**《数据库系统原理》**

**课程实验报告**

徽标

描述已自动生成

**学院：计算机（国家示范性软件）学院**

**班级：** 2021211304

**姓名：** **张梓良 杨晨 苗雨**

**学号：**2021212484 2021212171 2021212492

**前言**

**实验环境说明**

本实验环境为virtualBOX虚拟机openEuler20.03系统上的openGauss1.1.0/openGauss2.0.0数据库和华为云GaussDB(openGauss)数据库，实验数据采用电商数据库的八张表。

**第一章 完整性约束实验概述**

##### 1.1 实验目的

了解SQL语言和openGauss数据库提供的完整性（integrity）机制，通过实验掌握面向实际数据库建立实体完整性、参照完整性、断言、函数依赖等各种完整性约束的方法，验证各类完整性保障措施。

##### 1.2 实验内容

在前面完成的实验中已建立了本实验所需的8张表。本实验将针对这8张表，采用create table、alter table等语句，添加主键、候选键、外键、check约束、默认/缺省值约束，并观察当用户对数据库进行增、删、改操作时，DBMS如何维护完整性约束。

1． 建立完整性约束

2． 主键/候选键/空值/check/默认值约束验证

3． 外键/参照完整性验证分析

4． 函数依赖

5． 触发器

##### 1.3 实验要求

1. 罗列的实验内容比较多，不必都做。类似实验内容选做有代表性的，例如，

1） 主键验证、候选键验证只做一个；

2） 在一个实验中同时验证空值、默认值、主键、check等约束；

3） 级联、非级联外键约束实验二选一

2. 参照下面所给示例，选择电商数据库中不同的关系表，完成各个实验内容。

##### 1.4 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

# **第二章 实验过程**

##### 2.1 利用Create table/Alter table语句建立完整性约束

# **2.1.1 实验要求**

选择电商数据库中的一张表，如订单明细表LINEITEM，分析识别该表上所有约束；采用create table语句，建立该表的副本LINEITEMcopy，将数据导入LINEITEMcopy，后续实验可在上进行。

注意：后面的实验会使数据发生改动，可以在完成每项实验后，将数据发生改动的表清空，然后重新导入数据，从而使数据恢复原样，避免发生不必要的麻烦

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名称 | 字段中文名称 | 数据类型 | 数据取值范围，完整性/约束说明 |
| L\_ORDERKEY | 订单key | INTEGER | NOT NULL 主键 外键 |
| L\_PARTKEY | 零件key | INTEGER | NOT NULL 外键 |
| L\_SUPPKEY | 供应商key | INTEGER | NOT NULL 外键 |
| L\_LINENUMBER | 流水号 | INTEGER | NOT NULL 主键 |
| L\_QUANTITY | 数量 | DECIMAL  (15,2) | NOT NULL |
| L\_EXTENDEDPRICE | 价格 | DECIMAL  (15,2) | NOT NULL |
| L\_DISCOUNT | 折扣 | DECIMAL  (15,2) | NOT NULL |
| L\_TAX | 税 | DECIMAL  (15,2) | NOT NULL |
| L\_RETURNFLAG | 退货标志 | CHAR(1) | NOT NULL |
| L\_LINESTATUS | 明细状态 | CHAR(1) | NOT NULL |
| L\_SHIPDATE | 发货日期 | DATE | NOT NULL |
| L\_COMMITDATE | 预计到达日期 | DATE | NOT NULL |
| L\_RECEIPTDATE | 实际到达日期 | DATE | NOT NULL |
| L\_SHIPINSTRUCT | 运单处理策略 | CHAR(25) | NOT NULL |
| L\_SHIPMODE | 运送方式 | CHAR(10) | NOT NULL |
| L\_COMMENT | 备注 | VARCHAR  (44) | NOT NULL |

# **2.1.2 实验步骤**

使用create table在该表相关属性上，**添加主键、非空**。

CREATE TABLE LINEITEMcopy1(  
L\_ORDERKEY INTEGER NOT NULL,  
L\_PARTKEY INTEGER NOT NULL,  
L\_SUPPKEY INTEGER NOT NULL,  
L\_LINENUMBER INTEGER NOT NULL,  
L\_QUANTITY DECIMAL(15,2) NOT NULL,  
L\_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,  
L\_DISCOUNT DECIMAL(15,2) NOT NULL,  
L\_TAX DECIMAL(15,2) NOT NULL,  
L\_RETURNFLAG CHAR(1) NOT NULL,  
L\_LINESTATUS CHAR(1) NOT NULL,  
L\_SHIPDATE DATE NOT NULL,  
L\_COMMITDATE DATE NOT NULL,  
L\_RECEIPTDATE DATE NOT NULL,  
L\_SHIPINSTRUCT CHAR(25) NOT NULL,  
L\_SHIPMODE CHAR(10) NOT NULL,  
L\_COMMENT VARCHAR(44) NOT NULL,  
PRIMARY KEY (L\_ORDERKEY, L\_LINENUMBER),  
FOREIGN KEY (L\_PARTKEY) REFERENCES PART(P\_PARTKEY),  
FOREIGN KEY (L\_SUPPKEY) REFERENCES SUPPLIER(S\_SUPPKEY)  
);

创建成功

**创建一个不带约束的关系表**，使用alter table语句，在该表上添加约束。

CREATE TABLE LINEITEMcopy2(  
L\_ORDERKEY INTEGER NOT NULL,  
L\_PARTKEY INTEGER NOT NULL,  
L\_SUPPKEY INTEGER NOT NULL,  
L\_LINENUMBER INTEGER NOT NULL,  
L\_QUANTITY DECIMAL(15,2) NOT NULL,  
L\_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,  
L\_DISCOUNT DECIMAL(15,2) NOT NULL,  
L\_TAX DECIMAL(15,2) NOT NULL,  
L\_RETURNFLAG CHAR(1) NOT NULL,  
L\_LINESTATUS CHAR(1) NOT NULL,  
L\_SHIPDATE DATE NOT NULL,  
L\_COMMITDATE DATE NOT NULL,  
L\_RECEIPTDATE DATE NOT NULL,  
L\_SHIPINSTRUCT CHAR(25) NOT NULL,  
L\_SHIPMODE CHAR(10) NOT NULL,  
L\_COMMENT VARCHAR(44) NOT NULL  
);

