北京郵電大學

实验报告：GaussDB 数据库实验 4-7

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

学号：2022211683

姓名：张晨阳

目录

实验四 创建和管理用户

一、实验目的

1. 通过实验让学生熟悉并了解 GaussDB(for openGauss)数据库的基本机制与操作。
2. 通过用户管理、表管理、数据库对象等管理的操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)。

二、实验平台及环境

- 华为云: GaussDB 2.7.2
- 设备名称：数据库
- 设备型号：GaussDB(for openGauss) 8 核 | 64 GB
- 软件版本：GaussDB(for openGauss) 2020 主备版

三、实验内容

1. 本实验通过用户管理的操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)；
2. 本实验通过表管理、数据库对象等管理的操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)；
3. 本实验通过数据库对象管理的操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)。

四、实验步骤

1. 创建用户；
2. 管理用户；

五、实验结果及分析

1 创建用户

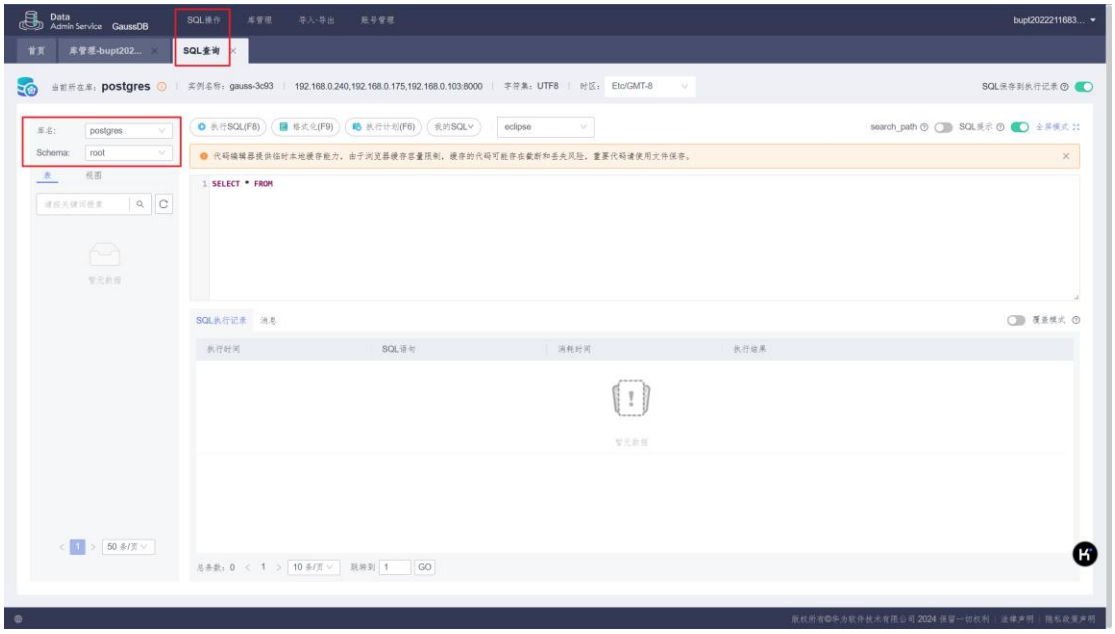
(1) 通过 CREATE USER 创建的用户，默认具有 LOGIN 权限；

(2) 通过 CREATE USER 创建用户的同时系统会在执行该命令的数据库中，为该用户创建一个同名的 SCHEMA；其他数据库中，则不自动创建同名的 SCHEMA；用户可使用 CREATE SCHEMA 命令，分别在其他数据库中，为该用户创建同名 SCHEMA；

(3) 系统管理员在普通用户同名 schema 下创建的对象，所有者为 schema 的同名用户（非系统管理员）。

a) 选择 SQL 操作，单击 SQL 查询，进入 SQL 查询页面；

b) 库名选择 postgres，Schema 选择 root：

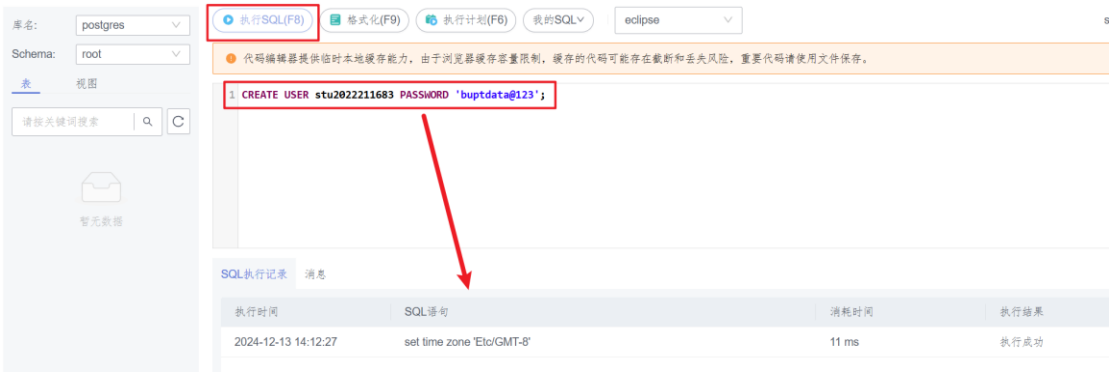


c) 创建用户。

创建用户 stu2022211683，登录密码为 buptdata@123，在 SQL 查询页面，输入如下 SQL 语句：

```
CREATE USER stu2022211683 PASSWORD 'buptdata@123';
```

截图如下：



同样的下面语句也可以创建用户。

```
CREATE USER stu2022211683 IDENTIFIED BY 'buptdata@123';
```

如果创建有“创建数据库”权限的用户，需要加 CREATEDB 关键字。

```
CREATE USER stu2022211683 CREATEDB PASSWORD 'buptdata@123';
```

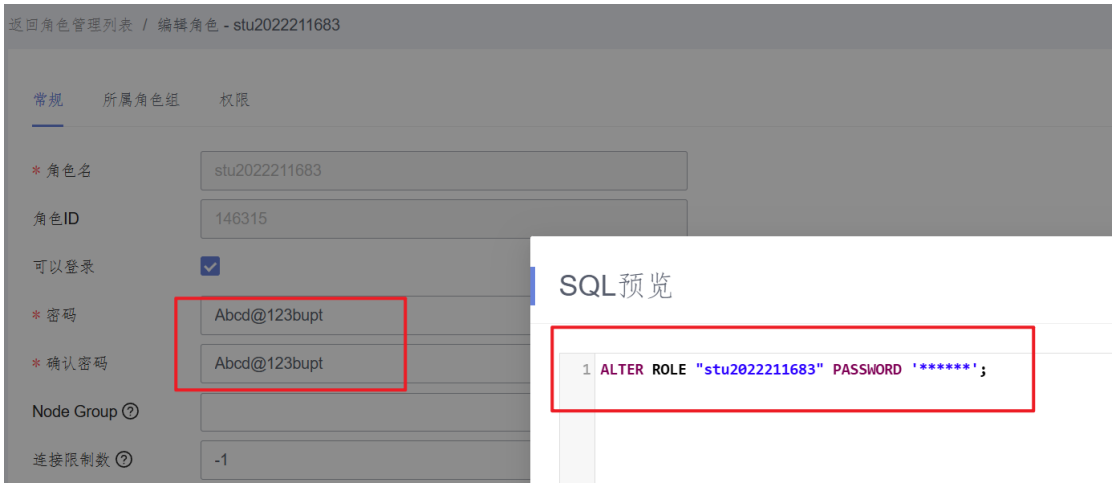
2 管理用户

（1） 选择账号管理，单击角色管理，进入角色管理页面：

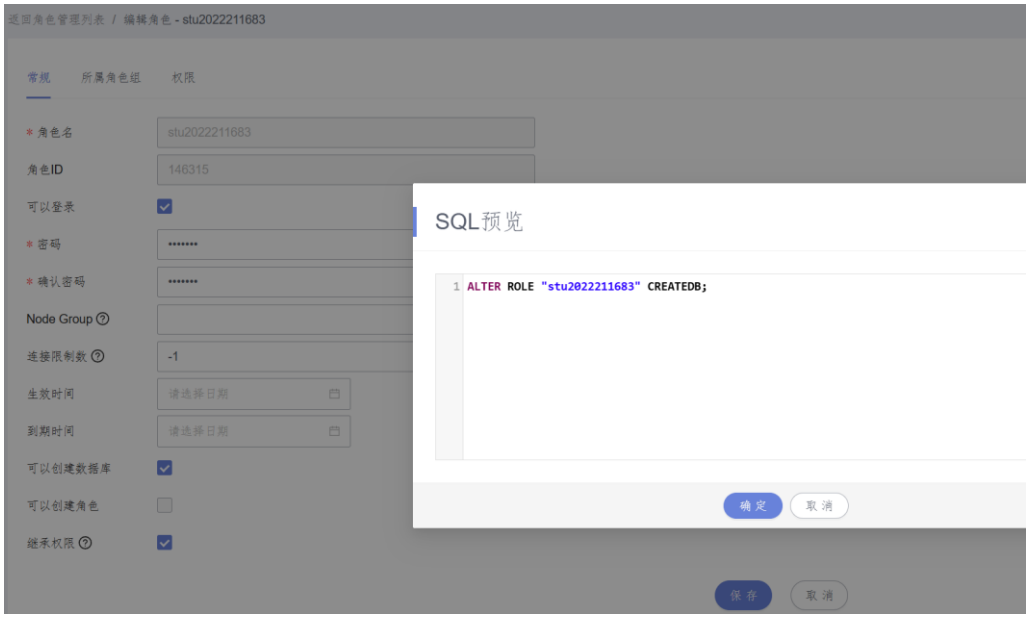
（2） 修改用户等登录密码：

a) 单击角色名 stu2022211683，进入编辑角色页面，在密码框和确认密码框输入新密码，将用户 stu2022211683 的登录密码由 buptdata@123 修改为 Abcd@123bupt，单击保存：

