福州大学

《数据库应用实践》课程报告

学号：102102145

姓名：胡嘉鑫

年级：大三

学院：计算机与大数据学院

专业：数据科学与大数据技术

本组其它成员：学号_____姓名_____

学号_____姓名_____

实验时间：2023－2024 学年第一学期

任课教师：程烨

《数据库应用实践》实验二：数据库管理系统的维护与管理

胡嘉鑫　　102102145

2023 年 11 月 25 日

目录

1 实验目的

掌握 DBMS 提供的数据库用户和权限管理机制；理解存储过程概念，掌握存储过程与触发器的使用；掌握数据库备份与恢复方法。

2 实验预备内容

1. 阅读教材《数据库系统概论》相关章节。
2. 阅读实验使用的数据库管理系统的相关帮助文档。

3 实验环境

- OS: Linux
- DBMS: OpenGauss DataBase

4 实验内容

4.1 数据库安全性

1. 数据库账户的添加、删除

```
-- 账户添加
CREATE USER JIM PASSWORD 'Bigdata@123';

-- 账户查看
SELECT * FROM pg_user;

-- 账户删除
DROP USER jim CASCADE;
```

图 1: 数据库账户的添加, 删除

2. 对账户进行授予权限、收回权限。

```sql
CREATE USER joe PASSWORD 'Bigdata@123';

-- 授予权限
 GRANT ALL PRIVILEGES TO joe;

-- 收回权限
REVOKE ALL PRIVILEGES FROM joe;
```

图 2: 对账户进行授予权限、收回权限

4.2 触发器，存储过程的使用

1. 创建存储过程并执行.

```sql
CREATE TABLE t_test(c1 INT, c2 INT);

-- 创建存储过程
CREATE OR REPLACE procedure insert_data
IS
a INT;
b INT;
BEGIN
a=1;
b=2;
```

```sql
INSERT INTO t_test VALUES(a,b);
INSERT INTO t_test VALUES(b,a);
END;
/

-- 执行存储过程
CALL insert_data();
SELECT * FROM t_test;
```

