==简介==

Oracle(收费，大型)和Mysql(免费，小型)目前都是oracle公司的

现在最新的SQL标准：SQL 2008

nosql技术，主要为了抵制SQL规范，目前翻译成not only SQL，最为代表的MongoDB数据库(Node JS开发)

==sqlplus介绍==

对于Oracle的前台管理工具

Oracle9i：OEM(企业管理器)、sqlplusw.exe、sqlplus.exe

Oracle10g：sqlplusw.exe、sqlplus.exe、EM(在线企业管理器)

Oracle11g：sqlplus.exe、sql developer

Oracle12c：sqlplus.exe、sql developer

sqlplus是Oracle提供的数据库管理的交互式工具，可以管理数据库的所有任务，sqlplus通过命令管理数据库，也可以通过sqlplus执行SQL语句的操作

sqlplus在执行SQL语句有一个缓存区

==基本命令==

Oracle登录命令：

普通用户：sqlplus(sqlplus 用户名/密码)

切换用户：conn 用户名/密码

解锁scott用户：alter user scott account unlock;

设置(修改)登录密码 alter user scott identified by wenge;

show user:显示当前用户名

disc[onnect]：断开与oracle的连接，不退出sqlplus窗口

exit：断开与oracle的连接，退出sqlplus窗口

修改密码：passw(ord) 用户名 –会提示输入新密码

spool(将屏幕显示的内容复制文件中)

1 spool e:/abc.sql---开始复制

2 spool off---结束复制

linesize和pagesize

set linesize 300；(之后查询表的大小为300，便于观察)

set pagesize 100；(之后查询表的行数为100，便于观察)

查询所用用户：select \* from all\_users;

查询所有用户信息(system登录) select \* from dba\_users;

查询用户及解锁状态select username,account\_status from dba\_users;

查询所有用户名,密码(system登录) select username,password from dba\_users;

查询当前用户信息 select \* from dba\_ustats;

删除用户

drop user 用户名 [cascade]

注：当我们删除一个用户时，如果这个用户创建过数据对象，在删除该用户时，必须加上cascade，把该用户数据对象和用户一起删除

别名

Select name [as] 姓名 from emp：没有引号括起来的别名不能使用特殊字符

Select name [as] “姓名()、/” from emp：Oracle只能用双引号括起别名，可以使用特殊符号(mysql可以用单、双引号括起)

==简单查询，基本==

简单查询：select [distinct] 列名称 [as] [别名] ...from 表名 [表别名];

执行顺序：第一步执行from子句，第二步执行select子句

连接符：||和concat(str1,str2,...)(mysql不能使用||，Oracle都能使用)

注：Oracle的concat只能连接2个，连接多个只能用||

select name || age from 表名;

select concat(name,age) from 表名 where name=concat('张','三');

sql语句支持四则运算

支持查询自定义常量(单引号括起来)：select '$' as 货币 from 表名;

限定查询：针对数据过多查询时可能会出现死机等问题(12c才有CDB与PDB)

1 nolog方式登录数据库：sqlplus /nolog(不登录数据库)

2 使用sys管理员进行登录：conn sys/change\_on\_install as sysdba;

3 切换到PDB中：alter session set container=pdbmldn;

4 打开PDB：alter database pdbmldn open;

进行SQL语句(用户名.表名)：where语句判断条件，限定运算符配合实现，限定运算符如下

关系运算符：> < >= <= = != <>(不等于)

逻辑运算符：and or not

判断控制：is null，is not null

范围查询：between 最小值 and 最大值(包括最大值与最小值)，in(值1,值2,...)，not in()，like模糊查询，not like

%：任意长度的字符 %%：任意长度中文 \_：单个字符

注意：in中有null可以正常查询，not in中有null查不到任何数据

执行顺序：from，where，select

实例：select \* from emp where sal>2000 and sal<4000;

Oracle的虚拟表：dual

==排序==

结果排序order by：放在SQL语句最后

order 字段|序号 [asc|desc]

asc：升序(默认)

desc：降序

执行顺序：from，where，select，order by

先按照sal排序，如果sal相等再按照hiredate排序：select \* from emp order by sal desc,hiredate desc;

按照序号查询(按照dept升序查询)：select name,depo,hiredate from emp order by 2;

==单行函数(数量巨大，只介绍常用的)==

语法：函数名(列|表达式[参数1,参数2])

PL/SQL编程可自己开发自定义的函数

1 字符函数

upper(列|字符串)：转大写

lower(列|字符串)：转小写

initcap(列|字符串)：开头字母转大写

replace(新的字符串,列|字符串)：新字符串替换旧字符串

length(列|字符串)：求出字符串长度

substr(列|字符串,开始点[,长度])：字符串截取(开始点和长度是数值)

ASCII(字符)：将字符转换成对应的十进制数字

chr(整数)：返回与之对应的字符

rpad(列|字符串,总长度,填充字符)：向右填充指定长度字符串

lpad(列|字符串,总长度,填充字符)：向左填充指定长度字符串

ltrim(字符串),rtrim(字符串)：去掉左或右空格

trim(列|字符串)：去掉左右空格

instr(列|字符串,要查找的字符串)：查找字符串(查找的到返回位置，查找不到返回0)

instr(列|字符串, 目标字符串, 起始位置, 匹配序号(第几个目标字符串))

简单举例：select substr(ename,1,length(ename)-2) from emp;//substr的开始点可以是0和1，如果是0最终会定义成1

2 数值函数

round(小数[,保留位数)：对小数进行四舍五入

trunc(小数[,截取位数])：对小数进行截取

mod(数字1,数字2)：取模(数字1除以数字2的余数)

ABS(数值)：绝对值

数据类型：

number(5,2)：3位整数2位小数(一般用于cast转换函数)

decimal(5,2)：同上，底层是number

trunc用于日期：

trunc(日期)：Navicat上获取日期的零点，命令行或pl/sql上截取年月日

trunc(日期,’dd’)：获取日期的零点(同上)

trunc(日期,’yy’/’yyyy’)：获取日期的当前年的第一天零点(同上)

trunc(日期,’mm’)：获取日期的当前月的第一天零点(同上)

3 日期函数\*

使用日期函数可以与数字进行计算，可以解决闰年问题

取得当前日期(利用伪列sysdate)：select sysdate from dual;

一般情况sysdate只能显示到日，systimestamp能显示到秒

日期格式化：alter session set nls\_date\_format='yyyy-mm-dd hh24:mi:ss';

日期计算公式：日期+/-数字=日期 日期-日期=天数

员工入职时间：select ename,trunc(sysdate-hiredate) from emp;

