## DQL语言

### 数据查询语言说明

/\*

#数据查询语言

\*/

### 进阶一

use employees;

SELECT \* FROM employees;

SELECT 100\*100;

SELECT 100;

SELECT 'JJONE';

SELECT VERSION();

#起别名

/\*

1.便于理解和区别;

\*/

#方式一 AS

SELECT 100%98 AS 结果;

SELECT last\_name AS 名 FROM employees;

#方式二 加空格

SELECT salary 薪水 FROM employees;

#如果别名有空格 可以用双引号""

SELECT salary "薪 水" FROM employees;

#去重

SELECT DISTINCT department\_id FROM employees;

#+号的作用

/\*

java中+号

1.运算符

2.连接符

mysql的+号只能是 运算符

SELECT 100+90 都为数值 做加法运算

SELECT '123'+90 字符型转换为数值 如果成功 在进行加法运算

SELECT 'jion'+90 字符型转换为数值 如果失败 字符转换为 0 在进行加法运算

SELECT 'NULL'+90 如果一方为null结果为null

\*/

#案例 连接两个字段

SELECT last\_name+first\_name AS 姓名

FROM employees;

SELECT 'jion'+90;

#如果需要进行连接用concat()函数

#案例

SELECT concat('A','B','C') AS 结果;

SELECT concat(last\_name,first\_name) AS 名 FROM employees;

SELECT CONCAT(last\_name,',',first\_name) AS 名 FROM employees;

#判断是否为空 ifnull(字段,返回值);

SELECT IFNULL(commission\_pct,0) AS

奖金率 ,commission\_pct FROM employees;

### 进阶二

进阶二

#条件查询

/\*

语法

select

查询列表

from

表名

where

筛选条件;

分类

1. 按条件表达式

条件运算符:

大于> 小于< 等于 = 不等!= <> >= <=

2.按逻辑表达式筛选

&& || ！

and or not

&& 和and ; true,true为true 否 false

|| 或 or ; 有true 为true 否 false

！ 或 not ; 取反

3.模糊查询

like

between and

is null

\*/

#案例一：查询工质>12000员工的信息

SELECT

\*

FROM

employees

WHERE

salary>12000;

#案例2：部门号不等于90的员工名和编号

SELECT

last\_name,

department\_id

FROM

employees

WHERE

department\_id<>90;

#按逻辑表达式筛选

#案例1：工资在10000到20000之间员工名，工资，奖金

SELECT

last\_name,

salary,

commission\_pct

FROM

employees

WHERE

salary>=10000 AND salary<=20000;

#模糊查询

/\*

关键字

like

特点

1.一般和通配符搭配使用

%任意多字符，包含0

\_任意单个字符

如果值里面有'\_'用转义符'\\_'

或者定义转义符

last\_name like '\_$\_%' ESPCAE '$';

betweeen

in

is NULL |is not NULL

\*/

#案例1：查询员工名中包含字符a的员工信##息

SELECT \* FROM employees WHERE

last\_name LIKE '%a%';

#案例2 查询员工第三个为n第五个为l;

SELECT \* FROM employees WHERE

last\_name LIKE '\_\_n\_l%';

#案例2 查询员工第二个为\_的员工 默认'\'为转义符 这里定义'$' 为转义符;

SELECT \* FROM employees WHERE

last\_name LIKE '\_$\_%' ESCAPE '$' ;

#BETWEEN AND

#案例3：查询编号在100到200之间的员工

SELECT \* FROM employees WHERE

employee\_id >=100 AND employee\_id <=200;

#等价与

#注：条件顺序不能错

SELECT \* FROM employees WHERE

employee\_id BETWEEN 100 AND 120;

# in 不支持通配符

#案例1：查询员工的工种编号是AD\_PRES AD\_VP 中的一个

SELECT \* FROM employees

WHERE job\_id IN('AD\_PRES','AD\_VP');

#类似

SELECT \* FROM employees

WHERE job\_id ='AD\_PRES' OR job\_id ='AD\_VP';

# is NULL

#案例1：没有奖金的员工

SELECT last\_name,commission\_pct FROM

employees

WHERE commission\_pct IS NULL;

#安全等于 <=>

#案例1：没有奖金的员工

SELECT last\_name,commission\_pct FROM

employees

WHERE commission\_pct <=>NULL;

SELECT last\_name FROM

employees

WHERE salary <=>12000;

#is null vs <=>

#is null 仅仅判断null可读性高

#<=> 判单null和数值 可读性低

### 进阶三

#排序查询

/\*

SELECT

字段，表达式 别名 函数 3

FROM

表名 1

WHERE 2

条件

ORDER BY 字段或表达式（升序）或别名或函数 asc |(降序)desc(默认升序) 4

ORDER BY 字段（升序）asc |(降序)desc ;

(前为主后为辅)（一般放在最后面）limit字句除外

\*/

#案例1：按工资排序

SELECT \* FROM employees ORDER BY salary ASC（按字段）

#案例2：按年薪排序 显示员工信息 和年薪（没有年薪的字段可以计算）

SELECT \* ,salary\*12\*(1+IFNULL(commission\_pct,0)) 年薪 FROM employees

ORDER BY salary\*12\*(1+IFNULL(commission\_pct,0)) ASC;（按表达式）

SELECT \* ,salary\*12\*(1+IFNULL(commission\_pct,0)) 年薪 FROM employees

ORDER BY 年薪 ASC; （按别名）

SELECT LENGTH(last\_name) 字节长度,last\_name FROM employees

ORDER BY LENGTH(last\_name) DESC; （按函数）

### 进阶四

#常见函数

/\*

功能：类似于java的方法，将一组逻辑语句封装在方法体中，对外暴露方法名

好处：1.隐藏了实现的字节 2.提高代码的重用性

调用: select 函数名（实参列表）from 表名

特点：

1.叫什么(函数名)

2.干什么（函数功能）

分类：

1.单行函数 作用处理

如cocat，length,iffull等

2.分组函数 作用统计 又称统计函数,聚合函数

\*/

#一字符函数

#1.length():获取数字的字节个数

SELECT LENGTH('jack');

SELECT LENGTH('杰克jack');

#2.concat():拼接函数

SELECT CONCAT(last\_name,'\_',first\_name) 姓名 FROM employees;

#3.upper() lower() 大小写转换

SELECT UPPER('jonh');

SELECT LOWER('JONH');

#示例

SELECT CONCAT(UPPER(last\_name),LOWER(first\_name)) 姓名 FROM employees;

#4.substr() ,substring()截取字符串 截取的是字符的长度

SELECT SUBSTR('mysql是个关系数据库',7) 结果;

SELECT SUBSTR('mysql是个关系数据库',8,12) 结果;

#示例

SELECT CONCAT(UPPER(last\_name),LOWER(first\_name)) 姓名 FROM employees;

#5.instr() 返回字串第一次出现的索引，如果找不到返回0

SELECT INSTR('mysql是个关系数据库','关系数据库') AS OU;

#6.trim() 去掉前后空格

SELECT LENGTH(TRIM(' mysql ')) as 输出;

也可以自定义去掉指定的字符

SELECT LENGTH(TRIM('a' FROM 'aaamyaasqlaaaaa')) as 输出;

#7.lpad ()用指定字符实现左填充

SELECT LPAD('mysql',10,'\*') AS 输出;

#8.rpad() 用指定字符实现右填充

SELECT RPAD('mysql',10,'\*') AS 输出;

#9.replace ()替换

SELECT REPLACE('mysql是个关系数据库','mysql','oracle') AS 输出;

#二、数学函数

#1.round() 四舍五入

SELECT ROUND(-1.55);

SELECT ROUND(1.567,2);

#2.ceil() 向上取整，返回>=该参数的最小整数

SELECT CEIL(-1.02);

#3.floor() 向下取整，返回<=该参数的最小整数

SELECT FLOOR(-1.02);

#4.truncate 截断

SELECT TRUNCATE(-1.02,1);

#5.mod取余

SELECT MOD(7,-2);

SELECT MOD(-7,-2);

#三、日期函数

#1.now() 返回当前系统的日期+时间

SELECT NOW();

#2.curdate() 返回当前日期，不包含时间

SELECT CURDATE();

#3.curtime() 返回当前时间，不包含日期

SELECT CURTIME();

#4.可以获取指定的部分，年月日时分秒

SELECT YEAR(NOW()) 年;

SELECT YEAR('1998-1-1') 年;

SELECT YEAR(hiredate) 年 FROM employees;

#年月日时分秒同理

#5.str\_to\_date() :常用将字符(时间) 转换成指定格式

#年月日的位置可以改变但 格式也要对应转换

SELECT STR\_TO\_DATE('1992-3-2','%Y-%c-%d') as 输出;

SELECT STR\_TO\_DATE('3-2-1992','%c-%d-%Y') as 输出;

#6.date\_format() 将日期转换成字符

SELECT DATE\_FORMAT(NOW(),'%Y年%m月%d日') AS 输出;

#四、其他函数 系统函数

SELECT VERSION();

SELECT DATABASE();

SELECT USER();

#五、流控制函数

#1.if函数： if else 的效果

SELECT IF(10>5,'大','小');

SELECT last\_name ,commission\_pct ,IF(commission\_pct IS NULL,'没奖金','有奖金') 备注

FROM employees;

#2.case 函数的使用一 ：switch case 的效果

/\*

java

switch(){

case 常量：语句; break;

... ...

default：语句; break;

}

mysql

case 要判断的字段或表达式

when 常量1 then 要显示的值或语句;

when 常量2 then 要显示的值或语句;

... ...

else 要显示的值或语句;

end

\*/

#3.case 函数的使用二: 类似 多重if

/\*

java

if(){

}else if(){

}else if(){

}else{

}

mysql

case （该处无值）

when 条件 then 要显示的值或语句;

when 条件 then 要显示的值或语句;

... ...

