Oracle 数据库 10g 版本
数据库管理员培训讲义

张烈 张建中

前言

开卷有益。

最好的学习教材是 ORACLE 的**文档**。但太繁杂，我们没有时间去钻研各个领域的数据库知识。根据我的教学实际情况，和大多数学员的实际需要情况，我将我的经验与大家分享。

最好的学习方法是**实验**。实验加深你对数据库的理解。

这是一本以实验为主的书.看到结果才是值得信赖的.

本书献给那些想学习 ORACLE 数据库的人。

本书含有六大部分：第一部分 sql 基础，第二部分 pl/sql 基础，第三部分数据库的体系结构和数据库一些包的应用，第四部分数据库的网络配置，第五部分数据库的备份和恢复，第六部分数据库的优化。

张烈　13701394033　zanglie@263.net

张建中 13601085651　zjz3@263.net

2006 年 10 月

Oracle 数据库学习常见问题问答

- Oracle 数据库的书很难看懂,oracle 真的很难学习吗?

Oracle 就是一个**小软件**,它把复杂的事情封装起来了,我们学习的是**管理**数据库.很简单.只要你掌握正确的学习方法,管理 oracle 数据库不难.

- 数据库有好多版本，我应该学习哪个?

万物一理，数据库的版本虽然多，本质是一样的，变化的只是表象，你是 oracle7 的专家，一定也是 oracle10g 的专家。

- 数据库存在好多平台，我应该学习哪个?

各个平台上有差别，很小。**Windows** 是最好的学习平台。

- 有好多管理数据库的工具，我应该使用哪个?

Sqlplus 最好的管理工具，当你只用 sqlplus 管理数据库的时候，你就掌握 oracle 了。

- 数据库学习中哪部分最难?

Sql 语句，永远是 sql,书写**高效的 sql** 是我们永恒的目标。**SQL 是数据库的颠峰**,说实话,oracle 就是 sql 引擎做的好,其它方面都不怎么样,浮夸之风日盛.

- 日常维护数据库最重要的工作是什么?

备份，永远是备份，有数据就有一切。

- 学习数据库的基本课程是哪部分?

体系结构，它是备份和优化数据库的**基石**。管理和维护之必备.

- 如何衡量我的数据库的水平?

你在 oracle 数据库中**想看什么就看得到**，你入门了。看什么都看得懂，你就学明白了。

- 图形界面对数据库学习有帮助吗?

有害无益！ 图形界面只能干一些糙活,你以为你什么都看到了,其实你什么都没有看到.

- 安装 ORACLE 简单吗?

顺利情况下很简单，但每次你都会碰到不一样的情况，需要你的**综合知识**，最简单的事情体现了最精华的部分，工作这么多年，还没有碰到一个大拿的主机工程师，都懂得点皮毛，可叹!

- 我看到的结果和你的实验不同,为什么?

你看到什么都是对的,看到才是真实的.在千变万化的结果中看到不变的真理!

- 我们学习完这本讲义可以达到什么水平?

如果你把这一百六十多个实验做一遍并**理解**了,你数据库入门了,能走多远就看你的日后的实践了.

- 我是**开发人员**,学习这本书有帮助吗?

非常有用,理解数据库的原理会**指导**我们书写高效的 SQL 语句.能用 SQL 实现的绝对不写程序,SQL 发展到今天已经很成熟了,掌握 oracle 的工作原理会使你的编程水平更上一层楼.

- 我是数据库**管理员**,学习 sql,pl/sql 有意义吗?

数据库管理员一定要会,因为数据库内有两个引擎,**sql 引擎和 pl/sql 引擎**,我们虽然不写程序,但要懂.

- 我没有什么计算机专业的基础,能学会数据库吗?

能!数据库**很简单**,人人都能学会.象汽车一样,我们是学开车,不是造汽车.我们不懂汽车的内部结构,但不影响我们驾驶汽车,我们的工作就是管理数据库,不难.

- 我是老程序员了,看你的教材有提高吗?

开卷有益!

- Oracle 的内容很多,我们应该掌握哪些产品?

Oracle 不是一个人做出来的,我们没有必要全面掌握,你掌握了基本的原理,在你的工作方向上深入一下.行业分工很细,一个人不能成为全能大师. **生命有涯,知识无边**.

目录

第一部分 sql 基础

基本查询语句

- Select 语句的作用

查询指定的**行**
查询指定的**列**
多张表**联合**查询
Select 语句可以查询指定的行，指定的列，也可以多张表联合查询来获得数据。上面的三句话，开宗明义的定义了 SQL 的基本功能，书写高效的 SQL 语句是我们永恒的追求，不管你是程序员还是数据库管理员。Select 既是入门所必备，又是数据库之颠峰。

- 简易语法

SELECT *|{[DISTINCT] column|expression [alias],...} FROM table;
大写的为关键字
小写的为我们指定的名称
SELECT 子句指定你所关心的列
FROM 子句指定你所要查询的表
之所以称之为简易语法，因为完全的 SELECT 语法很长，涉及到很多的逻辑关系，我们由浅入深。虽然不能大成，但小成总会有的。
一般我们将 select 叫做 select 子句，from 叫做 from 子句。

实验 1：书写一个最简单的 sql 语句，查询一张表的所有行和所有列

该实验的目的是初步认识 sql 语句,执行一个最简单的查询.

```
Select * from emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

你看到可能折行了,没有关系,后面我们会讲到.
查询 emp 表的所有行，所有列。对于小的表我们可以这样书写，对于大的表我们一般查询指定条件的行和我选定的列。Emp 表在 scott 用户下.scott 用户是一个练习帐号,密码是 tiger,如果你没有这个帐号可以随时建立,练习完了可以随时删除.招之既来,挥之既去.
10g 版本数据库中,scott 用户默认是锁定的.使用下面的语法解锁.

```
SQL> conn / as sysdba
Connected.
SQL> alter user scott account unlock identified by tiger;
```

解锁同时修改密码

```
User altered.
```

如果没有 scott 用户也不要紧.数据库内含了建立该用户的脚本

```
SQL> conn / as sysdba
Connected.
SQL> @%oracle_home%\rdbms\admin\utlsampl.sql
```
建立完成以后会自动退出 sqlplus,请重新登录既可.
运行 sqlplus /nolog
```
SQL> conn scott/tiger
Connected.
SQL> select * from tab;

TNAME
------------------------
DEPT
EMP
BONUS
SALGRADE
```
查看当前用户的所有的表.

实验 2：查询一张表的所有行，但列的顺序我们自己决定

该实验的目的是练习查询指定的列.
列的名称之间要使用逗号间隔,列的顺序由我们来指定.
Select ename,sal from emp;这里我们指定表中的两个列。其它的列我们不看。

ENAME	SAL
SMITH	806
ALLEN	1606
WARD	1256
JONES	2981
MARTIN	1256
BLAKE	2856
CLARK	2456
KING	5006
TURNER	1506
JAMES	956
FORD	3006
MILLER	1306

实验 3：查询表的某些列，在列上使用表达式

该实验的目的是使用表达式,对表的部分列进行运算.
Select ename,sal,sal+300 from emp;
其中 sal+300 是表达式，它并不存在于数据库中，是计算出来的结果。也可以使用函数.

ENAME	SAL	SAL+300
SMITH	806	1106
ALLEN	1606	1906
WARD	1256	1556
JONES	2981	3281
MARTIN	1256	1556
BLAKE	2856	3156
CLARK	2456	2756
KING	5006	5306
TURNER	1506	1806
JAMES	956	1256
FORD	3006	3306
MILLER	1306	1606

- 表达式的运算是有优先级的,和程序中的一样,先乘除后加减,括号强制优先级.

+ - * /

先乘除，后加减，括号强制优先级

```
Select ename,12*sal+300 from emp;
ENAME      12*SAL+300
---------- ----------
SMITH            9900
ALLEN           19500
WARD            15300
JONES           36000
MARTIN          15300
BLAKE           34500
CLARK           29700
KING            60300
TURNER          18300
JAMES           11700
FORD            36300
MILLER          15900
```

年终奖为 300 元。

```
Select ename,12*(sal+300) from emp;
ENAME      12*(SAL+300)
---------- ------------
SMITH             13200
ALLEN             22800
WARD              18600
JONES             39300
MARTIN            18600
BLAKE             37800
CLARK             33000
KING              63600
TURNER            21600
JAMES             15000
FORD              39600
MILLER            19200
```

每个月 300 元奖金。

- SQLPLUS 介绍

SQLPLUS 是 ORACLE 公司开发的很简洁的管理工具，初学者使用不习惯，但我使用了多年，从来没有使用过其它工具来管理数据库，因为你所要做的一切，SQLPLUS 都会很好的完成，其它的第三方所有的工具，一言以蔽之，狗尾续貂。请学员明大义，识大体，不要为虚浮的外表所迷惑，SQLPLUS 是最好的，最核心的 ORACLE 管理工具。SQLPLUS 简洁而高效，舍弃浮华，反璞归真。

实验 4：使用 sqlplus,进入 sqlplus 并进行简单的操作

该实验的目的是熟悉 oracle 的小工具 sqlplus 的使用.

- 如何进入 SQLPLUS 界面

进入 DOS,然后键入如下命令

```
C:\>sqlplus /nolog
```

进入字符界面

```
C:\>sqlplusw /nolog
```

进入 windows 界面，windows 平台特有的。

/nolog 是不登录的意思。只进入 SQLPLUS 程序提示界面。

等待你输入命令。

- SQLPLUS 的基本操作

```
Sql>connect / as sysdba
```

连接到本地的最高帐号
Sql>help index
Enter Help [topic] for help.

@	COPY	PAUSE	SHUTDOWN
@@	DEFINE	PRINT	SPOOL
/	DEL	PROMPT	SQLPLUS
ACCEPT	DESCRIBE	QUIT	START
APPEND	DISCONNECT	RECOVER	STARTUP
ARCHIVE LOG	EDIT	REMARK	STORE
ATTRIBUTE	EXECUTE	REPFOOTER	TIMING
BREAK	EXIT	REPHEADER	TTITLE
BTITLE	GET	RESERVED WORDS (SQL)	UNDEFINE
CHANGE	HELP	RESERVED WORDS (PL/SQL)	VARIABLE
CLEAR	HOST	RUN	WHENEVER OSERROR
COLUMN	INPUT	SAVE	WHENEVER SQLERROR
COMPUTE	LIST	SET	
CONNECT	PASSWORD	SHOW	

显示 SQLPLUS 命令的帮助，而不是 SQL 语法的帮助，它是查询的数据库内的一张表，所以你要得到帮助需要两个条件，一、数据库是打开的。二、存在 HELP 表
Sql>show all
显示当前 SQLPLUS 的环境设置
Sql>show user
显示当前所登录的用户信息

- Sqlplus 的屏幕缓冲的大小

在 oracle_home\sqlplus\admin\ sqlplus.ini 文件中描述了屏幕缓冲的大小
#Sql*Plus user initialzation file. DO NOT MODIFY
[WindowSize: L T R B] 0009 0000 1024 0735
[ScreenBuffer: W L] 0120 1000
其中 0120 表示每行 120 字符，默认为 100，有点小。 1000 表示每页为 1000 行，最大可以设置为 2000。

- Sqlplus 的基本操作

Spool 命令是将屏幕的显示输入到文本文件内，以便查看，有点象屏幕转存。
SPOOL C:\1.TXT
SELECT * FROM EMP;
SPOOL OFF
以上三行就将 SPOOL 和 SPOOL OFF 所夹的屏幕输出到 c:\1.txt 文件中。
Spool c:\1.txt append
Select * from dept;
Spool off
加 APPEND 命令的含义是续写 c:\1.txt,如果不加，将会把原来的 c:\1.txt 覆盖，这是 **10G 的新特性**，以前的数据库版本不能续写，只能指定新的文件名称。

Run
运行 SQLPLUS 缓冲区内的 SQL 语句，可以缩写为 r
/
与 run 命令相同，运行 SQLPLUS 缓冲区内的 SQL 语句

@脚本
@%oracle_home%\rdbms\admin\utlxplan.sql
该句话的含义为运行指定的脚本。
@@为运行相对路径下的脚本，一般是在大脚本调用小脚本的时候使用。

Save
将当前 SQLPLUS 缓冲区内的 SQL 语句保存到指定的文件中
如 save c:\2.txt

Get
将文件中的 SQL 语句调入到 SQLPLUS 缓冲区内。
如 get c:\2.txt

Eidt
编辑当前 SQLPLUS 缓冲区内的 SQL 语句
如 ed
--是注释当前行
/*　　*/是注释多行
● 　建立会话，和数据库发生连接
Sql>connect scott/tiger
连接到 SCOTT 用户，密码为 tiger
如果不写密码，你回车后会提示你输入密码。
Sql>Help index　会列出 sqlplus 命令的帮助.
SQL> help index

进一步的帮助
SQL> help LIST

LIST

Lists one or more lines of the most recently executed SQL command
or PL/SQL block which is stored in the SQL buffer. Enter LIST with
no clauses to list all lines. In SQL*Plus command-line you can also
use ";" to list all the lines in the SQL buffer. The buffer has no
command history list and does not record SQL*Plus commands.

L[IST] [n | n m | n * | n LAST | * | * n | * LAST | LAST]
其中中括号前为**缩写**.括号内的可以写,也可以不写
有的时候 sqlplus 会显示的有点乱,光标不在最后,请清屏.
SQL> clear screen

实验 5：查看当前用户的所有表和视图

该实验的目的是查看简单的数据字典,熟悉实验环境.
Select * from tab;

TNAME	TABTYPE	CLUSTERID
DEPT	TABLE	
EMP	TABLE	
BONUS	TABLE	
SALGRADE	TABLE	

显示当前用户所拥有的表和视图。其中 tab 是数据字典，你在每个用户下查看都看到是当前用户的表和视图，这是最基本的字典，我们一定要知道当前用户下的表和视图。

Select * from dept;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

显示 DEPT 表的所有行和所有列,各列的含义为:部门代码,部门名称,部门地址

Select * from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30

```
 7566 JONES      MANAGER         7839 02-APR-81          2975          20
 7654 MARTIN     SALESMAN        7698 28-SEP-81          1250  1400    30
 7698 BLAKE      MANAGER         7839 01-MAY-81          2850          30
 7782 CLARK      MANAGER         7839 09-JUN-81          2450          10
 7839 KING       PRESIDENT            17-NOV-81          5000          10
 7844 TURNER     SALESMAN        7698 08-SEP-81          1500     0    30
 7900 JAMES      CLERK           7698 03-DEC-81           950          30
 7902 FORD       ANALYST         7566 03-DEC-81          3000          20
 7934 MILLER     CLERK           7782 23-JAN-82          1300          10
```

显示 EMP 表的所有行和所有列

* 代表所有的列。

