GaussDB(for openGauss)数据库实验报告

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)。

三、 实验环境说明

1. 本实验环境为华为云 GaussDB(for openGauss)数据库；
2. 为了满足本实验需要，实验环境采用以下配置：

   1）设备名称：数据库

   2）设备型号：GaussDB(for openGauss) 8 核 | 64 GB

   3）软件版本：GaussDB(for openGauss) 2020 主备版

四、 实验步骤与要求

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。

i. 创建用户 stu119，登录密码为 buptdata@123，在 SQL 查询页面，输入如下 SQL 语句：

ii. `CREATE USER stu119 PASSWORD'buptdata@123';`

iii. 截图如下：

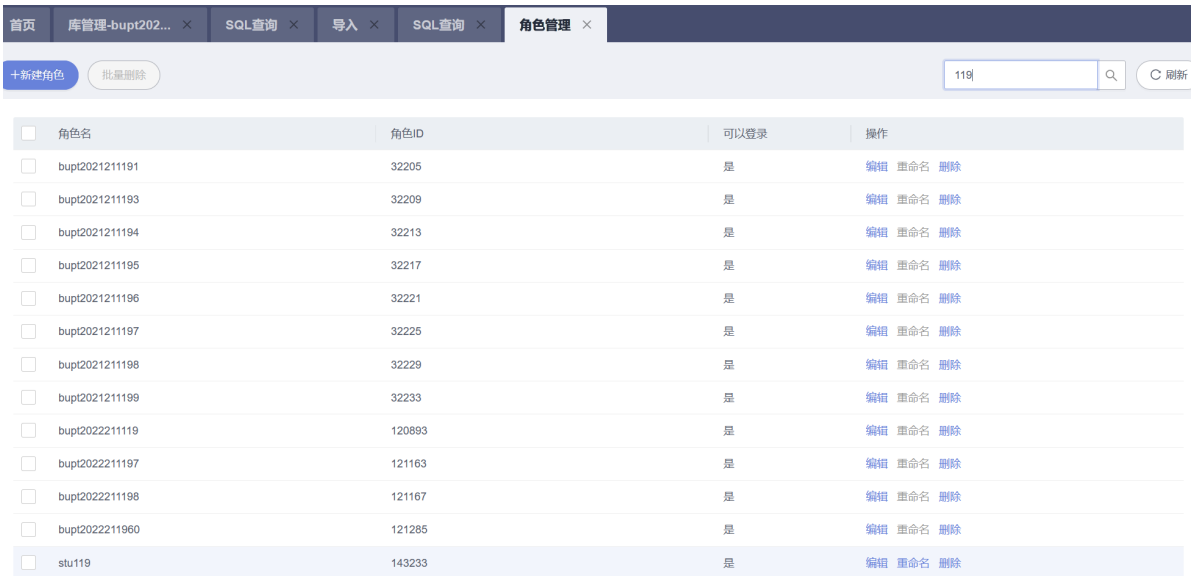
v. 同样的下面语句也可以创建用户。`CREATE USER stu IDENTIFIED BY 'buptdata@123';`

v. 如果创建有“创建数据库”权限的用户，需要加 CREATEDB 关键字。

`CREATE USER stu 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：

单击角色名 stu119，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 stu 的登录密码由 buptdata@123 修改为 Abcd@1231，单击保存：

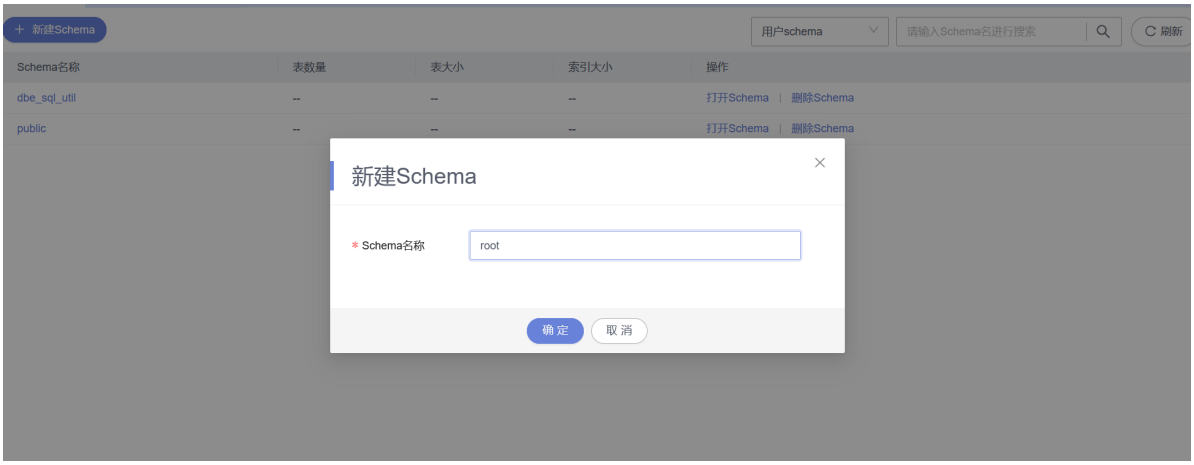
显示 SQL 预览，单击确定，修改成功。

（3） 为用户 stu 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 yiqing:

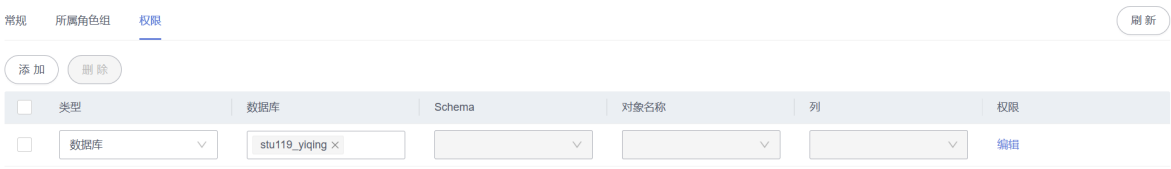
b) 单击库管理，创建 root 用户的同名 Schema:



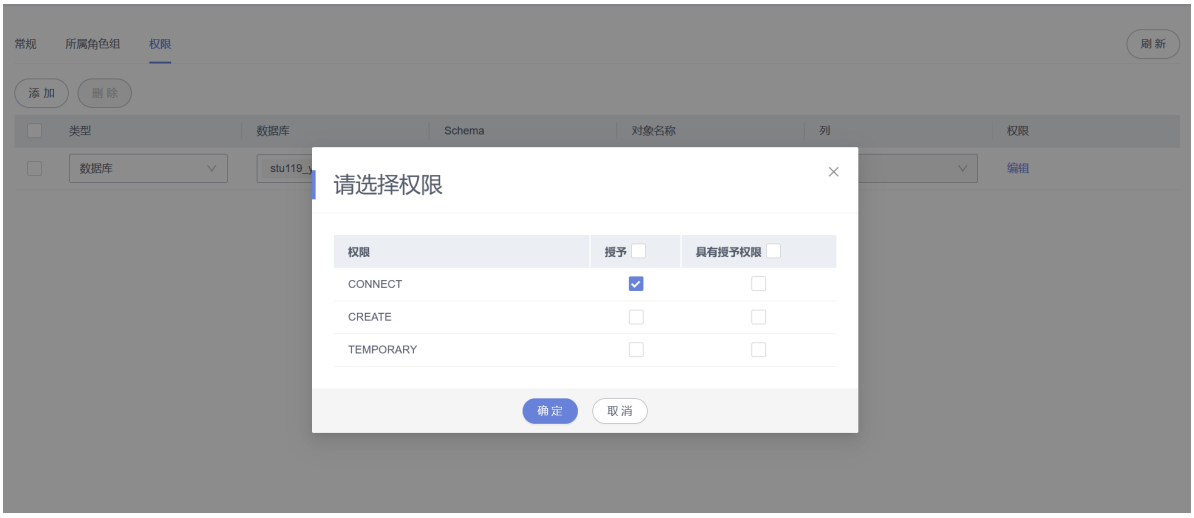
c) 创建成功后，单击 SQL 窗口，用以下语句在窗口中创建一张样例表，具体如下：

```
1  CREATE TABLE 样例(testid int);
```

