SUSTech CS308 DataBase Project1

Spring 2024

Start at April 11，End at April 29，2024

12210651贾适萌 12212751赵一诺 CS308 2024春 周三56班

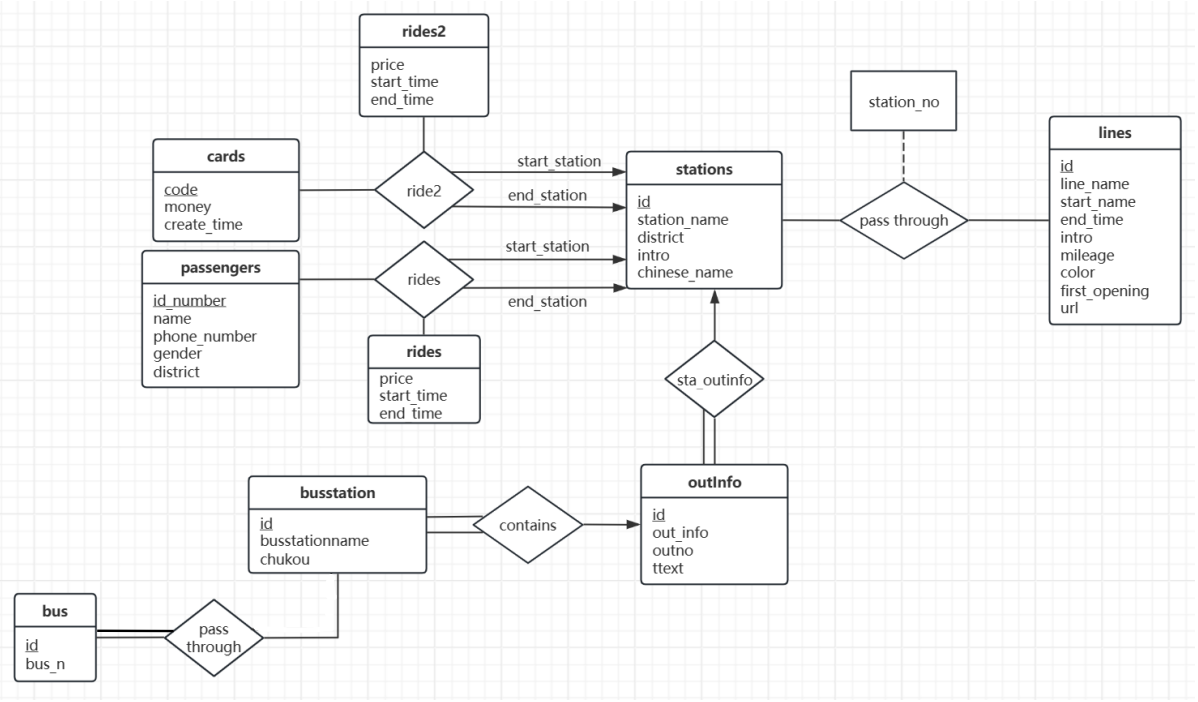
1 项目分工

姓名	TASK	贡献比
贾适萌	json文件与csv文件的格式转换与数据预处理， java、python、postgreSQL数据的导入与框架编写， 项目建表以及准确性查询的SQL语句编写， MySQL数据库的配置和数据导入，不同数据量导入比较， 项目报告写作	70%
赵一诺	ER图的绘制	30%

