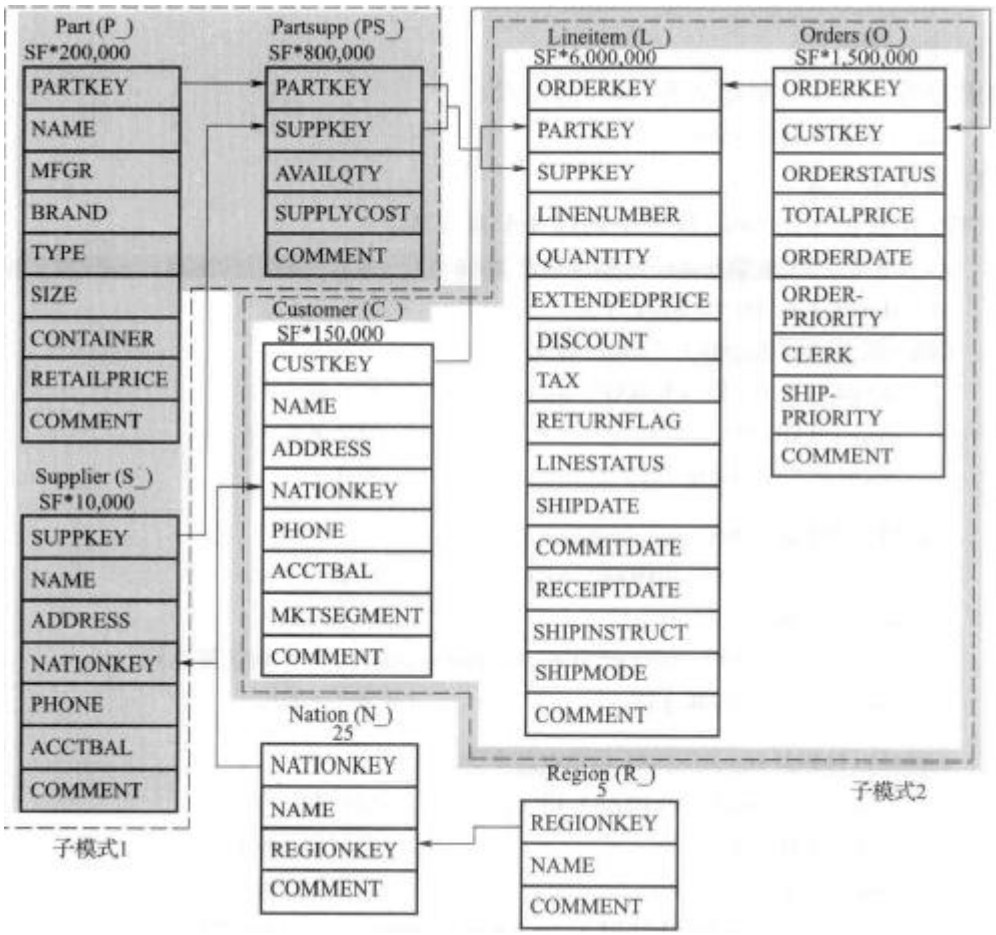
实验 1-数据库定义与操作语言实验

一、 实验环境

平台	Windows 10
SQL	MySQL 8.0

二、 数据库定义实验

本实验建立 TPC-H 数据库模式。TPC-H 数据库模式由零件表（Part）、供应商表（Supplier）、零件供应商表（Partsupp）、顾客表（Customer）、国家表（Nation）、地区表（Region）、订单表（Orders）和订单明细表（Lineitem）8 个基本表组成。



1) 定义数据库

采用中文字符集创建名为 TPCH 的数据库。“IF NOT EXISTS”语句判断 TPCH 数据库是否存在，如果没有则创建；设置数据库字符集的语句是“DEFAULT CHARACTER SET GBK COLLATE GBK_CHINESE_CI”，教材中“ENCODING = 'GBK'”的语法有误。

```
CREATE DATABASE IF NOT EXISTS TPCH DEFAULT CHARACTER SET GBK COLLATE
GBK_CHINESE_CI;
```

2) 定义模式

在数据库 TPCH 中创建名为 Sales 的模式。

```
CREATE SCHEMA IF NOT EXISTS Sales DEFAULT CHARACTER SET GBK COLLATE
GBK_CHINESE_CI;
```