b) 显示 SQL 预览，单击确定，修改成功。

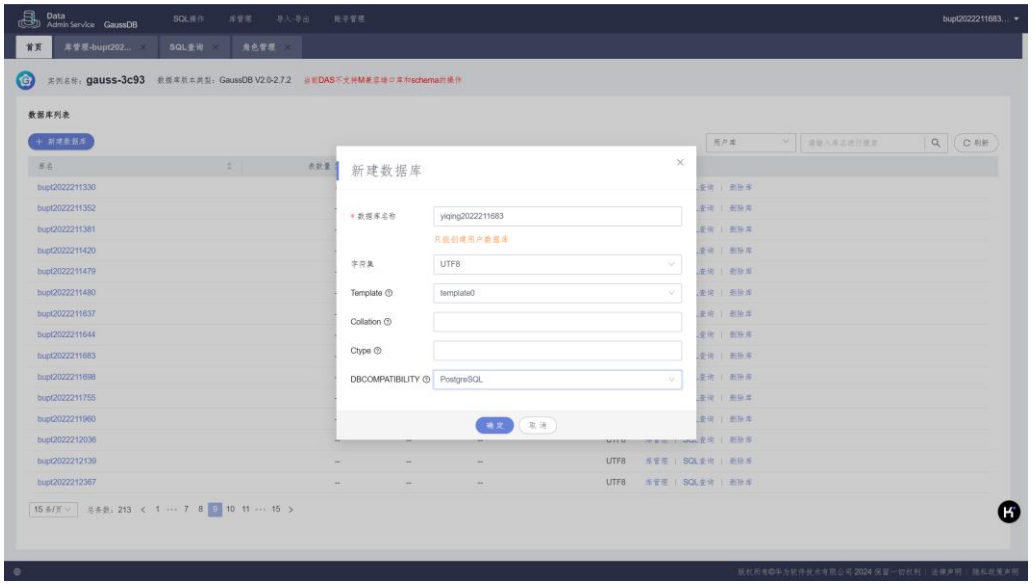


（3） 为用户 stu2022211683 追加可以创建数据库的权限。

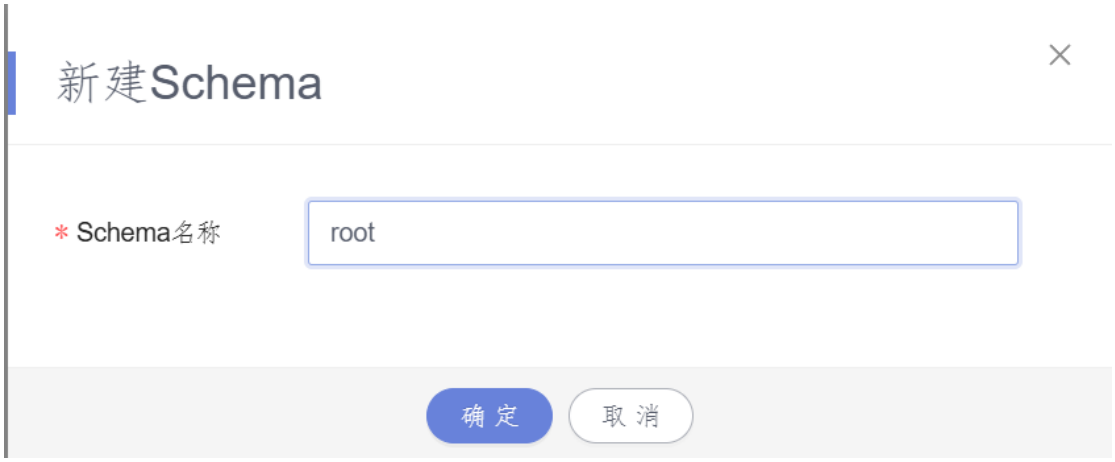


（4） 设置用户权限。

a) 创建数据库 yiqing2022211683：

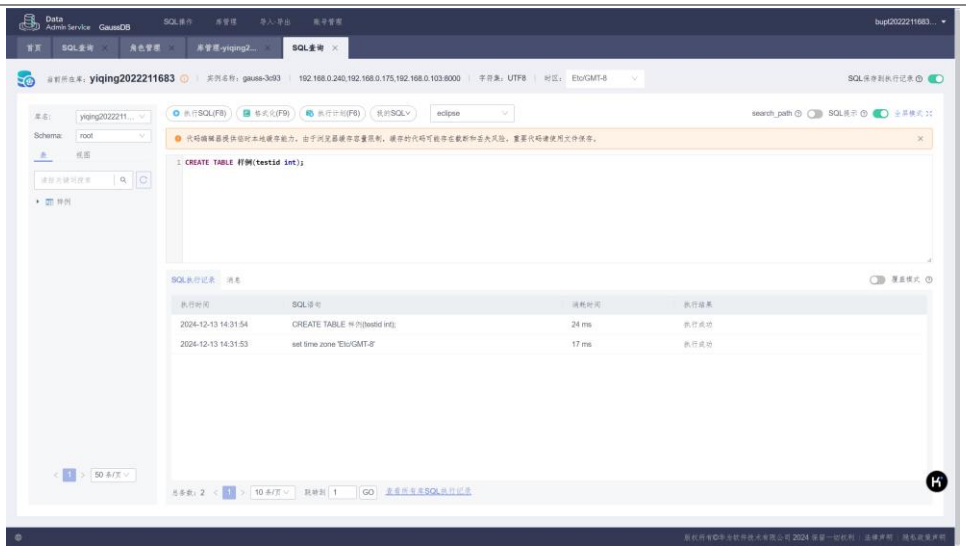


b) 单击库管理，创建 root 用户的同名 Schema:



c) 创建成功后，单击 SQL 窗口，用以下语句在窗口中创建一张样例表，具体如下：

```
CREATE TABLE 样例(testid int);
```



d) 单击账户管理->角色管理->单击角色名 stu2022211683->权限->添加，类型选择数据库，数据库选择 yiqing2022211683，然后单击编辑；

e) 勾选授予 CONNECT 权限；

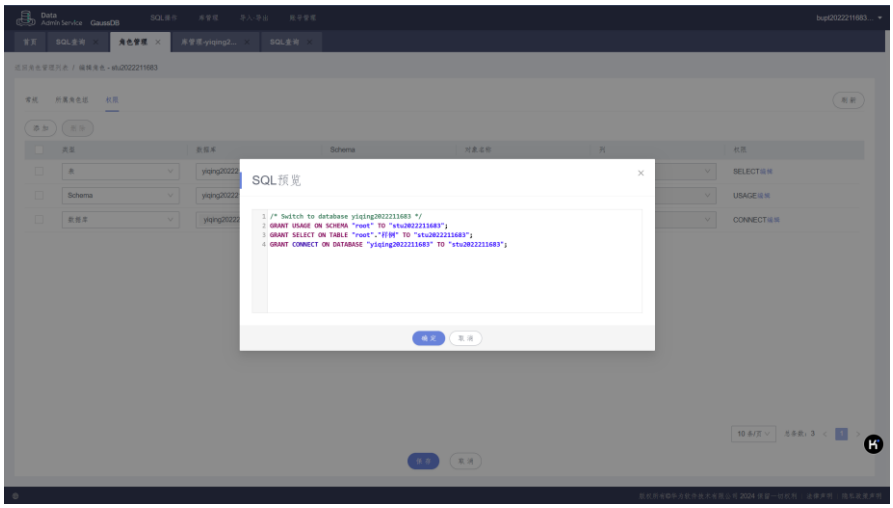
f) 再次单击添加，类型选择 Schema，数据库选择 yiqing2022211683，Schema 选择 root，单击编辑；

g) 勾选授予 USAGE 权限，单击确定；

h) 再次单击添加，类型选择表，数据库选择 yiqing2022211683，Schema 选择 root，对象名称选择样例，单击编辑；

i) 勾选授予 SELECT 权限；

添加完成后选择保存，单击确定后，权限添加完毕。



j) 验证用户权限：单击右上角账户名，选择切换连接

k) 选择 yiqing2022211683 数据库的 SQL 查询，输入查询语句:

```
select * from 样例;
```

l) 查询结果如下：

六、实验小结

本次实验让我掌握了用户创建、管理的基本操作，让我熟悉了用户和权限管理的相关知识。

本次实验也让我了解了不同的权限的区别，让我对管理用户权限产生了不小的兴趣，且去查阅了相关资料，了解了不同权限的作用。

对于本实验的建议：

可以适当增加别的权限操作；

希望可以维护网站，相关操作（如切换链接）响应速度很慢。

实验五 创建和管理索引和视图

一、实验目的

1. 通过实验让学生熟悉并了解 GaussDB(for openGauss)数据库的基本机制与操作。
2. 通过索引管理、视图管理等管理的操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)。

