

# CS307 Principle of Database Systems

## PROJECT 02 REPORT

### Database for Management Department of SUSTC

#### 1 项目基本信息

##### 1.1 项目作者

- 项目作者 1:
  - 姓名: 罗嘉诚 (Jiacheng Luo)
  - 学号: 12112910
  - 实验课: Lab 2
- 项目作者 2:
  - 姓名: 伦天乐 (Tianle Lun)
  - 学号: 12113019
  - 实验课: Lab 2

##### 1.2 贡献比

根据小组内部讨论并达成一致, 建议最终项目贡献比:

- 罗嘉诚: 50%
- 伦天乐: 50%

各任务详细贡献如下:

- 罗嘉诚
  - 1. 数据库设计、E-R 图绘制
  - 2. 基础 API 设计
  - 3. 报告写作
- 伦天乐
  - 1. 基础 API 设计
  - 2. GUI 前端设计、前后端交互
  - 3. 报告写作

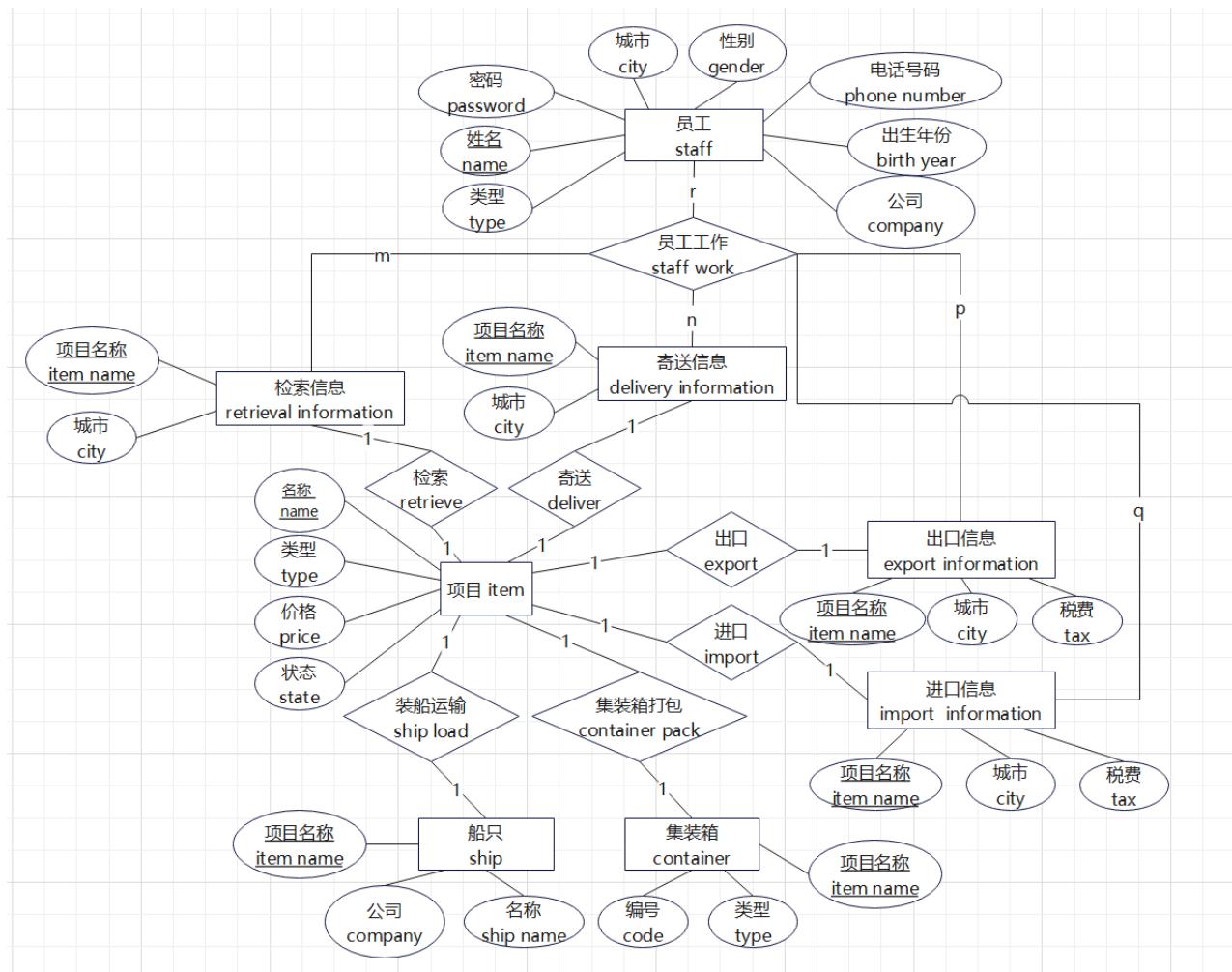
#### 2 Task 1: 数据库设计 15%

在 Project 1 的基础上, 我们修改了数据库的设计, 主要在如下方面进行了修改:

1. 根据给出的数据要求, 增加必要的属性, 删除冗余的属性: 如对 **项目 item** 增加 **状态 state** 属性、将 **快递员 courier** 替换为 **员工 staff** 并添加 **类型 type**、**密码 password**、**城市 city** 属性, 删除与 **时间 time** 有关的属性。
2. 根据数据库接口 **API** 修改数据库中存储各个属性的列的数据类: 如将 **项目 item** 中的 **价格 price** 属性的数据类型从 **int** 修改为 **numeric**, **员工 staff** 中的 **性别 gender** 属性由 **char** 类型修改为 **boolean** 类型。

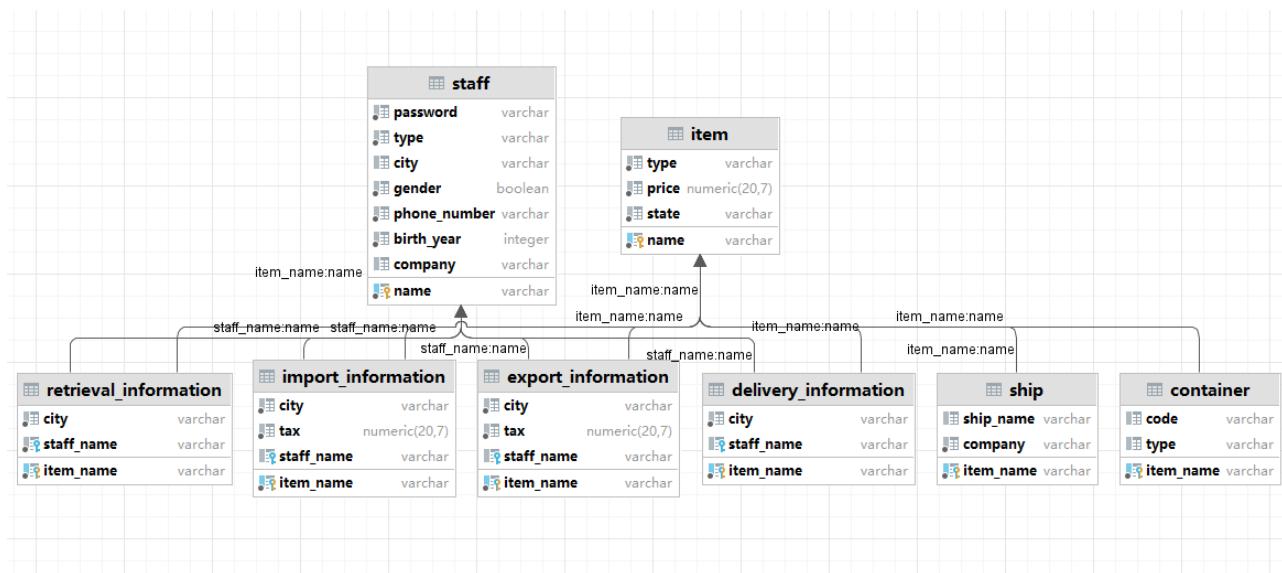
接下来，是我们修改数据库设计后的 E-R 图与数据库可视化图。

## 2.1 E-R 图



## 2.2 数据库可视化图

使用 [DataGrip](#) 中创建数据表后并全选，右键 [Diagram > Show Visualization](#)，显示表设计与关系：



数据库的构建满足三大范式，也满足 Project 1 中所指出的其他所有要求。

## 2.3 数据库角色创建与权限赋予

在数据库创建中，共计创建了 5 个不同的角色，根据 [API](#) 进行具体实现时，分别赋予不同的权限。

- 超级用户 [Super User](#)

拥有对数据库操作的最高权限，即拥有数据库操作的所有权限，包括但不限于：

1. 数据表创建权限。
2. 对于所有数据表的增、删、改、查权限。
3. 添加其他数据库角色权限。

我们只在创建数据表的两个 [API](#) (即 API 1 ~ API 2) 中使用该权限。

- SUSTC 部门经理用户 [SUSTC Department Manager User](#)

仅拥有对数据表查询的最高权限，包括：

1. 在所有数据表中查询所有字段和记录的权限。

注意：除了拥有对所有数据表查询的最高权限以外，本角色没有如插入、更改、删除等的其他权限。

在SUSTC 部门经理用户对应的 [API](#) (即 API 3 ~ API 10) 中，仅使用该权限。

- 快递员用户 [Courier User](#)