**新建主键**，如果表中没有建立主键，利用下面的语句添加主键

alter table 表名

add constraint 约束名

primary key (字段名)

说明：字段名为要在其上创建主键的字段名

ALTER TABLE LINEITEMcopy2 ADD CONSTRAINT LINEITEMcopy2\_PK PRIMARY KEY (L\_ORDERKEY, L\_LINENUMBER);

**候选键**，如果表中没有建立候选键，利用下面的语句添加候选键

alter table 表名

add constraint 约束名

unique (字段名)

ALTER TABLE LINEITEMcopy2 ADD CONSTRAINT LINEITEMcopy2\_UK UNIQUE (L\_ORDERKEY, L\_LINENUMBER, L\_PARTKEY);

**非空约束**，利用下面的语句添加非空约束

alter table 表名

alter column 字段名

set not null

ALTER TABLE LINEITEMcopy2 ALTER COLUMN L\_EXTENDEDPRICE SET NOT NULL;

将LINEITEM表的数据复制到LINEITEMcopy表中，保证LINEITEM表与LINEITEMcopy表内容一致

INSERT INTO lineitemcopy1  
SELECT \*  
FROM lineitem;

##### 2.2 主键/候选键约束验证

# **2.2.1 实验要求**

对于主键约束，选取定义了主键的关系表，如lineitemcopy1，

（1）使用分组聚集运算语句，判断是否满足主键约束

（2）向该表插入在主属性上取值为空的元组，观察DBMS反应；

（3）选取表中某些或某个元组，修改这些元组在主属性上的取值，或向表中插入新元组，使这些元组与表中已有其它元组的主属性取值相同，或者将选定的元组在主属性上的取值修改为null，观察DBMS反应；

对于候选键约束

（1）选取定义了候选键的关系表，如lineitemcopy1，使用分组聚集运算语句，判断是否满足候选键约束

（2）向该表插入在候选键属性上取值为空的元组，观察DBMS的反应；

（3）选取表中某些或某个元组，修改这些元组在候选键属性上的取值，或插入新元组，使这些元组与表中已有其它元组的候选键属性取值相同，或者将选定的元组在候选键属性上的取值修改为null，观察系统反应；

# **2.2.2实验步骤**

这里以**主键约束验证**为例，选取上面实验创建并导入数据的表lineitemcopy1。

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：

SELECT l\_orderkey, COUNT(\*)  
FROM lineitemcopy1  
GROUP BY (l\_orderkey, l\_linenumber)  
HAVING COUNT(\*)>1;

结果返回0行

表中并无重复主键的数据。

图形用户界面, 文本, 应用程序, 网站

描述已自动生成

**判断是否有主键为空的数据**

SELECT \*  
FROM lineitemcopy1  
WHERE l\_orderkey IS NULL  
AND l\_linenumber IS NULL;

结果返回0行

表中并无主键为空的数据。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**插入主键为空数据**，报错：

比如，插入值为(null,0,0,null,0,0,0,0,’a’,’b’,’2020-01-01’::date,

’2020-01-12’::date, ’2020-01-15’::date,'name3', 'name4', 'name5');

INSERT INTO lineitemcopy1  
VALUES(NULL,0,0,NULL,0,0,0,0, 'a', 'b', '2020-01-01'::DATE,   
'2020-01-12'::DATE, '2020-01-15'::DATE,'name3', 'name4', 'name5');

图片包含 图形用户界面

描述已自动生成

**修改原有数据行l\_orderkey, l\_linenumber字段为空**，报错：

比如，将l\_orderkey=1, l\_linenumber=5的数据行的l\_orderkey和l\_linenumber修改为空

UPDATE lineitemcopy1  
SET l\_orderkey=NULL, l\_linenumber=NULL  
WHERE l\_orderkey =1 AND l\_linenumber=5;

图形用户界面, 文本, 应用程序

描述已自动生成

**修改原有数据行l\_orderkey字段和l\_linenumber字段与表中已有其它元组的主属性取值相同**，报错：

比如，更新表中l\_orderkey=1，l\_linenumber=1的数据行，将其l\_linenumber字段值改为2，可以看到由于表中已经存在l\_orderkey=1，l\_linenumber=2的数据行，l\_orderkey，l\_linenumber作为主键，不允许重复值，因此执行失败。

UPDATE lineitemcopy1  
SET l\_linenumber =2  
WHERE l\_orderkey=1 AND l\_linenumber=1;

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

同样地，**插入主键重复的数据**，报错：

比如，插入值为(1,0,0,2,0,0,0,0,’a’,’b’,’2020-01-01’::date,

’2020-01-01’::date, ’2020-01-01’::date,'name3', 'name4', 'name5');

INSERT INTO lineitemcopy1   
VALUES(1,0,0,2,0,0,0,0, 'a', 'b', '2020-01-01'::DATE,   
'2020-01-01'::DATE, '2020-01-01'::DATE,'name3', 'name4', 'name5');

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

候选键约束的验证与上述步骤类似，因实验要求选择主键/候选键中一个进行验证，这里省略候选键的验证步骤

##### 2.3 空值约束验证

# **2.3.1 实验要求**

选取定义了not null属性约束的关系表，如lineitemcopy1及其属性l\_extendedprice，观察（1）向表中插入新元组，或（2）修改表中已有元组时，如果导致该属性上取值为空，DBMS的反应和处理方式。