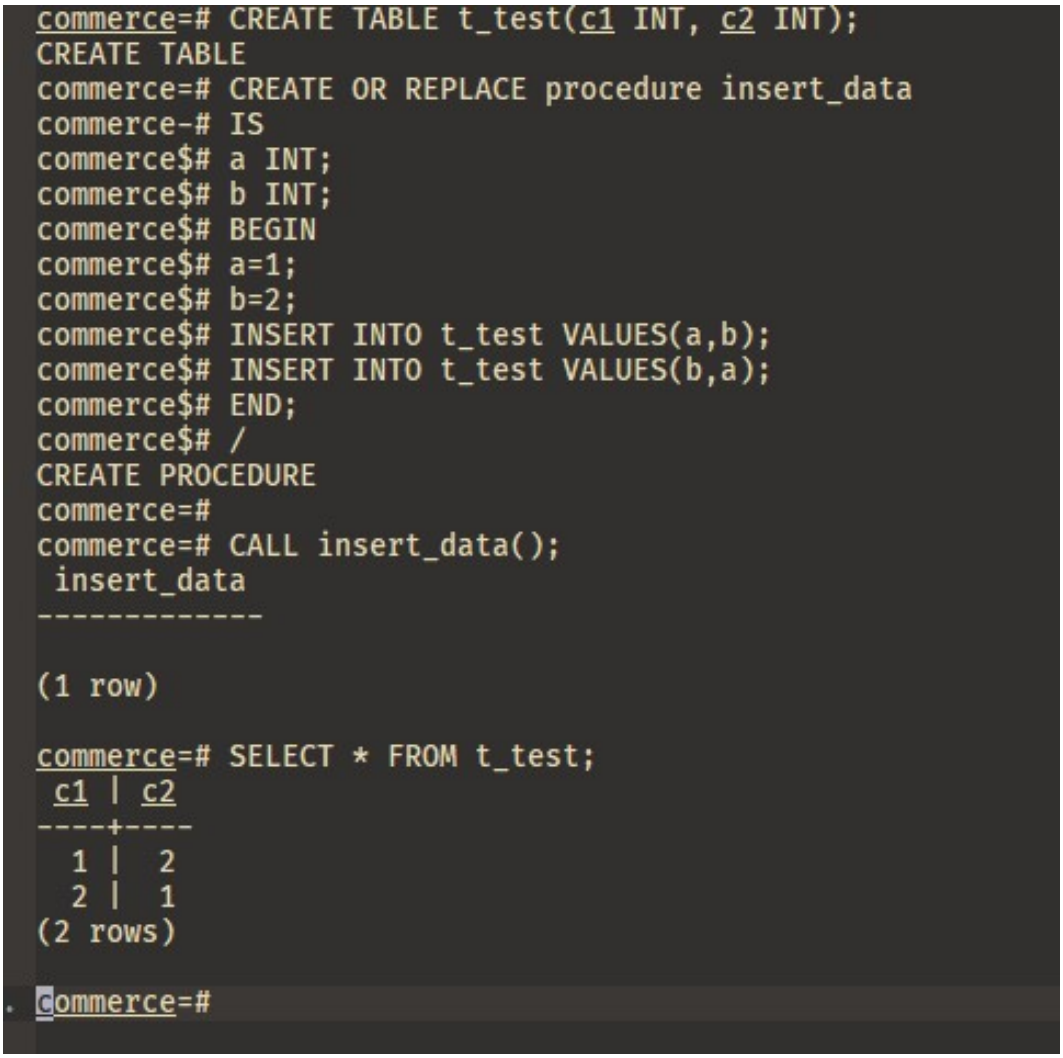


图 3: 创建存储过程并执行

2. 创建触发器并测试效果。

```
--创建源表及触发表
CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

--创建触发器函数
CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
  $$
  DECLARE
  BEGIN
    INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2,
↪   NEW.id3);
    RETURN NEW;
  END
  $$ LANGUAGE PLPGSQL;

CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
  $$
  DECLARE
  BEGIN
    UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
    RETURN OLD;
  END
  $$ LANGUAGE PLPGSQL;

CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
  $$
  DECLARE
  BEGIN
          DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
          RETURN OLD;
  END
  $$ LANGUAGE PLPGSQL;

-- 创建INSERT触发器
```

```sql
CREATE TRIGGER insert_trigger
  BEFORE INSERT ON test_trigger_src_tbl
  FOR EACH ROW
  EXECUTE PROCEDURE tri_insert_func();

-- 创建UPDATE触发器
CREATE TRIGGER update_trigger
  AFTER UPDATE ON test_trigger_src_tbl
  FOR EACH ROW
  EXECUTE PROCEDURE tri_update_func();

-- 创建DELETE触发器
CREATE TRIGGER delete_trigger
  BEFORE DELETE ON test_trigger_src_tbl
  FOR EACH ROW
  EXECUTE PROCEDURE tri_delete_func();

-- 执行INSERT触发事件并检查触发结果
INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
SELECT * FROM test_trigger_src_tbl;
SELECT * FROM test_trigger_des_tbl;

-- 执行UPDATE触发事件并检查触发结果
UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
SELECT * FROM test_trigger_src_tbl;
SELECT * FROM test_trigger_des_tbl;

-- 执行DELETE触发事件并检查触发结果
DELETE FROM test_trigger_src_tbl WHERE id1=100;
SELECT * FROM test_trigger_src_tbl;
SELECT * FROM test_trigger_des_tbl;

-- 修改触发器
ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO
↪  delete_trigger_renamed;
```

```sql
-- 禁用insert_trigger触发器
ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

-- 禁用当前表上所有触发器
ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;

--删除触发器
DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
DROP TRIGGER update_trigger ON test_trigger_src_tbl;
DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

