

一、数据库概述

2021年9月4日 19:36

内存层面：数组、集合

文件：硬盘--查询困难

数据库优点：

实现数据持久化

统一管理，方便查询

DB

数据库（database）：存储数据库的“仓库”。保存了一系列有组织的数据

DBMS

数据库管理系统（Database Management System），即数据库软件。数据库是通过DBMS创建和操作的容器。

常见的数据库软件：MySQL、Oracle（贵，服务收费）、DB2（适合处理海量数据）、SqlServer（微软公司，只能安装在win系统）等。

SQL

结构化查询语言（Structure Query Language）：专门用来与数据库通信的语言。

SQL的优点：

- ①不是某个特定数据库供应商专有的语言，几乎所有的DBMS都支持SQL
- ②简单易学
- ③实际上是一种强有力的语言，进行复杂和高级的数据库操作。

存储特点

- 1.将数据放到表中，表再放到库中
- 2.一个数据库中可以有多个表，每个表都有一个名字，用来标识自己。表名具有唯一性。
- 3.表具有一些特性，这些特性定义了数据在表中如何存储，类似java中的类的设计
- 4.表由列组成，我们也称为字段，所有表都是由一个或多个列组成，每一列类似java中的属性。
- 5.表中的数据是按行存储的，每一行类似于java中的对象

二、MySQL软件

2021年9月4日 20:32

历史：MySQLAB--sun--oracle

优点：

成本低：开放源代码，一般免费使用

性能高：执行很快，移植性好

简单：很容易安装和使用

DBMS分为两类：

- 基于共享文件系统的DBMS(Access)
- 基于客户机——服务器(C/S)的DBMS
(MySQL/Oracle/SqlServer)

属于cs架构的软件，一般来讲安装服务端

MySQL服务的启动与停止（官网下载社区版先配置好环境变量，见博客收藏安装并设置）

- ①以管理员身份打开cmd
- ②net start mysql(启动服务)
- ③net stop mysql (停止服务)
- ④登录服务：mysql （可省略 -h主机名 -P端口号） -uroot -p密码
- ⑤退出：exit

三、MySQL常用命令

2021年9月5日 12:42

一、常用命令

1.查看当前全部的数据库

show databases;

2.打开指定的库

use 库名

3.查看当前库的所有表

show tables;

4.查看其他库的所有表

show tables from 库名;

5.创建表

create table 表名 (

 列名 列类型,

 列名 列类型,

 .

 .

 .

) ;

6.查看表结构

desc 表名;

7.查看服务器版本

方式一：登录服务端 select version();

方式二：没登录服务端 mysql --version

二、语法规范

1.不区分大小写，但建议关键字大写，表名、列名小写

2.每条命令最好用；结尾

3.每条命令根据需要，可以进行缩进或换行

4.注释

单行注释：#注释文字

单行注释：-- (空格) 注释文字

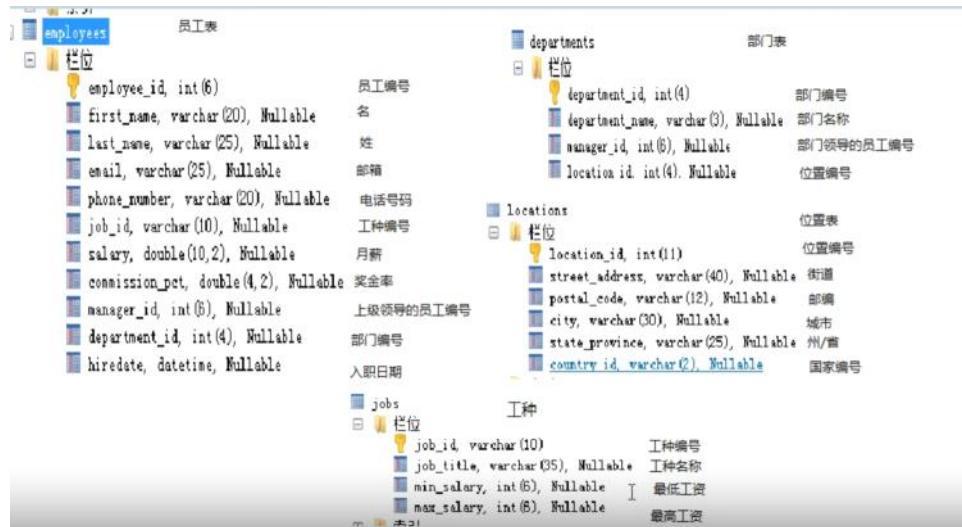
多行注释：/*注释文字*/

三、图形化用户界面客户端

SQLyog具体安装步骤看博客收藏

四、DQL语言

2021年9月5日 22:09



练习表介绍

一、基础查询

```
#进阶一：基础查询
/*
语法：
select 查询列表 from 表名；

特点：
1.查询列表可以是：表中的字段、常量、表达式、函数
2.查询结果是一个虚拟的表格
*/



#: 打开相应的库
USE myemployees;

#1: 查询表中的单个字段
SELECT last_name FROM employees;

#2: 查询表中的两个字段
SELECT last_name,salary,email FROM employees;

#3: 查询表中的所有字段
#双击表中字段名，使用F12格式化
SELECT
    `employee_id`,
    `first_name`,
    `last_name`,
    `email`,
    `phone_number`,
    `job_id`,
    `salary`,
    `commission_pct`,
    `manager_id`,
    `department_id`,
    `hiredate`
FROM
    employees ;

#: * 按表默认顺序排
SELECT * FROM employees;

#4. 查询常量值
SELECT 100;
```

```

SELECT * FROM employees;

#4. 查询常量值
SELECT 100;
SELECT 'join';

#5. 查询表达式
SELECT 100%98;

#6. 查询函数
SELECT VERSION();

#7. 起别名
/*
好处:
①便于理解
②如果要查询的字段有重名的情况, 使用别名可以区分开来
*/
#方式一
SELECT 100%98 AS 结果;
SELECT last_name AS 姓,first_name AS 名 FROM employees;
#方式二,如果别名中有特殊符号时, 将别名加"
SELECT last_name 姓 FROM employees;

#8. 去重
#例子: 查询员工表中涉及到的部门编号
SELECT DISTINCT department_id FROM employees;

#9. +号的作用
/*
mysql 中的+号, 只有一个功能: 运算符
select 100+90; 都是数值型做加法运算
select '123'+90; 会试图将字符串型转换为数值型,
                    转换成功就做加法运算;
                    转换失败, 将字符串型转换为0, 再运算;
                    只要其中一方为null, 则结果肯定为null;
*/
#例子: 查询员工名和姓连接成一个字段, 并显示为 姓名
SELECT
    CONCAT(first_name, last_name) AS 姓名
FROM
    employees