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



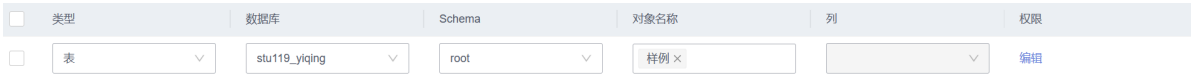
e) 勾选授予 CONNECT 权限：



f) 再次单击添加，类型选择 Schema，数据库选择 yiqing，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 yiqing，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 yiqing 数据库的 SQL 查询，输入查询语句：

```
1 select * from 样例;
```

l)查询结果如下：

五、 实验总结

在本次实验中遇到的第一个问题是角色名会有重名，而PPT给的密码也被判定为弱密码无法使用，所以我对用户名和密码都进行了一点修改，然后在为用户添加可以创建数据库的权限时，不知道什么原因网站里面的角色名以及角色ID总是无法加载出来，经过多次尝试后才成功添加权限，除此之外没有遇到其他问题。本次实验我学会了如何创建用户，并为用户添加和设置相应的权限，收获颇丰。

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)。

三、 实验环境说明

1. 本实验环境为华为云 GaussDB(for openGauss)数据库；
2. 为了满足本实验需要，实验环境采用以下配置：

（1）设备名称：数据库

（2）设备型号：GaussDB(for openGauss) 8 核 | 64 GB

（3）软件版本：GaussDB(for openGauss) 2020 主备版

四、 实验步骤与要求

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：

a) 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。

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

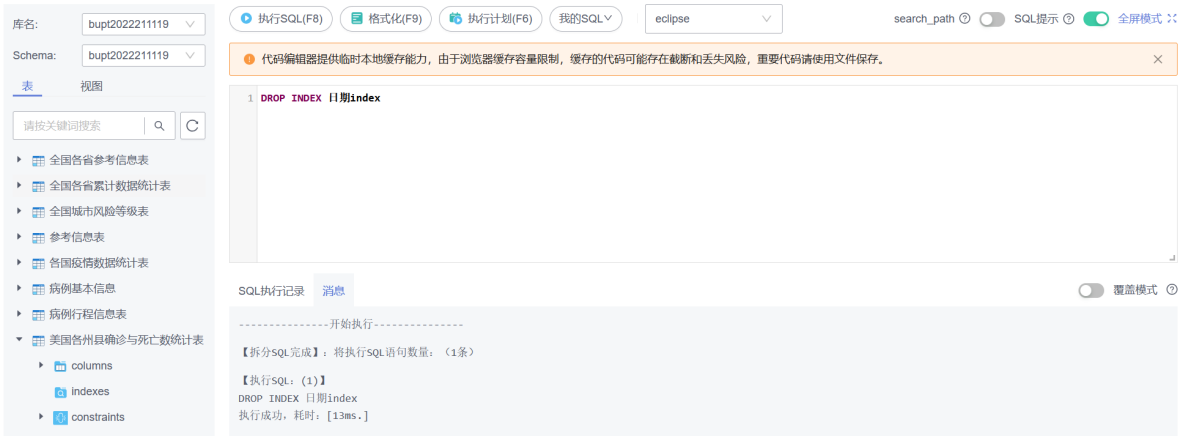
（3） 管理索引

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;



c) 索引创建练习

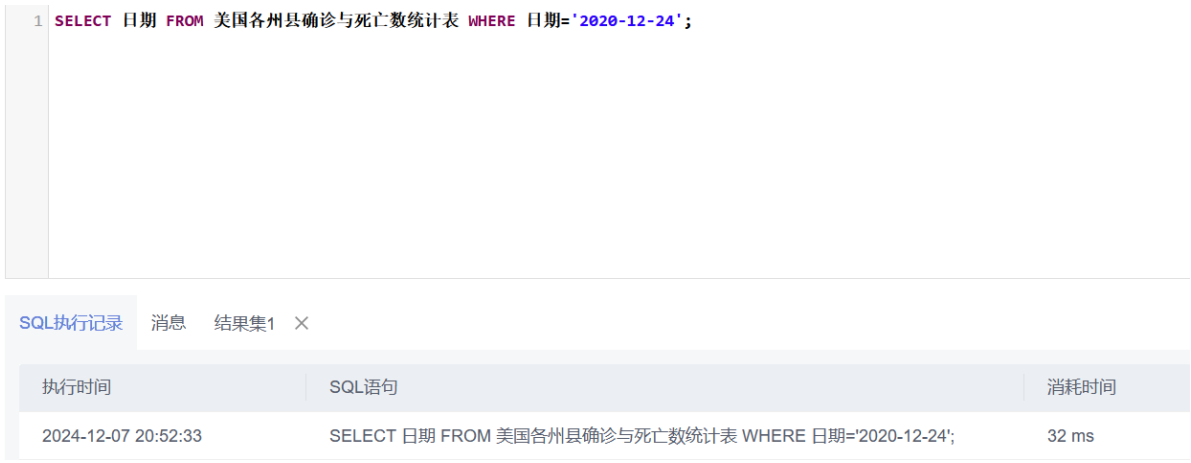
对美国各州县确诊与死亡数统计表创建以下四类索引，尝试比较未建索引与创建索引后，查询效率的不同。

d) 创建唯一索引：尝试比较未建索引后与创建索引后，查询效率的不同。

2 创建索引练习

（1） 如果对于“美国各州县确诊与死亡数统计表 1“，需要经常进行以下查询。

```
SELECT 日期 FROM 美国各州县确诊与死亡数统计表 1 WHERE 日期='2020-12-24';
```



使用以下命令创建索引。

```
CREATE INDEX 日期index1 ON 美国各州县确诊与死亡数统计表 1 (日期);
SELECT 日期 FROM 美国各州县确诊与死亡数统计表 1 WHERE 日期='2020-12-24';
```

比没有创建索引快了4倍左右。

（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数统计表 2”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数统计表 2 WHERE 日期= '2020-12-24' AND 累计确诊
>1000;
```

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

SQL执行记录 消息 结果集1 ×

执行时间	SQL语句	消耗时间
2024-12-07 20:55:44	SELECT * FROM 美国各州县确诊与死亡数统计表 WHERE 日期= '2020-12-24' AND 累计确诊>1000;	44 ms

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。

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

再进行查询得

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

SQL执行记录 消息 结果集1 ×

执行时间	SQL语句	消耗时间
2024-12-07 20:56:30	SELECT * FROM 美国各州县确诊与死亡数统计表 WHERE 日期= '2020-12-24' AND 累计确诊>1000;	11 ms

创建索引后查询速度也是原来的4倍。

（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。

由于在前面已展示过查询日期的时间，在这里不再展示。

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

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

SQL执行记录 消息 结果集1 ×

执行时间	SQL语句	消耗时间
2024-12-07 21:00:46	SELECT 日期 FROM 美国各州县确诊与死亡数统计表 WHERE 日期='2020-12-24';	8 ms

可以看到执行时间为8ms，与前面创建单个索引效率相近。

（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数统计表 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数统计表 (trunc(累计确诊));
```

