**MySQL数据库**

mysql：是一款开源的数据库软件，属于c/s架构，需要安装客户端和服务器

oracle：是一款商用的数据库软件，主要在大型项目中用的比较多

mysql与oracle同属于甲骨文公司旗下的

1. **mysql 安装介绍：**
2. **选择（Setup Type:）**

Developer default:开发人员模式（可以选这个，耗时）

sever only：只安装服务端

client only：只安装客户端

full：全安装

custom：自定义安装

1. **配置端口：**

mysql的默认端口是3306；【oracle的默认端口号：1521；sql server:1433】即使入口

提前检查3306端口是否被占用；如果被占用建议换一个

查询端口（pid）:netstat -ano——或者在任务管理器中查询

1. **账户设置**

配置root账户密码：root/密码（51testing）

配置一个自己的账户及密码：用户名/密码（ljh/147369ljh）

注意：安装mysql是需要依赖于.net framework 4.5.2(NDP452-KB2901907-x86-x64-AllOS-ENU.exe)

安装mysql还需要依赖于：visualC++（建议安装类型，选择：development 开发人员模式）

验证是否安装成功，有两种登录方式：

1. **通过命令行**

打开命令行：cmd

mysql -h 服务器设备 -P 端口号 -u 用户名 -p密码

-h host，即：连接哪一台设备上的mysql； 如果上本机：localhost 127.0.0.1

也可以不加-h选项； 默认连的就是自己机器上的mysql

如果是远端机器（另一台机器）： 那台机器的ip地址

-P port mysql的入口； 如果是默认的3306 可以不加-P选项

如果不是3306； 就必须加 -P

-u user 用户名

-p password 密码； 注意：密码直接在-p后面，没有空格

1. 连接自己的mysql：mysql -h localhost -P 3308 -u ljh -p147369ljh

show databases;显示当前mysql中所有的数据库； 默认自带4个数据库；mysql会自动维护的；

create database 数据库名字;创建数据库； 数据库名字自定义； 在实际工作中：数据库名字，一般都是项目名；

1. 连接远端的mysql（工作中一般都是这种情况）

mysql -h 远端设备的ip -P 远端设备mysql的端口号 -u 远端设备mysql的用户名 -p远端设备mysql的密码

1. **通过工具来连接登录mysql；**

sqlyog工具：是专业用来连接mysql 的；需要配置5个信息

需要配置5个信息:

主机： 设备的ip地址； 本机： localhost

端口号： mysql的端口号 (自己定义的端口或者使用默认的)

用户名：root

密码：root

数据库名： 自己数据库的名字

配置环境变量：在命令行找到mysql 的命令

配置环境变量：计算机--->右键-->属性-->高级系统设置-->环境变量-->系统变量（每个用户登录这个电脑都能使用到到mysql）---->找到path 这个变量，往这个里面添加mysql命令的目录即可；

1. 概念：

数据库软件： 是用来管理数据库的；可以创建数据库，删除数据库等等； 例如：mysql

即：一个数据库软件中可以创建包含多个数据表；

其他的数据库软件： oracle ，SQL Server, DB2等等

数据库：是用来管理存储数据的； 例如：wumeiDB

在表中来存储管理数据

数据：是存储中数据库中的东西

为什么要学习数据库？ 即：数据库跟软件，跟软件测试工作有什么关系

每个软件基本上都需要数据库； 因为每个软件都有数据的；

测试软件；就需要测试数据；例如:测试淘宝的查询商品功能,必须得准备商品数据；

测试人员： 会使用数据库； 能够从数据库中获取到数据（查询语句-日志相关）

开发人员： 会使用数据库

DBA (Database Administrator):数据库管理员； 管理数据库的； 会使用数据库，还能够维护，懂数据库的原理；懂数据库的结构；

1. **SQL语言：**

SQL:结构化的查询语言 Structured Query Language

SQL中主要包含4种语句：

1. DQL：（Data Query Language）数据查询语言； select语句
2. DML:（Data Manipulation Language）数据操作语言； 包含3个语句；
3. DDL:（Data Definition Language）数据定义语言；包含3个语句；
4. DCL:（Data Control Language）数据控制语言； 主要包含2个语句
5. **DQL数据查询语言： data query language**

