

Oracle 高级 SQL 培训与讲解

欢迎点击这里的链接进入精彩的[Linux公社](http://www.Linuxidc.com)网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](http://www.Linuxidc.com)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：[Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#) [Hadoop 专题](#)
[RedHat 专题](#) [SUSE 专题](#) [红旗 Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号：[linuxidc_com](#)

Linuxidc.com

微信扫一扫

订阅专业的最新Linux资讯及开源技术教程。

搜索微信公众号：[linuxidc_com](#)



1 With 子句

1.1 学习目标

掌握 with 子句用法，并且了解 with 子句能够提高查询效率的原因。

1.2 With 子句要点

1. with子句的返回结果存到用户的临时表空间中，只做一次查询，反复使用，提高效率。
2. 如果定义了with子句，而在查询中不使用，那么会报ora-32035 错误：未引用在with子句中定义的查询名。
3. 前面的with子句定义的查询在后面的with子句中使用。但是一个with子句内部不能嵌套with子句。
4. 当一个查询块名字和一个表名或其他的对象相同时，解析器从内向外搜索，优先使用子查询块名字。
5. with 查询的结果列有别名，引用的时候必须使用别名或*。
6. with 有可能影响执行计划。

1.3 with 子句语法



With alias_name **as** (select1), --as和select中的括号都不能省略

alias_name2 as (select2),--后面的没有with，逗号分割，同一个主查询同级别地方，with子查询只能定义一次

...

alias_namen as (select n) –与下面的实际查询之间没有逗号

Select

1.4with 使用例子:

1. 最简单的使用方法:

如查询部门名称包含“A”的所有员工信息

--with clause

```
with a as
(select deptno from dept where dname like '%A%')
select * from emp where deptno in (select * from a);
with a as
(select deptno from dept where dname like '%A%'), --a结果集
a2 as(select * from a where deptno>20) --a1结果集直接从a中筛选
select * from emp where deptno in (select * from a2);
```

2. 多层同级只能用一个with，并且后面的结果集可以使用前面的结果集：
查询部门名称包含“A”并且部门编号大于20的所有员工信息

```
with a as
(select deptno from dept where dname like '%A%'), --a结果集
a2 as(select * from a where deptno>20) --a1结果集直接从a中筛选
select * from emp where deptno in (select * from a2);
```

3. 不同级查询可以使用多个with：

查询部门名称包含“A”并且部门编号大于20的所有员工信息的另外一种实现方式如下

```
with a as
(select deptno from dept where dname like '%A%') --a结果集
select * from emp where deptno in ( --括号内层作为子查询，为第二级
with a2 as(select * from a where deptno>20) --a1结果集直接从a中筛选
select * from a2
);
```

1.5 使用场景

那什么情况下能使用到 with 子句呢？以下我就举几个简单的例子，简单的说明以下：

1. 我想测试一句 sql，而我不想专门建立一个测试表：

我想测试成绩大于 90 的学生，我不想建立学生表，可以用到 with 子句

```
with stu as(
select '张娜' sname,99 score from dual union
select '王杰' ,35 from dual union
select '宋丽' ,85 from dual union
select '陈晓' ,73 from dual union
select '李元' ,100 from dual
) --with 组成一个临时的结果集，存放在用户的临时表空间
select * from stu where score>90
```

2. 当一个 sql 重复用到某个相同的结果集作为子查询：

--查询销售部工资>1500 或者销售部工资小于1000 的员工

```
select * from emp where deptno=(select deptno from dept where dname
='SALES') and sal >1500
union all
select * from emp where deptno=(select deptno from dept where dname
```

```
='SALES') and sal <1000
--以上sql select deptno from dept where dname ='SALES'需要执行两次，影响效率
--可以使用with优化一下
with salno as(select deptno from dept where dname ='SALES')
select * from emp where deptno=(select * from salno) and sal >1500
union all
select * from emp where deptno=(select * from salno) and sal <1000
```

2 集合操作

2.1 学习目标

掌握union,union all,minus,intersect的使用,能够描述集合运算，了解内部运行原理。

2.2 要点

Operator	Returns	content
UNION	由每个查询选择的所有不重复的行	并集不包含重复值，默认按第1个查询的第1列升序排列
UNION ALL	由每个查询选择的所有行，包括所有重复的行	包括所有重复的行 完全并集包含重复值，不排序
INTERSECT	由每个查询选择的所有不重复的相交行	交集，不包含重复行，按第1个查询的第1列升序排列
MINUS	在第一个查询中，不在后面查询中的行	不包含重复行，按第1个查询的第1列升序排列

Union all 效率一般比union高。Union all内部不做排序工作，也不做剔除重复行工作，而union则做这个工作。所以当数据量比较大的时候，能用union all的时候尽量用union all。除了union all 默认不做排序和剔除重复行的操作外，union,minus, intersect都默认按第1个查询结果的第1列进行升序排列，并且不包含重复行。

2.3 语法

```
(select resource 1)
Union/union all/minus/intersect
(select resource 2)
```

Union/union all/minus/intersect

(select resource 3)

.....

其中查询结果集的各个字段的类型能够互相兼容，并且总的结果集字段名与第一个结果集相同。

2.4 使用案例

数据准备：

```
create table t1 as select rownum rn from dual connect by rownum<7;  
create table t2 as select rownum+3 rn from dual connect by rownum<7;
```

select * from t1

	RN
1	1
2	2
3	3
4	4
5	5
6	6

select * from t2

	RN
1	4
2	5
3	6
4	7
5	8
6	9

1. 查询 t1 和 t2 表的所有记录，不去除重复。

```
select rn r1 from t1--结果集的别名是r1  
union all  
select rn from t2
```

	R1
1	1
2	2
3	3
4	4
5	5
6	6
7	4
8	5
9	6
10	7
11	8
12	9

2. 查询 t1 和 t2 表的所有记录，去除重复。

```
select rn r1 from t1--结果集的别名是r1
union
select rn from t2
--结果集自动排序，以第一个字段为准
```

	R1
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

```
select rn r1 from t1--结果集的别名是r1
union
select rn from t2
--结果集自动排序，以第一个字段为准
```

	R1
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

3. 查询 t1 和 t2 表都存在的记录

```
select rn r1 from t1
intersect
select rn from t2
```

	R1
1	4
2	5
3	6

4. 查询 t1 表存在，t2 表不存在的记录

```
select rn r1 from t1
minus
select rn from t2
```

	R1
1	1
2	2
3	3

5. 排序操作:

```
select rn r1 from t1
minus
select rn from t2
order by r1 desc--排序只能放在最后一个结果及后面
```

	R1
1	3
2	2
3	1

6. 除了 union all 其他的都会在总的结果集中剔除重复，例如:

```
insert into t1 values(1);
commit;
```

现在 t1 表中有两条相同的记录，其 rn 的值为 1。

在进行集合运算时重复的记录被剔除:

```
select rn r1 from t1
minus
select rn from t2
```

	R1
1	1
2	2
3	3

2.5 使用场景

当要对多个结果集进行集合操作时，可是使用集合操作。

3 case 与 decode

3.1 学习目标

会使用 case 表达式和 decode 函数，理解各个参数和返回值的含义。

3.2 要点

Case 表达式：

1. When 后面的表达式类型应该全部保持一致，返回值类型也必须保持一致，或者能够进行隐式转换。
2. case 表达式 when 值，如果值是null，就算表达式也是null，结果也是返回false。也就是case 后面的表达式如果值为null，不会与when null 匹配，只会与else 匹配。

Decode函数的使用方法与case when相似，但是decode只能用等号匹配。

3.3 语法

Case 表达式第一种：

```
case exp when comexp then returnvalue
..when comexp then returnvalue
Else
Returnvalue
End
```

Case 表达式第二种：

```
case when Boolean then returnvalue
..when Boolean then return value
Else
Returnvalue
End
```

Decode 函数：

```
decode(exp,
value1,res1,
value2,res2,...,
valuen resn,
elsevalue)。
```

