Oracle SQL 编程最佳实践

第一部分 SQL 语言基础

第一章、关系型与非关系型数据库

1.1 关系型数据库由来

关系型数据库，是指采用了关系模型来组织数据的数据库。
关系模型是在 1970 年由 IBM 的研究员 E.F.Codd 博士首先提出的，在之后的几十年中，关系模型的概念得到了充分的发展并逐渐成为主流数据库结构的模型。
简单来说，关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。

1.2 关系型数据库优点

1）容易理解：
二维表结构是非常贴近逻辑世界的一个概念，关系模型相对之前的网状、层次等其他模型来说更容易理解
2）使用方便：
通用的 SQL 语言使得操作关系型数据库非常方便
3）易于维护：
丰富的完整性(实体完整性、参照完整性和用户定义的完整性)大大减低了数据冗余和数据不一致的概率
4）交易安全：
所有关系型数据库都不同程度的遵守事务的四个基本属性，因此对于银行、电信、证券等交易型业务的是不可或缺的。

1.3 关系型数据库瓶颈

1）高并发读写需求
网站的用户并发性非常高，往往达到每秒上万次读写请求，对于传统关系型数据库来说，硬盘 I/O 是一个很大的瓶颈。
2）海量数据的高效率读写
互联网上每天产生的数据量是巨大的，对于关系型数据库来说，在一张包含海量数据的表中查询，效率是非常低的。
3）高扩展性和可用性
在基于 web 的结构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，数据库却没有办法像 web server 和 app server 那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供 24 小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移。

1.4 非关系型数据库

1）NoSQL 特点：
可以弥补关系型数据库的不足。
针对某些特定的应用需求而设计，可以具有极高的性能。
大部分都是开源的，由于成熟度不够，存在潜在的稳定性和维护性问题。
2）NoSQL 分类：
面向高性能并发读写的 key-value 数据库
面向海量数据访问的面向文档数据库
面向可扩展性的分布式数据库

1.5 优势互补，相得益彰

1）关系型数据库适用结构化数据，NoSQL 数据库适用非结构化数据。
2）Oracle 数据库未来的发展方向：提供结构化、非结构化、半结构化的解决方案，实现关系型数据库和 NoSQL 共存互补。值得强调的是：目前关系型数据库仍是主流数据库，虽然 NoSql 数据库打破了关系数据库存储的观念，可以很好满足 web2.0 时代数据存储的要求，但 NoSql 数据库也有自己的缺陷。在现阶段的情况下，可以将关系型数据库和 NoSQL 数据库结合使用，相互弥补各自的不足。

第二章、SQL 的基本函数

2.1 关系型数据库命令类别

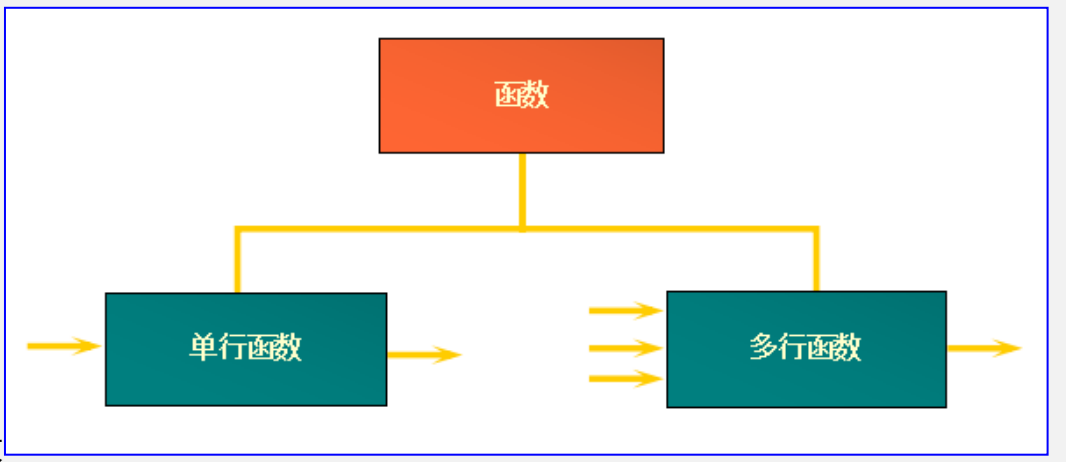
数据操纵语言：DML: select; insert; delete; update; merge.
数据定义语言：DDL: create; alter; drop; truncate; rename; comment.
事务控制语言：TCL: commit; rollback; savepoint.
数据控制语言：DCL: grant; revoke.

2.2 单行函数与多行函数

资料整理
火龙果软件
荐

单行函数：指一行数据输入，返回一个值的函数。所以查询一个表时，对选择的每一行数据都返回一个结果。
SQL>select empno,lower(ename) from emp;
多行函数：指多行数据输入，返回一个值的函数。所以对表的群组进行操作，并且每组返回一个结果。（典型的是聚合函数）
SQL>select sum(sal) from emp;
本小结主要介绍常用的一些单行函数，分组函数见第五章

2.3 单行函数的几种类型



2.3.1 字符型函数

lower('SQL Course')----->sql course 返回小写
upper('sql course')----->SQL COURSE 返回大写
initcap('SQL course')-----> Sql Course 每个单字返回首字母大写
concat('good','string')---->good string 拼接 只能拼接 2 个字符串
substr('String',1,3)---->Str 从第 1 位开始截取 3 位数，
演变：只有两个参数的
substr('String',3) 正数第三位起始，得到后面所有字符
substr('String',-2) 倒数第二位，起始，得到最后所有字符
instr('t#i#m#r#a#n#','#') --->找第一个#字符在那个绝对位置，得到的数值
Instr 参数经常作为 substr 的第三个参数值
演变：Instr 参数可有四个之多
如 select instr('aunfukk','u',-1,1) from dual; 倒数第一个 u 是哪个位置，结果返回 5
length('String')---->6 长度，得到的是数值
length 参数又经常作为 substr 的第三个参数
lpad('first',10,'$')左填充
rpad(676768,10,'*')右填充
replace('JACK and JUE','J','BL')---->BLACK and BLUE
trim('m' from 'mmtimranm')---->timran 两头截，这里的'm'是截取集，仅能有一个字符

处理字符串时，利用字符型函数的嵌套组合是非常有效的，试分析一道考题：

```
create table customers(cust_name varchar2(20));

insert into customers values('Lex De Hann');
insert into customers values('Renske Ladwig');
insert into customers values('Jose Manuel Urman');
insert into customers values('Joson Malin');

select * from customers;
CUST_NAME
--------------------
Lex De Hann
Renske Ladwig
Jose Manuel Urman
Joson Malin
```

一共四条记录，客户有两个名的，也有三个名的，现在想列出仅有三个名的客户，且第一个名字用*号略去
答案之一：

```
SELECT LPAD(SUBSTR(cust_name,INSTR(cust_name,' ')),LENGTH(cust_name),'*') "CUST NAME"
FROM customers
WHERE INSTR(cust_name,' ',1,2)<>0;
CUST NAME
```

资料整理
火龙果软件
荐

*** De Hann
**** Manuel Urman

分析：
先用 INSTR(cust_name,' ')找出第一个空格的位置，
然后，SUBSTR(cust_name,INSTR(cust_name,' '))从第一个空格开始往后截取字符串到末尾，结果是第一个空格以后所有的字符，
最后，LPAD(SUBSTR(cust_name,INSTR(cust_name,' '')),LENGTH(cust_name),'*')用 LPAD 左填充到 cust_name 原来的长度，不足的部分用*填充，也就是将第一个空格前　　的位置，用*填充。
where 后过滤是否有三个名字，INSTR(cust_name, ' ',1,2)从第一个位置，从左往右，查找第二次出现的空格，如果返回非 0 值，则说明有第二个空格，则有第三个名字。

2.3.2 数值型函数
round 对指定的值做四舍五入,round(p,s) s 为正数时，表示小数点后要保留的位数，s 也可以为负数，但意义不大。
round:按指定精度对十进制数四舍五入,如:round(45.923, 1),结果,45.9
round(45.923, 0),结果,46
round(45.923, -1),结果,50
trunc 对指定的值取整 trunc(p,s)
trunc:按指定精度截断十进制数,如:trunc(45.923, 1),结果,45.9
trunc(45.923),结果,45
trunc(45.923, -1),结果, 40
mod 返回除法后的余数
SQL> select mod(100,12) from dual;
2.3.3 日期型函数
因为日期在 oracle 里是以数字形式存储的，所以可对它进行加减运算，计算是以天为单位。
缺省格式：DD-MON-RR.
可以表示日期范围：（公元前）4712 至（公元）9999
时间格式
SQL> select to_date('2003-11-04 00:00:00' ,'YYYY-MM-DD HH24:MI:SS') FROM dual;
SQL> select sysdate+2 from dual;　　当前时间+2day
SQL> select sysdate+2/24 from dual;　　当前时间+2hour
SQL> select (sysdate-hiredate)/7 week from emp;　　两个 date 类型差，结果是以天为整数位的实数。
①MONTHS_BETWEEN　　计算两个日期之间的月数
SQL>select months_between('1994-04-01','1992-04-01') mm from dual;
查找 emp 表中参加工作时间>30 年的员工
SQL>select * from emp where months_between(sysdate,hiredate)/12>30;
很容易认为单行函数返回的数据类型与函数类型一致，对于数值函数类型而言的确如此，但字符和日期函数可以返回任何数据类型的值。比如 instr 函数是字符型的，months_between 函数是日期型的，但它们返回的都是数值。
②ADD_MONTHS　　给日期增加月份
SQL>select add_months('1992-03-01',4) am from dual;
③LAST_DAY　　日期当前月份的最后一天
SQL>select last_day('1989-03-28') l_d from dual;
④NEXT_DAY　　NEXT_DAY 的第 2 个参数可以是数字 1-7，分别表示周日--周六(考点）
比如要取下一个星期六，则应该是：
SQL>select next_day(sysdate,7) FROM DUAL;
⑤ROUND(p,s)，TRUNC(p,s)在日期中的应用，如何舍入要看具体情况，s 是 MONTH 按 30 天计，应该是 15 舍 16 入，s 是 YEAR 则按 6 舍 7 入计算。
SQL>SELECT empno, hiredate,round(hiredate,'MONTH') AS round,trunc(hiredate,'MONTH') AS trunc
FROM emp WHERE empno=7844;
SQL>SELECT empno, hiredate, round(hiredate,'YEAR') AS round,trunc(hiredate,'YEAR') AS trunc
FROM emp WHERE empno=7839;
2.3.4 几个有用的函数和表达式
1）DECODE 函数和 CASE 表达式：
实现 sql 语句中的条件判断语句，具有类似高级语言中的 if-then 语句的功能。
decode 函数源自 oracle, case 表达式源自 sql 标准，实现功能类似，decode 语法更简单些。
decode 函数用法：
SQL> SELECT job, sal,
decode(job, 'ANALYST', SAL*1.1, 'CLERK', SAL*1.15,'MANAGER', SAL*1.20, SAL) SALARY
FROM emp
/

资料整理
火龙果软件
荐

decode 函数的另两种常见用法：
SQL> select ename,job,decode (comm,null,'nonsale','sale') saleman from emp;
注：单一列处理，共四个参数：含义是：comm 如果为 null 就取'nonsale'，否则取'sale'
SQL> select ename,decode (sign(sal-2800), 1, 'HIGH','LOW') as "LEV" from emp;
注：sign()函数根据某个值是 0、正数还是负数，分别返回 0、1、-1，含义是：工资大于 2800, 返回 1，真取'HIGH'，假取'LOW'
CASE 表达式第一种用法：

```
SQL> SELECT job, sal,case job
when 'CLERK' then SAL*1.15
when 'MANAGER' then SAL*1.20
else sal end SALARY
FROM emp
/
```

CASE 表达式第二种用法：

```
SQL> SELECT job, sal,case
when job='ANALYST' then SAL*1.1
when job='CLERK'     then SAL*1.15
when job='MANAGER' then SAL*1.20
else sal end SALARY
FROM emp
/
```

以上三种写法结果都是一样的
CASE 第二种语法比第一种语法增加了搜索功能。形式上第一种 when 后跟定值，而第二种还可以使用表达式和比较符。
看一个例子

```
SQL> SELECT ename,sal,case
when sal>=3000 then '高级'
when sal>=2000 then '中级'
else '低级' end  级别
FROM emp
/
```

再看一个例子:使用了复杂的表达式

```
SQL> SELECT AVG(CASE
WHEN sal BETWEEN 500 AND 1000 AND JOB='CLERK'
THEN sal ELSE null END) "CLERK_SAL"
from emp;
```

比较;

```
SQL> select avg(sal) from emp where job='CLERK';
```

2）DISTINCT(去重)限定词的用法：
distinct 貌似多行函数，严格来说它不是函数而是 select 子句中的关键字。
SQL> select distinct job from emp;　　消除表行重复值。
SQL> select distinct job,deptno from emp;　　重复值是后面的字段组合起来考虑的
3）sys_context 获取环境上下文的函数（应用开发）
scott 远程登录

```
SQL>select SYS_CONTEXT('USERENV','IP_ADDRESS') from dual;
--------------------------------------------------------------------------------
192.168.0.136

SQL> select sys_context('userenv','sid') from dual;
SYS_CONTEXT('USERENV','SID')
--------------------------------------------------------------------------------
129

SQL> select sys_context('userenv','terminal') from dual;
SYS_CONTEXT('USERENV','TERMINAL')
--------------------------------------------------------------------------------
TIMRAN-222C75E5
```

4）处理空值的几种函数（见第四章）
5）转换函数 TO_CHAR、TO_DATE、TO_NUMBER （见第三章）

资料整理
火龙果软件
荐

第三章、SQL 的数据类型(表的字段类型）

3.1 四种基本的常用数据类型（表的字段类型）

1、字符型，2、数值型，3、日期型，4、大对象型

3.1.1 字符型：

数据类型	最大长度	说　明
CHAR(size)	2000	固定长度字符串，size表示存储的字符数量
NCHAR(size)	2000	固定长度的NLS(National Language Support)字符串，size表示存储的字符数量
NVARCHAR2(size)	4000	可变长度的NLS字符串，size表示存储的字符数量
VARCHAR2(size)	4000	可变长度字符串，size表示存储的字符数量
RAW	2000	可变长度二进制字符串

字符类型 char 和 varchar2 的区别

```
SCOTT@ prod>create table t1(c1 char(10),c2 varchar2(10));
SCOTT@ prod>insert into t1 values('a','ab');
SCOTT@ prod>select length(c1),length(c2) from t1;
LENGTH(C1) LENGTH(C2)
---------- ----------
        10          2
```

3.1.2 数值型：

数据类型	说　明
NUMBER(p,s)	包含小数位的数值类型。参数p表示精度，参数s表示小数点后面的位数。例如：NUMBER(10,2)表示小数点之前最多可以有8位数字，小数点后有2位数字
NUMERIC(p,s)	与NUMBER(p,s)相同
FLOAT	浮点数类型。它属于近似数据类型，并不存储数数字的精确值，只存储最近似值
DEC(p,s)	与NUMBER(p,s)相同
DECIMAL(p,s)	与NUMBER(p,s)相同
INTEGER	整数类型
INT	同INTEGER
SMALLINT	短整类型
REAL	实数类型，与FLOAT一样，属于近似数据类型
DOUBLE PRECISION	双精度类型

3.1.3 日期型：

资料整理
火龙果软件
荐

数据类型	说　明
DATE	日期类型
TIMESTAMP	与DATE数据类型相比，TIMESTAMP类型可以精确到微秒，微秒的精确范围为0-9，默认为6
TIMESTAMP WITH TIME ZONE	带时区偏移量的TIMESTAMP数据类型
TIMESTAMP WITH LOCAL TIME ZONE	带时区偏移量的TIMESTAMP数据类型
INTERVAL YEAR TO MONTH	使用YEAR和MONTH日期时间字段存储一个时间段。年份精度指定表示年份的数字的位数。默认为2
INTERVAL DAY TO SECOND	用于按照日、小时、分钟和秒来存储一个时段。日精度表示DAY字段的位数，默认为2；微秒的精度范围为0-9，默认为6

系统安装后，默认日期格式是 DD-MON-RR, RR 和 YY 都是表示两位年份，但 RR 是有世纪认知的，它将指定日期的年份和当前年份比较后确定年份是上个世纪还是本世纪（如表）。

当前年份	指定日期	RR 格式	YY 格式
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2013	27-OCT-95	1995	2095

3.1.4 LOB 型：

大对象是 10g 引入的，在 11g 中又重新定义，在一个表的字段里存储大容量数据，所有大对象最大都可能达到 4G

数据类型	说　明
BFILE	指向服务器文件系统上的二进制文件的文件定位器，该二进制文件保存在数据库之外
BLOB	保存非结构化的二进制大对象数据
CLOB	保存单字节或多字节字符数据
NCLOB	保存Unicode编码字符数据

CLOB，NCLOB，BLOB 都是内部的 LOB 类型，没有 LONG 只能有一列的限制。

保存图片或电影使用 BLOB 最好、如果是小说则使用 CLOB 最好。

虽然 LONG、RAW 也可以使用，但 LONG 是 oracle 将要废弃的类型，因此建议用 LOB。

虽说将要废弃，但还没有完全废弃，比如 oracle 11g 里的一些视图如 dba_views，对于 text(视图定义）仍然沿用了 LONG 类型。

Oracle 11g 重新设计了大对象，推出 SecureFile Lobs 的概念，相关的参数是 db_securefile，采用 SecureFile Lobs 的前提条件是 11g 以上版本，ASSM 管理等，符合这些条件的

BasicFile Lobs 也可以转换成 SecureFile Lobs。较之过去的 BasicFile Lobs, SecureFile Lobs 有几项改进：

1）压缩，2）去重，3）加密。

当 create table 定义 LOB 列时，也可以使用 LOB_storage_clause 指定 SecureFile Lobs 或 BasicFile Lobs

而 LOB 的数据操作则使用 Oracle 提供的 DBMS_LOB 包，通过编写 PL/SQL 块完成 LOB 数据的管理。

3.2 数据类型的转换

3.2.1 转换的需求

什么情况下需要数据类型转换

1）如果表中的某字段是日期型的，而日期又是可以进行比较和运算的，这时通常要保证参与比较和运算的数据类型都是日期型。

2）当对函数的参数进行抽（截）取、拼接，或运算等操作时，需要转换为那个函数的参数要求的数据类型。

3）制表输出有格式需求的，可将 date 类型，或 number 类型转换为 char 类型

资料整理
火龙果软件
荐

4）转换成功是有条件的，有隐性转换和显性转换两种方式

3.2.2 隐性类型转换：
是指 oracle 自动完成的类型转换。在一些带有明显意图的字面值上，可以由 Oracle 自主判断进行数据类型的转换。
一般规律：
①比较、运算或连接时：
SQL> select empno,ename from emp where empno='7788'
empno 本来是数值类型的，这里字符'7788'隐性转换成数值 7788
SQL> SELECT '12.5'+11 FROM dual;
将字符型‘12.5’隐转成数字型再求和
SQL> SELECT 10+('12.5'||11) FROM dual;
将数字型 11 隐转成字符与‘12.5’合并，其结果再隐转数字型与 10 求和
②调用函数时
SQL> select length(sysdate) from dual;
将 date 型隐转成字符型后计算长度
③向表中插入数据时
create table scott. t1 (id int,name char(10),birth date);
insert into scott.t1 values('123',456,'2017-07-15');
按照字段的类型进行隐式转换

3.2.3 显性类型转换
即强制完成类型转换（推荐），有三种形式的数据类型转换函数：
TO_CHAR
TO_DATE
TO_NUMBER

1）日期-->字符
SQL> select ename,hiredate, to_char(hiredate, 'DD-MON-YY') month_hired from emp
where ename='SCOTT';
ENAME HIREDATE MONTH_HIRED
---------- ---------- ----------
SCOTT 1987-04-19 00:00:00 19-4 月 -87
fm 压缩空格或左边的'0'
SQL> select ename, hiredate, to_char(hiredate, 'fmyyyy-mm-dd') month_hired from emp
where ename='SCOTT';
ENAME HIREDATE MONTH_HIRED
---------- ---------- ----------
SCOTT 1987-04-19 00:00:00 1987-4 19
其实 DD-MM-YY 是比较糟糕的一种格式，因为当日期中天数小于 12 时，DD-MM-YY 和 MM-DD-YY 容易造成混乱。
以下用法也很常见
SQL> select to_char(hiredate,'yyyy') FROM emp;
SQL> select to_char(hiredate,'mm') FROM emp;
SQL> select to_char(hiredate,'dd') FROM emp;
SQL> select to_char(hiredate,'DAY') FROM emp;

2）数字-->字符：9 表示数字，L 本地化货币字符
SQL> select ename, to_char(sal, 'L99,999.99') Salary from emp where ename='SCOTT';
ENAME SALARY

资料整理
火龙果软件
荐

---------- ----------
SCOTT ￥3,000.00
以下四个语句都是一个结果（考点），
SQL> select to_char(1890.55,'$99,999.99') from dual;
SQL> select to_char(1890.55,'$0G000D00') from dual;
SQL> select to_char(1890.55,'$99G999D99') from dual;
SQL> select to_char(1890.55,'$99G999D00') from dual; 9 和 0 可用，其他数字不行

3）字符-->日期
SQL> select to_date('1983-11-12', 'YYYY-MM-DD') tmp_DATE from dual;
4）字符-->数字：
SQL> SELECT to_number('$123.45','$9999.99') result FROM dual;
使用 to_number 时如果使用较短的格式掩码转换数字，就会返回错误。不要混淆 to_number 和 to_char 转换。
SQL> select to_number('123.56','999.9') from dual;
报错：ORA-01722: 无效数字

练习：建立 t1 表，包括出生日期，以不同的日期描述方法插入数据，显示小于 15 岁的都是谁
SQL> create table t1 (id int,name char(10),birth date);
insert into t1 values(1,'tim',sysdate);
insert into t1 values(2,'brian',sysdate-365*20);
insert into t1 values(3,'mike',to_date('1998-05-11','yyyy-mm-dd'));
insert into t1 values(4,'nelson',to_date('15-2 月-12','dd-mon-rr'));
SQL> select * from t1;
ID NAME BIRTH
---------- ---------- ----------
1 tim 2016-02-25 17:34:00
2brian 1996-03-01 17:34:22
3 mike 1998-05-11 00:00:00
4 nelson 2012-02-15 00:00:00
SQL>select name||'的年龄是'||to_char(months_between(sysdate,birth)/12,99) age from t1
where months_between(sysdate,birth)/12<15;
AGE
---
tim 的年龄是 0
nelson 的年龄是 4

第四章、WHERE 子句中常用的运算符

4.1 运算符及优先级：

算数运算符
*，/，+，-，
逻辑运算符
not, and ,or
比较运算符
1）单行比较运算 =,>, >=,<,<=, <>
2）多行比较运算 >any,>all,<any,<all,in， not in
3）模糊比较 like（配合“%”和“_”）
4）特殊比较 is null
5）()优先级最高
SQL>select ename, job, sal ,comm from emp where job='SALESMAN' OR job='PRESIDENT' AND sal> 1500;
注意：条件子句使用比较运算符比较两个选项，重要的是要理解这两个选项的数据类型。必须得一致，所以这里常用显性转换。
数值型、日期型、字符型都可以与同类型比较大小，但数值和日期比的是数值的大小，而字符比的是 acsii 码的大小
试比较下面语句，结果为什么不同
SQL> select * from emp where hiredate>to_date('1981-02-21','yyyy-mm-dd');
SQL> select * from emp where to_char(hiredate,'dd-mon-rr')>'21-feb-81';

4.2 常用谓词

4.2.1 用 BETWEEN AND 操作符来查询出在某一范围内的行.