仅拥有对数据表中查询和修改它自己本身负责的项目状态权限、查询和修改它自己个人信息的权限。

1. [item](#) 表中，对 [name](#)、[type](#)、[price](#)、[state](#) 字段的查询权限
2. [staff](#) 表中，对 [name](#)、[city](#) 字段的查询权限
3. [import\\_information](#) 表中，对 [item\\_name](#)、[tax](#)、[city](#) 字段的查询权限
4. [export\\_information](#) 表中，对 [item\\_name](#)、[tax](#)、[city](#) 字段的查询权限
5. [retrieval\\_information](#) 表中，对 [item\\_name](#)、[staff\\_name](#) 字段的查询权限
6. [delivery\\_information](#) 表中，对 [item\\_name](#)、[city](#)、[staff\\_name](#) 字段的查询权限
7. 修改 [item](#) 表中 [state](#) 字段的权限
8. 修改 [delivery\\_information](#) 表中 [staff\\_name](#) 字段的权限
9. 向 [item](#) 表中插入新记录的权限
10. 向 [delivery\\_information](#) 表中插入新记录的权限
11. 向 [export\\_information](#) 表中插入新记录的权限
12. 向 [import\\_information](#) 表中插入新记录的权限
13. 向 [retrieval\\_information](#) 表中插入新记录的权限

在快递员用户对应的 [API](#) (即 API 11 ~ API 12) 中，仅使用该权限。

- 公司经理用户 [Company Manager User](#)

仅拥有数据库中选择合适的船只运送至海港等待物品的权限、查询和修改公司信息的权限。

1. [item](#) 表中，对 [name](#)、[price](#)、[state](#)、[type](#) 字段的查询权限
2. [import\\_information](#) 表中，对 [item\\_name](#)、[tax](#)、[city](#) 字段的查询权限
3. [export\\_information](#) 表中，对 [item\\_name](#)、[tax](#)、[city](#) 字段的查询权限
4. [container](#) 表中，对 [item\\_name](#)、[code](#)、[type](#) 字段的查询权限
5. [ship](#) 表中，对 [item\\_name](#)、[ship\\_name](#)、[company](#) 字段的查询权限
6. [staff](#) 表中，对 [name](#)、[company](#) 字段的查询权限
7. 修改 [item](#) 表中 [state](#) 字段的权限
8. 修改 [container](#) 表中 [code](#)、[type](#) 字段的权限
9. 修改 [ship](#) 表中 [ship\\_name](#) 字段的权限

在公司经理用户对应的 **API** (即 API 13 ~ API 19) 中, 仅使用该权限。

- 海关用户 **Seaport Officer User**

1. **item** 表中, 对 **name**、**state** 字段的查询权限
2. **staff** 表中, 对 **name**、**city** 字段的查询权限
3. **import\_information** 表中, 对 **item\_name**、**city**、**staff\_name** 字段的查询权限
4. **export\_information** 表中, 对 **item\_name**、**city**、**staff\_name** 字段的查询权限
5. 修改 **item** 表中 **state** 字段的权限
6. 修改 **export\_information** 表中 **staff\_name** 字段的权限
7. 修改 **import\_information** 表中 **staff\_name** 字段的权限

在海关用户对应的 **API** (即 API 20 ~ API 21) 中, 仅使用该权限。

### 3 基础 API 实现 70%

为实现项目要求中的所有 API 的对应功能, 我们使用 **Java** 进行实现, 并在本地测试中通过了所有示例数据。

接下来, 我们将介绍如何使用 **SQL** 语句, 实现各个 **API**。

#### 3.1 数据表创建

在创建数据表的两个 **API** (即 API 1 ~ API 2) 中使用且仅使用 **Super User** 权限。

1. `Constructor(String database, String root, String pass);`

这个 **API** 主要实现对数据库超级用户的连接、数据表的创建、数据库中各角色的创建与连接。

- 实现对数据库超级用户的连接, 只需要使用 **root** 用户与数据库直接连接, 就获得了默认的具有最高的权限的数据库超级用户。

```
1 private Connection createConnection(String database, String user, String pass) throws
2   SQLException {
3     Properties properties = new Properties();
4     properties.setProperty("user", root);
5     properties.setProperty("password", pass);
6     properties.setProperty("useSSL", "false");
7     properties.setProperty("autoReconnect", "true");
8     return DriverManager.getConnection("jdbc:postgresql://" + database, user, pass);
9 }
10 public DBManipulation(String database, String root, String pass) {
11   this.database = database;
12   this.root = root;
13   this.pass = pass;
14   try {
15     this.rootConn = createConnection(database, root, pass);
16     // ...
17   } catch (SQLException e) {
18     e.printStackTrace();
19 }
```

- 实现数据表的创建，需要使用上面创建的 `root` 用户，使用 `create table` 语句进行数据表创建，为了实现表间相关约束，我们采用 `alter table` 添加相关限制。

创建数据表的全部代码，请在 `DBManipulation(String database, String root, String pass)` 中查看。

下面以创建 `staff` 表为例，仅作参考：

```
Statement sta = this.rootConn.createStatement();
sta.executeUpdate(sql: "create table if not exists staff (
    + "      name varchar not null,"
    + "      password varchar not null,"
    + "      type varchar not null,"
    + "      city varchar,"
    + "      gender boolean not null,"
    + "      phone_number varchar not null,"
    + "      birth_year integer not null,"
    + "      company varchar,"
    + "      primary key (name)"
    + ");"
    + "create table if not exists export_information (
    + "      item_name varchar not null,"
    + "      city varchar not null,"
```

下面展示添加表间相关约束部分的代码，仅作参考：

```
+ "      city varchar not null,"
+ "      staff_name varchar references staff(name),"
+ "      primary key (item_name)" + ");"
+ "create table if not exists item("
+ "      name varchar not null,"
+ "      type varchar not null,"
+ "      price numeric(20, 7) not null,"
+ "      state varchar not null,"
+ "      primary key (name)"
+ ");"
+ "alter table delivery_information drop constraint if exists foreignKey_deliveryInformation_itemName;"
+ "alter table retrieval_information drop constraint if exists foreignKey_retrievalInformation_itemName;"
+ "alter table export_information drop constraint if exists foreignKey_exportInformation_itemName;"
+ "alter table import_information drop constraint if exists foreignKey_importInformation_itemName;"
+ "alter table ship drop constraint if exists foreignKey_ship_itemName;"
+ "alter table container drop constraint if exists foreignKey_container_itemName;"
+ "alter table delivery_information add constraint foreignKey_deliveryInformation_itemName foreign key (item_name) references item(name);"
+ "alter table retrieval_information add constraint foreignKey_retrievalInformation_itemName foreign key (item_name) references item(name);"
+ "alter table export_information add constraint foreignKey_exportInformation_itemName foreign key (item_name) references item(name);"
+ "alter table import_information add constraint foreignKey_importInformation_itemName foreign key (item_name) references item(name);"
+ "alter table ship add constraint foreignKey_ship_itemName foreign key (item_name) references item(name);"
+ "alter table container add constraint foreignKey_container_itemName foreign key (item_name) references item(name);";
```

- 实现数据库中各角色的创建与连接，我们首先删除已经添加的数据库角色，然后添加所需的四个新角色。

首先，使用 `revoke` 语句回收和撤销所有数据表中的权限。

```
this.rootConn.createStatement().execute(
    sql: "REVOKE ALL ON TABLE delivery_information, export_information, import_information, item, retrieval_information, ship, staff, container from cs307_ll_sustom;" +
        "REVOKE ALL ON TABLE delivery_information, export_information, import_information, item, retrieval_information, ship, staff, container from cs307_ll_courier;" +
        "REVOKE ALL ON TABLE delivery_information, export_information, import_information, item, retrieval_information, ship, staff, container from cs307_ll_companym;" +
        "REVOKE ALL ON TABLE delivery_information, export_information, import_information, item, retrieval_information, ship, staff, container from cs307_ll_seaportm;" +
);
```

然后，使用 `drop user` 语句删除已经存在的数据库角色，并使用 `create user` 语句创建四个新的数据库角色：`cs307_ll_sustcm`、`cs307_ll_courier`、`cs307_ll_seaportm`、`cs307_ll_companym` 分别代表 SUSTC 部门经理用户 `SUSTC Department Manager User`、快递员用户 `Courier User`、海关用户 `Seaport Officer User`、公司经理用户 `Company Manager User` 四种角色。

```
this.rootConn.createStatement().execute(sql: "DROP USER IF EXISTS cs307_ll_sustcm;"  
    + "DROP USER IF EXISTS cs307_ll_courier;"  
    + "DROP USER IF EXISTS cs307_ll_seaportm;"  
    + "DROP USER IF EXISTS cs307_ll_companym;"  
    + "CREATE USER cs307_ll_sustcm WITH PASSWORD '123456';"  
    + "CREATE USER cs307_ll_courier WITH PASSWORD '123456';"  
    + "CREATE USER cs307_ll_seaportm WITH PASSWORD '123456';"  
    + "CREATE USER cs307_ll_companym WITH PASSWORD '123456';"
```

