《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](#_Toc92677741)

[1.1违反check约束的update操作 2](#_Toc92677742)

[1.2事务commit/rollback操作 10](#_Toc92677743)

[2.1.3 修改数据库模式 15](#_Toc92677744)

[1.4多条insert/delete操作执行比较 20](#_Toc92677745)

[1.5保存点Savepoint设置与回滚实验 25](#_Toc92677746)

[1.6事务内某条语句执行失败对其余语句的影响 26](#_Toc92677747)

[2.并发事务控制 28](#_Toc92677748)

[2.1 read committed 隔离级别下的脏读，不可重复读，幻读 28](#_Toc92677749)

[2.2 repeatable read 隔离级别下的脏读，不可重复读，幻读 38](#_Toc92677750)

[3.事务锁机制 49](#_Toc92677751)

[3.1 死锁分析 49](#_Toc92677752)

[示例1.隔离级别read-repeatable下死锁 49](#_Toc92677753)

[示例2.加的互斥锁的粒度 52](#_Toc92677754)

[4备份与恢复 56](#_Toc92677755)

[5.问题及解决 56](#_Toc92677756)

[问题一： 56](#_Toc92677757)

[问题二： 57](#_Toc92677758)

[问题三： 57](#_Toc92677759)

[问题四： 57](#_Toc92677760)

# 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tbcell表的副本tbcell\_1, tbcell\_3（注意tbcell\_2先前已被使用并被更改，故作tbcell\_3），并将tbcell表的数据导入进去

create table tbcell\_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,

PCl integer CHECK(PCl 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),

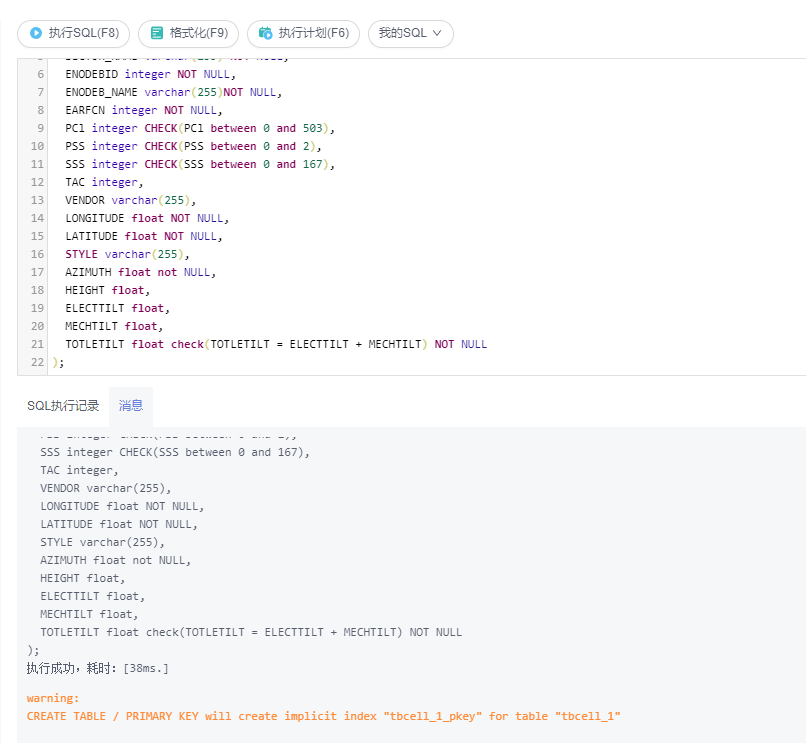
AZIMUTH float not NULL,

HEIGHT float,

ELECTTILT float,

MECHTILT float,

TOTLETILT float

);



克隆tbcell\_3



用Alter table add check添加约束

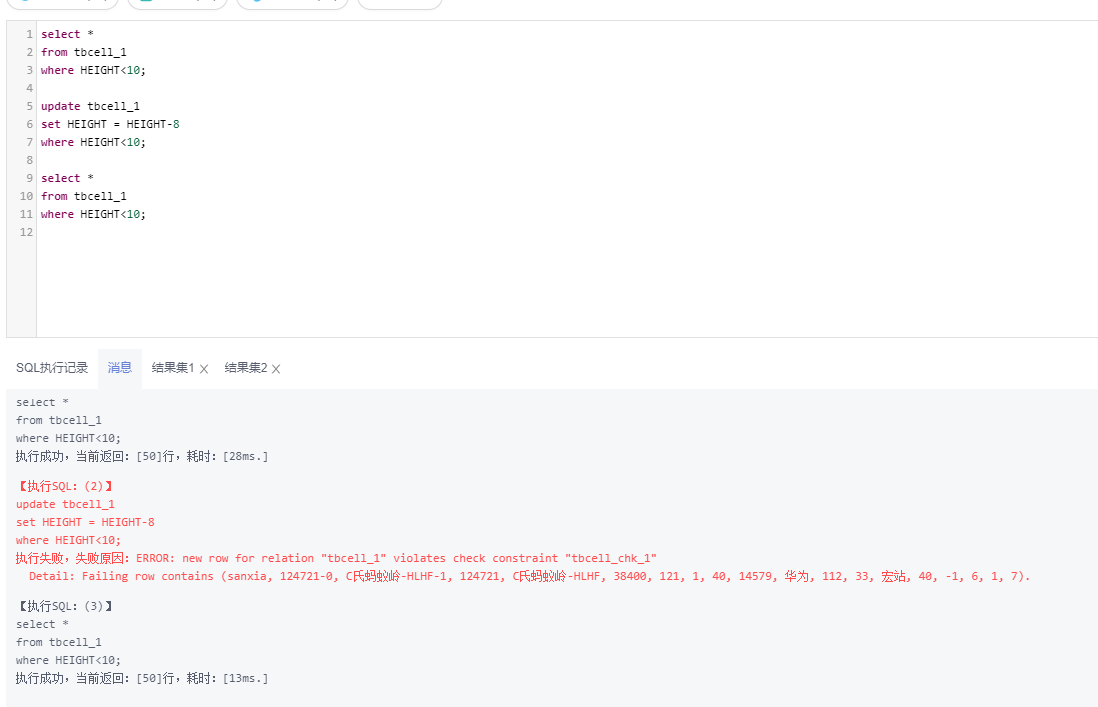
alter table tbcell\_1 add constraint tbcell\_chk\_1 check(HEIGHT>=0);

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

(已添加，此处用56代替)