2 项目任务

TASK1 ER Diagram

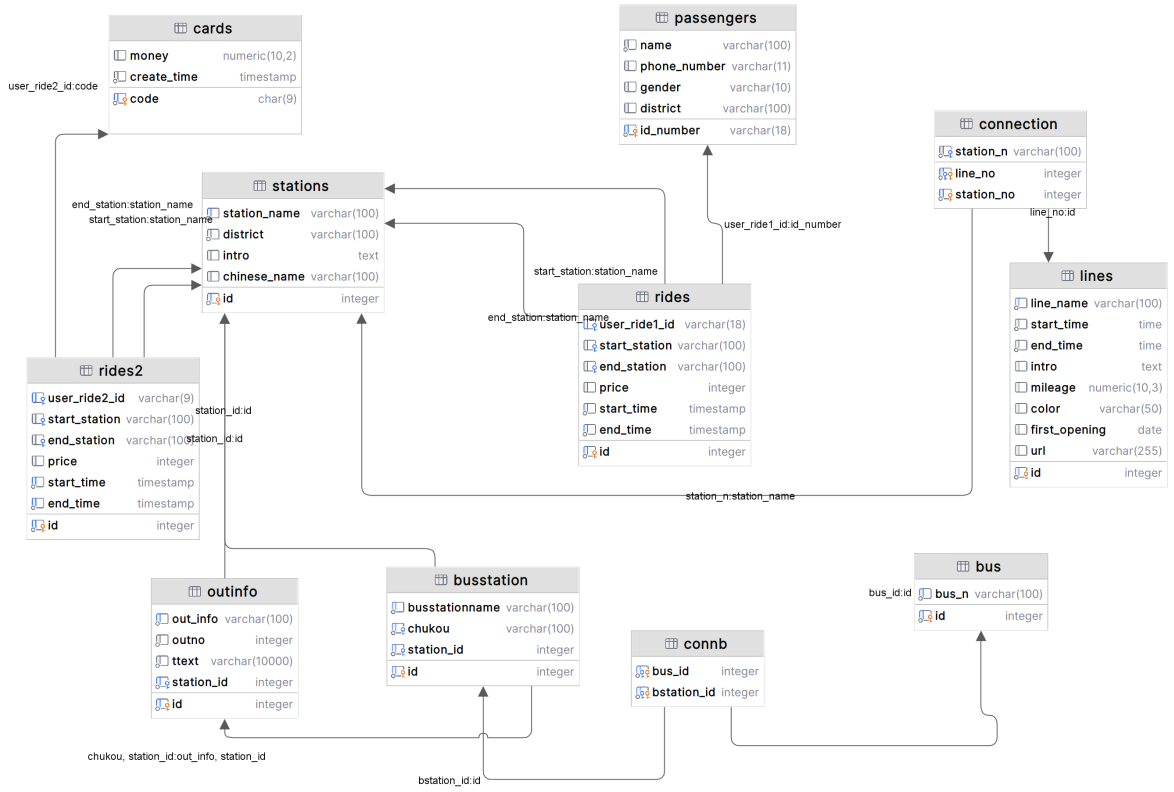
使用 **ProcessOn** 进行了ER图的绘制。



TASK2 Relational Database Design

DataGrip自动生成的关系图一览

在DataGrip选中所有表格，右键点击Diagrams，Show Diagrams，导出表格如下图。



所有数据表及其各列元素说明

在完成基本的思路构建后，开始数据库建表的工作，首先我们需要了解原始数据，为了符合3NF，需要拆解进行重新安排和连接的表格。先建立实体表格，再建立关系表格，明确存在外键关系的表格的添加顺序，最终建立11个表格如下（按建表顺序）：

num	表格名	note	Primary Key	各列元素说明
1	cards	存储所有的卡片信息	code	code:卡号，money: 卡片余额, create_time: 卡片创建时间
2	passengers	存储所有乘客信息	id_number	name: 姓名, id_number: 身份证号码, phone_number: 手机号码, gender: 性别, district: 地区
3	stations	存储所有地铁站信息	id	id:stations自增id，station_name: 站点名称, district: 区域, intro: 简介, chinese_name: 中文名
4	rides	存储以passengers的id_number为user_id的乘车记录	id	id: rides自增id，user_ride1_id: 用户乘车记录ID，start_station: 起始站点, end_station: 终点站, price: 乘车价格, start_time: 入站时间, end_time: 出站时间

num	表格名	note	Primary Key	各列元素说明
5	rides2	存储以cards的code为user_id的乘车记录	id	id: rides2自增id，user_ride2_id: 用户乘车记录ID，start_station: 起始站点, end_station: 终点站, price: 乘车价格, start_time: 入站时间, end_time: 出站时间
6	lines	存储所有地铁线路信息	id	id: lines自增id，line_name: 线路名称, start_time: 起始时间, end_time: 结束时间, intro: 简介, mileage: 里程, color: 标志颜色, first_opening: 首次开通时间, url: 百科链接
7	connection	存储每个地铁线路经过的地铁站及其站点顺序	(line_No, station_No)	line_No: 线路编号, station_No: 站点编号（每个地铁站是地铁线路的第几站）, station_n: 站点名称
8	outInfo	存储每个地铁站的所有出口信息	id	id: outInfo自增id，out_info: 出口信息，outNo: 出口编号, ttext: 出口描述, station_id: 站点ID
9	busStation	存储所有公交站信息	id	id: busStation自增id，busStationName: 公交站名称, chukou: 出口信息, station_id: 站点ID
10	bus	存储所有公交车	id	id: bus自增id，bus_n: 公交车名称
11	connB	存储每个公交经过的公交站信息	(Bstation_id, bus_id)	bus_id: 公交车ID, Bstation_id: 公交站ID