函数如下：

add\_months()：add\_months(sysdate,3)表示3个月之后的日期，add\_months(sysdate,-3)表示3个月之前的日期

next\_day()：next\_day(sysdate,'星期一')表示下一个星期一的日期，next\_day(sysdate,4)表示4天后的日期

last\_day(日期)：当前日期这个月的最后一天

months\_between(日期1,日期2)：两个日期间的月数(select ename 员工姓名,trunc(months\_between(sysdate,hiredate)/12) 入职年数 from emp;)

求员工入职具体时间(入职多少年多少月，多少天)：

select ename 员工姓名,trunc(months\_between(sysdate,hiredate)/12)||'年'||

trunc(mod(months\_between(sysdate,hiredate),12))||'月'||

trunc(sysdate-add\_months(hiredate,trunc(months\_between(sysdate,hiredate))))||'天' 入职时间 from emp;

extract()：

extract(year from date '2012-03-11')：显示2012

extract(month from systimestamp)：显示月数(day显示天数，hour显示小时(有点问题)，minute显示分钟，second显示分钟)

to\_timestamp转换函数，将字符串转换成日期时间：to\_timestamp('2014-02-13 12:20:33','yyyy-mm-dd hh24:mi:ss')

求得间隔的天数：select extract(day from to\_timestamp('2014-02-13 12:20:33','yyyy-mm-dd hh24:mi:ss')-to\_timestamp('2014-01-13 12:20:33','yyyy-mm-dd hh24:mi:ss')) from dual;

利用子查询：select extract(hour from datetime1-datetime2) hours,extract(minute from datetime1-datetime2) minutes from(select to\_timestamp('2014-02-13 13:25:33','yyyy-mm-dd hh24:mi:ss') datetime1,to\_timestamp('2014-01-13 12:20:33','yyyy-mm-dd hh24:mi:ss') datetime2 from dual);

timestamp：两个timestamp类型相减得到天数和时分秒毫秒的结合

timestamp时间戳与date类型之间的转换：

timestamp转换成date：timestamp+0，timestamp-0，cast(timestamp1 as date)或者利用to\_char与to\_date组合

date转换成timestamp：cast(date1 as timestamp) 或 to\_char与to\_timestamp组合

4 转换函数

oracle支持数据自动转换

数据类型有数字、字符串、日期(时间戳)

to\_char(日期|数字,转换格式)：将日期或数字转换成字符串

\*to\_char(sysdate,'yyyy-mm-dd')

to\_char(987654321.123,'999,999,999,999.99999')：987,654,321.12300

to\_char(987654321.123,'000,000,000,000.00000')：000,987,654,321.12300

to\_char(987654321.123,'$999,999,999,999.99999')：$987,654,321.12300

to\_char(987654321.123,'L999,999,999,999.99999')：根据语言不同自动选择货币

to\_char(sysdate,’DY’)：转化成星期(英文日期前三个大写字母)

to\_char(sysdate,’day’)：转化成英文星期

to\_char(sysdate,'day','NLS\_DATE\_LANGUAGE=''SIMPLIFIED CHINESE''')：转换汉字星期

查询每年2月份入职的员工：select \* from emp where to\_char(hiredate,'mm')='02';

to\_date(列|字符串,转换格式)：将字符串转换成date型

to\_date('2012-01-23','YYYY-MM-DD')：显示23-01月-12(看数据库有没有设置时间格式)

to\_timestamp()：to\_timestamp('2014-02-13 12:20:33','yyyy-mm-dd hh24:mi:ss')

\*to\_number(列|字符串)：将字符串转换成数字类型

to\_number('9')：转换成数字9

\*cast(数据A as格式B)：将数据A转换成格式B(字符串转换数字，时间转换成时间戳等等)

5 通用函数

通用函数：

\*NVL(数字|列,默认值)：查询comm如果有null则转换成0：select nvl(comm,0) 奖金 from emp;(null与任何数相加都是null)

NVL2()的使用：select nvl2(comm,sal+comm,sal)\*12 年薪 from emp;(如果comm不为空则执行sal+comm，如果为空则执行sal)

NULLIF(表达式1,表达式2)：如果1和2相等返回null，不相等返回表达式1

\*DECODE(列|值,判断值1,显示结果1,判断值2,显示结果2,...[,默认值])：多值判断，如果列|值与判断值相等，则输出对应的结果，否则输出默认值

select decode(2,1,'结果是1',2,'结果是2');

使用decode要判断所有内容，如果没有全部判断和默认值，则显示null

\*\*CASE 列|值 WHEN 值1 THEN 显示结果1 ... ,ELSE 显示结果n END [别名]

CASE WHEN 表达式1 THEN 显示结果1 ... ,ELSE 显示结果n END [别名]

SELECT ename,sal,case job WHEN 'CLERK' THEN sal\*1.1 WHEN 'SALEMAN' THEN sal\*1.2 ELSE sal\*1.5 END 新工资 from emp;

select ename,sal,case when sal<=1000 then '低' when sal>1000 and sal<=2000 then '中' else '高' end 薪资等级 from emp;

注：有些情况不想要别名的话，直接end结束即可

coalesce(表达式1,...表达式n)：如果1为null，则判断2，如果2不是空显示2，是空判断3，以此类推，如果n还是null，则显示null

listagg(合并的字段,连接符) within group：多行合并成一行:

select name,listagg(course||':'||score,',') within group(order by name) as courses from kecheng group by name

==多表查询==

在from后面跟上多张表即可

组函数count()：select count(\*) from emp;查询emp表的数据量

select \* from emp,dept;结果显示错误，笛卡尔积造成的(emp的数据量\*dept的数据量)

select \* from emp,dept where emp.deptno=dept.deptno;虽然消除了笛卡尔积，但笛卡尔积一直存在

select e.ename,e.empno,e.sal,s.grade from emp e,salgrade s where e.sal between s.losal and s.hisal;两张表没有相同的字段

select e.ename,e.empno,e.sal,decode(s.grade,1,'E等级工资',2,'D等级工资',3,'C等级工资'，4，'B等级工资'，5，'A等级工资') from emp e,salgrade s where e.sal between s.losal and s.hisal;

多表查询中每增加一张表都要消除笛卡尔积(表之间有相同的字段)

select e.ename,e.job,e.sal,d.dname,s.grade from emp e,dept d,salgrade s where e.deptno=d.deptno and e.sal between s.losal and s.hisal;

等值连接：

上面的e.deptno=d.deptno用的内连接(如果emp表中的员工的deptno是null，则不会显示这条数据)