然后，使用 `grant` 语句赋予各个角色相关数据库权限：

```
+ "GRANT SELECT ON staff, container, ship, item, import_information, export_information, retrieval_information, delivery_information TO cs307_ll_sustcm;"  
+ "GRANT SELECT(name, type, price, state) ON item TO cs307_ll_courier;"  
+ "GRANT SELECT(name, city) ON staff TO cs307_ll_courier;"  
+ "GRANT SELECT(item_name, tax, city) ON import_information, export_information TO cs307_ll_courier;"  
+ "GRANT SELECT(item_name, staff_name) ON retrieval_information TO cs307_ll_courier;"  
+ "GRANT SELECT(item_name, city, staff_name) ON delivery_information TO cs307_ll_courier;"  
+ "GRANT INSERT ON item, delivery_information, retrieval_information, import_information, export_information TO cs307_ll_courier;"  
+ "GRANT UPDATE(state) ON item TO cs307_ll_courier;"  
+ "GRANT UPDATE(staff_name) ON delivery_information TO cs307_ll_courier;"  
+ "GRANT INSERT ON delivery_information, export_information, import_information, item, retrieval_information TO cs307_ll_courier;"  
+ "GRANT UPDATE ON item, delivery_information TO cs307_ll_courier;"  
  
+ "GRANT SELECT(name, price, state, type) ON item TO cs307_ll_companym;"  
+ "GRANT SELECT(item_name, tax, city) ON import_information, export_information TO cs307_ll_companym;"  
+ "GRANT SELECT(item_name, code, type) ON container TO cs307_ll_companym;"  
+ "GRANT SELECT(item_name, ship_name, company) ON ship TO cs307_ll_companym;"  
+ "GRANT SELECT(name, company) ON staff TO cs307_ll_companym;"  
+ "GRANT UPDATE(state) ON item TO cs307_ll_companym;"  
+ "GRANT UPDATE(ship_name) ON ship TO cs307_ll_companym;"  
+ "GRANT UPDATE(code, type) ON container TO cs307_ll_companym;"  
  
+ "GRANT SELECT(name, state) ON item TO cs307_ll_seaportm;"  
+ "GRANT SELECT(name, city) ON staff TO cs307_ll_seaportm;"  
+ "GRANT SELECT(item_name, city, staff_name) ON import_information, export_information TO cs307_ll_seaportm;"  
+ "GRANT UPDATE(state) ON item TO cs307_ll_seaportm;"  
+ "GRANT UPDATE(staff_name) ON import_information, export_information TO cs307_ll_seaportm;");
```

最后，使用上述四种权限分别连接数据库。

```
this.sustcManagerConn = createConnection(database, user: "cs307_ll_sustcm", pass: "123456");  
this.companyManagerConn = createConnection(database, user: "cs307_ll_companym", pass: "123456");  
this.seaportConn = createConnection(database, user: "cs307_ll_seaportm", pass: "123456");  
this.courierConn = createConnection(database, user: "cs307_ll_courier", pass: "123456");
```

2. `void import(String recordsCSV, String staffsCSV);`

这个 `API` 主要完成的任务是：导入分别表示 `records.csv` 和 `staffs.csv` 中的文本内容两个字符串到数据库。由于我们认为数据是有效且合法的，因此在本部分中，我们复用 **Project 1** 中的较快的导入方案：在没有表间约束的情况下，批量导入所有数据，导入完毕后再进行约束检查。由于测试中的时限要求较宽，并且本次**Project 2**的数据量是 50000 条，是 **Project 1** 的 `1 / 10`，为了减小代码复杂度，我们并未采用多线程进行优化数据导入。

由于方法的重点在于如何进行字符串文本的处理，实现较为繁琐，如需查看此部分的完整代码，请在 `public void $import(String recordsCSV, String staffsCSV)` 中查看。

在本方法中，主要使用到了 `insert` 和 `alter table` 两种 `SQL`：

关闭数据库表间约束检查，我们可以使用：

```
this.rootConn.prepareStatement(sql: "ALTER TABLE delivery_information DISABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE retrieval_information DISABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE export_information DISABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE import_information DISABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE ship DISABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE container DISABLE TRIGGER ALL").execute();
```

向数据表中导入数据，我们可以使用：

```
private static final String containerSQL = "INSERT INTO container(item_name, code, type) VALUES(?, ?, ?)";
private static final String deliveryInformationSQL = "INSERT INTO delivery_information(item_name, city, staff_name) VALUES(?, ?, ?)";
private static final String retrievalInformationSQL = "INSERT INTO retrieval_information(item_name, city, staff_name) VALUES(?, ?, ?)";
private static final String exportInformationSQL = "INSERT INTO export_information(item_name, city, tax, staff_name) VALUES(?, ?, ?, ?)";
private static final String importInformationSQL = "INSERT INTO import_information(item_name, city, tax, staff_name) VALUES(?, ?, ?, ?)";
private static final String itemSQL = "INSERT INTO item(name, type, price, state) VALUES(?, ?, ?, ?)";
private static final String shipSQL = "INSERT INTO ship(item_name, ship_name, company) VALUES(?, ?, ?)";
private static final String staffSQL = "INSERT INTO staff(name, password, type, city, gender, phone_number, birth_year, company) VALUES(?, ?, ?, ?, ?, ?, ?, ?)";
```

开启数据库表间约束检查，我们可以使用：

```
this.rootConn.prepareStatement(sql: "ALTER TABLE delivery_information ENABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE retrieval_information ENABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE export_information ENABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE import_information ENABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE ship ENABLE TRIGGER ALL").execute();
this.rootConn.prepareStatement(sql: "ALTER TABLE container ENABLE TRIGGER ALL").execute();
```

## 3.2 SUSTC 部门经理用户

在后续的用户相关 API 中，我们都需要判断用户的登录状态，这将使用 root 权限（无论是数据库中的什么角色，直接拥有对 `password` 的直接查询权限非常不妥，因此在此我们采用root权限，并非将每个角色都赋予查询 `staff` 表中的 `password` 字段的权限），在 `staff` 表中查询用户的 `name`、`type`、`password` 并与给出值进行比较。

若用户登录不成功，则返回非法状态，如 `null`、`false` 或 `-1`。我们在 `private boolean checkUser(LogInfo logInfo)` 方法中完成这一过程：

```
1 private static final String checkUserSQL = "SELECT * FROM staff where name = ? AND
2   password = ? ";
3 private boolean checkUser(LogInfo logInfo) throws SQLException {
4     PreparedStatement statement = this.rootConn.prepareStatement(checkUserSQL);
5     statement.setString(1, logInfo.name());
6     statement.setString(2, logInfo.password());
7     ResultSet rs = statement.executeQuery();
8     if (!rs.next()) {
9         return false;
10    }
11 }
```

在SUSTC 部门经理用户的 8 个 API (即 API 3 ~ API 10) 中使用且仅使用 SUSTC 部门经理用户 SUSTC Department Manager User 权限。

3. `int getCompanyCount(LogInfo log);`

具体代码请见 `public int getCompanyCount(LogInfo logInfo)` 方法

使用 SQL 语句，进行查询即可：

```
private static final String getCompanyCountSQL = "SELECT count(*) FROM (SELECT DISTINCT company FROM staff WHERE company IS NOT NULL) tb";
```

4. `int getCityCount(LogInfo log);`

具体代码请见 `public int getCityCount(LogInfo logInfo)` 方法

使用 SQL 语句，进行查询即可：

```
private static final String getCityCountSQL = "SELECT count(*) from (" + "    SELECT DISTINCT city FROM (" + "        SELECT DISTINCT city FROM retrieval_information UNION DISTINCT" + "        (SELECT DISTINCT city FROM delivery_information) UNION DISTINCT" + "        (SELECT DISTINCT city FROM export_information) UNION DISTINCT" + "        (SELECT DISTINCT city from import_information)" + "    )t1)t0;";
```

5. `int getCourierCount(LogInfo log);`

具体代码请见 `public int getCourierCount(LogInfo logInfo)` 方法

使用 SQL 语句，进行查询即可：

```
private static final String getCourierCountSQL = "SELECT count(*) FROM staff WHERE type = 'Courier'";
```

6. `int getShipCount(LogInfo log);`

具体代码请见 `public int getShipCount(LogInfo logInfo)` 方法

使用 SQL 语句，进行查询即可：

```
private static final String getShipCountSQL = "SELECT count(*) FROM (SELECT DISTINCT ship_name FROM ship WHERE ship_name IS NOT NULL)t1";
```

7. `ItemInfo getItemInfo(LogInfo log, String name);`

具体代码请见 `public ItemInfo getItemInfo(LogInfo logInfo, String s)` 方法

使用 SQL 语句，进行查询，并将返回值组织成为 `ItemInfo` 类格式的数据即可：

```
private static final String getItemSQL = "SELECT * FROM item WHERE name = ?";  
private static final String getImportSQL = "SELECT city, staff_name, tax FROM import_information WHERE item_name = ?";  
private static final String getExportSQL = "SELECT city, staff_name, tax FROM export_information WHERE item_name = ?";  
private static final String getRetrievalSQL = "SELECT city, staff_name FROM retrieval_information WHERE item_name = ?";  
private static final String getDeliverySQL = "SELECT city, staff_name FROM delivery_information WHERE item_name = ?";
```

8. `ShipInfo getShipInfo(LogInfo log, String name);`

具体代码请见 `public ShipInfo getShipInfo(LogInfo logInfo, String s)` 方法

使用 SQL 语句，进行查询，并将返回值组织成为 `ShipInfo` 类格式的数据即可：

```
private static final String getShipInfoSQL = "SELECT item_name, company FROM ship WHERE ship_name = ?";
```

