

数据库系统 课程实验1

数据定义/数据操纵

计科210X 甘晴void 202108010XXX

目录

数据库系统 课程实验1
数据定义/数据操纵

实验目的

实验样例

实验环境

实验内容

1.1 数据库定义

- 1) 实验内容与要求
- 2) 实验重难点
- 3) 实验基础知识
 - ①模式的定义与删除
 - ②基本表的定义、删除与修改
 - ③模式与基本表
 - ④课本示例
 - ⑤数据类型
 - ⑥索引的建立与删除
 - ⑦数据字典（记录数据定义）
- 4) 实验过程
 - ①查看现存所有数据库
 - ②创建数据库
 - ③创建基本表

1.2 数据基本查询

- 1) 实验内容与要求
- 2) 实验重难点
- 3) 实验基础知识
 - ①单表查询
 - ②连接查询
 - ③嵌套查询
 - ④集合查询（了解即可）
 - ⑤基于派生表的查询
- 4) 实验过程
 - ①向三个表中填充提供的参考数据
 - ②单表查询

- ③分组统计查询
- ④多个表的连接查询

1.3 数据高级查询

- 1) 实验内容与要求
- 2) 实验重难点
- 3) 实验基础知识
- 4) 实验过程
 - ①相关子查询与不相关子查询
 - ②谓词ANY查询 和 带EXISTS的查询
 - ③多层嵌套EXISTS 与 集合查询

1.4 数据更新

- 1) 实验内容与要求
- 2) 实验重难点
- 3) 实验基础知识
 - ①插入数据
 - ②修改数据
 - ③删除数据
 - ④空值的处理
- 4) 实验过程
 - ①插入单个元组
 - ②带子查询的插入
 - ③修改多个元组的值

1.5 视图

- 1) 实验内容与要求
- 2) 实验重难点
- 3) 实验基础知识
- 4) 实验过程
 - ①创建行列子集视图
 - ②创建分组视图
 - ③创建视图（WITH CHECK OPTION）
 - ★出现问题:
 - ④可更新视图与不可更新视图
 - ⑤视图消解
 - ⑥删除视图

1.6 索引

- 1) 实验内容与要求
- 2) 实验重难点
- 3) 实验基础知识
- 4) 实验过程
 - ①生成一个足够大的数据集
 - ②展示索引
 - ③唯一索引
 - ④简单索引和复合索引

实验目的

通过 SQL 完成对数据库的定义、查询、更新、建立和使用视图及索引等相关操作。

实验样例

本实验使用学生-课程数据库 S-T，各位同学也可自行设计别的数据库进行实验。

以下为 S-T 数据库的描述。S-T 数据库如下：（红色的表示主码）

学生表：Student (**Sno**, Sname, Sex, Sage, Sdept)

学号 Sno	姓名 Sname	性别 Sex	年龄 Sage	所在系 Sdept
201215121	李勇	男	20	CS
201215122	刘晨	女	19	CS
201215123	王敏	女	18	MA
201215125	张立	男	19	IS

课程表：Course (**Cno**, Cname, Cpno, Ccredit)

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

学生选课表：SC (**Sno**, **Cno**, Grade)

学号 Sno	课程号 Cno	成绩 Grade
201215121	1	92
201215121	2	85
201215121	3	88
201215122	2	90
201215122	3	80

实验环境

DBMS: 8.0.33 MySQL Community Server - GPL

可视化: Navicat Premium 16.1.6

命令行: 由Microsoft商店提供的windows终端(version=1.18.2822.0)启动的windows powershell命令行

启动方式:

由于数据库是在本地的

```
mysql -u root -p
```

回车并输入密码

实验内容

1.1 数据库定义

1) 实验内容与要求

理解和掌握数据库 DDL 语言，能够熟练地使用 SQL DDL语句创建、修改和删除数据库和基本表。

2) 实验重难点

创建数据库、基本表，正确创建表级和列级完整性约束（例如列值是否允许为空，列值是否为主码、外码等），完整性约束可以在创建表时定义，也可以在创建表之后定义，但是需要注意被引用的表需要先创建。

3) 实验基础知识

```

SHOW DATABASES;
CREATE DATABASE HNUTEST;
DROP DATABASE HNUTEST;
USE HNUTEST;
CREATE TABLE STUDENT(/*这里写子句*/);
ALTER TABLE STUDENT ADD S1 CHAR(5);
DROP TABLE STUDENT RESTRICT|CASCADE;

```

①模式的定义与删除

#定义模式

```
create schema [模式名] authorization <用户名>;
```

#删除模式

```
drop schema <模式名><cascade|restrict> //级联与限制必须二选一（CASCADE会删除所有，而若该模式内有东西，RESTRICT会拒绝删除）
```

②基本表的定义、删除与修改

#基本表的定义

```
create table <表名>(<列名><数据类型>[列级完整性约束]
.....
[,<表级完整性约束>]);
```

#一些表级完整性约束

```
PRIMARY KEY(Sno,Cno);
```

```
FOREIGN KEY(Sno) REFERENCES Student(Sno);
```

列级完整性/表级完整性

#基本表的修改

```
ALTER TABLE <表名>
```

```
[ADD [COLUMN] <新列名><数据类型>[完整性约束]]
```

```
[ADD <表级完整性约束>]
```

```
[DROP [column] <列名> [cascade|restrict]]
```

```
[DROP constraint <完整性约束名> [restrict|cascade]]
```

```
[RENAME COLUMN <列名> to <新列名>]
```

```
[ALTER COLUMN <列名> TYPE <数据类型>]
```

#基本表的删除

```
DROP TABLE <表名> [restrict|cascade] #▲注意区别，默认RESTRICT
```

③模式与基本表

#基本表属于某一个特定的模式，下面是创建特定模式下基本表的方法。

#可以在指定模式下创建基本表

```
CREATE TABLE "SC-C-SC".Student (.....)
```

#可以设定模式然后创建基本表

```
SHOW SEARCH_PATH;#可以查看当前搜索路径
```

```
SET SEARCH_PATH TO "SC-C-SC",PUBLIC;#设置当前模式
```

#之后再创建基本表

④课本示例

#课本示例

/*建立一个学生表*/

```
CREATE TABLE Student(  
    Sno CHAR(9) PRIMARY KEY,  
    Sname CHAR(20) UNIQUE,  
    Ssex CHAR(2),  
    Sage SMALLINT,  
    Sdept CHAR(20)  
);
```

/*建立一个课程表*/

```
CREATE TABLE Course(  
    Cno CHAR(4) PRIMARY KEY,  
    Cname CHAR(40) NOT NULL,  
    Cpno CHAR(4),  
    Ccredit SMALLINT,  
    /*表级完整性约束条件*/  
    FOREIGN KEY (Cpno) REFERENCES Course(Cno)  
);
```

/*建立学生选课表SC*/

```
CREATE TABLE SC(  
    Sno CHAR(9),  
    Cno CHAR(4),  
    Grade SMALLINT,  
    PRIMARY KEY(Sno,Cno),  
    FOREIGN KEY(Sno) REFERENCES Student(Sno),  
    FOREIGN KEY(Cno) REFERENCES Course(Cno)  
);
```