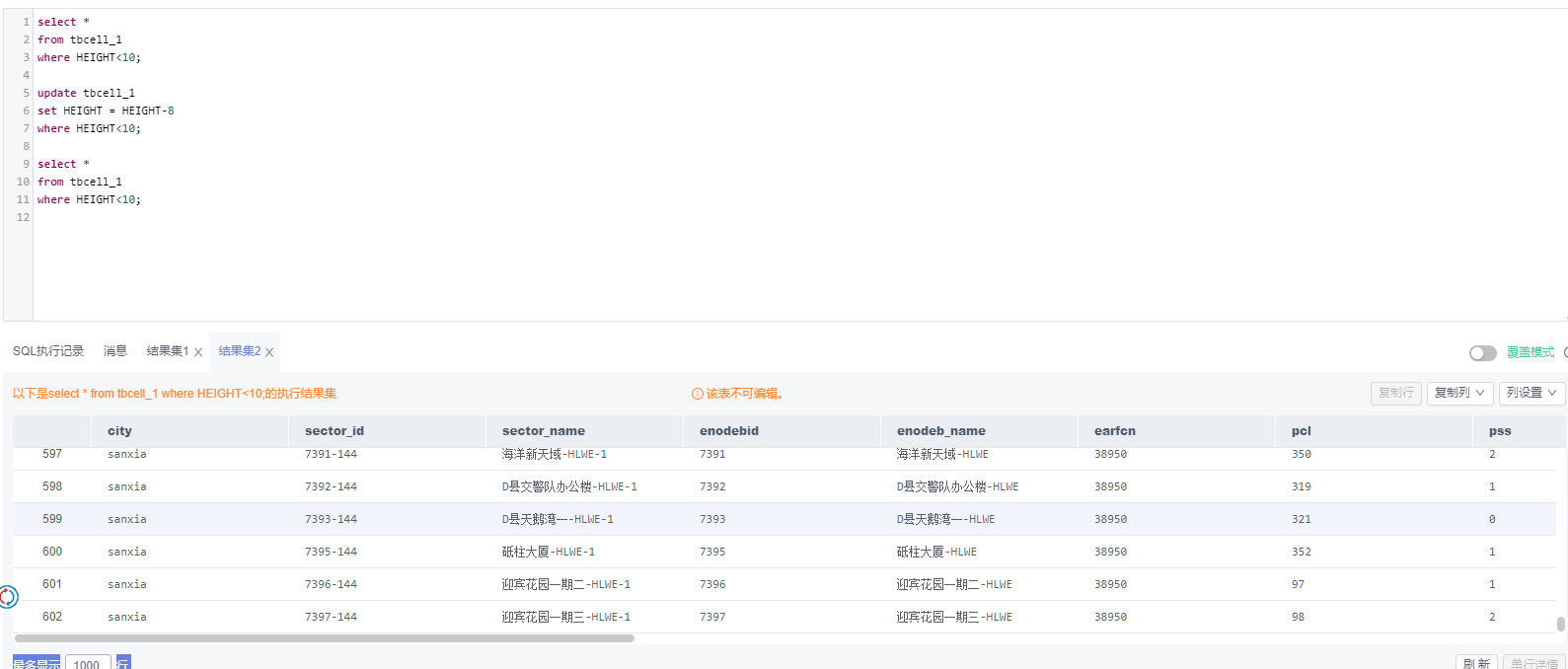
将实验内容在tbcell\_1表上组织成普通的SQL语句，顺序执行



更新时报错

两次查询结果一样（602行）



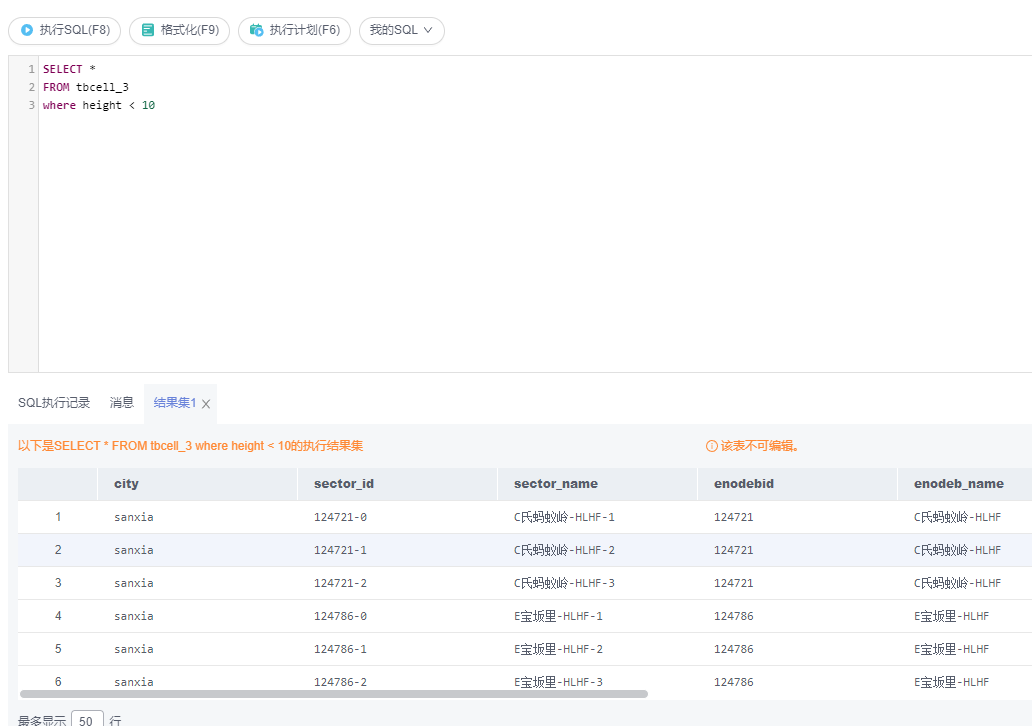


将实验内容在tbcell\_3表上组织成事务执行（以start transaction;开始，以end;结束)。

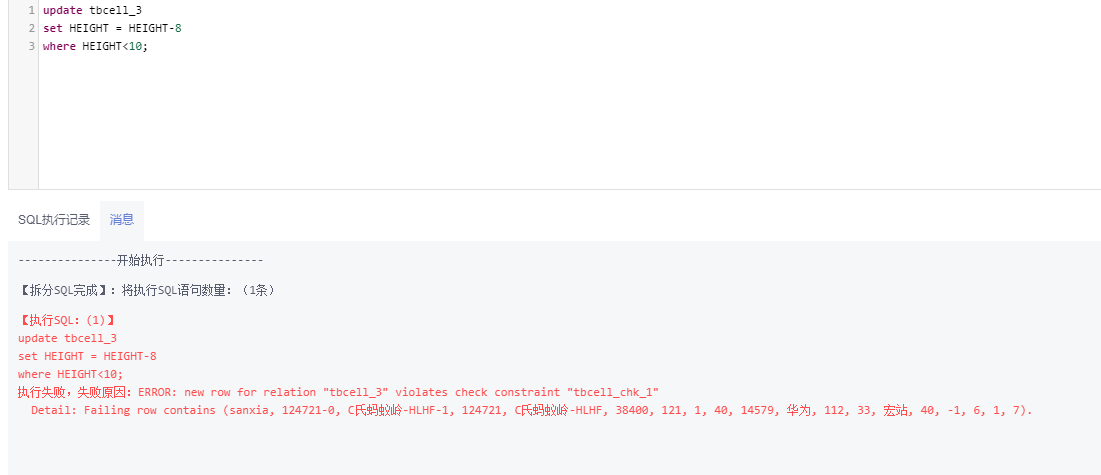


在事务中查询“小区天线高度（HEIGHT)小于10”的小区，结果与上面的查询结果一样



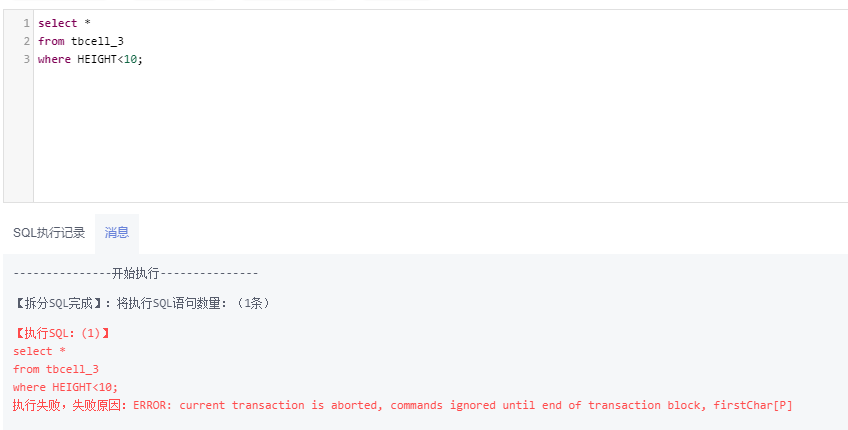


然后，在事务中更新：



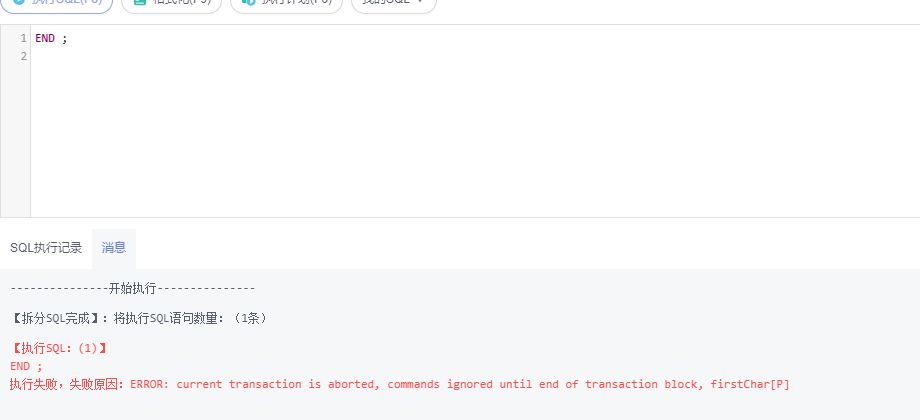
报错。

再次，在事务中查询“小区天线高度(HEIGHT)小于10”的小区



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



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



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""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。

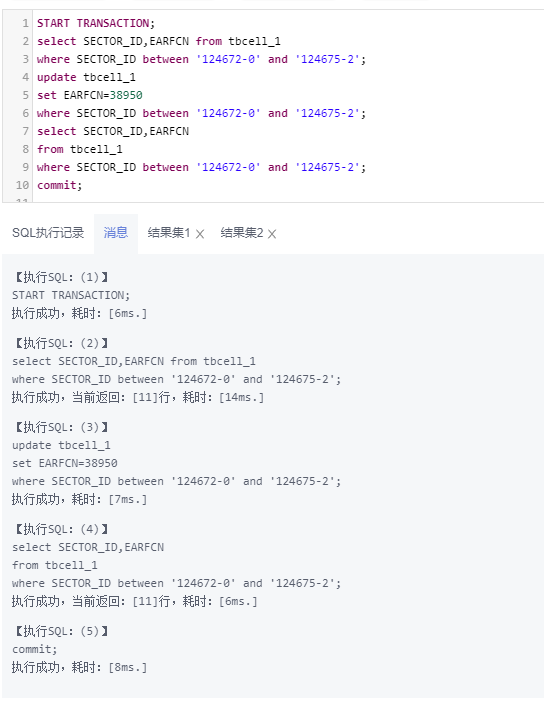
将step1、step2和step3 的数据库访问组织成1个单一事务T1，再将step3作为1个独立事务，提交DBMS,串行执行这2个事务，观察T1中的 rollback、commit对事务执行结果的影响。

由step1、step2和step3组成的事务T1采用以下2种结束方式:

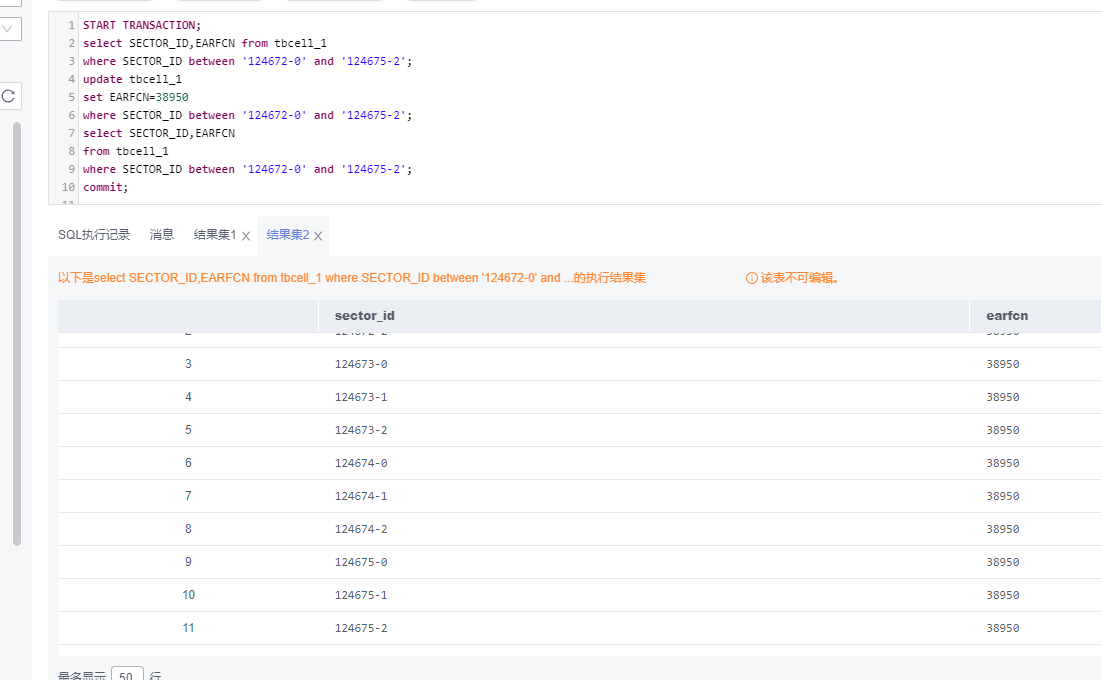
(1)以commit结束。

(2)以rollback结束。

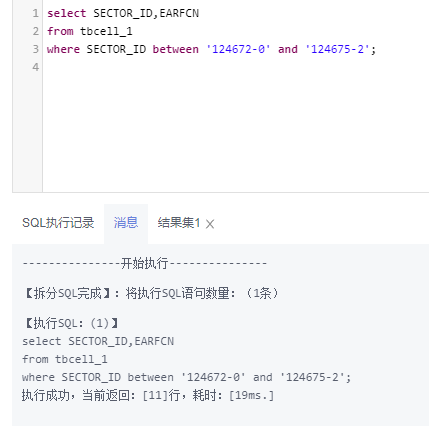
以commit结束的T1事务



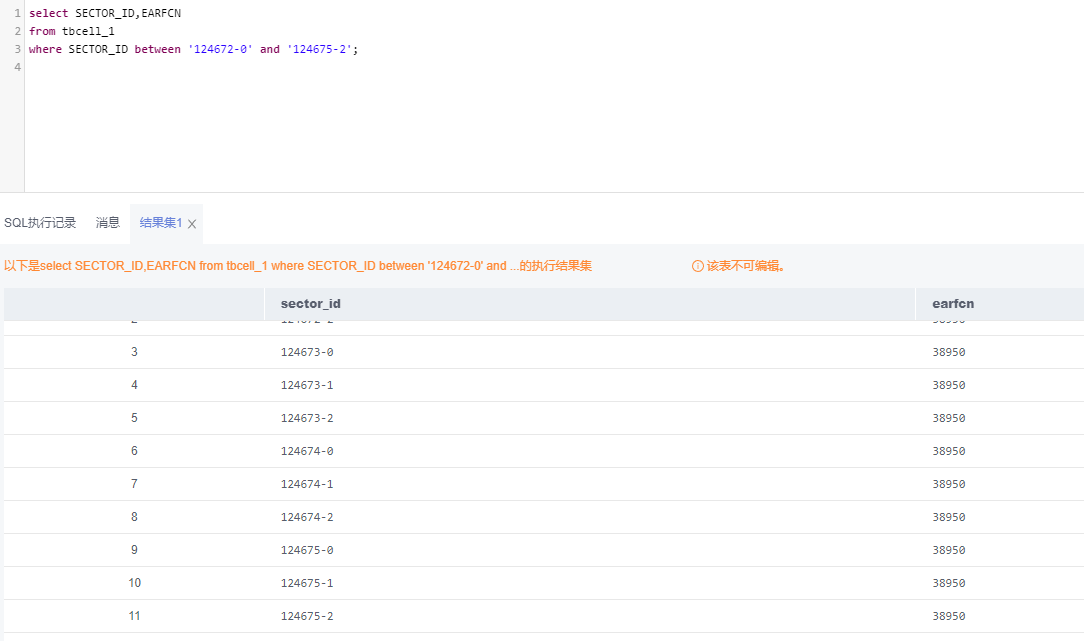
共查询到12条数据，并将EARFCN更新为38950，更新成功，将事务提交。

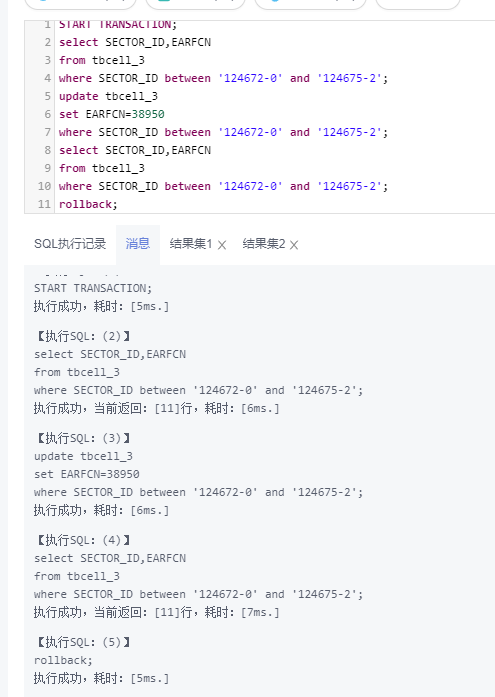


作为独立事务，再次进行查询



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。



以rollback 结束的T1事务  


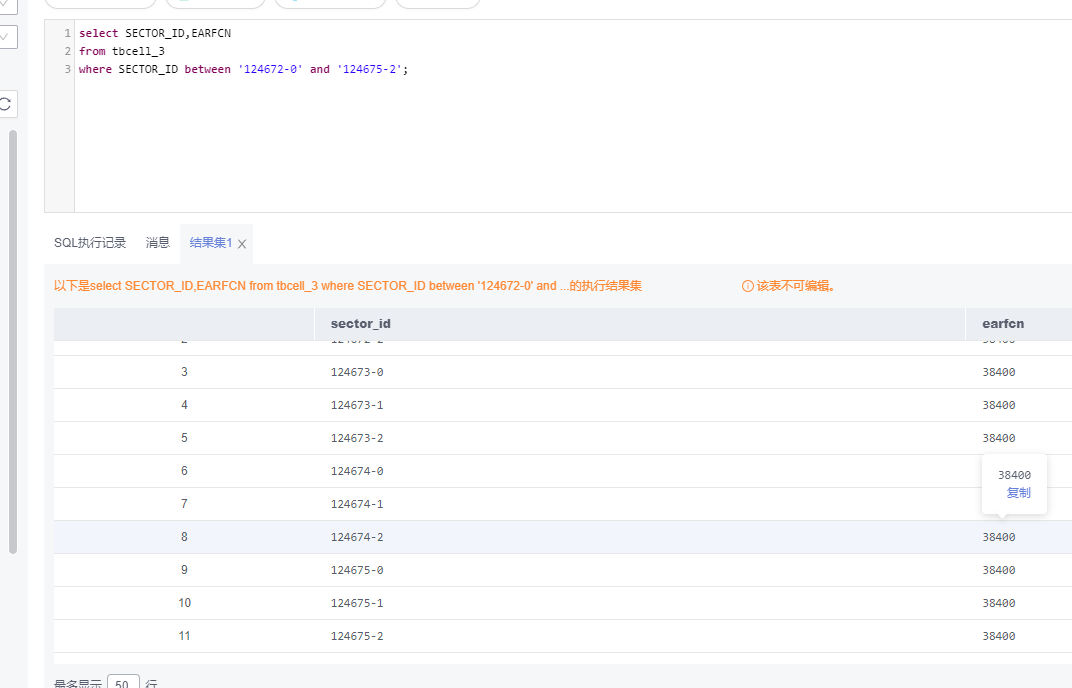
共查询到12条数据，并将EARFCN 更新为38950，更新成功，将事务回滚。



作为独立事务，再次进行查询



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。



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。

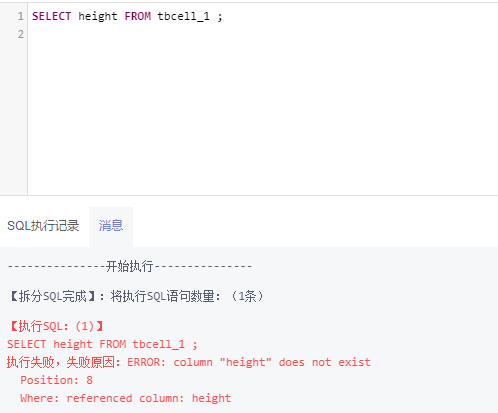
查看数据库（用select 语句查看被删除/增加的列)，观察数据库模式修改语句（alter table)，是否会受到rollback,commit语句的影响。

