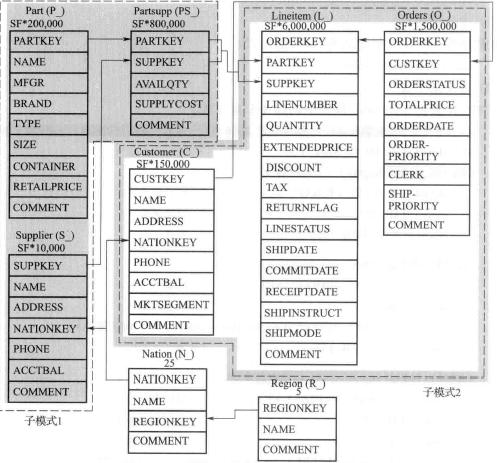
# 实验1-数据库定义与操作语言实验

## 实验环境

|  |  |
| --- | --- |
| 平台 | Windows 10 |
| SQL | SQL server 2017 |

## 数据库定义实验

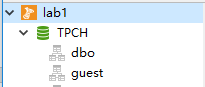
本实验建立TPC-H数据库模式。TPC-H数据库模式由零件表（Part）、供应商表（Supplier）、零件供应商表（Partsupp）、顾客表（Customer）、国家表（Nation）、地区表（Region）、订单表（Orders）和订单明细表（Lineitem）8个基本表组成。

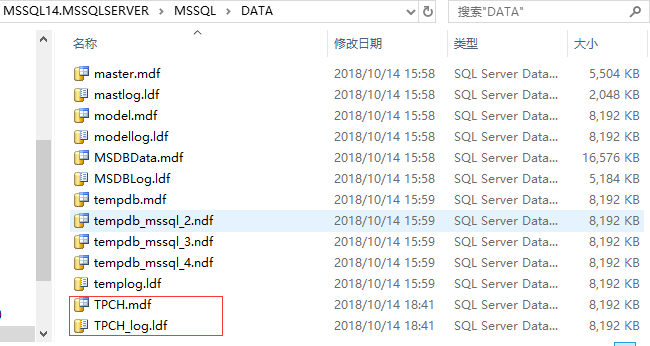


1. 定义数据库

采用中文字符集创建名为TPCH的数据库。设置数据库字符集的语句是“COLLATE Chinese\_PRC\_CI\_AS”，教材中“ENCODING = ‘GBK’”的语法有误。

CREATE DATABASE TPCH COLLATE Chinese\_PRC\_CI\_AS;





1. 定义模式

在数据库TPCH中创建名为Sales的模式。

CREATE SCHEMA Sales;

在SQL server中，数据库由多个模式组成，如同一间大楼有很多个房子一样。

1. 定义基本表

在Sales模式中创建8个基本表。新版本的SQL server无SEARCH\_PATH，因此创建表是必须加上模式名，否则表将创建者dbo模式下。

CREATE TABLE Sales.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

);

CREATE TABLE Sales.Nation ( -- 国家表

nationkey INTEGER PRIMARY KEY, -- 国家编号（主键）

name CHAR(25), -- 国家名称

regionkey INTEGER

REFERENCES Sales.Region (regionkey), -- 地区编号（外键）

comment VARCHAR(152) -- 备注

);

CREATE TABLE Sales.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 Sales.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

);

CREATE TABLE Sales.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

);

CREATE TABLE Sales.PartSupp ( -- 零件供应联系表

partkey INTEGER

REFERENCES Sales.Part (partkey), -- 零件编号（外键）

suppkey INTEGER

REFERENCES Sales.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

);

CREATE TABLE Sales.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 Sales.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

);

CREATE TABLE Sales.Orders ( -- 订单表

orderkey INTEGER PRIMARY KEY, -- 订单编号（主键）

custkey INTEGER REFERENCES Sales.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

);

CREATE TABLE Sales.Lineitem ( -- 订单明细表

orderkey INTEGER

REFERENCES Sales.Orders (orderkey), -- 订单编号

partkey INTEGER

REFERENCES Sales.Part (partkey), -- 零件编号

suppkey INTEGER

REFERENCES Sales.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

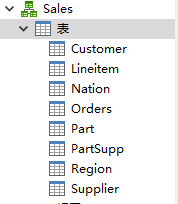
comment VARCHAR(44), -- 备注

PRIMARY KEY (orderkey , linenumber),

FOREIGN KEY (partkey , suppkey)

