**Oracle表的创建和管理**

**1.Oracle支持的数据类型**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **作用** | **对应java数据类型** |
| VARCHAR2(length) | 可变字符串,不会用空格填充至最大长度; length表示最大长度，一般保存长度比较小的内容 | String |
| CHAR2(length) | 定长字符串,当位数不足自动用空格填充来达到其最大长度; length表示最大长度 | String |
| NUMBER( p,s ) | NUMBER(p):p表示精度（总长度),表示整数数据最多不能超过p个长度，可以使用INT代替 | Integer/int Double/Float /double/float; |
| NUMBER(p,s):p表示精度（总长度),表示整数为占p-s个长度； s表示小数位数且四舍五入，可以使用FLOAT代替 |
| Date | 保存日期时间数据 | Date |
| TIMESTAMP | 保存日期时间数据 | Date |
| CLOB | 大文本数据，最多可以保存4G的文字 |  |
| BLOB | 二进制数据，最多可以保存4G的内容，可以保存文字、图片、音乐、电影 |  |

**2.Oracle支持的约束**

|  |  |  |
| --- | --- | --- |
| **约束** | **语法** | **说明** |
| 主键约束 | primary key(字段) | 非空+唯一 |
| 唯一约束 | unique(字段) |  |
| 非空约束 | not null |  |
| 检查约束 | check(约束) |  |
| 外键约束 | constraint FK\_外键字段\_主键表 foreign key(外键列) references 主键表（主键） |  |

**3.创建表的语法**

**3.1语法**

CREATE TABLE 表名(

字段 数据类型 [DEFAULT 默认值][约束],

字段 数据类型 [DEFAULT 默认值][约束],

......

字段 数据类型 [DEFAULT 默认值][约束]

);

**3.2实例**

CREATE TABLE t\_b\_user(

id NUMBER primary key,

userName VARCHAR2(18),

pass VARCHAR2(18),

sex CHAR(1) DEFAULT '1'

);

**4.对表的管理**

**删除表的语法**

drop table 表名;

**添加字段**

语法

ALTER TABLE 表名 ADD 字段名 数据类型[约束];

实例

ALTER TABLE t\_b\_user ADD status varchar2(3) default '10';

添加字段后位字段添加注释

alter table emp add e\_hobby varchar2(50);

comment on column emp.e\_hobby is '爱好';

### 修改字段名称

语法

alter table 表名rename column 原字段 to 新字段;

**新增表记录语法**

insert into 表名 values('值1','值2',...);

insert into 表名(列1,列2,...)values('值1','值2',...);

**修改表记录语法**

update 表名 set 列=值 [where 条件];

**删除表记录语法**

delete from 表名 [where 条件];

**查询表记录语法**

select \* from 表名 [where 条件]

**5.创建约束的两种方式**

1.创建表的的时候创建约束(不推荐)；  
2.创建表之后单独创建约束;(但是一般主键在创建表的时候创建)

# Oracle基础查询

## I.简单查询

### 1.语法

SELECT

[DISTINCT]\*|列名称 [别名],列名称 [别名],...

FROM 表名称[别名];

### 2.执行顺序

1.FROM:通过FROM子句确定数据来源;

2.SELECT:通过SELECT子句完成数据的筛选操作;

### 3.实例：

#### 3.1.通配符\*

**1.使用通配符"\*",查询全部的列,使用指定列名方式查询部分列；**

1.查询emp表中的全部记录;

SELECT \* FROM emp;

2.查询每个雇员的编号、姓名、职位、工资;

SELECT empno,ename,job,sal FROM emp;

#### 3.2.DISTINCT

**2.使用DISTINCT消除重复内容,如果查询的数据是多个列，那么只有在这多个列的数据都相同的时候才可以消除;**

1.查询所有的职位信息

SELECT DISTINCT job FROM emp;

SELECT DISTINCT empno,job FROM emp;

#### 3.3.四则运算

**3.针对查询的列返回结果进行四则运算(先乘除再加减);**

1.查询每个雇员的编号、姓名、年薪;

SELECT empno,ename,sal \*12 income FROM emp;

**4.在使用SELECT自己查询时，设置一些常量;  
   a.如果常量是字符串,则要求使用单引号声明,如 'hello' ;  
   b.如果常量是数字,则直接编写,如10;  
   c.如果常量是日期,则按日期风格格式编写;**

#### 3.4.常量

常量

1.直接查询常量

SELECT '雇员',empno,ename FROM emp;

**5.如果需要可以在SELECT子句中使用连接符"||"连接查询结果，在存储过程中可能会用到;**

1.使用||转换显示格式

SELECT '雇员编号:'||empno||,姓名：||ename||,收入：||sal info FROM emp;

## II.限定查询

### 1.语法

SELECT

[DISTINCT]\*|列名称 [别名],列名称 [别名],...

FROM 表名称[别名]

[WHERE 过滤条件];

### 2.执行顺序

1.FROM:通过FROM子句确定数据来源;

2.WHERE:通过WHERE确定满足条件的数据行;

3.SELECT:通过SELECT子控制显示的数据列

### 3.限定查询符合

1.关系运算符：>,<,<=,>=,<>(!=);

2.逻辑运算符：AND,OR,NOT

3.范围运算符：BETWEEN...AND,IN,NOT IN;

4.空判断：IS NULL,IS NOT NULL

5.模糊查询：LIKE  
说明：  
1."="可以在数字上使用,也可以在字符串中使用;

2.ORACLE中数据是区分大小的;

### 4.实例

#### 4.1.关系运算符

1.查询工资高于1500的员工信息

SELECT \* FROM emp c WHERE c.sal>1500;

2. 查询SMITH的基本信息

SELECT \* FROM emp c WHERE c.ename ='SMITH';

3.查询工资是5000的雇员信息

SELECT \* FROM emp c WHERE c.sal = 5000;

4.查询职位不是销售人员的雇员编号、姓名、职位

SELECT empno,ename,job FROM emp WHERE job !='SALESMAN';

SELECT empno,ename,job FROM emp WHERE job <>'SALESMAN';

#### 4.2.逻辑运算符

1.查询工资高于1500低于3000的员工信息

SELECT \* FROM emp c WHERE c.sal>1500 AND c.sal<3000;

2.查询工资大于2000或者职位是办事员的所有雇员信息

SELECT \* FROM emp c WHERE c.sal>2000 OR job ='CLERK';

3.查询所有工资小于2000的雇员信息

SELECT \* FROM emp c WHERE c.sal<2000;

SELECT \* FROM emp c WHERE NOT c.sal>=2000;

#### 4.3.范围查询

1.范围运算符：BETWEEN 最小值(数字|日期)AND 最大值,在此最小值与最大值直接的内容都满足条件,包含最小值和最大值;

1.查询工资在1500到2000之间的雇员(包含1500和2000)

SELECT \*FROM emp c WHERE c.sal >=1500 AND c.sal<=2000;

SELECT \*FROM emp c WHERE c.sal BETWEEN 1500 AND 2000;

2.查询所有在1981年雇用的雇员;

SELECT \*FROM emp c WHERE c WHERE c.hiredate BETWEEN '01-1 月-81' AND '31-12 月-81';