内连接[inner] join：作用于natural join相同，但语法不同(需要写关联字段)

作用：只返回两个表中连接字段相等的行

select \* from dept d join emp e on(e.deptno=d.deptno);

外连接：与where条件中的字段左右顺序有关，与from后面表的顺序无关

1)左外连接：显示左表所有数据(比如员工1没有部门编号，显示该员工，dept表中部门1没有员工不显示他)

select \* from emp e,dept d where e.deptno=d.deptno(+);

2)右外连接：显示右表所有数据

select \* from emp e,dept d where e.deptno(+)=d.deptno;

自身关联：存在笛卡尔积

select e1.empno,e1.ename,e1.mgr,e2.empno,e2.ename from emp e1,emp e2 where e1.mgr=e2.empno(+);

举例 ：查询在1981年雇佣的全部员工的编号，姓名，日期(按年-月-日显示)，工作，领导姓名，年工资，工资等级，部门编号，名称，位置，同时要求

这些雇员的月工资在1500-3500之间，按照工资降序排序，相同按工作排序

select e1.empno,e1.ename,e1.hiredate,e1.job,e2.ename,e1.sal,e1.sal\*12+nvl(e1.comm,0) 年薪,d.deptno,d.dname,d.loc,s.grade

from emp e1,emp e2,dept d,salgrade s

where to\_char(e1.hiredate,'yyyy')='1981' and

e1.sal between 1500 and 3500 and

e1.mgr=e2.empno(+) and

e1.sal between s.losal and s.hisal and

e1.deptno=d.deptno

order by e1.sal desc,e1.job;

内连接与等值连接的区别：

内连接后面的条件可以不等(比如 on e.deptno <> p.deptno)，等值连接必须是相等，从集合的角度上来看等值连接是内连接的子集

==SQL：1999语法的支持==

支持所有数据库

交叉连接(cross join)：第一张表每个数据与第二张表所有数据产生连接(笛卡尔积)

select \* from emp cross join dept;

自然连接(natural join)：消除两张表的笛卡尔积(两张表有共同字段)

select \* from emp natural join dept;

using子句：设置连接字段

select \* from emp join dept using(deptno);

on子句：设置连接条件

select \* from emp e join salgrade s on(e.sal between s.losal and s.hisal);

\*外连接

(+)只能Oracle使用

左外连接:(主要显示emp表)

select \* from emp e left [outer] join dept d on(e.deptno=d.deptno);

右外连接:

select \* from emp e right [outer] join dept d on(e.deptno=d.deptno);

全外连接(外连接中只有全外连接只能用SQL1999实现，但用的比较少)：

select \* from emp e full [outer] join dept d on(e.deptno=d.deptno);

数据的集合运算：查询结果的操作

集合运算时一种二目运算符，一共四种：并，差，交，笛卡尔积

并集union(不显示重复的)：select \* from dept union select \* from dept where deptno=10;

并集union all(显示重复的)：

尽量使用union/union all代替or：两个单表查询效率高

差集minus(效率很高)：select \* from dept minus select \* from dept where deptno=10;

交集intersect：select \* from dept intersect select \* from dept where deptno=10;

注：union拼接时第一个SQL查询的字段不能有聚合函数(可在外面套一层)

==分组统计查询==

统计函数：

count(\*|[distinct]列)：求出全部的记录数

sum求和

avg平均值

max最大值

min最小值

median中间值：是大小不是顺序

variance方差

stddev标准差

实际上只有5个标准函数：前面5个

可以对数据类型进行转换(如日期)

count(\*)与count(字段)的差别：count(字段)不计入null

执行顺序：from,where,group by,select,order by

单字段分组统计：

求出每个部门的人数：select deptno,count(\*) from emp group by deptno;

统计每种职业的最低和最高的工资：select job,max(sal),min(sal) from emp group by job;

注意事项：

1 没有group by，select子句中出现统计函数后不能出现其他函数(错误的select job,count(\*) from emp)

2 统计查询中select子句中只能出现分组函数和分组字段(group by后面字段)，不能出现其他字段(去掉job:select count(job),ename,job from emp group by ename;)

3 统计查询中允许统计函数嵌套使用，但之后的select子句不允许出现任何字段(去掉select后的deptno：select deptno,max(avg(sal)) from emp group by deptno)

举例 ：

查询每个部门的名称，人数，平均工资，平均服务年限

select d.dname,count(e.ename),round(avg(e.sal),2),round(avg(months\_between(sysdate,e.hiredate)/12),2) 平均入职年份

from emp e,dept d where e.deptno(+)=d.deptno group by d.dname;

查询公司各个工资等级雇员的数量、平均工资

select s.grade,count(e.empno),round(avg(e.sal),2)

from emp e,salgrade s

where e.sal between s.losal and s.hisal

group by s.grade;

查询领取comm与不领取comm的平均工资(不用group by)

select '领取佣金',round(avg(sal),2) from emp where comm is not null union

select '不领取佣金',round(avg(sal),2) from emp where comm is null;

单字段分组统计：

group by后面可以有多个字段

select d.deptno,d.dname,d.loc,count(e.deptno) count,nvl(round(avg(e.sal),2),0) avg from emp e,dept d

where e.deptno(+)=d.deptno group by d.deptno,d.dname,d.loc order by d.deptno;

having子句：

\*执行顺序：from，where，group by,having,select，order by,

与where的区别：针对分组之后的条件，使用having(having执行顺序在group by之后)

where：分组之前使用(可以有group by)，不允许使用统计函数

having：分组之后使用(必须有group by)，允许使用统计函数

举例：至少有一个员工的所有部门编号、名称，并统计这些部门的平均工资

select d.deptno,d.dname,round(avg(e.sal),2) from emp e,dept d where e.deptno(+)=d.deptno group by d.deptno,d.dname having count(e.empno)>1;

显示非销售人员的工作名称以及从事同一工作的员工工资的总和，并且要满足从事同一工作的员工工资合计大于5000，结果按照工资的合计升序排序

select distinct job,sum(sal) sum from emp where job!='CLERK' group by job having sum(sal)>5000 order by sum;

==子查询==

可以返回的数据类型：单行单列，单行多列，多行单列，多行多列(临时表)

一般where子句(一般单行单列，单行多列，多行单列)，having子句(一般单行单列，同时为了使用统计函数操作)，from子句(一般多行多列)使用子查询较多(select也可使用)

1-1 where子句使用子查询(单行单列)

查询工资最低的员工的具体信息：select \* from emp where sal=(select min(sal) from emp);

1-2 where子句使用子查询(单行多列)