3.4 使用案例

Case 第一种用法：

```
select rn,
case sign(rn-3) --此处为表达式
when -1 then '小于3'
when 0 then '等于3'
else '大于3' end range
from t1
```

	RN	RANGE
▶ 1	1	小于3
2	2	小于3
3	3	等于3
4	4	大于3
5	5	大于3
6	6	大于3
7	1	小于3

Case 第二种用法:

```
--相当于if boolean then value elseif ...
select rn,
case
when rn<3 then '小于3'
when rn<5 then '3到5'
else '5以上' end range
from t1
```

	RN	RANGE
▶ 1	1	小于3
2	2	小于3
3	3	3到5
4	4	3到5
5	5	5以上
6	6	5以上
7	1	小于3

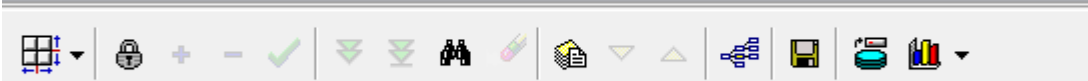
Decode 用法:

```
select rn,
decode(sign(rn-3),-1,'小于3',0,'等于3','大于3') range --此处注意参数个数及其参数怎么匹配
from t1
```

	RN	RANGE
▶ 1	1	小于3
2	2	小于3
3	3	等于3
4	4	大于3
5	5	大于3
6	6	大于3
7	1	小于3

上文提到过 null，碰到 null 的时候要注意，比如：


--此处表达式null与后面的null值不匹配, 返回值为2
select case null when null then '1' else '2' end nulltest from dual



	NULLTEST
1	2

这种情况可以这样处理:


--此处null is null 返回true, case表达式返回值是1
select case when null is null then '1' else '2' end nulltest from dual



	NULLTEST
1	1

如果用 decode 函数:

--此处null与null 匹配, decode返回值是1
select decode(null, null,1,2) nulltest from dual



	NULLTEST
1	1

3.5 使用场景

当我们的 sql 要求根据不同的条件返回不同的值时, 可以使用。

4 exists 与 in、not exists 与 not in

4.1 学习目标

掌握 exists 与 in 的、not exists 与 not in 的用法, 了解其内部的执行顺序 与执行原理, 知道什么情况下用 exists, 什么情况下用 in。

4.2 要点

1. Exists 用于只能用于子查询, 可以替代in, 若匹配到结果, 则退出内部查询, 并将条件标志为 true, 传回全部结果资料。
2. 若子查询结果集比较小, 优先使用in, 若外层查询比子查询小, 优先使用exists。因为若用in, 则oracle 会优先查询子查询, 然后匹配外层查询, 若使用exists, 则oracle 会优先

查询外层表，然后再与内层表匹配。最优化匹配原则，拿最小记录匹配大记录。

4.3 语法

In: Select select_fields from table_name where field_name in(select clause);

Exists: Select select_fields from table_name exists (select clause)

4.4 使用案例

查询员工部门编号在部门表中存在的员工记录:

```
select * from emp where deptno in (  
select deptno from dept--子查询先返回结果集，然后外层查询才能执行，与子查询的结果集匹配  
)
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1111	solaris			1984/7/1	1111.00		40
2	2222	suse			1976/7/1	2222.00		40
3	3333	ubuntu			1979/7/1	3333.00		40
4	2	ll	ll					10
5	7521	WARD	SALESMAN	7698	1981/2/22	1250.00	500.00	30
6	7566	JONES	MANAGER	7839	1981/4/2	2975.00		20
7	7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00	1400.00	30
8	7698	BLAKE	MANAGER	7839	1981/5/1	2850.00		30
9	7782	CLARK	MANAGER	7839	1981/6/9	2450.00		10
10	7788	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
11	7839	KING	PRESIDENT		1981/11/17	5000.00		10
12	7844	TURNER	SALESMAN	7698	1981/9/8	1500.00	0.00	30
13	7876	ADAMS	CLERK	7788	1987/5/23	1100.00		20
14	7900	JAMES	CLERK	7698	1981/12/3	950.00		30
15	7902	FORD	ANALYST	7566	1981/12/3	3000.00		20
16	7934	MILLER	CLERK	7782	1982/1/23	1300.00		10

以上语句可以用 Exist 替换:

```
--外层查询执行时，去判断子查询的记录存在不存在，如果存在就把当前行返回，否则跳入下一行判断
select * from emp where exists(
select 1 from dept where dept.deptno=emp.deptno
)
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1111	solaris			1984/7/1	1111.00		40
2	2222	suse			1976/7/1	2222.00		40
3	3333	ubuntu			1979/7/1	3333.00		40
4	2	ll	ll					10
5	7521	WARD	SALESMAN	7698	1981/2/22	1250.00	500.00	30
6	7566	JONES	MANAGER	7839	1981/4/2	2975.00		20
7	7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00	1400.00	30
8	7698	BLAKE	MANAGER	7839	1981/5/1	2850.00		30
9	7782	CLARK	MANAGER	7839	1981/6/9	2450.00		10
10	7788	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
11	7839	KING	PRESIDENT		1981/11/17	5000.00		10
12	7844	TURNER	SALESMAN	7698	1981/9/8	1500.00	0.00	30
13	7876	ADAMS	CLERK	7788	1987/5/23	1100.00		20
14	7900	JAMES	CLERK	7698	1981/12/3	950.00		30
15	7902	FORD	ANALYST	7566	1981/12/3	3000.00		20
16	7934	MILLER	CLERK	7782	1982/1/23	1300.00		10

另外 not in 和 not exists 在某些情况下也可以相互转换，但是要注意一点，not in 中的子查询返回的结果集包含 null 值的时候，查询会失效。例如我想查询对应员工记录数为 0 的部门。如下：

```
select * from emp where deptno is null--现在员工表中有两条记录部门编号是null
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	4444	fedore			1978/7/1	636.00		
2	5555	slacware			1986/7/1	2312.00		

用 not exists:

```
select * from dept where not exists(
select * from emp where deptno=dept.deptno--此时用not exists返回没有对应员工记录的部门
)
```

	DEPTNO	DNAME	LOC
1	60	LINUX	LINUX.D

以上语句不能用 not in 替换：

```
select * from dept where deptno not in(
select deptno from emp --此时用not in,因为子查询的结果集中包含null值，所以查询失效
)
```

	DEPTNO	DNAME	LOC

查询失效无记录返回。注意这并不是 oracle 的 bug，因为在 oracle 中 null 不表示空，而是表示未知，当使用 not in 的时候，如果子查询返回的结果集中包含 null 值，我们并不知道外层查询的记录在不在子查询返回的结果集之内，所以无记录返回。虽然这样，但是并不表示

not in 和 not exists 是完全不可以转换的，比如子查询所选的字段在对应的表中没有 null 值，这时 not in 和 not exists 是可以相互转换的。或者在某些情况下内层子查询加上 field_name is not null 限制条件也是可以的。

4.5 使用场景

当内层查询返回的结果集较小时，用 in 或者 not in 效率较高。当内层子查询返回的结果集比较大时，用 exists 或者 not exists 执行的效率较高。

5 行列互换

5.1 学习目标

掌握列转行技术和常用的行转列技术。

5.2 要点

行转列的情况有多种，不同的情况侧重点也不一样。

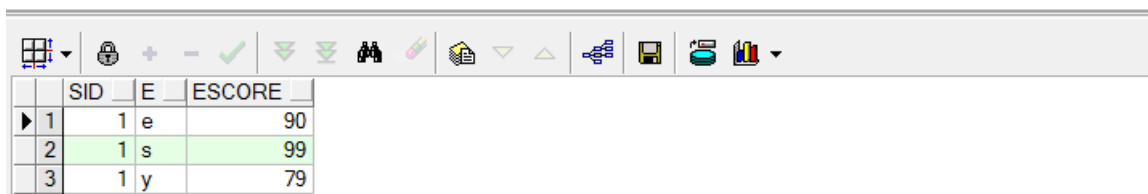
5.3 语法

5.4 使用案例

1. 列转行

第一种方法：需要用到 union 或者 union all:

```
--学生的英语、语文、数学成绩在一行上，我想把它拆成三行来分别记录该学生的三科成绩
with stu_score as(select 1 sid,'e' e,90 escore,'y' y,79 yscore,'s' s,99 sscore from dual)
select sid,e,escore from stu_score
union all
select sid,y,yscore from stu_score
union all
select sid,s,sscore from stu_score
order by sid
```



	SID	E	ESCORE
1	1	e	90
2	1	s	99
3	1	y	79

第二种方法：用到 model

```

with t as(select 1 sid,99 ys,88 ss,77 es from dual union all select 2,22,88,75 from dual )
select t,sc from t
model return updated rows
partition by(sid)
dimension by(0 as n)
measures('aaaaaaa' as t,100 as sc,ys,ss,es)
rules upsert all(
t[1]='语文',
t[2]='数学',
t[3]='英语',
sc[1]=ys[0],
sc[2]=ss[0],
sc[3]=es[0]
)

```

	T	SC
1	英语	77
2	数学	88
3	语文	99
4	英语	75
5	数学	88
6	语文	22

2. 行专栏，如我有 escore 表用来记录每个学生每个科目的成绩，如下：

```
select * from escore
```

	SID	TYPE	SCORE
1	1	e	20
2	2	y	33
3	2	s	22
4	2	e	
5	3	y	11
6	3	s	55
7	3	e	66
8	4	y	7
9	4	s	88
10	4	e	99

如果我想将每个学生的成绩统计在一行上，如：

3 语文 11 数学 55 英语 66

则我可以使用如下 sql：

```

select es.sid,
'英语',es.score,
'语文',ys.score,
'数学',ss.score from
(select * from escore where type='e') es,
(select * from escore where type='y') ys,
(select * from escore where type='s') ss
where es.sid=ys.sid(+) and ys.sid=ss.sid(+)

```

	SID	英语	SCORE	语文	SCORE	数学	SCORE
1	2	英语		语文	33	数学	22
2	3	英语	66	语文	11	数学	55
3	4	英语	99	语文	7	数学	88
4	1	英语	20	语文		数学	

这个 sql 表面上看没什么问题，但是仔细看一下三个结果集 es、ys 和 ss，他们来源于同一个表，而且查询方法也类似，都是根据 type 的值去筛选的，这样就会对 escore 表查询三遍，严重影响查询速率，那这个 sql 我们如何去优化呢！

首先在你的脑海里面要有一种思路，根据需求，原先每个学生成绩有多行记录，现在要显示到一行上，那一般情况下我们是需要根据学生分组的。所以 group by sid 这个是一定要有的，既然分组那我们可是使用 oracle 的聚合函数去求其他行的数据。至于科目字段目前都是已知的，也就是第 2,4,6 列显示的分别是英语、语文、数学这几个字，是常量，我们不用去考虑，那剩下的也就是最关键的，我们去求三科的成绩就可以了。

让我们再看一下 escore 表，当指针移到某一行数据时，当 type=e 时，我们就取到 score，加到第三列上，那第五列和第七列就加 0，也就是 sum(decode(type,'e',score,0))，其他列类似，这样 group by 时用到的聚合函数还有 decode 结合在一起使用，就可以完成我们的要求了，sql 写出来时这样的：

```

select sid,
'英语' e,sum(decode(type,'e',score)) es,
'语文' y,sum(decode(type,'y',score)) ys,
'数学' s,sum(decode(type,'s',score)) ss
from escore
group by sid

```

	SID	E	ES	Y	YS	S	SS
1	1	英语	20	语文		数学	
2	2	英语		语文	33	数学	22
3	4	英语	99	语文	7	数学	88
4	3	英语	66	语文	11	数学	55

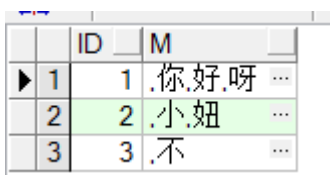
3. 字符串组合的多行转一列，例如我有一张测试表如下：


```
select * from mytest
```



	ID	NAME
1	1	你
2	1	好
3	1	呀
4	2	小
5	2	妞
6	3	不

我想根据 id 分组，将每一行的 name 连接起来，如下图是我想要的结果：



	ID	M
1	1	.你好呀 ...
2	2	.小妞 ...
3	3	.不 ...

这种行转列不是真正意义的行转列，是多行数据的值拼接后显示到一列上，那这种情况怎么处理呢，首先分析一下：多行 id 相同的值转换成一列，一般情况下需要用到 **group by**，但是对于字符串，**oracle** 中没有一个聚合函数适合用到此处的字符串连接，那该怎么办呢？

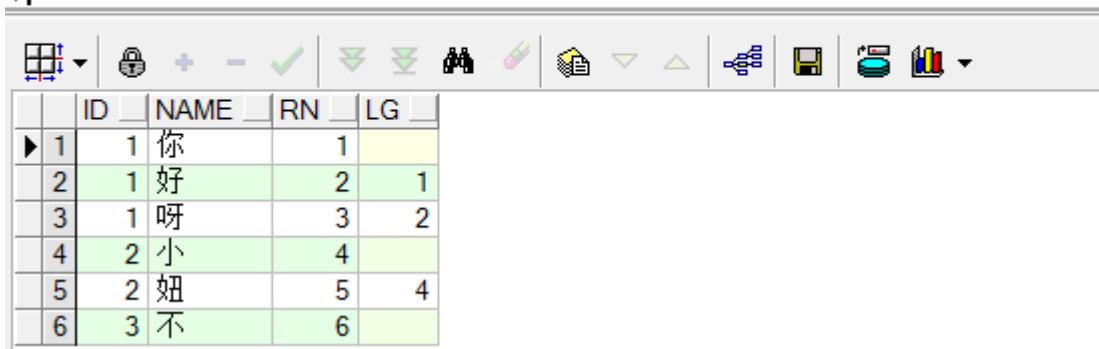
在 **oracle** 中，有 `sys_connect_by_path(field_name,concat_value)` 函数，可以通过 `connect by` 来依次连接每一行的数据，`connect by` 的语法是这样的：

`start with field1=1`--以当前表达式返回true的行开始

`connect by prior field2=field3`--通过当前行查找下一行，也就是说某一行数据的 `field3` 字段等于当前行的 `field2`，那就把这行数据作为下一行

有了这个思路，我们就可以用 `connect by` 通过使用 `sys_connect_by_path(field_name,concat_value)` 这个函数，并且根据 id 分组，将字符串连接在一起，然后通过 `max` 聚合函数，选出每组最长的字符串就可以了，那剩下的也就是最关键的问题就是我怎么去使用 `connect by`，通过当前行找到下一行呢？充分发散一下你的思维，看一下如下结果集：