在 MySQL 中，模式和数据库是等价的，因此执行这条语句后“SHOW DATABASES;”会发现有一个名为 Sales 的数据库。

3) 定义基本表

在 Sales 模式中创建 8 个基本表。

设置当前作用域的数据库为 Sales，基本表就会自动创建在 Sales 模式下。

```
USE Sales;
```

a) 地区表

```
CREATE TABLE IF NOT EXISTS Region (      --地区表
    regionkey INTEGER PRIMARY KEY,       --地区编号（主键）
    name CHAR(25),                       --地区名称
    comment VARCHAR(152)                 --备注
);
```

b) 国家表

```
CREATE TABLE IF NOT EXISTS Nation (      --国家表
    nationkey INTEGER PRIMARY KEY,       --国家编号（主键）
    name CHAR(25),                       --国家名称
    regionkey INTEGER REFERENCES Region (regionkey),    --地区编号（外键）
    comment VARCHAR(152)                 --备注
);
```

c) 供应商表

```
CREATE TABLE IF NOT EXISTS Supplier (    --供应商基本表
    suppkey INTEGER PRIMARY KEY,         --供应商编号（主键）
    name CHAR(25),                       --供应商名称
    address VARCHAR(40),                 --供应商地址
    nationkey INTEGER REFERENCES Nation (nationkey),    --国家编号（外键）
    phone CHAR(15),                      --供应商电话
    accbal REAL,                         --供应商账户余额
    comment VARCHAR(101)                 --备注
);
```

d) 零件表

```
CREATE TABLE IF NOT EXISTS Part (        --零件基本表
    partkey INTEGER PRIMARY KEY,         --零件编号（主键）
    name VARCHAR(55),                    --零件名称
    mfgr CHAR(25),                       --制造厂
    brand CHAR(10),                      --品牌
```

```
    type VARCHAR(25),                       --零件类型
    size INTEGER,                           --尺寸
    container CHAR(10),                     --包装
    retailprice REAL,                       --零售价格
    comment VARCHAR(23)                     --备注
);
```

e) 零件供应商表

```
CREATE TABLE IF NOT EXISTS PartSupp (    --零件供应联系表
    partkey INTEGER REFERENCES Part (partkey),  --零件编号（外键）
    suppkey INTEGER REFERENCES Supplier (suppkey),--零件供应商编号（外键）
    availqty INTEGER,                       --可用数量
    supplycost REAL,                        --供应价格
    comment VARCHAR(199),                   --备注
    PRIMARY KEY (partkey , suppkey)      --主键，表级约束
);
```

f) 顾客表

```
CREATE TABLE IF NOT EXISTS Customer (    --顾客表
    custkey INTEGER PRIMARY KEY,         --顾客编号（主键）
    name VARCHAR(25),                   --顾客名字
    address VARCHAR(40),                --地址
    nationkey INTEGER REFERENCES Nation (nationkey),    --国籍编号
    phone CHAR(15),                     --电话
    acctbal REAL,                       --账户余额
    mktsegment CHAR(10),                --市场分区
    comment VARCHAR(117)                --备注
);
```

g) 订单表

```
CREATE TABLE IF NOT EXISTS Orders (     --订单表
    orderkey INTEGER PRIMARY KEY,       --订单编号（主键）
    custkey INTEGER REFERENCES Customer (custkey),  --顾客编号（外键）
    orderstatus CHAR(1),                --订单状态
    totalprice REAL,                    --总金额
    orderdate DATE,                     --日期
    orderpriority CHAR(15),             --优先级
    clerk CHAR(15),                     --记账员
    shippriority INTEGER,               --运输优先级
    comment VARCHAR(79)                 --备注
);
```

h) 订单明细表

```
CREATE TABLE IF NOT EXISTS Lineitem (   --订单明细表
    orderkey INTEGER REFERENCES Orders (orderkey),  --订单编号
    partkey INTEGER REFERENCES Part (partkey),      --零件编号
    suppkey INTEGER REFERENCES Supplier (suppkey),  --供应商编号
    linenumber INTEGER,                             --订单明细编号
```

```
    quantity REAL,                                  --数量
    extendedprice REAL,                             --订单明细价格
    discount REAL,                                  --折扣[0.00, 1.00]
    tax REAL,                                       --税率[0.00,  0.08]
    returnflag CHAR(1),                             --退货标记
    linestatus CHAR(1),                             --订单明细状态
    shipdate DATE,                                  --装运日期
    commitdate DATE,                                --委托日期
    receipdate DATE,                                --签收日期
    shipinstruct CHAR(25),                          --装运说明
    shipmode CHAR(10),                              --装运方式
    comment VARCHAR(44),                            --备注
    PRIMARY KEY (orderkey , linenumber),
    FOREIGN KEY (partkey , suppkey)
        REFERENCES PartSupp (partkey , suppkey)
);
```