SQL> SELECT ename, sal FROM emp WHERE sal BETWEEN 1000 AND 1500;

资料整理 火龙果软件 荐

between 低值 and 高值，包括低值和高值。

4.2.2 模糊查询及其通配符：

在 where 子句中使用 like 谓词，常使用特殊符号"%"或"_"匹配查找内容，也可使用 escape 可以取消特殊符号的作用。

```
SQL>
create table test (name char(10));
insert into test values ('sFdL');
insert into test values ('AEdLHH');
insert into test values ('A%dMH');
commit;
SQL> select * from test;
NAME
----------
sFdL
AEdLHH
A%dMH
SQL> select * from test where name like 'A\%%' escape '\';
NAME
----------
A%dMH
```

4.2.3' '和" "的用法：

单引号的转义：连续两个单引号表示转义.

' '内表示字符或日期数据类型，而" " 一般用于别名中有大小写、保留字、空格等场合，引用 recyclebin 中的《表名》也需要" ".

```
SQL> select empno||' is Scott''s empno' from emp where empno=7788;
EMPNO||'ISSCOTT''SEMPNO'
--------------------------------------------------------
7788 is Scott's empno
```

4.2.4 交互输入变量符&和&&的用途：

①使用&交互输入

```
SQL> select empno,ename from emp where empno=&empnumber;
输入 empnumber 的值:  7788
```

&后面是字符型的，注意单引号问题，可以有两种写法：

```
SQL> select empno,ename from emp where ename='&emp_name';
输入 emp_name 的值:  SCOTT
SQL> select empno,ename from emp where ename=&emp_name;
输入 emp_name 的值:  'SCOTT'
```

②使用&&可以在当前 session 下将&保存为变量

作用是使后面的相同的&不再提问，自动取代。

```
SQL> select empno,ename,&&salary from emp where deptno=10 order by &salary;
输入 salary 的值:  sal
```

上例给的&salary 已经在当前 session 下存储了，可以使用 undefine salary 解除。

&&salary 和&的提示是按所在位置从左至右边扫描，&&salary 写在左边(首位)，&salary(第二位)

③define(定义变量）和 undefine 命令（解除变量）

```
SQL> define --显示当前已经定义的变量（包括默认值）
```

set define on|off 可以打开和关闭&。

```
SQL> define emp_num=7788      定义变量 emp_num
SQL>select empno,ename,sal from emp where empno=&emp_num;
SQL>undefine emp_num    取消变量
```

如果不想显示“原值”和“新值”的提示，可以使用 set verify on|off 命令

④Accept 接收一个变量

类似 define 功能，但通常和&配合使用

```
SQL> accept lowdate prompt 'Please enter the low date range ("MM/DD/YYYY"):';
SQL> accept highdate prompt 'Please enter the highdate range ("MM/DD/YYYY"):';
SQL> select ename||','||job as EMPLOYEES, hiredate from emp where hiredate between to_date('&lowdate','MM/DD/YYYY') and to_date('&highdate','MM/DD/YYYY');
```

4.2.5 使用逻辑操作符: AND; OR; NOT

AND 两个条件都为 TRUE ，则返回 TRUE

```
SQL> SELECT empno,ename,job,sal FROM emp WHERE sal>=1100 AND job='CLERK';
```

OR 两个条件中任何一个为 TRUE，则返回 TRUE

资料整理
火龙果软件
荐

SQL> SELECT empno,ename,job,sal FROM emp WHERE sal>=1100 OR job='CLERK';
NOT 如果条件为 FALSE，返回 TRUE
SQL> SELECT ename,job FROM emp WHERE job NOT IN ('CLERK','MANAGER','ANALYST');

4.2.6 什么是伪列

简单理解它是表中的列，但不是你创建的。
Oracle 数据库有两个著名的伪列 rowid 和 rownum
ROWID 的含义
rowid 可以说是物理存在的，表示记录在表空间中的唯一位置 ID，所以表的每一行都有一个独一无二的物理门牌号。
ROWNUM 的含义
rownum 是对结果集增加的一个伪列，即先查到结果集，当输出时才加上去的一个编号。rownum 必须包含 1 才有值输出。
SQL> select rowid,rownum,ename from emp;
SQL> select * from emp where rowid='AAARZ+AAEAAAAG0AAH';
SQL> select * from emp where rownum<=3;

第五章、分组函数

5.1 五个分组函数

sum(); avg(); count(); max(); min().

数值类型可以使用所有组函数
SQL> select sum(sal) sum, avg(sal) avg, max(sal) max, min(sal) min, count(*) count from emp;
MIN(),MAX(),count()可以作用于日期类型和字符类型
SQL> select min(hiredate), max(hiredate)，min(ename),max(ename),count(hiredate) from emp;
COUNT(*)函数返回表中行的总数，包括重复行与数据列中含有空值的行，而其他分组函数的统计都不包括空值的行。
COUNT(comm)返回该列所含非空行的数量。

```
SQL> select count(*),count(comm) from emp;
  COUNT(*)          COUNT(COMM)
---------- -----------
        14            4
```

5.2 GROUP BY 建立分组

SQL>select deptno, avg(sal）from emp group by deptno;
group by 后面的列也叫分组特性，一旦使用了 group by, select 后面只能有两种列，一个是组函数列，而另一个是分组特性列（可选）。
对分组结果进行过滤
SQL>select deptno, avg(sal) avgcomm from emp group by deptno having avg(sal)>2000;
SQL>select deptno, avg(sal) avgcomm from emp where avg(sal)>2000 group by deptno;　　错误的,应该使用 HAVING 子句
对分组结果排序
SQL>select deptno, avg(nvl(sal,0)) avgcomm from emp group by deptno order by avg(nvl(sal,0));
排序的列不在 select 投影选项中也是可以的，这是因为 order by 是在 select 投影前完成的。

5.3 分组函数的嵌套

单行函数可以嵌套任意层，但分组函数最多可以嵌套两层。
比如：count(sum(avg)))会返回错误“ORA-00935：group function is nested too deeply”.
在分组函数内可以嵌套单行函数，如：要计算各个部门 ename 值的平均长度之和
SQL> select sum(avg(length(ename))) from emp group by deptno;

第六章、数据限定与排序

6.1 SQL 语句的编写规则

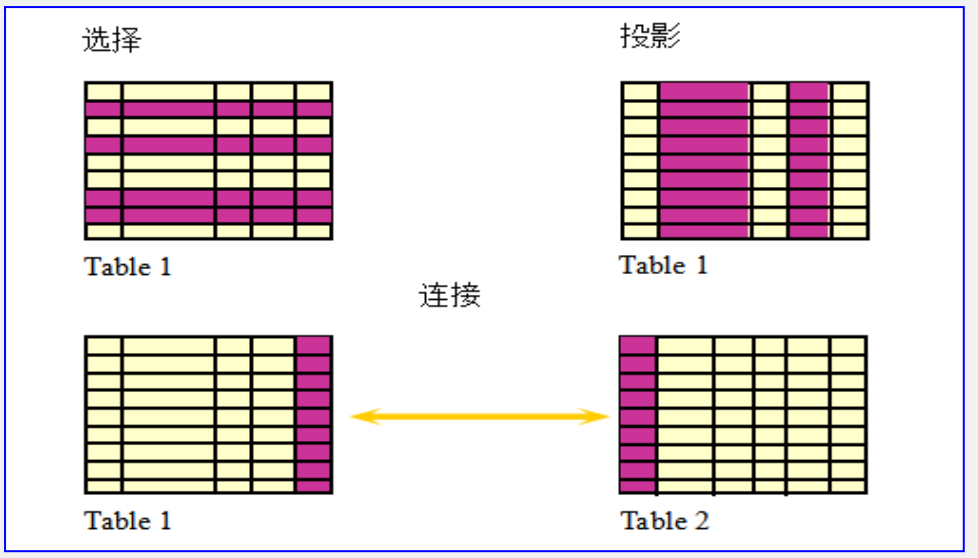
1）SQL 语句是不区分大小写的，关键字通常使用大写；其它文字都是使用小写
2）SQL 语句可以是一行，也可以是多行，但关键字不能在两行之间一分为二或缩写
3）子句通常放在单独的行中，这样可以增强可读性并且易于编辑
4）使用缩进是为了增强可读性

简单查询语句执行顺序

简单查询一般是指一个 SELECT 查询结构，仅访问一个表。

资料整理
火龙果软件
荐

基本语法如下：
SELECT 子句— 指定查询结果集的列组成，列表中的列可以来自一个或多个表或视图。
FROM 子句— 指定要查询的一个或多个表或视图。
WHERE 子句— 指定查询的条件。
GROUP BY 子句— 对查询结果进行分组的条件。
HAVING 子句— 指定分组或集合的查询条件。
ORDER BY 子句— 指定查询结果集的排列顺序



语句执行的一般顺序为①from，②where，③group by，④having，⑤order by，⑥select
where 限定 from 后面的表或视图，限定的选项只能是表的列或列单行函数或列表达式，where 后不可以直接使用分组函数
SQL> select empno,job from emp where sal>2000;
SQL> select empno,job from emp where length(job)>5;
SQL> select empno,job from emp where sal+comm>2000;
having 限定 group by 的结果，限定的选项必须是 group by 后的聚合函数或分组列，不可以直接使用 where 后的限定选项。
SQL> select sum(sal) from emp group by deptno having deptno=10;
SQL> select deptno,sum(sal) from emp group by deptno having sum(sal)>7000;
也存在一些不规范的写法：不建议采用。如上句改成 having 位置 在 group by 之前
SQL> select deptno,sum(sal) from emp having sum(sal)>7000 group by deptno;

6.2 排序(order by)

1）位置：order by 语句总是在一个 select 语句的最后面。
2）排序可以使用列名，列表达式，列函数，列别名，列位置编号等都没有限制，select 的投影列可不包括排序列，除指定的列位置标号外。
3）升序和降序，升序 ASC（默认), 降序 DESC。有空值的列的排序，缺省（ASC 升序）时 null 排在最后面（考点）。
4）混合排序，使用多个列进行排序，多列使用逗号隔开，可以分别在各列后面加升降序。
SQL> select ename,sal from emp order by sal;
SQL> select ename,sal as salary from emp order by salary;
SQL> select ename,sal as salary from emp order by 2;
SQL> select ename,sal,sal+100 from emp order by sal+comm;
SQL> select deptno,avg(sal) from emp group by deptno order by avg(sal) desc;
SQL> select ename,job,sal+comm from emp order by 3 nulls first;
SQL> select ename,deptno,job from emp order by deptno asc,job desc;

6.3 空值（null)

空值既不是数值 0,也不是字符" ", null 表示不确定。
6.3.1 空值参与运算或比较时要注意几点：
1）空值（null）的数据行将对算数表达式返回空值
SQL> select ename,sal,comm,sal+comm from emp;
2）分组函数忽略空值
SQL> select sum(sal),sum(sal+comm) from emp;　　为什么 sal+comm 的求和小于 sal 的求和？

```
  SUM(SAL) SUM(SAL+COMM)
---------- -------------
     29025          7800
```

3）比较表达式选择有空值（null）的数据行时，表达式返回为“假”，结果返回空行。
SQL>select ename,sal,comm from emp where sal>=comm;
4）非空字段与空值字段做"||"时, null 值转字符型""，合并列的数据类型为 varchar2。

资料整理
火龙果软件
荐

SQL> select ename,sal||comm from emp;
5）not in 在子查询中的空值问题（见第八章）
6）外键值可以为 null，唯一约束中，null 值可以不唯一（见十二章）
7）空值在 where 子句里使用“is null”或“is not null”
SQL> select ename,mgr from emp where mgr is null;
SQL> select ename,mgr from emp where mgr is not null;
8SQL> update emp set comm=null where empno=7788;

6.3.2 处理空值的几种函数方法：

1）nvl(expr1,expr2)
当第一个参数不为空时取第一个值，当第一个值为 NULL 时，取第二个参数的值。

```
SQL>select nvl(1,2) from dual;
   NVL(1,2)
----------
         1
SQL> select nvl(null,2) from dual;
NVL(NULL,2)
-----------
         2
```

nvl 函数可以作用于数值类型，字符类型，日期类型，但数据类型尽量匹配。
NVL(comm,0)
NVL(hiredate,'1970-01-01')
NVL(ename,'no manager')

2）nvl2(expr1,expr2,expr3)
当第一个参数不为 NULL，取第二个参数的值，当第一个参数为 NULL，取第三个数的值。

```
SQL> select nvl2(1,2,3) from dual;
NVL2(1,2,3)
-----------
         2

SQL> select nvl2(null,2,3) from dual;
NVL2(NULL,2,3)
--------------
             3

SQL> select ename,sal,comm,nvl2(comm,SAL+COMM,SAL) income,deptno from emp where deptno in (10,30);
```

考点:
1）nvl 和 nvl2 中的第二个参数不是一回事。
2）nvl2 的第二个参数和第三个参数要一致，如果不一致，第三个参数隐形转换成第二个参数
3）NULLIF（expr1,expr2)
当第一个参数和第二个参数相同时，返回为空，当第一个参数和第二个数不同时，返回第一个参数值，第一个参数值不允许为 null
SQL> select nullif(2,2) from dual;
SQL> select nullif(1,2) from dual;

4）coalesce（expr1,expr2.....)
返回从左起始第一个不为空的值，如果所有参数都为空，那么返回空值。
这里所有的表达式都是同样的数据类型
SQL> select coalesce(1,2,3,4) from dual;
SQL> select coalesce(null,2,null,4) from dual;

第七章、复杂查询(上)：多表连接技术

7.1 简单查询的解析方法：

全表扫描：指针从第一条记录开始，依次逐行处理，直到最后一条记录结束；
横向选择+纵向投影=结果集

7.2 多表连接

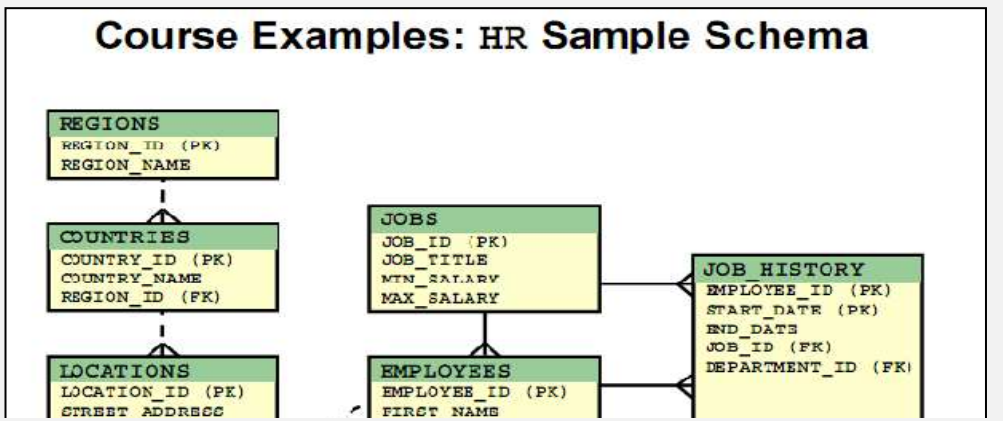
7.2.1 多表连接的优缺点

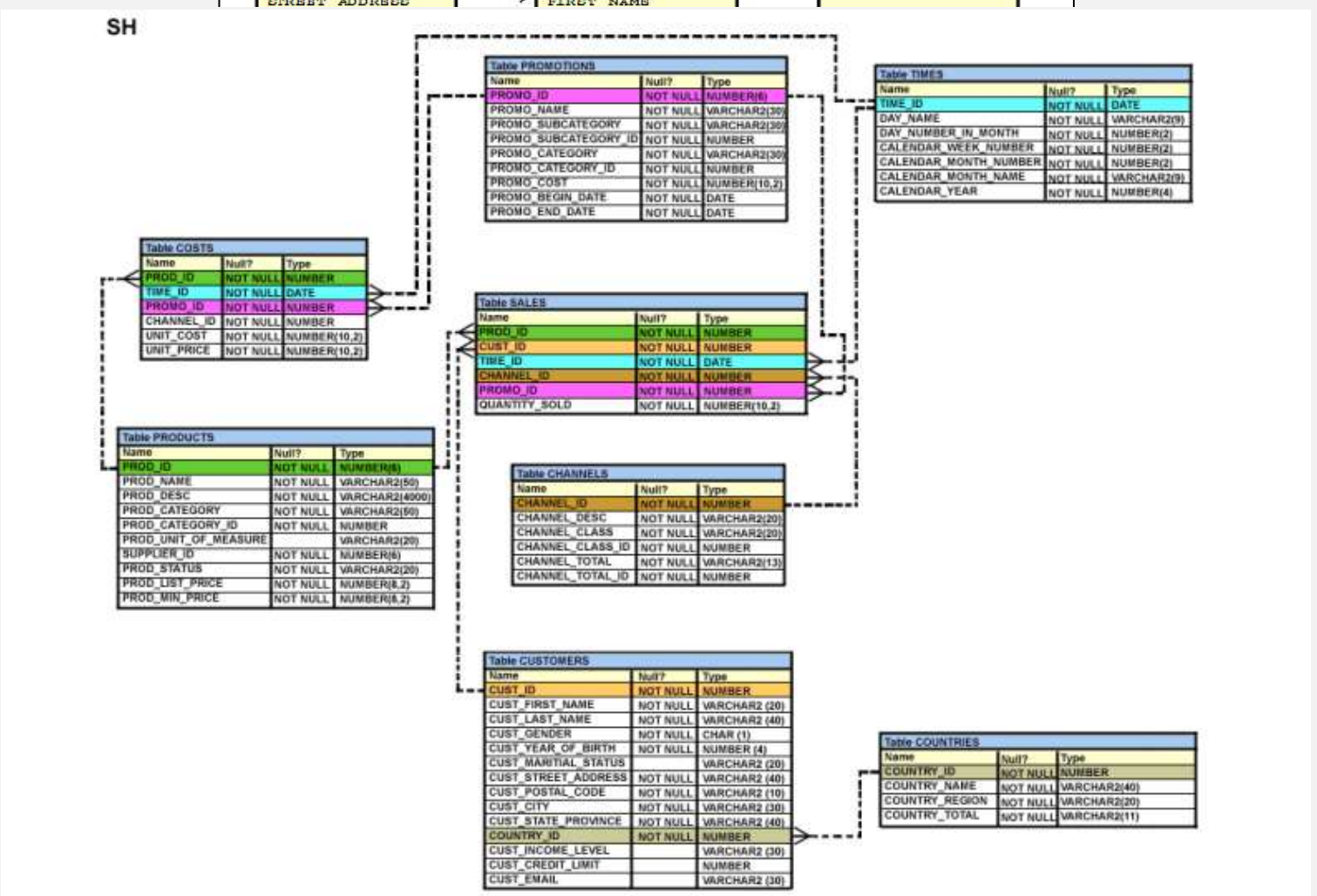
资料整理
火龙果软件
荐

优点：
1）减少冗余的数据，意味着优化了存储空间，降低了 IO 负担。
2）根据查询需要决定是否需要表连接。
3）灵活的增加字段，各表中字段相对独立（非主外键约束），增减灵活。
缺点：
1）多表连接语句可能冗长复杂，易读性差。
2）可能需要更多的 CPU 资源，一些复杂的连接算法消耗 CPU 和 Memory。
3）只能在一个数据库中完成多表连接查询。