else 要显示的值或语句;

end

\*/

#案例 查询工资的情况

/\*

如果工资>20000 ,显示A级别

如果工资>15000 ,显示B级别

如果工资>10000 ,显示C级别

\*/

SELECT last\_name, salary ,

CASE

WHEN salary>20000 THEN 'A'

WHEN salary>15000 THEN 'B'

WHEN salary>20000 THEN 'C'

ELSE 'D'

END AS 级别

FROM employees ;

#分组函数

/\*

功能：用作统计使用，又称聚合函数或统计函数或组函数

分类

sum 求和’avg 平均值、max 最大值、min 最小值、count 计数函数

\*/

#1.简单的使用

SELECT sum(salary) FROM employees;

SELECT AVG(salary) FROM employees;

SELECT MAX(salary) FROM employees;

SELECT MIN(salary) FROM employees;

SELECT COUNT(salary) FROM employees;

SELECT sum(salary) 和, AVG(salary) 平均, MIN(salary) 最小,

MAX(salary) 最大, COUNT(salary) 统计 FROM employees;

#2.参数支持那些类型

/\*

sum avg 数值

max min 数值 字符

count 都行 计算不为空的个数

\*/

#3.是否忽略null

/\*

分组都函数忽略null

\*/

#4.和distinct搭配使用

#去重

SELECT SUM(DISTINCT salary),SUM(salary) FROM employees;

#5.count()的详细介绍

SELECT COUNT(salary) FROM employees;

#统计行数 准确常用

SELECT COUNT(\*) FROM employees;

#可以加一个常量值来统计

SELECT COUNT(1) FROM employees;

#效率

#MYISAM存储引擎下 count(\*)的效率高

#INNODB存储引擎下 count(\*)和count(1)差不多 比count(字段) 高

#一般用count(\*)

#6.和分组函数一同查询的字段有限制

#统计函数只有一个值 其他的是一列 无意义

#一般同时查询的是 GROUP BY 后面的字段

### 进阶五

#分组查询

/\*

SELECT 分组函数，列(要求出现在group by后面)

FROM 表

WHERE 条件

GROUP BY 分组的列

ORDER BY 字句

注意：

查询列表必须特殊，要求是分组函数和group by 后面出现的字段

\*/

#引入：查询每个部门的平均工资

#所有部门

SELECT AVG(salary) FROM employees;

#group by 把表中的数据分成若干组

SELECT AVG(salary) ,department\_id FROM employees GROUP BY department\_id;

#1.添加筛选条件

#案例1：查询邮箱中包含a字符的，每个部门的平均工资

SELECT AVG(salary) , department\_id

FROM employees

WHERE email LIKE '%a%'

GROUP BY department\_id;

#案例2.查询查询有奖金的每个领导手下与员工的最高工资

SELECT MAX(salary) ,manager\_id

FROM employees

WHERE commission\_pct IS NOT NULL

GROUP BY manager\_id;

#添加复杂的筛选条件

#案例1.查询那个部门的员工个数>2

/\*

1.查询每个部门的员工个数

SELECT COUNT(\*),department\_id

FROM employees

GROUP BY department\_id;

2.根据1.的结果进行筛选,查询那个部门的员工个数>2

SELECT COUNT(\*),department\_id

FROM employees

GROUP BY department\_id;

HAVING COUNT(\*)>2

用HAVING关键字进行连接

\*/

SELECT COUNT(\*),department\_id

FROM employees

GROUP BY department\_id;

HAVING COUNT(\*)>2

#案例2:查询每个工种有奖金的最高工资>12000的工种编号和最高工资

/\*

\*/

#1.查询每个工种有奖金的最高工资

SELECT MAX(salary), job\_id

FROM employees

WHERE commission\_pct IS NOT NULL

GROUP BY job\_id;

#2.根据1.结果继续筛选，最高工资>12000

SELECT MAX(salary), job\_id

FROM employees

WHERE commission\_pct IS NOT NULL

GROUP BY job\_id

HAVING MAX(salary)>12000;

#案例3：查询领导编号>102的每个领导手下的最低工资>5000的领导编号是哪个,以及其最低工资

#1.查询领导编号的每个领导手下的最低工资

SELECT MIN(salary), manager\_id

FROM employees

GROUP BY manager\_id;

#2.添加筛选条件 manager\_id>102 表里存在放在里面

SELECT MIN(salary), manager\_id

FROM employees

WHERE manager\_id>102

GROUP BY manager\_id;

#3.添加筛选条件 MIN(salary)>5000 表里不存在放在外面

SELECT MIN(salary), manager\_id

FROM employees

WHERE manager\_id>102

GROUP BY manager\_id

HAVING MIN(salary)>5000;

#1.按表达式或函数分组

#案例：按员工姓名的长度分组，查询每一组的员工个数，筛选员工个数>5的有那些

#1.查询每个长度的员工个数

SELECT COUNT(\*) , LENGTH(last\_name)

FROM employees

GROUP BY LENGTH(last\_name);

#2.添加筛选条件 (支持别名)

SELECT COUNT(\*) , LENGTH(last\_name)

FROM employees

GROUP BY LENGTH(last\_name)

HAVING COUNT(\*) >5;

#按多个字段分组

#案例：查询每个部门每个工种的员工的平均工资

SELECT AVG(salary) ,department\_id,job\_id

FROM employees

GROUP BY department\_id,job\_id;

#GROUP BY 顺序可以交换

SELECT AVG(salary), department\_id,job\_id

FROM employees

GROUP BY job\_id,department\_id;

#3.添加排序

#案例：查询每个部门每个工种的员工的平均工资，并且按照平均工资高低显示

SELECT AVG(salary), department\_id,job\_id

FROM employees

GROUP BY job\_id,department\_id

ORDER BY AVG(salary) DESC;

#总结

/\*

1.分组查询中的筛选条件分为两类

数据源 位置 关键字

分组前筛选 原始表 GROUP BY子句的前面 WHERE

分组后筛选 分组后的结果集 GROUP BY子句的后面 HAVING(having)

#1.分组函数做条件肯定是放在having句子中

#2.能用分组前筛选的，就优先考虑分组前筛选

2.GROUP BY 子句支持单个字段分组，多字段分组(多个字段之间用逗号隔开没有顺序要求),

表达式和函数

3.也可添加排序(排序放在整个分组查询的最后)

\*/

### 进阶六

/\*

含义：又称多表查询，当要查询的字段来自于多个表或涉及多个表

SELECT NAME,boyName FROM boys ,beauty;

笛卡尔乘积现象：表1 有m行，表2有n行, 结果=m\*n行

发生原因：没有有效的连接条件

如何避免：添加添加有效的条件

分类：

按年代分类

sq192标准

sq199标准（推荐）

按功能分类：

内连接：

等值连接

非等值连接

自链接

外连接:

左外连接

右外连接

全外连接

交叉连接

#一、sq1 92语法

\*/

SELECT \* FROM beauty;

SELECT \* FROM boys;

SELECT NAME,boyName FROM boys ,beauty;

SELECT NAME,boyName FROM boys ,beauty

WHERE beauty.boyfriend\_id = boys.id ;

#案例2：查询员工名和对应的部门名

SELECT last\_name ,department\_name

FROM employees ,departments

WHERE employees.department\_id=departments.department\_id;

#2.为表起别名

/\*

1.提高语句的简洁度

2.区分多个重名的字段

注：如果为表起了表名，则查询的字段就不能用原来的表明限定

\*/

#案例3：查询员工名、工种号、工种名

#有歧义的字段用表明指明

SELECT last\_name ,e.job\_id,job\_title

FROM employees e,jobs j #表的位置可以调换

WHERE e.job\_id=j.job\_id;

#3.可以加筛选条件

#案例：查询有奖金的员工名、部门名

SELECT e.last\_name, d.department\_name,e.commission\_pct

FROM employees e, departments d

WHERE e.commission\_pct IS NOT NULL AND e.department\_id=d.department\_id;

#案例：查询城市名中第二个字母为o的部门名和城市名

SELECT department\_name , city

FROM departments d , locations l

WHERE d.location\_id=L.location\_id AND L.city LIKE '\_O%' ;

#4.添加分组？

#案例1：查询每个城市部门个数

SELECT COUNT(\*) 个数,city

FROM departments d,locations l

WHERE d.location\_id = l.location\_id

GROUP BY city;

#案例2：查询出有奖金的每个部门名和部门的领导编号和该部门的最低工资

SELECT department\_name , d.manager\_id, MIN(salary) 最低工资

FROM departments d , employees e

WHERE

e.commission\_pct IS NOT NULL

AND d.manager\_id IS NOT NULL

AND e.department\_id = e.department\_id

GROUP BY department\_name, d.department\_id

ORDER BY d.manager\_id ;

#5.添加排序

#案例2：查询出有奖金的每个部门名和部门的领导编号和该部门的最低工资按部门的领导编号

#排名

SELECT department\_name , d.manager\_id, MIN(salary) 最低工资

FROM departments d , employees e

WHERE

e.commission\_pct IS NOT NULL

AND d.manager\_id IS NOT NULL

AND e.department\_id = e.department\_id

GROUP BY department\_name, d.department\_id

ORDER BY d.manager\_id ;

#6.可以实现三表连接

#案例查询员工名，部门名和所在城市

SELECT last\_name ,department\_name ,city

FROM employees e ,departments d ,locations l;

WHERE e.department\_id = d.department\_id

AND d.location\_id = l.location\_id;

#非等值连接

#

#2.非等值连接

#案例1：员工的工资和工资级别

SELECT \* FROM job\_grades;

SELECT salary 工资 , grade\_level 级别

FROM employees e,job\_grades j

WHERE e.salary BETWEEN j.lowest\_sal AND j.highest\_sal;

#3.自连接(同一张表)

#案例：查询 员工名和上级的名

SELECT e.employee\_id ,e.last\_name ,m.employee\_id,m.last\_name

FROM employees e ,employees m;