功能： 从表中获取查询自己想要的数据；

在数据库中：数据都是存放在二维表格中的；一行就表示一条数据； 行又叫做：记录， 元组等等 列又叫做 属性；

1. **select \* from 表名:**

——查询表中所有的数据（全部信息）； \*表示所有列

1. **select 列名1,列名2,.......from 表名;**

——查询部分列的信息；

**select destinct 列名1,列名2,.......from 表名;**

————查询部分列时去重操作；

1. **select 列1名 别名, 列2名 , 列3名 别名 from 表名;**

——给列起别名

1. **select \* from 表名 where 查询条件;**

**select 列1名,列2名,.. from 表名 where 查询条件;**

查询条件具备的格式：

where 列名 运算符 搜索值

列名： 必须是表中存在的列

运算符： 重点要讲

搜索值： 由查询的人来决定

**运算符：=、like、!=、%、\_ （占位符）**

= 表示精确查找运算符：精确匹配；将跟搜索值长的一模一样的数据才会查询出来

like 表示模糊查找运算符：将跟搜索值长的像的数据查询出来；即：搜索值输入部分也能查询到

like 必须结合通配符一起使用才可以；

% 表示：0位或者多位任意字符；即：可以表示没有，可以表示有，可以表示有很多位；内容是什么无所谓

‘杨%’ 表示： ‘杨’ ‘杨逍’ ‘杨不悔’ ‘杨xxxx’ 即： 以杨开头的

‘%杨’ 表示：以杨结尾的

‘%杨%’ 表：包含杨的

\_ 表示：1位任意字符；即：占位符

！= <> ：表示不等于运算符

> >= < <= ：表示比较运算符

以上属于单条件查询，即：根据一个条件来查询数据

1. **单条件查询：**

**select \* from 表名 where 查询条件1 and 查询条件2;**

**select \* from 表名 where 查询条件1 or 查询条件2 or 查询条件3;**

and （与） 左右均满足条件才会被查询出来

or （或） 满足其中一个条件就会查询出来

and 与 or 优先级：and > or

查询条件中有and 有or 的时候，就一定会先执行and两侧的，然后再执行or

1. **多条件运算符：**
2. in 用法：where 列名 in （‘值1’,’值2’,’值3’,........）;

功能： 将这匹配这些值的数据都查询出来；

= 单值精确匹配

1. between...........and.....：

功能： 用来查询一个区间范围内的数据； 是闭区间 [a,b]

用法： where 列名 between 值1 and 值2

等价于 列名>=值1 and 列名<=值2

值1 是小的

值2 是大的

1. Not

用法： where not( 1个或者多个查询条件 )

功能： 括号中的查询条件，取反的意思 有点像逆向思维

如果运算符是 in

1. Where 列名 not in（‘值1’,’值2’,’值3’,........）;
2. **排序功能：order by**

将查询出来的数据进行排序

##### 排序关键字： order by 列名 asc|desc（默认升序）

##### asc ：升序 desc：降序

###### 语法：select \* from 表名 order by 列名 asc|desc

注意：先查询出数据，才能排序

总结：绝大部分的排序，都是按照某一个列来进行排序的；但是也会存在综合排序的情况（即：按照多个列进行排序）

select \* from 表名 [where 1个或者多个查询条件] order by 列1名 asc|desc, 列2名 asc|desc, 列3名 asc|desc,....;

功能：先按照第一个列进行排序， 第一个列取值一行的数据，再按照第二个列排序；如果前两个列取值都一样的数据，再按照第三个列排序

1. **limit：取范围内的值 取值范围默认是从1开始**

功能： 是用来取第几行到第几行的数据； 可以帮助开发来实现分页功能；

limit n,m （n表示从第几行开始取，第(n+1)行开始取； m表示取几行 ）

limit还可以结合排序一起使用； 实现top-n分析的功能；

select \* from 表名 order by 列名 asc|desc limit 0,3; # 取前三名

1. **比较命令行和sqlyog工具使用的小区别：**

Sqlyog: 在连接；

写了数据库名；连接成功之后：都是在这个特定的数据库中使用；

写sql语句来用： 写了查询语句select； 查询数据；

命令行： mysql命令连接：没有确定数据库名； -h -P -u -p

use 数据库名; 回车 use

正常写sql语句来使用数据库;

1. **补充： 如果在安装mysql软件的时候，忘记创建自己的用户；**

安装成功之后不需要卸载mysql，重装；

怎么解决：

用root用户登录mysql；

1. 用mysql命令，使用root账号连接mysql；
2. Show databases; 查看一下所有的数据库；
3. Use mysql 里面可以管理mysql用户信息；
4. 用语句创建用户： create user '用户名'@'%' identified by '密码';

### % 表示用户可以在非localhost上访问mysql

create user ’test’@’localhost’ identified by ’147369ljh’;

1. 给这个新用户赋权限

grant all on \*.\* to '用户名'@'%';

所有权限 所有数据库中的表

grant privileges on databasename.tablename to ‘username’@’host’;

1. flush privileges； 刷新
2. **拼接列，拼接字符串的函数**
3. concat（参数1,参数2,........）把这些参数拼接在一起

**备注**：列名不需要添加引号，字符串才需要添加引号

1. concat\_ws（**分隔符**,参数2,参数3,......）把从第二个参数开始的所有参数内容拼接在一起； 并且每个参数之间是有特定分隔符的；

**参数之间的分隔符**就是传入的第一个参数（**分隔符**）

\_ws ：with separte

1. **空值 null ：表示的是未知的，不确定的值；**

运算符： is

特性：任何值跟空值做运算都得空值

会带来一个问题： 运算的结果会跟实际生活中的情况不相符

解决： 通过空值转换函数可以解决

ifnull(参数1,参数2) 参数1是需要转换空值的那个列名 ；参数2 ： 当参数1为空值时，就用参数2的值来代替————ifnull（列名，0）

select \* from emp where ename is null；

select ifnull(name, “无名无姓”) from student; 查询学生表中姓名一栏为空的数据替换为“无名无姓”

1. **mysql数据类型**
2. **字符串类型：**

char 固定长度的字符串；char(10) 固定就是10位的字符串；分配10位空间[存储效率高一些]

varchar 可变长度的字符串；varchar(10)最长是10位的字符串；会根据实际的长度[节省空间]

