《完整性约束》  
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** |

目录

[一、利用Create table/Alter table语句建立完整性约束 3](#_Toc92331054)

[二、主键/空值/check/默认值约束验证 3](#_Toc92331055)

[1、主键约束 3](#_Toc92331056)

[2、空值 5](#_Toc92331057)

[3、Check约束 7](#_Toc92331058)

[4、默认值 8](#_Toc92331059)

[三、外键/参照完整性约束验证 8](#_Toc92331060)

[1、参照完整性约束验证 8](#_Toc92331061)

[2、级联外键关联下数据访问 10](#_Toc92331062)

[四、函数依赖分析验证 13](#_Toc92331063)

[方案一： 13](#_Toc92331064)

[方案二： 13](#_Toc92331065)

[五、触发器约束 14](#_Toc92331066)

[实验一： 14](#_Toc92331067)

[实验二： 15](#_Toc92331068)

[六、问题及解决 17](#_Toc92331069)

[问题一： 17](#_Toc92331070)

[问题二： 17](#_Toc92331071)

[问题三： 18](#_Toc92331072)

一、利用Create table/Alter table语句建立完整性约束

针对数据库表tbHandOver，创建其副本tbHandOver\_Copy，并在create table中定义主键，默认值，空值和check约束。

执行如下SQL语句：

1. create table "tbHandOver\_Copy"
2. (
3. "CITY" text,
4. "SCELL" varchar(50),
5. "NCELL" varchar(50),
6. "HOATT" **int**,
7. "HOSUCC" **int**,
8. "HOSUCCRATE" **float** **default** null,
9. primary key ("SCELL", "NCELL"),
10. check (("HOATT" >= 0)),
11. check (("HOSUCC" >= 0))
12. );

将tbHandOver的数据复制到tbHandOver\_Copy中，发现tbHandOver\_Copy内容和tbHandOver一致。



二、主键/空值/check/默认值约束验证

1、主键约束

使用分组聚集运算语句，判断是否满足主键约束，可以看出没有重复主键的数据行。

执行如下SQL语句：

1. select "SCELL", "NCELL", count(\*)
2. from "tbHandOver\_Copy"
3. group by ("SCELL", "NCELL")
4. having count(\*) > 1;

查看表中并无主键为空的数据，说明满足主键约束。



查看表中是否有主键为空的数据。

执行如下SQL语句：

1. select \*
2. from "tbHandOver\_Copy"
3. where "SCELL" = null and "NCELL" = null;

结果显示表中无主键为空的数据。

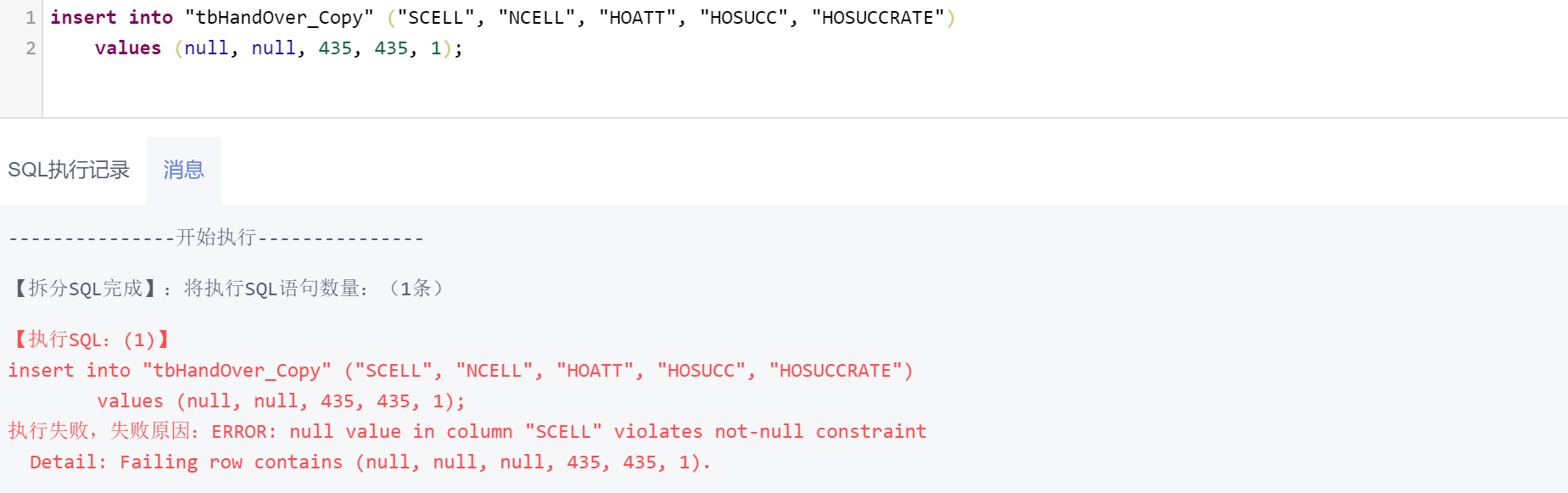


插入主键为空数据，观察DBMS反应。

执行如下SQL语句：

1. **insert** **into** "tbHandOver\_Copy" ("SCELL", "NCELL", "HOATT", "HOSUCC", "HOSUCCRATE")
2. **values** (null, null, 435, 435, 1);

结果显示违背了主键的非空约束，插入失败。

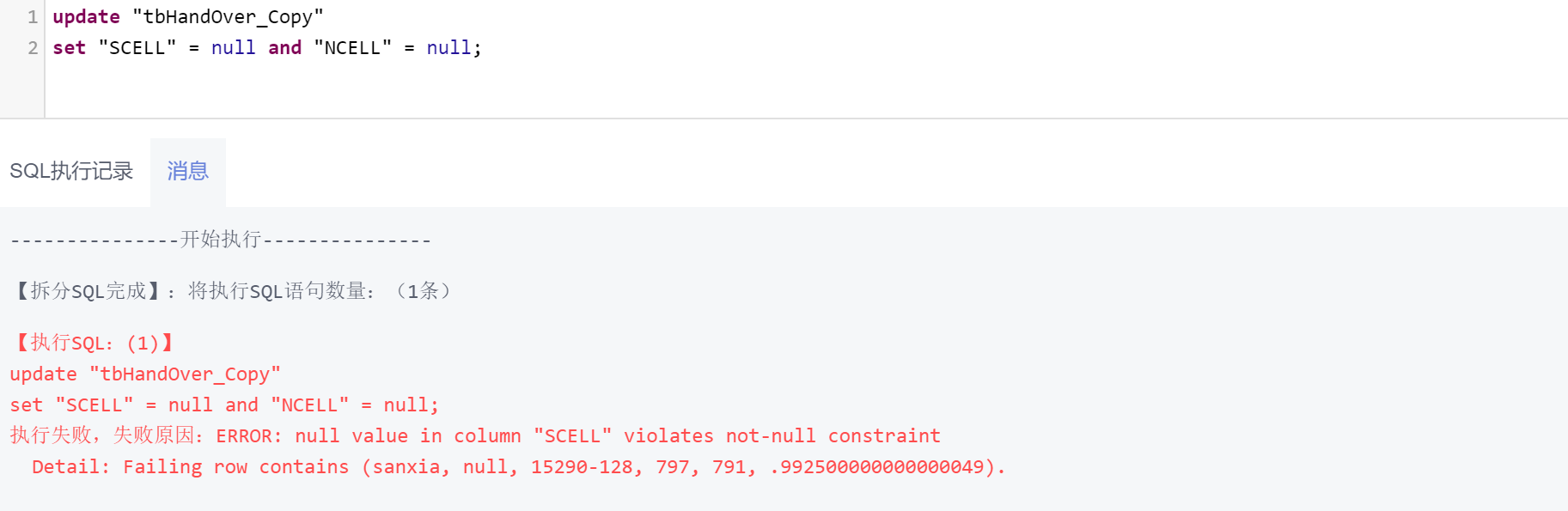


修改原有数据行SCELL，NCELL字段为空。

执行如下SQL语句：

1. **update** "tbHandOver\_Copy"
2. **set** "SCELL" = null and "NCELL" = null;

结果显示违背了主键的非空约束，更新失败。



更新表中SCELL字段值为124711-0和NCELL字段值为124711-1的数据行，将其字段值分别修改为124711-0和15290-128。

执行如下SQL语句：

1. **update** "tbHandOver\_Copy"
2. **set** "SCELL" = '124711-0', "NCELL" = '15290-128'
3. **where** "SCELL" = '124711-0' and "NCELL" = '124711-1';

