

2022 Summer CS213 Project

Contributors:

Zhu Yueming, Yu Tiancheng, Lu Hongyi, Wang Ziqin, Wang Weiyu, He Yirui, Yang Xiaosu, Chen Junfeng, Li Xin, Leng Ziyang, Niu Jingxuan, Qiu Yilun, Dai Guoyi

DDL

Deadline date: 2022, August 4th 18:00

Presentation date: 2022, August 5th, Lab class

Overview

It is a one-person project. Each student should finish the project by himself/herself and submit **one report in PDF**.

You should submit a report and code before the deadline, the **Topscore will be 100 (bonus, extra 10 points)** ; for the report submitted after the date, the **score will be 0** .

Please be honest. DO NOT copy ANY words, figures, and others from the Internet and others.

Task

Task 1: Database design

Design a database by **PostgreSQL** to manage all information mentioned in **course_info.json** and **select_course.csv** in the zip file **data/task1.zip**. Attention, there are some unreasonable data in these two files, correct them first. The total quantity of tables, the content in each table, all details should be determined by yourself.

Your design needs to meet the following **requirements**:

1. The tables created should satisfy the three normal forms;
2. Use the primary key and the foreign keys to indicate important attributes and relationships about your data. Every row in each table should be uniquely identified by its primary key;
3. Every table should be involved in a link. No isolate tables included;(每个表要有外键或者有其他表的外键指向，也就是说不能有孤立表)
4. Your design should contain no circular links; (对于表之间的外键方向，不能有环)
5. Each table should always have at least one mandatory ("Not Null") column (including the primary key but not the system-generated ID column); (每个表格中必须包含有至少一个非空的属性列，主键属于这个范畴，但是自增ID不属于)
6. Tables with no other unique columns than possibly a system-generated ID is not allowed; (除了主键自增的id之外，需要有其他unique约束的列)
7. Use appropriate types for different fields of data;

8. Your design should better be as easy to expand as possible.

Task 2: Import the data

Design programs/scripts to import data into your database from those two files (**course_info.json** and **select_course.csv**).

Your design needs to meet the following **requirements**:

1. Find ways to improve the efficiency of time-consuming during your importing process, and compare different importing methods.
2. Make sure all data are imported accurately, highly effectively, and automatically.

Task 3: Use DML to analyze your database in Task1

Design some experiments to show your database's performance, and record the execution time. Significant expressions, such as diagrams, the comparison will be welcome.

Your design needs to meet the following **requirements**:

1. The experiments should contain but are not limited to these manipulations: SELECT, DELETE, UPDATE, INSERT.
2. The experiments should be designed reasonably and comprehensively.

Task 4: Compare database and file

Design programs/scripts to make the comparison between the database and the file. And design some experiments to discuss their advantages and disadvantages. Significant expressions, such as diagrams, the comparison will be welcome.

Your design needs to meet the following **requirements**:

1. The programs/scripts could be any coding language if you like.
2. The experiments should be reasonable and comparable.
3. A reasonable analysis of the results from your experiments should be given.

Task 5: Modify your database tables to fit the required implementation codes

You will notice the database you designed in task 1 can not perfectly match what you need to do in the Service part in our provided codes. But you only need to modify some columns to match your needs in the Service part. So in this task, you need to modify your database tables to fit all interfaces you need to implement. We now upload the basic data for your test in **Task 6**, all of them are json files in the data folder. We just simply show the data tree directory here, the task1 folder can be ignored in this task. The files are not difficult to read, you can base on such files to modify your tables. By looking at the data, you can first see the requirements that follow.

```
| coursePrerequisites.json
| courses.json
| courseSectionClasses.json
| courseSections.json
| departments.json
| majorCompulsoryCourses.json
| majorElectiveCourses.json
| majors.json
```

```

|   sampleResult.txt
|   semesters.json
|   studentCourses.json
|   tree.txt
|   users.json
|
├─courseTable2
|   courseTable.json
|   courseTableResult.json
|
├─enrollCourse1
|   enrollCourse.json
|   enrollCourseResult.json
|
├─enrollCourse2
|   enrollCourse.json
|   enrollCourseResult.json
|
├─searchCourse1
|   searchCourseBasic.json
|   searchCourseBasicResult.json
|   searchCourseCid.json
|   searchCourseCidResult.json
|   searchCourseInstructor.json
|   searchCourseInstructorResult.json
|   searchCourseMixed.json
|   searchCourseMixedResult.json
|   searchCourseName.json
|   searchCourseNameResult.json
|
├─searchCourse2
|   searchCourseBasic.json
|   searchCourseBasicResult.json
|   searchCourseCid.json
|   searchCourseCidResult.json
|   searchCourseInstructor.json
|   searchCourseInstructorResult.json
|   searchCourseMixed.json
|   searchCourseMixedResult.json
|   searchCourseName.json
|   searchCourseNameResult.json
|
└─task1
    course_info.json
    select_course.csv

