day24_数据库查询

一 内容回顾（列举前一天重点难点内容）

1.1 教学重点:

```
1.熟练安装数据库
2.熟练掌握基本操作DDL,DML
3.熟练掌握DQL的基本操作
```

二 教学目标

```
1.掌握数据完整性
2.掌握多表查询之合并结果集
3.掌握多表查询之连接查询
4.掌握多表查询之子查询
5.掌握常用函数
6.了解数据库的备份与恢复
```

三 教学导读

接着昨天的内容继续学习数据库的基本操作

四 教学内容

4.5. 数据的完整性(会)

作用：保证用户输入的数据保存到数据库中是正确的。确保数据的完整性 = 在创建表时给表中添加约束

完整性的分类：实体完整性,域完整性,引用完整性.

4.5.1. 实体完整性

- 实体定义

即表中的一行(一条记录)代表一个实体（entity）实体完整性的作用：标识每一行数据不重复。

- 约束类型

主键约束（primary key）

唯一约束(unique)

自动增长列(auto_increment)

4.5.1.1. 主键约束（primary key）

```
注：每个表中要有一个主键。特点：数据唯一，且不能为null

例：第一种添加方式：

CREATE TABLE student(
    id int primary key,
    name varchar(50)
);

第二种添加方式：此种方式优势在于，可以创建联合主键

CREATE TABLE student(
  id int,
    name varchar(50),
    primary key(id)
);
```

```
17
18 CREATE TABLE student(
19     classid int,
20     stuid int,
21     name varchar(50),
22     primary key(classid,stuid)
23 );
24
25 第三种添加方式：单独添加主键
26
27 CREATE TABLE student(
28   id int,
29     name varchar(50)
30 );
31
32 ALTER TABLE student  ADD  PRIMARY  KEY (id);
```

4.5.1.2. 唯一约束(unique)

特点：数据不能重复。(可见null不算重复)

```
1 CREATE TABLE student(
2   Id int primary key,
3     Name varchar(50) unique
4 );
```

4.5.1.3. 自动增长列(auto_increment)

sqlserver数据库 (identity) oracle数据库(sequence)

特点:默认从最大值开始+1操作

给主键添加自动增长的数值，列只能是整数类型

```
CREATE TABLE student(
   Id int primary key auto_increment,
     Name varchar(50)
);

INSERT INTO student(name) values('tom');
```

4.5.2. 域完整性

- 域完整性的作用：

限制此单元格的数据正确，不对照此列的其它单元格比较,域代表当前单元格

- 域完整性约束分类：

数据类型

非空约束（not null）

默认值约束(default)

check约束（mysql不支持）check(sex='男' or sex='女')

- 详情

4.5.2.1. 数据类型:

数值类型、日期类型、字符串类型

4.5.2.2. 非空约束：not null

```
CREATE TABLE student(
    Id int primary key,
    Name varchar(50) not null,
    Sex varchar(10)
);

INSERT INTO student values(1,'tom',null);
```

4.5.2.3. 默认值约束 default

```
CREATE TABLE student(
    Id int primary key,
    Name varchar(50) not null,
    Sex varchar(10) default '男'
);

insert into student values(1,'tom','女');
insert into student values(2,'jerry',default);
```

4.5.3. 引用完整性（参照完整性）

外键约束：FOREIGN KEY

```
例：CREATE TABLE stu(
  sid int primary key,
    name varchar(50) not null,
    sex varchar(10) default '男'
);

create table score(
  id int primary key,
  score int,
  sid int , -- 外键列的数据类型一定要与主键的类型一致
```

```
  foreign key (sid) references stu(sid)
);

-- 撤销外键
先选中外键所在的表,在右下角找到info选项,拉到最后,找到创建score表的信息,我们可以发现主外键关联的标识score_ibfk_1
ALTER TABLE score DROP FOREIGN KEY score_ibfk_1

第二种添加外键方式。

ALTER TABLE score ADD CONSTRAINT fk_stu_score FOREIGN KEY(sid) REFERENCES stu(sid);
```

4.5.4. 表与表之间的关系

- 一对一

例如t_person表和t_card表，即人和身份证。这种情况需要找出主从关系，即谁是主表，谁是从表。人可以没有身份证，但身份证必须要有人才行，所以人是主表，而身份证是从表。设计从表可以有两种方案：在t_card表中添加外键列（相对t_user表），并且给外键添加唯一约束；给t_card表的主键添加外键约束（相对t_user表），即t_card表的主键也是外键。

- 一对多（多对一）

最为常见的就是一对多！一对多和多对一，这是从哪个角度去看得出来的。t_user和t_section的关系，从t_user来看就是一对多，而从t_section的角度来看就是多对一！这种情况都是在多方创建外键！

- 多对多

例如t_stu和t_teacher表，即一个学生可以有多个老师，而一个老师也可以有多个学生。这种情况通常需要创建中间表来处理多对多关系。例如再创建一张表t_stu_tea表，给出两个外键，一个相对t_stu表的外键，另一个相对t_teacher表的外键。

4.6. 多表查询（会)

4.6.1. 多表查询分类

合并结果集 UNION 、UNION ALL

了解连接查询内连接 [INNER] JOIN ON

外连接 OUTER JOIN ON

左外连接 LEFT [OUTER] JOIN

右外连接 RIGHT [OUTER] JOIN

全外连接（MySQL不支持）FULL JOIN

自然连接 NATURAL JOIN

4.6.2. 合并结果集

- 作用

合并结果集就是把两个select语句的查询结果合并到一起!