```
Desc emp;
Name                                                  Null?    Type
----------------------------------------------------- -------- ------------
EMPNO          注释:该列为员工代码                     NOT NULL NUMBER(4)
ENAME          注释:该列为员工名称                              VARCHAR2(10)
JOB            注释:该列为员工工作                              VARCHAR2(9)
MGR            注释:该列为员工的经理                            NUMBER(4)
HIREDATE       注释:该列为员工上班日期                          DATE
SAL            注释:该列为员工工资                              NUMBER(7,2)
COMM           注释:该列为员工奖金                              NUMBER(7,2)
DEPTNO         注释:该列为员工的部门代码                        NUMBER(2)
```

查看表结构

对这两张表大家一定要熟悉，因为我们所有的例题都是以这两个表为基础。

```
Sql>list
```

查看当前缓冲区内的语句。

简写为 l

```
Sql>help list      —可以查看缩写
```

- Scott 用户的建立

数据库默认建立以后 SCOTT 用户是存在的，在 10G 的版本以后默认为锁定的。我们需要解锁该用户。

```
SQL> conn / as sysdba    进入到数据库的最高用户。
Sql>alter user scott account unlock identified by tiger;
```

解锁 SCOTT 帐户，同时修改该用户的密码为 TIGER。10G 中该用户默认为锁定，而在 10G 前是没有锁定的。

```
Sql>alter user system identified by manager;
```

修改 SYSTEM 用户的密码为 MANGER，为了以后的实验方便，我以后默认的脚本都是使用该密码，数据库以前版本的默认密码也是 MANAGER

```
Sql>alter user sys identified by sys;
```

修改 SYS 用户的密码为 SYS，同样为了实验的方便，在生产环境请设定自己的密码。

SCOTT 用户可以随时被删除和建立，该用户存在的目的就是为了实验用的，表很少，但又代表了一定的典型数据库的应用。

```
Sql> @%oracle_home%\rdbms\admin\scott.sql
```

utlsampl.sql 脚本也是建立 scott 用户的,有点差别,但不大.

会重新建立 SCOTT 用户，并建立相应的表和表之间的关系。

- 书写 SQL 语句的原则

大小写不敏感，但单引和双引内的大小写是敏感的。切记！

关键字不能缩写

可以分行书写，但关键字不能被跨行书写，单引内也不要跨行书写。

一般每个子句是一行

可以排版来增加可读性

字符串用单引

列的别名用双引

实验 6：关于 null 值的问题

该实验的目的是练习数据库的一个重要值 null 的使用.

- Null 值

```
Select ename,sal,comm from emp;
ENAME             SAL  COMM
---------- ---------- -----
SMITH              800
ALLEN             1600   300
WARD              1250   500
JONES             2975
MARTIN            1250  1400
BLAKE             2850
CLARK             2450
KING              5000
TURNER            1500     0
JAMES              950
FORD              3000
MILLER            1300
```

其中 comm 列中有一些行没有值，是空值（null）。
Null 值不等于 0，也不等于空格。
Null 值是未赋值的值，不入索引。
NULL 是双刃剑，使用好了提高性能，你对它不了解，往往是错误的根源，切记！

实验 7：在列上起一个别名

该实验的目的是了解使用别名的目的和别名的使用方法.

- 别名的使用原则

1。区分同名列的名称
2。非法的表达式合法化
3。按照你的意愿显示列的名称
4。特殊的别名要双引
5。直接写列的后面
6。使用 as 增加可读性

```
Select sal as salary,hiredate  "上班日期" ,sal*12  total_salary from emp;
SALARY 上班日期     TOTAL_SALARY
------ ----------- ------------
   800 17-DEC-80            9600
  1600 20-FEB-81           19200
  1250 22-FEB-81           15000
  2975 02-APR-81           35700
  1250 28-SEP-81           15000
  2850 01-MAY-81           34200
  2450 09-JUN-81           29400
  3000 19-APR-87           36000
  5000 17-NOV-81           60000
  1500 08-SEP-81           18000
  1100 23-MAY-87           13200
   950 03-DEC-81           11400
  3000 03-DEC-81           36000
  1300 23-JAN-82           15600
```

- 重复的行

SELECT 语句显示重复的行。用 DISTINCT 语法来去掉重复的行。

```
Select deptno from emp;
DEPTNO
------
```

20
30
30
20
30
30
10
10
30
30
20
10

我们会看到很多重复的行，如果我们想去掉重复的行，我们需要 distinct 关键字。

实验 8：在显示的时候去掉重复的行

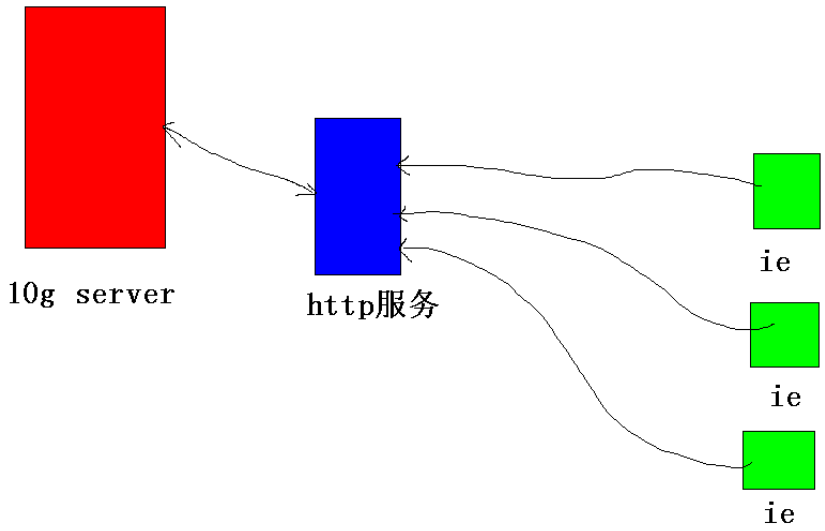
该实验的目的是使用 distinct 关键字,去掉重复的行.

```
Select distinct deptno from emp;
DEPTNO
------
    30
    20
    10
```

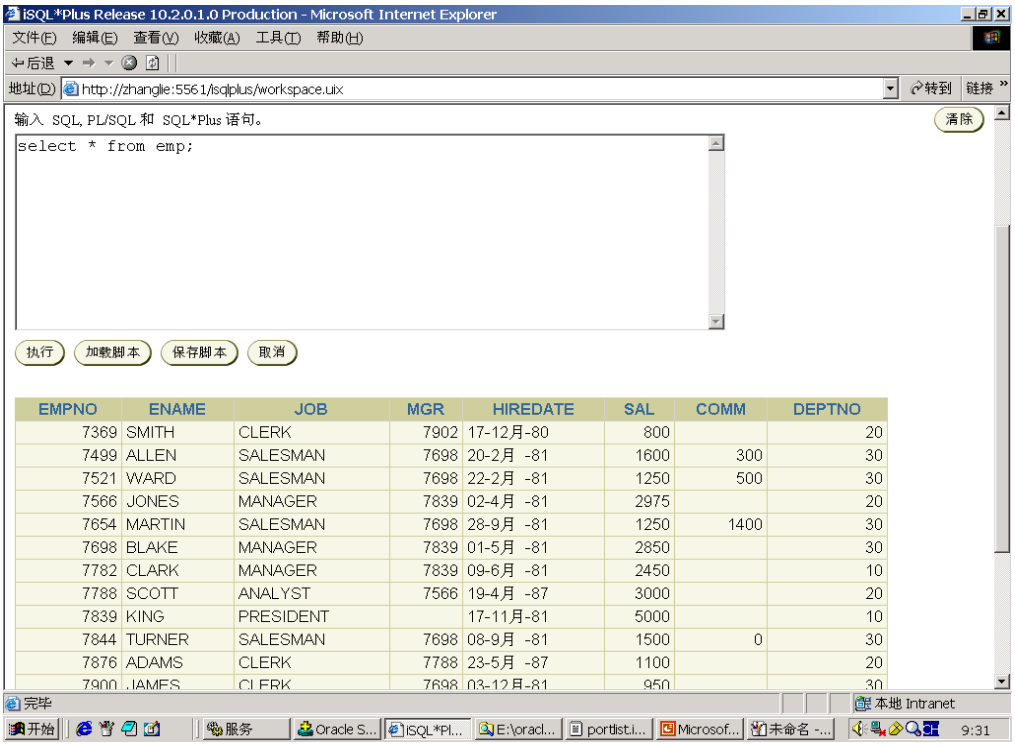
在 ORACLE 数据库的 10G 前版本，该语句需要排序才能去掉重复的行，而在 10G 中数据库并不需要排序，而是使用 HASH 算法来去掉重复的行，由于避免了排序，从而极大的提高了 SQL 语句的效率，因为 10G 的 SQL 内核改写了。效率更加的高。因为没有排序，所以输出也是无序的。

- Isqlplus

Isqlplus 是以 ie 方式连接的数据库，狗尾续貂，不建议使用。
显示结果美化
编辑功能加强
通过中间层的转换
Oracle_home\install\portlist.ini 文件中列出端口号
OracleOraDb10g_home2iSQL*Plus 服务要启动



http://hostname:5561/isqlplus



- Where 和 order by 子句

语法

```
SELECT    *|{[DISTINCT] column|expression [alias],...}
FROM table
[WHERE    condition(s)]
[order by column|expression| alias ];
```

Where 一定要放在 FROM 子句的后面。
符合条件的行会被筛选出来。

Order by 放在最后，用来排序显示结果

实验 9：显示表的部分行和部分列，使用 where 子句过滤出想要的行

该实验的目的是使用 where 子句.

```
Select deptno,ename from emp Where deptno=10;
DEPTNO ENAME
------ ------
    10 CLARK
    10 KING
    10 MILLER
```

只显示 10 号部门的员工名称。

```
Select * from emp where ename='KING';
EMPNO ENAME      JOB              MGR HIREDATE            SAL  COMM DEPTNO
----- ---------- --------- ---------- ------------ ---------- ----- ------
 7839 KING       PRESIDENT            17-NOV-81          5000            10
```

显示 KING 员工的详细信息。字符串要单引，字符串大小写敏感，日期格式敏感，牢记在心。

- 关系运算

=

<>,!=,^=

>=

<=

>

<

Between...and.....

```
SQL>  Select ename,sal from emp Where sal between 1000 and 3000;
ENAME             SAL
---------- ----------
ALLEN            1600
WARD             1250
JONES            2975
MARTIN           1250
BLAKE            2850
CLARK            2450
TURNER           1500
FORD             3000
MILLER           1300
```

含上下界

In 操作，穷举，据说穷举不能超过 1000 个值，我没有去验证。一般我们也不会穷举到 1000 个值，如果到 1000 请改写你的 SQL。

```
Select deptno,ename,sal from emp Where deptno  in(10,20);
DEPTNO ENAME             SAL
------ ---------- ----------
    20 SMITH             800
    20 JONES            2975
    10 CLARK            2450
    10 KING             5000
    20 FORD             3000
    10 MILLER           1300
```

实验 10：使用 like 查询近似的值

该实验的目的是掌握 like 的通配符.还有逻辑运算.

- Like 运算

_ 通配一个，仅匹配一个字符，
% 通配没有或多个字符
```
select ename,deptno from emp where ename like 'J%';
ENAME      DEPTNO
---------- ------
JONES          20
JAMES          30
```
首字母为 J 的员工，J 后有没有字符，有多少字符都不管。
```
select ename,deptno from emp where ename like '_A%';
ENAME      DEPTNO
---------- ------
WARD           30
MARTIN         30
JAMES          30
```
寻找第二个字母为 A 的员工，第一个字母必须有，是什么无所谓。
当你想查询_,%特殊字符时，请用 escape.

```
Select ename from emp where ename like '%s_%' escape 's ';
```
我们并不想查找 S 后必须有一个字符以上的员工，而是要剔除 S，S 出现的目的就是转义，将_转义了，这里的_不是通配符，而是实际意义的_。
```
Select ename from emp where ename like '%/_%' escape '/';
```
一般我们使用/来转义，以免产生歧异。

- 查询 NULL 值

```
SQL> Select ename,comm from emp where comm = null;
no rows selected
```
因为 null 不等于 null,所以没有行被选出。未知不等于未知,无穷不等于无穷.

```
SQL> Select ename,comm from emp where comm = 'null';
Select ename,comm from emp where comm = 'null'
                                        *
ERROR at line 1:
ORA-01722: invalid number
```
数据类型不匹配

```
SQL> Select ename,comm from emp where comm is null;
ENAME       COMM
---------- -----
SMITH
JONES
BLAKE
CLARK
KING
JAMES
FORD
MILLER
```

- And 运算

```
Select ename,deptno,sal From emp Where deptno=30 and sal>1200;
ENAME      DEPTNO        SAL
---------- ------ ----------
ALLEN          30       1600
WARD           30       1250
MARTIN         30       1250
BLAKE          30       2850
TURNER         30       1500
```
两个条件的交集，必须同时满足。

- OR 运算

```
Select ename,deptno,sal From emp Where deptno=30 or sal>1200;
```

```
ENAME      DEPTNO        SAL
---------- ------ ----------
ALLEN          30       1600
WARD           30       1250
JONES          20       2975
MARTIN         30       1250
BLAKE          30       2850
CLARK          10       2450
KING           10       5000
TURNER         30       1500
JAMES          30        950
FORD           20       3000
MILLER         10       1300
```

两个条件的并集，满足一个就可以。

- not 运算

```
Select ename,deptno,sal From emp Where ename not like 'T%';
ENAME      DEPTNO        SAL
---------- ------ ----------
SMITH          20        800
ALLEN          30       1600
WARD           30       1250
JONES          20       2975
MARTIN         30       1250
BLAKE          30       2850
CLARK          10       2450
KING           10       5000
JAMES          30        950
FORD           20       3000
MILLER         10       1300
```

补集，不是 T 打头的员工。

- 优先级

1。算术运算
2。连接运算
3。关系运算
4。 IS [NOT] NULL, LIKE, [NOT] IN
5。Between
6.not
7.and
8.or
括号强制优先级

- Order by 子句

不指明都是二进制排序，如果你想按照拼音，部首，笔画，法语等特殊的排序模式，请设定排序的环境变量，关于国家语言的支持问题我们再 DBA 体系结构中描述。
默认是升序 asc
降序要指定 desc

实验 11：使用 order by 子句来进行排序操作

该实验的目的是掌握排序操作.