```

Task 6: Implement the codes

Your work on this task is mainly divided into the following parts below:

1. Implement the service interfaces to pass the basic test cases. You need to **implement all methods in the service folder**. Don't worry about the workload, what you will really pay attention to are **CourseService.java** and **StudentService.java**. For other methods, just simple sql scripts will help.
2. Manipulate your data tables based on the database you designed.

3. Profile your implementation and find ways to speed it up.
4. (Optional) Find other ways to implement similar functionalities as our interfaces and compare (some of) them, are they better, worse, or have different use cases.
5. Make sure your results of codes will be **Correct (Most important)** and **Effective**.
6. After finishing all your codes in the Service interface, you had better test your program based on the **ProjectJudge.java** we give to you.
7. We prepare a simple implementation of a method, you can refer it to implement other methods.

Bonus

1. High concurrency and transaction management.
2. User privileges management.
3. Database index and file IO.
4. Compare the performance of multiple databases with the file system over different operating systems.
5. Some other effective explorations to improve the efficiency of a database system.

Report Structure

Part 1. Personal Info

Need to write down your name and sid.

Part 2. Task 1 and Task 5

Provide a **clearly formatted** diagram of the table structure in **Task 1 and Task 5** generated by DataGrip.



Give clear explanations for the design of your database, tables, and some columns (if needed).

Notice that you have designed two different databases, you need to describe one of your databases in detail and one in short due to the similarity of the two databases.

Part 3. Task 2

Introduce how to design the programs/scripts for importing data, and give the core codes of your scripts.

Introduce how to improve the efficiency of importing data and give the core codes. You can design some experiments to improve the efficiency of your work. Make sure that the experiments should be reasonable, and the improvements should be obvious. Also, you may need to prepare to show your scripts and how it works in the presentation. Prepare some queries in advance!

Part 4. Task 3 and Task 4

Show the performance analysis of your database designed in **Task 1**. You need to prepare some DMLs to provide the comparison between the database and the file system using programs/scripts. Analyze the results of your experiments designed.

Part 5. Bonus

Discuss your explorations of the bonus part.

Code packaging and submission

A Brief User Manual

Step 1: Open Project in IDEA.



Step 2: Design your database.

Please make sure your database in **Task 5** can **be rebuilt by the sql file and work properly**.

Step 3: Create your package and classes that implement interfaces in **cn.edu.sustech.cs307.service**.



Step 4: Create your Factory that extends from **cn.edu.sustech.cs307.factory.ServiceFactory** and register your service implements in the constructor:

Take CourseService as an example:



After registering all implementation (notice all services need to be registered):



For better judgment, please override getUIDs in your ServiceFactory implementation. This function should return your student ID like the picture shows:



Step 5: Modify the setting files (**config.properties**) as follows:

```
serviceFactory=me.yourname.impl.MyFactory
jdbcUrl=jdbc:postgresql://localhost:5432/project3
username=****
password=****
```

serviceFactory is the path of your **ServiceFactory**.

Step 6: Run the benchmark, and debug by yourselves.

You can refer to the reference implementation result in **sampleResult.txt** as shown below.



The integer after each test case is the number of your correct results. Your program should give out exactly the same result as the reference implementation.

Interface Specification

The structure of the interfaces is as follows.

- **database** folder stores connection information such as username, password, url, we only provides **PostgreSQL** as the DBMS.
- **dto** folder stores a set of data objects that will be accessed by the interfaces. Your implementations will use them as parameters or returned values.
- **service** folder stores **Service Interfaces**, this is the folder you should pay special attention to. There exist multiple **.java** file where the interface signatures are stored. You need to implement you own **class** to fit these signatures.
- **exception** folder stores exceptions that you should **throw** if something went wrong.
- **factory** folder stores the **ServiceFactory** abstract class that you need to implement to create your service instances.

What you need to do is implement all methods in the service folder, here is a a reference implementation, it shows how to implement one method of an interface.