要求：**被合并的两个结果：列数、列类型必须相同。**

- 创建四张表A,B,C,D

其中表A,B的数据类型完全一直

表D的id是tinyint类型(byte)

表C多一个字段

```
CREATE TABLE A(
    NAME VARCHAR(10),
    id   INT
    )
CREATE TABLE B(
    NAME VARCHAR(10),
    id   INT
    )
CREATE TABLE C(
    NAME VARCHAR(10),
    id   INT,
    sid  INT
    )
CREATE TABLE D(
    NAME VARCHAR(10),
    id   TINYINT
    )
```

- 添加数据

```
INSERT INTO A VALUES('a',20),('aa',40),('aaa',60)
INSERT INTO B VALUES('b',20),('bb',40),('bbb',60)
INSERT INTO C VALUES('c',20,30),('cc',40,50),
('ccc',60,70)
INSERT INTO D VALUES('d',20),('dd',40),('ddd',60)
```

- 执行语句

```
-- 注意:在进行合并结果集时,两个表的字段个数和类型必须一致
-- 合并结果集时不是对原表进行合并,是对查出来的虚拟表进行合并

-- 这里由于从A中查出的虚拟表和从C中查出的虚拟表字段个数不一样,所以报错
SELECT * FROM A UNION SELECT * FROM C
-- 这里由于从A中查出的虚拟表和从C中查出的虚拟表字段个数,类型一致,可以实现.
SELECT * FROM A UNION SELECT NAME,id FROM C
-- 当类型有高低等级,会转成高等级类型
SELECT * FROM D UNION SELECT * FROM A
```

- 再添加一些数据

```
INSERT INTO A VALUES('f',80)
INSERT INTO B VALUES('f',80)
```

- 再次执行语句

```
-- 对于union会自动进行去重
两个表中都有相同的数据'f',80.    会进行去重操作
SELECT * FROM A UNION SELECT * FROM B
-- 对于union all 不会去重
两个表中都有相同的数据'f',80.    不会进行去重操作
SELECT * FROM A UNION ALL SELECT * FROM B
```

4.6.3. 连接查询

- 创建表并添加数据(有主外键关系)

```
-- 创建表
CREATE TABLE IF NOT EXISTS student(
  id INT PRIMARY KEY,
```

```
  NAME VARCHAR(10)
)

CREATE TABLE IF NOT EXISTS scores(
  coruseId INT PRIMARY KEY,
  score INT,
  id INT ,
  FOREIGN KEY(id) REFERENCES student(id)
)

INSERT INTO student(id,NAME) VALUES(1,'zhangsan'),(2,'lisi')
INSERT INTO scores(coruseId,score,id) VALUES(1, 20,1),(2,40,2),(3,50,2)

```

- 直接连接查询两个表

```
SELECT * FROM student stu,scores sco
```

```
我们发现通过stu连接sco，查询出的结果就是stu*sco,即(2*3=6条记录)
```

结果出现了多个表的乘积,我们称为笛卡尔积.这里面有错误的数据.

假设集合A={a,b}，集合B={0,1,2}，则两个集合的笛卡尔积为{(a,0),(a,1),(a,2),(b,0),(b,1),(b,2)}。可以扩展到多个集合的情况。那么多表查询产生这样的结果并不是我们想要的，那么怎么去除重复的，不想要的记录呢，当然是通过条件过滤。通常要查询的多个表之间都存在关联关系，那么就通过关联关系去除笛卡尔积。

stu一共2行记录，sco表一共3行记录，那么连接后查询出的结果是6行记录。你只是想在查询stu表的同时，把每个学员的成绩显示出来，那么就需要使用主外键来去除无用信息了。

- 使用主外键关系做为条件来去除无用信息

注意:多表查询,如果一个字段的名字是唯一的,不需要使用.语法,可以直接使用.