数据表建表的合理性说明

- 本数据库的建立满足三大范式（3NF）

1. 每个数据表的每一列都是原子性的，不可分割的，仅有一个值。
2. 表中的每个非主属性都完全依赖于表的候选键。
3. 表中的每个非主属性都不能依赖于其他非主属性，而是直接依赖于主键。

- 所有的数据项都应该基于五个文件行，每个表中的每一行都应该由其主键进行唯一标识。每个表都包含一个外键，没有被隔离的表，不包含环形的外键链接。每个表都至少包含一个强制性的（“NotNull”）列，除了主键自增的id之外, 有其他unique约束的列，并且为不同的字段使用了适当的数据类型，并且具有可扩展性。

注： 1. 对原数据ride.json做拆分处理，rides的外键指向passengers，rides2的外键指向cards。

2. busStation的外键指向outInfo（所有地铁站的出口），在建表时添加联合外键约束。

TASK3 Data Import

Task 3.1 Basic Requirements:

Java脚本介绍

- 数据预处理(changeString.java)

  在changeString.java文件中有6个方法，来处理不同类型的字符串，增加数据库的可读性与统一性。

方法名	use in/apply	操作
changestn()	处理station名称	处理"\n"、"NBSP"、"6/MSP"等空格类型、首尾过多空格、station大小写等问题。
changeText()	outInfoGet.java	处理NBSP"、"6/MSP"等空格类型、过多空格等问题，统一括号类型"()、（）"。
changeOut()	outInfoGet.java busInfoGet.java	处理NBSP"、"6/MSP"等空格类型、过多空格等问题。
changeBusInfo()	busInfoGet.java	处理bus之间的分隔符（"","；","·","/","、、","、、、","，","；"）统一为"、"；处理所有小写m为大写，处理所有小写e为大写；处理换行符；去掉所有"路""号"；处理NBSP"、"6/MSP"等空格类型、过多空格等问题，统一括号类型"()、（）"。
changeBusName()	busInfoGet.java	处理bus之间的分隔符（"","；","·","/","、、","、、、"）统一为"、"；统一括号类型"()、（）"；
changeChuKou()	busInfoGet.java	处理NBSP"、"6/MSP"等空格类型、过多空格等问题，将"出人口"替换为"出入口"

  部分在非方法内修正的数据：

  1. 在bus分割数组busArray后对单个bus，去掉所有结尾的"线""号"，以及如果以"需删除"结束，直接continue，不录入表格。
  2. 对于chukou.equals("此站暂无数据")，进行continue不写入数据。

  部分手动修改的数据：

  1. （过铁路桥，距离G出入口约300米）去掉"，"。
  2. 将Peak Special Line102提前进行合并PeakSpecialLine102，避免空格处理造成的分割。
  3. "高峰专 线 31 路、高快巴士 47 号"合并空格，避免分割。
  4. 分割"M138高峰专线29"，避免重复。
  5. 对lines.json中Nonglin的district，修改"/西丽线"为"福田区"。

- json文件和csv文件的转换（目录src/Trans）

  这里我们使用了`common-csv-1.10.0.jar`和`gson-2.10.1.jar`，根据我们的ER图，将json文件拆分成xxx个csv表格，便于后面的导入，对于有较多数组嵌套的表格结构，通过JsonObject和JsonArray进行一层层的遍历和读取，详见src/Trans目录。

分别处理5个原始json文件为一下xxxx个csv文件，其中，对于`passenger.json`，`card.json`，`ride.json`不做大的变动，直接获取其对应的列和数据；对于`line.json`，首先我们读取除了"stations"的所有数据作为`lines.csv`的列，再读取每个stations和lines的联系，作为`connection.csv`；对于`stations.json`，我们先读取所有除了bus_info和out_info的数据作为`stations.csv`的属性列，再读取其中的out_info的数据作为每个地铁站的每个出口的数据（这里在一次遍历中找到了所有不重复的outt和chukou），再读取bus_info中所有公交站的数据，再读取所有bus的数据集，最后读取bus和对应的stations，并进行连接。

