

# 《事务管理》

## 实验报告



学院： 计算机学院（国家示范性软件学院）

---

班级： 2019211308 2019211308 2019211308

---

姓名： 顾天阳 曾世茂 庞仕泽

---

学号： 2019211539 2019211532 2019211509

---

# openGauss 事务管理

## 目录

1.单事务于串行事务.....	2
1.1 违反 check 约束的 update 操作.....	2
1.2 事务 commit/rollback 操作.....	10
2.1.3 修改数据库模式.....	15
1.4 多条 insert/delete 操作执行比较.....	20
1.5 保存点 Savepoint 设置与回滚实验.....	25
1.6 事务内某条语句执行失败对其余语句的影响.....	26
2.并发事务控制.....	28
2.1 read committed 隔离级别下的脏读，不可重复读，幻读.....	28
2.2 repeatable read 隔离级别下的脏读，不可重复读，幻读.....	38
3.事务锁机制.....	49
3.1 死锁分析.....	49
示例 1.隔离级别 read-repeatable 下死锁.....	49
示例 2.加的互斥锁的粒度.....	52
4 备份与恢复.....	56
5.问题及解决.....	56
问题一：.....	56
问题二：.....	57
问题三：.....	57
问题四：.....	57

## 1.单事务于串行事务

### 1.1 违反 check 约束的 update 操作

在 TD-LTE 网络数据库中，小区/基站工参表 tbCell 中的小区天线高度不能小于 0。在关系表 tbCell (注意:实验前备份该表，以防实验造成数据丢失)上，用 Alter table add check 添加约束，并在该备份表上完成以下实验内容:

Step1.查询小区/基站工参表的小区天线高度 (HEIGHT)小于 20 的 SECTOR\_ID、SECTOR\_NAME 和 HEIGHT;

Step2.更新小区/基站工参表将 step1 中的 HEIGHT 设置为当前值减去 15 (注意此时有可能违反 check 约束)

Step3.查询小区/基站工参表的小区天线高度 (HEIGHT)小于 20 的 SECTOR\_ID、SECTOR\_NAME 和 HEIGHT;针对以上操作分别进行如下的操作:

(1)将以上操作组织成普通的 SQL 语句，顺序执行。

(2)将以上操作组织成事务执行(以 start transaction;开始，以 end;结束查看数据库，观察两次  
的执行结果有何异同。

为方便起见，创建 tbccl 表的副本 tbccl\_1, tbccl\_3（注意 tbccl\_2 先前已被使用并被更改，  
故作 tbccl\_3），并将 tbccl 表的数据导入进去

```
create table tbccl_1(  
    CITY varchar(255) DEFAULT NULL,  
    SECTOR_ID varchar(50)PRIMARY KEY,  
    SECTOR_NAME varchar(255) NOT NULL,  
    ENODEBID integer NOT NULL,  
    ENODEB_NAME varchar(255)NOT NULL,  
    EARFCN integer NOT NULL,  
    PCI integer CHECK(PCI between 0 and 503),  
    PSS integer CHECK(PSS between 0 and 2),  
    SSS integer CHECK(SSS between 0 and 167),  
    TAC integer,  
    VENDOR varchar(255),  
    LONGITUDE float NOT NULL,  
    LATITUDE float NOT NULL,  
    STYLE varchar(255),  
    AZIMUTH float not NULL,  
    HEIGHT float,  
    ELECTTILT float,  
    MECHTILT float,  
    TOTLETILT float
```

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL ▾

6

ENODEBID integer NOT NULL,

7

ENODEB\_NAME varchar(255)NOT NULL,

8

EARFCN integer NOT NULL,

9

PCI integer CHECK(PCI between 0 and 503),

10

PSS integer CHECK(PSS between 0 and 2),

11

SSS integer CHECK(SSS between 0 and 167),

12

TAC integer,

13

VENDOR varchar(255),

14

LONGITUDE float NOT NULL,

15

LATITUDE float NOT NULL,

16

STYLE varchar(255),

17

AZIMUTH float not NULL,

18

HEIGHT float,

19

ELECTTILT float,

20

MECHTILT float,

21

TOTLETILT float check(TOTLETILT = ELECTTILT + MECHTILT) NOT NULL

22

);

SQL执行记录

消息

SSS integer CHECK(SSS between 0 and 167),

TAC integer,

VENDOR varchar(255),

LONGITUDE float NOT NULL,

LATITUDE float NOT NULL,

STYLE varchar(255),

AZIMUTH float not NULL,

HEIGHT float,

ELECTTILT float,

MECHTILT float,

TOTLETILT float check(TOTLETILT = ELECTTILT + MECHTILT) NOT NULL

);

执行成功, 耗时: [38ms.]

warning:

CREATE TABLE / PRIMARY KEY will create implicit index "tbcell\_1\_pkey" for table "tbcell\_1"

);

1

insert into tbcell\_1

2

select \*

3

from tbcell\_invariable;

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)】

insert into tbcell\_1

select \*

from tbcell\_invariable;

执行成功, 耗时: [52ms.]

克隆 tbcell\_3

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL v

```
1 insert into tbccll_3
2 select *
3 from tbccll_invariable;
```

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）】  
insert into tbccll\_3  
select \*  
from tbccll\_invariable;  
执行成功，耗时：[49ms.]

用 Alter table add check 添加约束

alter table tbccll\_1 add constraint tbccll\_chk\_1 check(HEIGHT>=0);

alter table tbccll\_3 add constraint tbccll\_chk\_2 check(HEIGHT>=0);

(已添加，此处用 56 代替)

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL v

```
1 alter table tbccll_1 add constraint tbccll_chk_6 check(HEIGHT>=0);
2 alter table tbccll_3 add constraint tbccll_chk_4 check(HEIGHT>=0);
3
```

SQL执行记录 消息

-----开始执行-----  
【拆分SQL完成】：将执行SQL语句数量：（2条）  
【执行SQL：（1）】  
alter table tbccll\_1 add constraint tbccll\_chk\_6 check(HEIGHT>=0);  
执行成功，耗时：[11ms.]  
【执行SQL：（2）】  
alter table tbccll\_3 add constraint tbccll\_chk\_4 check(HEIGHT>=0);  
执行成功，耗时：[12ms.]

将实验内容在 tbccll\_1 表上组织成普通的 SQL 语句，顺序执行

1

select \*

2

from tbccll\_1

3

where HEIGHT<10;

4

5

update tbccll\_1

6

set HEIGHT = HEIGHT-8

7

where HEIGHT<10;

8

9

select \*

10

from tbccll\_1

11

where HEIGHT<10;

12

SQL执行记录

消息

结果集1 X

结果集2 X

select \*

from tbccll\_1

where HEIGHT<10;

执行成功, 当前返回: [50]行, 耗时: [28ms.]

【执行SQL: (2)】

update tbccll\_1

set HEIGHT = HEIGHT-8

where HEIGHT<10;

执行失败, 失败原因: ERROR: new row for relation "tbccll\_1" violates check constraint "tbccll\_chk\_1"

Detail: Failing row contains (sanxia, 124721-0, C氏蚂蚁岭-HLHF-1, 124721, C氏蚂蚁岭-HLHF, 38400, 121, 1, 40, 14579, 华为, 112, 33, 宏站, 40, -1, 6, 1, 7).

【执行SQL: (3)】

select \*

from tbccll\_1

where HEIGHT<10;

执行成功, 当前返回: [50]行, 耗时: [13ms.]

更新时报错  
两次查询结果一样（602 行）

1

select \*

2

from tbccll\_1

3

where HEIGHT<10;

4

5

update tbccll\_1

6

set HEIGHT = HEIGHT-8

7

where HEIGHT<10;

8

9

select \*

10

from tbccll\_1

11

where HEIGHT<10;

12

SQL执行记录

消息

结果集1 X

结果集2 X

以下是select \* from tbccll\_1 where HEIGHT<10;的执行结果集

该表

	city	sector_id	sector_name	enod
597	sanxia	7391-144	海洋新天域-HLWE-1	7391
598	sanxia	7392-144	D县交警队办公楼-HLWE-1	7392
599	sanxia	7393-144	D县天鹅湾--HLWE-1	7393
600	sanxia	7395-144	砥柱大厦-HLWE-1	7395
601	sanxia	7396-144	迎宾花园一期二-HLWE-1	7396
602	sanxia	7397-144	迎宾花园一期三-HLWE-1	7397

```

1 select *
2 from tbccll_1
3 where HEIGHT<10;
4
5 update tbccll_1
6 set HEIGHT = HEIGHT+8
7 where HEIGHT<10;
8
9 select *
10 from tbccll_1
11 where HEIGHT<10;
12

```

SQL执行记录 消息 结果集1 X 结果集2 X

以下select \* from tbccll\_1 where HEIGHT<10;的执行结果

	city	sector_id	sector_name	enodeb_id	enodeb_name	earfcn	pci	psn
597	sancila	7391-144	海洋新无线-HLME-1	7391	海洋新无线-HLME	38950	350	2
598	sancila	7392-144	D县交警队办公楼-HLME-1	7392	D县交警队办公楼-HLME	38950	319	1
599	sancila	7393-144	D县天越寺-HLME-1	7393	D县天越寺-HLME	38950	321	0
600	sancila	7395-144	砥柱大厦-HLME-1	7395	砥柱大厦-HLME	38950	352	1
601	sancila	7396-144	迎宾花园一期二-HLME-1	7396	迎宾花园一期二-HLME	38950	97	1
602	sancila	7397-144	迎宾花园一期三-HLME-1	7397	迎宾花园一期三-HLME	38950	98	2

将实验内容在 tbccll\_3 表上组织成事务执行（以 start transaction;开始，以 end;结束）。

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL v

```

1 start TRANSACTION ;
2

```

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）】

```
start TRANSACTION ;
```

执行成功，耗时：[5ms.]

在事务中查询“小区天线高度（HEIGHT）小于 10”的小区，结果与上面的查询结果一样

```
1 SELECT *
2 FROM tbcell_3
3 where height < 10
```

SQL执行记录 消息 结果集1 X

-----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 tbcell\_3  
where height < 10  
执行成功，当前返回：[50]行，耗时：[21ms.]

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL v

```
1 SELECT *
2 FROM tbcell_3
3 where height < 10
```

SQL执行记录 消息 结果集1 X

以下是SELECT \* FROM tbcell\_3 where height < 10的执行结果集 ⓘ 该表不可编辑。

	city	sector_id	sector_name	enodebid	enodeb_name
1	saxia	124721-0	C氏蚂蚁岭-HLHF-1	124721	C氏蚂蚁岭-HLHF
2	saxia	124721-1	C氏蚂蚁岭-HLHF-2	124721	C氏蚂蚁岭-HLHF
3	saxia	124721-2	C氏蚂蚁岭-HLHF-3	124721	C氏蚂蚁岭-HLHF
4	saxia	124786-0	E宝坂里-HLHF-1	124786	E宝坂里-HLHF
5	saxia	124786-1	E宝坂里-HLHF-2	124786	E宝坂里-HLHF
6	saxia	124786-2	E宝坂里-HLHF-3	124786	E宝坂里-HLHF

最多显示 50 | 行

然后，在事务中更新：

```
1 update tbcell_3
2 set HEIGHT = HEIGHT-8
3 where HEIGHT<10;
```

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）】**  
update tbcell\_3  
set HEIGHT = HEIGHT-8  
where HEIGHT<10;  
执行失败，失败原因: ERROR: new row for relation "tbcell\_3" violates check constraint "tbcell\_chk\_1"  
Detail: Failing row contains (saxia, 124721-0, C氏蚂蚁岭-HLHF-1, 124721, C氏蚂蚁岭-HLHF, 38400, 121, 1, 40, 14579, 华为, 112, 33, 宏站, 40, -1, 6, 1, 7).

报错。

再次，在事务中查询“小区天线高度(HEIGHT)小于 10”的小区



```
1 select *
2 from tbcell_3
3 where HEIGHT<10;
```

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）】

select \*

from tbcell\_3

where HEIGHT<10;

执行失败，失败原因：ERROR: current transaction is aborted, commands ignored until end of transaction block, firstChar[P]

报错，由于事务中前面的命令执行失败，之后的命令便无法执行了，需要将事务回滚  
END 无法结束事务，rollback 也报同样的错，最后关闭该查询解决

```
1 END ;
2
```

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）】

END ;

执行失败，失败原因：ERROR: current transaction is aborted, commands ignored until end of transaction block, firstChar[P]

代码：

START transaction;

select \*

from tbcell\_1

where HEIGHT<10;

update tbcell\_1

set HEIGHT = HEIGHT-8

where HEIGHT<10;

select \*

from tbcell\_1

where HEIGHT<10;

end

退出事务后，再次查询

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```

1 select *
2 from tbcell_3
3 where HEIGHT<10;

```

SQL执行记录

消息

结果集1

以下select \* from tbcell\_3 where HEIGHT<10;的执行结果集

该表不可编辑。

	city	sector_id	sector_name	enodebid	enodeb_name
1	sanxia	124721-0	C氏蚂蚁岭-HLHF-1	124721	C氏蚂蚁岭-HLH
2	sanxia	124721-1	C氏蚂蚁岭-HLHF-2	124721	C氏蚂蚁岭-HLH
3	sanxia	124721-2	C氏蚂蚁岭-HLHF-3	124721	C氏蚂蚁岭-HLH
4	sanxia	124786-0	E宝坂里-HLHF-1	124786	E宝坂里-HLHF
5	sanxia	124786-1	E宝坂里-HLHF-2	124786	E宝坂里-HLHF
6	sanxia	124786-2	E宝坂里-HLHF-3	124786	E宝坂里-HLHF
7	sanxia	124799-16	A池万洋国际城-HLME-1	124799	A池万洋国际城
8	sanxia	124804-2	E宝孟家河-HLHF-3	124804	E宝孟家河-HLH
9	sanxia	124804-3	E宝后孟家河-HLHF-4	124804	E宝孟家河-HLH
10	sanxia	124804-4	E宝孟家河-HLHF-5	124804	E宝孟家河-HLH

与上面的查询结果一致，数据没有发生变动

分析:当执行 update 语句之后发现报错，check 约束不合法，会导致回滚，单语句顺序执行的话，会导致单语句回滚，事务执行的话，则是整个事务回滚，说明当有 check 约束时，某行更新失败会使得整条语句（或者整个事务）全部回滚，并非是只跳过 check 不通过的那些行。

## 1.2 事务 commit/rollback 操作

分别以两种事务执行模式，即自动提交、显式提交，在关系表 tbCell 上执行以下操作，并观察、分析、解释执行结果。注意:实验前备份该表，以防实验造成数据丢失。

Step1.查看小区 ID 在'122880-0'和 122882-2'之间的小区配置的频点编号;

Step2.将小区 ID 在 122880-0'和 122882-2'之间的小区配置的频点编号更新为 37900;Step3.再次查看小区 ID 在 122880-0'和'122882-2'之间的小区配置的频点编号。

将 step1、step2 和 step3 的数据库访问组织成 1 个单一事务 T1，再将 step3 作为 1 个独立事务，提交 DBMS,串行执行这 2 个事务，观察 T1 中的 rollback、commit 对事务执行结果的影响。

由 step1、step2 和 step3 组成的事务 T1 采用以下 2 种结束方式:

- (1)以 commit 结束。
- (2)以 rollback 结束。

以 commit 结束的 T1 事务