WHERE e.manager\_id = m.manager\_id; #2.非等值连接

#案例1：员工的工资和工资级别

SELECT \* FROM job\_grades;

SELECT salary 工资 , grade\_level 级别

FROM employees e,job\_grades j

WHERE e.salary BETWEEN j.lowest\_sal AND j.highest\_sal;

#3.自连接(同一张表)

#案例：查询 员工名和上级的名

SELECT e.employee\_id ,e.last\_name ,m.employee\_id,m.last\_name

FROM employees e ,employees m;

WHERE e.manager\_id = m.manager\_id;

#二、sql 99语法

/\*

语法：

SELECT 查询列表

FROM 表一 别名 【连接类型】

join 表二 别名 on 连接条件

WHERE 筛选条件

GROUP BY 分组

HAVING 筛选条件

ORDER BY 排序

筛选条件与连接条件分开提高可读性

分类：

1.内连接 inner

2.外连接

左外: left 【outer可省略】

右外：right 【outer可省略】

全外：full 【outer可省略】

交叉连接

\*/

#1.内连接

/\*

SELECT 查询列表

FROM 表1 别名

inner jion 表2 别名

等值

非等值

自连接

特点：

1.添加排序、分组、筛选

2.inner 可以省略

3.筛选条件放在where后面，连接条件放在on的后面提高分离性，便于阅读

4.inner join 和sql92语法的等值连接效果一样，都是查询多表的交集部分

\*/

#99语法

#1、等值连接 内连接

#案例1.查询员工名、部门名

SELECT last\_name , department\_name

FROM employees e

INNER JOIN departments d

ON e.department\_id = d.department\_id;

#案例2.查询名字中包含e的员工名和工种名（添加筛选条件）

SELECT last\_name ,job\_title

FROM employees e

INNER JOIN jobs j

ON e.job\_id = j.job\_id

WHERE e.last\_name LIKE '%e%';

#案例3.查询部门个数>3的城市名和部门名

SELECT l.city 城市,COUNT(\*) 部门个数

FROM departments d

INNER JOIN locations l

ON d.location\_id = l.location\_id

GROUP BY l.city

HAVING COUNT(\*)>3;

#案例4：查询那个部门的员工个数>3的部门名和员工个数，并按个数降序（添加排序）

SELECT d.department\_name 部门名,COUNT(\*) 员工个数

FROM departments d

INNER JOIN employees e

ON d.department\_id = e.department\_id

GROUP BY d.department\_name

HAVING COUNT(\*)>3

ORDER BY COUNT(\*) DESC;

#案例5：查询员工名、部门名、工种名、并按部门名降序

#注：连接条件需要顺序

SELECT last\_name ,department\_name ,job\_title

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

INNER JOIN jobs j ON e.job\_id = j.job\_id

ORDER BY job\_title;

#二、非等值连接

#查询员工的工资级别

SELECT salary ,grade\_level

FROM employees e

INNER JOIN job\_grades j

ON e.salary BETWEEN j.lowest\_sal AND j.highest\_sal;

#查询工资级别的个数>20的个数，并且按工资级别降序

SELECT COUNT(\*) ,grade\_level

FROM employees e

INNER JOIN job\_grades j

ON e.salary BETWEEN j.lowest\_sal AND j.highest\_sal

GROUP BY grade\_level

HAVING COUNT(\*)>20

ORDER BY grade\_level DESC;

#三、自连接

#查询员工的名字、上级的名字

SELECT e.last\_name , m.last\_name

FROM employees e

INNER JOIN employees m

ON e.manager\_id = m.manager\_id

WHERE e.last\_name LIKE '%k%';

#二、外连接

/\*

应用：1.一个表有 一个表没有的记录

特点：

1.外连接的查询结果为主表中的所有记录

如果从表中有和它匹配的,则显示匹配的值

如果从表中没有和它匹配的,则显示null

外连接查询结果=内连接结果+主表有而从表没有的记录

2.左外连接，left jion 左边的是主表

右外连接，right jion 右边的是主表

3.左外和右外交换两个表的顺序，可以实现同样的效果

4.全外连接=内连接的结果+表1中有但表2没有+表2中有但表1没有

\*/

#引入 ：查询没有男朋有的女神名

SELECT \* FROM beauty;

SELECT \* FROM boys;

#查询男朋友不在男神表的 女神名

#要查询的信息主要来自哪里哪个是主表

#左外连接

SELECT b.name,bo.\*

FROM beauty b

LEFT JOIN boys bo

ON b.boyfriend\_id = bo.id;

SELECT b.name

FROM beauty b

LEFT JOIN boys bo

ON b.boyfriend\_id = bo.id

WHERE bo.id IS NULL;

#一般选从表的主键

#右外连接

SELECT b.name

FROM boys bo

RIGHT JOIN beauty b

ON b.boyfriend\_id = bo.id

WHERE bo.id IS NULL;

#案例：那个部门没有员工

#左外

SELECT d.\* , e.employee\_id

FROM departments d

LEFT JOIN employees e

ON d.department\_id = e.department\_id

WHERE e.employee\_id IS NULL;

#右外

SELECT d.\* , e.employee\_id

FROM employees e

RIGHT JOIN departments d

ON d.department\_id = e.department\_id

WHERE e.employee\_id IS NULL;

#全外连接

USE girls;

SELECT b.\*,bo.\*

FROM beauty b

FULL JOIN boys bo

ON b.boyfirebds\_id=bo.id;

#交叉连接

#相当与笛卡尔积

SELECT b.\*,bo.\*

FROM beauty b

CROSS JOIN boys bo;

#sql92 和sql99 pk

#功能：sql99支持较多可读性高

### 进阶七

/\*

含义：

出现在其他语句中的select语句中，称为子查询或内查询

外部的查询语句，称为主查询或外查询

分类：

按子查询出现的位置：

SELECT后面：

仅仅自持标量子查询

FROM后面：

支持表子查询

WHERE或having后面（重点）

标量子查询 (单行) \*

列子查询 (多行) \*

行子查询

EXISTS后面(相关子查询)

表子查询

按结果集的行列数不同

标量子查询(结果集只有一行一列)

列子查询(结果集只有一行多列)

行子查询(结果集有一行多列)

表子查询(结果集一般为多行多列)

#一、where或having后面

#1.标量子查询（单行子查询）

#2.列子查询（多行子查询）

#3.行子查询(多行多列)

#特点：

#1.子查询放在小括号内

#2.子查询一般放在条件的右侧

#3.标量子查询，一般搭配着单行操作符使用

> < >= <= = <>

#列子查询 一般搭配着多行操作符使用

in、any/some、all

\*/

#1.标量子查询

#案例1：谁的工资比Abel高？

#1.查询Abel的工资

SELECT salary

FROM employees

WHERE last\_name ='Abel';

#2.查询员工信息，满足salary>1.的结果

SELECT \*

FROM employees

WHERE salary > (

SELECT salary

FROM employees

WHERE last\_name ='Abel'

);

#案例2：返回job\_id与141员工相同，salary比143员工多的员工 姓名，job\_id 和工资

#1.查询141号员工的 job\_id

SELECT job\_id

FROM employees

WHERE employee\_id = 141

#2.查询143号的 salary

SELECT salary

FROM employees

WHERE employee\_id = 143

#3.查询员工的姓名，job\_id和工资 ，要求job\_id=1.并且 salary>2.

SELECT last\_name,job\_id,salary

FROM employees

WHERE job\_id =(SELECT job\_id

FROM employees

WHERE employee\_id = 141

) AND salary>(SELECT salary

FROM employees

WHERE employee\_id = 143)

#案例3：返回公司工资最少的员工的last\_name,job\_id和salary

#1.查询公司最低工资

SELECT MIN(salary)

FROM employees

#2.查询last\_name,job\_id和salary，要求c=1.

SELECT last\_name,job\_id,salary

FROM employees

WHERE salary=(

SELECT MIN(salary)

FROM employees

)

#案例4：查询最低工资大于50号部门最低工资的部门id和其最低工资

#1.查询50号部门的最低工资

SELECT MIN(salary)

FROM employees

WHERE department\_id = 50;

#查询每个部门的最低工资

SELECT MIN(salary),department\_id

FROM employees

GROUP BY department\_id;

#3.筛选2，满足min(salary)>1.

SELECT MIN(salary),department\_id

FROM employees

GROUP BY department\_id

HAVING MIN(salary)>(

SELECT MIN(salary)

FROM employees

WHERE department\_id = 50

);

#非法使用标量子查询

#子查询的值不能为空且为单行

SELECT MIN(salary),department\_id

FROM employees

GROUP BY department\_id

HAVING MIN(salary) > (

SELECT salary

FROM employees

WHERE department\_id = 250

);

#多行子查询(列子查询)

# 多行比较符 in/not in(等于列表中任意一个)(in默认为 in())

# any|some(与子查询返回值里面的任意一个值比较 与in类似弹药写比较符)

# all(与子查询返回值里面的所有值比较)

#案例1：返回location\_id是1400或1700的部门中的所有员工姓名

#1.查询location\_id是1400或1700的部门编号

SELECT department\_id

FROM departments

WHERE location\_id IN(1400,1700)

#2.查询员工姓名，要求部门号是1.列表中的某一个

SELECT last\_name

FROM employees

WHERE department\_id IN(

SELECT department\_id

FROM departments

WHERE location\_id IN(1400,1700)

)

#案例2：返回其它工种中比job\_id为‘IT\_PROG’工种任一

#工资低的员工的员工号、姓名、job\_id 以及salary

#1.查询job\_id为‘IT\_PROG’部门任一工资

SELECT DISTINCT salary

FROM employees

WHERE job\_id = 'IT\_PROG'

#2.查询员工号、姓名、job\_id 以及salary，salary<(1.)的任意一个

SELECT employee\_id ,last\_name ,job\_id ,salary

FROM employees

WHERE salary < ANY(

SELECT DISTINCT salary

FROM employees

WHERE job\_id = 'IT\_PROG'

