CS307 Database Project1

成员：沈泓立（12311016），郑袭明（12311011）

成员分工及贡献百分比

沈泓立：

- E-R图绘制
- 进行非Postgres导入测试（MySQL）
- Python和C++数据导入框架与编写
- Python和C++部分batch导入优化以及相关对比测试
- 多平台导入数据测试（Windows、macOS）
- 项目报告写作

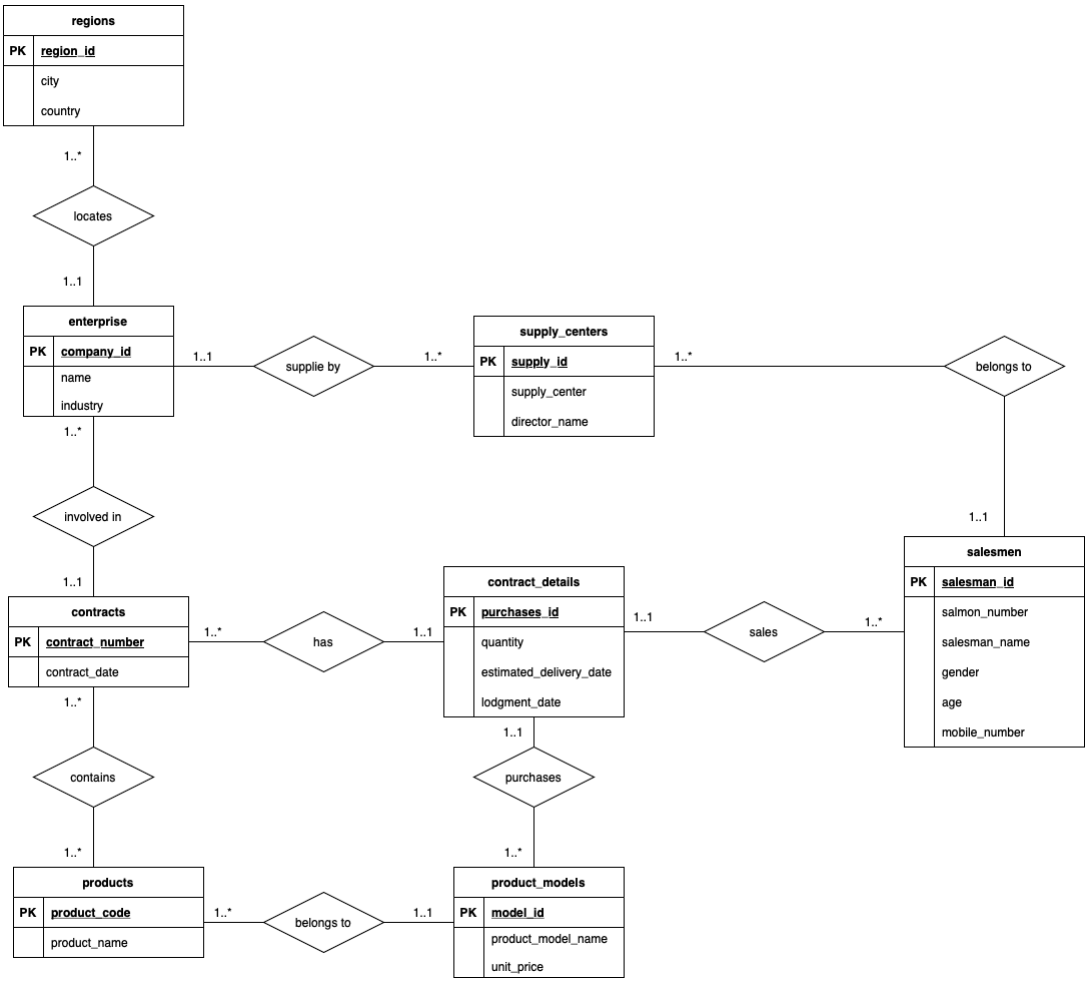
郑袭明：

- 数据库建表设计
- Java数据筛选与导入
- 基于Java的多种导入优化，如Batch、多线程等，并进行比较
- 项目相关的accuracy checking SQL语句编写
- 不同的Data Volume测试优化
- 项目报告写作