9. `ContainerInfo getContainerInfo(LogInfo log, String code);`

具体代码请见 `public ContainerInfo getContainerInfo(LogInfo logInfo, String s)` 方法

使用 SQL 语句，进行查询，并将返回值组织成为 `ContainerInfo` 类格式的数据即可：

```
private static final String getContainerInfoSQL = "SELECT item_name, type FROM container WHERE code = ?";
```

10. `StaffInfo getStaffInfo(LogInfo log, String name);`

具体代码请见 `public StaffInfo getStaffInfo(LogInfo logInfo, String s)` 方法

使用 SQL 语句，进行查询，并将返回值组织成为 `StaffInfo` 类格式的数据即可：

```
private static final String getStaffInfoSQL = "SELECT * FROM staff WHERE name = ?";
```

### 3.3 快递员用户

11. `bool newItem(LogInfo log, ItemInfo item);`

具体代码请见 `public boolean newItem(LogInfo logInfo, ItemInfo itemInfo)` 方法

首先，先根据如下 SQL 语句以及给出的 `ItemInfo` 类状态 `itemInfo` 判断当前新增的项目是否合法：

```
private static final String checkItemSQL = "select case (select count(*) from item where name = ?)" +
    " when 0 then true" +
    " else false" +
    " end";

private static final String checkStaffCity = "select case (select count(*) from staff where name = ? and city = ?)" +
    " when 0 then false" +
    " else true" +
    " end";
```

然后再使用如下 SQL 语句，判断给出的 `itemInfo` 的税率是否符合对应进出口城市对应种类物品的税率。

```
private static final String getImportTaxRateSQL = "with" +
    " tmp1 as (select name, price from item where type = ?)," +
    " tmp2 as (select item_name, tax from import_information where city = ?)" +
    " select case (select count(*) from tmp1) when 0 then -1 else (" +
    "     select case (select count(*) from tmp2) when 0 then -1 else (" +
    "         select sum(tax)/sum(price) as ImportTaxRate from (" +
    "             select tmp1.name, tmp1.price, tmp2.tax from tmp1" +
    "                 inner join tmp2" +
    "                     on tmp1.name = tmp2.item_name) as foo" +
    "     )" +
    "     end" +
    " )" +
    " end";
```

```

private static final String getExportTaxRateSQL = "with" +
    "    tmp1 as (select name, price from item where type = ?)," +
    "    tmp2 as (select item_name, tax from export_information where city = ?)" +
    "    select case (select count(*) from tmp1) when 0 then -1 else (" +
    "        select case (select count(*) from tmp2) when 0 then -1 else (" +
    "            select sum(tax)/sum(price) as ExportTaxRate from (" +
    "                select tmp1.name, tmp1.price, tmp2.tax from tmp1" +
    "                    inner join tmp2" +
    "                        on tmp1.name = tmp2.item_name) as foo" +
    "        )" +
    "    end" +
    ")" +
    "end";

```

若上述两项检查都有效，则使用如下 SQL 语句，将该条数据导入数据库：

```

private static final String insertNewItemSQL2 = "insert into delivery_information (item_name, city, staff_name) values (?, ?, ?)";
private static final String insertNewItemSQL3 = "insert into export_information (item_name, city, tax, staff_name) values (?, ?, ?, ?)";
private static final String insertNewItemSQL4 = "insert into import_information (item_name, city, tax, staff_name) values (?, ?, ?, ?)";
private static final String insertNewItemSQL5 = "insert into item (name, type, price, state) values (?, ?, ?, ?)";
private static final String insertNewItemSQL6 = "insert into retrieval_information (item_name, city, staff_name) values (?, ?, ?)";

```

12. `bool setItemState(LogInfo log, String name, ItemState s);`

具体代码请见 `public boolean setItemState(LogInfo logInfo, String name, ItemState s)` 方法

使用如下 API 检查当前项目状态，再根据 快递员 所拥有的权限，尝试更新状态。

```

private static final String getItemStateSQL = "select state from item where name = ?";
private static final String getItemRetrievalCourier = "select staff_name from retrieval_information where item_name = ?";
private static final String getItemDeliveryCourierAndCity = "select staff_name, city from delivery_information where item_name = ?";
private static final String updateItemState = "update item set state = ? where name = ?";
private static final String updateItemDeliveryCourier = "update delivery_information set staff_name = ? where item_name = ?";
private static final String checkItemDeliveryCourier = "select case " +
    "(select count(*) from staff where name = ? and city = ?) when 0 then false" +
    " else true end";

```

### 3.4 公司经理用户

13. `double getImportTaxRate(LogInfo log, String city, String itemClass);`

具体代码请见 `public double getImportTaxRate(LogInfo logInfo, String city, String itemType)` 方法

使用 SQL 语句，进行查询即可：

```
private static final String getImportTaxRateSQL = "with" +
    "    tmp1 as (select name, price from item where type = ?)," +
    "    tmp2 as (select item_name, tax from import_information where city = ?)" +
    "    select case (select count(*) from tmp1) when 0 then -1 else (" +
    "        select case (select count(*) from tmp2) when 0 then -1 else (" +
    "            select sum(tax)/sum(price) as ImportTaxRate from (" +
    "                select tmp1.name, tmp1.price, tmp2.tax from tmp1" +
    "                    inner join tmp2" +
    "                        on tmp1.name = tmp2.item_name) as foo" +
    "        )" +
    "    end" +
    ")" +
    "end";
```

14. `double getExportTaxRate(LogInfo log, String city, String itemClass);`

具体代码请见 `public double getExportTaxRate(LogInfo logInfo, String city, String itemType)` 方法

使用 SQL 语句，进行查询即可：

```
private static final String getExportTaxRateSQL = "with" +
    "    tmp1 as (select name, price from item where type = ?)," +
    "    tmp2 as (select item_name, tax from export_information where city = ?)" +
    "    select case (select count(*) from tmp1) when 0 then -1 else (" +
    "        select case (select count(*) from tmp2) when 0 then -1 else (" +
    "            select sum(tax)/sum(price) as ExportTaxRate from (" +
    "                select tmp1.name, tmp1.price, tmp2.tax from tmp1" +
    "                    inner join tmp2" +
    "                        on tmp1.name = tmp2.item_name) as foo" +
    "        )" +
    "    end" +
    ")" +
    "end";
```

15. `bool loadItemToContainer(LogInfo log, String itemName, String containerCode);`

具体代码请见 `public boolean loadItemToContainer(LogInfo logInfo, String itemName, String containerCode)` 方法

首先使用下列 SQL 语句先找出当前项目所对应的 `container code` 是否为 null 或者与当前给出箱子编号一样。

```
private static final String getItemContainerCode = "select code from container where item_name = ?";
private static final String checkContainerCode = "select case (select count(*) from container where code = ?) when 0 then false else true end";
```

然后使用下列 SQL 语句查看当前的箱子是否正在运输货物。

```
private static final String checkPackingContainerSQL = "with" +
    "    tmp1 as (select name from item where state = 'Packing to Container' or state = 'Shipping' or state = 'Unpacking from Container')," +
    "    tmp2 as (select item_name from container where code = ? and item_name != ?)" +
    "    select case (select count(tmp1.name) from tmp1 inner join tmp2 on tmp1.name = tmp2.item_name) when 0 then true else false end";
```

若检查当前合法，则使用下列 SQL 语句查询箱子的类型，并更新项目的状态。

```
private static final String getContainerTypeSQL = "select type from container where code = ?";  
  
private static final String updateItemContainerCode = "update container set code = ?, type = ? where item_name = ?";  
  
16. bool loadContainerToShip(LogInfo log, String shipName, String containerCode);
```

具体代码请见 `public boolean loadContainerToShip(LogInfo logInfo, String shipName, String containerCode)` 方法

首先使用下列 SQL 语句找到所有处于 `Packing to Container` 状态并且 container code 为当前值的项目，这意味着货物以及与当前的箱子绑定，最多查询结果为一项。

```
private static final String getItemNameFromContainerCodeSQL = "with" +  
    "    tmp1 as (select item_name from container where code = ?)," +  
    "    tmp2 as (select name from item where state = 'Packing to Container') " +  
    "    select tmp1.item_name from tmp1 inner join tmp2 on tmp2.name = tmp1.item_name;" ;
```

然后使用下列 SQL 语句查询一艘船是否在航行。

```
private static final String checkShipSailing = "with" +  
    "    tmp1 as (select item_name from ship where ship_name = ?)," +  
    "    tmp2 as (select name from item where state = 'Shipping') " +  
    "    select case (select count(*) from tmp1) when 0 then false else (" +  
    "        select case (select count(tmp1.item_name) from tmp1 inner join tmp2 on tmp1.item_name = tmp2.name)" +  
    "        when 0 then true else false end" +  
    "    ) end";
```

然后使用下列 SQL 语句获取货物、船、员工是否为同一公司。

```
private static final String getShipCompany = "select company from ship where ship_name = ?";  
private static final String getStaffCompany = "select company from staff where name = ?";  
private static final String getItemCompany = "select company from item where item_name = ?";
```