```
Select ename,sal from emp order by sal;
ENAME             SAL
---------- ----------
SMITH              800
JAMES              950
WARD              1250
```

```
MARTIN            1250
MILLER            1300
TURNER            1500
ALLEN             1600
CLARK             2450
BLAKE             2850
JONES             2975
FORD              3000
KING              5000
```

不说排序的类型就是升序。

```
Select ename,sal from emp order by sal desc;
ENAME             SAL
---------- ----------
KING              5000
FORD              3000
JONES             2975
BLAKE             2850
CLARK             2450
ALLEN             1600
TURNER            1500
MILLER            1300
MARTIN            1250
WARD              1250
JAMES              950
SMITH              800
```

降序要明确的指出。

隐式排序，显示的结果里没有工资，但是按照工资的顺序显示的。

```
Select ename from emp order by sal;
ENAME
------
SMITH
JAMES
WARD
MARTIN
MILLER
TURNER
ALLEN
CLARK
BLAKE
JONES
FORD
KING
```

别名排序

```
Select sal*12  salary from emp order by salary;
```

表达式排序

```
Select sal*12  salary from emp order by sal*12;
```

位置排序,对集合操作时比较方便.

```
Select ename,sal from emp order by 2;
```

多列排序

```
Select deptno,job,ename,sal from emp order by deptno,job;
DEPTNO JOB       ENAME             SAL
------ --------- ---------- ----------
    10 CLERK     MILLER            1300
    10 MANAGER   CLARK             2450
    10 PRESIDENT KING              5000
    20 ANALYST   FORD              3000
    20 CLERK     SMITH              800
```

```
    20 MANAGER   JONES            2975
    30 CLERK     JAMES             950
    30 MANAGER   BLAKE            2850
    30 SALESMAN  TURNER           1500
    30 SALESMAN  WARD             1250
    30 SALESMAN  ALLEN            1600
    30 SALESMAN  MARTIN           1250
```

先按照部门排序，部门相同的再按照工作排序。

练习
1。查询 30 号部门的员工，显示名称和工资。
2。查询第三个字母为 A 的员工。
3。查询员工的名称和上班日期，日期反序排列。

- 函数

使用函数的目的是为了操作数据
将输入的变量处理，返回一个结果。
变量可以有好多。
传入的变量可以是列的值，也可以是表达式。
函数可以嵌套。
内层函数的结果是外层函数的变量。

单行函数：每一行都有一个返回值，但可以有多个变量。
多行函数：多行有一个返回值。

- 单行函数的分类

字符操作函数
数字操作函数
日期操作函数
数据类型转换函数
综合数据类型函数

- 字符操作函数

大小写操作函数
Lower,upper,initcap
字符串操作函数
Concat,length,substr,instr,trim,replace,lpad,rpad

实验 12：操作字符串的函数

该实验的目的是掌握常用的字符串操作的函数.
字符串的大小写操作

```
Select lower(ename), upper(ename), initcap(ename) from emp;
LOWER(ENAM UPPER(ENAM INITCAP(EN
---------- ---------- ----------
smith      SMITH      Smith
allen      ALLEN      Allen
ward       WARD       Ward
jones      JONES      Jones
martin     MARTIN     Martin
blake      BLAKE      Blake
clark      CLARK      Clark
king       KING       King
turner     TURNER     Turner
james      JAMES      James
ford       FORD       Ford
miller     MILLER     Miller
小写       大写       首字母大写
Select lower('mf  TR'), upper('mf  TR'), initcap('mf  TR') from dual;
```

```
LOWER('MFTR')                  UPPER('MFTR')                  INITCAP('MFTR')
------------------------------ ------------------------------ ---------------
mf  tr                         MF  TR                         Mf  Tr
```

Dual 是虚表，让我们用表的形式来访问函数的值。

其它字符串操作函数

```
select ename,job,concat(ename,job) from emp;
ENAME      JOB       CONCAT(ENAME,JOB)
---------- --------- ----------------
SMITH      CLERK     SMITHCLERK
ALLEN      SALESMAN  ALLENSALESMAN
WARD       SALESMAN  WARDSALESMAN
JONES      MANAGER   JONESMANAGER
MARTIN     SALESMAN  MARTINSALESMAN
BLAKE      MANAGER   BLAKEMANAGER
CLARK      MANAGER   CLARKMANAGER
KING       PRESIDENT KINGPRESIDENT
TURNER     SALESMAN  TURNERSALESMAN
JAMES      CLERK     JAMESCLERK
FORD       ANALYST   FORDANALYST
MILLER     CLERK     MILLERCLERK
```

将两个字符连接到一起

下面三句话是求字符串的长度，字符串要单引。

```
select length('张三') from dual;--按照字
select lengthb('张三') from dual;--按字节
select lengthc('张三') from dual;--unicode 的长度
SQL> select length('张三') from dual;

LENGTH('张三')
--------------
             2

SQL> select lengthb('张三') from dual;

LENGTHB('张三')
---------------
              4

SQL> select lengthc('张三') from dual;

LENGTHC('ÕÅÈý')
---------------
              2

select ename,substr(ename,1,1) "first",substr(ename,-1) "last"from emp;
ENAME      first                last
---------- -------------------- ----
SMITH      S                    H
ALLEN      A                    N
WARD       W                    D
JONES      J                    S
MARTIN     M                    N
BLAKE      B                    E
CLARK      C                    K
KING       K                    G
TURNER     T                    R
JAMES      J                    S
FORD       F                    D
```

```
MILLER     M                    R
```
substr(字符串,m,n),m 是从第几个字符开始，如果为负的意思是从后边的第几个开始。N 是数多少个，如果不说就是一直到字符串的结尾。

```
select ename,instr(ename,'A') "A 在第几位" from emp;
ENAME       A 在第几位
---------- ----------
SMITH               0
ALLEN               1
WARD                2
JONES               0
MARTIN              2
BLAKE               3
CLARK               3
KING                0
TURNER              0
JAMES               2
FORD                0
MILLER              0
```
求子串在父串中的位置，0 表示没有在父串中找到该子串。

截断字符串和添加字符串的函数
```
SQL> select trim(leading 'a' from 'aaaaababababaaaaa') from dual;
TRIM(LEADING'A'FROM'AAAAABABAB
------------------------------
bababaaaaa
```
截掉连续的前置的 a

```
SQL> select trim(trailing 'a' from 'aaaaababababaaaaa') from dual;
TRIM(TRAILING'A'FROM'AAAAABABA
------------------------------
aaaaababab
```
截掉连续的后置的 a

```
SQL> select trim(both 'a' from 'aaaaababababaaaaa') from dual;
TRIM(BOTH'A'FROM'AAAAABABABAAA
------------------------------
babab
```
截掉连续的前置和后置的 a

```
SQL> select trim('a' from 'aaaaababababaaaaa') from dual;
TRIM('A'FROM'AAAAABABABAAAAA')
------------------------------
Babab
```
如果不说明是前置还是后置就是 both 全截断。
Trim 函数是截掉头或者尾连续的字符，一般我们的用途是去掉空格。

```
SQL> select lpad(ename,20,'-') ename,rpad(ename,20,'-') ename from emp;

ENAME                                ENAME
------------------------------------ ------------------------------
----------------SMITH                SMITH----------------
----------------ALLEN                ALLEN----------------
----------------WARD                 WARD----------------
----------------JONES                JONES----------------
----------------MARTIN               MARTIN----------------
----------------BLAKE                BLAKE----------------
----------------CLARK                CLARK----------------
----------------KING                 KING----------------
```

```
----------TURNER                    TURNER----------
---------JAMES                      JAMES---------
---------FORD                       FORD---------
----------MILLER                    MILLER----------
```
左铺垫和右铺垫，20 是总共铺垫到多少位，-是要铺垫的字符串。

```
SQL> select lpad(sal,2,' ') ename,rpad(sal,10,' ') ename,sal from emp;
别名                                    别名
ENAME    位数不足                       ENAME    左对齐                                  SAL
---------------------------------------- ---------------------------------------- ----------
80                                      800                                       800
16                                      1600                                     1600
12                                      1250                                     1250
29                                      2975                                     2975
12                                      1250                                     1250
28                                      2850                                     2850
24                                      2450                                     2450
50                                      5000                                     5000
15                                      1500                                     1500
95                                      950                                       950
30                                      3000                                     3000
13                                      1300                                     1300
```
Lpad 左铺垫，rpad 右铺垫，一般的用途是美化输出的结果。
如果位数不足，按照截取后的结果显示，不报错。

```
SELECT REPLACE('JACK and JUE','J','BL')  FROM DUAL;
BLACK and BLUE
```
将字符串中的 J 全部替换位 BL

实验 13：操作数字的函数

该实验的目的是掌握常用的关于数字操作的函数.

- 数字操作函数

```
SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,-1) FROM   DUAL;
ROUND(45.923,2) ROUND(45.923,0) ROUND(45.923,-1)
--------------- --------------- ---------------
          45.92              46              50
SELECT  TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-2) FROM   DUAL;
TRUNC(45.923,2) TRUNC(45.923) TRUNC(45.923,-2)
--------------- ------------- ---------------
          45.92            45               0
```
以小数点位核心，2 是小数点后两位，0 可以不写，表示取整，-1 表示小数点前一位
ROUND 是四舍五入，TRUNC 是截断，全部舍弃。

```
select ceil(45.001) from dual;取整，上进位，和 trunc 全部去掉正好相反
CEIL(45.001)
------------
          46
```

```
select abs(-23.00) from dual;取绝对值
select mod(sal,2000) from emp;取余数
```

实验 14：操作日期的函数

该实验的目的是掌握常用的关于日期操作的函数.

- 系统日期的操作

日期是很特殊的数据类型，用好了可以提高数据库的性能，而使用不当往往是错误的根源，如果你使用字符型数据来存储日期，就放弃日期特有的计算功能。
函数 SYSDATE 求当前数据库的时间。
```
select sysdate from dual;
SYSDATE
---------
01-MAY-07
```
日期的显示格式和客户端的配置相关。

查看当前的日期显示格式
```
select * from nls_session_parameters where parameter='NLS_DATE_FORMAT';
PARAMETER                VALUE
------------------------ ---------
NLS_DATE_FORMAT          DD-MON-RR
```
如果你的显示是如下样子：
```
PARAMETER
----------------
VALUE
----------------
NLS_DATE_FORMAT
DD-MON-RR
```
这表示折行了，请限制 value 列的宽度
```
col value for a20
```
代表的含义是凡是列的名称是 value 的，都按照 20 个宽度来显示，你想取消该列的定义
col value **clear**，其中 col 是 column 的缩写。你想查看帮助 help column 即可

```
alter session set NLS_DATE_FORMAT='yyyy/mm/dd:hh24:mi:ss';
```
重新设定为我们想要的格式。
```
select sysdate from dual;
SYSDATE
-------------------
2007/05/01:16:32:54
```
查看系统时间，数据库本身没有时间，它有 scn 号，和我们的时间不同。
```
alter session set NLS_DATE_FORMAT='DD-MON-RR';
```
设定为默认的显示格式
```
select sysdate from dual;
```
再次查看，我们发现日期的显示随着客户端的格式变化而变化。

日期的内部存储都是以 yyyymmddhh24miss 存在数据库中

- 日期的操作函数

```
select round(sysdate-hiredate) days,sysdate,hiredate from emp;

 DAYS SYSDATE             HIREDATE
----- ------------------- -------------------
 9632 2007/05/01:16:37:33 1980/12/17:00:00:00
 9567 2007/05/01:16:37:33 1981/02/20:00:00:00
 9565 2007/05/01:16:37:33 1981/02/22:00:00:00
 9526 2007/05/01:16:37:33 1981/04/02:00:00:00
 9347 2007/05/01:16:37:33 1981/09/28:00:00:00
 9497 2007/05/01:16:37:33 1981/05/01:00:00:00
 9458 2007/05/01:16:37:33 1981/06/09:00:00:00
 9297 2007/05/01:16:37:33 1981/11/17:00:00:00
 9367 2007/05/01:16:37:33 1981/09/08:00:00:00
 9281 2007/05/01:16:37:33 1981/12/03:00:00:00
 9281 2007/05/01:16:37:33 1981/12/03:00:00:00
 9230 2007/05/01:16:37:33 1982/01/23:00:00:00
```
两个日期相减的结果单位为天，往往是带小数点。我们通过函数可以取整。
```
SQL> select months_between(sysdate,hiredate) ,sysdate,hiredate from emp;
```

```
MONTHS_BETWEEN(SYSDATE,HIREDATE) SYSDATE             HIREDATE
-------------------------------- ------------------- -------------------
                      316.506237 2007/05/01:16:38:24 1980/12/17:00:00:00
                      314.409462 2007/05/01:16:38:24 1981/02/20:00:00:00
                      314.344946 2007/05/01:16:38:24 1981/02/22:00:00:00
                      312.990108 2007/05/01:16:38:24 1981/04/02:00:00:00
                      307.151398 2007/05/01:16:38:24 1981/09/28:00:00:00
                             312 2007/05/01:16:38:24 1981/05/01:00:00:00
                      310.764301 2007/05/01:16:38:24 1981/06/09:00:00:00
                      305.506237 2007/05/01:16:38:24 1981/11/17:00:00:00
                      307.796559 2007/05/01:16:38:24 1981/09/08:00:00:00
                      304.957849 2007/05/01:16:38:24 1981/12/03:00:00:00
                      304.957849 2007/05/01:16:38:24 1981/12/03:00:00:00
                      303.312688 2007/05/01:16:38:24 1982/01/23:00:00:00
```

取两个日期的月间隔

```
select add_months(hiredate,6) ,hiredate from emp;
ADD_MONTHS(HIREDATE HIREDATE
------------------- -------------------
1981/06/17:00:00:00 1980/12/17:00:00:00
1981/08/20:00:00:00 1981/02/20:00:00:00
1981/08/22:00:00:00 1981/02/22:00:00:00
1981/10/02:00:00:00 1981/04/02:00:00:00
1982/03/28:00:00:00 1981/09/28:00:00:00
1981/11/01:00:00:00 1981/05/01:00:00:00
1981/12/09:00:00:00 1981/06/09:00:00:00
1982/05/17:00:00:00 1981/11/17:00:00:00
1982/03/08:00:00:00 1981/09/08:00:00:00
1982/06/03:00:00:00 1981/12/03:00:00:00
1982/06/03:00:00:00 1981/12/03:00:00:00
1982/07/23:00:00:00 1982/01/23:00:00:00
```

六个月过后是哪天。