```
commerce=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);
ERROR:  relation "test_trigger_des_tbl" already exists
commerce=#
commerce=# --创建触发器函数
commerce=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
commerce-#   $$
commerce$#   DECLARE
commerce$#   BEGIN
commerce$#     INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
commerce$#     RETURN NEW;
commerce$#   END
commerce$#   $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
commerce=#
commerce=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
commerce-#   $$
commerce$#   DECLARE
commerce$#   BEGIN
commerce$#     UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
commerce$#     RETURN OLD;
commerce$#   END
commerce$#   $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
commerce=#
commerce=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
commerce-#   $$
commerce$#   DECLARE
commerce$#   BEGIN
commerce$#           DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
commerce$#           RETURN OLD;
commerce$#   END
commerce$#   $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
commerce=#
commerce=# -- 创建INSERT触发器
commerce=# CREATE TRIGGER insert_trigger
commerce-#   BEFORE INSERT ON test_trigger_src_tbl
commerce-#   FOR EACH ROW
commerce-#   EXECUTE PROCEDURE tri_insert_func();
CREATE TRIGGER
commerce=#
commerce=# -- 创建UPDATE触发器
commerce=# CREATE TRIGGER update_trigger
commerce-#   AFTER UPDATE ON test_trigger_src_tbl
commerce-#   FOR EACH ROW
commerce-#   EXECUTE PROCEDURE tri_update_func();
CREATE TRIGGER
commerce=#
commerce=# -- 创建DELETE触发器
commerce=# CREATE TRIGGER delete_trigger
commerce-#   BEFORE DELETE ON test_trigger_src_tbl
commerce-#   FOR EACH ROW
commerce-#   EXECUTE PROCEDURE tri_delete_func();
CREATE TRIGGER
commerce=#
```

图 4: 创建触发器并测试效果

```
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)

commerce=#
commerce=# -- 执行UPDATE触发事件并检查触发结果
commerce=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
UPDATE 1
commerce=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 400
(1 row)

commerce=# SELECT * FROM test_trigger_des_tbl;
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 400
(1 row)

commerce=#
commerce=# -- 执行DELETE触发事件并检查触发结果
commerce=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
DELETE 1
commerce=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
(0 rows)

commerce=# SELECT * FROM test_trigger_des_tbl;
 id1 | id2 | id3
-----+-----+-----
(0 rows)

commerce=#
commerce=# -- 修改触发器
commerce=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;
ALTER TRIGGER
commerce=#
commerce=# -- 禁用insert_trigger触发器
commerce=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;
ALTER TABLE
commerce=#
commerce=# -- 禁用当前表上所有触发器
commerce=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;
ALTER TABLE
commerce=#
commerce=# --删除触发器
commerce=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
DROP TRIGGER
commerce=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
DROP TRIGGER
commerce=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
DROP TRIGGER
commerce=#
```

图 5: 创建触发器并测试效果

4.3 数据库备份与恢复

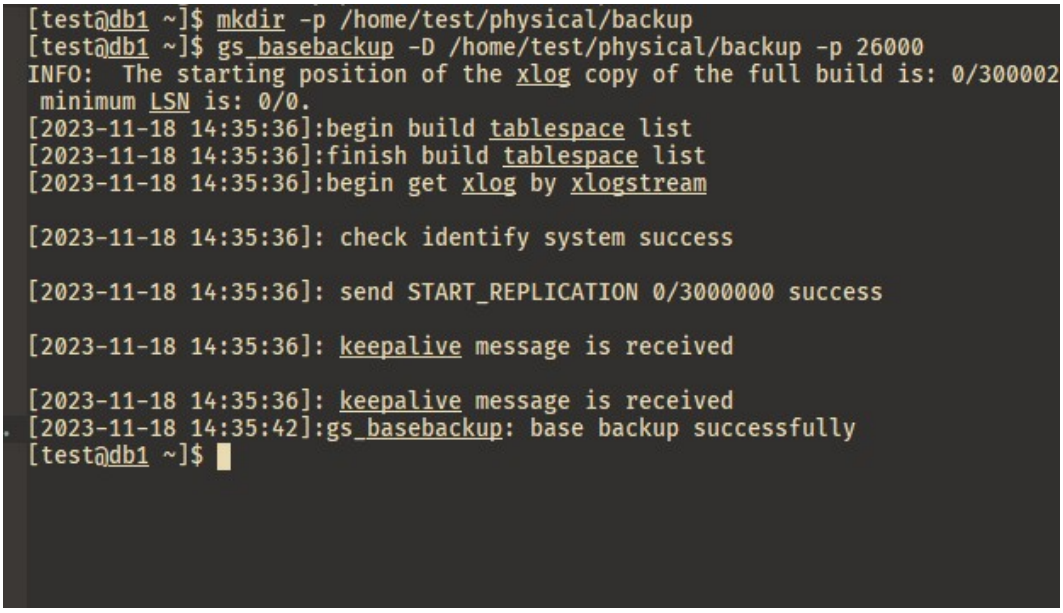
1. 对所创建的数据库进行备份