7.2.2 多表连接中表的对应关系

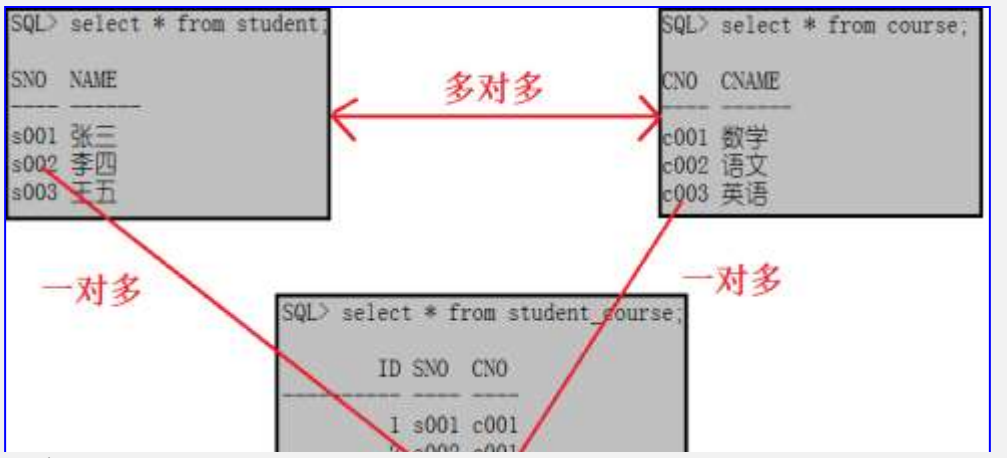
1）一对一关系
将表一份为二，最简单的对应关系
2）一对多关系
两表通过定义主外键约束，符合第三范式标准的对应关系。





3）多对多关系

资料整理
火龙果软件
荐



非标准的对应关系

当两表为多对多关系的时候，通常需要建立一个中间表，中间表至少要有两表的主键，这样，可使中间表分别与每个表为一对多关系。

7.2.3 多表连接的种类和语法

交叉连接（笛卡尔积）

非等值连接

等值连接(内连)

外连接（内连的扩展，左外，右外，全连接）

自连接

自然连接（内连，隐含连接条件,自动匹配连接字段）

复合连接（多个结果集进行并、交、差）

- SQL99语法:

```
SELECT   table1.column, table2.column
FROM     table1
JOIN     连接方法 table2 on 连接条件;
```

- Oracle语法:

```
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column1 = table2.column2;
```

7.2.1 交叉连接（笛卡尔积）

连接条件无效或被省略，两个表的所有行都发生连接，所有行的组合都会返回（n*m）

SQL99 写法：

SCOTT@ prod>select * from emp e cross join dept d;

Oracle 写法：

SCOTT@ prod>select * from emp e,dept d;

7.2.2 非等值连接：（连接条件没有“=”号）

SQL99 写法：

SCOTT@ prod>select empno,ename,sal,grade,losal,hisal from emp join salgrade on sal between losal and hisal;

Oracle 写法：

SCOTT@ prod>select empno,ename,sal,grade,losal,hisal from emp,salgrade where sal between losal and hisal;

7.2.3 等值连接，典型的内连接

SQL99 写法：

SCOTT@ prod>select e.ename,d.loc from emp e inner join dept d on e.deptno=d.deptno;

Oracle 写法：

SCOTT@ prod>select e.ename,d.loc from emp e,dept d where e.deptno=d.deptno;

7.2.4 等值连接的 using 字句（常用）

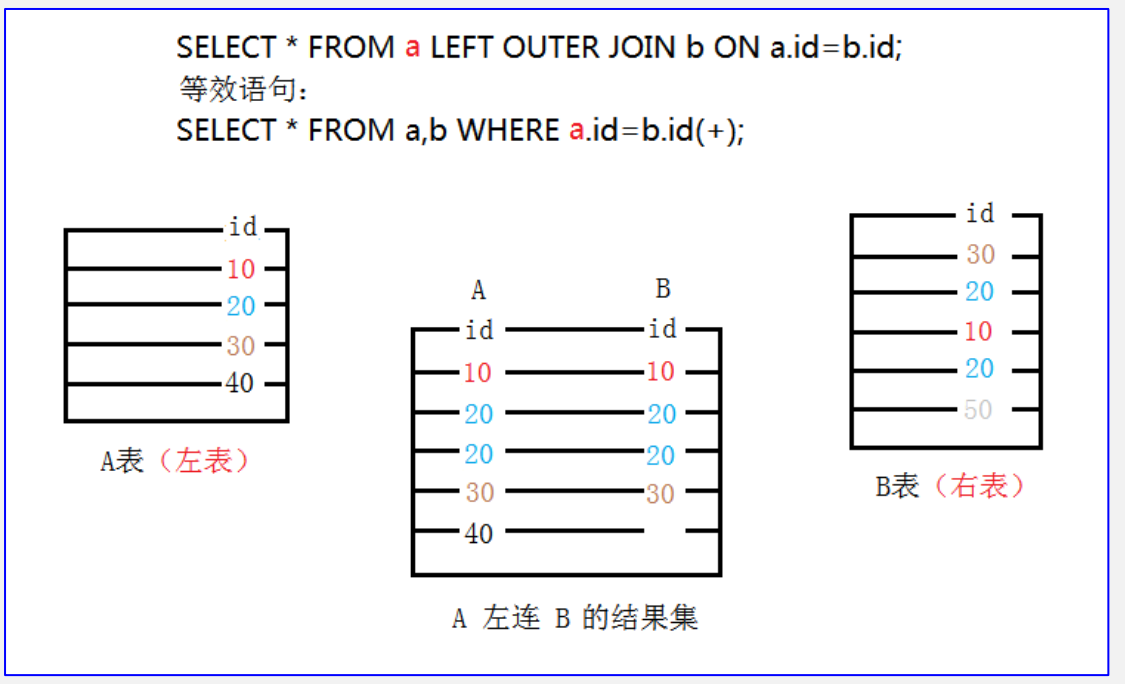
资料整理
火龙果软件
荐

等值连接的连接字段可以相同，
比如 on e.deptno=d.deptno;
也可以不同
比如 on e.empno=e.mgr
如果连接字段相同，可以使用 using 字句简化书写
如 on e.deptno=d.detpno.; 换成 using(deptno)
例:
SCOTT@ prod>select e.ename,d.loc from emp e inner join dept d using(deptno); using 里也可以多列

使用 using 关键字注意事项
1、如果 select 的结果列表项中包含了 using 关键字所指明的那个关键字，那么，不要指明该关键字属于哪个表。
2、using 中可以指定多个列名。
3、on 和 using 关键字是互斥的，也就是说不能同时出现。

7.2.5 外连接（包括左外连接，右外连接，全外连接）
1）左外连接语法
SQL99 语法:
SCOTT@ prod>select * from emp e left outer join dept d on e.deptno=d.deptno;
Oracle 语法:
SCOTT@ prod>select * from emp e,dept d where e.deptno=d.deptno(+);

2）左连接要理解两个关键点
1、如何确定左表和右表（左连右连都一样）
SQL99 写法：通过 from 后面表的先后顺序确定，第一个表为左表
SCOTT@ prod>select e.ename,d.loc from emp e left join dept d on e.deptno=d.deptno;
from 后第一个表是 emp 表，为左表，“=”左右位置无所谓
Oracle 写法：通过 where 后面的“=”的位置确定，“=”号左边的为左表
SCOTT@ prod>select e.ename,d.loc from emp e,dept d where e.deptno=d.deptno(+);
“=”左边是 emp 表，为左表，from 后面表位置无所谓
2、左连是左表为主
①左连是以左表为驱动，每行都参与匹配右表的行，匹配上就连成一行，如果匹配不上，左表行也不缺失该连接行，这时右表内容填空就是了。
②左连后，左表的行是不缺失的，即左连后的结果集的行数>=左表行数，存在>的可能是因为左表的一行可能匹配了右表的多行。
③也可以左表、右表都是同一个表，即“自左连”。

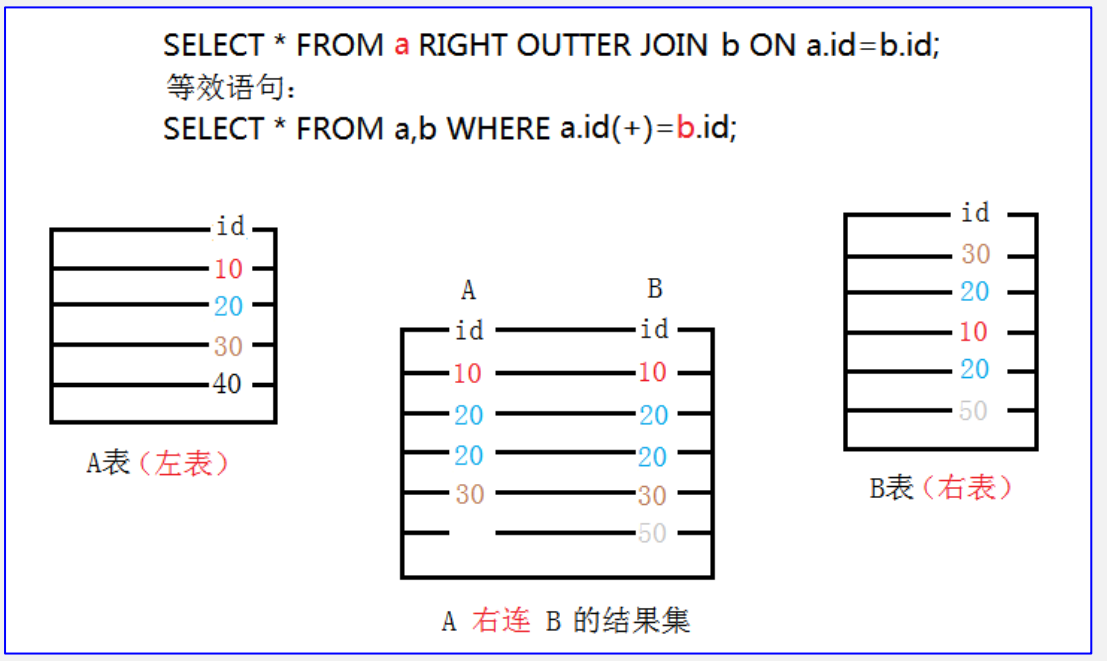


3、到底哪个表当左表好
无一定之规，根据业务需求来决定。
两表之间一般以主外键确定一对多关系，外键表是明细表，比如 emp 和 dept 的关系，以 deptno 确定父子关系，emp 是外键表
你要查每个员工的工作地点，这时以外键表（emp 明细表）做左表理所当然
SCOTT@ prod>select e.ename,d.loc from emp e left outer join dept d on e.deptno=d.deptno;
你要查每个部门有多少员工，这时以主键表(dept)做左表更合理
SCOTT@ prod>select d.deptno,count(e.deptno) from dept d left outer join emp e on e.deptno=d.deptno group by d.deptno;
推导一下：
SCOTT@ prod>select d.deptno,e.deptno from dept d left outer join emp e on e.deptno=d.deptno;

资料整理
火龙果软件
荐

以上三点，属于个人理解，这套法则同样适用于右连。

2）右外连接
SQL99 语法:
SCOTT@ prod>select * from emp e right join dept d on e.deptno=d.deptno;
Oracle 语法:
SCOTT@ prod> select * from emp e,dept d where e.deptno(+)=d.deptno;



3）全外连接
SQL99 语法:
SCOTT@ prod> select * from emp e full join dept d on e.deptno=d.deptno;
Oracle 语法:（无，等同于 union 连接）
SCOTT@ prod>
select * from emp e,dept d where e.deptno=d.deptno(+)
Union
select * from emp e,dept d where e.deptno(+)=d.deptno;

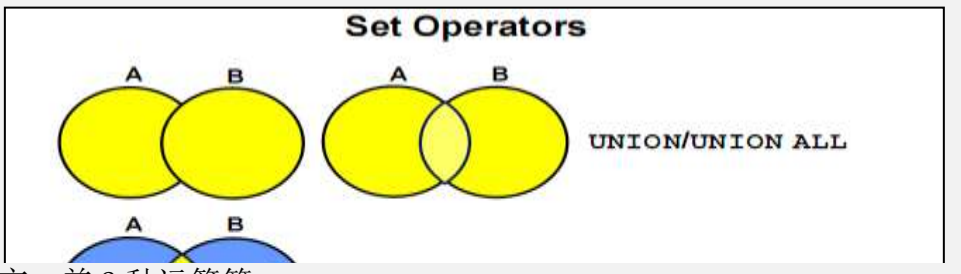
7.2.6 自连接
SQL99 语法：
SCOTT@ prod>select e1.empno,e2.mgr from emp e1 cross join emp e2;
Oracle 语法:
SCOTT@ prod> select e1.empno,e2.mgr from emp e1,emp e2;
必须使用别名区别不同的表

7.2.6 自然连接（属于内连中等值连接）
使用关键字 natural join,就是自然连接。
SCOTT@ prod>select e.ename,d.loc from emp e natural join dept d;
如果有多列复合匹配条件，则自动多列匹配

7.3 复合查询（使用集合运算符）

Union，对两个结果集进行并集操作，重复行只取一次，同时进行默认规则的排序；
Union All，对两个结果集进行并集操作，包括所有重复行，不进行排序；
Intersect，对两个结果集进行交集操作，重复行只取一次，同时进行默认规则的排序；
Minus，对两个结果集进行差操作，不取重复行，同时进行默认规则的排序。

资料整理
火龙果软件
荐



复合查询操作有并，交，差 3 种运算符。

示例：

```
SQL> create table dept1 as select * from dept where rownum <=1;
SQL> insert into dept1 values (80, 'MARKTING', 'BEIJING');
SQL> select * from dept;
     DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
SQL> select * from dept1;
     DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        80 MARKTING       BEIJING
```

1）union

```
SQL>
select * from dept
union
select * from dept1;
     DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        80 MARKTING       BEIJING
```

2）union all

```
SQL>
select * from dept
union all
select * from dept1;
     DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        10 ACCOUNTING     NEW YORK
        80 MARKTING       BEIJING
```

特别注意：可以看出只有 union all 的结果集是不排序的。

3）intersect

```
SQL>
```

资料整理
火龙果软件
荐

```
select * from dept
intersect
select * from dept1;
      DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
```

4）minus（注意谁 minus 谁）

```
SQL>
select * from dept
minus
select * from dept1;
      DEPTNO DNAME          LOC
---------- -------------- -------------
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL>
select * from dept1
minus
select * from dept;
      DEPTNO DNAME          LOC
---------- -------------- -------------
        80 MARKTING       BEIJING
```

7.4 复合查询几点注意事项

1）列名不必相同，但要类型匹配且顺序要对应，大类型对上就行了，比如 char 对 varchar2，date 对 timestamp 都可以，字段数要等同，不等需要补全。

```
create table a (id_a int,name_a char(10));
create table b (id_b int,name_b char(10),sal number(10,2));

insert into a values (1, 'sohu');
insert into a values (2, 'sina');

insert into b values (1, 'sohu', 1000);
insert into b values (2, 'yahoo', 2000);
commit;

SQL> select * from a;
        ID_A NAME_A
---------- ----------
         1 sohu
         2 sina

SQL> select * from b;

        ID_B NAME_B          SAL
---------- ---------- ----------
         1 sohu           1000
         2 yahoo          2000

SQL>
select id_a,name_a from a
union
select id_b,name_b from b;
```

2）四种集合运算符优先级按先后出现的顺序执行，如有特殊要求可以使用（）。

3）关于复合查询中 order by 使用别名排序的问题：

资料整理 火龙果软件 荐

①缺省情况下，复合查询后的结果集是按所有字段的组合隐式排序的（除 union all 外）
如果不希望缺省的排序，也可以使用 order by 显式排序

```
select id_a, name_a name from a
union
select id_b, name_b name from b
order by name;

select id_a, name_a from a
union
select id_b, name_b from b
order by 2;
```

②显式 order by 是参照第一个 select 语句的列元素。所以，order by 后的列名只能是第一个 select 使用的列名、别名、列号（考点）。如果是补全的 null 值需要 order by，则需要使用别名。

```
SQL>
select id_a, name_a name,to_number(null) from a
union
select id_b, name_b name,sal from b
order by sal;
```

报错：ORA-00904: "SAL": 标识符无效

以下三种写法都是正确的

```
SQL>
select id_a, name_a name,to_number(null) from a
union
select id_b, name_b name,sal from b
order by 3;

SQL>
select id_b, name_b name,sal from b
union
select id_a, name_a name,to_number(null) from a
order by sal;

SQL>
select id_a, name_a name,to_number(null) aa from a
union
select id_b, name_b name,sal aa from b
order by aa;
```

③排序是对复合查询结果集的排序，不能分别对个别表排序，order by 只能一次且出现在最后一行；

```
SQL>
select id_a, name_a from a order by id_a
union
select id_b, name_b from b order by id_b;
```

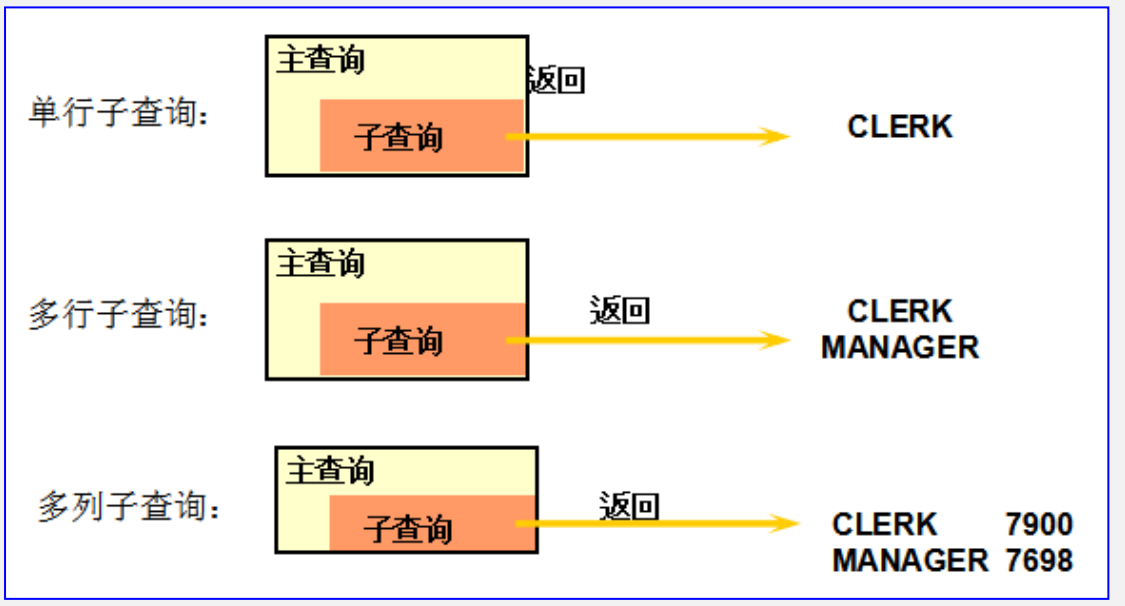
报错：ORA-00933: SQL 命令未正确结束

第八章、复杂查询(下)：子查询

8.1 非关联子查询：

返回的值可以被外部查询使用。子查询可以独立执行的（且仅执行一次）。

资料整理
火龙果软件
荐



8.1.1 单行单列子查询

子查询仅返回一个值，也称为标量子查询，采用单行比较运算符（>，<，=，<>,>=，<=）

例：内部 SELECT 子句只返回单值

```
SQL>select ename,sal
from emp
where sal > (
        select sal from emp
        where ename='JONES')
/
```

8.1.2 多行单列子查询

子查询返回一列多行，采用多行比较运算符（in，not in，all, any）

1）使用 in　(逐个比较是否有匹配值）

```
SQL> select ename, sal from emp where sal in (800,3000,4000);
```

例：显示出 emp 表中那些员工不是普通员工（属于大小领导的）。

```
SQL>select ename from emp where empno in (select mgr from emp);
```

2）使用 not in　(子查询不能返回空值）

"in"与"not in"遇到空值时情况不同，对于"not in"　如果子查询的结果集中有空值，那么主查询得到的结果集也是空。

例：查找出没有下属的员工,即普通员工，（该员工号不在 mgr 之列的）

```
SQL>select ename from emp where empno not in (select mgr from emp);
no rows selected
```

上面的结果不出所料，主查询没有返回记录。这个原因是在子查询中有一个空值，而对于 not　in 这种形式，一旦子查询出现了空值，则主查询记录结果也就返回空了

排除空值的影响

```
SQL>select ename from emp where empno not in (select nvl(mgr,0)from emp);
```

注意：not 后不能跟单行比较符，只有 not in 组合，也没有 not any　和 not all 的组合，但 not 后可以接表达式　如：where empno not in(...）与 where not empno in(...)两个写法都是同样结果，前者是 not in 组合，后者是 not 一个表达式。

3）使用 all　（>大于最大的，<小于最小的）

```
SQL> select ename,sal from emp where sal >all (2000,3000,4000);
```

例：查找高于所有部门的平均工资的员工（>比子查询中返回的列表中最大的大才行）

```
SQL> select ename, job, sal from emp where sal > all(select avg(sal) from emp group by deptno);
ENAME        JOB              SAL
---------- --------- ----------
JONES       MANAGER          2975
SCOTT       ANALYST          3000
KING        PRESIDENT        5000
FORD        ANALYST          3000
```

4）在多行子查询中使用 any　（>大于最小的，<小于最大的）

>any 的意思是：>比子查询中返回的列表中最小的大就行，注意和 all 的区别，all 的条件苛刻，any 的条件松阔，any 强调的是只要有任意一个符合就行了，所以>any 只要比最小的那个大就行了，没必要比最大的还大。

```
select ename, sal from emp where sal >any (2000,3000,4000);
```