备注：字符串类型，可以存放字母、数字、符号、汉字

当长度超过了字符串设定的长度之后，会默认取字符串设定的最大的长度来存放数据，后面的数据就存不进去了。

create table t07072(

num1 int unsigned,#unsigned表示：无符号的

num2 int

);

insert into t07072(num1,num2) values(2147483648,2147483648);

select \* from t07072;=>2147483648 2147483647

2147483648

1. **整型：**

tinyint：存储空间占1个字节；

取值范围是：-128~127

smallint：占2个字节

int：占4个字节

备注：

存放的是整数；支持负数，支持正数； 但是必须匹配取值范围；如果超出了取值范围，采用的是边界的值；

如果要存入小数： 进行四舍五入然后存放； 依然是整数

如果要存入字母； 依然是整数，值是0

create table t07073(

num1 float,

num2 double

);

insert into t07073(num1,num2) values(99.6541784521,99.6541784521);

select \* from t07073;

1. **浮点型：支持小数**

float：单精度

double：双精度（更加准确一些）

1. **自定义的数值类型：**

decimal（m,d）

m：表示**整体**数字的位数

d：表示**小数点**后保留几位

create table t07074(

num1 decimal(5,2)

);

Insert into t07074(num1) values(1000); #表示：999.999999

select \* from t07074;

1. **日期时间类型：**

date：年月日

time：时分秒

detatime：年月日时分秒

create table t07075(

riqi date,

shijian time,

riqishijian datetime

);

insert into t07075(riqi,shijian,riqishijian)

values('2020-07-07 10:40:33','2020-07-07','2020-07-07 10:40:33');

select \* from t07075;

备注：如何查看表中的包含哪些列，以及列的数据类型；

1. 如果使用的是sqlyog工具：直接在左侧点开查看；
2. 如果使用的是命令行：mysql命令登录 desc 表名; describe 描述；查看表结构的；查看表中 包含的列，以及数据类型
3. **DML数据操作语言： data manipulation language**

功能：主要是用来管理表数据的维护表中数据：跟数据有关

1. **insert into 表名 values 插入语句**

insert into t0707(id1,id2) values('12$%哈哈aabbcdew','12$%哈哈aabbcdew');

select \* from t0707;

===>

语法：

1. **insert into 表名(列1名,列2名,列3名,....) values(值1,值2,值3,....);——列名与值名一一对应；**
2. **insert into 表名 values(值1,值2,值3,....);——简写语法：所有列都要插入数据；**

**插入多行的语法：**

1. **insert into 表名(列1名,列2名,列3名,....) values(值1,值2,值3,....),(值1,值2,值3,....),(值1,值2,值3,....),....; 指定的列名下都插入数据**
2. **insert into 表名 values(值1,值2,值3,....),(值1,值2,值3,....),(值1,值2,值3,....),......; 不指定列名的插入数据**
3. **delete from 删除语句**

**delete from 表名 [where 1个或者多个查询条件];**

select 列名 from 表名 where 一个和多个查询语句；

1. 要删除的数据在哪个表中； 确定表名
2. 要删除这个表中哪一行或者是哪几行数据； 确定行； where 查询条件

备注：删除数据要慎重；建议在删除数据之前，要先查询一下数据；确定要删除这些数据，再写delete语句；

**清空整个表的数据：**

1. **delete from 表名;**
2. **truncate table 表名;** 清空效率高；表中数据量比较大的时候建议采用这个
3. **update 表名 set 更新修改语句**

修改之前，需要明确的：表名、列、行 + where查询条件

update 表名 set 列1名 = 新的值,列2名=新的值,... [where 查询条件];

重置 赋值 控制的行

update student set sname = ‘郭琪’ where sid = ‘SH6110’;

修改数据的几种情况：

1. 可以将原本是空值的，修改为一个明确的值；
2. 可以将原本有值的，修改为一个空值；
3. 可以将原本有值的，修改为一个新的值
4. **事务是用来保证数据的一致性，即：保证操作的一致性；**

即：一个事务中的所有DML操作，要么都成功，要么都失败； 不会出现一部分成功，一部分失败的情况；

DML语句会自己开启一个事务，执行第一个DML 语句时就会自动开启事务

在mysql数据库中，默认的情况下，事务是自动提交的，每执行完一个dml语句，就会自动开启一个事务，但是同时也会自动提交一个结束这个事务。

在mysql数据库中想要实现多个DML语句在同一个事务中进行控制：

1. 把自动提交事务关闭；

**select@@autocommit;**先查看事务自动提交的状态：{1：表示自动提交；0：表示手动提交}

**select@@autocommit=0;**

1. 编辑DML语句；
2. 手动结束事务：

**commit：**提交、让事务中的DML都操作成功

**rollback：**回退、回滚；让事务中的DML都操作失败

**事务的特性：**

1. 原子性（把事务当成是一个原子，表示不可分割是一体；要么都成功、要么都失败）
2. 隔离性（侧重指事务之间相互隔离，不受影响，与事务设置的隔离级别有关）
3. 一致性（执行事务前后状态一致，所有窗口看到的数据是一样的）
4. 永久性[持久性]（在事务结束之后，操作的数据就永远的存在于表中，除非手动删除修改）

我的理解是：原子性、隔离性、永久性都是为了保障一致性而存在的，一致性也是最最终的目的。

1. **DDL 数据定义语言： data definition language**

功能：主要是用来管理数据库对象的 维护表结构：跟表、列有关

1. **创建表：create table 表名( ); 表名包含数字、字母、下划线**

语法：**create table 表名**(

列名1 数据类型,——字符串类型的是需要确定长度的

列名2 数据类型,——整型、日期、浮点型不需要确定长度

·

·

）;