```

二、条件查询

```

#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.按条件表达式筛选, > < = != <> (不等于)  <= >=
2.按逻辑运算符: && || !
            and or not
3.模糊查询
    like
    between and
    in
    is null
*/
#1.按条件表达式筛选
#查询工资大于12000
SELECT
    *
FROM
    employees
WHERE
    salary > 12000 ;

#2按逻辑表达式筛选
#查询工资在10000到20000之间的员工名、工资及奖金
SELECT
    last_name,
    salary,
    commission_pct
FROM
    employees
WHERE salary >= 10000
    AND salary <= 20000 ;

```



|_-| 数也可

```

WHERE salary >= 10000
AND salary <= 20000;

#3模糊查询
曰/*like
特点:
一般与通配符搭配使用, 通配符: %:任意多个字符含0个
:_:任意单个字符
\转义字符
\$:ESCAPE '\$', 自定义转义字符

*/
#查询员工名包含字符a的员工信息
SELECT * FROM employees WHERE last_name LIKE '%at';
SELECT * FROM employees WHERE last_name LIKE '%_\_%';
SELECT * FROM employees WHERE last_name LIKE '%_S_%' ESCAPE '$';

曰/*between and
①可以提高语句的简洁度
②包含临界值
③两个临界值不能反
*/
#查询员工编号在100到120之间的员工信息
SELECT * FROM employees WHERE employee_id BETWEEN 100 AND 120;

曰/*in
含义: 判断某字段的值是否属于in 列表中的某一项
特点:
①使用in 提高简洁度
②in列表的值类型必须一致或兼容
*/
#查询员工的工种编号是'IT_PROT','AD_VP','AD_PRES'中一个的员工名和工种编号
SELECT last_name,job_id FROM employees WHERE job_id IN('IT_PROT','AD_VP','AD_PRES');

曰/*is null
=或<>不能用于判断null值
is null 或 is not null 可以判断是否为null值
<=>:安全等于, 可以判断null值和普通值
*/
#查询没有奖金的员工名和奖金率
SELECT last_name,commission_pct FROM employees WHERE commission_pct IS NULL;

```

三、排序查询

```

#进阶三: 排序查询
曰/*
语法:
    select 查询列表
    from 表
    【where 筛选条件】
    order by 排序列表 【asc|desc】(升/降)

特点:
    ①desc 降序
    ②asc 升序, 不写默认升序排列
    ③order by子句中可以支持单个字段、多个字段、表达式、函数、别名.
    ④order by 子句一般是放在查询语句的最后面, limit子句除外
*/
#查询员工信息, 工资高到低
SELECT * FROM employees ORDER BY salary DESC;
#查询部门编号>=90的员工信息, 按入职时间先后排序[添加筛选条件]
SELECT * FROM employees WHERE department_id>=90 ORDER BY hiredate ASC;
#按年薪高低显示员工的信息和年薪[按别名排序]
SELECT *,salary*12*(1+IFNULL(commission_pct,0)) AS 年薪 FROM employees
ORDER BY 年薪 DESC;
#根据员工名字长度排序[按函数排序]
SELECT LENGTH(last_name) AS 字节长度 ,last_name,salary
FROM employees
ORDER BY 字节长度 ASC;
#查询员工信息, 先按工资排序, 再按员工编号排序[按多个字段排序]
SELECT * FROM employees ORDER BY salary ASC ,employee_id DESC;

```

四、常见函数---单行函数

```
#进阶四： 常见函数
/*
调用: select 函数名(实参列表) from 表;
特点:
    函数名
    函数功能
分类:
    1. 单行函数
        如: concat/length/ifnull等
    2. 分组函数 (统计函数, 聚合函数, 组函数)

    功能: 做统计使用

*/
#单行函数
#1. 字符函数
#@length 获取参数值的字节数
#@concat 拼接字符串
#@upper lower 大小写
SELECT CONCAT(UPPER(last_name), '-', LOWER(first_name)) AS 姓名 FROM employees;
#@substr/substring 截取字符串
注意: mysql中索引从1开始
#截取从指定索引处及后面所有字符
SELECT SUBSTR('abcdefghijkl',7) out_put;
#截取从指定索引处开始指定长度的字符
SELECT SUBSTR('abcdefghijkl',1,3) out_put;
#@instr 返回子串第一次出现的索引, 找不到返回-1
SELECT INSTR('abcdefghijkl','fgh') AS out_put;
#@trim 去除字符串前后的空格
SELECT TRIM('   -- ') AS out_put;
#去除前后指定的字符
SELECT TRIM('a' FROM 'aaaaaaaaaa123aa123aaaaaaaa') AS out_put;
#@lpad 用指定的字符实现左填充指定长度
SELECT LPAD('123',10,'*') AS out_put;
#@rpad 用右填充实现指定的长度
#@replace 替换
SELECT REPLACE('张前分666','张前分','张前锋') AS out_put;

#2. 数学函数
#@round 四舍五入
SELECT ROUND(1.6);
#小数点后保留两位
SELECT ROUND(1.57,2);
#@ceil 向上取整
SELECT CEIL(1.2);
#@floor 向下取整
SELECT FLOOR(1.2);
#@truncate 截断
SELECT TRUNCATE(1.6666666,1);
#@mod 取余
SELECT MOD(10,3);

#
#3. 日期函数
#now 返回当前系统的日期加时间
SELECT NOW();
#curdate 返回当前系统的日期, 不包含时间
SELECT CURDATE();
#curtime 返回当前时间, 不包含日期

#可以获取指定的部分: 年、月、日、小时、分钟、秒
SELECT YEAR(NOW());
SELECT YEAR('1998-1-1');
SELECT YEAR(hiredate) 年 FROM employees;
SELECT MONTH(NOW());
#英语月份
SELECT MONTHNAME(NOW());
#str_to_date 将字符串通过指定的格式转换成日期
SELECT STR_TO_DATE('1998-3-2','%Y-%c-%d') AS out_put;
#date_format 将日期转换成字符串
SELECT DATE_FORMAT (NOW(),'%Y年%m月%d日') AS out_put;

#4. 其他函数
SELECT VERSION();
SELECT DATABASE();
SELECT USER();

#5. 流程控制元素
#if 函数: 相当于三元运算符
SELECT IF(10>5,'big','small');
#case函数的使用情况一: switch case 的效果
/*
case 要判断的字段或表达式
when 常量1 then 要显示的值1或语句1
when 常量2 then 要显示的值2或语句2
.
.
else 要显示的语句n
end
*/
SELECT salary 原始工资,department_id,
CASE department_id
WHEN 30 THEN salary*1.1
WHEN 40 THEN salary*1.2
```

```
/*
SELECT salary 原始工资, department_id,
CASE department_id
WHEN 30 THEN salary*1.1
WHEN 40 THEN salary*1.2
WHEN 50 THEN salary*1.3
ELSE salary
END AS 新工资
FROM employees;
```

```
#case 函数使用情形二:
/*
case
when 条件1 then 要显示的语句1
when 条件2 then 要显示的语句2
...
else 要显示的语句n
end
*/
SELECT salary,
CASE
WHEN salary>20000 THEN 'A'
WHEN salary>15000 THEN 'B'
WHEN salary>10000 THEN 'C'
ELSE 'D'
END
AS 工资级别
FROM
employees;
```