可以看到查询耗时少了53ms。

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`

```
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 AS y ON x.病例号 = y.病例号 WHERE y.省 = '北京市'
```

（3） 管理视图

a) 查询普通视图

i. 执行如下命令查询 bj_yq 视图。`SELECT * FROM bj_yq;`

ii. 截图如下：

```
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 "bupt2022211119"."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 ("病例行程信息表" x
```

关闭

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。

   由于前面创建的`bi_yq`视图就为北京市病例信息的试图==视图，过程也已展示，在这里不再展示。

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）

执行SQL指令如下：

```
SELECT 病例号, 行程号, 性别, 日期信息
FROM bj_yq
WHERE 行程信息 LIKE '%普通型%'
ORDER BY 病例号
LIMIT 5
```

查询结果如下：

```sql
SELECT 病例号, 行程号, 性别, 日期信息
FROM bj_yq
WHERE 行程信息 LIKE '%普通型%'
ORDER BY 病例号
LIMIT 5
```

SQL执行记录 消息 结果集1 覆盖模式

以下是SELECT 病例号, 行程号, 性别, 日期信息 FROM bj_yq WHERE ... ⓘ该表不可编辑。 复制行 复制列 列设置

	病例号	行程号	性别	日期信息
1	25	1708	男	1月19日
2	27	1825	男	1月19日
3	28	1899	女	1月19日
4	29	1959	女	1月19日
5	31	2096	男	1月19日

五、 实验总结

在本次实验中，刚开始创建索引时，我对创建索引的语法不熟悉，而课件上所给的语法直接复制粘贴的话是错误的，所以我去查询了一下相关的语法，并进行了修正，然后将索引效率进行比较的时候，我刚开始疏忽了已有索引对后续查找的效率影响，所以我后来发现这个问题后在每次对比前都把原来已经创建的索引删除来控制变量，以便最好地比较有无索引的效率差异，对于视图部分则没有遇到什么比较大的问题。

在本次实验中，我学习了如何创建索引和视图，并了解了几种不同类型的索引，同时巩固了课堂上所学的索引和视图的知识。

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。

三、 实验环境说明

1. 本实验环境为华为云 GaussDB(for openGauss)数据库；
2. 为了满足本实验需要，实验环境采用以下配置：

   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

四、 实验步骤与要求

1. 创建存储过程：

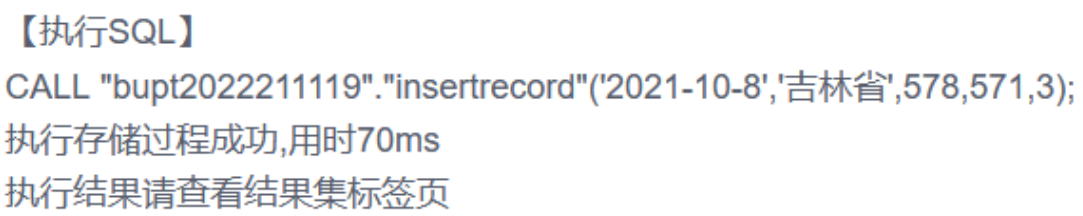
1. 创建存储过程：在全国各省累计数据统计表中增加一条记录。执行存储过程：增加 2021 年 10 月 8 日吉林省累计确诊 578 例，累计治愈 571 例，累计死亡 3 例。

执行指令如下：

```
CREATE OR REPLACE PROCEDURE insertRecord ("日期" DATE , "省" VARCHAR(50) , "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 VALUES (日期, 省, 累计确诊, 累计治愈, 累计死亡);
END;
```

在对象列表的存储过程执行插入操作：

执行结果：



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。

过程创建指令如下：

```
CREATE OR REPLACE PROCEDURE queryRecord ("指定州" VARCHAR , "指定日期" DATE , OUT "累计确诊总数" INT , OUT "累计死亡总数" INT )
AS DECLARE
BEGIN
SELECT SUM(累计确诊), SUM(累计死亡) INTO 累计确诊总数, 累计死亡总数 FROM 美国各州县确诊与死亡数统计表 WHERE 日期 = 指定日期 AND 州 = 指定州;
END;
```

创建完成后开始执行，在这里我们换成使用CALL进行调用：

```
CALL bupt2022211119.queryrecord('California', '2021-1-1', 累计确诊总数, 累计死亡总数);
```

结果如下：

以下是CALL bupt2022211119.queryrecord('California', '2021-1-1', 累计… ⓘ 该表不可编辑。

	累计确诊总数	累计死亡总数
1	2365024	26363

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 queryRecord_3 ("指定日期" DATE , OUT "中国累计确诊总数" INT , OUT "美国累计确诊总数" INT )
AS DECLARE
BEGIN
SELECT 中国累计确诊,美国累计确诊 INTO 中国累计确诊总数, 美国累计确诊总数
FROM (
    SELECT SUM(累计确诊) AS 中国累计确诊
    FROM 全国各省累计数据统计表
    GROUP BY 日期
    HAVING 日期 = 指定日期
) AS 中国确诊
NATURAL JOIN(
    SELECT SUM(累计确诊) AS 美国累计确诊
    FROM 美国各州县确诊与死亡数统计表
    GROUP BY 日期
    HAVING 日期 = 指定日期
) AS 美国确诊;
END;
```

查询结果如下：

```
CALL bupt2022211119.queryrecord_3('2020-12-15', 中国累计确诊总数, 美国累计确诊总数);
```

SQL执行记录 消息 结果集1

以下是CALL bupt2022211119.queryrecord_3('2020-12-15', 中国累计确... 该表不可编辑。

	中国累计确诊总数	美国累计确诊总数
1	94538	16810792

4. 创建存储过程：向全国各省累计数据统计表增加记录。

在第一题中我们就已经创建了这样一个存储过程：

```
CREATE OR REPLACE PROCEDURE insertRecord ("日期" DATE , "州" VARCHAR(50) , "县" VARCHAR(50),"累计确诊" INT, "累计死亡" INT)
AS DECLARE BEGIN
INSERT INTO 美国各州县确诊与死亡数统计表 VALUES (日期, 州, 县, 累计确诊, 累计死亡);
END;
```

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。

若使用触发器的话则需要创建对应的函数：

```
CREATE OR REPLACE FUNCTION check_on_insert()
RETURNS trigger AS
$$
BEGIN
    IF NEW.州 NOT IN (SELECT 省州 FROM 参考信息表 WHERE 国家地区 = 'US')
       OR NOT EXISTS (
            SELECT 市县
            FROM 参考信息表
            WHERE 国家地区 = 'US' AND 省州 = NEW.州 AND 市县 = NEW.县
       )
    THEN
        RAISE EXCEPTION '未通过数据一致性验证';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

然后创建触发器：

```
CREATE TRIGGER trigger_insert BEFORE INSERT ON 美国各州县确诊与死亡数统计表 FOR EACH ROW EXECUTE PROCEDURE check_on_insert();
```