最后使用下列 SQL 语句更新货物的状态。

```
private static final String updateItemShip = "update item set ship_name = ? where item_name = ?";  
  
17. bool shipStartSailing(LogInfo log, String shipName);
```

具体代码请见 `public boolean shipStartSailing(LogInfo logInfo, String shipName)` 方法

使用如下 SQL 获取所有正在当前船上，`Waiting for Shipping` 状态的物品：

```
private static final String getWaitingShippingItems = "with" +  
    "    tmp1 as (select item_name from item where ship_name = ? and company = ?)," +  
    "    tmp2 as (select name from item where state = 'Waiting for Shipping') " +  
    "    select tmp1.item_name from tmp1 inner join tmp2 on tmp2.name = tmp1.item_name";
```

若条件合法，将这些物品的状态，转化为 `Shipping`。

```
18. bool unloadItem(LogInfo log, String item);
```

具体代码请见 `public boolean unloadItem(LogInfo logInfo, String item)` 方法。

若情况合法，修改货物状态从 `Shipping` 到 `Unpacking from Container`。

```
19. itemWaitForChecking(LogInfo log, String item);
```

具体代码请见 `public boolean itemWaitForChecking(LogInfo logInfo, String item)` 方法。

若情况合法，修改货物状态从 `Unpacking from Container` 到 `Import Checking`。

### 3.5 海关用户

20. `String[] getAllItemsAtPort(LogInfo log);`

具体代码请见 `public String[] getAllItemsAtPort(LogInfo logInfo)` 方法

使用下列 SQL 获取海关所在城市，并获得当前城市等待检查的所有货物。

```
private static final String getStaffCity = "select city from staff where name = ?";
private static final String getItemAtPort = "(select name from item where state = 'Import Checking'" +
    "INTERSECT (select item_name from import_information where city = ?))" +
"union" +
"(select name from item where state = 'Export Checking'" +
"INTERSECT (select item_name from export_information where city = ?))";
```

21. `bool setItemCheckState(LogInfo log, String itemName, bool success);`

具体代码请见 `public boolean setItemCheckState(LogInfo logInfo, String itemName, boolean success)` 方法

使用下列 SQL 获取对应物品在检查时对应的海关用户名，检查是否与当前登录者信息一致。

```
private static final String getItemExportStaff = "select staff_name from export_information where item_name = ?";
private static final String getItemImportStaff = "select staff_name from import_information where item_name = ?";
```

若没有相应海关用户名，那么当前登录信息就是其对应海关，使用如下SQL更新：

```
private static final String updateItemExportStaff = "update export_information set staff_name = ? where item_name = ?";
private static final String updateItemImportStaff = "update import_information set staff_name = ? where item_name = ?";
```

然后根据是否通过海关，更改物品的状态。

### 3.6 测试

本地测试中，上述所有 21 个 API 运行结果均正确、效率也相对较高。



## 4 前后端设计与交互

在本部分中，我们设计了图形化界面 GUI 前端，与后端进行通讯。

因此，用户只需操作图形化界面即可操作数据库，大大简化了用户使用数据库管理系统的难度。

### 4.1 前端设计

#### 4.1.1 前置资源

- Java Development Kit 17
- JavaFX SDK 17.0.2
- Ikonli-javafx-12.3.1
- JFoenix 9.0.10

#### 4.1.2 网络连接原理

- 采用 TCP/IP 协议  
使用 `java.net` 包，将一系列查询、登录等行为封装为数据包，在客户端和服务端之间收发。
- 仿照 session/cookie 的登录模式

客户端登录时发送用户名与密码，若用户名与密码正确，则由服务端生成并返回 `cookie` 供客户端后续使用，客户端后续操作通过 `cookie` 来进行。客户端每次登录时服务端都会生成唯一 `cookie`，此 `cookie` 为内存式 `cookie`，且客户端主动登出时会发送 `Logout` 包删除在服务端保存的 `session` 以节省空间。

- 为了方便，我们本次Project使用 `UUID` 作为 `cookie` 进行模拟。

#### 4.1.3 主体设计包与类说明

- `Packet.class`：所有数据包类的父类

```
1 package cn.edu.sustech.dbms2.client.packet;
2 public abstract class Packet {
3     public abstract String getContext();
4     public abstract int getCode();
5 }
```

- `packet.client` 和 `packet.server` 包

两个包皆同时存在于客户端与服务端。

- `client` 包下的类为客户端发送的数据包类，即服务端接收的数据包类；
- `server` 包下的类为服务端发送的数据包类，即客户端接收的数据包类，负责封装和解封数据包。

以下以负责创建新项目的 `packet.client.NewItemPacket.class` 以及 `packet.server.NewItemPacketInfoPacket.class` 为例。

以下为客户端的两个类代码展示：

```
1 // packet.client.NewItemPacket.class
2 public class NewItemPacket extends Packet {
3     private String context;
4     public NewItemPacket(String cookie, ItemInfo itemInfo) {
```

```
5         this.context = cookie + "$" + DBUtils.toItemInfoString(itemInfo);
6     }
7
8     @Override
9     public String getContext() {
10        return this.context;
11    }
12
13    @Override
14    public int getCode() {
15        return getStaticCode();
16    }
17
18 }
```

```
1 // packet.server.NewItemPacketInfoPacket.class
2 public class NewItemInfoPacket extends Packet {
3
4     private String context;
5     private boolean success;
6
7     public NewItemInfoPacket(String context) {
8
9         this.context = context;
10        this.success = Boolean.parseBoolean(context);
11    }
12
13    public boolean isSuccess() {
14
15        return this.success;
16    }
17
18    @Override
19    public String getContext() {
20
21        return this.context;
22    }
23
24    @Override
25    public int getCode() {
26
27        return getStaticCode();
28    }
29
30    public static int getStaticCode() {
31
32        return 20;
33    }
34 }
```

以下为服务端的两个对应类代码展示：

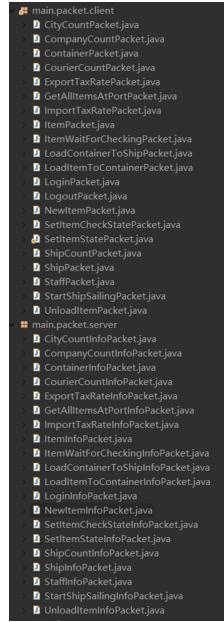
```
1 public class NewItemPacket extends Packet {
2
3     private String cookie;
4     private ItemInfo info;
5
6     public NewItemPacket(String context) {
```

```
5     String info[] = context.split("[\$]");
6     this.cookie = info[0];
7     this.info = DBUtils.loadItemInfo(info[1]);
8 }
9 public String getCookie() {
10     return this.cookie;
11 }
12 public ItemInfo getInfo() {
13     return this.info;
14 }
15 @Override
16 public String getContext() {
17     return cookie;
18 }
19 @Override
20 public int getCode() {
21     return getStaticCode();
22 }
23 public static int getStaticCode() {
24     return 19;
25 }
26 }
```

```
1 public class NewItemInfoPacket extends Packet {
2     private String context;
3     public NewItemInfoPacket(boolean success) {
4         this.context = "" + success;
5     }
6     @Override
7     public String getContext() {
8         return this.context;
9     }
10    @Override
11    public int getCode() {
12        return getStaticCode();
13    }
14    public static int getStaticCode() {
15        return 20;
16    }
17 }
18 }
```

其中，`context` 为数据包包含的内容，`getCode()` 返回该数据包类型的唯一识别码。

所有数据包一览：



上图为服务端架构，客户端与此相同。

- **PacketManager.class** : 数据包管理类

负责识别数据包的类型并且分发到对应类，核心方法代码如下：

```

1 public Packet receivePacket(int len, byte[] packetBytes) {
2     String msg = new String(packetBytes, 0, len);
3     int index = msg.indexOf('0');
4     if (index == -1) {
5         return null;
6     }
7     int code = Integer.parseInt(msg.substring(0, index));
8     String context = msg.substring(index + 1);
9     Class<? extends Packet> packetClazz = packetCodes.get(code);
10    try {
11        Constructor<? extends Packet> constructor =
12            packetClazz.getConstructor(String.class);
13        return constructor.newInstance(context);
14    } catch (Exception e) {
15        ThrowableHandler.handleThrowable(e);
16    }
17    return null;
}

```

- **DBClient.class**: 客户端核心类

负责链接客户端和收发包，主要代码如下：

```

1 public Packet sendAndReceivePacket(Packet packet) throws IOException {
2     try {
3         Socket socket = new Socket();
4         socket.setSoTimeout(10000);

```

```

5         socket.setKeepAlive(true);
6         socket.setOOBInline(true);
7         socket.connect(new InetSocketAddress(host, port));
8         BufferedOutputStream writer = new
9             BufferedOutputStream(socket.getOutputStream());
10        writer.write((packet.getCode() + "@" + packet.getContext().getBytes()));
11        writer.flush();
12        BufferedInputStream input = new
13            BufferedInputStream(socket.getInputStream());
14        byte[] bytes = new byte[1024 * 8];
15        int len;
16        if ((len = input.read(bytes)) != -1) {
17            writer.close();
18            input.close();
19            socket.close();
20            return PackageManager.getInstance().receivePacket(len, bytes);
21        }
22        writer.close();
23        input.close();
24        socket.close();
25    } catch (SocketException e) {
26        e.printStackTrace();
27    }