REFERENCES Sales.PartSupp (partkey , suppkey)

);

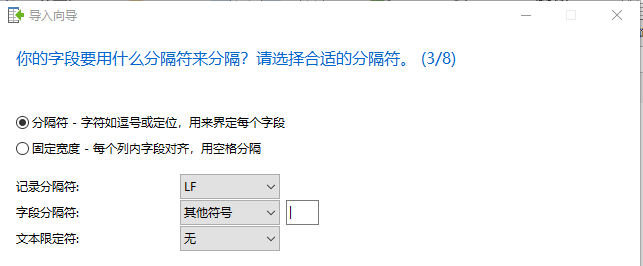


## 数据基本查询实验

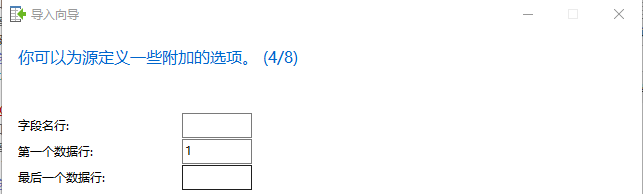
1. 导入数据

在表上右键“导入向导”，使用txt文件导入，过程中的设置如下：

1. 设置字段分隔符为‘|’（竖线）和行结束符为LF：



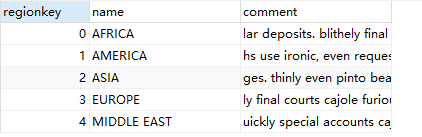
1. 第一个数据行设置为1



1. 勾选表的主键，选择表的属性和文件的字段的对应关系



结果如下，其他表导入过程类似，注意表导入的先后关系，被引用的表先导入：

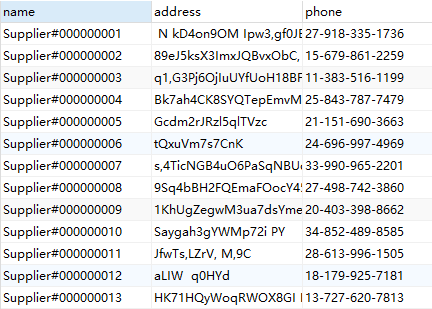


1. 单表查询（投影）

查询供应商的名称、地址和联系电话。

SELECT name, address, phone FROM Sales.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：



1. 单表查询（选择）

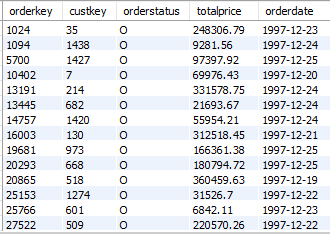
查询最近一周内提交的总价大于1000元的订单的编号、顾客编号等订单的所有信息。由于表中日期是很久之前的，因此假设CURRENT\_DATE为DATE("1997-12-25")。

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。

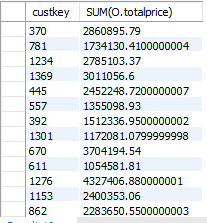


1. 不带分组过滤条件的分组统计查询

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。



1. 带分组过滤条件的分组统计查询

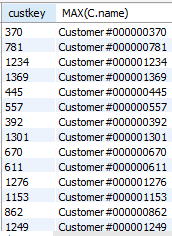
由于name属性不在分组条件中，因此需要用聚集函数（如max）。

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。



1. 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。



1. 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。



1. 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。



1. 三表连接查询

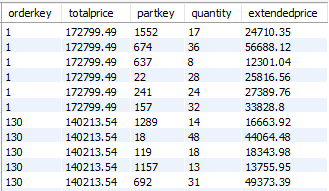
三表连接与两表连接同理，customer表与orders表都具有custkey属性，orders表和lineitem表都具有orderkey属性，在WHERE子句保证取值一样。

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。



## 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';



1. 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)