```
DROP TABLE IF EXISTS customer_t1;
CREATE TABLE customer_t1
(
 c_customer_sk integer,
```

```
 c_customer_id char(5),
 c_first_name char(6),
 c_last_name char(8)
);
INSERT INTO customer_t1 (c_customer_sk, c_customer_id,
↪  c_first_name) VALUES
(3769, 'hello', DEFAULT) ,
(6885, 'maps', 'Joes'),
(4321, 'tpcds', 'Lily'),
(9527, 'world', 'James');

DROP TABLE IF EXISTS customer_t2;
CREATE TABLE customer_t2
(
 c_customer_sk integer,
 c_customer_id char(5),
 c_first_name char(6),
 c_last_name char(8)
);
INSERT INTO customer_t2 (c_customer_sk, c_customer_id,
↪  c_first_name) VALUES
(3769, 'hello', DEFAULT) ,
(6885, 'maps', 'Joes'),
(9527, 'world', 'James');

DROP user IF EXISTS lucy;
CREATE USER lucy WITH PASSWORD "Bigdata@123";
\c - lucy
DROP TABLE IF EXISTS lucy.mytable;
CREATE TABLE mytable (firstcol int);
INSERT INTO mytable values (100);

mkdir -p /home/test/physical/backup
gs_basebackup -D /home/test/physical/backup -p 26000
```

```
. INSERT INTO customer_t2 (c_customer_sk, c_customer_id, c_first_name) VALUES
  (3769, 'hello', DEFAULT) ,
  (6885, 'maps', 'Joes'),
  (9527, 'world', 'James');