```
select next_day(hiredate,'friday') ,hiredate from emp;
NEXT_DAY(HIREDATE,' HIREDATE
------------------- -------------------
1980/12/19:00:00:00 1980/12/17:00:00:00
1981/02/27:00:00:00 1981/02/20:00:00:00
1981/02/27:00:00:00 1981/02/22:00:00:00
1981/04/03:00:00:00 1981/04/02:00:00:00
1981/10/02:00:00:00 1981/09/28:00:00:00
1981/05/08:00:00:00 1981/05/01:00:00:00
1981/06/12:00:00:00 1981/06/09:00:00:00
1981/11/20:00:00:00 1981/11/17:00:00:00
1981/09/11:00:00:00 1981/09/08:00:00:00
1981/12/04:00:00:00 1981/12/03:00:00:00
1981/12/04:00:00:00 1981/12/03:00:00:00
1982/01/29:00:00:00 1982/01/23:00:00:00
```

当前的日期算起，下一个星期五是哪一天，这句话你可能运行失败，因为日期和客户端的字符集设置有关系，如果你是英文的客户端，就的用 Friday 来表达，日期是格式和字符集敏感的。如果你是中文的客户端，就的用‘星期五’来表达。

```
select last_day(hiredate) ,hiredate from emp;
```

该日期的月底是哪一天。

```
LAST_DAY(HIREDATE)  HIREDATE
------------------- -------------------
1980/12/31:00:00:00 1980/12/17:00:00:00
1981/02/28:00:00:00 1981/02/20:00:00:00
1981/02/28:00:00:00 1981/02/22:00:00:00
```

1981/04/30:00:00:00 1981/04/02:00:00:00
1981/09/30:00:00:00 1981/09/28:00:00:00
1981/05/31:00:00:00 1981/05/01:00:00:00
1981/06/30:00:00:00 1981/06/09:00:00:00
1981/11/30:00:00:00 1981/11/17:00:00:00
1981/09/30:00:00:00 1981/09/08:00:00:00
1981/12/31:00:00:00 1981/12/03:00:00:00
1981/12/31:00:00:00 1981/12/03:00:00:00
1982/01/31:00:00:00 1982/01/23:00:00:00

- 日期的进位和截取

```
select hiredate,round(hiredate,'mm') ,round(hiredate,'month')  from emp;
select hiredate,round(hiredate,'yyyy') ,round(hiredate,'year')  from emp;
select hiredate,trunc(hiredate,'mm') ,trunc(hiredate,'month')  from emp;
select hiredate,trunc(hiredate,'yyyy') ,trunc(hiredate,'year')  from emp;
```

数字的进位和截取是以小数点为中心，我们取小数点前或后的值，而日期的进位和截取是以年，月，日，时，分，秒为中心。

- 数据类型的隐式转换

字符串可以转化为数字和日期。
数字要合法，日期要格式匹配。

```
select ename,empno from emp where empno='7900';
```

数字和日期在赋值的时候也可以转为字符串，但在表达式的时候不可以转换。

```
select ename,empno from emp where ename='123';
select ename,empno from emp where ename=123;
```

- 数据类型的显式转换

To_char,to_date,to_number
日期转化为字符串，请说明字符串的格式。

```
select ename,to_char(hiredate,'yyyy/mm/dd') from emp;
```

```
ENAME                TO_CHAR(HIREDATE,'YYYY/MM/DD')
-------------------- ------------------------------
SMITH                1980/12/17
ALLEN                1981/02/20
WARD                 1981/02/22
JONES                1981/04/02
MARTIN               1981/09/28
BLAKE                1981/05/01
CLARK                1981/06/09
KING                 1981/11/17
TURNER               1981/09/08
JAMES                1981/12/03
FORD                 1981/12/03
MILLER               1982/01/23
```

FM 消除前置的零和空格。

```
select ename,to_char(hiredate,'fmyyyy/mm/dd') from emp;
```

```
ENAME                TO_CHAR(HIREDATE,'FMYYYY/MM/DD
-------------------- ------------------------------
SMITH                1980/12/17
ALLEN                1981/2/20
WARD                 1981/2/22
JONES                1981/4/2
MARTIN               1981/9/28
BLAKE                1981/5/1
CLARK                1981/6/9
KING                 1981/11/17
TURNER               1981/9/8
JAMES                1981/12/3
FORD                 1981/12/3
```

```
MILLER               1982/1/23
```

其他格式：year,month,mon,day,dy,am,ddsp,ddspth
格式内加入字符串请双引。

```
 select to_char(hiredate,'fmyyyy "年" mm "月"')  from emp;

SQL> col ss for a6
SQL> select sysdate,to_char(sysdate,'sssss') ss  from dual;

SYSDATE      SS
------------ ------
23-SEP-07    41175
```

当前距离零点的秒数.

```
SQL> select to_char(sysdate,'yyyy year mm month mon dd day dy ddsp ddspth') from dual;

TO_CHAR(SYSDATE,'YYYYYEARMM
-------------------------------------------------------------------------------------------
2007 two thousand seven 10 october   oct 04 thursday  thu four fourth
```

数据类型的显式转换

- 数字转为字符串

格式为 9,0,$,l,.

```
col salary for a30
SQL> select ename,to_char(sal,'9999.000') salary from emp;

ENAME      SALARY
---------- --------------------
SMITH        808.000
ALLEN       1608.000
WARD        1258.000
JONES       2983.000
MARTIN      1258.000
BLAKE       2858.000
CLARK       2458.000
SCOTT       3008.000
KING        5008.000
TURNER      1508.000
ADAMS       1108.000
JAMES        958.000
FORD        3008.000
MILLER      1308.000

SQL>  select ename,to_char(sal,'$00099999000.00' ) salary  from emp;

ENAME      SALARY
---------- --------------------
SMITH       $00000000808.00
ALLEN       $00000001608.00
WARD        $00000001258.00
JONES       $00000002983.00
MARTIN      $00000001258.00
BLAKE       $00000002858.00
CLARK       $00000002458.00
SCOTT       $00000003008.00
KING        $00000005008.00
TURNER      $00000001508.00
ADAMS       $00000001108.00
JAMES       $00000000958.00
```

```
FORD          $00000003008.00
MILLER        $00000001308.00

SQL> select ename,to_char(sal,'l99,999.000') salary from emp;

ENAME      SALARY
---------- ------------------------------
SMITH                $808.000
ALLEN              $1,608.000
WARD               $1,258.000
JONES              $2,983.000
MARTIN             $1,258.000
BLAKE              $2,858.000
CLARK              $2,458.000
SCOTT              $3,008.000
KING               $5,008.000
TURNER             $1,508.000
ADAMS              $1,108.000
JAMES                $958.000
FORD               $3,008.000
MILLER             $1,308.000

SQL> select ename,TO_char(sal,'9G999D99') salary from emp;

ENAME      SALARY
---------- ------------------------------
SMITH          808.00
ALLEN        1,608.00
WARD         1,258.00
JONES        2,983.00
MARTIN       1,258.00
BLAKE        2,858.00
CLARK        2,458.00
SCOTT        3,008.00
KING         5,008.00
TURNER       1,508.00
ADAMS        1,108.00
JAMES          958.00
FORD         3,008.00
MILLER       1,308.00
```

9 是代表有多少宽度，如果不足会显示成######,0 代表强制显示 0，但不会改变你的结果。G 是千分符，D 是小数点。
在数据库中 16 进制的表达是按照字符串来描述的，所以你想将十进制的数转换为十六进制的数使用 to_char 函数。

```
select to_char(321,'xxxxx') from dual;
TO_CHAR(321,'XXXXX')
--------------------
  141
```

其中 xxxxx 的位数要足够，不然报错，你就多写几个，足够大就可以。

数据类型的显式转换

- To_number,to_date

如果你想将十六进制的数转换为十进制的数请使用 to_number 函数。

```
SQL> select to_number('abc32','xxxxxxxx') from dual;

TO_NUMBER('ABC32','XXXXXXXX')
-----------------------------
                       703538
```

```
SELECT TO_DATE('January 15, 1989, 11:00 A.M.', 'Month dd, YYYY, HH:MI A.M.', 'NLS_DATE_LANGUAGE
= American') FROM DUAL;
```

日期是格式和语言敏感的，切记！

```
select TO_NUMBER('100.00', '9G999D99')  from dual;
```

G 为千分符，D 为小数点

RR 和 yy 日期数据类型

```
select to_char(sysdate,'yyyy') "当前",
to_char(to_date('98','yy'),'yyyy') "yy98",
to_char(to_date('08','yy'),'yyyy') "yy08",
to_char(to_date('98','rr'),'yyyy') "rr98",
to_char(to_date('08','rr'),'yyyy') "rr08" from dual;
```

结果为

```
2007  2098  2008  1998  2008
```

yy 是两位来表示年，世纪永远和说话者的当前世纪相同。

RR 比较灵活，它将世纪分为上半世纪和下半世纪。如果你处于上半世纪，描述的是 0-49，那么就和当前世纪相同，描述的是 50-99 就是上世纪。如果你处于下半世纪，描述的是 0-49，那么是下个世纪，描述的是 50-99 就是当前世纪。从而可以看出，RR 的设计完全为了 1990 年到 2010 之间我们的思维习惯而设计的。当我们时间到 2050 前后，使用起来就非常的别扭。

实验 15：操作数据为 null 的函数

该实验的目的是掌握常用的关于 NULL 值操作的函数.

- 综合数据类型函数

NVL (expr1, expr2)

如果 expr1 为非空，就返回 expr1, 如果 expr1 为空返回 expr2，两个表达式的数据类型一定要相同。

NVL2 (expr1, expr2, expr3)

如果 expr1 为非空，就返回 expr2, 如果 expr1 为空返回 expr3

NULLIF (expr1, expr2)

如果 expr1 和 expr2 相同就返回空，否则返回 expr1

COALESCE (expr1, expr2, ..., exprn)

返回括号内第一个非空的值。

```
SQL> select ename,comm,nvl(comm,-1) from emp;

ENAME                COMM NVL(COMM,-1)
-------------------- ----- ------------
SMITH                              -1
ALLEN                 300          300
WARD                  500          500
JONES                              -1
MARTIN               1400         1400
BLAKE                              -1
CLARK                              -1
KING                               -1
TURNER                  0            0
JAMES                              -1
FORD                               -1
MILLER                             -1
```

有奖金就返回奖金，奖金为空就返回-1。

```
select sal+comm ,sal+nvl(comm,0),nvl2(comm,'工资加奖金','纯工资') "收入类别" from emp;
```

有奖金就返回‘工资加奖金‘，奖金为空就返回‘纯工资’。

```
select ename,nullif(ename,'KING') from emp;
ENAME                NULLIF(ENA
-------------------- ----------
SMITH                SMITH
ALLEN                ALLEN
WARD                 WARD
JONES                JONES
MARTIN               MARTIN
BLAKE                BLAKE
CLARK                CLARK
KING
TURNER               TURNER
JAMES                JAMES
FORD                 FORD
MILLER               MILLER
```

如果员工的名称为 king 就返回空，否则返回自己的名字。

```
select COALESCE(comm,sal,100)  "奖金"  from emp;
```

如果有奖金就返回奖金，如果没有奖金就返回工资作为奖金，如果奖金和工资都为空就返回 100，起个别名叫做"奖金"。

实验 16:分支的函数

该实验的目的是掌握分支操作的函数.

- Case 语句

9I 以后才支持的新特性，说叫语句其实是函数。目的是为了分支。

```
CASE expr WHEN comparison_expr1 THEN return_expr1
         [WHEN comparison_expr2 THEN return_expr2
          WHEN comparison_exprn THEN return_exprn
          ELSE else_expr]
END

SELECT ename, job, sal,
       CASE job WHEN 'CLERK'   THEN  1.10*sal
                WHEN 'SALESMAN' THEN  1.15*sal
                WHEN 'ANALYST'   THEN  1.20*sal
       ELSE      sal END     "REVISED_SALARY"
FROM   emp;
ENAME                JOB              SAL REVISED_SALARY
-------------------- --------- ---------- --------------
SMITH                CLERK             800            880
ALLEN                SALESMAN         1600           1840
WARD                 SALESMAN         1250         1437.5
JONES                MANAGER          2975           2975
MARTIN               SALESMAN         1250         1437.5
BLAKE                MANAGER          2850           2850
CLARK                MANAGER          2450           2450
KING                 PRESIDENT        5000           5000
TURNER               SALESMAN         1500           1725
JAMES                CLERK             950           1045
FORD                 ANALYST          3000           3600
MILLER               CLERK            1300           1430
```

Decode 函数，和 CASE 语句一样都是分支语句，但 Decode 函数是 ORACLE 自己定义的，其它数据库可能不支持。

语法如下：

```
DECODE(col|expression, search1, result1
```

```
                    [, search2, result2,...,]
                    [, default])
```

例题：判别 job,不同工作的人赋予不同的工资，除了 CLERK，SALESMAN，ANALYST 以外，其它的人工资不变，将函数的值起一个别名为 REVISED_SALARY。

```
SELECT ename, job, sal,
       decode(job ,'CLERK'     ,   1.10*sal
                  ,'SALESMAN' ,   1.15*sal
                  ,'ANALYST'  ,   1.20*sal
                              ,    sal     )    "REVISED_SALARY"
FROM   emp;
```

ENAME	JOB	SAL	REVISED_SALARY
SMITH	CLERK	800	880
ALLEN	SALESMAN	1600	1840
WARD	SALESMAN	1250	1437.5
JONES	MANAGER	2975	2975
MARTIN	SALESMAN	1250	1437.5
BLAKE	MANAGER	2850	2850
CLARK	MANAGER	2450	2450
KING	PRESIDENT	5000	5000
TURNER	SALESMAN	1500	1725
JAMES	CLERK	950	1045
FORD	ANALYST	3000	3600
MILLER	CLERK	1300	1430

下面的例题是求税率：不同工资上的税率不同。每 2000 一个台阶，8000 以上一律 40%的税。

```
SELECT ename, sal,
       DECODE (TRUNC(sal/2000, 0),
                        0, 0.00,
                        1, 0.09,
                        2, 0.20,
                        3, 0.30
                        , 0.40
                        ) TAX_RATE
FROM   emp;
```

ENAME	SAL	TAX_RATE
SMITH	800	0
ALLEN	1600	0
WARD	1250	0
JONES	2975	.09
MARTIN	1250	0
BLAKE	2850	.09
CLARK	2450	.09
KING	5000	.2
TURNER	1500	0
JAMES	950	0
FORD	3000	.09
MILLER	1300	0

不管 CASE 语句还是 DECODE 函数，他们都是单行函数，每一行都有一个返回值。
从 ORACLE 角度来讲，DECODE 更好，因为各个版本的数据库都支持，横向来说，CASE 语句更好，因为它是国标，不同的数据库间都认可。

实验 17：分组统计函数

该实验的目的是掌握常用的组函数.理解 group by 的操作.

- 组函数

这种函数每次处理多行，给出一个返回值
Avg 平均
Sum 求和
Max 最大
Min 最小
Count 计数