⑤数据类型

教材P75

⑥索引的建立与删除

在后面涉及

⑦数据字典（记录数据定义）

4) 实验过程

①查看现存所有数据库

我想知道现在我的DBMS内都有哪些数据库

```
SHOW DATABASES;
```

②创建数据库

```
CREATE DATABASE HNUTEST;
```

截图如下

```
mysql> CREATE DATABASE HNUTEST;
Query OK, 1 row affected (0.00 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| hnutest  |
| homework|
| information_schema |
| mysql    |
| performance_schema |
| sakila    |
| sys      |
| world    |
+-----+
8 rows in set (0.00 sec)
```

创建完数据库之后，记得要“选中”与“进入”你创建的数据库。

③创建基本表

根据题目提供的参考数据，创建三个表

- Student表中以Sno作为主码
- Course表中Cno作为主码，引用本身的Cno（主码）为Cpno属性列（非主码）作为外码
- 表SC以Sno、Cno属性组作为主码，Sno引用自Student的主键Sno，Cno引用自Course表的主键Cno

SQL代码如下

```
CREATE TABLE Student( /*学生信息表*/
    Sno CHAR(9) PRIMARY KEY,
    Sname CHAR(20) NOT NULL,
    Ssex CHAR(2) NOT NULL,
    Sage SMALLINT NOT NULL,
    Sdept CHAR(20) NOT NULL
);

CREATE TABLE Course( /*课程信息表*/
    Cno CHAR(9) PRIMARY KEY,
    Cname CHAR(20) NOT NULL,
    Cpno CHAR(9),
    Ccredit SMALLINT,
    FOREIGN KEY(CPNO) REFERENCES COURSE(CNO)
);

CREATE TABLE SC( /*选课信息表*/
    Sno CHAR(9),
    Cno CHAR(4),
    Grade SMALLINT,
    PRIMARY KEY(SNO,CNO),
    FOREIGN KEY(SNO) REFERENCES STUDENT(SNO),
    FOREIGN KEY(CNO) REFERENCES COURSE(CNO)
);
```

本部分实验完毕，在navicat中查看，可见符合效果。



1.2 数据基本查询

1) 实验内容与要求

掌握 SQL 程序设计基本规范，熟练运用 SQL 语言实现数据基本查询，包括单表查询、分组统计查询和连接查询；设计单个表对自身的连接查询，多个表的连接查询，按照 SQL 程序设计规范写出具体的 SQL 查询语句，并能够正确运行。

2) 实验重难点

分组统计查询，单表自身连接查询和多表连接查询，确定连接属性，正确设计连接条件。

3) 实验基础知识

基本形式如下：

```
select[all|distinct]<目标列表表达式>[别名][,<目标列表表达式>[别名]].....  
from <>  
[where<条件表达式>]  
[group by <某个列名> [having <条件表达式>]]  
[order by <某个列名> [asc|desc]]  
[limit <行数1>[offset <行数2>]]
```

知识量过多，详见书本。在实验过程中我尽量涉及到。下面是一些具体叙述。

①单表查询

#选择表中的若干列（投影操作）

```
SELECT Sno FROM Student;
```

#选择表中的若干组（选择操作）

```
SELECT DISTINCT Sno FROM Student;
```

#比较大小

```
WHERE = > < <> != !> !<
```

#确定范围

```
SELECT Sno FROM Student WHERE Sage BETWEEN 20 AND 21;
```

```
SELECT Sno FROM Student WHERE Sage NOT BETWEEN 20 AND 21;
```

#确定集合

```
IN | NOT IN
```

#字符匹配

% #任意长度字符串

_ #任意单个字符

```
WHERE Sname LIKE '___寅';
```

```
WHERE Sname NOT LIKE '%寅';
```

#涉及到含有通配符，可以后跟ESCAPE'\ '进行转义

```
#WHERE Sname NOT LIKE '%寅\_ ' ESCAPE '\ ';
```

#涉及空值的查询

```
WHERE GRADE IS NULL
```

```
WHERE GRADE IS NOT NULL
```

#多重条件查询

```
AND OR 括号
```

#ORDER BY子句

```
ORDER BY grade DESC | ASC #（排序操作）允许第二关键字
```

#查询结果排序，不影响原表

#聚集函数（数理统计）

#不能用作WHERE子句条件表达式

#只能作为SELECT子句和GROUP BY子句的HAVING短语

```
COUNT(*)
```

```
COUNT([ DISTINCT | ALL ]<列名>)
```

支持 sum avg max min

#e.g.

```
SELECT COUNT(DISTINCT Sno) From SC;
```

#GROUP BY子句（分组筛选）★★★

```
GROUP BY
```

```
SELECT Sno FROM SC GROUP BY Sno HAVING COUNT(*) > 10;
```

#Limit子句（只选择前多少个子句，常与ORDER BY连用）

```
LIMIT 10 OFFSET 5 #显示10个忽略前5个
```

②连接查询

自然连接查询 复合条件连接查询

```
SELECT student.sno,sname
FROM student,sc
WHERE student.sno=sc.sno AND
      sc.cno='2' AND sc.grade>90;
```

自身连接，需要为表起别名

#e.g. 间接先修课（先修课的先修课）

```
SELECT first.cno,seconde.cjno
FROM course first,course second
WHERE first.cjno=second.cno;
```

外连接，把表中的悬浮元组保存在结果关系中

```
SELECT student.sno,sname,ssex,sage,sdept,cno,grade
FROM student LEFT OUTER JOIN sc ON (student.sno=sc.sno);
```

多表连接和以上类似

③嵌套查询

查询块：一个select-from-where

嵌套查询，外层查询（父查询），内层查询（子查询）

【子查询不能用order by】

any,all,

exist谓词：只返回逻辑真假

存在量词与全称量词

不相关子查询

```
SELECT name
FROM student
```

```

WHERE dept IN
    (
        SELECT dept
        FROM student
        WHERE name='刘晨'
    );
# 相关子查询
SELECT sno,cno
FROM SC x
WHERE Grade >= (
    SELECT AVG(Grade)
    FROM SC y
    WHERE y.sno=x.sno
);
# ANY(SOME)/ALL
SELECT name,age
FROM student
WHERE age<ALL
    (
        SELECT age
        FROM student
        WHERE dept='CS'
    )
    AND dept!='CS';
# EXIST
SELECT name
FROM student
WHERE EXISTS
    (
        SELECT *
        FROM SC
        WHERE sno=student.sno AND cno='1'
    );

```

★全称量词的转换

看书

逻辑蕴含的转换

④集合查询（了解即可）

```
# UNION
SELECT *
FROM student
WHERE dept='CS'
UNION (UNION ALL)
SELECT *
FROM student
WHERE age<=19;

# INTERSECT
SELECT *
FROM student
WHERE dept='CS'
INTERSECT
SELECT *
FROM student
WHERE age<=19;

# EXCEPT
SELECT *
FROM student
WHERE dept='CS'
EXCEPT
SELECT *
FROM student
WHERE age<=19;
```

⑤基于派生表的查询

```
SELECT Sno,Cno
FROM SC,(SELECT Sno,Avg(Grade) FROM SC GROUP BY Sno)
      AS Avg_SC(Avg_sno,Avg_grade)
WHERE SC.Sno = Avg_SC.Avg_sno AND SC.Grade >= Avg_SC.Avg_grade;
```

4) 实验过程

①向三个表中填充提供的参考数据

这一部分填充数据比较多，我直接使用navicat来填充了，使用SQL语句也是可以操作的。

The screenshot displays three database tables in Navicat, each with a menu bar (文件, 编辑, 查看, 窗口, 帮助) and a toolbar (开始事务, 文本, 筛选, 排序, 列, 导入, 导出, 数据生成, 创建图表).