```
select id ,name,rn,
lag(rn) over(partition by id order by rn) lg from (
    select id,name,row_number() over(order by id) rn from mytest
)|
```



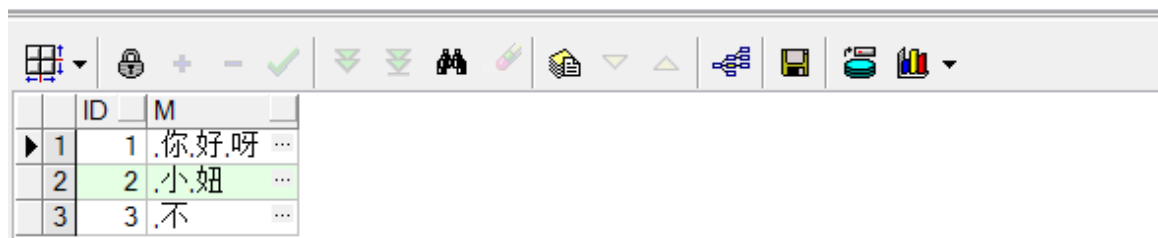
	ID	NAME	RN	LG
1	1	你	1	
2	1	好	2	1
3	1	呀	3	2
4	2	小	4	
5	2	妞	5	4
6	3	不	6	

那我下一步用如下思路使用 `connect by` 将所要的结果查询上来：

start with lg is null--以lg为null的行作为起始行
connect by prior rn=lg and prior id=id --当前行与其他行比较，满足这个条件的
就作为下一行数据

总的查询结果如下：

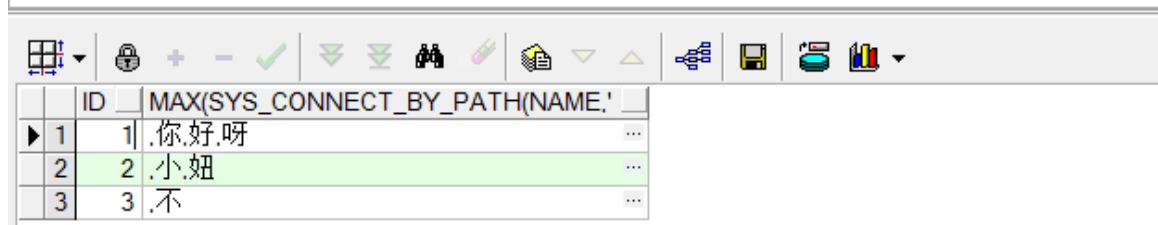
```
with t as(
  select id,name,rn,lag(rn) over(partition by id order by rn) lg from (
    select id,name,row_number() over(order by id) rn from mytest
  )
)
select id,max(sys_connect_by_path(name','')) m from t
start with lg is null
connect by prior rn=lg and prior id=id
group by id
```



	ID	M
▶ 1	1	.你好,呀 ...
2	2	.小姐 ...
3	3	.不 ...

其实怎么使用 connect by 方法很多，例如如下 sql 也能完成：

```
with t as(
  select id,name,row_number() over(partition by id order by id) rn from mytest
)
select id,max(sys_connect_by_path(name','')) from t
start with rn=1
connect by prior rn=rn-1 and prior id=id
group by id
```



	ID	MAX(SYS_CONNECT_BY_PATH(NAME,'
▶ 1	1	.你好,呀 ...
2	2	.小姐 ...
3	3	.不 ...

5.5 使用场景

当开发过程中，需要我们将多列转换成多行或者将多行转换成多列的时候，就需要用到行列转换，要根据不同的情况确定不同的结果方案，典型的行列转换就这几种，还有一种比较复杂的是不定不定行专列，不定行转列需要用到 oracle 的 package，在次先不做讲解。

6 merge into

6.1 学习目标

掌握 merge into，学会使用 merge into 批量处理数据。

6.2 要点

1. MERGE 语句是 Oracle9i 新增的语法，用来合并 UPDATE 和 INSERT 语句。
2. 过 MERGE 语句，根据一张表或子查询的连接条件对另外一张表进行查询，连接条件匹配上的进行 UPDATE，无法匹配的执行 INSERT。
3. 这个语法仅需要一次全表扫描就完成了全部工作，执行效率要高于 INSERT+UPDATE。

6.3 语法

```
MERGE [INTO [schema .] table [t_alias]
USING [schema .] { table | view | subquery } [t_alias]
ON ( condition )
WHEN MATCHED THEN merge_update_clause
WHEN NOT MATCHED THEN merge_insert_clause;
```

6.4 使用案例

1. 基于escore表创建escore2表，英语成绩每个同学加上5分的课时分，新增政治成绩，如下：

```
create table escore2 as select sid,type,score+5 score from escore where
type='e'
insert into escore2 values (1,'z',31);
insert into escore2 values (2,'z',45);
insert into escore2 values (3,'z',66);
insert into escore2 values (4,'z',76);
commit;
```

```
select * from escore2
```

	SID	TYPE	SCORE
1	1	e	25
2	2	e	
3	3	e	71
4	4	e	104
5	1	z	31
6	2	z	45
7	3	z	66
8	4	z	76

根据 escore2 表更新 escore 表，根据 sid 和 type 匹配，对于已经存在的记录进行更新操作，对于不存在的记录进行插入操作。

```
merge into escore t
```

```
using escore2 t2--此处可以是表、视图和查询结果集
```

```
on (t.sid=t2.sid and t.type=t2.type)--匹配条件，需要加括号
```

```
when matched then
```

```
update set t.score=nvl(t2.score,decode(t.type,'e',5,0))--根据匹配条件，更新escore
```

```
when not matched then
```

```
insert values(t2.sid,t2.type,t2.score)--无匹配条件的记录，插入新记录
```

执行 merge 以后的 escore 表如下：

```
select * from escore
```

	SID	TYPE	SCORE
1	1	e	25
2	2	e	5
3	3	e	71
4	4	e	104
5	1	s	88
6	2	s	98
7	3	s	35
8	1	y	76
9	2	y	58
10	3	y	75
11	4	s	
12	4	y	70
13	3	z	66
14	2	z	45
15	4	z	76
16	1	z	31

注意 update 的时候不能修改匹配的连接字段，否则就会报错。

6.5 使用场景

当要依赖别的表、视图或者结果集批量修改和插入目标表数据时，可以使用此方法，运行速率快，而且简单。

7 group by 高级特性

7.1 学习目标

学会使用 group by 语句，学会使用 group by 输出小计、合计。

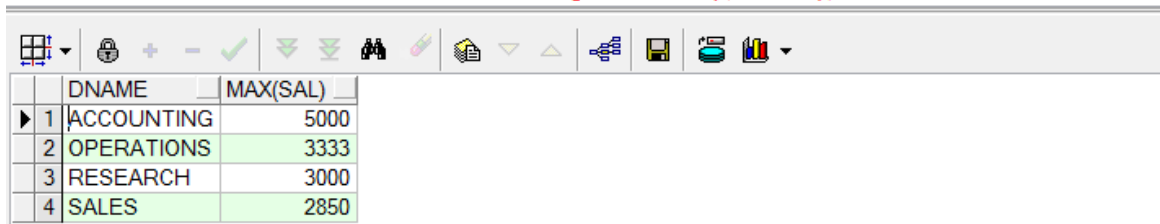
7.2 要点

1. 在 select 语句中可以使用 group by 子句将行划分成较小的组，然后，使用聚合函数返回每一个组的汇总信息。
2. 可以使用 having 子句限制返回的结果集。
3. 在带有 group by 子句的查询语句中，在 select 列表中指定的列要么是 group by 子句中指定的列，要么包含聚合函数语法。
4. 使用 rollup 操作符产生 subtotal(小计)的值，cube 操作符产生 cross-tabulation(列联交叉表)的值。
5. 使用 grouping 函数标识通过 rollup 和 cube 建立的行的值。
6. 使用 grouping sets 产生一个 single result set(结果集)。
7. 使用 grouping_id 和 group_id 函数。

7.3 使用案例

1. 查询部门员工的最高工资大于 1500 的部门，如下：

```
select dname,max(sal) from dept,emp --select的字段只能是group by后面的字段还有聚合函数
where dept.deptno=emp.deptno
group by dname having max(sal)>1500--having在分组之后筛选结果集
```



	DNAME	MAX(SAL)
1	ACCOUNTING	5000
2	OPERATIONS	3333
3	RESEARCH	3000
4	SALES	2850

相信这样的案例大家已经很熟悉了，以下讲解一下 group by 的高级特性。

2. Rollup 可以返回合计，例如：

rollup(a,b) 包括: (a,b)、(a)、()的合计

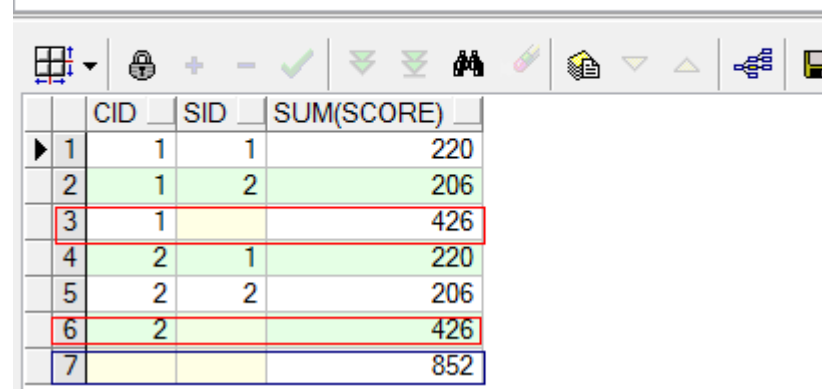
rollup(a,b,c) 包括: (a,b,c)、(a,b)、(a)、()的合计

我想按班级和学生分组, 查询班级下学生的总分, 并且做一下小计, 使用以上的 escore 表, 并且新建学生班级的关系表如下:

```
create table refcs as
select 1 cid,1 sid from dual
union all select 1,2 from dual
union all select 2,1 from dual
union all select 2,2 from dual
```

则查询的sql如下, 红色区域是每个班级的总分合适, 蓝色区域是所有的总分合计:

```
select cid,e.sid,sum(score) from refcs r,escore e
where e.sid=r.sid
group by rollup(cid,e.sid)
```



	CID	SID	SUM(SCORE)
1	1	1	220
2	1	2	206
3	1		426
4	2	1	220
5	2	2	206
6	2		426
7			852

3. cube 可以返回交叉的合计, 例如:

cube(a,b) 包括: (a,b)、(a)、(b)、()

cube(a,b,c) 包括: (a,b,c)、(a,b)、(a,c)、(b,c)、(a)、(b)、(c)、()

欢迎点击这里的链接进入精彩的[Linux公社](http://www.Linuxidc.com)网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](http://www.Linuxidc.com)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：[Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#) [Hadoop 专题](#)
[RedHat 专题](#) [SUSE 专题](#) [红旗 Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号：[linuxidc_com](#)



微信扫一扫

Linuxidc.com

订阅专业的最新Linux资讯及开源技术教程。

搜索微信公众号：[linuxidc_com](#)

```
select cid,e.sid,sum(score) from refcs r,escore e
where e.sid=r.sid
group by cube(cid,e.sid)
```

	CID	SID	SUM(SCORE)
1			923
2		1	220
3		2	206
4		3	247
5		4	250
6	1		426
7	1	1	220
8	1	2	206
9	2		497
10	2	3	247
11	2	4	250

与 rollup 相比，多了灰色区域，灰色区域是对分组的第二个字段 sid 的小计,用来统计每个学生的总分数，但在此是没有多大意义的，因为学生和班级是 1->n 的关系，统计每个学生的总分数和统计每个班级下每个学生的总分数没有区别。

4. GROUPING 函数可以接受一列，返回 0 或者 1。如果列值为空，那么 GROUPING()返回 1；如果列值非空，那么返回 0。GROUPING 只能在使用 ROLLUP 或 CUBE 的查询中使用。当需要在返回空值的地方显示某个值时，GROUPING()就非常有用了。

看如下 sql，红色区域和蓝色区域的 grouping 参数字段是 null，因此 grouping 字段返回 1，非 null 时返回 0：

```
select grouping(cid),cid,grouping(e.sid),e.sid,sum(score) from refcs r,escore e
where e.sid=r.sid
group by cube(cid,e.sid)
```

	GROUPING(CID)	CID	GROUPING(E.SID)	SID	SUM(SCORE)
1		1		1	923
2		1	0	1	220
3		1	0	2	206
4		1	0	3	247
5		1	0	4	250
6	0	1		1	426
7	0	1	0	1	220
8	0	1	0	2	206
9	0	2		1	497
10	0	2	0	3	247
11	0	2	0	4	250

但是返回 0 和 1 似乎没有多大意义，通常返回一些有意义的字符串可读性能好一些，如

下:

```
select decode(grouping(e.sid),0,cid||'|',cid||'班合计') cid,  
       decode(grouping(cid),0,e.sid||'|',e.sid||'的总分') sid,sum(score)  
from refcs r,escore e  
where e.sid=r.sid  
group by cube(cid,e.sid)
```

	CID	SID	SUM(SCORE)
1	班合计	的总分	923
2		1的总分	220
3		2的总分	206
4		3的总分	247
5		4的总分	250
6	1班合计		426
7	1	1	220
8	1	2	206
9	2班合计		497
10	2	3	247
11	2	4	250

其实你可以再优化一下，让第一行只显示“总计”两个字。

5. Grouping sets: 以上用 GROUP BY ROLLUP 或 GROUP BY CUBE 替代 GROUP BY，来计算高级的统计，不过它们会生成所有可能的总数，而你可能不需要全部总数，可以用 GROUP BY GROUPING SETS 来代替 GROUP BY CUBE。你可以应用来指定你感兴趣的总数组组合。因为它不必计算它不需要集合（也不会产生太多结果），所以对 SQL 引擎来说更为高效。例如：

```
select decode(grouping(cid),0,decode(grouping(e.sid),0,cid||'|',cid||'班合计'),'总计'),  
       e.sid,sum(score) s  
from escore e,refcs cs  
where e.sid=cs.sid  
group by grouping sets ((), (cid), (cid,e.sid))  
order by cid,sid nulls last
```

	DECODE(GROUPING(CID),0,DECODE(GROUPING(E.SID),0,CID ' ',CID '班合计'),'总计')	SID	S
1	班合计	1	220
2		2	206
3	1班合计		426
4		3	247
5		4	250
6	2班合计		497
7	总计		923

在图中我已经用不同颜色的边框和箭头指明某行数据来源于 grouping sets 之后的哪些集合。

7.4 使用场景

分组时常用。

8 分析函数

8.1 学习目标

掌握分析函数的使用，能用分析函数解决复杂查询等问题。

8.2 要点

1. 分析函数可分为四类：等级函数(ranking)，聚合函数(aggregate)，行比较函数(row comparison)，统计函数(statistical)。
- 2.

8.3 语法

函数名（参数）over(partition by ...order by...windows clause)

8.4 使用案例

1. Row_number, rank, dense_rank 属于等级函数，例如，我想根据部分分区，查询部门内部员工公司的排名，看一下用这三个等级函数会有什么区别：

```
select e.deptno,e.empno,e.ename,e.sal,
row_number()over(partition by deptno order by sal) rn,--以deptno分区，以sal排序计算row_number
rank()over(partition by deptno order by sal) r,|
dense_rank()over(partition by deptno order by sal) dr
from emp e
```

	DEPTNO	EMPNO	ENAME	SAL	RN	R	DR
1	10	7934	MILLER	1300.00	1	1	1
2	10	7782	CLARK	2450.00	2	2	2
3	10	7839	KING	5000.00	3	3	3
4	10	2	II		4	4	4
5	20	7876	ADAMS	1100.00	1	1	1
6	20	7566	JONES	2975.00	2	2	2
7	20	7788	SCOTT	3000.00	3	3	3
8	20	7902	FORD	3000.00	4	3	3
9	30	7900	JAMES	950.00	1	1	1
10	30	7521	WARD	1250.00	2	2	2
11	30	7654	MARTIN	1250.00	3	2	2
12	30	7844	TURNER	1500.00	4	4	3
13	30	7698	BLAKE	2850.00	5	5	4
14	40	1111	solaris	1111.00	1	1	1
15	40	2222	suse	2222.00	2	2	2
16	40	3333	ubuntu	3333.00	3	3	3
17	50	6666	opensuse	3333.00	1	1	1
18		4444	fedore	636.00	1	1	1

以deptno分区，作为一组，并且返回以sal排序的结果

注意观察选中的区域，可以发现什么问题呢？

我们发现三个函数都是返回分区排序后的序号，不同之处在于排序字段相同时，row_number 是从 1 到 n 连续不跳号的，rank 是给予值相同的两行相同的序号，而且跳

号，dense_rank 也是给予值相同的两行同样的序号，但是不跳号。

2. 常用分析函数：看一下以下一个 sql 中包含了多个常用的分析函数，图中需要注意的地方我已经圈出来并且标明了：