```

- **Main.class** : 服务端核心类（即服务端主类）

负责接收客户端链接和收发包，由于代码较长，完整代码不在此展示。

#### 4.1.4 用户图形化界面

- 使用 `JavaFX`、`JFoenix` 组件进行设计，`Ikonli` 提供图标组件支持。
- 包含 `LobbyView.class` 和 `UserView.class` 两个主体界面类，分别负责职员登录和职员操作。
- `ThrowableHandler.class` 类负责错误栈输出的可视化。

#### 4.1.5 使用方法

##### 客户端

1. 下载 [Open JavaFX SDK 17.0.2](#)
2. 通过 `pom.xml` 导入 `maven` 项目（包含多个依赖，部分依赖下载需要设置 `maven` 镜像为 `alibaba` 镜像）
3. 设置虚拟机参数如下（替换 `sdk` 路径为安装路径）

```

1 --module-path "D:\javafx-sdk-17.0.2\lib"
2 --add-modules javafx.controls,javafx.fxml
3 --add-opens java.base/java.lang.reflect=ALL-UNNAMED

```

4. 运行客户端

## 服务端

- 环境同Project `Local Judge` 环境即可运行，无额外依赖包。
- 运行 `Main.class` 开启服务端即可。

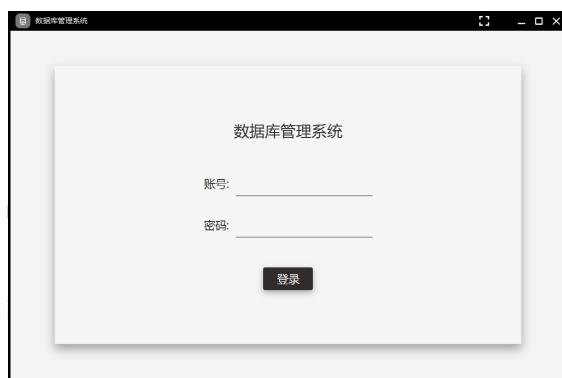
## 注意事项

- 请勿在服务端开启时间运行 `Local Judge`。同时，因为服务端没有运行数据导入方法，请先运行完 `Local Judge` 导入测试数据后开启服务端。
- 服务端重启后请重开客户端，因为所有 `session` 都被清除。
- 对于其他意料之外的错误，若无法解决。请关闭服务端和客户端，运行 `Local Judge` 重新导入数据后，再重新打开服务端和客户端，以解决错误。

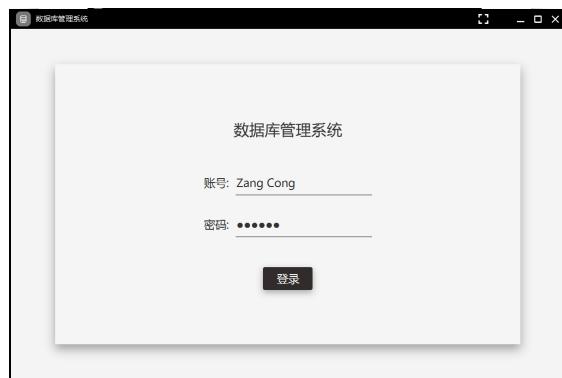
## 4.2 效果展示

### 4.2.1 登录界面

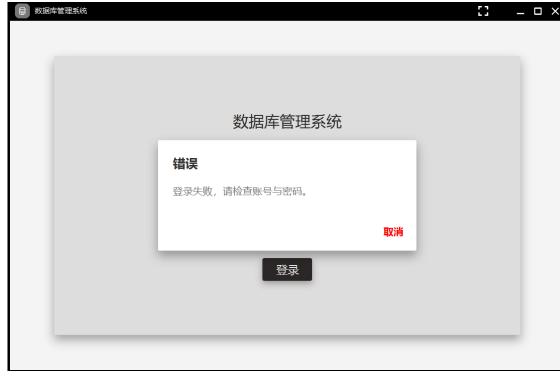
此处是数据库管理系统的登录界面，用户可以使用账号和密码进行登录。



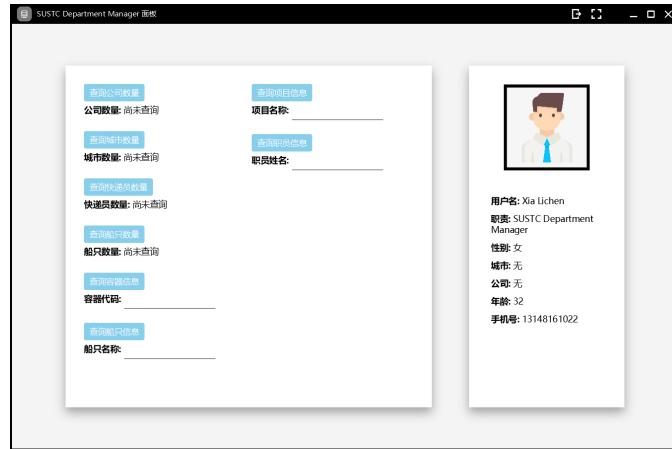
用户输入 账号 和 密码 后，点击 登录 按钮，数据库管理系统会在数据库中查询是否存在有相应账号和密码的用户。



- 如果 **不存在** 相关用户信息，前端将进行相关提示，提醒用户修改账号或密码。



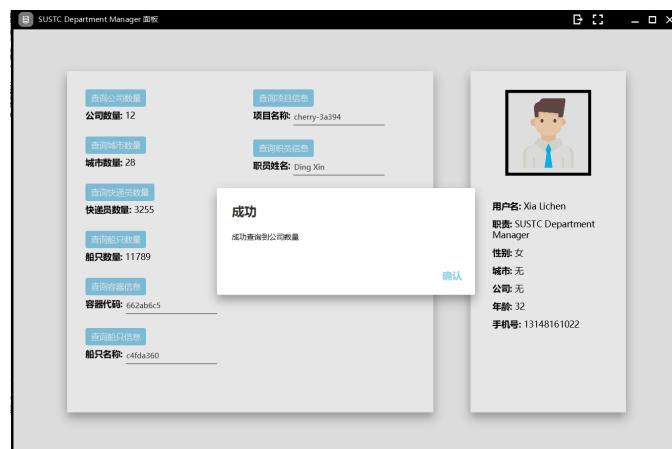
- 如果存在相关用户信息，前端会进入相应用户的对应管理面板，用户能够进行权限内的操作。