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

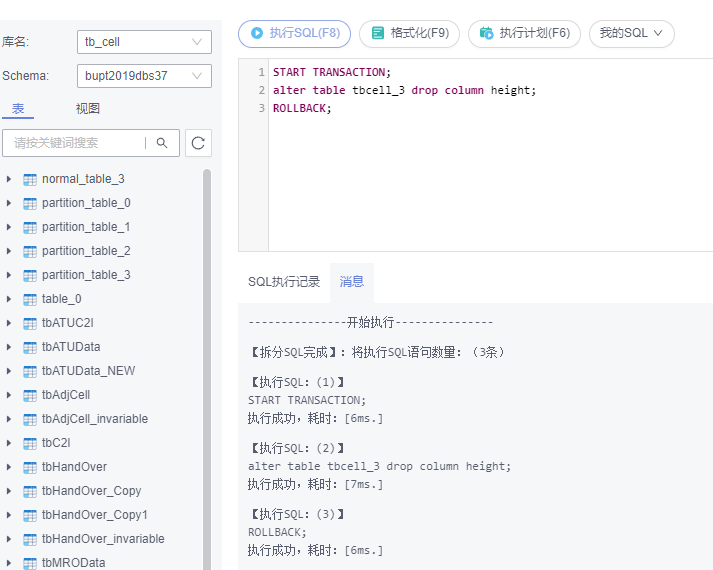


查看height列

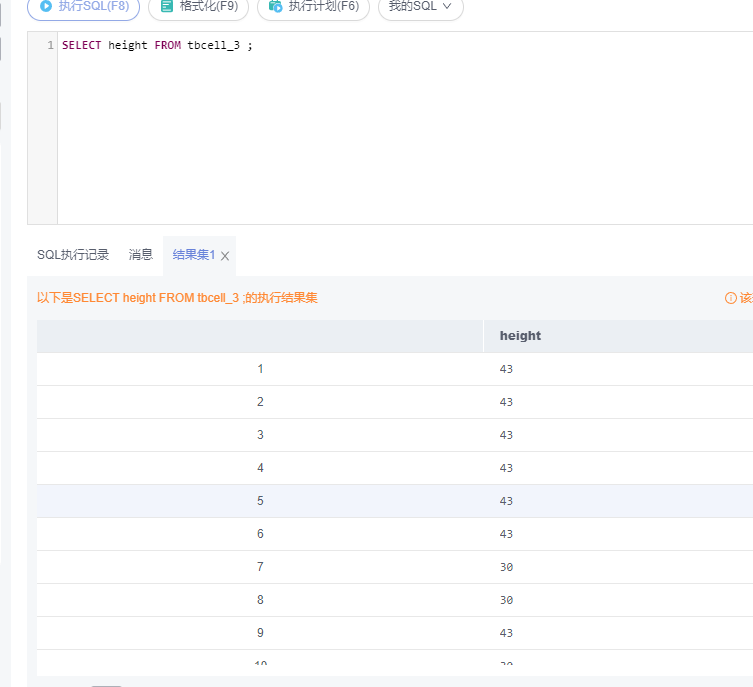


报错，显示该列以不存在，事务提交后，删除操作有效

删除height列，以rollback结束



查看height列



该列仍然存在，事务回滚，删除操作无效。

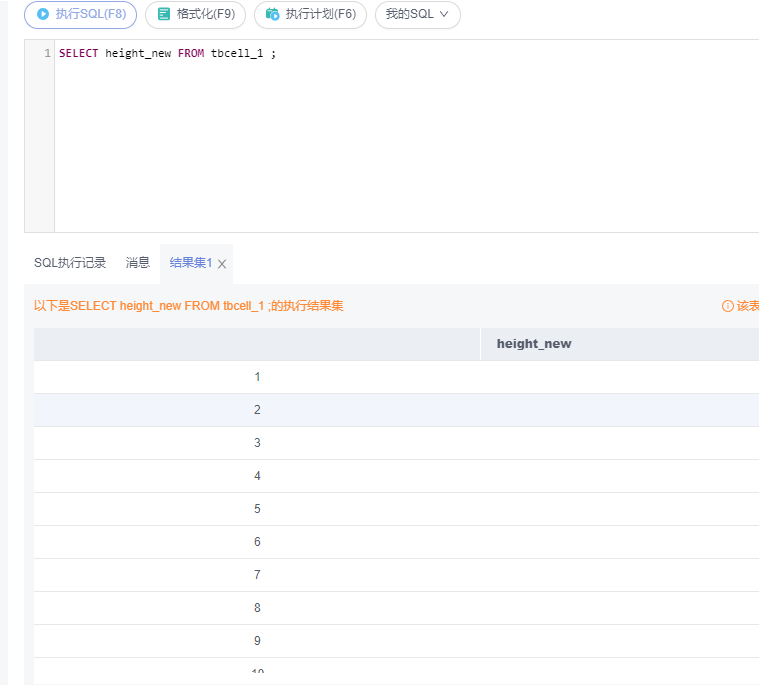
增加height\_new列，以commit结束



查看height\_new列



显示该列存在，内容为空，事务提交后，增加操作有效



增加height\_new列，以rollback结束



查看height\_new列



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);

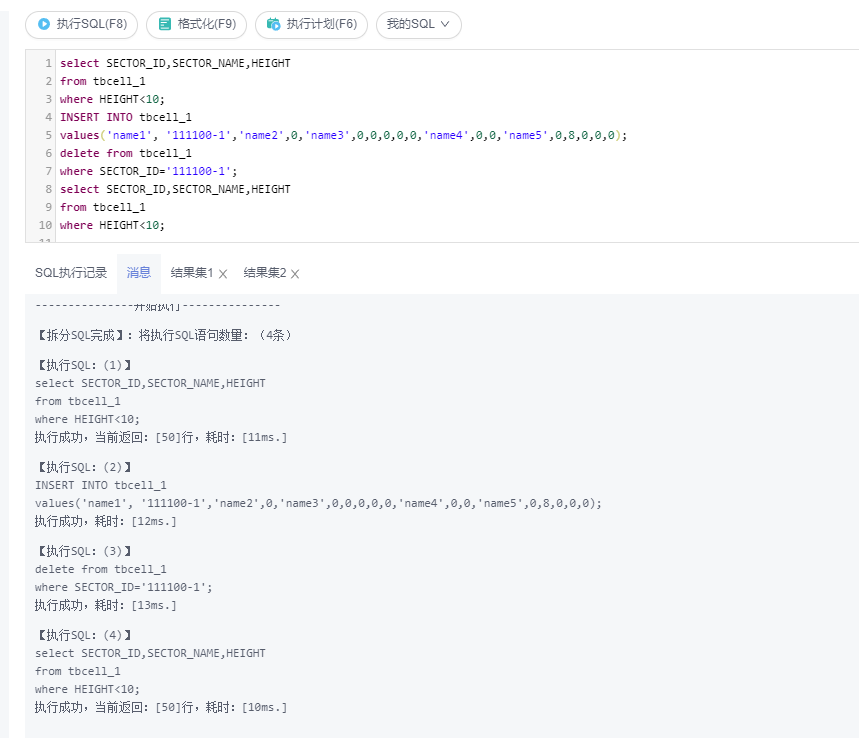
delete from tbcell\_1

where SECTOR\_ID='111100-1';

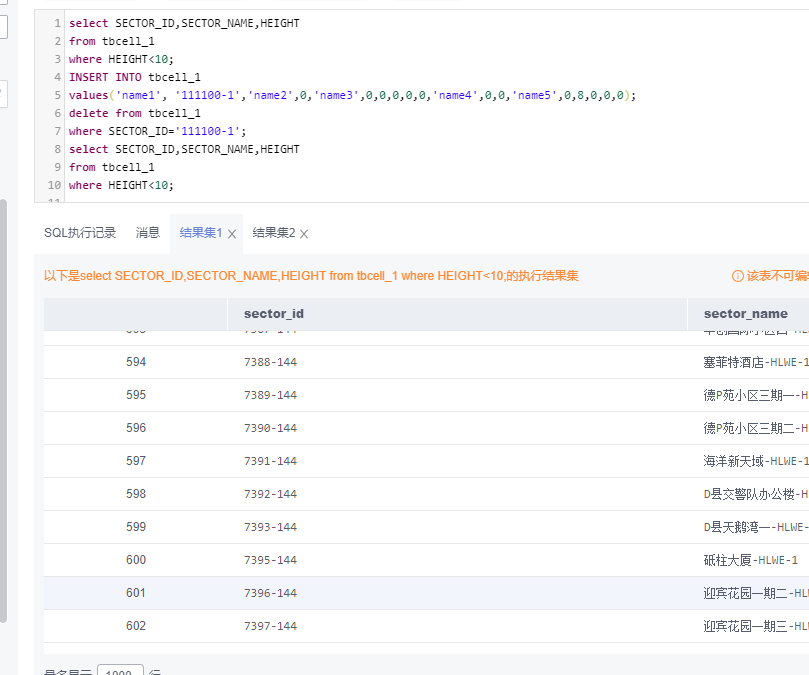
select SECTOR\_ID,SECTOR\_NAME,HEIGHT

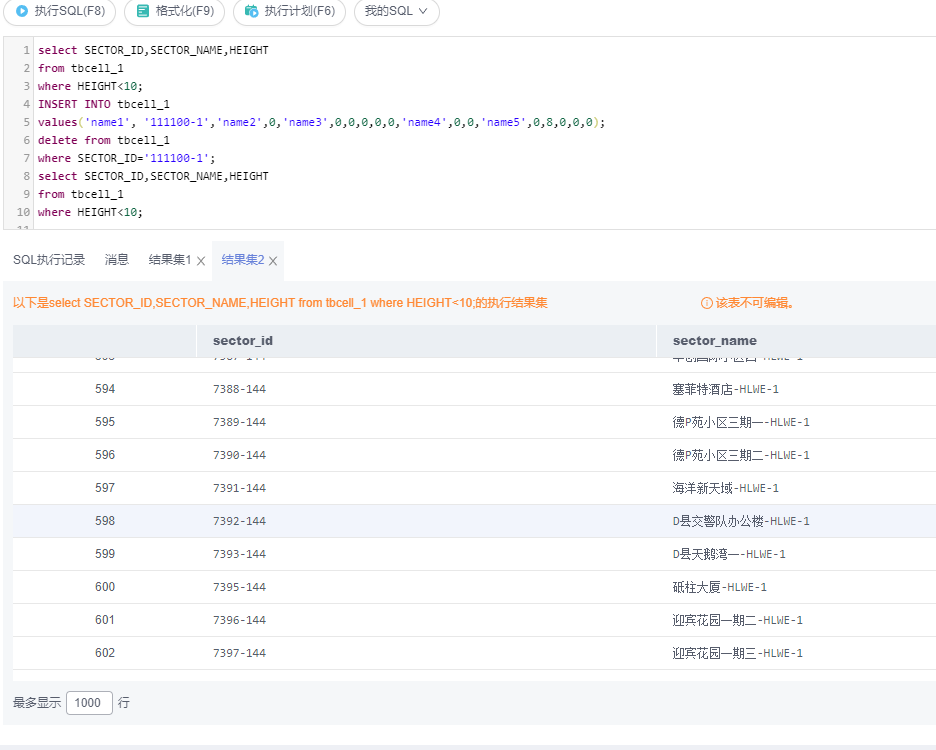
from tbcell\_1

where HEIGHT<10;