结果显示主键重复，违背了主键唯一性约束。

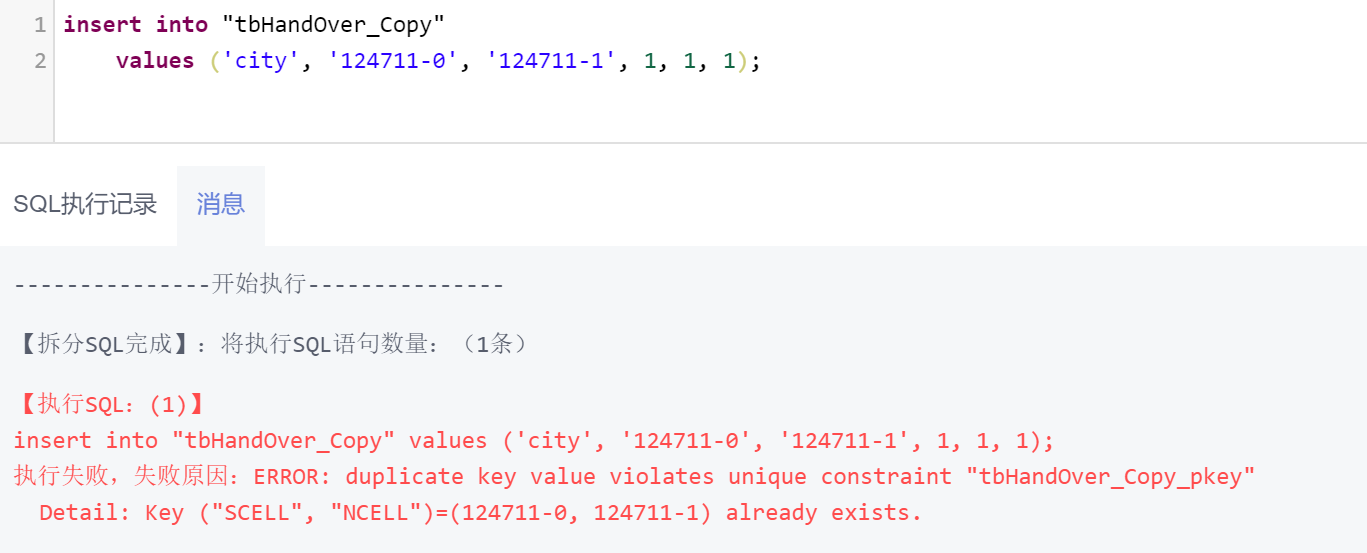


同样地，我们插入主键重复的数据。

执行如下SQL语句：

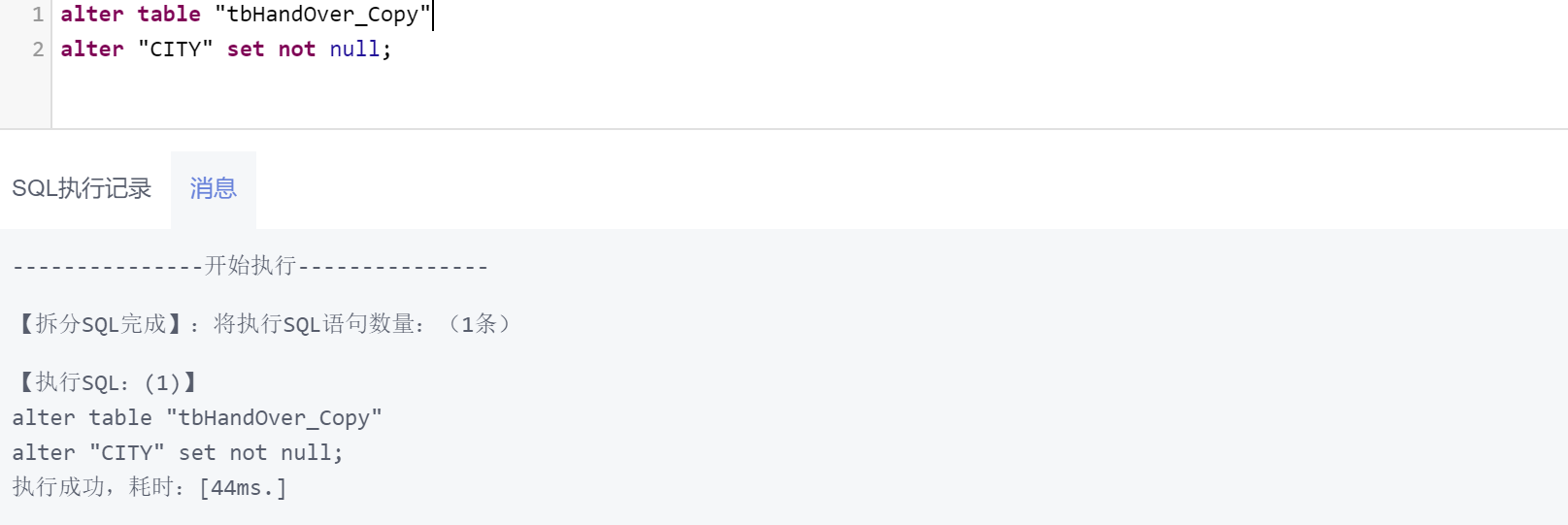
1. **insert** **into** "tbHandOver\_Copy"
2. **values** ('city', '124711-0', '124711-1', 1, 1, 1);

结果显示主键重复，插入失败。



2、空值

首先通过alter table添加not null约束。



插入一数据行，其CITY字段为空。

执行如下SQL语句：

1. **insert** **into** "tbHandOver\_Copy"
2. **values** (null, '12', '13', 1, 1, 1);

结果显示违背CITY非空约束，插入失败。

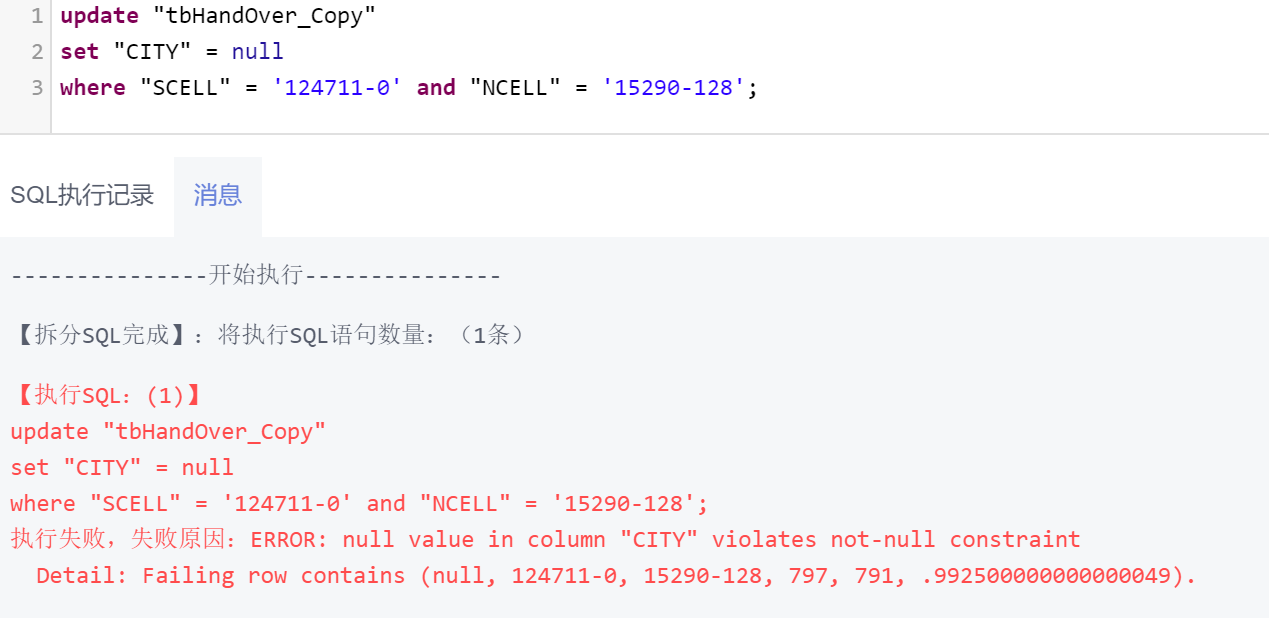


更新其中一行数据，修改其CITY字段为空。

执行如下SQL语句：

1. **update** "tbHandOver\_Copy"
2. **set** "CITY" = null
3. **where** "SCELL" = '124711-0' and "NCELL" = '15290-128';