8.1.3 多行多列子查询

资料整理
火龙果软件
荐

子查询返回多列结果集。有成对比较、非成对比较两种形式。
测试准备

```
SQL>create table emp1 as select * from emp;
SQL>update emp1 set sal=1600,comm=300 where ename='SMITH';      SMITH 是 20 部门的员工
SQL>update emp1 set sal=1500,comm=300 where ename='CLARK';      CLARK 是 10 部门的员工
SQL> select * from emp1;
SQL> select empno,ename,sal,comm,deptno from emp1;
     EMPNO ENAME                SAL       COMM     DEPTNO
---------- ---------- ---------- ---------- ----------
      7369 SMITH              1600        300         20
      7499 ALLEN              1600        300         30
      7521 WARD               1250        500         30
      7566 JONES              2975                    20
      7654 MARTIN             1250       1400         30
      7698 BLAKE              2850                    30
      7782 CLARK              1500        300         10
      7788 SCOTT              3000                    20
      7839 KING               5000                    10
      7844 TURNER             1500          0         30
      7876 ADAMS              1100                    20
      7900 JAMES               950                    30
      7902 FORD               3000                    20
      7934 MILLER             1300                    10
```

查询条件：查找 emp1 表中是否有与 30 部门的员工工资和奖金相同的其他部门的员工。
注意看一下：现在 20 部门的 SIMTH 符合这个条件，它与 30 部门的 ALLEN 有相同的工资和奖金

1）成对比较多列子查询：
特点是主查询每一行中的列都要与子查询返回列表中的相应列同时进行比较，只有各列完全匹配时才显示主查询中的该数据行。

```
SQL>
select ename,deptno,sal,comm from emp1
   where (sal,comm) in (select sal,comm from emp1 where deptno=30)
and deptno<>30
/
ENAME                DEPTNO        SAL       COMM
---------- ---------- ---------- ----------
SMITH                    20       1600        300
```

考点：1）成对比较是不能使用>any 或>all 等多行单列比较符的。2）成对比较时的多列顺序和类型必须一一对应。

2）非成对比较成对多列子查询（含布尔运算）
例：非成对比较

```
SQL>select ename,deptno,sal,comm from emp1
where sal in(
            select sal
            from emp1
            where deptno=30)
and
            nvl(comm,0) in (
            select nvl(comm,0)
            from emp1
            where deptno=30)
and deptno<>30
/
ENAME                DEPTNO        SAL       COMM
---------- ---------- ---------- ----------
SMITH                    20       1600        300
CLARK                    10       1500        300
```

两个子查询返回的值分别与主查询中的 sal 和 comm 列比较，
如果员工的工资与 30 部门任意一个员工相同，同时，奖金也与 30 部门的其他员工相同，那么得到了两个员工的信息。
可见，成对比较(使用 where （列，列）)比非成对比较（使用 where 列 and 列) 更为严苛。

资料整理
火龙果软件
荐

8.1.4 关于布尔运算符 not

not 就是否定后面的比较符，基本的形式如下

where empno=7788	where NOT (empno=7788)
where ename LIKE 'S%'	where ename NOT LIKE 'S%'
where deptno IN (20,30)	where deptno NOT IN (20,30)
where sal BETWEEN 1500 AND 3000	where sal NOT BETWEEN 1500 AND 3000
where comm IS NULL	where comm IS NOT NULL
where EXISTS (select子查询)	where NOT EXISTS (select子查询)

8.1.5 from 子句中使用子查询(也叫内联视图）

例：员工的工资大于他所在的部门的平均工资的话，显示其信息。

分两步来考虑：

第一步，先看看每个部门的平均工资，再把这个结果集作为一个内联视图。

```
SQL> select deptno,avg(sal) salavg from emp group by deptno;
      DEPTNO        SALAVG
---------- ----------
            30 1566.66667
            20          2175
            10 2916.66667
```

第二步，把这个内联视图起一个别名 b，然后和 emp 别名 e 做连接，满足条件即可。

```
SQL>
select e.ename, e.sal, e.deptno, b.salavg
from emp e, (select deptno,avg(sal) salavg from emp group by deptno) b
where e.deptno=b.deptno and e.sal > b.salavg
/
ENAME                    SAL        DEPTNO        SALAVG
---------- ---------- ---------- ----------
ALLEN                   1600            30     1566.66667
JONES                   2975            20          2175
BLAKE                   2850            30     1566.66667
SCOTT                   3000            20          2175
KING                    5000            10     2916.66667
FORD                     3000            20          2175
```

8.1.6 Update 使用非关联查询（考点）

范例：

```
SCOTT@ prod>create table emp1 as select empno,ename,sal from emp where sal>2500;
SCOTT@ prod>select * from emp1;

        EMPNO ENAME                    SAL
---------- ---------- ----------
          7566 JONES                  2975
          7698 BLAKE                  2850
          7788 SCOTT                  3000
          7839 KING                   5000
          7902 FORD                    3000
SCOTT@  prod>  UPDATE  (SELECT  empno,sal  FROM  emp1)  SET  sal=10000  WHERE  empno  in  (SELECT  empno  FROM  emp  WHERE
sal >=3000 );
```

这里 update 后面的 select 子句仅提供可选择更新的列

```
SCOTT@ prod>select * from emp1;

        EMPNO ENAME                    SAL
---------- ---------- ----------
          7566 JONES                  2975
          7698 BLAKE                  2850
          7788 SCOTT                 10000
          7839 KING                  10000
          7902 FORD                   10000
```

资料整理
火龙果软件
荐

8.1.7 delete 使用非关联查询

8.2 关联子查询

8.2.1 特点：
其子查询（内部，inner)会引用主查询(外部，outer)查询中的一列或多列。在执行时，外部查询的每一行都被一次一行地传递给子查询，子查询依次读取外部查询传递来的每一值，并将其用到子查询上,直到外部查询所有的行都处理完为止，最后返回查询结果。
理论上主查询有 n 行，子查询被调用 n 次。
示例 1 关联查询用于 select 语句
使用关联查询，显示员工的工资大于他所在部门的平均工资（对比非关联查询的例子）。
SQL> select ename,sal,deptno from emp outer where sal> (select avg(sal) from emp inner where inner.deptno=outer.deptno);

示例 2 关联查询中的特殊形式，使用 EXISTS 或 NOT EXISTS
EXISTS 关心的是在子查询里能否找到一个行值（哪怕有 10 行匹配，只要找到一行就行），如果子查询有行值，则立即停止子查询的搜索，然后返回逻辑标识 TRUE, 如果子查询没有返回行值，则返回逻辑标识 FALSE， 子查询要么返回 T，要么返回 F，以此决定了主查询的调用行的去留，然后主查询指针指向下一行，继续调用子查询...

①EXISTS 的例子：显示出 emp 表中那些员工不是普通员工（属于大小领导的）。
SQL> select empno,ename,job,deptno from emp outer where exists (select 'X' from emp where mgr=outer.empno);
说明：exists 子查询中 select 后的‘X'只是一个占位，它返回什么值无关紧要，它关心的是子查询中否‘存在’，即子查询的 where 条件能否有‘结果’，一旦子查询查到一条记录满足 where 条件，则立即返回逻辑‘TRUE’，（就不往下查了）。否则返回‘FALSE’。
②NOT EXISTS 的例子：显示 dept 表中还没有员工的部门。
SQL> select deptno,dname from dept d where not exists (select 'X' from emp where deptno=d.deptno);

```
    DEPTNO DNAME
---------- --------------
        40 OPERATIONS
```

对于关联子查询，在某种特定的条件下，比如子查询是个大表，且连接字段建立了索引，那么使用 exists 比 in 的效率可能更高。

8.2. Update 使用关联查询
范例
1）建立 EMP1 表
SQL> create table emp1 as select e.empno,e.sal,d.loc,d.deptno from emp e,dept d
where e.deptno=d.deptno and e.sal>2000 order by 1;
SQL> update emp1 set loc=null;
SQL> commit;
2）将 EMP1 表 LOC 字段清空，并提交，然后再将数据回填，恢复到提交前的状态。
SQL> update emp1 e set e.loc=(select d.loc from dept d where d.deptno=e.deptno);
SQL> commit;

8.3 别名的使用

有表别名和列别名, 表别名用于多表连接或子查询中，列别名用于列的命名规范。
如果别名的字面值有特殊字符，需要使用双引号。如："AB C"
必须使用别名的地方：
1）两表连接后，select 投影中有相同命名的列，必须使用表别名区别标识（自然连接中的公共列则采用相反规则）
SQL>select ename,d.deptno from emp e,dept d where e.deptno=d.deptno;
2）使用 create * {table |view} as select ...语句创建一个新的对象，其字段名要符合对象中字段的规范，不能是表达式或函数等非规范字符，而使用别名可以解决这个问题。
SQL>create table emp1 as select deptno,avg(sal) salavg from emp group by deptno;
SQL>create or replace view v as select deptno,avg(sal) salavg from emp group by deptno;
或 SQL> create or replace view v(deptno,salavg) as select deptno,avg(sal) from emp group by deptno;
3）使用内联视图时，若 where 子句还要引用其 select 中函数的投影, 使用别名可以派上用场。
SQL>select * from (select avg(sal) salavg from emp) where salavg>2000;
4）当以内联视图作为多表连接，主查询投影列在形式上不允许单行字段（或函数）与聚合函数并列，解决这个问题是使在内联视图中为聚合函数加别名，然后主查询的投影中引用其别名。
5）rownum 列是 Oracle 的伪列，加别名可以使它成为一个表列，这样才可以符合 SQL99 标准中的连接和选择。

资料整理
火龙果软件
荐

SQL> select * from (select ename,rownum rn from emp) where rn>5;
6）不能使用别名的地方：
在一个独立的 select 结构的投影中使用了列别名，不能在其后的 where 或 having 中直接引用该列别名（想想为什么？）。
SQL> select ename,sal salary from emp where salary>2000;　　错
SQL> select deptno,avg(sal) salavg from emp group by deptno having salavg>2000;　　错

8.4 综合练习

练习一、 找出员工工资最高的那个人的工作地点（非关联查询）
两类思路：1）一网打尽型的 2）顺藤摸瓜型的。
1）SQL>select ename,sal,loc from emp e,dept d where e.deptno=d.deptno and sal=(select max(sal) from emp);
2）SQL>select ename,sal,loc from (select * from emp where sal=(select max(sal) from emp)) a,dept b where a.deptno=b.deptno(+);
3）SQL>select ename,sal,loc from (select * from emp where sal>=all(select sal from emp)) a,dept b where a.deptno=b.deptno;
4）SQL>select ename,sal,loc from (select * from emp e,dept d where e.deptno=d.deptno order by sal desc) where rownum=1;

练习二、 员工的工资大于他所在的部门的平均工资的话，显示其信息。
1）SQL>select e.ename, e.sal, e.deptno, b.salavg
from emp e, (select deptno,avg(sal) salavg from emp group by deptno) b
where e.deptno=b.deptno and e.sal > b.salavg;　　(非关联查询)

2）SQL> select ename,sal,deptno from emp outer where sal> (select avg(sal) from emp inner where inner.deptno=outer.deptno);　　（关联查询）

练习三、
1）显示出 emp 表中哪些员工不是普通员工（属于大小领导的）。
SQL>select ename from emp where empno in (select mgr from emp);　　(非关联查询)
SQL>select empno,ename,job,deptno from emp outer where exists (select 'X' from emp where mgr=outer.empno);　　（关联查询）

2）查找出没有下属的员工（属于普通员工）
SQL>select ename from emp where empno not in (select distinct (nvl(mgr,0)) from emp);
SQL>select empno,ename,job,deptno from emp outer where not exists (select 'X' from emp where mgr=outer.empno);

练习四、查找高于所有部门的平均工资的员工（>比子查询中返回的列表中最大的大才行）
SQL> select ename, job, sal from emp where sal > all(select avg(sal) from emp group by deptno);

练习五、要求列出 emp 表第 5-第 10 名员工（按 sal 大--小排序）的信息（结果集的分页查询技术）
关于 rownum 伪列使用需要注意两点：
1，使用 rownum 筛选时必须显示或隐示的包含第一行，否则不会返回任何行
2，rownum 是在结果集确定后才编号的。
SQL>select * from (select t1.*,rownum rn from (select * from emp order by sal desc) t1 where rownum<=10) where rn>=5;　　高效
SQL>select * from (select t1.*, rownum rn from (select * from emp order by sal desc) t1) where rn between 5 and 10;　　低效

练习六、
1）从列出 emp 表中显示员工和经理对应关系表。（自连接）
SQL>select ,a.ename, WORKER,b.ename BOSS from emp a, emp b where a.mgr=b.empno;

2）一个叫 team 的表，里面只有一个字段 name，一共有 4 条纪录，分别是 a,b,c,d，对应四个球队，现在四个球队进行比赛，用一条 sql 语句显示所有可能的比赛组合（面试题）
create table team (name char(4));
insert into team values('a');
insert into team values('b');
insert into team values('c');
insert into team values('d');
答案：
select * from team a,team b where a.name<b.name;

练习七、根据一张表修改另一张表 （面试题）
有两个表 A 和 B ，均有 key 和 value 两个字段，如果 B 的 key 在 A 中也有，就把 B 的 value 换为 A 中对应的 value
这道题的 SQL 语句怎么写？（关联查询）
。

资料整理
火龙果软件
荐

准备：
create table emp1 as select * from emp;
update emp1 set empno=empno+1000 where sal>2000;
update emp1 set sal=sal+1000;
commit;
答案：
update emp1 b set sal=(select a.sal from emp a where a.empno=b.empno) where b.empno in (select a.empno from emp a);

练习八、表 dept1 增加一列 person_count，要求根据 emp 表填写 dept1 表的各部门员工合计数。（关联查询）
CTAS 方法建立 dept1，将 dept1 增加一列 person_count
create table dept1 as select * from dept;
alter table dept1 add person_count int;
update dept1 d set person_count=(select count(*) from emp e where e.deptno=d.deptno);
select * from dept1;

练习九、消除重复行（面试题）
1）建立 EMP1 表
SQL> create table emp1 as select e.empno,e.sal,d.loc,d.deptno from emp e,dept d
where e.deptno=d.deptno and e.sal>2000 order by 1;
2）EMP1 表插入两条重复记录（比如：选择 7788 和 7902），并提交，然后再删掉重复记录，恢复到提交状态。
SQL> insert into emp1 select * from emp1 where empno in (7788,7902);
SQL> commit；
SQL> delete emp1 where rowid not in (select min(rowid) from emp1 group by empno,sal,loc,deptno);
SQL> commit;

练习十、分组函数练习（面试题）
环境准备：
create table s1(name char(6),subject char(8),score int);

insert into s1 values('张三','语文',79);
insert into s1 values('张三','数学',75);
insert into s1 values('李四','语文',76);
insert into s1 values('李四','数学',90);
insert into s1 values('王五','语文',90);
insert into s1 values('王五','数学',100);
insert into s1 values('王五','英语',81);
commit;

要求 1：查询出每门课都大于 80 分的学生姓名
1 SQL> select name from s1 group by name having min(score)>=80; （利用分组函数）
2 SQL> select distinct name from s1 where name not in (select distinct name from s1 where score<80);（利用 distinct 关键字）

要求 2：查出表中哪一门课程的平均分数最高
SQL> select subject from s1 group by subject having avg(score)=(select max(avg(score)) from s1 group by subject);

练习十一、行列转换典型示例（面试题）

表 t1

YEAR	MONTH	AMOUNT
1991	1	1.1
1991	2	1.2
1991	3	1.3
1991	4	1.4
1992	1	2.1
1992	2	2.2
1992	3	2.3
1992	4	2.4

表 t2

资料整理
火龙果软件
荐

YEAR	M1	M2	M3	M4
1991	1.1	1.2	1.3	1.4
1992	2.1	2.2	2.3	2.4

行转列，将表 1 转换生成表 2（t1----->t2）

利用 decode 函数及分组特性

a）建立原始表 t1

```
create table t1(year number(4),month number(2),amount number(2,1));

insert into t1 values(1991,1,1.1);
insert into t1 values(1991,2,1.2);
insert into t1 values(1991,3,1.3);
insert into t1 values(1991,4,1.4);
insert into t1 values(1992,1,2.1);
insert into t1 values(1992,2,2.2);
insert into t1 values(1992,3,2.3);
insert into t1 values(1992,4,2.4);
commit;
```

b) 转换语句

```
create table t2 as select year,
max(decode(month,1,amount,null)) m1,
max(decode(month,2,amount,null)) m2,
max(decode(month,3,amount,null)) m3,
max(decode(month,4,amount,null)) m4
from t1 group by year;
```

练习十二，三表连接，显示学生与选学课程的对应关系（面试题）

环境准备：

```
create table student(sno char(4),name char(6));
create table course(cno    char(4),cname char(6));

insert into student values('s001','张三');
insert into student values('s002','李四');
insert into student values('s003','王五');
commit;

insert into course values('c001','数学');
insert into course values('c002','语文');
insert into course values('c003','英语');
commit;

create table student_course (id int,sno char(4),cno char(4));

insert into student_course values(1,'s001','c001');
insert into student_course values(2,'s002','c001');
insert into student_course values(3,'s002','c002');
insert into student_course values(4,'s003','c003');
insert into student_course values(5,'s003','c001');
commit;

SQL> select * from student;
SNO    NAME
---- ------
s001  张三
s002  李四
```

资料整理
火龙果软件
荐

s003 王五

```
SQL> select * from course;
CNO    CNAME
---- ------
c001 数学
c002 语文
c003 英语

SQL> select * from student_course;
        ID SNO    CNO
---------- ---- ----
         1 s001 c001
         2 s002 c001
         3 s002 c002
         4 s003 c003
         5 s003 c001
```

一个学生可能选修多门课程，而一门课程也会有多个学生选修，所以 student 和 course 是多对多关系，这两个表之间没有直接关系 stuent 和 course 的关系由中间表 student_course 给出。
现在要列出每个学生的选修课程，分析一下它的连接过程：

第一步（中间过程）
SQL99 写法：

```
SQL> select * from student_course sc left join student s on sc.sno=s.sno;
```

Oracle 写法：

```
SQL> select * from student_course sc,student s where sc.sno=s.sno(+);
        ID SNO    CNO    SNO    NAME
---------- ---- ---- ---- ------
         1 s001 c001 s001 张三
         3 s002 c002 s002 李四
         2 s002 c001 s002 李四
         5 s003 c001 s003 王五
         4 s003 c003 s003 王五
```

第二步（中间过程）
SQL99 写法：

```
SQL> select * from student_course sc left join student s on sc.sno=s.sno left join course c on sc.cno=c.cno;
```

Oracle 写法：

```
SQL> select * from student_course sc,student s,course c where sc.sno=s.sno(+) and sc.cno=c.cno(+);
        ID SNO    CNO    SNO    NAME     CNO    CNAME
---------- ---- ---- ---- ------ ---- ------
         5 s003 c001 s003 王五     c001 数学
         2 s002 c001 s002 李四     c001 数学
         1 s001 c001 s001 张三     c001 数学
         3 s002 c002 s002 李四     c002 语文
         4 s003 c003 s003 王五     c003 英语
```

第三步（最终结果）
SQL99 写法：

```
SQL> select s.name,c.cname from student_course sc left join student s on sc.sno=s.sno left join course c on sc.cno=c.cno;
```

Oracle 写法：

```
SQL> select s.name,c.cname from student_course sc,student s,course c where sc.sno=s.sno(+) and sc.cno=c.cno(+);
NAME     CNAME
------ ------
王五     数学
李四     数学
张三     数学
李四     语文
王五     英语
```

资料整理
火龙果软件
荐

练习十三 影院票房统计（面试题）

数据表 1

```
create table busdata(ydate date,cid number(3),box number(5),person number(5));
insert into busdata values(to_date('2016-01-01','yyyy-mm-dd'),1,100,10);
insert into busdata values(to_date('2016-01-01','yyyy-mm-dd'),2,110,15);
insert into busdata values(to_date('2016-01-01','yyyy-mm-dd'),3,120,20);
insert into busdata values(to_date('2016-01-02','yyyy-mm-dd'),1,150,25);
insert into busdata values(to_date('2016-01-02','yyyy-mm-dd'),2,160,30);
insert into busdata values(to_date('2016-01-02','yyyy-mm-dd'),3,170,35);
insert into busdata values(to_date('2016-01-03','yyyy-mm-dd'),1,200,40);
insert into busdata values(to_date('2016-01-03','yyyy-mm-dd'),2,210,45);
insert into busdata values(to_date('2016-01-03','yyyy-mm-dd'),3,220,50);
commit;
```

数据表 2

```
create table cinema(cid number(3),cname varchar2(12));
insert into cinema values(1,'五棵松店');
insert into cinema values(2,'马连道店');
insert into cinema values(3,'慈云寺店');
commit;
```

数据表 3

```
create table mstak(mdate date,cid number(3),mnum number(5));
insert into mstak values(to_date('2016-01-01','yyyy-mm-dd'),1,500);
insert into mstak values(to_date('2016-01-01','yyyy-mm-dd'),2,600);
commit;
```

1.查询 2016/1/1 各影城营业数据

结果表头：营业日期、影城名称、票房、人次。

```
select b.ydate 营业日期，c.cname 影城名称,b.box 票房,b.person 人次
from busdata b,cinema c where b.cid=c.cid
and ydate='2016-01-01 00:00:00';
```

2.查询 2016/1/1 所有影城总营业数据

结果表头：营业日期、总票房、总人数

```
SELECT ydate,SUM(box),SUM(person) FROM busdata WHERE ydate='2016-01-01 00:00:00' GROUP BY ydate;
```