两次查询的结果相同





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

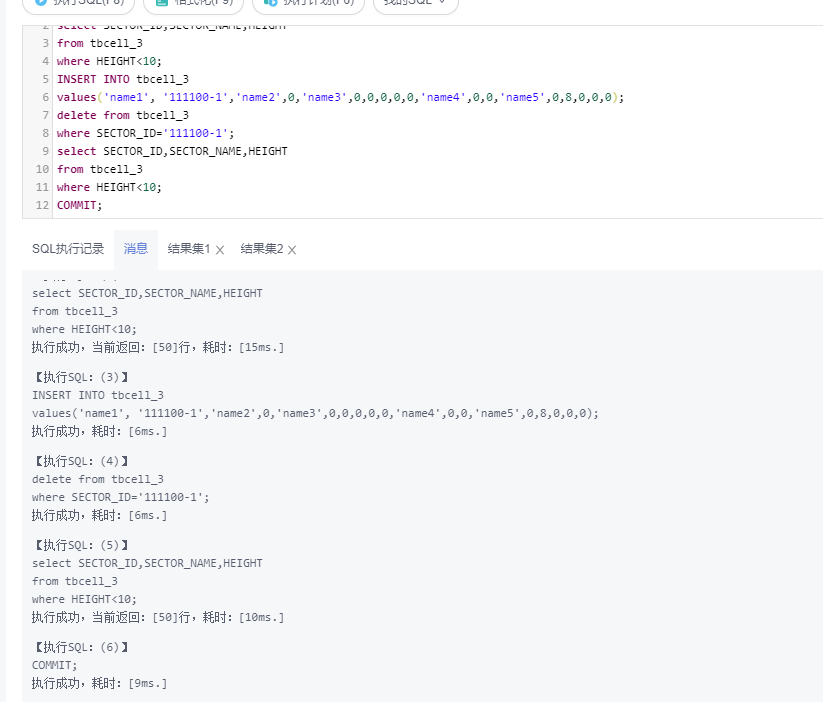
where SECTOR\_ID='111100-1';

select SECTOR\_ID,SECTOR\_NAME,HEIGHT

from tbcell\_3

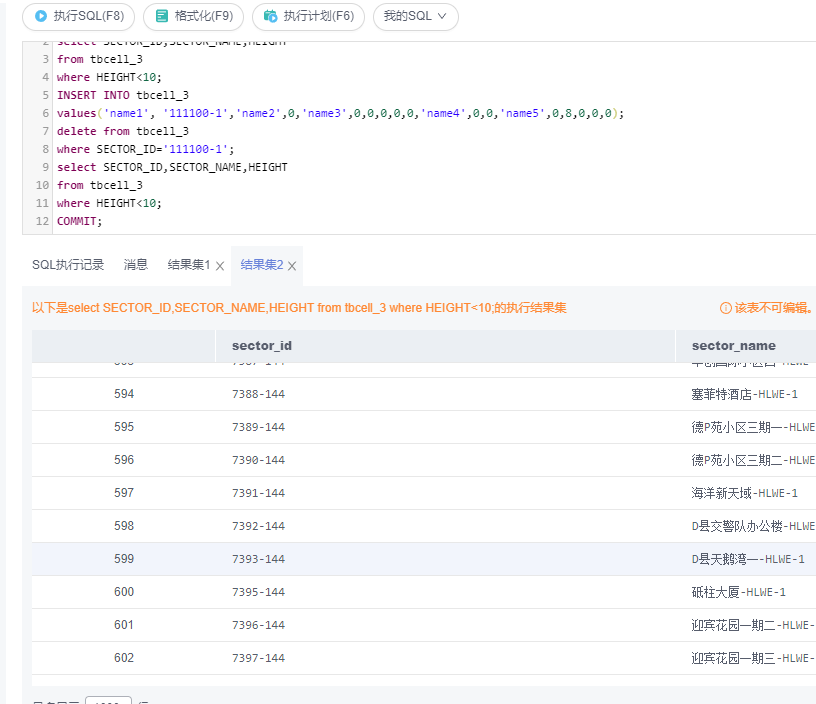
where HEIGHT<10;

COMMIT ;



两次查询的结果相同（一共602行)





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、添加之刖、删乐之石夺，史用 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;

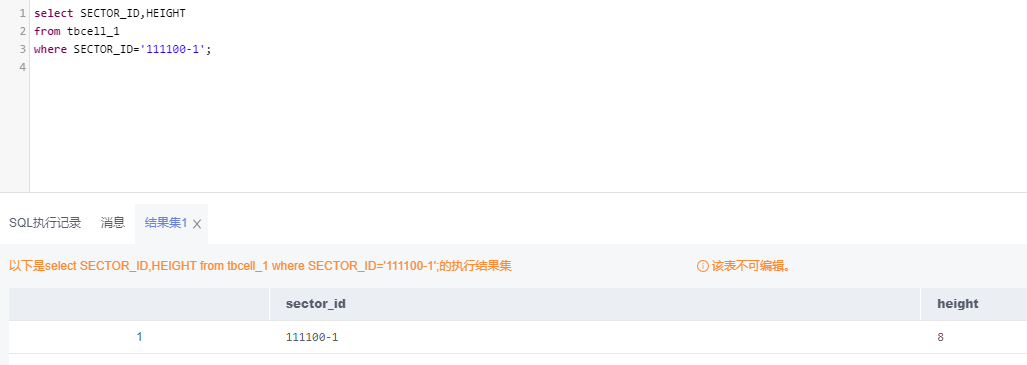
以下事务以tbCell为访问对象，在插入操作之后设置了保存点:



查看是否插入成功，插入的数据是否被删除



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



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(PCl 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



插入数据，并更新HEIGHT的值



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;

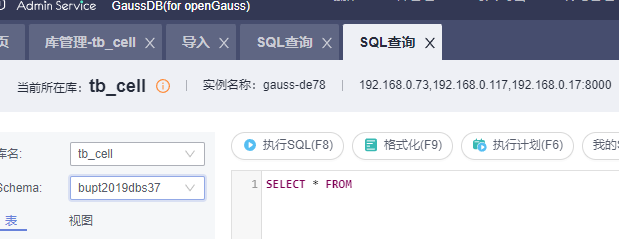
Step4.在事务T2中，查询SECTOR\_ID为“124672-O”的HEIGHT值，并COMMIT 提交事务T2;Step5.用ROLLBACK回滚事务T1;

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



查询SECTOR\_ID为“15500-128”的HEIGHT值





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，将SECTOR\_ID为“15500-128”的HEIGHT修改为6



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



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;



在事务T2中，将SECTOR ID为“15500-128”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



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;

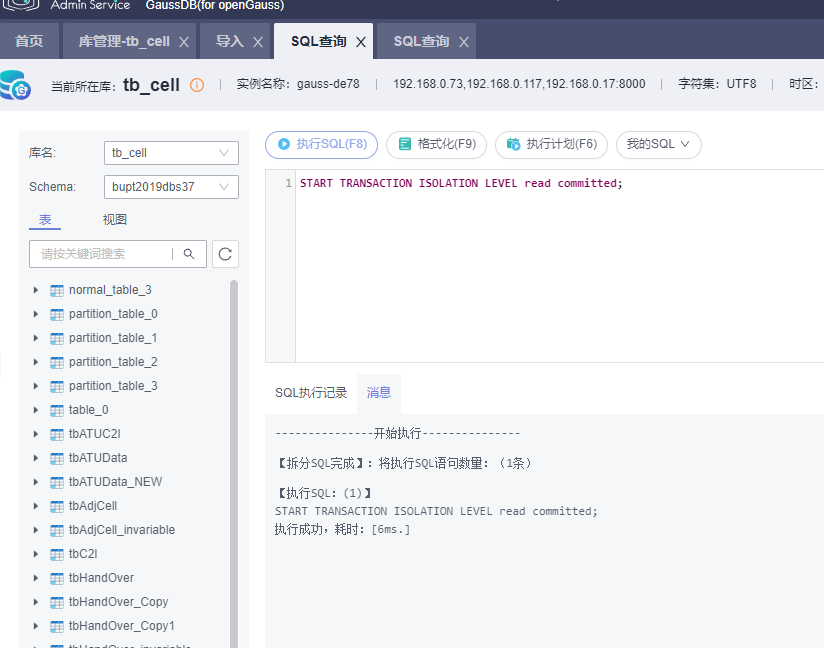
Step2.在事务T1中，查询SECTOR\_ID在“124672-0”与“124672-4”之间的元组;Step3.在事务T2中，插入SECTOR\_ID为“124672-3”的元组，并COMMIT提交事务T2;

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)

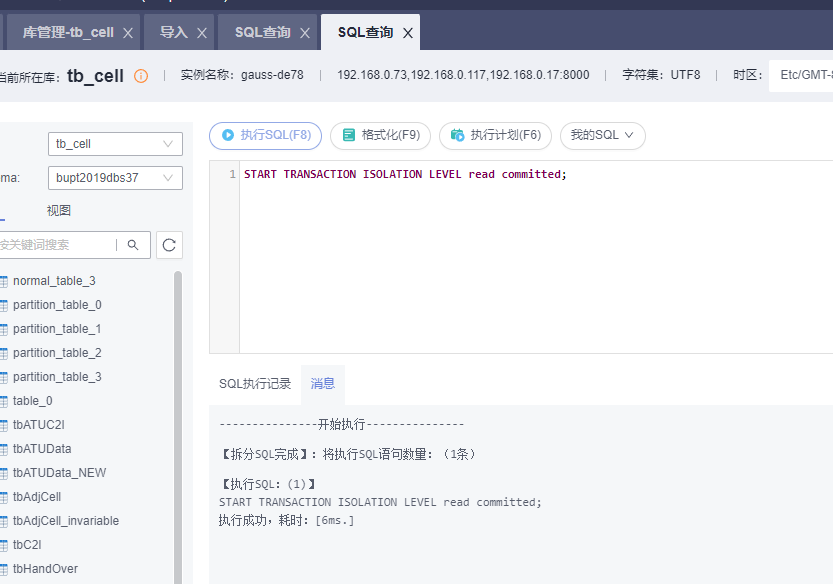
Step4.在事务T1中，查询在“124672-0”与“124672-4”之间的元组，并COMMIT提交事务T1;要求:

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



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';

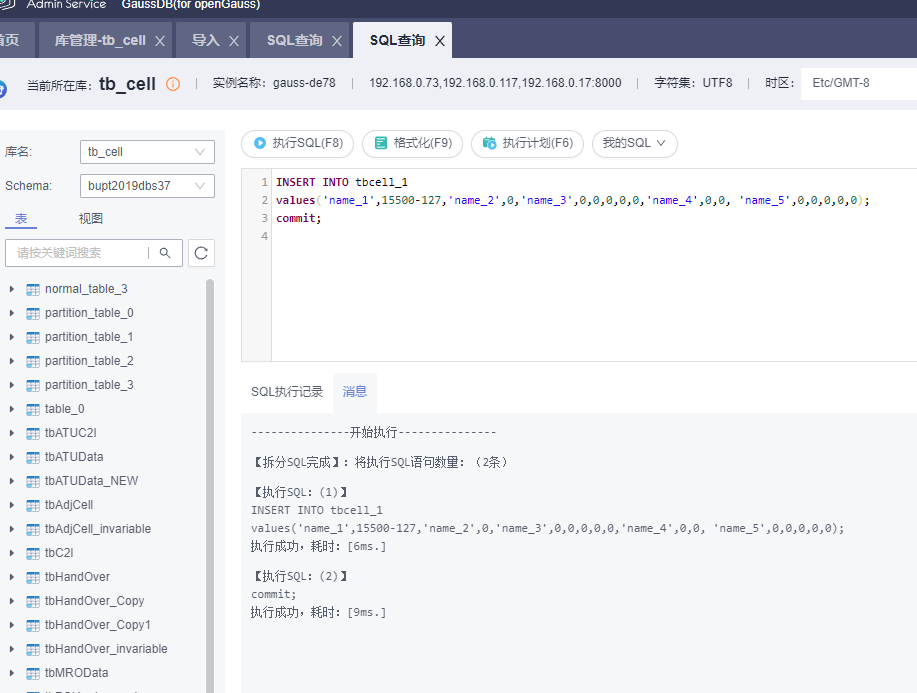


共有3行数据

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;