二、实验环境

- 华为云: GaussDB 2.7.2
- 设备名称：数据库
- 设备型号：GaussDB(for openGauss) 8 核 | 64 GB
- 软件版本：GaussDB(for openGauss) 2020 主备版

三、实验内容

1. 本实验通过索引管理、视图管理等管理的操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)；
2. 本实验通过视图管理等管理的操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss) 。

四、实验步骤

1. 创建北京市病例信息的视图，包括行程号，病例号，性别，日期信息（选用病例行程信息表日期）和行程信息。
2. 通过上述视图查询临床分型为普通型的病例号、行程号、性别和日期信息，按照病例号进行升序显示（截前五条记录）。

五、实验结果及分析

1 创建和管理索引

（1） 索引可以提高数据的访问速度，但同时也增加了插入、更新和删除表的处理时间。所以是否要为 表增加索引，索引建立在哪些字段上，是创建索引前必须要考虑的问题。需要分析应用程序的业务处理、数据使用、经常被用作查询条件或者被要求排序的字段来确定是否建立索引。openGauss 支持 4 种创建索引的方式：唯一索引、多字段索引、部分索引、表达式索引。

（2） 创建索引：

在“美国各州县确诊与死亡数统计表”输入以下语句，创建分区表索引索引名，不指定索引分区的名称。

```
CREATE INDEX 日期 index ON 美国各州县确诊与死亡统计表(日期);
```

截图如下：

（3） 管理索引

a) 查询索引：创建索引后刷新页面，左下角会显示表视图，单击 indexes 显示当前表的所有索引。

b) 删除索引：输入以下语句，删除索引：

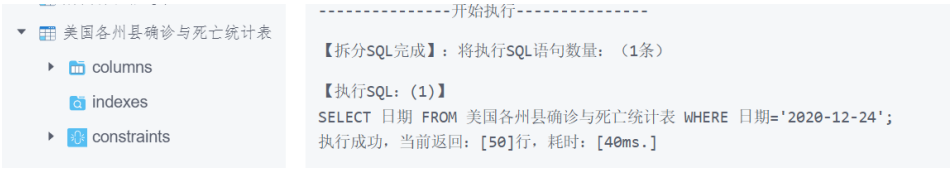
```
DROP INDEX 日期 index
```

2 创建索引练习

（1） 如果对于“美国各州县确诊与死亡统计表”，需要经常进行以下查询。

```
SELECT 日期 FROM 美国各州县确诊与死亡统计表 WHERE 日期='2020-12-24';
```

创建索引之前的效率如下：



使用以下命令创建索引。

```
CREATE INDEX 日期 index ON 美国各州县确诊与死亡统计表(日期);
```

执行效率得到提升：

（2）创建多字段索引：尝试比较未建索引后与创建索引后，查询效率的不同。

若需要经常查询“美国各州县确诊与死亡统计表”中日期是‘2020-12-24’，且’累计确诊’大于 1000 的记录，使用以下命令进行查询。

```
SELECT * FROM 美国各州县确诊与死亡统计表 WHERE 日期= '2020-12-24' AND 累计确诊>1000;
```

无索引时的效率如下：

使用以下命令在字段’日期’和‘累计确诊’上定义一个多字段索引。

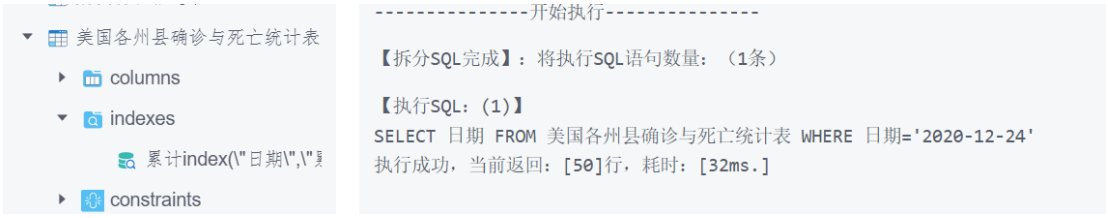
```
CREATE INDEX 累计 index ON 美国各州县确诊与死亡统计表(日期, 累计确诊);
```

执行效率提升：

（3）创建部分索引：尝试比较未建索引后与创建索引后，查询效率的不同。

如果只需要查询日期=‘2020-12-24’的记录，可以创建部分索引来提升查询效率。

创建索引前的效率为：



使用如下命令创建部分索引：

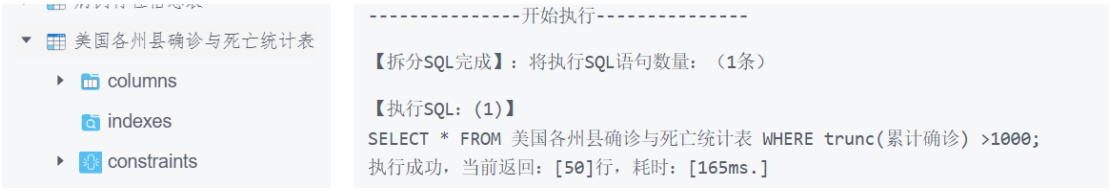
```
CREATE INDEX 日期index ON 美国各州县确诊与死亡统计表(日期) WHERE 日期 = '2020-12-24';
```

执行效率提升：

（4）创建表达式索引：尝试比较未建索引后与创建索引后，查询效率的不同。

若经常需要查询’累计确诊’>1000 的信息，执行如下命令进行查询。

```
SELECT * FROM 美国各州县确诊与死亡统计表 WHERE trunc(累计确诊) >1000;
```



可以为上面的查询创建表达式索引：

```
CREATE INDEX 累计确诊_index ON 美国各州县确诊与死亡统计表(trunc(累计确诊));
```

效率得到提升：

3 创建和管理视图

（1） 基本概念

a) 当用户对数据库中的一张或者多张表的某些字段的组合感兴趣，而又不想每次键入这些查询时，用户就可以定义一个视图，以便解决这个问题。

b) 视图与基本表不同，不是物理上实际存在的，是一个虚表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。视图每次被引用的时候都会运行一次。

（2） 创建视图

a) 执行如下命令创建普通视图 bj_yq：

```
CREATE VIEW bj_yq AS
SELECT 行程号, x.病例号, 性别, x.日期信息, 行程信息
FROM 病例行程信息表 AS x LEFT JOIN 病例基本信息表 AS y
ON x.病例号 = y.病例号 WHERE y.省 = '北京市'
```

b) 截图如下：

（3） 管理视图

a) 查询普通视图

执行如下命令查询 bj_yq 视图。

```
SELECT * FROM bj_yq;
```

b) 查看普通视图的具体信息

切换到 库管理 -> 对象列表，单击 视图，查看视图列表，选中 myview 视图的

操作，单击查看视图详情：

查看视图详情

```
CREATE OR REPLACE VIEW "buptceshi"."bj_yq" as
    SELECT x."行程号", x."病例号", y."性别", x."日期信息", x."行程信息" FROM (buptceshi.'
```

4 实验步骤

1) 创建北京市病例信息的视图，包括行程号，病例号，性别，日期信息（选用病例行程信息表日期）和行程信息。

输入以下 SQL 语句：

```
CREATE VIEW bj_yq AS
SELECT 行程号, x.病例号, 性别, x.日期信息, 行程信息
FROM 病例行程信息表 AS x LEFT JOIN 病例基本信息表 AS y
ON x.病例号 = y.病例号
WHERE y.省 = '北京市';
```

2) 通过上述视图查询临床分型为普通型的病例号、行程号、性别和日期信息，按照病例号进行升序显示（截前五条记录）。

输入以下 SQL 语句：

```
SELECT 病例号, 行程号, 性别, 日期信息
FROM bj_yq
ORDER BY 病例号 ASC
LIMIT 5;
```

结果如下：

六、实验总结

本次实验使我初步掌握了 SQL 相关的索引和视图语法，并且了解了几种不同类型的索引，也让我对数据库中的实际应用有了更多了解。

同时，我也更加理解了视图的作用，它使得数据库在面向用户时更加方便便捷。

实验六 创建和管理存储过程

一、实验目的

1. 通过实验让学生熟悉并了解 GaussDB(for openGauss)数据库的基本机制与操作。

2. 通过创建和管理存储过程操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)。

二、实验环境

- 华为云: GaussDB 2.7.2
- 设备名称：数据库
- 设备型号：GaussDB(for openGauss) 8 核 | 64 GB
- 软件版本：GaussDB(for openGauss) 2020 主备版