**create table t0707(**

**id1 char(10),**

**id2 varchar(10)**

**);**

**crate table tester.student（**

**sid char(7）unique,**

**sname varchar(12) notnull,**

**ssex char(1),**

**sage int,**

**sphone varchar(11)**

**);**

1. **改变表结构：alter table 表名;**

##### 在表中添加列：

alter table 表名 add 列名1 数据类型,add 列名2 数据类型,..........;

alter table student add sid varchar(10), add scn int, add sdm char(10);

##### 修改表中的列：

alter table 表名 change 表中列名 new列名 new数据类型,change 表中列名 new列名 new数据类型,······;

alter table student change sname stu\_name varchar(12),change sage stu\_age int,change ssex stu\_sex char(1);

##### 删除表中的列：

alter table 表名 drop 列名1 , drop 列名2，·····;

alter table student drop sname, drop sid;

alter table 表名

添加列：add 列名1 数据类型，add 列名2 数据类型，···

修改列：change 表中列名1 新的列名1 新的数类型，change ···

删除列：drop 列名1，drop 列名2，···

1. **删除表：drop table 表名;**
2. **约束：自动保持数据库完整性的一种方法，定义了可输入表或表的单个列中的数据的限制条件**
3. **非空约束——not null**

约束是在创建表的时候创建的

1. **唯一约束——unique**

值可以为空，但是有值就必须唯一

1. **主键约束——primary key**

就是非空与唯一约束这两个限制的集合

create table t07075(

id char(18) primary key,——主键约束

name varchar(15) not null,——非空约束

phone char(11) unique,——唯一约束

pro int

);

select \* from t07075;

每个表中都必须要有主键，主键列一般都是编号，不是名字；——主键是用来唯一表示一行数据的

主键列： 添加了主键约束的列；

主键值： 主键列中的取值

主键约束：primary key

1. **外键约束——foreign key**

涉及到两个表： 其中一个表 是父表； 另一个表是子表

父表，子表：涉及的就是从属关系： xxxx属于yyyy

|  |  |
| --- | --- |
| 如果两个表中，涉及到了外键约束：  如果想要往子表中插入数据，那么父表中必须得先有这条数据；  如果想要删除父表中的数据；那么就要先在子表中把这个对应的数据要先删掉； | 外键约束创建使用过程：   1. 先创建父表； 父表中必须要有主键 2. 再创建子表； 子表中添加外键约束；   外键约束建立了，关系就建立  **foreign key (子表中外键列名) references 父表名(主键列名)**  备注：子表中的外键列名 跟 父表中的主键列名 可以是一样的，也可以不一样；  foteign key (pro) references t07074 (pro\_id)   1. 插入数据时，先插父表； 删除数据时，先删儿子 |
| create table t07074(  pro\_id int primary key,  pro\_name varchar(10)  );  select \* from t07074;  ——父表（先创建） | create table t07075(  id char(18) primary key,——主键约束  name varchar(15) not null,——非空约束  phone char(11) unique,——唯一约束  pro int  foteign key (pro) references t07074 (pro\_id)  );  select \* from t07075;  ————子表 |

1. **检查约束**

（在mysql中不支持，在oracle中支持）

1. **分组查询：**

功能： 主要是用来做统计的； 即：按照xxxx ，来统计xxxx

例如： 企业：按照部门统计每个部门的平均薪水，部门员工人数

1. **分组函数：**

可以单独使用；可以不结合分组查询使用

count(参数) count(列名)统计的是查询返回的行数；

即：数量 ；查询数量 count(\*) 表示查询出所有的行数

max(参数) max(列名) 查询出来的这一列的最大值

min(参数) min(列名) 查询出来的这一列的最小值

avg(参数) avg(列名) 查询出来的这一列的平均值

SELECT site\_id, count FROM access\_logWHERE count > (SELECT AVG(count) FROM access\_log); # 先查出整体平均值，然后再判断大于平均值的site\_id 和 count。

sum(参数) sum(列名) 查询出来的这一列的和

select count(\*),max(sal),min(sal),sum(sal),avg(sal)from emp;

# 根据列名name在employee表中查询名字出现的次数

SELECT name, COUNT(\*) FROM employee\_tbl GROUP BY name;

SELECT coalesce(name, '总数'), SUM(signin) as signin\_count FROM employee\_tbl GROUP BY name WITH ROLLUP;

+--------------------------+--------------+

| coalesce(name, '总数') | signin\_count |

+--------------------------+--------------+

| 小丽 | 2 |

| 小明 | 7 |

| 小王 | 7 |

| 总数 | 16 |

+--------------------------+--------------+

4 rows in set (0.01 sec)

1. **分组查询： group by col\_name**

关键字：**group by 列名** 按照xxx统计，就按这个列来分组的

# WITH ROLLUP 可以实现在分组统计数据基础上再进行相同的统计（SUM,AVG,COUNT…）

SELECT name, SUM(signin) as signin\_count FROM employee\_tbl GROUP BY name WITH ROLLUP;

+--------+--------------+

| name | signin\_count |

+--------+--------------+

| 小丽 | 2 |

| 小明 | 7 |

| 小王 | 7 |

| NULL | 16 |

+--------+--------------+

4 rows in set (0.00 sec)