```
select sum(sal),min(sal),max(sal),avg(sal),count(sal) from emp;
SUM(SAL)   MIN(SAL)   MAX(SAL)   AVG(SAL) COUNT(SAL)
-------- ---------- ---------- ---------- ----------
   24925        800       5000 2077.08333         12

select min(hiredate),max(hiredate) from emp;
MIN(HIREDATE)       MAX(HIREDATE)
------------------- -------------------
1980/12/17:00:00:00 1982/01/23:00:00:00
```
日期的小为早，大为晚。

```
select count(*),count(comm) from emp;
COUNT(*) COUNT(COMM)
-------- -----------
      12           4
```
所有组函数，除了 count(*)以外，都忽略 null 值，count 是计数，查看有多少行，count(列)是查看该列有多少非空的行。

```
SQL> select avg(comm),avg(nvl(comm,0)) from emp;
 AVG(COMM) AVG(NVL(COMM,0))
---------- ----------------
       550       183.333333
```
求平均的奖金，奖金为非空的人的平均；和大平均，所有的人参加平均，如果奖金为空，就用零来替代。

```
SQL> select sum(comm),count(comm),count(*)  from emp;
 SUM(COMM) COUNT(COMM)   COUNT(*)
---------- ----------- ----------
      2200           4         12
```
上面的语法验证了组函数忽略 null 值。

```
SQL> select count(distinct deptno) from emp;
COUNT(DISTINCTDEPTNO)
---------------------
                    3
```
计算有多少不同的部门代码的个数。

- Group by 子句

```
SQL> select deptno,sum(sal) from emp   group by deptno;
DEPTNO   SUM(SAL)
------ ----------
    30       9400
    20       6775
    10       8750
```
按照部门号码分组，同组的进行统计。9i 需要排序，10g 不要排序。

```
select sum(sal) from emp group by deptno;
SUM(SAL)
--------
    9400
    6775
    8750
```
分组的列不在 SELECT 列表中，这样写有利于子查询，只列出各个部门的工资总和而不显示部门名称。

```
select deptno,sum(sal) from emp;
select deptno,sum(sal) from emp
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```
这句话不会运行，因为 deptno 要求每行都显示，而 sum 要求多行统计后再显示，违反了原则。
在有组函数的 SELECT 中，不是组函数的列，一定要放在 GROUP BY 子句中。

```
select deptno,job,sum(sal) from emp group by deptno,job;
DEPTNO JOB         SUM(SAL)
------ --------- ----------
    20 CLERK            800
    30 SALESMAN        5600
    20 MANAGER         2975
    30 CLERK            950
    10 PRESIDENT       5000
    30 MANAGER         2850
    10 CLERK           1300
    10 MANAGER         2450
    20 ANALYST         3000
```
多列分组，每列都一样的才放到一起进行统计。30 号部门中有四个销售。合并成一行了。

```
select job,avg(sal) from emp group by job;
JOB         AVG(SAL)
--------- ----------
CLERK     1016.66667
SALESMAN        1400
PRESIDENT       5000
MANAGER   2758.33333
ANALYST         3000
```
要求只显示平均工资大于 2000 的工作。如何过滤掉其它的行？

```
select job,avg(sal) from emp
 where avg(sal)>2000
 group by job;
```
这句话不会运行，WHERE 是条件， avg(sal)是结果。
条件中就使用了结果，违反了因果关系。不但 ORACLE 完成不了您的需求，我们人类的认知阶段就实现不了。

```
select job,avg(sal) from emp  group by job  having avg(sal)>2000;
JOB         AVG(SAL)
--------- ----------
PRESIDENT       5000
MANAGER   2758.33333
ANALYST         3000
```
Having 是在结果中再次筛选。Having 一定得出现在 group by 子句得后面。不能独立存在。

```
select deptno,avg(sal) from emp where job='CLERK' group by deptno having avg(sal)>1000;
```
Where 和 having
可以同时出现再一句话中，起作用的时间不同。

```
SQL> select max(avg(sal)) from emp group by deptno;

MAX(AVG(SAL))
-------------
   2916.66667
```
求各个部门平均工资中的最大的。
组函数的嵌套注意要使用 GROUP BY 子句。

巧用 DECODE 函数，改变排版方式

```
select sum(decode(to_char(hiredate,'yyyy'),'1980',1,0)) "1980",
sum(decode(to_char(hiredate,'yyyy'),'1981',1,0)) "1981",
sum(decode(to_char(hiredate,'yyyy'),'1982',1,0)) "1982",
sum(decode(to_char(hiredate,'yyyy'),'1987',1,0)) "1987",
count(ename) "总人数" from emp;
1980       1981       1982       1987      ×ÜÈËÊý
---- ---------- ---------- ---------- ----------
   1         10          1          0         12
```

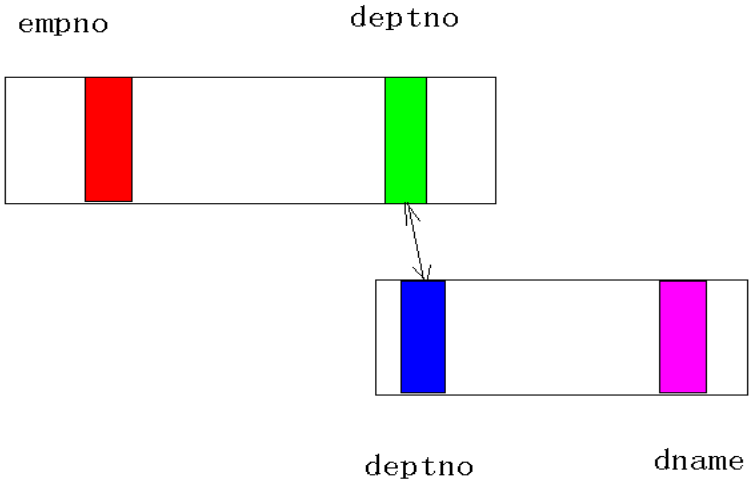
需要掌握的知识点：
1。组函数
2。分组统计
3。NULL 值在组函数中的作用
4。HAVING 的过滤作用
5。组函数的嵌套

实验 18：表的连接查询

该实验的目的是掌握基本的联合查询.

- 表的连接

我们要从多张表中要得到信息，就得以一定的条件将表连接在一起查询。



- Cartesian（笛卡儿）连接

当多张表在一起查询时，没有给定正确的连接条件，结果是第一张表的所有行和第二张表的所有行进行矩阵相乘，得到 n*m 行的结果集。
一般来说笛卡儿连接不是我们需要的结果。
但表如果有一行的情况下，结果有可能正确。

```
select ename,dname from emp,dept;
select ename,dname from emp cross join dept;
ENAME      DNAME
---------- ----------
SMITH      ACCOUNTING
ALLEN      ACCOUNTING
WARD       ACCOUNTING
JONES      ACCOUNTING
MARTIN     ACCOUNTING
BLAKE      ACCOUNTING
CLARK      ACCOUNTING
```

```
KING       ACCOUNTING
TURNER     ACCOUNTING
JAMES      ACCOUNTING
FORD       ACCOUNTING
MILLER     ACCOUNTING
SMITH      RESEARCH
ALLEN      RESEARCH
WARD       RESEARCH
JONES      RESEARCH
MARTIN     RESEARCH
BLAKE      RESEARCH
CLARK      RESEARCH
KING       RESEARCH
TURNER     RESEARCH
JAMES      RESEARCH
FORD       RESEARCH
MILLER     RESEARCH
SMITH      SALES
ALLEN      SALES
WARD       SALES
JONES      SALES
MARTIN     SALES
BLAKE      SALES
CLARK      SALES
KING       SALES
TURNER     SALES
JAMES      SALES
FORD       SALES
MILLER     SALES
SMITH      OPERATIONS
ALLEN      OPERATIONS
WARD       OPERATIONS
JONES      OPERATIONS
MARTIN     OPERATIONS
BLAKE      OPERATIONS
CLARK      OPERATIONS
KING       OPERATIONS
TURNER     OPERATIONS
JAMES      OPERATIONS
FORD       OPERATIONS
MILLER     OPERATIONS

48 rows selected.
```

结果为每个员工在每个部门上了一次班，4*12=48，这并不是我们想得到的结果。
要避免笛卡儿连接一定要给定一个正确的连接条件。

- 等值连接

在连接中给定一个相等的连接条件。

```
select ename,dname from emp,dept
where emp.deptno=dept.deptno;
ENAME      DNAME
---------- ----------
SMITH      RESEARCH
ALLEN      SALES
WARD       SALES
JONES      RESEARCH
MARTIN     SALES
BLAKE      SALES
CLARK      ACCOUNTING
```

```
KING        ACCOUNTING
TURNER      SALES
JAMES       SALES
FORD        RESEARCH
MILLER      ACCOUNTING
```
当列的名称在两张表内重复的时候，要加表的前缀来区分，避免不明确的定义。

- 表的别名

1。便于书写
2。将同名的表区分
3。一旦定义了别名，表的本名就无效
4。只在该语句内有效
5。定义方式为表名后紧跟别名，用空各间隔。

```
select ename,dname  from emp e,dept d where e.deptno=d.deptno;
```

SQL99 的书写方式

```
select ename,dname
 from emp e join dept d
on (e.deptno=d.deptno);
```

效率是相同的，SQL99 是国标

列的别名，为了区分相同的列的名称，这是**别名的本质**。

```
select ename,dname,e.deptno,d.deptno
from emp e,dept d
where e.deptno=d.deptno;
```

```
ENAME      DNAME          DEPTNO DEPTNO
---------- -------------- ------ ------
SMITH      RESEARCH           20     20
ALLEN      SALES              30     30
WARD       SALES              30     30
JONES      RESEARCH           20     20
MARTIN     SALES              30     30
BLAKE      SALES              30     30
CLARK      ACCOUNTING         10     10
KING       ACCOUNTING         10     10
TURNER     SALES              30     30
JAMES      SALES              30     30
FORD       RESEARCH           20     20
MILLER     ACCOUNTING         10     10
```

上述显示有两个列名称都叫 deptno,我们无法区分。

```
select ename,dname,e.deptno "员工表",d.deptno "部门表"
from emp e,dept d
where e.deptno=d.deptno;
```

- 不等连接

连接条件不是一个相等的条件。

```
select ename,sal,grade
from emp,salgrade
where sal between LOSAL and hisal;
```

```
ENAME             SAL      GRADE
---------- ---------- ----------
SMITH             800          1
JAMES             950          1
WARD             1250          2
MARTIN           1250          2
MILLER           1300          2
TURNER           1500          3
```

```
ALLEN           1600          3
CLARK           2450          4
BLAKE           2850          4
JONES           2975          4
FORD            3000          4
KING            5000          5
```

- 外键连接

将一张表有，而另一张表没有的行也显示出来。

```
select ename,dname,emp.deptno from emp,dept
where emp.deptno=dept.deptno;
ENAME      DNAME          DEPTNO
---------- -------------- ------
SMITH      RESEARCH           20
ALLEN      SALES              30
WARD       SALES              30
JONES      RESEARCH           20
MARTIN     SALES              30
BLAKE      SALES              30
CLARK      ACCOUNTING         10
KING       ACCOUNTING         10
TURNER     SALES              30
JAMES      SALES              30
FORD       RESEARCH           20
MILLER     ACCOUNTING         10
```

这句话不会显示 40 号部门，因为 40 部门没有员工。

```
select ename,dname,dept.deptno from emp,dept
where emp.deptno(+)=dept.deptno;

ENAME      DNAME          DEPTNO
---------- -------------- ------
SMITH      RESEARCH           20
ALLEN      SALES              30
WARD       SALES              30
JONES      RESEARCH           20
MARTIN     SALES              30
BLAKE      SALES              30
CLARK      ACCOUNTING         10
KING       ACCOUNTING         10
TURNER     SALES              30
JAMES      SALES              30
FORD       RESEARCH           20
MILLER     ACCOUNTING         10
           OPERATIONS         40
```

+号的意思为将没有员工的部门，用 NULL 来匹配

+号不能同时放在等号的两边，只能出现在一边。

- 自连接

表的一列和同一个表的另一列作为连接的条件。

```
select w.ename "下级" ,m.ename "上级"
from emp w,emp m
where w.mgr=m.empno(+);
下级       上级
FORD       JONES
JAMES      BLAKE
TURNER     BLAKE
MARTIN     BLAKE
WARD       BLAKE
ALLEN      BLAKE
MILLER     CLARK
```

```
CLARK      KING
BLAKE      KING
JONES      KING
SMITH      FORD
KING
```
其中“下级”和“上级”为列的别名。区分相同的列。
W 和 m 为表的别名。区分相同的表。别名的本质。
（+）为了将没有上级的人也显示。

过滤结果
想在结果中过滤去一些内容请用 and 运算。
```
select w.ename "下级" ,m.ename "上级"
from emp w,emp m
where w.mgr=m.empno(+)
and w.deptno=30;
```

实验 19：sql99 规则的表连接操作

该实验的目的是掌握新的 ORACLE 表之间的联合查询语法.

SQL99 规则的书写格式
- Nature(自然)连接

这是 SQL99 规则。
所有同名的列都作为等值条件。
同名的列的数据类型必须匹配。
列的名称前不能加表的前缀。
```
select ename,deptno,dname from emp natural join dept;
ENAME      DEPTNO DNAME
---------- ------ ----------
SMITH          20 RESEARCH
ALLEN          30 SALES
WARD           30 SALES
JONES          20 RESEARCH
MARTIN         30 SALES
BLAKE          30 SALES
CLARK          10 ACCOUNTING
KING           10 ACCOUNTING
TURNER         30 SALES
JAMES          30 SALES
FORD           20 RESEARCH
MILLER         10 ACCOUNTING
```
Using 指定列的连接
当有多列同名，但想用其中某一列作为连接条件时使用。

```
select ename,deptno ,dname
from emp join dept using (deptno);
```

- SQL99 的外键连接

SQL99 写法
```
select ename,dname,dept.deptno
from dept left outer join emp
on(dept.deptno=emp.deptno);
```

9I 前的写法
```
select ename,dname from emp,dept
where emp.deptno(+)=dept.deptno;
```

知识点

1。笛卡儿连接
2。等值连接
3。不等连接
4。外键连接
5。自连接
6。SQL99 的书写格式

练习
1。查询员工的名称(ename)和上班地址（loc）.
2.查询上下级的关系（emp 表）。
3。查询员工的工资级别，只显示 3 级以上的员工名称。
4。查询部门名称 DNAME 和部门的平均工资。

实验 20：子查询

该实验的目的是掌握子查询的语法和概念.