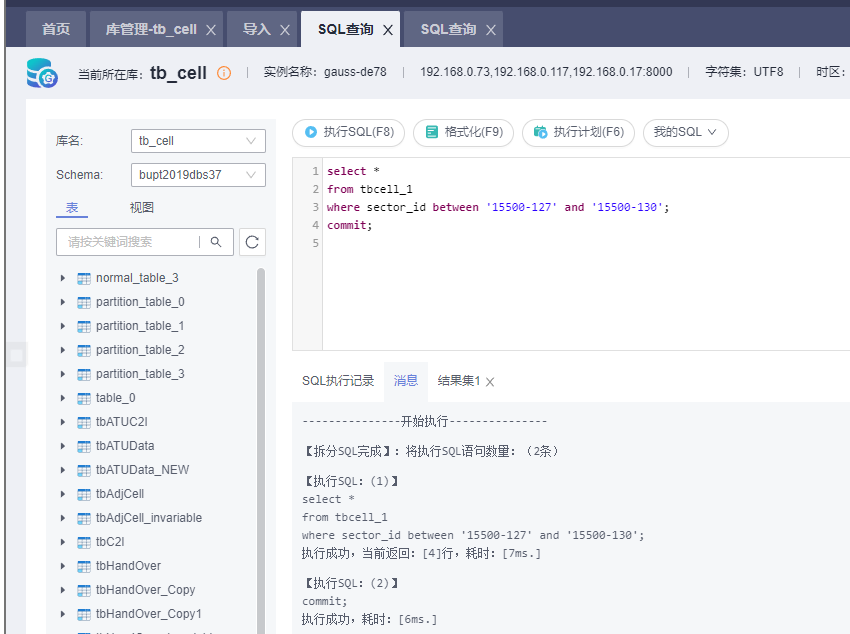
在事务T1中，再次查询SECTOR ID在“15500-127”与“15500-130”之间的元组,并COMMIT提交事务

select \*

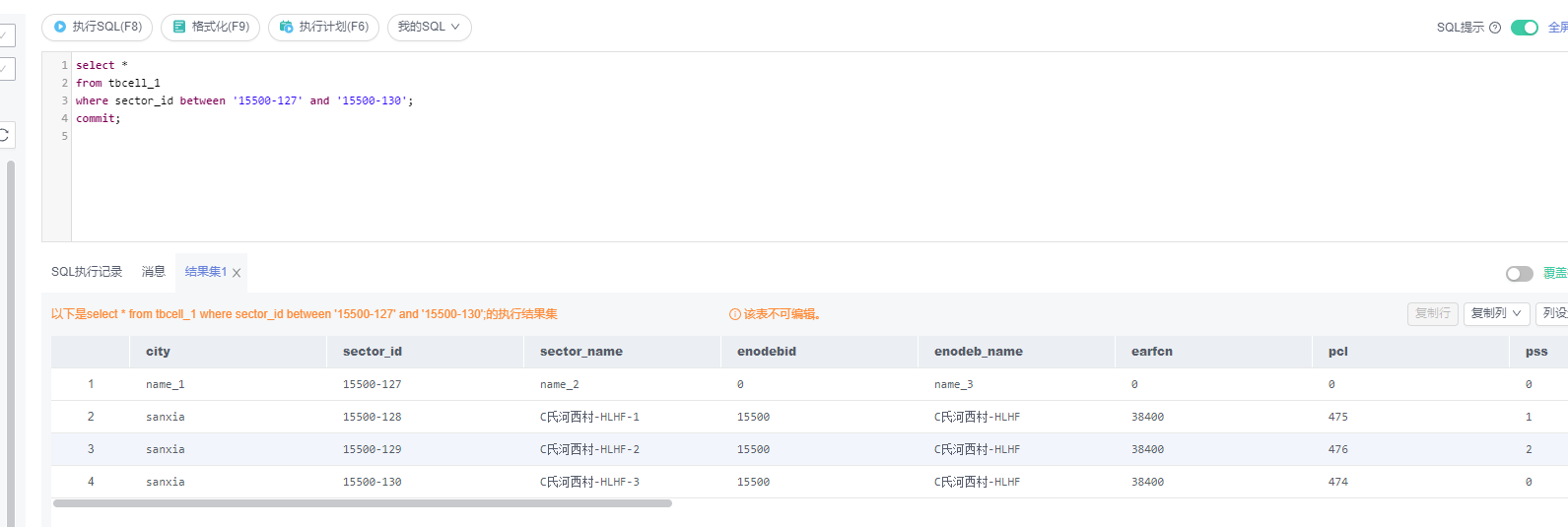
from tbcell\_1

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

commit;



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



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

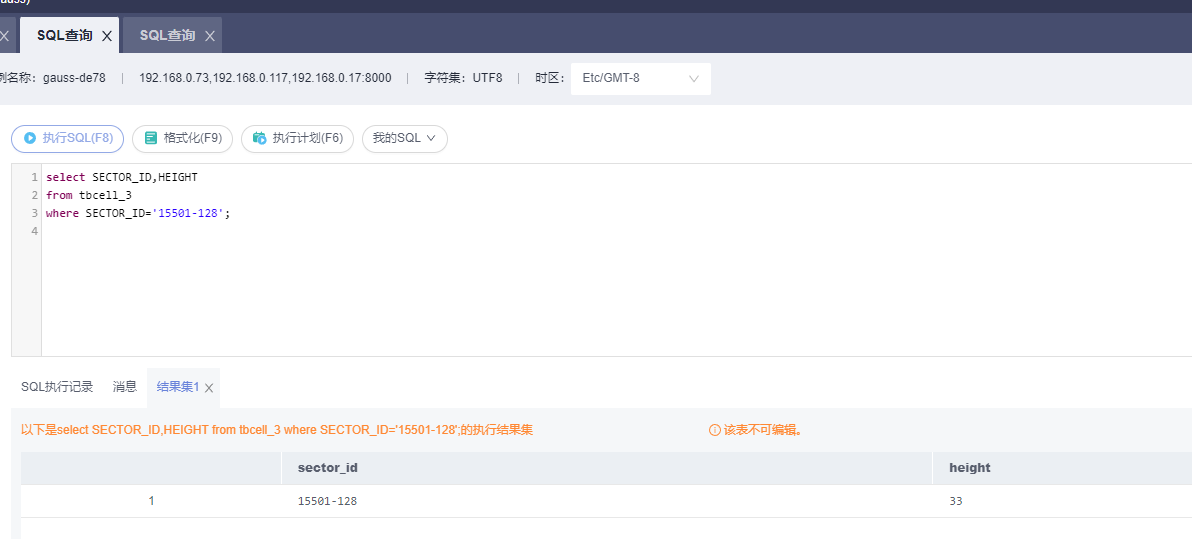


查询SECTOR\_ID为“15501-128”的HEIGHT值

select SECTOR\_ID,HEIGHT

from tbcell\_3

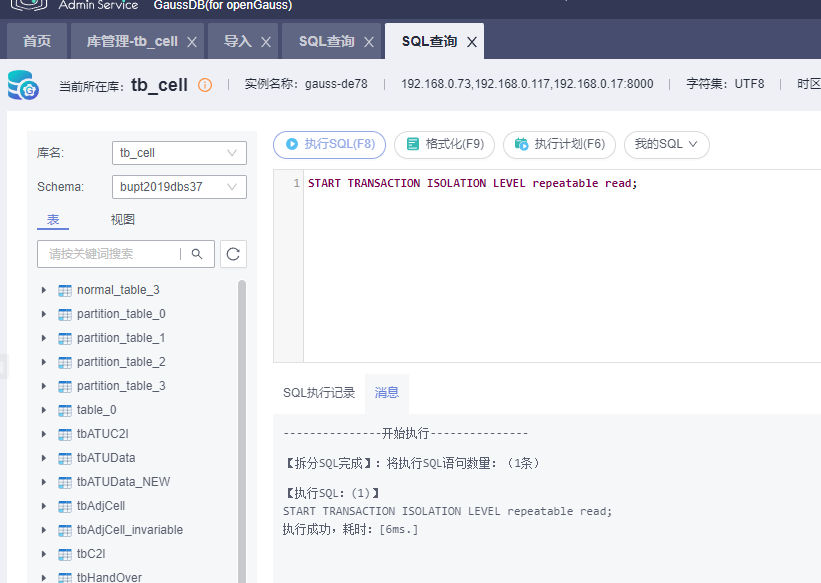
where SECTOR\_ID='15501-128';



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

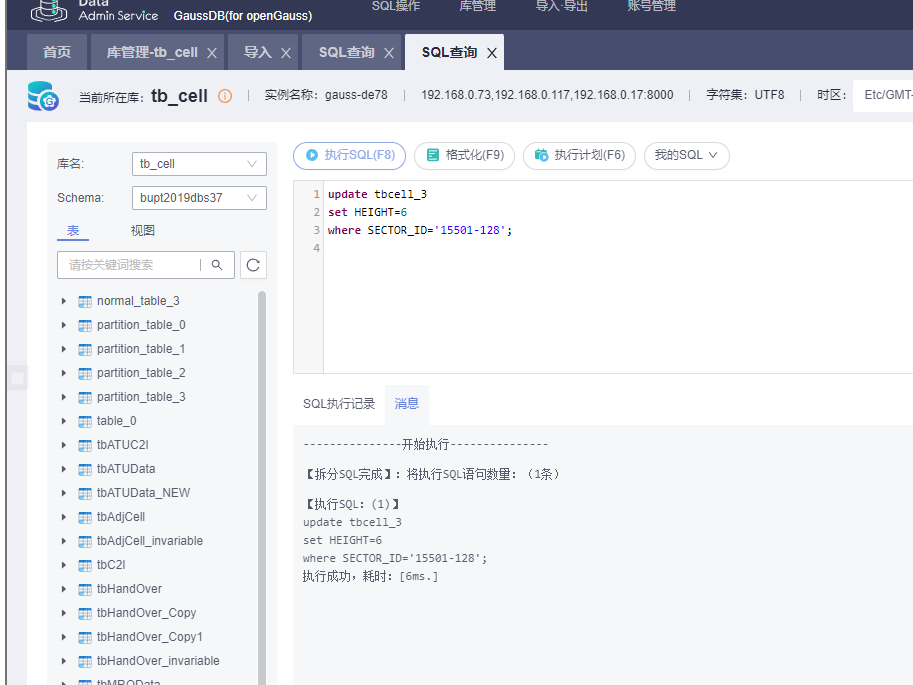


在事务T1中，将SECTOR\_ID为“15501-128”的HEIGHT修改为6;

update tbcell\_3

set HEIGHT=6

where SECTOR\_ID='15501-128';

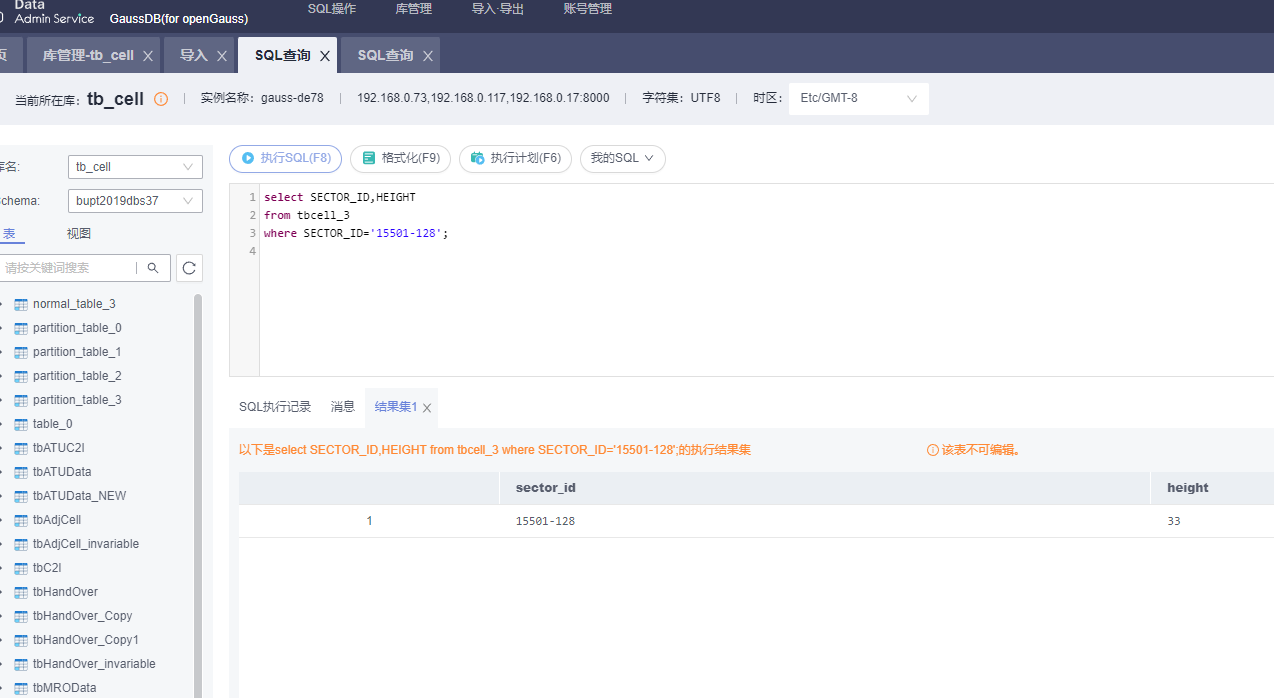


在事务T2中，查询SECTOR\_ID为“15500-128”的HEIGHT值;

select SECTOR\_ID,HEIGHT

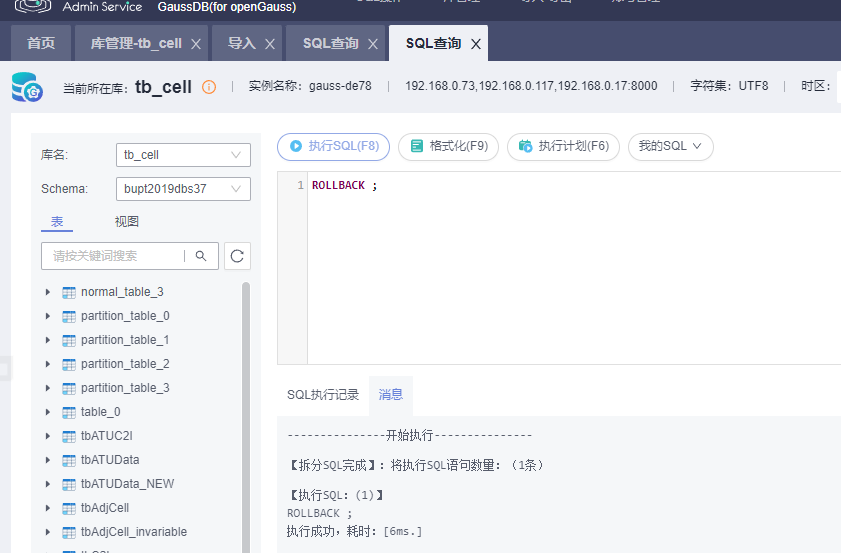
from tbcell\_3

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;