但是如果多表中同一个字段出现了多次,要使用.语法

这里直接使用的是默认的查询方式--相当于内连接

```
SELECT * FROM student stu,scores sco WHERE stu.id=sco.id
```

4.6.3.1. 内连接

上面的连接语句就是内连接，但它不是SQL标准中的查询方式，可以理解为方言！SQL标准的内连接为：

```
SELECT *
FROM student stu
INNER JOIN scores sco
ON stu.id=sco.id;
```

测试:

取消主外键

```
ALTER TABLE scores DROP FOREIGN KEY scores_ibfk_1;
```

再次查询,得到的结果说明,没有主外键也可以,但是一定要在两张表中,进行条件控制的字段有对应的关系

```
INSERT INTO student(id,NAME) VALUES(3,'wangwu')

SELECT * FROM student stu INNER JOIN scores sco ON
stu.id=sco.id
```

总结:内连接的特点：只能查出两个表都有的id记录,如果一个记录只有主表有,副表没有,使用内连接,无法查出这条记录

4.6.3.2. 外连接（左外连接、右外连接）

外连接的特点：查询出的结果存在不满足条件的可能。

左外连接

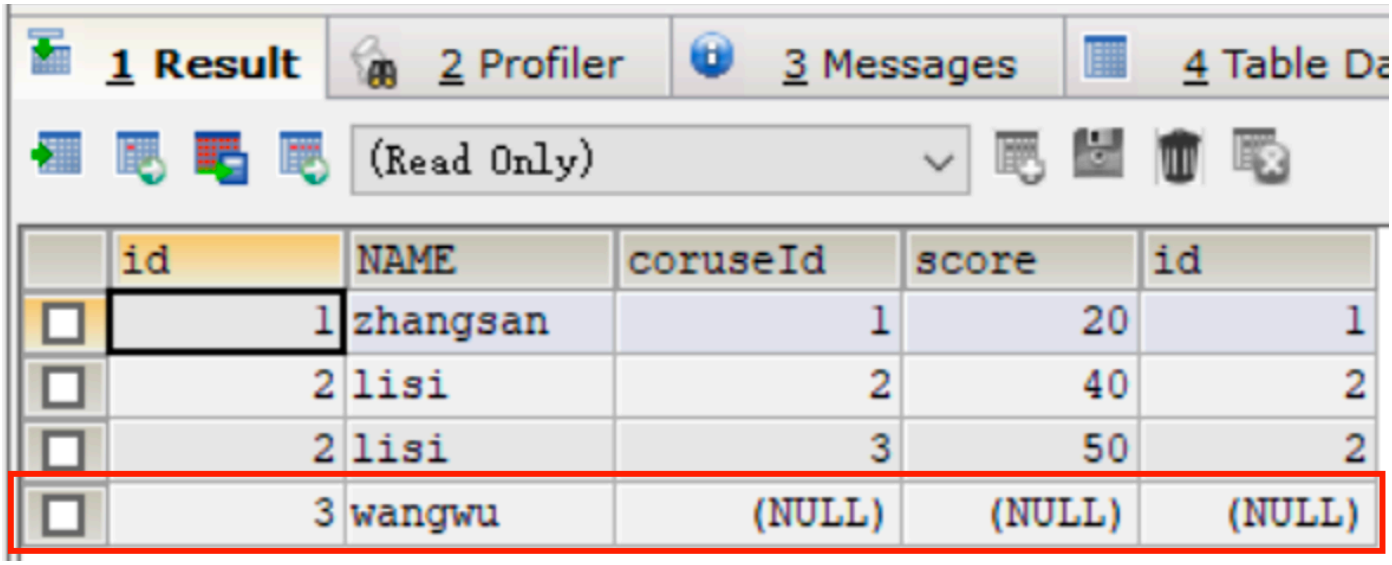
```
SELECT * FROM emp e
LEFT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

左连接是先查询出左表（即以左表为主），然后查询右表，右表中满足条件的显示出来，不满足条件的显示NULL。

工作原理:以左表为主,左表的信息会全部查出来,右表中只能查出与左表中id相同的记录,其他的查不出.只有左表有数据,右表没有对应数据的显示null

左连接:写法 select from 表一 left outer(可省略) join 表二 on 表一.id=表二.id