使用存储过程的话，则使用以下SQL语句：

```
CREATE OR REPLACE PROCEDURE insert_record_5(
    日期 DATE,
    州 VARCHAR,
    县 VARCHAR,
    累计确诊数 INT,
    累计死亡数 INT
)
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM 参考信息表 WHERE 国家 = 'US' AND 省州 = 州) THEN
        RAISE EXCEPTION '该州不存在于参考信息表中';
    END IF;

    IF NOT EXISTS (
        SELECT 1 FROM 参考信息表 WHERE 国家 = 'US' AND 省州 = 州 AND 市县 = 县
    ) THEN
        RAISE EXCEPTION '该州县不存在于参考信息表中';
    END IF;

    INSERT INTO 美国各州县确诊与死亡数统计表 (日期, 州, 县, 累计确诊, 累计死亡)
    VALUES (日期, 州, 县, 累计确诊数, 累计死亡数);
END;
```

执行结果如下：

```
CALL bupt2022211119.insert_record_5('2021-12-30', 'abcd', 'CAlifornia', 100, 50)
```

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)】
CALL bupt2022211119.insert_record_5('2021-12-30', 'abcd', 'CAlifornia', 100, 50)
执行失败，失败原因：ERROR: 该州不存在于参考信息表中
```

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。

相应SQL语句如下：

```
CREATE OR REPLACE PROCEDURE delete_record_6("删除病例号" INT)
AS
BEGIN
    DELETE FROM 病例基本信息 WHERE 病例号 = 删除病例号;
    DELETE FROM 病例行程信息表 WHERE 病例号 = 删除病例号;
END;
```

7. 创建存储过程：查询某城市的风险地区等级数量，输入石家庄市，中/高风险地区，输出对应的中/高风险地区数量

执行SQL语句为：

```
CREATE OR REPLACE PROCEDURE query_record_7("查询城市" VARCHAR,  "查询风险等级" VARCHAR, OUT "相应风险地区数目" INT)
AS
BEGIN
    SELECT COUNT(*) INTO 相应风险地区数目
    FROM 全国城市风险等级表
    WHERE 市 = 查询城市 AND 风险等级 = 查询风险等级;
END;
```

现在调用执行：

```
CALL bupt2022211119.query_record_7('石家庄市', '中风险地区', 相应风险地区数)
```

SQL执行记录　消息　结果集1 ×

以下是CALL bupt2022211119.query_record_7('石家庄市', '中风险地区',... ⓘ该表不可编辑。

	相应风险地区数目
1	40

```
CALL bupt2022211119.query_record_7('石家庄市', '高风险地区', 相应风险地区数)
```

SQL执行记录 消息 结果集1

以下是CALL bupt2022211119.query_record_7('石家庄市', '高风险地区',... 该表不可编辑。

	相应风险地区数目
1	2

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; 截图如下：

```
drop procedure insertRecord;
```

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 insertRecord;
执行成功，耗时：[10ms.]
```

五、实验总结

本次实验我遇到的第一个问题就是对存储过程的创建的语法不熟悉，刚开始无从下手，在研究了课件并上手实践后，虽然时常犯一些忘记加分号的错误，但是还是基本掌握了存储过程相关的基本语法。

第二个问题则是在执行过程时，在一开始我是在库管理->对象列表->存储过程->修改或执行里面对过程进行执行的，这样只需要在弹出的表格中填好相应参数即可，但是在我的过程有两个输入两个输出时，执行就一直失败，提示我有两个参数不存在，经多次查询我发现是输出参数也需要申明，但是弹出来的表格里面并没有输出参数，所以后来执行过程我采用了`CALL`命令来执行，解决了这个问题。

在本次实验中我学会了对存储过程进行创建和管理，并能够使用存储过程进行一些操作，同时使用存储过程实现类似触发器的功能，使我对SQL语法的熟悉程度大大增加，收获颇丰。

对本次实验，题目5，6，7都可以使用触发器实现，而且课件里面也有触发器相关语法，但是题目里面要求都是使用存储过程，而没有触发器相关的题目，希望能够更新课件，同时希望课件中对于存储过程和触发器的相关语法更加清晰一些，最好能够给出一些简单的例子，方便我们学习。