三、实验内容

本实验通过对存储过程管理等操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss) 创建和调用及管理存储过程。

四、实验步骤与要求

1. 创建存储过程：

1) 创建存储过程：在全国各省累计数据统计表中增加一条记录。执行存储过程：增加 2021 年 10 月 8 日吉林省累计确诊 578 例，累计治愈 571 例，累计死亡 3 例。

2) 创建存储过程：查询美国指定州指定日期的新冠肺炎累计确诊总数与累计死亡总数。通过该存储过程统计 California 州截至 2021 年 1 月 1 日的新冠疫情数据情况。

3) 创建存储过程：查询中美某天累计确诊病例数。

4) 创建存储过程：向全国各省累计数据统计表增加记录。

5) 创建存储过程：向美国各州县确诊与死亡数统计表中插入记录时，检查该记录的州县在参考信息表中是否存在。如果不存在，则不允许插入。

6) 创建存储过程：在病例基本信息表中删除某记录时，该病例 ID 对应的行程信息记录也进行删除操作。

7) 创建存储过程：查询某城市的风险地区等级数量。调用语句的输入是【石家庄市，中/高风险地区】，输出对应的中/高风险地区数量

2. 管理存储过程：

3. 管理存储过程，切换到库管理 -> 对象列表，选择存储过程，选择 insertRecord 存储过程中的操作，单击查看存储过程详情。

4. 切换到 SQL 查询界面，删除存储过程。命令：drop procedure insertRecord;

五、实验结果及分析

1 创建存储过程

1) 创建存储过程：在全国各省累计数据统计表中增加一条记录。执行存储过程：增加 2021 年 10 月 8 日吉林省累计确诊 578 例，累计治愈 571 例，累计死亡 3 例。

输入 sql 语句如下：

```
CREATE OR REPLACE PROCEDURE add_province_record(
    日期 DATE,
    省 VARCHAR(255),
    累计确诊 INT,
    累计治愈 INT,
    累计死亡 INT
)
AS
BEGIN
    INSERT INTO 全国各省累计数据统计表 (日期, 省, 累计确诊, 累计治愈, 累计死亡)
    VALUES (日期, 省, 累计确诊, 累计治愈, 累计死亡);
END;

CALL add_province_record('2021-10-08', '吉林省', 578, 571, 3);
```

2) 创建存储过程：查询美国指定州指定日期的新冠肺炎累计确诊总数与累计死亡总数。通过该存储过程统计 California 州截至 2021 年 1 月 1 日的新冠疫情数据情况。

执行 sql 语句如下：

```
CREATE OR REPLACE PROCEDURE buptceshi.queryUSstatistics(
    state VARCHAR(255),
    day DATE,
    OUT totalall INT,
    OUT dead INT
)
AS DECLARE
BEGIN
    SELECT SUM(累计确诊), SUM(累计死亡)
    FROM 美国各州县确诊与死亡统计表
    WHERE 州 = state AND 日期 = day
    INTO totalall, dead;
END;

CALL queryUSstatistics('California', '2021-01-01', @totalall, @dead);
```

3) 创建存储过程：查询中美某天累计确诊病例数。

执行 sql 语句如下：

```
CREATE OR REPLACE PROCEDURE buptceshi.queryCNUSstatistics(
    dat DATE,
    OUT cn INT,
    OUT us INT
)
AS DECLARE
BEGIN
    SELECT SUM(累计确诊)
    FROM 全国各省累计数据统计表
    WHERE 日期 = dat
    INTO cn;
    SELECT SUM(累计确诊)
    FROM 美国各州县确诊与死亡统计表
    WHERE 日期 = dat
    INTO us;
END;

CALL queryCNUSstatistics('2021-01-01', @cn, @us);
```

4) 创建存储过程：向全国各省累计数据统计表增加记录。

```
CREATE OR REPLACE PROCEDURE buptceshi.insertProvince (
    省 VARCHAR(255),
    日期 DATE,
    累计确诊 INT,
    累计治愈 INT,
    累计死亡 INT
)
AS DECLARE
BEGIN
    INSERT INTO 全国各省累计数据统计表（省，日期，累计确诊，累计治愈，累计死亡）
```

```
    VALUES (省, 日期, 累计确诊, 累计治愈, 累计死亡);
END;

CALL insertProvince('江苏省', '2021-11-01', 10, 10, 0);
```

5) 创建存储过程：向美国各州县确诊与死亡数统计表中插入记录时，检查该记录的州县在参考信息表中是否存在。如果不存在，则不允许插入。

执行 sql 语句如下：

```
CREATE OR REPLACE PROCEDURE insertUSRecord(
    state_name VARCHAR(255),
    county_name VARCHAR(255),
    day DATE,
    totalall INT,
    death INT
)
AS DECLARE
BEGIN
    IF EXISTS (
        SELECT * FROM 参考信息表
        WHERE 省州 = state_name AND 市县 = county_name
    ) THEN
        INSERT INTO 美国各州县确诊与死亡统计表 (州, 县, 日期, 累计确诊, 累计死亡)
        VALUES (state_name, county_name, day, totalall, death);
    ELSE
        RAISE EXCEPTION 'not exist in the reference table';
    END IF;
```

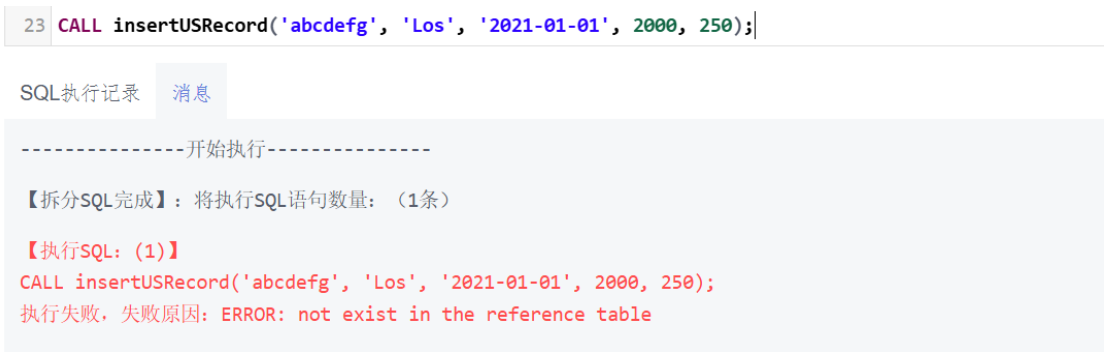
```
END;
```

尝试插入在参考信息表中存在的记录：

```
CALL insertUSRecord('California', 'Los Angeles', '2021-01-01', 2000, 250);
```

尝试插入在参考信息表中不存在的记录：

```
CALL insertUSRecord('abcdefg', 'Los', '2021-01-01', 2000, 250);
```



6) 创建存储过程：在病例基本信息表中删除某记录时，该病例 ID 对应的行程信息记录也进行删除操作。

执行 sql 语句如下：

```
CREATE OR REPLACE PROCEDURE deleteRecord(
    caseID INT
)
AS DECLARE
BEGIN
    DELETE FROM 病例行程信息表 WHERE 病例号 = caseID;
    DELETE FROM 病例基本信息表 WHERE 病例号 = caseID;
END;
```

```
CALL deleteRecord(1);
```

```
CREATE OR REPLACE PROCEDURE deleteRecord(
caseID INT
)
AS DECLARE
BEGIN
DELETE FROM 病例行程信息表 WHERE 病例号 = caseID;
DELETE FROM 病例基本信息表 WHERE 病例号 = caseID;
END;

CALL deleteRecord(1);
```

SQL执行记录 消息 结果集1

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（2条）
【执行SQL：(1)】
CREATE OR REPLACE PROCEDURE deleteRecord(
caseID INT
)
AS DECLARE
BEGIN
DELETE FROM 病例行程信息表 WHERE 病例号 = caseID;
DELETE FROM 病例基本信息表 WHERE 病例号 = caseID;
END;
执行成功，耗时：[32ms.]
【执行SQL：(2)】
CALL deleteRecord(1);
执行成功，当前返回：[1]行，耗时：[16ms.]
```

7) 创建存储过程：查询某城市的风险地区等级数量。调用语句的输入是【石家庄市，中/高风险地区】，输出对应的中/高风险地区数量

执行 sql 语句如下：

```
CREATE OR REPLACE PROCEDURE queryRiskCount(
    cname VARCHAR(255),
    risk_level VARCHAR(255),
    OUT result INT
)
AS DECLARE
BEGIN
    SELECT COUNT(*)
    FROM 全国城市风险等级表
    WHERE 市 = cname AND 风险等级 = risk_level
    INTO result ;
END;