  DROP user IF EXISTS lucy;
  CREATE USER lucy WITH PASSWORD "Bigdata@123";
  \c - lucyDROP TABLE
  commerce=# CREATE TABLE customer_t1
  commerce-# (
  commerce(#  c_customer_sk integer,
  commerce(#  c_customer_id char(5),
  commerce(#  c_first_name char(6),
  commerce(#  c_last_name char(8)
  commerce(# );
  CREATE TABLE
  commerce=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
  commerce-# (3769, 'hello', DEFAULT) ,
  commerce-# (6885, 'maps', 'Joes'),
  commerce-# (4321, 'tpcds', 'Lily'),
  commerce-# (9527, 'world', 'James');
  INSERT 0 4
  commerce=#
  commerce=# DROP TABLE IF EXISTS customer_t2;
  DROP TABLE
  commerce=# CREATE TABLE customer_t2
  commerce-# (
  commerce(#  c_customer_sk integer,
  commerce(#  c_customer_id char(5),
  commerce(#  c_first_name char(6),
  commerce(#  c_last_name char(8)
> commerce(# );
  CREATE TABLE
  commerce=# INSERT INTO customer_t2 (c_customer_sk, c_customer_id, c_first_name) VALUES
  commerce-# (3769, 'hello', DEFAULT) ,
  commerce-# (6885, 'maps', 'Joes'),
  commerce-# (9527, 'world', 'James');
  INSERT 0 3
  commerce=#
  commerce=# DROP user IF EXISTS lucy;
  NOTICE:  role "lucy" does not exist, skipping
  DROP ROLE
  commerce=# CREATE USER lucy WITH PASSWORD "Bigdata@123";
  CREATE ROLE
  commerce=# \c - lucy
  Password for user lucy:
  Non-SSL connection (SSL connection is recommended when requiring high-security)
  You are now connected to database "commerce" as user "lucy".
  commerce=> DROP TABLE IF EXISTS lucy.mytable;
  NOTICE:  table "mytable" does not exist, skipping
  DROP TABLE
  commerce=> CREATE TABLE mytable (firstcol int);
  CREATE TABLE
  commerce=> INSERT INTO mytable values (100);
  INSERT 0 1
  commerce=>
```

图 6: 对所创建的数据库进行备份

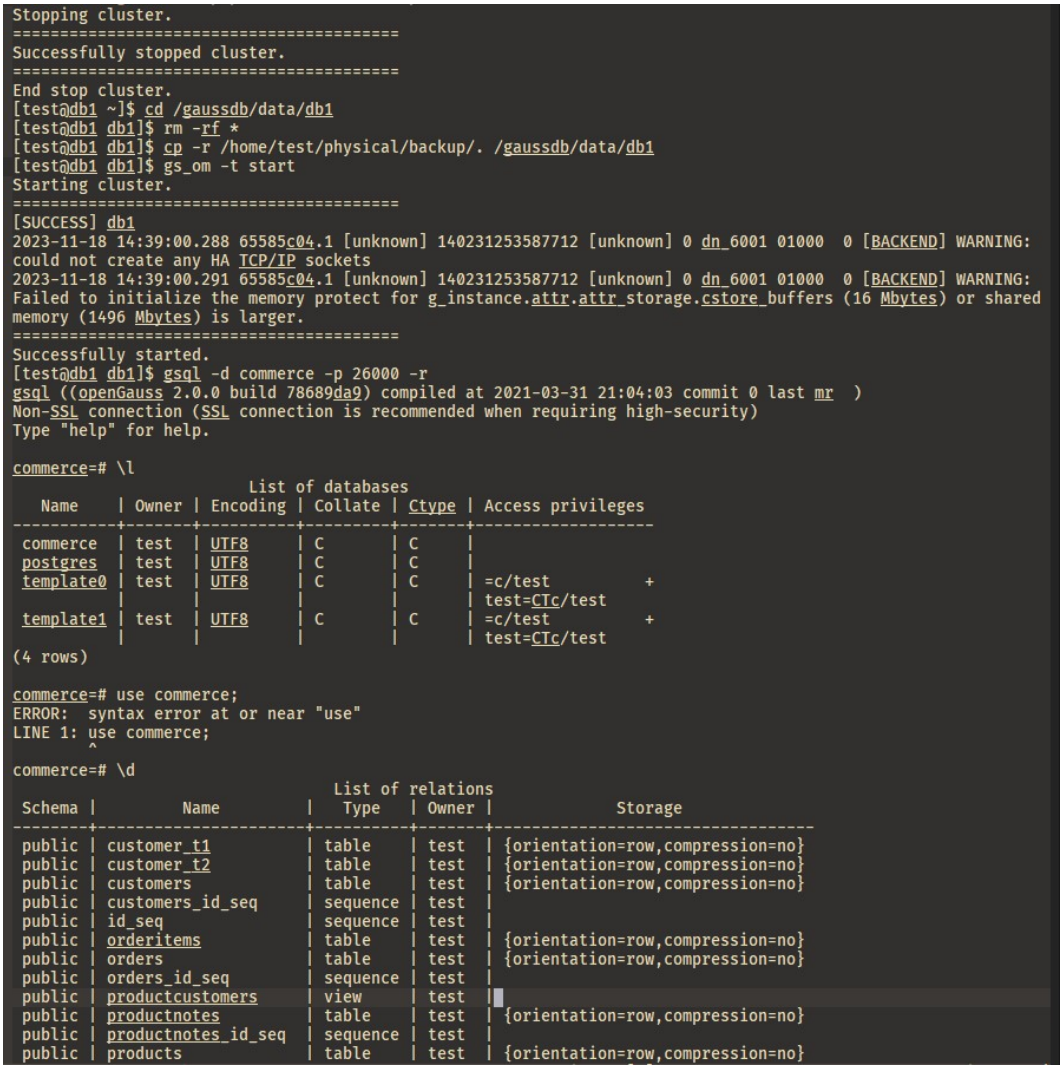