3.查询日期从 2016/1/1 到 2016/1/3 各影城营业数据

结果表头：影城名称、总票房、总人次

```
SELECT c.cname,t.zpf,t.zrs
FROM cinema c,(SELECT cid,SUM(box) zpf,SUM(person) zrs
                FROM busdata
                WHERE ydate BETWEEN '2016-01-01 00:00:00' AND '2016-01-03 00:00:00' GROUP BY cid) t
WHERE c.cid=t.cid;
```

4.查询日期从 2016/1/1 到 2016/1/3 所有影城总营业数据

结果表头：总票房、总人次

```
SELECT SUM(box) zpf, SUM(person) zrs
   FROM busdata
  WHERE ydate BETWEEN '2016-01-01 00:00:00' AND '2016-01-03 00:00:00';
```

5.首先查询 2016/1/1 各影城营业数据，再查询 2016/1/1 至 2016/1/3 各影城营业数据。将二者合并在一个查询结果集中显示

查询结果如下

资料整理
火龙果软件
荐

A	B	C	D	E	F
营业日期	影城名称	票房	人次	累计票房	累计人次
2016/1/1	五棵松店	100	10	330	45
2016/1/1	马连道店	150	25	450	50
2016/1/1	慈云寺店	200	40	630	135

```
SELECT a.w,a.x,a.y,a.z,b.aa,b.bb FROM
(SELECT t.ydate w,c.cname x,t.box y,t.person z
FROM cinema c,busdata t
WHERE ydate='2016-01-01 00:00:00' AND c.cid=t.cid) a
JOIN
(SELECT b.cname xx,t.zpf aa,t.zrs    bb
FROM cinema b,(SELECT cid,SUM(box) zpf,SUM(person) zrs
FROM busdata
WHERE ydate BETWEEN '2016-01-01 00:00:00' AND '2016-01-03 00:00:00' GROUP BY cid) t
WHERE b.cid=t.cid) b
ON a.x=b.xx;
```

6.查询 1 月份各影城票房完成情况在本例中则为 2016/1/1 至 2016/1/3，说明：“任务完成率”。“总票房”/“票房月任务”。要求：显示全部的影城信息，也就说会有三行数据影城名称分别为：“五棵松店”、“马连道店”、“慈云寺点”。
查询结果如下：

影城名称	总票房	总人次	月任务	完成率
五棵松店	130	45	500	60%
马连道店	480	50	600	50%
慈云寺店	630	125	***	***

```
SELECT a.x,a.y,a.z,b.mnum,a.y/b.mnum*100||'%' wcl
FROM
(SELECT c.cid w,c.cname x,t.zpf y,t.zrs z
FROM cinema c,(SELECT cid,SUM(box) zpf,SUM(person) zrs
                FROM busdata
                WHERE    ydate BETWEEN '2016-01-01 00:00:00' AND '2016-01-03 00:00:00' GROUP BY cid) t
WHERE c.cid=t.cid) a
LEFT JOIN
(select cid,mnum from MSTAK) b
ON a.w=b.cid;
```

第二部分、管理用户及对象

第九章、用户访问控制

9.1 认识 Oracle 用户

Oracle 数据库是多用户系统，每个用户一个账户。

9.1.1 查看数据库里有多少用户？

SQL>select username from dba_users;

9.1.2 用户默认的存储空间

每个用户账户都可以指定默认的表空间，用户创建的任何对象（如表或索引）将缺省保存在此表空间中，如果创建用户时没有指定默认表空间，那么该用户缺省使用数据库级的默认表空间。

9.1.3 数据库默认表空间

SQL> select * from database_properties where rownum<=10;

设置数据库的默认表空间

SQL> alter database default tablespace TABLESPACENAME;

9.2 密码认证介绍

用户登录时需要密码认证，认证主要有几种方法

1）OS 认证　　用于 sys 用户本地登录

2）数据字典口令认证　　用户普通用户登录

3）口令文件认证　　用于 sys 用户远程登录

以上三种用户认证是最常用的。

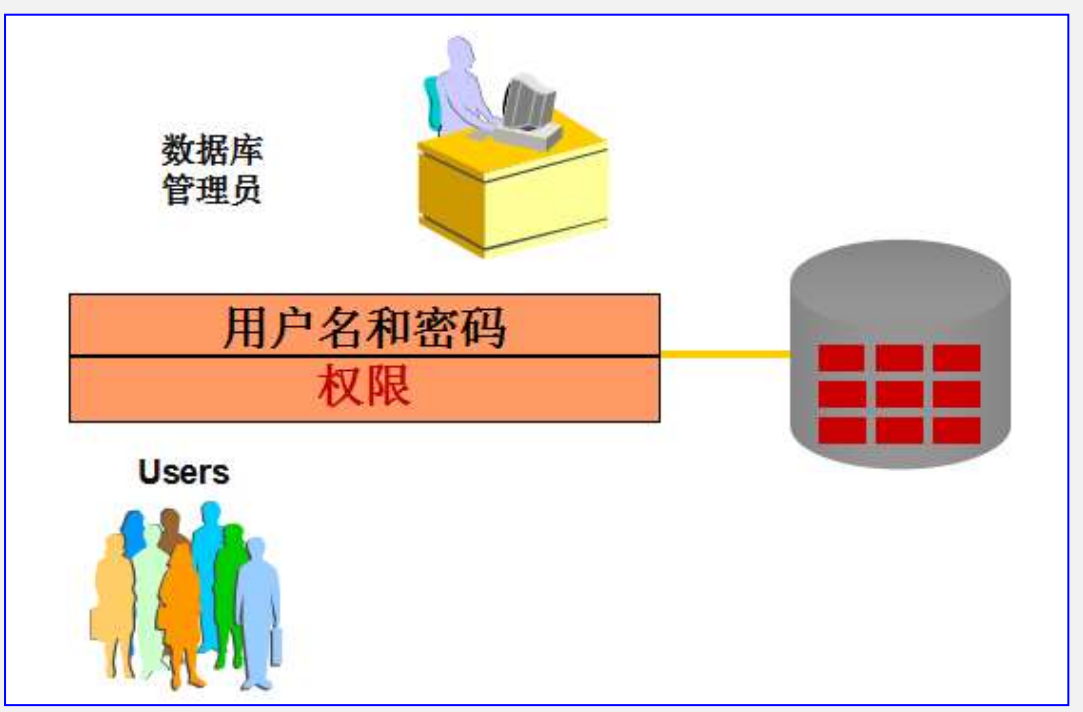
另外还有一些认证方式，考试会提及，比如：

资料整理
火龙果软件
荐

4）外部密码认证
如果配置了 os_authent_prefix 参数，如缺省值为'ops$'，当数据库中存在用户'ops$tim'，且对该用户启用了外部验证。那么在操作系统上以 tim 用户登录成功后，就可以直接键入 sqlplus / 登录用户是 ops$tim，密码由操作系统外部提供，不是数据字典认证。

示例：
1)用户前缀缺省是 OPS$
sqlplus / as sysdba
SQL> show parameter os_authent_prefix
NAME TYPE VALUE
---------- ----------- ----------
os_authent_prefix string ops$

2) 建立 Oracle 的用户，密码是操作系统提供（外部）
SQL> create user ops$tim identified externally;
SQL> grant connect to ops$tim;
SQL> select USERNAME,PASSWORD from dba_users where username='OPS$TIM';
USERNAME PASSWORD
---------- ----------
OPS$TIM EXTERNAL
3) 建立操作系统用户
[root@cuug ~]# useradd tim
4）设置用户环境变量
[root@cuug ~]#su - tim
[tim@cuug ~]$ vi .bash_profile (添加环境变量)
ORACLE_BASE=/u01
ORACLE_HOME=$ORACLE_BASE/oracle
ORACLE_SID=prod
PATH=$ORACLE_HOME/bin:$PATH
export ORACLE_BASE ORACLE_HOME ORACLE_SID PATH
[tim@cuug ~]$ source .bash_profile
5）以操作系统身份登录
[tim@cuug ~]$ id
uid=501(tim) gid=502(tim) groups=502(tim)
[tim@cuug ~]$ sqlplus /
SQL*Plus: Release 11.2.0.1.0 Production on Sat Jul 2 22:25:12 2016
Copyright (c) 1982, 2009, Oracle. All rights reserved.
.....
SQL> show user;
USER is "OPS$TIM"
SQL>



9.3 空间配额的概念

资料整理
火龙果软件
荐

配额（quota)是表空间中为用户的对象使用的空间量，dba 建立用户时就应该考虑限制用户的磁盘空间配额，否则无限制配额的用户可能把你的表空间撑爆（甚至损坏 system 表空间）。
创建用户时，可以指定用户的缺省表空间及其配额，然后才可以建表。
单一的用户不需要临时表空间和 UNOD 表空间的配额
更改用户配额的一些命令：
ALTER USER tim QUOTA 10m ON test_tbs;　　给 tim 用户在 test_tbs 上使用了 10m 空间的配额
ALTER USER tim QUOTA unlimited on test_tbs　　--不受限制
ALTER USER tim QUOTA 0 ON test_tbs;　　收回剩余配额

9.4 概要文件的概念

9.4.1 **作用：**对用户访问数据库做一些限制。
1）概要文件（profile)具有两个功能，一个是 KERNEL 资源（如 CPU 资源）限制，另一个是 PASSWORD 资源（如登录）限制。
2）始终要实施口令控制，而对于 KERNEL 资源限制，则只有实例参数 RESOURE_LIMIT 为 TRUE 时（默认是 FALSE)才会实施。
3）系统自动使用概要文件，有一个默认的 default profile,限制很宽松，作用较小。
4）可以使用 create profile 为用户创建它自己的概要文件，没有指定值的参数，其值就从 default profile 的当前版本中提取。
9.4.2 **概要文件示例**
创建一个概要文件，对 tim 用户施加两个限制
1）如出现两次口令失误，将账户锁定。
2）tim 用户最多同时以两个 session 登录
步骤一、创建概要文件并命名 two_error
SQL> create profile two_error limit failed_login_attempts 2;
SQL> alter profile two_error limit Sessions_per_user 2;　　dba_profiles 视图可以列出所有 profile 资源;
SQL> select * from dba_profiles where PROFILE='TWO_ERROR';
步骤二、将概要文件分配给 tim 用户
SQL> alter user tim profile two_error;
步骤三、tim 用户尝试两次登录使用错误密码
SQL> conn tim/fdfd
ERROR: ORA-28000: 帐户已被锁定
步骤四、sys 为 tim 解锁
SQL> alter user tim account unlock;
步骤五、resource_limit=true 才可以使得 KERNEL 资源限制对 profile 起作用(考点)
SQL> alter system set resource_limit=true;
步骤六、测试 tim 只能同时有两个 session 登录
步骤七、sys 删掉了 two_error 概要文件
SQL> drop profile two_error cascade;
删除 two_error 后 tim 用户又绑定到 default profile 上。

9.5 权限和角色

9. 5. 1 **数据库的安全问题**
1）系统安全：
用户名、口令、概要文件、磁盘配额等
2）数据库安全：
对数据库系统和数据库对象的访问及操作
用户具备系统权限才能够访问数据库
具备对象权限才能访问数据库中的对象

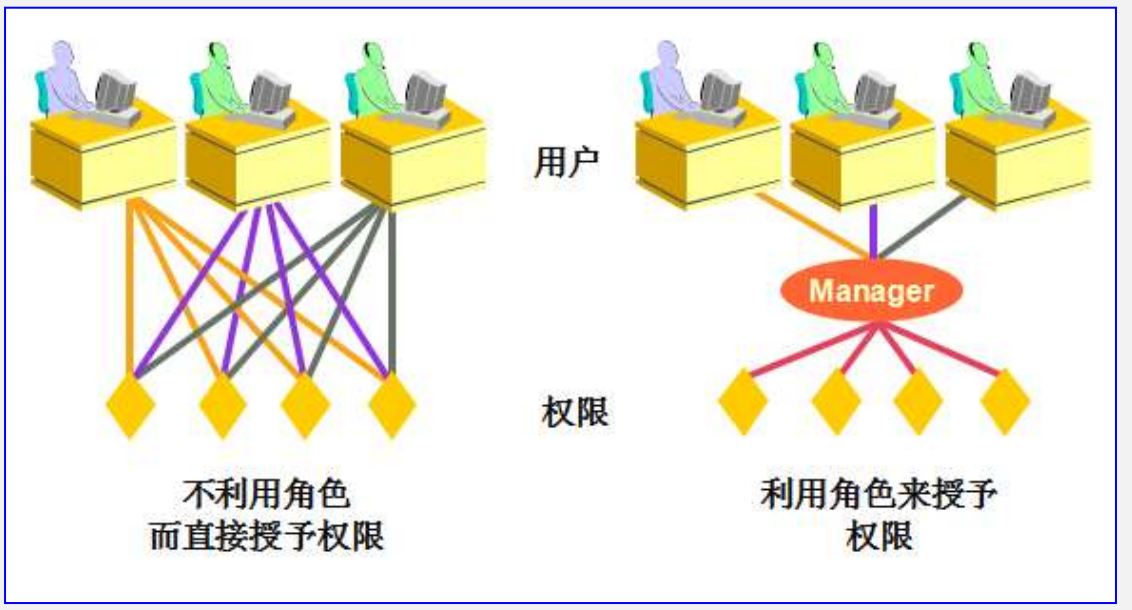
9.5.2 **系统权限**
针对于 database 的相关权限，通常由 DBA 授予（11g 已有 200 多种）
典型的 DBA 权限：
CREATE USER
DROP　USER
SELECT ANY TABLE
CREATE ANY TABLE
典型的一般用户的系统权限：
CREATE SESSION
CREATE TABLE
CREATE VIEW
CREATE PROCEDURE

9.5.3 **角色**

资料整理
火龙果软件
荐

1）为什么会有角色

系统权限太过繁杂，Oracle 建议将系统权限打包成角色，通过角色授权权限，目的就是为了简化用户访问管理



Oracle 有一些预定义角色，如：connect、resource、dba 角色等

也可以创建和删除自定义角色

SQL> CREATE role myrole;

SQL> DROP role myrole;

2）授予和回收权限的语法

1、授予系统权限和角色的语法

GRANT sys_privs,[role] TO user|role|PUBLIC [WITH ADMIN OPTION]

2、回收系统权限和角色的语法

Revoke sys_prvs[role] FROM user|role|PUBLIC

3）使用预定义角色的考虑

connect 和 resource 两个角色，可以满足一般用户大多数需求

SQL> grant connect, resource to tim;

SQL> revoke unlimited tablespace from tim;

SQL> alter user tim quota 10m on test_tbs;

查看用户表空间配额：

SQL> select tablespace_name,username,max_bytes from DBA_TS_QUOTAS where username='TIM';

当 unlimited tablespace 和 quota 共存时，以 unlimited 为准。

关于权限与空间配额的考虑

用户有空间配额才能建表。

空间配额不是权限的概念，而是用户属性的概念。

作为 DBA，在生产系统上一定要为普通用户分配空间配额，限制使用存储空间。

有两种办法分配空间配额：

第一种：在建立用户时指定空间配额，这个方法只能分配用户缺省的表空间上的配额，不是很灵活。

第二种：授予 resource 角色，包含有 unlimited tablespace 权限此权限对所有表空间不设限，包括系统表空间，请立即收回该权限，然后再分配空间配额。

系统权限

sys, system 拥有普通用户的所有对象权限，并有代理授权资格。

系统权限里的 any 含义：

SQL> conn / as sysdba;

SQL> grant create any table to tim;

tim 可以为 SCOTT 建表，这张表是属于 SCOTT 用户下的一个对象。

SQL> conn tim/tim

SQL> create table scott.t100 (id int);

9.5.4 对象权限

资料整理
火龙果软件
荐

• 对象权限	表	视图	序列	过程
• ALTER	Ö		Ö	
• DELETE	Ö	Ö		
• EXECUTE				Ö
• INDEX	Ö			
• INSERT	Ö	Ö		
• REFERENCES	Ö			
• SELECT	Ö	Ö	Ö	
• UPDATE	Ö	Ö		

1、授予对象权限的语法
GRANT object_privs ON object TO user|role|PUBLIC [WITH GRANT OPTION]
2、回收对象权限的语法
REVOKE object_prvs ON object FROM user|role|PUBLIC
角色授权的示例

9.5.5、授权注意事项
1）系统权限和对象权限语法格式不同，不能混合使用
SQL>grant create table,select on emp to tim　　错
2）系统权限和角色语法相同可以并列授权
SQL>grant connect,create table to tim;
3、可以使 update 对象权限精确到列
SQL> grant select, update(sal) on scott.emp to tim;
4）可以一条语句并列授予多个用户
SQL>grant connect to tim,ran;
5）可以通过授权建立用户,如 ran 用户不存在
SQL>grant connect,resource to ran identified by ran;

9.5.6 权限的传递与回收
系统权限的级联授予 WITH ADMIN OPTION
对象权限的级联授予 WITH GRANT OPTION
系统权限和对象权限的级联收回

sys 能分别从 scott 和 tim 收回系统权限，但仅从 scott 收回系统权限时并不能级联收回 tim 系统的权限。
scott 只能从 tim 收回对象权限，但同时会级联收回 ran 的对象权限。

9.5.7 对象权限在存储过程中的使用
存储过程 proc1 中包含了一些 tim 没有权限的 DML 操作，scott 将 proc1 的 execute 权限赋给 tim，，那么 tim 能成功地执行存储过程 proc1 吗？这个问题涉及到了 create procedure 时 invoker_rights_clause 的两个选项：
1、AUTHID CURRENT_USER
当执行存储过程时，检查用户 DML 操作的对象权限。
2、AUTHID DEFINER（默认）
当执行存储过程时，不检查用户 DML 操作的对象权限。
测试：默认情况下 AUTHID DEFINER
第一步

资料整理
火龙果软件
荐

SQL> create table scott.a (d1 date);
SQL> grant connect,resource to tim identified by tim;
SQL> conn scott/scott
第二步
create or replace procedure proc1 as
begin
insert into scott.a values(sysdate);
commit;
end;
第三步
SQL> grant execute on proc1 to tim;
第四步
SQL> conn tim/tim
SQL> exec scott.proc1;
测试 AUTHID CURRENT_USER 选项
第一步
SQL>conn scott/scott
SQL>create or replace procedure proc1
AUTHID CURRENT_USER as
begin
insert into scott.a values(sysdate);
commit;
end;
第二步，结果报错!
SQL> conn tim/tim
SQL> exec scott.proc1;
第三步，让 scott 在 execute 权限和 insert 权限都具备的情况下再执行上次操作
SQL> grant all on a to tim;
第四步，重复第二步，结果 OK!