查询与SCOTT从事同一工作且工资相同的员工信息

select \* from emp where (job,sal)=(select job,sal from emp where ename='SCOTT') and ename<>'SCOTT';

1-3 where子句使用子查询(多行单列)：需要使用in、any、some或者all操作符

1）in

查询出于每个部门中不是最低工资相同的全部员工的信息

select \* from emp where sal not in(select min(sal) from emp group by deptno);

2）any

=any：基本同in(<>any返回所有数据，没意义)

select \* from emp where sal=any(select min(sal) from emp group by deptno);

>any：比子查询返回结果的最小的要大(还有>=any)

<any：比子查询返回结果的最大的要小(还有<=any)

3）同any(为了避免any的歧义出现)

4）all

<>all：等价于not in(=all不等价于in，没有任何数据返回)

>all：比子查询返回结果的最大的要大(还有>=all)

<all：比子查询返回结果的最小的要小(还有<=all)

1-4 where子句使用子查询(exists)：exists判断，如果是true显示数据，false不显示数据

select \* from emp where and sal>2000 exists(select \* from emp where sal=12445);//由于子句查询不到东西，所有不显示数据

select \* from emp where and sal>2000 exists(select \* from emp);//由于子句能查询，显示sal>2000的数据

2 having子句中使用子查询

查询部门编号、人数和平均工资，并要求这些部门的平均工资高于公司平均工资

select deptno,count(empno),round(avg(sal),2) from emp group by deptno having avg(sal)>(select avg(sal) from emp);

查询平均工资最高的部门的名称和平均工资

select d.dname,round(avg(e.sal),2) from emp e,dept d where e.deptno=d.deptno group by d.dname having

avg(e.sal)=(select max(avg(sal)) from emp group by deptno);

3 在from子句中使用子查询(注意子查询要使用别名)

4 在select中使用子查询(查询数据明细时增加一个统计值列)

select name,(select count(course) from kecheng where name=k.name group by name) as 个数 from kecheng k;--同分析函数有点相似

with as子句(Navicat不能用)

定义临时表，下面是将select name from temp1定义成了名为temp的临时表：

with

t1 as(select name from temp1), //定义临时表

t2 as(select name from temp1) //定义多个用逗号隔开

select \* from t2; //查询临时表

优点：表(temp1)只会扫描一次(上面只用到了t1)

分析函数

可以在数据中进行分组然后计算基于组的某种统计值，并且每一组的每一行都可以返回一个统计值(即查询数据同时每个数据返回一个统计值)

分析函数和聚合函数的不同之处是什么？

普通的聚合函数用group by分组，每个分组返回一个统计值，而分析函数采用partition by分组，并且每组每行都可以返回一个统计值。

ROWS BETWEEN unbounded preceding AND current row

加order by 字段(不加每个分组的数据是一样的，不用累计计算直接计算)：从每个分组的第一条数据开始，一直到最后一条(比如sum，第一个是第一个数据，第二个是前2个数据的和，，以此类推；比如max，第一个是第一个数据，第二个是前2个最大的一个数据，，以此类推。。。。。就是累计计算)

SELECT name,sum(SCORE)

over(PARTITION BY "NAME" ORDER BY SCORE ROWS

BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) MAX\_SAL FROM KECHENG;

等同于(上面没有between。。。的情况下)：

SELECT A.NAME,B.SUMVAL FROM KECHENG A,(SELECT NAME,SUM(SCORE) SUMVAL FROM KECHENG GROUP BY NAME ORDER BY NAME) B WHERE A.NAME=B.NAME ORDER BY A.NAME

行列转换

<https://www.cnblogs.com/liunanjava/p/4961923.html>

表中数据如下：



1 固定行列转换

* 1. decode方式(聚合函数用户max最好)

SELECT id,name,

SUM(decode(course,'语文',score,0)) 语文, //对course字段进行了5次判断，其中course=’语文’的那条结果是67，剩下的都是0，所以和是67

SUM(decode(course,'数学',score,0)) 数学,

SUM(decode(course,'英语',score,0)) 英语,

SUM(decode(course,'历史',score,0)) 历史,

SUM(decode(course,'化学',score,0)) 化学

FROM KECHENG GROUP BY id,name;

* 1. case when方式

SELECT ID,NAME,

MAX(CASE WHEN course='语文' THEN score ELSE 0 END) 语文, //SUM和MAX都行

MAX(CASE WHEN course='数学' THEN score ELSE 0 END) 数学,

MAX(CASE WHEN course='英语' THEN score ELSE 0 END) 英语,

MAX(CASE WHEN course='历史' THEN score ELSE 0 END) 历史,

MAX(CASE WHEN course='化学' THEN score ELSE 0 END) 化学

FROM kecheng

GROUP BY ID ,NAME;

* 1. wmsys.wm\_concat行列转换函数

注：11g和12c已经摒弃了该函数，如果别人用了该函数，只有通过手工创建该函数，来临时解决该问题，但是在使用时需要用to\_char(wm\_concat())方式，才能完全替代之前的应用

SELECT ID,NAME,

wmsys.wm\_concat(course || ':'||score) course

FROM kecheng GROUP BY ID ,NAME;

* 1. 使用over(partition by t.u\_id)用法

partition by。。。。。

动态行列转换

需要使用存储过程(下面使用创建视图的方法，也可创建表(前面需要查询该表是否存在，存在要删掉))

CREATE OR REPLACE PROCEDURE P\_TEST IS

V\_SQL VARCHAR2(2000);

CURSOR CURSOR\_1 IS

SELECT DISTINCT id,name FROM TEST ORDER BY id;--id(字符串)和name是对应的，这里将name转换成列(字段)

V\_CUR CURSOR\_1%ROWTYPE;

BEGIN

V\_SQL := 'SELECT time';

open CURSOR\_1

LOOP

FETCH CURSOR\_1 into V\_CUR;

EXIT WHEN CURSOR\_1%NOTFOUND;

V\_SQL := V\_SQL || ',' || 'COUNT(CASE WHEN id=''' || V\_CUR.id ||

''' then 1 else null end) AS ' || V\_CUR.name;

END LOOP;

CLOSE CURSOR\_1;

V\_SQL := V\_SQL || ' FROM TEST GROUP BY time ORDER time;

V\_SQL := 'CREATE OR REPLACE VIEW RESULT AS '|| V\_SQL; --select语句外不能套括号(可能是select里有括号的原因造成的)

--DBMS\_OUTPUT.PUT\_LINE(V\_SQL);

EXECUTE IMMEDIATE V\_SQL;

END P\_TEST;

