**中山大学计算机学院**

**数据库系统**

**本科生实验报告**

**（2024学年秋季学期）**

课程名称：Database System

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | **202410380** | 专业（方向） | **计算机科学与技术** |
| 学号 | **22320107** | 姓名 | **饶鉴晟** |

# 实验题目

1. 课内实验

问题：赵老师当了2008级电子商务班的班主任，他要能查到全校的课程信息以及本班学生的选课信息，如何让他有权查到这些信息? 主要内容如下：

1. **登录管理**

为新老师创建登录账号logzhao,验证该账号与数据库的连接访问是否正确?

1. **对用户授权**

**问题1:**试解决赵老师能查询本年级学生的选课信息?

首先创建2008级学生选课信息的视图 scview,把访问该视图的权限授予赵老师，最后 验证赵老师能否访问该视图?

**问题2:**试解决让赵老师了解某课程的选课情况?

首先创建能查询指定课程选课信息的存储过程 scpro,把执行该存储过程的权限授予赵老师，最后验证赵老师能否执行存储过程?

**补充内容：**撤销赵老师查询某课程的选课情况，再验证赵老师能否执行存储过程?

1. **角色管理**

**问题：**假如学校新增10个辅导员，都要在student表中添加、修改和删除学生，要个个设置权限，方便吗?

可以考虑利用数据库的角色管理来实现：

首先创建辅导员角色m\_role,然后对角色进行插入操作授权，再创建各个辅导员的登 录以及对应的登录用户，使这些用户成为角色成员，再验证用户是否有插入操作的权限?

还可以考虑应用程序角色来实现：

创建应用程序角色，激活该角色，对其进行插入操作的授权，验证是否具有该操作的权限?

# 实验目的

设计和完成一个综合性数据库安全实践案例，深入理解PostgreSQL数据库安全性控制的关键概念和技术方法，掌握如何创建和管理用户及角色、配置权限、应用存储过程、实现行级安全策略、启用审计日志等具体操作，确保数据库的访问控制、数据传输安全及操作权限管理的有效性，

# 实验环境

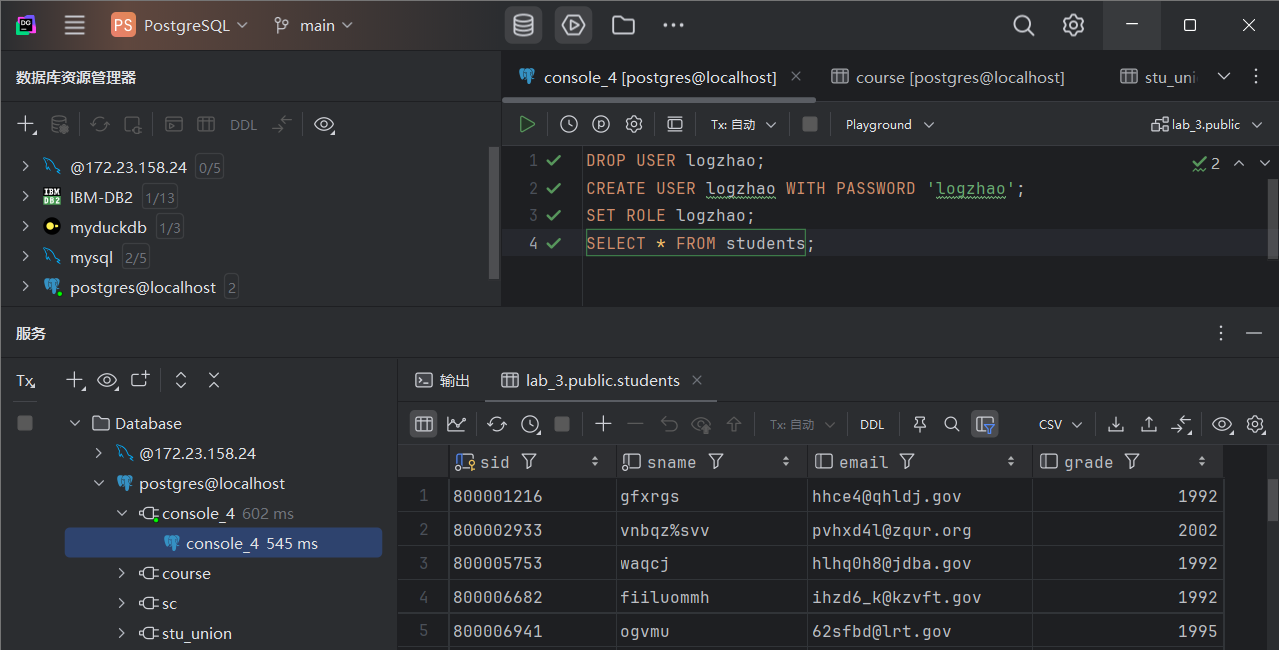
* 操作系统：Windows 11 Workstation Pro 24H2
* IDE：DataGrip 2024.2.2 Build #DB-242.21829.162
* Database：PostgreSQL 16.4, compiled by Visual C++ build 1940, 64-bit