得到的每个csv文件对应如下：

num	file_name	read_json	file_head	out_csv
1	RideTrans.java	ride.json	"user", "start_station", "end_station", "price", "start_time", "end_time"	ride.csv
2	passTrans.java	passenger.json	"name", "id_number", "phone_number", "gender", "district"	ride.csv
3	cardsTrans.java	cards.json	"code", "money", "create_time"	passenger.csv
4	linesTrans.java	lines.json	"line_name", "stations", "start_time", "end_time", "intro", "mileage", "color", "first_opening", "url"	cards.csv
5	connectionGet.java	lines.json	"line_No, line_n, stationNo, station_n"	lines.csv
6	StationTrans.java	stations.json	"station_name", "district","intro", "chinese_name"	connection.csv
7	outInfoGet.java	stations.json	"id, out_info, outNo ,ttext, station_id"	stations.csv
8	busInfoGet.java	stations.json	"busStationName, chukou, bus, station_id"	outInfo.csv
9	busGet.java	stations.json	"bus, busStationName,"	busInfo.csv
10	connBGet.java	stations.json	"id, bus_id, Bstation_id"	bus.csv

在读取所有bus的信息时，使用了**Hashmap和HashSet**的数据结构来避免bus的重复录入。

```
HashMap<String, Integer> bus = new HashMap<>();
```

```
HashSet<String> gaokuai = new HashSet<>();
```

考虑到部分中文数字同时存在的HashMap/HashMap无法准确识别的情况，建立3个新的HashMap，存储该bus_n后面的数字部分，经观察得知只有大鹏假日专线、高峰专线、高快巴士有识别不准确的情况，且保证后面的编号彼此均不重复，所以可以放在一起处理。这里使用**正则表达式**匹配数字部分：

```
Pattern pattern = Pattern.compile("\\d+");
Matcher matcher = pattern.matcher(input);
while (matcher.find()) {
    String number = matcher.group();
        }
```

在处理所有chukuo和outt时，也使用Hashmap来进行数据的读取，对于该地铁站已经存在的outt，不进行重复读取，（"out_info"和"bus_info"中的出口都进行遍历！！）保证outInfo表中包含了地铁站的所有出口，使busStation表可以和outInfo相联系。

- 将csv文件导入数据库（目录src/Import）

  在第一次尝试导入中，我使用了之前lab课讲过的最快的导入数据的方法**Loader5Batch**，在resource文件夹中建立了dbUser.properties，存储 host, database, user, password, port，和数据库建立连接，其中每个import文件要修改的主要方法为以下部分（以ImportCard.java为例，仅展示部分代码）：

```
setPrepareStatement()
     stmt = con.prepareStatement("INSERT INTO cards (code, money,
create_time) " + "VALUES (?,?,?);");
```

```
loadCSVFile()
    Reader reader = Files.newBufferedReader(Path.of("cards.csv"));
        CSVParser csvParser = new CSVParser(reader, CSVFormat.DEFAULT);
```

```
loadData(CSVRecord record)
            String code = record.get(0);
            double money = Double.parseDouble(record.get(1));
            String createTime = record.get(2);
            stmt.setString(1, code);
            stmt.setDouble(2, money);
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
            java.util.Date parsedDate = dateFormat.parse(createTime);
            java.sql.Timestamp timestamp = new
java.sql.Timestamp(parsedDate.getTime());
            stmt.setTimestamp(3, timestamp);
            stmt.addBatch();
```

之后，我们在DataGrip对应的数据库中进行刷新，确保表格的成功导入。

11个Import文件与读取的csv文件对应如下表：

num	file_name	read_csv
1	ImportRide1.java	ride.csv
2	ImportRide2.java	ride.csv
3	ImportPassenger.java	passenger.csv
4	ImportCard.java	cards.csv
5	ImportLines.java	lines.csv
6	ImportConnection.java	connection.csv
7	ImportStations.java	stations.csv
8	ImportOutInfo.java	outInfo.csv
9	ImportBusInfo.java	busInfo.csv
10	ImportBus.java	bus.csv
11	ImportConnB.java	cinnB.csv