```
添加一条数据
INSERT INTO scores(coruseId,score,id) VALUES(4, 20,4)
查询
SELECT * FROM student stu  LEFT OUTER JOIN scores sco ON stu.id=sco.id
```



id	NAME	coruseId	score	id
1	zhangsan	1	20	1
2	lisi	2	40	2
2	lisi	3	50	2
3	wangwu	(NULL)	(NULL)	(NULL)

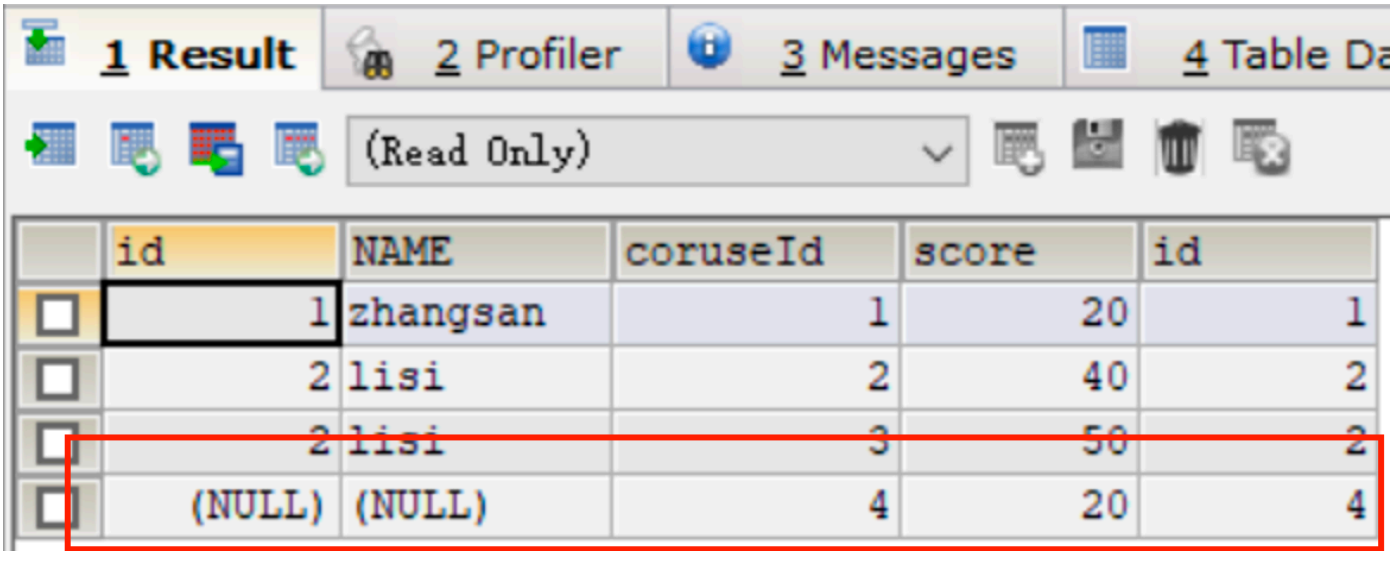
从图中可以看出,最后一行,左表有数据:3,wangwu,右表没有对应数据,显示null

右外连接

右连接:写法 select from 表一 right outer(可省略) join 表二 on 表一.id=表二.id

工作原理:以右表为主,右表的信息会全部查出来,左表中只能查出与右表中id相同的记录,其他的查不出,只有右表有数据,左表没有对应数据的显示null

```
1 SELECT * FROM student stu  RIGHT OUTER JOIN scores sco
  ON stu.id=sco.id
```



id	NAME	coruseId	score	id
1	zhangsan	1	20	1
2	lisi	2	40	2
2	lisi	3	50	2
(NULL)	(NULL)	4	20	4

从图中可以看出,最后一行,右表有数据:4,20,4,左表没有对应数据,显示null

连接查询心得:

连接不限于两张表，连接查询也可以是三张、四张，甚至N张表的连接查询。通常连接查询不可能需要整个笛卡尔积，而只是需要其中一部分，那么这时就需要使用条件来去除不需要的记录。这个条件大多数情况下都是使用主外键关系去除。两张表的连接查询一定有一个主外键关系，三张表的连接查询就一定有两个主外键关系，所以在大家不是很熟悉连接查询时，首先要学会去除无用笛卡尔积，那么就是用主外键关系作为条件来处理。如果两张表的查询，那么至少有一个主外键条件，三张表连接至少有两个主外键条件。

4.6.3.3. 自然连接--了解

大家也都知道，连接查询会产生无用笛卡尔积，我们通常使用主外键关系等式来去除它。而自然连接无需你去给出主外键等式，它会自动找到这一等式:

两张连接的表中名称和类型完全一致的列作为条件，例如stu和sco表都存在id列，并且类型一致，所以会被自然连接找到!

当然自然连接还有其他的查找条件的方式，但其他方式都可能存在问题!

```
-- 注意:1.要有主外键,而且主外键的的名字要相同.如果没有主外键,他默认会找字段相同的所有字段进行比较,如果没有字段相同的字段报错
-- 2.自然连接时,会将相同的字段合并
-- 3.如果只写natural 默认当作内连接
SELECT * FROM student stu NATURAL  JOIN scores sco
SELECT * FROM student stu NATURAL LEFT  JOIN scores sco
SELECT * FROM student stu NATURAL RIGHT  JOIN scores sco
```

4.6.4. 子查询（非常重要）

- 定义

一个select语句中包含另一个完整的select语句。

子查询就是嵌套查询，即SELECT中包含SELECT，如果一条语句中存在两个，或两个以上SELECT，那么就是子查询语句了。

- 子查询出现的位置：

where后，作为被查询条件的一部分；

from后，作表；

注意:当子查询出现在where后作为条件时，还可以使用如下关键字：(很少用)any all

- 子查询结果集的形式：

单行单列（用于条件）

单行多列（用于条件）

多行单列（用于条件）

多行多列（用于表）

- 课上练习

1.工资高于JONES的员工。

分析：查询条件：工资>JONES工资，其中JONES工资需要一条子查询。

第一步：查询JONES的工资