# 实验步骤

1. 课内实验

* + 1. **为新老师创建登录账号logzhao,验证该账号与数据库的连接访问是否正确?**

**注意：postgresql中的user等价于拥有登陆权限的role。**



**图1 实验结果**

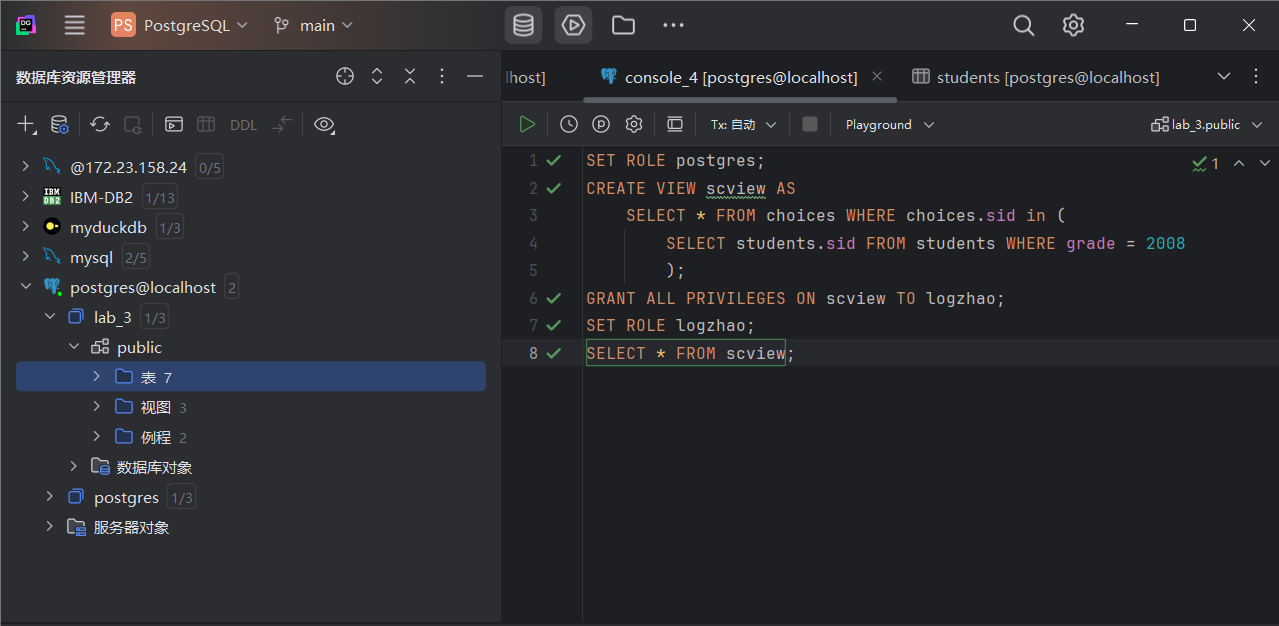
首先，通过DROP USER logzhao;删除可能已存在的用户logzhao，确保实验环境清洁，避免用户创建时发生冲突。