) AND job\_id <> 'IT\_PROG';

#或

SELECT employee\_id ,last\_name ,job\_id ,salary

FROM employees

WHERE salary < ANY(

SELECT MAX(salary)

FROM employees

WHERE job\_id = 'IT\_PROG'

) AND job\_id <> 'IT\_PROG';

#案例3：返回其它部门中比job\_id为‘IT\_PROG’部门所有工资都低的员工的员工号、姓名、job\_id #以及salary

SELECT employee\_id ,last\_name ,job\_id ,salary

FROM employees

WHERE salary < ALL(

SELECT DISTINCT salary

FROM employees

WHERE job\_id = 'IT\_PROG'

) AND job\_id <> 'IT\_PROG';

#或

SELECT employee\_id ,last\_name ,job\_id ,salary

FROM employees

WHERE salary < ANY(

SELECT MIN(salary)

FROM employees

WHERE job\_id = 'IT\_PROG'

) AND job\_id <> 'IT\_PROG';

#3、行子查询（结果集一行多列或多行多列）

#案例：查询员工编号最小并且工资最高的员工信息

#(原先)

#1.查询最小员工编号

SELECT MIN(employee\_id)

FROM employees

#2.查询工资最高

SELECT MAX(salary)

FROM employees

#符合1. 2. 员工的信息

SELECT \*

FROM employees

WHERE employee\_id = (

SELECT MIN(employee\_id)

FROM employees

)

AND salary =(

SELECT MAX(salary)

FROM employees

)

#(行子查询)

#比较符相同

SELECT \*

FROM employees

WHERE (employee\_id,salary)=(

SELECT MIN(employee\_id),MAX(salary)

FROM employees

)

#二、select 后面

/\*

仅仅支持标量子查询

\*/

#案例：查询每个部门的员工个数

SELECT d.\* ,(

SELECT COUNT(\*)

FROM employees e

WHERE d.department\_id = e.department\_id

)

FROM departments d;

#案例2：查询员工号=102的部门名

#一行一列

SELECT (

SELECT department\_name,e.department\_id

FROM departments d

INNER JOIN employees e

ON d.department\_id=e.department\_id

WHERE e.employee\_id=102

) 部门名;(false)

SELECT (

SELECT department\_name

FROM departments d

INNER JOIN employees e

ON d.department\_id=e.department\_id

WHERE e.employee\_id=102

) 部门名;(ture)

三、from 后面

/\*

将子查询结果充当一张表，要求必须起别名

\*/

#案例：查询每个部门的平均工资的工资等级

#1.查询每个部门的平均工资

SELECT AVG(salary) ,department\_id

FROM employees

GROUP BY department\_id

#2.

SELECT sa.\*, jo.grade\_level

FROM (SELECT AVG(salary) sal,department\_id

FROM employees

GROUP BY department\_id) sa

INNER JOIN job\_grades jo

ON sa.sal BETWEEN lowest\_sal AND highest\_sal

#四、exists后面（相关子查询）

/\*

语法：

exists(完整的查询语句)

结果：

1或0

\*/

#EXISTS是否存在

SELECT EXISTS(SELECT employee\_id FROM employees)

案例1：查询有员工的部门名

#in

SELECT department\_name

FROM departments d

WHERE d.`department\_id` IN(

SELECT department\_id

FROM employees

)

#exists

SELECT department\_name

FROM departments d

WHERE EXISTS(

SELECT \*

FROM employees e

WHERE d.`department\_id`=e.`department\_id`

);

#案例2：查询没有女朋友的男神信息

#in

SELECT bo.\*

FROM boys bo

WHERE bo.id NOT IN(

SELECT boyfriend\_id

FROM beauty

)

#exists

SELECT bo.\*

FROM boys bo

WHERE NOT EXISTS(

SELECT boyfriend\_id

FROM beauty b

WHERE bo.`id`=b.`boyfriend\_id`

);

### 进阶八

#分页查询 ★

/\*

应用场景：当要显示的数据，一页显示不全，需要分页提交sql请求

语法：

select 查询列表

from 表

【join type join 表2

on 连接条件

where 筛选条件

group by 分组字段

having 分组后的筛选

order by 排序的字段】

limit 【offset,】size;

offset要显示条目的起始索引（起始索引从0开始）(不写默认0)

size 要显示的条目个数

特点：

①limit语句放在查询语句的最后

②公式

要显示的页数 page，每页的条目数size

select 查询列表

from 表

limit (page-1)\*size,size;

size=10

page

1 0

2 10

3 20

\*/

#案例1：查询前五条员工信息

SELECT \* FROM employees LIMIT 5;

SELECT \* FROM employees LIMIT 0 , 5;

#案例2：查询第11条——第25条

SELECT \* FROM employees LIMIT 10,15;

#案例3：有奖金的员工信息，并且工资较高的前10名显示出来

SELECT

\*

FROM

employees

WHERE commission\_pct IS NOT NULL

ORDER BY salary DESC

LIMIT 10 ;

### 进阶九

#联合查询

/\*

union 联合 合并 ：将多条查询语句的结果合并成一个结果

应用场景：

要查询的结果来自于多个表，且多个表没有直接的连接关系，但查询的信息一致时

特点：★

1、要求多条查询语句的查询列数是一致的！

2、要求多条查询语句的查询的每一列的类型和顺序最好一致

3、union关键字默认去重，如果使用union all 可以包含重复项

\*/

#引入的案例：查询部门编号>90或邮箱包含a的员工信息

SELECT \* FROM employees WHERE email LIKE '%a%' OR department\_id>90;

SELECT \* FROM employees WHERE email LIKE '%a%'

UNION

SELECT \* FROM employees WHERE department\_id>90;

#案例：查询中国用户中男性的信息以及外国用户中年男性的用户信息

SELECT id,cname FROM t\_ca WHERE csex='男'

UNION ALL

SELECT t\_id,tname FROM t\_ua WHERE tGender='male';

## DML语言

### 数据库操作语言说明

/\*

数据操作语言

插入：insert

修改：update

修改：delete

\*/

### 一、插入语句

/\*

语法 insert into 表名(列名,...) values(值1,...);

\*/

SELECT \* FROM beauty;

#1.插入的值的类型要与列的类型一致或兼容

INSERT INTO beauty(id,NAME,sex,borndate,phone,photo,boyfriend\_id)

VALUES(13,'唐艺昕','女','1990-4-23','1898888888',NULL,2);

#2.不可以为null的列必须插入值。可以为null的列如何插入值？

#方式一：

INSERT INTO beauty(id,NAME,sex,borndate,phone,photo,boyfriend\_id)

VALUES(13,'唐艺昕','女','1990-4-23','1898888888',NULL,2);

#方式二：

INSERT INTO beauty(id,NAME,sex,phone)

VALUES(15,'娜扎','女','1388888888');

#3.列的顺序是否可以调换

INSERT INTO beauty(NAME,sex,id,phone)

VALUES('蒋欣','女',16,'110');

#4.列数和值的个数必须一致

INSERT INTO beauty(NAME,sex,id,phone)

VALUES('关晓彤','女',17,'110');

#5.可以省略列名，默认所有列，而且列的顺序和表中列的顺序一致

INSERT INTO beauty

VALUES(18,'张飞','男',NULL,'119',NULL,NULL);

#方式二：

/\*

语法：

insert into 表名

set 列名=值,列名=值,...

\*/

INSERT INTO beauty

SET id=19,NAME='刘涛',phone='138143';

#1、方式一支持插入多行,方式二不支持

INSERT INTO beauty

VALUES(23,'唐艺昕1','女','1990-4-23','1898888888',NULL,2)

,(24,'唐艺昕2','女','1990-4-23','1898888888',NULL,2)

,(25,'唐艺昕3','女','1990-4-23','1898888888',NULL,2);

#2、方式一支持子查询，方式二不支持

INSERT INTO beauty(id,NAME,phone)

SELECT 26,'宋茜','1898888888';

INSERT INTO beauty(id,NAME,phone)

SELECT id,boyname,'1234567'

FROM boys WHERE id<3;

二、

### 二、修改语句

/\*

1.修改单表记录

语法：

update 表名 1

set 列=新值，列=新值... ... 3

where 筛选条件; 2

2.修改多表记录【补充】

sql92

语法：

update 表名1 别名 ,表名1 别名

set 列 = 值 ，，，，

where 连接条件

and 筛选条件

sql99

语法：

update 表名1 别名

inner |left|right| join 表2 别名

on 连接条件

set 列=值

where 筛选条件

\*/

#1.修改单表记录

#案例1：修改beauty 表中姓唐的女神电话为1388888888

UPDATE beauty

SET phone = '13888888'

WHERE `name` LIKE '唐%';

#案例2：

#2.修改多表的记录

#案例1：修改张无忌女朋友的手机号为114

UPDATE boys bo

INNER JOIN beauty b ON bo.id = b.boyfriend\_id

SET b.phone = '11111111'

WHERE bo.boyName='张无忌';

#案例2：修改没有男朋友的女神的男朋有编号都为2号

UPDATE boys bo

RIGHT JOIN beauty b ON bo.id = b.boyfriend\_id

SET b.boyfriend\_id = 2

WHERE bo.id IS NULL;

### 三、删除语句

/\*

方式一：DELETE

语法：

单表删除【\*】

delete from 表名 where 筛选条件

多表删除【补充】

sql92语法：

delete 表的别名

from 表1 别名,表2 别名,

where 连接条件

and 筛选条件；

sql99

delete 表的别名,表的别名,

from 表1 别名

inner | left | right join 表2 别名 on 连接条件

where 筛选条件

limit 条目

方式二：truncate

语法：

truncate table 表名

\*/

#方式一：DELETE

#1.单表删除【\*】

#案例1： 删除手机号以9结尾的女神信息

DELETE FROM beauty

WHERE phone LIKE '%8';

SELECT \* FROM beauty;

#多表的删除

#案例：删除张无忌的女神信息

DELETE b

FROM beauty b

INNER JOIN boys bo ON b.boyfriend\_id = bo.id

WHERE bo.boyName ='张无忌'

#方式二：trancate 语句

#案例：将魅力值>100男神信息删除

#{z只能清空数据 ，不能添加条件}

TRUNCATE TABLE boys WHERE userCP >100;（x）

#{清空数据}

TRUNCATE TABLE boys;

/\*

1.delete 可以加where 条件，truncate不能加

2.truncate删除，效率高一丢丢

3.假如要删除的表中有自增长列，

如果用delete删除后，再插入数据，自增长列的值从断点开始，

而truncate删除后，再插入数据，自增长列的值从1开始。

4.truncate删除没有返回值，delete删除有返回值

5.truncate删除不能回滚，delete删除可以回滚.