Table 1: student @hnutest (localhost_3306) - 表

Sno	Sname	Ssex	Sage	Sdept
201215121	李勇	男	20	CS
201215122	刘晨	女	19	CS
201215123	王敏	女	18	MA
201215125	张立	男	19	IS

Table 2: course @hnutest (localhost_3306) - 表

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学	(Null)	2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	(Null)	2
7	PASCAL语言	6	4

Table 3: sc @hnutest (localhost_3306) - 表

Sno	Cno	Grade
201215121	1	92
201215121	2	85
201215121	3	88
201215122	2	90
201215122	3	80

②单表查询

我按照书上介绍的单表查询，自己编了一些问题，然后解题。

查询选择了课程的学生的学号

```
SELECT DISTINCT SNO  
FROM SC;
```

查询所在系为“CS”的学生姓名

```
SELECT SNAME  
FROM STUDENT  
WHERE SDEPT='CS';
```

查询所在系不为“CS”或“MA”的学生学号

```

SELECT SNO
FROM STUDENT
WHERE SDEPT NOT IN ('CS','MA');

# 查询成绩在90分以下的学生学号
SELECT DISTINCT SNO
FROM SC
WHERE GRADE<90;

# 查询 Student 表中为姓刘的学生的学号和姓名
SELECT SNO,SNAME
FROM STUDENT
WHERE SNAME LIKE '刘%';

# 查询选修了2号课程的学生学号和2号课程的成绩，并将其按照降序排列
SELECT SNO,GRADE
FROM SC
WHERE CNO='2'
ORDER BY GRADE DESC;

# 查询学号为“201215121”的同学的最高分
SELECT MAX(GRADE)
FROM SC
WHERE SNO='201215121';

# 查询学号为“201515121”的同学的平均分
SELECT AVG(GRADE)
FROM SC
WHERE SNO='201215121';

```

运行结果如下

```

mysql> # 查询选择了课程的学生学号
mysql> SELECT DISTINCT SNO
-> FROM SC;
+-----+
| SNO      |
+-----+
| 201215121 |
| 201215122 |

```

```
+-----+
2 rows in set (0.00 sec)
```

mysql> # 查询所在系为“CS”的学生姓名

```
mysql> SELECT SNAME
      -> FROM STUDENT
      -> WHERE SDEPT='CS';
```

```
+-----+
```

```
| SNAME |
```

```
+-----+
```

```
| 李勇  |
```

```
| 刘晨  |
```

```
+-----+
```

```
2 rows in set (0.00 sec)
```

mysql> # 查询所在系不为“CS”或“MA”的学生学号

```
mysql> SELECT SNO
      -> FROM STUDENT
      -> WHERE SDEPT NOT IN ('CS', 'MA');
```

```
+-----+
```

```
| SNO      |
```

```
+-----+
```

```
| 201215125 |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```

mysql> # 查询成绩在90分以下的学生学号

```
mysql> SELECT DISTINCT SNO
      -> FROM SC
      -> WHERE GRADE<90;
```

```
+-----+
```

```
| SNO      |
```

```
+-----+
```

```
| 201215121 |
```

```
| 201215122 |
```

```
+-----+
```

```
2 rows in set (0.00 sec)
```


mysql> # 查询 Student 表中为姓刘的学生的学号和姓名

```
mysql> SELECT SNO,SNAME
-> FROM STUDENT
-> WHERE SNAME LIKE '刘%';
```

```
+-----+-----+
| SNO      | SNAME |
+-----+-----+
| 201215122 | 刘晨  |
+-----+-----+
1 row in set (0.00 sec)
```

mysql> # 查询选修了2号课程的学生学号和2号课程的成绩，并将其按照降序排列

```
mysql> SELECT SNO,GRADE
-> FROM SC
-> WHERE CNO='2'
-> ORDER BY GRADE DESC;
```

```
+-----+-----+
| SNO      | GRADE |
+-----+-----+
| 201215122 | 90    |
| 201215121 | 85    |
+-----+-----+
2 rows in set (0.00 sec)
```

mysql> # 查询学号为“201215121”的同学的最高分

```
mysql> SELECT MAX(GRADE)
-> FROM SC
-> WHERE SNO='201215121';
```

```
+-----+
| MAX(GRADE) |
+-----+
| 92          |
+-----+
1 row in set (0.00 sec)
```

mysql> # 查询学号为“201515121”的同学的平均分

```
mysql> SELECT AVG(GRADE)
-> FROM SC
-> WHERE SNO='201215121';
```

```

+-----+
|  AVG(GRADE)  |
+-----+
|    88.3333   |
+-----+
1 row in set (0.00 sec)

```

读者可以和原表比对一下，应该是正确的，这一部分比较基础。

③分组统计查询

```

# 求每个学生的学号与平均成绩
SELECT SNO,AVG(GRADE)
FROM SC
GROUP BY SNO;

```

运行结果如下：

```

mysql> # 求每个学生的学号与平均成绩
mysql> SELECT SNO,AVG(GRADE)
      -> FROM SC
      -> GROUP BY SNO;

+-----+-----+
| SNO      |  AVG(GRADE)  |
+-----+-----+
| 201215121 |    88.3333   |
| 201215122 |    85.0000   |
+-----+-----+
2 rows in set (0.00 sec)

```

③单个表对自身的连接查询

```
# 查询与'信息系统'学分相同的课程的课程号和课程名
SELECT FIRST.CNO,FIRST.CNAME
FROM COURSE FIRST,COURSE SECOND
WHERE FIRST.CCREDIT=SECOND.CCREDIT AND
      SECOND.CNAME='信息系统';

# 其实用嵌套实现更为直观，但自身连接也是解决问题的方法
SELECT CNO,CNAME
FROM COURSE
WHERE CCREDIT=
      (SELECT CCREDIT
      FROM COURSE
      WHERE CNAME='信息系统');
```

运行结果如下：

```
mysql> # 查询与'信息系统'学分相同的课程的课程号和课程名
mysql> SELECT FIRST.CNO,FIRST.CNAME
-> FROM COURSE FIRST,COURSE SECOND
-> WHERE FIRST.CCREDIT=SECOND.CCREDIT AND
-> SECOND.CNAME='信息系统';

+-----+-----+
| CNO | CNAME      |
+-----+-----+
| 1   | 数据库      |
| 3   | 信息系统    |
| 5   | 数据结构    |
| 7   | PASCAL语言  |
+-----+-----+
4 rows in set (0.00 sec)

mysql> # 其实用嵌套实现更为直观，但自身连接也是解决问题的方法
mysql> SELECT CNO,CNAME
-> FROM COURSE
-> WHERE CCREDIT=
-> (SELECT CCREDIT
-> FROM COURSE
-> WHERE CNAME='信息系统');
```

CNO	CNAME
1	数据库
3	信息系统
5	数据结构
7	PASCAL语言