贡献百分比相同，均为**50%**

Task 1: E-R Diagram

本小组使用 drawio 绘图工具，绘制本项目的 E-R 图，截图如下：

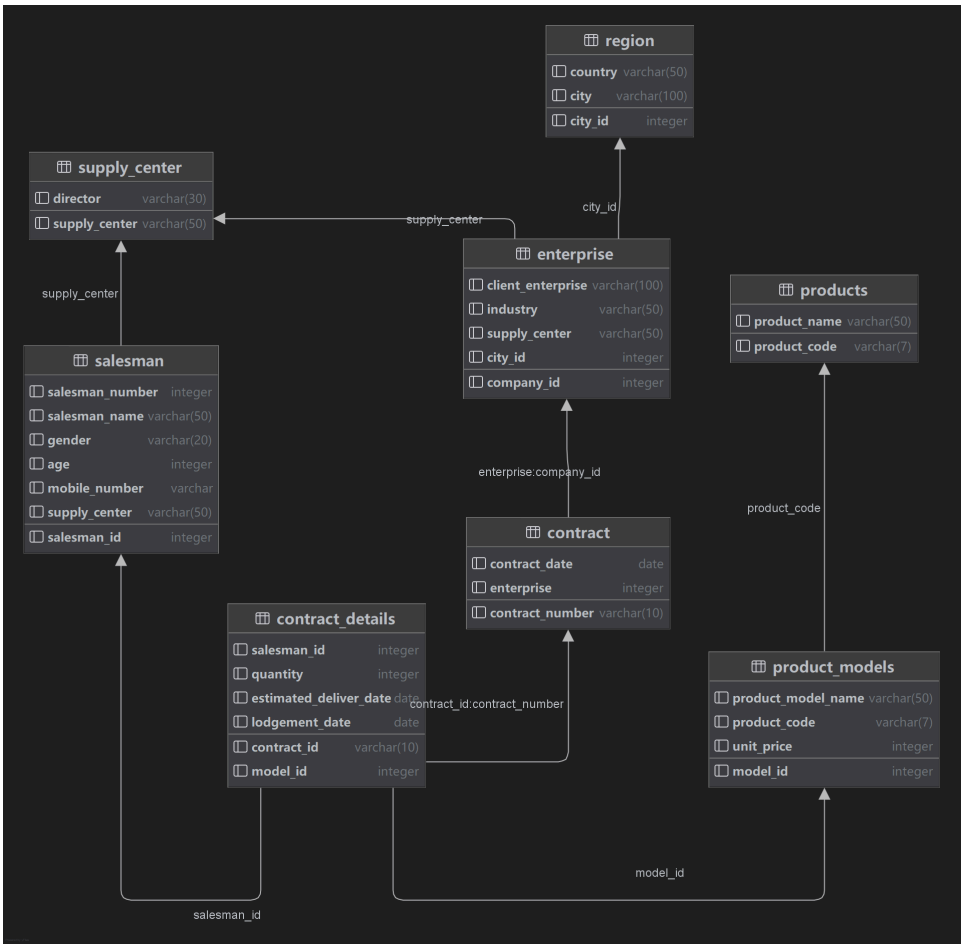


Task 2: Relational Database Design

本项目使用 `DDL.sql` 文件创建数据表，使用 `PostgreSQL` 的 `DDL` 语法编写。（后续在MySQL导入中，使用`MySQL`的语法改写进行导入。）