结果显示违背CITY非空约束，插入失败。



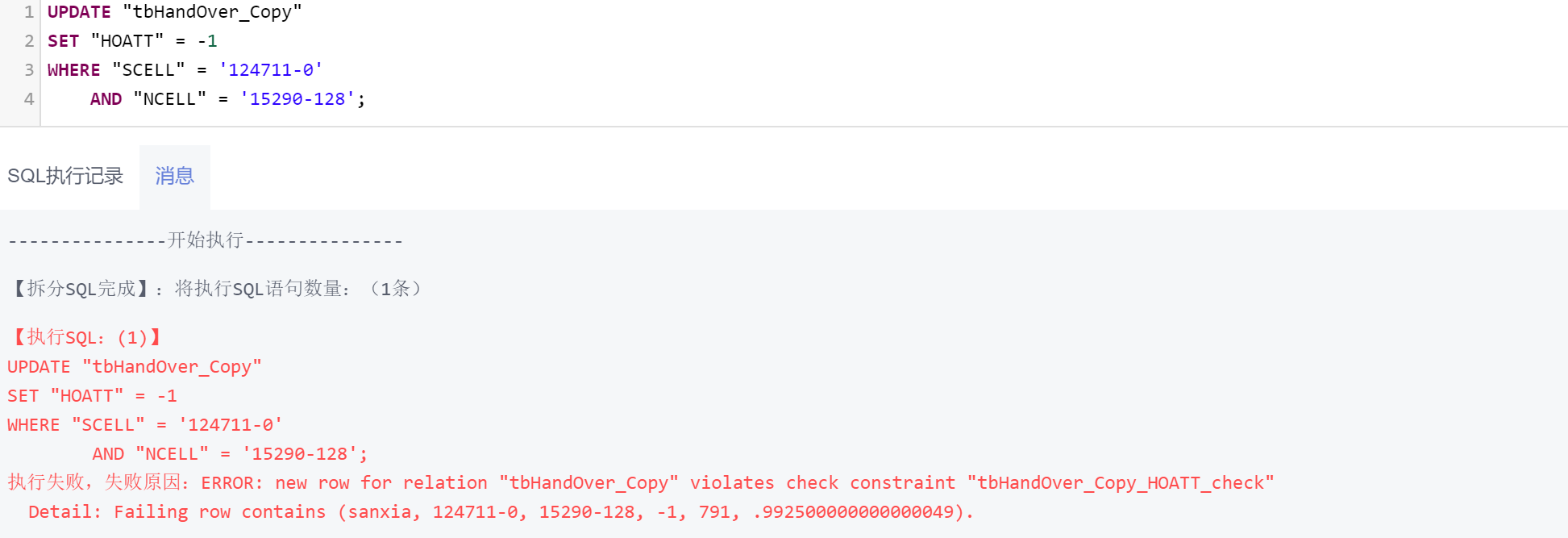
3、Check约束

针对HOATT属性上的check约束关系，修改SCELL和NCELL字段值为124711-0和15290-128的数据，将其HOATT改为-1。

执行SQL语句如下：

1. **UPDATE** "tbHandOver\_Copy"
2. **SET** "HOATT" = -1
3. **WHERE** "SCELL" = '124711-0'
4. AND "NCELL" = '15290-128';

结果显示违背HOATT\_check约束，更新失败。

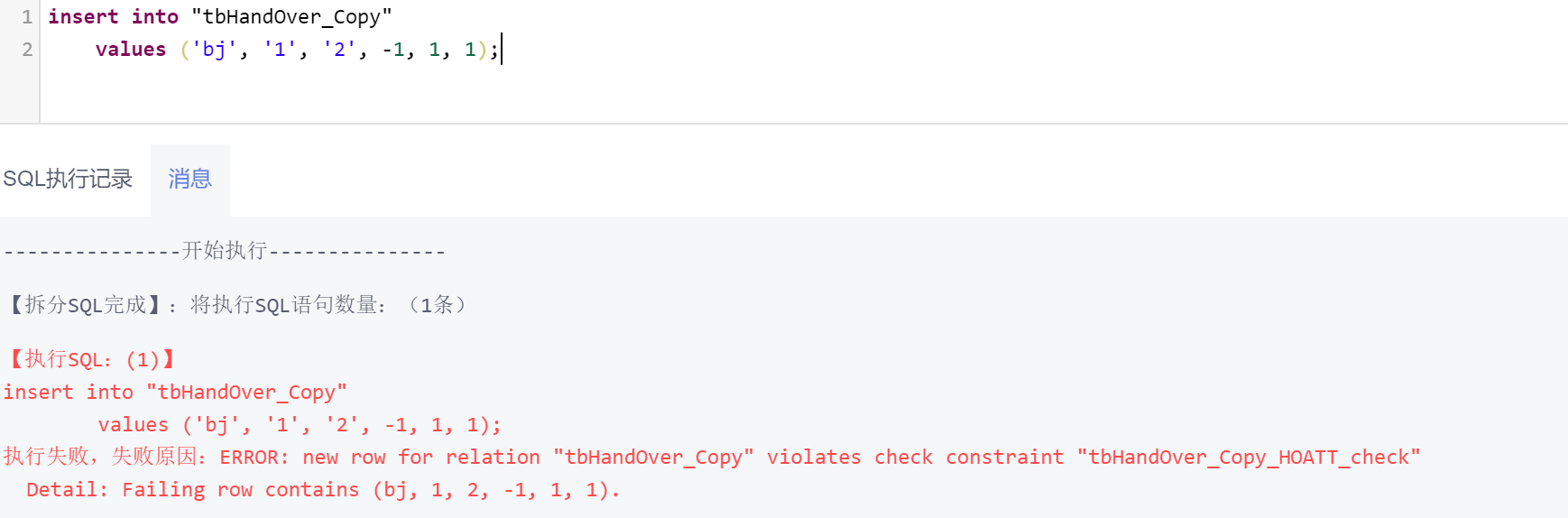


同样的，插入不满足HOATT>=0的数据行。

执行SQL语句如下：

1. **insert** **into** "tbHandOver\_Copy"
2. **values** ('bj', '1', '2', -1, 1, 1);

结果显示违背HOATT\_check约束，插入失败。



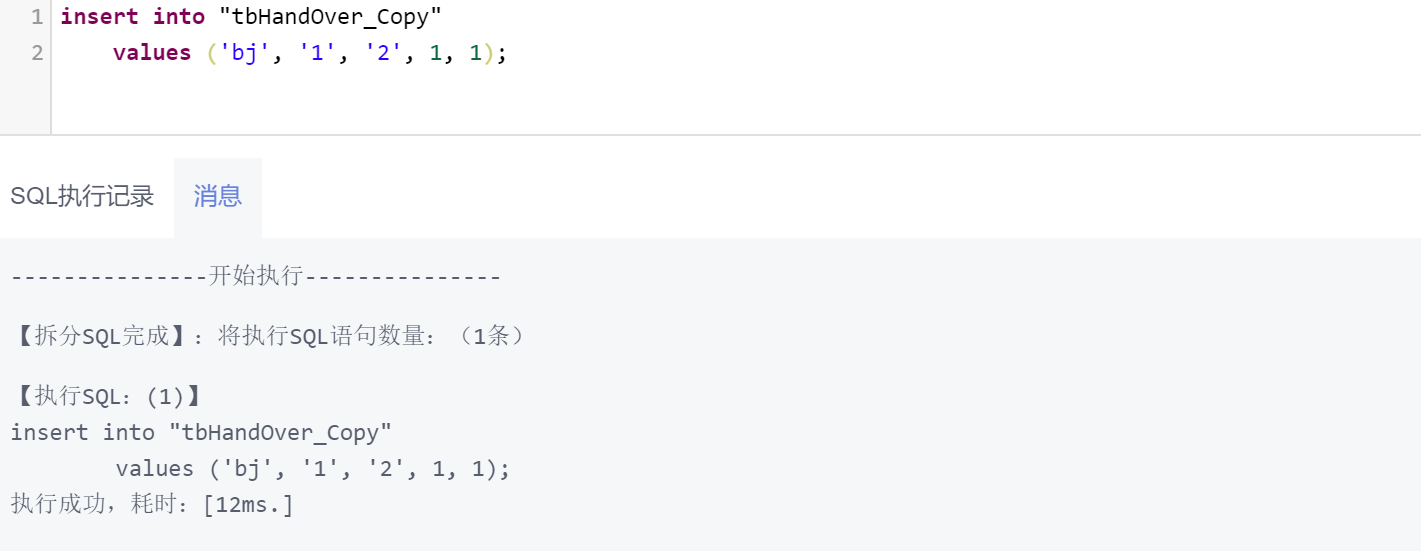
4、默认值

字段HOSUCCRATE默认值为null，向表中插入一行数据，不给出HOSUCCRATE字段。

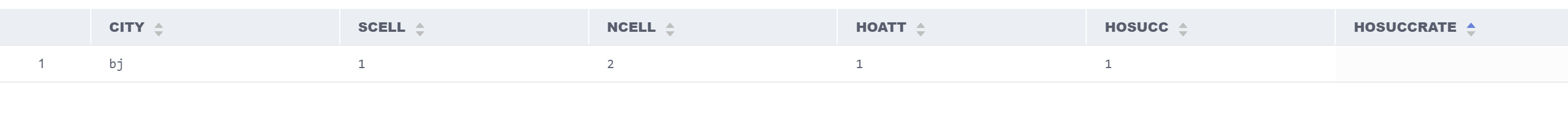
执行SQL语句如下：

1. **insert** **into** "tbHandOver\_Copy"
2. **values** ('bj', '1', '2', 1, 1);

结果显示插入成功。



查询刚插入的数据，发现其HOSUCCRATE被设成默认值空值。



三、外键/参照完整性约束验证

1、参照完整性约束验证

首先，判断tbHandOver\_Copy在属性NCELL上的取值是否都出现在tbCell表的SECTOR\_ID列中。

执行如下SQL语句：

1. **select** "NCELL"
2. **from** "tbHandOver\_Copy"
3. **where** "NCELL" not in (**select** "SECTOR\_ID"
4. **from** "tbcell\_invariable");

结果不为空，说明两张表间不满足参照完整性约束。



去除上述SQL语句中查出的NCELL所在元组，使得两表间参照完整性约束关系成立。

执行如下SQL语句：

1. **delete** **from** "tbHandOver\_Copy" **as** t1
2. **where** t1."NCELL" not in (**select** "SECTOR\_ID"
3. **from** "tbcell\_invariable");