```

+-----+-----+
| 1 | 数据库 |
| 3 | 信息系统 |
| 5 | 数据结构 |
| 7 | PASCAL语言 |
+-----+-----+
4 rows in set (0.00 sec)

```

④多个表的连接查询

```

# 输出三表连接的结果
SELECT
STUDENT.SNO, SNAME, SSEX, SAGE, SDEPT, COURSE.CNO, CNAME, CPNO, CCREDIT, GRA
DE
FROM STUDENT, COURSE, SC
WHERE STUDENT.SNO=SC.SNO AND COURSE.CNO=SC.CNO;

# 探究STUDENT和SC两表外连接查询
# 以STUDENT为主体，关注每一个学生的选课情况
SELECT *
FROM STUDENT
LEFT JOIN SC ON (STUDENT.SNO=SC.SNO);

# 以课程对应关系为主体，没有课程对应关系的学生不列出
SELECT *
FROM SC
LEFT JOIN STUDENT ON (STUDENT.SNO=SC.SNO);

```

运行结果如下：

```

mysql> # 输出三表连接的结果
mysql> SELECT
STUDENT.SNO, SNAME, SSEX, SAGE, SDEPT, COURSE.CNO, CNAME, CPNO, CCREDIT, GRA
DE
-> FROM STUDENT, COURSE, SC
-> WHERE STUDENT.SNO=SC.SNO AND COURSE.CNO=SC.CNO;

+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
| SNO      | SNAME | SSEX | SAGE | SDEPT | CNO | CNAME      | CPNO |
CCREDIT | GRADE |

```

```

+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
| 201215121 | 李勇 | 男 | 20 | CS | 1 | 数据库 | 5 |
| 4 | 92 |
| 201215121 | 李勇 | 男 | 20 | CS | 2 | 数学 | NULL |
| 2 | 85 |
| 201215121 | 李勇 | 男 | 20 | CS | 3 | 信息系统 | 1 |
| 4 | 88 |
| 201215122 | 刘晨 | 女 | 19 | CS | 2 | 数学 | NULL |
| 2 | 90 |
| 201215122 | 刘晨 | 女 | 19 | CS | 3 | 信息系统 | 1 |
| 4 | 80 |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
5 rows in set (0.00 sec)

```

mysql> # 以STUDENT为主体，关注每一个学生的选课情况

```

mysql> SELECT *
      -> FROM STUDENT
      -> LEFT JOIN SC ON (STUDENT.SNO=SC.SNO);

```

```

+-----+-----+-----+-----+-----+-----+-----+
--+
| Sno      | Sname | Ssex | Sage | Sdept | Sno      | Cno |
Grade |
+-----+-----+-----+-----+-----+-----+-----+
--+
| 201215121 | 李勇 | 男 | 20 | CS | 201215121 | 3 | 88
|
| 201215121 | 李勇 | 男 | 20 | CS | 201215121 | 2 | 85
|
| 201215121 | 李勇 | 男 | 20 | CS | 201215121 | 1 | 92
|
| 201215122 | 刘晨 | 女 | 19 | CS | 201215122 | 3 | 80
|
| 201215122 | 刘晨 | 女 | 19 | CS | 201215122 | 2 | 90
|
| 201215123 | 王敏 | 女 | 18 | MA | NULL | NULL | NULL
|
| 201215125 | 张立 | 男 | 19 | IS | NULL | NULL | NULL
|

```

```

+-----+-----+-----+-----+-----+-----+-----+
--+
7 rows in set (0.00 sec)

mysql> # 以课程对应关系为主体，没有课程对应关系的学生不列出
mysql> SELECT *
      -> FROM SC
      -> LEFT JOIN STUDENT ON (STUDENT.SNO=SC.SNO);
+-----+-----+-----+-----+-----+-----+-----+
--+
| Sno          | Cno | Grade | Sno          | Sname | Ssex | Sage | Sdept |
|
+-----+-----+-----+-----+-----+-----+-----+
--+
| 201215121 | 1   | 92    | 201215121 | 李勇  | 男   | 20   | CS    |
|
| 201215121 | 2   | 85    | 201215121 | 李勇  | 男   | 20   | CS    |
|
| 201215121 | 3   | 88    | 201215121 | 李勇  | 男   | 20   | CS    |
|
| 201215122 | 2   | 90    | 201215122 | 刘晨  | 女   | 19   | CS    |
|
| 201215122 | 3   | 80    | 201215122 | 刘晨  | 女   | 19   | CS    |
|
+-----+-----+-----+-----+-----+-----+-----+
--+
5 rows in set (0.00 sec)

```

1.3 数据高级查询

1) 实验内容与要求

掌握 SQL 嵌套查询和集合查询等各种高级查询的设计方法等；针对自定义的数据库，设计各种嵌套查询和集合查询。

2) 实验重难点

嵌套查询，相关子查询。

3) 实验基础知识

这一部分知识还是比较多且复杂。

重点在于书上的

- 相关与不相关子查询
- ANY与EXISTS谓词
- ★对于全称量词怎么处理（转化为双重否定）
- ★对于逻辑蕴含怎么处理（A推出B转化为非A析取B双重否定）

这些比较复杂，难以一下子概括清楚，故在下面实验过程中体现。对于逻辑蕴含我没有找到合适的例子，这个没有涉及到，读者可以自己思考。

4) 实验过程

①相关子查询与不相关子查询

```
# 不相关子查询 查询与李勇在同一个系的同学的学号、姓名和所在系
SELECT SNO, SNAME, SDEPT
FROM STUDENT
WHERE SDEPT IN
    (SELECT SDEPT
     FROM STUDENT
     WHERE SNAME='李勇');

# 相关子查询 查询每个学生超过他自己平均成绩的课程号和成绩
SELECT SNO, CNO, GRADE
FROM SC X
WHERE GRADE >
    (SELECT AVG(GRADE)
     FROM SC Y
     WHERE Y.SNO=X.SNO);
```

第一个查询表示不相关子查询，第一步的查询确定李勇所在的系名，第二部查询是查找所有在CS系中的学生，第一部查询嵌入到第二部查询中，先执行子查询再进行父查询；

第二个查询是相关子查询，旨在找出每个学生查过他自己选修课程平均成绩的课程号；X是表SC的别名，用来表示SC的一个元组，内层循环是求一个学生所有选修课程平均成绩，至于是哪一个学生的平均成绩需要看x.Sno的值，与父查询相关；其中该语句的执行过程如下：

- 外层查询中取出SC的一个元组x，将x的Sno传入内层查询
- 执行内层查询，得到平均值，用该值代替内层查询，得到外层查询
- 执行该外层查询

运行结果如下：

```
mysql> # 不相关子查询 查询与李勇在同一个系的同学的学号、姓名和所在系
```

```
mysql> SELECT SNO,SNAME,SDEPT
-> FROM STUDENT
-> WHERE SDEPT IN
-> (SELECT SDEPT
-> FROM STUDENT
-> WHERE SNAME='李勇');
```

```
+-----+-----+-----+
| SNO      | SNAME | SDEPT |
+-----+-----+-----+
| 201215121 | 李勇  | CS    |
| 201215122 | 刘晨  | CS    |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> # 相关子查询 查询每个学生超过他自己平均成绩的课程号和成绩
```