# **2.3.2 实验步骤**

**插入一数据行，其l\_extendedprice字段为空**，报错：

比如，插入值为(1,0,0,2,0,null,0,0,’a’,’b’,’2020-01-01’::date, ’2020-01-01’::date, ’2020-01-01’::date,'name3', 'name4', 'name5');

INSERT INTO lineitemcopy1   
VALUES (1, 0, 0, 2, 0, NULL, 0, 0, 'name1', 'name2', '2020-01-01'::DATE, '2020-01-01'::DATE, '2020-01-01'::DATE, 'name3', 'name4', 'name5');

文本

中度可信度描述已自动生成

**修改原有数据行l\_extendedprice字段为空**，报错：

比如，将l\_orderkey=2 and l\_linenumber=1的数据行的l\_extendedprice修改为空

UPDATE lineitemcopy1  
SET l\_extendedprice=NULL  
WHERE l\_orderkey=2 AND l\_linenumber=1;

图片包含 文本

描述已自动生成

##### 2.3 check/默认值约束验证

# **2.3.1 准备工作**

在上述创建表的SQL语句中，并没有包含具体的检查约束和默认值约束。首先创建检查约束和默认值约束

检查约束（Check Constraint）：允许定义一些条件，以确保插入或更新的数据满足特定的约束条件

例如，想要在LINEITEMcopy1表中**添加一个检查约束，确保L\_QUANTITY列的值大于0**，可以使用以下代码：

ALTER TABLE LINEITEMcopy1  
ADD CONSTRAINT CHK\_L\_QUANTITY CHECK (L\_QUANTITY > 0);

这将确保在插入或更新数据时，L\_QUANTITY列的值始终大于0。

默认值约束（Default Constraint）：允许为列定义默认值，以便在插入新行时自动为该列赋予默认值。

例如，想要在LINEITEMcopy1表的L\_DISCOUNT列上**添加一个默认值约束，将默认值设置为0.00**，可以使用以下代码：

ALTER TABLE LINEITEMcopy1  
ALTER COLUMN L\_DISCOUNT SET DEFAULT 0.00;

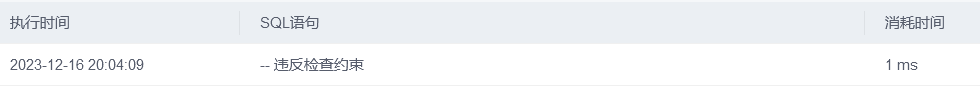
这将使得在插入新行时，如果未显式指定L\_DISCOUNT列的值，将自动使用默认值0.00。

# **2.3.2 实验步骤**

检查约束验证：

尝试**插入一行数据，其中L\_QUANTITY的值小于等于0**。如果检查约束生效，数据库会抛出错误，拒绝插入操作。

INSERT INTO LINEITEMcopy1 (L\_QUANTITY) VALUES (-10);



默认值约束验证：

尝试**插入一行数据，未提供L\_DISCOUNT的值**。如果默认值约束生效，数据库会自动将L\_DISCOUNT列的值设置为默认值0.00。

INSERT INTO LINEITEMcopy1 (L\_DISCOUNT) VALUES (NULL);



##### 2.4 外键/参照完整性约束验证

# **2.4.1 实验要求**

表1 可能存在外键关联的表

|  |  |  |  |
| --- | --- | --- | --- |
| 参照关系r1 | | 被参照关系r2 | |
| 表 | 外键属性 | 表 | 主键 |
| ORDERS | O\_CUSTKEY | CUSTOMER | C\_CUSTKEY |
| SUPPLIER | S\_NATIONKEY | NATION | N\_NATIONKEY |
| NATION | N\_REGIONKEY | REGION | R\_REGIONKEY |
| PARTSUPP | PS\_PARTKEY、PS\_SUPPKEY | PART、SUPPLIER | P\_PARTKEY、S\_SUPPKEY |
| CUSTOMER | C\_NATIONKEY | NATION | N\_NATIONKEY |
| LINEITEM | L\_ORDERKEY、L\_PARTKEY、L\_SUPPKEY | ORDERS、PART | O\_ORDERKEY、P\_PARTKEY、S\_SUPPKEY |

# **2.4.2 实验步骤**

步骤1：判断参照完整性约束是否满足

从表1中选定一组表r1和r2，编写SQL语句，判断两表间是否满足参照完整性约束。例如，r1= ORDERS，r2= CUSTOMER，判断ORDERS在属性O\_CUSTKEY上的取值是否都出现在ORDERS表的C\_CUSTKEY列中。

步骤2：改造参照关系表，满足完整性要求

如果两张表间不满足参照完整性约束。则使用delete语句，去除参照关系表中相关元组，使得两表间参照完整性约束关系成立。

步骤3：在改造后的参照表r1和被参照r2上，建立非级联外键关联。

为方便起见，分别创建关系表orders和customer的副本orderscopy和customercopy，并将数据导入进去

CREATE TABLE customercopy1(  
c\_custkey INTEGER,  
c\_name VARCHAR(25),  
c\_address VARCHAR(40),  
c\_nationkey INTEGER,  
c\_phone CHAR(15),  
c\_acctbal DECIMAL(15,2),  
c\_mktsegment CHAR(10),  
c\_comment VARCHAR(117),  
PRIMARY KEY (c\_custkey),  
FOREIGN KEY (c\_nationkey) REFERENCES nation(n\_nationkey)  
);  
INSERT INTO customercopy1  
SELECT \*  
FROM customer;