```
SELECT sal FROM emp WHERE ename='JONES'
```

第二步：查询高于JONES工资的员工

```
SELECT * FROM emp WHERE sal > (SELECT sal FROM emp WHERE ename='JONES')
```

结果：

```
SELECT * FROM emp WHERE sal > (SELECT sal FROM emp WHERE ename='JONES')
```

2.查询与SCOTT同一个部门的员工。

子查询作为条件 子查询形式为单行单列

4.6.5. 自连接

- 自己连接自己，起别名
- 课上练习

1.求7369员工编号、姓名、经理编号和经理姓名

```
SELECT e1.empno , e1.ename,e1.mgr,e2.ename
FROM emp e1, emp e2
WHERE e1.mgr = e2.empno AND e1.empno = 7369;
```

2.求各个部门薪水最高的员工所有信息

```
(select max(sal) maxsal,deptno from emp
group by deptno) a
where e.deptno = a.deptnoand e.sal =a.maxsal
```

4.7. MySQL中的函数

4.7.1. 常用系统函数(会)

4.7.1.1. 日期函数

ADDTIME (date2 ,time_interval)	**将time_interval加到date2**
CURRENT_DATE ( )	当前日期
CURRENT_TIME ( )	当前时间
CURRENT_TIMESTAMP ( )	当前时间戳
DATE (datetime)	返回datetime的日期部分
DATE_ADD (date2 , INTERVAL d_value d_type)	在date2中加上日期或时间
DATE_SUB (date2 , INTERVAL d_value d_type)	在date2上减去一个时间
DATEDIFF (date1 ,date2)	两个日期差
NOW ()	当前时间
YEAR|Month|Day(datetime)	年月日

4.7.1.2. 字符串函数

CHARSET(str)	返回字串字符集
CONCAT (string2 [...])	连接字串
INSTR (string ,substring)	返回substring在string中出现的位置,没有返0
UCASE (string2)	转换成大写
LCASE (string2)	转换成小写
LEFT (string2 ,length)	从string2中的左边起取length个字符
LENGTH (string)	string长度
REPLACE (str ,search_str ,replace_str)	在str中用replace_str替换search_str
STRCMP (string1 ,string2)	逐字符比较两字串大小,
SUBSTRING (str , position [,length])	从str的position开始,取length个字符
LTRIM (string2) RTRIM (string2) trim	去除前端空格或后端空格

4.7.1.3. 常规函数

ABS (number2)	绝对值
BIN (decimal_number)	十进制转二进制
CEILING (number2)	向上取整
CONV(number2,from_base,to_base)	进制转换
FLOOR (number2)	向下取整
FORMAT (number,decimal_places)	保留小数位数
HEX (DecimalNumber)	转十六进制
LEAST (number , number2 [...])	求最小值
MOD (numerator ,denominator)	求余
RAND([seed])	RAND([seed])

4.7.1.4. 操作示例

```
-- mysql常用的函数
SELECT ADDTIME('2007-12-30 21:50:50', '1:1:1')
SELECT ADDTIME(NOW(),'1:1:1')
SELECT CURRENT_DATE()
SELECT CURRENT_TIME()
SELECT CURRENT_TIMESTAMP()
SELECT DATE(NOW())-- 获取当前时间的日期部分

SELECT DATE_ADD('2016-6-6' ,INTERVAL  -1 DAY)
SELECT DATEDIFF('2016-6-8','2016-6-10')  -- 前面的时间-后面的时间的差值
SELECT NOW();

SELECT YEAR|MONTH|DAY(DATETIME )-- 不能这样做
SELECT YEAR(NOW())-- 单独获取年
SELECT MONTH(NOW())-- 单独获取月
```

```sql
SELECT DAY(NOW())-- 单独获取日

-- 对字符串的操作
SELECT CHARSET('hello') -- 当前的编码格式
SELECT CONCAT('he','lo') -- 合并字符串
SELECT *,CONCAT(ename,job) FROM emp;
SELECT INSTR('hello','e')-- 当在原字符串中找不到子字符串时,会返回0
SELECT LEFT('hello',2)-- -- 不能这样做
SELECT REPLACE('hello','h','wo')
SELECT STRCMP('hello','heloo') -- 比较两个字符串    前面大返回1    后面大返回-1   相等返回0
SELECT LTRIM('        hello') -- 去除左边的空格
SELECT RTRIM('hello      ') -- 去除右边的空格
SELECT MOD(34,0); -- 返回null
```

4.7.2. 自定义函数(了解)

4.7.2.1. 什么是函数

mysql中的函数与存储过程类似，都是一组SQL集；

4.7.2.2. 与存储过程的区别

函数可以return值，存储过程不能直接return，但是有输出参数可以输出多个返回值；

函数可以嵌入到sql语句中使用，而存储过程不能；

函数一般用于实现较简单的有针对性的功能（如求绝对值、返回当前时间等），存储过程用于实现复杂的功能（如复杂的业务逻辑功能）；

函数的关键字是function 存储过程是:procedure

4.7.2.3. 自定义函数