**过滤关键字：having**

SELECT Websites.name, SUM(access\_log.count) AS nums FROM Websites

INNER JOIN access\_log

ON Websites.id=access\_log.site\_id

WHERE Websites.alexa < 200

GROUP BY Websites.name

HAVING SUM(access\_log.count) > 200;

1. **结合分组函数一起使用：**

语法：

select 分组的那个列名,1个或者多个分组函数 from 表名 group by 列名;

select did,count(\*),avg(sage) from student group by did;

###### 按照多列分组：

select 分组的那几列, 1个或者多个分组函数 from 表名 group by 列1名,列2名,..;

select sname,ssex,did,sphone,max(sage) from student group by did,ssex,sname,sage;

select did,ssex,count(\*) from student group by ssex,did order by did desc;

###### 分组之后，针对分组之后的在进行过滤

select 分组的那个列名，1个或多个分组函数 from 表名 group by 列名 **having 分组函数 运算符 搜索值；**

select did,count(\*),avg(sage) from student group by did having count(\*) <= 3;

计算**平均成绩大于75分**的课程名称和平均成绩：

select avg(grade),cname from score join course on score.cid = course.cid group by cname having avg(grade) > 75;

1. **多表联合查询：**

关系模型数据库中，主要有3种关系：

#### 一对一的关系： 例如：学号跟学生 系别编号跟系 课程编号跟课程 员工工号跟员工 身份证号跟公民

这种关系在表中： **是通过 主键 来体现**

select \* from student,department where student.did=department.did;

#### 一对多的关系:例如: 学生跟系的关系 员工跟部门的关系 一个学生是属于一个系的；一个系可以包含多个学生

即：从属关系

这种关系在表中： **是通过 外键 来体现**

select \* from student,department where student.did=department.did and sage>22;

多对多的关系:例如:学生跟课程是通过选课成绩表来体现 一个学生可以选择多门课，一个课程也可以被多个学生选

这种关系在表中： **是通过 第三张表 来体现**

select \* from student s,department d where s.did=d.did;

select \* from student,score,course where student.did=score.did and score.cid=course.cid;

总结：多表联合查询就是从多个表中查询出自己想要的数据，但是必须要添加**连接条件**；否则就会查询出大量的错误数据。

连接条件是根据两个表的关系来写的

绝大部分的关系：都是外键关系； 而且一般习惯上：子表跟父表的关联列的列名是一样的；

连接条件的格式： 第一个表的列名 运算符 第二个表的列名

**语法：select \* from 表1名 别名1，表2名 别名2，表3名 别名3 where 1个或者多个连接条件 [and 查询条件]；**

select \* from student s,department d **where s.did=d.did and sage>22;**

1个连接条件最多只能连接2个表；

如果要连接 n个表， 至少需要n-1个连接条件

补充：表跟表之间的关系可以是不是外键关系；特别少：

员工表：emp 薪资级别表： salgrade

例如：查询员工的姓名，薪水，以及这个员工所在的薪资级别

员工表的员工薪水，要在薪资级别表的最低薪水跟最高薪水之间；

select \* from emp; # 查看员工表

select \* from salgrade ,emp where sal between losal and hisal;

select \* from salgrade ,emp where sal>=losal and sal<=hisal;

补充： 多表联合查询，可以是从同一个表来查询； 自连接

select \* from **student s,course c ,score sc** where s.sid = sc.sid and sc.cid=c.cid;

例如： 查询员工的姓名，岗位，薪水，以及这个员工的主管姓名

emp既是员工表； emp也是主管表

select yuangong.ename,yuangong.sal, yuangong.job,zhuguan.ename from emp yuangong, emp zhuguan where yuangong.mgr = zhuguan.empno;

1. **内连接、外连接：**

内连接的关键字：表1 inner join 表2 on 表1.主键/外键 = 表2.主键/外键;

内连接的功能：就是多表联合查询；是多表查询的另外一种写法

比较：多表联合查询 和 内连接：功能是一模一样的；

区别：

**多表联合查询的语法： 语法简单 （跟单表的思路是相同的）**

**内连接的： 优势： 查询效率要高一些； 即：查询的快一些 （开发人员会用到）；**

**表数据量比较大的**

内连接（多表联合查询）：将左右两个表中符合连接条件的数据查询出来；

select \* from student , course , score where student.sid = score.sid and score.cid=course.cid;

select \* from student inner join course inner join score on student.sid = score.sid and score.cid=course.cid;

左连接（左外连接）：left join 会将两个表中符合连接条件的数据查询出来，但是还会将左表中没有匹配上的数据也会查询出来； 即：左表中不管是符合连接条件还是不符合的数据都会查询出来； 但是右表只会显示符合连接条的数据；

右连接（右外连接）：right join 会将两个表中符合连接条件的数据查询出来，但是还会将右表中没有匹配上的数据也会查询出来； 即：右表中不管是符合连接条件还是不符合的数据都会查询出来； 但是左表只会显示符合连接条的数据；

select \* from emp,dept where emp.`DEPTNO` = dept.`DEPTNO`;

select \* from emp inner join dept on emp.`DEPTNO` = dept.`DEPTNO`;

select \* from emp right join dept on emp.`DEPTNO` = dept.`DEPTNO`;

select \* from dept left join emp on emp.`DEPTNO` = dept.`DEPTNO`;