三、 数据基本查询实验

1) 导入数据

加载数据时遇到“–secure-file-priv”错误，使用“SHOW GLOBAL VARIABLES LIKE ‘%secure%’;”发现 secure-file-priv 为 NULL，即不允许导入导出数据，需要更改 MySQL 服务默认配置文件 my.ini 中的 secure-file-priv 选项为“”（空字符串），以表示允许任意目录的导入导出，或者设置为某个目录的路径。

```
USE Sales;
LOAD DATA INFILE 'region.tbl' REPLACE INTO TABLE Sales.Region FIELDS
TERMINATED BY '|' LINES TERMINATED BY '\n';
LOAD DATA INFILE 'nation.tbl' REPLACE INTO TABLE Sales.Nation FIELDS
TERMINATED BY '|' LINES TERMINATED BY '\n';
LOAD DATA INFILE 'supplier.tbl' REPLACE INTO TABLE Sales.Supplier
FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n';
LOAD DATA INFILE 'part.tbl' REPLACE INTO TABLE Sales.Part FIELDS
TERMINATED BY '|' LINES TERMINATED BY '\n';
LOAD DATA INFILE 'partsupp.tbl' REPLACE INTO TABLE Sales.PartSupp
FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n';
LOAD DATA INFILE 'customer.tbl' REPLACE INTO TABLE Sales.Customer
FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n';
LOAD DATA INFILE 'orders.tbl' REPLACE INTO TABLE Sales.Orders FIELDS
TERMINATED BY '|' LINES TERMINATED BY '\n';
LOAD DATA INFILE 'lineitem.tbl' REPLACE INTO TABLE Sales.Lineitem
FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n';
```

2) 单表查询（投影）

查询供应商的名称、地址和联系电话。

```
SELECT name, address, phone FROM Supplier;
```

此命令查询了 Supplier 表中的所有元组，并打印其 name、address、phone 属性。部分结果如下：

name	address	phone
Supplier#000000001	N kD4on9OM Ipw3,gf0JBoODd7tgrzrddZ	27-918-335-1736
Supplier#000000002	89eJ5ksX3ImxJOBvxObC.	15-679-861-2259
Supplier#000000003	q1,G3Pi6OiIuUYfUoH18BFTKP5aU9bEV3	11-383-516-1199
Supplier#000000004	Bk7ah4CK8SYOTepEmvMkkgMwg	25-843-787-7479
Supplier#000000005	Gcdm2rJRzl5qlTVzc	21-151-690-3663
Supplier#000000006	tOxuVm7s7CnK	24-696-997-4969
Supplier#000000007	s,4TicNGB4uO6PaSqNBUq	33-990-965-2201
Supplier#000000008	9Sq4bBH2FOEmaFOocY45sRTxo6yuoG	27-498-742-3860
Supplier#000000009	1KhUgZegwM3ua7dsYmekYBsK	20-403-398-8662
Supplier#000000010	Saygah3gYWMp72i PY	34-852-489-8585
Supplier#000000011	JfwTs,LZrV, M,9C	28-613-996-1505
Supplier#000000012	aLIW q0HYd	18-179-925-7181
Supplier#000000013	HK71HQyWoqRWOX8GI FpqAifW,2PoH	13-727-620-7813
Supplier#000000014	EXsnO5pTNi4iZRm	25-656-247-5058
Supplier#000000015	olXVbNBfVzRqgokr1T,Ie	18-453-357-6394
Supplier#000000016	YiP5C55zHDXL7LalK27zfQnwejdpin4AMpvh	32-822-502-4215
Supplier#000000017	c2d,ESHRSkK3WYnxpgw6aOqN0q	29-601-884-9219
Supplier#000000018	PGGVE5PWAMwKDZw	26-729-551-1115
Supplier#000000019	edZT3es,nBFD8lBXTGeTl	34-278-310-2731
Supplier#000000020	ivbAE,RmTymrZVYaFZva2SH,i	13-715-945-6730

3) 单表查询（选择）

查询最近一周内提交的总价大于 1000 元的订单的编号、顾客编号等订单的所有信息。由于表中日期是很久之前的，因此假设 CURRENT_DATE 为 DATE("1997-12-25")。

```
SELECT * FROM Sales.Orders WHERE DATE("1997-12-25") - orderdate >= 0
AND DATE("1997-12-25") - orderdate < 7 AND totalprice > 1000;
```

结果如下，可以看到每条记录的 orderdate 都是在 7 天之内的。

orderkey	custkey	orderstatus	totalprice	orderdate
1024	35	O	248306.79	1997-12-23
1094	1438	O	9281.56	1997-12-24
5700	1427	O	97397.92	1997-12-25
10402	7	O	69976.43	1997-12-20
13191	214	O	331578.75	1997-12-24
13445	682	O	21693.67	1997-12-24
14757	1420	O	55954.21	1997-12-24
16003	130	O	312518.45	1997-12-21
19681	973	O	166361.38	1997-12-25
20293	668	O	180794.72	1997-12-25
20865	518	O	360459.63	1997-12-19
25153	1274	O	31526.7	1997-12-22
25766	601	O	6842.11	1997-12-23
27522	509	O	220570.26	1997-12-22

4) 不带分组过滤条件的分组统计查询

```
SELECT C.custkey, SUM(O.totalprice) FROM customer C, orders O
WHERE C.custkey = O.custkey GROUP BY C.custkey;
```

结果如下，每条记录都列出了客户的 custkey 及其在订单表中的购物总价。

	custkey	SUM(O.totalprice)
	370	2860895.79
	781	1734130.4100000004
	1234	2785103.37
	1369	3011056.6
	445	2452248.7200000007
	557	1355098.93
	392	1512336.9500000002
	1301	1172081.0799999998
	670	3704194.54
	611	1054581.81
	1276	4327406.880000001
	1153	2400353.06
	862	2283650.5500000003