执行成功！



再次执行查询：

1. **select** "NCELL"
2. **from** "tbHandOver\_Copy"
3. **where** "NCELL" not in (**select** "SECTOR\_ID"
4. **from** "tbcell\_invariable");

结果显示为空，说明两表已经满足参照完整性约束。

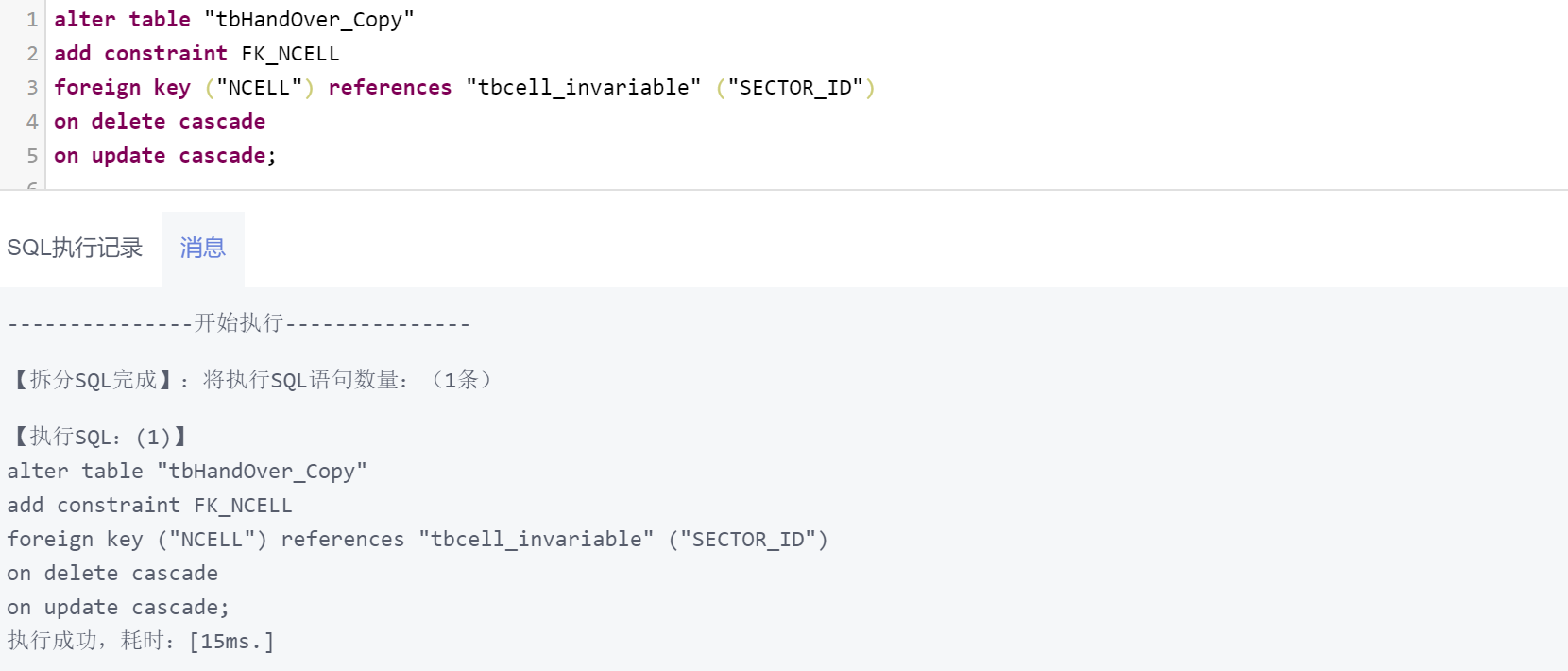


定义tbHandOver和tbCell之间的级联关联。

执行如下SQL语句：

1. **alter** **table** "tbHandOver\_Copy"
2. **add** **constraint** FK\_NCELL
3. **foreign** **key** ("NCELL") **references** "tbcell\_invariable" ("SECTOR\_ID")
4. **on** **delete** **cascade**
5. **on** **update** **cascade**;

执行成功，表明级联外键定义成功。



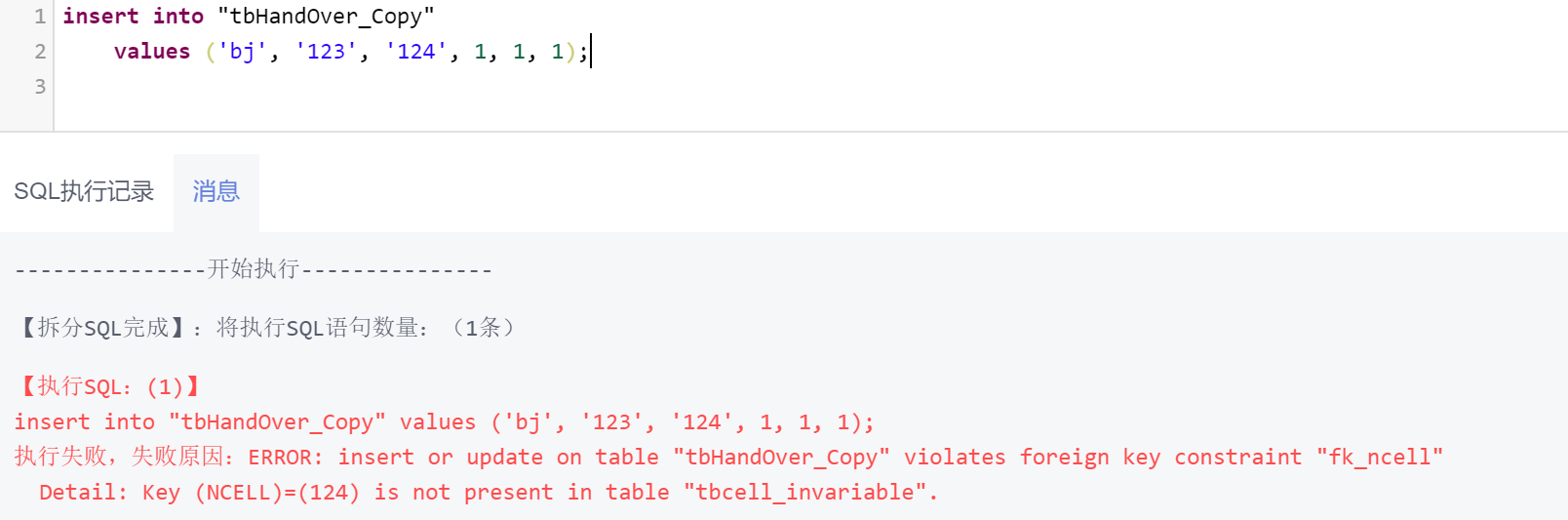
2、级联外键关联下数据访问

建立好级联关系后，向tbHandOver表中插入一行数据，其NCELL值设为124。

执行如下SQL语句：

1. **insert** **into** "tbHandOver\_Copy"
2. **values** ('bj', '123', '124', 1, 1, 1);

结果显示，由于tbCell表中不存在SECTOR\_ID为124的数据行，违反了外键约束，因此插入失败。



再向tbHandOver表中插入一行数据，其NCELL值设为124672-0。

执行如下SQL语句：

1. **insert** **into** "tbHandOver\_Copy"
2. **values** ('bj', '123', '124672-0', 1, 1, 1);

结果显示，由于tbCell表中存在SECTOR\_ID为124672-0的数据行，不违反外键约束，因此插入成功。



向tbCell中插入一行SECTOR\_ID为12的数据，而tbHandOver中不存在NCELL为12的数据，插入成功。



将tbHandOver表中一行NCELL值为124711-1的数据的NCELL值修改为13。

执行如下SQL语句：

1. **update** "tbHandOver\_Copy"
2. **set** "NCELL" = '13'
3. **where** "NCELL" = '124711-1';