层次查询connect by

<https://www.cnblogs.com/king-xg/p/6794562.html>

查询员工和他的lead的名字：

select emp\_id,lead\_id,emp\_name,prior emp\_name as lead\_name from employee

start with lead\_id=0 --emp\_id=1(祖节点)

connect by prior emp\_id = lead\_id; -- lead\_id = prior emp\_id

查询员工id是6以及他所有祖先节点

select emp\_id,lead\_id,emp\_name,salary

from employee

start with emp\_id=6

connect by prior lead\_id=emp\_id;

prior：父行限定符(父行的字段前需要加)

nocycle：如果有循环行要加上这个(结果会去重)，connect by后面

其他用法看链接

2 动态转化

PL/SQL

Pivot与unpivot函数(11g之后)：

SELECT \*FROM (select name,course,SCORE from KECHENG) pivot(

SUM(score)

FOR course IN(

'语文' AS 'yuwen',

'数学' AS 'shuxue'

)

) GROUP BY name;

下面的SQL语句显示内容为XML格式：

SELECT \*FROM (select name,course,SCORE from KECHENG) pivot(

SUM(score)

FOR course IN(ANY)

) GROUP BY name;

==rowid伪列(包含删除重复数据)==

查询rowid：select ''||ROWID,id,name from kecheng;//单独查不出来，要转换成字符串

rowid存在真实的表或视图中，不存在临时表、基表中

每行数据的地址编号，不会相同的，最早插入的数据rowid值最小

删除重复字段只保留一个(rowid最小的那条数据)：

delete from mydept where rowid not in(select min(rowid) from mydept group by deptno,dname,loc);//删除所有deptno重复，并且dname重复，并且loc重复的数据

(复制表：create table mydept as select \* from dept;)

第二种方法如下(没上面那种好用)：

单个字段重复数据

1-1 查询重复的数据(单个字段)

select peopleId from people group by peopleId having count(peopleId) > 1;

1-2 查询重复的数据(单个字段)

select peopleId,peoplename,count(\*) as num from people group by peopleId, peoplename having num > 1;

2 删除重复字段(最小的rowid除外)

delete from people

where peopleId in (select peopleId from people group by peopleId having count(peopleId) > 1)

and rowid not in (select min(rowid) from people group by peopleId having count(peopleId )>1)

11g闪回技术

闪回查询：

查询从2018-06-05 10:20:00之前的所有数据：SELECT \*FROM TEMP1 AS OF timestamp to\_timestamp('2018-06-05 10:20:00','yyyy-mm-dd hh24:mi:ss');

闪回表：

视图

也叫做虚表，不占物理内存(相对)，产生视图的表叫做基表，修改视图(有些视图)的数据也会改变基表的数据

作用：

1)可以使用各种不同的方式将基表的数据展现在用户面前, 以便符合用户的使用习惯

2)隐藏数据的逻辑复杂性并简化查询语句

3)执行某些必须使用视图的查询. 某些查询必须借助视图的帮助才能完成. 比如, 有些查询需要连接一个分组统计后的表和另一表, 这时就可以先基于分组统计的结果创建一个视图, 然后在查询中连接这个视图和另一个表就可以了；

4)安全性保证.可以让不同的用户看见不同的列,这样就可以保证敏感数据不被用户看见；

5)简化用户权限的管理. 可以将视图的权限授予用户, 而不必将基表中某些列的权限授予用户, 这样就简化了用户权限的定义。

创建视图：(1sysdba赋权grant create view to scott;)

create [ or replace ] [ force ] view [schema.]view\_name //force :强制创建视图，不考虑基表是否存在和使用基表的权限

[ (column1,column2,...) ] //视图的列名，个数必须与select查到的个数一样

as

select ...

[ with check option ] //增删改的数据, 必须是select查询所能查询到的数据(检查约束)

[ with read only ]; //创建的视图只能查询不能进行DML操作

例如：CREATE VIEW KECHENG\_VIEW (名字,课程,成绩) AS SELECT "NAME",COURSE,SCORE FROM KECHENG WITH READ ONLY;

对视图的 增删改查 同表

函数

--定义一个拼接功能的函数

CREATE OR REPLACE FUNCTION "GET\_PINGJIE"

--定义三个参数

(

N1 NUMBER,

N2 NUMBER,

N3 NUMBER

)

--查询结果返回类型

Return VARCHAR2

IS

--定义函数内部用到的变量

S0 VARCHAR2(400);

S1 VARCHAR2(40);

S2 VARCHAR2(40);

S3 VARCHAR2(40);

--运行计算

BEGIN

--给变量赋值

S0:='@';

S1:='第一级:';

S2:='\_第二级';

S3:='\_第三级';

--拼接功能的实现过程

IF N1 IS NOT NULL AND LENGTHB(N1)>0 THEN

S0:=S0 || S1 || N1;

END IF;

IF N2 IS NOT NULL AND LENGTHB(N1)>0 THEN

S0:=S0 || S2 || N2;

END IF;

IF N3 IS NOT NULL AND LENGTHB(N1)>0 THEN

S0:=S0 || S3 || N3;

END IF;

--返回的变量

RETURN S0;

END;

SQL使用：SELECT GET\_PINGJIE(12.5,null,14) FROM DUAL;

显示：

序列

CREATE SEQUENCE 序列名

INCREMENT BY 1 –每次增长1

START WITH 1 –从1开始

NOMAXvalue/MAXvalue –最大值

NOCYCLE/CYCLE –达到最大值是否循环

NOCACHE;

使用：序列名.nextval

==rownum伪列(分页查询)==

即行号(每行数据的编号，从1开始)

1. 查询前面10条数据

select \* from emp where ROWNUM<=10;

1. 从第6条开始，提取5条数据

select \* from (select rownum rn,empno,ename from emp where rownum <=10) emp2 where emp2.rn>5;

注：rownum只能查询前面多少行(包括1所以可以用rownum=1)，所以不能用rownum=2(大于1)或者rownum>n

==fetch分页(12c才支持)==

(放在SQL语句最后)

查询前5行：select \* from emp fetch first 5 row only;

查询第4行开始的2条数据：select \* from emp offset 3 rows fetch next 2 rows only;

按照百分比取得数据：fetch next百分比 percent rows only

==创建存储过程==

(命令行中结束运行时输入 /)

or replace：加上这句就是创建并覆盖

create [or replace] procedure pro\_name

[(argment[{IN|OUT}]Type,argment[{IN}OUT|IN OUT}]Type]