9.5.8 有关权限的常用视图
1）用户查看自己当前拥有的系统权限（包括角色中的系统权限）
SESSION_PRIVS （常用）
2）使用数据字典查看权限的一般规律
dba_xxx_privs sys 用户查询用（常用）
all_xxx_privs
user_xxx_privs 普通用户查询用（常用）
3）使用数据字典查看角色的一般规律 普通用户查询用
role_xxx_privs
role_xxx_privs
role_xxx_privs
其中 xxx：role 表示角色，sys 表示系统权限，tab 表示对象权限

9.6 相关视图示例：
从哪个角度看，非常重要！三个用户，分别是 sys、scott 和 tim
第一步 sys 用户操作
1）建立 myrole 角色，把 connect 角色和 create table 系统权限以及 update on scott.emp 对象权限放进 myrole。
2）把 myrole 角色授给 tim。
3）把 create table 系统权限授给 tim。
第二步 scott 用户操作
把 select on emp 表的对象权限授给 tim
从 dba 角度看权限：
select * from dba_sys_privs where grantee='TIM'; 看用户 tim 所拥有的系统权限
select * from dba_tab_privs where grantee='TIM'; 看用户 tim 所拥有的对象权限
select * from dba_role_privs where grantee='TIM'; 看用户 tim 所拥有的角色
select * from dba_role_privs where grantee='MYROLE'; 看用 MYROLE 角色包含的角色
select * from dba_sys_privs where grantee='MYROLE'; 查看 MYROLE 角色里包含的系统权限
select * from dba_tab_privs where grantee='MYROLE'; 查看 MYROLE 角色里包含的对象权限
从 tim 用户角度看权限：

资料整理
火龙果软件
荐

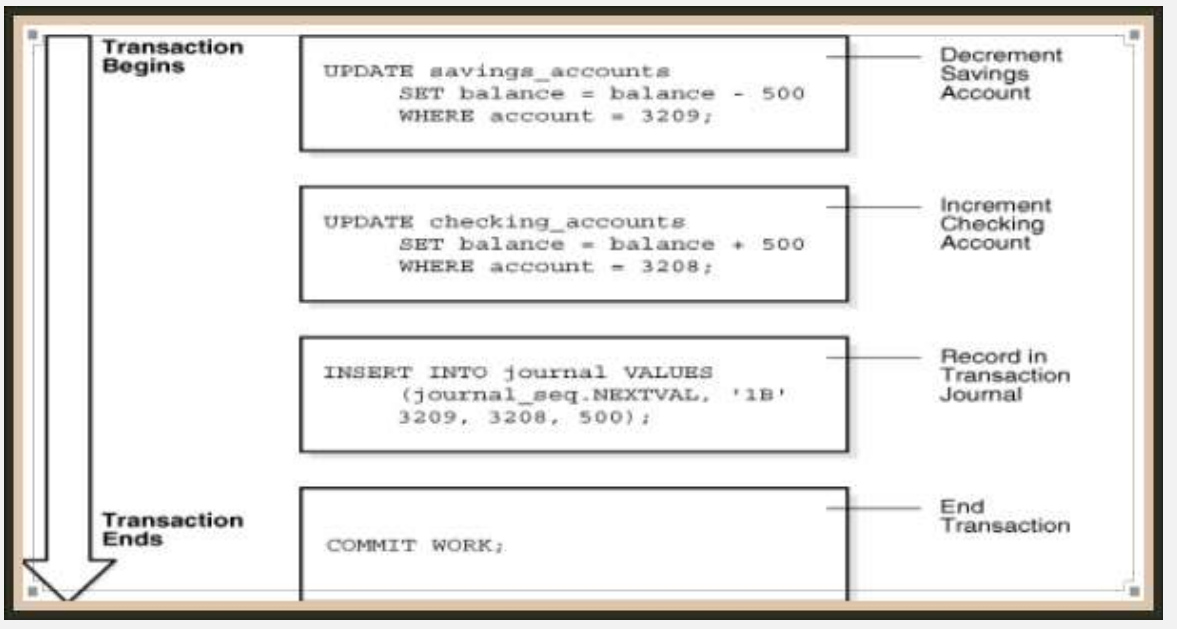
select * from user_role_privs;　查看和自己有关的角色（不含角色中含有的角色）
select * from user_sys_privs;　查看和自己有关的系统权限（不含角色中的系统权限）
select * from user_tab_privs;　查看和自己有关的对象权限（不含角色中的对象权限）
select * from role_role_privs;　角色里包含的角色
select * from role_sys_privs;　角色里包括的系统权限
select * from role_tab_privs;　角色里包括的对象权限
select * from session_privs;　查看和自己有关的系统权限(包括角色里的权限）
从 scott 角度看权限
select * from all_tab_privs where grantee='TIM';
select * from all_tab_privs where table_name='EMP';

第十章、事务和锁

10.1 事务具备的四个属性（简称 ACID 属性）：

1）**原子性**（Atomicity）：事务是一个完整的操作。事务的各步操作是不可分的（如原子不可分）；各步操作要么都执行了，要么都不执行。
2）**一致性**（Consistency）：事务执行的结果必须使数据库从一个一致的状态到另一个一致的状态。。
3）**隔离性**（Isolation）：事务的执行不干扰其他事务。一般来说数据库的隔离性都提供了不同程度的隔离级别。
4）**持久性**（Durability）：事务一旦提交完成后，数据库就不可以丢失这个事务的结果，数据库通过日志能够保持事务的持久性。

10.2 事务的开始和结束



10.2.1 事务采用隐性的方式，
1）起始于 session 的第一条 DML 语句
2）一个 session 某个时刻只能有一个事务
10.2.2 事务结束的方式：
1）TCL 语句被执行，COMMIT（提交）或 ROLLBACK（回滚）
2）DDL 语句被执行（提交）
3）DCL 语句被执行（提交）
4）用户退出 SQLPLUS（正常退出是提交，非正常退出是回滚）
5）服务器故障或系统崩溃（回滚）
6）shutdowm immediate(回滚）
在一个事务里如果某个 DML 语句失败，之前其他任何 DML 语句将保持完好，且不会提交！

10.3 Oracle 的事务保存点功能

savepoint 命令允许在事务进行中设置一个标记（保存点），回滚到这个标记可以保留该点之前的事务存在，并使事务继续执行。
实验:
savepoint sp1;
delete from emp1 where empno=7900;
savepoint sp2;
update emp1 set ename='timran' where empno=7788;
select * from emp1；
rollback to sp2;
select * from emp1；

资料整理 火龙果软件 荐

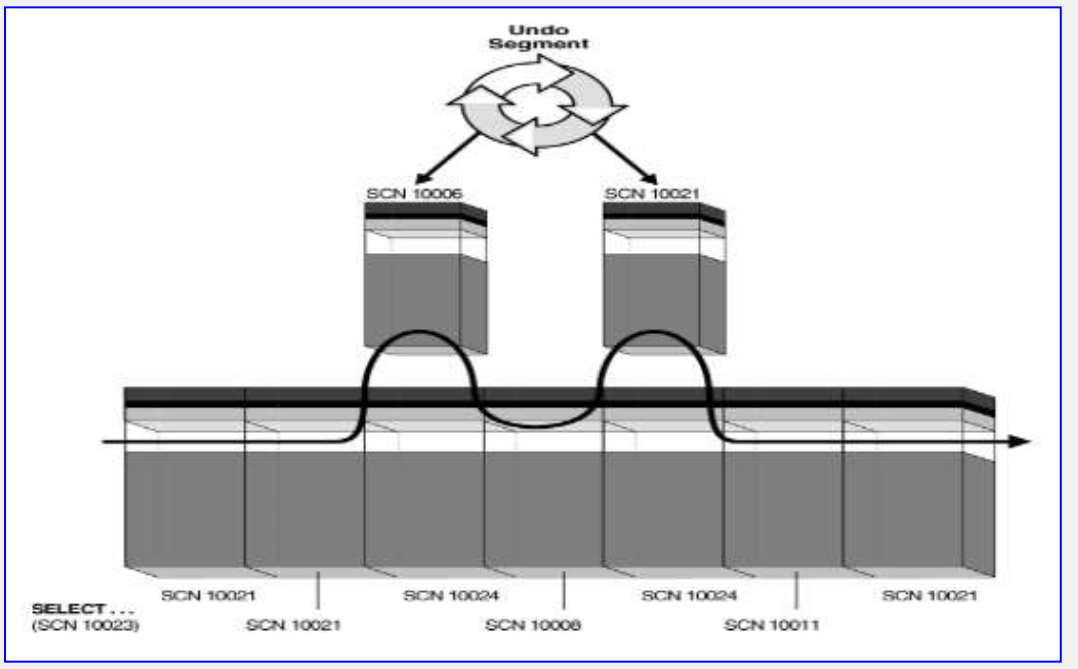
rollback to sp1;
注意 rollback to XXX 后，左侧的事务不会结束。

10.4 SCN 的概念

10.4.1 概念：

SCN 全称是 System Change Number，它是一个不断增长的整数，相当于 Oracle 内部的一个时钟，只要数据库一有变更，这个 SCN 就会增加，Oracle 通过 SCN 记录数据库里事务的一致性。SCN 涉及了实例恢复和介质恢复的核心概念，它几乎无处不在：控制文件，数据文件，日志文件都有 SCN，包括 block 上也有 SCN。

10.4.2 理解一致性读



实际上，我们所说的保证同一时间点一致性读的概念，其背后是物理层面的 block 读，Oracle 会依据你发出 select 命令，记录下那一刻的 SCN 值，然后以这个 SCN 值去同所读的每个 block 上的 SCN 比较，如果读到的块上的 SCN 大于 select 发出时记录的 SCN，则需要利用 Undo 得到该 block 的前镜像，在内存中构造 CR 块(Consistent Read)。

10.4.3 获得当前 SCN 的两个办法：

SQL> select current_scn from v$database;
SQL> select dbms_flashback.get_system_change_number from dual;

有两个函数可以实现 SCN 和 TIMESTAMP 之间的互转
scn_to_timestamp
timestamp_to_scn
SQL> select scn_to_timestamp(current_scn) from v$database;

10.5 共享锁与排他锁的基本原理：

10.5.1 基本原则

排他锁，排斥其他的排他锁和共享锁。
共享锁，排斥其他的排他锁，但不排斥其他的共享锁。

10.5.2 Oracle 锁的种类

因为有事务才有锁的概念。Oracle 数据库锁可以分为以下几大类：
DML 锁（data locks，数据锁），用于保护数据的完整性。
DDL 锁（dictionary locks，数据字典锁），用于保护数据库对象的结构，如表、索引等的结构定义。
SYSTEM 锁（internal locks and latches），保护数据库的内部结构。

我们探讨的是 Oracle 的 DML 操作（insert、update、delete），它包括两种锁：TX（行锁）和 TM（表锁）。
TX 是面向事务的行锁，它表示你锁定了表中的一行或若干行。update 和 delete 操作都会产生行锁，insert 操作除外。
TM 是面向对象的表锁，它表示你锁定了系统中的一个对象，在锁定期间不允许其他人对这个对象做 DDL 操作。目的就是为了实施 DDL 保护。（理解一下参数 ddl_lock_timeout）
比如一个 update 语句，有表级锁（即 TM)和行锁（即 TX 锁）。Oracle 是先申请表级锁 TM（其中的 RX 锁），获得后系统再自动申请行锁(TX)。并将实际锁定的数据行的锁标志置位。

对于 DML 操作
行锁(TX)只有一种
表锁(TM)共有五种，分别是 RS,RX,S,SRX,X。

资料整理
火龙果软件
荐

10.6 五种 TM 表锁的含义：

1）ROW SHARE 行共享(RS)，
允许其他用户同时更新其他行，允许其他用户同时加共享锁，不允许有独占（排他性质）的锁
2）ROW EXCLUSIVE 行排他(RX)，允许其他用户同时更新其他行，只允许其他用户同时加行共享锁或者行排他锁
3）SHARE 共享(S)，不允许其他用户同时更新任何行，只允许其他用户同时加共享锁或者行共享锁
4）SHARE ROW EXCLUSIVE(SRX) 共享行排他，不允许其他用户同时更新其他行，只允许其他用户同时加行共享锁
5）EXCLUSIVE (X)排他，其他用户禁止更新任何行，禁止其他用户同时加任何排他锁。

sql语句	许可其他用户的加锁
select * from table_name	RS,RX,S,SRX,X
insert, update, delete(DML操作)	RS,RX
select * from table_name for update	RS,RX

10.7 加锁模式

第一种方式：自动加锁
做 DML 操作时，如 insert，update，delete，以及 select....for update 由 oracle 自动完成加锁
Session1/scott:
SQL> select * from dept1 where deptno=30 for update;　　用 for update 加锁
Session2/sys:
SQL>select * from scott.dept1 for update;　　不试探，被锁住
SSession2/sys:
SQL>select * from scott.dept1 for update nowait;　　试探，以防被锁住
SQL>select * from scott.dept1 for update wait 5;
SQL> select * from scott.dept1 for update skip locked;　　跳过加锁的记录，锁定其他记录
1）对整个表 for update 是不锁 insert 语句的。
2）wait 5：等 5 秒自动退出。nowait：不等待。skip locked：跳过。都可起到防止自己被挂起的作用。
语法：lock table 表名 in exclusive mode.（一般限于后三种表锁）

10.8 死锁和解锁

Oracle 自动侦测死锁，自动解决锁争用。
制作死锁案例：
session1
update scott.emp1 set sal=8000 where empno=7369;
session2
update scott.emp1 set sal=9000 where empno=7934;
session1
update scott.emp1 set sal=8000 where empno=7934;
session2
update scott.emp1 set sal=9000 where empno=7369;
报错：ORA-00060: 等待资源时检测到死锁

10.9 管理员如何解鎖

可以根据以下方法准确定位要 kill session 的 sid 号和 serial#号，
SQL> select * from v$lock where type in ('TX','TM');
SQL> select a.sid,a.serial#,b.sql_text from v$session a,v$sql b where a.prev_sql_id=b.sql_id and a.sid=127;
```
       SID       SERIAL# SQL_TEXT
---------- ---------- ----------------------------------------------------------------------------
```

资料整理
火龙果软件
荐

```
          127          2449 update emp1 set sal=8000 where empno=7788
SQL> select sid,serial#,blocking_session,username,event from v$session where blocking_session_status='VALID';
          SID       SERIAL# BLOCKING_SESSION USERNAME                                EVENT
---------- ---------- ---------------- ------------------------------ ----------------------------------------
          127          2449                134       SCOTT                                enq: TX - row lock contention
```

也可以根据 v$lock 视图的 block 和 request 确定 session 阻塞关系,确定无误后再杀掉这个 session

```
SQL>ALTER SYSTEM KILL SESSION '127,2449';
```

更详细的信息，可以从多个视图得出，相关的视图有：v$session,v$process,vsql,vlocked,v$sqlarea 等等

阻塞(排队）从 EM 里看的更清楚 EM-->Performance-->Additional Monitoring Links-->Blocking Sessions(或 Instance Locks)

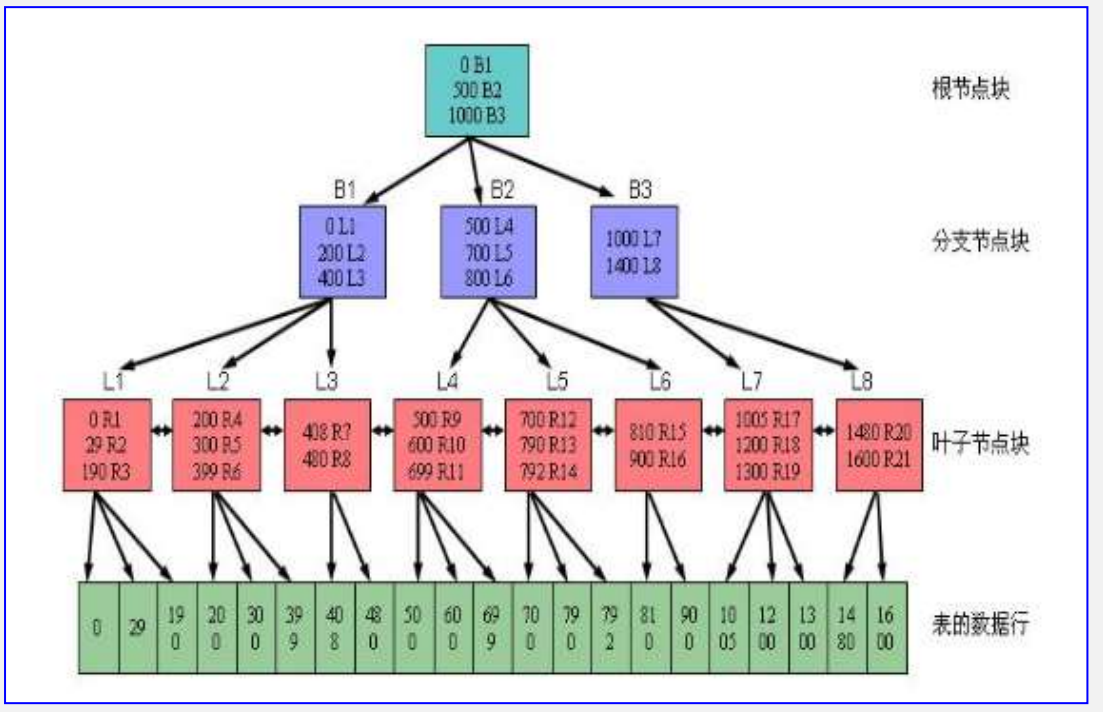
第十一章、索引

11.1 索引结构及特点

两大类：B 树索引，2）位图索引

11. 1. 1 B 树索引结构

根节点，分支节点，叶子节点，以及表行，rowid，键值，双向链等概念。



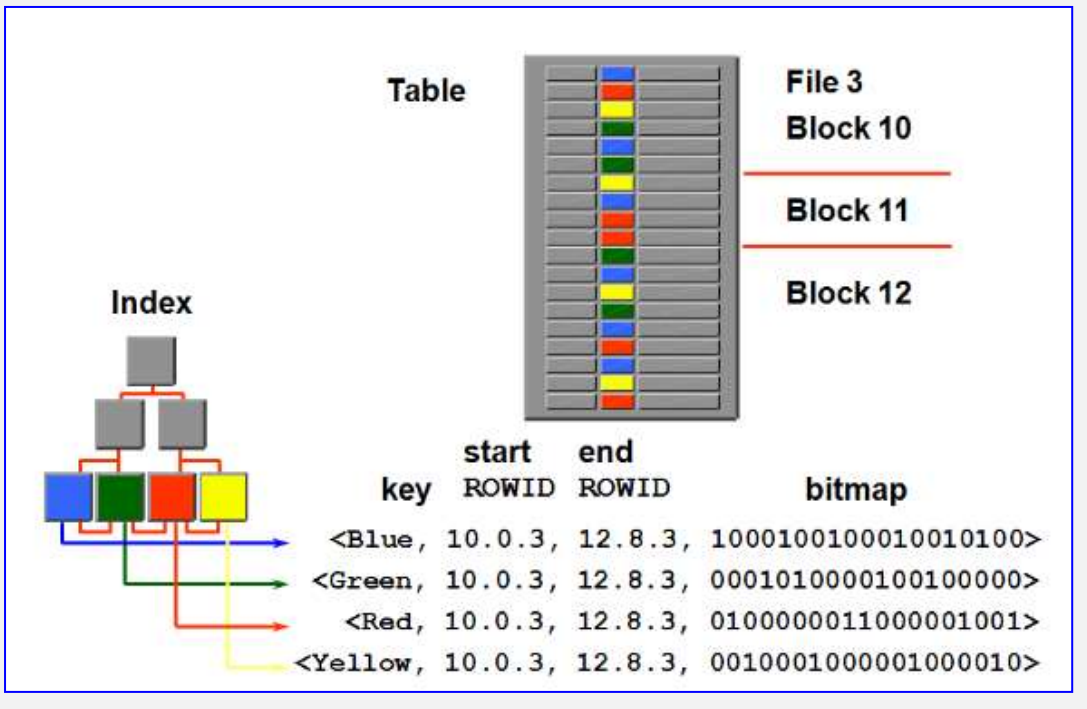
1）叶块之间使用双向链连接，

2）删除表行时索引叶块也会更新，但只是逻辑更改，并不做物理的删除叶块。

3）索引叶块中不保存表行键值的 null 信息。

11. 1. 2 位图索引结构

位图索引适用于离散度较低的列，它的叶块中存放 key, start rowid-end rowid,并应用一个函数把位图中相应 key 值置 1。



建立位图索引：

```
SQL>create bitmap index job_bitmap on emp1(job);
```

资料整理
火龙果软件
荐

11.1.3 查看约束的两个数据字典视图

select * from user_indexes;

select * user_ind_columns;

11.2 B 树索引和位图索引的适用环境

1）索引是与表相关的一个可选结构，在逻辑上和物理上都独立于表的数据，索引能优化查询，不能优化 DML 操作

2）由于是 Oracle 自动维护索引，所以频繁的 DML 操作反而会引起大量索引维护的开销。3）如果 SQL 语句仅访问被索引的列，那么数据库只需从索引中读取数据，而不用读取表，如果该语句同时还要访问除索引列之外的列，那么数据库会使用 rowid 来查找表中的行。

B树适合情况	位图适合情况

Scott 用户使用 autotrace 工具

SQL>conn / as sysdba

SQL>@$ORACLE_HOME/sqlplus/admin/plustrce.sql

SQL> grant plustrace to scott;

SQL> conn scott/scott

SQL> set autotrace on;

11.3 几种常用的 B 树索引创建方法：

1）唯一索引，指键值不重复。

SQL> create unique index empno_idx on emp1(empno);

2）非唯一索引（Unique or non_unique):

SQL> create index empno_idx on emp1(empno);

3）组合索引(Composite)：绑定了两个或更多列的索引。

SQL> create index job_deptno_idx on emp1(job,deptno);

4）反向键索引(Reverse)：将字节倒置后组织键值。当使用序列产生主键索引时，可以防止叶节点出现热块现象（考点）。缺点是无法提供索引范围扫描。

SQL> create index mgr_idx on emp1(mgr) reverse;

5）函数索引(Function base)：以索引列值的函数值为键值去组织索引

SQL> create index fun_idx on emp1(lower(ename));

6）压缩（Compress)：重复键值只存储一次，就是说重复的键值在叶块中就存一次，后跟所有与之匹配的 rowid 字符串。

SQL> create index comp_idx on emp1(sal) compress;

7）升序或降序（Ascending or descending)：叶节点中的键值排列默认是升序的。

SQL> create index deptno_job_idx on emp1(deptno desc, job asc);

11.4 索引的优化

11.4.1 查询优化器使用索引

1）**索引唯一扫描**(index unique scan)

通过唯一索引查找一个数值返回单个 ROWID。对于唯一组合索引，要在 where 的谓词“=”后包含所有列的“布尔与”。

2）**索引范围扫描**(index range scan)

在非唯一索引上，可能返回多行数据，所以在非唯一索引上都使用索引范围扫描。

使用 index rang scan 的 3 种情况：

(在唯一索引列上使用了 range 操作符(> < <> >= <= between)

在唯一组合索引上，对组合索引使用部分列进行查询(含引导列），导致查询出多行

对非唯一索引列上进行的任何查询。不含‘布尔或’

3）**索引全扫描**(index full scan)

对整个 index 进行扫描，并且顺序的读取其中数据。

CBO 根据统计数值得知进行全 Oracle 索引扫描比进行全表扫描更有效时，才进行全 Oracle 索引扫描，

4）**索引快速扫描**(index fast full scan)

扫描索引中的所有的数据块，fast full scan 在读取叶子块时的顺序完全由物理存储位置决定，并采取多块读，每次读取

资料整理 火龙果软件 荐

DB_FILE_MULTIBLOCK_READ_COUNT 个块。
CBO 能够索引全扫描和快速扫描的前提是：所要的数据必须能从索引中可以直接得到，因此不再需要查询基表。

11.5 索引的碎片问题

对基表做 DML 操作，便导致对索引表块的自动更改操作，尤其是基表的 delete 操作会引起 index 表的 index entries 的逻辑删除。
注意：只有当一个索引块中的全部 index entry 都被删除了，这个块才会被收回。
如果 update 基表索引列，则索引块会发生 entry delete，再 entry insert，这都些动作都可能产生索引碎片
测试：
第一步，搭建环境

```
SQL> create table t (id int);
SQL> create index ind_1 on t(id);
SQL>
begin
   for i in 1..1000000 loop
     insert into t values (i);
          if mod(i, 100)=0 then
          commit;
          end if;
   end loop;
end;
/
```

第二步，收集统计信息，并查看视图

```
SQL> analyze index ind_1 validate structure;
SQL> select name,HEIGHT,PCT_USED,DEL_LF_ROWS/LF_ROWS from index_stats;
```