CALL queryRiskCount('石家庄市', '中风险地区', @result);
CALL queryRiskCount('石家庄市', '高风险地区', @result);
```

以下是CALL queryRiskCount('石家庄市', '中风险地区', @result);的执行结果集 ⓘ 该表不可编辑。

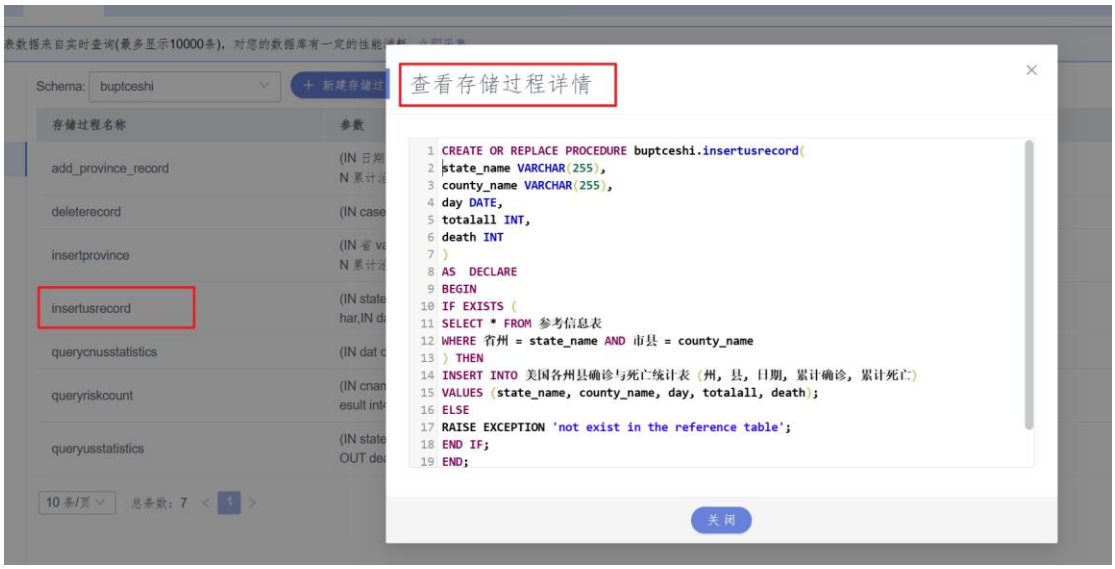
	result
1	40

以下是CALL queryRiskCount('石家庄市', '高风险地区', @result);的执行结果集　　ⓘ 该表不可编辑。

	result
1	2

2 管理存储过程

管理存储过程，切换到库管理 -> 对象列表，选择存储过程，选择 insertusrecord 存储过程中的操作，单击查看存储过程详情。



切换到 SQL 查询界面，删除存储过程。命令：

```
drop procedure insertusrecord;
```

```
1 drop procedure insertusrecord;
```

SQL执行记录　消息

```
----------开始执行----------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
drop procedure insertusrecord;
执行成功，耗时：[7ms.]
```

六、实验总结

本次实验让我掌握了存储过程的相关知识和操作，也让我学习了存储过程、触发器和函数的使用以及语法。

除此之外，在处理一些额外需求时，我发现使用 sql 语句也可以一定程度上代替触发器等，使得实现逻辑更清晰。

课件的相关知识点也拓宽了我的知识面。

一些实验建议：

1. 希望指导书里的内容可以再详细一点，刚开始处理存储过程时仍有点不知半解，导致一些小错误；

2. 希望实验内容里增加触发器的内容，虽然可以使用存储过程代替一些触发器，但更希望可以针对触发器的使用，适当增添实验内容。

实验七 数据库接口实验

一、实验目的

1. 华为的 GaussDB(for openGauss)支持基于 C、Java 等应用程序的开发。了解它相关的系统结构和相关概念，有助于更好地开发和使用 GaussDB(for openGauss)数据库。

2. 通过实验了解通用数据库应用编程接口 ODBC/JDBC 的基本原理和实现机制，熟悉连接 ODBC/JDBC 接口的语法和使用方法。

3. 熟练 GaussDB(for openGauss)的各种连接方式与常用工具的使用。

4. 利用 C 语言(或其它支持 ODBC/JDBC 接口的高级程序设计语言)编程实现简单的数据库应用程序，掌握基于 ODBC 的数据库访问基本原理和方法。

二、实验环境

- 数据库：MySQL8.0
- ODBC 9.1.0
- C++ 17
- Visual Studio Code 1.96.0

三、实验内容

1. 本实验内容通过使用 ODBC/JDBC 等驱动开发应用程序。

2. 连接语句访问数据库接口，实现对数据库中的数据进行操作（包括增、删、改、查等）；

3. 要求能够通过编写程序访问到华为数据库，该实验重点在于 ODBC/JDBC 数据源配置和高级语言(C/C++/JAVA/PYTHON)的使用。

四、实验步骤与要求

1. 在 Windows 控制面板中通过管理工具下的 ODBC 数据源工具在客户端新建连接到华为分布式数据库服务器的 ODBC 数据源，测试通过后保存，注意名字应与应用程序中引用的数据源一致。

1）编译程序并调试通过；

2）实验过程要求：

（1） 以 PGSQL 语言相关内容为基础，课后查阅、自学 ODBC/JDBC 接口有关内容，包括 ODBC 的体系结构、工作原理、数据访问过程、主要 API 接口的语法和使用方法等。

（2）以实验二建立的数据库为基础，编写 C 语言(或其它支持 ODBC/JDBC 接口的高级程序设计语言) 数据库应用程序，按照如下步骤访问数据库：

a) Step1. ODBC 初始化，为 ODBC 分配环境句柄；

b) Step2. 建立应用程序与 ODBC 数据源的连接；

c) Step3. 实现数据库应用程序对数据库中表的数据查询、修改、删除、插入等操作。

【程序设计逻辑举例：可以先打印出所有记录，接下来删除某一行；再进行打印，继续修改；最后打印一遍，然后插入。】

d) Step4. 结束数据库应用程序。

注意：

e) 由于不是程序设计练习，因此针对一张表进行操作，即可完成基本要求。

f) 若程序结构和功能完整，界面友好，可适当增加分数。

（3） 实验相关语句要求：

所编写的数据库访问应用程序应使用到以下主要的 ODBC API 函数：

(a) SQLALLocEnv：初始化 ODBC 环境，返回环境句柄；

(b) SQLALLocConnect：为连接句柄分配内存并返回连接句柄；

(c) SQLConnect：连接一个 SQL 数据资源；

(d) SQLDriverConnect：连接一个 SQL 数据资源，允许驱动器向用户询问信息；

(e) SQLALLocStmt：为语句句柄分配内存, 并返回语句句柄；

(f) SQLExecDirect：把 SQL 语句送到数据库服务器，请求执行由 SQL 语句定义的数据库访问；

(g) SQLFetchAdvances：将游标移动到查询结果集的下一行(或第一行)；

(h) SQLGetData：按照游标指向的位置，从查询结果集的特定的一列取回数据；

(i) SQLFreeStmt：释放与语句句柄相关的资源；

(j) SQLDisconnect：切断连接；

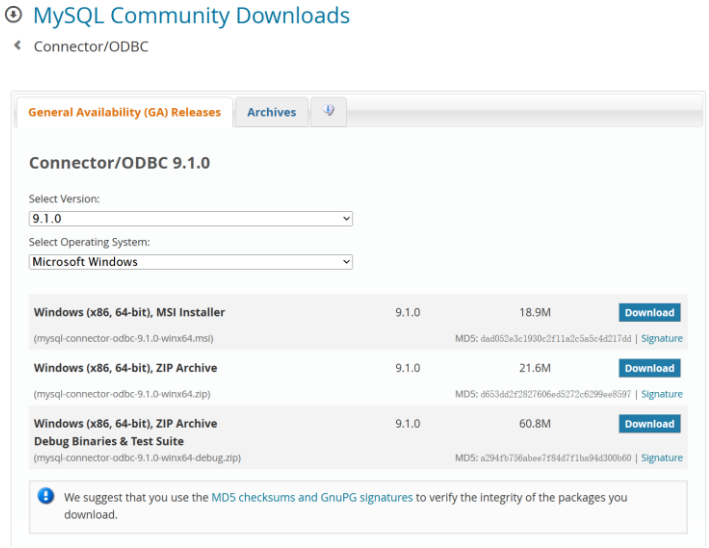
(k) SQLFreeConnect：释放与连接句柄相关的资源；

(l) SQLFreeEnv：释放与环境句柄相关的资源。

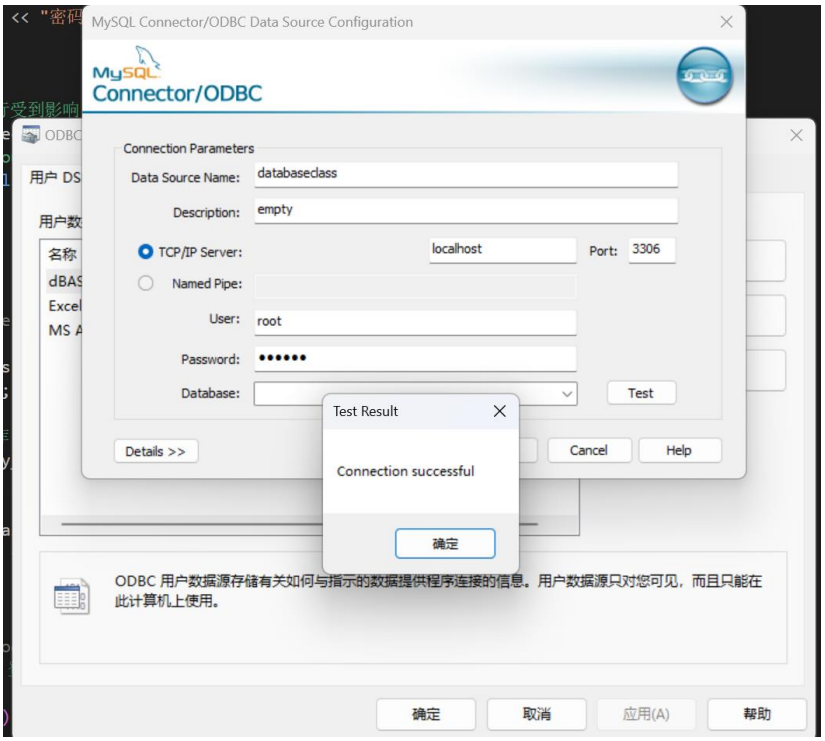
五、实验结果及分析

1 安装 ODBC 数据源工具

通过网址 MySQL :: Download Connector/ODBC 下载安装 mysql-connector-odbc-9.1.0。



进入 C:\Windows\System32\odbcad32.exe，选择用户 DSN-＞ 添加->MySQL ODBC 8.0 Unicode Driver，输入用户名和密码，点击测试，连接成功后保存并退出。



2 程序设计