#### 4.4.空判断

空判断:数据库上解释为不确定的内容；

1.如果在数字列上使用null绝对不表示0;

2.对应空判断不能使用关系运算符完成;

3.空的操作只能使用IS NULL或IS NOT NULL (NOT ...IS NULL)

查询领取佣金的雇员

SELECT \* FROM emp c WHERE c.comm IS NOT NULL;

SELECT \* FROM emp c WHERE NOT comm IS NULL;

#### 4.5.IN操作符

1.IN给定的是一个指定的可选范围;

2.如果不使用IN操作符可以使用或操作;

3.如果不在指定范围之中可以使用NOT IN;

查询出雇员编号是7369、7566、7788、9999的雇员信息

SELECT \* FROM emp c WHERE c.empno IN(7369,7566,7788,9999);

SELECT \* FROM emp c WHERE c.empno = 7369 OR c.empno = 7566 OR c.empno = 7788 OR c.empno = 9999;

查询出雇员编号不是7369、7566、7788、9999的雇员信息

SELECT \* FROM emp c WHERE c.empno NOT IN(7369,7566,7788,9999);

SELECT \* FROM emp c WHERE NOT c.empno IN(7369,7566,7788,9999);

#### 4.6.NOT IN 和NULL的问题

1.在使用NOT IN进行范围判断时，如果范围里面包含NULL,那么不会有任何的结果返回;

2.使用IN无任何影响;

#### 4.7.模糊查询 like

7.1.通配符   1."\_":匹配任意的一位字符；  
 2."%":匹配任意位的字符;  
说明：  
1.like可以用在各种类型上，不一定是字符串;  
2.在使用like模糊查询的时候，如果不设置查询关键字，那么表示查询全部;

1.在使用like查询时

查询姓名是以字母A开头的雇员信息

SELECT \* FROM emp c WHERE c.ename like 'A%';

查询姓名的第二个字母是A开头的雇员信息

SELECT \* FROM emp c WHERE c.ename like '\_A%';

查询姓名包含A字母的雇员信息

SELECT \* FROM emp c WHERE c.ename like '%A%';

### 总结

1.where子句控制显示的数据行的操作,select子句控制显示数据的列的操作; 2.select子句后于where子句执行,所有在select中定义的别名，无法在where子句中使用; 3.各个限定查询的符号用法; 4.在使用NOT IN进行范围判断时，如果范围里面包含NULL,那么不会有任何的结果返回; 5.在使用like模糊查询的时候，如果不设置查询关键字，那么表示查询全部;

## III.排序查询

### 1.语法

select

[distinct]\*|列名称[别名],列名称[别名]...

from 表名 [别名]

[where] 条件

order by 字段[ASC(默认)/DESC,字段[ASC/DESC],....]

### 2.执行顺序：

1.FROM:通过FROM子句确定数据来源;

2.WHERE:通过WHERE确定满足条件的数据行;

3.SELECT:通过SELECT子控制显示的数据列

4.ORDER BY:对SELECT返回的数据进行排序;

### 3.ORDER BY说明

ORDER BY子句在SELECT子句后面执行,所有ORDER BY子句可以使用select子句定义的别名;

排序的方式有两种：

ASC:按升序方式排列，默认方式,不写也是ASC;

DESC:按降序方式排列;

### 4.实例

查询所有雇员信息，要求按工资由高到低

SELECT \* FROM emp ORDER BY sal DESC;

查询所有的销售人员的信息，要求按照雇员日期由早到晚的顺序排序;

SELECT \* FROM emp WHERE JOB='SALESMAN' ORDER BY hiredate ASC;

查询所有的销售人员的信息，要求按照工资由高到低的顺序排序，如果工资相同，按照雇员日期由早到晚的顺序排序;

SELECT \* FROM emp WHERE JOB='SALESMAN' ORDER BY sal desc,hiredate ASC;

查询每个雇员的编号、姓名、年薪，按年薪降序排序

SELECT empno,ename,sal \*12 income FROM emp ORDER BY income DESC;

### 5.总结

1.使用ORDER BY查询时，排序是最好完成的；  
2.ORDER BY子句是最后一个执行的子句；  
3.ORDER BY之中可以设置多个排序字段；  
4.ORDER BY有两种排序模式：ASC(默认),DESC；  
5.ORDER BY是唯一一个可以使用SELECT子句定义的别名的子句;

# Oracle中基本函数

[Oracle](http://localhost:9093/itour/work/detail?id=81) [SQL](http://localhost:9093/itour/work/detail?id=81)

阅读  最后发布于 2022-07-10 08:44

## 1.字符串函数

### 1.upper-字符串转位大写

语法:UPPER(字符串|列)

返回值：字符串;

说明：主要用于查询时用户输入的区分大小写的数据;一般会统一转小写或大写;

实例:SELECT UPPER('hello') FROM dual;

SELECT UPPER(ename) FROM emp;

### 2.lower-字符串转位小写

语法：LOWER(字符串|列)

返回值:字符串

说明：主要用于查询时用户输入的区分大小写的数据;一般会统一转小写或大写;

实例:SELECT LOWER('HELLO') FROM dual;

SELECT LOWER(ename) FROM emp;

### 3.initcap-首字母大写，其他字母小写

语法：INITCAP(字符串|列)

返回值：字符串

实例：SELECT INITCAP(ename) FROM emp c;

### 4.length-获取指定字符的长度

语法：LENGTH(字符串|列)

返回值：数字

实例：SELECT LENGTH(ename) FROM emp c;

### 5.substr-字符串的截取

在程序中所有的字符的首字母的索引都是0，在Oracle中，所有的字符的首字母的索引都是1，如果设置为0，也会按照1的方式处理;

语法：substr(字符串|列,开始索引 [,长度])

返回值：字符串

说明：1.进行字符串的截取,如果没有设置长度，表示从开始索引截取到结尾；

2.开始索引，从1开始，设置为0也会按1的方式处理；

3.开始索引设置为负数，截取的开始位置为字符串右端向左数第N个字符;

实例

实例：SELECT SUBSTR('helloworld',6) FROM dual;//world

说明:如果没有设置长度，表示从开始索引第一位截取到结尾,开始索引从1开始；

实例:SELECT SUBSTR('helloworld',0,5) FROM dual;//hello

SELECT SUBSTR('hello world',5,3) FROM dual;//o w

说明:开始索引，从1开始，设置为0也会按1的方式处理

实例: SELECT ename,SUBSTR(ename,LENGTH(ename)-2,3) FROM emp;

说明：1.截取ename的后三位;

2. 2.使用LENGTH(字符串|列)方法得到开始位置;

实例：SELECT ename,SUBSTR(ename,-3) FROM emp;---可以设置负数，-3

说明：1.截取ename的后三位;

2.设置为负数，截取的开始位置为字符串右端向左数第N个字符,Oracle特有;

### 6.replace-字符串替换

语法：REPLACE(字符串|列,旧字符串,新字符串 )

返回值：返回字符串

实例：SELECT REPLACE('hello oracle','oracle','world') FROM dual;//hello world

## 2.数值函数

### 1.round-四舍五入

语法：round(列|数字[,小数位])