在oracle： 全外连接

1. **子查询：**

又叫做嵌套查询 ：指的是一个语句中包含来一个查询语句；子查询语句主要是用在查询条件中：

语法：**select \* from 表名 where 列名 运算符 （子查询语句）；**

select \* from student where sage>(select sage from student where sname='张无忌') ;

delete from student where sage>( select sage from student where sname='张无忌' );

简单理解为： 子查询语句代替来值（搜索值）

子查询在查询条件中的使用原则：

子查询语句必须写到小括号 ()

子查询语句必须要写到查询条件的右侧； 因为子查询代替的是搜索值

子查询语句只能查询一个列

**如果是单行子查询，采用的运算符是： = > < >= <= !=**

单行子查询，即：跟一个值比较；

**如果是多行子查询；采用的运算符： in all any （all any 会结合> >= < <= 一起使用）**

**select \* from student where did in(select did from department where dname='计算机' or dname='数学');**

多行子查询，即：跟多个值比较

All: 跟所有值进行比较；要比赢所有人 要求严格

>all(子查询) : 大于最大的

<all(子查询): 小于 最小的

Any:跟其中一个值进行比较； 比赢其中一个人就好 宽松

>any(子查询): 大于最小的

<any(子查询) : 小于最大的

根据子查询返回的行数来区分；如果子查询返回一行，就是单行子查询；如果返回>=2行，就是多行子查询

select \* from student where sage> all( select sage from student where sname in ('张无忌','宋青书','赵敏'));

select \* from student where sage>( select MAX(sage) from student where sname in ('张无忌','宋青书','赵敏'));

如果子查询比较麻烦，建议可以先分步理思路：

例如：查询性别跟张无忌不一样的学生

先查询出张无忌的性别 select ssex from student where sname=”张无忌”;

再查询性别跟他不一样的学生 select \* from student where ssex !=(select ssex from student where sname=”张无忌”);

select \* from student where ssex not in (select ssex from student where sname=”张无忌”);

**子查询还有一种用法: 子查询可以来代替一个表**

语法

Select \* from 表名 [where 查询条件];

**Select \* from (子查询语句) 别名 [where 查询条件];**

**例如： 先查询出薪水>2000的员工， 再从这些员工中查询岗位为“CLERK”的员工**

**select \* from (select \* from emp where sal>2000) a where job='CLERK';**

课后练习： 查询薪水比自己部门平均薪水高的员工 （分组，子查询；多表）

查询薪水比自己部门平均薪水（按部门名称分组统计）高的员工 （分组，子查询，多表）

1. **mysql常见函数：**

Concat(参数1,参数2,...) 功能：将多个参数的内容拼接在一起

Concat\_ws(分隔符,参数2,参数3) 功能：将多个参数拼接中一起，每个参数之间有分隔符

ifnull(参数1,参数2) 功能：转换空值

分组函数： 统计函数， 集合函数

Count(\*) avg(参数) max(参数) min(参数) sum(参数)

**数值函数：**

abs() round() truncate() ceil() floor() power()

Round(参数1,参数2):处理数字的格式；参数1是需要处理的数据； 参数2：保留小数点后几位； **并且会进行四舍五入**

Truncate(参数1,参数2):同上； **不会进行四舍五入**；直接截断

Abs(参数) 取绝对值的

Ceil(参数) 向上取整的

Floor(参数) 向下取整的

Power(x,y) x的y次方

**字符串函数：**

**参数可以是普通字符串，也可以是字符串类型的列名**

Lower(参数) 全部变小写

Upper(参数) 全部变大写

Length(参数) 取长度

Substr(参数1,参数2,参数3) 取子串； 参数1是原字符串； 参数2表示从第几位开始取；

参数3表示取几位

Trim(参数) 将参数的前后空格去掉

Ltrim(参数) 将参数的前面的，左边的空格去掉

Rtrim(参数) 将参数后面的， 右边的空格去掉

Replace(参数1,参数2,参数3) 替换函数； 参数1原字符串； 参数2表示要将什么替换掉； 参数3表示要替换成什么

Reverse(参数) 反转； 将字符串的参数进行反转

**日期函数：**

CURRENT\_DATE(), 获取当前年月日

CURRENT\_TIME(), 获取当前时分秒

CURRENT\_TIMESTAMP() 获取当前年月份时分秒 now()

获取日期时间中的部分内容：

EXTRACT(type FROM d)从日期 d 中获取指定的值，type 指定返回的值

extract(type from 日期)

type可取值为：

MICROSECOND 微秒

SECOND 秒

MINUTE 分

HOUR 时

DAY 日

MONTH 月份

YEAR

SECOND\_MICROSECOND

MINUTE\_MICROSECOND

MINUTE\_SECOND

HOUR\_MICROSECOND

HOUR\_SECOND

HOUR\_MINUTE

DAY\_MICROSECOND

DAY\_SECOND

DAY\_MINUTE

DAY\_HOUR

YEAR\_MONTH

1. **数据库**

数据库对象：简单的理解为就是存储中数据库中的东西

常见的数据库对象

1. **表（table）：**

由行和列构成， 作用：放数据

1. **索引（index）：**

功能：主要用来提高查询效率的；类似于书的目录；主要用在数据量比较大的表中；

使用索引：

1. 如果经常根据某个列进行查询的话，那么可以给这个列添加索引；

**create index 索引名 on 表名（列名）;**

1. 加完索引之后，就正常写查询语句；即：根据这个列查询的语句；自动查的会快点
2. 视图（view）：功能：主要是用来简化查询语句的；可以将一个复杂查询语句简单化；
3. 创建视图：**create view 视图名 as select语句；【列名不能有重复的】**
4. **要查询语句，就**可以从视图中查询；**select \* from 视图名**；
5. 数据还是存放在表中，而不是存放在视图中；
6. **存储过程（procedure）**

