

数据库系统原理实验四、实验五、实验六、实验七 实验报告

毛子恒

2019211397

北京邮电大学 计算机学院

日期：2021 年 12 月 16 日

Part I

实验四

1 概述

1.1 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)。

1.2 实验平台及环境

- GaussDB(for openGauss) 8 核 | 64 GB
- GaussDB(for openGauss) 2020 主备版

1.3 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)。

2 实验步骤

2.1 创建用户

选择 SQL 操作，单击 SQL 查询，进入 SQL 查询页面。库名选择 postgres，Schema 选择 root。
创建用户，输入以下 SQL 语句：

```
1 CREATE USER stu2019211397 PASSWORD 'buptdata@123';
```

结果如图 1。

```
-----开始执行-----  
  
【拆分SQL完成】：将执行SQL语句数量：（1条）  
  
【执行SQL：（1）】  
CREATE USER stu2019211397 PASSWORD 'buptdata@123';  
执行成功，耗时：[22ms.]
```

图 1

2.2 管理用户

2.2.1 角色管理

选择账号管理，单击角色管理，进入角色管理页面，如图 2。

<input type="checkbox"/>	角色名	角色ID
<input type="checkbox"/>	bupt2019211397	17433
<input type="checkbox"/>	stu2019211397	40335

图 2

单击角色名 stu2019211397，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 stu2019211397 的登录密码由 buptdata@123 修改为 Abcd@123，单击保存，显示 SQL 预览，单击确定，修改成功，如图 3。

为用户 stu2019211397 追加可以创建数据库的权限，勾选“可以创建数据库”复选框，保存，如图 4。

将 bupt2019211397 角色（用户）的权限赋予 stu 用户，选择所属角色组，勾选 bupt2019211397 角色后的“授予”复选框，保存，如图 5：

2.2.2 设置用户权限

创建数据库 yiqing_2019211397，如图 6。

创建名为 root 的 schema，如图 7。

单击 SQL 操作->SQL 查询，创建一张样例表，输入以下 SQL 语句：



图 3



图 4

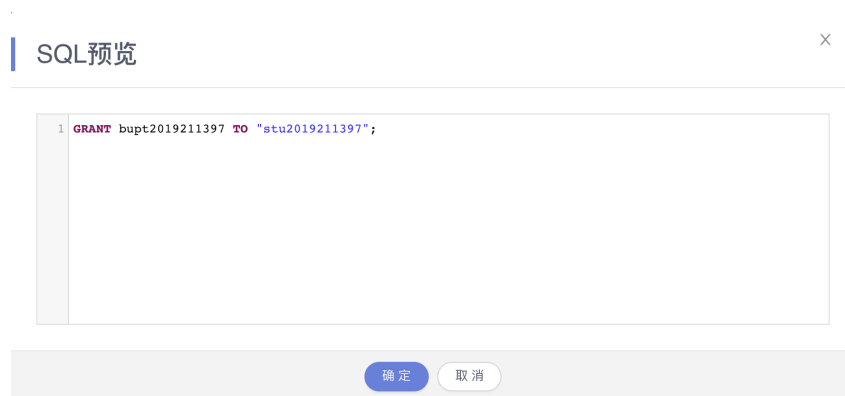


图 5

1 CREATE TABLE 样例 (testid int);

结果如图 8。

单击账户管理-> 角色管理-> 单击角色名 stu2019211397-> 权限-> 添加, 类型选择数据库, 数据库选择 yiqing_2019211397, 然后单击编辑, 勾选授予 CONNECT 权限, 单击确定。

再次单击添加, 类型选择 Schema, 数据库选择 yiqing_2019211397, Schema 选择 root, 单击编辑, 勾选授予 USAGE 权限, 单击确定。

再次单击添加, 类型选择表, 数据库选择 yiqing_2019211397, Schema 选择 root, 对象名称

新建数据库

数据库名称

yiqing_2019211397

只能创建用户数据库

字符集

UTF8

Template

template0

Collation

Ctype

DBCOMPATIBILITY

PostgreSQL

确定

取消

图 6

新建Schema

* Schema名称

root

确定

取消

图 7

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】

CREATE TABLE 样例(testid int);

执行成功，耗时：[7ms.]

图 8

选择样例，单击编辑，勾选授予 SELECT 权限，单击确定。

如图 9。

<input type="checkbox"/>	类型	数据库	Schema	对象名称	列	权限
<input type="checkbox"/>	表	yiqing_2019211397	root	样例		SELECT编辑
<input type="checkbox"/>	Schema	yiqing_2019211397	root			USAGE编辑
<input type="checkbox"/>	数据库	yiqing_2019211397				CONNECT编辑

图 9

添加完成后选择保存，单击确定后，权限添加完毕，如图 10。



图 10

2.2.3 验证用户权限

单击右上角账户名，选择切换连接，用 stu2019211397 账户登录，如图 11。

The image shows a "数据库登录" (Database Login) dialog box with a close button (X) in the top right corner. It contains the following fields and options:

- "当前实例:" (Current Instance): A dropdown menu showing "gauss-fc46".
- "* 数据库名称:" (Database Name): A text input field containing "postgres".
- "* 登录用户名:" (Login Username): A text input field containing "stu2019211397".
- "密码:" (Password): A password input field with a masked password "....." and a clear button (X).
- Two checkboxes: "记住密码" (Remember Password) and "SQL执行记录" (SQL Execution Record).
- A description for the "SQL执行记录" checkbox: "开启此项后，您可以在DAS中，方便的查看到您的SQL窗口执行历史记录，并且可以直接再次执行，无需重复输入。" (After enabling this item, you can conveniently view the execution history of your SQL window in DAS, and you can directly execute it again without repeating the input).
- A large blue "登录" (Login) button at the bottom.