5) 带分组过滤条件的分组统计查询

由于 name 属性不在分组条件中，因此需要用聚集函数（如 max）。

```
SELECT C.custkey, MAX(C.name) FROM customer C, orders O
WHERE C.custkey = O.custkey
GROUP BY C.custkey HAVING AVG(O.totalprice) > 1000;
```

下表列出了平均购物总价大于 1000 的顾客的 custkey 和名字。

custkey	MAX(C.name)
370	Customer#000000370
781	Customer#000000781
1234	Customer#000001234
1369	Customer#000001369
445	Customer#000000445
557	Customer#000000557
392	Customer#000000392
1301	Customer#000001301
670	Customer#000000670
611	Customer#000000611
1276	Customer#000001276
1153	Customer#000001153
862	Customer#000000862
1249	Customer#000001249

6) 单表自身连接查询

```
SELECT F.suppkey, F.name, F.address
FROM supplier F, supplier S WHERE F.nationkey = S.nationkey
AND S.name = 'Supplier#000000009';
```

连接计算 F 与 S 的笛卡尔积。之所以有两条记录，是因为第一条是 9 号和 9 号连接，第二条是 34 号和 9 号连接。34 和 9 号的 nationkey 相同，即他们属于同一个国家。

suppkey	name	address
9	Supplier#000000009	1KhUgZegwM3ua7dsYmekYBsK
34	Supplier#000000034	mYRe3KvA2O4lL4HhxDKkkrPUDPMKRCSp.Xpa

7) 两表连接查询（普通连接）

```
SELECT P.name, P.mfgr, P.retailprice, PS.supplycost
FROM part P, partsupp PS
WHERE P.retailprice > PS.supplycost;
```

下图列出了在笛卡尔积产生的元组中 retailprice 大于 supplycost 的元组。

name	mfgr	retailprice	supplycost
goldenrod lavender spring chocolate lace	Manufacturer#1	901	771.64
blush thistle blue yellow saddle	Manufacturer#1	902	771.64
spring green yellow purple cornsilk	Manufacturer#4	903	771.64
cornflower chocolate smoke green pink	Manufacturer#3	904	771.64
forest brown coral puff cream	Manufacturer#3	905	771.64
bisque cornflower lawn forest magenta	Manufacturer#2	906	771.64
moccasin green thistle khaki floral	Manufacturer#1	907	771.64
misty lace thistle snow royal	Manufacturer#4	908	771.64
thistle dim navajo dark gainsboro	Manufacturer#4	909	771.64
linen pink saddle puff powder	Manufacturer#5	910.01	771.64
spring maroon seashell almond orchid	Manufacturer#2	911.01	771.64

8) 两表连接查询（自然连接）

自然连接需要保证同名属性值相同。在 part 和 partsupp 表中同名属性使 partkey，因此 WHERE 条件子句需要判断 P.partkey = PS.partkey。

```
SELECT P.name, P.mfgr, P.retailprice, PS.supplycost
FROM part P, partsupp PS
WHERE P.partkey = PS.partkey AND P.retailprice > PS.supplycost;
```

下图与普通连接结果的不同之处在于去掉了那些在 partkey 属性上取值不同的元组。

name	mfgr	retailprice	supplycost
goldenrod lavender spring chocolate lace	Manufacturer#1	901	771.64
goldenrod lavender spring chocolate lace	Manufacturer#1	901	337.09
goldenrod lavender spring chocolate lace	Manufacturer#1	901	357.84
blush thistle blue yellow saddle	Manufacturer#1	902	378.49
blush thistle blue yellow saddle	Manufacturer#1	902	438.37
blush thistle blue yellow saddle	Manufacturer#1	902	306.39
spring green yellow purple cornsilk	Manufacturer#4	903	498.13
spring green yellow purple cornsilk	Manufacturer#4	903	645.4
spring green yellow purple cornsilk	Manufacturer#4	903	191.92
cornflower chocolate smoke green pink	Manufacturer#3	904	113.97
cornflower chocolate smoke green pink	Manufacturer#3	904	591.18

9) 三表连接查询

三表连接与两表连接同理，customer 表与 orders 表都具有 custkey 属性，orders 表和 lineitem 表都具有 orderkey 属性，在 WHERE 子句保证取值一样。

```
SELECT O.orderkey, O.totalprice, L.partkey, L.quantity, L.extendedprice
FROM customer C, orders O, lineitem L
WHERE C.custkey = O.custkey AND O.orderkey = L.orderkey AND C.name =
'Customer#000000370';
```

下图列出了名字为 000000370 的客户的订单编号、总价及其订购的零件信息。

orderkey	totalprice	partkey	quantity	extendedprice
1	172799.49	1552	17	24710.35
1	172799.49	674	36	56688.12
1	172799.49	637	8	12301.04
1	172799.49	22	28	25816.56
1	172799.49	241	24	27389.76
1	172799.49	157	32	33828.8
130	140213.54	1289	14	16663.92
130	140213.54	18	48	44064.48
130	140213.54	119	18	18343.98
130	140213.54	1157	13	13755.95
130	140213.54	692	31	49373.39