Task 3.2 Data Accuracy checking:

1. The numbrer of stations, in each district, on each line or in total.

```
--each district
select district, count(*) from stations group by district;
--each line
select line_name, count(c.line_No) as station_count
from connection c
join lines l on c.line_No = l.id
group by l.line_name;
```

2. Number of female passengers and male passengers respectively.

```
select gender, count(gender) from passengers group by gender;
```

3. List the number of passengers from Mainland China, Hong Kong, Macau, and Taiwan.

```
select district, count(district) from passengers group by district;
```

4. List the buses, buildings, or landmarks near a specific **station exit**.

   以Guomao的C出口为例

```
--building,landmark
select ttext from outInfo where station_id in(
    select id from stations where station_name = 'Guomao'
    ) and out_info = 'C出入口';
--buses
select distinct bus_n from bus where id in (
    select bus_id from connB where Bstation_id in (
            select id from busstation where chukou = 'C出入口'and
                   station_id in (select id from stations where station_name
= 'Guomao')));
```

5. List all information about a specific passenger's journey, including passenger name, entry station, exit station, date, and time.

```
select name, start_station, end_station, start_time, end_time from rides
    join passengers on user_ride1_id = passengers.id_number where
user_ride1_id
            in(select id_number from passengers where name = '朱阳');
```

6. List all journey records for a specific travel card, including card number, entry station, exit station, date, and time.

```
select user_ride2_id, start_station, end_station, start_time, end_time from
rides2
     where user_ride2_id in(select code from cards where code = '887046252');
```

7. Query information about a specific subway station, including Chinese name, English name, number of exits, the district it is located in, and the subway line it belongs to.

```
select station_name, chinese_name, district, count(out_info) ,line_name as
outNum from outInfo
    join stations on station_id = stations.id
    join connection on stations.station_name = connection.station_n
    join lines l on connection.line_No = l.id
                        where station_id in
            (select id from stations where station_name = 'Tanglang') group
by stations.id, line_name;
```

8. Query information about a specific subway line, including start time, end time, first opening time, number of stations, and an introduction

```
select line_name, start_time, end_time, first_opening, intro,
count(station_No) from lines
join connection on lines.id = connection.line_No
where line_name = '5号线' group by line_name, start_time, end_time,
first_opening, intro;
```

Task 3.3 Advanced requirements:

- **不同编程语言导入数据的效率对比**

1. **Java导入：**

   对于Java语言，选取其中效率最高的 `Batch` 的导入方法，导入130993条数据用时22.34s，导入速度为 **5869 record/s。**

```
"D:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Program Files\IntelliJ IDEA 2023.2.2\lib\idea_rt.ja
Card's 10000 records successfully loaded
Passenger's 10000 records successfully loaded
Stations's 306 records successfully loaded
Ride1's 63503 records successfully loaded
Ride2's 36497 records successfully loaded
Line's 16 records successfully loaded
Connection's 373 records successfully loaded
OutInfo's 1276 records successfully loaded
BusInfo's 877 records successfully loaded
Bus's 1120 records successfully loaded
ConnB's 7025 records successfully loaded
130993 records successfully loaded
Loading speed: 5869 records/s

Process finished with exit code 0
```

2. **Python导入：**

   对于python语言，使用psycopg2（Python 数据库适配器）来连接和操作 PostgreSQL 数据库，导入数据方法：`ImportAll.py`，导入130993条数据用时20.38s，导入速度为 **6425 record/s。**

```
"D:\Program Files\PyCharm 2023.3.3\HappyTry\venv\Scripts\python.exe" "D:\Program Files\PyCharm 2023.3.3\HappyTry\Import
bus Total records loaded: 1120
stations Total records loaded: 306
passengers Total records loaded: 10000
outInfo Total records loaded: 1276
busInfo Total records loaded: 877
cards Total records loaded: 10000
connB Total records loaded: 7025
lines Total records loaded: 16
connection Total records loaded: 373
rides Total records loaded: 63503
rides2 Total records loaded: 36497
All record: 130993
Loading speed: 6425.007155272902 records/s

Process finished with exit code 0
```

3. **PostgreSQL导入**

   在DataGrip中，通过copy语句可以实现外部文件的导入。

```
copy cards from 'C:\Users\Administrator\Desktop\Allcsv\cards.csv' with
csv header;
```