结果显示，tbCell表中并没有SECTOR\_ID为13的数据行，因此违反了外键约束，更新失败。

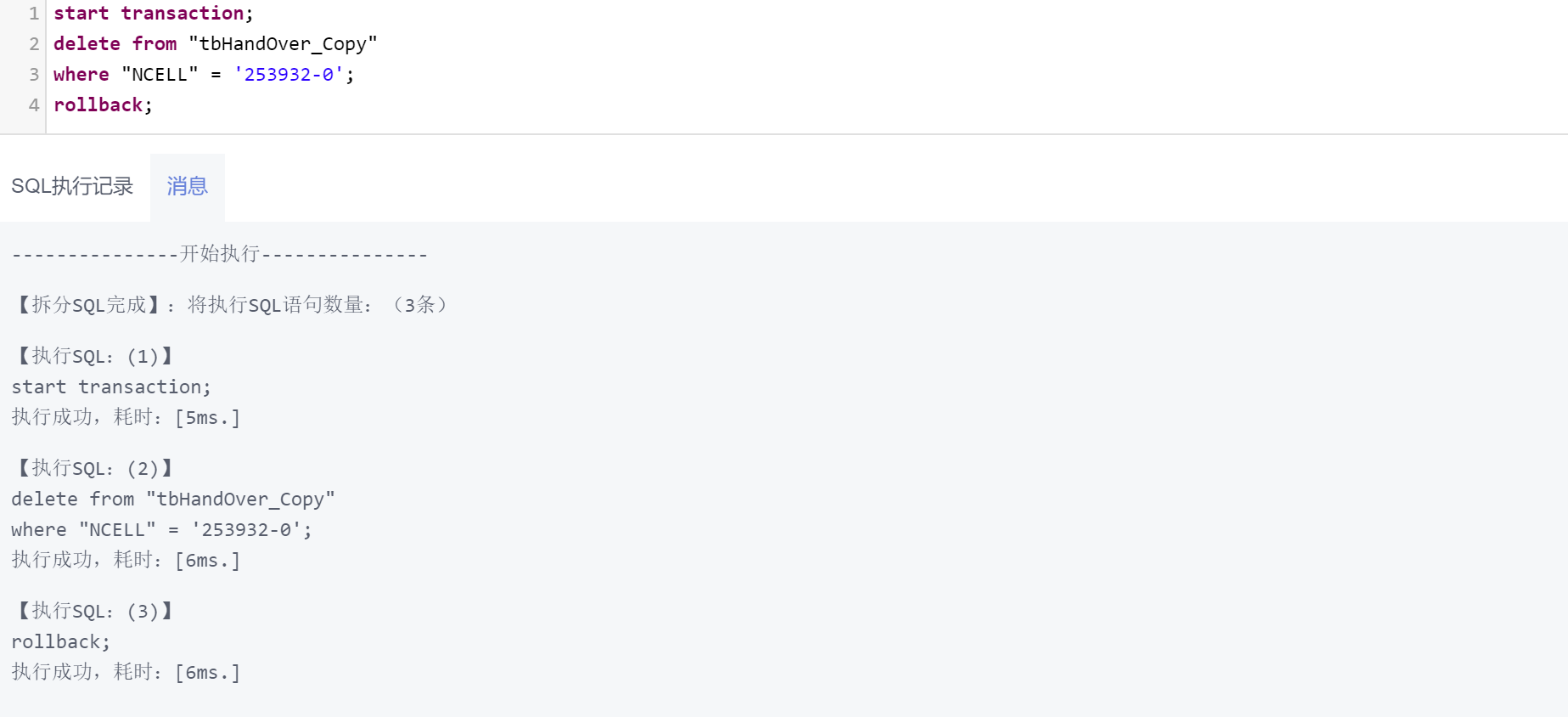
将tbHandOver表中一行NCELL值为124711-1的数据的NCELL值修改为124672-0。

执行如下SQL语句：

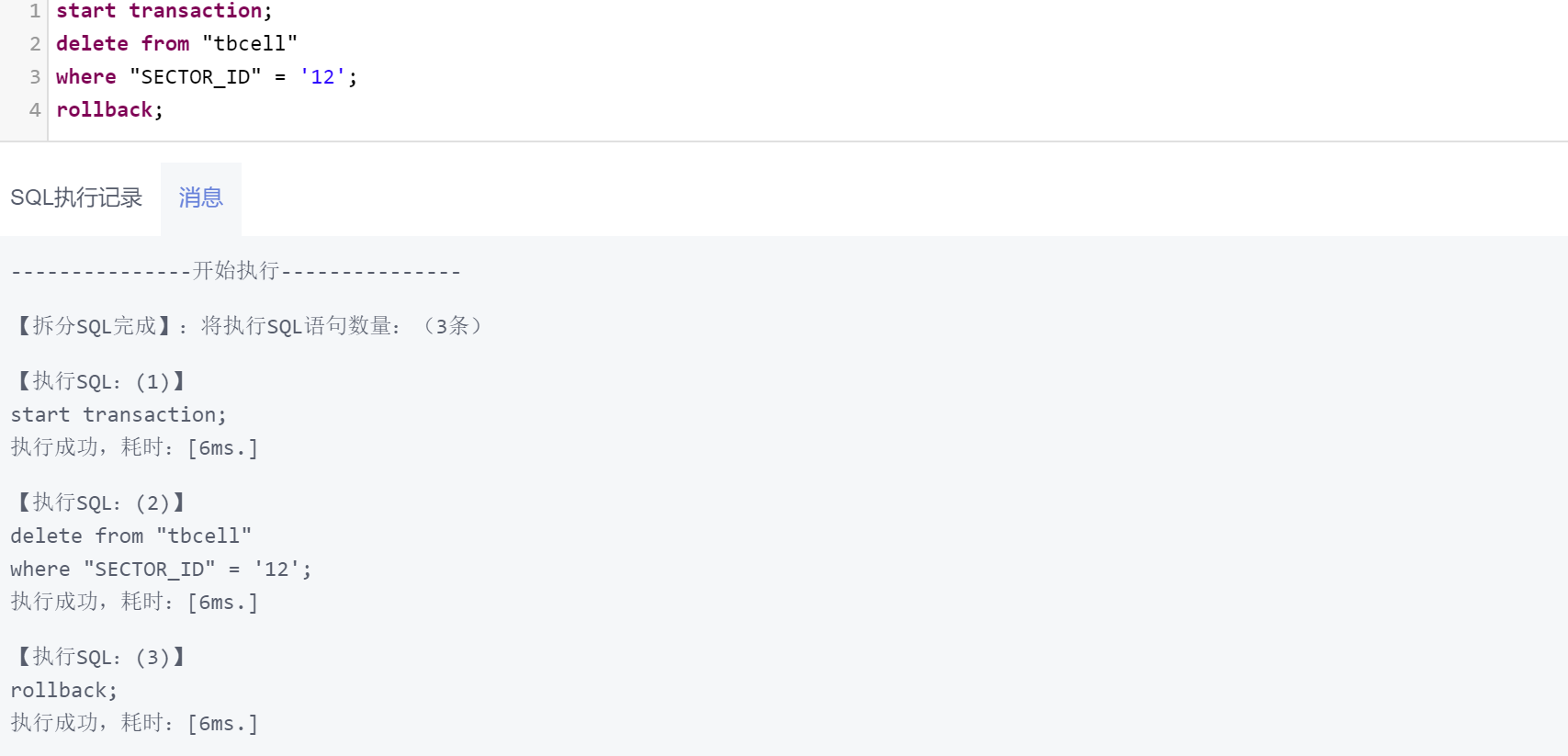
1. **update** "tbHandOver\_Copy"
2. **set** "NCELL" = '124672-0'
3. **where** "NCELL" = '124711-1';