图 11

进入 yiqing_2019211397 数据库的 SQL 查询，输入以下 SQL 语句：

```
1 SELECT * FROM 样例;
```

结果如图 12。

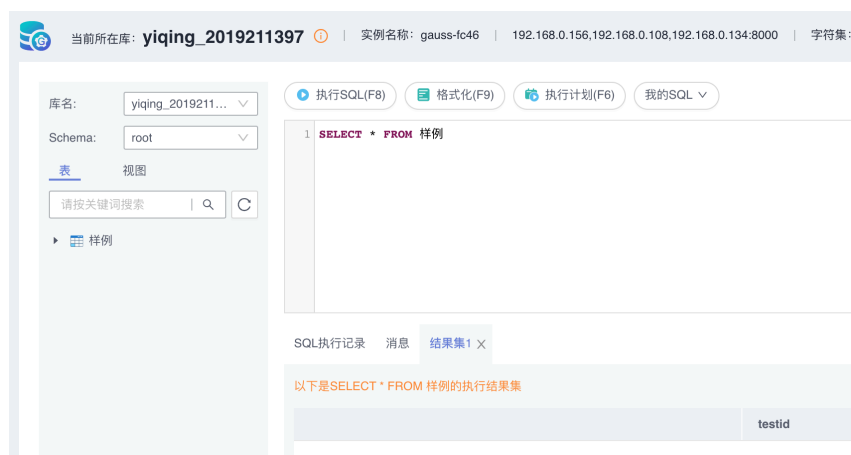


图 12

3 实验总结

本次实验使我初步认识华为云 DAS 的权限管理系统，同时巩固了课堂上所学的有关用户和权限管理的知识。

Part II

实验五

4 概述

4.1 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)。

4.2 实验平台及环境

- GaussDB(for openGauss) 8 核 | 64 GB
- GaussDB(for openGauss) 2020 主备版

4.3 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);

5 实验步骤

5.1 创建和管理索引

创建索引 进入 bupt2019211397 数据库，输入以下 SQL 语句：

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

结果如图 13。



图 13

管理索引 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，如图 13。

删除索引，输入以下 SQL 语句：

```
1 DROP INDEX 日期index;
```

索引创建练习 创建唯一索引，输入以下 SQL 语句：

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

输入以下 SQL 语句进行查询：

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

创建索引前和索引后的结果分别如图 14和图 15。

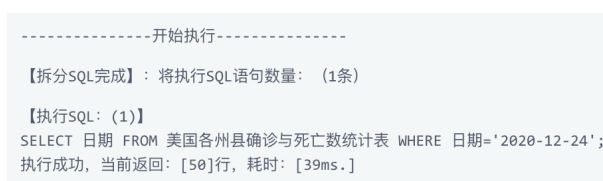


图 14

可见查询效率有提升。

创建多字段索引，输入以下 SQL 语句：

```
-----开始执行-----  
  
【拆分SQL完成】：将执行SQL语句数量：（1条）  
  
【执行SQL：（1）】  
SELECT 日期 FROM 美国各州县确诊与死亡数统计表 WHERE 日期='2020-12-24';  
执行成功,当前返回：[50]行,耗时：[19ms.]
```

图 15

-
- 1 **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）;
-

输入以下 SQL 语句进行查询:

-
- 1 **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;
-

创建索引前和索引后的结果分别如图 16和图 17。

```
-----开始执行-----  
  
【拆分SQL完成】：将执行SQL语句数量：（1条）  
  
【执行SQL：（1）】  
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;  
执行成功,当前返回：[50]行,耗时：[14ms.]
```

图 16

```
-----开始执行-----  
  
【拆分SQL完成】：将执行SQL语句数量：（1条）  
  
【执行SQL：（1）】  
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;  
执行成功,当前返回：[50]行,耗时：[8ms.]
```

图 17

可见查询效率有提升。

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，输入以下 SQL 语句：

-
- 1 **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';
-

输入以下 SQL 语句进行查询:

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

创建索引前和索引后的结果分别如图 14和图 18。

可见查询效率有提升。

创建表达式索引，输入以下 SQL 语句：


```

-----开始执行-----
【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】
SELECT 日期 FROM 美国各州县确诊与死亡数统计表 WHERE 日期='2020-12-24';
执行成功，当前返回：[50]行，耗时：[6ms.]

```

图 18

1 **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));

输入以下 SQL 语句进行查询：

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

创建索引前和索引后的结果分别如图 19和图 20。

```

-----开始执行-----
【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】
SELECT * FROM 美国各州县确诊与死亡数统计表 WHERE trunc(累计确诊)>1000;
执行成功，当前返回：[50]行，耗时：[215ms.]

```

图 19

```

-----开始执行-----
【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】
SELECT * FROM 美国各州县确诊与死亡数统计表 WHERE trunc(累计确诊)>1000;
执行成功，当前返回：[50]行，耗时：[167ms.]

```

图 20

可见查询效率有提升。

5.2 创建和管理视图

创建普通视图 bj_yq，输入以下 SQL 语句：

```

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

```

进入对象列表，单击视图，如图 21。

查询视图 bj_yq，输入以下 SQL 语句：

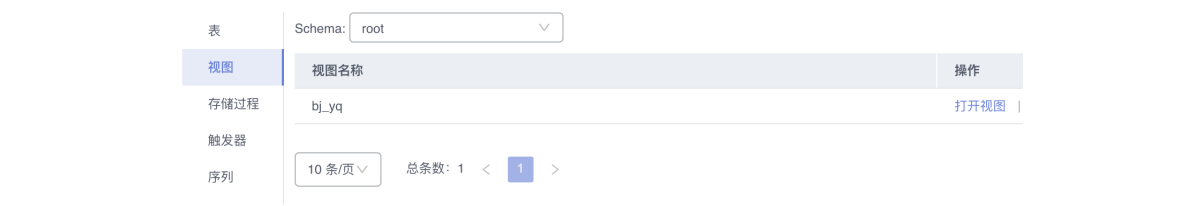


图 21

1 `SELECT * FROM bj_yq;`

结果如图 22。

SQL 执行记录 消息 结果集 1 X						
以下是SELECT * FROM bj_yq的执行结果集						
该表不可编辑。						
	行程号	病例号	性别	日期信息	行程信息	
1	41	1007	女	12月28日	作为确诊病例的密切接触者进行集中隔离医学观察	
2	42	1007	女	1月4日	核酸检测结果为阳性，由120负压救护车转运至地坛	
3	121	1020	男	12月26日	作为确诊病例的密切接触者进行集中隔离医学观察	
4	122	1020	男	1月3日	核酸检测结果为阳性，由120负压救护车转运至地坛	
5	123	1021	男	12月26日	作为确诊病例的密切接触者进行集中隔离医学观察	
6	124	1021	男	1月3日	核酸检测结果为阳性，由120负压救护车转运至地坛	
7	125	1022	女	12月28日	作为确诊病例的密切接触者进行集中隔离医学观察	
8	126	1022	女	1月3日	核酸检测结果为阳性，由120负压救护车转运至地坛	

图 22

进入对象列表，单击视图，单击查看视图详情，如图 23。



图 23

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：

```
1 SELECT 病例号, 行程号, 性别, 日期信息
2 FROM bj_yq
3 WHERE 行程信息 LIKE '%普通型%'
4 ORDER BY 病例号;
```