返回值：数字

说明: 1.如果没有设置小数点的保留位数，那么会不保留小数直接进位；

2.如果设置了小数位的话，会在指定的小数位上进行四舍五入；

3.可以设置负数,表示进行整数位进行四舍五入;

实例：SELECT ROUND(788.5)FROM dual;//789

SELECT ROUND(726.6,-2) FROM dual;--700

### 2.trunc-数据截取,不进位

语法：trunc(列|数字 [,小数位])

返回值：数字

说明：数据截取,不进位;

实例： SELECT TRUNC(769.5641234),TRUNC(769.5641234,2),TRUNC(769.5661234,2) FROM dual;//769 769.56 769.56

### 3.mod -求模（求余数）

语法：mod(列|数字,列|数字)

返回值：数字

说明：求模（求余数）

SELECT MOD(10,3) FROM dual;//1

## 3.日期函数

### 1.获取当期日期

oracle提供了一个伪列 sysdate；

select sysdate from dual;

### 2.日期操作公式

1.日期+数字=日期(表示若干天之后的日期)

2.日期-数字=日期(表示若干天之前的日期)

3.日期-日期=数字（天数）  
说明：  
对于日期而言，由于每个月的天数是不同的,所以直接进行天数加法实现月数的计算是不准确的; 如需要查询每个雇员的已经被雇用的天数时，直接使用天数来实现年或月的计算,那么最终的结果一定是不准确的，为了实现准确的日期操作，oracle 提供了日期函数来处理日期；

### oracle提供的日期函数

#### 1.add\_months

语法：add\_months(列|日期,数字)

返回值：日期

作用：计算在指定日期上增加若干个月之后的日期

#### 2.months\_between

语法：months\_between(列|日期,列|日期)

返回值:数字

作用：计算两个日期之间所经历的月数

#### 3.last\_day

语法：last\_day(列|日期)

返回值：日期

作用：取得指定日期所在月的最后一天

#### 4.next\_day

语法：next\_day(列|日期,星期X)

返回值：日期

作用：计算下一个指定的一周时间数对应的日期

实例：

--查询4个月以后的日期

SELECT ADD\_MONTHS (SYSDATE,4) FROM dual;

--计算所有的雇员到今天为止的雇用月数

SELECT MONTHS\_BETWEEN(SYSDATE,hiredate) FROM emp;

--当前月份的最后一天日期

SELECT LAST\_DAY(SYSDATE) FROM dual;

--查询下个星期二的日期

SELECT NEXT\_DAY(SYSDATE,'星期二') FROM dual;

用年,月，日的方式计算出每个雇员到现在为止的雇用年限

1.每一位雇员到现在为止雇用的年份;

oracle 提供了两种方式计算年份

a.日期-日期=天数/365 （无法去除润年）

b.Months\_between(sysdate,hiredate)/12=年

SELECT TRUNC(MONTHS\_BETWEEN(SYSDATE,hiredate)/12) FROM emp;

2.求出雇佣的月数

当前时间和入职时间经历的约束除以12余下的就是剩下的月数+天数，截取小数位得到对应的月数

SELECT TRUNC(MOD(MONTHS\_BETWEEN(SYSDATE,hiredate),12)) FROM emp e;

3.求出雇佣的天数 （日期-日期=天数）

tip:由于时间跨度太长，必须回避年和月的问题

雇用日期+雇用日期到现在日期为止所经历的月数+天数=当前时间

SELECT TRUNC(SYSDATE -ADD\_MONTHS(hiredate,MONTHS\_BETWEEN(SYSDATE,hiredate))) FROM emp c;

tip:雇用日期+雇用日期到现在日期为止所经历的月数

## 4.转换函数

转换格式标记：  
     日期：年(yyyy) 月(mm),日(dd)  
     时间： 时（hh/hh24）分(mi) 秒(ss)  
     数字：任意数字（9） 本地货币符号（L）

### 1.to\_char

语法：to\_char(列|日期|数字,转换格式)

返回值：字符串

说明：将日期或数字格式化为指定的字符串

实例

1.转换日期为指定格式

SELECT TO\_CHAR(SYSDATE,'yyyy-mm-dd') FROM dual;

2.利用to\_char()实现日期的拆分

SELECT TO\_CHAR(SYSDATE,'yyyy'),TO\_CHAR(SYSDATE,'mm'),TO\_CHAR(SYSDATE,'dd') FROM dual;

3.查询在二月份雇用的雇员信息

SELECT \* FROM emp WHERE TO\_CHAR(hiredate,'mm')='02'

4.货币符合转换

SELECT to\_char(12345678,'L999,999,999') FROM dual; //￥12,345,678

### 2.to\_date

语法：to\_date(列|字符串,转换格式)

返回值：日期

说明：按照指定的转换格式编写字符串后将其变为日期型数据

实例

1.将字符串日期转为Date

SELECT TO\_DATE('2018-11-26','yyyy-mm-dd') FROM dual;

### 3.to\_number

语法：to\_number(列|字符串)

返回值：数字

说明：将字符串变为数字，Oracle提供了许多的自动转换机制，to\_number，TO\_DATE很少使用;

## 5.替换函数

### 1.nvl

语法：nvl(列，默认值)

返回值：数字

说明：如果传入的内容是null，则使用默认数值处理，如果不是空则使用原始数值处理

实例

--查询雇员的编号、姓名、值为、佣金（没有佣金显示0）

SELECT e.empno,e.ename,e.job ,nvl(comm,0)comm FROM emp e;

### 2.nvl2

语法：nvl2(列名,非空值,空值)

返回值：字符串

说明：如果传入的内容是null,则使用空值的方式处理，如果出入的内容不是null，则使用非空值的方式处理；

实例

1.查询雇员信息及工资(工资+佣金)

SELECT e.\* ,nvl2(comm,sal+comm,sal)salary FROM emp e;

说明：null+0=null;所以必须处理空的情况;

### 3.decode

decode（列|字符串|数值,比较内容1,显示内容1,...[,默认选择内容]） 返回值：字符串 说明：相当于 case when...then end，设置内容会和每个比较内容比较，如果内容相同则使用显示内容,如果不相同则使用默认选择内容显示; 实例

SELECT e.\*,

DECODE(job,'CLERK','办事员',

'PRESIDENT','总裁',

'MANAGER','经理',

'ANALYST','分析师',

'销售员'

)post

FROM emp e;

**Oracle多表查询**

[Oracle](http://localhost:9093/itour/work/detail?id=83) [SQL](http://localhost:9093/itour/work/detail?id=83)

阅读  最后发布于 2022-07-10 08:49

**1.多表查询语法**

SELECT [DISTINCT]\*|列[别名],....|常量

FROM 表名[别名],表名[别名]

[WHERE 条件]

[ORDER BY 字段 [ASC|DESC},字段 [ASC|DESC},...]