```
mysql> SELECT SNO,CNO,GRADE
-> FROM SC X
-> WHERE GRADE>
-> (SELECT AVG(GRADE)
-> FROM SC Y
-> WHERE Y.SNO=X.SNO);
```

```
+-----+-----+-----+
| SNO      | CNO  | GRADE |
+-----+-----+-----+
| 201215121 | 1    | 92    |
| 201215122 | 2    | 90    |
```



```
+-----+-----+-----+
2 rows in set (0.00 sec)
```

②谓词**ANY**查询 和 带**EXISTS**的查询

子查询返回单值时可以用比较运算符，返回多值时要用**ANY**；**EXISTS**代表存在量词，带有该谓词的子查询不返回任何数据，只产生逻辑真或假。

ANY查询 非CS系中比任意一个CS系学生年龄小的学生姓名与学号(这是书上很经典的题了，注意最后一个**AND**是父查询的)

```
SELECT SNAME,SNO
FROM STUDENT
WHERE SAGE<ANY
      (SELECT SAGE
       FROM STUDENT
       WHERE SDEPT='CS')
AND SDEPT<>'CS';
```

EXISTS查询 所有选择了2号课程的学生

```
SELECT SNAME
FROM STUDENT
WHERE EXISTS
      (SELECT *
       FROM SC
       WHERE SC.SNO=STUDENT.SNO AND SC.CNO='2');
```

前者为带**ANY**谓词的查询，查找非计算机系中比任意一个计算机系学生年纪小的学生姓名与学号。后者为带**EXISTS**的查询，查询所有选修了1号课程的学生。

运行结果如下：

```
mysql> # ANY查询 非CS系中比任意一个CS系学生年龄小的学生姓名与学号(这是书上很经典的题了，注意最后一个AND是父查询的)
```

```
mysql> SELECT SNAME,SNO
-> FROM STUDENT
-> WHERE SAGE<ANY
->      (SELECT SAGE
->         FROM STUDENT
->         WHERE SDEPT='CS')
-> AND SDEPT<>'CS';
```

```
+-----+-----+-----+
```

```
| SNAME | SNO          |
+-----+-----+
| 王敏   | 201215123   |
| 张立   | 201215125   |
+-----+-----+
2 rows in set (0.00 sec)
```

mysql> # EXISTS查询 所有选择了2号课程的学生

```
mysql> SELECT SNAME
-> FROM STUDENT
-> WHERE EXISTS
-> (SELECT *
-> FROM SC
-> WHERE SC.SNO=STUDENT.SNO AND SC.CNO='2');

+-----+
| SNAME |
+-----+
| 李勇   |
| 刘晨   |
+-----+
2 rows in set (0.00 sec)
```

③多层嵌套EXISTS 与 集合查询

与EXISTS相对应的是NOT EXISTS谓词，若内层查询结果为空，则外层的WHERE子句返回真值，否则返回假值。

SELECT语句的查询结果是元组的集合，所以多个SELECT语句的结果可进行集合操作，包括并操作UNION、交操作INTERSET和差操作EXCEPT。

注意，参加集合操作的各查询结果列数必须相同，对应项的数据类型也必须相同。

多层嵌套EXISTS 查询选修了全部课程的学生的学号和姓名

由于SQL谓词逻辑没有全称量词这个概念。这里需要经过一个变换，该学生选修了所有课程，即不存在任意一个课程该学生没有选修。由此可以用三层选择去写

```
SELECT SNO,SNAME
FROM STUDENT
WHERE NOT EXISTS
(SELECT *
FROM COURSE
WHERE NOT EXISTS
```

```
(SELECT *
FROM SC
WHERE SC.SNO=STUDENT.SNO AND SC.CNO=COURSE.CNO));
```

集合查询 查询先行课为5的课程与学分为4的课程的并集，输出课程名称

```
SELECT CNAME
FROM COURSE
WHERE CPNO='5'
UNION
SELECT CNAME
FROM COURSE
WHERE CCREDIT=4;
```

其实可以看到，第一个的结果是空集。

而第二个则是用UNION连接两个查询，很形象地求并集，将UNION替换为INTERSET或EXCEPT，可以得到交操作和差操作。

运行结果如下：

```
mysql> # 多层嵌套EXISTS 查询选修了全部课程的学生的学号和姓名
mysql> SELECT SNO,SNAME
-> FROM STUDENT
-> WHERE NOT EXISTS
-> (SELECT *
-> FROM COURSE
-> WHERE NOT EXISTS
-> (SELECT *
-> FROM SC
-> WHERE SC.SNO=STUDENT.SNO AND SC.CNO=COURSE.CNO));
Empty set (0.00 sec)
```

```
mysql> SELECT CNAME
-> FROM COURSE
-> WHERE CPNO='5'
-> UNION
-> SELECT CNAME
-> FROM COURSE
-> WHERE CCREDIT=4;
```

```

+-----+
| CNAME      |
+-----+
| 数据库      |
| 信息系统    |
| 数据结构    |
| PASCAL语言  |
+-----+
4 rows in set (0.00 sec)

```

此时，如果类似于“UNION”这样的操作符前后连接的SELECT对象不一致，会触发报错如下：

```

mysql> SELECT *
-> FROM COURSE
-> WHERE CPNO='5'
-> UNION
-> SELECT CNAME
-> FROM COURSE
-> WHERE CCREDIT=4;
ERROR 1222 (21000): The used SELECT statements have a different
number of columns

```

1.4 数据更新

1) 实验内容与要求

熟悉数据库的数据更新操作，能够使用 SQL 语句对数据库进行数据的插入、修改、删除操作。理解和掌握 INSERT,UPDATE,DELETE 语法结构，结合嵌套查询，设计不同形式的插入、修改和删除的语句。

2) 实验重难点

与嵌套查询相结合的插入，修改和删除的 SQL 语句设计，利用一个表的数据来插入，修改和删除另外一个表的数据。

3) 实验基础知识

①插入数据

插入一个元组

```
INSERT INTO <表名> [( <属性列1>, <属性列2>, .....)]
```

省略

#如果与原表一致则可以

```
VALUES ( <常量1>, <常量2>, ..... );
```

插入子查询结果

```
insert into <表名> [( <属性列1>, <属性列2>, .....)]
```

子查询;

#e.g.

插入元组

```
INSERT INTO student(sno,sname,ssex,sdept,sage)
```

出属性名, 但VALUE要保持顺序

#可以不指

```
VALUES('201215128','陈东','男','IS',18);
```

空值

#未给出的列将自动赋

插入子查询结果

```
CREATE TABLE DEPT_AGE
    (SDEPT CHAR(15)
    AVG_AGE SMALLINT);
INSERT DEPT_AGE(SDEPT,AVG_AGE)
SELECT SDEPT,AVG(SAGE)
FROM STUDENT
GROUP BY SDEPT;
```

②修改数据

```
update <表名>
set <列名>=<表达式>,<列名>=<表达式>.....
[where <条件>];
```

修改一个元组的值

```
UPDATE student
SET sage=22
WHERE sno='201215121';
```

修改多个元组的值

```
UPDATE student
SET sage=sage+1;
```

```
# 带子查询的修改语句
UPDATE SC
SET grade=0
WHERE sno IN
    (SELECT sno
     FROM student
     WHERE sdept='CS');
```

③删除数据