`datediff(日期1, 日期2)`: 返回相差天数

`md5('字符')`: 返回该字符的md5加密形式

五、常见函数--分组函数

#分组函数

曰/*

功能: 用作统计作用, 又称为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 计算个数

特点1:

sum avg:	处理数值型	忽略null
max min:	处理任何类型	忽略null
count :	处理任何类型	忽略null

以上分组函数都忽略null值

特点2:

可以和`distinct`搭配实现去重运算

特点3 count 详细:

一般使用`count(*)`统计行数

特点4: 和分组函数一同查询的字段有限制
要求一同查询的字段是`group by` 后的字段+

*/

#简单的使用

```
SELECT SUM(salary) FROM employees;
SELECT MAX(salary) AS 最大值, COUNT(salary), AVG(salary) FROM employees;
```

#搭配`diatinct`去重

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

#count 函数的具体介绍

```
SELECT COUNT(salary) FROM employees;
```

#统计行数

```
SELECT COUNT(*) FROM employees;
```

```
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(1)` 与 `count(*)` 效率差不多, 都高于`count(字段)`

六、分组查询

#进阶5: 分组查询

曰/*

语法:

```
select 分组函数, 列 (要求出现在group by的后面)
from 表
//可以添加分组前查询where
```

```

#进阶5：分组查询
/*语法：
   select 分组函数，列（要求出现在group by的后面）
   from 表
   //可以添加分组前查询where
   group by 分组的列表
   //可以添加分组后查询having
   [order by 子句]

特点：
1.
      数据源          位置          关键字
分组前筛选： 原始表      group by 前      where
分组后筛选： 分组后的结果集      group by 后      having
      ⚠分组函数做条件肯定是在having子句中
      ⚠能用分组前筛选的优先使用分组前端选
2.
group by 子句支持单个字段分组，多个字段分组（逗号隔开，无顺序要求），表达式
或者函数（用的少），可以添加排序放在最后
*/

```

#例子：查询每个工种的最高工资

```

SELECT MAX(salary), job_id
FROM employees
GROUP BY job_id;

```

#添加分组前筛选条件

#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;

```

#添加分组后的筛选条件

#例：查询哪个部门的员工个数>2

```

SELECT COUNT(*), department_id
FROM employees
GROUP BY department_id
HAVING COUNT(*)>2;

```

#例

```

SELECT MAX(salary), job_id
FROM employees
WHERE commission_pct IS NOT NULL
GROUP BY job_id
HAVING MAX(salary)>12000;

```

#按表达式或函数分组

#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

```

SELECT COUNT(*), LENGTH(last_name) AS lenname
FROM employees
GROUP BY lenname
HAVING COUNT(*)>5;

```

#按多个字段分组

#例：查询每个部门每个工种的平均工资

```

SELECT AVG(salary), department_id, job_id
FROM employees
GROUP BY department_id, job_id;

```

#添加排序

#例：查询每个部门每个工种的员工的平均工资，并高到低排序

```

SELECT AVG(salary), department_id, job_id
FROM employees
WHERE department_id IS NOT NULL
GROUP BY department_id, job_id
HAVING AVG(salary)>10000
ORDER BY AVG(salary) DESC;

```

七、连接查询

```

#进阶6：连接查询
/*含义：又称多表查询，当查询的字段来自于多个表时使用。
笛卡尔乘积现象：表1有m行，表2有n行，总数m*n行情况
发生原因：没有有效的连接条件
如何避免：添加有效的连接条件

分类：
按年代分类：
sql92标准：仅支持内连接
sql99标准[推荐]：支持内连接+外连接（左外、右外）+交叉连接

按功能分：
内连接：
    等值连接
    非等值连接
    自连接
外连接：
    左外连接
    右外连接
    全外连接
交叉连接

*/

```

#一、sql92标准

#一、等值连接

```

    /**
     * 一、sql92标准
     * 一.等值连接
     */
    /*多表等值连接的结果为多表的交集部分
    on表连接，至少需要n-1个连接条件
    多表的顺序没有要求
    一般为表起别名
    可以搭配排序、分组、筛选

    */
    #例1：查询女生名对应男生名
    SELECT name,boyName
    FROM boys,beauty
    WHERE beauty.`boyfriend_id`=boys.`id`;

    #1.为表起别名
    /*
    提高语句的简洁度
    区分多个重名的字段

    注意：如果为表起了别名，则查询的字段就不能使用原来的表名去限定
    两个表的顺序可以调换
    */
    #查询员工名、工种号、工种名
    SELECT last_name,e.job_id,job_title
    FROM employees e,jobs j
    WHERE e.`job_id`=j.`job_id`;

    #2.加筛选
    #例：查询有奖金的员工名、部门名
    SELECT last_name,department_name
    FROM employees e,departments d
    WHERE e.`department_id`=d.`department_id`
    AND e.`commission_pct`IS NOT NULL;

    #3.加分组
    #例：查询每个城市的部门个数
    SELECT COUNT(*) 个数,city
    FROM departments d,locations l
    WHERE d.`location_id`=l.`location_id`
    GROUP BY city;

    #4.加排序
    #例：查询每个工种的工种名和员工的个数，按个数降序
    SELECT job_title,COUNT(*)
    FROM employees e,jobs j
    WHERE e.`job_id`=j.`job_id`
    GROUP BY job_title
    ORDER BY COUNT(*) DESC;

    #5.三表连接
    #例：查询员工名、部门名和所在的城市
    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`
    AND city LIKE's%'
    ORDER BY department_name;

    #添加等级表
    CREATE TABLE job_grades
    (grade_level VARCHAR(3),
    lowest_sal INT,
    highest_sal INT);

    INSERT INTO job_grades

```

#二、非等值连接

```

    #例：查询员工的工资和工资级别
    SELECT salary,grade_level
    FROM employees e,job_grades j
    WHERE j.`lowest_sal`<=e.`salary`<=j.`highest_sal`
    AND grade_level='A';

```

#三、自连接

```

    #例：查询员工名以及上级的名字
    SELECT e1.employee_id,e1.last_name,e2.employee_id,e2.last_name
    FROM employees e1,employees e2
    WHERE e1.`manager_id`=e2.`employee_id`;

```

```

    #sql99标准
    /*
    语法：
        select 查询列表
        from 表1 别名 [连接类型]
        join 表2 别名
        on 连接条件
        where 筛选条件
        .....
    */