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

三、 实验环境说明

1. 本实验环境为华为云 GaussDB(for openGauss)数据库；
2. 为了满足本实验需要，实验环境采用以下配置：

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

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；

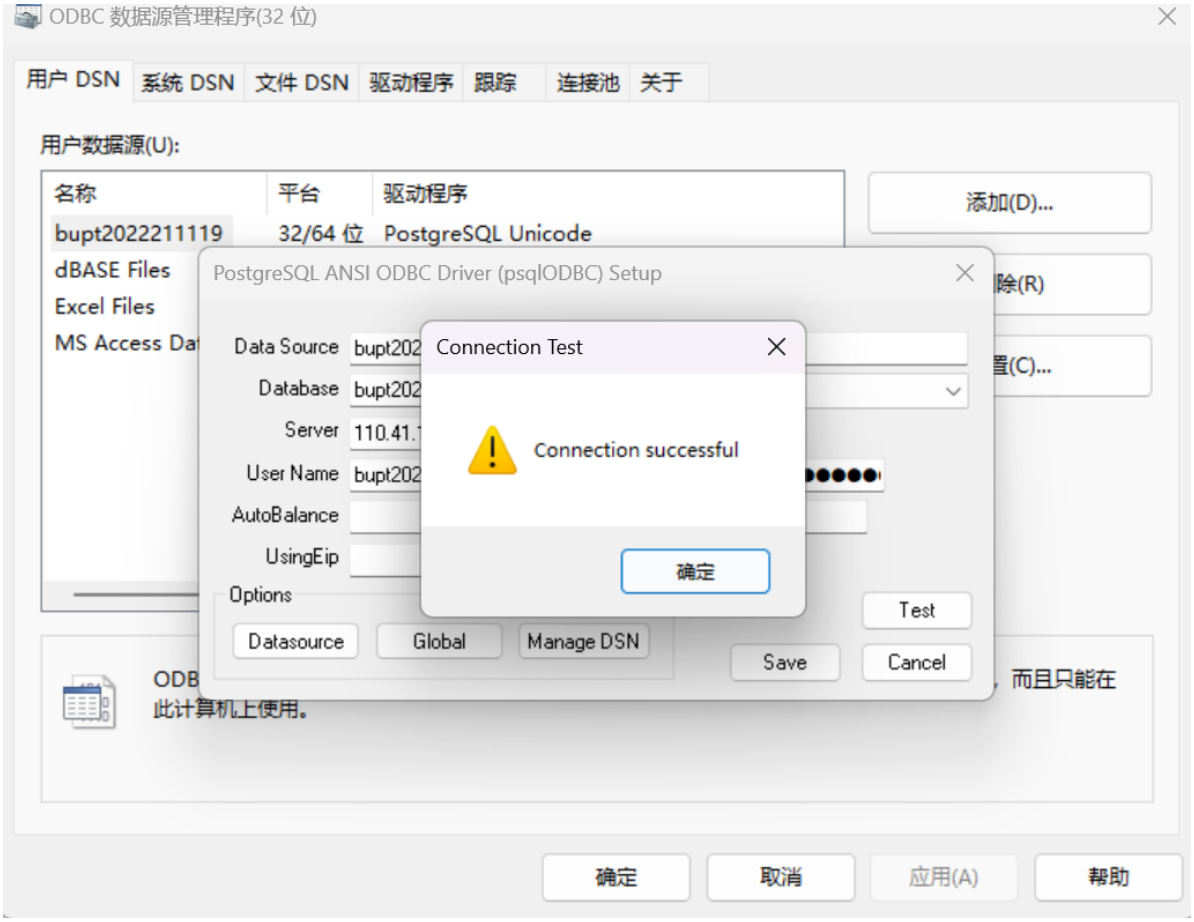
(i) SQLFreeStmt：释放与语句句柄相关的资源；

(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。

首先安装驱动并配置好数据源：



然后撰写代码：

```
// 此示例演示如何通过ODBC方式获取GaussDB(for openGauss)中的数据。
// DBtest.c (compile with: libodbc.so)
//病例号，省，市，区，日期，性别，年龄，患者信息，其他信息，信息来源
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#endif
SQLHENV      V_OD_Env;      // Handle ODBC environment
SQLHSTMT     V_OD_hstmt;    // Handle statement
```

```
SQLHDBC       V_OD_hdbc;       // Handle connection
SQLINTEGER    V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
char          query[1000];     //储存SQL语句
SQLLEN        result_len;

void query_data(int flag) {
      // 查询数据
      if(flag == 1) {
            getchar();
            printf("Please input query SQL: \n");
            gets(query);
      }
      else
            snprintf(query, sizeof(query),
                  "SELECT * FROM 病例基本信息 WHERE 省 = '北京市' AND 市 = '北京市' AND 区 = '北京市'");
      V_OD_erg = SQLExecDirect(V_OD_hstmt, (SQLCHAR *)query, SQL_NTS);
      if (V_OD_erg == SQL_SUCCESS || V_OD_erg == SQL_SUCCESS_WITH_INFO) {
            SQLINTEGER case_no, age;
            char province[50], city[50], district[50], date[20], gender[10], info[100], other_info[100], source[50];
            while (SQLFetch(V_OD_hstmt) != SQL_NO_DATA) {
                  SQLGetData(V_OD_hstmt, 1, SQL_C_SLONG, &case_no, 0, &result_len);
                  SQLGetData(V_OD_hstmt, 2, SQL_C_CHAR, province, sizeof(province), &result_len);
                  SQLGetData(V_OD_hstmt, 3, SQL_C_CHAR, city, sizeof(city), &result_len);
                  SQLGetData(V_OD_hstmt, 4, SQL_C_CHAR, district, sizeof(district), &result_len);
                  SQLGetData(V_OD_hstmt, 5, SQL_C_CHAR, date, sizeof(date), &result_len);
                  SQLGetData(V_OD_hstmt, 6, SQL_C_CHAR, gender, sizeof(gender), &result_len);
                  SQLGetData(V_OD_hstmt, 7, SQL_C_SLONG, &age, 0, &result_len);
                  SQLGetData(V_OD_hstmt, 8, SQL_C_CHAR, info, sizeof(info), &result_len);
                  SQLGetData(V_OD_hstmt, 9, SQL_C_CHAR, other_info, sizeof(other_info), &result_len);
                  SQLGetData(V_OD_hstmt, 10, SQL_C_CHAR, source, sizeof(source), &result_len);

                  printf("病例号: %d, 省: %s, 市: %s, 区: %s, 日期: %s, 性别: %s, 年龄: %d, 患者信息: %s 其他信息: %s, 信息来源: %s\n",
                        case_no, province, city, district, date, gender, age, info, other_info, source);
            }
      } else {
            printf("Error querying data.\n");
            // 获取并打印错误信息
            SQLCHAR sqlState[6], messageText[SQL_MAX_MESSAGE_LENGTH];
            SQLINTEGER nativeError;
            SQLSMALLINT textLength;
```

```
            SQLGetDiagRec(SQL_HANDLE_STMT, V_OD_hstmt, 1, sqlState,
&nativeError, messageText, sizeof(messageText), &textLength);
            printf("SQL Error: %d, SQL State: %s, Message: %s\n",
nativeError, sqlState, messageText);
      }
      SQLFreeStmt(V_OD_hstmt, SQL_CLOSE);//重置语句句柄
}

void delete_data(int flag) {
      // 删除数据
      if(flag == 1) {
            getchar();
            printf("Please input delete SQL: \n");
            gets(query);
      }
      else
            snprintf(query, sizeof(query),
                  "DELETE FROM 病例基本信息 WHERE 病例号 = 26");
      V_OD_erg = SQLExecDirect(V_OD_hstmt, (SQLCHAR *)query, SQL_NTS);
      if (V_OD_erg == SQL_SUCCESS || V_OD_erg == SQL_SUCCESS_WITH_INFO) {
            printf("Delete data successfully.\n");
      } else {
            printf("Error deleting data.\n");
            // 获取并打印错误信息
            SQLCHAR sqlState[6], messageText[SQL_MAX_MESSAGE_LENGTH];
            SQLINTEGER nativeError;
            SQLSMALLINT textLength;
            SQLGetDiagRec(SQL_HANDLE_STMT, V_OD_hstmt, 1, sqlState,
&nativeError, messageText, sizeof(messageText), &textLength);
            printf("SQL Error: %d, SQL State: %s, Message: %s\n",
nativeError, sqlState, messageText);
      }
      SQLFreeStmt(V_OD_hstmt, SQL_CLOSE);
}

void update_data(int flag) {
      // 更新数据
      if(flag == 1) {
            getchar();
            printf("Please input update SQL: \n");
            gets(query);
      }
      else
            snprintf(query, sizeof(query),
                  "UPDATE 病例基本信息 SET 其它信息 = '现居住于海淀区北太平庄街道北
京邮电大学社区' WHERE 病例号 = 25");
      V_OD_erg = SQLExecDirect(V_OD_hstmt, (SQLCHAR *)query, SQL_NTS);
      if (V_OD_erg == SQL_SUCCESS || V_OD_erg == SQL_SUCCESS_WITH_INFO) {
            printf("Update data successfully.\n");
      } else {
            printf("Error updating data.\n");
            // 获取并打印错误信息
            SQLCHAR sqlState[6], messageText[SQL_MAX_MESSAGE_LENGTH];
            SQLINTEGER nativeError;
            SQLSMALLINT textLength;
```

```
            SQLGetDiagRec(SQL_HANDLE_STMT, V_OD_hstmt, 1, sqlState, &nativeError, messageText, sizeof(messageText), &textLength);
            printf("SQL Error: %d, SQL State: %s, Message: %s\n", nativeError, sqlState, messageText);
      }
      SQLFreeStmt(V_OD_hstmt, SQL_CLOSE);
}

void insert_data(int flag) {
      // 插入数据
      if(flag == 1) {
            getchar();
            printf("Please input insert SQL: \n");
            gets(query);
      }
      else
            snprintf(query, sizeof(query),
                  "INSERT INTO 病例基本信息 (病例号, 省, 市, 区, 日期, 性别, 年龄, 患者信息, 其它信息, 信息来源) VALUES (26, '北京市', '北京市', '北京市', '2021-09-01', '男', 25, '现居住于海淀区北太平庄街道北京邮电大学社区。', '无', '北京市卫健委')");
      V_OD_erg = SQLExecDirect(V_OD_hstmt, (SQLCHAR *)query, SQL_NTS);
      if (V_OD_erg == SQL_SUCCESS || V_OD_erg == SQL_SUCCESS_WITH_INFO) {
            printf("Insert data successfully.\n");
      } else {
            printf("Error inserting data.\n");
            // 获取并打印错误信息
            SQLCHAR sqlState[6], messageText[SQL_MAX_MESSAGE_LENGTH];
            SQLINTEGER nativeError;
            SQLSMALLINT textLength;
            SQLGetDiagRec(SQL_HANDLE_STMT, V_OD_hstmt, 1, sqlState, &nativeError, messageText, sizeof(messageText), &textLength);
            printf("SQL Error: %d, SQL State: %s, Message: %s\n", nativeError, sqlState, messageText);
      }
      SQLFreeStmt(V_OD_hstmt, SQL_CLOSE);
}

int main(int argc,char *argv[])
{
      // 1. 申请环境句柄
      V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);

      if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))

      {
           printf("Error AllocHandle\n");
           exit(0);
      }
      // 2. 设置环境属性（版本信息）
      SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
      // 3. 申请连接句柄
      V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
```

```
146       if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
147       {
148            SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
149            exit(0);
150       }
151 //4. 设置连接属性
152       SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)SQL_AUTOCOMMIT_ON, 0);
153       printf("*****",V_OD_hdbc);
154 // 5. 连接数据源，这里的“userName”与“password”分别表示连接数据库的用户名和用户密码，
请根据实际情况修改。
155       // 如果odbc.ini文件中已经配置了用户名密码，那么这里可以留空（""）；但是不建议这
么做，因为一旦odbc.ini权限管理不善，将导致数据库用户密码泄露。
156       V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "PostgreSQL35W", SQL_NTS,
157                             (SQLCHAR*) "bupt2022211119", SQL_NTS,
(SQLCHAR*) "bupt2022211119@", SQL_NTS);
158       if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
159       {
160           printf("Error SQLConnect %d\n",V_OD_erg);
161           SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
162           exit(0);
163       }
164       printf("Connected !\n");
165       // 6. 设置语句属性
166       SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
167 // 7. 申请语句句柄
168       SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
169
170       int i = 0;
171       printf("Please input the number of operations: \n");
172       printf(" 1. Default\n 2. Quety\n 3. Delete\n 4. Update\n 5. Insert\n
6. Exit\n");
173       scanf("%d", &i);
174       while(TRUE) {
175             if (i == 1) {
176                   query_data(0);
177                   delete_data(0);
178                   query_data(0);
179                   update_data(0);
180                   query_data(0);
181                   insert_data(0);
182                   query_data(0);
183             } else if (i == 2) {
184                   query_data(1);
185             } else if (i == 3) {
186                   delete_data(1);
187             } else if (i == 4) {
188                   update_data(1);
189             } else if (i == 5) {
190                   insert_data(1);
191             } else if (i == 6) {
192                   break;
```

```
            } else {
                  printf("Invalid input.\n");
            }
            printf("Please input the number of operations: \n");
            printf(" 1. Default\n 2. Quety\n 3. Delete\n 4. Update\n 5. Insert\n 6. Exit\n");
            scanf("%d", &i);
      }

      //  断开数据源连接并释放句柄资源
      SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
      SQLDisconnect(V_OD_hdbc);
      SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
      SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
      return(0);
 }