结果显示，tbCell表中已有SECTOR\_ID值为124672-0的数据行，因此执行成功。

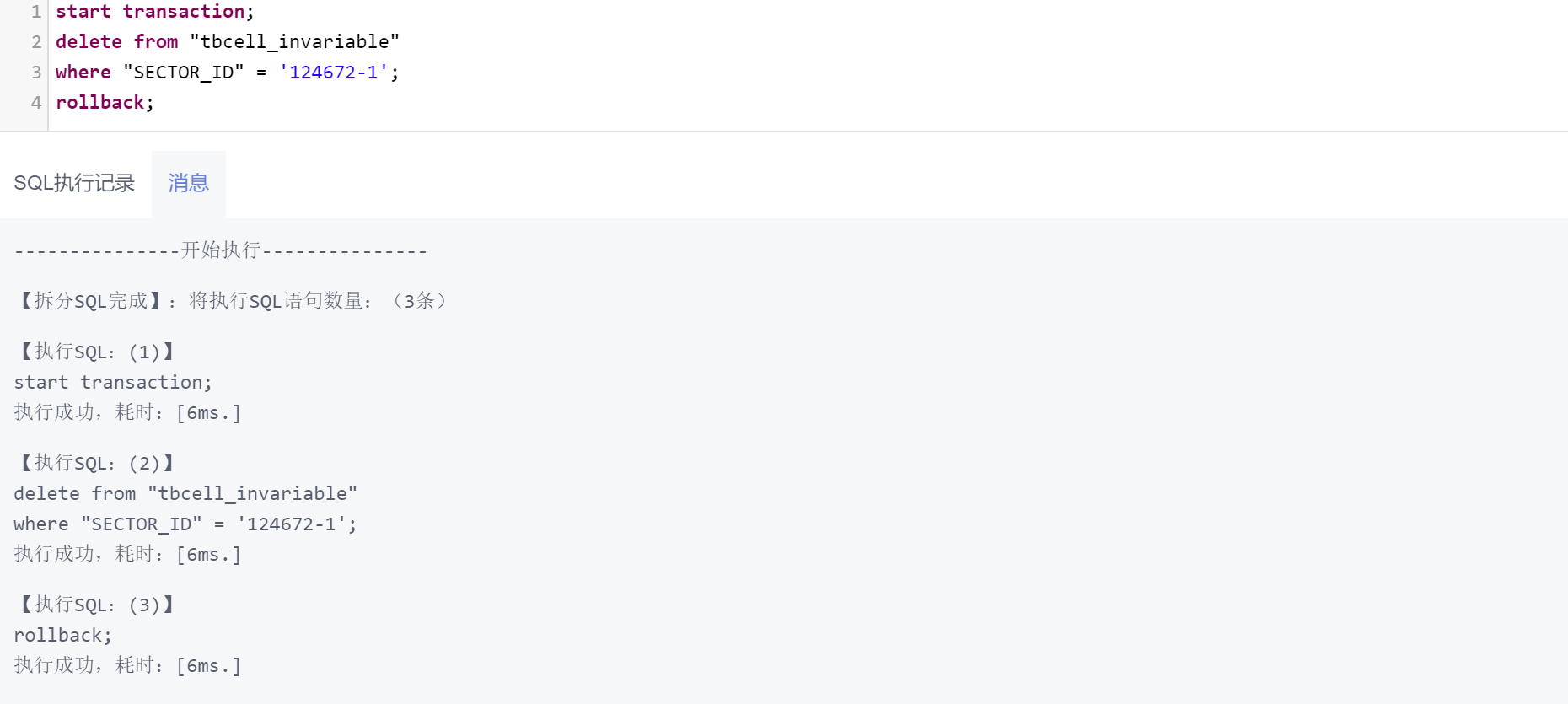
删除tbHandOver表中NCELL字段值为的数据行，执行成功。



从tbCell表中删除SECTOR\_ID为12的数据行，表tbHandOver中不存在NCELL为12的数据行，执行成功。



从tbCell表中删除SECTOR\_ID值为124672-1的数据行，表tbHandOver存在NCELL值为的数据行，执行成功。



四、函数依赖分析验证

在PCI优化分配表tbPCIAssignment中，验证函数依赖ENODEB\_ID->PCI是否成立。

方案一：

直接选取ENODB\_ID值相等，但PCI字段不同的元组进行查看；若存在结果不为空，则其之间不满足函数依赖。

执行如下SQL语句：

1. **select** \*
2. **from** "tbPCIAssignment" **as** t1, "tbPCIAssignment" **as** t2
3. **where** (t1."ENODEB\_ID" = t2."ENODEB\_ID") and (t1."PCI" <> t2."PCI");

结果不为空，说明该函数依赖不成立。



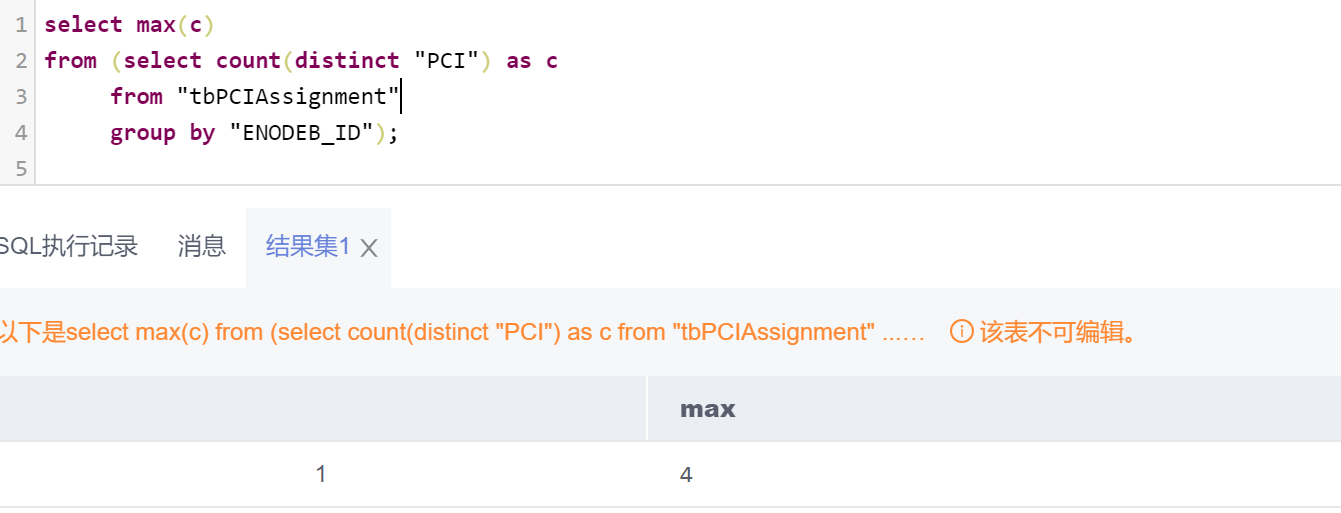
方案二：

首先对ENODEB\_ID分组，对于有相同ENODEB\_ID的元组，统计去重之后的PCI值，对于每组结果，只要不同的PCI值的数量大于1，则说明相同的ENODEB\_ID对于不同得PCI，不满足函数依赖，若所有分组结果的最大值都为1，则满足函数依赖，否则不满足。

执行如下SQL语句：

1. **select** **max**(c)
2. **from** (**select** count(**distinct** "PCI") **as** c
3. **from** "tbPCIAssignment"
4. **group** **by** "ENODEB\_ID");

结果显示最大值为4，所以不满足函数依赖。



五、触发器约束

实验一：

在tbHandOver上定义触发器，用于在插入数据时根据HOATT和HOSUCC字段的值进而计算出HOSUCCRATE的值。

首先定义触发器函数，如果HOATT字段为0，则HOSUCCRATE赋值为空；否则HOSUCCRATE=HOSUCC/HOATT。

执行如下SQL语句：

1. **create** **function** insert\_trig\_func\_1() **returns** **trigger** **as** $$
2. **begin**
3. if new."HOATT" = 0 **then**
4. new."HOSUCCRATE" := null;
5. **else**
6. new."HOSUCCRATE" := new."HOSUCC" / new."HOATT";
7. **end** if;
8. **return** new;
9. **end**;
10. $$ language plpgsql;

接着定义触发器，在PostgreSQL中，触发器对每一行调用一个过程。

执行如下SQL语句：

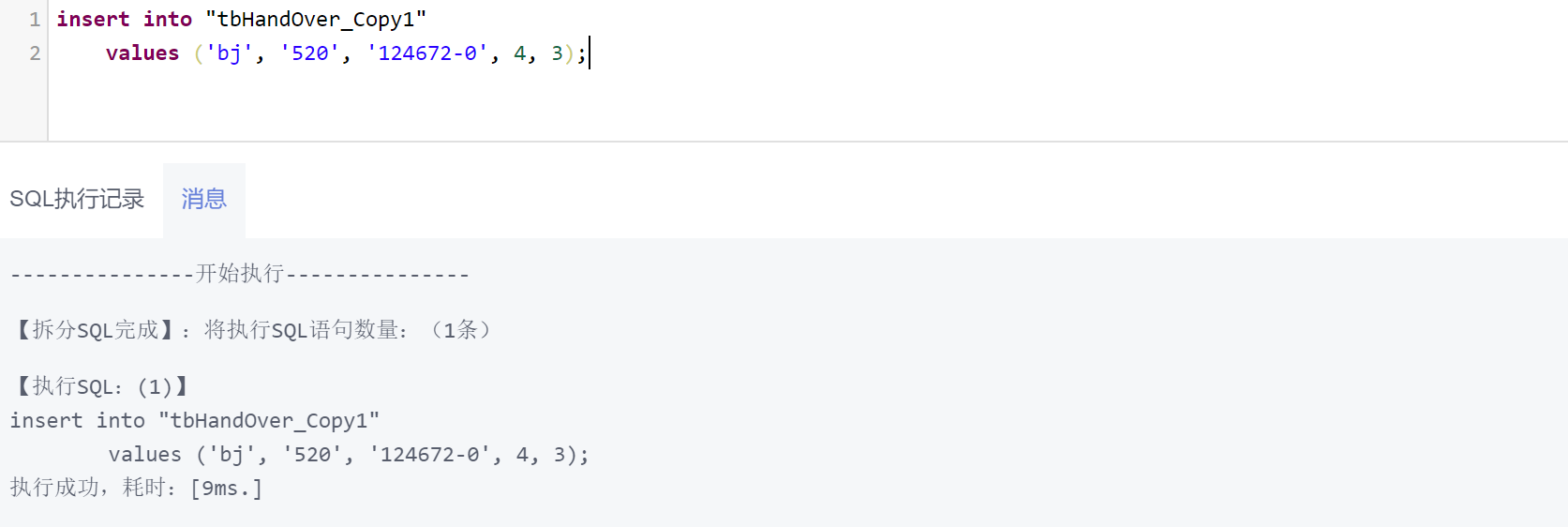
1. **create** **trigger** insert\_trig\_before\_1 before **insert** **on** "tbHandOver\_Copy1"
2. **for** each row **execute** **procedure** insert\_trig\_func\_1();

下面向tbHandOver里插入一行数据，HOATT的值不为0。

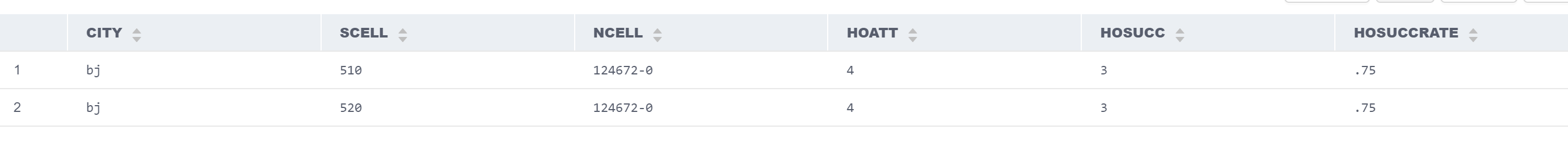
执行如下SQL语句：

1. **insert** **into** "tbHandOver\_Copy1"
2. **values** ('bj', '520', '124672-0', 4, 3);