结果如图 24。

以下是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 病例号的执行...  该表不可编辑。

	病例号	行程号	性别	日期信息
1	25	1788	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	2896	男	1月19日

图 24

6 实验总结

本次实验使我对 SQL 的索引和视图有关语法更加熟悉，并且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。

Part III

实验六

7 概述

7.1 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、触发器等操作,让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss)。

7.2 实验平台及环境

- GaussDB(for openGauss) 8 核 | 64 GB
- GaussDB(for openGauss) 2020 主备版

7.3 实验内容

1. 本实验通过存储过程管理、触发器管理等操作，让学生熟悉并了解 DAS 环境下如何使用 GaussDB(for openGauss);

8 实验步骤

8.1 创建存储过程

在全国各省累计数据表中增加一条记录。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。

创建存储过程 insertRecord，内容如图 25。

```
1 CREATE OR REPLACE PROCEDURE root."insertRecord"("日期" date, "省" character varying, "累计确诊" integer, "累计治愈" integer, "累计死亡" integer)
2 AS DECLARE
3 BEGIN
4 INSERT INTO 全国各省累计数据表 VALUES (日期, 省, 累计确诊, 累计治愈, 累计死亡);
5 END
6 ;
```

图 25

执行存储过程，设置参数如图 26。

请设置存储过程的入口参数值

	参数	数据类型	参数模式	传递null值	传递默认值	参数值
1	日期	date	IN	<input type="checkbox"/>	<input type="checkbox"/>	2021-10-08
2	省	varchar	IN	<input type="checkbox"/>	<input type="checkbox"/>	吉林省
3	累计确诊	int4	IN	<input type="checkbox"/>	<input type="checkbox"/>	578
4	累计治愈	int4	IN	<input type="checkbox"/>	<input type="checkbox"/>	571
5	累计死亡	int4	IN	<input type="checkbox"/>	<input type="checkbox"/>	3

确定 取消

图 26

执行结果如图 27。

【执行SQL】
SELECT "root"."insertRecord"('2021-10-08','吉林省',578,571,3);
执行存储过程成功,用时14ms
执行结果请查看结果集标签页

图 27

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。

创建存储过程 queryRecord，内容如图 28。

```
1 CREATE OR REPLACE PROCEDURE root."queryRecord"("指定日期" character varying, "指定州" character varying, OUT "累计确诊总数" integer, OUT "累计死亡总数" integer)
2 AS DECLARE
3 BEGIN
4 SELECT SUM(累计确诊), SUM(累计死亡) 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 INTO 累计确诊总数, 累计死亡总数;
5 END
6 ;
```

图 28

执行存储过程，设置参数如图 29。

请设置存储过程的入口参数值

	参数	数据类型	参数模式	传递null值	传递默认值	参数值
1	指定日期	varchar	IN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text" value="'2021-01-01'"/>
2	指定州	varchar	IN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text" value="'California'"/>

确定

取消

图 29

执行结果如图 30。

消息结果集1

	queryRecord
1	(2365024,26363)

图 30

8.2 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 insertRecord 存储过程中的操作，单击查看存储过程详情，如图 31。

查看存储过程详情

```
1 CREATE OR REPLACE PROCEDURE root."insertRecord"("日期" date, "省" character varying, "治愈" integer, "死亡" integer)
2 AS DECLARE
3 BEGIN
4 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);
5 END
6 ;
7 /
8
```

关闭

图 31

删除存储过程，输入以下 SQL 语句：

```
1 drop procedure "insertRecord";
```

结果如图 32。

```
-----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：[6ms.]
```

图 32

8.3 创建触发器

创建 INSERT 触发器，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。

创建函数 check_on_insert，如图 33。

```
1 CREATE OR REPLACE FUNCTION root.check_on_insert()  
2 RETURNS trigger  
3 LANGUAGE plpgsql  
4 NOT FENCED NOT SHIPPABLE  
5 AS $function$  
6 BEGIN  
7     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  
8         RAISE EXCEPTION '未通过数据一致性验证';  
9     END IF;  
10    RETURN NEW;  
11 END  
12 $function$;
```

图 33

创建触发器，输入以下 SQL 语句：

```
1 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();
```

输入以下 SQL 语句测试：

```
1 INSERT INTO 美国各州县确诊与死亡数统计表 VALUES ('2021-10-19', 'US', 'AAA',  
  ↳ 'BBB', 123, 1);
```

结果如图 34。

```
-----开始执行-----  
  
【拆分SQL完成】：将执行SQL语句数量：（1条）  
  
【执行SQL：（1）】  
INSERT INTO 美国各州县确诊与死亡数统计表 VALUES ('2021-10-19', 'US', 'AAA', 'BBB', 123, 1);  
执行失败，失败原因：ERROR: 未通过数据一致性验证
```

图 34

创建 DELETE 触发器，当从病例基本信息表中删除一条记录时，该病例 ID 对应的行程信息记录也进行删除操作。

创建函数 on_delete，如图 35。

```
1 CREATE OR REPLACE FUNCTION root.on_delete()  
2 RETURNS trigger  
3 LANGUAGE plpgsql  
4 NOT FENCED NOT SHIPPABLE  
5 AS $function$  
6 BEGIN  
7     DELETE FROM 病例行程信息表1  
8     WHERE 病例号 = OLD.病例号;  
9     RETURN OLD;  
10 END  
11 $function$;
```

图 35

创建触发器，输入以下 SQL 语句：

```
1 CREATE TRIGGER trigger_delete AFTER DELETE ON 病例基本信息表1 FOR EACH ROW  
  → EXECUTE PROCEDURE on_delete();
```

查看病例基本信息表和病例行程信息表中病例号为 1 的患者的信息，如图 36和图 37。

以下是SELECT * FROM 病例基本信息表1 WHERE 病例号 = 1的执行结果集

该表不可编辑。

病例号	省	市	区	日期	性别	年龄	患者信息
1	河北省	石家庄市	藁城区	2021-01-20	女	54	确诊病例1:

图 36

以下是SELECT * FROM 病例行程信息表1 WHERE 病例号 = 1的执行结果集

该表不可编辑。

行程号	病例号	日期信息	行程信息
1	1	2021年1月1日至10日	除1月3日12时步行到南桥寨惠兰粮油店买菜外，均居家无外出
2	1	1月11日	转运至藁城区指定隔离点进行集中隔离医学观察，期间12日、
3	1	1月18日	转运至石家庄市第三医院发热门诊就诊，核酸检测呈阳性，当
4	1	1月19日	诊断为确诊病例。

图 37

输入以下 SQL 语句测试：

```
1 DELETE FROM 病例基本信息表1 WHERE 病例号 = 1;
```

结果如图 38。

再次查看病例基本信息表和病例行程信息表中病例号为 1 的患者的信息，如图 39和图 40。发现两个表中该患者的信息都已经删除。

创建 UPDATE 触发器，禁止修改全国各省累计数据统计表中的累计确诊、累计治愈和累计死亡数据。

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】

```
DELETE FROM 病例基本信息表1
WHERE 病例号 = 1;
```

执行成功，耗时：[8ms.]