\*/

SELECT \* FROM boys;

DELETE FROM boys;

TRUNCATE TABLE boys;

INSERT INTO boys (boyname,usercp)

VALUES('张飞',100),('刘备',100),('关云长',100);

## DDL语言

### 数据库定义语言说明

/\*

数据库定义语言

库和表的管理

一、库的管理

创建、修改、删除

二、表的管理

创建、修改、删除

创建：create

修改：update

删除：drop

\*/

### 一、库的管理

#1.库的创建

/\*

#:语法

create database (if not exists) 库名;

\*/

#案例1：创建books

CREATE DATABASE books;

#添加判断

CREATE DATABASE IF NOT EXISTS books;

#案例2：库的修改RENAME

RENAME DATABASE books TO 新库名;(已经废弃,不安全)

#如需要修改可以在根目录的data文件修改

#更改字符集 ALTER

ALTER DATABASE books CHARACTER SET utf8;

#案例3:库的删除 DROP

#添加判断

DROP DATABASE IF EXISTS books ;

### 二、表的管理

#1.表的创建

/\*

create table 表名(

列名 列的类型 【长度 、 约束】,

列名 列的类型 【长度 、 约束】,

列名 列的类型 【长度 、 约束】,

... ...

列名 列的类型 【长度 、 约束】

)

\*/

#案例创建表Book

CREATE TABLE book(

id INT ,#编号

bName VARCHAR(20), #图书名

price DOUBLE ,#价格

authorId INT, #作者编号

publishDate DATETIME #出版日期

);

#查询创建的表

DESC book;

#案例：创建表author

CREATE TABLE author(

id INT ,

au\_name VARCHAR(20),

nation VARCHAR(10)

)

DESC author;

#2.表的修改 alter

/\*

语法：

alter table 表名

add|drop|modify|change column 列名 【列名类型 约束】

修改表名

ALTER TABLE 表名

RENAME TO 新表名;

\*/

#1.修改表的列名

ALTER TABLE book

CHANGE COLUMN publishDate pubDate DATETIME;

DESC book;

#2.修改列的类型或约束

ALTER TABLE book

MODIFY COLUMN pubDate TIMESTAMP;

DESC book;

#3.添加新列

ALTER TABLE author

ADD COLUMN annual DOUBLE;

#4.删除列

ALTER TABLE author

DROP COLUMN annual;

#5.修改表名

ALTER TABLE author

RENAME TO book\_author;

DESC author;

#3.表的删除

DROP TABLE book\_author;

#可以添加判断 删除创建都可以

DROP TABLE IF EXISTS book\_author;

CREATE TABLE IF NOT EXISTS author(

id INT ,

au\_name VARCHAR(20),

nation VARCHAR(10)

)

#通用写法：

DROP DATABASE IF EXISTS 旧库名;

CREATE DATABASE 新库名;

DROP TABLE IF EXISTS 旧表名;

CREATE TABLE 新表名;

#4.表的复制

INSERT INTO author

values

(1,'村上春树','日本'),

(2,'莫言','中国'),

(3,'冯唐','中国'),

(4,'金庸','中国');

SELECT \* FROM author ;

#例1.仅仅复制表的结构

CREATE TABLE copy1 Like author ;

#查看

SELECT \* FROM author ;

SELECT \* FROM copy1 ;

#例2.复制表的结构+数据

#复制全部数据

CREATE TABLE copy2 SELECT \*FROM author;

#查看

SELECT \* FROM copy2 ;

DROP TABLE copy\_author ;

#复制部分数据

CREATE TABLE copy3

SELECT id,au\_name

FROM author

WHERE nation='中国';

#查看

SELECT \* FROM copy3 ;

#复制部分字段不复制数据

#(添加不成立条件)

CREATE TABLE copy4

SELECT id,au\_name

FROM author

WHERE 1=2;

## 常见的数据类型

### 数据类型说明

/\*

说明：

例如:tinyint (微整形)

占用 1 个字节 八位 1111 1111

有符号：-128~127

无符号：0~255

为什么：

1个字节有八位：

1111 1111

2^8 = 256

范围：从0开始时 0~255

有符号首位是符号位

1(符号位)111 1111(数值)

第一位为符号位 + -

数值是 2^7=128

范围：-128~127

数值型：

整形

小数：

定点数

浮点数

字符型：

较短的文本：char varchar

较长的文本：text,bolb(较长的二进制数据)

\*/

### 一、整形

/\*

分类：

tinyint ,smallint mediumint ,int/integer, big

1 2 3 4 5

特点:

1.如果不设值有符号还是无符号,默认有符号

如果想设置无符号，需要添加unsigned

2.如果插入范围超出整数范围报错(ut of range )并且插入临界值(MYSQL8.0报错不插入值)

3.如果不设值长度，会有默认长度

#注：此长度位数据显示的长度(如果数值不够默认为数值，如果要在左边填充0,必须搭配zerofills使用zerofill(会默认无符号UNSIGNED))，不是数据范围，数据范围是由类型决定的

\*/

#1.如何设置有符号和无符号

DROP TABLE IF EXISTS tab\_int;

CREATE TABLE tab\_int(

t1 INT ,

t2 INT UNSIGNED

);

DESC tab\_int ;

INSERT INTO tab\_int VALUES(-123);

INSERT INTO tab\_int VALUES(-123,-12);

INSERT INTO tab\_int VALUES(-123,455125415515212512);

SELECT \* FROM tab\_int;

### 二、小数

/\*

分类：

1.浮点型

float(M,D)

double(M,D)

2.定点型

dec(M,D)

decimal(M,D)

特点：

1.M和D

M 是整数部位+小数部位的长度

D 是小数部位

如果超过范围，则插入临界值(mysql8.0会报错不插入值)

2.M 和 D都能省略

如果是decimal，则M默认位10,D 默认为0

如果是float 和double ,则会根据插入的数值精度来决定精度

3.定点型的精度较高,如果说要求插入的精度较高需要使用定点型 如货币运算等

\*/

#测试M和D

DROP TABLE tab\_float;

CREATE TABLE tab\_float(

f1 FLOAT(5,2),

f2 DOUBLE(5,2),

f3 DECIMAL(5,2)

);

CREATE TABLE tab\_float(

f1 FLOAT,

f2 DOUBLE,

f3 DECIMAL

);

SELECT \* FROM tab\_float;

# D的作用

#正常

INSERT INTO tab\_float

VALUES (123.45,123.45,123.45);

#多一位 四舍五入

INSERT INTO tab\_float

VALUES (123.456,123.456,123.456);

#少一位 补0

INSERT INTO tab\_float

VALUES (123.4,123.4,123.4);

# M的作用 是总长度

INSERT INTO tab\_float

VALUES (15123.4,15123.4,15123.4);

SELECT \* FROM tab\_float;

#原则

/\*

所选择的类型越简洁越好，能报存数值的类型越小越好

\*/

### 三、字符型

/\*

较短的文本

char 固定

varchar 可变

其他

binary

varbinary 较短的二进制

enum枚举 例如 性别 季节

set集合

超长的文本

text

blob(较长的二进制)

特点：

写法: char(M)

M的意思:最大字符数，可以省略默认为0

特点: 固定

空间的消耗:高

效率:高

写法: varchar(M)

M的意思: 最大字符数，不可以可以省略

特点: 固定

空间的消耗:省

效率:低

\*/

#枚举

CREATE TABLE tab\_char(

c1 ENUM('a','b','c')

)

DESC tab\_char;

#一个一个 不区分大小写

INSERT INTO tab\_char VALUES ('a');

INSERT INTO tab\_char VALUES ('A');

INSERT INTO tab\_char VALUES ('c');

INSERT INTO tab\_char VALUES ('d');（x）

SELECT \* FROM tab\_char;

#set类型 集合 和enum类似 保存0~64个成员

CREATE TABLE tab\_set(

S1 SET('a','b','c')

);

#一个或多个 不区分大小写

INSERT INTO tab\_set VALUES ('a');

INSERT INTO tab\_set VALUES ('A','b');

INSERT INTO tab\_set VALUES ('c');

SELECT \* FROM tab\_set;

#四、日期型

/\*

分类

date 只保存日期

time 只保存时间

year 只保存年

datetime 保存日期加时间

timestamp 保存日期加时间

datetime

字节 8

范围 1000-9999

时区等影响 不受

datetime

字节 4

范围 1970-2038

时区等影响 受

\*/

CREATE TABLE tab\_date(

t1 DATETIME,

t2 TIMESTAMP

);

INSERT INTO tab\_date VALUES(NOW(),NOW());

SELECT \* FROM tab\_date;

#时区

SHOW VARIABLES LIKE 'time\_zone';

#设置时区

SET time\_zone='+9:00';