```

```

from 表1 别名 [连接类型]
join 表2 别名
on 连接条件
where 筛选条件
group by 分组
having 筛选条件
order by 排序列表

内连接 (*) : inner
外连接
    左外 (*) ::left [outer]
    右外 (*) :right [outer]
    全外:   :full [outer]
交叉连接: cross [join]
注: [] 标记为可省略

*/
#一、内连接
/*
select 查询列表
from 表1 别名
inner join 表2 别名 on 连接条件
inner join 表3...;

分类:
等值
非等值
自连接

特点:
①添加排序、分组、筛选
②inner可以省略
③筛选条件放在where后面，连接条件放在on后面，提高分离性，便于阅读

*/
#1.等值连接
#案例: 查询名字中包含e的员工名和工种名（添加筛选）
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%';
#查询员工名、部门名、工种名，并按部门降序
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 department_name DESC;

#2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;

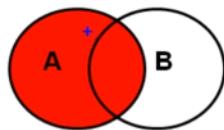
#3.自连接
#查询员工名字与上级名字
SELECT e1.last_name AS 员工,e2.last_name 上级
FROM employees e1
INNER JOIN employees e2
ON e1.manager_id=e2.employee_id;

#二、外连接
/*
用于查询一个表中有，另一个表中没有的记录
特点:
①外连接的查询结果为主表中的所有记录
    如果有匹配的显示匹配的值
    如果没有匹配的则显示null
    外连接查询结果=内连接结果+主表中有而从表没有的记录
②左外连接，left join 左边的是主表
    右外连接，right join右边的是主表
③左外和右外交换两个表的顺序，可以实现同样的结果
④全外连接=内连接的结果+表1中有但表2没有的+表2中有但是表1中没有的
*/
#案例: 查询哪个部门没有员工
#1.左外
SELECT d.*,e.employee_id
FROM departments d
LEFT OUTER JOIN employees e
ON d.department_id=e.department_id
WHERE e.employee_id IS NULL;

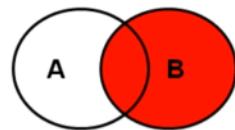
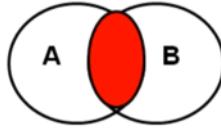
#2.全外
USE girls
SELECT b.* ,bo.*
```

```
#2.全外
USE girls
SELECT b.* , bo.*
FROM beauty b
FULL OUTER JOIN boys bo
ON b.biyfriend_id=bo.id;
```

#三、交叉连接(实现笛卡尔乘积)
 SELECT b.* , bo.*
 FROM beauty b
 CROSS JOIN boys bo;

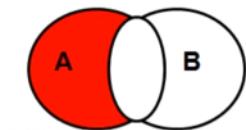


SELECT <select_list>
 FROM A
 LEFT JOIN B
 ON A.key=B.key

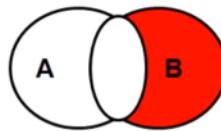


SELECT <select_list>
 FROM A
 RIGHT JOIN B
 ON A.key=B.key

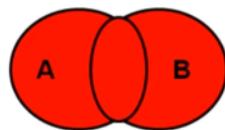
SELECT <select_list>
 FROM A
 INNER JOIN B
 ON A.key=B.key



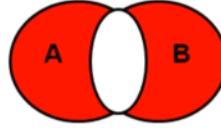
SELECT <select_list>
 FROM A
 LEFT JOIN B
 ON A.key=B.key
 WHERE B.key is null;



SELECT <select_list>
 FROM A
 RIGHT JOIN B
 ON A.key=B.key
 WHERE A.key is null;



SELECT <select_list>
 FROM A
 FULL JOIN B
 ON A.key=B.key



SELECT <select_list>
 FROM A
 FULL JOIN B
 ON A.key=B.key
 WHERE A.key is null
 OR B.key is null;

八、子查询

#子查询

/*

含义：

出现在其他语句中的select语句，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：
 只支持标量子查询
 from后面：
 支持表子查询
 where或having后面： ***
 标量子查询 *
 列子查询 *
 行子查询

```

    标量子查询*
    列子查询 *
    行子查询

    exists后面 (相关子查询)
    表子查询

按结果集行列数不同:
    标量子查询 (结果集只有一行一列)
    列子查询 (结果集只有一列多行)
    行子查询 (结果集只有一行多列)
    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*
2.列子查询 *
3.行子查询

标量子查询特点:
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，搭配着单行操作符使用
>< >= <= <>
④子查询的执行优先于主查询执行，主查询的条件用到了子查询的结果
子查询结果必须是一行一列

列子查询一般搭配多行操作符使用
    in/not in 等于列表中的任意一个
    any/some(少) 和子查询返回的某一个比较,满足任意一个
    all (少) 和子查询返回的所有值比较,需全满足
*/
#1.标量子查询
#例：谁的工资比Abel高？
SELECT *
FROM employees
WHERE salary>(
    SELECT salary
    FROM employees
    WHERE last_name='Abel'
);
#例返回工资工资最少的员工的last_name,job_id和salary
#①查最低工资
SELECT MIN(salary)
FROM employees;
#②查询last_name,job_id,salary,salary=①
SELECT last_name,job_id,salary
FROM employees
WHERE salary=(  

    SELECT MIN(salary)
    FROM employees
);
#2.列子查询（多行子查询）
#案例：返回location_id是1400或1700的部门中所有员工姓名
#①查询location_id是1400或1700的部门编号
SELECT DISTINCT department_id
FROM departments
WHERE location_id IN(1400,1700);
#②查询员工姓名，要求部门号是①列表中的一个
SELECT last_name
FROM employees
WHERE department_id IN(  

    SELECT DISTINCT department_id
    FROM departments
    WHERE location_id IN(1400,1700)
);
#3.行子查询（结果集一行多列或多行多列）
#例：查询员工编号最小并且工资最高的员工信息
SELECT *
FROM employees
WHERE (employee_id,salary)=(
    SELECT MIN(employee_id),MAX(salary)
    FROM employees
);
普通方式
#①查询员工编号最小
SELECT MIN(employee_id)
FROM employees;
#②查询最高工资
SELECT MAX(salary)
FROM employees;
#③查询员工信息
SELECT *
-- ...