CREATE TABLE orderscopy1(  
o\_orderkey INTEGER,  
o\_custkey INTEGER,  
o\_orderstatus CHAR(1),  
o\_totalprice DECIMAL(15,2),  
o\_orderdate DATE,  
o\_orderpriority CHAR(15),  
o\_clerk CHAR(15),  
o\_shippriority INTEGER,  
o\_comment VARCHAR(79),  
PRIMARY KEY (o\_orderkey),  
FOREIGN KEY (o\_custkey) REFERENCES customercopy1(c\_custkey)  
);  
INSERT INTO orderscopy1  
SELECT \*  
FROM orders;

**判断两表间是否满足参照完整性约束**

SELECT COUNT(O\_CUSTKEY)  
FROM orderscopy1  
WHERE O\_CUSTKEY NOT IN (  
SELECT C\_CUSTKEY   
FROM customercopy1  
);

若结果不为0，则不满足参照完整性约束

若结果为0，则两表已经满足参照完整性约束。

图片包含 应用程序

描述已自动生成

由于级联、非级联外键约束实验二选一，此处选择非级联进行实验

定义orderscopy和customercopy之间的非级联关联如下：

ALTER TABLE orderscopy1  
ADD CONSTRAINT FK\_O\_CUSTKEY  
FOREIGN KEY(O\_CUSTKEY) REFERENCES customercopy1(C\_CUSTKEY);

定义成功

##### 2.5 级联/非级联外键关联下数据访问

# **2.5.1 实验要求**

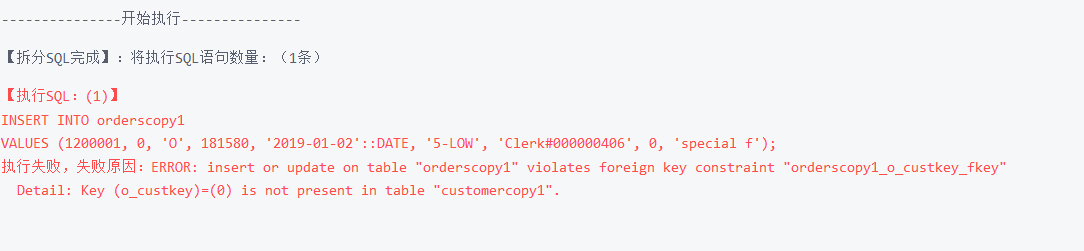
选取相互间定义了非级联外键关联的一组表r1和r2，分别在参照关系r1、被参照关系r2上，对表的主属性/外键属性作插入insert、删除delete、更新update操作，观察当其中1个表（如参照关系表r1、被参照关系表r2）在外键属性或主属性上的取值发生变化时，DBMS对这些操作的反应，以及另外一个表（如被参照关系表、参照关系表）在主属性或外键属性上的取值的变化，并记录实验结果。

上述插入、删除、更新操作操作分为违反约束和不违反约束两种情况。

# **2.5.2 实验步骤**

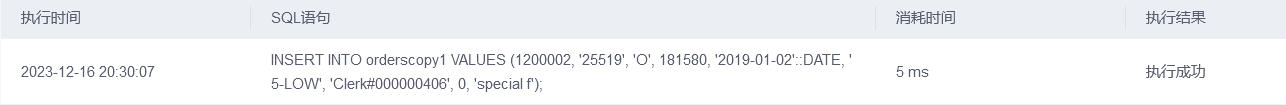
建立好外键关联后，**向orderscopy表中插入一行数据，其O\_CUSTKEY值设为0**，由于customercopy表中不存在C\_CUSTKEY值为0的数据行，违反了外键约束，因此插入失败：

INSERT INTO orderscopy1  
VALUES (1200001, 0, 'O', 181580, '2019-01-02'::DATE, '5-LOW', 'Clerk#000000406', 0, 'special f');



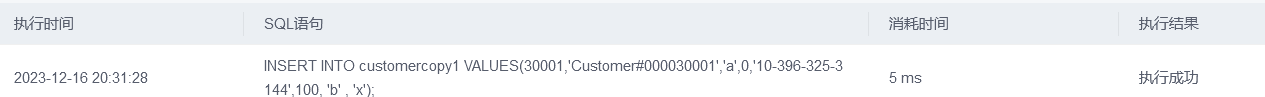
**再向orderscopy表中插入一行数据，其O\_CUSTKEY值设为25519**，由于customercopy表中存在C\_CUSTKEY值为25519的数据行，不违反外键约束，因此插入成功：

INSERT INTO orderscopy1  
VALUES (1200002, '25519', 'O', 181580, '2019-01-02'::DATE, '5-LOW', 'Clerk#000000406', 0, 'special f');



**向customercopy中插入一行C\_CUSTKEY为30001的数据**，而customercopy中不存在C\_CUSTKEY值为30001的数据，插入成功：

INSERT INTO customercopy1  
VALUES(30001,'Customer#000030001','a',0,'10-396-325-3144',100, 'b' , 'x');



**将orderscopy表中一行O\_CUSTKEY值为7828的数据的O\_CUSTKEY字段值修改为31000**，而customercopy表中并没有C\_CUSTKEY值为31000的数据行，因此违反了外键约束，更新失败：

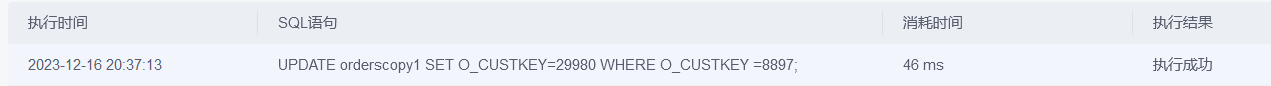
UPDATE orderscopy1  
SET O\_CUSTKEY=31000  
WHERE O\_CUSTKEY=7828;