{IS|AS}

<类型.变量的说明>

BEGIN

<执行部分>

EXCEPTION

<可选的异常错误处理程序>

END;

create or replace procedure proname is

begin

insert into tabpro(ename,logdate) values(user,sysdate);

end;

调用存储过程：

1 SQL窗口：

begin

GETUSERS1('LMM','LMM');

end;

1. 命令窗口：exec pro\_name[(参数)]

https://blog.csdn.net/zdx1515888659/article/details/46237311

游标

作用：同jdbc中的ResultSet，将查询结果放在一个集合当中，一般配合存储过程使用

游标是将数据保存到内存中的，所以操作的数据量不能太大

create or replace procedure aa(var varchar2) is

定义变量名 定义变量类型;

cursor 游标名 is select语句(查询分组的字段);

变量名a 游标名%ROWTYPE; --变量a的类型与游标类型一直

Begin

WHILE开始时间(sysdate-1) < 结束时间(sysdate) LOOP ---一般可以不用WHILE

open 游标名;

loop

fetch 游标名 into 变量名a; --将游标的值赋给变量a

exit when 游标名%notfound; --当游标中没有数据时

查询统计值的SQL语句（sysdate-1<=时间字段<sysdate[按天统计]）

SQL（查询sysdate-1的数据数量是0,0下面if中执行插入，>0执行更新）

if 表达式1 then

sql语句(插入);--取出值：变量名.字段(游标后面的sql语句中select的字段)

end if;

if then 更新语句 end if;

commit;

end loop;

close 游标名;

开始时间 :=开始时间+1;---while的循环条件

end loop;---关闭while循环

end aa;

静态游标

1 隐式游标

通过使用名字SQL来访问，但是通过SQL游标名只能访问前一个DML操作或单行SELECT操作的游标属性。所以通常在刚刚执行完操作之后，立即使用SQL游标名来访问属性。游标的属性有四种，如下所示:

sql%found （布尔类型，默认值为null）

sql%notfound（布尔类型,默认值为null）

sql%rowcount(数值类型默认值为0)：游标所影响的行数

sql%isopen(布尔类型)

2 显式游标

1）声明游标

DECLEAR

CURSOR 游标名[(参数 参数类型…])]

IS SELECT语句;

2）打开游标

open 游标名[(实际参数..)];

3）提取数据

fetch 游标名 into 变量名1,变量名2…;//变量的个数和类型必须和SELECT语句中的字段一致

或

FETCH 游标名 INTO 记录变量;// 记录变量定义：变量名 表名|游标名%ROWTYPE

4）关闭游标：CLOSE 游标名;

提取数据：fetch或者for循环

For循环(显式)：

cursor c\_rate is select \* from exchangerate t where t.currentmonth is null;

begin

for p\_rate in c\_rate loop

for i in 1 .. 3 loop

insert into exchangerate(quarter, Description)

values(p\_rate.quarter, p\_rate.description);

end loop;

end loop;

For循环(隐式)：

begin

for p\_rate in (select \* from exchangerate t where t.currentmonth is null) loop

for i in 1 .. 3 loop

insert into exchangerate(quarter, Description)

values(p\_rate.quarter, p\_rate.description);

end loop;

end loop;

显示游标参数传递：不确定查询条件中的值时使用

declare

p\_rate exchangerate%rowtype;

cursor c\_rate(p\_quarter varchar2) --声明游标带参数

is

select \* from exchangerate t where t.quarter<=p\_quarter;

begin

open c\_rate(p\_quarter=>'2011Q3');--打开游标,传递参数值

loop

fetch c\_rate into p\_rate;

update exchangerate set rate=p\_rate.rate+1 where id=p\_rate.id;

exit when c\_rate%notfound;

end loop;

close c\_rate;

end;

动态游标(REF游标)

创建REF游标+游标变量:

declare

p\_rate EMP%rowtype;

type rate is ref cursor;--定义游标变量(可用于其他表)type rate is ref cursor RETURN EMP%TYPE(只能用于表EMP)

c\_rate rate; --声明游标变量

begin

open c\_rate for select \* from EMP t where t.quarter='2011Q3';--打开游标变量

loop

fetch c\_rate into p\_rate;--提取游标变量

update EMP set rate=p\_rate.rate+1 where id=p\_rate.id;

exit when c\_rate%notfound;

end loop;

close c\_rate;--关闭游标变量

end;

REF游标分类:

强类型：带RETURN返回类型

弱类型：不带RETURN返回类型

动态语句拼接：:n

OPEN cur FOR

'select \* from emp where sal>:1 order by sal desc'

--填充占位符

USING p\_salary;

DBMS\_OUTPUT.PUT\_LINE('薪水大于'|| p\_salary ||'的员工有：');--打印输出(调试)

Job

常用操作和间隔值：<https://www.cnblogs.com/Chestnuts/articles/7066333.html>

间隔值：要用’’包起来(包括函数)

创建定时job(每隔一定时间执行一次存储过程或sql语句)：

declare

job number;

BEGIN

DBMS\_JOB.SUBMIT(

JOB => job, /\*自动生成JOB\_ID\*/

WHAT => 'proc\_add\_test;', /\*需要执行的存储过程名称或SQL语句(注意冒号)\*/

NEXT\_DATE => sysdate+3/(24\*60), /\*初次执行时间-下一个3分钟\*/

INTERVAL => 'trunc(sysdate,''mi'')+1/(24\*60)' /\*每隔1分钟执行一次\*/

);

commit;

end;

查询job：select \* from all\_jobs/ user\_jobs;

删除job：

begin

dbms\_job.remove(40);--40是job的编号

end;

job\_queue\_processes 参数：

job\_queue\_processes参数越大消耗资源越多

（sys用户下）

show parameter job：查询job\_queue\_processes参数

ALTER SYSTEM SET JOB\_QUEUE\_PROCESSES=个数：修改job\_queue\_processes参数

如果job\_queue\_processes参数=1，一个存储过程被4个job定时在同一时间点执行，存储过程中设置了延迟60秒，每个job执行的间距为60秒，如果此时参数=4，job基本上同时执行的。

==索引==

专门用于数据库查询操作性能的手段，在Oracle之中为了维护这种查询性能，需要对某一类数据进行指定结构的排序。

select \* from emp where sal>2000;这个SQL语句执行后会扫描emp表中的所有数据，

打开跟踪器(查看SQL的执行计划on改成off即关闭)：

在conn sys/change\_on\_install as sysdba;中执行set autotrace on命令(此命令不能在pl/sql上运行，能在sql\*plus上运行)，之后执行select \* from scott.emp where sal>2000;可查看执行计划。

如果查询rowid的话，会更快一些，不会查询到所有数据，如select \* from emp where rowed=’…’;