**2.执行顺序**

1.FROM:确定数据来源  
2.WHERE:对数据进行筛选  
3.SELECT:确定要使用的数据列  
4.ORDER BY:针对返回的结果进行排序

**3.笛卡儿积的处理**

**1.什么是笛卡尔积**

SELECT \* FROM emp,dept; 返回数据的条数=emp表数据条数 \* dept表数据条数，这种现象称为笛卡尔积问题；

**2.如何消除笛卡尔积**

1.想要消除笛卡尔积必须想办法为两张数据表设置关系;

SELECT \* FROM emp e,dept d WHERE e.deptno = d.deptno;

**3.相关说明：**

a.由于在进行多表关联查询时，有可能不同的数据表会存在相同的列名称，在这种情况下，需要在列的前面加上表明称或表的别名；在Oracle中，可使用AS为列取别名，使用As方式取别名时，AS可以省略，为表取别名是不可以使用AS关键字;

b.以上代码实际上只是消除了显示的笛卡尔积，而数据库的多表查询中，笛卡尔积会一直存在，只要存在数据表，就一定会存在有笛卡尔积；

c.一旦增加数据表之后需要增加消除笛卡尔积的关联字段或关联条件；

d.使用多表查询操作，永远都有一个前提，要关联的数据表一定要存在有关联字段或者是关联条件，否则一定不能使用多表查询;

**4.多表查询带来的问题：**

1.当总的数据量大的时候，执行的速度会很慢，多表查询的性能一定比较差，所以在开发中应该尽可能回避;

**5.实例**

--显示每个雇员的编号、姓名、职位、工资、部门名称、部门位置  
基本步骤:  
1.确定使用的数据表  
2.确定已知的关联字段

SELECT e.empno,e.ename,e.job,e.sal,d.dname,d.loc FROM emp e,dept d WEHRE e.deptno = d.deptno;

**4.数据表的连接操作**

在进行表连接的时候，有时需要设置表数据的主控方，主要有两种方式：  
1.内连接（等值连接）：所有满足条件的数据都会显示出来;  
2.外连接（左外连接|右外连接|全外连接）：控制左表与右表的数据是否全部显示；

**1.内连接**

内连接  
1.[INENER] JOIN|，

SELECT \* FROM emp e INNER JOIN dept d on e.deptno = d.deptno;

SELECT \* FROM emp e JOIN dept d on e.deptno = d.deptno;

SELECT \* FROM emp e,dept d WEHRE e.deptno = d.deptno;

说明：没有部门的雇员不会被查询出；

**2.外连接**

让等值判断左右两边有一边可以全部查询出来，如果想要实现外连接，则需要使用"(+)"语法来实现，(+)标记只有Oracle支持;  
1.左外连接  
字段 = 字段(+)  
2.右外连接  
字段(+) = 字段

左外连接,查询出所有的雇员信息，如果雇员没有对应的部门信息时，使用null来填充;

SELECT e.empno,e.ename,e.job,e.sal,d.dname,d.loc FROM emp e,dept d WEHRE e.deptno = d.deptno(+);

右外连接,查询出所有的部门信息，如果部门没有雇员时，使用null来填充；

SELECT e.empno,e.ename,e.job,e.sal,d.dname,d.loc FROM emp e,dept d WEHRE e.deptno(+) = d.deptno;

查询每个雇员的姓名、职位、领导姓名  
1.确定数据表  
2.确定已知的关联字段

SELECT e.ename,e.job ,memp.ename FROM emp e,emp memp WEHRE e.mgr(+) =memp.empno;

**5.SQL：1999语法支持**

(+)标记只有Oracle支持，其他数据库如何处理？  
可以使用SQL:1999语法来实现;

**1.SQL:1999语法定义的多表查询结果语法：**

SELECT [DISTINCT]\*|列[别名],....|常量

FROM 表名1 [CROSS JOIN tableName2] [NATURAL JOIN tableName2] [JOIN tableName2 on(条件)|USING（字段）] [LEFT|RIGHT|FULL OUTER JOIN tableName2]

[WHERE 条件]

[ORDER BY 字段[ASC|DESC],字段[ASC|DESC],...]

**2.相关说明**

1.交叉连接：CROSS JOIN，主要功能是产生笛卡尔积，简单的实现多表查询;

SELECT \* FROM emp e CROSS JOIN dept d;

2.自然连接：NATURAL JOIN，自动使用关联字段消除笛卡尔积(一般关联字段是外键，但此处它是以名称相同的字段)，属于内连接的概念;

SELECT \* FROM emp e NATURAL JOIN dept d;

3.USING子句：如果现在一张表里面有多个关联字段存在，那么可以使用USING子句明确的设置一个关联字段；

SELECT \* FROM emp e NATURAL JOIN dept d USING(deptno);

4.ON子句：如果没有设置关联字段，则可以使用ON子句设置条件,"()"括号可以省略；

SELECT \* FROM emp e NATURAL JOIN salgrade s on(e.sal BETWEEN s.losal AND s.hisal);

5.外连接：在SQL：1999语法标准里明确定义了数据表的左外连接、右外连接、全外连接的操作； 左外连接：

SELECT \* FROM emp e LEFT OUTER JOIN dept d USING(deptno);

右外连接:

SELECT \* FROM emp e RIGHT OUTER JOIN dept d USING(deptno);

全外连接：

SELECT \* FROM emp e FULL OUTER JOIN dept d USING(deptno);

**6.数据的集合操作**

**1.简介**

数据的集合操作主要负责连接的是查询结果，对于查询结果的集合操作主要提供了四种操作符：UNION,UNION ALL、INTERSECT、MINUS,利用这几个操作符可以实现多个查询语句的连接;

**2.对查询的要求**

数据集合查询相当于多个查询连接为一个查询结果返回，那么就有一个明确的要求，多个查询返回的列的结构必须相同；

**3.语法：**

SELECT [DISTINCT]\*|列[别名],....|常量

FROM 表名1 [CROSS JOIN tableName2] [NATURAL JOIN tableName2] [JOIN tableName2 on(条件)|USING（字段）] [LEFT|RIGHT|FULL OUTER JOIN tableName2]

[WHERE 条件]

[ORDER BY 字段[ASC|DESC],字段[ASC|DESC],...]

[UNION |UNION ALL|INTERSECT|MINUS]

SELECT [DISTINCT]\*|列[别名],....|常量

FROM 表名2 [CROSS JOIN tableName2] [NATURAL JOIN tableName2] [JOIN tableName2 on(条件)|USING（字段）] [LEFT|RIGHT|FULL OUTER JOIN tableName2]

[WHERE 条件]

[ORDER BY 字段[ASC|DESC],字段[ASC|DESC],...]

.....