结果显示插入成功。



打开表tbHandOver，我们发现数据已经成功插入，且HOSUCCRATE的值已经被计算。



下面，继续插入一个HOATT字段值为0的数据行。

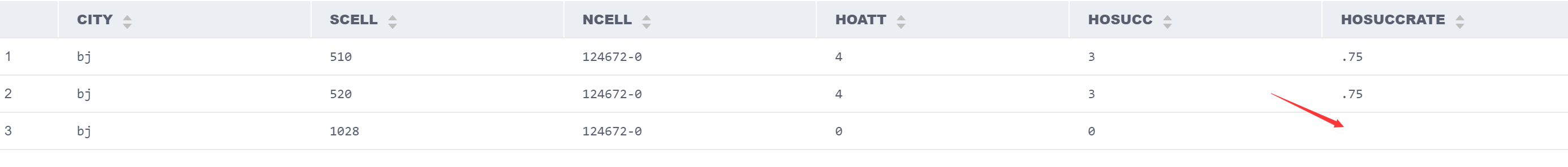
执行如下SQL语句：

1. **insert** **into** "tbHandOver\_Copy1"
2. **values** ('bj', '1028', '124672-0', 0, 0);

插入成功！



打开tbHandOver表，我们发现数据已经成功插入，HOSUCCRATE为空值，完美符合预期结果！



实验二：

在tbHandOver上定义触发器，用于在修改数据时根据HOATT和HOSUCC字段的值进而计算出HOSUCCRATE的值。

首先定义触发器函数，如果HOATT字段为0，则HOSUCCRATE赋值为空；否则HOSUCCRATE=HOSUCC/HOATT。

执行如下SQL语句：

1. **create** **function** update\_trig\_func() **returns** **trigger** **as** $$
2. **begin**
3. if new."HOATT" = 0 **then**
4. new."HOSUCCRATE" := null;
5. **else**
6. new."HOSUCCRATE" := new."HOSUCC" / new."HOATT";
7. **end** if;
8. **return** new;
9. **end**;
10. $$ language plpgsql;

结果显示，定义触发器函数成功。



接着定义触发器，在PostgreSQL中，触发器对每一行调用一个过程。

执行如下SQL语句：

1. **create** **trigger** update\_trig\_before before **update** **on** "tbHandOver\_Copy1"
2. **for** each row **execute** **procedure** update\_trig\_func();

执行成功！

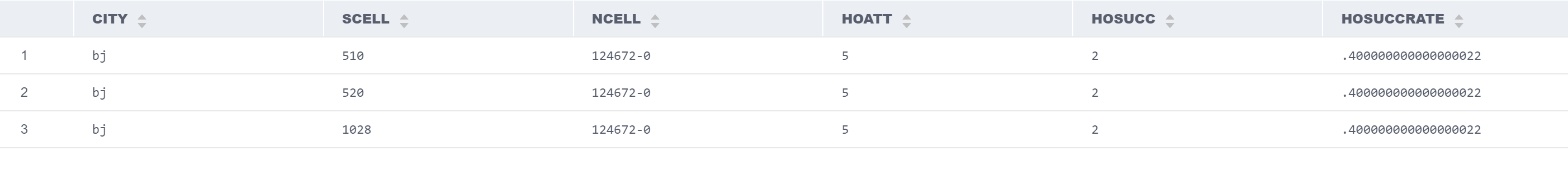


下面在tbHandOver里修改数据，HOATT的值不为0。

执行如下SQL语句：

1. **update** "tbHandOver\_Copy1"
2. **set** "HOATT" = 5, "HOSUCC" = 2
3. **where** "CITY" = 'bj';

打开tbHandOver表，发现数据已经成功更新，HOSUCCRATE的值被自动计算。

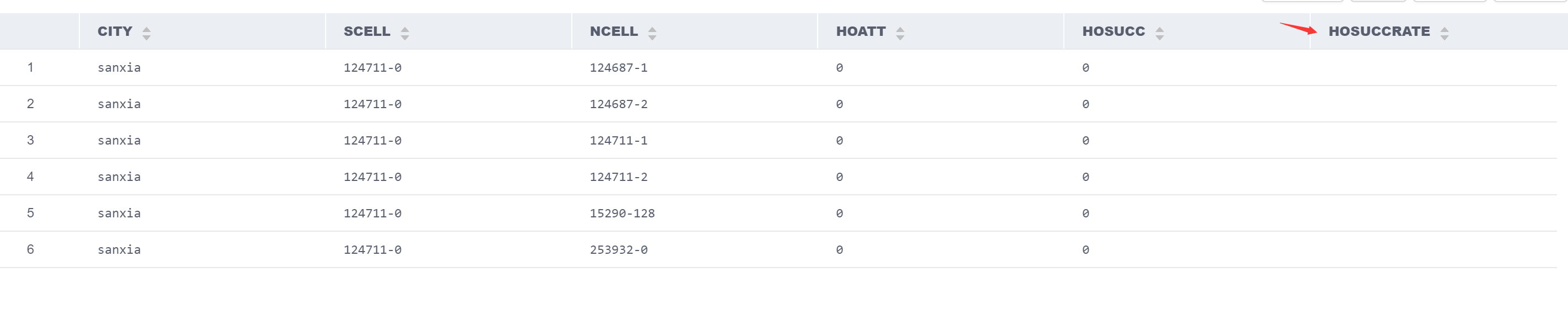


继续修改数据，令HOATT字段为0。

执行如下SQL语句：

1. **update** "tbHandOver\_Copy1"
2. **set** "HOATT" = 0, "HOSUCC" = 0
3. **where** "SCELL" = '124711-0';

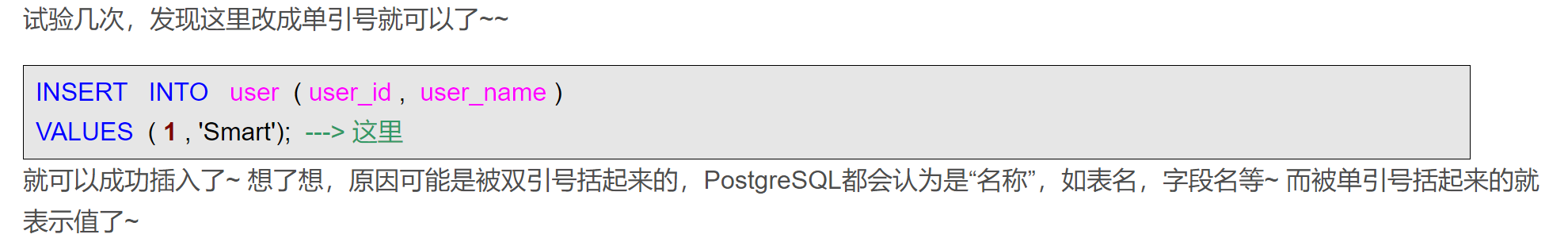
打开tbHandOver表，发现数据已经成功更新，HOSUCCRATE的值为空。



六、问题及解决

问题一：

在主键约束部分，首先出现了类似column "124711-0" does not exist问题，仔细检查并无语法问题。后来试着改成了单引号，便运行成功，查阅资料发现，PostgreSQL会把双引号认为是“名称”。