图形用户界面

低可信度描述已自动生成

**将orderscopy表中一行O\_CUSTKEY值为8897的数据的O\_CUSTKEY字段值修改为29980**，customercopy表中已有C\_CUSTKEY值为29980的数据行，不违反外键约束，因此执行成功：

UPDATE orderscopy1  
SET O\_CUSTKEY=29980  
WHERE O\_CUSTKEY =8897;



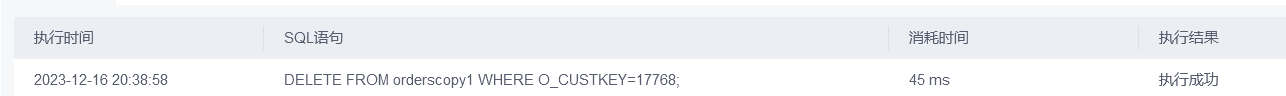
**将customercopy表中一行C\_CUSTKEY值为12的数据的C\_CUSTKEY字段值修改为31001**，表orderscopy中不存在O\_CUSTKEY值为12的数据行,不违反外键约束，执行成功：

UPDATE customercopy1  
SET C\_CUSTKEY=31001  
WHERE C\_CUSTKEY=12;



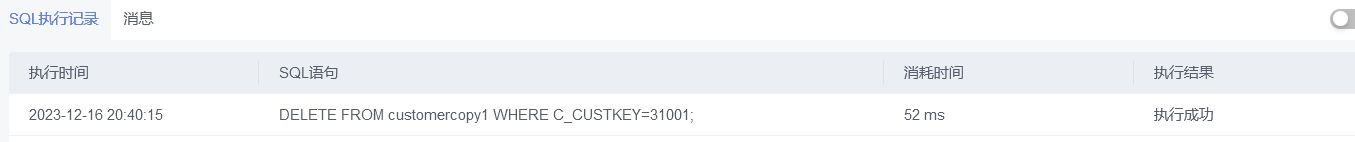
**删除orderscopy表中O\_CUSTKEY字段值为17768的数据行**，执行成功：

DELETE FROM orderscopy1  
WHERE O\_CUSTKEY=17768;



**从customercopy表中删除C\_CUSTKEY值为31001的数据行**，表orderscopy中不存在C\_CUSTKEY值为31001的数据行，不违反外键约束，执行成功：

DELETE FROM customercopy1  
WHERE C\_CUSTKEY=31001;



以上步骤在级联外键关联和非级联外键关联下都一样，没有本质区别。

**级联/非级联区别**

**查看orderscopy表中O\_CUSTKEY=8890的数据项**

SELECT \*  
FROM orderscopy1  
WHERE O\_CUSTKEY=8890;

表格

描述已自动生成

共有24条

**将customercopy表中一行C\_CUSTKEY值为8890的数据的C\_CUSTKEY字段值修改为30005**，表orderscopy中存在O\_CUSTKEY值为8890的数据行,因此违反了外键约束，执行失败：

UPDATE customercopy1  
SET C\_CUSTKEY=30005  
WHERE C\_CUSTKEY=8890;

图形用户界面, 文本, 应用程序

描述已自动生成

**从customercopy表中删除C\_CUSTKEY值为8890的数据行**，表orderscopy中存在O\_CUSTKEY值为8890的数据行，因此违反了外键约束，执行失败：

DELETE FROM customercopy1  
WHERE C\_CUSTKEY =8890;

图形用户界面, 应用程序

描述已自动生成

非级联外键关联下的操作，只要违法了参照完整性约束，便无法执行。而级联外键关联下，当被参照关系中的主键发生修改，删除时，参照关系中的外键会跟着进行相应地修改，删除。

##### 2.6 函数依赖分析验证

# **2.6.1 实验要求**

函数依赖反映了关系表中属性间的依赖关系。主键、候选键、外键约束都属于函数依赖，对于这三类函数依赖的验证参见上面的实验。下面考虑验证非主属性间的函数依赖关系。

在零部件表PART中，一种零件品牌只能由一个零件厂商生产，因此零件品牌与零件厂商间存在函数依赖：

P\_BRAND → P\_MFGR

要求：

（1）用SQL语句判断P\_BRAND与P\_MFGR间是否存在函数依赖关系。

（2）如果P\_BRAND与P\_MFGR间函数依赖不存在，用SQL语句找出导致该函数依赖不存在的元组。

# **2.6.2 实验步骤**

步骤1：判断函数依赖P\_BRAND → P\_MFGR是否满足。

首先对P\_BRAND进行分组，对于有相同P\_BRAND的元组，统计去重之后的零件厂商数量，对于每组结果，只要不同的零件厂商数量大于1，则说明相同的P\_BRAND对应不同的P\_MFGR，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。

SELECT MAX(a) AS a\_MFGR   
FROM(  
SELECT COUNT(DISTINCT P\_MFGR) AS a   
FROM part  
GROUP BY P\_BRAND  
);



结果满足依赖。

步骤2：对于其他不满足依赖关系的例子，可以尝试找出导致该函数依赖不存在的元组。

例如

SELECT T1.P\_BRAND, T2.P\_BRAND, T1.P\_MFGR, T2.P\_MFGR  
FROM part AS T1, part AS T2   
WHERE T1.P\_BRAND = T2.P\_BRAND AND T1.P\_MFGR <> T2.P\_MFGR;

图片包含 图形用户界面

描述已自动生成

##### 2.7 触发器约束验证

# **2.7.1 实验要求**

实验1：开发一个数据插入查重触发器，实现：

向一张表中插入一行新数据时，如果新数据的主键与表中已有其它元组的主键不相同，则直接插入；如果新数据的主键与表中已有元组的主键相同，则根据新插入元组的属性值修改已有元组的属性值，或者：先删除主键相同的已有元组，再插入新元组。

