CS307 Spring 2021 Database Project 2

1. Source code

Download link:

For java: https://github.com/NewbieOrange/SUSTech-SQL-Project2-Public

For python: https://github.com/ziqin/SUSTech-CS307-Project2-Python

Interface Specification

We provide the interfaces code in two language (Java, Python). You can use either of them upon your preference. Notice
that due to the inherent differences between these two language, the **Score Policy** may also be different.

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 interfaces. Your implementation 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.

Your Tasks

- Implement the `service` and `factory` interfaces to pass the base testcases.
- Design your (PostgreSQL) database to satisfy the requirements of interfaces.
- Profile your implementation and find ways to speed it up.
- (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.

Here is a reference implementation, it shows you how to implement one method of an interface. To get a service working, you'll have to implement **all its interfaces**

The following code is just a guide, the code interacts with 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();
```

```java
            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 */
}
```

```java
public class ReferenceServiceFactory extends ServiceFactory {
    public ReferenceServiceFactory() {
        registerService(StudentService.class, new ReferenceStudentService());
        // TODO: register other service implementations here
    }
}
```

Additional requirements of 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.
- Do **NOT** modify anything in the provided interfaces, or any of the framework code.
- Your implementation should throw `java.lang.UnsupportedOperationException` if a
  method is not actually implemented,
  so the tests can fail quickly.

Python

- All `add*()` functions with return value of int should return the (presumably auto-generated)
  ID.
- All arguments are guaranteed to follow the type hints. Arguments passed won't be `None`
  unless explicitly annotated
  with `Optional`.
- For return types with type hints, return values (and their fields) should not be `None`, unless
  explicitly documented.
  Use `[]`, `{}`, `set()` or their equivalents instead of `None` for empty container in return
  values.
- Your implementation should raise `NotImplementedError` if a method is not actually
  implemented, so that tests can
  fail quickly.

Rules

- Data should be persisted on disk after each write operation instead of only modified in RAM. If you introduced a cache
  layer, you have to enforce the consistency. You should also ensure the 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.

Python-specific rules

TO BE ADDED

2. What to submit?

If you are using Java:

- The entire Java repository (all source codes and configs, without the `target` folder).
- The database dump from `pg_dump` without data in tables (by adding `-s` argument to `pg_dump`).
- The jar file built using maven-assembly (Run `IDEA-Maven-Plugins-assembly-single`)
  in `/target/xxxx-with-dependencies.jar`.

If you are using Python:

3. What to deliver?

- **PASS BASE TEST:** First and foremost, you should pass the base testcases, this is the basic requirement.
- **IMPROVE YOUR EFFICIENCY:** After you passed the base tests, you need to find ways to improve the performance of your implementation. You can work on the following aspects.

  ### Resource Consumption

  - Memory Consumption: How much memory your database takes?
  - Disk Consumption: How much disk space your database takes? How are they distributed? (index, data, other info?)

Speed

- Data Import Speed: How much time your database need to import all data?
- Data Modify Speed (Insertion, Update, Deletion): How much time your database need to change one/one hundred/one million rows?
- Data Query Speed: How much time your database need to fetch one/one hundred/one million rows?
- Cache Policy: How much time your database need to fetch a row if the row was just accessed by others?

Concurrency

- Simultaneous Query Number: How many queries can your database handles simultaneously?
- Simultaneous Query Latency: How long does it take to query if there are many users connect to your database simultaneously.
- Transaction Safety: Is your database safe with many users concurrently writing/reading to it?

Correctness

- Malformed Data Identification: Can your database identify the malformed data automatically?
- ACID Principle

- **(Optional) DIFFERENT WAYS SAME GOAL?** Can you find other ways to implement these functionalities? Are they **BETTER/WORSE/USECASE-RELATED?** Please do share us your amazing ideas.

Project Timeline

Code Submission Deadline: **June 6th**

Presentation Time: **June 15th** by Tencent Meeting