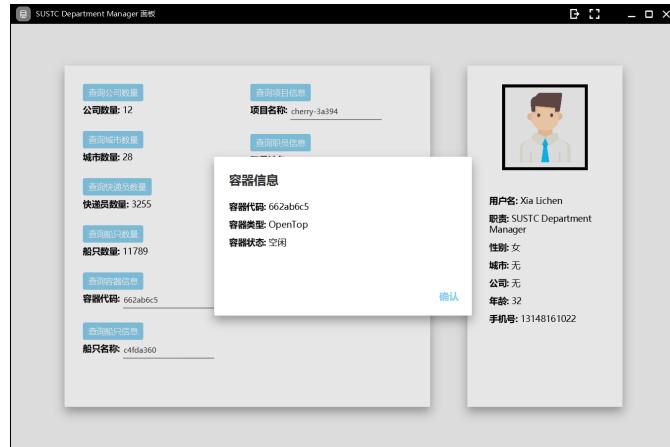
#### 4.2.2 SUSTC 部门经理用户 管理面板

该用户能实现的功能是：

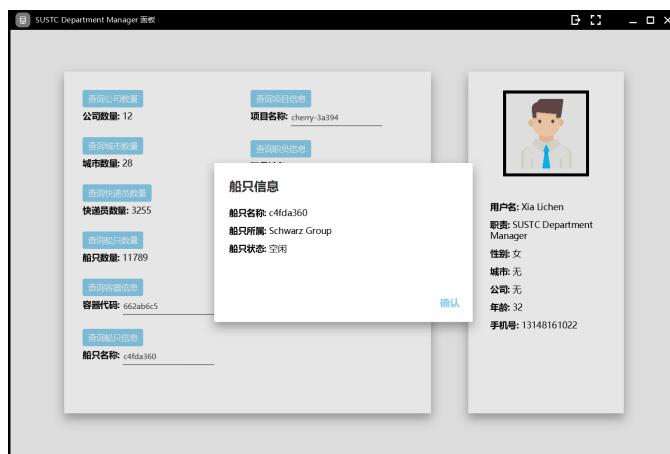
- 查询公司数量
- 查询城市数量
- 查询快递员数量
- 查询船只数量



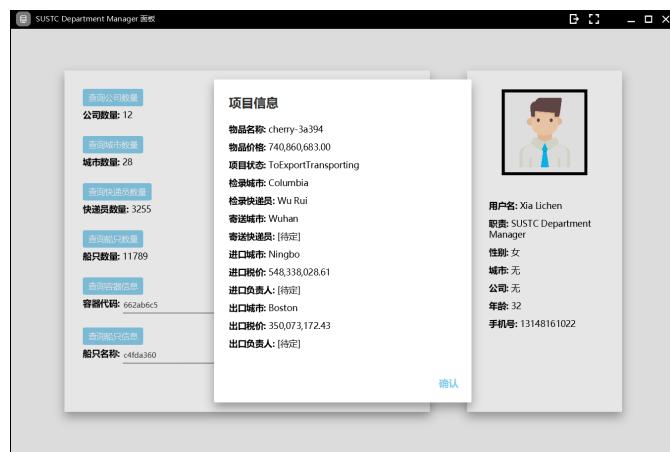
- 查询容器信息



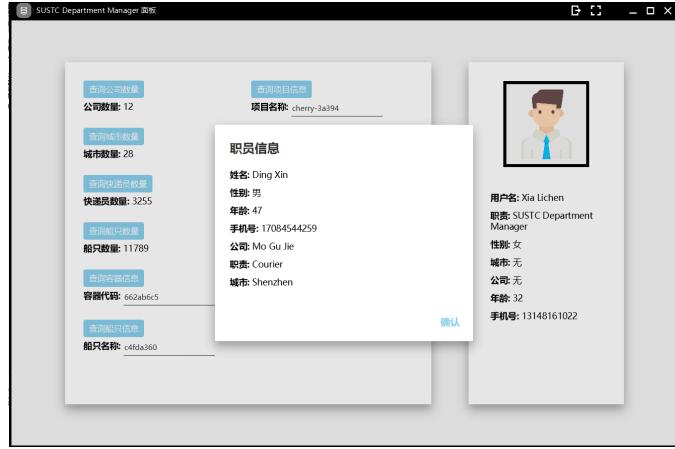
- 查询船只信息



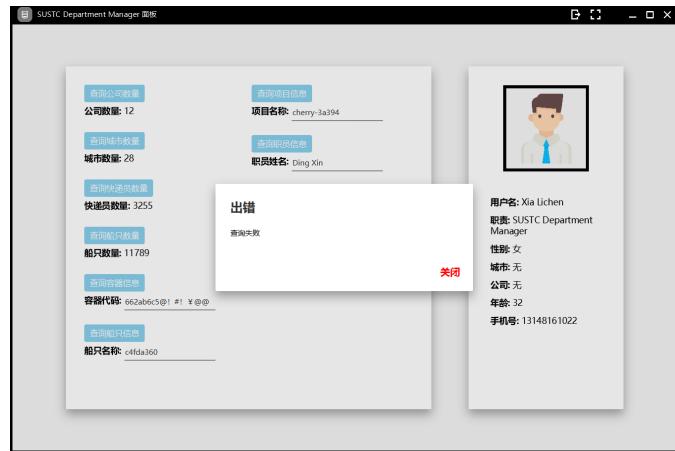
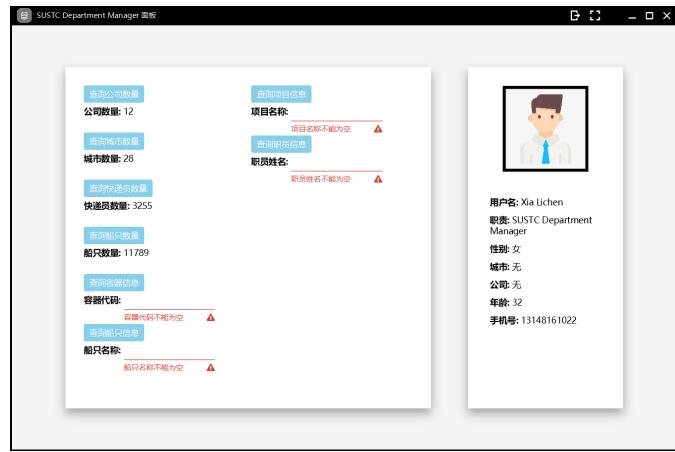
- 查询项目信息



- 查询职员信息



当然，当查询容器信息、船只信息、项目信息、职员信息时，不输入相关信息或者输入错误值时，会提示错误：

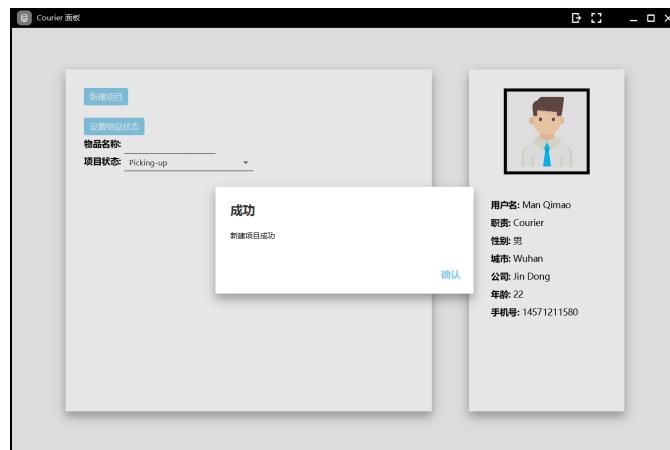
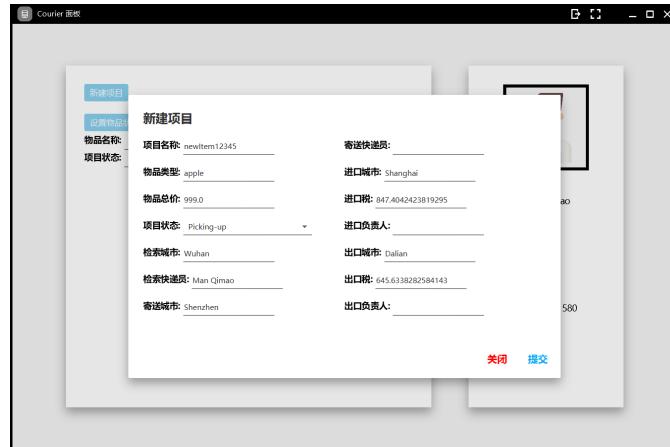