- 子查询

谁的工资最大
1。求出最大工资。
```
Select max(sal) from emp;
5000
```
2。找到最大工资的人。
```
Select ename from emp where sal=5000;
ENAME
-----
KING
```
将两句话写在一起

```
Select ename from emp
where sal=(Select max(sal) from emp);
ENAME
-----
KING
```

- 简单子查询

1。先于主查询执行。
2。主查询调用了子查询的结果。
3。注意列的个数和类型要匹配。
4。子查询返回多行要用多行关系运算操作。
5。子查询要用括号括起来。

查询工资总和高于 10 号部门工资总和的部门。
```
select deptno,sum(sal)
from emp
group by deptno
having sum(sal)>(select sum(sal) from emp where deptno=10);

DEPTNO   SUM(SAL)
------ ----------
    30       9400
```

查询每个部门的最大工资是谁。
```
select deptno,ename,sal
from emp
where (deptno,sal) in (select deptno,max(sal) from emp group by deptno);
DEPTNO ENAME             SAL
------ ---------- ----------
    30 BLAKE            2850
    10 KING             5000
    20 FORD             3000
```
子查询返回多行，用=不可以，得用 in。
子查询返回多列，所以对比的列也要匹配。

Any 和 all 操作
```
SQL> select ename,sal from emp where sal<any(1000,2000);
ENAME             SAL
---------- ----------
SMITH              800
ALLEN             1600
WARD              1250
MARTIN            1250
TURNER            1500
JAMES              950
```

```
MILLER           1300

7 rows selected.
```
小于 2000 就可以
```
SQL> select ename,sal from emp where sal<all(1000,2000);

ENAME             SAL
---------- ----------
SMITH             800
JAMES             950
```
必须小于 1000

小于 all 小于最小，大于 all 大于最大
```
SQL> select ename,sal,deptno from emp
  2  where sal<all(select avg(sal) from emp group by deptno);

ENAME             SAL DEPTNO
---------- ---------- ------
SMITH             800     20
WARD             1250     30
MARTIN           1250     30
TURNER           1500     30
JAMES             950     30
MILLER           1300     10
```
小于 any 小于最大，大于 any 大于最小
```
select ename,sal,deptno from emp
where sal>any(select avg(sal) from emp group by deptno)
ENAME             SAL DEPTNO
---------- ---------- ------
ALLEN            1600     30
JONES            2975     20
BLAKE            2850     30
CLARK            2450     10
KING             5000     10
FORD             3000     20
```

- From 子句中的子查询

查询工资大于本部门平均工资的员工。
```
select ename,e.deptno,sal,asal
from emp e,
(select deptno ,avg(sal) asal from emp group by deptno) a
where e.deptno=a.deptno and sal>asal;
```
A 为视图，为什么要使用别名 asal,因为表达式不能当列的名称，别名的本质使用方法是使非法的合法化。
```
ENAME      DEPTNO        SAL       ASAL
---------- ------ ---------- ----------
BLAKE          30       2850 1566.66667
ALLEN          30       1600 1566.66667
FORD           20       3000 2258.33333
JONES          20       2975 2258.33333
KING           10       5000 2916.66667
```

- 相互关联的子查询

```
select ename,sal ,deptno
from emp o
where sal>(select avg(sal) from emp where deptno=o.deptno) ;
ENAME             SAL DEPTNO
---------- ---------- ------
ALLEN            1600     30
JONES            2975     20
```

```
BLAKE             2850      30
KING              5000      10
FORD              3000      20
```
先运行主查询，得到第一行，将 DEPTNO 传入到子查询，由子查询求出 AVG（SAL），在判定主查询的行是否符合查询的条件。
执行计划是将子查询看作视图的关联。这叫做 SQL 的自动改写。

- Exists 操作

```
select ename,empno,mgr from emp o
where exists(select 3 from emp where mgr=o.empno);
ENAME           EMPNO        MGR
---------- ---------- ----------
FORD              7902       7566
BLAKE             7698       7839
KING              7839
JONES             7566       7839
CLARK             7782       7839
```
找领导，其中 3 是常量，你写什么都可以。
当子查询有行时，Exists 返回 true
查到行后就不再继续查询
当子查询没有行时为假，Exists 返回 false

知识点
简单子查询
多行多列子查询
相互关联子查询
Exists 子句

练习
1。列出没有下级的员工。
2。每个部门工资最少的员工。
3。所有比平均工资高的员工。

DDL 和 DML 语句

- 表的基本操作

表有名称。
表由行和列组成
表是存放数据的最基本对象
我们将一般的表叫做 heap table（堆表），其含义为杂乱无章的存储数据，堆表是数据库的重要组织形式。它有别于索引组织表和 cluster 表。

- 表的名称规则

标准 ASCII 码可以描述
字母开头
30 个字母内
不能是保留字
可以包含大小写字母，数字，_,$,#
不能和所属用户的其它对象重名。千万不要使用汉语做表和列的名称，因为汉语是 ascii 码所不能描述的，ORACLE 的核心是 ASCII 编写的，你使用汉语只是一时痛快，后患无穷。

数据字典

```
Select object_name,object_type from user_objects;
OBJECT_NAME                    OBJECT_TYPE
------------------------------ -----------
DEPT                           TABLE
PK_DEPT                        INDEX
EMP                            TABLE
PK_EMP                         INDEX
BONUS                          TABLE
SALGRADE                       TABLE
```

user_objects 当前用户所拥有的所有对象。不包含你建立的 public 对象。

```
select table_name from user_tables;
TABLE_NAME
----------
DEPT
EMP
BONUS
SALGRADE
```

user_tables 你自己的表，你有一切的权利。你所拥有的表。

```
select * from tab;
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
```

tab 你所拥有的表和视图，显示的较简洁，列较少。

实验 21：建立简单的表，并对表进行简单 ddl 操作

该实验的目的是掌握简单的 ddl 语法.学习建立表,修改表,验证表,删除表

- Create table 语句建立表

要指明表的名称
列的名称
列的数据类型
列的宽度

是否有默认值

常见的数据类型
Char(n)定长
Varchar2(n)变长，最大到4000
Number(p,s)
Date
Long
Lob
raw

```
Create table t1
(name char(8),
Salary number(5) default 0,
Content char(4 char),
Hiredate  date );
```
其中name为8个字节，content为4个字。

描述表结构
```
desc t1
Name                                              Null?    Type
------------------------------------------------- -------- ------------
NAME                                                       CHAR(8)
SALARY                                                     NUMBER(5)
CONTENT                                                    CHAR(4 CHAR)
HIREDATE                                                   DATE
```
验证表
```
Select table_name from user_tables;
TABLE_NAME
----------
DEPT
EMP
BONUS
SALGRADE
T1
select TABLE_NAME,COLUMN_NAME,DATA_TYPE,DATA_LENGTH
from user_tab_columns where table_name='T1';
TABLE_NAME                     COLUMN_NAME          DATA_TYPE      DATA_LENGTH
------------------------------ -------------------- ------------- -----------
T1                             NAME                 CHAR                    8
T1                             SALARY               NUMBER                 22
T1                             CONTENT              CHAR                    8
T1                             HIREDATE             DATE                    7
```

- 在现有表的基础上建立表

```
Create table t2
as select ename name,sal salary from emp;
```
当t2诞生时就会有子查询中所查出的数据。
如果想改变列的名称，请用别名。

如果不想要数据，只建立表结构，请加一个假条件。
```
Create table t3 (c1,c2,c3) as
Select ename,empno,sal from emp where 9=1;
```

- 修改表结构

如果列为null，可以随便修改列的类型和宽度。
如果有数据，修改会受到限制。但不会破坏数据。
如果不改变数据类型，只改变宽度的话加大是可以的。
```
alter table t1 modify(name char(4));
```

```
alter table t2 modify(name char(8));
```

- 修改表的名称

```
 rename t1 to t_1;
```

必须是表的 owner 才可以修改表名称

- 修改列的名称（10g 才可以）

```
 alter table t3 rename column c1 to name;
```

- 表注释

```
comment on table emp is 'employee table';
select COMMENTS from user_tab_comments where table_name='EMP';
```

- 列注释

```
COMMENT ON COLUMN EMP.SAL IS '员工工资';
select COMMENTS  from  user_col_comments
where table_name='EMP' AND column_name='SAL';
```

- 丢弃表

```
SQL>  Select * from tab;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
T1                             TABLE
T2                             TABLE

6 rows selected.

SQL> Drop table t2;

Table dropped.
```

并没有将表真的删除，只是改了名称。

```
Select * from tab;
TNAME                          TABTYPE
------------------------------ -------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
T1                             TABLE
BIN$9WMmfz1CRnqF6c5/hfJmaA==$0 TABLE
```

显示回收站的信息

```
SQL> show recyclebin
ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
T2               BIN$9WMmfz1CRnqF6c5/hfJmaA==$0 TABLE        2007-05-01:17:48:01

SQL> SELECT * FROM USER_RECYCLEBIN;

OBJECT_NAME               ORIGINAL_NAME                    OPERATIO TYPE
------------------------- -------------------------------- -------- -------------------
TS_NAME                        CREATETIME          DROPTIME               DROPSCN
------------------------------ ------------------- ------------------- ----------
PARTITION_NAME                   CAN CAN    RELATED BASE_OBJECT PURGE_OBJECT      SPACE
-------------------------------- --- --- ---------- ----------- ------------ ----------
```

```
BIN$9WMmfz1CRnqF6c5/hfJma T2                                    DROP      TABLE
A==$0
USERS                           2007-05-01:17:47:48 2007-05-01:17:48:01    3166795
                                  YES YES       53883        53883         53883           8
```

- 将回收站的表还原

```
FLASHBACK TABLE t2 TO BEFORE DROP;
```

还原表的同时修改表的名称。

```
FLASHBACK TABLE T2 TO BEFORE DROP RENAME TO TT2;
```

- 清空回收站内指定的表

```
PURGE TABLE T2;
```

清除当前用户的回收站，不会影响其它用户的回收站

```
PURGE   RECYCLEBIN;
```

绕过回收站，彻底的删除表，在 10G 前是没有回收站的，就是彻底的删除。回收站内没有的表是不容易恢复的，我只能取备份来恢复了。

```
Drop table t2   PURGE;
```

知识点
建立表
修改表
注释表
改名称
丢弃表
恢复丢弃的表
初步认识数据字典

- 数据操作语言 dml

Data Manipulation Language (dml)
从无到有　insert into
数据变化　update set
删除数据　delete where
表的融合　merge into

实验 22：dml 语句，插入删除和修改表的数据

该实验的目的是掌握 DML 语法.插入删除和修改表中的数据.

- insert into 语句

```
Insert  into t1(c1,c2,c3) values(v1,v2,v3);
```
要点关键字写全，列的个数和数据类型要匹配。
这种语法每次只能插入一行。
可以使用函数
```
Insert into t1(c2) values(sysdate);
```
将当前的日期插入。

隐式插入 null
在插入中没有列出的列，就会被插入 NULL，如果该列有 DEFAULT 值，那么就插入默认值。
```
Insert into dept(deptno) values(50);
```
其中 dname,loc 没有说明，都为 null.
显式插入 null
明确的写出该列的值为 NULL
```
Insert into dept values(60,null,null);
Insert into dept(loc,dname,deptno) values(null,null,70);
```

日期和字符串
日期格式敏感
当插入的列为日期的时候，最好强制转化为日期类型，默认的转换在环境变化的时候会报错。
```
Insert into t3(hiredate) values(to_date('080599 ','mmddrr');
```
字符串大小写敏感
```
Insert into t2(c1) values('BEIJING ');
Insert into t2(c1) values('beijing ');
```

子查询插入
不加 values 关键字，一次可以插入多行
列的类型和位置要匹配
```
Insert into d1 select * from dept;
Insert into emp2 select * from emp where deptno=30;
```

- Update 语句

修改表中的数据
```
Update emp set sal=sal+1;
Update emp set sal=2000 where empno=7900;
Update emp set comm=null where deptno=30;
```

用子查询来更新
```
connect scott/tiger
drop table emp2 purge;
create table emp2 as select * from emp;
alter table emp2 add(dname varchar2(10));
update emp2 set dname=(select dname from dept where dept.deptno=emp2.deptno);
select * from emp2;
```
将部门号码和部门名称对应起来
结果如下：
```
SQL> connect scott/tiger
```

```
Connected.
SQL> drop table emp2 purge;
drop table emp2 purge
           *
ERROR at line 1:
ORA-00942: table or view does not exist
```
因为 emp2 不存在，所以报错，这是预防下面的语句失败，scott 用户你可以随意操作，不行了就删除重新建立。

```
SQL> create table emp2 as select * from emp;

Table created.
```
建立 EMP2 表，和 EMP 表的结构，数据都相同
```
SQL> alter table emp2 add(dname varchar2(10));

Table altered.
```
将 emp2 表结构修改，增加一列。该列的值为 null
```
SQL> update emp2 set dname=(select dname from dept where dept.deptno=emp2.deptno);

12 rows updated.
```
使用相互关联子查询来更新 emp2 表的 dname 列。
```
SQL> select * from emp2;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DNAME
7369	SMITH	CLERK	7902	17-DEC-80	800		20	RESEARCH
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	SALES
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30	SALES
7566	JONES	MANAGER	7839	02-APR-81	2975		20	RESEARCH
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	SALES
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30	SALES
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10	ACCOUNTING
7839	KING	PRESIDENT		17-NOV-81	5000		10	ACCOUNTING
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30	SALES
7900	JAMES	CLERK	7698	03-DEC-81	950		30	SALES
7902	FORD	ANALYST	7566	03-DEC-81	3000		20	RESEARCH
7934	MILLER	CLERK	7782	23-JAN-82	1300		10	ACCOUNTING

- 使用 default 值进行表的修改

```
connect scott/tiger
drop table t1 purge;
create table t1(c1 number(2) default 10,c2 char(10) default 'bj');
insert into t1(c1)values(20);
select * from t1;
C1 C2
-- --
20 bj
update t1 set c1=default;
select * from t1;
C1 C2
-- --
10 bj
```
如果没有指定 default 的值，那么为 null 值。
```
SQL> update emp2 set EMPNO=default;

12 rows updated.