图 38

以下是SELECT * FROM 病例基本信息表1 WHERE 病例号 = 1的执行结果集

该表不可编辑。

复制行 复制列 列设置

病例号	省	市	区	日期	性别	年龄	患者信息
-----	---	---	---	----	----	----	------

图 39

以下是SELECT * FROM 病例行程信息表1 WHERE 病例号 = 1的执行结果集

该表不可编辑。

复制行 复制列 列设置

行程号	病例号	日期信息	行程信息
-----	-----	------	------

图 40

```
1 CREATE OR REPLACE FUNCTION root.check_on_update()
2 RETURNS trigger
3 LANGUAGE plpgsql
4 NOT FENCED NOT SHIPPABLE
5 AS $function$
6 BEGIN
7     IF OLD.累计确诊 != NEW.累计确诊 OR OLD.累计治愈 != NEW.累计治愈 OR OLD.累计死亡 != NEW.累计死亡 THEN
8         RAISE EXCEPTION '禁止修改数据';
9     END IF;
10    RETURN NEW;
11 END
12 $function$;
```

图 41

创建函数 check_on_update，如图 41。

创建触发器，输入以下 SQL 语句：

```
1 CREATE TRIGGER trigger_update BEFORE UPDATE ON 全国各省累计数据统计表 FOR EACH
↪ ROW EXECUTE PROCEDURE check_on_update();
```

输入以下 SQL 语句测试：

```
1 UPDATE 全国各省累计数据统计表
2 SET 累计确诊 = 1
3 WHERE 日期 = '2020-12-31';
```

结果如图 42。

8.4 管理触发器

删除 INSERT 触发器、DELETE 触发器、UPDATE 触发器，输入以下 SQL 语句：


```

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】
UPDATE 全国各省累计数据统计表
SET 累计确诊 = 1
WHERE 日期 = '2020-12-31';
执行失败，失败原因：ERROR：禁止修改数据

```

图 42

```

1 DROP TRIGGER trigger_insert ON 美国各州县确诊与死亡数统计表;
2 DROP TRIGGER trigger_delete ON 病例基本信息表1;
3 DROP TRIGGER trigger_update ON 全国各省累计数据统计表;

```

结果如图 43。

```

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（3条）

【执行SQL：（1）】
DROP TRIGGER trigger_insert ON 美国各州县确诊与死亡数统计表;
执行成功，耗时：[5ms.]

【执行SQL：（2）】
DROP TRIGGER trigger_delete ON 病例基本信息表1;
执行成功，耗时：[5ms.]

【执行SQL：（3）】
DROP TRIGGER trigger_update ON 全国各省累计数据统计表;
执行成功，耗时：[6ms.]

```

图 43

8.5 事务级全局临时表

创建临时表 t_test2，输入以下 SQL 语句：

```

1 CREATE GLOBAL TEMPORARY TABLE t_test2(
2     id integer,
3     lbl text
4 ) ON COMMIT DELETE ROWS;

```

结果如图 44。

首先，用 begin 开始一个事务，其次，向表中插入数据，最后，对表进行查询。可以查出相应数据，输入以下 SQL 语句：

```

1 BEGIN;
2 INSERT INTO t_test2 VALUES(1,'data1');
3 SELECT * FROM t_test2;

```

```
-----开始执行-----  
  
【拆分SQL完成】： 将执行SQL语句数量： （1条）  
  
【执行SQL： (1)】  
CREATE GLOBAL TEMPORARY TABLE t_test2(  
id integer,  
lbl text  
) ON COMMIT DELETE ROWS;  
执行成功，耗时： [8ms.]
```

图 44

以下是select * from t_test2;的执行结果集 ⓘ 该表不可编辑。

	id	lbl
1	1	data1

图 45

结果如图 45。

先用 **commit** 提交来结束事务，此时再对表进行查询，可以发现已经查询不出数据了，输入以下 SQL 语句：

```
1 COMMIT;  
2 SELECT * FROM t_test2;
```

结果如图 46。

以下是select * from t_test2;的执行结果集 ⓘ 该表不可编辑。

	id	lbl

图 46

删除临时表，输入以下 SQL 语句：

```
1 drop table t_test2;
```

结果如图 47。

```
-----开始执行-----  
  
【拆分SQL完成】： 将执行SQL语句数量： （1条）  
  
【执行SQL： (1)】  
drop table t_test2;  
执行成功，耗时： [7ms.]
```

图 47

9 实验总结

实验中，我参考 PostgreSQL 的语法说明和教材上的内容编写存储过程和触发器的 SQL 语句，使我对 SQL 语法的熟悉程度大大增加，同时增强了我的英文文献阅读能力。

本次实验使我对触发器的理解更加深刻，同时巩固了课堂上所学的有关触发器的知识。

Part IV

实验七

10 概述

10.1 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。

10.2 实验平台及环境

- GaussDB(for openGauss) 8 核 | 64 GB
- GaussDB(for openGauss) 2020 主备版
- Windows 10 Pro 21H1
- MinGW-w64 8.1.0

10.3 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。

11 实验步骤

11.1 配置数据源

下载 GaussDB-Kernel-V500R001C10-Windows-Odbc.tar.gz 并解压,点击 psqlodbc_x64.msi 进行安装。

进入 C:\Windows\System32\odbcad32.exe, 选择用户 DSN-> 添加->PostgreSQL Unicode, 输入用户名和密码, 点击测试, 如图 48所示。测试成功后保存并退出。