接着，使用CREATE USER logzhao WITH PASSWORD 'logzhao';创建了一个名为logzhao的新用户，并设置了密码logzhao。此操作为用户分配了登录权限。

然后，通过SET ROLE logzhao;切换到logzhao用户，模拟其实际登录场景，以验证新用户的连接和操作权限。

最后，执行SELECT \* FROM students;查询students表，检查用户是否具备访问权限。如果查询成功，说明用户创建及连接配置正确。

* + 1. **首先创建2008级学生选课信息的视图 scview,把访问该视图的权限授予赵老师，最后 验证赵老师能否访问该视图?**



**图2 SQL语句和执行结果**

赵老师可以访问该视图。

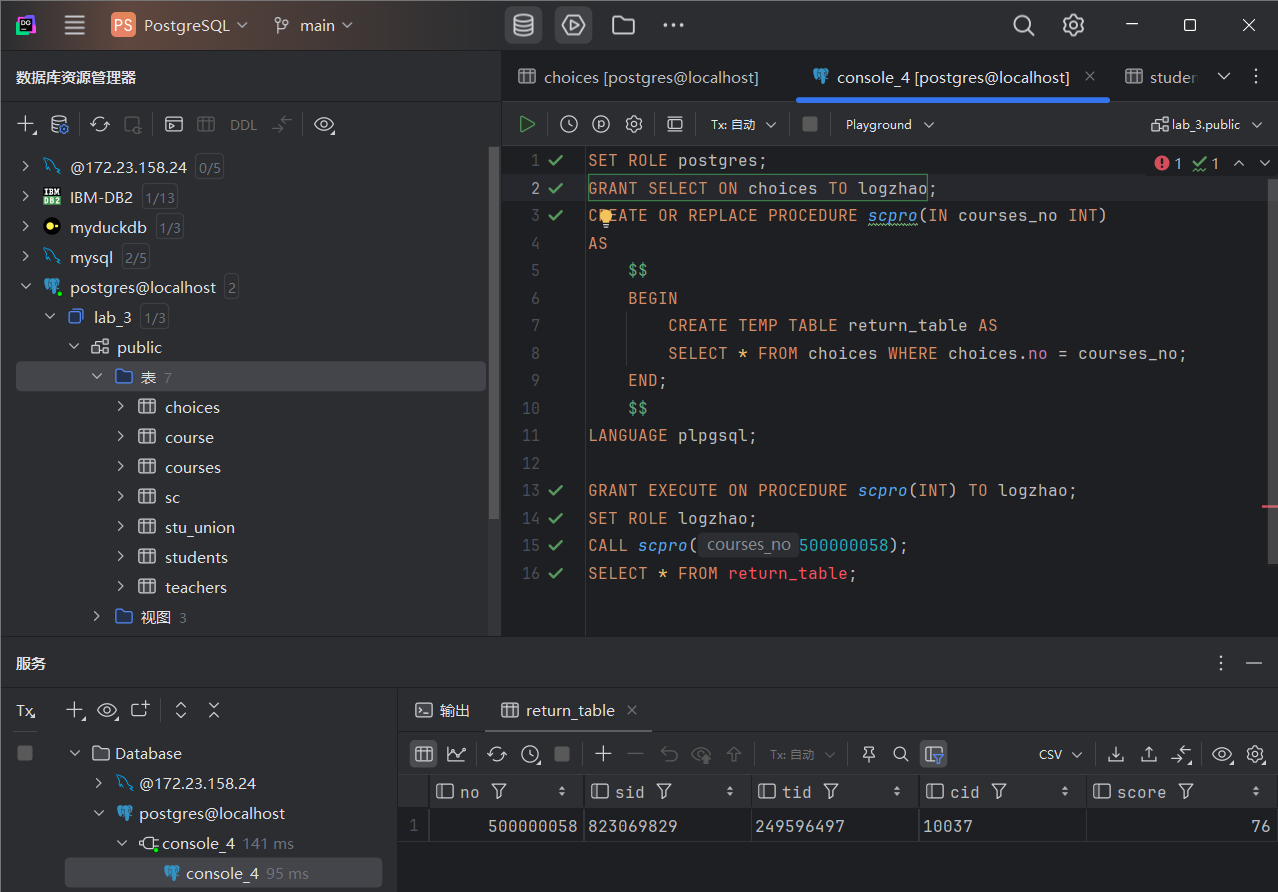
在本实验中，我们首先通过SET ROLE postgres;切换到超级用户postgres，以便进行视图创建和权限管理的操作。

