we will consider using the knowledge to redesign our database: introducing trigger functions to correct data so that the database can be more resilient to dirty data; granting different authority for different user so that we can share our database without risking getting hacked.

Due to time limits, we have yet to figure out how to compile the `MySQL` C++ loader on MacOS, we will consider fulfilling this task later.

We can only use `psql` to upload csv files at the moment, we might consider using python to implement a file transfer service for our csv using ftp.

We may consider using locally deployed `Llama 3 8b` to conduct data cleaning task.

# 5 Conclusion

In conclusion, our CS307 Principle of Database System Project 1 Report reflects the dedication and expertise of Group 212 members, Yunwang Chen and Feiyang Zheng. Through meticulous data handling, innovative database design, and thorough data accuracy checking, we have successfully navigated the complexities of database projects. Our collaborative efforts have not only met the project requirements but have also provided valuable insights into the principles of database design and implementation. We have learned the significance of attention to detail, problem-solving skills, and effective collaboration in ensuring the success of database projects. Moving forward, we aim to further enhance our database by implementing trigger functions for data correction and establishing different user authorities for enhanced security. Despite encountering challenges along the way, our project has been a testament to our commitment to excellence in database management.

# Q1: Station Queries

## – Q1.1: Number of Stations in Each District

| District | 宝安区 | 龙岗区 | 罗湖区 | 南山区 | 光明区 | 盐田区 | 坪山区 | 龙华区 | 福田区 |
|---|---|---|---|---|---|---|---|---|---|
| Number of Stations | 51 | 56 | 29 | 60 | 13 | 9 | 15 | 21 | 52 |

Table 1: The number of metro stations in each district as part of the database accuracy check.

## –Q1.2: Number of Stations on Each Line

| Line Name | 3 号线 | 14 号线 | 11 号线 | 8 号线 | 10 号线 | 7 号线 | 6 号线 | 16 号线 | 2 号线 | 5 号线 | 9 号线 | 20 号线 | 4 号线 | 12 号线 | 6 号线支线 | 1 号线 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of Stations | 31 | 18 | 19 | 10 | 24 | 27 | 27 | 24 | 32 | 34 | 32 | 5 | 23 | 33 | 4 | 30 |

Table 2: Distribution of metro stations across various lines.

## –Q1.3: Total Number of Stations

| Total Number of Stations |
|---|
| 306 |

Table 3: The total number of stations across all metro lines as reported in the system database.

# Q2: Passenger Gender Counts

| Gender | 男 | 女 |
|---|---|---|
| Count | 5069 | 4931 |

Table 4: Count of female and male passengers to verify gender data accuracy.

# Q3: Passenger Origin

| Region | Chinese Macao | Chinese Hong Kong | Chinese Taiwan | Chinese Mainland |
|---|---|---|---|---|
| Count | 1010 | 970 | 1039 | 6981 |

Table 5: Number of passengers from Mainland China, Hong Kong, Macau, and Taiwan.

# Q4: Nearby Buses, Buildings, or Landmarks for Specific Station Exit

## –Q4.1: Buses

| 43 | 74 | 81 | M217 | M299 | M343 | M385 | M393 | M459 | M460 | M554 | M566 | 高峰专线 | 高峰专线 | 高快巴士 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | 119 | 150 | 23 |

Table 6: A transposed list of bus lines, presenting all names horizontally to facilitate easier viewing of the range of services.

## –Q4.2: Buildings or Landmarks

| Location | Nearby Features |
|---|---|
| Text | 京基御景峰、留仙大道南侧（西）、长源居委会 |

Table 7: Details of buses and landmarks near specific station exits as provided in the data.

# Q5: Specific Passenger Journey Information

| Name | Entry Station | Exit Station | Start Time | End Time |
|---|---|---|---|---|
| 臧宜君 | Liangmao Hill | Zhucun | 2023-02-02 18:36:42 | 2023-02-02 19:12:42 |
| 臧宜君 | Yuyuan station | Xitou | 2023-09-12 19:54:04 | 2023-09-12 19:58:04 |
| 臧宜君 | Ludancun | Yonghu | 2023-10-29 22:56:20 | 2023-10-29 23:03:20 |
| 臧宜君 | Songgang | Huangtian | 2023-12-08 21:53:04 | 2023-12-08 22:16:04 |
| 臧宜君 | Taiziwan | Yangmei | 2023-11-13 07:39:15 | 2023-11-13 08:33:15 |

Table 8: Journey details for passenger 臧宜君 including times and stations.

# Q6: Specific Travel Card Journey Records

| **Code** | **Entry Station** | **Exit Station** | **Start Time** | **End Time** |
|---|---|---|---|---|
| 883706296 | Buxin | Guangming | 2023-04-09 13:40:38 | 2023-04-09 14:03:38 |
| 883706296 | Tangkeng | Huaxin | 2023-03-14 11:39:13 | 2023-03-14 12:30:13 |
| 883706296 | Tongle South | Pingshan Center | 2023-12-02 06:46:02 | 2023-12-02 07:24:02 |
| 883706296 | Longcheng Square | Pingshan Station | 2023-04-24 15:32:01 | 2023-04-24 16:26:01 |
| 883706296 | Shangtang | Qianwan Park | 2023-11-02 14:35:34 | 2023-11-02 14:48:34 |
| 883706296 | ShaTian | Taiziwan | 2023-08-15 22:38:03 | 2023-08-15 23:29:03 |
| 883706296 | Guanlan | Taoyuancun | 2023-06-16 07:40:07 | 2023-06-16 08:16:07 |
| 883706296 | Tongle South | Taoyuancun | 2023-09-12 21:37:45 | 2023-09-12 21:58:45 |
| 883706296 | Hi-Tech Park | Xiangmi | 2023-07-30 16:57:38 | 2023-07-30 17:50:38 |

Table 9: Detailed journey records for the travel card number 883706296, showing entry and exit stations with corresponding times.

# Q7: Specific Subway Station Information

| Chinese Name | Station Name | Number of Exits | District | Line Name |
|---|---|---|---|---|
| 长岭陂 | Changlingpi | 4 | 南山区 | 5 号线 |

Table 10: Detailed information about the Changlingpi subway station.

# Q8: Specific Subway Line Information

| Name | Start Time | End Time | First Opening | Number of Stations | Intro |
|---|---|---|---|---|---|
| 1 号线 | 06:20:00 | 23:00:00 | 2004-12-28 | 30 | 深圳地铁 1 号线（Shenzhen Metro Line 1）... |

Table 11: Details of the Shenzhen Metro Line 1, including operating times and history with some of the intro being omitted due to length issue.