```

```

SELECT MAX(salary)
FROM employees;
#①查询员工信息
SELECT *
FROM employees
WHERE employee_id=
    (SELECT MIN(employee_id)
     FROM employees
    )
AND salary=
    (SELECT MAX(salary)
     FROM employees
    );
;

#二、放在select后面
#例查询每个部门的员工的个数
SELECT d.*,
       (SELECT COUNT(*)
        FROM employees e
        WHERE e.department_id=d.department_id
       ) AS 个数
     FROM departments d;

#三、from后面
/*
将子查询结果冲当为一张表，要求必须起别名
*/
#例：查询每个部门的平均工资的工资等级
#①查平均工资
SELECT AVG(salary),department_id
FROM employees
GROUP BY department_id;
#②连接①的结果和job_grades表
SELECT ag_dep.*,g.grade_level
FROM (
    SELECT AVG(salary) ag,department_id
    FROM employees
    GROUP BY department_id
   ) AS ag_dep
INNER JOIN job_grades g
ON ag_dep.ag BETWEEN g.lowest_sal AND g.highest_sal;

#四、放在exists后面（相关子查询）
/*
语法：
exists(完整的查询语句)
或
not exists(完整的查询语句)
结果：
1或0
*/
SELECT EXISTS(SELECT employee_id FROM employees WHERE salary=30000);
#例查询有员工的部门名
SELECT department_name
FROM departments d
WHERE EXISTS(
    SELECT *
    FROM employees e
    WHERE d.department_id=e.department_id
   );

```

sql执行顺序 (博客收藏)

from			获取虚拟数据源表
join			与第一表生成笛卡尔乘积表
on			筛选各个行将满足条件的筛选出来形成虚拟表

on		筛选各个行将满足条件的筛选出来形成虚拟表
where		再次筛选形成虚拟表
group by		形成分组后虚拟表
having		再次筛选
select		选出要查看的部分形成虚拟表显示
distinct		移除相同行，形成新虚拟表
order by		排序形成新表
limit		分页显示表

/*

语法：

select	查询列表
from	表1 别名
连接类型	join 表2 别名
on	连接条件
where	筛选条件
group by	分组列表
having	筛选条件
order by	排序列表
limit	起始条目，条目数

*/

九、分页查询

#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 分组后筛选
order by 排序后字段
limit offset,size;
```

offset要显示条目的起始索引（从0开始）
size 要显示的条目个数

特点：

- ① limit子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;

*/

#例查询第11到25条信息

```
SELECT *
FROM employees
LIMIT 10,15;
```

#例有奖金的员工信息，且工资最高的前10名显示

```
SELECT *
FROM employees
LIMIT 10,15;
#例有奖金的员工信息，且工资最高的前10名显示
SELECT *
FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY salary DESC
LIMIT 10;
```

十、联合查询

#联合查询

```
/*
union 联合：将多条查询语句的结果合并成一个结果
语法：
    查询语句1
    union
    查询语句2
    union
    ...
应用场景：
要查询的结果来自于多个表，且多个表没有连接关系，但查询的信息一致时

特点：
    ◎要求多条查询语句的查询列数一致
    ◎要求多条查询语句查询的每一列的类型和顺序最好一致
    ◎union默认会自动去重，不想去重使用union all

*/
#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%'
UNION
SELECT * FROM employees WHERE department_id>90;
```

五、DML语言

2021年9月11日 22:02

```
#DML语言
/*
数据操作语言:
插入: insert
修改: update
删除: delete
*/



#一、插入语句
#方式1
/*
语法:
    insert into 表名(列名,...)
    values(值1,...),values(...),...
    可以插入多行
*/
#1.插入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,'蔡依林','女','1990-4-23','41111114000',NULL,2);
SELECT * FROM beauty;
#2.不可以为null的列必须插入值, 可以为null的列如何插入值?
#列名写上, 值为null; 或列名不写, 值null也不写
INSERT INTO beauty(id,NAME,sex,borndate,phone,boyfriend_id)
VALUES (15,'吴亦凡','女','1991-5-23','11111114000',2);
#3.列的顺序调换
INSERT INTO beauty(NAME,sex,id,phone)
VALUES ('目目','女',16,'22222222222');
#4.列数和值的个数必须一致
#5.可以省略列名, 默认为所有列, 顺序和表中一致
INSERT INTO beauty
VALUES (17,'蔡依林','女','1990-4-23','41111114000',NULL,2);

#插入方式2
/*
语法:
    insert into 表名
    set 列名1=值1, 列名2=值2...
*/



#两种方式的对比:
#1.方式1支持插入多行, 方式二不支持
#2.方式1支持子查询, 将子查询的结果插入指定表, 方式二不支持
INSERT INTO beauty(id,NAME,phone)
SELECT 26,'宋茜','120';



#二、修改语句
/*
    修改单表的记录*
语法:
    update 表名
    set 列=新值, 列=新值...
    where 筛选条件;

    修改多表的记录[补充]
语法: (sql92)
    update 表1 别名, 表2 别名
    set 列=值, ...
    where 连接条件
    and 筛选条件;
语法: (sql99)
    update 表1 别名,
    inner|left|right join 表2 别名
    on 连接条件
    set 列=值, ...
    where 筛选条件;
*/
#修改单表的记录
#修改beauty表中姓吴的电话改为110
UPDATE beauty
```

```
/*  
#修改单表的记录  
#修改beauty表中姓吴的电话改为110  
UPDATE beauty  
SET phone='110'  
WHERE NAME LIKE '%吴%';  
SELECT * FROM beauty;  
  
#修改多表的记录  
#例修改张无忌的女朋友的手机号为114  
UPDATE boys bo  
INNER JOIN beauty b  
ON bo.`id`=b.`boyfriend_id`  
SET b.`phone`='114'  
WHERE bo.`boyName`='张无忌';
```

#三、删除语句

```
/*  
方式一: delete  
语法:  
①单表的删除*  
    delete from 表名  
    where 筛选条件;  
②多表的删除[补充]  
    [sql99]  
    delete 表1的别名, 表2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
```

方式二: truncate

```
语法:  
    truncate table 表名;
```

两种方式区别:

```
①delete 可以加where条件, truncate不能加  
②truncate删除, 效率高一些  
③假如要删除的表中有自增长列, 如果使用delete删除后,  
再插入数据, 自增长列的值从断点开始;  
而使用truncate删除后, 再插入数据, 自增长列的值从1开始。  
④truncate删除没有返回值, delete有返回值, 返回受影响行数  
⑤truncate删除不能回滚, delete删除可以回滚  
*/
```

```
#方式一: delete  
#单表的删除  
#删除手机号以9结尾的女生信息  
DELETE FROM beauty WHERE phone LIKE '%0';
```

```
#方式二: truncate语句, 删除表中全部数据  
TRUNCATE TABLE boys;
```

六、DDL语言

2021年9月12日 13:50

一、数据定义语言

```
#DDL
/*
数据定义语言

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: alter
删除: drop

*/
#一、库的管理
#1.库的创建
/*
语法:
create database [if not exists]库名;
*/
#案例: 创建book库
CREATE DATABASE IF NOT EXISTS book;
#2.库的修改
#库名尽量别改, 需要改的话关闭服务去data改名字
#更改库的字符集, 不指定会使用默认字符集
ALTER DATABASE book CHARACTER SET gbk;

#库的删除
DROP DATABASE IF EXISTS book;