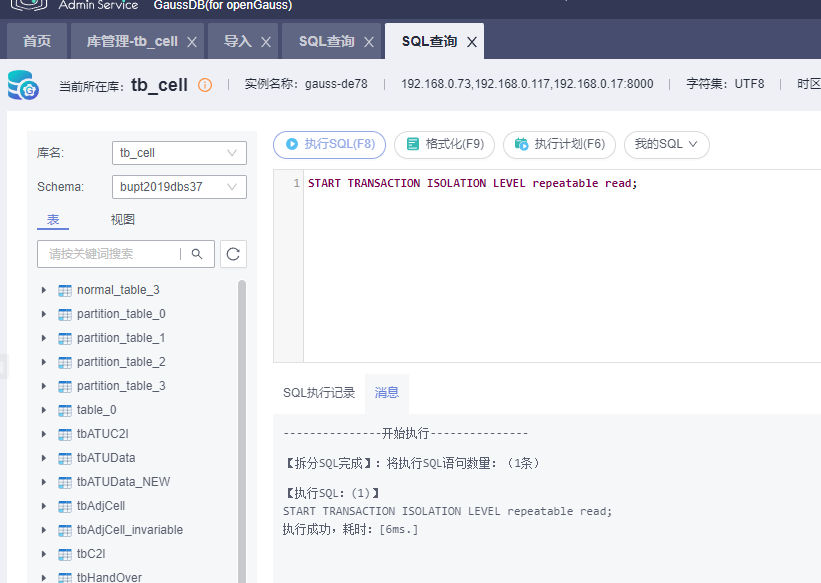
Step4.在事务T1中，查询SECTOR\_ID为“124673-0”的HEIGHT值，并COMMIT提交事务T1;要求:

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



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;

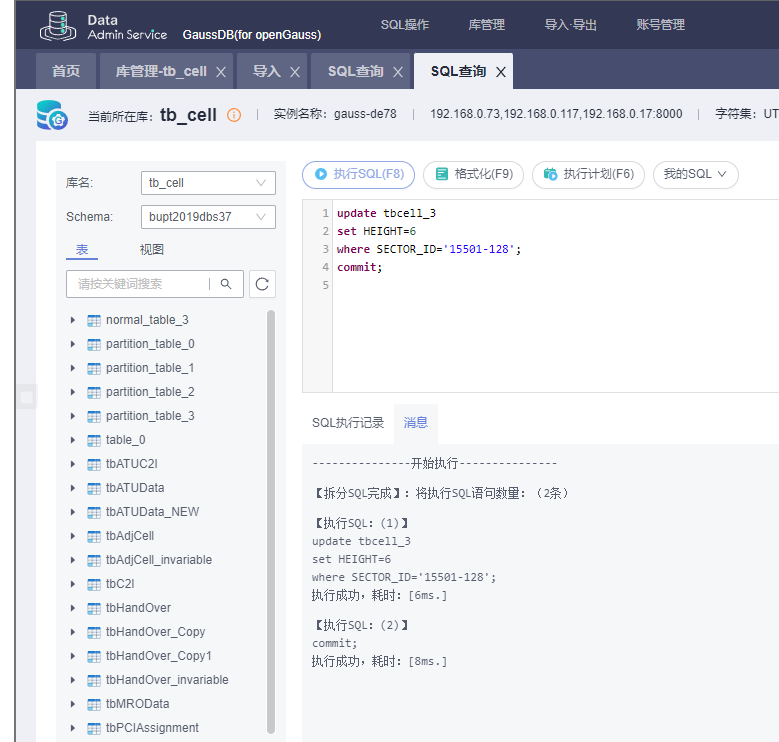
select SECTOR\_ID,HEIGHT

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 ;



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产生变化。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-O”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)

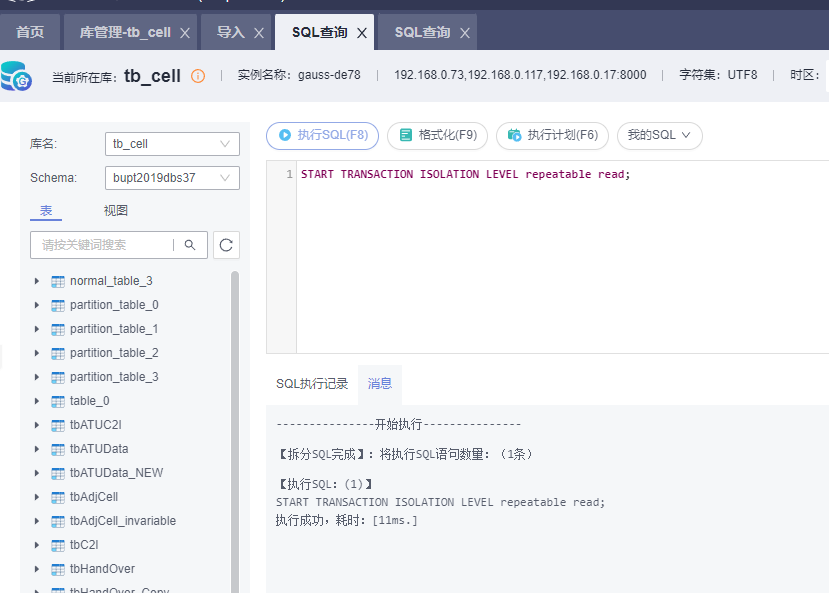
Step4.在事务T1中，查询在“124673-O”与“124673-4”之间的元组，并COMMIT提交事务T1;要求:

观察Step2和Step4两次查询到的元组是否相同，事务T1在Step4

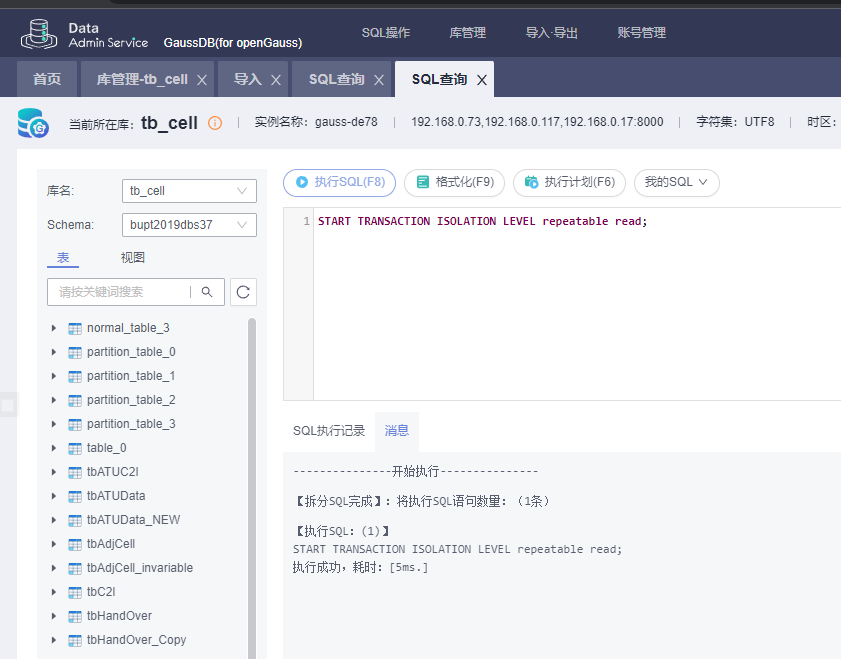
查询到的元组是否包含事务T2中插入的

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

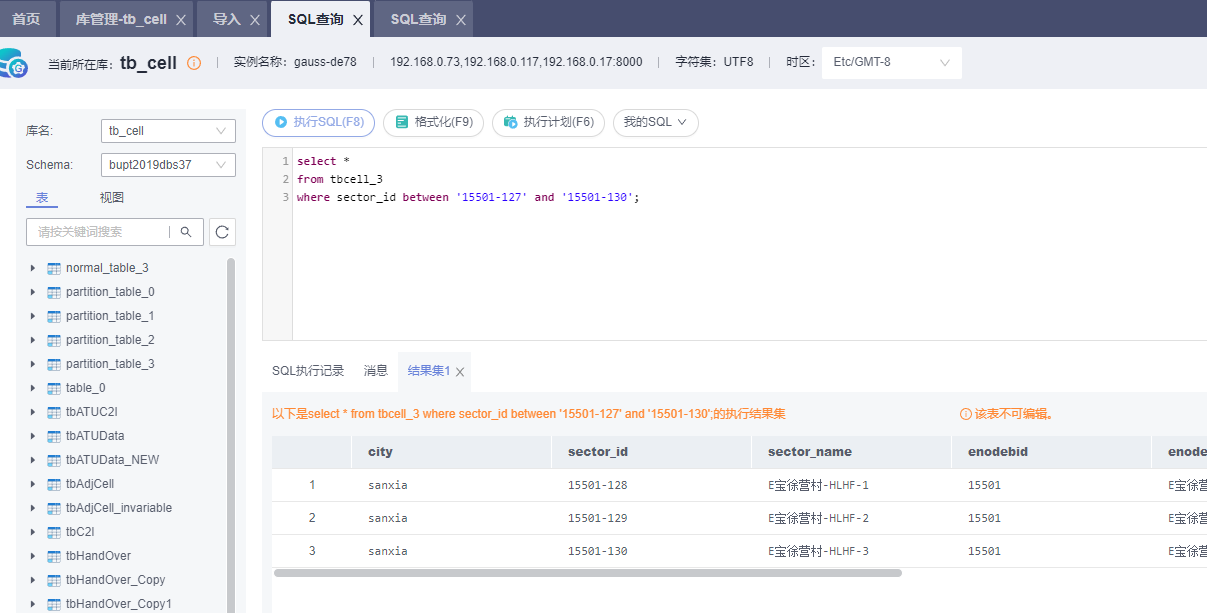


在事务T1中，查询SECTOR\_ID在“15501-127”与“15501-130”之间的元组

select \*

from tbcell\_3

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



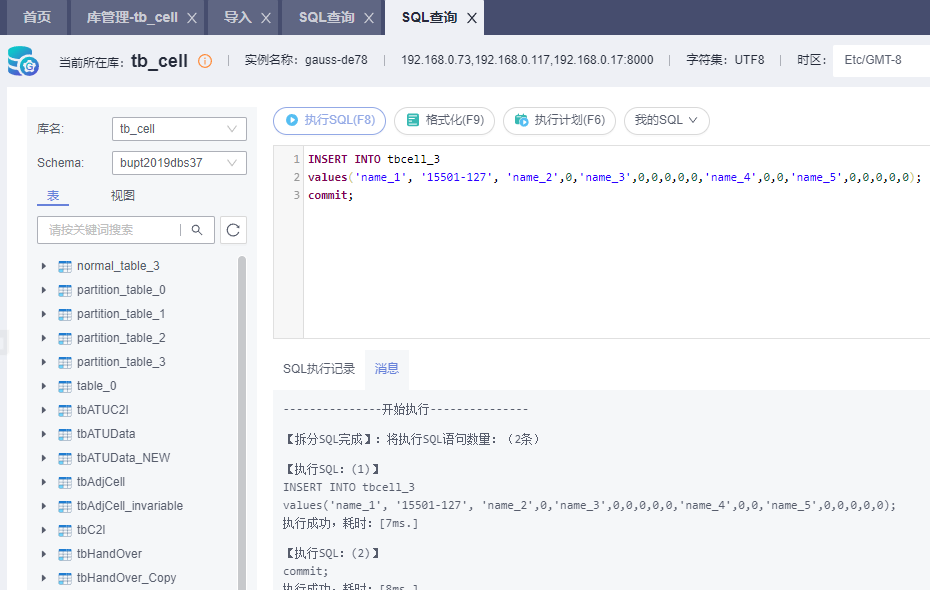
—共有3行数据

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;