```
1 START TRANSACTION;
2 select SECTOR_ID,EARFCN from tbcell_1
3 where SECTOR_ID between '124672-0' and '124675-2';
4 update tbcell_1
5 set EARFCN=38950
6 where SECTOR_ID between '124672-0' and '124675-2';
7 select SECTOR_ID,EARFCN
8 from tbcell_1
9 where SECTOR_ID between '124672-0' and '124675-2';
10 commit;
```

SQL执行记录 消息 结果集1 X 结果集2 X

【执行SQL: (1)】  
START TRANSACTION;  
执行成功, 耗时: [6ms.]

【执行SQL: (2)】  
select SECTOR\_ID,EARFCN from tbcell\_1  
where SECTOR\_ID between '124672-0' and '124675-2';  
执行成功, 当前返回: [11]行, 耗时: [14ms.]

【执行SQL: (3)】  
update tbcell\_1  
set EARFCN=38950  
where SECTOR\_ID between '124672-0' and '124675-2';  
执行成功, 耗时: [7ms.]

【执行SQL: (4)】  
select SECTOR\_ID,EARFCN  
from tbcell\_1  
where SECTOR\_ID between '124672-0' and '124675-2';  
执行成功, 当前返回: [11]行, 耗时: [6ms.]

【执行SQL: (5)】  
commit;  
执行成功, 耗时: [8ms.]

共查询到 12 条数据，并将 EARFCN 更新为 38950，更新成功，将事务提交。

SQL执行记录 消息 结果集1 X 结果集2 X

以下是select SECTOR\_ID,EARFCN from tbcell\_1 where SECTOR\_ID between '124672-0' and ...的执行结果集 该表不可编辑。

	sector_id	earfcn
3	124673-0	38950
4	124673-1	38950
5	124673-2	38950
6	124674-0	38950
7	124674-1	38950
8	124674-2	38950
9	124675-0	38950
10	124675-1	38950
11	124675-2	38950

最多显示 50 行

作为独立事务，再次进行查询

```

1 select SECTOR_ID,EARFCN
2 from tbcell_1
3 where SECTOR_ID between '124672-0' and '124675-2';
4

```

SQL执行记录

消息

结果集1 X

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】

```

select SECTOR_ID,EARFCN
from tbcell_1
where SECTOR_ID between '124672-0' and '124675-2';
执行成功，当前返回：[11]行，耗时：[19ms.]

```

数据为更新后的数据，可知，T1 事务内的更新操作有效，数据表发生更改。

```

1 select SECTOR_ID,EARFCN
2 from tbcell_1
3 where SECTOR_ID between '124672-0' and '124675-2';
4

```

SQL执行记录

消息

结果集1 X

以下是select SECTOR\_ID,EARFCN from tbcell\_1 where SECTOR\_ID between '124672-0' and ...的执行结果集

ⓘ 该表不可编辑。

	sector_id	earfcn
3	124673-0	38950
4	124673-1	38950
5	124673-2	38950
6	124674-0	38950
7	124674-1	38950
8	124674-2	38950
9	124675-0	38950
10	124675-1	38950
11	124675-2	38950

以 rollback 结束的 T1 事务

```

1 START TRANSACTION;
2 select SECTOR_ID,EARFCN
3 from tbcell_3
4 where SECTOR_ID between '124672-0' and '124675-2';
5 update tbcell_3
6 set EARFCN=38950
7 where SECTOR_ID between '124672-0' and '124675-2';
8 select SECTOR_ID,EARFCN
9 from tbcell_3
10 where SECTOR_ID between '124672-0' and '124675-2';
11 rollback;

```

SQL执行记录

消息

结果集1 X

结果集2 X

```

START TRANSACTION;
执行成功,耗时: [5ms.]

【执行SQL: (2)】
select SECTOR_ID,EARFCN
from tbcell_3
where SECTOR_ID between '124672-0' and '124675-2';
执行成功,当前返回: [11]行,耗时: [6ms.]

【执行SQL: (3)】
update tbcell_3
set EARFCN=38950
where SECTOR_ID between '124672-0' and '124675-2';
执行成功,耗时: [6ms.]

【执行SQL: (4)】
select SECTOR_ID,EARFCN
from tbcell_3
where SECTOR_ID between '124672-0' and '124675-2';
执行成功,当前返回: [11]行,耗时: [7ms.]

【执行SQL: (5)】
rollback;
执行成功,耗时: [5ms.]

```

共查询到 12 条数据，并将 EARFCN 更新为 38950，更新成功，将事务回滚。

SQL执行记录

消息

结果集1 X

结果集2 X

以下是select SECTOR\_ID,EARFCN from tbcell\_3 where SECTOR\_ID between '124672-0' and ...的执行结果集

ⓘ 该表不可编辑。

	sector_id	earfcn
3	124673-0	38950
4	124673-1	38950
5	124673-2	38950
6	124674-0	38950
7	124674-1	38950
8	124674-2	38950
9	124675-0	38950
10	124675-1	38950
11	124675-2	38950

最多显示 50 行

作为独立事务，再次进行查询

1 select SECTOR\_ID,EARFCN  
2 from tbcell\_3  
3 where SECTOR\_ID between '124672-0' and '124675-2';

SQL执行记录消息结果集1 X

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】  
select SECTOR\_ID,EARFCN  
from tbcell\_3  
where SECTOR\_ID between '124672-0' and '124675-2';  
执行成功，当前返回：[11]行，耗时：[7ms.]

数据为更新前的数据，由于事务回滚，事务内的更新操作无效，数据表不发生更改。

1 select SECTOR\_ID,EARFCN  
2 from tbcell\_3  
3 where SECTOR\_ID between '124672-0' and '124675-2';

SQL执行记录消息结果集1 X

以下是select SECTOR\_ID,EARFCN from tbcell\_3 where SECTOR\_ID between '124672-0' and ...的执行结果集

① 该表不可编辑。

	sector_id	earfcn
3	124673-0	38400
4	124673-1	38400
5	124673-2	38400
6	124674-0	38400
7	124674-1	38400
8	124674-2	38400
9	124675-0	38400
10	124675-1	38400
11	124675-2	38400

最多显示 50 行

分析:

当显式执行事务时，一定要记得在事务结尾处执行 commit 操作，否则对数据表的更改就不会持久化生效。

### 2.1.3 修改数据库模式

针对小区/基站信息表 tbCell(注意备份原表),

Step1.修改 TD-LTE 数据库中的 tbCell 表,删除列 height(使用 alter table drop);Step2.修改 tbCell 表,增加列 height (使用 alter table add)。

将 step1、step2 的数据库访问组织成 1 个单一事务 (以显式事务的方式),采用以下 2 种结束方式:(1)以 commit 结束;

(2)以 rollback 结束。

查看数据库 (用 select 语句查看被删除/增加的列),观察数据库模式修改语句 (alter table),是否会受到 rollback,commit 语句的影响。

也可以自行选择或创建表、删除其它关系表,重复以上两步,查看数据库,观察数据库模式定义语句(createtable)模式修改语句(drop table)是否会受到 rollback,commit 语句的影响。

删除 height 列,以 commit 结束

```
1 START TRANSACTION;
2 alter table tbcell_1 drop column HEIGHT;
3 COMMIT;
4
```

SQL执行记录

消息

-----开始执行-----

【拆分SQL完成】: 将执行SQL语句数量: (3条)

【执行SQL: (1)】

START TRANSACTION;

执行成功,耗时: [6ms.]

【执行SQL: (2)】

alter table tbcell\_1 drop column HEIGHT;

执行成功,耗时: [6ms.]

【执行SQL: (3)】

COMMIT;

执行成功,耗时: [10ms.]

查看 height 列

```
1 SELECT height FROM tbc11_1 ;
2
```

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）】

SELECT height FROM tbc11\_1 ;

执行失败，失败原因：ERROR: column "height" does not exist

Position: 8

Where: referenced column: height

报错，显示该列以不存在，事务提交后，删除操作有效  
删除 height 列，以 rollback 结束

库名: tb\_cell  
Schema: bupt2019dbs37  
表 视图  
请按关键词搜索 | 搜索 清除

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUData
- tbATUData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I
- tbHandOver
- tbHandOver\_Copy
- tbHandOver\_Copy1
- tbHandOver\_invariable
- tbMROData

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL v

```
1 START TRANSACTION;
2 alter table tbc11_3 drop column height;
3 ROLLBACK;
```

SQL执行记录 消息

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（3条）

【执行SQL：（1）】  
START TRANSACTION;  
执行成功，耗时：[6ms.]

【执行SQL：（2）】  
alter table tbc11\_3 drop column height;  
执行成功，耗时：[7ms.]

【执行SQL：（3）】  
ROLLBACK;  
执行成功，耗时：[6ms.]

查看 height 列



执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

1

```
SELECT height FROM tbcell_3 ;
```

SQL执行记录

消息

结果集1 X

以下是SELECT height FROM tbcell\_3 的执行结果集

height

143

243

343

443

543

643

730

830

943

1043

该列仍然存在，事务回滚，删除操作无效。

增加 height\_new 列，以 commit 结束

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

1

```
START TRANSACTION;
```

2

```
alter table tbcell_1 add column HEIGHT_NEW float;
```

3

```
COMMIT;
```

4

SQL执行记录

消息

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（3条）

【执行SQL：（1）】  
START TRANSACTION;  
执行成功，耗时： [6ms.]

【执行SQL：（2）】  
alter table tbcell\_1 add column HEIGHT\_NEW float;  
执行成功，耗时： [7ms.]

【执行SQL：（3）】  
COMMIT;  
执行成功，耗时： [11ms.]

查看 height\_new 列

```
1 SELECT height_new FROM tbccll_1 ;
```

SQL执行记录 消息 结果集1 X

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】

SELECT height\_new FROM tbccll\_1 ;

执行成功，当前返回：[50]行，耗时：[17ms.]

显示该列存在，内容为空，事务提交后，增加操作有效

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL v

1 SELECT height\_new FROM tbccll\_1 ;

SQL执行记录 消息 结果集1 X

以下是SELECT height\_new FROM tbccll\_1 .的执行结果集

height_new
1
2
3
4
5
6
7
8
9
...

增加 height\_new 列，以 rollback 结束

```
1 START TRANSACTION;  
2 alter table tbcell_3 add column HEIGHT_NEW float;  
3 ROLLBACK;  
4
```

SQL执行记录

消息

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（3条）

【执行SQL：（1）】

START TRANSACTION;

执行成功，耗时：[5ms.]

【执行SQL：（2）】

alter table tbcell\_3 add column HEIGHT\_NEW float;

执行成功，耗时：[6ms.]

【执行SQL：（3）】

ROLLBACK;

执行成功，耗时：[5ms.]

查看 height\_new 列

```
1 SELECT height_new FROM tbcell_3 ;
```

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）】

SELECT height\_new FROM tbcell\_3 ;

执行失败，失败原因：ERROR: column "height\_new" does not exist

Position: 8

Where: referenced column: height\_new

显示该列不存在，事务回滚，增加操作无效

分析:

数据库模式定义语句(比如: create table)，模式修改语句(比如: drop table)是会受到 rollback, commit 语句的影响。

## 1.4 多条 insert/delete 操作执行比较

针对小区/基站信息表 tbCell(注意备份原表)，

Step1.查询小区/基站工参表的小区天线高度(HEIGHT)小于 7 的 SECTOR\_ID、SECTOR\_NAME 和 HEIGHT;Step2.在小区/基站工参表中，添加一条 SECTOR\_ID 为“211100-2”、HEIGHT 为 6 的信息;

Step3.删除 step2 所添加的信息;

Step4.查询小区/基站工参表的小区天线高度(HEIGHT)小于 7 的 SECTOR\_ID、SECTOR\_NAME 和 HEIGHT;针对以上操作分别进行如下的操作:

(1) 将以上操作组织成普通的 SQL 语句，顺序执行;

(2) 将以上操作组织成事务执行(以 start transaction;开始，以 commit;结束)查看数据库，观察两次的执行结果有何异同。

组织成普通的 SQL 语句，顺序执行（需要还原 tbcell\_1）

```
select SECTOR_ID,SECTOR_NAME,HEIGHT
```

```
from tbcell_1
```

```
where HEIGHT<10;
```

```
INSERT INTO tbcell_1
```

```
values('name1', '111100-1', 'name2',0,'name3',0,0,0,0,0,'name4',0,0,'name5',0,8,0,0,0);
delete from tbcell_1
where SECTOR_ID='111100-1';
select SECTOR_ID,SECTOR_NAME,HEIGHT
from tbcell_1
where HEIGHT<10;
```

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```
1 select SECTOR_ID,SECTOR_NAME,HEIGHT
2 from tbcell_1
3 where HEIGHT<10;
4 INSERT INTO tbcell_1
5 values('name1', '111100-1', 'name2',0,'name3',0,0,0,0,0,'name4',0,0,'name5',0,8,0,0,0);
6 delete from tbcell_1
7 where SECTOR_ID='111100-1';
8 select SECTOR_ID,SECTOR_NAME,HEIGHT
9 from tbcell_1
10 where HEIGHT<10;
```

SQL执行记录 消息 结果集1 X 结果集2 X

开始执行

【拆分SQL完成】：将执行SQL语句数量：（4条）

【执行SQL：（1）】  
select SECTOR\_ID,SECTOR\_NAME,HEIGHT  
from tbcell\_1  
where HEIGHT<10;  
执行成功，当前返回：[50]行，耗时：[11ms.]

【执行SQL：（2）】  
INSERT INTO tbcell\_1  
values('name1', '111100-1', 'name2',0,'name3',0,0,0,0,0,'name4',0,0,'name5',0,8,0,0,0);  
执行成功，耗时：[12ms.]

【执行SQL：（3）】  
delete from tbcell\_1  
where SECTOR\_ID='111100-1';  
执行成功，耗时：[13ms.]