The following code is just a guide, the code that interacts with the database will usually be written in the DAO layer.

```
@ParametersAreNonnullByDefault
public class ReferenceStudentService implements StudentService {
    /* Some codes are omitted */
    @Override
    public void dropCourse(int studentId, int sectionId) {
        try (Connection connection =
SQLDataSource.getInstance().getSQLConnection();
        PreparedStatement stmt = connection.prepareStatement("call
drop_course(?, ?)") {
            stmt.setInt(1, studentId);
            stmt.setInt(2, sectionId);
            stmt.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    /* Some codes are omitted */
}
```

```
public class ReferenceServiceFactory extends ServiceFactory {
    public ReferenceServiceFactory() {
        registerService(StudentService.class, new ReferenceStudentService());
        registerService(CourseService.class, new ReferenceCourseService());
        // registerService(<interface name>.class, new <your implementation>());
    }
}
```

After you have implemented the factory class, be sure to put your factory class name into the file `./config.properties`. So that we can find your implementation and test.

Additional requirements of the interface

Java

- All add*() functions with int as return value should return the (presumably auto-generated) ID.
- All arguments are guaranteed to be non-null, unless marked as @Nullable.
- All return values (and their fields) should be non-null, unless explicitly documented otherwise. If a list/map is empty, put List.of()/Map.of() or equivalents instead of null.
- Your implementation should throw java.lang.UnsupportedOperationException if a method is not actually implemented, so the tests can fail quickly.

Rules

- Data should be persisted on disk after each writes operation instead of only modified in RAM. If you introduced a cache layer, you have to enforce consistency. You should also ensure durability in case of a sudden shutdown.
- You should **NOT** use frameworks such as **ORM**.
- You don't need to spend time on **GUI/WEB**, as we do **NOT** give extra scores for them.

Java-specific rules

- You should **NOT** modify or add any class in package cn.edu.sustech.cs307. Use another package for your implementations.
- You should **NOT** extend any class in package cn.edu.sustech.cs307.dto.
- In this project, we use Maven to manage dependent libraries. If you want to introduce a new library, you need to record it in pom.xml. Your dependencies should be downloadable from the Maven Central repository.

SQL file

You need to record all your DDLs in the sql file. Don't submit two or more sql files because our testing script will be perplexed so you may get a low score! You can record your sql scripts in **Task 1** in this file if you ensure all tables will be updated again (you can drop the useless tables in **Task 5**, or maybe you can reserve some unless columns). Try to avoid the repeated name of the tables unless they are completely similar.

Submission

1. The source code of your project.
2. Only one SQL file named strictly **cs213.sql**, we will use this file to rebuild your database in the server. Please place this file into the root path(the same level of folder *src*, *data*,...) as the picture shows:



3. Delete documents as follow:

README.md,

pic

Choose all files and folder in root path, and make a zip **using your sid as zip name**.

Please make sure your database can **be rebuilt by the SQL file and work properly** as we stated above.

1. A personal report. Make a zip (**NOT RAR**, recommend to use 7z) file of 2, and submit this **zip file and a pdf report** on sakai .
2. **Do not import jar manually, use maven instead.** In this project, we use Maven to manage dependence libraries. If you want to introduce a new library, you need to record it in pom.xml. Your dependencies should be downloadable from the Maven Central repository, or your codes will fail to be compiled.

Tips

1. How to check the locale of a database: Connect to the database in Postgres shell, run command show all; , then find the keyword locale:



2. How should I know where I am wrong?

Read the interface documents carefully.

You are free to **modify any class** in **cn.edu.sustech.cs307**, we will replace them when testing your code on the server. But you need to ensure your program can be compiled and run based on the origin cn.edu.sustech.cs307. (所有实现的service方法必须在原有的，即我们所给的，项目结构中能正确地编译和运行得到结果)

For example, you can modify the **ProjectJudge.java** and dto classes to print your wrong test cases, also we have prepared the methods for you to detect what the bugs are.

Here is just a sample output in the console, try it by yourself, we hope your program will run perfectly.



However, DO NOT TRY TO modify the test cases or the judger to make your code "correct". We are not idiots.

3. How to generate the SQL file that ensure database is fully rebuildable?

Right click the schema of your project database, click **SQL Generator** in **SQL Scripts**.



Please make sure your settings are consistent with the following image:



1. Then you can copy the generated SQL, which contains all tables, indexes, and functions.
2. It is recommended to use git to manage group collaboration.

How to calculate the number of weeks during two days?

It's recommended to use the following SQL code:

```
SELECT (floor((day_end - day_start) / 7.0)::integer + 1) AS weekoffset
```

Please create database with **LC_COLLATE='C'**, which provides the platform-independent sorting result.

Here is a sample command:

```
CREATE DATABASE project2 WITH ENCODING='UTF8' LC_COLLATE = 'C';
```


Presentation

Since time is limited, show the shining points of your total work.