```
select e.deptno,e.empno,e.ename,e.sal,
first_value(ename)over(partition by deptno order by sal) fv,--以over后的计算方式取得某个字段在当前窗口的第一个值
last_value(ename)over(partition by deptno) lv,--取最后一个值
avg(e.sal)over(partition by deptno) vg,--取得当前窗口的平均值
max(sal) over(partition by deptno) m,--取的当前窗口的最大值
min(sal) over(partition by deptno) n,--取得当前窗口的最小值
sum(sal) over(partition by deptno order by sal) s,--取得当前窗口的合计值，加了order by当前窗口为第一行到当前行
count(empno)over(partition by deptno order by ename) c,--取得当前窗口的条数，加了order by当前窗口为第一行到当前行
from emp e
```

	DEPTNO	EMPNO	ENAME	SAL	FV	LV	VG	M	N	S	C
1	10	7934	MILLER	1300.00	MILLER II		2916.66666666667	5000	1300	1300	1
2	10	7782	CLARK	2450.00	MILLER II		2916.66666666667	5000	1300	3750	1
3	10	7839	KING	5000.00	MILLER II		2916.66666666667	5000	1300	8750	2
4	10	2	II		MILLER II		2916.66666666667	5000	1300	8750	4
5	20	7876	ADAMS	1100.00	ADAMS SCOTT		2518.75	3000	1100	1100	1
6	20	7566	JONES	2975.00	ADAMS SCOTT		2518.75	3000	1100	4075	3
7	20	7902	FORD	3000.00	ADAMS SCOTT		2518.75	3000	1100	10075	2
8	20	7788	SCOTT	3000.00	ADAMS SCOTT		2518.75	3000	1100	10075	4
9	30	7900	JAMES	950.00	JAMES BLAKE		1560	2850	950	950	2
10	30	7654	MARTIN	1250.00	JAMES BLAKE		1560	2850	950	3450	3
11	30	7521	WARD	1250.00	JAMES BLAKE		1560	2850	950	3450	5
12	30	7844	TURNER	1500.00	JAMES BLAKE		1560	2850	950	4950	4
13	30	7698	BLAKE	2850.00	JAMES BLAKE		1560	2850	950	7800	1

多个分析函数的排序方式不一致，返回的总结果集按照前面的排序，也就是sal

加了order by返回的是当前窗口的第一行到当前行的sum

此处由于是按照ename排序，所以返回的顺序是错乱的，而且由于存在order by，所以是从当前窗口的第一行到当前行

常用的分析函数还有 lag、lead、percent_rank、PERCENTILE_COUNT 等，大家可以自己回去研究一下。

3. First、last 返回通过 dense_rank 排序后的第一个或者最后一个。例如我想查询员工的最小工资最小的部门和员工的最大工资最大的部门：

```
select
min(deptno) keep(dense_rank first order by min(sal) nulls last) x,
min(deptno) keep(dense_rank last order by max(sal) nulls last) d
from emp
where sal is not null and deptno is not null
group by deptno
```

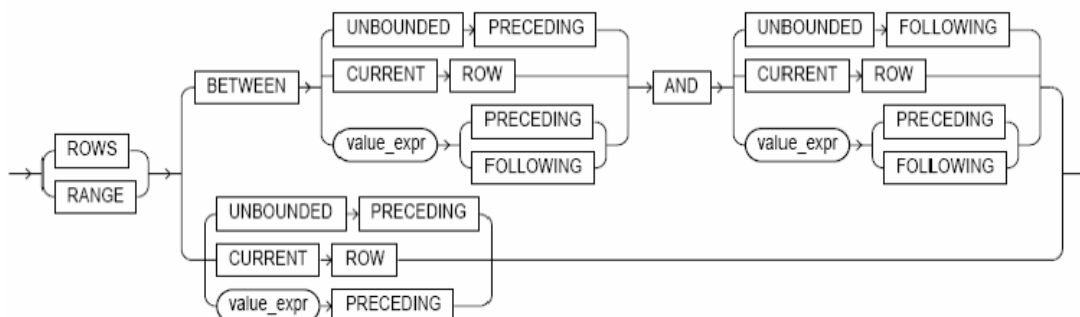
按照min(sal)排序，取得第一行deptno

按照max(sal)排序，取得第一行deptno

1. 先以deptno分组，求出每组的min(sal)和max(sal)

	X	D
1	30	10

4. Windows 子句，用来指明分析函数的计算窗口，语法如下：



窗口的划分方式有range和rows两种，Rows 表示物理偏移量，range表示逻辑偏移量。用rows或range划分窗口，按照起点在上，终点在下的原则（如果违反这个原则，则分析函数的计算结果为null），对窗口中的每一行应用分析函数计算结果。如果我想取：当前sal=500到当前sal为一个窗口，并按照窗口来计算工资总和，可以用如下sal来实现：

```
select empno,ename,sal,
sum(sal)
over(order by sal range between 500 preceding and current row ) s--以sal排序，讲比当前行小500以内的行到当前行作为一个窗口
from emp
```

	EMPNO	ENAME	SAL	S
1	4444	fedore	636.00	636
2	7900	JAMES	950.00	586
3	7876	ADAMS	1100.00	2386
4	1111	solaris	1111.00	3797
5	7654	MARTIN	1200.00	5561
6	7521	WARD	1200.00	5561
7	7934	MILLER	1300.00	6961
8	7844	TURNER	1500.00	7511
9	2222	suse	2222.00	2222
10	5555	slacware	2312.00	4534
11	7782	CLARK	2450.00	6984
12	7698	BLAKE	2850.00	5300
13	7566	JONES	2975.00	5825
14	7902	FORD	3000.00	11825
15	7788	SCOTT	3000.00	11825
16	6666	opensuse	3333.00	18491
17	3333	ubuntu	3333.00	18491
18	7839	KING	5000.00	5000
19	2	ll		

5. 分析函数与 group by 结合，进行 topN 查询，例如我想查询总工资前三名的部门，如下：

```
select * from (
select deptno,sum(sal),
row_number() over(order by sum(sal) desc) n--把分好组的结果集按照sum(sal)排序，输出序号
from emp
where deptno is not null
group by deptno
) where n<4
```

	DEPTNO	SUM(SAL)	N
1	20	10075	1
2	10	8750	2
3	30	7800	3

8.5 使用场景

逻辑比较复杂的查询，往往需要用到分析函数。

9 rownum

9.1 学习目标

理解 oracle 内部 rownum 的原理，会使用 rownum 进行 top-N 查询和其他范围查询。

9.2 要点

1. rownum 和 where 在同一层查询中，where 条件之后使用 rownum 比较，只能使用 <=, <, !=, <>, 不能使用 >, >=(>=1是可以的, 和不加效果一样), =(使用=, 只能是where rownum=1才可以)。否则不返回任何数据。如果使用 !=或<>, 那么只是返回前n-1行，其

他按照rownum工作原理推算。

2. 当rownum 和order by 在一个语句级别中(同一层)使用的时候,看这个查询的数据是否从索引中获取(或者根据索引先得到rowid然后定位行)的,如果不是,那么就是先查询出来,每行标上rownum,然后order by 将结果重新排序,那么rownum的顺序是乱的。如果排序的数据是从索引中查询的,这样结果有序。这取决于执行计划,执行计划又和oracle优化器相关。
3. 在执行语句时,有关rownum执行的顺序是这样的:
 - 1) 执行查询操作,初始化rownum值为1。
 - 2) 指针指向第一行,将该行的rownum标记为1。
 - 3) 进行where条件匹配,如果where条件返回false,则抛弃行,返回true,则返回该行,并且将rownum值自增1。
 - 4) Oracle获取下一行,将该行的rownum标记为当前rownum值。
 - 5) 返回第三步。

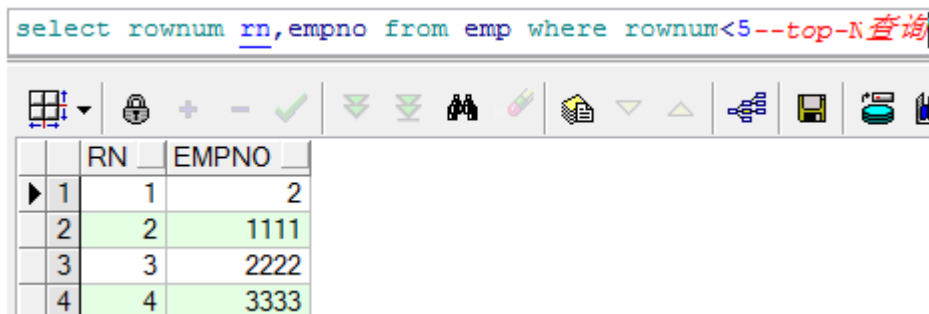
9.3 语法

Rownum 可以用在 where 条件中,如:

```
Select * from emp where rownum<5;
```

9.4 使用案例

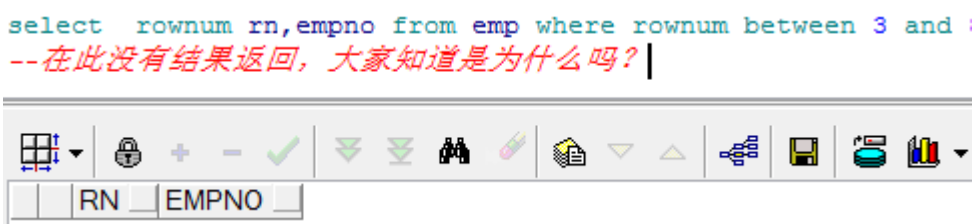
1. Top-N 查询:



```
select rownum rn,empno from emp where rownum<5--top-N查询
```

	RN	EMPNO
1	1	2
2	2	1111
3	3	2222
4	4	3333

2. 查询中间几行:



```
select rownum rn,empno from emp where rownum between 3 and 5  
--在此没有结果返回,大家知道是为什么吗?
```

	RN	EMPNO
--	----	-------

看一下 oracle 执行的原理,当指针移动到第一行的时候, rownum=1, rownum between 3 and 5 返回 false, 第一行被抛弃, 指针指向第二行, 此时 rownum 还是为 1, 第二行也被抛弃, 以此类推, 无结果返回。这种情况可以使用子查询, 先把 rownum 最为 rn 字段缓存到结果集里面, 然后对结果集进行筛选:

```
select * from(--需要先把rownum查询上来，缓存到一个结果集里面，然后对结果集进行筛选
select rownum rn,empno from emp order by empno
) where rn between 3 and 8
```

	RN	EMPNO
1	3	2222
2	4	3333
3	5	4444
4	6	5555
5	7	6666
6	8	7521

3. 上述查询其实存在隐患，不知道大家注意没有？子查询中 `select rownum rn,empno from emp order by empno`，oracle 的执行顺序是先取到结果集，标记上 rownum，然后进行排序，这样 rownum 的序号不一定是排序后的序号，所以取到的数据可能不是我想要的。那此处为什么我能取到正确的数据呢？这取决于执行计划，在 emp 表中，empno 作为表的主键，也就是唯一索引，在取得子查询结果集得时候，oracle 是根据索引读取数据的，而索引一般是在 oracle 的内存中，并且索引是有序的，优化器选择索引的方式访问 emp 表，所以 oracle 读取数据的同时为当前行标记上 rownum，所以 rownum 顺序不会错乱。如果我是通过 ename 排序取 3-8 行，emp 在 ename 上并没有建立索引，所以读取的数据时错乱的，如下：

```
select rownum rn,ename from emp order by ename
--以下rownum顺序错误
```

	RN	ENAME
1	16	ADAMS
2	11	BLAKE
3	12	CLARK
4	18	FORD
5	17	JAMES
6	9	JONES
7	14	KING
8	10	MARTIN
9	19	MILLER
10	13	SCOTT
11	15	TURNER
12	8	WARD
13	4	fedore
14	7	ll
15	6	opensuse
16	5	slacware
17	1	solaris
18	2	suse
19	3	ubuntu

此时再用外层查询取得 rn between 3 and 8，就会取得错误的数据。

```
select * from (
  select rownum rn,ename from (--第二层缓存rownum, 并且筛选<9的数据
    select ename from emp order by ename--最内层查询根据ename排序
  ) where rownum<9
) where rn>2--筛选>2的数据
```

	RN	ENAME
1	3	CLARK
2	4	FORD
3	5	JAMES
4	6	JONES
5	7	KING
6	8	MARTIN

此时能够保证数据是正确的。有关案例 2 的 order by 和 rownum 要慎用，因为即使在相关字段上有索引，oracle 的优化器也不一定会选择索引访问数据，这跟表的状态和其他很多原因都有关系，有关索引和执行计划的相关知识，这里不做讲解，将会在以后的课程中放在 oracle 优化的科目中进行讲解。

9.5 使用场景

在进行 top-N 查询或者取中间数据时可能用到。

10 rowid

10.1 学习目标

了解 rowid 的组成部分，会使用 rowid 进行删除重复行等查询。

10.2 要点

1. rowid 的是基于 64 位编码的 18 个字符，由数据对象编号+文件编号+块编号+行编号组成（数据对象编号(6)+文件编号(3)+块编号(6)+行编号(3)=18 位）。

10.3 语法

10.4 使用案例

1. Rowid 经常用于删除重复行，如我用如下语句加入 escore2 两条重复数据，如下：


```
insert into escore2 select * from escore2 where sid=1;
```

```
commit;
```

```
select * from escore2 order by sid,type
```

	SID	TYPE	SCORE
1	1	e	25
2	1	e	25
3	1	z	31
4	1	z	31
5	2	e	
6	2	z	45
7	3	e	71
8	3	z	66
9	4	e	104
10	4	z	76

重复数据

很显然，圈出的数据为重复数据，如下我可以用rowid来删除重复数据：

```
delete from escore2 where rowid not in(
select min(rowid) from escore2 group by sid ,type,score
);
```

```
commit;
```

再来看一下escore2表，重复数据没有了：

```
select * from escore2
```

	SID	TYPE	SCORE
1	1	e	25
2	2	e	
3	3	e	71
4	4	e	104
5	1	z	31
6	2	z	45
7	3	z	66
8	4	z	76

2. 分页，例如我想取 escore2 表的第 4-6 条数据，也就是上图 4、5、6 行，如下：

```
select * from escore2 where rowid not in(select rowid from escore2 where rownum<4) and rownum<4
```

	SID	TYPE	SCORE
1	4	e	104
2	1	z	31
3	2	z	45

10.5 使用场景

Rowid 可用于删除重复数据或者分页，还可以用于其他的需要唯一标识行的 sql。

11 Dade 的使用

11.1 学习目标

掌握 date 数据类型，会使用 date 类型，并且掌握 date 类型的常用函数。

11.2 要点

1. 一些常用的数据格式要牢记，他们就像 date 对象的属性，当你要访问 date 对象的相应属性时，需要将这个属性作为参数传入，属性对应的值才能被返回，如：

1) Y 或 YY 或 YYYY 年的最后一位，两位或三位

2) Q 季度

3) MM 月份

4) Month 用 9 个字符长度表示的月份名

5) WW 当年第几周

6) W 本月第几周

7) DDD 当年第几天

8) DD 当月第几天

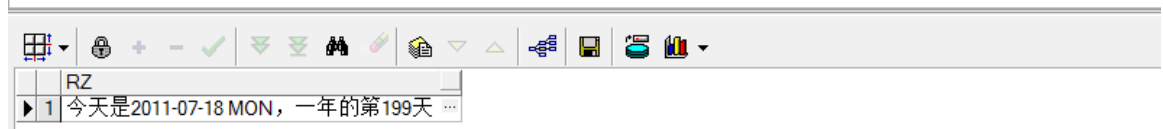
9) D 周内第几天

10) DY 中文的星期几

11) HH 或 HH12 12 进制小时数 HH24 24 小时制

例如今天是 2011 年 7 月 18 日，星期一，执行如下 sql 看一下结果：