功能概念：存储过程就是数据库中的编程;可以通过写代码来实现更加复杂的功能，或者是可以通过写代码实现sql语句实现不了的功能；

存储过程的特点优势：

1. 执行速度比较快； 效率高
2. 提高了语句的复用性； 存储过程是会存储在数据库中的；

测试人员在实际工作中： 一般是在做性能测试时，如果需要准备大量的数据，这个时候使用存储过程比较方便。 可以写循环

存储过程的使用步骤：

创建存储过程； 即：写代码 create procedure

调用存储过程，使用存储过程： call 存储过程名(0个或者多个参数); 在写SQL语句的窗口中写的；

1. **介绍存储过程：**

创建： 在左侧存储过程，上选中，右键-->创建存储过程 -->输入：存储过程的名字

注意：只能创建一次； 执行一次

编辑，修改存储过程： 在左侧存储过程，选中要修改的那个存储过程，右键-->改变存储过程

备注：改变多次，执行多次

结构：

delimiter $$ #定义存储过程的分隔符；即：结束符； 表示：存储过程从 create procedure开始，

到 $$ 这里结束 ； 不可以是 分号 ;

Create procedure 数据库名.存储过程名() #创建存储过程；给当前数据库来创建存储过程

Begin **#存储过程中的代码：来写到 begin end 之间的**

存储过程体;

End$$ #存储过程结束

Delimiter ; #把分隔符，又变回默认的分隔符，结束符

1. **打印语句：**

Create procedure 存储过程名()

Begin

Select “”;

Select \* from 表名 where 查询条件;

End $$

|  |
| --- |
| create definer=`ljh`@`%` procedure `stu\_02`()  begin  declare stu\_name varchar(20);  declare stu\_age int;  declare stu\_sex char(2) default "男";  #直接给局部变量赋值  set stu\_name = "liangzai";  set stu\_age = 18;  select stu\_name,stu\_age,stu\_sex;  end$$  delimiter ; |

打印：select语句

call stu\_02;调用stu\_02 函数

1. **变量：**

存储过程中：主要包含2种变量； 局部变量 和 用户变量(全局)

局部变量： 只在当前这个存储过程里面可以使用的；

用户变量（全局变量):在当前登录的用户中都可以使用的； 在存储过程外面也可以使用；

**局部变量**

使用过程：

先定义局部变量； 声明局部变量

declare 变量名 数据类型;

给局部变量赋值； 不赋值：就默认是空值

定义局部变量的同时来赋值：

declare 变量名 数据类型 default 值;

set 变量名 = 值;

使用局部变量

Select 变量名; 打印这个变量

|  |
| --- |
| create definer=`wuwu`@`%` procedure `var\_procedure`()  begin  declare age int; #定义一个 int 类型的局部变量 age；  declare emp\_name varchar(20); #定义一个字符串类型的局部变量 name；  declare sex CHAR(2) default '男';#定义了一个字符串类型的局部变量sex，并且赋值为‘男’；  declare emp\_job varchar(20);  set @emp\_sal = 0;#全局变量  select ename,job,sal into emp\_name,emp\_job,@emp\_sal from emp where ename='smith';  select emp\_name,emp\_job; #打印这个两个变量的值 |

**注意： declare必须直接写到begin的下面**

**用户变量（全局变量）**

**注意：**

1. **用户变量：在赋值的时候，来同时创建声明的； 并且根据赋的值确定了数据类型；**
2. **用户变量：在变量名之前 有一个 @**

用户变量使用：

Set @变量名 = 值;

例如： set @phone = ‘18111111111’;

Select @变量名; 打印这个变量的

**给变量赋值：**

通用的：set 局部变量名= 值; set @全局变量名 = 值;

局部变量独有的： declare 局部变量名 类型 default 值;

通用的：只在数据库中才有可能用到的； 从表中查询出数据，然后赋值给变量 （用的多的）

Select 列1名,列2名 ,... into 局部变量名, @全局变量名 from 表 where 查询条件;

|  |
| --- |
| #set age = 20; #给局部变量age赋值  #set name = "haha"; #给局部变量name赋值  #set @phone = "18112345678";#给全局变量 @phone赋值，即：创建了一个全局变量，并且赋值了  #select age,name, sex,@phone; #打印显示这几个变量的值  #set age = 40;  #set sex='女';  #set @phone="13333333333";  #select age,name, sex,@phone;  select ename,job into emp\_name,emp\_job from emp where sname= ‘smith’;  #从员工表中查询smith的姓名和岗位赋值给 emp\_name,emp\_job  end$$ |

1. **参数：**

**存储过程中有3种参数： in out inout**

1. **In**

给存储过程里面传的参数； 就类似于 python函数的参数

存储过程中需要这个东西，但是这个东西存储过程自己没办法自给自足，需要外面给传

例如：指定某一个学生，将这个学生的性别，年龄，赋值到变量中；并且打印

select ssex,sage into stu\_sex,@stu\_age from student where sname=学生的名字;