#### 4.2.3 快递员用户 管理面板

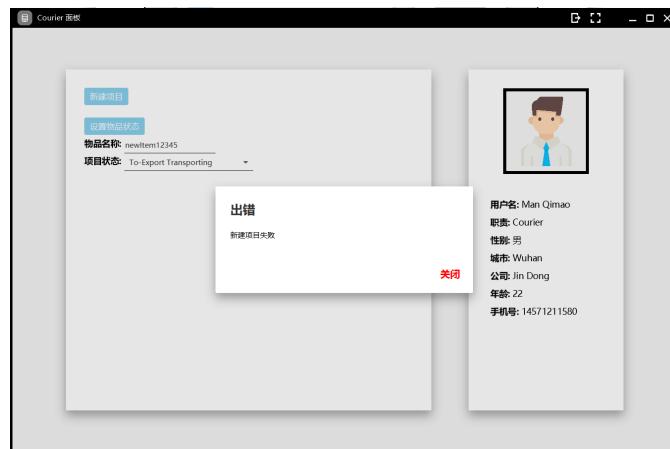
该用户能实现的功能为：

- 新建项目

当新建项目信息，符合相关要求时，会新建项目。

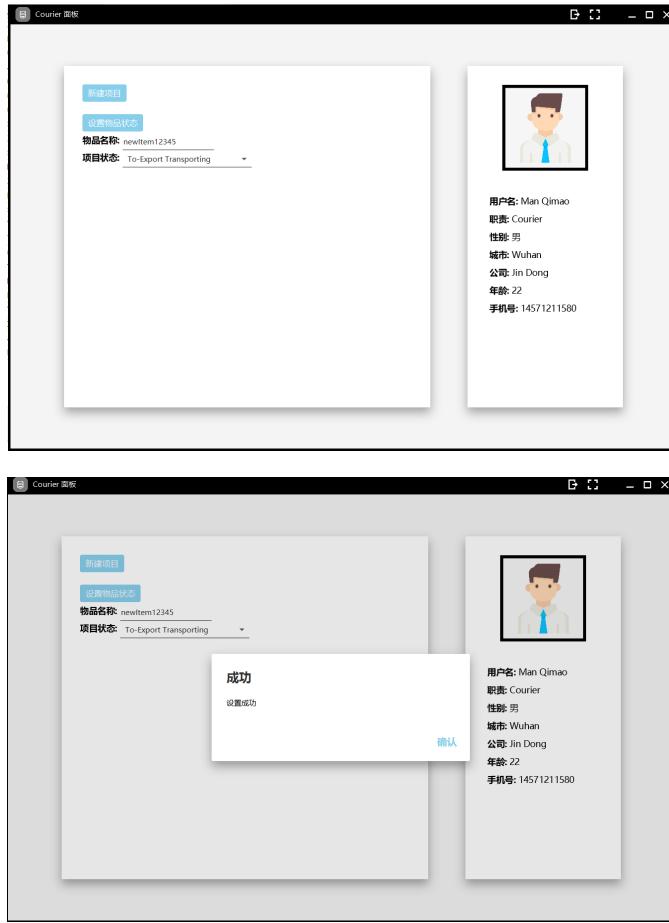


当新建项目信息，不符合相关要求时，会显示错误提示。

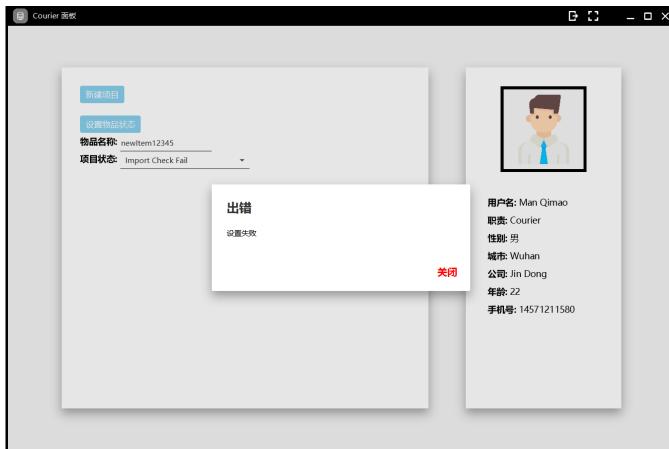


- 更改项目状态

当更改项目状态，符合相关要求时，会更改项目状态。



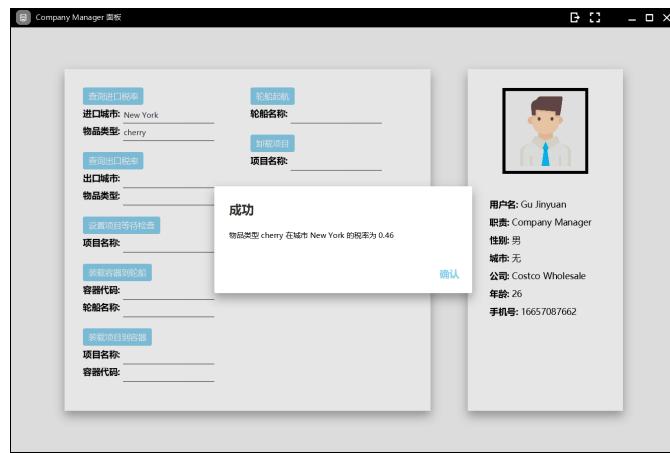
当更改项目状态，不符合相关要求时，不会更改项目状态，并提示错误。



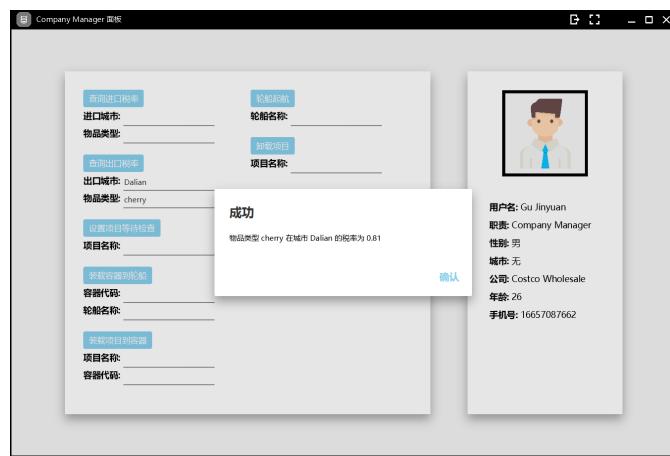
#### 4.2.4 公司经理用户 管理面板

该用户能实现的功能为：

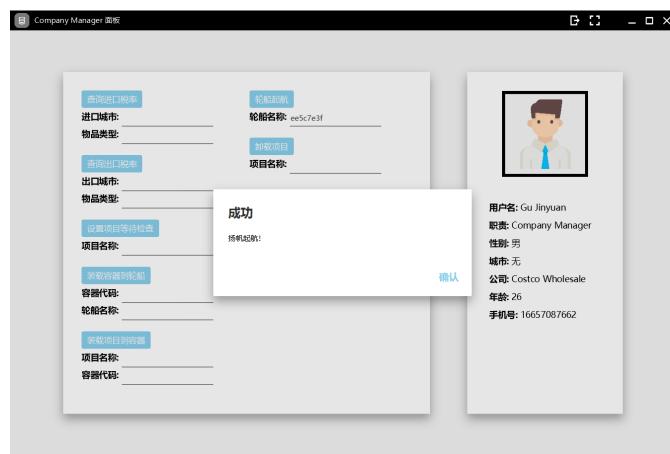
- 查询进口城市特定类型物品的税率



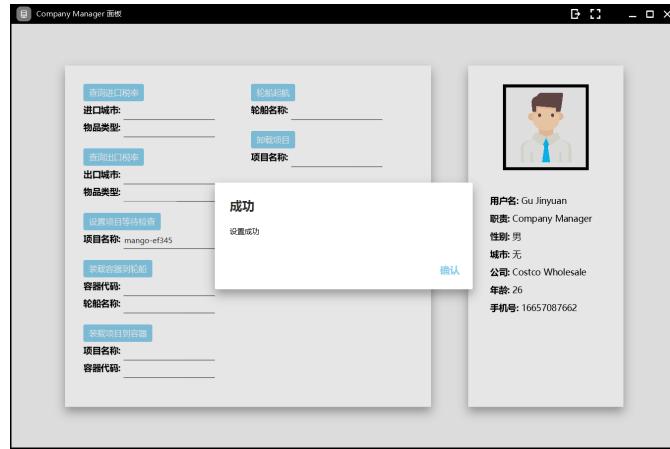
- 查询出口城市特定类型物品的税率



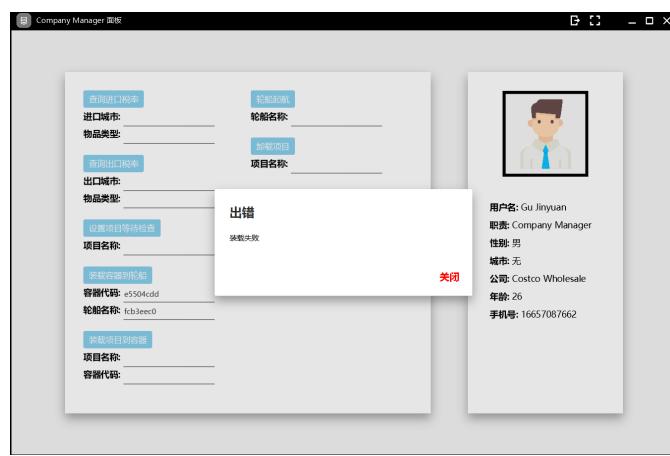
- 轮船起航



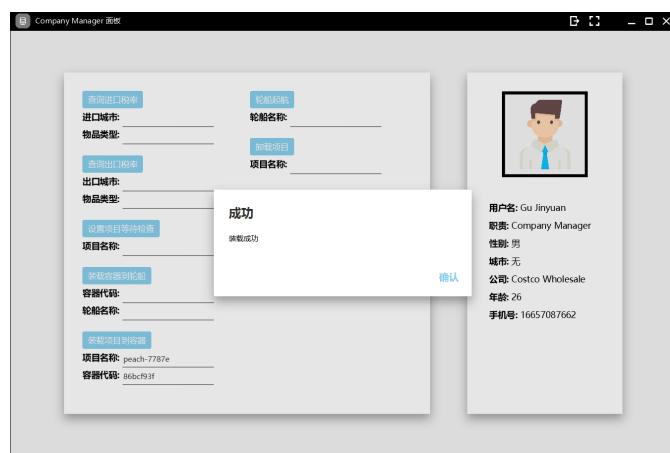
- 设置项目等待检查



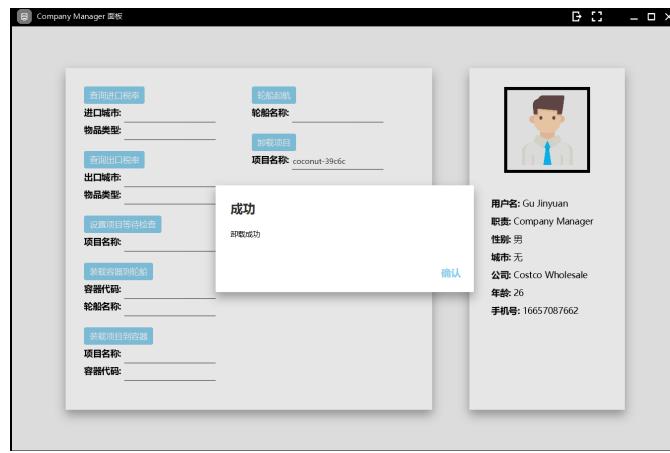
- 装载容器到轮船



- 装载项目到容器



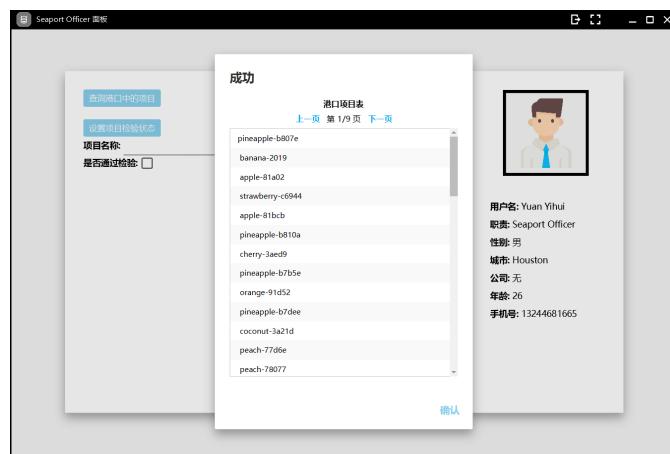
- 卸载项目



#### 4.2.5 海关用户

该用户能实现的功能为：

- 查询港口中的所有项目



- 设置海关项目检验状态