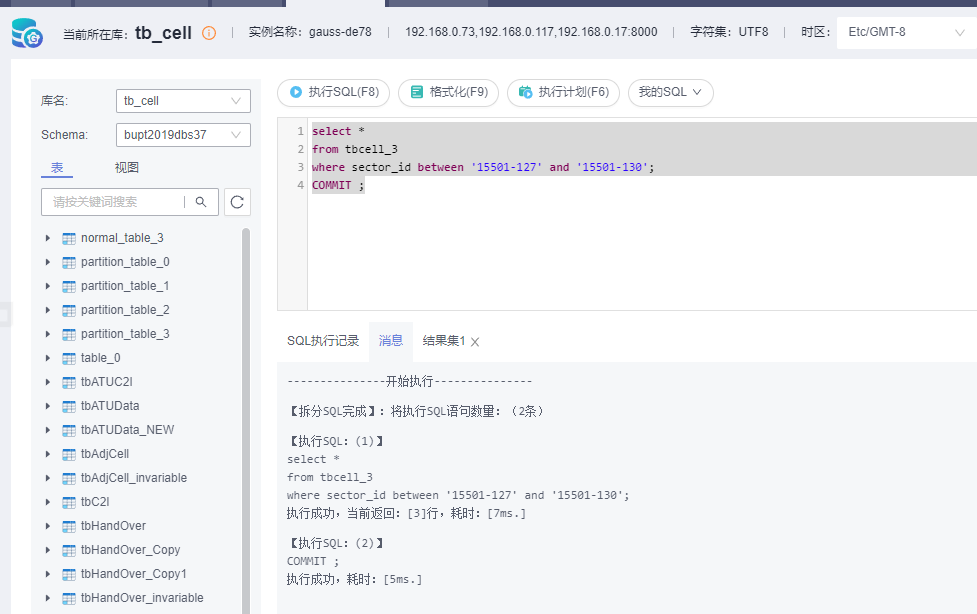
在事务T1中，再次查询SECTOR ID在“15500-127”与“15500-130”之间的元组,并COMMIT提交事务

select \*

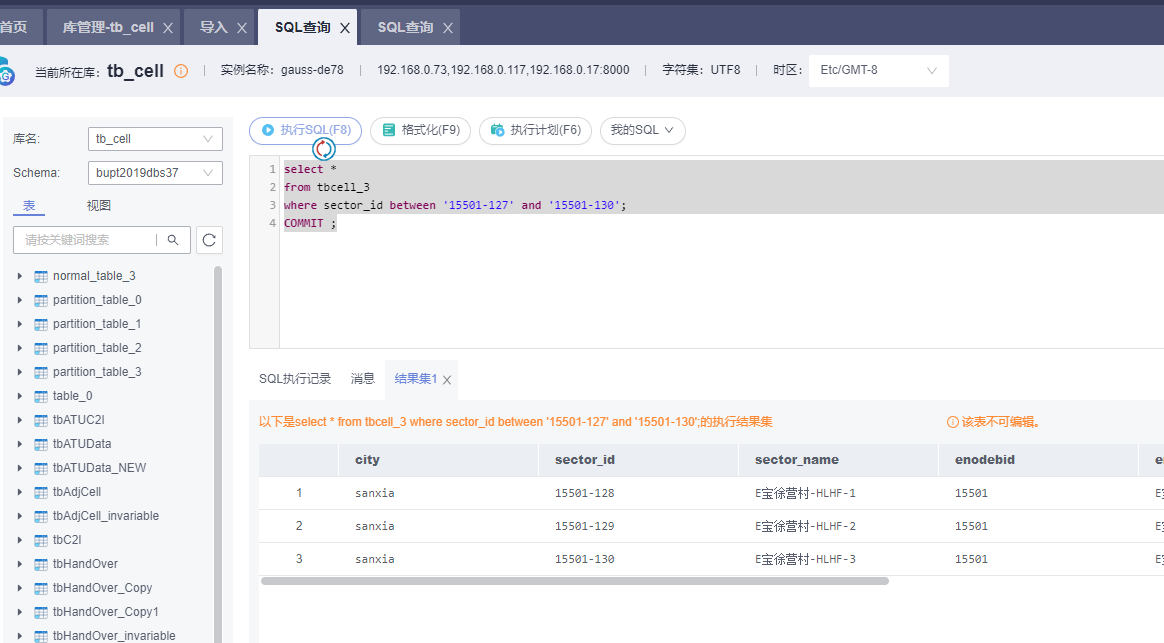
from tbcell\_3

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

COMMIT ;



查询结果一共有3行，与之前的查询结果相同



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REPEATEABLE-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 COMMITED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。

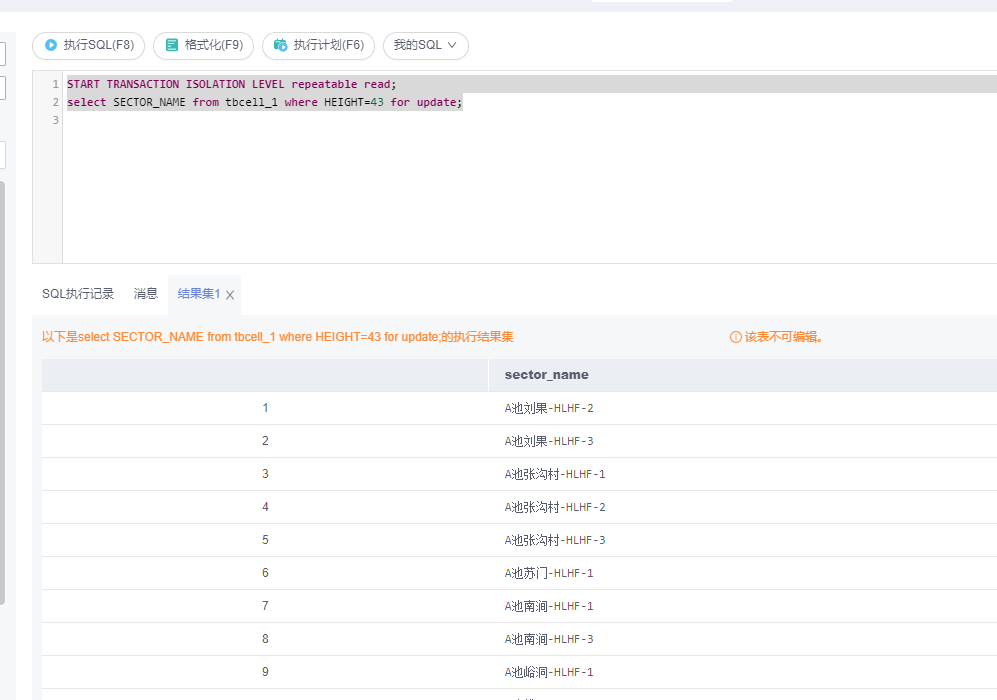
## 示例1.隔离级别read-repeatable下死锁

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;

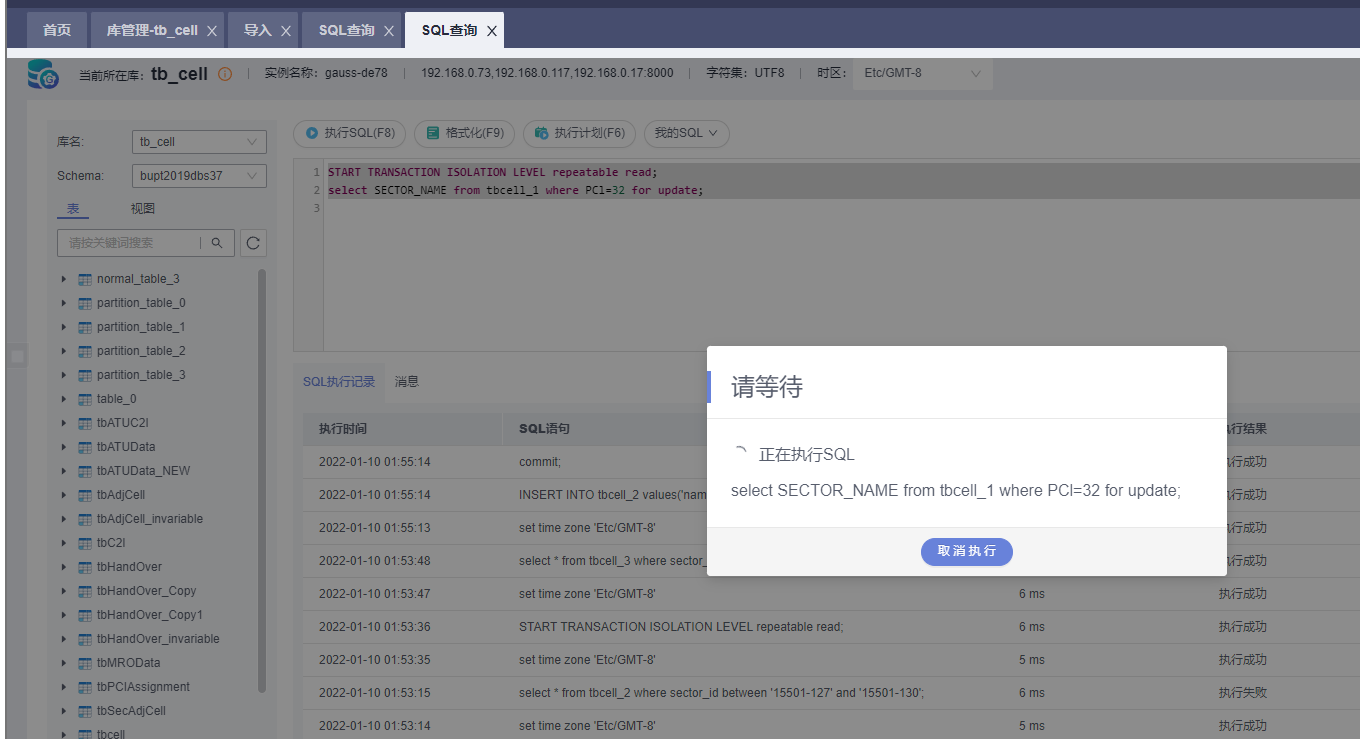




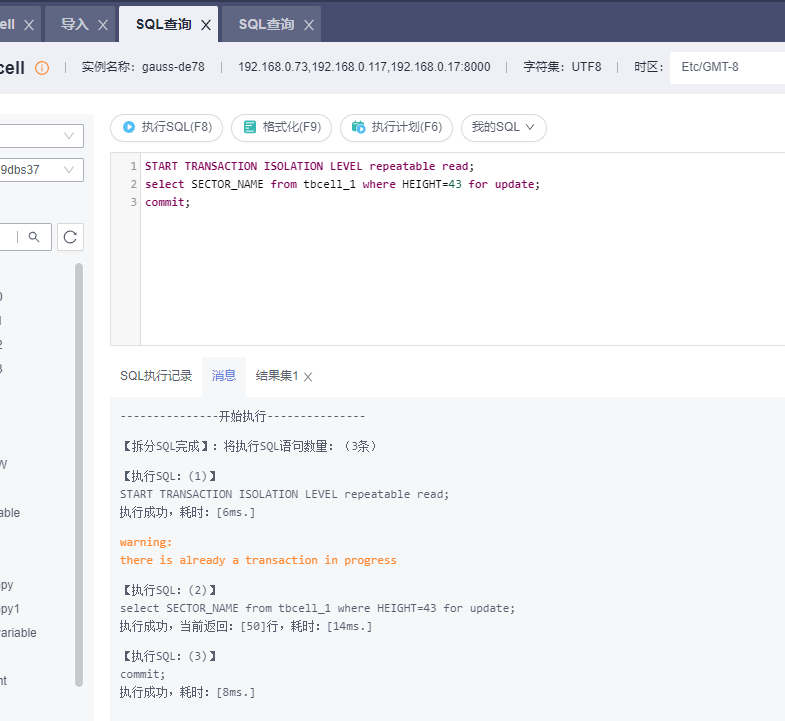
在session2中开启一个事务，也是执行上述select for update语句。此时检测到符合筛选条件的部分数据行上加了互斥锁(两次查询的数据行有重叠，存在PCI=32且HEIGHT=43的数据行)，查询进入申请锁的等待阶段。

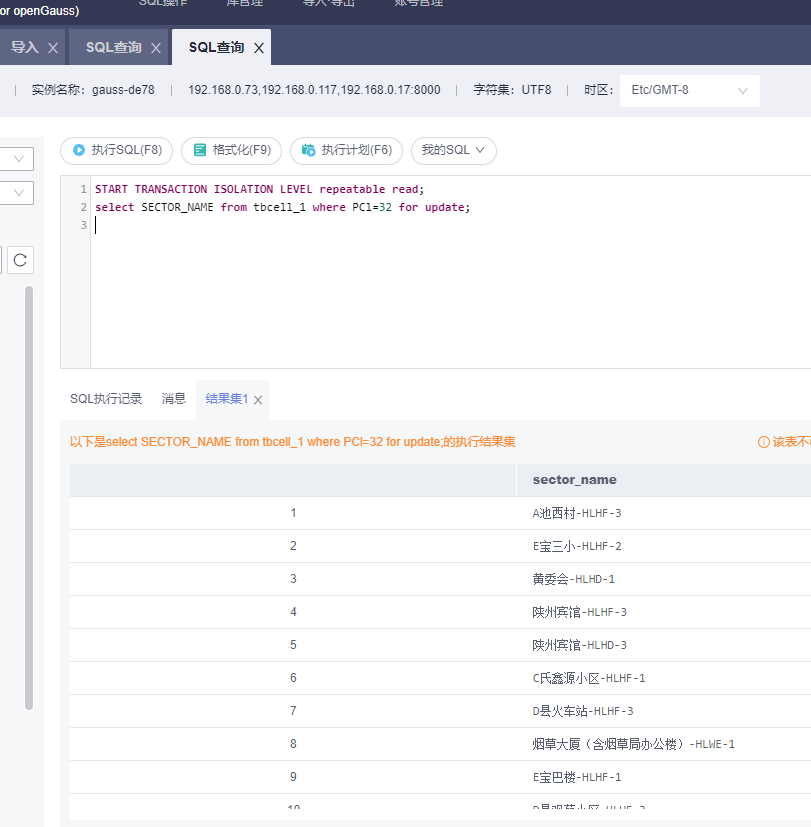
START TRANSACTION ISOLATION LEVEL repeatable read;