```
DROP FUNCTION IF EXISTS func_compare;     -- 丢掉已经存在的函数
DELIMITER ;; -- 自定义分隔符,这里定义的分隔符是;;,定义好后,只有遇到;;才会结束

-- 这里是在连接数据库    +    创建函数

CREATE DEFINER='root'@'localhost' FUNCTION func_compare(a INT) RETURNS VARCHAR(200) CHARSET utf8
BEGIN    -- 函数开始

 -- Routine body goes here...
 String temp = "";
 if(a>=10){temp = "大于等于10";}else{}
-- 这里写的是if语句

IF a >= 10 THEN
  RETURN '大于等于10';
  ELSE
      RETURN '小于10';
  END IF;
END -- 函数结束
;;
DELIMITER ; -- 重新将分隔符定义成;

使用函数
select func_compare(4)
结果:小于10
```

4.8. mysql中的存储过程(了解)

4.8.1. 为什么要使用存储过程

mysql从1.5开始支持procedure

存储过程简单来说，就是为以后的使用而保存的一条或多条MySQL语句的集合。可将其视为批件，虽然它们的作用不仅限于批处理。在我看来， 存储过程就是有业务逻辑和流程的集合， 可以在存储过程中创建表，更新数据， 删除等等。

通过把处理封装在容易使用的单元中，简化复杂的操作(正如前面例子所述)。 由于不要求反复建立一系列处理步骤，这保证了数据的完整性。如果所有开发人员和应用程序都使用同一(试验和测试)存储过程，则所使用的代码都是相同的。这一点的延伸就是防止错误。需要执行的步骤越多，出错的可能性就越大。防止错误保证了数据的一致性。 简化对变动的管理。如果表名、列名或业务逻辑(或别的内容)有变化，只需要更改存储过程的代码。使用它的人员甚至不需要知道这些变化。

4.8.2. 过程说明

```
DROP PROCEDURE IF EXISTS math1; -- 丢掉存储过程math1
DELIMITER ;; -- 重新设置分隔符为;;
CREATE PROCEDURE math1 -- 创建存储过程 math1
(IN a INT,IN b INT)  -- 传入参数 a,b     IN的意思是传入参数 OUT的意思是传出参数     INOUT是既可以传入又可以传出

BEGIN

    declare c int default 0; -- 定义局部变量,只能在存储过程中使用

```

```
    set  c = 3;

    SET @var1 = 1; -- 设置用户变量,全局使用
    SET @var2 = 2;
    SELECT @SUM:=(a + b) AS SUM, @dif:=(a - b) AS dif;
END;
;; -- 结束
DELIMITER ;-- 重新设置分隔符为;

CALL math1(1,2)  -- 调用存储过程
SELECT @var1; -- 调用用户变量

```

4.8.3. 参数说明

4.8.3.1. 参数in的使用

(代表输入，意思说你的参数要传到存过过程的过程里面去)

为了避免存储过程中分号(";")结束语句，我们使用分隔符告诉mysql解释器,该段命令是否已经结束了。

案例功能：求1-n的和

```
delimiter $
create procedure p1(in n int)
begin
declare total int default 0;
declare num int default 0;
while num < n do
set num:=num+1;
set total:=total+num;
end while;
select total;
end$
call p1(10)$
```

4.8.3.2. 参数out的使用

(代表往外输出)

这里还要注意一点的就是我们的输出参数一定要设置相应类型的初始，否则不管你怎么计算得出的结果都为NULL值

案例功能：求1-n的和

```
create procedure p2(in n int,out total int)
begin
declare num int default 0;
set total:=0;
while num < n do
set num:=num+1;
set total:=total+num;
end while;
end$总结in、out区别：
in:表示输入一个值，你需要一个值，我给你一个值
out:你往外输出一个值，你输出的那个值我就拿一个变量来接收你给我输出的那个值
```

4.8.3.3. 参数inout的使用

(既能输入一个值又能传出来一个值)

功能：传一个年龄，自动让年龄增长10岁