```
delete from <表名>
[where <条件>];
```

④空值的处理

#空值与另一个值的算数运算结果为空值
#空值与另一个值的比较运算结果为UNKNOWN，不参与/影响是非判定

4) 实验过程

①插入单个元组

```
INSERT INTO STUDENT VALUES('202108010','甘晴void','男','21','CS');
INSERT INTO COURSE VALUES('8','离散数学','2',4);
INSERT INTO COURSE VALUES('9','提瓦特导论','2',5);
```

②带子查询的插入

```
# 对每一个课程求平均成绩，并存入数据库
CREATE TABLE AVG_GRADE(CNO CHAR(9),AVG_GRADE SMALLINT);

INSERT INTO AVG_GRADE(CNO,AVG_GRADE)
SELECT CNO,AVG(GRADE)
FROM SC
GROUP BY CNO;
```

运行截图如下：

localhost_3306	对象	course @hnutest (localhost_3306) - 表	avg_grade @hnutest (localhost_3306)...
hnutest	开始事务	文本	筛选
表	排序	列	导入
avg_grade	CNO	AVG_GRADE	导出
course	1	92	数据生成
sc	2	88	创建图表
student	3	84	
视图			
函数			
查询			
备份			
homework			
information_schema			
mysql			
performance_schema			
sakila			
sys			
world			

③修改多个元组的值

现在发福利，给每个同学多加上1分

UPDATE SC

SET GRADE=GRADE+1;

学课程3的同学额外加上2分

UPDATE SC

SET GRADE=GRADE+2

WHERE CNO='3';

运行截图如下：

localhost_3306	对象	course @hnutest (localhost_...	avg_grade @hnutest (localh...
hnutest	开始事务	文本	筛选
表	排序	列	导入
avg_grade	Sno	Cno	Grade
course	201215121	1	93
sc	201215121	2	86
student	201215121	3	91
视图	201215122	2	91
函数	201215122	3	83
查询			
备份			
homework			
information_schema			
mysql			
performance_schema			
sakila			
sys			
world			

④删除与批量删除

```
# 现在删除编号为8的课程
DELETE
FROM COURSE
WHERE CNO='8';

# 删除所有（上一步中创建的）AVG_GRADE记录
DELETE
FROM AVG_GRADE;
```

1.5 视图

1) 实验内容与要求

熟悉 SQL 语言有关视图的操作，能够熟练使用 SQL 语句来创建需要的视图，定义数据库外模式，并能使用所创建的视图实现数据管理。

针对给定的数据库模式以及相应的应用需求，创建视图和带有 WITH CHECK OPTION 的视图，并验证 WITH CHECK OPTION 选项的有效性。理解并掌握视图消解执行原理，能够区分可更新视图和不可更新视图。

2) 实验重难点

设计需求并创建视图，区分可更新视图和不可更新视图，创建并验证 WITH CHECK OPTION 选项。

3) 实验基础知识


```
CREATE VIEW <视图名>[(<列名1>,<列名2>,...)]
as <子查询>      #子查询可以嵌套,此时称建立在多个基本表上
[with check option];      #加上该句之后插入值时会自动检查是否符合子查询的要求,
若不满足则拒绝
DROP VIEW <视图名> [CASCADE]

UPDATE SET WHERE
INSERT INTO VALUES
DELETE FROM WHERE
```

4) 实验过程

①创建行列子集视图

省略视图列名, 建立计科专业学生的视图, 该视图由单个表导出, 且保留了主码, 为行列子集视图。

```
CREATE VIEW CS_STUDENT
AS
SELECT SNO,SNAME,SAGE,SDEPT
FROM STUDENT
WHERE SDEPT='CS'
```

运行截图如下:

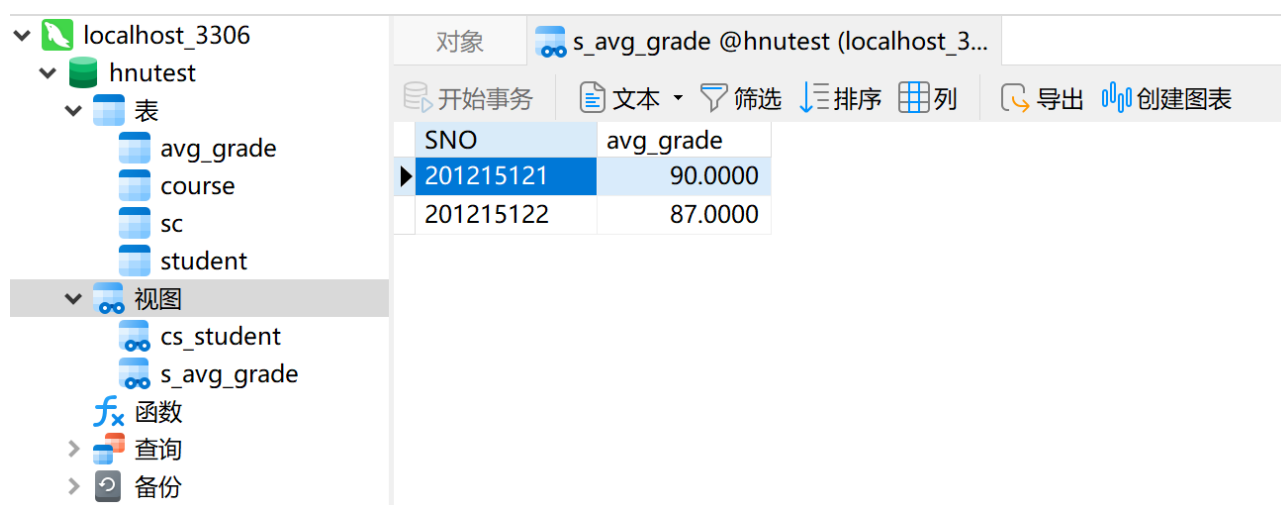
SNO	SNAME	SAGE	SDEPT
201215121	李勇	20	CS
201215122	刘晨	19	CS
202108010	甘晴void	21	CS

②创建分组视图

建立学号和平均成绩的视图，这种视图带有聚集函数和GROUP BY子句查询，称为分组视图。

```
CREATE VIEW S_AVG_GRADE(SNO,avg_grade)
AS
SELECT SNO,AVG(GRADE)
FROM SC
GROUP BY SNO;
```

运行结果如下：



The screenshot shows a database management tool interface. On the left, a tree view displays the database structure for 'localhost_3306' and 'hnutest'. Under '视图' (Views), 's_avg_grade' is selected. The main pane shows the view's definition and its data. The view is named 's_avg_grade' and has two columns: 'SNO' and 'avg_grade'. The data is as follows:

SNO	avg_grade
201215121	90.0000
201215122	87.0000

③创建视图（WITH CHECK OPTION）

使用WITH CHECK OPTION，建立MA专业学生的视图。

```
CREATE VIEW MA_STUDENT
AS
SELECT SNO,SNAME,SAGE,SDEPT
FROM STUDENT
WHERE SDEPT="MA"
WITH CHECK OPTION;
```

运行截图略。通过该视图对计科专业学生的记录分别进行增加，删除，修改。

我试图向该视图插入一条新的数据如下。