**4.执行顺序**

1.FROM:确定数据来源  
2.WHERE:对数据进行筛选  
3.SELECT:确定要使用的数据列  
4.ORDER BY:针对返回的结果进行排序

**5.集合操作符的特点：**

UNION:将两个查询结果合并为一个查询结果返回，如果遇到重复的数据不会重复显示;

SELECT \* FROM emp e WHERE e.deptno ='10'

UNION

SELECT \* FROM emp e;

UNION ALL:将两个查询结果合并为一个查询结果返回，如果遇到重复的数据会重复显示；

SELECT \* FROM emp e WHERE e.deptno ='10'

UNION ALL

SELECT \* FROM emp e;

INTERSECT:回两个查询中的相同数据,属于交集操作；

SELECT \* FROM emp e WHERE e.deptno ='10'

INTERSECT

SELECT \* FROM emp e;

MINUS:返回差集，差集的计算是以一个集合减去第二个集合的数据集；

SELECT \* FROM emp e

MINUS

SELECT \* FROM emp e WHERE e.deptno ='10';

**总结：**

1.只要是多表查询永远存在笛卡尔积,所有优秀的系统设计时候觉得不可以出现多表查询操作;

2.多表连接查询，必须存在有关联字段或关联条件,否则无法进行多表查询

3.大部分情况下，使用的都是内连接，外连接oracle提供了"(+)"来操作

4.在Oracle以外的数据（sqlserver，mysql）需理由sql:1999语法实现;

5.可以使用集合操作将多个查询结果合并到一起显示，但是要求，多个查询的返回的列的结果相同;

**Oracle分组统计查询**

[Oracle](http://localhost:9093/itour/work/detail?id=85) [SQL](http://localhost:9093/itour/work/detail?id=85)

阅读  最后发布于 2022-07-10 08:51

**1.统计函数**

在SQL中定义了5个常用的统计函数

统计个数：count(\*|[distinct]字段|常量)

最大值:max(字段（日期或数字）)

最小值：min(字段（日期或数字）)

求和：sum(数字字段)

平均值：avg（数字字段)

说明：

1.max()和min（）可以使用数字或日期字段；

2.sum()和avg()只要数值类型字段可以使用;

相关实例：

--查询员工的最高工资和最低工资

SELECT MAX(sal),MIN(sal) FROM emp e;

--查询最晚和最早被雇佣的员工日期

SELECT MAX(e.hiredate),MIN(e.hiredate) FROM emp e;

--统计所有员工的总工资已经平均工资

SELECT SUM(e.sal),AVG(e.sal) FROM emp e;

---统计员工的平均服务年限

SELECT TRUNC(AVG(MONTHS\_BETWEEN(SYSDATE,e.hiredate))/12) FROM emp e;

**2.count(\*),count(字段),count(distinct 字段)的区别**

1.count(\*):明确的返回表中的数据个数,是最准确的;

2.count(字段)：不统计为null的数据个数,如果某一列的数据不可能为空,那么结果与count(\*)相同;

3.count(distinct 字段):统计消除重复数据后的数据个数;

4.COUNT(常量) 和 COUNT(*)表示的是直接查询符合条件的数据库表的行数,推荐使用count(*);

**3.什么情况下可能分组？如何实现分组？**

1.)能分组说明部分数据具备某些共性；

2.）可以使用GROUP BY子句实现分组；

**4.分组语法结构**

SELECT

[DISTINCT] 分组字段[别名],....|统计函数

FROM 表名1

[CROSS JOIN tableName2] [NATURAL JOIN tableName2]

[JOIN tableName ON(条件)|USING(字段)] [LEFT|RIGHT|FULL OUTER JOIN tableName2]

[WHERE 条件]

[GROUP BY 分组字段,分组字段,...]

[HAVING 分组后的过滤条件]

[ORDER BY 字段[ASC|DESC,...]]

**5.SQL执行顺序**

1.FROM:确定数据来源;  
2.WHERE:针对数据行进行筛选;  
3.GROUP BY:针对数据实现分组操作;  
4.HAVING:针对分组后的数据进行过滤;  
5.SELECT:确定要使用的数据列;  
6.ORDER BY:针对返回的结果进行排序;

**6.简单实例**

1.按照职位分组,统计每个职位的名称,人数，平均工资;

SELECT job,COUNT(\*),AVG(sal) FROM emp GROUP BY job;

2.要求查询出每个部门编号，以及每个部门的人数、最高和最低工资

SELECT c.deptno,COUNT(\*),MAX(sal),MIN(sal) FROM emp c GROUP BY c.deptno;

**7.分组统计的限制**

1.在没有编写group by子句的时候(全表作为一组）,那么select子句中只允许出现统计函数，不允许出现任何的其他字段;

错误实例：SELECT COUNT(\*),ename FROM emp;

正确实例：SELECT COUNT(\*) FROM emp；

2.在使用group by字句分组的时候,select子句之中,只允许出现分组字段和统计函数,其他字段不允许出现;

错误实例：SELECT job,COUNT(\*),ename FROM emp e GROUP BY job;

正确实例：SELECT job,COUNT(\*) FROM emp e GROUP BY job;

3.统计查询允许嵌套查询，但是嵌套后的统计查询中，select子句里不允许再出现任何的字段，包括分组字段,只能使用嵌套的统计函数;

错误实例：SELECT e.deptno,MAX(AVG(sal)) FROM emp e GROUP BY deptno;

正确实例：SELECT MAX(AVG(sal)) FROM emp e GROUP BY deptno;

**8.SQL编写基本步骤**

I.确定数据表  
II.确定已知的关联字段或关联条件

**9.多表分组查询简单实例**

1.查询每个部门的名称，平均工资，人数  
a.查询每个雇员的编号、部门名称  
b.进行分组统计

SELECT

d.dname,avg(sal),count(\*)

FROM emp e ,dept d

WHERE e.deptno(+)=d.deptno

GROUP BY d.dname;

2.查询每个部门的部门编号，部门名称，位置，部门人数，平均服务年限  
a.查询每个部门的部门编号，部门名称，位置,雇员编号，雇员日期  
b.进行分组统计

SELECT

d.deptno,d.dname,d.loc,COUNT(\*),TRUNC(AVG(MONTHS\_BETWEEN(SYSDATE,e.hiredate)/12))

FROM dept d, emp e

WHERE d.deptno(+)=e.deptno

GROUP BY d.deptno,d.dname,d.loc；

说明：多字段分组时，只有多个列的数据完全重复的时候才可以使用；

**10.where和having的区别**

1.where 发生在Group by操作之前,属于分组前的数据筛选，where子句不允许使用统计函数;（从所有的数据中筛选出可以分组的数据）

2.having 发生在Group by 操作之后，属于分组后的数据筛选;having子句可以使用统计函数;

**11.简单实例实例**

1.查询出平均工资高于2000的职位和平均工资  
错误实例：

SELECT e.job,AVG(sal)

FROM emp e

WHERE AVG(sal)>2000

group by e.job;

正确实例

SELECT

e.job,AVG(sal)

FROM emp e

GROUP BY e.job

HAVING AVG(sal)>2000;

2.显示非销售人员工作名称以及从事同一工作雇员的月工资的总和,并且满足从事同一工作雇员的月工资的总和高于￥5000，输出结果按工资的合计升序排列;  
a.显示非销售的人员信息  
b.根据工作名称分组统计月工资的总和；  
c.筛选出月工资总和高于5000的记录；  
d.对数据进行排序

SELECT e.job,SUM(sal) num

FROM emp e

WHERE job!='SALESMAN'

GROUP BY job

HAVING SUM(sal)>5000

order by num asc;

3.统计所有领取佣金和不领取佣金的雇员人数和平均工资  
错误实例：

SELECT e.comm,COUNT(\*),AVG(sal) FROM emp e GROUP BY comm;