实验2：开发一个日期校对触发器，实现：

当向订单明细表LINEITEM中插入一行，或者修改现有订单的发货日期时，判断新插入的、或修改后的发货日期是否合法，即发货日期必须在预计到达日期和实际到达日期之前。如果不合法，回滚。

实验3：开发一个触发器，实现：

当客户账户余额小于50时，不允许向订单表中插入来自该客户的新订单。

# **2.7.2 实验步骤**

由于触发器中禁止增删改操作的嵌套使用，因此为了完成实验需求，再对lineitemcopy表进行一个备份，新表为lineitemcopy\_new.为了验证触发器正确性，删除新表上的相关约束。在实际应用中需保持两表的数据一致性，本实验仅验证触发器效果。

CREATE TABLE lineitemcopy1\_new(  
L\_ORDERKEY INTEGER NOT NULL,  
L\_PARTKEY INTEGER NOT NULL,  
L\_SUPPKEY INTEGER NOT NULL,  
L\_LINENUMBER INTEGER NOT NULL,  
L\_QUANTITY DECIMAL(15,2) NOT NULL,  
L\_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,  
L\_DISCOUNT DECIMAL(15,2) NOT NULL,  
L\_TAX DECIMAL(15,2) NOT NULL,  
L\_RETURNFLAG CHAR(1) NOT NULL,  
L\_LINESTATUS CHAR(1) NOT NULL,  
L\_SHIPDATE DATE NOT NULL,  
L\_COMMITDATE DATE NOT NULL,  
L\_RECEIPTDATE DATE NOT NULL,  
L\_SHIPINSTRUCT CHAR(25) NOT NULL,  
L\_SHIPMODE CHAR(10) NOT NULL,  
L\_COMMENT VARCHAR(44) NOT NULL  
);  
INSERT INTO lineitemcopy1\_new  
SELECT \*  
FROM lineitemcopy1;  
 

下面是实验1的触发器定义，使用了一个名为 lineitemcopy1\_new\_insert 的触发器。这个触发器会在插入新数据时检查是否存在具有相同主键的元组。如果存在，它将删除旧元组并插入新元组。

CREATE OR REPLACE FUNCTION lineitemcopy1\_new\_insert() RETURNS TRIGGER AS $$  
BEGIN  
    -- 如果找到相同主键的元组  
    IF EXISTS (SELECT 1 FROM lineitemcopy1\_new WHERE L\_ORDERKEY = NEW.L\_ORDERKEY AND L\_LINENUMBER = NEW.L\_LINENUMBER) THEN  
        -- 删除旧元组  
        DELETE FROM lineitemcopy1\_new WHERE L\_ORDERKEY = NEW.L\_ORDERKEY AND L\_LINENUMBER = NEW.L\_LINENUMBER;  
        -- 插入新元组  
        INSERT INTO lineitemcopy1\_new VALUES (NEW.\*);  
        -- 跳过默认的插入操作  
        RETURN NULL;  
    END IF;  
    -- 如果没有找到相同主键的元组，执行默认的插入操作  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
   
CREATE TRIGGER lineitemcopy1\_new\_insert BEFORE INSERT ON lineitemcopy1\_new FOR EACH ROW EXECUTE PROCEDURE lineitemcopy1\_new\_insert();

为了验证触发器的功能，可以使用以下代码执行插入操作并观察结果：

INSERT INTO lineitemcopy1\_new (L\_ORDERKEY, L\_PARTKEY, L\_SUPPKEY, L\_LINENUMBER, L\_QUANTITY, L\_EXTENDEDPRICE, L\_DISCOUNT, L\_TAX, L\_RETURNFLAG, L\_LINESTATUS, L\_SHIPDATE, L\_COMMITDATE, L\_RECEIPTDATE, L\_SHIPINSTRUCT, L\_SHIPMODE, L\_COMMENT)  
VALUES (1, 1, 1, 1, 10, 100, 0.1, 0.05, 'R', 'F', '2023-12-16', '2023-12-16', '2023-12-16', 'Sample Ship Instruct', 'Sample', 'Sample Comment');

这个插入语句尝试将具有相同主键值的数据插入表中。如果触发器正常工作，它将删除原来具有 (L\_ORDERKEY, L\_LINENUMBER) = (1, 1) 的记录，并插入新的记录。

-- 插入一个新元组  
INSERT INTO lineitemcopy1\_new VALUES (1, 1, 1, 1, 1.0, 1.0, 1.0, 1.0, 'A', 'A', CURRENT\_DATE, CURRENT\_DATE, CURRENT\_DATE, 'A', 'A', 'A');  
   
-- 插入一个具有相同主键但不同属性值的元组  
INSERT INTO lineitemcopy1\_new VALUES (1, 2, 2, 1, 2.0, 2.0, 2.0, 2.0, 'B', 'B', CURRENT\_DATE, CURRENT\_DATE, CURRENT\_DATE, 'B', 'B', 'B');  
   
-- 检查结果  
SELECT \* FROM lineitemcopy1\_new WHERE L\_ORDERKEY = 1 AND L\_LINENUMBER = 1;

这段代码应该返回最后插入的元组，而不是最初插入的元组。这证明了触发器的功能。

图片包含 应用程序

描述已自动生成

下面是实验2的触发器定义，创建触发器函数，将新数据插入到lineitemcopy中

CREATE OR REPLACE FUNCTION tri\_insert\_func() RETURNS TRIGGER AS $$ DECLARE BEGIN   
INSERT INTO lineitemcopy1   
VALUES  
(NEW.L\_ORDERKEY, NEW.L\_PARTKEY,NEW.L\_SUPPKEY,NEW.L\_LINENUMBER,NEW.L\_QUANTITY,NEW.L\_EXTENDEDPRICE,  
NEW.L\_DISCOUNT, NEW.L\_TAX, NEW.L\_RETURNFLAG, NEW.L\_LINESTATUS, NEW.L\_SHIPDATE, NEW.L\_COMMITDATE,   
NEW.L\_RECEIPTDATE, NEW.L\_SHIPINSTRUCT, NEW.L\_SHIPMODE, NEW.L\_COMMENT);  
RETURN NEW; END $$ LANGUAGE PLPGSQL;