```
INSERT INTO MA_STUDENT VALUES('202100001','可莉',21,'MA');
```

★出现问题:

```
mysql> INSERT INTO MA_STUDENT VALUES('202100001','可莉',21,'MA');  
ERROR 1423 (HY000): Field of view 'hnutest.ma_student' underlying  
table doesn't have a default value
```

错误就在于，我们MA_STUDENT所定义在的STUDENT表，它本来有5个属性如下。

```
CREATE TABLE Student( /*学生信息表*/  
    Sno CHAR(9) PRIMARY KEY,  
    Sname CHAR(20) NOT NULL,  
    Ssex CHAR(2) NOT NULL,  
    Sage SMALLINT NOT NULL,  
    Sdept CHAR(20) NOT NULL  
);
```

而我们节选的MA_STUDENT视图，只有其中的4个，但是我们对视图进行修改时，实际上最后还是要对底层的表进行操作，这就导致我们没有节选来的SSEX属性始终是NULL，而NULL是不可被接受的。

我决定更改为对有先行课的课程建立视图

```
CREATE VIEW COURSE_WITHOUT_PREVIOUS  
AS  
SELECT CNO,CNAME,CPNO,CCREDIT  
FROM COURSE  
WHERE CPNO IS NOT NULL  
WITH CHECK OPTION;
```

运行结果如下:

localhost_3306	对象	s_avg_grade @...	student @hnu...	ma_student @...
hnutest	开始事务	文本	筛选	排序
表	列	导出	创建图表	
avg_grade	CNO	CNAME	CPNO	CCREDIT
course	3	信息系统	1	4
sc	9	提瓦特导论	2	5
student	1	数据库	5	4
视图	4	操作系统	6	3
函数	7	PASCAL语言	6	4
查询	5	数据结构	7	4
备份				

接下来尝试一些与OPTION有关的操作

```

/*不满足CPNO IS NOT NULL, CHECK OPTION FAILED, 不能增加*/
mysql> INSERT INTO COURSE_WITHOUT_PREVIOUS VALUES('10','高等元素
论',NULL,5);
ERROR 1369 (HY000): CHECK OPTION failed
'hnutest.course_without_previous'

/*增加*/
INSERT INTO COURSE_WITHOUT_PREVIOUS VALUES('10','高等元素论','9',5);
INSERT INTO COURSE_WITHOUT_PREVIOUS VALUES('11','征服深渊：从入门到满
星','10',5);

```

运行结果如下：

localhost_3306	对象	s_avg_grade @...	student @hnu...	ma_student @...	course @
hnutest	开始事务	文本	筛选	排序	列
表	导出	创建图表			
avg_grade	CNO	CNAME	CPNO	CCREDIT	
course	1	数据库	5	4	
sc	10	高等元素论	9	5	
student	11	征服深渊：从入门到满星	10	5	
视图	3	信息系统	1	4	
course_without_prev	4	操作系统	6	3	
cs_student	5	数据结构	7	4	
ma_student	7	PASCAL语言	6	4	
s_avg_grade	9	提瓦特导论	2	5	
函数					

继续操作：

```

/*不满足CHECK OPTION的修改*/
UPDATE COURSE_WITHOUT_PREVIOUS
SET CPNO=NULL

```

```

WHERE CNAME='高等元素论';

mysql> /*不满足CHECK OPTION的修改*/
mysql> UPDATE COURSE_WITHOUT_PREVIOUS
    -> SET CPNO=NULL
    -> WHERE CNAME='高等元素论';
ERROR 1369 (HY000): CHECK OPTION failed
'hnutest.course_without_previous'

/*删除*/
DELETE
FROM COURSE_WITHOUT_PREVIOUS
WHERE CNAME='征服深渊：用入门到满星';
/*修改*/
UPDATE COURSE_WITHOUT_PREVIOUS
SET CCREDIT=6
WHERE CNAME='高等元素论';

/*这两条运行是成功的*/

```

④可更新视图与不可更新视图

- 一般情况下，只有行列子集视图是可以更新的
- 如果视图由两个表导出，或定义中含有聚集函数和GROUP BY子句，以及定义中含有DISTINCT短语等情况，视图都是不可更新的

例如对①中建立的学号和平均成绩的视图就是不可更新的，因为使用了聚集函数，不可以通过修改其他数据将平均成绩修改。

<div>localhost_3306</div> <div>hnutest</div> <div>表</div> <div>avg_grade</div> <div>course</div> <div>sc</div> <div>student</div> <div>视图</div> <div>course_without_prev</div> <div>cs_student</div> <div>ma_student</div> <div>s_avg_grade</div>	<div>对象</div> <div>s_avg_grade @... student @hnu... ma_student @...</div> <div>打开视图 设计视图 新建视图 删除视图 导出向导</div> <table> <thead> <tr> <th>名</th><th>可以更新</th></tr> </thead> <tbody> <tr> <td>course_without_previous</td><td>是</td></tr> <tr> <td>cs_student</td><td>是</td></tr> <tr> <td>ma_student</td><td>是</td></tr> <tr> <td>s_avg_grade</td><td>否</td></tr> </tbody> </table>	名	可以更新	course_without_previous	是	cs_student	是	ma_student	是	s_avg_grade	否
名	可以更新										
course_without_previous	是										
cs_student	是										
ma_student	是										
s_avg_grade	否										

⑤视图消解

视图消解就是指将对视图的操作转换成对基本表的操作

```
SELECT CNO,CNAME,CCREDIT
FROM COURSE_WITHOUT_PREVIOUS
WHERE CCREDIT=4;

SELECT CNO,CNAME,CCREDIT
FROM COURSE
WHERE CCREDIT=4 AND CPNO IS NOT NULL;
```

运行结果如下：

```
mysql> SELECT CNO,CNAME,CCREDIT
      -> FROM COURSE_WITHOUT_PREVIOUS
      -> WHERE CCREDIT=4;
+----+-----+-----+
| CNO | CNAME      | CCREDIT |
+----+-----+-----+
| 3   | 信息系统   | 4       |
| 1   | 数据库     | 4       |
| 7   | PASCAL语言 | 4       |
| 5   | 数据结构   | 4       |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT CNO,CNAME,CCREDIT
      -> FROM COURSE
      -> WHERE CCREDIT=4 AND CPNO IS NOT NULL;
+----+-----+-----+
| CNO | CNAME      | CCREDIT |
+----+-----+-----+
| 3   | 信息系统   | 4       |
| 1   | 数据库     | 4       |
| 7   | PASCAL语言 | 4       |
| 5   | 数据结构   | 4       |
+----+-----+-----+
4 rows in set (0.00 sec)
```

可见是一致的。

⑥删除视图

```
/*RESTRICT删除*/  
DROP VIEW CS_STUDENT RESTRICT;  
/*CASCADE删除*/  
DROP VIEW CS_STUDENT CASCADE;
```

1.6 索引

1) 实验内容与要求

掌握索引设计原则和技巧，能够创建合适的索引以提高数据库查询、统计分析效率。针对给定的数据库模式和具体应用需求，创建唯一索引、函数索引、复合索引等；修改索引；删除索引。设计相应的 SQL 查询验证索引有效性。学习利用 EXPLAIN 命令分析 SQL 查询是否使用了所创建的索引，并能够分析其原因，执行 SQL 查询并估算索引提高查询效率的百分比。

2) 实验重难点

创建索引，设计查询验证索引的有效性。

3) 实验基础知识

建立索引：加快查询速度

索引是关系数据库管理系统的内部实现技术，属于内模式的范畴。

#索引的建立

```
CREATE [unique][cluster]index <索引名>
```

```
on <表名>(<列名>[<次序>],.....)
```

#索引的修改

```
ALTER index <旧索引名> rename to <新索引名>
```

#索引的删除

```
DROP index <索引名>
```

#e.g.