#二、表的管理
#1.表的创建**
/*
create table [if not exists]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]
);

*/
#例创建history表
CREATE TABLE history(
    id INT,
    bName VARCHAR(20),
    price DOUBLE,
    authorId INT,
    publicDate DATETIME
);
DESC history;
#创建表author
CREATE TABLE author(
    id INT,
    au_name VARCHAR(20),
    nation VARCHAR(10)
);

#2.表的修改
/*
语法:
alter table 表名 add/drop/change/modify/rename to    column 列名 [列类型 约束];
*/
#①修改列名
ALTER TABLE history CHANGE COLUMN publicDate pubDate DATETIME;
#②修改列的类型或约束
ALTER TABLE history MODIFY COLUMN pubDate TIMESTAMP;
#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 ADD COLUMN annual DOUBLE;
#①删除列
ALTER TABLE author DROP COLUMN annual;
#②修改表名
ALTER TABLE author RENAME TO book_author;

#3.表的删除
DROP TABLE IF EXISTS book_author;
SHOW TABLES;

#4.表的复制
#可以跨库复制，需要使用 库.表名
INSERT INTO author
VALUES(1,'村上中树','日本'),
(2,'莫言','中国'),
(3,'金庸','中国');
#①仅仅复制表的结构
CREATE TABLE copytable LIKE author;
#②复制表的结构和数据
CREATE TABLE copy2
SELECT * FROM author;
#③复制部分数据
CREATE TABLE copy3
SELECT id,au_name
FROM author
WHERE nation='中国';
#④仅仅复制某些字段,无数据
CREATE TABLE copy4
SELECT id,au_name
FROM author
WHERE 0;

```

添加列补充：

后面可以跟[frist|after 字段名]，将字段加在指定位置（用的少）

二、数值类型

#常见的数据类型

```

/* 数值型
   整形
   小数:
     定点数
     浮点数

  字符型
    较短文本: char、varchar
    较长文本: text、blob(较长的二进制数据)

  日期型
  */

#一、整形
/* 分类:
   tinyint    smallint   mediumint   int/integer   bigint
   1          2           3           4           8
特点:
①默认为有符号，如果想使用无符号，需要添加unsigned [int unsigned]
②如果插入的数值超出了整形的范围，会报错。
③如果不设置宽度，会有默认的宽度。
搭配 num1 int(7) zerofill 位数不够会左补0，超过直接显示原数
*/
#二、小数:
/* 1.浮点型
float(M,D)
  4
double(M,D)
  8字节
  */

```

```

float(M,D)
    4
double(M,D)
    8字节
2.定点型
dec(M,D) /
decimal(M,D)
    M+D字节

特点:
① M: 整数部位+小数部位
D: 小数部位
如果超过则报错(版本), 小数mysql 5.7, 5舍6入, 整数超过报错out of range
② M,D都可以省略
如果是decimal, 则M默认10, D默认为0
如果是float和double, 则会根据插入的数值的精度来决定精度
③ 定点型的精度较高, 如果要求插入的数值的精度较高如货币运算等考虑使用。
*/

```

#原则
/*
所选择的类型越简单越好, 节省空间
*/

#三、字符型

```

/*  
较短文本:  
字符串类型    最多字符数    描述级存储需求    特点:          空间    效率  
char(M)        M            0<=M<=255整数    固定长度的字符    费        高  
varchar(M)     M           0-65535整数      可变长度的字符    省        低  
char中M可以省略, 默认1; varchar M不可省略。  
  
较长文本:  
test  
blob(较大的二进制数据)  
  
enum、set、binary、varbinary了解  
*/

```

#四、日期型

```

/*  
date          只保存日期  
time          只保存时间  
year          只保存年  
  
datetime      保存日期+时间  
timestamp     保存日期+时间  
  
特点:  
      字节          范围          时区等的影响  
datetime      8             1000-9999    不受  
timestamp     4             1970-2038    受  
*/  
SHOW VARIABLES LIKE 'time_zone'; #查看时区  
SET time_zone='+9:00'; #设置时区

```

三、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性和可靠性

分类: 六大约束
not null: 非空, 用于保证该字段的值不能非空
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

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
比如员工表的部门编号, 员工的工种编号

添加约束的时机:

- 创建表时
- 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

主键和唯一的对比:

	保证唯一性	是否	字段数量	组合
主键	√	✗	最多一个	√(不推荐)
唯一	√	√	多个	√(不推荐)

多个列可以组合成一个主键, 都重复才算重复

CONSTRAINT pk PRIMARY KEY(id, seat) #组合主键

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一致或兼容, 名称无要求
3. 要求主表中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。

*/

#一、创建表时添加约束

#1. 添加列级约束

/*

直接在字段名和类型后面追加 约束类型 即可。

只支持: 默认、非空、主键、唯一

一个字段需要多个约束时, 直接写在字段 空格隔开

*/

CREATE DATABASE students;

USE students;

CREATE TABLE stuinfo (

id INT PRIMARY KEY,
stuname VARCHAR(20) NOT NULL,
gender CHAR(1),
seat INT UNIQUE,
age INT DEFAULT 18

);

CREATE TABLE major (

id INT PRIMARY KEY,
majorName VARCHAR(20)

);

DESC stuinfo;

SHOW INDEX FROM stuinfo; #查看表中所有的索引, 主键、外键、索引

#2. 添加表级约束

/*

语法: 在各个字段的最后面

[constraint 约束名] 约束类型(字段名)

```

#二. 添加约束到表
/*
语法: 在各个字段的最后面
[constraint 约束名] 约束类型(字段名)

通常写法:
表级约束只使用外键, 约束名: fk_表1_表2
*/
DROP TABLE IF EXISTS stuinfo;
CREATE TABLE stuinfo(
    id INT,
    dtuname VARCHAR(20),
    gender CHAR(1),
    seat INT,
    age INT,
    majorid INT,
    CONSTRAINT pk PRIMARY KEY(id), #主键
    CONSTRAINT uq UNIQUE(seat), #唯一键
    CONSTRAINT fk_stuinfo_major
        FOREIGN KEY(majorid) REFERENCES major(id) #外键
);

```

```

#二、修改表时添加约束
/*
列级约束:
alter table 表名 modify column 字段名 字段类型 新约束类型;
表级约束:
alter table 表名 add 约束类型(字段名) [外键的引用];
*/
#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 DEFAULT 18;
#3.添加主键
ALTER TABLE stuinfo MODIFY COLUMN id INT PRIMARY KEY;
#4. 添加唯一键
#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);

```

```

#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 INDEX seat;
#5.删除外键
ALTER TABLE stuinfo DROP FOREIGN KEY fk_stuinfo_major;

```