在lineitemcopy1\_new上定义插入触发器，如果发货日期满足插入条件，则插入到lineitemcopy中，若不满足条件，则不进行插入操作。

CREATE TRIGGER insert\_trig\_before BEFORE INSERT ON lineitemcopy1\_new FOR EACH ROW  
WHEN(NEW.l\_shipdate <= NEW.l\_commitdate AND NEW.l\_shipdate <= NEW.l\_receiptdate)  
EXECUTE PROCEDURE tri\_insert\_func();

通过pg\_get\_triggerdef(oid)获取触发器定义信息

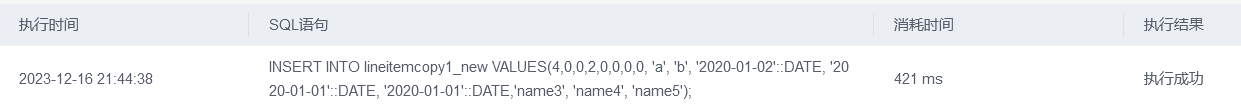
SELECT pg\_get\_triggerdef(oid) FROM pg\_trigger;

图形用户界面, 文本, 应用程序

描述已自动生成

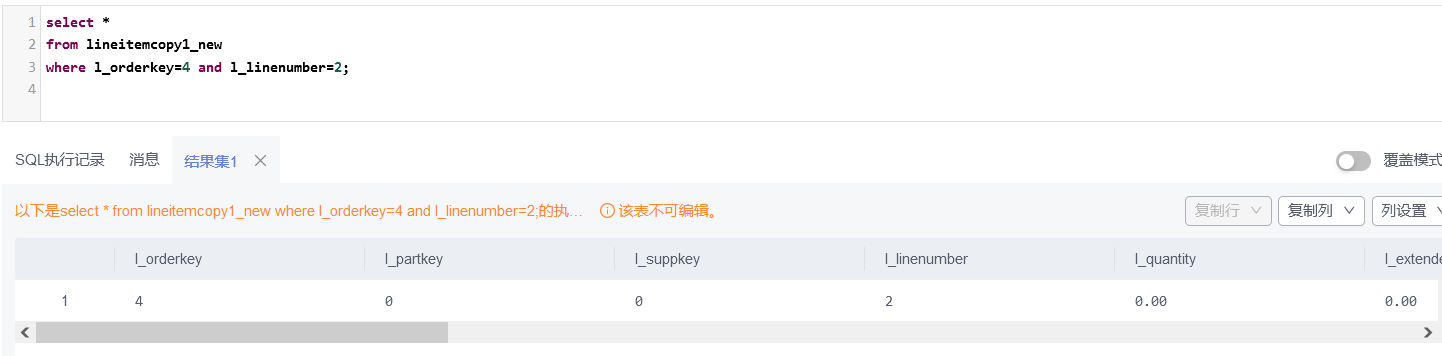
向lineitemcopy\_new插入一行数据，发货日期为2020年1月2日，预计到达日期和实际到达日期均为2020年1月1日，不满足约束条件。

INSERT INTO lineitemcopy1\_new  
VALUES(4,0,0,2,0,0,0,0, 'a', 'b', '2020-01-02'::DATE,   
'2020-01-01'::DATE, '2020-01-01'::DATE,'name3', 'name4', 'name5');



查看订单key为4，流水号为2的数据行，发现数据插入到了lineitemcopy1\_new表中

SELECT \*  
FROM lineitemcopy1\_new  
WHERE l\_orderkey=4 AND l\_linenumber=2;



但没有插入进lineitemcopy1表

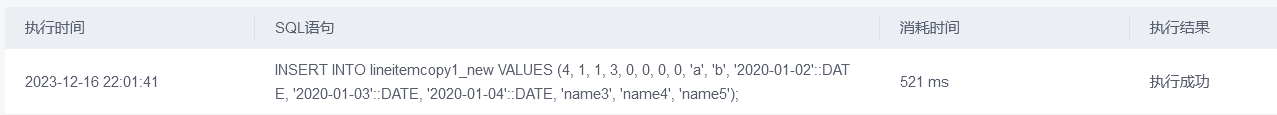
SELECT \*  
FROM lineitemcopy1  
WHERE l\_orderkey=4 AND l\_linenumber=2;

图形用户界面, 应用程序

描述已自动生成

插入一行发货日期为2020年1月2日，预计到达日期为2020年1月3日，实际到达日期均为2020年1月4日的数据，满足约束，插入成功。

INSERT INTO lineitemcopy1\_new  
VALUES (4, 1, 1, 3, 0, 0, 0, 0, 'a', 'b', '2020-01-02'::DATE, '2020-01-03'::DATE, '2020-01-04'::DATE, 'name3', 'name4', 'name5');



查看订单key为4，流水号为3的数据行，在lineitemcopy1\_new中查看刚才插入的数据，发现插入成功。

SELECT \*  
FROM lineitemcopy1\_new  
WHERE l\_orderkey=4 AND l\_linenumber=3;

图形用户界面, 应用程序

中度可信度描述已自动生成

在lineitemcopy1中查看刚才插入的数据，发现插入也成功了。

SELECT \*  
FROM lineitemcopy1  
WHERE l\_orderkey=4 AND l\_linenumber=3;