第三步，基表高强度更新，造成索引碎块

```
SQL> delete t where rownum<700000;
```

第四步，重复第二步，查看视图

11.6 重建索引

查询索引的两个动态视图
DBA_INDEXES
DBA_IND_COLUMNS
在 Oracle 文档里并没有清晰的给出索引碎片的量化标准。
Oracle 建议通过 Segment Advisor(段顾问）解决表和索引的碎片问题，如果你想自行解决，可以通过查看 index_stats 视图，当以下三种情形之一发生时，说明积累的碎片应该整理了。
1.HEIGHT >=4
2 PCT_USED< 50% (相对值）
3 DEL_LF_ROWS/LF_ROWS>0.2

```
SQL> alter index ind_1 rebuild online;
```

11.7 索引不可用（unusable）和不可见（invisible）

1）仅仅保存索引定义，不删除索引，也不更新索引。

```
SQL> alter index ind_1 unusable;
```

索引被设定为 unusable 后，如再次使用需要做 rebuild。

```
SQL> alter index ind_1 rebuild;
```

2）在 11g 里，Oracle 提供了一个新特性(Index Invisible)来降低直接删除索引或禁用索引的风险。我们可以在创建索引时指定 invisible 属性或者用 alter 语句来修改索引为 invisible(visible)

```
SQL> alter index ind_1 invisible;
SQL> select index_name,status,VISIBILITY from user_indexes;
```

索引不可见其实是对优化器来说不可见，索引维护还是正常进行的。

第十二章、约束

12.1 什么是约束

1）约束的作用：
约束是数据库能够实施业务规则以及保证数据遵循实体--关系模型的一种手段。如果违反约束，将自动回滚出现问题的整个语句，而不是语句中的单个操作，也不是整个事务。
2）约束的语法：

资料整理
火龙果软件
荐

列级定义：只能引用一个列，表中可以有多个列级约束。
表级定义：引用一个或多个列，通常用来定义主键。
追加定义：建表后，再通过 alter table 命令追加的约束。
查看约束的两个数据字典视图
select * from user_constraints;
select * user_cons_columns;

12.2 五种约束的建立

12.2.1 非空约束

列级定义：
create table t1 (id number(2) not null, name varchar2(4))
追加非空约束：
如：alter table t1 modify ename not null;
或（alter table t1 modify ename constraint aaa not null）;
非空约束没有表级定义

12.2.2 唯一性约束

1）unique 不包括空值的唯一性
2）单列可设 unique+not null 约束，约束之间没有“，”
create table a (id int unique not null, name char(10));
3）单表中 unique 约束没有数量限制。
4）unique 约束的列上有索引。
列级定义
create table t1(id number(2) unique,name varchar2(4));
表级定义
create table t2(id number(2),name varchar2(4),constraint id_uk unique(id));
追加定义
alter table t2 add CONSTRAINT id_uk UNIQUE (id);

12.2.3 主键约束

主键约束语法上与唯一约束类似
1）每个表只能建立一个主键约束，primary key=unique key + not null，主键约束可以是一列，也可以是组合多列。
2）主键列上需要索引，如果该列没有索引会自动建立一个 unique index, 如果该列上已有索引（非唯一也可以），那么就借用这个索引，由于是借用的索引，当主键约束被删除后借用的索引不会被删除。同理，多列组合的主键，需要建立多列组合索引，而多列主键的单列上还可以另建单列索引。
3）主键约束和唯一约束不能同时建立在一个列上。
4）主机约束和外键约束可以在同一列上。（典型的例子是一个表上的两列是组合主键，但这两列的每列都是其他表的外键）
关于主键和索引关联的问题:（这个地方考点较多)

```
SQL>
create table t (id int, name char(10));
insert into t values (1, 'sohu');
insert into t values (2, 'sina');
commit;
SQL> create index t_idx on t(id);
```

下面这两句话是一样的效果，因为缺省情况下 id 列已经有索引 t_id 了，建主键时就会自动用这个索引（考点）。

```
SQL> alter table t add constraint pk_id primary key (id);
SQL> alter table t add constraint pk_id primary key (id) using index t_idx;
SQL> select CONSTRAINT_NAME,TABLE_NAME,INDEX_NAME    from user_constraints;
SQL> alter table t drop constraint pk_id;      删除了约束，索引还在，本来就是借用的索引。
SQL> select index_name from user_indexes;
INDEX_NAME
------------------------------
PK_EMP
PK_DEPT
T_IDX
SQL> drop table t purge;      t_idx 是和 t 表关联的，关键字 purge 使表和索引一并永久删除了。
```

也可以使用 using 子句在建表、建约束、建索引一条龙下来，当然 primary key 也会自动使用这个索引(考点）。删除该约束，索引还存在。

```
SQL> create table t (id int,name char(10),constraint pk_id primary key(id) using index (create index t_idx on t(id)));
```

资料整理
火龙果软件
荐

12.2.4.外键约束
作用：引用主键构成完整性约束
1）外键约束和 unique 约束都可以有空值。
2）外键需要参考主键约束，但也可以参考唯一键约束。
3）外键和主键一般分别在两个表中，但也可以同处在一个表中。
SQL> create table emp1 as select * from emp;
SQL> create table dept1 as select * from dept;
SQL> alter table dept1 add constraint pk_dept1 primary key(deptno);
SQL> ALTER TABLE emp1 ADD CONSTRAINT fk_emp1 FOREIGN KEY(deptno) REFERENCES dept1(deptno);
主外键构成完成性约束，有两个要点：
1）插入或更新有外键的记录时，必须参照主键，（子要依赖父）。
2）有外键表时，主键表的记录不能做 DML 删除，（父不能舍子），
如果一定要删除，需要在建立外键时加 on delete cascade 子句或 on delete set null 子句
测试 ON DELETE CASCADE 子句
报错：ORA-02292: integrity constraint (SCOTT.E_FK) violated - child record found
删除外键约束，使用 ON DELETE CASCADE 关键字重建外键,
SQL>alter table emp1 drop constraint fk_emp1;
SQL>ALTER TABLE emp1 ADD CONSTRAINT fk_emp1 FOREIGN KEY(deptno) REFERENCES dept1(deptno) ON DELETE CASCADE;
视图 delete_rule 列会显示 CASCADE,否则显示 NO ACTION(考点)
SQL>select constraint_name,constraint_type,status,delete_rule from user_constraints;
测试：
SQL>delete from dept1 where deptno=30
再查看 emp1 表的 deptno 已经没有 30 号部门了，如果再对 dept1 的操作进行 rollback，emp1 的子记录也随之 rollback
ON DELETE CASCADE 要慎用，父表中删除一行数据就可能引发子表中大量数据丢失。
为此，还有 on delete set null 子句，顾名思义是子表不会删除（丢失）记录，而是将外键的值填充 null。
如果 disable dept1 主键约束并使用级联 cascade 关键字，则 emp1 的外键也会 disable，若再次 enable dept1 主键，则 emp1 外键仍然保持 disable.
SQL> alter table dept1 disable constraints pk_dept1 cascade;
SQL> alter table dept1 enable constraints pk_dept1;
SQL>select constraint_name,constraint_type,status,delete_rule from user_constraints;
SQL> drop table dept1 purge;
报错：ORA-02449: 表中的唯一/主键被外键引用
SQL> drop table dept1 cascade constraint purge;
注意：这时外键约束也被删除了（考点）

12.2.5.CHECK 约束
1、作用：CHECK 约束是检查一个表达式，表达式的逻辑结果为”F“时，违反约束则语句失败。
表达式使用”( )“扩起来
涉及多列的表达式，只能使用表级定义描述。
2、语法:
列级定义
SQL> create table emp1 (empno int,sal int check (sal>500),comm int);
表级定义
SQL> create table emp2 (empno int,sal int,comm int,check(sal>500));
追加定义
SQL> alter table emp2 add constraint e_no_ck check (empno is not null);
验证
SQL> insert into emp2 values(null,1,1);
报错：ORA-02290: 违反检查约束条件 (SCOTT.E_NO_CK)
3、check 约束中的特殊情况
1）check 约束中的表达式中不能使用变量日期函数（考点）
SQL> alter table emp1 add constraint chk check(hiredate<sysdate);
报错：ORA-02436: 日期或系统变量在 CHECK 约束条件中指定错误
SQL> alter table emp1 add constraint chk check(hiredate<to_date('2000-01-01','yyyy-mm-dd'));　这句是可以的
2）级联约束
```
CREATE TABLE test2 (
    pk NUMBER PRIMARY KEY,
    fk NUMBER,
    col1 NUMBER,
```
资料整理
火龙果软件
荐

```
col2 NUMBER,
CONSTRAINT fk_constraint FOREIGN KEY (fk) REFERENCES test2,
CONSTRAINT ck1 CHECK (pk > 0 and col1 > 0),
CONSTRAINT ck2 CHECK (col2 > 0)
)
/
```

当删除列时，看看会发生什么？

SQL> ALTER TABLE test2 DROP (col1);　这句不能执行，在 constraint ck1 中使用了该列

如果一定要删除级联约束的列，带上 cascade constraints 才行

SQL> ALTER TABLE test2 DROP (col1) cascade constraints;

3）check 中可以使用 in 表达式，如(check deptno in(10,20))

12.3 约束的四种状态

enable validate　:无法插入违反约束的行，而且表中所有行都要符合约束

enable novalidate　:表中可以存在不合约束的状态，,但对新加入数据必须符合约束条件.

disable novalidate　:可以输入任何数据，表中或已存在不符合约束条件的数据.

disable validate　:不能对表进行插入/更新/删除等操作,相当于对整个表的 read only 设定.

（如是主键，会删除索引，当 enable 后，又建立了索引）

更改约束状态是一个数据字典更新，将对所有 session 有效。

1）测试 enable novalidate 这种组态的用法

常用于当在表中输入了一些测试数据后，而上线后并不想去清除这些违规数据，但想从此开始才执行约束。

假设已经建立了一个 emp1 表，也插入了数据，如果有一天想在 empno 上加入 primary key 但是之前有不符合（not null+unique)约束的，怎样才能既往不咎呢？

SQL>create table emp1 as select * from emp;　没有约束考过来

SQL>update emp1 set empno=7788 where empno=7369;　设置一个重值

SQL>alter table emp1 add constraint pk_emp1 primary key (empno);　因要检查主键唯一性，拒绝建立此约束。

alter table emp1 add constraint pk_emp1 primary key (empno) enable novalidate;　这句话也不行，原因是唯一索引在捣乱

SQL>create index empno_index on emp1(empno);　建一个普通索引不受 unquie 的限制

SQL>alter table emp1 add constraint pk_emp1 primary key (empno) enable novalidate;。

从此之后，这个列的 DML 操作还是要符合主键约束（not null+unique)。

2）将 disable novalidate，enable novalidate 和 enable validate 三种状态组合起来的用法：

这种组合，可以避免因有个别不符合条件的数据而导致大数据量的传输失败。

假设有 a 表是源数据表，其中有空值，b 表是 a 表的归档表，设有非空约束，现要将 a 表数据（远程）大批量的插入到 b 表（本地）。

alter table b modify constraint b_nn1 disable novalidate;　先使 B 表非空约束无效。

insert into b select * from a;　大批数据可以无约束插入，空值也插进 B 表里了。

alter table b modify constraint b_nn1 enable novalidate;　既往不咎，但若新输入数据必须符合要求。

update b set channel='NOT KNOWN'where channel is null;　将所有空值填充了，新老数据都符合要求了。

alter table b modify constraint b_nn1 enable validate;　最终是约束使能+验证生效，双管齐下。

12.4 延迟约束

12.4.1 作用：

在插入大批量数据时先不检查约束（节约时间），而当提交时集中在一起检查约束。注意：一旦有一条 DML 语句违反了约束，整个提交都将失败，全军覆没。

yyyyyyyyyy 分为可延迟（deferrable）可以通过查询 User_Constraints 视图获得当前所有关于约束的系统信息.

查看 user_constraints 中的两个字段

Deferrable　是否为延迟约束 值为:Deferrable 或 Not Deferrable(缺省）.

Deferred　是否采用延迟 值为:Immediate（缺省）或 Deferred.

这两个字段的关系是：

1）如果创建约束时没有指定 Deferrable 那么以后也无法使约束成为延迟约束（只有通过重建约束时再指定它是延迟约束）

2）一个约束只有被定义成 Deferrable，那么这个约束 session 级才可以在 deferred 和 immediate 两种状态间相互转换

12.4.2 延迟约束示例：

SQL> alter table emp1 add constraint chk_sal check(sal>500) deferrable;　建立可延迟约束

已将 chk_sal 约束设为 Deferrable 了，下面可有两种面向 session 的方案：

1）约束不延迟，插入数据立刻检查约束

SQL>set constraint chk_sal immediate;

2）约束延迟，提交时将整个事务一起检查约束

SQL>set constraint chk_sal deferred;

资料整理 火龙果软件 荐

使用 set constraint 切换只影响当前会话，
也可以在建立约束时一次性指定系统级(全局设定所有会话）的延迟约束
①SQL>alter table emp1 add constraint chk_sal check(sal>500) deferrable initially immediate;
②SQL>alter table emp1 add constraint chk_sal check(sal>500) deferrable initially deferred;

第十三章、视图

13.1 为什么使用视图

1）限制数据的存取：
用户只能看到基表的部分信息。方法：赋予用户访问视图对象的权限，而不是表的对象权限。
2）使得复杂的查询的书写变得容易：
对于多表连接等复杂语句的映射，或内联视图的使用。
3）提供数据的独立性：
基表的多个独立子集的映射

13.2 简单视图和复杂视图

1） 简单视图：
视图与基表的记录一对一，故而可以通过视图修改基表。
2） 复杂视图：
视图与基表的记录一对多，无法修改视图。

特性	简单视图	复杂视图
表的个数	一个	一个或多个
含函数	无	有
含组函数	无	有
含DISTINCT	无	有
DML操作	可以	不一定

13.2 语法

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
   [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY];
```

13.2.1 FORCE 作用：可以先建视图，后建基表
SQL>create force view view1 as select * from test1;
13.2.2 WITH CHECK OPTION 作用：对视图 where 子句进行约束，使视图结果集保持稳定。
SQL>create view view2 as select * from emp where deptno=10 with check option;
insert 不许插入非 10 号部门的记录
update 不许将 10 号部门修改为其他部门
13.2.3 WITH READ ONLY 作用：禁止对视图执行 DML 操作
SQL>create view view3 as select * from emp where deptno=10 with read only;
如果建立了视图 想查看其中的定义，可以访问如下视图 dba_views 中的 text 字段(long 型)；

```
declare
v_text    dba_views.text%type;
v_name    dba_views.view_name%type;
begin
select text, view_name into v_text,v_name FROM dba_views WHERE view_name='V1';
dbms_output.put_line(v_name||' define is :'||v_text);
end;
/
```

第十四章、同义词

14.1 作用

资料整理
火龙果软件
荐

从字面上理解就是别名的意思，和视图的功能类似。就是一种映射关系。

14.2 公有同义词

同义词通常是数据库对象的别名
公有同义词一般由 DBA 创建，使所有用户都可使用,
创建者需要 create public synonym 权限。
示例:
SQL>conn / as sysdba
SQL>create view v1 as select ename,sal,deptno from scott.emp where deptno=10;
SQL>create public synonym syn1 for v1;
SQL>grant select on syn1 to public;

14.3 私有同义词

一般是普通用户自己建立的同义词，创建者需要 create synonym 权限。
sys:
SQL> grant create synonym to scott;
scott:
SQL> create synonym abc for emp; scott 建立了一个私有同义词
SQL> select * from abc; scott 可以使用这个私有同义词了
SQL> grant select on abc to tim; 把访问同义词的对象权限给 tim
SQL> select * from scott.abc; tim 使用同义词时要加模式名前缀(考点)
查看同义词的视图：dba_synonyms
删除私有同义词：drop synonym 同义词名
删除公有同义词：drop public synonym 同义词名
SQL>select * from dba_synonyms where synonym_name='SYN1';

14.3 同义词的要点

1）私有同义词是模式对象，一般在自己的模式中使用，如其他模式使用则必须用模式名前缀限定。
2）公有同义词不是模式对象，不能用模式名做前缀。
3）私有和公有同义词同名时，如果指向不同的对象，私有同义词优先。
4）引用的同义词的对象（表或视图）被删除了，同义词仍然存在，这同视图类似，重新创建该对象名，下次访问同义词时自动编译。

第十五章、序列

15.1 序列的作用:

主要目的是为了自动生成主键值。
因为一个数据库序列被所有 session 共享使用。当多个 session 向一个表中插入数据时，序列保证主键值不会冲突，按照序列的要求自动增长并不会重复

15.2 序列的两个伪列：

Currval: 序列的当前值，反复引用时该值不变。
Nextval: 序列的下一个值，每次引用按步长自增。

15.3 用法及示例

15. 3. 1 示例 1：
CREATE SEQUENCE dept_deptno 序列名
INCREMENT BY 10 步长 10
START WITH 5 起点 5
MAXVALUE 100 最大 100
CYCLE 循环
NOCACHE 不缓存

第一次要引用一下 nextval 伪列
SQL>select dept_deptno.nextval from dual;

资料整理
火龙果软件
荐

以后就有 currval 伪列值了。
SQL>select dept_deptno.nextval from dual;

15.3.2 示例 2：
主键使用序列
Session1/scott:
SQL> conn scott/scott
SQL> create table t1 (id int primary key,name char(8));
SQL> create sequence seq1;

SQL> grant all on t1 to tim;
SQL> grant all on seq1 to tim;

SQL> insert into t1 values(seq1.nextval,'a');
SQL> insert into t1 values(seq1.nextval,'b');
SQL> select * from t1;
```
        ID NAME
---------- --------
         1 a
         2 b
```

Session2/tim:
SQL> conn tim/tim
SQL> insert into scott.t1 values(scott.seq1.nextval,'c');
SQL> select * from scott.t1;
```
        ID NAME
---------- --------
         3 c
```
两个 session 由于使用序列发号而不会造成主键冲突，当它们都提交后，看到的数据是一样的。
SQL> drop sequence seq1;　　删除序列不会影响之前的引用

15.3.3 几点说明：
1）建立一个最简单序列只需要 create sequence 序列名，其他使用缺省值，注意：起始是 1，步长也是 1。
2）如果启用选项 cache，缺省只有 20 个号，经验表明这个数量会不够，可以设置多一些，根据需要 10000 个也可以。
3）选项 cycle 其实没有什么意义，因为它使序列发出重复值，这对于基于序列是主键值的用法是个问题。
4）可以使用 alter 命令修改序列配置。但 alter 命令不能修改起始值。如果要重启该序列，唯一的办法是删除并重新创建它。
5）循环后初始是从 1 开始的，不管原来的值是如何设的（考点）
6）currval 的值显示的是当前 session 的最新 nextval
7）建表时 DEFALT(缺省值)不能使用 nextval 定义（考点）
8）删除序列的命令是 drop sequence 序列名

第三部分、SQL 语言的扩展

第十六章、DML 语句总结

16.1 Insert 语句的基本类型

1）单表一次插入一行，有 values 关键字。
2）单表一次插入多行，使用子查询，没有 values 关键字。
3）多表插入，又分 Insert all 和 Insert first 两种形式。

16, 2 单表单行插入
1）SQL> create table a (id int,name char(10) default 'aaa');　　name 列指定了 default 值
2）SQL> insert into a values(1,'abc');　　表 a 后没有所选列，values 必须指定所有字段的值。
3）SQL> insert into a values(2,default);　　同上，name 字段用 default 占位。
4）SQL> insert into a values(3,null);　　表 a 后没有所选列，name 字段用 null 占位。
5）SQL> insert into a (id) values(4);　　未选定的字段如果指定了 default,则以 default 的值代替 null
6）SQL> insert into (select id from a) values (5);　　这种形式本质同上，只不过表 a 的形式以结果集代之。
7）SQL> insert into a values(6,(select dname from dept where deptno=10));　　values 里的某列使用 subquery 引用另一个表的数据
注意几点：
1）不符合约束条件的 insert 不能成功。

资料整理
火龙果软件
荐

2）default 不但可以用于 insert 语句，也可以用于 update 语句（考点）
3）values 后面不可以跟多列子查（考点）。
SQL> insert into a values(select deptno,dname from dept where deptno=10);
报错：ORA-00936: 缺失表达式
更正如下：
SQL> insert into a values((select deptno from dept where deptno=10), (select dname from dept where deptno=10));
SQL> commit;
8）insert WITH CHECK OPTION 的用法
SQL> insert into (select id from a where id<100 WITH CHECK OPTION) values (20);
SQL> rollback;
SQL> insert into (select id from a where id<100 WITH CHECK OPTION) values (101);
报错: ORA-01402: view WITH CHECK OPTION where-clause violation
看看这句话的另一种情况：
SQL> insert into (select name from a where id<100 WITH CHECK OPTION) values ('NBA');
报错：ORA-01402: view WITH CHECK OPTION where-clause violation
上例是想说明如果插入的列不在 where 条件里，则不允许插入。
16. 3 单表一次插入多行
语法：主句去掉了 values 选项。使用 select 子查询，
SQL> create table b as select * from a where 1>2;　建立一个空表 b。结构来自 a 表
SQL> insert into b select * from a where name='aaa';　插入的是结果集，注意没有 values 选项。
SQL> insert into b(id) select id from a where id in(1,3);　使用子查询（结果集）插入，对位
16. 4 多表插入
16. 4. 1 作用：
一条 INSERT 语句可以完成向多张表的插入任务（Multitable insert）。有两种形式：insert all 与 insert first，
准备测试环境：
1.创建表 T 并初始化测试数据，此表作为数据源。
create table t (x number(10), y varchar2(10));
insert into t values (1,'a');
insert into t values (2,'b');
insert into t values (3,'c');
insert into t values (4,'d');
insert into t values (5,'e');
insert into t values (6,'f');
commit;

2.创建表 T1 和 T2，作为我们要插入数据的目标表。
SQL>create table t1 as select * from t where 0=1;
SQL>create table t2 as select * from t where 0=1;

16. 4. 1 第一种多表插入方法 INSERT ALL
SQL>insert all into t1 into t2 select * from t;　（无条件 insert all)
12 rows created.
这里之所以显示插入了 12 条数据，实际上表示在 T1 表中插入了 6 条，T2 表插入了 6 条，一共是 12 条数据。（不分先后，各插各的）
SQL>rollback;
SQL> insert all when x>=3 then into t1 when x>=2 then into t2 select * from t;　（有条件 insert all)。

16. 4. 2 第二种多表插入方法 INSERT FIRST
1）清空表 T1 和 T2
SQL> truncate table t1;
SQL> truncate table t2;
2）完成 INSERT FIRST 插入
SQL> insert first when x>=3 then into t1 when x>=2 then into t2 select * from t;　(有条件 insert first)
处理逻辑是这样的，首先检索 T 表查找 X 列值大于等于 3 的数据插入到 T1 表，然后将前一个查询中出现的数据排除后再查找 T 表，找到 X 列值大于等于 2 的数据再插入到 T2 表，注意 INSERT FIRST 的真正目的是将同样的数据只插入一次。
3）验证 T1 和 T2 表中被插入的数据。
SQL> select * from t1;
SQL> select * from t2;

资料整理
火龙果软件
荐

第十七章、其他 DML 和 DDL 语句的用法

17.1 DML 语句-MERGE

根据一个表的数据组织另一个表的数据，一般是对 merge 的目标表插入新数据或替换掉老数据。
Oracle 10g 中 MERGE 有如下一些改进：
1、UPDATE 或 INSERT 子句是可选的
2、UPDATE 和 INSERT 子句可以加 WHERE 子句
3、ON 条件使用常量过滤谓词来 insert 所有的行到目标表中,不需要连接源表和目标表
4、UPDATE 子句后面可以跟 DELETE 子句来去除一些不需要的行

示例：首先创建表：