   计算11张表格的导入总效率为 **19100.76 record/s。**

```
总导入条数：130993条；
总执行时间 = 75 + 124 + 107 + 13 + 5 + 4175 + 2177 + 13 + 14 + 23 + 18 +
114 = 6858 ms = 6.858 s
Loading speed : 130993/6.858 = 19100.76 record/s
```

```
[2024-04-28 12:20:38] completed in 75 ms
p2.public> copy cards from 'C:\Users\Administrator\Desktop\Allcsv\cards.csv' with csv header
[2024-04-28 12:20:42] 10,000 rows affected in 124 ms
p2.public> copy passengers from 'C:\Users\Administrator\Desktop\Allcsv\passenger.csv' with csv header
[2024-04-28 12:20:42] 10,000 rows affected in 107 ms
p2.public> copy stations from 'C:\Users\Administrator\Desktop\Allcsv\stationsI.csv' with csv header
[2024-04-28 12:20:42] 306 rows affected in 13 ms
p2.public> copy lines from 'C:\Users\Administrator\Desktop\Allcsv\linesI.csv' with csv header
[2024-04-28 12:20:42] 16 rows affected in 5 ms
p2.public> copy rides from 'C:\Users\Administrator\Desktop\Allcsv\ride1.csv' with csv header
[2024-04-28 12:20:46] 63,503 rows affected in 4 s 175 ms
p2.public> copy rides2 from 'C:\Users\Administrator\Desktop\Allcsv\ride2.csv' with csv header
[2024-04-28 12:20:48] 36,497 rows affected in 2 s 177 ms
p2.public> copy connection from 'C:\Users\Administrator\Desktop\Allcsv\connectionI.csv' with csv header
[2024-04-28 12:20:48] 373 rows affected in 13 ms
p2.public> copy bus from 'C:\Users\Administrator\Desktop\Allcsv\bus.csv' with csv header
[2024-04-28 12:20:48] 1,120 rows affected in 14 ms
p2.public> copy outInfo from 'C:\Users\Administrator\Desktop\Allcsv\outInfo.csv' with csv header
[2024-04-28 12:20:49] 1,276 rows affected in 23 ms
p2.public> copy busstation from 'C:\Users\Administrator\Desktop\Allcsv\busInfo.csv' with csv header
[2024-04-28 12:20:49] 877 rows affected in 18 ms
p2.public> copy connB from 'C:\Users\Administrator\Desktop\Allcsv\connB.csv' with csv header
[2024-04-28 12:20:49] 7,025 rows affected in 114 ms
```

激活 Windows

对比来说，在数据量相同的前提下，导入效率 **Postgres > python > java。**

- **Java中五种导入方法的效率对比**

  分别在Java中选取五种不同的导入脚本：

  `Awful.java`，`Connect.java`，`Prepare.java`，`Transaction.java`，`Batch.java`

  这里我们以大数据集的导入作为代表：rides表，提前导入有外键关联的stations表和passenger表，分别使用以上五种方法进行数据导入，导入效率如下图：

  1. **ImportAwful.java**: 13 record/s

     ```
     Ride1's 63503 records successfully loaded
     Awful Loading speed : 13 records/s
     ```

  2. **ImportConnect.java**: 3357 records/s

     ```
     Ride1's 63503 records successfully loaded
     Connect Loading speed : 3357 records/s
     ```

  3. **ImportPrepare.java**: 4112 records/s

     ```
     Ride1's 63503 records successfully loaded
     Prepare Loading speed : 4112 records/s
     ```

  4. **ImportTransaction.java**: 4329 records/s

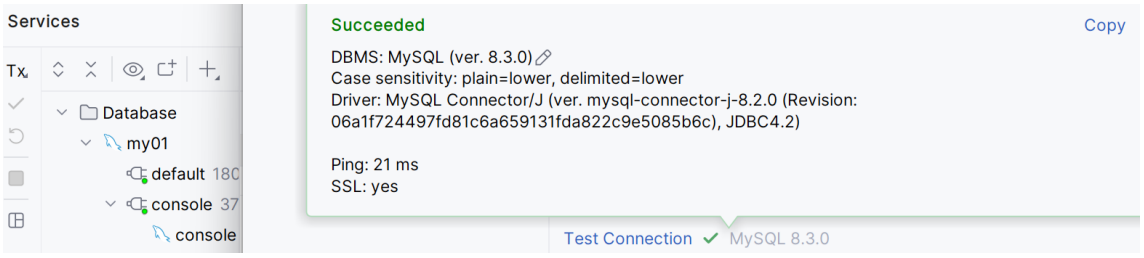
     ```
     Ride1's 63503 records successfully loaded
     Transaction Loading speed : 4329 records/s
     ```