数据库设计

使用 `DataGrip` 创建数据表并全选后通过右键 `Diagram > Show Diagram` 显示如下数据表设计及关系。



设计思路及说明

数据表及其各列含义说明

在整个项目中共创建了 8 个数据表，数据表和其中各列、外键的含义如下：

1. **Supply_center** 表存储供应中心的信息。包括供应中心名称 `supply_center`（主键），所属区域的管理员 `director`。
2. **Region** 表存储地区信息。包括城市编号 `region_id`（主键），国家 `country`，城市名 `city`。
3. **Enterprise** 表存储公司的信息。包括公司编号 `company_id`（主键），客户公司名 `client_enterprise`，所属行业 `industry`，关联的供应中心 `supply_center`（外键），所在地区编号 `region_id`（外键）。
4. **Contract** 表存储合同信息。包括合同编号 `contract_number`（主键），合同签订日期 `contract_date`，签约企业编号 `enterprise`（外键）。
5. **Products** 表存储产品信息。包括产品编码 `product_code`（主键），产品名称 `product_name`。
6. **Product_models** 表存储产品型号的相关信息。包括产品型号编号 `model_id`（主键），型号名称 `product_model_name`，所属产品的编码 `product_code`（外键），该型号的单价 `unit_price`。
7. **Salesman** 表存储销售人员的信息。包括销售员编号 `salesman_id`（主键），工号 `salesman_number`，姓名 `salesman_name`，性别 `gender`，年龄 `age`，手机号 `mobile_number`，所属供应中心 `supply_center`（外键）。
8. **Contract_details** 表存储合同的详细内容。包括合同编号 `contract_id`、产品型号编号 `model_id`（复合主键）、销售员编号 `salesman_id`（外键），销售数量 `quantity`，预计交货日期 `estimated_deliver_date`，付款到账日期 `lodgement_date`。

数据库构建的合理性

- **满足三大范式**
  - 通过示意图可以看到，每个数据表的每一列都是不可分割的，仅有一个值。
  - 每个数据表都有主关键字，且主关键字都是 `UNIQUE` 的，其它数据元素能和主关键字一一对应。
  - 通过设计外键连接，我们将同一数据表中具有“传递”关系的数据列设计成不同的表格进行设计，不存在非关键字段对任一候选关键字段的传递函数依赖。
  - 可见，按以上设计思想设计的数据库满足三大范式的要求。
- 满足项目要求文档所要求的其它详细注意点，如外键无环、Unique约束列等。

Task 3: Data Import

Task 3.1 Basic Requirements:

脚本名称	作者	描述
CSVFormatAdjustment.java	郑袭明	数据筛选与格式调整调整。
CSVReader.java	郑袭明	通过运行这个Java脚本可以将全部数据分割为8个txt文件作为中间文件，分别对应数据库设计的8个表格。
SQLGenerator.java	郑袭明	将Resources中的8个txt文件作为输入，运行该脚本可以得到所有的建表语句以及插入内容的sql文件
Loader.java	郑袭明	运行这个Java脚本可以导入所有的数据到数据库中
mysql_loader.java	沈泓立	运行这个Java脚本可以导入所有的数据到MySQL数据库中
cpp_loader.cpp	沈泓立	运行这个C++脚本可以成功导入所有数据，使用libpqxx库连接PostgreSQL数据库。其中有两种导入方式，一种是普通逐行导入，另一种是批量管道导入
python_loader.python	沈泓立	运行这个python脚本可以成功导入所有数据，使用了psycopg2库来连接和操作PostgreSQL数据库。其中有两种导入方式，一种是普通逐行导入，另一种是依赖于文本读取的批量导入

在处理数据的过程中，我们通过创造了中间文件的方式来处理数据。

我们首先使用CSVFormatAdjustment.java对数据进行调整，把原有数据中Supply Center 名为 “Hong Kong, Macao and Taiwan regions of China” 改为 “Hong Kong and Macao and Taiwan regions of China” 以便后续数据导入。

然后我们使用CSVReader.java从原有csv文件中按行读入数据，以 “,” 分割，从而获得每个instance的具体信息。然后我们按建表语句分别将这些信息写入对应的txt文件，从而获得每个表对应的具体信息。

然后我们使用SQLGenerator.java分别读取Resources中的8个txt文件，生成对应的sql插入语句。生成的Data.sql中即包含导入数据的全部建表语句。

然后可以使用Loader.java从Data.sql中读入sql插入语句并执行，从而将全部数据导入数据库。由于Loader.java使用普通 `Statement` 逐条执行 SQL 文件中的完整插入语句，因此效率低下。后续3.3中我们会给出更高效的插入方式。