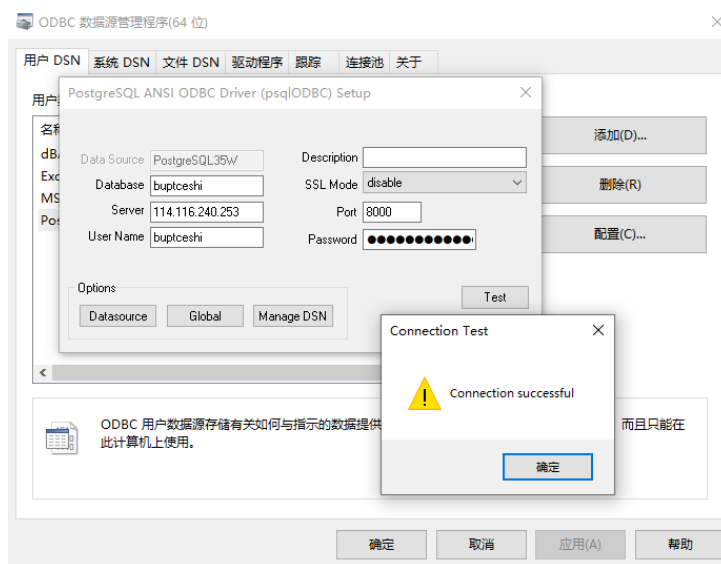


图 48: 配置数据源

11.2 编写程序

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <wchar.h>
4  #ifdef WIN32
5  #include <windows.h>
6  #endif
7  #include <sqlext.h>
8  #include <locale.h>
9
10 SQLHENV V_OD_Env; // 环境句柄
11 SQLHSTMT V_OD_hstmt; // 句柄
12 SQLHDBC V_OD_hdbc; // 连接属性
13 SQLINTEGER V_OD_erg, V_OD_err; // 存放返回值
14 SQL_DATE_STRUCT date;
15 SQLWCHAR prov[50];
16 SQLINTEGER defi, cure, death;
17 SQLLEN cb_date, cb_prov, cb_defi, cb_cure, cb_death;
18 SQLLEN res_cnt;
19 char query[200]; // 存放查询语句
20
```

```

21 int main(int argc, char *argv)
22 {
23     // 区域设置, 用于处理中文
24     setlocale(LC_ALL, "");
25     // 申请环境句柄
26     V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &V_OD_Env);
27     if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
28     {
29         printf("Error AllocHandle\n");
30         exit(0);
31     }
32     // 设置环境属性
33     SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void *)SQL_OV_ODBC3, 0);
34     // 申请连接句柄
35     V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
36     if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
37     {
38         SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
39         exit(0);
40     }
41     // 设置连接属性
42     SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,
    → (SQLPOINTER)SQL_AUTOCOMMIT_ON, 0);
43     // 连接数据源
44     V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR *)"PostgreSQL35W", SQL_NTS,
    → (SQLCHAR *)"buptceshi", SQL_NTS, (SQLCHAR *)"bupt20211201@", SQL_NTS);
45     if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
46     {
47         printf("Error SQLConnect %d\n", V_OD_erg);
48         SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
49         exit(0);
50     }
51     printf("Connected!\n");
52     // 设置语句属性
53     SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT, (SQLPOINTER *)3, 0);
54     // 申请语句句柄
55     SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
56
57     // 将长字符的字符串转为多字节的字符串
58     wcstombs_s(NULL, query, 200, L"SELECT * FROM root. 全国各省累计数据统计表
    → WHERE 日期 = '2020-12-8'", _TRUNCATE);
59     // 执行查询
60     SQLExecDirect(V_OD_hstmt, query, SQL_NTS);
61     // 绑定结果集
62     SQLBindCol(V_OD_hstmt, 1, SQL_C_TYPE_DATE, (SQLPOINTER)&date, 50,
    → &cb_date);
63     SQLBindCol(V_OD_hstmt, 2, SQL_C_WCHAR, (SQLPOINTER)prov, 50, &cb_prov);
64     SQLBindCol(V_OD_hstmt, 3, SQL_C_SLONG, (SQLPOINTER)&defi, 50, &cb_defi);
65     SQLBindCol(V_OD_hstmt, 4, SQL_C_SLONG, (SQLPOINTER)&cure, 50, &cb_cure);
66     SQLBindCol(V_OD_hstmt, 5, SQL_C_SLONG, (SQLPOINTER)&death, 50, &cb_death);
67     // 取一条数据

```

```

68     V_OD_erg = SQLFetch(V_OD_hstmt);
69     while (V_OD_erg != SQL_NO_DATA)
70     {
71         printf("%S: %d-%d-%d %S: %S %S: %d %S: %d %S: %d\n", L" 日期",
↪ date.year, date.month, date.day, L" 省", prov, L" 累计确诊", defi, L" 累计
↪ 治愈", cure, L" 累计死亡", death);
72         // 取下一条数据
73         V_OD_erg = SQLFetch(V_OD_hstmt);
74     }
75     printf("Query Done!\n\n");
76     // 释放游标
77     SQLCloseCursor(V_OD_hstmt);
78
79     wcstombs_s(NULL, query, 200, L"INSERT INTO root. 全国各省累计数据统计表
↪ values(?,?,?,?), _TRUNCATE);
80     // 准备执行
81     SQLPrepare(V_OD_hstmt, query, SQL_NTS);
82     // 准备参数
83     date.year = 2021;
84     date.month = 10;
85     date.day = 8;
86     wcscpy(prov, L" 吉林省");
87     defi = 578;
88     cure = 571;
89     death = 3;
90     // 绑定参数
91     SQLBindParameter(V_OD_hstmt, 1, SQL_PARAM_INPUT, SQL_C_TYPE_DATE,
↪ SQL_TYPE_DATE, sizeof(date), 0, (SQLPOINTER)&date, 0, &cb_date);
92     SQLBindParameter(V_OD_hstmt, 2, SQL_PARAM_INPUT, SQL_C_WCHAR, SQL_WCHAR,
↪ 50, 0, (SQLPOINTER)prov, 0, &cb_prov);
93     SQLBindParameter(V_OD_hstmt, 3, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
↪ 0, 0, (SQLPOINTER)&defi, 0, &cb_defi);
94     SQLBindParameter(V_OD_hstmt, 4, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
↪ 0, 0, (SQLPOINTER)&cure, 0, &cb_cure);
95     SQLBindParameter(V_OD_hstmt, 5, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
↪ 0, 0, (SQLPOINTER)&death, 0, &cb_death);
96     // 执行
97     SQLExecute(V_OD_hstmt);
98     // 提取行计数
99     SQLRowCount(V_OD_hstmt, &res_cnt);
100    printf("Inserted %d row.\n\n", res_cnt);
101
102    wcstombs_s(NULL, query, 200, L"SELECT * FROM root. 全国各省累计数据统计表
↪ WHERE 日期 = '2021-10-8'", _TRUNCATE);
103    SQLExecDirect(V_OD_hstmt, query, SQL_NTS);
104    V_OD_erg = SQLFetch(V_OD_hstmt);
105    while (V_OD_erg != SQL_NO_DATA)
106    {
107        // 通过 SQLGetData 取数据
108        SQLGetData(V_OD_hstmt, 1, SQL_C_TYPE_DATE, (SQLPOINTER)&date, 50,
↪ &cb_date);

```