四、 数据更新实验

1) INSERT 基本语句（插入全部列的数据）

```
INSERT INTO customer VALUES(2030, '张三', '北京市', 40, '010-51001199',
0.00, 'Northeast', 'vip customer');
```

按教材上的代码输入会发生错误：Error Code: 1062. Duplicate entry '30' for key 'PRIMARY'。这是因为 custkey 是主键，而 custkey 为 30 的客户已经在 customer 表中。

```
SELECT * FROM customer WHERE name = '张三';
```

custkey	name	address	nationkey	phone	acctbal	mktsegment	comment
2030	张三	北京市	40	010-51001199	0	Northeast	vip customer
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2) INSERT 基本语句（插入部分列的数据）

教材中(partkey, suppkey)为(479, 1)，外键引用错误，因为 PartSupp 中没有这个主键，修改为(1, 27)。

```
INSERT INTO lineitem(orderkey, Linenumber, partkey, suppkey, quantity,
shipdate)
VALUES (862, ROUND(RAND() * 100.0), 1, 27, 10, '2012-3-6');
```

3) 批量数据 INSERT 语句

a) 创建一个新的顾客表，把所有中国国籍顾客插入到新的顾客表中。

“WITH NO DATA”的方法语法错误，改为 MySQL 使用 LIKE 方法。

```
CREATE TABLE NewCustomer LIKE customer;
```

查看 NewCustomer 表的结构，发现其不仅有何 customer 一样的属性，连主键信息也是一样的。

Columns in table

Column	Type	Nullable	Indexes
custkey	int(11)	NO	PRIMARY
name	varchar(25)	YES	
address	varchar(40)	YES	
nationkey	int(11)	YES	
phone	char(15)	YES	
acctbal	double	YES	
mktsegment	char(10)	YES	

```
INSERT INTO newcustomer
SELECT C.*
FROM customer C, nation N
WHERE C.nationkey = N.nationkey AND N.name = 'CHINA';
```

从下图的结果中可以发现所有记录的 nationkey 都是 18，即 CHINA 的国家编号。

custkey	name	address	nationkey
7	Customer#000000007	TcGe5gaZNgVePxU5kRrvXBfkasDTea	18
19	Customer#000000019	uc,3bHIx84H,wdrmLOiVsiqXCq2tr	18
75	Customer#000000075	Dh 6iZ,cwxWLKOfRKkiGrzv6pm	18
82	Customer#000000082	zhG3EZbap4c992Gi3bK,3Ne,Xn	18
118	Customer#000000118	OVnFuHygK9wx3xpg8	18
124	Customer#000000124	aTbvVAW5tCd,v09O	18
147	Customer#000000147	6VvIwbVdmcsMzuu,C84GtBWPaipGfi7DV	18
150	Customer#000000150	zeoGShTiCwGPplOWFkLURrh41O0AZ8dwNEEN4	18
169	Customer#000000169	NihmHa7xrciE	18
196	Customer#000000196	68RstNo6a2B	18
234	Customer#000000234	ILvuJbixVmrNEVxsfOOMFxBvvSs	18

b) 创建一个顾客购物统计表，记录每个顾客极其购物总数和总价等信息。

```
CREATE TABLE ShoppingStat(
    custkey INTEGER,
    quantity REAL,
    totalprice REAL
);
INSERT INTO shoppingstat
SELECT C.custkey, SUM(L.quantity), SUM(O.totalprice)
FROM customer C, orders O, lineitem L
WHERE C.custkey = O.custkey AND O.orderkey = L.orderkey
GROUP BY C.custkey;
```

custkey	quantity	totalprice
370	2117	13509419.099999996
781	1324	7628113.619999997
1234	2014	14707309.719999993
1369	2228	15370767.310000006
445	1823	11688852.799999997
557	1015	6396499.809999999
392	1068	6364371.840000003
1301	843	5717341.740000001
670	2659	17850711.669999987
611	821	4514647.38
1276	3029	23694012.25000002

c) 倍增零件表的数据，多次重复执行，直到总记录数达到 50 万行为止。

```
INSERT INTO part
SELECT partkey + (SELECT COUNT(*) FROM part),
       name, mfgr, brand, type, size, container, retailprice, comment
FROM part;
```

多次执行上面的语句，用 `SELECT COUNT(*) FROM part;`语句查看表的大小，最终 part 表的大小为 512000。

	count(*)
	512000

4) UPDATE 语句（修改部分记录的部分列值）

```
UPDATE partsupp
SET supplycost = supplycost * 0.9
WHERE suppkey = (SELECT suppkey FROM supplier WHERE name =
'Supplier#000000009');
```