Task 3.2 Data Accuracy checking

Q1. How many salesmans are there for each gender?

```
select gender, count(*)
from salesman
group by gender;
```

	gender	count
1	Unknown	96
2	Male	444
3	Female	441

Q2. How many companies are there in each supply center?

```
select Sc.supply_center, count(company_id)
from Enterprise e
         join Supply_center Sc on Sc.supply_center = e.supply_center
group by Sc.supply_center;
```

	supply_center	count
1	America	63
2	Hong Kong and Macao and Taiwan regions of China	6
3	Northern China	13
4	Southwestern China	1
5	Eastern China	4
6	Asia	28
7	Europe	52
8	Southern China	6

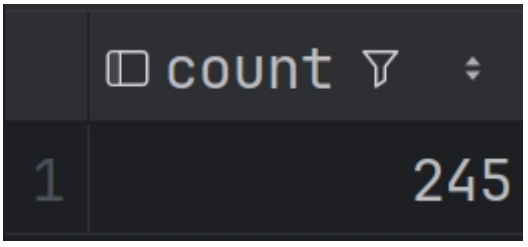
Q3. How many salesmans are there in each supply center?

```
select s.supply_center, count(salesman_id)
from salesman s
group by s.supply_center;
```

	supply_center	count
1	Eastern China	124
2	Asia	124
3	Northern China	122
4	America	124
5	Southwestern China	121
6	Hong Kong and Macao and Taiwan regions of China	119
7	Europe	123
8	Southern China	124

Q4. How many salesmans are there within a given age range (lower and upper bounds)? In this case: (30--40)

```
select count(*)
from salesman
where age > 30
  and age <= 40;
```



Q5. How many countries are there in each supply center?

```
select supply_center, count(distinct r.country) as country
from region r
         join Enterprise E on r.region_id = E.region_id
group by supply_center;
```

	supply_center	country
1	America	4
2	Asia	2
3	Eastern China	1
4	Europe	10
5	Hong Kong and Macao and Taiwan regions of China	1
6	Northern China	1
7	Southern China	1
8	Southwestern China	1

Q6. Given a country name (Italy or Canada), list all company names and their respective industries.

```
select country, client_enterprise, industry
from Enterprise e
         join Region R on R.region_id = e.region_id
where country = 'Italy'
   or country = 'Canada';
```

	country	client_enterprise	industry
1	Canada	Nortel Networks	Electrical and electronic
2	Italy	ENI	Oil refining
3	Italy	Fiat	Car
4	Italy	Assicurazioni Generali	Insurance

Q7. Given a product code (L8N0649), list all product models and their corresponding unit prices.

```
select product_code, product_model_name, unit_price
from product_models
where product_code = 'L8N0649';
```

	product_code	product_model_name	unit_price
1	L8N0649	LaptopA9	7357
2	L8N0649	LaptopC6	1242
3	L8N0649	Laptop78	8566
4	L8N0649	LaptopI7	7452

Q8. Given a contract number (CSE0000003), list all order details in the contract, including (product model, order quantity, salesmans name, and lodgement_date).

```
select contract_number, product_model_name, quantity, salesman_name,
lodgement_date
from contract c
         join Contract_details Cd on c.contract_number = Cd.contract_id
         join Product_models Pm on Pm.model_id = Cd.model_id
         join Salesman S on S.salesman_id = Cd.salesman_id
where contract_number = 'CSE0000003';
```

	contract_number	product_model_name	quantity	salesman_name	lodgement_date
1	CSE0000003	HotPot14	780	Phyllis Evans	2021-10-18
2	CSE0000003	RadioYogurtMaker82	830	Hua Zhaoyu	2021-09-28
3	CSE0000003	MultiplexerL3	810	Jessica Jones	2021-11-02
4	CSE0000003	VacuumCleaner75	110	Zhou Yaokun	2021-10-06
5	CSE0000003	WaterDispenserY1	980	Xie Yuyang	2021-10-28
6	CSE0000003	Cabinet97	970	Sonia Wright	2021-10-01
7	CSE0000003	PhysicalSecurityIsolationD9	910	Lv Yinxiang	2021-11-01
8	CSE0000003	CellPhoneBattery21	330	Oz Smith	2021-10-26
9	CSE0000003	DestroyEquipmentM9	720	Carrie Clarke	2021-10-21
10	CSE0000003	MultifunctionalT4	190	Chen Ziyan	2021-10-16
11	CSE0000003	MobilePhoneFilm18	210	Phyllis Evans	2021-10-30
12	CSE0000003	CashRegisterY2	830	Liu Wenjing	2021-11-14
13	CSE0000003	TelephoneConference05	410	Dou Ruichen	2021-10-16
14	CSE0000003	SlrHandleU6	850	Lindsay Davies	2021-10-29