总结：在分组的时候需要注意只有基数（列的值）固定的时候才可以分组；

正确实例:  
a.查询出领取佣金的雇员人生和平均工资 b.查询没有领取佣金的雇员人生和平均工资

SELECT '领取佣金' title, COUNT(empno),AVG(sal) FROM emp e WHERE e.comm IS NOT NULL

UNION

SELECT '不领取佣金' title,COUNT(empno),avg(sal) FROM emp e WHERE e.comm IS NULL;

**总结：**

1.不管是单表分组还是多表分组，重点看重复列,如果是一个重复列在group by后加一个字段，如果有多个重复再group by后添加多个字段;

2.分组中的使用限制:  
a.分组函数嵌套后不允许出现任何字段;  
b分组查询可以单独使用,如果在使用group by字句分组的时候,select子句之中只允许出现分组字段和统计函数,其他字段不允许出现;

3.多表查询与分组统计查询的时候相当于是一张临时表,所有的分组是在这种临时表中完成的;

# Oracle子查询

[Oracle](http://localhost:9093/itour/work/detail?id=87) [SQL](http://localhost:9093/itour/work/detail?id=87)

阅读  最后发布于 2022-08-28 20:57

## 1.什么是子查询

子查询是指在一个查询中继续嵌套其他的查询语句;

## 2.子查询使用方法：

1.出现where 子句上：  
子查询一般返回单行单列，单行多列，多行单列数据；

2.出现在having子句上：  
子查询会返回单行单列，同时表示要使用统计函数；

3.出现在From子句上：  
子查询返回多行多列数据（表结构）

4.select子句上：  
返回单行单列，一般不使用;

## 3.出现where 子句上

### 1）出现在where子句上--返回单行单列

---查询低于公司平均水平的员工信息(where 子句返回单行单列)

SELECT \* FROM emp WHERE sal<(SELECT AVG(sal) FROM emp e);

---查询最早雇佣的员工(where 子句返回单行单列)

SELECT \* FROM emp e WHERE e.hiredate = (SELECT MIN(hiredate) FROM emp);

### 2）出现在where子句上--返回单行多列

查询和scott从事同一工作，且工资相同的雇员信息(where子句返回单行多列，实际中很少使用)

SELECT \* FROM emp

WHERE (job,sal)=(SELECT job,sal FROM emp WHERE ename='SCOTT') AND ename!='SCOTT'

### 3）出现在where子句上--返回多行单列

如果子查询返回的内容是多行单列的话就表示一个操作数据的范围,对应此类的操作 SQL提供了三个操作符 in ,any,all;

#### 3.1.IN操作

##### 1）IN：与子查询返回的内容完全相同的数据；

SELECT sal FROM emp WHERE sal IN( SELECT sal FROM emp WHERE job='MANAGER')

##### 2）NOT IN 操作;

如果使用的是NOT IN ，子查询中不能放回null，如果返回了null,不会返回任何数据;

SELECT \* FROM emp WHERE empno NOT IN (SELECT mgr FROM emp);

#### 3.2.any 操作

##### 1.=any:功能和in一样

SELECT \* FROM WHERE sal=any(

SELECT sal FROM emp WHERE job='MANAGER');

##### 2.>any: 比子查询返回的最小值要大

SELECT \* FROM WHERE sal>any(

SELECT sal FROM emp WHERE job='MANAGER')

##### 3.<any:比子查询返回的最大值要小;

SELECT \* FROM WHERE sal<any(

SELECT sal FROM emp WHERE job='MANAGER')

#### 3.3.ALL

##### 1.>all：比子查询返回的最大值要大

SELECT \* FROM WHERE >all(

SELECT sal FROM emp WHERE job='MANAGER');

##### 2.<all：比子查询返回的最小值要小

SELECT \* FROM WHERE sal <all(SELECT sal FROM emp WHERE job='MANAGER');

## 4.在Having中使用子查询

一般返回单行单列的数据只在where和having语句中出现；  
如果子查询在having中出现，只有一种情况返回单行单列数据且需要进行统计函数计算；

查询高于公司平均工资的的职位名称，职位人生，平均工资

SELECT job,COUNT(\*) count,AVG(sal) avgsal

FROM emp

GROUP BY job

HAVING AVG(sal)>(SELECT AVG(sal)FROM emp);

## 5.在select中使用子查询一般不使用

会产生1+N次查询问题（每次查询出emp表的一条数据就需要读取查询下dept表），相当于发送1条查询指令，却发生了1+N次的查询， 一般不使用影响性能;

查询每个雇员的姓名，职位，部门名称

SELECT e.ename,e.job,(SELECT d.dname FROM dept d WHERE d.deptno=e.deptno )

FROM emp e;

## 6.在From子句中的子查询

1.如果子查询返回的结构是多行多列的而数据,那么就相当于是一张临时表的形式，所以可以直接在From子句中使用;

2.在from子句中出现子查询的另一种情况,在整个查询语句之中需要使用到统计函数，但是又无法直接使用统计函数统计的时候， 可以先在From子句中利用子查询实现统计;

查询出每个部门名称，位置，部门人数

方式1:

SELECT

d.dname,d.loc,count(empno) counts

FROM emp e,dept d ON e.deptno(+) = d.deptno

GROUP BY d.dname,d.loc

;

方式二：

SELECT

d.dname,d.loc,e.counts

FROM dept d ,(SELECT deptno, COUNT(\*) counts FROM emp e GROUP BY deptno) e

WHERE e.deptno(+)=d.deptno;

## 7. 多表查询和子查询都可以实现统计，那种方式更好呢？

1.如果emp表中有1400条记录，dept表有400条记录；  
**使用多表查询及分组统计：**  
emp表中的1400dept表中的400=560000条记录;**使用子查询**子查询的数据量：只使用了emp表中的1400条记录,最多返回400行记录子查询的返回数据量400dept表中的400=160000行记录  
两个操作加在一起最多产生160000条记录  
在实际的工作中，子查询的目的是解决多表查询所带来的性能问题,在开发中使用的是最多的;

## 总结

1.子查询，首先考虑的一定是where子句或from子句里面出现子查询的操作;

2.select基本不用，having子句出现的子查询只有在使用统计函数的时候才会使用;

3.子查询最大的作用是解决多表查询所带来的笛卡儿积影响的性能的问题;

4.复杂查询=简单查询+限定查询+多表查询+子查询

**Oracle子查询SQL编写**

[Oracle](http://localhost:9093/itour/work/detail?id=89) [SQL](http://localhost:9093/itour/work/detail?id=89)

阅读  最后发布于 2022-07-10 08:52

**1.编写子查询的步骤**

I.确定数据表;  
II.确定已知的关联字段;  
III.按条件逐步编写sql语句;

**2.实例**

1.列出薪资高于在部门30工作的所有员工的薪资的员工的姓名和薪资、部门名称，部门人数;

**I.确定数据表**

 emp e：员工姓名和薪资;

 dept d:部门名称;

 emp tmp:部门人数;

**II.确定已知的关联字段**  
 e.deptno = d.deptno

**III.逐步编写SQL语句**  
1）查询出在部门30工作的员工的薪资|查询出在部门30工作员工的最高薪资;--返回多行单列数据;

SELECT sal FROM emp e WHERE e.deptno='30';