```
#include <windows.h>

#include <sqlext.h>

#include <iostream>
#include <string>

// 辅助函数：检查 SQL 操作的返回值并报告结果
void checkResult(SQLRETURN retcode, const std::string& action) {
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
        std::cout << action << " succeeded.\n";
    } else {
        std::cerr << action << " failed.\n";
        exit(EXIT_FAILURE);
    }
}

int main() {
    SQLHENV hEnv;       // ODBC 环境句柄
    SQLHDBC hDbc;       // 数据库连接句柄
    SQLHSTMT hStmt;     // SQL 语句句柄
    SQLRETURN retcode;  // SQL 操作返回值

    // Step 1: 初始化 ODBC 环境
    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hEnv);  // 分配环境句柄
    checkResult(retcode, "SQLAllocHandle (Environment)");

    // 设置 ODBC 版本为 3.0
    retcode = SQLSetEnvAttr(hEnv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 0);
    checkResult(retcode, "SQLSetEnvAttr");

    // 分配数据库连接句柄
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, hEnv, &hDbc);
    checkResult(retcode, "SQLAllocHandle (Connection)");

    // Step 2: 建立数据库连接
    SQLCHAR connStr[] = "DSN=databaseclass;UID=root;PWD=abc123;DATABASE=test3;";
    retcode = SQLDriverConnect(hDbc, NULL, connStr, SQL_NTS, NULL, 0, NULL, SQL_DRIVER_COMPLETE);
```

```
39.     checkResult(retcode, "SQLDriverConnect");
40.
41.     std::cout << "Connection established successfully!\n";
42.
43.     // 分配 SQL 语句句柄
44.     retcode = SQLAllocHandle(SQL_HANDLE_STMT, hDbc, &hStmt);
45.     checkResult(retcode, "SQLAllocHandle (Statement)");
46.
47.     // Step 3: 查询所有记录
48.     std::cout << "\nQuerying all records in the diary table:\n";
49.     const char* query = "SELECT title, authorID, destination, content, popularity,
rating, score FROM diary;";
50.     retcode = SQLExecDirect(hStmt, (SQLCHAR*)query, SQL_NTS);  // 执行查询
51.     checkResult(retcode, "SQLExecDirect (Query All)");
52.
53.     // 用于存储查询结果的变量
54.     SQLCHAR title[256], authorID[256], destination[256], content[256];
55.     SQLINTEGER popularity, rating;
56.     SQLDOUBLE score;
57.
58.     // 提取并显示查询结果
59.     while (SQLFetch(hStmt) == SQL_SUCCESS) {
60.         SQLGetData(hStmt, 1, SQL_C_CHAR, title, sizeof(title), NULL);
61.         SQLGetData(hStmt, 2, SQL_C_CHAR, authorID, sizeof(authorID), NULL);
62.         SQLGetData(hStmt, 3, SQL_C_CHAR, destination, sizeof(destination), NULL);
63.         SQLGetData(hStmt, 4, SQL_C_CHAR, content, sizeof(content), NULL);
64.         SQLGetData(hStmt, 5, SQL_C_SLONG, &popularity, 0, NULL);
65.         SQLGetData(hStmt, 6, SQL_C_SLONG, &rating, 0, NULL);
66.         SQLGetData(hStmt, 7, SQL_C_DOUBLE, &score, 0, NULL);
67.         std::cout << "Title: " << title << ", AuthorID: " << authorID << ",
Destination: " << destination
68.                   << ", Content: " << content << ", Popularity: " << popularity
69.                   << ", Rating: " << rating << ", Score: " << score << "\n";
70.     }
71.     SQLFreeStmt(hStmt, SQL_CLOSE);
72.
73.     // Step 4: 删除一条记录
74.     std::cout << "\nDeleting a record from the diary table:\n";
75.     const char* deleteStmt = "DELETE FROM diary WHERE authorID = 'Kimi';";
76.     retcode = SQLExecDirect(hStmt, (SQLCHAR*)deleteStmt, SQL_NTS);
77.     checkResult(retcode, "SQLExecDirect (Delete)");
78.     std::cout << "Record deleted successfully!\n";
79.     SQLLEN rowCount = 0;
80.     SQLRowCount(hStmt, &rowCount);
```

```
81.     printf("%d rows affected.\n", rowCount);
82.
83.     // Step 5: 修改记录
84.     std::cout << "\nUpdating a record in the diary table:\n";
85.     const char* updateStmt = "UPDATE diary SET content = 'Updated Content',
popularity = 100 WHERE authorID = 'Lucia';";
86.     retcode = SQLExecDirect(hStmt, (SQLCHAR*)updateStmt, SQL_NTS);
87.     checkResult(retcode, "SQLExecDirect (Update)");
88.     std::cout << "Record updated successfully!\n";
89.     rowCount = 0;
90.     SQLRowCount(hStmt, &rowCount);
91.     printf("%d rows affected.\n", rowCount);
92.
93.     // Step 6: 插入一条记录
94.     std::cout << "\nInserting a record into the diary table:\n";
95.     const char* insertStmt = "INSERT INTO diary (title, authorID, destination,
content, popularity, rating, score) VALUES ('New Title', 'A123', 'New York', 'New
Content', 5000, 500, 4.5);";
96.     retcode = SQLExecDirect(hStmt, (SQLCHAR*)insertStmt, SQL_NTS);
97.     checkResult(retcode, "SQLExecDirect (Insert)");
98.     std::cout << "Record inserted successfully!\n";
99.     rowCount = 0;
100.    SQLRowCount(hStmt, &rowCount);
101.    printf("%d rows affected.\n", rowCount);
102.
103.    // Step 7: 再次查询并显示所有记录
104.    std::cout << "\nQuerying and displaying all records from the diary table:\n";
105.    const char* queryagain = "SELECT title, authorID, destination, content,
popularity, rating, score FROM diary;";
106.    retcode = SQLExecDirect(hStmt, (SQLCHAR*)queryagain, SQL_NTS);  // 执行查询
107.    checkResult(retcode, "SQLExecDirect (Query All)");
108.
109.    // 提取并显示查询结果
110.    while (SQLFetch(hStmt) == SQL_SUCCESS) {
111.        SQLGetData(hStmt, 1, SQL_C_CHAR, title, sizeof(title), NULL);
112.        SQLGetData(hStmt, 2, SQL_C_CHAR, authorID, sizeof(authorID), NULL);
113.        SQLGetData(hStmt, 3, SQL_C_CHAR, destination, sizeof(destination), NULL);
114.        SQLGetData(hStmt, 4, SQL_C_CHAR, content, sizeof(content), NULL);
115.        SQLGetData(hStmt, 5, SQL_C_SLONG, &popularity, 0, NULL);
116.        SQLGetData(hStmt, 6, SQL_C_SLONG, &rating, 0, NULL);
117.        SQLGetData(hStmt, 7, SQL_C_DOUBLE, &score, 0, NULL);
118.        std::cout << "Title: " << title << ", AuthorID: " << authorID << ",
Destination: " << destination
119.                  << ", Content: " << content << ", Popularity: " << popularity
```

```
                  << ", Rating: " << rating << ", Score: " << score << "\n";
    }
    SQLFreeStmt(hStmt, SQL_CLOSE);

    // Clean up
    std::cout << "\nCleaning up resources...\n";
    SQLFreeStmt(hStmt, SQL_CLOSE);        // 释放语句句柄
    SQLDisconnect(hDbc);                  // 断开数据库连接
    SQLFreeHandle(SQL_HANDLE_DBC, hDbc);  // 释放数据库连接句柄
    SQLFreeHandle(SQL_HANDLE_ENV, hEnv);  // 释放环境句柄
    std::cout << "All resources freed.\n";

    return 0;
}
```

3 编译运行

```
> g++ -o ODBC ODBC.cpp -lodbc32 -lodbccp32
> ./ODBC
```

4 运行结果

```
SQLAllocHandle (Environment) succeeded.
SQLSetEnvAttr succeeded.
SQLAllocHandle (Connection) succeeded.
SQLDriverConnect succeeded.
Connection established successfully!
SQLAllocHandle (Statement) succeeded.