Task 3.3 Advanced requirements

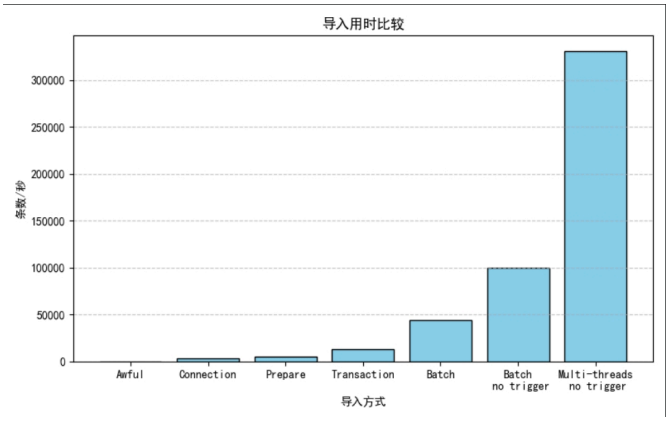
1. Try to optimize your script, and find more than one ways to import data, and provide a comparative analysis of the computational efficiencies between these ways.

我们使用多种方法试图优化，包括恒定Connection，引入Prepare Statement，引入Transaction机制，批量导入，多线程优化和在导入时Disable All Triggers的优化，共六条。

其中，前四条为Lab课上所提到的优化方式的实现。相关测试代码呈现在`/src/Loaders`中。为了方便比较，我们选用了多个表中的`Contract_details`一表作为比较对象，共 500000 条数据。

其中，Windows 测试的硬件环境为：

- **Lenovo Legion R9000P ARX8，AMD Ryzen 7 7745HX，32GB RAM，2TB SSD**
- **Windows 11，IntelliJ IDEA 2023.3.4，PostgreSQL 17.3**

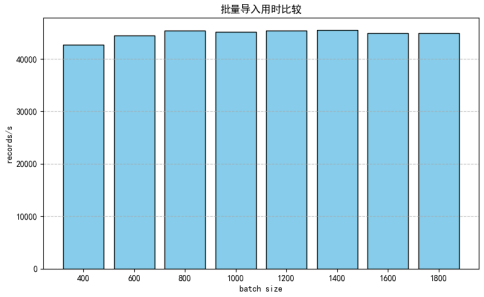
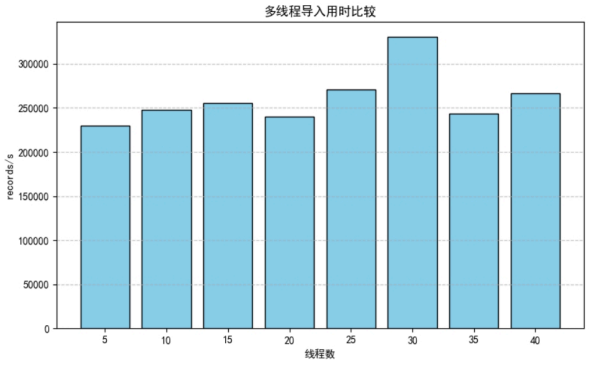
采用以下的输出形式	导入用时比较
insert 495000 data successfully! insert 496000 data successfully! insert 497000 data successfully! insert 498000 data successfully! insert 499000 data successfully! insert 500000 data successfully! 500000 records successfully loaded Loading speed : 44927 records/s	

其中，Batch导入方式和Multi-threads导入方式均采取其中最优的方式：

- Batch导入采用`BATCH_SIZE=1000`。
- Multi-threads导入采用`THREADS=30, BATCH_SIZE=1000`。

不难发现，各种导入方式速度逐渐得到提升，而最优的方式即为Multi-threads (With batches, no trigger)的方式，达到了330687条数/秒的高速。