```
create procedure p3(inout age int)
begin
set age:=age+10;
end$
注意：调用的时候，我这里需要和大家声明一下，inout型的参数值既是输入类型又是输出类型，你给它一个值，值不是变量，不是变量那out的时候它怎么赋给这个值是不是?
因此我们需要先设置一个变量并初始化这个值，调用的时候直接传这个变量即可。
set @currentAge=8$
call p3(@currentAge)$
select @currentAge$
```

4.8.4. 变量说明

mysql的变量分为两种：系统变量和用户变量,但是在实际使用中，还会有局部变量、会话变量等分法

4.8.4.1. 局部变量

局部变量一般用在sql语句块中，比如存储过程的begin/end。其作用域仅限于该语句块，在该语句块执行完毕后，局部变量就消失了。

局部变量一般用declare来声明，可以使用default来说明默认值。

4.8.4.2. 用户变量

用户变量的作用域要比局部变量要广。用户变量可以作用于当前整个连接，但是当当前连接断开后，其所定义的用户变量都会消失。

select @变量名

对用户变量赋值有两种方式，一种是直接用"="号，另一种是用":="号。其区别在于使用set命令对用户变量进行赋值时，两种方式都可以使用；当使用select语句对用户变量进行赋值时，只能使用":="方式，因为在select语句中，"="号被看作是比较操作符。

4.8.4.3. 会话变量

服务器为每个连接的客户端维护一系列会话变量。在客户端连接时，使用相应全局变量的当前值对客户端的会话变量进行初始化。设置会话变量不需要特殊权限，但客户端只能更改自己的会话变量，而不能更改其它客户端的会话变量。会话变量的作用域与用户变量一样，仅限于当前连接。当当前连接断开后，其设置的所有会话变量均失效。

```
设置会话变量有如下三种方式：

set session var_name = value;

set @@session.var_name = value;

set var_name = value;

查看一个会话变量也有如下三种方式：

select @@var_name;

select @@session.var_name;

```

```
show session variables like "%var%";

mysql> show session variables;
```

4.8.4.4. 全局变量

全局变量影响服务器整体操作。当服务器启动时，它将所有全局变量初始化为默认值。这些默认值可以在选项文件中或在命令行中指定的选项进行更改。要想更改全局变量，必须具有SUPER权限。全局变量作用于server的整个生命周期，但是不能跨重启。即重启后所有设置的全局变量均失效。要想让全局变量重启后继续生效，需要更改相应的配置文件。

```
要设置一个全局变量，有如下两种方式：

set global var_name = value; //注意：此处的global不能省略。根据手册，set命令设置变量时若不指定GLOBAL、SESSION或者LOCAL，默认使用SESSION

set @@global.var_name = value; //同上

要想查看一个全局变量，有如下两种方式：

select @@global.var_name;

show global variables like "%var%";

mysql> show global variables;
```

4.9. MySQL数据库的备份与恢复

4.9.1. 通过脚本实现数据的备份与恢复

4.9.1.1. 生成SQL脚本 导出数据

在CMD下 命令不能加;

在控制台使用mysqldump命令可以用来生成指定数据库的脚本文本，但要注意，脚本文本中只包含数据库的内容，而不会存在创建数据库的语句！所以在恢复数据时，还需要自己手动创建一个数据库之后再去恢复数据。

```
mysqldump –u用户名 –p密码 --databases 数据库名 > 生成的脚本文件路径

例如:执行下面的终端命令
C:\Users\bihai\Desktop>mysqldump -uroot -p123456 --databases dbtest1 > dbtest1.sql
//下面的是警告,不用管
mysqldump: [Warning] Using a password on the command line interface can be insecure.
```

现在可以在桌面找到dbtest1.sql文件了！

注意，mysqldump命令是在Windows控制台下执行，无需登录mysql！！！

4.9.1.2. 执行SQL脚本 恢复数据

前提：必须先创建数据库名

通过下面的方式来执行脚本文件：