Querying all records in the diary table:
SQLExecDirect (Query All) succeeded.
Title: 美好的一天, AuthorID: William, Destination: 北京邮电大学, Content: 今天在北京邮电大学打了羽毛球，体验很好, Popularity: 7440, Rating: 39204, Score: 0
Title: 秋日漫步, AuthorID: Redcap, Destination: 中山公园, Content: 秋天的中山公园格外美丽, Popularity: 5670, Rating: 30000, Score: 1.1
Title: 校园生活, AuthorID: Redcap, Destination: 复旦大学, Content: 复旦的图书馆真是学习的好地方, Popularity: 2340, Rating: 25000, Score: 0
Title: 雨后清新, AuthorID: Redcap, Destination: 武汉大学, Content: 雨后的武大樱花道清新宜人, Popularity: 4560, Rating: 25000, Score: 0
Title: 历史之旅, AuthorID: William, Destination: 故宫博物院, Content: 故宫的历史沉淀让人着迷, Popularity: 1000, Rating: 40000, Score: 0
```

Title: 自然探索, AuthorID: Lucia, Destination: 黄山, Content: Updated Content, Popularity: 5055, Rating: 23000, Score: 0.4
Title: 文化体验, AuthorID: Lucia, Destination: 颐和园, Content: Updated Content, Popularity: 1008, Rating: 45000, Score: 0
Title: 春日游园, AuthorID: Lucia, Destination: 南京大学, Content: Updated Content, Popularity: 100111, Rating: 12000, Score: 2.2
Title: 夏日海边, AuthorID: William, Destination: 青岛海滨, Content: 夏日的青岛海边凉爽宜人, Popularity: 5000, Rating: 11000, Score: 0
Title: 古迹探访, AuthorID: William, Destination: 长城, Content: 长城的雄伟令人震撼, Popularity: 6000, Rating: 21000, Score: 1
Title: 秋色宜人, AuthorID: 测试, Destination: 杭州西湖, Content: 秋天的西湖美景如画, Popularity: 7000, Rating: 33000, Score: 0
Title: 冬日暖阳, AuthorID: 测试, Destination: 哈尔滨工业大学, Content: 哈工大的冬日阳光温暖人心, Popularity: 8000, Rating: 44000, Score: 0.6
Title: 山川壮丽, AuthorID: 测试, Destination: 九寨沟, Content: 九寨沟的山水让人心旷神怡, Popularity: 9060, Rating: 41000, Score: 0
Title: 城市夜景, AuthorID: 测试, Destination: 上海外滩, Content: 外滩的夜景璀璨夺目, Popularity: 1500, Rating: 24000, Score: 5
Title: 乡村风光, AuthorID: William, Destination: 婺源, Content: 婺源的乡村风光宁静祥和, Popularity: 2500, Rating: 35000, Score: 0
Title: 节日庆典, AuthorID: Seven, Destination: 广州塔, Content: 广州塔的跨年烟火绚丽多彩, Popularity: 3510, Rating: 45000, Score: 4
Title: 学术氛围, AuthorID: Seven, Destination: 浙江大学, Content: 浙大的学术氛围浓厚, Popularity: 4500, Rating: 13000, Score: 0
Title: 自然奇观, AuthorID: Seven, Destination: 张家界, Content: 张家界的自然景观令人赞叹, Popularity: 5500, Rating: 23000, Score: 0
Title: 历史遗迹, AuthorID: William, Destination: 秦始皇兵马俑, Content: 参观兵马俑，感受千年文化, Popularity: 6500, Rating: 34000, Score: 2
Title: 海滨日落, AuthorID: William, Destination: 三亚海滨, Content: 观看三亚的日落，宁静美好, Popularity: 7610, Rating: 47500, Score: 0
Title: 忙碌的一天, AuthorID: William, Destination: 北京邮电大学, Content: 好忙啊哦哦哦, Popularity: 5000, Rating: 40000, Score: 3.4
Title: 大功告成, AuthorID: William, Destination: 北京邮电大学, Content: 今天下雨了，好凉快, Popularity: 5030, Rating: 46250, Score: 0
Title: 你好, AuthorID: William, Destination: 教三 537, Content: 不太好, Popularity: 5010, Rating: 42500, Score: 2.4
Title: 疯狂星期三, AuthorID: 路徐, Destination: 教三 537, Content: 再见, Popularity: 5000, Rating: 40000, Score: 0
Title: 你好好, AuthorID: 路徐, Destination: 教三 539, Content: 好好好好！, Popularity: 5000, Rating: 40000, Score: 0
Title: 你好, AuthorID: 测试用户 612, Destination: 北京邮电大学, Content: 今天写报告, Popularity: 5000, Rating: 40000, Score: 0
Title: 数据库验收, AuthorID: Kimi, Destination: 教二, Content: 数据库验收, Popularity: 5949, Rating: 55135, Score: 1.5

```
Title: 临时补充, AuthorID: Kimi, Destination: 教三, Content: 验收ing, Popularity: 1111,
Rating: 2222, Score: 2

Deleting a record from the diary table:
SQLExecDirect (Delete) succeeded.
Record deleted successfully!
2 rows affected.

Updating a record in the diary table:
SQLExecDirect (Update) succeeded.
Record updated successfully!
3 rows affected.

Inserting a record into the diary table:
SQLExecDirect (Insert) succeeded.
Record inserted successfully!
1 rows affected.

Querying and displaying all records from the diary table:
SQLExecDirect (Query All) succeeded.
Title: 美好的一天, AuthorID: William, Destination: 北京邮电大学, Content: 今天在北京邮电大学
打了羽毛球，体验很好, Popularity: 7440, Rating: 39204, Score: 0
Title: 秋日漫步, AuthorID: Redcap, Destination: 中山公园, Content: 秋天的中山公园格外美丽,
Popularity: 5670, Rating: 30000, Score: 1.1
Title: 校园生活, AuthorID: Redcap, Destination: 复旦大学, Content: 复旦的图书馆真是学习的好
地方, Popularity: 2340, Rating: 25000, Score: 0
Title: 雨后清新, AuthorID: Redcap, Destination: 武汉大学, Content: 雨后的武大樱花道清新宜人,
Popularity: 4560, Rating: 25000, Score: 0
Title: 历史之旅, AuthorID: William, Destination: 故宫博物院, Content: 故宫的历史沉淀让人着
迷, Popularity: 1000, Rating: 40000, Score: 0
Title: 自然探索, AuthorID: Lucia, Destination: 黄山, Content: Updated Content, Popularity:
100, Rating: 23000, Score: 0.4
Title: 文化体验, AuthorID: Lucia, Destination: 颐和园, Content: Updated Content,
Popularity: 100, Rating: 45000, Score: 0
Title: 春日游园, AuthorID: Lucia, Destination: 南京大学, Content: Updated Content,
Popularity: 100, Rating: 12000, Score: 2.2
Title: 夏日海边, AuthorID: William, Destination: 青岛海滨, Content: 夏日的青岛海边凉爽宜人,
Popularity: 5000, Rating: 11000, Score: 0
Title: 古迹探访, AuthorID: William, Destination: 长城, Content: 长城的雄伟令人震撼,
Popularity: 6000, Rating: 21000, Score: 1
Title: 秋色宜人, AuthorID: 测试, Destination: 杭州西湖, Content: 秋天的西湖美景如画,
Popularity: 7000, Rating: 33000, Score: 0
Title: 冬日暖阳, AuthorID: 测试, Destination: 哈尔滨工业大学, Content: 哈工大的冬日阳光温暖人
心, Popularity: 8000, Rating: 44000, Score: 0.6
```