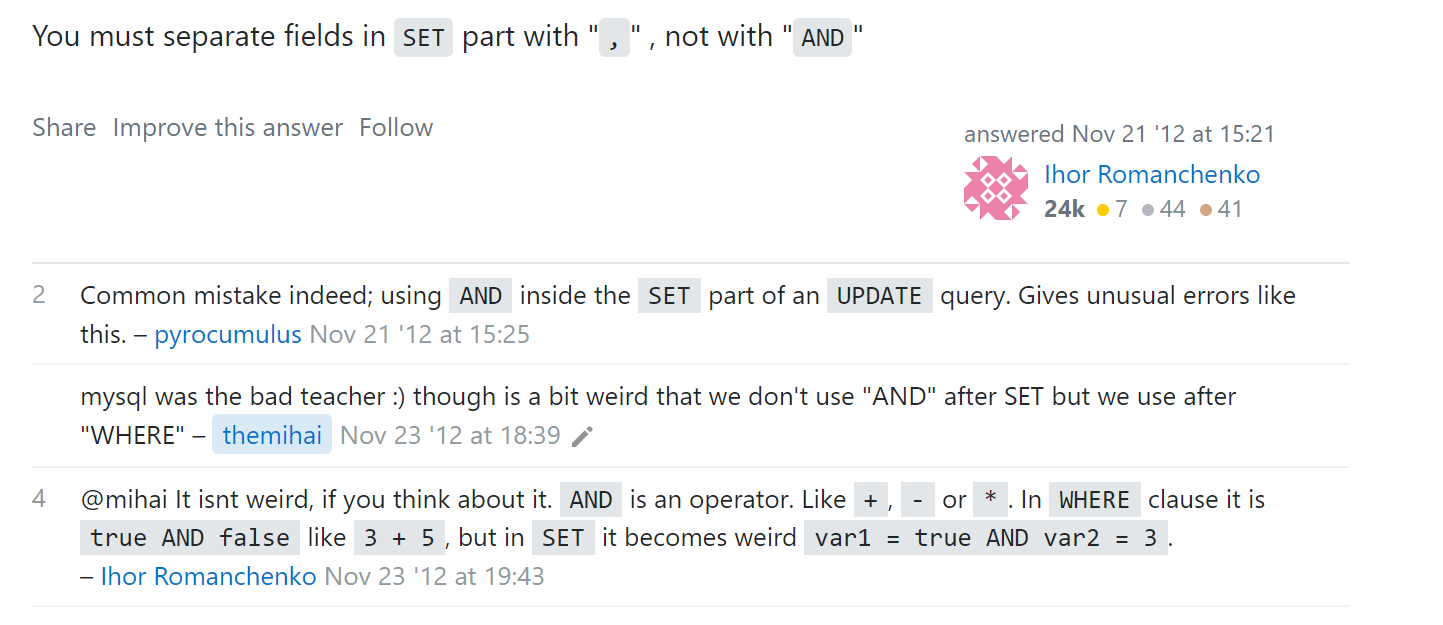


问题二：

在修改数据时，出现了如下SET问题：



因为这本就是主键重复的试验，执行并不会成功，但我却困惑于失败原因，没有显示重复键相关的信息。后来查阅了相关资料，得知需要把and改为逗号，便出现了预期的结果。



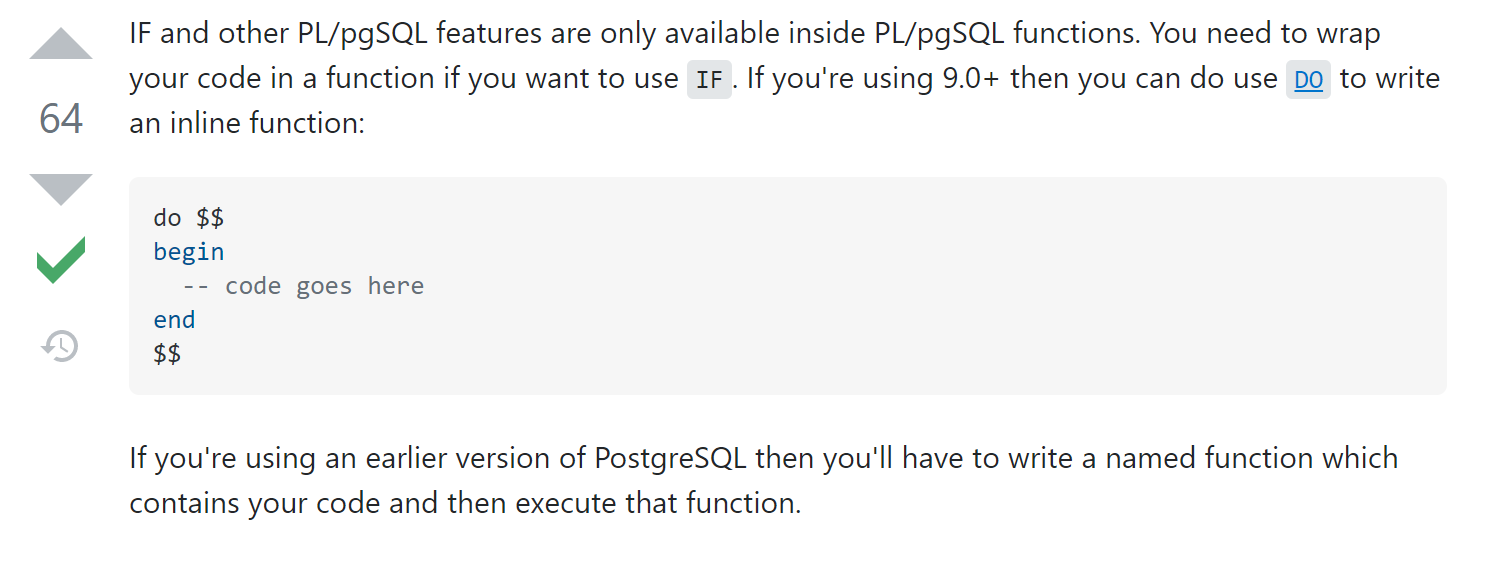
问题三：

在做触发器实验时，首先按照教材上的方法编写SQL语句，发现执行失败。后来得知，PostgreSQL里触发器对每一行调用一个过程，需要添加execute procedure语句。

这样依然执行失败，一直在if附近出现语法错误：

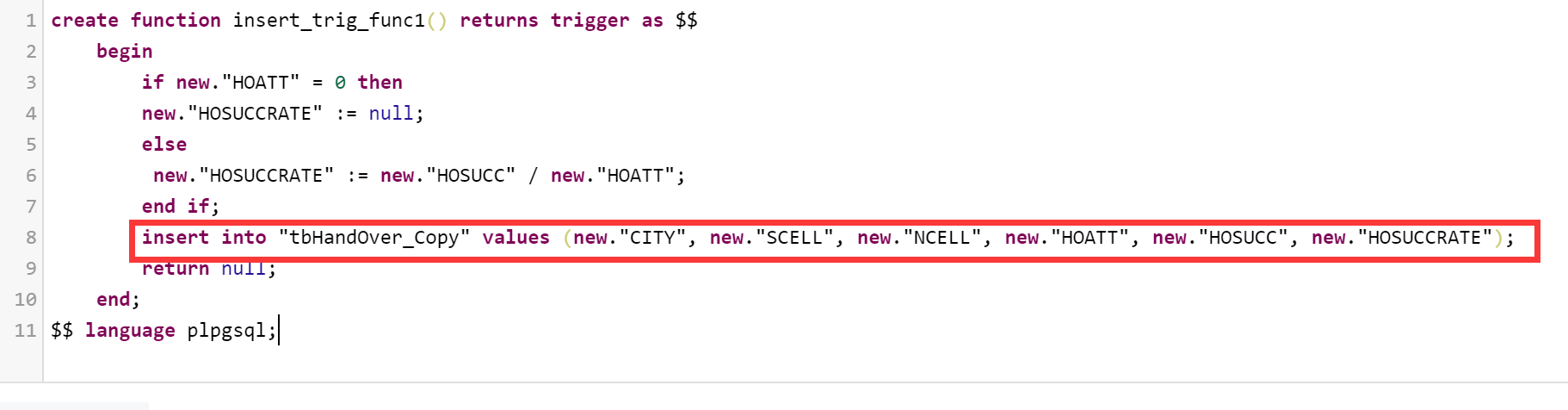


后来经过查阅Stack Overflow得知，在PostgreSQL里，if一般需要放在触发器函数里：

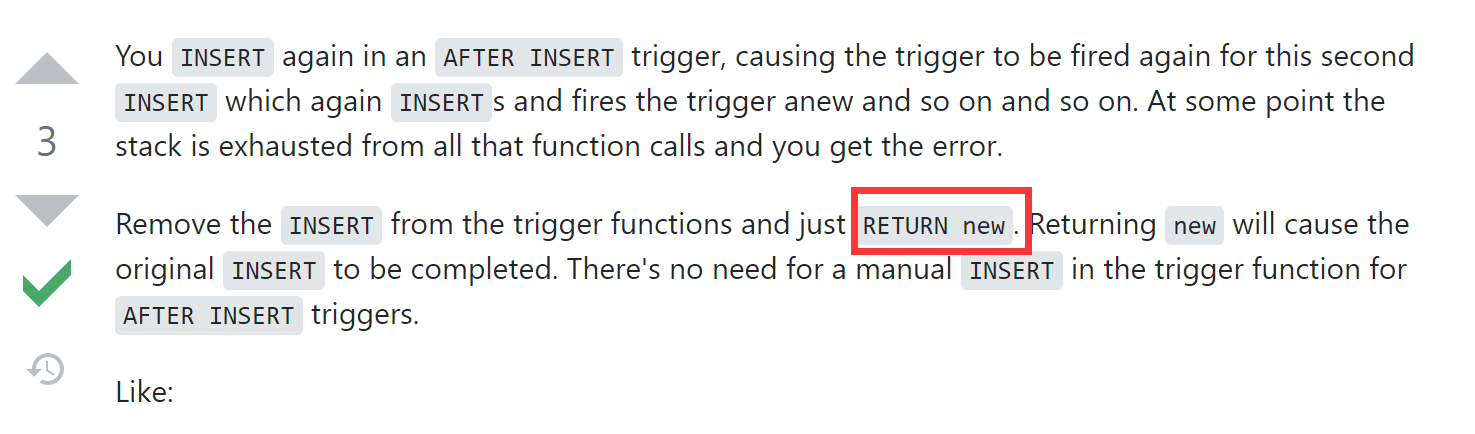


终于，等我编写了触发器函数之后，再通过定义触发器调用触发器函数，执行成功！

此时的触发器函数代码：



但不幸的事情又发生了，当我试图去插入一个数据时，出现了爆栈错误，即Error: stack limit exceeded，我考虑在触发器函数调用过程中，可能发生了死循环。但思考了一会，仍然无法解决这个问题，在Stack Overflow上，我再次获得了解答，在手动调用insert时，当前函数会不断进行循环，将其替换为return new即可。



至此，再插入数据时，便得到了预期中的结果。