接着，使用CREATE VIEW scview AS语句创建了一个名为scview的视图。该视图基于表choices和students，查询结果为2008级学生的选课信息。视图通过嵌套查询实现：内层查询从students表中选出2008级学生的sid，外层查询从choices表中筛选出这些学生的选课信息。该视图的创建简化了针对特定年级学生选课信息的复杂查询。

随后，通过GRANT ALL PRIVILEGES ON scview TO logzhao;将访问视图scview的所有权限授予用户logzhao，确保该用户可以查询视图。

切换到用户logzhao后（SET ROLE logzhao;），使用SELECT \* FROM scview;成功查询出2008级学生的选课信息，验证了用户logzhao对视图的访问权限配置正确。

* + 1. **首先创建能查询指定课程选课信息的存储过程 scpro,把执行该存储过程的权限授予赵 老师，最后验证赵老师能否执行存储过程。**



**图3 SQL语句和执行结果**

赵老师可以执行该存储过程。

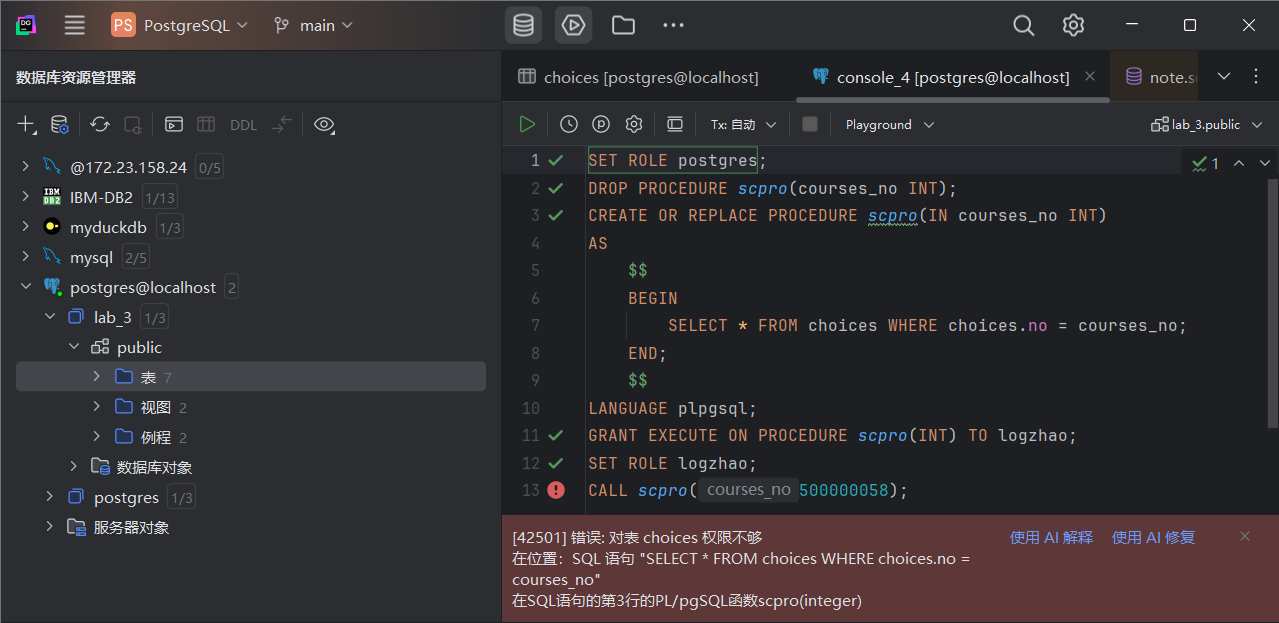
首先，通过 SET ROLE postgres; 切换到超级用户 postgres，以便执行权限管理和存储过程创建操作。随后，通过 GRANT SELECT ON choices TO logzhao; 授予用户 logzhao 对表 choices 的查询权限。这一步确保 logzhao 在执行存储过程时能够访问 choices 表中的数据。