```

#外键删除字段补充
#方式一: 级联删除
#删除主表数据时删除对应从表数据
ALTER TABLE stuinfo ADD COLUMN fk_stu_major
FOREIGN KEY(majorid) REFERENCES major(id) ON DELETE CASCADE;
#方式二: 级联置空
#删除主表数据时将对应外键null
ALTER TABLE stuinfo ADD COLUMN fk_stu_major
FOREIGN KEY(majorid) REFERENCES major(id) ON DELETE SET NULL;

```

主键约束、唯一约束和外键约束，都会生成索引，不能直接修改表的列属性来删除

主键约束、唯一约束和外键约束，都会生成索引，不能直接修改表的列属性来删除
需要使用drop

四、标识列

```
#标识列
/*
又称为自增长列
含义：
可以不用手动插入值，系统提供默认的序列值，起始值为1

特点：
①必须和一个key搭配
②一个表最多只有一个自增长列
③类型只能是数值型
④
SET auto_increment_increment=3;#设置步长

手动插入第一个可以设置起始索引：VALUES(10,'join')
从10开始。
*/
#一、创建表时设置标识列
CREATE TABLE tab_identity(
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(20)
);
INSERT INTO tab_identity VALUES(NULL,'join');//使用null占位置

SHOW VARIABLES LIKE '%auto_increment%';

#二、修改表时设置标识列
#添加
ALTER TABLE tab_identity MODIFY COLUMN id INT PRIMARY KEY AUTO_INCREMENT;
#删除
ALTER TABLE tab_identity MODIFY COLUMN id INT ;
```

七、TCL语言

2021年9月15日 13:02

一、事务

```
#TCL
/*
Transaction Control Language 事务控制语言
事务：
事务由单独单元的一个或者多个SQL语句组成，在这个单元中，每个
MySQL语句是相互依赖的。

事务的ACID(acid)属性：
①原子性(Atomicity)
原子性指事务是一个不可分割的工作单位
②一致性(Consistency)
事务必须使一个数据从一个一致性状态变换到另一个一致性状态。
③隔离性(Isolation)
事务的隔离性是指一个事务的执行不能被其他并发事务干扰。
④持久性(Durability)
持久性是指一个事务一旦被提交，对数据库的改变就是永久性的。
```

事务的创建：
隐式事务：事务没有明显的开启和结束的标记：
比如：insert、update、delete语句

显式事务：
事务具有明显的开启和结束的标记
前提：必须先设置自动提交功能为禁用

步骤1：开启事务
set autocommit=0;
start transaction; 可选的
步骤2：编写事务中的sql语句(select/insert/update/delete)
语句1;
语句2;
...
步骤3：结束事务
commit; 提交事务
rollback; 回滚事务，

savepoint 节点名; 设置保存点;
 rollback to 节点名; 回滚到保存点

```
/*
#事务创建的例子
#开启事务
SET autocommit=0;
#编写语句
balance=500 WHERE username='用户1';
UPDATE account SET balance=1500 WHERE username='用户2';
#结束事务
COMMIT;
```

```
/*
事务并发问题介绍：
并发事务：多个事务同操作一个数据库的相同时数据时
脏读：
一个事务读取了被另一个事务更新但未提交的字段。
不可重复读：
对于两个事务T1, T2, T1读取了一个字段，单后T2更新了该字段,
之后，T1再次读取同一个字段，值就不同了。
```

幻读：一个事务读取了其他事务还没有提交的数据，读到其他事务插入的数据
delete和truncate在事务使用时的区别
delete可以回滚， truncate不能回滚。

八、其他

2021年9月16日 20:16

一、视图

#视图

/*

含义：虚拟表，和普通表一样使用
mysql 5.1版本出现的新特性，是通过表动态生成的数据

使用视图的好处：

- ①重用sql语句。
- ②简化复杂的sql语句，不必知道它的查询细节。
- ③保护数据，提高安全性。

1. 创建视图：

```
create view 视图名  
as  
查询语句；
```

2. 使用视图

和表的使用一样，当成表使用

3. 视图的修改

方式一：

```
create or replace view 视图名  
as  
查询语句；
```

方式二：

```
alter view 视图名  
as  
查询语句
```

4. 删除视图

```
drop view 视图1, 视图2, ...;
```

5. 查看视图

```
desc 视图名;  
或  
show create view 视图名;
```

6. 视图的更新

对视图进行增删改都是在修改原表中的数据

注意：视图一般用于查询的，而不是更新的，所以具备以下特点的视图都不允许更新

- ①包含分组函数、group by、distinct、having、union。

②join

③常量视图

④where后的子查询用到了from中的表

⑤用到了不可更新的视图

7. 视图和表的对比

	关键字	是否占用物理空间	使用
视图	create view	占用较小，只保存sql逻辑	一般用于查询
表	create table	保存实际的数据	增删改查

```
*/  
CREATE VIEW v2  
AS  
SELECT * FROM employees WHERE 100<employee_id<110;  
  
SELECT * FROM v2 WHERE employee_id=103;  
  
DROP VIEW v1,v2;
```

二、变量

```
#变量
/*
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

```
/*
全局变量
```

作用域：服务器每次启动将为所有的全局变量赋初始值，
针对所有的会话（连接）有效，但不能跨重启。

会话变量：

作用域：仅仅针对当前会话（连接）无效

变量由系统提供，不是用户定义，属于服务器层面
使用的语法：

1. 查看所有的系统变量

```
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

```
set @@global|[session] [.]. 系统变量名=值;
```

注意：全局级别加global

会话级别session，不写默认session

```
*/
```

```
SHOW GLOBAL VARIABLES;
```

```
SHOW SESSION VARIABLES;
```

```
SELECT @@global.autocommit;
```

#二、自定义变量

```
/*
说明：变量是用户自定义的，不是系统的
```

使用步骤：

声明

赋值

使用（查看、比较、运算等）

用户变量

作用域：针对当前会话（连接）有效，等同于会话变量的作用域
应用在任何地方，也就是begin end 里面或外面。

1. 声明并初始化

赋值操作符 =,:=

```
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）

```
set @用户变量名=值;
```

或

```
set @用户变量名 :=值;
```

或

```
select @用户变量名:=值;
```

或

```
select 字段 into 变量名
```

from 表;

3. 使用（查看）

```
select @用户变量名;
```

局部变量

作用域：仅在定义它的begin end中有效

局部变量

作用域：仅在定义它的begin end中有效

```
1. 声明  
declare 变量名 类型;  
declare 变量名 类型 default 值;  
2. 赋值  
set 局部变量名=值;  
或  
set 局部变量名 :=值;  
或  
select @局部变量名:=值;  
或  
select 字段 into 局部变量名  
from 表;  
3. 使用  
select 局部变量名;  
*/
```

三、存储过程

#存储过程和函数

```
/*
存储过程和函数：类似于java中的方法
```

#存储过程

```
/*
含义：一组预先编译好的sql语句的集合，理解成批处理语句
好处：
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连接的连接次数，提高了效率。
```