```
select '今天是'||to_char(sysdate,'yyyy-MM-dd DY') ||', 一年的第'||to_char(sysdate,'ddd')||'天' rz
from dual
```

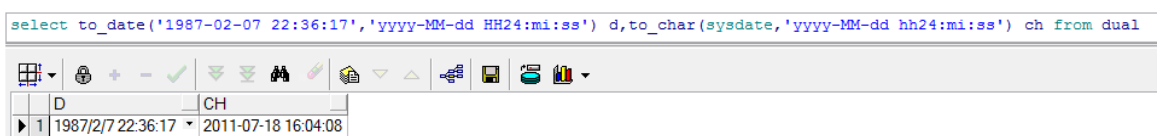


RZ
今天是2011-07-18 MON, 一年的第199天

2. 一些常见函数的用法：

- 1) 对于 to_date 和 to_char 函数大家应该很熟悉了，这应该是 oracle 里面最常用的函数了，如下：

```
select to_date('1987-02-07 22:36:17','yyyy-MM-dd HH24:mi:ss') d,to_char(sysdate,'yyyy-MM-dd hh24:mi:ss') ch from dual
```



D	CH
1987/2/7 22:36:17	2011-07-18 16:04:08

需要强调一点的是，oracle 有默认的显示格式，对于这个格式的字符串，oracle 是可以识别的，并且能通过隐式转换将其转换为 date 类型，如下 sql(在命令行执行)：

```
SQL> select sysdate from dual;

SYSDATE
-----
18-JUL-11    date类型输出显示格式就是oracle默认格式
```

由以上结果的输出可以看出我当前数据库的时间匹配格式是 18-JUL-11 的，那 oracle 可以接收这种类型的字符串将其隐式转换为 date 类型，如下 sql:

```
SQL> select 1 from dual where sysdate>'18-JUL-11';
          1
-----
          1

SQL> select 1 from dual where sysdate>'1987-02-07';
select 1 from dual where sysdate>'1987-02-07'
                                     *
ERROR at line 1:
ORA-01861: literal does not match format string    报错
```

oracle能够识别默认时间格式的字符串，将其转换为date类型与sysdate比较，故不报错。

此处时间格式oracle不识别，不能进行隐式转换。

如下我修改了当前 session 的默认时间格式，则执行不报错，但是只在当前 session 有效：

```
SQL> alter session set nls_date_format='yyyy-MM-dd';    修改当前session的默认时间匹配格式
Session altered.

SQL> select 1 from dual where sysdate>'1987-02-07';
          1
-----
          1

修改后字符串'1987-02-07'能够被识别，并且进行隐式转换
```

- 2) Last_day(mydate)，此函数返回 mydate 所在月份的最后一天。

```
SQL> select sysdate, last_day(sysdate) ls from dual;

SYSDATE      LS
-----
19-JUL-11    31-JUL-11    本月最后一天
```

- 3) Add_month(mydate,n)，返回 mydate 推后 n 个月后的日期。

```
SQL> select sysdate,add_months(sysdate,3) from dual;

SYSDATE      ADD_MONTHS(S
-----
19-JUL-11    19-OCT-11    返回当前日期加3个月后的日期
```

- 4) Months_between(date1,date2)，返回 date1 与 date2 间隔的月数。

```
SQL> select months_between(sysdate,to_date('2011-01-01','yyyy-mm-dd')) m from dual;
```

M
6.59465726

当前日期与2011-01-01间隔的月份数

- 5) Next_day(mydate,dayofweek), 返回自 mydate 日期起, 下一个 dayofweek (星期几) 的日期。

```
SQL> select to_char(sysdate,'day') td,next_day(sysdate,'MONDAY') nd from dual;
```

TD	ND
tuesday	25-JUL-11

今天是tuesday, 下一个monday是25-jul-11

- 6) Trunc(mydate,格式字符串), 返回对 mydate 截断到指定位置后的日期。Round(mydate,格式字符串), 返回对 mydate 四舍五入到指定位置的字符串, 如下 sql:

```
select sysdate,trunc(sysdate,'hh') tr,round(sysdate,'mm') rd from dual;
```

	SYSDATE	TR	RD
1	2011/7/19 11:08:06	2011/7/19 11:00:00	2011/8/1

截断时间到小时
对月份进行四舍五入

11.3 使用场景

对于日期的计算需要用到日期函数, 如:

- 1) 上个月末: `trunc(sysdate,'mm')-1`
- 2) 本月最后一秒: `trunc(last_day(sysdate)+1,'dd')-1/24/60/60`
- 3) 本月的天数: `trunc(last_day(sysdate)+1)-trunc(sysdate,'mm')`

12 字符串函数的使用

12.1 学习目标

掌握常用的字符串函数, 能够用字符串函数解决相关问题。

12.2 要点

1. `ascii(char)`: 返回字符串首字符串的 ASCII 码值。`Chr(n)`: 返回 ASCII 码值 n 对应的字符。
如下:

```
SQL> select chr(97) c,ascii('a') a from dual;
```

C	A
a	97

的ASCII码值是97

2. `concat(str1,str2,...)`, 返回 `str1`, `str2`,连接后的字符串。
3. `initcap(str)`, 返回每个单词首字母大写的字符串。
4. `instr(char1,char2[,n[,m]])`: 用于取得子串在字符串中的位置, 其中数字 `n` 为起始搜索位置, 数字 `m` 为子串出现次数。如果数字 `n` 为负数, 则从尾部开始搜索。数字 `m` 必须为正整数, 并且 `n` 和 `m` 的默认值为 1。
5. `length(char)`: 返回字符串的长度。如果字符串的类型为 `char`, 则其长度包括所有的后缀空格; 如果 `char` 是 `null`, 则返回 `null`。
6. `lower(char)`: 用于将字符串转换为小写格式; `upper(char)`: 将字符串转换为大写格式。
7. `lpad(char1,n,char2)`: 用于在字符串 `char1` 的左端填充字符串 `char2`, 直至字符串总长度为 `n`, `char2` 的默认值为空格。如果 `char1` 长度大于 `n`, 则该函数返回 `char1` 左端的 `n` 个字符; `rpadd(char1,n,char2)`用于在字符串 `char1` 的右端填充字符串 `char2`, 直至字符串总长度为 `n`, `char2` 的默认值为空格。如果 `char1` 长度大于 `n`, 则该函数返回 `char1` 左端的 `n` 个字符。
8. `trim(char from string)`用于从字符串的头部、尾部或两端截取特定字符; `ltrim(char1[,set])`: 用于去掉字符串 `char1` 左端所包含的 `set` 中的任何字符。Oracle 从左端第一个字符开始扫描, 逐一去掉在 `set` 中出现的字符, 当遇到不是 `set` 中出现的字符时终止, 然后返回剩余结果; `rtrim(char1[,set])`: 用于去掉字符串 `char1` 右端所包含的 `set` 中的任何字符。Oracle 从右端第一个字符开始扫描, 逐一去掉在 `set` 中出现的字符, 当遇到不是 `set` 中出现的字符时终止, 然后返回剩余结果。
9. `replace(char,search_string[,replacement_string])`: 用于将字符串的子串替换为其他子串。如果 `search_string` 为 `null`, 则返回原有字符串; 如果 `replacement_string` 为 `null`, 则会去掉指定子串。

欢迎点击这里的链接进入精彩的[Linux公社](http://www.Linuxidc.com)网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](http://www.Linuxidc.com)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：[Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#) [Hadoop 专题](#)
[RedHat 专题](#) [SUSE 专题](#) [红旗 Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号：[linuxidc_com](#)

Linuxidc.com

微信扫一扫

订阅专业的最新Linux资讯及开源技术教程。

搜索微信公众号：[linuxidc_com](#)