接着，使用 CREATE OR REPLACE PROCEDURE 创建了存储过程 scpro，该存储过程接收一个输入参数 courses\_no，用于筛选课程编号为 courses\_no 的选课信息，并将查询结果存储到一个名为 return\_table 的临时表中。临时表的作用域为当前数据库会话，仅对会话内的查询可见。

随后，通过 GRANT EXECUTE ON PROCEDURE scpro(INT) TO logzhao; 授予用户 logzhao 执行存储过程的权限。这一步允许 logzhao 调用 scpro 进行选课信息的查询。

切换到用户 logzhao（SET ROLE logzhao;）后，执行存储过程 CALL scpro(500000058); 成功调用了存储过程，并根据指定课程编号 500000058 筛选了选课信息。紧接着，执行 SELECT \* FROM return\_table; 查询临时表，成功获取到了指定课程的选课信息，验证了用户 logzhao 对存储过程和相关数据的访问权限。

* + 1. **撤销赵老师查询某课程的选课情况，再验证赵老师能否执行存储过程?**



**图4 SQL语句和执行结果**

赵老师不能执行该存储过程。

通过CREATE OR REPLACE PROCEDURE语句创建了一个名为scpro的存储过程。该存储过程接受一个输入参数courses\_no，用于查询表choices中课程编号为该参数值的所有选课信息。此存储过程的逻辑使用了PL/pgSQL语言并封装在BEGIN和END块中。

随后，使用GRANT EXECUTE ON PROCEDURE scpro(INT) TO logzhao;将执行scpro存储过程的权限授予用户logzhao，这允许其调用该存储过程。

切换到用户logzhao后（SET ROLE logzhao;），我尝试调用存储过程CALL scpro(500000058);以查询指定课程的选课信息。然而，执行过程中返回了权限不足的错误。这表明用户logzhao缺少对表choices的SELECT权限，存储过程中的查询无法成功执行。这个结果是显然的，因为我撤销了赵老师查询选课记录表的权限。