```

执行结果如下：

```
Please input the number of operations:
 1. Default
 2. Quety
 3. Delete
 4. Update
 5. Insert
 6. Exit
```

首先输出查询病例基本信息表中省，市，区都为北京市的病例：

```
PS C:\Codefield\SQL> .\2.exe
*****Connected !
病例号：25，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：43，患者信息：男，43岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：26，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：38，患者信息：男，38岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：27，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：40，患者信息：男，40岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：28，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：38，患者信息：女，38岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：29，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：64，患者信息：女，64岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：30，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：9，患者信息：男，9岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：31，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：男，31岁 其他信息：现住顺义区北务镇南辛庄户村，信息来源：北京日报
病例号：32，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：33，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：34，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：35，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：31，患者信息：某女 其他信息：暂无，信息来源：北京青年报
病例号：36，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：31，患者信息：某女 其他信息：暂无，信息来源：北京青年报
病例号：37，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：83，省：北京市，市：北京市，区：北京市，日期：2021-01-19，性别：女，年龄：63，患者信息：女，63岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京市疾病预防控制中心
病例号：84，省：北京市，市：北京市，区：北京市，日期：2021-01-19，性别：男，年龄：9，患者信息：男，9岁 其他信息：无症状感染者，现住大兴区天宫院街道融汇社区，为上述确诊病例之外孙，信息来源：北京市疾病预防控制中心
病例号：178，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：女，年龄：63，患者信息：女，63岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京市疾病预防控制中心
病例号：179，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：男，年龄：46，患者信息：男，46岁 其他信息：现住大兴区天宫院街道融汇社区，在某报社工作，信息来源：北京市疾病预防控制中心
病例号：180，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：男，年龄：34，患者信息：男，34岁 其他信息：境外输入病例，中国籍，在塞尔维亚工作，信息来源：北京市疾病预防控制中心
```

然后对病例号26号的病例进行删除操作：

```
Delete data successfully.
病例号：25，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：43，患者信息：男，43岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：27，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：40，患者信息：男，40岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：28，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：38，患者信息：女，38岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：29，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：64，患者信息：女，64岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：30，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：9，患者信息：男，9岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：31，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：男，31岁 其他信息：现住顺义区北务镇南辛庄户村，信息来源：北京日报
病例号：32，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：33，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：34，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：35，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：31，患者信息：某女 其他信息：暂无，信息来源：北京青年报
病例号：36，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：31，患者信息：某女 其他信息：暂无，信息来源：北京青年报
病例号：37，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：83，省：北京市，市：北京市，区：北京市，日期：2021-01-19，性别：女，年龄：63，患者信息：女，63岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京市疾病预防控制中心
病例号：84，省：北京市，市：北京市，区：北京市，日期：2021-01-19，性别：男，年龄：9，患者信息：男，9岁 其他信息：无症状感染者，现住大兴区天宫院街道融汇社区，为上述确诊病例之外孙，信息来源：北京市疾病预防控制中心
病例号：178，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：女，年龄：63，患者信息：女，63岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京市疾病预防控制中心
病例号：179，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：男，年龄：46，患者信息：男，46岁 其他信息：现住大兴区天宫院街道融汇社区，在某报社工作，信息来源：北京市疾病预防控制中心
病例号：180，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：男，年龄：34，患者信息：男，34岁 其他信息：境外输入病例，中国籍，在塞尔维亚工作，信息来源：北京市疾病预防控制中心
```

可以看到26号病例号已经被删除了，然后进行`update`修改操作，将病例号为25号的患者的其他信息修改为`现居住于海淀区北太平庄街道北京邮电大学社区`：

```
病例号：25，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：43，患者信息：男，43岁 其他信息：现居住于海淀区北太平庄街道北京邮电大学社区，信息来源：北京日报
```