下图中左图是 partsupp 表开始时的数据，右图是执行上面的语句之后的结果。可以看到，每一行的 supplycost 都变成了左边 supplycost 的 0.9 倍。

suppkey	supplycost
9	249.63
9	327.19
9	512.24
9	399.64
9	558.85
9	883.99
9	252.66
9	519.36
9	135.51
9	736.86
9	729.13

suppkey	supplycost
9	224.667
9	294.471
9	461.016
9	359.676
9	502.96500000000003
9	795.591
9	227.394
9	467.42400000000004
9	121.95899999999999
9	663.174
9	656.217

5) UPDATE 语句（利用一个表中的数据修改另外一个表中的数据）

MySQL 中把表 P 放在 FROM 子句的语法是错误的，正确方式是放在 UPDATE 之后。

```
UPDATE lineitem L, part P
SET L.extendedprice = P.retailprice * L.quantity
WHERE L.partkey = P.partkey;
```

查询 lineitem 中的 extendedprice 属性，发现从原来的 NULL 值变成了具体数值，是 lineitem 和 part 中 partkey 属性值相同即同一个零件的 retailprice 和 quantity 的乘积。

orderkey	linenumber	extendedprice
1	1	24710.35
1	2	56688.12
1	3	12301.04
1	4	25816.559999999998
1	5	27389.760000000002
1	6	33828.8
2	1	36596.28
3	1	42436.799999999996
3	2	53468.310000000005
3	3	32029.559999999998
3	4	2388.58
3	5	48519.24

6) DELETE 基本语句（删除给定条件的所有记录）

```
DELETE FROM lineitem
WHERE orderkey IN (SELECT orderkey FROM orders O, customer C
                   WHERE O.custkey = C.custkey AND C.name =
'Customer#000000001');
DELETE FROM orders
WHERE custkey = (SELECT custkey FROM customer WHERE name =
'Customer#000000001');
```

下图中左图是 Customer#000000001 一开始的订单记录，右图是执行上面语句之后的结果，可以看到 orders 表中已经没有他的订单记录。

orderkey	linenumber
9154	1
9154	2
9154	3
9154	4
9154	5
9154	6
9154	7
14656	1
24322	1
24322	2
24322	3
24322	4
24322	5

orderkey	linenumber
NULL	NULL

五、 视图实验

1) 创建视图（省略视图列名）

```
CREATE VIEW V_DLMU_PartSupp1 AS
SELECT P.partkey, P.name, PS.availqty, P.retailprice, PS.supplycost,
P.comment
FROM part P, partsupp PS, supplier S
WHERE P.partkey = PS.partkey AND S.suppkey = PS.suppkey AND S.name =
'Supplier#000000004';
```

省略视图列名时 SELECT 语句中的所有属性作为视图的属性，且属性名与原表面相同，当 SELECT 的是一个函数的结果时，属性名包含函数名称。

partkey	name	availqty	retailprice	supplycost	comment
3	spring green yellow purple cornsilk	4651	903	920.92	egular deposits hag
28	navajo yellow drab white misty	9988	928.02	666.53	x-ray pending. iron
53	bisque rose cornsilk seashell purple	8200	953.05	388.08	motot
78	blush forest slate seashell puff	7246	978.07	577.23	icing deposits wake
103	navy sky spring orchid forest	5913	1003.1	905.88	e blithely blith
125	mint ivory saddle peach midnight	5546	1025.12	806.66	kages against
151	chartreuse linen violet ghost thistle	1484	1051.15	71.68	ccounts nag i
177	indian turquoise purple green spring	4349	1077.17	63.36	ermanently eve
203	beige cornflower gainsboro chiffon wheat	1677	1103.2	399.16	usual instructio
222	aquamarine puff antique drab beige	8855	1122.22	926.1	ross the ironic. un
249	hot sandy lavender saddle rosy	1861	1149.24	914.16	excuses kindle f
276	orange yellow drab grey blanched	2309	1176.27	612.63	iously.
303	almond chocolate firebrick black bisque	9794	1203.3	499.32	ily according

2) 创建视图（不能省略列名的情况）

```
CREATE VIEW V_CustAvgOrder(custkey, cname, avgprice, avgquantity) AS
SELECT C.custkey, MAX(C.name), AVG(O.totalprice), AVG(L.quantity)
FROM customer C, orders O, lineitem L
WHERE C.custkey = O.custkey AND L.orderkey = O.orderkey
GROUP BY C.custkey;
```

在不省略列名时，MAX、AVG 函数并没有出现在最终的属性名中，而是被对应给定的属性名替代。

custkey	cname	avgprice	avgquantity
370	Customer#000000370	146841.5119565217	23.01086956521739
781	Customer#000000781	152562.27239999996	26.48
1234	Customer#000001234	171015.2293023255	23.41860465116279
1369	Customer#000001369	163518.80117021283	23.70212765957447
445	Customer#000000445	169403.6637681159	26.420289855072465
557	Customer#000000557	177680.55027777774	28.194444444444443
392	Customer#000000392	135412.1668085107	22.72340425531915
1301	Customer#000001301	178666.92937500004	26.34375
670	Customer#000000670	173307.880291262	25.815533980582526
611	Customer#000000611	125406.87166666666	22.805555555555557
1276	Customer#000001276	199109.3466386556	25.45378151260504
1153	Customer#000001153	150603.13712328763	24.383561643835616
862	Customer#000000862	169082.6746774194	26.919354838709676