【执行SQL：（4）】  
select SECTOR\_ID,SECTOR\_NAME,HEIGHT  
from tbcell\_1  
where HEIGHT<10;  
执行成功，当前返回：[50]行，耗时：[10ms.]

两次查询的结果相同

1

select

SECTOR\_ID,SECTOR\_NAME,HEIGHT

2

from

tbcell\_1

3

where

HEIGHT<10;

4

INSERT

INTO

tbcell\_1

5

values

(

'name1'

,

'111100-1'

,

'name2'

,

'name3'

,

0

,

0

,

0

,

0

,

'name4'

,

0

,

0

,

'name5'

,

0

,

8

,

0

,

0

,

0

);

6

delete

from

tbcell\_1

7

where

SECTOR\_ID='111100-1';

8

select

SECTOR\_ID,SECTOR\_NAME,HEIGHT

9

from

tbcell\_1

10

where

HEIGHT<10;

SQL执行记录

消息

结果集1 X

结果集2 X

以下是select SECTOR\_ID,SECTOR\_NAME,HEIGHT from tbcell\_1 where HEIGHT<10;的执行结果集

ⓘ 该表不可编辑。

	sector_id	sector_name
594	7388-144	塞菲特酒店-HLWE-1
595	7389-144	德P苑小区三期一-H
596	7390-144	德P苑小区三期二-H
597	7391-144	海洋新天域-HLWE-1
598	7392-144	D县交警队办公楼-H
599	7393-144	D县天鹅湾--HLWE-
600	7395-144	砥柱大厦-HLWE-1
601	7396-144	迎宾花园一期二-HL
602	7397-144	迎宾花园一期三-HL

最多显示 1000 行

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL v

1

select

SECTOR\_ID,SECTOR\_NAME,HEIGHT

2

from

tbcell\_1

3

where

HEIGHT<10;

4

INSERT

INTO

tbcell\_1

5

values

(

'name1'

,

'111100-1'

,

'name2'

,

'name3'

,

0

,

0

,

0

,

0

,

'name4'

,

0

,

0

,

'name5'

,

0

,

8

,

0

,

0

,

0

);

6

delete

from

tbcell\_1

7

where

SECTOR\_ID='111100-1';

8

select

SECTOR\_ID,SECTOR\_NAME,HEIGHT

9

from

tbcell\_1

10

where

HEIGHT<10;

SQL执行记录

消息

结果集1 X

结果集2 X

以下是select SECTOR\_ID,SECTOR\_NAME,HEIGHT from tbcell\_1 where HEIGHT<10;的执行结果集

ⓘ 该表不可编辑。

	sector_id	sector_name
594	7388-144	塞菲特酒店-HLWE-1
595	7389-144	德P苑小区三期一-HLWE-1
596	7390-144	德P苑小区三期二-HLWE-1
597	7391-144	海洋新天域-HLWE-1
598	7392-144	D县交警队办公楼-HLWE-1
599	7393-144	D县天鹅湾--HLWE-1
600	7395-144	砥柱大厦-HLWE-1
601	7396-144	迎宾花园一期二-HLWE-1
602	7397-144	迎宾花园一期二-HLWE-1

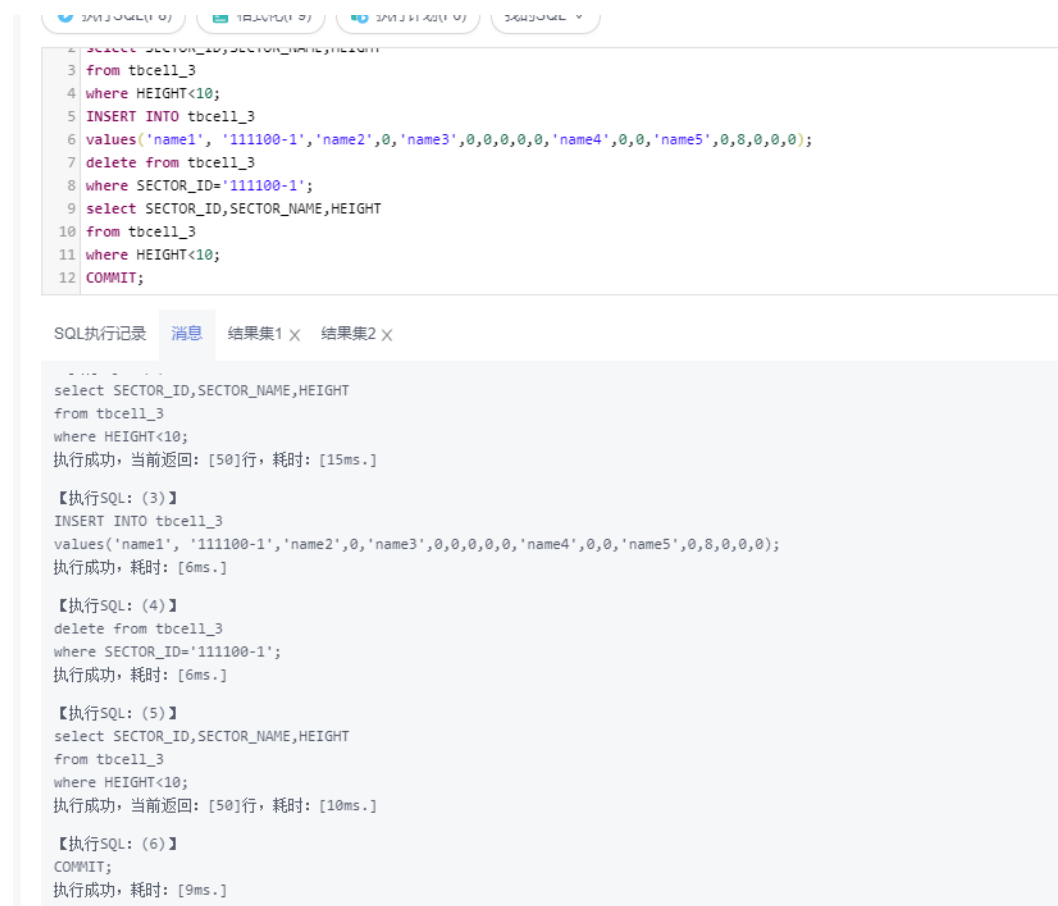
最多显示 1000 行

组织成事务执行(以 start transaction;开始，以 commit;结束)

```

start TRANSACTION ;
select SECTOR_ID,SECTOR_NAME,HEIGHT
from tbcell_3
where HEIGHT<10;
INSERT INTO tbcell_3
values('name1', '111100-1','name2',0,'name3',0,0,0,0,0,'name4',0,0,'name5',0,8,0,0,0);
delete from tbcell_3
where SECTOR_ID='111100-1';
select SECTOR_ID,SECTOR_NAME,HEIGHT
from tbcell_3
where HEIGHT<10;
COMMIT ;

```



The screenshot displays a SQL execution interface with a query editor at the top and a results pane below. The query editor contains 12 lines of SQL code, including a transaction start, two queries on the tbcell\_3 table, an insert, a delete, and a transaction commit. The results pane shows the execution of these statements one by one, providing details such as the SQL statement, success status, and execution time in milliseconds.

```

1 select SECTOR_ID,SECTOR_NAME,HEIGHT
2 from tbcell_3
3 where HEIGHT<10;
4 INSERT INTO tbcell_3
5 values('name1', '111100-1','name2',0,'name3',0,0,0,0,0,'name4',0,0,'name5',0,8,0,0,0);
6 delete from tbcell_3
7 where SECTOR_ID='111100-1';
8 select SECTOR_ID,SECTOR_NAME,HEIGHT
9 from tbcell_3
10 where HEIGHT<10;
11 COMMIT;

```

SQL执行记录 消息 结果集1 X 结果集2 X

```

select SECTOR_ID,SECTOR_NAME,HEIGHT
from tbcell_3
where HEIGHT<10;
执行成功, 当前返回: [50]行, 耗时: [15ms.]

【执行SQL: (3)】
INSERT INTO tbcell_3
values('name1', '111100-1','name2',0,'name3',0,0,0,0,0,'name4',0,0,'name5',0,8,0,0,0);
执行成功, 耗时: [6ms.]

【执行SQL: (4)】
delete from tbcell_3
where SECTOR_ID='111100-1';
执行成功, 耗时: [6ms.]

【执行SQL: (5)】
select SECTOR_ID,SECTOR_NAME,HEIGHT
from tbcell_3
where HEIGHT<10;
执行成功, 当前返回: [50]行, 耗时: [10ms.]

【执行SQL: (6)】
COMMIT;
执行成功, 耗时: [9ms.]

```

两次查询的结果相同（一共 602 行）

SQL执行记录

消息

结果集1 X

结果集2 X

以下是select SECTOR\_ID,SECTOR\_NAME,HEIGHT from tbcell\_3 where HEIGHT<10,的执行结果集

	sector_id	sector_name
594	7388-144	塞菲特酒
595	7389-144	德P苑小
596	7390-144	德P苑小
597	7391-144	海洋新天
598	7392-144	D县交警
599	7393-144	D县天鹅
600	7395-144	砥柱大厦
601	7396-144	迎宾花园
602	7397-144	迎宾花园

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL v

以下是select SECTOR\_ID,SECTOR\_NAME,HEIGHT from tbcell\_3 where HEIGHT<10,的执行结果集

	sector_id	sector_name
594	7388-144	塞菲特酒店-HLWE-1
595	7389-144	德P苑小区三期一-HLWE
596	7390-144	德P苑小区三期二-HLWE
597	7391-144	海洋新天域-HLWE-1
598	7392-144	D县交警队办公楼-HLWE
599	7393-144	D县天鹅湾一-HLWE-1
600	7395-144	砥柱大厦-HLWE-1
601	7396-144	迎宾花园一期二-HLWE-
602	7397-144	迎宾花园一期三-HLWE-

分析:

两次执行的结果没有不同，多条 insert/delete 操作的显式事务操作跟隐式事务操作的结果集一样。



## 1.5 保存点 Savepoint 设置与回滚实验

本实验要求在事务内部不同执行位置设置，例如添加之后、添加之删、删乐之石夺，使用 SAVV 1A 的法甲 savepoint name 语句创建保存点,使用 ROLLBACK savepoint\_name 语句将事务回滚，观察每次操作的结果。

保存点提供了回滚部分事务的机制，而不是回滚到事务的开始。

以小区 PCI 优化调整结果表 tbPCIAssignment 为访问对象，在创建的事务中 insert 插入语句后设置保存点，然后删除添加的信息，并回滚至保存点并提交事务;事务完成后再查询相应的行，观察执行结果是否插入成功，具体如下:

Step1.查询小区 PCI 优化调整结果表的小区 PCI 为 106 的小区的 SECTOR\_ID;

Step2.在小区 PCI 优化调整结果表中，添加一条 SECTOR\_ID 为“220102-5”、PCI 为 106 的信息;Step3.设置保存点;

Step4.删除 step2 所添加的信息;Step5.回滚至保存点;

Step6.事务提交结束;

Step7.查询小区 PCI 优化调整结果表的小区 PCI 为 106 的小区的 SECTOR\_ID;

以下事务以 tbCell 为访问对象，在插入操作之后设置了保存点：

SQL(F8)
格式化(F9)
执行计划(F6)
我的SQL v

```

1  ALTER SECTOR_ID=1;
2  select SECTOR_ID,SECTOR_NAME,HEIGHT
3  from tbcell_1
4  where HEIGHT<10;
5  INSERT INTO tbcell_1
6  values('name1','111100-1','name2','0','name3','0,0,0,0,0','name4','0,0','name5','0,0,0,0,0');
7  SAVEPOINT sp;
8  delete from tbcell_1
9  where SECTOR_ID='111100-1';
10 ROLLBACK to sp;
11 COMMIT;
```

SQL执行记录
消息
结果集1 X

```

FROM tbcell_1
where HEIGHT<10;
执行成功，当前返回: [50]行，耗时: [9ms.]
```

【执行SQL: (3)】

```

INSERT INTO tbcell_1
values('name1','111100-1','name2','0','name3','0,0,0,0,0','name4','0,0','name5','0,0,0,0,0');
执行成功，耗时: [7ms.]
```

【执行SQL: (4)】

```

SAVEPOINT sp;
执行成功，耗时: [6ms.]
```

【执行SQL: (5)】

```

delete from tbcell_1
where SECTOR_ID='111100-1';
执行成功，耗时: [5ms.]
```

【执行SQL: (6)】

```

ROLLBACK to sp;
执行成功，耗时: [5ms.]
```

【执行SQL: (7)】

```

COMMIT;
执行成功，耗时: [8ms.]
```

查看是否插入成功，插入的数据是否被删除

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL v

```
1 select SECTOR_ID,HEIGHT
2 from tbcell_1
3 where SECTOR_ID='111100-1';
4
```

SQL执行记录

消息

结果集1 x

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】

select SECTOR\_ID,HEIGHT

from tbcell\_1

where SECTOR\_ID='111100-1';

执行成功，当前返回：[1]行，耗时：[7ms.]

局部回滚在插入操作之后，删除操作之前，插入成功，插入的数据没有被删除

```
1 select SECTOR_ID,HEIGHT
2 from tbcell_1
3 where SECTOR_ID='111100-1';
4
```

SQL执行记录

消息

结果集1 x

以下是select SECTOR\_ID,HEIGHT from tbcell\_1 where SECTOR\_ID='111100-1'的执行结果集

ⓘ 该表不可编辑。

	sector_id	height
1	111100-1	8

分析:

在事务内部，用 savepoint\_name 语句创建保存点,使用 ROLLBACK to savepoint\_name 语句将事务回滚到保存点的时刻，而不是回滚到事务的开始。保存点提供了回滚部分事务的机制利用 savepoint 来达到局部回滚的目的。

## 1.6 事务内某条语句执行失败对其余语句的影响

根据现网实际情况，小区 PCI 优化调整结果表 tbPCIAssignment 中的小区 PCI 在 0 到 503 之间。在 tbPCIAssignment 的备份表 tbPCIAssignment\_new 上，用 Alter table add check 添加约束，并在该备份表上完成以下实验内容:

Step1.在 tbPCIAssignment\_ new 表上添加约束:加入约束 check(PCI between 0 and 503)。

Step2.以备份表 tbPCIAssignment\_new 表为访问对象，依次添加 PCI 为'300'和'600'的两条数据，将 2 条对备份表 tbPCIAssignment\_new 表进行顺序访问的 insert 语句组织成 1 个显示执行模式下的事务;

Step3.观察并对比当事务执行违反约束时，事务结束后 tbPCIAssignment\_new 的内容。由于之前曾还原 tbcell\_1，在 tbcell\_1 表上加入 HEIGHT>=0 的约束

```
1 alter table tbcell_1 add constraint tbcell_chk_1 check(HEIGHT>=0);
2
```

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)】

```
alter table tbcell_1 add constraint tbcell_chk_1 check(HEIGHT>=0);
```

执行成功,耗时: [11ms.]