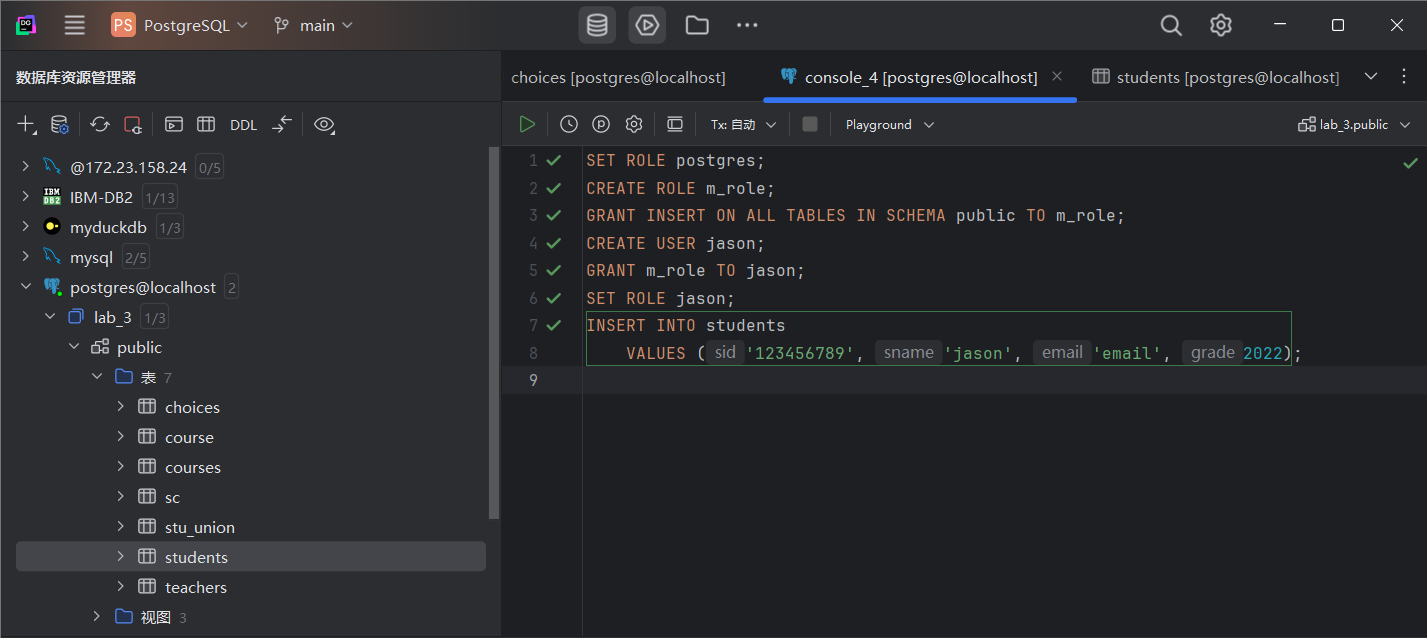
* + 1. **假如学校新增10个辅导员，都要在student表中添加、修改和删除学生，要个个 设置权限，方便吗?**

如果学校新增10个辅导员，要求他们都能够在student表中添加、修改和删除学生记录，那么逐一为每个辅导员设置权限会变得非常繁琐。对于每个用户，我们需要分别授予INSERT、UPDATE和DELETE等权限。这种逐个操作的方法不仅费时费力，还容易出错，比如遗漏某些权限或配置不一致。

为了解决这种问题，数据库提供了角色（Role）的机制。可以创建一个角色，例如student\_manager，为该角色一次性赋予对student表的所有必要权限，然后将10个辅导员分配到这个角色中。这样一来，只需要管理角色的权限即可，无需为每个用户单独设置。当新增或移除辅导员时，只需将用户加入或移出该角色，无需重新配置权限。

使用角色的好处是提高了权限管理的灵活性和效率，特别是在用户数量多或权限要求复杂的场景下，角色机制显得更加简洁和规范。通过这一机制，可以有效减少手动操作的冗余，同时降低出错的风险。

* + 1. **首先创建辅导员角色m\_role,然后对角色进行插入操作授权，再创建各个辅导员的登录以及对应的登录用户，使这些用户成为角色成员，再验证用户是否有插入操作的权限?**