```

109         SQLGetData(V_OD_hstmt, 2, SQL_C_WCHAR, (SQLPOINTER)prov, 50,
↪ &cb_prov);
110         SQLGetData(V_OD_hstmt, 3, SQL_C_SLONG, (SQLPOINTER)&defi, 50,
↪ &cb_defi);
111         SQLGetData(V_OD_hstmt, 4, SQL_C_SLONG, (SQLPOINTER)&cure, 50,
↪ &cb_cure);
112         SQLGetData(V_OD_hstmt, 5, SQL_C_SLONG, (SQLPOINTER)&death, 50,
↪ &cb_death);
113         printf("%S: %d-%d-%d %S: %S %S: %d %S: %d %S: %d\n", L" 日期",
↪ date.year, date.month, date.day, L" 省", prov, L" 累计确诊", defi, L" 累计
↪ 治愈", cure, L" 累计死亡", death);
114         V_OD_erg = SQLFetch(V_OD_hstmt);
115     }
116     printf("Query Done!\n\n");
117     SQLCloseCursor(V_OD_hstmt);
118
119     wcstombs_s(NULL, query, 200, L"UPDATE root. 全国各省累计数据统计表 SET 累计
↪ 确诊 = 0 WHERE 日期 = '2021-10-8'", _TRUNCATE);
120     SQLExecDirect(V_OD_hstmt, query, SQL_NTS);
121     SQLRowCount(V_OD_hstmt, &res_cnt);
122     printf("Updated %d row.\n\n", res_cnt);
123
124     wcstombs_s(NULL, query, 200, L"SELECT * FROM root. 全国各省累计数据统计表
↪ WHERE 日期 = '2021-10-8'", _TRUNCATE);
125     SQLExecDirect(V_OD_hstmt, query, SQL_NTS);
126     V_OD_erg = SQLFetch(V_OD_hstmt);
127     while (V_OD_erg != SQL_NO_DATA)
128     {
129         SQLGetData(V_OD_hstmt, 1, SQL_C_TYPE_DATE, (SQLPOINTER)&date, 50,
↪ &cb_date);
130         SQLGetData(V_OD_hstmt, 2, SQL_C_WCHAR, (SQLPOINTER)prov, 50,
↪ &cb_prov);
131         SQLGetData(V_OD_hstmt, 3, SQL_C_SLONG, (SQLPOINTER)&defi, 50,
↪ &cb_defi);
132         SQLGetData(V_OD_hstmt, 4, SQL_C_SLONG, (SQLPOINTER)&cure, 50,
↪ &cb_cure);
133         SQLGetData(V_OD_hstmt, 5, SQL_C_SLONG, (SQLPOINTER)&death, 50,
↪ &cb_death);
134         printf("%S: %d-%d-%d %S: %S %S: %d %S: %d %S: %d\n", L" 日期",
↪ date.year, date.month, date.day, L" 省", prov, L" 累计确诊", defi, L" 累计
↪ 治愈", cure, L" 累计死亡", death);
135         V_OD_erg = SQLFetch(V_OD_hstmt);
136     }
137     printf("Query Done!\n\n");
138     SQLCloseCursor(V_OD_hstmt);
139
140     wcstombs_s(NULL, query, 200, L"DELETE FROM root. 全国各省累计数据统计表
↪ WHERE 日期 = '2021-10-8'", _TRUNCATE);
141     SQLExecDirect(V_OD_hstmt, query, SQL_NTS);
142     SQLRowCount(V_OD_hstmt, &res_cnt);
143     printf("Deleted %d row.\n\n", res_cnt);

```

```

144
145     wcstombs_s(NULL, query, 200, L"SELECT * FROM root. 全国各省累计数据统计表
↪ WHERE 日期 = '2021-10-8'", _TRUNCATE);
146     SQLExecDirect(V_OD_hstmt, query, SQL_NTS);
147     V_OD_erg = SQLFetch(V_OD_hstmt);
148     while (V_OD_erg != SQL_NO_DATA)
149     {
150         SQLGetData(V_OD_hstmt, 1, SQL_C_TYPE_DATE, (SQLPOINTER)&date, 50,
↪ &cb_date);
151         SQLGetData(V_OD_hstmt, 2, SQL_C_WCHAR, (SQLPOINTER)prov, 50,
↪ &cb_prov);
152         SQLGetData(V_OD_hstmt, 3, SQL_C_SLONG, (SQLPOINTER)&defi, 50,
↪ &cb_defi);
153         SQLGetData(V_OD_hstmt, 4, SQL_C_SLONG, (SQLPOINTER)&cure, 50,
↪ &cb_cure);
154         SQLGetData(V_OD_hstmt, 5, SQL_C_SLONG, (SQLPOINTER)&death, 50,
↪ &cb_death);
155         printf("%S: %d-%d-%d %S: %S %S: %d %S: %d %S: %d\n", L" 日期",
↪ date.year, date.month, date.day, L" 省", prov, L" 累计确诊", defi, L" 累计
↪ 治愈", cure, L" 累计死亡", death);
156         V_OD_erg = SQLFetch(V_OD_hstmt);
157     }
158     printf("Query Done!\n\n");
159     SQLCloseCursor(V_OD_hstmt);
160
161     // 断开数据源连接并释放句柄
162     SQLFreeHandle(SQL_HANDLE_STMT, V_OD_hstmt);
163     SQLDisconnect(V_OD_hdbc);
164     SQLFreeHandle(SQL_HANDLE_DBC, V_OD_hdbc);
165     SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
166     return 0;
167 }

```

程序首先查询国内 2021 年 12 月 8 日的所有确诊信息，之后向表中插入一条 2021 年 10 月 8 日吉林省的数据，紧接着查询该日的所有数据，之后将该行数据中的确诊数改为 0，再查询一次，最后删除该行数据，再查询一次。

注：SQLAllocEnv、SQLAllocConnect 等函数在 ODBC 3.0 版本中已弃用，用相同功能的函数替代。

11.3 编译