1. **in 参数的使用:**

create procedure 存储过程名 ( in 参数名 数据类型 )

形参：形式上的参数；知道要传一个什么类型的参数；

begin

**#如果要使用in参数； 直接通过参数名使用**

Select \* from 表名 where 列名 = 参数名;

end $$;

1. **调用存储过程**

call 存储过程名(参数的值);

实参：实际的参数； 知道了实际要传的数据是什么

好处：有了in参数，功能就可以变的更加灵活，更加强大一点；

|  |
| --- |
| 1. 查询薪水比每个部门平均薪水都要高的员工  create definer=`wuwu`@`%` procedure `parameter\_pro`(in **emp\_name** varchar(20) )  begin  declare **emp\_sal** double;  declare **emp\_dname** varchar(10);  drop table if exists 表名; #drop procedure if exists `stu\_06`$$;（存储过程名）  select sal,dname into emp\_sal,emp\_dname from emp,dept  where emp.`DEPTNO`=dept.`DEPTNO` and **ename= emp\_name;**  select **emp\_sal,emp\_dname**;  end$$ |

|  |
| --- |
| create definer=`wuwu`@`%` procedure `if\_course\_num`(in stu\_cname CHAR(10),out kcsc int)  begin  declare stu\_tname CHAR(10); #讲师姓名  declare stu\_count int; #这门课的学生数量  select chour,tname,COUNT(sid) into kcsc,stu\_tname,stu\_count from course, score  where course.`cid`=score.`cid` and cname=stu\_cname;  select stu\_tname,stu\_count; |

1. **out**

类似于python函数的返回值

把数据从存储过程中返回出去； 给外面的； out类型的参数

如果有数据，需要返回给调用的人；那么就可以定义out类型的参数

将数据返回到外面；即：在存储过程外面也可以使用到这个数据：

1. 可以把这个数据，赋值到一个全局变量中；在外面就可以直接使用 @全局变量名
2. 也可以把这个数据，放到out类型的参数中往外返。

Out参数使用过程：

1. **创建存储过程时，定义out类型的参数**

**Create procedure 存储过程名(out 参数名 数据类型)**

**Begin**

**#把需要往外返的数据，赋值到out 类型的参数中**

**Set 参数名 = 值; #把要返回的值，赋值到out类型的参数中；**

**Select xxxx into 参数名 from 表名 where 查询条件;**

**End$$**

1. **调用这个存储过程了 ； 要接收这个返回值； 用全局变量接收**

**Call 存储过程名(@全局变量名); 接收返回值；**

**返回的数据就会自动放入到 @全局变量中**

1. **inout 参数：**

这个参数既可以往存储过程中传数据，又可以把要返回的数据放到这个参数中；

把1个in参数，和 1个out参数，合并到一起； 很少

1. **if语句：**

|  |  |
| --- | --- |
| 功能：实现逻辑判断  if stu\_count >= 3 then  select "学生挺多的";  elseif stu\_count >= 1 and  stu\_count < 3 then  select "正常";  else  select '需要招生';  end if; | case jixiao  when 'A' then  select "优秀";  when 'B' then  select "良好";  when 'C' then  select "一般";  else  select "注意";  end case; |

1. **case when语句：**

功能：实现逻辑判断；

一般如果是针对某个变量，来判断这个变量是什么值的时候，建议使用case when，比较简单方便；

例如： 给存储过程，传入一个绩效等级； 接下来针对绩效等级进行判断

如果是： A 打印：优秀 如果是B 打印：良好 如果是C 打印：一般 否则：打印：

1. **循环语句：**

如果是循环，就一定要进行循环控制； 控制循环的次数；不可以：无限循环，即：死循环

如何进行循环控制： 可以利用循环变量进行控制

定义声明一个变量 ； 用来做循环控制； 叫做：循环变量

Declare 变量名 int default 初始值;

给这个变量赋值； 在循环之前；

明确循环条件： 循环变量<=结束值 循环变量>=结束值 while 循环条件

循环是从初始值开始的，到结束值结束

改变循环变量的值： set 循环变量 = xx; 循环体

**存储过程中主要提供了:while repeat循环**

比较：

1. 语法不一样；
2. 循环条件控制那块不一样
3. While 循环的条件 符合条件就循环
4. Repeat until 退出循环条件； 符合条件就退出
5. Repeat至少会循环一次的； While可能一次都不会进行循环

|  |
| --- |
| while i<=10 do  insert into t0711 values(i,CONCAT("hehe",i));  set i=i+1;  end while;  repeat  select "hello world!";  set m=m+1;  until m>5  end repeat; |

declare i int default 1;

#明确循环条件的

while i<=5 do

select "hello world!";

set i = i+1; #改变循环变量的值

end while;

declare m int default 1;

repeat

select "hello world!";

set m=m+1;

until m>5

end repeat;

课后练习：

创建一个存储过程，实现以下功能：

drop table if exists 表名;

创建一个表；包含：id name sex age 这4列

往这个表中插入20行数据 ; 用循环

其中：id name 两列的值不可以重复

性别是：男 女 更替 ； 要求：用case when语句

年龄是从20开始； 往上递增； 当递增到30时，年龄就一直是30 ； 用if语句