****

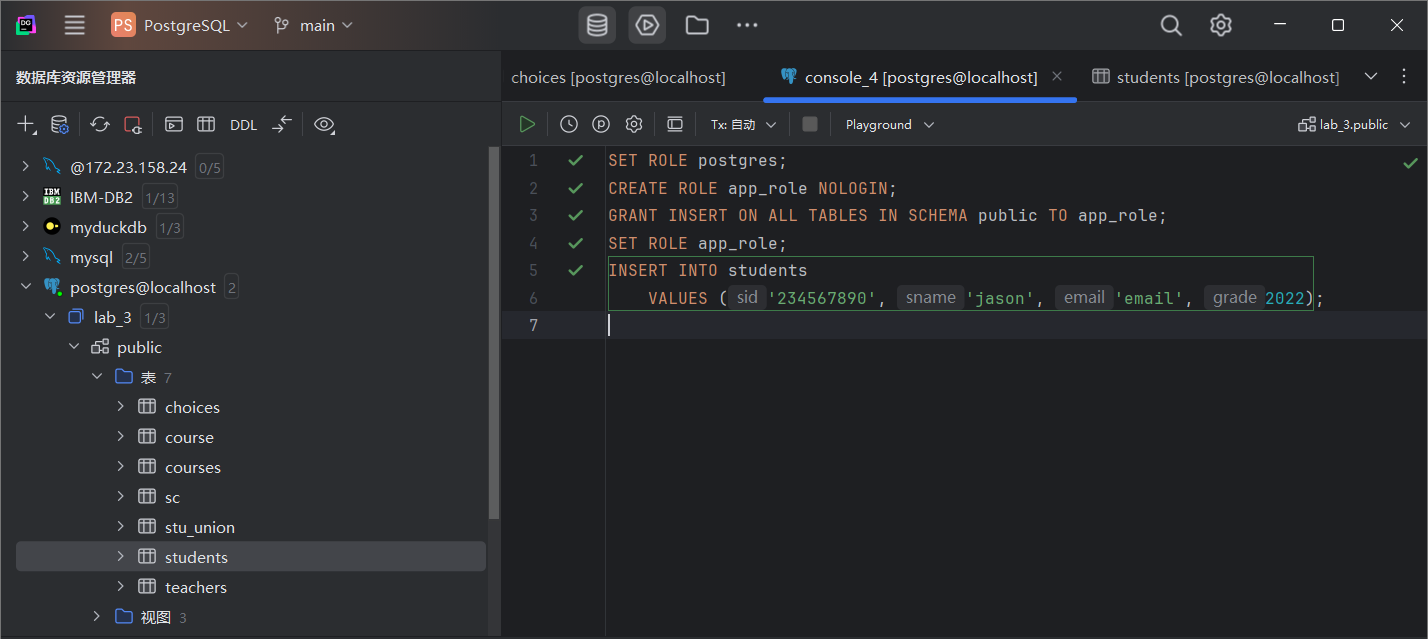
**图5 SQL语句和执行结果**

在本实验中，使用CREATE ROLE m\_role;创建了一个名为m\_role的角色。随后授予该角色对public模式中所有表的插入权限。这一步确保了拥有m\_role角色的用户可以在public模式的表中进行数据插入操作。

然后创建了一个登录用户jason，并将角色m\_role赋予用户jason，使其成为该角色的成员。这样，用户jason继承了m\_role的插入权限。

接下来，执行插入语句成功将一条新记录插入到students表中。这表明用户jason通过继承m\_role的权限，具备了对表students进行插入操作的能力。

* + 1. **创建应用程序角色，激活该角色，对其进行插入操作的授权，验证是否具有该操作的 权限?**



**图6 SQL语句和执行结果**

在本实验中，我们通过创建和激活一个应用程序角色 app\_role，验证了其插入操作的权限。以下是代码的详细解释和实验结果：

首先，通过 SET ROLE postgres; 切换到超级用户 postgres，以确保具备执行角色创建和权限授予操作的权限。

接着，使用 CREATE ROLE app\_role NOLOGIN; 创建了一个名为 app\_role 的应用程序角色。NOLOGIN 属性表明该角色不能直接登录，而是需要通过其他用户在会话中显式激活。这种设计通常用于在特定场景中临时启用角色的权限。

随后，使用 GRANT INSERT ON ALL TABLES IN SCHEMA public TO app\_role; 授予 app\_role 对 public 模式中所有表的插入权限。这一步确保了角色拥有插入数据的能力。

通过 SET ROLE app\_role; 激活了 app\_role，当前会话的权限切换为 app\_role 的权限。激活后，执行插入操作，成功向 students 表插入了一条新的学生记录。由于 app\_role 已被正确授予插入权限，该操作可以正常执行。

# 参考资料

[1] PostgreSQL Global Development Group. (2023).

*PostgreSQL 16 Documentation*. Available at:

<https://www.postgresql.org/docs/16/index.html>

[2] Oracle Corporation. (2019). *Oracle® Database SQL Language Reference and Standard, 19c*. Available at:

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/SQL-Standards.html>