## 常见约束

### 常见约束

/\*

含义：一种限制，用于限制表中的数据，为了保证表中的数据的准确和可靠性

分类：六大约束

NOT NULL：非空，用于保证该字段的值不能为空

比如姓名、学号等

DEFAULT:默认，用于保证该字段有默认值

比如性别

PRIMARY KEY:主键，用于保证该字段的值具有唯一性，并且非空

比如学号、员工编号等

UNIQUE:唯一，用于保证该字段的值具有唯一性，可以为空

比如座位号

CHECK:检查约束【mysql中不支持】

比如年龄、性别

FOREIGN KEY:外键，用于限制两个表的关系，用于保证该字段的值必须来自于主表的关联列的值

在从表添加外键约束，用于引用主表中某列的值

比如学生表的专业编号，员工表的部门编号，员工表的工种编号

添加约束的时机：

1.创建表时

2.修改表时

约束的添加分类：

列级约束：

六大约束语法上都支持，但外键约束没有效果

表级约束：

除了非空、默认，其他的都支持

主键和唯一的大对比：

保证唯一性 是否允许为空 一个表中可以有多少个 是否允许组合

主键 √ × 至多有1个 √，但不推荐

唯一 √ √ 可以有多个 √，但不推荐

两种约束

主键：唯一 不能为空 至多一个 可以组合不推荐

唯一：唯一 可以为空 可以多个 可以组合不推荐

组合：两列 变成一列的功用 主键

例如：两列都不能重复 constraint pk PRIMARY KEY(id,name), #主键

外键：

1、要求在从表设置外键关系

2、从表的外键列的类型和主表的关联列的类型要求一致或兼容，名称无要求

3、主表的关联列必须是一个key（一般是主键或唯一）

4、插入数据时，先插入主表，再插入从表

删除数据时，先删除从表，再删除主表

\*/

CREATE DATABASE students;

### 一、创建表时添加约束

#1.添加列级约束

/\*

#mysql中

语法：

直接在字段名后面和类型后面追加 约束类型即可。

只支持：默认、非空、主键、唯一

\*/

USE students ;

#学生信息表

CREATE TABLE stuinfo(

id INT PRIMARY KEY, #主键

stuName VARCHAR(20) NOT NULL, #非空

gender CHAR(1), CHECK(gender='男' OR gender='女'), #检查

seat INT DEFAULT 18, #默认约束

majorId INT REFERENCES major(id) #外键

);

#主表

CREATE TABLE major (

id INT PRIMARY KEY ,

majorName VARCHAR(20)

);

DESC stuinfo;

#查看stuinfo的所有的索引，包括主键，外键，唯一。

SHOW INDEX FROM stuinfo;

#2.添加表级约束

/\*

语法：在各个字段的后面

【constraint 约束名】(可以不写) 约束类型(字段名)

\*/

DROP TABLE IF EXISTS stuinfo;

CREATE TABLE stuinfo(

id INT,

stuName VARCHAR(20),

gender CHAR(1),

seat INT,

age INT,

majorid INT,

constraint pk PRIMARY KEY(id), #主键

constraint up UNIQUE(seat), #唯一键

constraint ck CHECK(gender='男' OR gender='女'), #检查

constraint fk\_stuinfo\_major FOREIGN KEY(majorid) REFERENCES major(id) #外键

);

SHOW INDEX FROM stuinfo;

#通用的写法：

CREATE TABLE IF NOT EXISTS stuinfo(

id INT PRIMARY KEY,

stuname VARCHAR(20) NOT NULL,

sex CHAR(1),

age INT DEFAULT 18,

seat INT UNIQUE,

majorid INT ,

constraint fk\_stuinfo\_major FOREIGN KEY(majorid) REFERENCES major(id)

);

### 二、修改表时添加约束

/\*

1、添加列级约束

alter table 表名 modify column 字段名 新约束

2、添加表级约束

alter table 表名 add 【constraint 约束名】约束类型(字段名) 【外键的引用】

\*/

DROP TABLE IF EXISTS stuinfo;

CREATE TABLE stuinfo(

id INT,

stuName VARCHAR(20),

gender CHAR(1),

seat INT,

age INT,

majorid INT

);

#1.添加非空约束

ALTER TABLE stuinfo MODIFY COLUMN stuname VARCHAR(20) NOT NULL;

#2.添加默认约束

ALTER TABLE stuinfo MODIFY COLUMN age INT UPDATE 18;

#3.添加主键约束

#1.列级约束

ALTER TABLE stuinfo MODIFY COLUMN id INT PRIMARY KEY;

#2.表级约束

ALTER TABLE stuinfo ADD PRIMARY KEY(id);

#4.添加唯一

#1.列级约束

ALTER TABLE stuinfo MODIFY COLUMN seat INT UNIQUE;

#2.表级约束

ALTER TABLE stuinfo ADD UNIQUE(seat);

#5.添加外键

ALTER TABLE stuinfo ADD FOREIGN KEY(majorid) REFERENCES major(id);

ALTER TABLE stuinfo ADD CONSTRAINT fk\_stuinfo\_major FOREIGN KEY(majorid) REFERENCES major(id);

### 三、修改表时删除约束

#1.删除非空约束

ALTER TABLE stuinfo MODIFY COLUMN stuname VARCHAR(20) NULL;

#2.删除默认约束

ALTER TABLE stuinfo MODIFY COLUMN age INT ;

#3.删除主键

ALTER TABLE stuinfo DROP PRIMARY KEY;

#4.删除唯一

ALTER TABLE stuinfo DROP seat;

#5.删除外键

ALTER TABLE stuinfo DROP FOREIGN KEY fk\_stuinfo\_major;

SHOW INDEX FROM stuinfo;

## 标识列

### 标识列说明

/\*

又称为自增长列

含义：可以不用手动的插入值,系统提供默认的的序列值

特点：

1、标识列必须和主键搭配吗？ 不一定 ，必须是一个key

2、一个表中可以有多个标识列？ 至多一个

3、标识列的类型只能是数值

4、标识列可以通过SET auto\_increment\_increment = 3 设置步长

也可以手动设置值

\*/

### 一、创建表时设置标标识列

#mysql8.0版本修复自增列值重复利用情况

#mysql8.0以前的版本

DROP TABLE IF EXISTS tab\_identity;

CREATE TABLE tab\_identity(

id INT PRIMARY KEY AUTO\_INCREMENT,#默认步长1

NAME VARCHAR(20)

);

INSERT INTO tab\_identity VALUES (1,'join');

INSERT INTO tab\_identity(NAME) VALUES ('join');

SELECT \* FROM tab\_identity;

SHOW VARIABLES LIKE '%AUTO\_INCREMENT%';

#更改步长

SET auto\_increment\_increment = 3;

### 二、修改设置标识列

ALTER TABLE tab\_identity MODIFY COLUMN id INT PRIMARY KEY AUTO\_INCREMENT;

### 三、修改表时删除标识列

ALTER TABLE tab\_identity MODIFY COLUMN id INT PRIMARY KEY ;#不添加即可

## Tcl语言

### 事务控制语言说明

/\*

Transaction Control Language 事务控制语言

事务：

一个或一组sql 语句组成一个执行单元，这个执行单元要么全部执行，要么全部不执行

失败数据回滚

回滚：回复之前的状态

存储引擎 inndb 支持

#查看存储引擎

SHOW engines ;

案例：转账

张三丰 1000

郭襄 1000

update 表 set 张三丰的余额=500 where name='张三丰';

意外

update 表 set 郭襄的余额=1500 where name='郭襄';

事务的特性：

ACID

原子性：一个事务不可在分割，要么都执行要么都不执行

一致性：一个事务执行会使数据从一个一致的状态切换到另外一个一致状态

隔离性：一个事务的执行不受其他事务的干扰

持久性：一个事务一旦提交，会永久的改变数据库的数据

### 事务的创建

隐式事务：没有明显的开启和结束

比如：insert , update ,delete 语句

delete from 表 where id = 1;

显示事务：事务具有明显的开启和结束

前提：必须先设置自动提交功能为禁用

set autocommit = 0;

步骤一：开启事务

set autocommit = 0;

start transaction ; 可选的

步骤二：编写事务中的sql语句(select insert update delete)

语句1;

语句2;

语句3;

...

步骤三：结束事务

commit;提交事务

rollback;回滚事务

#2.演示事务对于delete和truncate的处理的区别

SET autocommit=0;

START TRANSACTION;

DELETE FROM account;

ROLLBACK;

Delete 可以回滚

Truncate 不可以回滚

savepoint 节点名;设置保存点

#查看变量

SHOW variables LIKE '%autocommit%';

\*/

#查看存储引擎

SHOW engines ;

#查看变量

SHOW VARIABLES LIKE '%autocommit%';

#演示事务的使用步骤

/\*

导入表

DROP TABLE IF EXISTS account;

CREATE TABLE account(

id INT PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(20),

balance DOUBLE

);

INSERT INTO account(username ,balance)

VALUES('张无忌',1000),('赵敏',1000);

SELECT \* FROM account;

\*/

事务的隔离级别

脏读 幻读 不可重复读

read uncommitted : √ √ √

read committed : x √ √

repeatable read： × × √

serializable: × × ×

mysql中默认 第三个隔离级别 repeatable read

oracle中默认第二个隔离级别 read committed

查看隔离级别

select @@tx\_isolation;

SHOW VARIABLES LIKE 'transaction\_isolation';

设置隔离级别

set session|global transaction isolation level 隔离级别;

#开启事务

SET autocommit=0;

START TRANSACTION;

#编写一组事务的语句

update account set balance=1000 where username='张三丰';

update account set balance=1000 where username='赵敏';

#结束事务 (提交事务) 回滚或提交

#ROLLBACK; 回滚不会改变数据

COMMIT; 提交修改数据

SELECT \* FROM account;

#演示savepoint 的使用 搭配rollback使用

SET autocommmit=0;

START TRANSACTION;

DELETE FROM account WHERE id = 25;

SAVEPOINT a ; #设置保存点

DELETE FROM account WHERE id = 28;