```
gcc -o A A.c -lodbc32
```

11.4 运行

运行截图如图 49。


```

Connected!
日期: 2020-12-8 省: 天津市 累计确诊: 301 累计治愈: 292 累计死亡: 3
日期: 2020-12-8 省: 安徽省 累计确诊: 992 累计治愈: 986 累计死亡: 6
日期: 2020-12-8 省: 山东省 累计确诊: 857 累计治愈: 841 累计死亡: 7
日期: 2020-12-8 省: 山西省 累计确诊: 222 累计治愈: 218 累计死亡: 0
日期: 2020-12-8 省: 宁夏回族自治区 累计确诊: 75 累计治愈: 75 累计死亡: 0
日期: 2020-12-8 省: 广西壮族自治区 累计确诊: 264 累计治愈: 261 累计死亡: 2
日期: 2020-12-8 省: 湖南省 累计确诊: 1020 累计治愈: 1016 累计死亡: 4
日期: 2020-12-8 省: 河南省 累计确诊: 1295 累计治愈: 1266 累计死亡: 22
日期: 2020-12-8 省: 海南省 累计确诊: 171 累计治愈: 165 累计死亡: 6
日期: 2020-12-8 省: 浙江省 累计确诊: 1295 累计治愈: 1288 累计死亡: 1
日期: 2020-12-8 省: 湖北省 累计确诊: 68149 累计治愈: 63635 累计死亡: 4512
日期: 2020-12-8 省: 江西省 累计确诊: 935 累计治愈: 934 累计死亡: 1
日期: 2020-12-8 省: 江苏省 累计确诊: 681 累计治愈: 675 累计死亡: 0
日期: 2020-12-8 省: 新疆维吾尔自治区 累计确诊: 980 累计治愈: 977 累计死亡: 3
日期: 2020-12-8 省: 四川省 累计确诊: 822 累计治愈: 783 累计死亡: 3
日期: 2020-12-8 省: 广东省 累计确诊: 2009 累计治愈: 1966 累计死亡: 8
日期: 2020-12-8 省: 河北省 累计确诊: 373 累计治愈: 367 累计死亡: 6
日期: 2020-12-8 省: 吉林省 累计确诊: 157 累计治愈: 155 累计死亡: 2
日期: 2020-12-8 省: 北京市 累计确诊: 952 累计治愈: 939 累计死亡: 9
日期: 2020-12-8 省: 台湾省 累计确诊: 1 累计治愈: 1 累计死亡: 0
日期: 2020-12-8 省: 西藏自治区 累计确诊: 1 累计治愈: 1 累计死亡: 0
日期: 2020-12-8 省: 贵州省 累计确诊: 147 累计治愈: 145 累计死亡: 2
日期: 2020-12-8 省: 黑龙江省 累计确诊: 949 累计治愈: 936 累计死亡: 13
日期: 2020-12-8 省: 青海省 累计确诊: 18 累计治愈: 18 累计死亡: 0
日期: 2020-12-8 省: 陕西省 累计确诊: 502 累计治愈: 480 累计死亡: 3
日期: 2020-12-8 省: 重庆市 累计确诊: 590 累计治愈: 583 累计死亡: 6
日期: 2020-12-8 省: 香港特别行政区 累计确诊: 7075 累计治愈: 5696 累计死亡: 112
日期: 2020-12-8 省: 福建省 累计确诊: 500 累计治愈: 454 累计死亡: 1
日期: 2020-12-8 省: 甘肃省 累计确诊: 182 累计治愈: 180 累计死亡: 2
日期: 2020-12-8 省: 澳门特别行政区 累计确诊: 46 累计治愈: 46 累计死亡: 0
日期: 2020-12-8 省: 上海市 累计确诊: 1376 累计治愈: 1300 累计死亡: 7
日期: 2020-12-8 省: 云南省 累计确诊: 221 累计治愈: 210 累计死亡: 2
日期: 2020-12-8 省: 内蒙古自治区 累计确诊: 336 累计治愈: 308 累计死亡: 1
日期: 2020-12-8 省: 辽宁省 累计确诊: 289 累计治愈: 287 累计死亡: 2
Query Done!

Inserted 1 row.

日期: 2021-10-8 省: 吉林省 累计确诊: 578 累计治愈: 571 累计死亡: 3
Query Done!

Updated 1 row.

日期: 2021-10-8 省: 吉林省 累计确诊: 0 累计治愈: 571 累计死亡: 3
Query Done!