然后再进行插入操作，将26号患者的信息进行一定的修改并插入回去：

```
病例号：27，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：40，患者信息：男，40岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：28，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：38，患者信息：女，38岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：29，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：64，患者信息：女，64岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：30，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：9，患者信息：男，9岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京日报
病例号：31，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：男，31岁 其他信息：现住顺义区北务镇南辛庄户村，信息来源：北京日报
病例号：32，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：33，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：34，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：35，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：31，患者信息：某女 其他信息：暂无，信息来源：北京青年报
病例号：36，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：女，年龄：31，患者信息：某女 其他信息：暂无，信息来源：北京青年报
病例号：37，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：31，患者信息：某男 其他信息：暂无，信息来源：北京青年报
病例号：25，省：北京市，市：北京市，区：北京市，日期：2021-01-20，性别：男，年龄：43，患者信息：男，43岁 其他信息：现居住于海淀区北太平庄街道北京邮电大学社区，信息来源：北京日报
病例号：83，省：北京市，市：北京市，区：北京市，日期：2021-01-19，性别：女，年龄：63，患者信息：女，63岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京市疾病预防控制中心
病例号：84，省：北京市，市：北京市，区：北京市，日期：2021-01-19，性别：男，年龄：9，患者信息：男，9岁 其他信息：无症状感染者，现住大兴区天宫院街道融汇社区，为上述确诊病例之外孙，信息来源：北京市疾病预防控制中心
病例号：178，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：女，年龄：63，患者信息：女，63岁 其他信息：现住大兴区天宫院街道融汇社区，信息来源：北京市疾病预防控制中心
病例号：179，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：男，年龄：46，患者信息：男，46岁 其他信息：现住大兴区天宫院街道融汇社区，在某报社工作，信息来源：北京市疾病预防控制中心
病例号：180，省：北京市，市：北京市，区：北京市，日期：2021-01-18，性别：男，年龄：34，患者信息：男，34岁 其他信息：境外输入病例，中国籍，在塞尔维亚工作，信息来源：北京市疾病预防控制中心
病例号：26，省：北京市，市：北京市，区：北京市，日期：2021-09-01，性别：男，年龄：25，患者信息：现居住于海淀区北太平庄街道北京邮电大学社区。 其他信息：无，信息来源：北京市卫健委
```

可以看到成功插入了26号病例的新信息。

同时我还使用python写了一份带有图形页面的ODBC查询程序，代码如下：

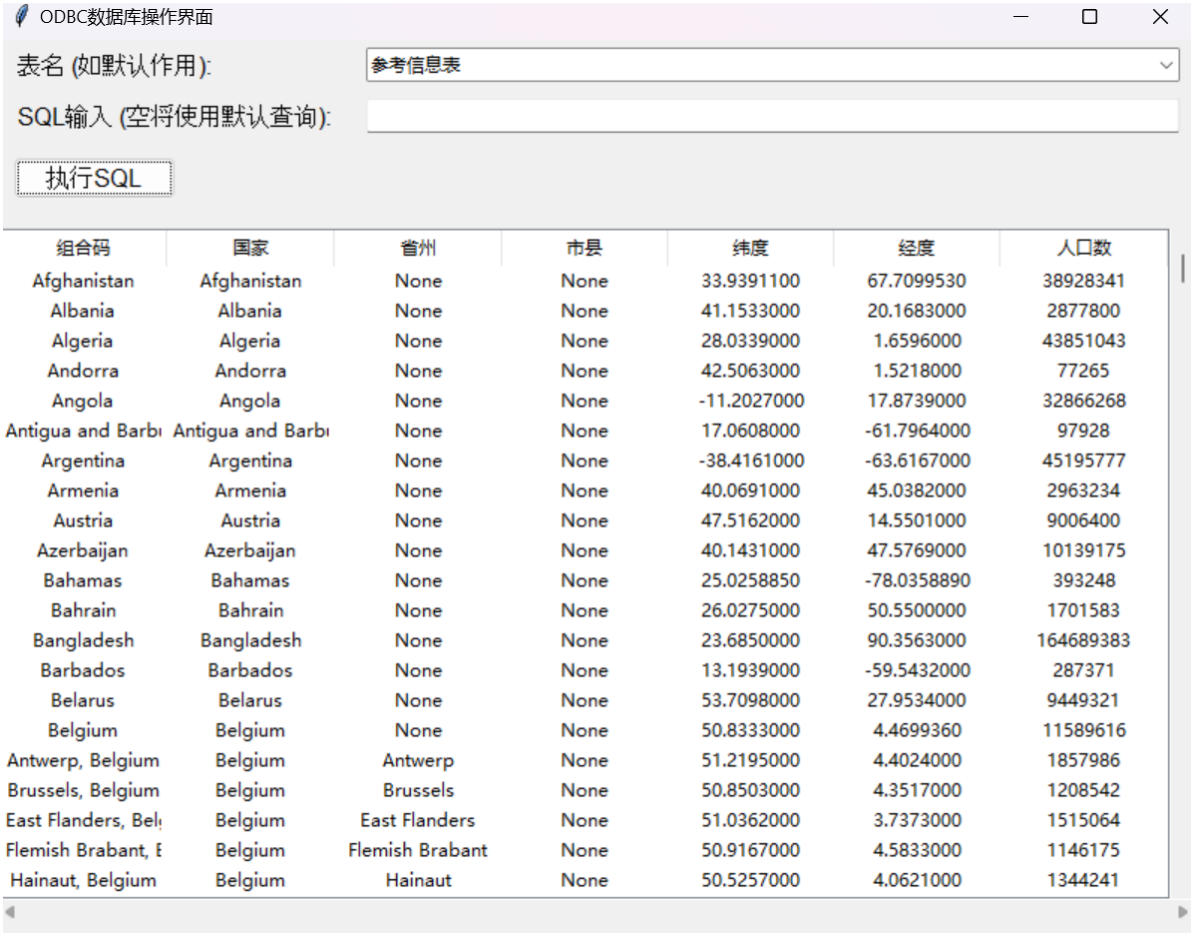
```python
import tkinter as tk
from tkinter import ttk, messagebox
import pyodbc
import datetime

# 数据库连接函数
def connect_to_db():
```

```
    try:
        connection = pyodbc.connect(
            'DSN=PostgreSQL35W;UID=bupt2022211119;PWD=bupt2022211119@'
        )
        return connection
    except Exception as e:
        messagebox.showerror("连接错误", f"无法连接到数据库：{e}")
        return None

# 执行SQL操作
def execute_query(query):
    conn = connect_to_db()
    if conn is None:
        messagebox.showerror("执行错误", "数据库连接失败，无法执行SQL操作！")
        return [], []

    try:
        cursor = conn.cursor()
        print("执行的 SQL:", query)

        cursor.execute(query)

        if query.strip().upper().startswith("SELECT"):
            if cursor.description:
                columns = [column[0] for column in cursor.description]
                results = cursor.fetchall()
            else:
                columns = []
                results = []
        else:
            conn.commit()
            columns = []
            results = []

        return columns, results

    except Exception as e:
        messagebox.showerror("执行错误", f"数据库操作失败：{e}")
        conn.rollback()
        return [], []

    finally:
        conn.close()

# 解析SQL和显示结果
def handle_sql():
    sql = sql_entry.get()

    if not sql.strip():
        table_name = table_var.get()
        sql = f"SELECT * FROM {table_name}"

    columns, results = execute_query(sql)
    display_results(columns, results)