3) 创建视图（WITH CHECK OPTION）

WITH CHECK OPTION 使 VIEW V_DLMU_PartSupp2 可以被更改，而且在视图上的更改会被同步到 partsupp 表中。

```
CREATE VIEW V_DLMU_PartSupp2 AS
SELECT partkey, suppkey, availqty, supplycost
FROM partsupp
WHERE suppkey = (SELECT suppkey FROM supplier
                 WHERE name = 'Supplier#000000004')
WITH CHECK OPTION;
```

partkey	suppkey	availqty	supplycost
3	4	4651	920.92
28	4	9988	666.53
53	4	8200	388.08
78	4	7246	577.23
103	4	5913	905.88
125	4	5546	806.66
151	4	1484	71.68
177	4	4349	63.36
203	4	1677	399.16
222	4	8855	926.1
249	4	1861	914.16
276	4	2309	612.63
303	4	9794	499.32

Supplier#000000004 的 suppkey 为 4，因此新插入的值的 suppkey 也必须为 4，否则会“check failed”。

```
INSERT INTO v_dlmu_partsupp2 VALUES(58889, 4, 704, 77760);
```

查询 v_dlmu_partsupp2 表，发现上面的记录被成功插入到其中。

partkey	suppkey	availqty	supplycost
58889	4	704	77760

再查询 PartSupp 的更改，发现表中也多出了 partkey 为 58889 的元组。

partkey	suppkey	availqty	supplycost	comment
58889	4	704	77760	NULL
NULL	NULL	NULL	NULL	NULL

使用 UPDATE 更新 v_dlmu_partsupp2。教材中搞错了，WHERE 的条件应该是 partkey = 58889 而不是 suppkey。

```
UPDATE v_dlmu_partsupp2 SET supplycost = 12
WHERE partkey = 58889;
```

分别查询 v_dlmu_partsupp2 视图和 partsupp 表，发现 partkey 为 58889 的元组的 supplycost 都变成了 12。

partkey	suppkey	availqty	supplycost
58889	4	704	12

partkey	suppkey	availqty	supplycost	comment
58889	4	704	12	NULL

```
DELETE FROM v_dlmu_partsupp2 WHERE partkey = 58889;
SELECT * FROM v_dlmu_partsupp2 WHERE partkey = 58889;
```

分别查询 v_dlmu_partsupp2 视图和 partsupp 表，都已经没有 partkey 为 58889 的元组。

partkey	suppkey	availqty	supplycost

partkey	suppkey	availqty	supplycost	comment
NULL	NULL	NULL	NULL	NULL

4) 可更新的视图（行列子集视图）

```
CREATE VIEW V_DLMU_PartSupp3 AS
SELECT partkey, suppkey, availqty, supplycost FROM partsupp
WHERE suppkey = (SELECT suppkey FROM supplier
                 WHERE name = 'Supplier#000000004');
```

结果同上。

5) 不可更新的视图

```
INSERT INTO v_custavgorder VALUES(100000, NULL, 20, 2000);
```

无法更新，错误如下：

Error Code: 1471. The target table v_custavgorder of the INSERT is not insertable-into

6) 删除视图（RESTRICT/CASCADE）

```
CREATE VIEW V_CustOrd(custkey, cname, qty, extprice) AS
SELECT C.custkey, C.name, L.quantity, L.extendedprice
FROM customer C, orders O, lineitem L
WHERE C.custkey = O.custkey AND O.orderkey = L.orderkey;

CREATE VIEW V_CustAvgOrder2(custkey, cname, avgqty, avgprice) AS
SELECT custkey, MAX(cname), AVG(qty), AVG(extprice)
FROM v_custord GROUP BY custkey;

DROP VIEW v_custord RESTRICT;
DROP VIEW v_custord CASCADE;
```

RESTRIC 和 CASCADE 都只是删除了 v_custord 视图，并没有删除 V_CustAvgOrder2 视图，而且在刷新是发生错误：Error Code: 1356 View 'sales.v_custavgorder2' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them. 即 v_custavgorder2 引用了一个不存在的 v_custord 视图。

上网查找原因，RESTRIC 保证只有当不存在依赖的外键时才能删除成功，CASACDE 会删除表及依赖的表，也许是 MySQL 中行为与 SQL server 不同。

六、 索引实验

1) 创建唯一索引

在 part 表中有些零件的名字相同，不同创建唯一索引，因此选择 nation 表。

```
CREATE UNIQUE INDEX Idx_nation_name ON nation(name);
```

查看 nation 表的所有 index，发现多了一个 Idx_nation_name，且为唯一，另一个 index 是其主键。

Indexes in Table

Visible	Key	Type	Uni	Columns
☑	PRIMARY	BTREE	YES	nationkey
☑	Idx_nation_name	BTREE	YES	name