插入数据，并更新 HEIGHT 的值

```
1 START TRANSACTION;
2 select SECTOR_ID,HEIGHT
3 from tbcell_1
4 where HEIGHT<10;
5 INSERT INTO tbcell_1
6 values ('name_1', '999999-1', 'name_2',0,'name_3',0,0,0,0,0,'name_4',0,0,'name_5',0,8,0,0,0);
7 update tbcell_1
8 set HEIGHT = HEIGHT-8
9 where HEIGHT<10;
10 COMMIT;
```

SQL执行记录 消息 结果集1 X

START TRANSACTION;  
执行成功,耗时: [6ms.]

【执行SQL: (2)】

```
select SECTOR_ID,HEIGHT
from tbcell_1
```

where HEIGHT<10;

执行成功,当前返回: [50]行,耗时: [9ms.]

【执行SQL: (3)】

```
INSERT INTO tbcell_1
```

```
values ('name_1', '999999-1', 'name_2',0,'name_3',0,0,0,0,0,'name_4',0,0,'name_5',0,8,0,0,0);
```

执行成功,耗时: [5ms.]

【执行SQL: (4)】

```
update tbcell_1
```

```
set HEIGHT = HEIGHT-8
```

```
where HEIGHT<10;
```

执行失败,失败原因: ERROR: new row for relation "tbcell\_1" violates check constraint "tbcell\_chk\_1"

Detail: Failing row contains (sanxia, 124721-0, C氏蚂蚁蛉-HLHF-1, 124721, C氏蚂蚁蛉-HLHF, 38400, 121, 1, 40, 14579, 华为, 112, 33, 宏站, 40, -1, 6, 1, 7).

【执行SQL: (5)】

```
COMMIT;
```

执行失败,失败原因: ERROR: current transaction is aborted, commands ignored until end of transaction block, firstChar[P]

由于不满足 check 约束，更新 HEIGHT 为当前高度减去 8 时失败

用 COMMIT 提交该事务时，由于事务内有命令执行失败，无法执行，关闭查询后自动回滚查询在上面事务中插入的数据



什么都没查到。

为查询到该数据，不只是更新语句回滚，插入语句也跟着回滚，回滚到整个事务开始前

## 2.并发事务控制

### 2.1 read committed 隔离级别下的脏读，不可重复读，幻读

#### (1) 脏读

Step1. 用隐式的独立事务查询 SECTOR\_ID 为“124672-O”的 HEIGHT 值;Step2.创建 read committed 隔离级别下的并发事务 T1 和 T2;

Step3.在事务 T1 中，将 SECTOR\_ID 为“124672-O”的 HEIGHT 修改为 6;

Step4.在事务 T2 中，查询 SECTOR\_ID 为“124672-O”的 HEIGHT 值，并 COMMIT 提交事务 T2;Step5.用 ROLLBACK 回滚事务 T1;

要求:

观察 Step4 的 HEIGHT 值，与 Step1 和 Step3 的 HEIGHT 值进行对比

说明事务 T2 是否读取了事务 T1 未提交的数据，是否存在脏读;

先打开两个查询，并连接相同的数据库

Admin Service GaussDB(for openGauss)

库管理-tb\_cell x 导入 x SQL查询 x SQL查询 x

当前所在库: **tb\_cell** | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000

库名: tb\_cell 执行SQL(F8) 格式化(F9) 执行计划(F6) 我的:

Schema: bupt2019dbs37

表 视图

1 SELECT \* FROM

查询 SECTOR\_ID 为“15500-128”的 HEIGHT 值

```

1 select SECTOR_ID,HEIGHT
2 from tbcell_1
3 where SECTOR_ID='15500-128';
4
5

```

SQL执行记录 消息 结果集1 x

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】

```

select SECTOR_ID,HEIGHT
from tbcell_1
where SECTOR_ID='15500-128';

```

执行成功，当前返回：[1]行，耗时：[7ms.]

```

1 select SECTOR_ID,HEIGHT
2 from tbcell_1
3 where SECTOR_ID='15500-128';
4
5

```

SQL执行记录 消息 结果集1 x

以下是select SECTOR\_ID,HEIGHT from tbcell\_1 where SECTOR\_ID='15500-128'的执行结果集 ⓘ 该表不可编辑。

	sector_id	height
1	15500-128	20

在第一个窗口创建 read committed 隔离级别下的事务 T1

SQL查询 X SQL查询 X

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL v

```
1 START TRANSACTION ISOLATION LEVEL read committed;
```

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)】

START TRANSACTION ISOLATION LEVEL read committed;

执行成功, 耗时: [6ms.]

在第二个窗口创建 read committed 隔离级别下的事务 T2

SQL查询 X SQL查询 X

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Et

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL v

```
1 START TRANSACTION ISOLATION LEVEL read committed;
```

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)】

START TRANSACTION ISOLATION LEVEL read committed;

执行成功, 耗时: [7ms.]

在事务 T1 中, 将 SECTOR\_ID 为“15500-128”的 HEIGHT 修改为 6

SQL查询

SQL查询

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```
1 update tbcell_1
2 set HEIGHT = 6
3 where SECTOR_ID = '15500-128';
```

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)】  
update tbcell\_1  
set HEIGHT = 6  
where SECTOR\_ID = '15500-128';  
执行成功, 耗时: [7ms.]

在事务 T2 中，查询 SECTOR\_ID 为“15500-128”的 HEIGHT 值;

Gauss)

SQL操作 库管理 导入导出 账号管理

SQL查询

SQL查询

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```
1 select SECTOR_ID,HEIGHT
2 from tbcell_1
3 where SECTOR_ID = '15500-128';
4
```

SQL执行记录

消息

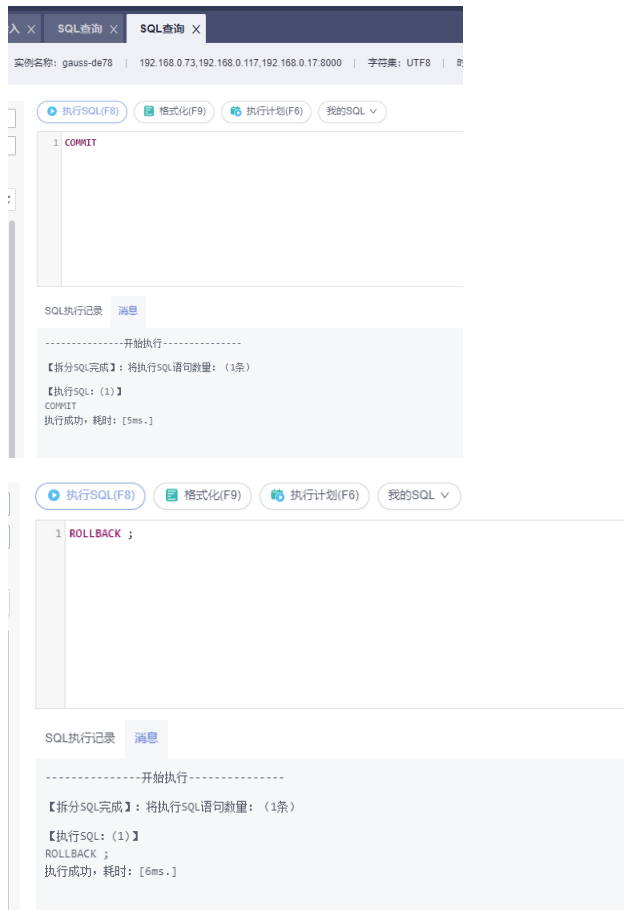
结果集1

以下是select SECTOR\_ID,HEIGHT from tbcell\_1 where SECTOR\_ID = '15500-128'的执行结果集

该表不可编辑。

	sector_id	height
1	15500-128	20

与之前查询的结果相同，都为 20。还未提交的事务 T1 对 HEIGHT 的更改并没有影响之后事务 T2 的查询结果，这样若事务 T1 之后回滚，事务 T2 也不会读到脏数据  
将事务 T2 提交，并将事务 T1 回滚



分析:

read committed 隔离级别下只能读到已经提交的数据, 无法读到未提交的数据, 不会出现脏读。

## (2) 不可重复读

Step1.创建 read committed 隔离级别下的并发事务 T1 和 T2;

Step2.在事务 T1 中, 查询 SECTOR\_ID 为“124672-O”的 HEIGHT 值;

Step3.在事务 T2 中, 将 SECTOR\_ID 为“124672-O”的 HEIGHT 修改为 6, 并 COMMIT 提交事务 T2;Step4.在事务 T1 中, 查询 SECTOR\_ID 为“124672-O”的 HEIGHT 值, 并 COMMIT 提交事务 T1;

要求:

观察 Step2 和 Step4 两次查询中的 HEIGHT 值是否相等, 说明是否为不可重复读;

在第一个窗口创建 read committed 隔离级别下的事务 T1





在第二个窗口创建 read committed 隔离级别下的事务 T2



在事务 T1 中，查询 SECTOR\_ID 为“15500-128”的 HEIGHT 值;

SQL查询 X SQL查询 X

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17-8000 | 字符集: UTF8 | 时区: Etc/GMT-8

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```
1 select SECTOR_ID,HEIGHT
2 from tbcell_1
3 where SECTOR_ID = '15500-128';
4
```

SQL执行记录 消息 结果集1 X

以下是select SECTOR\_ID,HEIGHT from tbcell\_1 where SECTOR\_ID = '15500-128'的执行结果集

	sector_id	height
1	15500-128	20

在事务 T2 中，将 SECTOR ID 为“15500-128”的 HEIGHT 修改为 6，并 COMMIT 提交事务 T2

SQL查询 X SQL查询 X

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17-8000 | 字符集: UTF8 | 时区: Etc/GMT-8

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```
1 update tbcell_1
2 set HEIGHT = 6
3 where SECTOR_ID='15500-128';
4 commit;
5
```

SQL执行记录 消息

-----开始执行-----  
【拆分SQL完成】：将执行SQL语句数量：（2条）  
【执行SQL：（1）】  
update tbcell\_1  
set HEIGHT = 6  
where SECTOR\_ID='15500-128';  
执行成功，耗时：[6ms.]  
【执行SQL：（2）】  
commit;  
执行成功，耗时：[10ms.]

在事务 T1 中，再次查询 SECTOR\_ID 为“15500-128”的 HEIGHT 值并 COMMIT 提交事务 T1

SQL查询 X SQL查询 X

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17-8000 | 字符集: UTF8 | 时区: Etc/GMT-8

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```
1 select SECTOR_ID,HEIGHT
2 from tbcell_1
3 where SECTOR_ID = '15500-128';
4 COMMIT ;
5
```

SQL执行记录 消息 结果集1 X

以下是select SECTOR\_ID,HEIGHT from tbcell\_1 where SECTOR\_ID = '15500-128'的执行结果集

	sector_id	height
1	15500-128	6

两次查询的值不一样，第二次查询的值为 6，与之前提交的事务 T2 中更改的值相同  
分析:

受到其它已经提交的事务的影响，不同的时刻读到的同一批数据不一样。该例中，T1 事务的查询操作受到之前已经提交的 T2 事务的数据更改操作的影响，使得两次对同一个数据的查询结果不一样。read committed 隔离级别下可能出现不可重复读。

### (3) 幻读

Step1.创建 read committed 隔离级别下的并发事务 T1 和 T2;

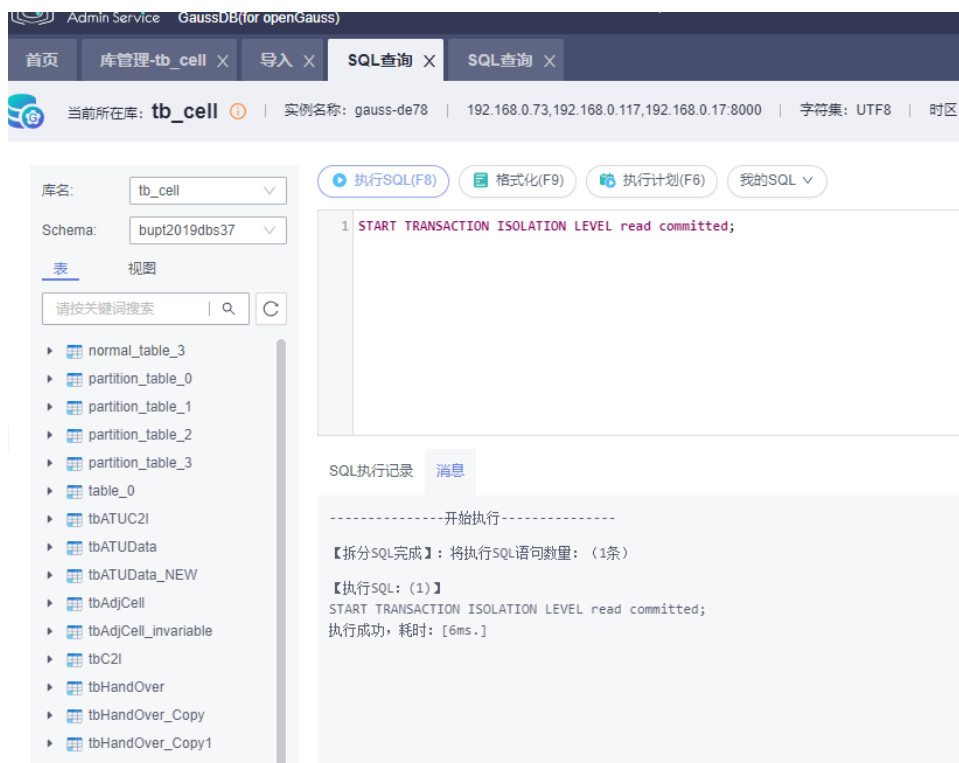
Step2.在事务 T1 中，查询 SECTOR\_ID 在“124672-0”与“124672-4”之间的元组;Step3.在事务 T2 中，插入 SECTOR\_ID 为“124672-3”的元组，并 COMMIT 提交事务 T2;

比如: values('name\_1,124672-3, 'name\_2',0,'name\_3',0,0,0,0,0,'name\_4',0,0,'name\_5',0,0,0,0,0)

Step4.在事务 T1 中，查询在“124672-0”与“124672-4”之间的元组，并 COMMIT 提交事务 T1;  
要求:

观察 Step2 和 Step4 两次查询到的元组是否相同，事务 T1 在 Step4 查询到的元组是否包含事务 T2 中插入的元组，说明是否为幻读;

在第一个窗口创建 read committed 隔离级别下的事务 T1



在第二个窗口创建 read committed 隔离级别下的事务 T2

库管理tb\_cell × 导入 × SQL查询 × SQL查询 ×

当前所在库: tb\_cell | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-4

tb\_cell

ma: bupt2019dbs37

视图

安关键词搜索

normal\_table\_3  
partition\_table\_0  
partition\_table\_1  
partition\_table\_2  
partition\_table\_3  
table\_0  
tbATUC2I  
tbATUData  
tbATUDData\_NEW  
tbAdjCell  
tbAdjCell\_invariable  
tbC2I  
tbHandOver

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL

1 START TRANSACTION ISOLATION LEVEL read committed;

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)】  
START TRANSACTION ISOLATION LEVEL read committed;  
执行成功, 耗时: [6ms.]

在事务 T1 中，查询 SECTOR\_ID 在“15500-127”与“15500-130”之间的元组

select \*

from tbcell\_1

where sector\_id between '15500-127' and '15500-130';

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL

SQL提示 全屏

1 select \*  
2 from tbcell\_1  
3 where sector\_id between '15500-127' and '15500-130';  
4

SQL执行记录 消息 结果集1 X

以下select "from tbcell\_1 where sector\_id between '15500-127' and '15500-130'"的执行结果表 该表不可编辑。

复制行 复制列 列设

	city	sector_id	sector_name	enodebid	enodeb_name	earfcn	pci	psn
1	saxxia	15500-128	C氏河西村-HLHF-1	15500	C氏河西村-HLHF	38400	475	1
2	saxxia	15500-129	C氏河西村-HLHF-2	15500	C氏河西村-HLHF	38400	476	2
3	saxxia	15500-130	C氏河西村-HLHF-3	15500	C氏河西村-HLHF	38400	474	0

共有 3 行数据

在事务 T2 中，插入 SECTOR\_ID 为“15500-127”的元组，并 COMMIT 提交事务 T2

INSERT INTO tbcell\_1

values('name\_1','15500-127','name\_2',0,'name\_3',0,0,0,0,0,'name\_4',0,0, 'name\_5',0,0,0,0,0);

commit;

Admin Service GaussDB(for openGauss)

库管理-tb\_cell x 导入 x SQL查询 x SQL查询 x

当前所在库: tb\_cell | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell Schema: bupt2019dbs37

表 视图

请按关键词搜索 | 搜索

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUDData
- tbATUDData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I
- tbHandOver
- tbHandOver\_Copy
- tbHandOver\_Copy1
- tbHandOver\_invariable
- tbMROData

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL v