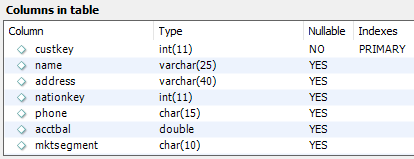
VALUES (862, ROUND(RAND() \* 100.0), 1, 27, 10, '2012-3-6');

1. 批量数据INSERT语句
   1. 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。



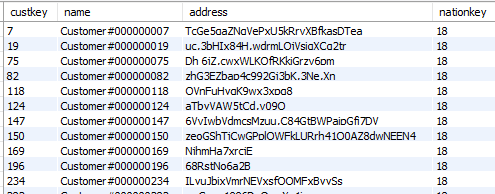
INSERT INTO newcustomer

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。



* 1. 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

);

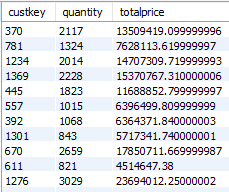
INSERT INTO shoppingstat

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;



* 1. 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。



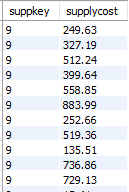
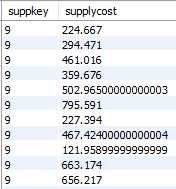
1. UPDATE语句（修改部分记录的部分列值）

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。

1. UPDATE语句（利用一个表中的数据修改另外一个表中的数据）

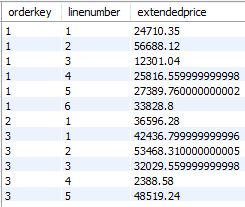
MySQL中把表P放在FROM子句的语法是错误的，正确方式是放在UPDATE之后。

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。



1. DELETE基本语句（删除给定条件的所有记录）

DELETE FROM lineitem

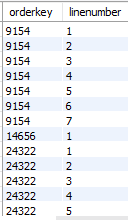
WHERE orderkey IN (SELECT orderkey FROM orders O, customer C

                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。

## 视图实验

1. 创建视图（省略视图列名）

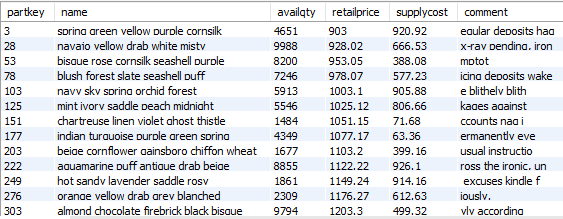
CREATE VIEW V\_DLMU\_PartSupp1 AS

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。



1. 创建视图（不能省略列名的情况）

CREATE VIEW V\_CustAvgOrder(custkey, cname, avgprice, avgquantity) AS

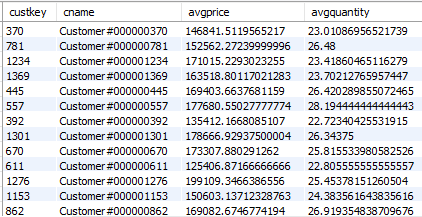
SELECT C.custkey, MAX(C.name), AVG(O.totalprice), AVG(L.quantity)

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。



1. 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

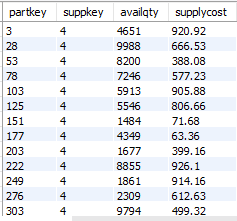
SELECT partkey, suppkey, availqty, supplycost

FROM partsupp

WHERE suppkey = (SELECT suppkey FROM supplier

                 WHERE name = 'Supplier#000000004')

WITH CHECK OPTION;



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。



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



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。





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。





1. 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。

1. 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

1. 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。

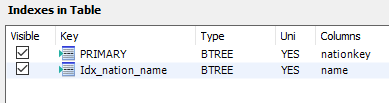
## 索引实验

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。



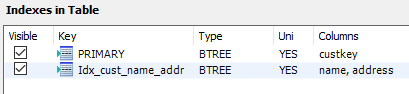
1. 创建函数索引

MySQL中没有函数索引。

1. 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。



1. 创建聚簇索引

MySQL中没有聚簇索引。

1. 创建Hash索引

MySQL中没有Hash索引。

1. 修改索引名称
2. 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';



1. 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；



将part记录数增加到512000条，无索引时一次查询要0.515秒；



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。



## 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。