Deleted 1 row.
Query Done!

```

图 49: 运行截图

12 实验总结

在配置数据源环节，按照华为云的文档完成，没有产生问题，但是测试华为云的样例时，发现没有建表权限，并且无论如何替换表名都查询不到表，通过 `SQLGetDiagField` 诊断函数得到报错 42501 和 42P01。

之后尝试使用 `SELECT * FROM pg_catalog.pg_tables` 语句查看所有表的属性，发现打印出乱码，于是加上了长字符 (`wchar_t`) 的字符串和地区设置，正确打印了结果。

发现长字符的查询语句无法直接执行，于是使用 `wcstombs_s` 函数转为多字节的字符串再执行，终于查询出了结果。

在查询完成后，下一条查询报 HY010 错误，查看文档发现需要通过 `SQLCloseCursor` 释放游标。

12.1 数据库驱动的概念

数据库驱动是应用程序和数据库存储之间的一种接口，如图 50 所示，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。

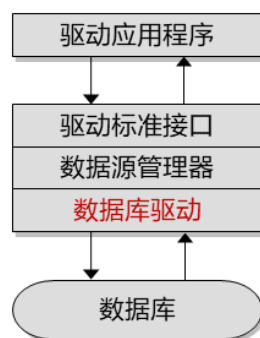


图 50: 数据库驱动

12.2 使用 ODBC 开发的流程

使用 ODBC 开发的流程如图 51 所示。

1. 任何应用程序中的第一步是连接到数据源。连接到数据源的第一步是加载驱动程序管理器，然后使用 `SQLAllocHandle` 分配环境句柄。然后，应用程序使用 `SQL_ATTR_APP_ODBC_VER` 属性调用 `SQLSetEnvAttr` 来注册它所遵循的 ODBC 版本。接下来，应用程序使用 `SQLAllocHandle` 分配连接句柄，然后使用 `SQLConnect`、`SQLDriverConnect` 或 `SQLBrowseConnect` 连接到数据源。然后，应用程序设置任何连接属性，例如是否手动提交事务。
2. 第二步是初始化应用程序。此时，通常使用 `SQLGetInfo` 来发现驱动程序的功能。所有应用程序都需要使用 `SQLAllocHandle` 来分配语句句柄，许多应用程序使用 `SQLSetStmtAttr` 设置语句属性（如游标类型）。
3. 第三步是生成并执行 SQL 语句。用于执行此步骤的方法可能会有很大差异。应用程序可能会提示用户输入 SQL 语句，根据用户输入生成 SQL 语句，或者使用硬编码的 SQL 语句。如果 SQL 语句包含参数，则应用程序会通过每个参数调用 `SQLBindParameter`，将这些参数绑定到应用程序变量中。生成 SQL 语句并绑定任何参数后，将通过 `SQLExecDirect` 执行语句。如果语句将多次执行，则可以通过 `SQLPrepare` 准备，并通过 `SQLExecute` 执行。应用程序还可以放弃地执行 SQL 语句，而调用函数返回包含目录信息的结果集，如可用的列或表。
4.
 - 如果在步骤 3 中执行的语句为 `SELECT` 语句或目录函数，应用程序将首先调用 `SQLNumResultCols` 以确定结果集中的列数。如果应用程序已知道结果集列的数目，则不需要执行此步骤，例如，当在垂直或自定义应用程序中对 SQL 语句进行硬编码时。接下来，应用程序通过 `SQLDescribeCol` 检索每个结果集列的名称、数据类型、精度和小数位数。同样，对于已知道此信息的应用程序（如垂直和自定义应用程序），这并不是必需的。应用程序

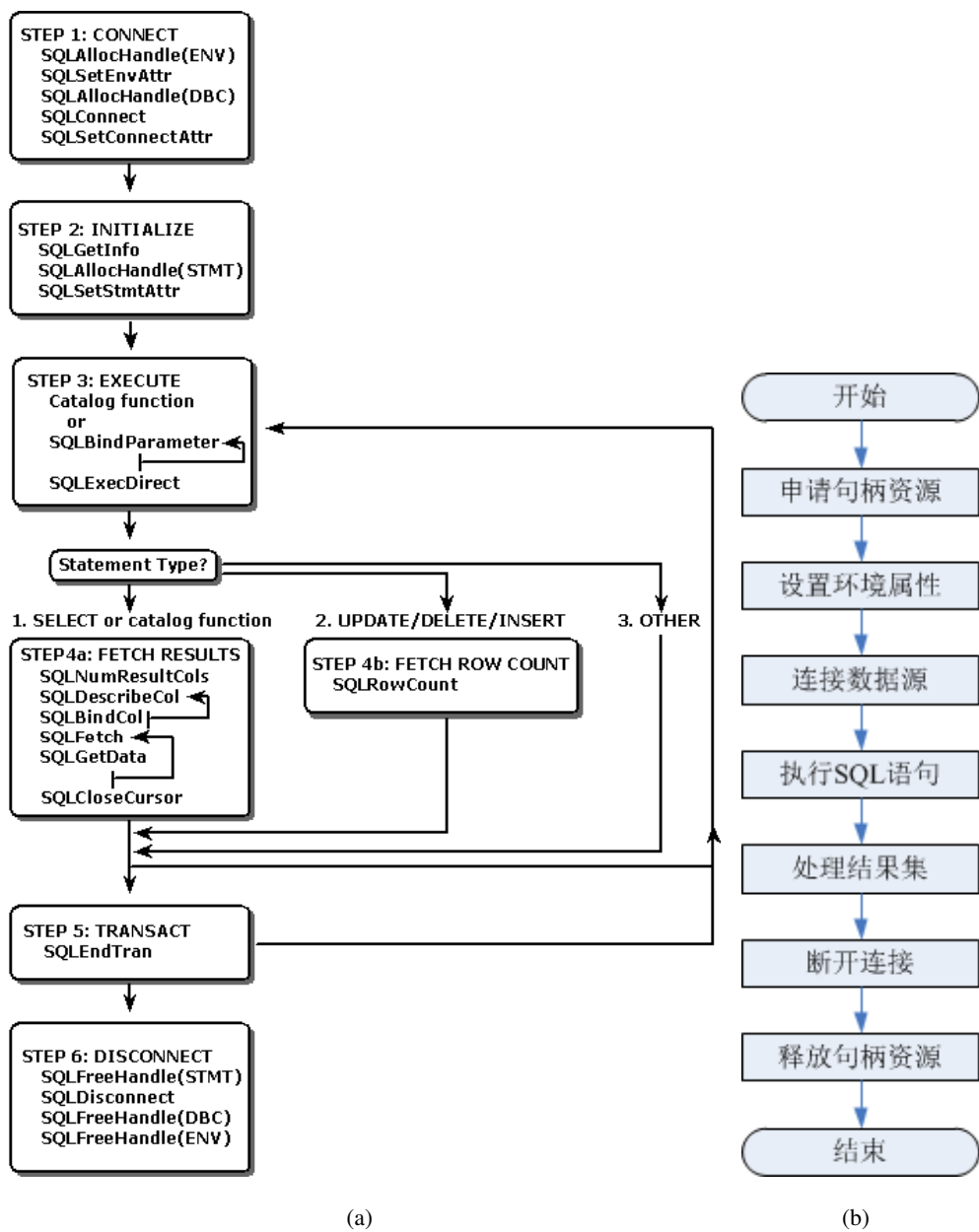


图 51: 使用 ODBC 开发的流程

将此信息传递给 `SQLBindCol`，这会将应用程序变量绑定到结果集中的列。现在，应用程序会调用 `SQLFetch` 来检索第一行数据，并将该行中的数据置于与 `SQLBindCol` 绑定的变量中。如果行中有任何长数据，则它会调用 `SQLGetData` 来检索该数据。应用程序继续调用 `SQLFetch` 和 `SQLGetData` 以检索其他数据。完成数据提取后，它将调用 `SQLCloseCursor` 以关闭游标。现在，应用程序返回到步骤 3 来执行同一事务中的另一个语句；或转到步骤 5 以提交或回滚事务。

- 如果步骤 3 中执行的语句是 `UPDATE`、`DELETE` 或 `INSERT` 语句，则应用程序使用 `SQLRowCount` 检索受影响的行的计数。应用程序现在返回到步骤 3，以在同一事务中执行另一个语句，或继续执行步骤 5 以提交或回滚事务。
5. 第五步是调用 `SQLEndTran` 来提交或回滚事务。仅当应用程序将事务提交模式设置为手动提交时，应用程序才会执行此步骤；如果事务提交模式为自动提交（这是默认值），则执行语句时，将自动提交事务。若要在新事务中执行语句，应用程序会返回到步骤 3。若要断开与数据源的连接，应用程序将继续执行步骤 6。
 6. 最后一步是断开与数据源的连接。首先，应用程序通过调用 `SQLFreeHandle` 释放所有语句句柄。接下来，应用程序与 `SQLDisconnect` 断开与数据源的连接，并通过 `SQLFreeHandle` 释放连接句柄。最后，应用程序通过 `SQLFreeHandle` 释放环境句柄，并卸载驱动程序管理器。