```
1 INSERT INTO tbcell_1
2 values('name_1',15500-127,'name_2',0,'name_3',0,0,0,0,0,'name_4',0,0, 'name_5',0,0,0,0,0);
3 commit;
4
```

SQL执行记录 消息

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（2条）

【执行SQL：（1）】  
INSERT INTO tbcell\_1  
values('name\_1',15500-127,'name\_2',0,'name\_3',0,0,0,0,0,'name\_4',0,0, 'name\_5',0,0,0,0,0);  
执行成功，耗时：[6ms.]

【执行SQL：（2）】  
commit;  
执行成功，耗时：[9ms.]

在事务 T1 中，再次查询 SECTOR ID 在“15500-127”与“15500-130”之间的元组,并 COMMIT 提交事务

```
select *
from tbcell_1
where sector_id between '15500-127' and '15500-130';
commit;
```

当前所在库: tb\_cell | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区:

库名: tb\_cell | Schema: bup2019dbs37

SQL 语句:

```
1 select *
2 from tbcell_1
3 where sector_id between '15500-127' and '15500-130';
4 commit;
```

SQL 执行记录 消息 结果集 1 x

-----开始执行-----

【拆分SQL完成】: 将执行SQL语句数量: (2条)

【执行SQL: (1)】

```
select *
from tbcell_1
where sector_id between '15500-127' and '15500-130';
```

执行成功, 当前返回: [4]行, 耗时: [7ms.]

【执行SQL: (2)】

```
commit;
```

执行成功, 耗时: [6ms.]

查询结果一共有 4 行, 多出的一行数据为在已经提交的事务 T2 中插入的新元组

SQL 语句:

```
1 select *
2 from tbcell_1
3 where sector_id between '15500-127' and '15500-130';
4 commit;
```

SQL 执行记录 消息 结果集 1 x

以下select \* from tbcell\_1 where sector\_id between '15500-127' and '15500-130'的执行结果

city	sector_id	sector_name	enodeb_id	enodeb_name	earfcn	pci	gsm
name_1	15500-127	name_2	0	name_3	0	0	0
saxia	15500-128	C氏河西村-HLHF-1	15500	C氏河西村-HLHF	38400	475	1
saxia	15500-129	C氏河西村-HLHF-2	15500	C氏河西村-HLHF	38400	476	2
saxia	15500-130	C氏河西村-HLHF-3	15500	C氏河西村-HLHF	38400	474	0

分析:

在事务 T1 第一次查询以后, 事务 T1 还未提交, 事务 T2 就插入了一条在事务 T1 的查询范围内的元组, 并且事务 T2 提交了, 之后事务 T1 在进行一遍相同的查询, 受事务 T2 的插入操作的影响, 查询结果多出了一个数据。read committed 隔离级别下可能出现幻读。

## 2.2 repeatable read 隔离级别下的脏读, 不可重复读, 幻读

针对小区/基站信息表 tbCell(注意备份原表), 在 repeatable read 隔离级别下, 在并发事务内部进行数据的查询, 更新, 添加等, 观察并发事务之间的影响, 验证 repeatable read 隔离级别下是否存在脏读, 不可重复读, 幻读。

(1)脏读

Step1.用隐式的独立事务查询 SECTOR\_ID 为“124673-O”的 HEIGHT 值;

Step2.创建 repeatable read 隔离级别下的并发事务 T1 和 T2;

Step3.在事务 T1 中, 将 SECTOR\_ID 为“124673-O”的 HEIGHT 修改为 6;

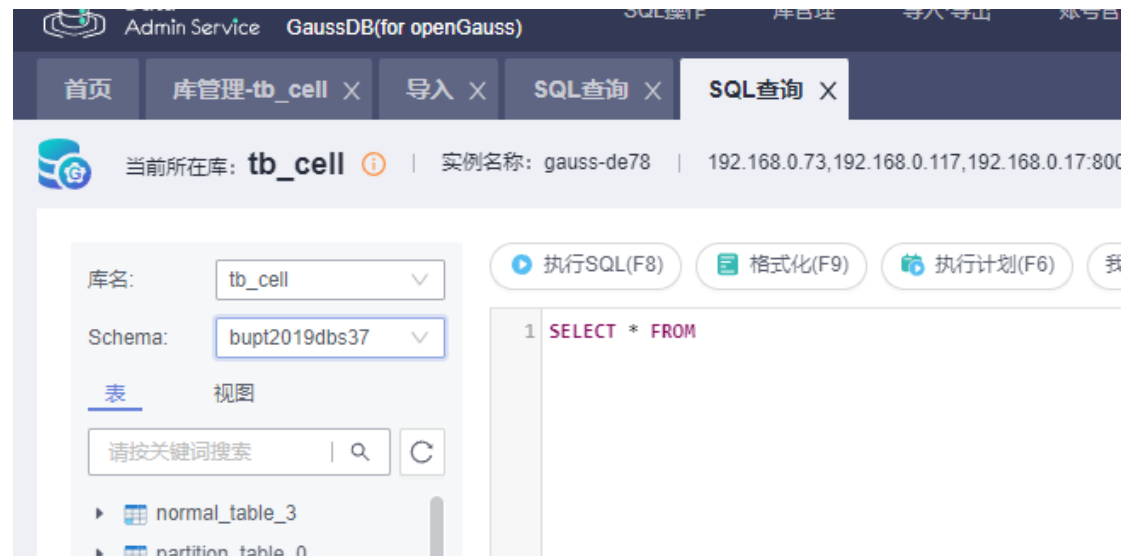
Step4.在事务 T2 中，查询 SECTORLID 为“124673-O”的 HEIGHT 值，并 COMMIT 提交事务 T2;

Step5.用 ROLLBACK 回滚事务 T1;

要求:

观察 Step4 的 HEIGHT 值，与 Step1 和 Step3 的 HEIGHT 值进行对比  
说明事务 T2 是否读取了事务 T1 未提交的数据，是否存在脏读;

先打开两个查询，并连接相同的数据库



查询 SECTOR\_ID 为“15501-128”的 HEIGHT 值

select SECTOR\_ID,HEIGHT

from tbcell\_3

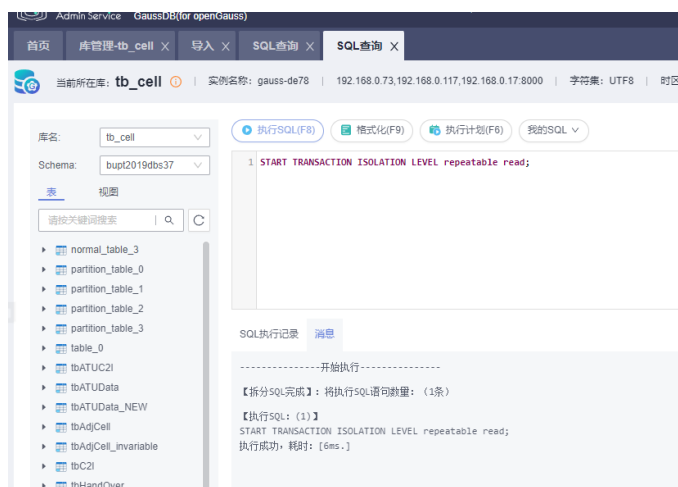
where SECTOR\_ID='15501-128';



在第一个窗口创建 repeatable read 隔离级别下的事务 T1

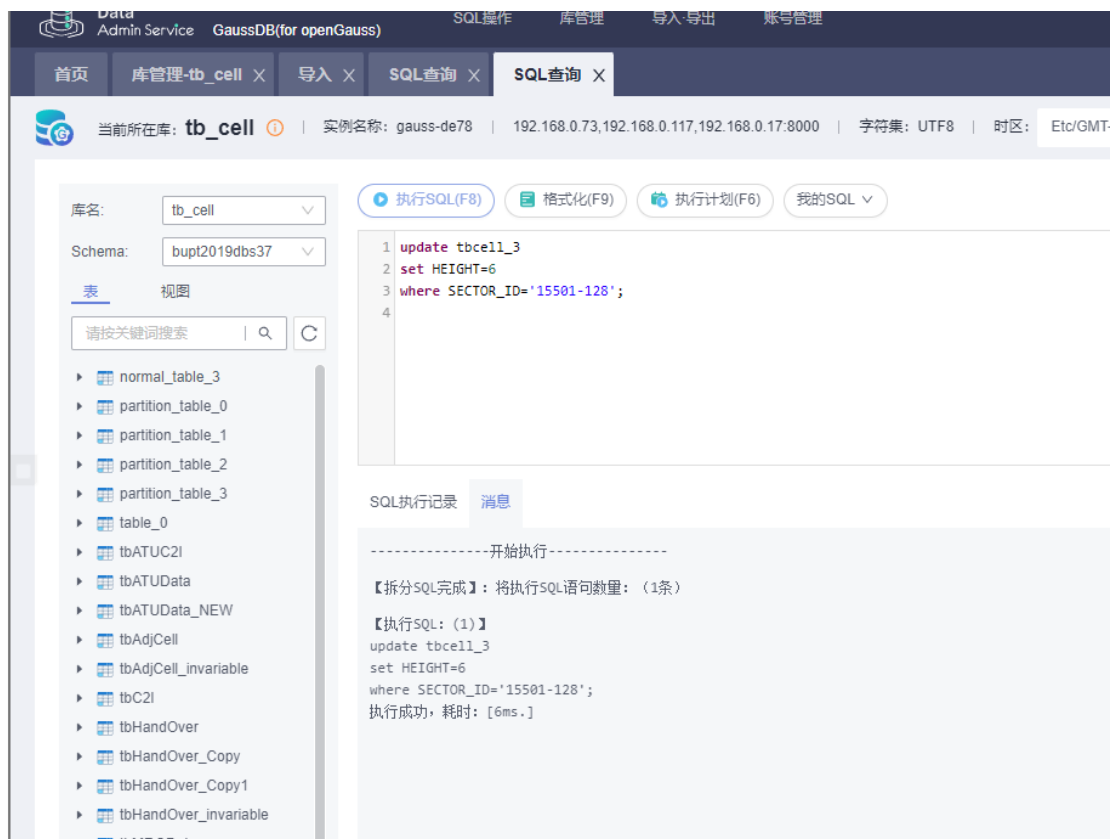


在第二个窗口创建 repeatable read 隔离级别下的事务 T2



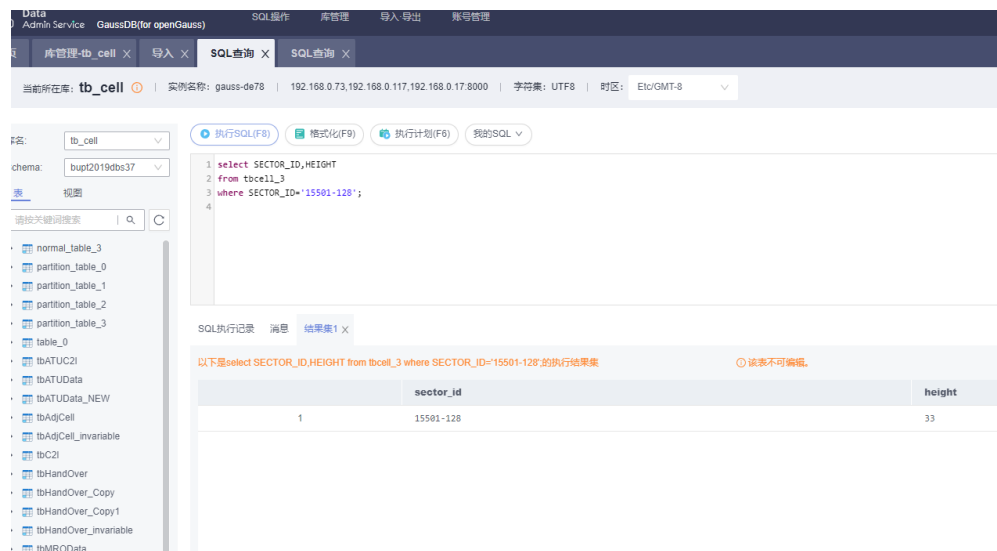
在事务 T1 中, 将 SECTOR\_ID 为“15501-128”的 HEIGHT 修改为 6;  
update tbccll\_3  
set HEIGHT=6  
where SECTOR\_ID='15501-128';





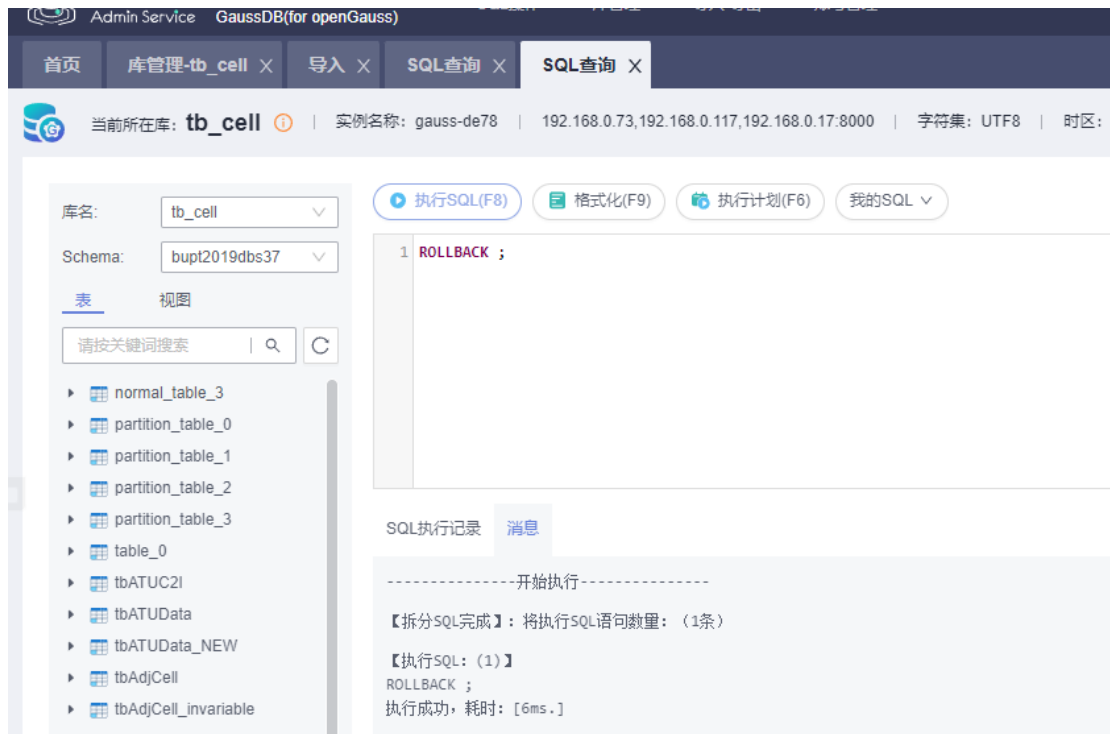
在事务 T2 中，查询 SECTOR\_ID 为“15500-128”的 HEIGHT 值;