所有的Batch导入尝试和Multi-threads导入尝试罗列如下：

Batch导入尝试	Multi-threads导入尝试：线程数为变量
	

可以看到，导入速度在 `batch size` = 800 到 `batch size` = 1800 的区间内并无显著区别，均为45000 records/s 左右，且每次运行得到的导入速度也不尽相同，可能会上下浮动500 records/s 左右。因此选取上述区间中任意作为`batch size`均可，并无显著区别。

而对于多线程，可以看出在指定Batch_Size大小下，随着线程数增加，速度先变快后遇到瓶颈，只有在合理数量时（比如30线程），才能最大化性能；超出最佳点，反而因为争用导致效率下降。

对于在导入时Disable Trigger的行为，我们也有所实践：

Data	Initial time	Optimized time	Multiple of optimization
Contract_details	11.23s	4.96s	2.26

可以看出，通过Disable Trigger，可以显著加快导入速度。当然，这么做的前提是所有数据已经经过前置检测，确认无误，否则可能导致数据库导入了错误的数据。

2. Try to import data across multiple systems.

在这一部分中，我们实现了两个平台的测试，Windows，macOS。

其中，macOS硬件的测试环境为：

- **MacBook Air, Apple M3, 16GB RAM, 512GB SSD**
- **macOS Sequoia 15.1, Visual Studio Code 1.99.0**

  Windows硬件环境同Task1

Operating System	Task	Time(s)	Velocity(Records/s)
MacOS	Loader.java	114.28	4834
Windows	Loader.java	48.16	11473

我们可以看出，在通过相同的Java demo导入数据时，macOS操作系统显示出明显更加强大的性能，数据导入速度是windows系统的2.37倍。

3. Try to import data using various programming languages (e.g., Java, Python, C++).

以下测试结果均建立在Task2的macOS测试环境下：

普通模式：

Language	Total_Time(s)	Velocity(Records/s)
Java	48.16	11473
C++	63.92	8644
Python	45.71	12086

可以看出，在普通模式下，Java和Python的导入时间相当，均为45-50s，而C++的导入时间则较长，要超过60s。

优化模式（仅对比C++和Python的优化效果）：

`C++`：使用PostgreSQL的libpqxx库的管道(pipeline)功能实现高效批量插入，它通过`pqxx::pipeline`将多个INSERT语句批量发送到数据库，每积累1000条记录就提交一次事务，相比逐条插入减少了事务开销和网络往返次数，显著提高了大数据量导入的性能。

`Python`：采用了一种简单直接的数据导入方式，它一次性读取整个SQL文件内容并通过`cursor.execute()`方法执行所有INSERT语句，相比逐条执行的方式减少了与数据库的交互次数，但由于是单次大事务提交，可能会产生较大的事务日志和锁竞争问题，适合中小规模数据导入但不适合真正的高性能批量导入场景。

Language	Initial time(s)	Optimized time(s)
C++	63.92	7.61
Python	45.71	7.41

可以看出，两种语言的优化方式均取得了显著的优化效果，将数据导入时间控制在了10s以内

4. Experiment with other databases.

我们实现了两个平台的测试：Postgres(基本)、MySQL.

以下测试结果均建立在Task2的macOS测试环境下：

DBMS	Time(s)	Velocity(records/s)
Postgres	48.16	11473
MySQL	319.71	1728

可以看出，Postgres数据库管理系统的数据导入速度显著高于MySQL数据库的数据导入速度，具有更优的性能和效率。

5. Try to import data with different data volumes.

在这一部分中，我们修改了Contract_details表的导入量，使用 Loader3Prepare 导入Contract_details表中，基于不同数据量分别观察导入效率。详细数据见下表：

可以看到，并没有产生明显的性能差异，导入效率基本在 5500 records/s左右。

N（总指令数）	总时间	每秒插入指令数量
1w	1.89s	5279

N（总指令数）	总时间	每秒插入指令数量
5w	9.10s	5495
10w	18.08s	5531
25w	45.05s	5548
50w	89.74s	5571