SQL> select * from emp2;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DNAME
	SMITH	CLERK	7902	17-DEC-80	800		20	RESEARCH
	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	SALES
	WARD	SALESMAN	7698	22-FEB-81	1250	500	30	SALES
	JONES	MANAGER	7839	02-APR-81	2975		20	RESEARCH
	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	SALES
	BLAKE	MANAGER	7839	01-MAY-81	2850		30	SALES
	CLARK	MANAGER	7839	09-JUN-81	2450		10	ACCOUNTING
	KING	PRESIDENT		17-NOV-81	5000		10	ACCOUNTING
	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30	SALES
	JAMES	CLERK	7698	03-DEC-81	950		30	SALES
	FORD	ANALYST	7566	03-DEC-81	3000		20	RESEARCH
	MILLER	CLERK	7782	23-JAN-82	1300		10	ACCOUNTING

- Delete 删除行

```
Delete t1;--所有的行都删除。
Delete  emp2 where sal>2000;
将符合条件的行删除
```

融合语句

```
MERGE INTO table_name table_alias
  USING (table|view|sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
    col1 = col_val1,
    col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

建立实验表，e1 和 e2 表中有重复的人，但工资不同

```
CONN SCOTT/TIGER
DROP TABLE E1;
DROP TABLE E2;
CREATE TABLE E1 AS SELECT EMPNO,ENAME,SAL
FROM EMP WHERE DEPTNO=10;
CREATE TABLE E2 AS SELECT EMPNO,ENAME,SAL
FROM EMP WHERE DEPTNO IN (10,20);
UPDATE E2 SET SAL=SAL+100;
COMMIT;
SQL> select * from e1;
```

EMPNO	ENAME	SAL
7782	CLARK	2450
7839	KING	5000
7934	MILLER	1300

```
SQL> select * from e2;
```

EMPNO	ENAME	SAL
7369	SMITH	900
7566	JONES	3075
7782	CLARK	2550
7839	KING	5100
7902	FORD	3100

```
      7934 MILLER           1400
```
这两张表有同名称的人，但工资不相同。
E1<----E2　　将 e2 融合到 e1 表中
```
merge into E1
USING E2
ON (E1.EMPNO=E2.EMPNO)
WHEN MATCHED THEN
UPDATE SET
E1.SAL=E1.SAL
WHEN NOT MATCHED THEN
INSERT  VALUES(E2.EMPNO,E2.ENAME,E2.SAL);
COMMIT;
SQL> SELECT * FROM E1 order by 1;

     EMPNO ENAME              SAL
---------- ---------- ----------
      7369 SMITH             900
      7566 JONES            3075
      7782 CLARK            2450
      7839 KING             5000
      7902 FORD             3100
      7934 MILLER           1300

6 rows selected.

SQL> SELECT * FROM E2 order by 1;

     EMPNO ENAME              SAL
---------- ---------- ----------
      7369 SMITH             900
      7566 JONES            3075
      7782 CLARK            2550
      7839 KING             5100
      7902 FORD             3100
      7934 MILLER           1400
```
E1 表的前两行是自己的，没有变化，后面的行是 e2 表追加的。
Merge 是 update 和 insert 的结合体，有做 upate ，没有做 insert

实验 23：事务的概念和事务的控制

该实验的目的是了解事务的概念,提交回退和控制事务.

- Transaction 事务的概念

开始：第一个 dml 语句
结束：commit 或者 rollback
未完成的事务可以撤消
未完成的事务，其它会话看不到结果，只能看到已经提交的结果。
维护事务需要锁和回退段的参与

提交事务 commit
1. 手工直接提交 commit
2.自动提交
　　ddl,dcl 语句
　　exit 退出 sqlplus
3.提交后
　事务结束
　释放锁和回退
　其它用户可以看到结果，修改过的结果

撤消事务 rollback
1. 手工直接撤消 rollback
2.自动撤消
    网络或数据库崩溃
     强制退出 sqlplus
3.撤消后
    事务结束
    修改前的数据恢复了
    释放锁和回退
    其它用户可以看到结果，未修改的结果

事务的控制

```
connect scott/tiger
update emp set sal=1000 where deptno=10;
savepoint u10;
update emp set sal=2000 where deptno=20;
savepoint u20;
update emp set sal=3000 where deptno=30;
savepoint u30;
delete emp;
```

事务的控制

```
rollback to savepoint u30;
select ename,sal,deptno from emp order by deptno;
rollback to savepoint u20;
select ename,sal,deptno from emp order by deptno;
rollback to savepoint u10;
select ename,sal,deptno from emp order by deptno;
rollback;
select ename,sal,deptno from emp order by deptno;
```

知识点
1。Dml 语句
2。事务的概念

- 约束(constraint)

保证数据的有效
保证数据的安全
可以自己书写代码
使用触发器
使用 oracle 提供的五类约束

实验 24：在表上建立不同类型的约束

该实验的目的是掌握 oracle 提供的五种约束.

- Not null

定义在表的列上，表明该列必须要有值，不能为 null
可以在建立表的时候说明
也可以在表建立后修改为 not null
可以给约束指定名称。
如果不指定名称，数据库会给一个系统自动指定名称，SYS_C#######
User_constraints,user_cons_columns 可以查看到约束的信息

建立表的时候指定 not null 约束

```
connect scott/tiger
drop table t1 purge;

create table t1 (name char(9) not null,
telenum  char(8) constraint t1_tele_nl not null);
```

建立表的时候指明非空约束，一个是系统命名，一个是我们命名。

```
select CONSTRAINT_NAME,CONSTRAINT_type,TABLE_NAME,column_name
from user_constraints natural join user_cons_columns;
```

```
CONSTRAINT_NAME                C TABLE_NAME                    COLUMN_NAME
------------------------------ - ------------------------------ -----------
PK_DEPT                        P DEPT                           DEPTNO
FK_DEPTNO                      R EMP                            DEPTNO
PK_EMP                         P EMP                            EMPNO
T1_TELE_NL                     C T1                             TELENUM
SYS_C005978                    C T1                             NAME
```

我们通过数据字典来验证我们的实验结果。

建立表后指定 not null 约束。要使用 modify 语法。你不指定名称，数据库自己命名。

```
connect scott/tiger
drop table t1 purge;
create table t1 as select * from dept;
alter table t1 modify(dname   not null);
```

- 唯一约束 UNIQUE

列的值不能重复
可以为 NULL
是用索引来维护唯一的
索引的名称和约束的名称相同

建立表的时候指定 UNIQUE 约束

```
connect scott/tiger
drop table t1 PURGE;

create table t1 (name char(9) UNIQUE,
mail  char(8) constraint t1_mail_u UNIQUE);
```

建立表的时候指明唯一约束，一个是系统命名，一个是我们命名。

```
select CONSTRAINT_NAME,CONSTRAINT_type,TABLE_NAME,column_name
from user_constraints natural join user_cons_columns;
CONSTRAINT_NAME                C TABLE_NAME                    COLUMN_NAME
------------------------------ - ------------------------------ -----------
PK_DEPT                        P DEPT                         DEPTNO
FK_DEPTNO                      R EMP                          DEPTNO
PK_EMP                         P EMP                          EMPNO
SYS_C005980                    U T1                           NAME
T1_MAIL_U                      U T1                           MAIL
```

建立表之后指定 UNIQUE 约束

```
connect scott/tiger
drop table t1 purge;
create table t1 as select * from dept;
alter table t1 add constraint u_dname unique (dname);

select CONSTRAINT_NAME,CONSTRAINT_type,TABLE_NAME,column_name
from user_constraints natural join user_cons_columns;
CONSTRAINT_NAME                C TABLE_NAME                    COLUMN_NAME
------------------------------ - ------------------------------ -----------
PK_DEPT                        P DEPT                         DEPTNO
FK_DEPTNO                      R EMP                          DEPTNO
PK_EMP                         P EMP                          EMPNO
U_DNAME                        U T1                           DNAME

Select table_name,index_name,column_name from user_ind_columns;
TABLE_NAME                     INDEX_NAME                     COLUMN_NAME
------------------------------ ------------------------------ -----------
DEPT                           PK_DEPT                        DEPTNO
EMP                            PK_EMP                         EMPNO
T1                             U_DNAME                        DNAME
```

- Check 检测约束

```
Drop table t1 purge;
Create table t1(name varchar2(8)
check (length(name)>4),
Mail varchar2(10) );

alter table t1 add constraint check_mail check (length(mail)>4);
insert into t1 (name,mail)values('abc','sdsfs');
ERROR at line 1:
ORA-02290: check constraint (SCOTT.SYS_C005983) violated

insert into t1 (name,mail)values('abcsdfs','sds');
ERROR at line 1:
ORA-02290: check constraint (SCOTT.CHECK_MAIL) violated
```

- Primary key 主健约束

一个表只能有一个主健
主健要求唯一并且非空
可以是联合主健，联合主健每列都要求非空
主健能唯一定位一行,所以主键也叫逻辑 rowid
主健不是必需的，可以没有
主健是通过索引实现的
索引的名称和主健名称相同

建立表的时候指定主健，系统命名

```
drop table t1 purge;
create table t1(mail char(8) primary key,name char(8));
```

```
select CONSTRAINT_NAME,CONSTRAINT_type,TABLE_NAME,column_name
from user_constraints natural join user_cons_columns;
CONSTRAINT_NAME                C TABLE_NAME                    COLUMN_NAME
------------------------------ - ------------------------------ -----------
PK_DEPT                        P DEPT                         DEPTNO
FK_DEPTNO                      R EMP                          DEPTNO
PK_EMP                         P EMP                          EMPNO
SYS_C005985                    P T1                           MAIL

Select table_name,index_name,column_name from user_ind_columns;
TABLE_NAME                     INDEX_NAME                     COLUMN_NAME
------------------------------ ------------------------------ -----------
DEPT                           PK_DEPT                        DEPTNO
EMP                            PK_EMP                         EMPNO
T1                             SYS_C005985                    MAIL
```

表建立后指定自命名的主健

```
drop table t1 purge;
create table t1(mail char(8) ,name char(8));
alter table t1 add constraint pk_t1_mail primary key (mail) ;

select CONSTRAINT_NAME,CONSTRAINT_type,TABLE_NAME,column_name
from user_constraints natural join user_cons_columns;

CONSTRAINT_NAME                C TABLE_NAME                    COLUMN_NAME
------------------------------ - ------------------------------ -----------
PK_DEPT                        P DEPT                         DEPTNO
FK_DEPTNO                      R EMP                          DEPTNO
PK_EMP                         P EMP                          EMPNO
PK_T1_MAIL                     P T1                           MAIL

Select table_name,index_name,column_name from user_ind_columns;

TABLE_NAME                     INDEX_NAME                     COLUMN_NAME
------------------------------ ------------------------------ ------------
DEPT                           PK_DEPT                        DEPTNO
EMP                            PK_EMP                         EMPNO
T1                             PK_T1_MAIL                     MAIL
```

- foreign key 外健

指定在表的列上
引用本表其它列，或其它表的其它列
被引用的列得有唯一约束或者主健约束，因为引用的是索引的键值，而不是真正的表。
目的是维护数据的完整性
核心是一列是另外一列的子集，null 除外

建立主健，建立一个外健来引用主健

```
conn scott/tiger
drop table e purge;
drop table d purge;

create table d as select * from dept;
create table e as select * from emp;

alter table d add constraint pk_d primary key  (deptNO) ;
alter table e add constraint fk_e foreign key (deptno)  references d(deptno) ;
```

违反约束

```
delete d where deptno=10
```

ORA-02292: 违反完整约束条件 (SCOTT.FK_E) - 已找到子记录

update e set deptno=50
ORA-02291: 违反完整约束条件 (SCOTT.FK_E) - 未找到父项关键字

建立被级连的外健
alter table e drop constraint FK_E;
alter table e add constraint fk_e foreign key (deptno)
references d(deptno) on delete set null;
父表的值被删除，子表的相关列自动被赋予 NULL 值。
alter table e drop constraint FK_E;
alter table e add constraint fk_e foreign key (deptno)
references d(deptno) on delete cascade;
父表的值被删除，子表的相关行自动被删除。

- 删除约束

任何约束都可以用约束名称来删除
Alter table ### drop constraint ****;
因为主健只能有一个，所以删除主健约束的时候也可以
Alter table ### drop primary key;
如果主健和唯一性约束被删除，自动建立的索引也会同时被清除。
Not Null 约束也可以用 alter table modify 来删除。
删除有外健引用的主健或唯一约束的时候，外健也要被级连删除。
Alter table ### drop primary key cascade;
如果不加 cascade，你删不了，报有外键在使用，不能删除。

按约束的名称来删除约束（可以删除各种约束）
Alter table t1 drop constraint sys_c03033;
非空约束的第二种删除方式
alter table t1 modify (name not null);
alter table t1 modify (name null);

知识点
掌握 oracle 提供的五类约束
建立
查看
删除

- Sequence 序列

序列是一类对象
它可以自动的产生唯一的整数
通常用来产生主健的值
每个用户可以建立多个序列

实验 25：序列的概念和使用

该实验的目的是操作序列,使用序列进行插入操作.

建立和查看序列 s1

```
CREATE SEQUENCE s1
       INCREMENT BY 2
       START WITH 1
       MAXVALUE 10
       MINVALUE -10
       NOCYCLE
       NOCACHE;
Select * from user_sequences;
SEQUENCE_NAME                   MIN_VALUE  MAX_VALUE INCREMENT_BY C O CACHE_SIZE LAST_NUMBER
------------------------------ ---------- ---------- ------------ - - ---------- -----------
S1                                     -10         10            2 N N          0           1
```

引用序列的值

```
Select s1.nextval from dual;取序列 s1 的下一个值。
Select s1.currval from dual; 取序列 s1 的当前值。
Drop table d purge;
Create table d as select * from dept where 0=9;
```

建立一个和 dept 表结构相同，但不含任何行的空表。

```
Insert into d(deptno) values(s1.nextval);
Insert into d(deptno) values(s1.nextval);
Select * from user_sequences;
```

当序列没有值在内存中时，currval 属性无效，先 nextval 后，才会有效。

修改序列

序列的当前值一定得在最大和最小之间
步长不能为零

```
alter SEQUENCE s1  INCREMENT BY -2;
alter SEQUENCE s1 CYCLE;
```

序列值的不连续问题

事物回退，序列号不会退
其它语句引用了该序列