2) 创建函数索引

MySQL 中没有函数索引。

3) 创建复合索引

```
CREATE UNIQUE INDEX Idx_cust_name_addr ON customer(name, address);
```

同样去查询 customer 的 index，多出了一个 Idx_cust_name_addr，用到了 name 和 address 属性，且是唯一的。

Indexes in Table

Visible	Key	Type	Uni	Columns
☑	PRIMARY	BTREE	YES	custkey
☑	Idx_cust_name_addr	BTREE	YES	name, address

4) 创建聚簇索引

MySQL 中没有聚簇索引。

5) 创建 Hash 索引

MySQL 中没有 Hash 索引。

6) 修改索引名称

7) 分析某个 SQL 查询语句执行时是否使用了索引

如下图，key 这一列表明查询过程用到了 Idx_nation_name 索引。

```
EXPLAIN SELECT * FROM nation WHERE name = 'CHINA';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered
1	SIMPLE	nation	NULL	const	Idx_nation_name	Idx_nation_name	51	const	1	100.00

8) 验证索引效率

由于在函数定义是遇到很多语法上的问题，而且 MySQL workbench 本身能看到一条语句执行的时间，因此直接查看。最坏情况下应该是表中没有对应记录，因此将筛选条件为 name = 'lalala'。

```
SELECT * FROM part WHERE name = 'lalala';
```

一开始 part 中有 2000 条记录，没有索引一次查询时间极短；

Time	Action	Message	Duration / Fetch
17:42:29	SELECT * FROM part WHERE name = 'lalala' LIMI...	0 row(s) returned	0.000 sec / 0.000 sec

将 part 记录数增加到 512000 条，无索引时一次查询要 0.515 秒；

17:43:48	SELECT * FROM part WHERE name = 'lalala' LI...	0 row(s) returned	0.515 sec / 0.000 sec

```
CREATE INDEX part_name ON part(name);
```

在 part 上创建 name 的索引，一次查询时间极短，与 2000 条时相当，可以看出索引极大地提升了查询效率。

17:44:34	SELECT * FROM part WHERE name = 'lalala' LI...	0 row(s) returned	0.000 sec / 0.000 sec

七、 实验总结

这次实验可以总结为两个字——曲折。一是因为用的是 MySQL 而书上的参考代码是 SQL server 的，而是 MySQL 版本为 8.0 相对较新，很多东西已经发生变化。

由于之前学过一段时间的 MySQL，比较偏爱 MySQL 的语法和语义，因此一开始就决定用 MySQL 来做这个实验。原本想着 MySQL 的语法和 SQL server 不会有多大区别，毕竟都是关系型数据库。事实也是如此，然而仅仅是那一小部分的区别就足以让人头秃。在 1.1 数据库定义实验中，需要给数据库设置编码格式，参考代码用的是 ENCODING='GBK'的方式，上网查教程了解到 MySQL 要设置字符集和 COLLATE；在 1.4 数据库更新实验中需要创建一个与旧表结构相同的表，SQL server 用了 WITH NO DATA 的方式。在百度上搜索，发现 MySQL 中有两种方式，一种是用 create table tableA select * from tableB 创建新表并把旧表的所有数据拷贝到新表，然后用 delete from tableA 删除新表的所有元组；另一种方式比较简单，create table tableA like tableB，效率也相对较高。之后的几个实验也遇到不少的问题，比如索引实验中 MySQL 没有对应的几个机制。

其次，由于新版本的 MySQL 有些语法、配置改变，也给实验带来了一定的麻烦。比如，MySQL 8.0 默认的数据导入目录为安装目录下的 Uploads 目录下，想要导入数据时，需要填写文件的绝对路径，因为工作目录不在 Uploads 中。为了能够导入其他目录的数据，需要设置 my.ini 文件中的 secure-file-priv 选项，而不是像旧版本一样直接在 MySQL 客户端中 set。不得不说，找 my.ini 的过程本身就很让人抓狂，因为 8.0 的 MySQL 有两个 my.ini，一个在安装目录（D 盘），另一个在 C 盘同名的目录下。一开始修改的是 C 盘的 my.ini，结果无论如何重启 MySQL 服务 secure-file-priv 都不变，后来在服务中查看详情，才发现 MySQL 服务用的是 D 盘的 my.ini。

当然，这次实验也让我的理论知识得到了很好的巩固。理论课进度比较快，而且又比较抽象，很多地方都没有学习得很扎实，基本语法也不是很熟悉。这次实验一次性回顾了理论课本的大部分内容，让我在实践的过程中进一步熟悉了语法和语义，更重要的是理清了语句的逻辑关系。我觉得这是很关键的，因为 SQL 的语句很简单很巧妙，很多看似简单的工作，想要转换成具体的 SQL 语句往往不是那么容易。这次实验中强迫自己不看参考代码先实现一遍看效果，逐渐熟悉了某些类型的问题转换成 SQL 语句的方法。

以上就是我在这次实验中遇到的问题和收获。