应用程序

低可信度描述已自动生成

对于修改发货日期的操作，创建触发器函数

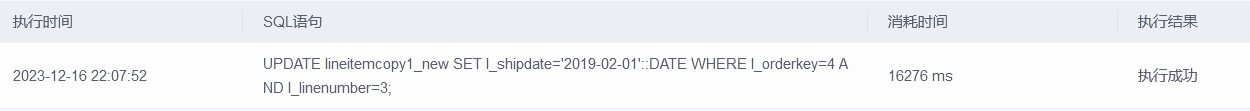
CREATE OR REPLACE FUNCTION tri\_update\_func() RETURNS TRIGGER AS $$ DECLARE BEGIN  
UPDATE  lineitemcopy1   
SET L\_SHIPDATE =NEW. L\_SHIPDATE   
WHERE OLD. L\_ORDERKEY = NEW. L\_ORDERKEY AND OLD. L\_LINENUMBER = NEW. L\_LINENUMBER;  
RETURN NEW; END $$ LANGUAGE PLPGSQL;

同样对lineitemcopy\_new设置一个更新触发器，插入前比较发货日期与预计到达日期、实际到达日期，当发货日期满足约束时，将新值更新到lineitemcopy对应行中，否则不进行更新。

CREATE TRIGGER updata\_trig\_before BEFORE UPDATE ON lineitemcopy1\_new FOR EACH ROW  
WHEN(NEW.l\_shipdate <= NEW.l\_commitdate AND NEW.l\_shipdate <= NEW.l\_receiptdate)  
EXECUTE PROCEDURE tri\_update\_func();

将刚才插入的订单key为4，流水号为3的数据进行修改，发货日期新值为2019-02-01，满足约束条件。

UPDATE lineitemcopy1\_new  
SET l\_shipdate='2019-02-01'::DATE  
WHERE l\_orderkey=4 AND l\_linenumber=3;



查看订单key为4，流水号为3的数据行，在lineitemcopy\_new中查看更新的数据，发现更新成功。

图形用户界面, 应用程序, Teams

描述已自动生成

在lineitemcopy中查看更新的数据，发现更新也成功了。

图形用户界面, 应用程序

描述已自动生成

同样的，将订单key为4，流水号为3的数据再进行修改，PCI新值为2020-02-01，不满足约束条件。

UPDATE lineitemcopy1\_new  
SET l\_shipdate='2020-02-01'::DATE  
WHERE l\_orderkey=4 AND l\_linenumber=3;

查看订单key为4，流水号为3的数据行，在lineitemcopy\_new中查看更新的数据，发现更新成功

图形用户界面, 应用程序

描述已自动生成

在lineitemcopy中查看更新的数据，发现更新失败，表中维持原数据不变。

图形用户界面, 应用程序

描述已自动生成

实验3的触发器定义如下，这个触发器会在插入新数据时检查相关客户的账户余额。

CREATE OR REPLACE FUNCTION check\_balance() RETURNS TRIGGER AS $$  
DECLARE   
    customer\_balance DECIMAL(15,2);  
BEGIN  
    SELECT c\_acctbal INTO customer\_balance FROM customercopy1 WHERE c\_custkey = NEW.o\_custkey;  
    IF customer\_balance < 50 THEN  
        RAISE EXCEPTION '账户余额不足，无法插入新的订单';  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
   
CREATE TRIGGER check\_balance\_before\_insert  
BEFORE INSERT ON orderscopy1  
FOR EACH ROW EXECUTE PROCEDURE check\_balance();

可以通过以下代码来进行验证

首先找一个余额小于50的项目

SELECT \*  
FROM customercopy1  
WHERE c\_acctbal < 50;

图形用户界面, 文本, 应用程序

描述已自动生成

为这个客户添加一个订单记录

INSERT INTO orderscopy1 (o\_orderkey, o\_custkey, o\_orderstatus, o\_totalprice, o\_orderdate, o\_orderpriority, o\_clerk, o\_shippriority, o\_comment)  
VALUES (1, 11, 'O', 100.00, '2023-12-16', 'High', 'John', 1, 'Comment for order');

图片包含 文本

描述已自动生成

插入失败

# **第三章 总结**

在本次实验中，我们探索了数据库中的完整性约束。完整性约束是一种用于确保数据库中数据的准确性和一致性的技术。通过定义一组规则和条件，我们可以限制数据库中数据的插入、更新和删除操作，以确保数据的完整性。

在实验中，我们首先创建了一个简单的数据库，并设计了几个表格来存储数据。接下来，我们使用不同类型的完整性约束来保护这些表格中的数据。

实验中使用的完整性约束包括：

1. 主键约束：我们在表格中定义了主键，确保每个记录都具有唯一标识符。这样可以防止数据的重复和冗余。

2. 外键约束：我们在表格之间建立了关联，并使用外键约束来确保关联的有效性。外键约束可以防止删除或更新主表中的关键数据，而不会破坏相关的从表数据。

3. 唯一约束：我们使用唯一约束来限制某些列的取值不能重复。这样可以确保数据的唯一性。

4. 默认约束：我们为某些列定义了默认值，确保在插入新记录时，这些列具有默认的初始值。

通过实验，我们发现完整性约束对于维护数据库的数据一致性和准确性非常重要。它们可以帮助我们避免插入无效或冗余的数据，保护数据库的完整性。

在实验过程中，我们还注意到完整性约束在一些情况下可能会导致操作失败。例如，当插入一条违反唯一约束的记录时，数据库会抛出错误并拒绝插入。这提示了我们在设计数据库结构和完整性约束时需要谨慎考虑，并确保约束的正确性和适用性。

总的来说，本次实验帮助我们深入理解了数据库中的完整性约束，并展示了它们在维护数据一致性方面的重要性。通过合理地设计和使用完整性约束，我们可以提高数据库的数据质量和可靠性，为应用程序和用户提供更好的体验。