SELECT max(sal) FROM emp e WHERE e.deptno='30';

2)查询出薪资高于部门30工作的员工的姓名和薪资  
如果返回的是多行单列的数据，只能在where子句中使用子查询；此时需要使用判断符号(in any all),应该使用>all进行数据筛选|不是用直接查出薪资大于部门30的最高工资的信息；

SELECT e.ename,e.sal FROM emp e WHERE sal>ALL(SELECT sal FROM emp e WHERE e.deptno='30');

SELECT e.ename,e.sal FROM emp e WHERE sal>(SELECT max(sal) FROM emp e WHERE e.deptno='30');

3）查询部门名称  
需要在FROM子句中引入dept表,一旦引入了新表（前提：存在关联字段或关联条件），需要在where条件子句中添加消除笛卡儿积的条件;

SELECT

   e.ename,e.sal,d.dname

FROM emp e,dept d

WHERE e.deptno = d.deptno AND sal>(SELECT max(sal) FROM emp e WHERE e.deptno='30')

4)统计部门人数  
对应部门人数可以先在子查询中完成统计；

SELECT tmp.deptno,COUNT(\*) deptcount FROM emp tmp group by tmp.deptno;

最终SQL

SELECT

   e.ename,e.sal,d.dname,tmp.deptcount

FROM emp e, dept d,(SELECT e.deptno,COUNT(\*) deptcount FROM emp e GROUP BY e.deptno) tmp

WHERE e.deptno = d.deptno and e.deptno = tmp.deptno and sal>(SELECT max(sal) FROM emp e WHERE e.deptno='30');

SELECT

   e.ename,e.sal,d.dname,tmp.deptcount

FROM emp e, dept d,(SELECT e.deptno,COUNT(\*) deptcount FROM emp e GROUP BY e.deptno) tmp

WHERE e.deptno = d.deptno and e.deptno = tmp.deptno and sal>All(SELECT sal FROM emp e WHERE e.deptno='30');

2.列出与"SCOTT"从事相同工作的所有员工的姓名，工资，部门名称，部门人数，领导姓名;  
**I.确定数据表**  
 1.emp e：员工姓名、工资  
 2.dept d：部门名称  
 3.emp tmp：部门人数  
 4.emp temp:领导姓名  
**II.确定已知的关联字段**  
e.deptno = d.deptno e.empno = temp.mgr **III.逐步编写SQL语句**  
1.查询出"SCOTT"从事的工作  
返回单行单列值,一般用于where或having子句上，但是现在没有统计需要，所有放到where子句中;

SELECT job FROM emp e WHERE e.ename ='SCOTT';

2.查询与"SCOTT"从事相同工作的所有员工的姓名，工资

SELECT e.ename,e.sal FROM emp e WHERE job = (SELECT job FROM emp e WHERE e.ename ='SCOTT');

3.查找部门的部门信息,加入dept表，同时添加消除笛卡儿积的条件

 SELECT e.ename,e.sal,d.dname FROM emp e,dept d WHERE e.deptno =d.deptno job = (SELECT job FROM emp e WHERE e.ename ='SCOTT');

4.查询部门人数,进行单独的统计操作，而后在Form子句之后使用

SELECT deptno，COUNT(\*) counts FROM emp tmp GROUP BY tmp.deptno;

SELECT

e.ename,e.sal,d.dname,tmp.counts

FROM emp e,dept d ,(SELECT deptno，COUNT(\*) counts FROM emp tmp GROUP BY tmp.deptno) tmp

WHERE e.deptno =d.deptno job = (SELECT job FROM emp e WHERE e.ename ='SCOTT') AND e.deptno= tmp.deptno;

5.查找出雇员信息对应的领导姓名，使用emp表实现自身关联操作

SELECT

e.ename,e.sal,d.dname,tmp.counts,temp.ename leader

FROM emp e,dept d ,(SELECT deptno，COUNT(\*) counts FROM emp tmp GROUP BY tmp.deptno) tmp,

emp temp

WHERE e.deptno =d.deptno job = (SELECT job FROM emp e WHERE e.ename ='SCOTT') AND e.deptno= tmp.deptno and e.empno =temp.mgr;

3.列出薪资比‘SMITH’ 或‘ALLEN’多的所有员工的员工编号，姓名，部门名称，其领导姓名，部门人数，部门平均工资，部门最高和最低工资；  
**I.确定数据表**  
 1.emp e:员工编号、姓名  
 2.dept d:部门名称  
 3.emp tmp:员工对应领导姓名  
 4.emp temp:部门人数、部门平均工资、部门最高和最低工资  
**II.确定已知的关联字段**  
e.deptno = d.deptno  
e.empno = tmp.mgr  
**III.逐步编写SQL语句**  
1.查询出‘SMITH’ 或‘ALLEN’的薪资，返回多行单列数据，应该在where子句中使用它；

SELECT sal FROM emp e WHERE e.ename in('SMITH','ALLEN');

2.查询薪资比‘SMITH’ 或‘ALLEN’多的所有员工的员工编号，姓名;

SELECT e.empno,e.ename FROM emp e WHERE e.sal>any(SELECT sal FROM emp e WHERE e.ename in('SMITH','ALLEN'));

3.查找部门的部门信息,加入dept表，同时添加消除笛卡儿积的条件

SELECT e.empno,e.ename,d.dname FROM emp e,dept d WHERE e.sal>any(SELECT sal FROM emp e WHERE e.ename in('SMITH','ALLEN')) AND e.deptno = d.deptno;

4.查找出雇员信息对应的领导姓名，使用emp表实现自身关联操作

SELECT

   e.empno,e.ename,d.dname,tmp.ename leader

 FROM emp e,dept d,emp tmp

 WHERE e.sal>any(SELECT sal FROM emp e WHERE e.ename in('SMITH','ALLEN')) AND e.deptno = d.deptno AND e.empno = tmp.mgr;

5.统计部门信息，进行单独的统计操作，而后在Form子句之后使用

SELECT deptno,COUNT(\*) dcount,AVG(sal) avgsal，MAX（sal）maxsal，MIN（sal）minsal FROM emp GROUP BY deptno;

SELECT

   e.empno,e.ename,d.dname,tmp.ename leader,temp.dcount,temp.avgsal,temp.maxsal,temp.minsal

 FROM emp e,dept d,emp tmp,(SELECT deptno,COUNT(\*) dcount,AVG(sal) avgsal，MAX（sal）maxsal，MIN（sal）minsal FROM emp GROUP BY deptno) temp

 WHERE e.sal>any(SELECT sal FROM emp e WHERE e.ename in('SMITH','ALLEN')) AND e.deptno = d.deptno AND e.empno = tmp.mgr AND e.deptno = temp.deptno;

4.查询受雇日期早于其直接上级的所有员工的编号，姓名，部门名称，部门位置，部门人数;  
**I.确定数据表**  
 1.emp e:员工的编号，姓名  
 2.dept d:部门名称、部门位置  
 3.emp tmp:部门人数  
 4.emp temp 其直接上级受雇日期，作为自身关联使用  
**II.确定已知的关联字段**  
 e.deptno =d.deptno  
 e.empno =d.mgr  
**III.逐步编写SQL语句**  
1.查询受雇日期早于其直接上级的所有员工的编号，姓名,实现emp表的自身关联