```
SQL>create table PRODUCTS
      (
      PRODUCT_ID INTEGER,
      PRODUCT_NAME VARCHAR2(30),
      CATEGORY VARCHAR2(30)
      );

insert into PRODUCTS values (1501, 'VIVITAR 35MM', 'ELECTRNCS');
insert into PRODUCTS values (1502, 'OLYMPUS IS50', 'ELECTRNCS');
insert into PRODUCTS values (1600, 'PLAY GYM', 'TOYS');
insert into PRODUCTS values (1601, 'LAMAZE', 'TOYS');
insert into PRODUCTS values (1666, 'HARRY POTTER', 'DVD');
commit;

SQL>create table NEWPRODUCTS
      (
      PRODUCT_ID INTEGER,
      PRODUCT_NAME VARCHAR2(30),
      CATEGORY VARCHAR2(30)
      );

insert into NEWPRODUCTS values (1502, 'OLYMPUS CAMERA', 'ELECTRNCS');
insert into NEWPRODUCTS values (1601, 'LAMAZE', 'TOYS');
insert into NEWPRODUCTS values (1666, 'HARRY POTTER', 'TOYS');
insert into NEWPRODUCTS values (1700, 'WAIT INTERFACE', 'BOOKS');
commit;

SQL>select * from products;
PRODUCT_ID PRODUCT_NAME                    CATEGORY
---------- ------------------------------ ------------------------------
      1501 VIVITAR 35MM                   ELECTRNCS
      1502 OLYMPUS IS50                   ELECTRNCS
      1600 PLAY GYM                       TOYS
      1601 LAMAZE                             TOYS
      1666 HARRY POTTER                   DVD
SQL> select * from newproducts;
PRODUCT_ID PRODUCT_NAME                    CATEGORY
---------- ------------------------------ ------------------------------
      1502 OLYMPUS CAMERA                  ELECTRNCS
      1601 LAMAZE                             TOYS
      1666 HARRY POTTER                   TOYS
      1700 WAIT INTERFACE                 BOOKS
```

下面我们从表 NEWPRODUCTS 中合并行到表 PRODUCTS 中，但删除 category 为 ELECTRNCS 的行.

```
SQL>MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
```

资料整理
火龙果软件
荐

```
UPDATE
SET p.product_name = np.product_name,p.category = np.category
DELETE WHERE (p.category = 'ELECTRNCS')
WHEN NOT MATCHED THEN
INSERT
VALUES (np.product_id, np.product_name, np.category);

SQL>select * from products;
PRODUCT_ID PRODUCT_NAME                    CATEGORY
---------- ------------------------------ ------------------------------
      1501 VIVITAR 35MM                   ELECTRNCS
      1600 PLAY GYM                       TOYS
      1601 LAMAZE                         TOYS
      1666 HARRY POTTER                   TOYS
      1700 WAIT INTERFACE                 BOOKS
```

为什么 1502 不在了，但 1501 还在？ 因为 1502 是 matched,先被 update,然后被 delete, 而 1501 是 not matched.

注意几点：

1）例子里有 update,delete 和 insert。它们是否操作是取决于 on 子句的，两个表如果符合 on 条件就是匹配，不符合就是不匹配。

2）匹配了就更新，不匹配则插入。10g 后加入了 delete 语句，这个语句必须在匹配条件下出现。它是一种补充。

3）你必须对操作的表有对象权限

4）ON 子句里的字段不能被 update 子句更新

17.2 WITH 语句

可以使用一个关键字 WITH， 为一个子查询块（subquery block）起一个别名。然后在后面的查询中引用该子查询块的别名。

好处：

1）使用 with 语句，可以避免在 select 语句中重复书写相同的语句块。

2）with 语句将该子句中的语句块执行一次并存储到用户的临时表空间中。

3）使用 with 语句可以避免重复解析，提高查询效率。

示例：这个 with 语句完成三个动作

建立一个 dept_costs，保存每个部门的工资总和，

建立一个 avg_cost，根据 dept_costs 求出所有部门总工资的平均值，

最后显示出部门总工资值小于部门总工资平均值的那些部门的信息(dname)。

```
WITH
dept_costs AS (
SELECT d.dname, SUM(e.sal) AS dept_total
FROM emp e, dept d
WHERE e.deptno = d.deptno
GROUP BY d.dname ),
avg_cost AS (SELECT SUM(dept_total)/COUNT(*) AS dept_avg FROM dept_costs)
SELECT * FROM dept_costs
WHERE dept_total <
(SELECT dept_avg FROM avg_cost)
ORDER BY dname
/
DNAME          DEPT_TOTAL
-------------- ----------
ACCOUNTING           8750
SALES                 9400、
```

可以分三个部分来看:

第一 AS 建立 dept_costs，保存每个部门的工资总和。

第二个 AS 建立 avg_cost，根据第一个 AS dept_costs 求出所有部门总工资的平均值（两个 with 子程序用逗号分开，第二个使用了第一个别名）。

最后是查询主体，SELECT * FROM... 调用了前两个 with 下的别名（子查询），显示部门总工资值小于部门总工资平均值的那些部门的信息。

1）with 语句中只能有 select 子句，没有 DML 子句（注意和 merge 的区别）。

资料整理 火龙果软件 荐

2）一般将主查询放在最后描述，因为查询主体中要引用的 with 别名需要在之前定义过。

17.3 表的 DDL 操作

1）在数据库打开的情况下，可以使用 DDL 语句修改数据字典信息，主要 DDL 语句对列的操作有：
增加（add）一列，
修改（midify)列的宽度,
删除（drop colunm)一列,
更名（rename column）列名
2）当想要 add 一列，并约束该列为 not null 时，如果该表已经有数据了，加的列本身是 null,则与 not null 约束矛盾，报错 。

```
SQL> select * from a;
            ID NAME
---------- ----------
             1 a
             2 b

SQL> alter table a add C number(5) not null;
```

报错：ORA-01758: 要添加必需的 (NOT NULL) 列，则表必须为空

```
SQL> alter table a add C number(5) default 0 not null;
```

修改成功了，可以看到 C 列全是 0，这样才能使 C 列的约束为 not null （考点）
3)要删除某一个表格上的某个字段,但是由于这个表格拥有非常大量的资料,如果你在尖峰时间直接执行 ALTER TABLE ABC DROP（COLUMN）；可能会收到 ORA-01562 -
failed to extend rollback segment number string
Oracle 推荐：使用 SET UNUSED 选项标记一列（或多列），使该列不可用。
然后，当业务量下降后再使用 DROP UNUSED column 选项删除被被标记为不可用的列。SET UNUSED COLUMNS 用于 drop 多列时效率更高，
SET UNUSED COLUMNS 方法系统开销比较小，速度较快，但效果等同于直接 drop column。就是说这两种方法都不可逆，无法再还原该字段及其内容了。
语法：
ALTER TABLE table SET UNUSED [column] (COLlist 多个)
ALTER TABLE table DROP UNUSED [COLUMN];
查看 unused 后的视图
select * from user_unused_col_tabs;
1）如果 set unused 某列，该列上有索引，约束，并定义了视图，引用过序列，结果如何，索引和约束自动删除，序列无关，视图保留定义。
2）无法删除属于 SYS 的表中的列，会报 ORA-12988 错误，哪怕你是 sys 用户都不可以。
实验：scott 下

```
SQL>create table a (id int, name char(10));
SQL>create index id_idx on a(id);
SQL>alter table a add constraint unq_id unique(id);
SQL>create sequence a_id start with 1 increment by 1;
SQL>create view v as select id from a;
SQL>insert into a values(a_id.nextval,'tim');
SQL>insert into a values(a_id.nextval,'ran');
SQL>commit;

SQL> select * from a;
SQL> select index_name from user_indexes where table_name='A';      查看有关索引
SQL> select constraint_name from user_constraints where table_name='A';      查看有关约束
SQL> select object_name from user_objects where object_type='SEQUENCE';      查看有关序列
SQL> select text from user_views where view_name='V';      查看有关视图
SQL> alter table a drop column id;
```

重复查看有关信息，索引和约束随该列数据虽然被删除，但序列和视图的定义还在。

17.4 模式及名称空间

模式(Schema)是一种逻辑结构，它对应于用户，每建一个用户就有一套模式与之对应。
我们通常说对象的唯一标识符是前缀为模式名加上对象名称，如 scott.emp。
同一模式下的同类对象是不可以重名的。比如在 scott 模式里，表 emp 是唯一的，但在不同模式下可以重名。
名称空间定义了一组对象类型，同一个名称空间里的不同对象不能同名，而不同的名称空间中的不同对象可以共享相同的名称。
1）表，视图，序列，同义词是不同类型的对象，但它们属于同一名称空间，因此在同一模式下也是不可以重名的，比如 scott 下不可以让一个表名和一个视图名同名。

资料整理
火龙果软件
荐

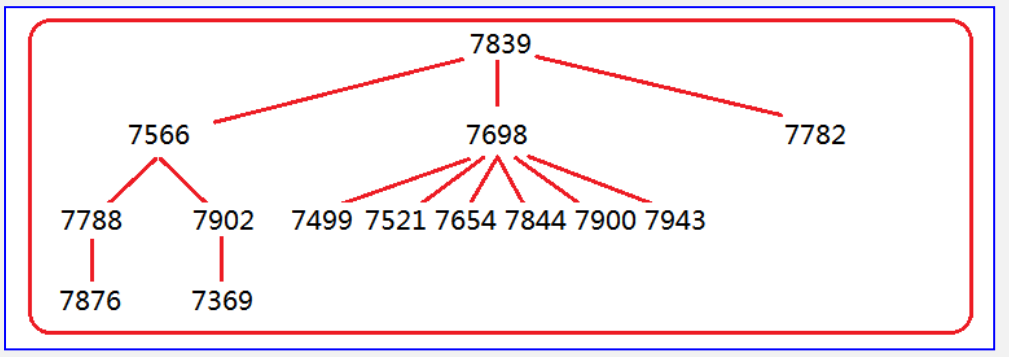
2）索引、约束有自己的名称空间，所以在 scott 模式下，可以有表 A,索引 A 和约束 A 共存
3）表名最长可以 30 个字符，表名中若包含特殊字符需要使用“ ”将表名全部包括在内，表的最大列数为 1000。

第十八章、ORACLE 分层查询

18.1 树的遍历

分层查询是 select 语句的扩展，目的是迅速找出表中父列--子列的隶属关系。
首先我们应该熟悉一下树的遍历方法
ORACLE 是一个关系数据库管理系统，在某些表中的数据还呈现出树型结构的联系。例如，在 EMP 表中含有雇员编号（EMPNO）和经理（MGR）两列，这两列反映出来的就是雇员之间领导和被领导的关系。这种关系就是一种树结构。

图 1.1 EMP 表树结构图



树的遍历有两个方向
top--down 自上而下
即父亲找儿子，一个父亲可能有几个儿子，一个儿子可能有几个孙子，遍历不能丢了儿子，顺序以左为先。
down--top 自底向上
即儿子找父亲，一个儿子只能有一个父亲，所以顺序应该是：孙子->儿子-->父亲-->爷爷。

18.2 分层查询语法

在 SELECT 命令中使用 CONNECT BY 和 START WITH 子句可以查询表中的树型结构关系。其命令格式如下：
SELECT ...
CONNECT BY {PRIOR 列名 1=列名 2|列名 1=PRIOR 列名 2}
[START WITH]；

18.3.1 CONNECT BY 子句

理解 CONNECT BY PRIOR 子句至关重要，它确定了树的检索方向：是 top --> down（父-->子）还是 down --> top（子-->父）。
在分层表中，表的父列与子列是确定的（身份固定），如：在 emp 表中 empno 是子列（下级）， mgr 是父列（上级）。

PRIOR 关键字就像一个箭头("-->")，
①connect by prior empno = mgr
②connect by mgr = prior empno
两句语法等同，都是说 mgr（父）--> empno(子），因此树的检索方向是 top --> down。
①connect by empno = prior mgr
②connect by prior mgr = empno
两句语法等同，都是说 empno(子）--> mgr(父），因此树的检索方向是 down --> top。

18.3.2 START WITH

子句为可选项，用来标识哪个节点作为查找树型结构的根节点。若该子句被省略，则表示所有满足查询条件的行作为根节点（每一行都会成为一个树根）。
例 1 以树结构方式显示 EMP 表的数据。

```
SQL>select empno,ename,mgr from emp
connect by prior empno=mgr
start with empno=7839
/
     EMPNO ENAME          MGR
---------- ---------- ----------
      7839 KING
      7566 JONES          7839
      7788 SCOTT          7566
```

资料整理 火龙果软件 荐

```
      7876 ADAMS          7788
      7902 FORD           7566
      7369 SMITH          7902
      7698 BLAKE          7839
      7499 ALLEN          7698
      7521 WARD           7698
      7654 MARTIN         7698
      7844 TURNER         7698
      7900 JAMES          7698
      7782 CLARK          7839
      7934 MILLER         7782
```

仔细看 empno 这一列输出的顺序，就是上图树状结构每一条分支（从根节点开始）的结构。

例 2 从 SMITH 节点开始自底向上查找 EMP 的树结构。

```
SQL>select empno,ename,mgr
from emp
connect by empno=prior mgr
start with empno=7369
/
     EMPNO ENAME          MGR
---------- ---------- ----------
      7369 SMITH          7902
      7902 FORD           7566
      7566 JONES          7839
      7839 KING
```

在这种自底向上的查找过程中，只有树中的一枝被显示。

18.3 定义起始节点

在自顶向下查询树结构时，不但可以从根节点开始，还可以定义任何节点为起始节点，以此开始向下查找。这样查找的结果就是以该节点为开始的结构树的一枝。

例 3 查找 7566(JONES)直接或间接领导的所有雇员信息。

```
SQL＞SELECT EMPNO,ENAME,MGR
FROM EMP
CONNECT BY PRIOR EMPNO=MGR
START WITH EMPNO=7566
/
     EMPNO ENAME          MGR
---------- ---------- ----------
      7566 JONES          7839
      7788 SCOTT          7566
      7876 ADAMS          7788
      7902 FORD           7566
      7369 SMITH          7902
```

START WITH 不但可以指定一个根节点，还可以指定多个根节点。

例 4 查找由 FORD 和 BLAKE 领导的所有雇员的信息。

```
SQL>SELECT EMPNO,ENAME,MGR
FROM EMP
CONNECT BY PRIOR EMPNO=MGR
START WITH ENAME IN ('FORD','BLAKE')
/
     EMPNO ENAME          MGR
---------- ---------- ----------
      7902 FORD           7566
      7369 SMITH          7902
      7698 BLAKE          7839
      7499 ALLEN          7698
      7521 WARD           7698
```

资料整理
火龙果软件
荐

```
      7654      MARTIN              7698
      7844      TURNER              7698
      7900      JAMES               7698
8 rows selected.
```

18.4 使用 LEVEL 伪列

在查询中，可以使用伪列 LEVEL 显示每行数据的有关层次。LEVEL 将返回树型结构中当前节点的层次。伪列 LEVEL 为数值型，可以在 SELECT 命令中用于各种计算。

例 5

1）使用 LEVEL 伪列显示所有员工隶属关系。

```
SQL> COLUMN LEVEL FORMAT A20
SQL> SELECT LPAD(LEVEL,LEVEL*3,' ')
as "LEVEL",EMPNO,ENAME,MGR
FROM EMP
CONNECT BY PRIOR EMPNO=MGR
START WITH ENAME='KING'
/
LEVEL                    EMPNO ENAME                MGR
-------------------- ---------- ---------- ----------
  1                        7839 KING
     2                     7566 JONES            7839
        3                  7788 SCOTT            7566
           4               7876 ADAMS            7788
        3                  7902 FORD             7566
           4               7369 SMITH            7902
     2                     7698 BLAKE            7839
        3                  7499 ALLEN            7698
        3                  7521 WARD             7698
        3                  7654 MARTIN           7698
        3                  7844 TURNER           7698
        3                  7900 JAMES            7698
     2                     7782 CLARK            7839
        3                  7934 MILLER           7782
14 rows selected.
```

在 SELECT 使用了函数 LPAD，该函数表示以 LEVEL*3 个空格进行填充，由于不同行处于不同的节点位置，具有不同的 LEVEL 值，因此填充的空格数将根据各自的层号确定，空格再与层号拼接，结果显示出这种缩进的层次关系。

2）只查看第 2 层的员工信息：

```
SQL> select t1.* from (select level LNUM ,ename,mgr from emp connect by prior empno=mgr start with ename='KING') t1 where LNUM=2;
      LNUM ENAME                MGR
---------- ---------- ----------
         2      JONES                7839
         2      BLAKE                7839
         2      CLARK                7839
SQL> select lnum,avg(sal) avgsal from (select level LNUM ,ename,mgr,sal from emp connect by prior empno=mgr start with ename='KING') group by lnum having lnum=2;
      LNUM        AVGSAL
---------- ----------
         2       2758.33333
```

18.5 节点和分支的裁剪

在对树结构进行查询时，可以去掉表中的某些行，也可以剪掉树中的一个分支，使用 WHERE 子句来限定树型结构中的单个节点，以去掉树中的单个节点，但它却不影响其后代节点

（自顶向下检索时）或前辈节点（自底向顶检索时）。

```
SQL>SELECT LPAD(LEVEL,LEVEL*3,' ')
as "LEVEL",EMPNO,ENAME,MGR
FROM EMP
WHERE ENAME<>'SCOTT'
```

资料整理 火龙果软件 荐

```
CONNECT BY PRIOR EMPNO=MGR
START WITH ENAME='KING'
/
LEVEL                                EMPNO ENAME                MGR
-------------------- ---------- ---------- ----------
   1                                  7839      KING
       2                              7566      JONES              7839
               4                      7876      ADAMS              7788
           3                          7902      FORD               7566
               4                      7369      SMITH              7902
       2                              7698      BLAKE              7839
           3                          7499      ALLEN              7698
           3                          7521      WARD               7698
           3                          7654      MARTIN            7698
           3                          7844      TURNER            7698
           3                          7900      JAMES              7698
       2                              7782      CLARK              7839
           3                          7934      MILLER             7782

13 rows selected.
```

在这个查询中，仅剪去了树中单个节点 SCOTT。若希望剪去树结构中的某个分支，则要用 CONNECT BY 子句。CONNECT BY 子句是限定树型结构中的整个分支，既要剪除分支
上的单个节点，也要剪除其后代节点（自顶向下检索时）或前辈节点（自底向顶检索时）。
例．显示 KING 领导下的全体雇员信息，除去 SCOTT 领导的一支。

```
SQL>SELECT LPAD(LEVEL,LEVEL*3,' ')
as "LEVEL",EMPNO,ENAME,MGR
FROM EMP
CONNECT BY PRIOR EMPNO=MGR
AND ENAME!='SCOTT'
START WITH ENAME='KING'
/
```

这个查询结果就除了剪去单个节点 SCOTT 外，还将 SCOTT 的子节点 ADAMS 剪掉，即把 SCOTT 这个分支剪掉了。当然 WHERE 子句可以和 CONNECT BY 子句联合使用，这样能够同时剪掉单个节点和树中的某个分支。
在使用 SELECT 语句来报告树结构报表时应当注意，CONNECT BY 子句不能作用于出现在 WHERE 子句中的表连接。如果需要进行连接，可以先用树结构建立一个视图，再将这个视图与其他表连接，以完成所需要的查询。

--完--

资料整理
火龙果软件
荐