```
在终端执行下面的命令

mysql -uroot -p123456 新创建的库 < c:\dbtest1.sql

mysql –u用户名 –p密码 数据库<要执行脚本文件路径
```

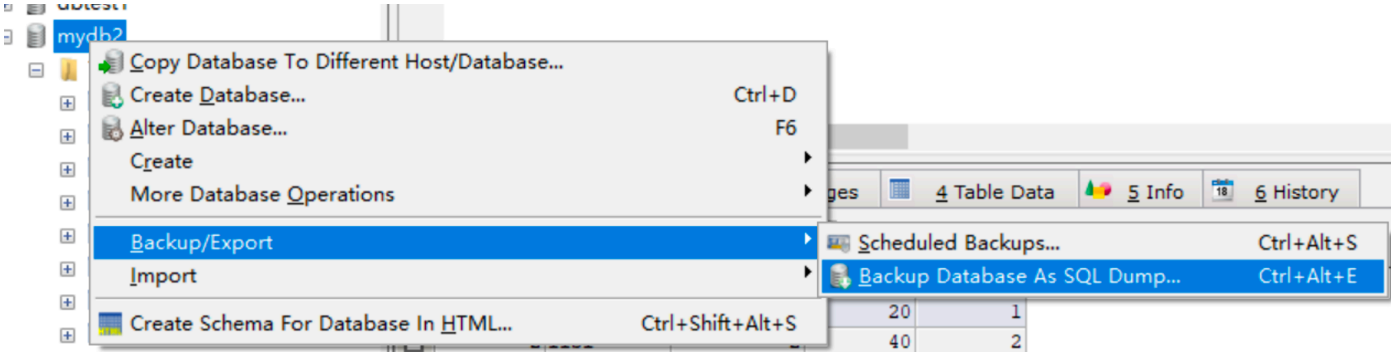
这种方式无需登录mysql!

4.9.2. 通过可视化工具实现数据的备份与恢复

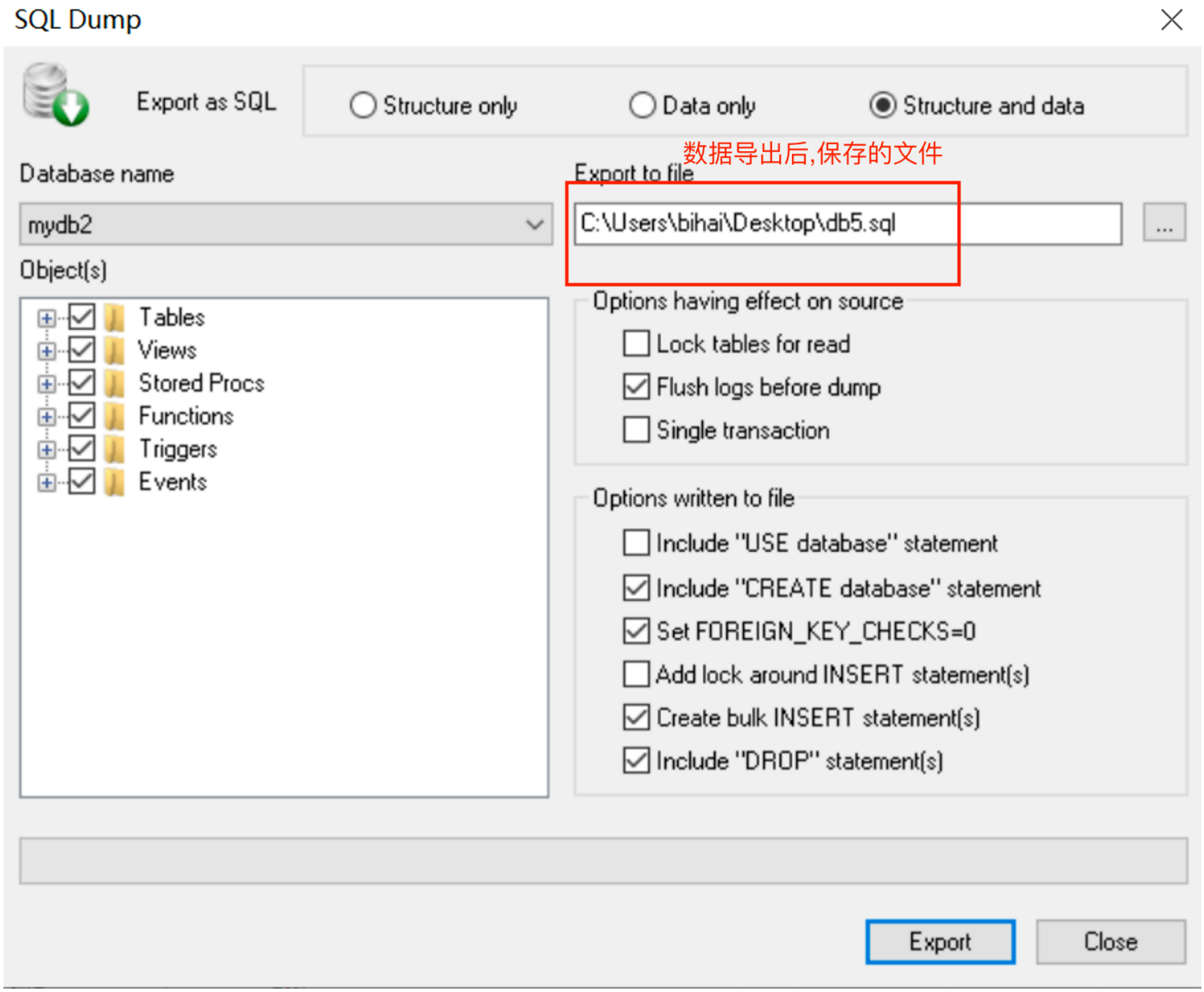
4.9.2.1. 使用sqlyog实现数据的备份

1.选中要备份的数据库

2.右键,选中Backup Database As SQL Dump



3.在弹出的窗口中按照我的提示选择,最后点击按钮export,当下面的进度条变成绿色,说明导出成功



4.9.2.2. 使用sqlyog实现数据的恢复

1.必须要新建一个数据库

2.选中数据库,右键选中Restore From SQL Dump,导入即可

**