语法

```
##创建语法：  
create procedure 存储过程名(参数列表)  
begin  
    存储过程体(一组合法有效的sql语句)  
end
```

注意：

o参数列表包含三部分
参数模式 参数名 参数类型

参数模式

```
in      :该参数可以作为输入，调用方的传入值  
out     :该参数可以作为输出，也就是可以作为返回值  
inout   :既可以作为输入又可以作为输出
```

o如果存储过程体只有一句话，begin end 可以省略
存储过程体中每条sql语句结尾必须加分号
存储过程的结尾可以使用 delimiter 重新设置结束标记
语法：sqlyog不支持
delimiter 结束标记
例： delimiter \$

##调用语法：

```
call 存储过程名(实参列表);
```

##删除存储过程

语法：

```
drop procedure 存储过程名
```

##查看存储过程的信息

```
show create procedure myp2;
```

```

##查看存储过程的信息
show create procedure myp2;

##没有修改方法

/*
#1.空参列表
#例：插入到admin表中5条记录
DELIMITER $
CREATE PROCEDURE myp1()
BEGIN
    INSERT INTO admin(username,password) VALUES('join1','0000');
    INSERT INTO admin(username,password) VALUES('join2','0000');
    INSERT INTO admin(username,password) VALUES('join3','0000');
    INSERT INTO admin(username,password) VALUES('join4','0000');
    INSERT INTO admin(username,password) VALUES('join5','0000');
END $

#调用
CALL myp1()$


#2.带in的
#创建存储过程实现，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;#初始化变量并赋值
    SELECT COUNT(*) INTO result#赋值
    FROM admin
    WHERE admin.username=username
    AND admin.password=PASSWORD;
    SELECT IF(result>0,'成功','失败');#变量使用
END $


#3.带out的,如果返回多个时，赋值时into左右对应，用，隔开
#例：根据女神名，返回对应的男神名
CREATE PROCEDURE myp4(IN beautyName VARCHAR(20),OUT boynam VARCHAR(20))
BEGIN
    SELECT bo.boyName INTO boynam
    FROM boys bo
    INNER JOIN beauty b ON bo.id=b.boyfriend_id
    WHERE b.name=beautyName;
END $

#调用
SET @bname$;
CALL myp4('小昭',@bname) $;
SELECT @bname;

#4.创建inout模式参数的存储过程
#例：传入a,b两个值，最终a, b都翻倍并返回
CREATE PROCEDURE myp7(INOUT a INT,INOUT b INT)
BEGIN
    SET a=a*2;
    SET b=b*2;
END $

SET @m=10$;
SET @n=20$;
CALL myp7(@m,@n)$;
SELECT @m,@n$;

```

四、函数

#函数

```

/*
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连接的次数

```

区别：

存储过程：可以有0个返回，也可以有多个返回，
 适合左批量插入、批量更新
 函数：有且只有一个返回，适合做外键数据后返回一个结果

存储过程：可以有0个返回，也可以有多个返回，
适合左批量插入、批量更新
函数：有且只有一个返回，适合做处理数据后返回一个结果

#1. 创建语法
create function 函数名(参数列表) returns 返回类型
begin
 函数体
end

注意：
①参数列表 包含两部分：
参数名 参数类型
②函数体
肯定会有return语句，位置随意，通常放在最后。

return 值；
③函数体只一条语句时，可以省略begin 和 end
④使用 delimiter 语句设置结束标记

#2. 调用语法
select 函数名(参数列表) #执行函数并显示返回值

#3. 查看函数
show create function myf1;

#4. 删除函数
drop function myf1;
|
*/

#无参有返回
#例：返回公司的员工个数
DELIMITER \$
CREATE FUNCTION myf1() RETURNS INT
BEGIN
 DECLARE c INT DEFAULT 0;
 SELECT COUNT(*) INTO c
 FROM employees;
 RETURN c;
END \$

SELECT myf1();

USE myemployees;

五、流程控制结构

#流程控制结构
曰/*
顺序结构：
程序从上往下一次执行
分支结构：
程序从两条或多条路径中选择一条执行
循环结构：
满足条件基础上，重复执行一段代码

*/
#一、分支结构
#1.if函数
曰/*
功能：实现简单的双分支
语法：
IF(表达式1, 表达式2, 表达式3)
相当于三元运算符
*/

#2.case结构
曰/*
情况1：
类似switch语句
*/

```

#2.case结构
/*
情况1:
类似switch语句
语法:
    case 变量|表达式|字段
        when 要判断的值 then 返回的值1或语句;
        when 要判断的值 then 返回的值2或语句;
        ...
        else 要返回的值n或语句;
    end [case];
情况2:
类似多重if语句
语法:
    case 变量|表达式|字段
        when 要判断的条件1 then 返回的值1或语句;
        when 要判断的条件2 then 返回的值2或语句;
        ...
        else 要返回的值n或语句;
    end [case];
特点:
① 可以作为独立的表达式，嵌套在其他语句中使用，放在任何地方，begin end 中或 begin end 外面；可以作为独立的语句使用，只能放在begin end 中；
② 如果when中的值满足或条件成立，则执行对应的then后面的语句，并且结束case。如果都不满足，则执行else中的语句或值
③ else可以省略，如果else省略了，并且所有的when条件都不满足，则返回null
*/

```

```

#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;
...
[else 语句n;]
end if;

应用在begin end中
*/
CREATE FUNCTION test_if(score INT) RETURNS CHAR
BEGIN
    IF score>=90 AND score<=100 THEN RETURN 'A'
    ELSE RETURN 'B';
    END IF;
END $
```

```

#二、循环结构
/*
分类：
while、loop、repeat

循环控制：
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
/*
[标签名:]while 循环条件 do
    循环体;
end while[标签名];

*/
#2.loop
/*
语法：
[标签名:]loop
    循环体;
end loop[标签名];

可以用来模拟简单的死循环
搭配leave跳出循环
*/

```

```

#3.repeat
/*
语法：
[标签名:]repeat
    循环体;
until 结束循环的条件
end repeat[标签名];

*/
#例：批量插入，根据次数插入到admin表中多条记录

```

```
end repeat[标签名];  
/*  
#例：批量插入，根据次数插入到admin表中多条记录  
CREATE PROCEDURE pro_hwile1(IN insertCount INT)  
BEGIN  
    DECLARE i INT DEFAULT 1;  
    a:WHILE i<=insertCount DO  
        INSERT INTO admin(username,`password`)  
        VALUES(CONCAT('Rose',i),'666')  
        SET i=i+1;  
        IF i=3 THEN ITERATE a;  
        END IF;  
        IF i=20 THEN LEAVE a;  
        END IF;  
    END WHILE a;  
END $  
  
CALL pro_hwile1(100)$
```