ROLLBACK TO a; #回滚到保存点

## 视图

### 视图说明

/\*

含义：虚拟表，和普通表一样使用

mysql5.1版本出现的新特性，是通过表的动态生成的数据

比如：舞蹈班和普通班的对比

比如：舞蹈班和普通班级的对比

创建语法的关键字 是否实际占用物理空间 使用

视图 create view 只是保存了sql逻辑 增删改查，只是一般不能增删改

表 create table 保存了数据 增删改查

\*/

#案例：查询姓张的学生名和专业名

SELECT stuname,majorName

FROM stuinfo s

INNER JOIN major m ON s.majorid = m.id

WHERE s.stuname LIKE '张%';

CREATE VIEW V1

AS

SELECT stuname,majorName

FROM stuinfo s

INNER JOIN major m ON s.majorid = m.id

WHERE s.stuname LIKE '张%';

SELECT \* FROM V1 WHERE stuname LIKE '张%';

### 一、创建视图

/\*

语法：

create view 视图名

as

查询语句;

\*/

USE myemployees;

#案例1.查询姓名中包含a字符的员工名、部门名和工种信息

##1.创建

CREATE VIEW myv1

AS

SELECT last\_name , department\_name , job\_title ,email

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

INNER JOIN jobs j ON e.job\_id = j.job\_id

##1.使用

SELECT \* FROM myv1 WHERE last\_name LIKE '%a%';

#2.查询各个部门的平均工资级别

#创建视图查看每个部门的平均工资

CREATE VIEW myv2

AS

SELECT AVG(salary) ag , department\_id

FROM employees

GROUP BY department\_id;

SELECT \* FROM myv2 ;

#使用

SELECT myv2.`ag`,g.grade\_level

FROM myv2

JOIN job\_grades g

ON myv2.`ag` BETWEEN g.`lowest\_sal` AND g.`highest\_sal`;

#3.查询平均工资最低的部门信息

SELECT \* FROM myv2 ORDER BY ag LIMIT 1;

#4.查询平均工资最低的部门名和工资

CREATE VIEW myv3

AS

SELECT \* FROM myv2 ORDER BY ag LIMIT 1;

SELECT d.\* ,m.ag

FROM myv3 m

INNER JOIN departments d

ON m.department\_id = m.department\_id;

### 二、修改视图

#方式一

/\*

create or replace view 视图名

as

查询语句

\*/

#例：

SELECT \* FROM myv3;

CREATE OR REPLACE VIEW myv3

AS

SELECT AVG(salary) ,job\_id

FROM employees

GROUP BY job\_id;

#方式二

/\*

语法：

alter view 视图名

as

查询语句;

\*/

ALTER VIEW myv3

AS

SELECT \*

FROM employees;

SELECT \* FROM myv3;

### 三、删除视图

/\*

语法：

drop view 视图名,视图名,....

\*/

#例

DROP VIEW myv3;

#四、查看视图

DESC myv3;

SHOW CREATE VIEW myv3;

#五、视图的更新

CREATE OR REPLACE VIEW myv1

AS

SELECT last\_name ,email

FROM employees ;

SELECT \* FROM myv1;

#1.插入

INSERT INTO myv1 VALUES('张飞','ZF@qq.com');

#2.修改

UPDATE myv1 SET last\_name='张无忌' WHERE last\_name='张飞';

#3.删除

DELETE FROM myv1 WHERE last\_name='张无忌';

##具备以下特点的视图不允许更新

#包含以下关键字的sql语句：分组函数、distinct、group by、having、union或者union all

CREATE OR REPLACE VIEW myv1

AS

SELECT MAX(salary) m,department\_id

FROM employees

GROUP BY department\_id;

SELECT \* FROM myv1;

#更新

UPDATE myv1 SET m=9000 WHERE department\_id=10;

#②常量视图

CREATE OR REPLACE VIEW myv2

AS

SELECT 'john' NAME;

SELECT \* FROM myv2;

#更新

UPDATE myv2 SET NAME='lucy';

#Select中包含子查询

CREATE OR REPLACE VIEW myv3

AS

SELECT department\_id,(SELECT MAX(salary) FROM employees) 最高工资

FROM departments;

#更新

SELECT \* FROM myv3;

UPDATE myv3 SET 最高工资=100000;

#join

CREATE OR REPLACE VIEW myv4

AS

SELECT last\_name,department\_name

FROM employees e

JOIN departments d

ON e.department\_id = d.department\_id;

#更新

SELECT \* FROM myv4;

UPDATE myv4 SET last\_name = '张飞' WHERE last\_name='Whalen';

INSERT INTO myv4 VALUES('陈真','xxxx');

#from一个不能更新的视图

CREATE OR REPLACE VIEW myv5

AS

SELECT \* FROM myv3;

#更新

SELECT \* FROM myv5;

UPDATE myv5 SET 最高工资=10000 WHERE department\_id=60;

#where子句的子查询引用了from子句中的表

CREATE OR REPLACE VIEW myv6

AS

SELECT last\_name,email,salary

FROM employees

WHERE employee\_id IN(

SELECT manager\_id

FROM employees

WHERE manager\_id IS NOT NULL

);

#更新

SELECT \* FROM myv6;

UPDATE myv6 SET salary=10000 WHERE last\_name = 'k\_ing';

## 变量

### 变量说明

/\*

系统变量：

全局变量

会话变量

自定义变量：

用户变量

局部变量

\*/

### 一、系统变量

/\*

说明：变量由系统提供，不是用户定义，属于服器层面

使用的语法:

1、查看所有的系统变量

SHOW GLOBAL | 【SESSION】 VARIABLES;

show global | session variables ;

2、查看满足条件的部分系统变量

show global | session variables like '%char%' ;

3、查看指定的某个系统变量的值

select @@global |【session】.系统变量名;

4、为某个系统变量赋值

方式一

set global | 【session】系统变量名 = 值;

方式二

set @@global | 【session】.系统变量名 = 值;

注意：

如果是全局级别，则需要加global ,如果是会话级别,则加session 不加默认session

\*/

#1.全局变量

/\*

作用域：服务器每次启动将会为所有的全局变量赋初始值，

针对于说有的会话（有效），但不能跨重启（只能修改配置才能重启有效）

\*/

##1.查看所有的全局变量

show global variables ;

##2.查看部分全局变量

show global variables like '%char%';

##3.查看指定变量的值

select @@global.autocommit;

select @@tx\_isolation;

##4.为某个指定的全局变量赋值

set @@global.autocommit=1 ;

#2.会话变量

/\*

作用域：仅仅针对于当前的会话有效

\*/

##1.查看所有的会话变量

show variables ;

show session variables ;

##2.查看部分会话变量

show variables like '%char%';

show session variables like '%char%';

##3.查看指定变量的值

select @@session.autocommit;

select @@autocommit;

##4.为某个指定的会话变量赋值

方式一:

set @@autocommit=0 ;

方式二:

set session autocommit=1 ;

### 二、自定义变量

/\*

说明：变量是用户自定义的，不是由系统的

使用步骤：

声明

赋值

使用（查看 、比较、运算）

\*/

#1.用户变量

/\*

作用域：针对于当前的会话（连接）有效，同于会话变量的作用域

应用在任何地方，也就是begin end的里面或外面

\*/

##1.声明并初始化

set @用户变量名 = 值;或

set @用户变量名:=值;或

select @用户变量名:=值;

##2.赋值（更新用户变量的值）

方式一：通过set 或select

set @用户变量名 = 值;或

set @用户变量名:=值;或

select @用户变量名:=值;

方式二：通过select into

select 字段(一个值) into 变量名

from 表;

##3.使用（查看用户变量名）;

select @用户变量名;

#案例

set @name='john';

set @name=100;

set @count=1;

select count(\*) into @count

FROM employees;

select @count;

#2、局部变量

/\*

作用域：仅仅在定义它的begin end 中有效

应用在 begin end 中 必须是第一句话

\*/

#声明

declare 变量名 类型

declare 变量名 类型 default 值;

#赋值

方式一：通过set 或select

set 局部变量名 = 值;或

set 局部变量名:=值;或

select @用户变量名:=值;

方式二：通过select into

select 字段(一个值) into 局部变量名

from 表;

#使用

select 变量名;

对比用户变量和局部变量

#用户变量和局部变量的对比

作用域 定义位置 语法

用户变量 当前会话 会话的任何地方 加@符号，不用指定类型

局部变量 定义它的BEGIN END中 BEGIN END的第一句话 一般不用加@,需要指定类型

#案例：声明两个变量，求和并打印

#1.用户变量

set @m=1;

set @n=2;

set @sum = @m +@n;

SELECT @sum;

#2.局部变量(会报错 只能在 begin end 中使用)

DECLARE m INT DEFAULT 1;

DECLARE n INT DEFAULT 2;

DECLARE sum INT;

SELECT sum;

## 存储过程

### 存储过程说明

/\*

存储过程和函数：类似于java中的方法

好处：

1、提高代码的重用性

2、简化操作

\*/

#存储过程 #需要在命令窗口运行

/\*

含义：一组预先编译好的SQL语句的集合，理解成批处理的语句

1、提高了代码的重用性

2、简化操作

3、减少了编译次数并且减少了和数据库服务器的连接次数，提高了效率

\*/

### 一、创建语法

create procedure 存储过程名(参数列表)

begin

存储过程体(一组合法的SQL语句)

end

注意：

1、参数列表包含三部分

参数模式 参数名 参数列表

in stuname varchar(20)

参数模式：

in :该参数可以作为输入 ,也就是该参数需要调用方传入值

out:该参数可以作为输出，也就是该参数可以作为返回值

inout:该参数既可以作为输入又可以作为输出，也就是该参数既需要传入值，又可以返回值

2、如果存储过程仅仅只有句话，begin end 可以省略

存储过程中的每条SQL语句的结尾要求必须加分号。

存储过程的结尾可以使用 delimiter 重新设置

语法：

delimiter + 结束标记

案例：

delimiter $ (结束标记)

### 二、调用语法

call 存储过程名(实参列表);

#1.空参列表

#案例：插入到admin 表中五条记录

select \* from admin ;