```
CREATE UNIQUE INDEX Stusno ON Student(Sno);
```

```
CREATE UNIQUE INDEX Cousno ON Cource(Cno);
```

```
CREATE UNIQUE INDEX SCno ON SC(Sno ASC,Cno DESC);
```

4) 实验过程

①生成一个足够大的数据集

由于验证索引效率要求数据集较大，故我们这里可以扩展这个STUDENT表，这里我重新生成一个STUDENT_EXTENT表，以免干扰。

```
CREATE TABLE STUDENT_EXTENT( /*学生信息表*/  
    Sno CHAR(9) PRIMARY KEY,  
    Sname CHAR(20),  
    Ssex CHAR(2),  
    Sage SMALLINT,  
    Sdept CHAR(20)  
);
```

然后使用NAVICAT自带的生成数据行功能进行生成，这个功能很方便，使用截图如下：

100% * 数据生成

localhost_3306
hnutest

预览

表: student_extent 重新生成

Sno	Sname	Ssex	Sage	Sdept
▶ 202108000	姚睿	女	17	CS
202108001	陆子韬	女	24	IS
202108002	廖云熙	男	22	EE
202108003	尹杰宏	女	20	MA
202108004	叶岚	男	19	EE
202108005	熊云熙	女	24	CS
202108006	张秀英	女	23	EE
202108007	卢诗涵	男	18	IS
202108008	冯子异	男	18	EE
202108009	曹云熙	男	17	MA
202108010	顾子异	女	21	MA
202108011	贾诗涵	女	21	AI
202108012	崔子异	男	20	IS
202108013	曹嘉伦	男	21	EE
202108014	廖子异	女	16	IS
202108015	龚秀英	男	25	CS
202108016	吕云熙	男	16	MA

保存配置文件 加载配置文件 选项 上一步 开始

一共生成100,000条数据。

100% - 数据生成

对象: 1
生成的数据: 100,000
插入的数据: 100,000
错误: 0
时间: 00:02.02

[DGR] Data Generation started
[DGR] 1> student_extent: --Transaction - Begin--
[DGR] 1> student_extent: Preparing field
[DGR] 1> student_extent: Start records generation
[DGR] 1> student_extent: Records generated
[DGR] 1> student_extent: --Transaction - Commit--
[DGR] Finished successfully

保存配置文件 加载配置文件 上一步 关闭

②展示索引

```
SHOW INDEX FROM STUDENT_EXTENT;
```

索引展示如下：

```
Windows PowerShell
Your MySQL connection id is 11
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use hnutest
Database changed
mysql> SHOW INDEX FROM STUDENT_EXTENT;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table           | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| student_extent | 0          | PRIMARY | 1            | Sno         | A         | 0           | NULL    | NULL  | NULL | BTREE     |         |               | YES    | NULL       |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> |
```

此时有处于主键上的索引，类型为BTREE

③唯一索引

尝试采用有重复属性值的键SDEPT（所在系）建立唯一索引，报错如下。

```
CREATE UNIQUE INDEX SDEPT_INDEX ON STUDENT_EXTENT(SDEPT);
```

```
mysql> CREATE UNIQUE INDEX SDEPT_INDEX ON STUDENT_EXTENT(SDEPT);
ERROR 1062 (23000): Duplicate entry 'AI' for key
'student_extent.SDEPT_INDEX'
```

可见，唯一索引必须不能有重复属性值，受UNIQUE约束。

④简单索引和复合索引

```
# 建立简单索引
CREATE INDEX SDEPT_INDEX ON STUDENT_EXTENT(SDEPT);

# 建立复合普通索引
CREATE INDEX SEX_AGE_INDEX ON STUDENT_EXTENT(SSEX,SAGE);

mysql> CREATE INDEX SDEPT_INDEX ON STUDENT_EXTENT(SDEPT);
Query OK, 0 rows affected (0.42 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> CREATE INDEX SEX_AGE_INDEX ON STUDENT_EXTENT(SSEX,SAGE);
Query OK, 0 rows affected (0.42 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

可见建立索引需要花费一定时间。

⑤使用索引与不使用索引的对比

```
SELECT SNO FROM STUDENT_EXTENT WHERE SAGE=20;
SELECT SNO FROM STUDENT_EXTENT IGNORE
INDEX(PRIMARY,SEX_AGE_INDEX,SDEPT_INDEX)WHERE SAGE=20;

10969 rows in set (0.01 sec)
10969 rows in set (0.02 sec)
```

可以很明显地发现，慢了一倍多。这里是因为数据量还不够大，如果是百万级，千万级甚至亿万级别的数据量，效果就会差地很大了。

```
EXPLAIN SELECT SNO FROM STUDENT_EXTENT WHERE SAGE=20;
EXPLAIN SELECT SNO FROM STUDENT_EXTENT IGNORE
INDEX(PRIMARY,SEX_AGE_INDEX,SDEPT_INDEX)WHERE SAGE=20;
```

这个运行截图如下：

```
mysql> EXPLAIN SELECT SNO FROM STUDENT_EXTENT WHERE SAGE=20;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | STUDENT_EXTENT | NULL | range | SEX_AGE_INDEX | SEX_AGE_INDEX | 12 | NULL | 9964 | 100.00 | Using where; Using index for skip scan |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT SNO FROM STUDENT_EXTENT IGNORE INDEX(PRIMARY,SEX_AGE_INDEX,SDEPT_INDEX)WHERE SAGE=20;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | STUDENT_EXTENT | NULL | ALL | NULL | NULL | NULL | NULL | 99645 | 10.00 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql>
```

explain 显示了 MYSQL 如何使用索引来处理 select 语句以及连接表，查看是否使用索引，看 type 类型即可，如果是 all，则说明该查询语句遍历了所有行，没有使用索引。

可见在忽略索引的情况下，确实是使用了“ALL”去遍历的。

实验感悟

本次实验，我自己创建数据库，并通过一些较为简单的数据，实践学习了书本上对于数据库的DDL语句的语法、使用SQL 语句创建、修改和删除数据库的语句，SQL 查询语句，SQL嵌套查询和集合查询等各种高级查询，SQL语句对数据库进行数据的插入、修改、删除操作，创建视图和带WITH CHECK OPTION的视图，索引设计原则和技巧，创建合适的索引以提高数据库查询这些知识点。

“纸上得来终觉浅，绝知此事要躬行”

对着书本看几遍都不如做一遍实验来得透彻。

美中不足在于这学期课程数量多，知识与实验压力巨大，所以没能够对于数据库做更多的探索，仅仅局限在书本所限定的范围内。这不得不说是一大遗憾。但是我相信以后还能用上，下次相遇一定能更加亲切。