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



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。





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的业务的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。

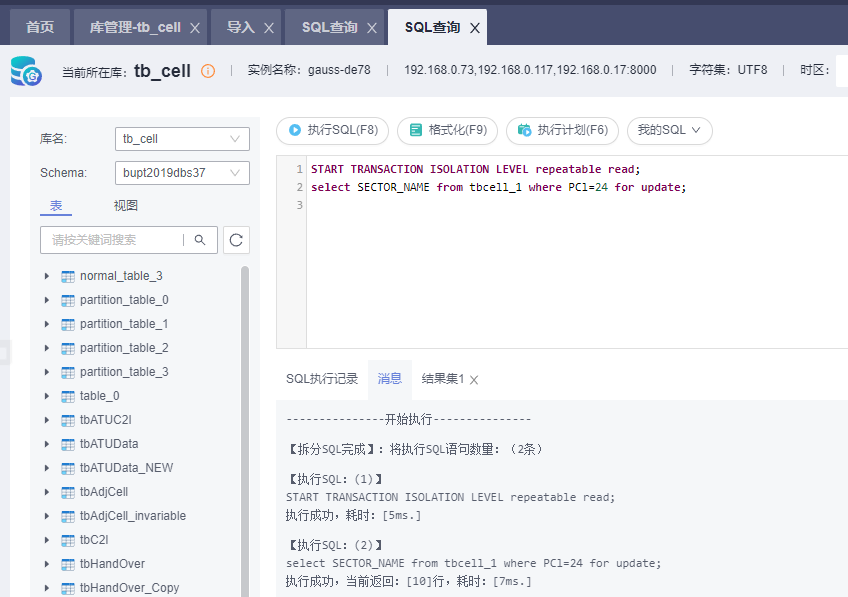
START TRANSACTION ISOLATION LEVEL repeatable read;

select SECTOR\_NAME from tbcell\_1 where SECTOR\_ID='124672-1' for update;





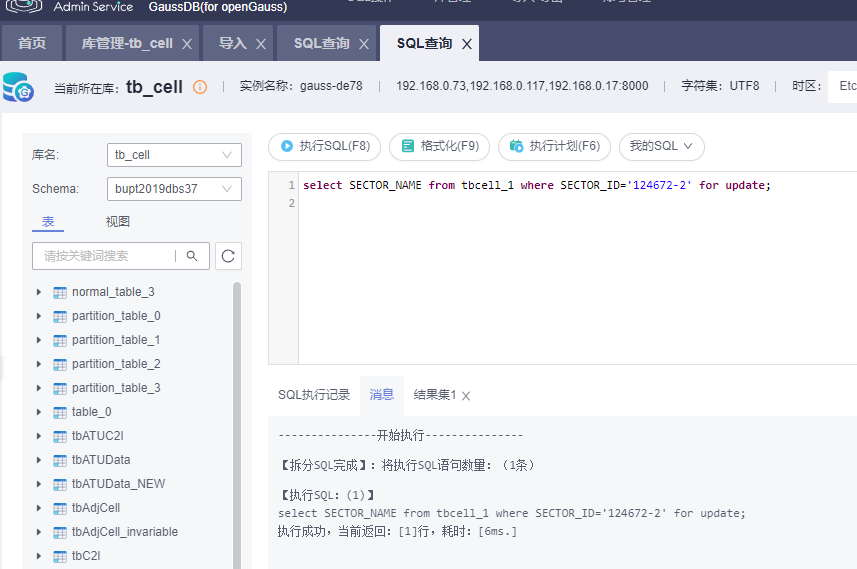
在session2中开启一个事务，也是执行上述select for update语句。



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;

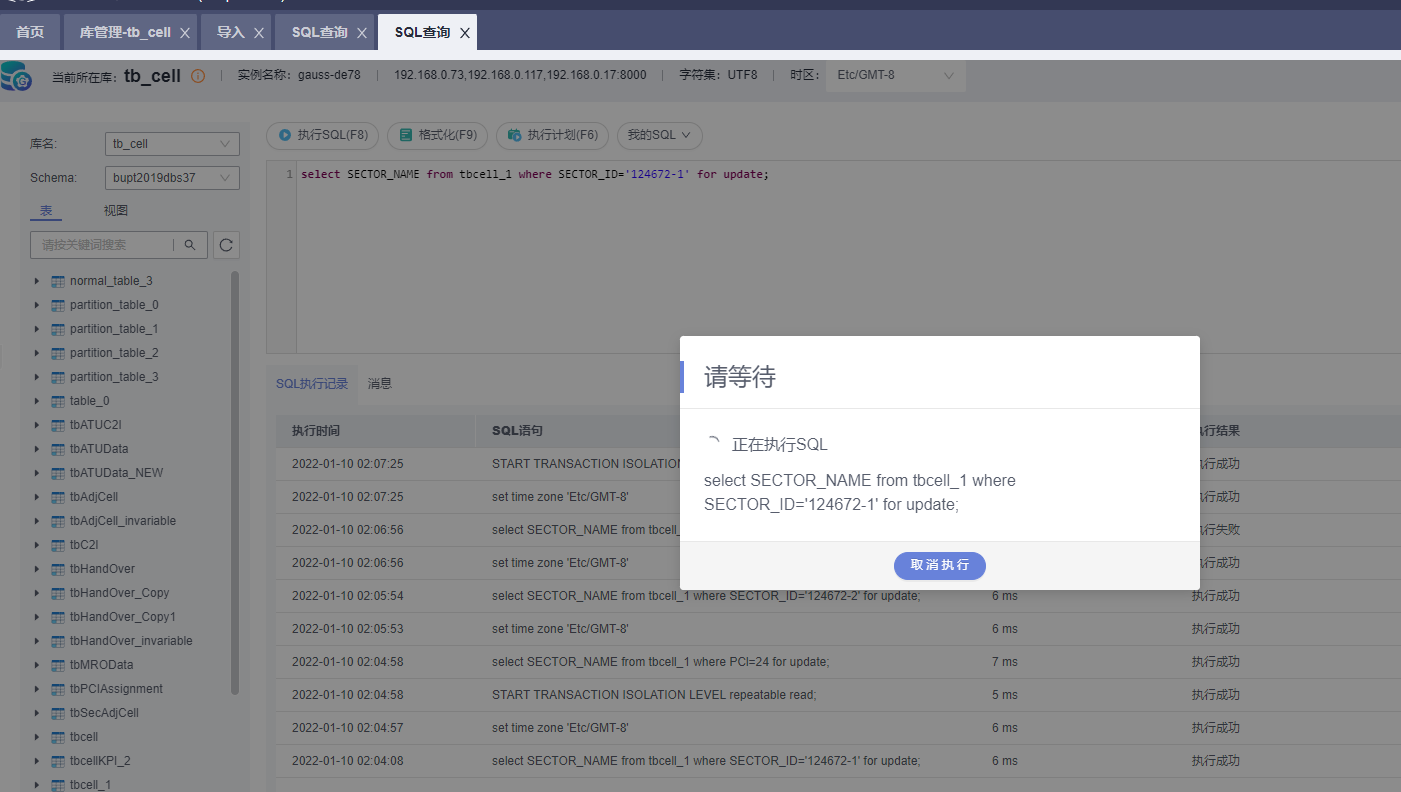


正常查询，没有发生锁等待，说明select for update语句，不是对查询条件所在的列加互斥锁。

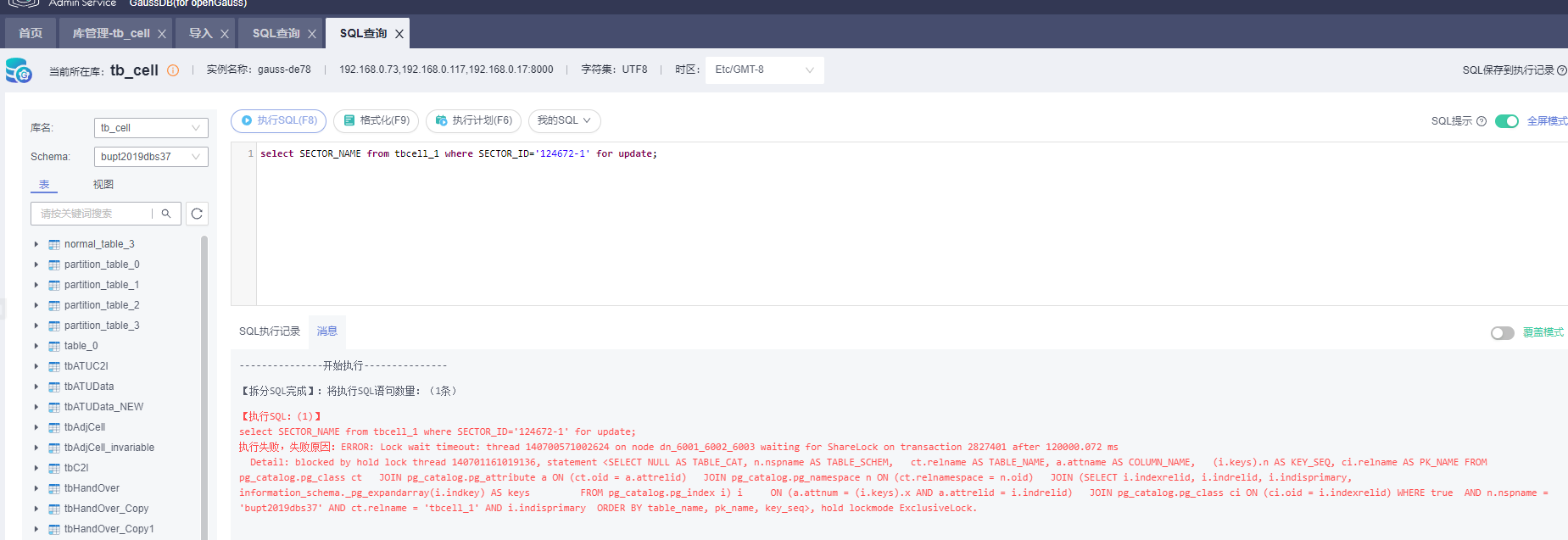


在session2上，继续执行上述select for update语句，这次与session1上的查询条件一样

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

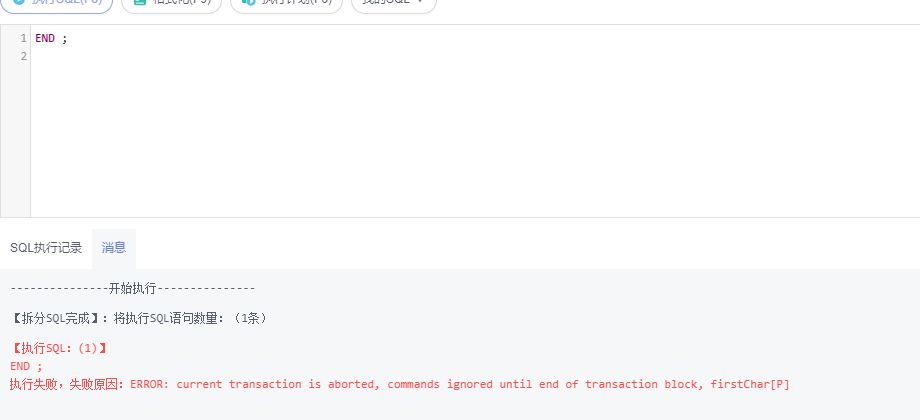
进行备份与恢复。

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：



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

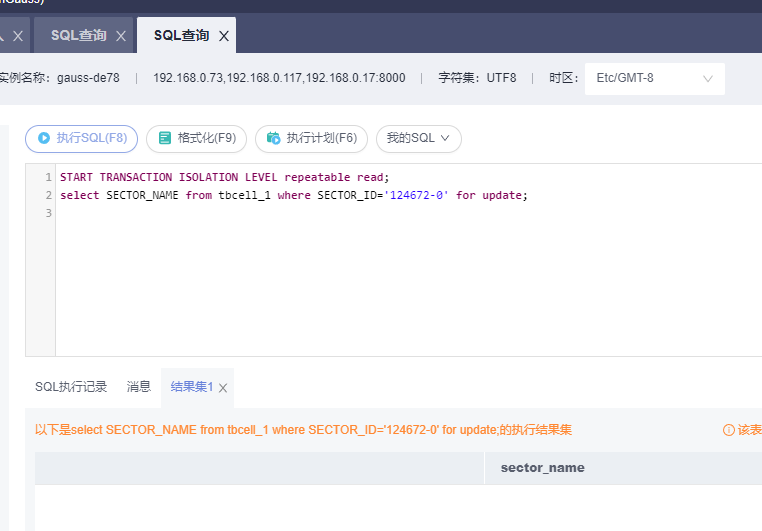
## 问题四：

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



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.