索引组成部分如下

索引项头：存储了行数和锁的信息

索引列长度和索引列值：同时存在，定义了长度，在长度之后保存的是内容

Rowid：指向数据表中数据行的rowid，通过rowid找到完整数据

最好的索引是在高基数列上使用，表中的主键或者唯一约束会自动创建索引

查询所有索引对象：select \* from user\_indexes;// select \* from user\_ind\_columns;

删除索引：索引属于Oracle对象，由于自身需要数据结构的维护，所以一般占用较大的磁盘空间，并随着表的增长而增长，所以要删除不常用的索引，数据库性能也可通过固态硬盘等设备提升

drop index [用户名.]索引名;

**B树索引：**

create index [用户名.]索引名称 on[用户名.]表名称(列[ASC|DESC]);

降序索引：B树索引衍生产品语法相同，只不过在SQL中使用了降序排序

函数索引：B树索引衍生产品

举例create index emp\_ename\_ind on emp(lower(ename));

Sql：select \* from emp where lower(ename)=’smith’;

**位图索引：**



举例：create bitmap index emp\_deptno\_ind on emp(deptno);

Oracle查询性能优化

1. ORACLE采用自下而上的顺序解析WHERE子句,根据这个原理,表之间的连接必须写在其他WHERE条件之前, 那些可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾，尤其是“主键ID=？”这样的条件
2. ORACLE在解析的过程中, 会将'\*' 依次转换成所有的列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费更多的时间 。
3. 查询语句时由于全表扫描读取的数据多，尤其是对于大型表不仅查询速度慢，而且对磁盘IO造成大的压力，通常都要避免，而避免的方式通常是使用索引Index

--索引的优势和代价：

优势：提高检索数据的效率，提供了主键(primary key)的唯一性验证，在大型表中使用索引特别有效，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，一张表的索引一般不超过6个

代价：索引需要空间来存储,也需要定期维护, 每当有记录在表中增减或索引列被修改时, 索引本身也会被修改. 这意味着每条记录的INSERT , DELETE , UPDATE将为此多付出4 , 5 次的磁盘I/O . 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢。而且表越大，影响越严重。

使用索引注意的地方：

索引列上避免进行或使用null判断、<>和!=、or、in和not in、通配符%、表达式操作(四则运算)、函数操作，否则将会进行全盘扫描

PL/SQL编程

PL/SQL是对SQL语言存储过程语言的扩展。分为 数据库引擎部分 和 可嵌入到许多产品 （如C语言，JAVA语言等）工具中的独立引擎。可以将这两部分称为：数据库PL/SQL和工具PL/SQL。这里主要介绍 数据库PL/SQL内容。

与SQL的区别：

SQL没有变量，没有流程控制(分支，循环)，用户在执行多条SQL语句时，每条数据都是逐一发到数据库，而PL/SQL可以一次将多条数据发送到数据库，减少网络流量

打印(可用于调试)：dbms\_output.put\_line('test1='||test1);

开启打印(命令行运行pl/sql开启后才能打印)：set serveroutput on;

dbms\_output.enable()：默认输出超过2000字节不输出(可设置dbms\_output)

拼接SQL中两个’代表SQL中一个’(用于字符串)

拼接SQL：v\_sql=’select count(1) from ’||tablename||’where name=’’’||v\_cur||’’’ and …’…

执行拼接SQL(将值赋给v\_exist)：EXECUTE IMMEDIATE v\_sql INTO v\_exist

语法：

declare

--声明部分，定义变量，常量，游标

begin

--程序编写、SQL语句

exception

--处理异常

end;

v\_eno :-= &empno ：由键盘输入(不用声明empno)

定义常量：常量名 constant 数据类型 [not null] :=值;(不加constant是变量)

定义变量：变量名 emp.empno%type(与emp表中empno字段相同类型)

Rowtype：一条数据，用变量名.字段名取出各字段值

分支结构：

1 if then end if;

2 if then else end if;

3 if then elseif then … else end if;

case语句：同sql，pl/sql也可以使用case when ….语句

循环结构：

1 loop:循环

LOOP

循环执行的语句块;

EXIT WHEN 循环结束条件;

循环条件修改;(游标中可以没有这个)

END LOOP;

2 while循环

WHILE (循环结束条件) LOOP

循环执行的语句块;

循环条件修改;

EDN LOOP;

3 for循环

FOR 循环索引 IN [REVERSE] 下限 .. 上限 LOOP -- REVERSE表示反转，从上限开始循环

循环语句;

END LOOP;

EXIT：结束循环　　continue：结束本次循环

GOTO：

GOTO temp;

…

<<temp>>

直接跳到<<temp>>处，不执行中间的语句

其他

RAISE\_APPLICATION\_ERROR(-20600,'不能在周末修改表tb\_emp');：自定义错误

dbms\_lock.sleep(60)：存储过程延迟执行60秒

集合

1 定义记录类型

Declare声明：

TYPE emp\_type IS RECODE(

ename emp.ename%type,--定义成员类型，也可定义varchar2,number…

job emp.job%type :=’YUANGONG ’ --可以为成员复制

);

v\_emp emp\_type; --声明复合类型变量

调用：v\_emp.ename;…即可

注：rowtype只能根据已有的表决定复合类型，记录类型可以由用户自定义类型组成

2 集合

2-1 索引表

同程序数组区别：下标可以是字符串也可以是数字(可以是负的)，索引不一定要连续，不需要定义索引表

定义索引表：

TYPE 类型名称 IS TABLE OF 数据类型 [NOT NULL]

INDEX BY [PLS\_INTEGER | BINARY\_INTEGER] VARCHAR2(长度)]

动态SQL

1 使用excute immediate

excute immediate 动态SQL语句(字符串) using 绑定参数列表 returning into 输出参数列表;

2 使用动态游标

触发器

<https://www.cnblogs.com/sharpest/p/7764662.html>

语句级触发器：在某些语句执行前或执行后被触发

行级触发器：触发的表中的行数据改变时就会被触发一次

创建语法：

create [or replace] trigger触发器名 触发时间(before或after) 触发事件(insert,update,delete)

on 表名

[FOR EACH ROW]--对表的每一行触发器执行一次(没有这句是语句级)

[for each row]

[DECLARE]---同存储过程，用来声明变量

begin

pl/sql语句

end

create or replace trigger auth\_secure before insert or update or DELETE

on tb\_emp

begin

IF(to\_char(sysdate,'DY')='星期日') THEN

RAISE\_APPLICATION\_ERROR(-20600,'不能在周末修改表tb\_emp');