#需要在命令窗口运行

delimiter $

CREATE PROCEDURE myp1()

BEGIN

INSERT INTO admin(username,`password`)

VALUES('john1','0000'),

('john2','0000'),

('john3','0000'),

('john4','0000'),

('john5','0000');

END $

#需要在命令窗口运行

#调用

CALL myp1()$

SELECT \* FROM admin;

DROP PROCEDURE myp1;

#2、创建带in模式参数的存储过程

#案例1：创建存储过程实现 根据女神名，查询对应的男神信息

CREATE PROCEDURE myp2 (IN beautyName VARCHAR(20))

BEGIN

SELECT bo.\*

FROM beauty b

LEFT JOIN boys bo

ON b.boyfriend\_id = bo.id

WHERE b.`name`= beautyName;

END $

#调用

CALL myp2('柳岩')$

DROP PROCEDURE myp2;

#案例2：创建存储过程判断用户是否登录成功

CREATE PROCEDURE myp3(IN username VARCHAR(20),IN password VARCHAR(20))

BEGIN

DECLARE result INT DEFAULT 0;#声名并初始化

SELECT count(\*) INTO result #赋值

FROM admin a

WHERE a.username = username

AND a.password = password;

SELECT IF(result>0,'成功','失败'); #使用

END $

#调用

DROP PROCEDURE myp3;

call myp3('张飞','8888') $

#创建带out模式的存储过程

#案例1：根据女神名，返回对应的男神名 --带一个返回值的

CREATE PROCEDURE myp5( IN bveautyName VARCHAR(20),OUT boyName VARCHAR(20))

BEGIN

SELECT bo.boyName INTO boyName

FROM boys bo

INNER JOIN beauty b

ON bo.id = b.boyfriend\_id

WHERE b.name = bveautyName;

END $

CALL myp5('柳岩', @bname)$

SELECT @bname$

DROP PROCEDURE myp5;

#案例2：根据输入的女神名，返回对应的男神名和魅力值

CREATE PROCEDURE myp7(IN beautyName VARCHAR(20),OUT boyName VARCHAR(20),OUT usercp INT)

BEGIN

SELECT boys.boyname ,boys.usercp INTO boyname,usercp

FROM boys

RIGHT JOIN

beauty b ON b.boyfriend\_id = boys.id

WHERE b.name=beautyName ;

END $

#调用

CALL myp7('小昭',@name,@cp)$

SELECT @name,@cp$

#4.创建带inout 模式参数的存储过程

#案例1：传入a和b两个值，最终a和b 都翻倍并返回

CREATE PROCEDURE myp8(INOUT a INT ,INOUT b INT)

BEGIN

SET a=a\*2;

SET b=b\*2;

END $

#调用

SET @m=10$

SET @n=20$

CALL myp8(@m,@n)$

SELECT @m,@n$ (查询一下a,b的返回结果)

### 三、删除存储过程

语法：DROP PROCEDURE 存储过程名

DROP PROCEDURE myp1;

### 四、查看存储过程的信息

DESC myp2;× 不可以

SHOW CREATE PROCEDURE myp2;

## 函数

### 函数说明

/\*

含义：一组预先编译好的SQL语句的集合，理解成批处理语句

1、提高代码的重用性

2、简化操作

3、减少了编译次数并且减少了和数据库服务器的连接次数，提高了效率

区别：

存储过程：可以有0个返回，也可以有多个返回

函数：有且仅有一个返回值 不能没有不能多

适合处理数据后返回一个结果

\*/

### 一、创建语法

create function 函数(参数列表) returns 返回类型

begin

函数体

end

/\*

注意：

参数列表 包含两部分

参数列表 参数类型

函数体：肯定会有return语句，如果没有会报错

如果return语句没有放在函数体最后也不报错，但不建议

return 值;

3.函数体仅有一句话，则可以省略begin end

4.使用delimiter 语句设置结束标记

\*/

### 二、调用语法

SELECT 函数名(参数列表)

#案例演示

#1.无参返回值

#案例：返回公司的员工个数

delimiter $

CREATE FUNCTION myf1() RETURNS INT

BEGIN

DECLARE c INT DEFAULT 0 ;#定义变量

SELECT COUNT(\*) INTO c

FROM employees;

RETURN c;

END $

#调用

SELECT myf1()$

/\*

如果报：

ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you \*might\* want to use the less safe log\_bin\_trust\_function\_creators variable)

一下是解决方案：

这是我们开启了bin-log, 我们就必须指定我们的函数是否是

1 DETERMINISTIC 不确定的

2 NO SQL 没有SQl语句，当然也不会修改数据

3 READS SQL DATA 只是读取数据，当然也不会修改数据

4 MODIFIES SQL DATA 要修改数据

5 CONTAINS SQL 包含了SQL语句

其中在function里面，只有 DETERMINISTIC, NO SQL 和 READS SQL DATA 被支持。如果我们开启了 bin-log, 我们就必须为我们的function指定一个参数。

在MySQL中创建函数时出现这种错误的解决方法：

set global log\_bin\_trust\_function\_creators=TRUE;

\*/

#2.有参有返回

#案例1：根据员工名，返回它的工资

delimiter $

CREATE FUNCTION myf2(empName VARCHAR(20)) RETURNS DOUBLE

BEGIN

SET @sal = 0; #定义用户变量

SELECT salary INTO @sal

FROM employees

WHERE last\_name = empName;

RETURN @sal;

END $

SELECT myf2('Kochhar')$

#案例2：根据部门名，返回该部门的平均工资

CREATE FUNCTION myf3(deptName VARCHAR(20)) RETURNS DOUBLE

BEGIN

DECLARE sal DOUBLE ;

SELECT AVG(salary) INTO sal

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

WHERE d.department\_name=deptName;

RETURN sal;

END $

SELECT myf3('IT')$

### 三、查看函数

SHOW CREATE FUNCTION myf3;

show create function myf3;

### 四、删除函数

DROP FUNCTION myf3;

drop function myf3;

## 流程控制结构

### 流程控制说明

/\*

顺序、分支、循环

\*/

### 一、分支结构

#1.if函数

/\*

语法：if(条件,值1，值2)

功能：实现双分支

应用在begin end中或外面

\*/

#2.case结构

/\*

语法：

情况1：类似于switch

case 变量或表达式

when 值1 then 语句1;

when 值2 then 语句2;

...

else 语句n;

end

情况2：

case

when 条件1 then 语句1;

when 条件2 then 语句2;

...

else 语句n;

end

应用在begin end 中或外面

\*/

#3.if结构

/\*

功能：实现多重分支

语法：

if 条件1 then 语句1;

elseif 条件2 then 语句2;

elseif 条件3 then 语句3;

...

【else 语句n;】

end if ;

\*/

#案例1：创建函数，实现传入成绩，如果成绩>90,返回A，如果成绩>80,返回B，如果成绩>60,返回C，否则返回D

DELIMITER 定好结束符为"$$", 然后最后又定义为";", MYSQL的默认结束符为";".

delimiter $ #结束符号

CREATE FUNCTION test\_if(score INT) RETURNS CHAR

BEGIN

IF score >= 90 AND score <=100 THEN RETURN 'A';

ELSEIF score >= 80 THEN RETURN 'B';

ELSEIF score >= 60 AND score <=100 THEN RETURN 'C';

ELSE RETURN 'A';

END IF;

END $

SELECT test\_if(85)$

### 二、循环结构

/\*

分类：

while 、loop 、repeat

循环控制结构

iterate 类似于 continue ,继续，结束本次循环，继续下一次

leave 类似于 break，跳出，结束当前所在的循环

\*/

#1.while

/\*

语法：

【标签label：】while 循环条件 do

循环体;

end while 【标签label】;

while (循环条件){

循环体;

}

\*/

#2.loop

/\*

语法：

【标签：】loop

循环体;

end loop 【标签】

用来模拟死循环(没有结束条件)

可以搭配leave 结束死循环

\*/

#3.repeat

/\*

语法：

【标签：】repeat

循环体;

until 结束条件

end repeat 【标签】;

类似于

do{

}while();

\*/

#案例：批量插入，根据次数插入到admin 表中多条记录

delimiter $

CREATE PROCEDURE pro\_while1(IN insertCount INT)

BEGIN

DECLARE i INT DEFAULT 1;

WHILE i<= insertCount DO

INSERT INTO admin (username,`password`)

VALUES (CONCAT('Rose',i),'6666');

SET i = i+1;

END WHILE ;

END $

CALL pro\_while1(100)$

DROP PROCEDURE pro\_while1

#2、添加leave语句

#leave的使用

搭配标签label 使用

类似break

#案例：批量插入，根据次数插入到admin 表中多条记录,如果次数多于20则停止

delimiter $

TRUNCATE TABLE admin$

DROP PROCEDURE pro\_while1$

CREATE PROCEDURE pro\_while1(IN insertCount INT)

BEGIN

DECLARE i INT DEFAULT 1;

a:WHILE i<= insertCount DO

INSERT INTO admin (username,`password`)

VALUES (CONCAT('Rose',i),'6666');

IF i >=20

THEN LEAVE a;

END IF;

SET i = i+1;

END WHILE a;

END $

CALL pro\_while1(100)$

DROP PROCEDURE pro\_while1;

select \* from admin;

#3.添加iterate语句

搭配标签label 使用

类似continue

#案例：批量插入，根据次数插入到admin表中多条记录，只插入偶数次

TRUNCATE TABLE admin$

DROP PROCEDURE test\_while1$

CREATE PROCEDURE test\_while1(IN insertCount INT)

BEGIN

DECLARE i INT DEFAULT 0;

a:WHILE i<=insertCount DO #a是标签

SET i=i+1;

IF MOD(i,2)!=0 THEN ITERATE a;

END IF;

INSERT INTO admin(username,`password`) VALUES(CONCAT('xiaohua',i),'0000');

END WHILE a;

END $

CALL test\_while1(100)$