SELECT e.empno,e.ename FROM emp e,emp temp WHERE e.mgr = temp.empno(+) and e.hiredate<temp.hiredate;

2.查找部门的部门信息,加入dept表，同时添加消除笛卡儿积的条件

SELECT e.empno,e.ename,d.dname,d.loc

FROM emp e,emp temp,dept d

WHERE e.mgr = temp.empno(+) and e.hiredate<temp.hiredate AND e.deptno =d.deptno;

3.统计部门人数;

SELECT deptno,COUNT(\*) counts FROM emp GROUP BY deptno;

SELECT e.empno,e.ename,d.dname,d.loc,tmp.counts

FROM emp e,emp temp,dept d ,(SELECT deptno,COUNT(\*) counts FROM emp GROUP BY deptno) tmp

WHERE e.mgr = temp.empno(+) and e.hiredate<temp.hiredate AND e.deptno =d.deptno;

5.列出所有‘CLERK’（办事员）的姓名及部门名称，部门人数，工资等级;  
**I.确定数据表**  
 1.emp e:姓名  
 2.dept d:部门名称  
 3.emp tmp:部门人数  
 4.salgrade g：工资等级  
**II.确定已知的关联字段**  
e.deptno = d.deptno  
e.sal between g.losal and g.hisal  
**III.逐步编写SQL语句**  
1.查询所有‘CLERK’（办事员）的姓名

SELECT e.ename FROM emp e WHERE job ='CLERK';

2.查找部门的部门信息,加入dept表，同时添加消除笛卡儿积的条件

SELECT e.ename,d.dname FROM emp e,dept d WHERE job ='CLERK' and c.deptno = d.deptno;

3.统计部门信息

SELECT e.ename,d.dname tmp.couts

FROM emp e,dept d,(SELECT deptno, COUNT(\*) counts FROM emp c GROUP BY c.deptno) tmp

WHERE job ='CLERK' and c.deptno = d.deptno AND e.deptno = tmp.deptno;

4.工资等级

SELECT e.ename,d.dname tmp.couts,g.grade

FROM emp e,dept d,(SELECT deptno, COUNT(\*) counts FROM emp c GROUP BY c.deptno) tmp

WHERE job ='CLERK' and c.deptno = d.deptno AND e.deptno = tmp.deptno AND sal BETWEEN g.losal AND g.hisal;

**Oracle视图**

**1.什么是视图？**

视图可以理解为数据库中一张虚拟的表,质上是对表进行关联查询。视图的本身是不包含任何数据，只是一个查询结果，当基表的数据发生变化时，视图里面的数据也会跟着发生变化。

**2.视图作用**

1.简化查询操作;

**3.语法**

CREATE [OR REPLACE] VIEW view\_name

AS

SELECT查询

[WITH READ ONLY];

说明：

1.OR REPLACE：如果视图已经存在，则替换旧视图；

2.当视图对象被创建后会保存在user\_views数据字典中；

select \* from user\_views;

3.WITH READ ONLY：表示视图只读，不能更新；

4.建议使用只读视图，因为视图中的数据不是真实存在的数据；

**4.删除视图**

DROP VIEW 视图名称；

**Oracle之序列**

**1.序列的作用**

许多数据库都提供了自动增长列的操作，几乎所有的关系性数据库都是支持自动增长的操作的，Oracle比较特殊，只有在Oracle12c后才提供自动增长列，在之前类似的操作必须使用序列的方式处理；

**2.序列的创建语法**

CREATE SEQUENCE 序列名称

[INCREMENT BY 步长]

[START WITH 开始值]

[MAXVALUE 最大值 | NOMAXVALUE]

[MINVALUE 最小值 | NOMIMVALUE]

[CYCLE 缓存大小|NOCYCLE]

说明：大部分情况只需要创建默认的序列（步长为1,最小值为0，没有最大值，开始值为0，且设置有20个缓存数据）

实例：创建默认的序列

CREATE SEQUENCE SEQ\_T\_T\_USER;

说明：

1.序列对象创建后，会被保存到数据字典之中,如果需要查看可以使用"user\_sequences"

SELECT \* FROM user\_sequences;

sequence\_name:序列名称；

min\_value：最小值

max\_value：最大值；

increment\_by：步长；

cycle\_flag：循环标记；Y:循环；N:不循环

order\_flag：是否按顺序生成标记；

cache\_size：缓存量；

last\_number：最后一次操作的值；

**3.序列的使用**

Oracle在序列中提供了两个伪类进行操作；  
1.序列名称.NEXTVAL：表示进行序列的增长，每调用一次序列加上指定的步长；

2.序列名称.CURRVAL:表示获取当前序列的内容，重复调用多次，序列的内容不会有任何的变化；

3.在序列被创建后，严格来讲序列是无法直接执行CURRVAL，必须先执行NEXTVAL才可以使用CURRVAL；

SELECT SEQ\_T\_T\_USER.NEXTVAL FROM dual;

SELECT SEQ\_T\_T\_USER.CURRVAL FROM dual;

说明：只有执行了NEXTVAL序列才才进入可用状态；

4.所谓缓存，严格的来讲是缓存了20个数据，而最终的last\_number是利用缓存个数\*步长计算得来的； 如果是用了缓存有可能出现调号的问题；

5.如果序列想要为某个表的字段进行自动编号操作，那么只能利用insert语句完成；

INSERT INTO T\_USER(id,name) values(SEQ\_T\_T\_USER.NEXTVAL,'admin');

**4.序列的删除**

DROP SEQUENCE 序列名;

**Oracle 伪列（rownum和rowid）**

**1.ROWNUM**

**1.rownum可以做什么？**

在Oracle中RowNum可以做两件事情:  
1）取出第一条记录

SELECT \* FROM emp WHERE ROWNUM=1;

2）取出前N行记录；（分页）

a.查询出前N条记录；

b.将前N行数据当作临时表取出(start，end]行记录；

语法：

结束行:N=cureentPage\*pageSize

开始行:S=(cureentPage-1)\*pageSize;

SELECT \* FROM

(SELECT ROWNUM ROW\_ID ,TMP.\* FROM emp TMP WHERE rownum <=N)

WHERE ROW\_ID>s;

**2.ROWID**

在进行表中数据保存的时候，除了用户可以见到的数据列之外，所以的表中的数据的行都会有一个唯一的一块物理地址编号，这种编号可以通过rowid查找到。 任何情况下，都可以通过rowid确定唯一的一条;

**1.ROWID含义**

AAAR6E AAH AAAACb AAA  
数据的对象编号：AAAR6E  
数据保存文件编号：AAH  
数据保存的块号：AAAACb  
数据的保存行号：AAA

**2.使用**

1.删除所有重复记录，且只保留最早增加的一条记录

DELETE FROM dept WHERE ROWID NOT IN(

SELECT MIN(ROWID)FROM dept GROUP BY deptno,dname,loc

);