END IF;

END;

触发器实现ID自增长：

CREATE OR REPLACE TRIGGER MY\_TGR

BEFORE INSERT ON TAB\_USER

FOR EACH ROW--对表的每一行触发器执行一次

DECLARE

NEXT\_ID NUMBER;

BEGIN

SELECT MY\_SEQ.NEXTVAL INTO NEXT\_ID FROM DUAL;

:NEW.ID := NEXT\_ID; --:NEW表示新插入的那条记录

END;

查找触发器名：select trigger\_name from all\_triggers [where table\_name='XXX'];

查找触发器代码：select text from all\_source where type='TRIGGER' AND name='TRIGGER01';

删除触发器：DROP TRIGGER TRIGGER01;

包的定义及使用

将任何出现在块声明的语句(存储过程,函数,游标,类型,变量)放于包中,相当于一个容器.将声明语句放入包中的好处是:用户可以从其他PL/SQL块中对其进行引用,因此包为PL/SQL提供了全程变量

包分为两部分:包头和包体(创建了包头之后才能创建包体)

添加任意数量的测试数据

1. 创建一个表，并同时添加1000000条数据，代码：

create table TestTable as

select rownum as id,

to\_char(sysdate + rownum/24/3600, 'yyyy-mm-dd hh24:mi:ss') as inc\_datetime,

trunc(dbms\_random.value(0, 100)) as random\_id,

dbms\_random.string('x', 20) random\_string

from dual

connect by level <= 1000000;

2）在创建表后，原来表的基础上追加记录，比如在方法一创建的TestTable表中追加1000000条数据，代码：

insert into TestTable

(ID, INC\_DATETIME,RANDOM\_ID,RANDOM\_STRING)

select rownum as id,

to\_char(sysdate + rownum / 24 / 3600, 'yyyy-mm-dd hh24:mi:ss') as inc\_datetime,

trunc(dbms\_random.value(0, 100)) as random\_id,

dbms\_random.string('x', 20) random\_string

from dual

connect by level <= 1000000;

其他

1 查找表名为KC2的表是否存在(存在返回1不存在返回0)：select count(1) from user\_tables where table\_name='KC2';

问题

1 Select SUM(value) 和,max(4) 值 from temp where id=4; //Temp表中没有id=4的数据

显示和、值两个字段为null(查询的结果没有行，即没有rownum，所以max(4)的值加不进去)

1. SELECT K."NAME","COUNT"(DISTINCT K.SCORE) FROM KECHENG K LEFT JOIN EMPLOYEE E ON K."ID"=E.EMP\_ID GROUP BY K."NAME";

(关联表用聚合函数求数量)

K表和E表关联主要显示K表，此时想根据name分组求出每个name的数量，需要用distinct去掉和E表关联后多余的重复数据，但是K表中的SCORE成绩可能有重复的(例如语文数学都是50)，此时DISTINCT K.SCORE也会将他们去重，所以DISTINCT后面的字段必须是唯一值，可以改成DISTINCT K.rowid

3 A1表和A2表都是以A表为基础做的临时表，两表都有一个时间在两个参数之间 ，在能查到A1的数据可能查不到A2的数据(数据都是聚合函数得到的)，此时将两表拼在一起，只需用逗号 连接2表即可，select后面用别名点出数据即可

4 求2张表中name字段相同的数据

Select e.id,e.name,e.age from emp e where (select count(\*) from emp where name=e.name)>1;

5 SELECT ID,NAME,BM,CASE WHEN BM LIKE '1%' THEN (SELECT COUNT(1) FROM TEMP1 WHERE BM LIKE '1%') WHEN BM LIKE '2%' THEN (SELECT COUNT(1) FROM TEMP1 WHERE BM LIKE '2%') ELSE NULL END 数量 FROM TEMP1;



笔试题

一、有3个表

Student（SNO，SNAME, SEX）代表（学号，姓名，性别）

Cource（CNO，CNAME，CTEACHER）代表（课号，课名，教师）

Stu\_Cource（SNO，CNO，SCGRADE）代表（学号，课号，成绩）

问题：

1，找出没选过“黎明”老师的所有学生姓名。

2，列出2门以上（含2门）不及格学生姓名及平均成绩。

3，即学过1号课程有学过2号课所有学生的姓名。

4.检索每一门课程成绩都大于等于90分的学生学号、姓名和性别。

5.检索全部学生都选修的课程的课程号与课程名。

1）select Sname from Student where Sno not in( select Sno from Stu\_Cource where Cno in (select cno from Cource where Cteacher='黎明') )

SELECT sname FROM STUDENT WHERE SNAME NOT in (SELECT DISTINCT SNAME FROM STUDENT s INNER JOIN STU\_COURCE sc ON s.sno=sc.sno INNER JOIN COURCE c ON sc.cno=c.cno WHERE c.Cteacher = '黎明');

注：有时候不用关联表查询，只需一层层查出所需要的字段

2）SELECT s.SNAME 姓名,AVG(st.SCGRADE) 平均成绩 FROM STUDENT s INNER JOIN STU\_COURCE st ON s.SNO=ST.SNO WHERE s.SNAME in (SELECT TE.name FROM (SELECT s.SNAME name,"COUNT"(ST.SCGRADE) num FROM STUDENT s INNER JOIN STU\_COURCE st ON s.SNO=ST.SNO WHERE ST.SCGRADE<60 GROUP BY s.SNAME) te

WHERE TE.NUM>=2) GROUP BY s.SNAME;

注：先查出60分以下的所有数据再根据name分组求出每个人scgrade的数量作为临时表，查出此临时表scgrade数量大于等于2的名字作为一个子查询，再查询这些名字以及平均成绩

3）SELECT SNAME FROM (SELECT SNAME,COUNT(CNO) num FROM STUDENT s INNER JOIN STU\_COURCE st ON s.SNO=ST.SNO WHERE CNO='1001' OR CNO='1002' GROUP BY SNAME) te WHERE TE.num>=2;

4）SELECT \* FROM STUDENT WHERE SNAME IN (SELECT SNAME FROM (SELECT SNAME,MIN(SCGRADE) num FROM STUDENT s INNER JOIN STU\_COURCE st ON s.SNO=ST.SNO GROUP BY SNAME) TE WHERE num>=90);

5）SELECT CNO,CNAME FROM COURCE WHERE CNO IN (SELECT CNO FROM (SELECT CNO,"COUNT"(SNO) NUM FROM Stu\_Cource GROUP BY CNO) TE WHERE NUM=(SELECT "COUNT"(SNO) FROM STUDENT));