```
select SECTOR_ID,HEIGHT
from tbcell_3
where SECTOR_ID='15501-128';
```



与之前查询的结果相同，都为 33。还未提交的事务 T1 对 HEIGHT 的更改并没有影响之后事务 T2 的查询结果,这样若事务 T1 之后回滚，事务 T2 也不会读到脏数据。

将事务 T2 提交，并将事务 T1 回滚



分析:

repeatable read 隔离级别下只能读到已经提交的数据，无法读到未提交的数据，不会出现脏读。

## (2) 不可重复读

Step1.创建 repeatable read 隔离级别下的并发事务 T1 和 T2;

Step2.在事务 T1 中，查询 SECTOR\_ID 为“124673-O”的 HEIGHT 值;

Step3.在事务 T2 中，将 SECTOR ID 为“124673-O”的 HEIGHT 修改为 6，并 COMMIT 提交事务 T2;

Step4.在事务 T1 中，查询 SECTOR\_ID 为“124673-O”的 HEIGHT 值，并 COMMIT 提交事务 T1;

要求:

观察 Step2 和 Step4 两次查询中的 HEIGHT 值是否相等，说明是否为不可重复读;

在第一个窗口创建 repeatable read 隔离级别下的事务 T1

当前所在库: **tb\_cell** | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bupt2019dbs37

表 | 视图

请按关键词搜索 | 搜索 | 清除

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUDData
- tbATUDData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I
- tbHandOver
- tbHandOver\_Conv

执行SQL(F8) | 格式化(F9) | 执行计划(F6) | 我的SQL v

```
1 START TRANSACTION ISOLATION LEVEL repeatable read;
```

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)】  
START TRANSACTION ISOLATION LEVEL repeatable read;  
执行成功, 耗时: [6ms.]

在第二个窗口创建 repeatable read 隔离级别下的事务 T2

Admin Service GaussDB(for openGauss)

当前所在库: **tb\_cell** | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bupt2019dbs37

表 | 视图

请按关键词搜索 | 搜索 | 清除

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUDData
- tbATUDData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I
- tbHandOver

执行SQL(F8) | 格式化(F9) | 执行计划(F6) | 我的SQL v

```
1 START TRANSACTION ISOLATION LEVEL repeatable read;
```

SQL执行记录 | 消息

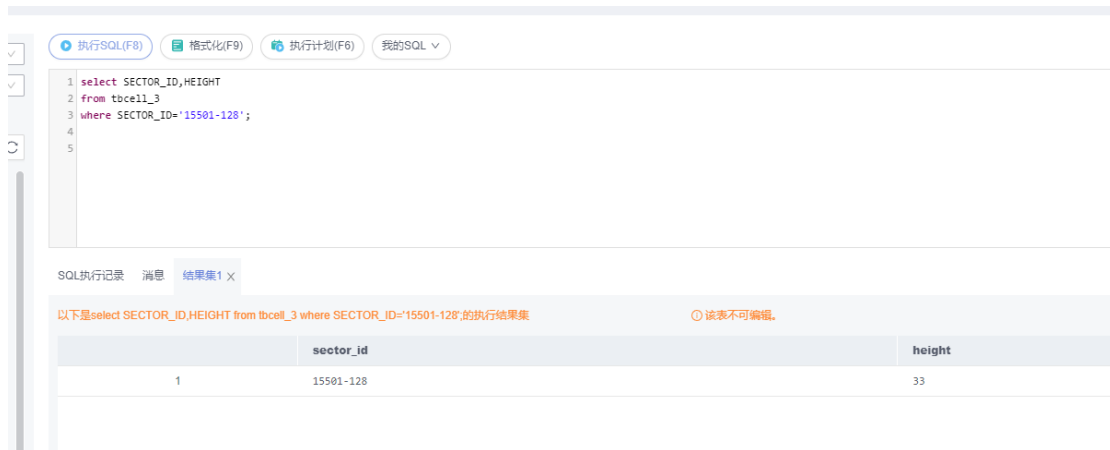
-----开始执行-----

【拆分SQL完成】: 将执行SQL语句数量: (1条)

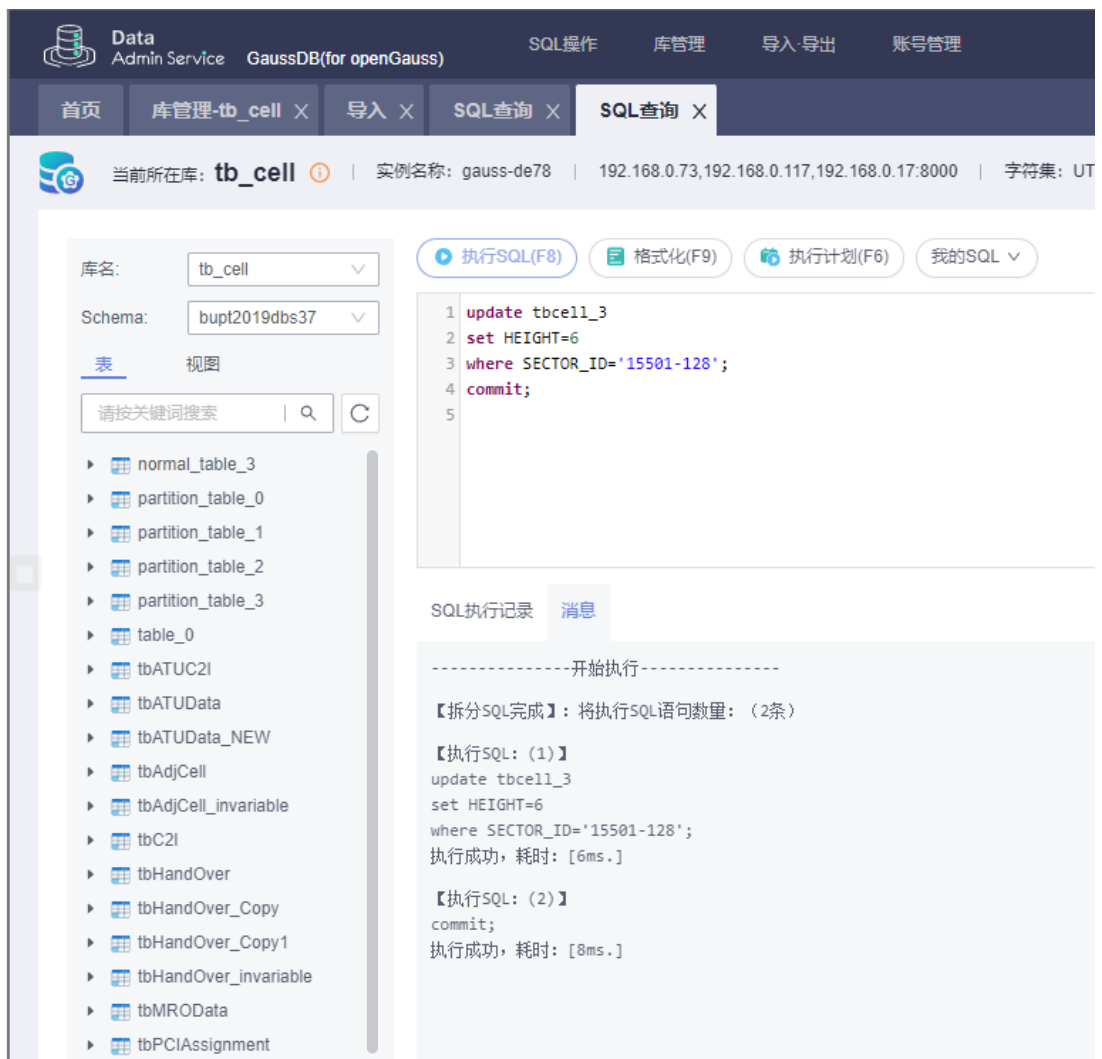
【执行SQL: (1)】  
START TRANSACTION ISOLATION LEVEL repeatable read;  
执行成功, 耗时: [6ms.]

在事务 T1 中, 查询 SECTOR\_ID 为“15501-128”的 HEIGHT 值;

```
select SECTOR_ID,HEIGHT  
from tbcell_3  
where SECTOR_ID='15501-128';
```



在事务 T2 中，将 SECTOR\_ID 为“15501-128”的 HEIGHT 修改为 6，并 COMMIT 提交事务 T2



在事务 T1 中，再次查询 SECTOR\_ID 为“15500-128”的 HEIGHT 值，并 COMMIT 提交事务 T1

```
select SECTOR_ID,HEIGHT
from tbcell_3
where SECTOR_ID='15501-128';
COMMIT ;
```

The screenshot shows a database management interface with a sidebar on the left listing various tables and schemas. The main area displays a SQL query in a text editor:

```
1 select SECTOR_ID, HEIGHT
2 from tbc11_3
3 where SECTOR_ID='15501-128';
4 COMMIT ;
5
6
```

Below the query editor, the results are displayed in a table format. The table has two columns: **sector\_id** and **height**. The results show a single row with **sector\_id** 1 and **height** 33.

sector_id	height
1	33

两次查询的值相同，都为 33，前面已经提交的事务 T2 对 HEIGHT 值的修改未影响事务 T1 的查询

repeatable read 隔离级别下，一个事务仅仅看到本事务开始之前提交的数据，它不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。该例中，T1 事务只能看到在该事务开始之前提交的数据，而 T2 事务是在 T1 事务执行期间提交的，则 T1 事务无法看到 T2 事务修改的数据，T1 事务的查询结果不会发生变化。repeatable read 隔离级别下不会出现不可重复读。

### (3) 幻读

Step1.创建 repeatable read 隔离级别下的并发事务 T1 和 T2;

Step2.在事务 T1 中，查询 SECTOR\_ID 在“124673-0”与“124673-4”之间的元组;Step3.在事务 T2 中，插入 SECTOR\_ID 为“124673-3”的元组，并 COMMIT 提交事务 T2;

比如: values('name\_1;124673-3'; 'name\_2',0,'name\_3',0,0,0,0,0 'name\_4,0,0,'name\_5',0,0,0,0,0)

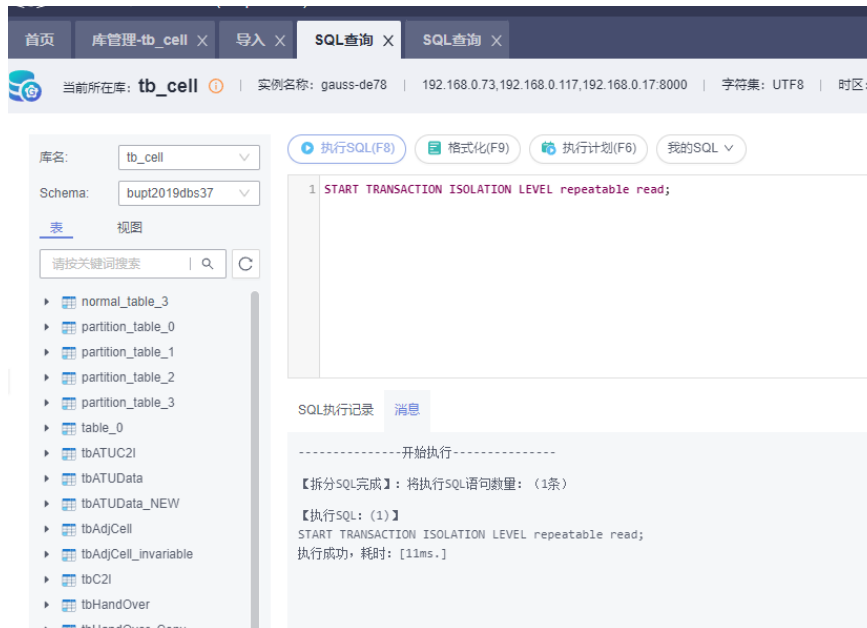
Step4.在事务 T1 中，查询在“124673-0”与“124673-4”之间的元组，并 COMMIT 提交事务 T1; 要求:

观察 Step2 和 Step4 两次查询到的元组是否相同，事务 T1 在 Step4

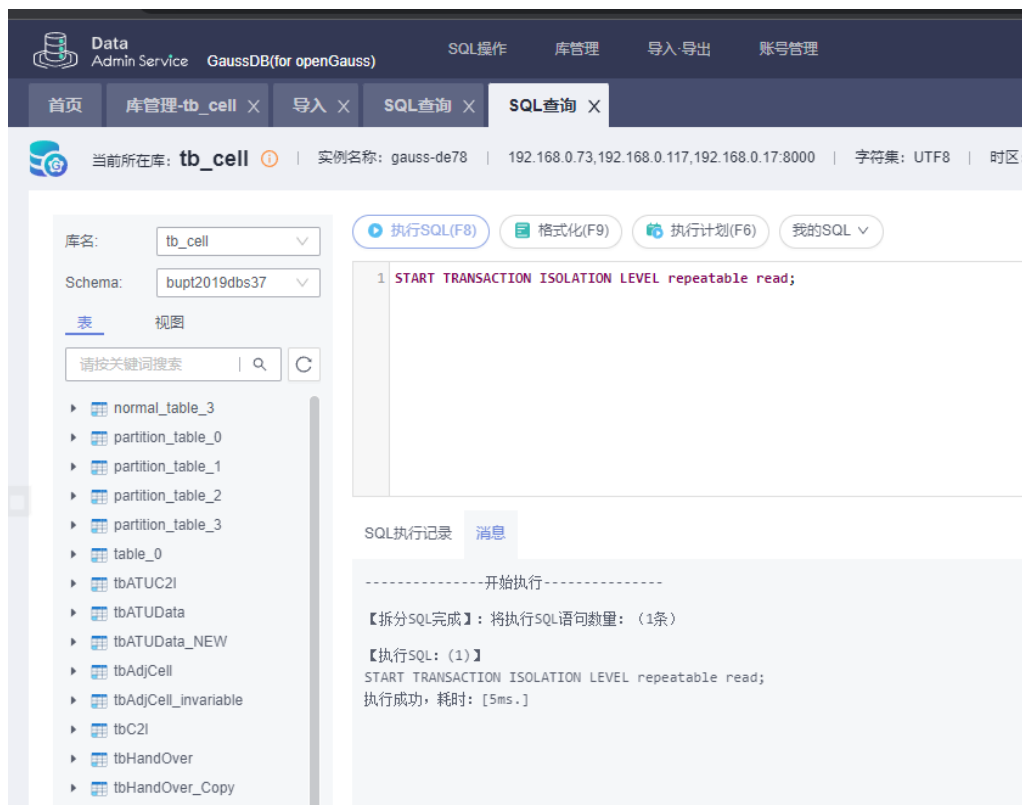
查询到的元组是否包含事务 T2 中插入的

元组，说明是否为幻读;

在第一个窗口创建 repeatable read 隔离级别下的事务 T1



在第二个窗口创建 repeatable read 隔离级别下的事务 T2



在事务 T1 中，查询 SECTOR\_ID 在“15501-127”与“15501-130”之间的元组  
select \*  
from tbcell\_3  
where sector\_id between '15501-127' and '15501-130';

当前所在库: **tb\_cell** | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bupl2019dbs37

SQL 查询

```
1 select *
2 from tbcell_3
3 where sector_id between '15501-127' and '15501-130';
```

SQL 执行记录 消息 结果集 1 X

以下是select \* from tbcell\_3 where sector\_id between '15501-127' and '15501-130'的执行结果集

	city	sector_id	sector_name	enodebid	enode
1	saxxia	15501-128	E宝徐营村-HLHF-1	15501	E宝徐营
2	saxxia	15501-129	E宝徐营村-HLHF-2	15501	E宝徐营
3	saxxia	15501-130	E宝徐营村-HLHF-3	15501	E宝徐营

一共有 3 行数据

在事务 T2 中，插入 SECTOR\_ID 为“15501-127”的元组，并 COMMIT 提交事务 T2

INSERT INTO tbcell\_3

values('name\_1', '15501-127', 'name\_2',0,'name\_3',0,0,0,0,0,'name\_4',0,0,'name\_5',0,0,0,0,0);  
commit;

当前所在库: **tb\_cell** | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bupl2019dbs37

SQL 查询

```
1 INSERT INTO tbcell_3
2 values('name_1', '15501-127', 'name_2',0,'name_3',0,0,0,0,0,'name_4',0,0,'name_5',0,0,0,0,0);
3 commit;
```

SQL 执行记录 消息

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（2条）

【执行SQL：（1）】  
INSERT INTO tbcell\_3  
values('name\_1', '15501-127', 'name\_2',0,'name\_3',0,0,0,0,0,'name\_4',0,0,'name\_5',0,0,0,0,0);  
执行成功，耗时：[7ms.]

【执行SQL：（2）】  
commit;  
执行成功，耗时：[6ms.]

在事务 T1 中，再次查询 SECTOR ID 在“15500-127”与“15500-130”之间的元组,并 COMMIT 提交事务

select \*

from tbcell\_3

where sector\_id between '15501-127' and '15501-130';

COMMIT ;

当前所在库: tb\_cell | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bupt2019dbs37

表 | 视图

请按关键词搜索

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUDData
- tbATUDData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I
- tbHandOver
- tbHandOver\_Copy
- tbHandOver\_Copy1
- tbHandOver\_invariable

```
1 select *
2 from tbcell_3
3 where sector_id between '15501-127' and '15501-130';
4 COMMIT ;
```

SQL执行记录 消息 结果集1 X

-----开始执行-----

【拆分SQL完成】: 将执行SQL语句数量: (2条)

【执行SQL: (1)】

```
select *
from tbcell_3
where sector_id between '15501-127' and '15501-130';
```

执行成功, 当前返回: [3]行, 耗时: [7ms.]

【执行SQL: (2)】

```
COMMIT ;
```

执行成功, 耗时: [5ms.]

查询结果一共有 3 行, 与之前的查询结果相同

首页 | 库管理-tb\_cell X | 导入 X | SQL查询 X | SQL查询 X

当前所在库: tb\_cell | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bupt2019dbs37

表 | 视图

请按关键词搜索

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUDData
- tbATUDData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I
- tbHandOver
- tbHandOver\_Copy
- tbHandOver\_Copy1
- tbHandOver\_invariable

```
1 select *
2 from tbcell_3
3 where sector_id between '15501-127' and '15501-130';
4 COMMIT ;
```

SQL执行记录 消息 结果集1 X

以下是select \* from tbcell\_3 where sector\_id between '15501-127' and '15501-130'的执行结果集

	city	sector_id	sector_name	enodebid	e
1	saxxia	15501-128	E宝徐营村-HLHF-1	15501	E
2	saxxia	15501-129	E宝徐营村-HLHF-2	15501	E
3	saxxia	15501-130	E宝徐营村-HLHF-3	15501	E

分析:

该例中, T1 事务只能看到在该事务开始之前提交的数据, 而 T2 事务是在 T1 事务执行期间提交的, 则 T1 事务无法看到 T2 事务插入的数据, T1 事务的查询结果不会产生变化。repeatable read 隔离级别下不会出现幻读。



## 3.事务锁机制

### 3.1 死锁分析

在 openGauss 中，当两个或以上的事务相互持有和请求锁，并形成一个循环的依赖关系，就会产生死锁。多个事务同时锁定同一个资源时，也可能产生死锁。死锁无法完全避免的，数据库会自动检测事务死锁，立即回滚其中某个事务，并且返回一个错误。它根据某种机制来选择回滚代价最小的事务来进行回滚。

以下两种情况可能导致并发事务死锁:

(1)在 REPEATABLE-READ 隔离级别下，如果两个事务同时对关系表中满足 where 条件的元组记录用 select-from-where-for-update 加互斥锁，在没有符合条件记录的情况下，两个事务都会加锁成功。当 2 个事务发现满足 where 条件的元组记录尚不存在，都试图插入一条新记录，就会出现死锁。这种情况下，将隔离级别改成 READ COMMITTED，可以避免死锁问题。

(2)当隔离级别为 READ COMMITTED 时，如果两个并发事务都先执行 select-from-where-for-update，判断是否存在符合 where 条件的记录，如果没有，就插入一条记录。此时，只有一个事务能插入成功，另一个事务会出现锁等待，当第 1 个事务提交后，第 2 个事务会因主键重合出错。但虽然这个事务出错，却会获得一个互斥锁，这时如果有第 3 个事务又来申请互斥锁，也会出现死锁。对于这种情况，可以直接做插入操作，然后再捕获主键重异常，或者在遇到主键重合错误时，总是执行 ROLLBACK 释放获得的互斥锁。

#### 示例 1.隔离级别 read-repeatable 下死锁

在 session1 中开启事务，执行 select for update 语句，该语句对表中符合条件的元组/数据行加上互斥锁。

```
START TRANSACTION ISOLATION LEVEL repeatable read;  
select SECTOR_NAME from tbcell_1 where HEIGHT=43 for update;
```

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

1

START TRANSACTION ISOLATION LEVEL repeatable read;

2

select SECTOR\_NAME from tbcell\_1 where HEIGHT=43 for update;

3

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）】

START TRANSACTION ISOLATION LEVEL repeatable read;

执行成功，耗时：[6ms.]

【执行SQL：（2）】

select SECTOR\_NAME from tbcell\_1 where HEIGHT=43 for update;

执行成功，当前返回：[50]行，耗时：[15ms.]

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

1

START TRANSACTION ISOLATION LEVEL repeatable read;

2

select SECTOR\_NAME from tbcell\_1 where HEIGHT=43 for update;

3

SQL执行记录

消息

结果集1

以下是select SECTOR\_NAME from tbcell\_1 where HEIGHT=43 for update;的执行结果集

该表不可编辑。

	sector_name
1	A池刘果-HLHF-2
2	A池刘果-HLHF-3
3	A池张沟村-HLHF-1
4	A池张沟村-HLHF-2
5	A池张沟村-HLHF-3
6	A池苏门-HLHF-1
7	A池南涧-HLHF-1
8	A池南涧-HLHF-3
9	A池峪洞-HLHF-1

在 session2 中开启一个事务，也是执行上述 select for update 语句。此时检测到符合筛选条件的部分数据行上加了互斥锁(两次查询的数据行有重叠，存在 PCI=32 且 HEIGHT=43 的数据行)，查询进入申请锁的等待阶段。

```
START TRANSACTION ISOLATION LEVEL repeatable read;
select SECTOR_NAME from tbcell_1 where PCI=32 for update;
```

当前所在库: tb\_cell | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17-8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bup2019dbs37

SQL执行记录 消息

执行时间	SQL语句	执行结果
2022-01-10 01:55:14	commit;	执行成功
2022-01-10 01:55:14	INSERT INTO tbcell_2 values('name');	执行成功
2022-01-10 01:55:13	set time zone 'Etc/GMT-8';	执行成功
2022-01-10 01:53:48	select * from tbcell_3 where sector;	执行成功
2022-01-10 01:53:47	set time zone 'Etc/GMT-8';	6 ms 执行成功
2022-01-10 01:53:36	START TRANSACTION ISOLATION LEVEL repeatable read;	6 ms 执行成功
2022-01-10 01:53:35	set time zone 'Etc/GMT-8';	5 ms 执行成功
2022-01-10 01:53:15	select * from tbcell_2 where sector_id between '15501-127' and '15501-130';	6 ms 执行失败
2022-01-10 01:53:14	set time zone 'Etc/GMT-8';	5 ms 执行成功

请等待

正在执行SQL

select SECTOR\_NAME from tbcell\_1 where PCI=32 for update;

取消执行

若在等待过程中，马上释放数据行的互斥锁，比如 COMMIT 提交 session1 上占用数据行互斥锁的事务，则查询正常进行，此查询的业务对符合查询条件的数据行加入新的互斥锁。

当前所在库: tb\_cell | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17-8000 | 字符集: UTF8 | 时区: Etc/GMT-8

库名: tb\_cell | Schema: bup2019dbs37

SQL执行记录 消息 结果集1 X

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（3条）

【执行SQL: (1)】  
START TRANSACTION ISOLATION LEVEL repeatable read;  
执行成功，耗时：[6ms.]

warning:  
there is already a transaction in progress

【执行SQL: (2)】  
select SECTOR\_NAME from tbcell\_1 where HEIGHT=43 for update;  
执行成功，当前返回：[50]行，耗时：[14ms.]

【执行SQL: (3)】  
commit;  
执行成功，耗时：[8ms.]

for openGauss)

SQL查询 x SQL查询 x

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL v

```
1 START TRANSACTION ISOLATION LEVEL repeatable read;
2 select SECTOR_NAME from tbcell_1 where PCI=32 for update;
3
```

SQL执行记录 消息 结果集1 x

以下是select SECTOR\_NAME from tbcell\_1 where PCI=32 for update;的执行结果集 ① 该表不

	sector_name
1	A池西村-HLHF-3
2	E宝三小-HLHF-2
3	黄委会-HLHD-1
4	陕州宾馆-HLHF-3
5	陕州宾馆-HLHD-3
6	C氏鑫源小区-HLHF-1
7	D县火车站-HLHF-3
8	烟草大厦(含烟草局办公楼)-HLWE-1
9	E宝巴楼-HLHF-1
10	

分析:

在本例中,session1 的业务先占用了符合 HEIGHT=43 的元组/数据行的互斥锁,之后 session2 的业务要申请符合 PCI=32 的元组/数据行的互斥锁时,由于存在 PCI=32 且 HEIGHT=43 的数据行,则在 session1 的业务释放符合 HEIGHT=43 的元组/数据行的互斥锁前,session2 的业务无法占用符合 PCI=32 的元组/数据行的互斥锁。因此,session2 的业务进入申请锁的等待阶段,这个等待阶段设定了最大的锁等待时间,超时后,系统会自动判定发生死锁,回滚当前事务。若在超时前,session1 的业务占用的互斥锁释放了,则 session2 的业务能够对符合 PCI=32 的元组/数据行加上互斥锁,session2 的业务查询便能正常进行了。

## 示例 2.加的互斥锁的粒度

在 session1 中开启事务,执行 select for update 语句,该语句对表中符合条件的元组/数据行加上互斥锁。