  5. **ImportBatch.java**: 7149 records/s

     ```
     Ride1's 63503 records successfully loaded
     Batch Loading speed : 7149 records/s
     ```

- **尝试MySQL数据库进行导入**

  在完成postgreSQL数据库导入后，尝试新的数据库的配置和导入，这里我们选择了MySQL，使用了python脚本进行数据的导入（使用了mysql-connector包），导入脚本见 `ImportMysql.py`。



导入130993条数据耗时24.93s，导入效率为 **5254.14 records/s**。在MySQL导入的时候注意到我们的lines表的名称似与MySQL的关键字重合，故在建表时对 `lines` 进行添加双反反引号处理。

```
"D:\Program Files\PyCharm 2023.3.3\HappyTry\venv\Scripts\python.exe" "D:\Program Files\PyCharm 2023.3.3\HappyTry\Import\ImportMysql.py"
bus Total records loaded: 1120
stations Total records loaded: 306
passengers Total records loaded: 10000
outInfo Total records loaded: 1276
busInfo Total records loaded: 877
cards Total records loaded: 10000
connB Total records loaded: 7025
lines Total records loaded: 16
connection Total records loaded: 373
rides Total records loaded: 63503
rides2 Total records loaded: 36497
All record: 130993
Loading speed: 5254.140857282496 records/s

Process finished with exit code 0
```

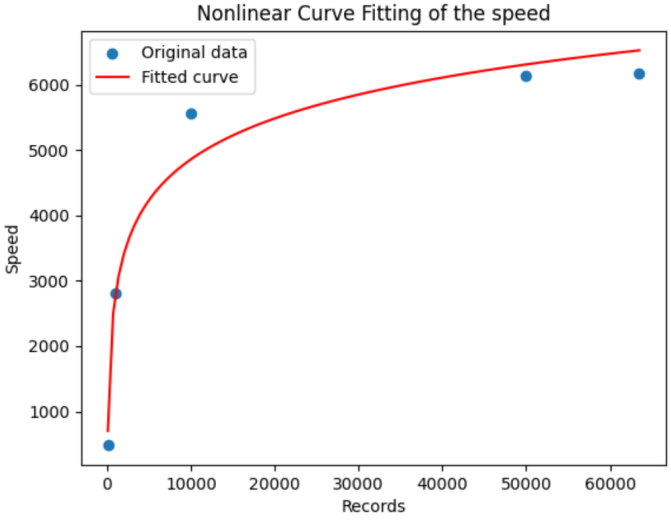
对比 `ImportAll.py` 的导入效率，我们可以看到 **PostgreSQL (6424) > MySQL (5254)**，导入PostgreSQL更快。

- **尝试不同数据量的导入**

  对导入过程中不同数据量的导入效率的对比，分别在导入rides表的过程中，对导入100条，1000条，10000条，50000条，全部（63503条）的时候的导入效率进行对比，脚本见 `DifferentVolume.py`，绘制拟合曲线如图所示。

```
"D:\Program Files\PyCharm 2023.3.3\HappyTry\venv\Scripts\python.exe" "D:\Program Files\PyCharm 2023.3.3\HappyTry\Import\Dif
The speed of 100 records is 480.1853742976338 records/s
The speed of 1000 records is 2815.7344781635275 records/s
The speed of 10000 records is 5567.647120119585 records/s
The speed of 50000 records is 6140.300229256395 records/s
rides Total records loaded: 63503
All record: 63503
Loading speed: 6166.019360148271 records/s

Process finished with exit code 0
```



可以看到，随着数据导入量的增加，导入速度也随之增加，总体导入速率呈对数增长的趋势（先极速增长，后稳速增长）。使用python的 numpy matplotlib scipy 包进行了非线性拟合如上图所示，拟合脚本见 `Draw/match.py`。

——项目1报告到此结束