```

```python
# 显示SQL查询结果
def display_results(columns, results):
    # 清空现有的数据
    for i in tree.get_children():
        tree.delete(i)

    # 设置 Treeview 的列标题
    tree["columns"] = columns
    tree["show"] = "headings"

    for col in columns:
        tree.heading(col, text=col)
        tree.column(col, width=150, anchor="center")

    # 格式化结果，处理 datetime 对象
    formatted_results = []
    for row in results:
        formatted_row = []
        for value in row:
            if isinstance(value, datetime.datetime):  # 如果是 datetime 对象
                formatted_row.append(value.strftime("%Y-%m-%d %H:%M:%S"))  # 转为字符串格式
            else:
                formatted_row.append(value)
        formatted_results.append(tuple(formatted_row))

    # 插入格式化后的数据
    for row in formatted_results:
        tree.insert("", "end", values=row)

# 主界面
app = tk.Tk()
app.title("ODBC数据库操作界面")
app.geometry("800x600")

# 自适应布局设置
app.grid_rowconfigure(6, weight=1)
app.grid_columnconfigure(1, weight=1)

# 样式调整
style = ttk.Style()
style.configure("TButton", font=("Arial", 12))
style.configure("TLabel", font=("Arial", 12))
style.configure("TEntry", font=("Arial", 12))

# 表名输入
tk.Label(app, text="表名 (如默认作用):", font=("Arial", 12)).grid(row=0, column=0, padx=10, pady=5, sticky="w")
table_var = tk.StringVar()
table_dropdown = ttk.Combobox(app, textvariable=table_var, width=40, state="readonly")
table_dropdown["values"] = [
    "病例基本信息",
    "病例行程信息表",
    "参考信息表",
```

```
    "各国疫情数据统计表",
    "美国各州县确诊与死亡数统计表",
    "全国城市风险等级表",
    "全国各省参考信息表",
    "全国各省累计数据统计表"
]
table_dropdown.grid(row=0, column=1, padx=10, pady=5, sticky="ew")

# SQL输入框
tk.Label(app, text="SQL输入 (空将使用默认查询):", font=("Arial",
12)).grid(row=1, column=0, padx=10, pady=5, sticky="w")
sql_entry = ttk.Entry(app, width=60)
sql_entry.grid(row=1, column=1, padx=10, pady=5, sticky="ew")

# 操作按钮
button_frame = tk.Frame(app)
button_frame.grid(row=2, column=0, columnspan=2, pady=10, sticky="ew")

execute_btn = ttk.Button(button_frame, text="执行SQL", command=handle_sql)
execute_btn.pack(side="left", padx=10)

# 查询结果显示区域
result_frame = tk.Frame(app)
result_frame.grid(row=6, column=0, columnspan=2, pady=10, sticky="nsew")

scrollbar_x = ttk.Scrollbar(result_frame, orient="horizontal")
scrollbar_y = ttk.Scrollbar(result_frame, orient="vertical")

scrollbar_x.pack(side="bottom", fill="x")
scrollbar_y.pack(side="right", fill="y")

tree = ttk.Treeview(result_frame, show="headings",
xscrollcommand=scrollbar_x.set, yscrollcommand=scrollbar_y.set)
tree.pack(fill="both", expand=True)

scrollbar_x.config(command=tree.xview)
scrollbar_y.config(command=tree.yview)

app.mainloop()

```

实现效果如下：



组合码	国家	省州	市县	纬度	经度	人口数
Afghanistan	Afghanistan	None	None	33.9391100	67.7099530	38928341
Albania	Albania	None	None	41.1533000	20.1683000	2877800
Algeria	Algeria	None	None	28.0339000	1.6596000	43851043
Andorra	Andorra	None	None	42.5063000	1.5218000	77265
Angola	Angola	None	None	-11.2027000	17.8739000	32866268
Antigua and Barbı	Antigua and Barbı	None	None	17.0608000	-61.7964000	97928
Argentina	Argentina	None	None	-38.4161000	-63.6167000	45195777
Armenia	Armenia	None	None	40.0691000	45.0382000	2963234
Austria	Austria	None	None	47.5162000	14.5501000	9006400
Azerbaijan	Azerbaijan	None	None	40.1431000	47.5769000	10139175
Bahamas	Bahamas	None	None	25.0258850	-78.0358890	393248
Bahrain	Bahrain	None	None	26.0275000	50.5500000	1701583
Bangladesh	Bangladesh	None	None	23.6850000	90.3563000	164689383
Barbados	Barbados	None	None	13.1939000	-59.5432000	287371
Belarus	Belarus	None	None	53.7098000	27.9534000	9449321
Belgium	Belgium	None	None	50.8333000	4.4699360	11589616
Antwerp, Belgium	Belgium	Antwerp	None	51.2195000	4.4024000	1857986
Brussels, Belgium	Belgium	Brussels	None	50.8503000	4.3517000	1208542
East Flanders, Bel	Belgium	East Flanders	None	51.0362000	3.7373000	1515064
Flemish Brabant, E	Belgium	Flemish Brabant	None	50.9167000	4.5833000	1146175
Hainaut, Belgium	Belgium	Hainaut	None	50.5257000	4.0621000	1344241

五、实验总结

本次实验遇到的第一个问题是配置好数据源后代码还是一直无法和数据源连接，后来输出错误信息发现原因是实验所给的驱动是32位的，我直接安装了实验所给压缩包中的哪一个驱动，导致驱动和计算机的位数不一致，所以连接失败，于是我又按照课件所给网址和流程进行了重新装载和配置，测试后程序能正常连通。

遇到的第二个问题是中文字符乱码问题，发现按照示例程序绑定结果集后输出查询结果，输出会是乱码，于是我认为是字符编码的问题，但是不管怎么修改字符编码都没有解决，后来我直接使用`SQLGetData`API函数并把结果集直接放到相应的参数中，问题得以解决，程序能够正常的输入输出中文。

遇到的第三个问题是在执行插入操作的时候无法执行成功，经输出报错信息，发现报错一直提示命令没有闭合，于是发现是我储存SQL语句的数组设置的太小了，直接把插入语句给截断了，经修改后程序能够正常执行。

本次实验我了解了数据库驱动的概念，并掌握了ODBC开发应用的流程，能够利用C语言程序访问ODBC数据源中的数据并进行修改，收获颇丰。

数据库驱动是应用程序和数据库存储之间的一种接口，数据库厂商为了某一种开发语言环境（比如Java，C）能够实现数据库调用而开发的类似翻译员功能的程序，将复杂的数据库操作与通信抽象成为了当前开发语言的访问接口，它提供了一组标准化的接口和功能，使得开发人员能够通过一致的方法与各种数据库管理系统（DBMS）进行交互，而不需要关心底层数据库的实现细节。

而ODBC 开发的基本流程如下：

1. **安装 ODBC 驱动**
   确保目标数据库的 ODBC 驱动程序已经安装，并通过 ODBC 数据源管理器配置对应的数据源名称（DSN）。

2. **初始化环境**
   通过 ODBC 提供的 API 初始化环境：

分配环境句柄（SQLAllocHandle）。
设置环境属性（例如 ODBC 版本）。

3. **连接到数据库**
   分配连接句柄。
   调用 SQLConnect 或 SQLDriverConnect 连接数据库。
4. **执行 SQL 语句**
   分配语句句柄（SQLAllocHandle）。
   准备和执行 SQL 语句（SQLExecDirect 或 SQLPrepare + SQLExecute）。
   检索结果（通过 SQLFetch 或绑定列值）。
5. **处理结果**
   解析查询结果，或检查执行成功的状态。
   获取返回的错误代码或警告信息（如有）。
6. **释放资源**
   按照以下顺序释放资源，防止内存泄漏：
   释放语句句柄。
   断开数据库连接。
   释放连接句柄。
   释放环境句柄。

通过上述步骤，开发者可以在不同数据库之间轻松切换，极大地提升了数据库访问的灵活性和兼容性。