```
START TRANSACTION ISOLATION LEVEL repeatable read;
```

```
select SECTOR_NAME from tbcell_1 where SECTOR_ID='124672-1' for update;
```

库名: tb\_cell Schema: bupt2019dbs37

SQL查询

```
1 START TRANSACTION ISOLATION LEVEL repeatable read;
2 select SECTOR_NAME from tbcell1_1 where SECTOR_ID='124672-1' for update;
3
```

SQL执行记录 消息 结果集1 X

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（2条）

【执行SQL：（1）】  
START TRANSACTION ISOLATION LEVEL repeatable read;  
执行成功，耗时：[6ms.]

【执行SQL：（2）】  
select SECTOR\_NAME from tbcell1\_1 where SECTOR\_ID='124672-1' for update;  
执行成功，当前返回：[1]行，耗时：[6ms.]

库名: tb\_cell Schema: bupt2019dbs37

SQL查询

```
1 START TRANSACTION ISOLATION LEVEL repeatable read;
2 select SECTOR_NAME from tbcell1_1 where SECTOR_ID='124672-1' for update;
3
```

SQL执行记录 消息 结果集1 X

以下是select SECTOR\_NAME from tbcell1\_1 where SECTOR\_ID='124672-1' for update,的执行结果集

	sector_name
1	A池刘果-HLHF-2

该表不可编辑。

在 session2 中开启一个事务，也是执行上述 select for update 语句。

SQL查询 X

当前所在库: **tb\_cell** | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区:

库名: tb\_cell Schema: bupt2019dbs37

表 视图

请按关键词搜索

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUDData
- tbATUDData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I
- tbHandOver
- tbHandOver\_Copy

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL

```
1 START TRANSACTION ISOLATION LEVEL repeatable read;
2 select SECTOR_NAME from tbcell_1 where PC1=24 for update;
3
```

SQL执行记录 消息 结果集1 X

-----开始执行-----

【拆分SQL完成】：将执行SQL语句数量：（2条）

【执行SQL：（1）】  
START TRANSACTION ISOLATION LEVEL repeatable read;  
执行成功，耗时：[5ms.]

【执行SQL：（2）】  
select SECTOR\_NAME from tbcell\_1 where PC1=24 for update;  
执行成功，当前返回：[10]行，耗时：[7ms.]

正常查询，没有发生锁等待，说明 select for update 语句，不是对整表加互斥锁。

在 session2 上，继续执行上述 select for update 语句

select SECTOR\_NAME from tbcell\_1 where SECTOR\_ID='124672-2' for update;

Admin Service GaussDB(for openGauss)

SQL查询 X

当前所在库: **tb\_cell** | 实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc

库名: tb\_cell Schema: bupt2019dbs37

表 视图

请按关键词搜索

- normal\_table\_3
- partition\_table\_0
- partition\_table\_1
- partition\_table\_2
- partition\_table\_3
- table\_0
- tbATUC2I
- tbATUDData
- tbATUDData\_NEW
- tbAdjCell
- tbAdjCell\_invariable
- tbC2I

执行SQL(F8) 格式化(F9) 执行计划(F6) 我的SQL

```
1 select SECTOR_NAME from tbcell_1 where SECTOR_ID='124672-2' for update;
2
```

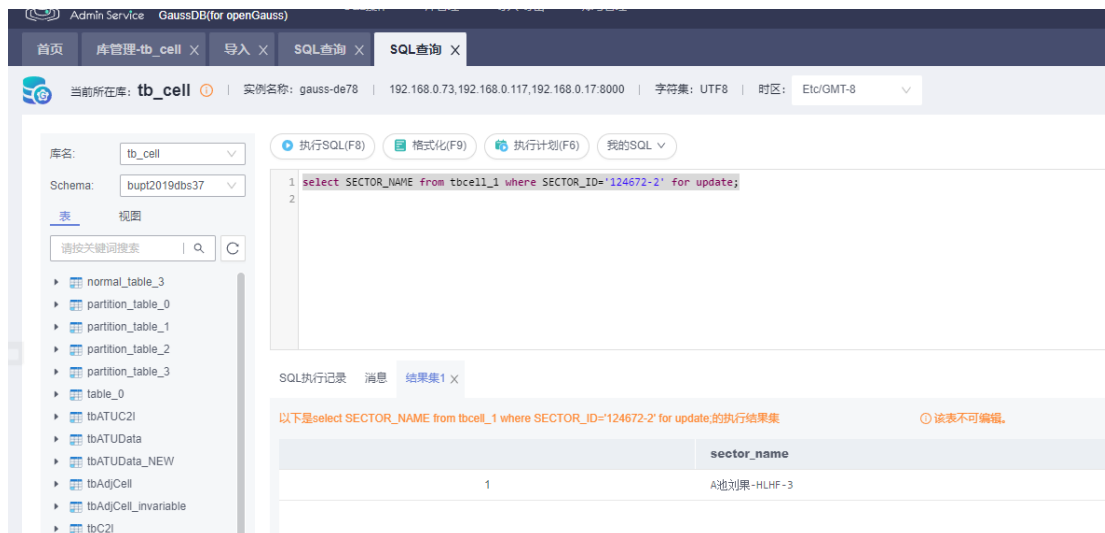
SQL执行记录 消息 结果集1 X

-----开始执行-----

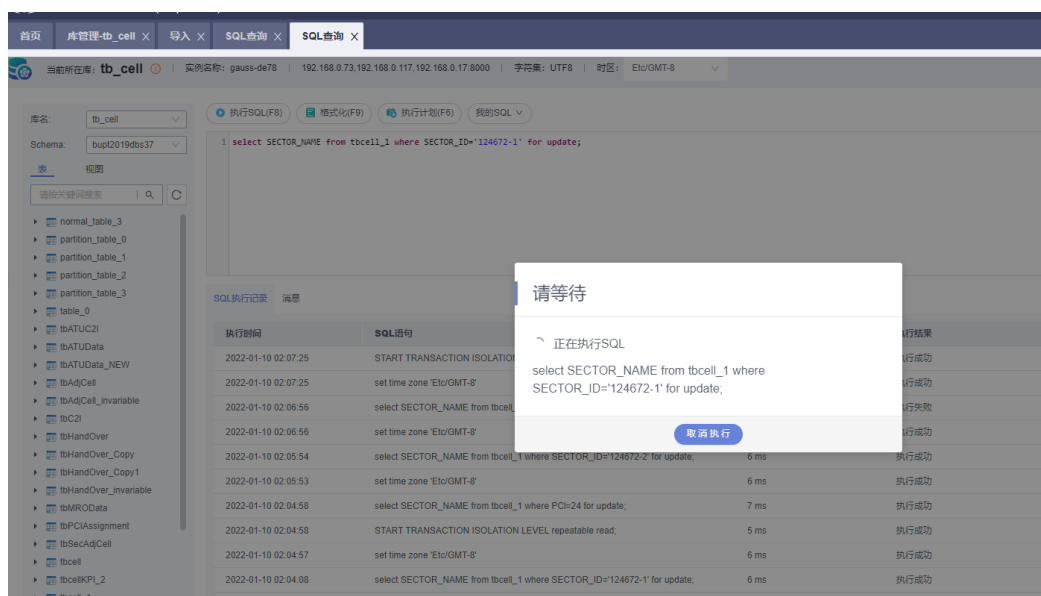
【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：（1）】  
select SECTOR\_NAME from tbcell\_1 where SECTOR\_ID='124672-2' for update;  
执行成功，当前返回：[1]行，耗时：[6ms.]

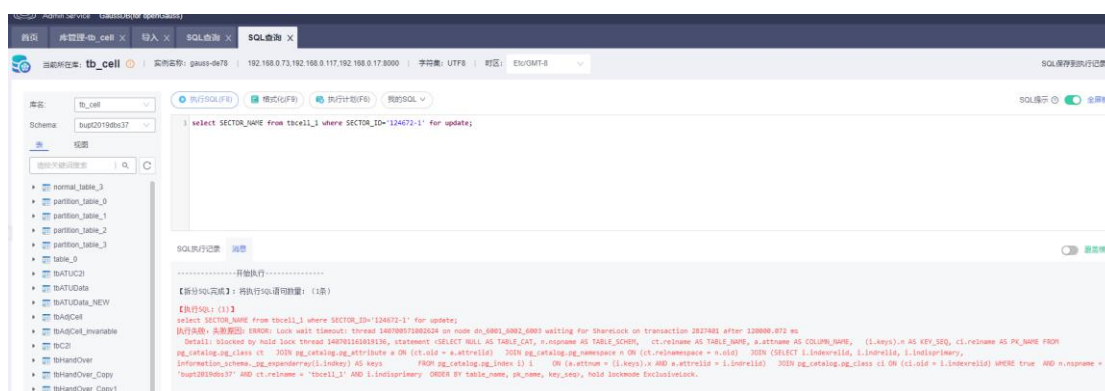
正常查询，没有发生锁等待，说明 select for update 语句，不是对查询条件所在的列加互斥锁。



在 session2 上，继续执行上述 select for update 语句，这次与 session1 上的查询条件一样 `select SECTOR_NAME from tbcell_1 where SECTOR_ID='124672-1' for update;`



发生锁等待，并且等待一段时间后，系统会自动判定发生死锁，回滚当前事务。



说明 select for update 语句，是对表中符合查询条件的元组/数据行加上互斥锁分析：  
select for update 语句，加入的互斥锁的粒度一般为行锁。

## 4 备份与恢复

数据备份是保护数据安全的重要手段之一，为了更好的保护数据安全, openGauss 数据库支持两种备份恢复类型、多种备份恢复方案，备份和恢复过程中提供数据的可靠性保障机制。备份与恢复类型可分为逻辑备份与恢复、物理备份与恢复。

- 逻辑备份与恢复：通过逻辑导出对数据进行备份，逻辑备份只能基于备份时刻进行数据转储，所以恢复时也只能恢复到备份时保存的数据。对于故障点和备份点之间的数据，逻辑备份无能为力，逻辑备份适合备份那些很少变化的数据，当这些数据因误操作被损坏时，可以通过逻辑备份进行快速恢复。如果通过逻辑备份进行全库恢复，通常需要重建数据库，导入备份数据来完成，对于可用性要求很高的数据库，这种恢复时间太长，通常不被采用。由于逻辑备份具有平台无关性，所以更为常见的是，逻辑备份被作为一个数据迁移及移动的主要手段。

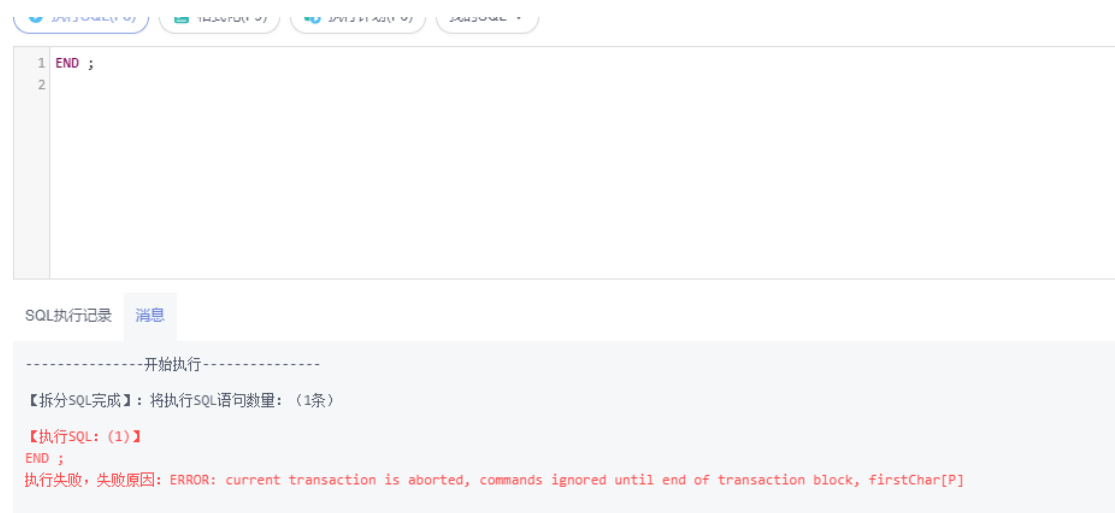
- 物理备份与恢复：通过物理文件拷贝的方式对数据库进行备份，以磁盘块为基本单位将数据备份。通过备份的数据文件及归档日志等文件，数据库可以进行完全恢复。物理备份速度快，一般被用作对数据进行备份和恢复，用于全量备份的场景。通过合理规划，可以低成本进行备份与恢复。

由于无管理员权限，此实验略。

## 5.问题及解决

### 问题一：

在事务出错后，无法通过 end、commit、rollback 等操作自动回滚，均会报错



我们查阅 google 等，只找到建议 commit、abort 等，但在 opengauss 中无法执行，最后关闭查询，使其回滚并终止了事务。



问题二：

在为 tbcell\_1 的 totletilt 添加约束后，原表部分数据不符合约束：

```
from tbcell_invariable ;
执行失败，失败原因：ERROR: null value in column "totletilt" violates not-null constraint
Detail: Failing row contains (yiyang, 11317-128, 栢阳上庄F-HLH-1, 11317, 栢阳上庄F-HLH, 38400, 373, 1, 124, 14400, 华为, 113, 34, 宏站, 10, null, null, null, null).
```

经过研究发现，本实验不需要对 totletilt 以及约束作任何要求，故删去了约束。

问题三：

在操作 tbcell\_2，发现 tbcell\_2 有很多数据缺失。

经过回顾发现，之前的实验也使用到了 tbcell\_2 并作出了修改，故从原表再克隆一份 tbcell\_3 代替 tbcell\_2 进行操作，并将实验中对 tbcell\_2 的操作移至 tbcell\_3

问题四：

在进行死锁实验的示例二时，执行  
START TRANSACTION ISOLATION LEVEL repeatable read;  
select SECTOR\_NAME from tbcell\_1 where SECTOR\_ID='124672-0' for update;  
并未找到数据

SQL查询

实例名称: gauss-de78 | 192.168.0.73,192.168.0.117,192.168.0.17:8000 | 字符集: UTF8 | 时区: Etc/GMT-8

执行SQL(F8)

格式化(F9)

执行计划(F6)

我的SQL

```
1 START TRANSACTION ISOLATION LEVEL repeatable read;
2 select SECTOR_NAME from tbcell_1 where SECTOR_ID='124672-0' for update;
3
```

SQL执行记录

消息

结果集1

以下是select SECTOR\_NAME from tbcell\_1 where SECTOR\_ID='124672-0' for update:的执行结果集

sector_name
-------------

经过与数据库比对，发现确实没有这个数据元，所以搜索改为 124672-1 与 124672-2.