```
[test@db1 ~]$ mkdir -p /home/test/physical/backup
[test@db1 ~]$ gs_basebackup -D /home/test/physical/backup -p 26000
INFO:  The starting position of the xlog copy of the full build is: 0/300002
 minimum LSN is: 0/0.
[2023-11-18 14:35:36]:begin build tablespace list
[2023-11-18 14:35:36]:finish build tablespace list
[2023-11-18 14:35:36]:begin get xlog by xlogstream

[2023-11-18 14:35:36]: check identify system success

[2023-11-18 14:35:36]: send START_REPLICATION 0/3000000 success

[2023-11-18 14:35:36]: keepalive message is received

[2023-11-18 14:35:36]: keepalive message is received
[2023-11-18 14:35:42]:gs_basebackup: base backup successfully
[test@db1 ~]$
```

图 7: 对所创建的数据库进行备份

2. 利用备份进行数据库恢复

```
gs_om -t stop
cd /gaussdb/data/db1
rm -rf *
cp -r /home/test/physical/backup/. /gaussdb/data/db1
gs_om -t start
```



```
Stopping cluster.
=========================================
Successfully stopped cluster.
=========================================
End stop cluster.
[test@db1 ~]$ cd /gaussdb/data/db1
[test@db1 db1]$ rm -rf *
[test@db1 db1]$ cp -r /home/test/physical/backup/. /gaussdb/data/db1
[test@db1 db1]$ gs_om -t start
Starting cluster.
=========================================
[SUCCESS] db1
2023-11-18 14:39:00.288 65585c04.1 [unknown] 140231253587712 [unknown] 0 dn_6001 01000  0 [BACKEND] WARNING:
could not create any HA TCP/IP sockets
2023-11-18 14:39:00.291 65585c04.1 [unknown] 140231253587712 [unknown] 0 dn_6001 01000  0 [BACKEND] WARNING:
Failed to initialize the memory protect for g_instance.attr.attr_storage.cstore_buffers (16 Mbytes) or shared
memory (1496 Mbytes) is larger.
=========================================
Successfully started.
[test@db1 db1]$ gsql -d commerce -p 26000 -r
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:04:03 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

commerce=# \l
                              List of databases
   Name    | Owner | Encoding | Collate | Ctype | Access privileges
-----------+-------+----------+---------+-------+-------------------
 commerce  | test  | UTF8     | C       | C     |
 postgres  | test  | UTF8     | C       | C     |
 template0 | test  | UTF8     | C       | C     | =c/test          +
           |       |          |         |       | test=CTc/test
 template1 | test  | UTF8     | C       | C     | =c/test          +
           |       |          |         |       | test=CTc/test
(4 rows)

commerce=# use commerce;
ERROR:  syntax error at or near "use"
LINE 1: use commerce;
        ^
commerce=# \d
                                  List of relations
 Schema |          Name          |   Type   | Owner |              Storage
--------+------------------------+----------+-------+-----------------------------------
 public | customer_t1            | table    | test  | {orientation=row,compression=no}
 public | customer_t2            | table    | test  | {orientation=row,compression=no}
 public | customers              | table    | test  | {orientation=row,compression=no}
 public | customers_id_seq       | sequence | test  |
 public | id_seq                 | sequence | test  |
 public | orderitems             | table    | test  | {orientation=row,compression=no}
 public | orders                 | table    | test  | {orientation=row,compression=no}
 public | orders_id_seq          | sequence | test  |
 public | productcustomers       | view     | test  |
 public | productnotes           | table    | test  | {orientation=row,compression=no}
 public | productnotes_id_seq    | sequence | test  |
 public | products               | table    | test  | {orientation=row,compression=no}
```

图 8: 利用备份进行数据库恢复

5 实验总结

5.1 实验涉及到的相关知识

- 数据库账户的添加和删除;
- 对账户进行授予权限, 收回权限;
- 创建并执行存储过程;

- 创建触发器并测试效果;
- 删除触发器;
- 数据库的备份和恢复.

5.2 实验遇到的问题及其解决

没有问题.