```
Title: 山川壮丽, AuthorID: 测试, Destination: 九寨沟, Content: 九寨沟的山水让人心旷神怡,
Popularity: 9060, Rating: 41000, Score: 0
Title: 城市夜景, AuthorID: 测试, Destination: 上海外滩, Content: 外滩的夜景璀璨夺目,
Popularity: 1500, Rating: 24000, Score: 5
Title: 乡村风光, AuthorID: William, Destination: 婺源, Content: 婺源的乡村风光宁静祥和,
Popularity: 2500, Rating: 35000, Score: 0
Title: 节日庆典, AuthorID: Seven, Destination: 广州塔, Content: 广州塔的跨年烟火绚丽多彩,
Popularity: 3510, Rating: 45000, Score: 4
Title: 学术氛围, AuthorID: Seven, Destination: 浙江大学, Content: 浙大的学术氛围浓厚,
Popularity: 4500, Rating: 13000, Score: 0
Title: 自然奇观, AuthorID: Seven, Destination: 张家界, Content: 张家界的自然景观令人赞叹,
Popularity: 5500, Rating: 23000, Score: 0
Title: 历史遗迹, AuthorID: William, Destination: 秦始皇兵马俑, Content: 参观兵马俑，感受千年
文化, Popularity: 6500, Rating: 34000, Score: 2
Title: 海滨日落, AuthorID: William, Destination: 三亚海滨, Content: 观看三亚的日落，宁静美
好, Popularity: 7610, Rating: 47500, Score: 0
Title: 忙碌的一天, AuthorID: William, Destination: 北京邮电大学, Content: 好忙啊哦哦哦,
Popularity: 5000, Rating: 40000, Score: 3.4
Title: 大功告成, AuthorID: William, Destination: 北京邮电大学, Content: 今天下雨了，好凉快,
Popularity: 5030, Rating: 46250, Score: 0
Title: 你好, AuthorID: William, Destination: 教三 537, Content: 不太好, Popularity: 5010,
Rating: 42500, Score: 2.4
Title: 疯狂星期三, AuthorID: 路徐, Destination: 教三 537, Content: 再见, Popularity: 5000,
Rating: 40000, Score: 0
Title: 你好好, AuthorID: 路徐, Destination: 教三 539, Content: 好好好好！, Popularity:
5000, Rating: 40000, Score: 0
Title: 你好, AuthorID: 测试用户 612, Destination: 北京邮电大学, Content: 今天写报告,
Popularity: 5000, Rating: 40000, Score: 0
Title: New Title, AuthorID: A123, Destination: New York, Content: New Content,
Popularity: 5000, Rating: 500, Score: 4.5

Cleaning up resources...
All resources freed.
```

六、实验总结

1 实验问题

(1) 代码编写完成后，编译失败

执行命令：

```
> g++ -o ODBC ODBC.cpp -lodbc32 -lodbccp32
```

报错：

```
(base) PS E:\WILLIAMZHANG\DataBase\lab> g++ -o ODBC ODBC.cpp -lodbc32
In file included from D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/sql.h:13,
                 from D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/sqlext.h:9,
                 from ODBC.cpp:1:
D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/sqltypes.h:37:11: error: '__LONG32' does not name a type; did you mean '__LINE__'?
   typedef __LONG32 SQLINTEGER;
           ^~~~~~~~
           __LINE__
D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/sqltypes.h:38:29: error: expected initializer before 'SQLUINTEGER'
   typedef unsigned __LONG32 SQLUINTEGER;
                             ^~~~~~~~~~~
D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/sqltypes.h:41:11: error: 'INT64' does not name a type; did you mean 'INT64_C'?
   typedef INT64 SQLLEN;
           ^~~~~
           INT64_C
D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/sqltypes.h:42:11: error: 'UINT64' does not name a type; did you mean 'WIN64'?
   typedef UINT64 SQLULEN;
           ^~~~~~
           WIN64
D:/x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0/mingw64/x86_64-w64-mingw32/include/sqltypes.h:43:11: error: 'UINT64' does not name a type; did you mean 'WIN64'?
   typedef UINT64 SQLSETPOSIROW;
           ^~~~~~
           WIN64
```

解决：

由于 vscode 的自动格式化，使得头文件<sqlext.h>出现在<windows.h>前面，导致编译时报错，将格式改为如下，避免格式化同时解决报错：

```
#include <windows.h>

#include <sqlext.h>
```

(2) 中文输出乱码导致程序崩溃

在 MySQL 中使用的 UTF-8 编码，并且存储了中文内容，这导致在输出到终端时乱码，程序无法正常运行：

```
(base) PS E:\WILLIAMZHANG\DataBase\lab> g++ -o ODBC ODBC.cpp -lodbc32 -lodbccp32
(base) PS E:\WILLIAMZHANG\DataBase\lab> ./ODBC
SQLAllocHandle (Environment) succeeded.
SQLSetEnvAttr succeeded.
SQLAllocHandle (Connection) succeeded.
SQLDriverConnect succeeded.
Connection established successfully!
SQLAllocHandle (Statement) succeeded.

Querying all records in the diary table:
SQLExecDirect (Query All) succeeded.
Title: 缇庡ソ鐨勫瓨澶?
```

在终端执行如下命令解决问题：

```
> [Console]::OutputEncoding = [System.Text.Encoding]::UTF8
```

2 数据库驱动的概念

数据库驱动是应用程序与数据库之间的中介软件组件，主要负责实现二者的通信。它通过标准化的接口，屏蔽了数据库底层通信的复杂性，使应用程序可以方便地与数据库交互。

驱动的核心功能包括将应用程序发出的操作（如 SQL 语句）转换为数据库服务器能够理解的协议，并将数据库返回的结果解析为应用程序可以使用的格式。

常见的数据库驱动类型有 ODBC 驱动、JDBC 驱动和原生驱动等，其中 ODBC 驱动适用于多种关系型数据库，JDBC 驱动是 Java 专用的连接标准，原生驱动则针对特定数据库进行优化，性能最佳但局限于特定数据库。

通过使用数据库驱动，开发者可以通过统一的接口高效地实现与数据库的连接、查询、更新和数据处理操作，而不必关心底层通信的复杂细节。这不仅提升了开发效率，还增强了应用程序的跨数据库兼容性和灵活性。

说明：

1. 应用程序通过标准接口（如 ODBC、JDBC）与驱动交互。
2. 驱动负责将应用程序的操作转化为数据库协议。
3. 驱动接收数据库返回的数据并解析给应用程序。

3 ODBC 开发应用流程

在使用 ODBC（Open Database Connectivity）开发数据库应用程序时，开发流程通常包括以下步骤：

1. 加载 ODBC 环境

分配环境句柄：调用 SQLAllocHandle 分配一个 ODBC 环境句柄（SQLHENV）。

```
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hEnv);
```

设置环境属性：设置 ODBC 的版本属性（例如 ODBC 3.0）。

```
SQLSetEnvAttr(hEnv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 0);
```

2. 建立数据库连接

分配数据库连接句柄：调用 SQLAllocHandle 分配一个数据库连接句柄。

```
SQLAllocHandle(SQL_HANDLE_DBC, hEnv, &hDbc);
```

建立连接：使用 SQLDriverConnect 或 SQLConnect 连接到数据库。\

```
SQLDriverConnect(hDbc, NULL,
(SQLCHAR*)"DSN=databaseclass;UID=root;PWD=abc123;DATABASE=test3;", SQL_NTS, NULL,
0, NULL, SQL_DRIVER_COMPLETE);
```

3. 发送 SQL 语句

分配语句句柄：调用 SQLAllocHandle 分配 SQL 语句句柄（SQLHSTMT）。

```
SQLAllocHandle(SQL_HANDLE_STMT, hDbc, &hStmt);
```

执行 SQL 语句：调用 SQLExecDirect 执行 SQL 查询或更新语句。

```
SQLExecDirect(hStmt, (SQLCHAR*)"SELECT * FROM myTable;", SQL_NTS);
```

4. 处理查询结果

提取数据：使用 SQLFetch 获取查询结果的每一行，然后调用 SQLGetData 提取具体的字段值。

```
while (SQLFetch(hStmt) == SQL_SUCCESS) {
    SQLGetData(hStmt, 1, SQL_C_CHAR, title, sizeof(title), NULL);
    SQLGetData(hStmt, 2, SQL_C_CHAR, authorID, sizeof(authorID), NULL);
    SQLGetData(hStmt, 3, SQL_C_CHAR, destination, sizeof(destination), NULL);
    SQLGetData(hStmt, 4, SQL_C_CHAR, content, sizeof(content), NULL);
    SQLGetData(hStmt, 5, SQL_C_SLONG, &popularity, 0, NULL);
    SQLGetData(hStmt, 6, SQL_C_SLONG, &rating, 0, NULL);
    SQLGetData(hStmt, 7, SQL_C_DOUBLE, &score, 0, NULL);
```

```
    std::cout << "Title: " << title << ", AuthorID: " << authorID << ", Destination:
" << destination << ", Content: " << content << ", Popularity: " << popularity <<
", Rating: " << rating << ", Score: " << score << "\n";
}
```

处理更新操作：对于非查询操作（如插入、更新、删除），执行后可以检查返回码或受影响的行数。

```
SQLRowCount(hStmt, &rowCount);
printf("%d rows affected.\n", rowCount);
```

5. 释放资源

释放语句句柄：释放分配的 SQL 语句句柄。

```
SQLFreeStmt(hStmt, SQL_CLOSE);
```

断开数据库连接：调用 SQLDisconnect 断开数据库连接。

```
SQLDisconnect(hDbc);
```

释放连接句柄和环境句柄：释放分配的句柄，清理资源。

```
SQLFreeHandle(SQL_HANDLE_DBC, hDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hEnv);
```

如下图：