```
Insert into d(deptno) values(s1.nextval);
Rollback;
Insert into d(deptno) values(s1.nextval);
Insert into dept(deptno) values(s1.nextval);
```

Cache 序列的值到内存中
NOCACHE 每次取值都要计算。
CACHE n 一次就放入内存 n 个值。默认值为 20，如果你要连续的使用序列，如定单流水号的产生，请将 N 设大点。如果你不使用序列，而是自己写代码，请注意你的程序效率和竞争锁死情况的发生。
停止数据库后，内存中存放的序列值会丢失
user_sequences 中的 last_number 列代表重新计算的起始值
序列中有一个 order 的选项，在单实例没有差别，它体现在集群 rac 环境中。Order 和 cache 是相互排斥的，一旦你使用了 order 项，会是 cache 项的值无效。Order 的意思为集群中的每个节点想要获得序列的值都要重新从序列的定义中计算获得。确保按照顺序得到序列的值。

删除序列
Drop sequence s1;
一旦被删除，就不可以再引用序列的值。

使用序列的原则如下：
如果你想用做主键，请使用不可循环的序列。
如果想快速发生序列的值，请将 cache 的值加大。
如果序列内的值要很长时间才能使用完，可以考虑使用可以循环的序列。

知识点
序列是共享的对象
主要用来产生主健的。

- View 视图

使用视图为了我们的方便
增加了数据库的负担
视图下降数据库的性能
视图是查询语句的别名
视图的定义存在于数据字典中
User_views 查看视图的定义
视图是好东西，用好了提高开发的效率和安全性，反之数据库的性能极大的下降，数据库是给明白人用的。物有所长，必有所短，万物一理，视图也不例外。Oracle 的所有字典都是视图。极大的提高了管理性，极大的提高了可使用性，我们学习 oracle,就要跟 oracle 学习,他怎么玩，我们就怎么玩，肯定可以成为数据库大师。

- 使用视图的目的

限制对数据库的访问
将复杂的查询包起来，化繁为简
提供给用户独立的数据
在同一个表上建立不同的视图，减少基表的个数。

实验 26：建立和使用视图

该实验的目的是建立简单的视图,了解视图的使用和工作原理.

- 建立和使用视图

如果没有权限，请先授权。在 10G 以前 SCOTT 用户是有 create view 的权限的，10g 以后就没有了，因为默认的角色的权限发生了变化。

```
Conn / as sysdba
Grant create view to scott;
Conn scott/tiger
create view v1
as select deptno,min(sal) salary from emp  group by deptno;
SQL> Desc v1;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ---------
 DEPTNO                                                         NUMBER(2)
 SALARY                                                         NUMBER
```

其中 salary 是 min(sal)的别名，因为函数不能作为列的名称，列的别名的本质用法，将非法的合法化。

```
SQL> Select * from v1;

DEPTNO     SALARY
------ ----------
    30        950
    20        800
    10       1300
```

视图 v1 只是一个定义，没有独立的数据，数据还是存放在 emp 表中。

```
SQL> Select VIEW_NAME, text from user_views;
```

查看视图的定义

```
VIEW_NAME                      TEXT
------------------------------ --------------------------------------------------------
V1                             select deptno,min(sal) salary from emp  group by deptno
```

视图的使用
你使用视图的时候可以把视图当做表
数据库在解释查询语句的时候会查找视图的定义
视图中不存放数据，除了物化视图，物化视图是快照，真正的存放数据，需要刷新。
每次查询的时候都要进行数据的查找

视图的执行过程

```
Select * from v1;
```

查找 user_views 取出 v1 视图的定义。
select deptno,min(sal) salary from emp group by deptno;
数据库执行查找到的定义
所以说视图下降数据库的性能，尤其在复杂的视图相互关联查询的时候，你觉得语法很简单，其实视图内调用了大量的表，容易使我们麻痹大意。

Force 强制建立视图，视图使存在于数据字典中的定义，那么就可以先存在定义，再有基表。
先存在视图的定义，再建立基本表得使用强制的语法。
Drop table t2 purge;
Create force view v2 as select * from t2;
Create table t2 as select * from emp;
查看状态为 INVALID
select object_name,object_type,status from user_objects;
调用视图后再查看状态为 VALID
Select * from v2;
select object_name,object_type,status from user_objects;

修改视图
Create or replace view v1 as select * from dept;
就是将视图的定义替换。

删除视图
Drop view v1;
在数据字典中将视图的定义清除
不影响基本表中的数据

视图上运行 DML
简单的视图可以运行 dml
等于直接操作基本表的数据
但受到一些限制
- Delete 的限制

1。有组函数
2。有 group by 子句
3。用了 distinct 关键字
4。有 rownum 列
Delete v1 where deptno=10;
ORA-01732: 此视图的数据操纵操作非法

- Update 的限制

1。有组函数
2。有 group by 子句
3。用了 distinct 关键字
4。有 rownum 列
5。有表达式的列

- insert 的限制

1。有组函数
2。有 group by 子句
3。用了 distinct 关键字
4。有 rownum 列
5。有表达式的列
6。基本表中有 not null 的列，但该列没有出现在视图的定义里

- WITH CHECK OPTION 选项

```
CREATE OR REPLACE VIEW empvu20
AS SELECT*
   FROM     emp
   WHERE    deptno = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
```

建立视图的时候带有检测约束，约束就是 where 的条件。确保在 update 视图的时候，视图所选择的行不会发生变化，是对视图中数据的一种保障。

```
update empvu20 set deptno=10;
ORA-01402: 视图 WITH CHECK OPTIDN where 子句违规
```

- Read only 选项

```
CREATE OR REPLACE VIEW empvu10
    (employee_number, employee_name, job_title)
AS SELECT empno, ename, job
   FROM     emp
   WHERE    deptno = 10
   WITH READ ONLY;
```

禁止 dml 操作视图

- inline 内嵌式视图

将 from 子句中的子查询起别名

```
select ename,sal from
emp,
(select deptno,avg(sal) salary from emp  group by deptno) a
where emp.deptno=a.deptno
and emp.sal>a.salary;
```

A 就是内嵌式视图
当前语句内起作用
一次性的，在语句外不可引用

Top-n 查询
查询工资的前三名:

```
SELECT ROWNUM as RANK, ename, sal
FROM  (SELECT ename,sal FROM emp
       ORDER BY sal DESC)
WHERE ROWNUM <= 3;
```

From 子句后为视图，内嵌式。

```
SQL> conn scott/tiger
Connected.
SQL> select rownum,ename from emp;
```

按照取出数据的顺序显示,我们使用了 rownum 伪列.

```
    ROWNUM ENAME
---------- ----------
         1 SMITH
         2 ALLEN
         3 WARD
         4 JONES
         5 MARTIN
         6 BLAKE
         7 CLARK
         8 SCOTT
         9 KING
        10 TURNER
        11 ADAMS
        12 JAMES
        13 FORD
        14 MILLER
```

现在我们要查询位置在 8 到 12 的员工.

```
SQL> select rownum#,ename from
  2  (select rownum rownum#,ename from emp order by 1)
  3  where rownum# between 8 and 12;
```

```
   ROWNUM# ENAME
---------- ----------
         8 SCOTT
         9 KING
        10 TURNER
        11 ADAMS
        12 JAMES
```

知识点
掌握视图的原理
建立简单的视图
在视图上进行 dml
内嵌式视图
查看视图的定义
删除视图

- 集合操作

并集 union /union all
交集 INTERSECT
补集 MINUS

实验 27：查询结果的集合操作

该实验的目的是掌握集合操作的语法,对多个结果进行交,并,补集的集合操作.

建立实验表

```
conn scott/tiger
drop table t2 purge;
drop table t1 purge;

create table t1 as select deptno,ename,sal
from emp where deptno in(10,20) order by deptno;
create table t2 as select deptno,ename,sal
from emp where deptno in(20,30) order by deptno;

select * from t1;
DEPTNO ENAME             SAL
------ ---------- ----------
    10 CLARK            2450
    10 KING             5000
    10 MILLER           1300
    20 FORD             3000
    20 JONES            2975
    20 SMITH             800

6 rows selected.

select * from t2;

DEPTNO ENAME             SAL
------ ---------- ----------
    20 FORD             3000
    20 JONES            2975
    20 SMITH             800
    30 TURNER           1500
    30 JAMES             950
    30 BLAKE            2850
    30 WARD             1250
    30 ALLEN            1600
    30 MARTIN           1250
```

这两张表既有相同部分，又有不同。

```
select * from t1
union all
select * from t2;
DEPTNO ENAME             SAL
------ ---------- ----------
    10 CLARK            2450
    10 KING             5000
    10 MILLER           1300
    20 FORD             3000
    20 JONES            2975
    20 SMITH             800
    20 FORD             3000
    20 JONES            2975
```

```
    20 SMITH             800
    30 TURNER           1500
    30 JAMES             950
    30 BLAKE            2850
    30 WARD             1250
    30 ALLEN            1600
    30 MARTIN           1250
```

将 t1 的结果和 t2 的结果联合显示。不排序操作，也不去掉重复的行。

```
select * from t1
union
select * from t2;
DEPTNO ENAME             SAL
------ ---------- ----------
    10 CLARK            2450
    10 KING             5000
    10 MILLER           1300
    20 FORD             3000
    20 JONES            2975
    20 SMITH             800
    30 ALLEN            1600
    30 BLAKE            2850
    30 JAMES             950
    30 MARTIN           1250
    30 TURNER           1500
    30 WARD             1250
```

将 t1 的结果和 t2 的结果联合显示。含有排序操作，也去掉重复的行。

```
select * from t1
INTERSECT
select * from t2;
DEPTNO ENAME             SAL
------ ---------- ----------
    20 FORD             3000
    20 JONES            2975
    20 SMITH             800
```

将 t1 的结果和 t2 的结果的共有部分显示。含有排序操作，也去掉重复的行。

```
select * from t1
MINUS
select * from t2;
DEPTNO ENAME             SAL
------ ---------- ----------
    10 CLARK            2450
    10 KING             5000
    10 MILLER           1300
```

T1 表有，而 t2 表没有的行，去掉重复的行。

```
select * from t2
MINUS
select * from t1;
DEPTNO ENAME             SAL
------ ---------- ----------
    30 ALLEN            1600
    30 BLAKE            2850
    30 JAMES             950
    30 MARTIN           1250
    30 TURNER           1500
```

```
    30 WARD              1250
```
t2 表有，而 t1 表没有的行，去掉重复的行。

实验 28：高级分组 rollup,cube 操作

该实验的目的是掌握高级分组的语法.理解高级分组的工作原理.

- 组函数中的集合操作
- Rollup 分组

按部门分组
```
select deptno,sum(sal) from emp group by deptno;
DEPTNO   SUM(SAL)
------ ----------
    30       9400
    20       6775
    10       8750
```
按部门分组,并求总计
```
select deptno,sum(sal) from emp  group by rollup(deptno);
DEPTNO   SUM(SAL)
------ ----------
    10       8750
    20       6775
    30       9400
            24925
```
Rollup 分组，一次全表扫描

```
select deptno,sum(sal) from emp group by rollup(deptno);
```

分解为下列语句
```
select deptno,sum(sal) from emp group by deptno
union all
select null,sum(sal) from emp
order by 1;
```
两次扫描表，效率低

Group by **Rollup**(a,b,c,d)
的结果集为,共 n+1 个集
```
Group by a,b,c,d
Union all
Group by a,b,c
Union all
Group by a,b
Union all
Group by a
Union all
Group by null
```

```
select deptno,job,sum(sal) from emp group by rollup(deptno,job);
DEPTNO JOB         SUM(SAL)
------ --------- ----------
    10 CLERK           1300
    10 MANAGER         2450
    10 PRESIDENT       5000
    10                 8750
    20 CLERK            800
    20 ANALYST         3000
    20 MANAGER         2975
    20                 6775
```

```
    30 CLERK             950
    30 MANAGER          2850
    30 SALESMAN         5600
    30                  9400
                       24925
```

结果为

```
select deptno,job,sum(sal) from emp group by deptno,job
union all
select deptno,null,sum(sal) from emp group by deptno
union all
select null,null,sum(sal) from emp;
```

Grouping(列名称）的使用，为了表达该列是否参加了分组活动。
0 为该列参加了分组，1 为该列未参加分组操作

```
select deptno,job,grouping(deptno),grouping(job),sum(sal)
from emp group by rollup(deptno,job);
```

DEPTNO	JOB	GROUPING(DEPTNO)	GROUPING(JOB)	SUM(SAL)
10	CLERK	0	0	1300
10	MANAGER	0	0	2450
10	PRESIDENT	0	0	5000
10		0	1	8750
20	CLERK	0	0	800
20	ANALYST	0	0	3000
20	MANAGER	0	0	2975
20		0	1	6775
30	CLERK	0	0	950
30	MANAGER	0	0	2850
30	SALESMAN	0	0	5600
30		0	1	9400
		1	1	24925

- Cube 分组

```
select deptno,job,grouping(deptno),
grouping(job) ,sum(sal) from emp group by cube(deptno,job);
```

结果集为，2**n 个结果集

```
select deptno,job,sum(sal) from emp group by deptno,job
union all
select deptno,null,sum(sal) from emp group by deptno
union all
select null,job,sum(sal) from emp group by job
union all
select null,null,sum(sal) from emp;
```

实验 29：树结构的查询 start with 子句

该实验的目的是会使用 start with 语法来处理层次结构的数据.

层次结构的数据查询

```
select empno,ename,mgr from emp
 start with (ename='SMITH')
 connect by prior mgr=empno;
```

EMPNO	ENAME	MGR
7369	SMITH	7902
7902	FORD	7566
7566	JONES	7839

7839 KING

伪列 level

```
select level,empno,ename,mgr from emp
start with (ename='SMITH')
connect by prior mgr=empno;
```

LEVEL	EMPNO	ENAME	MGR
1	7369	SMITH	7902
2	7902	FORD	7566
3	7566	JONES	7839
4	7839	KING	

从树根开始查询

```
select level,empno,ename,mgr from emp
start with (ename='KING')
connect by prior empno=mgr;
```

LEVEL	EMPNO	ENAME	MGR
1	7839	KING	
2	7566	JONES	7839
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

美化层次关系

```
select lpad('-',level,'-')||ename from emp
start with (ename='KING')
connect by prior empno=mgr;
LPAD('-',LEVEL,'-')||ENAME
--------------------------------
-KING
--JONES
---FORD
----SMITH
--BLAKE
---ALLEN
---WARD
---MARTIN
---TURNER
---JAMES
--CLARK
---MILLER
```

删除节点，下级保留

```
select level,empno,ename,mgr from emp
where ename<>'BLAKE'
start with (ename='KING')
connect by prior empno=mgr;
```

BLAKE 一个人删除，不影响他的下属

删除枝干
```
select level,empno,ename,mgr from emp
start with (ename='KING')
connect by prior empno=mgr
and ename<>'BLAKE';
```
BLAKE 和他的整个部门

实验 30：高级 dml 操作

该实验的目的是掌握高级 dml 操作语法.将一个查询结果插入到多张表.

Insert 的进一步学习

将一张表的数据分别插入到多张表中
建立实验表 e1,e2
```
drop table e1 purge;
drop table e2 purge;
create table e1 as select ename,sal,hiredate
from emp where 9=0;
create table e2 as select ename,deptno,mgr
from emp where 9=0;

insert all
into e1 values(ename,sal,hiredate)
into e2 values(ename,deptno,mgr)
select ename,sal,hiredate,deptno,mgr
from emp where deptno=10;

select * from e1;
select * from e2;
```
All 的含义为：emp 表中的一行将插入到 e1,e2 中

```
insert first
when sal>3000 then
into e1 values(ename,sal,hiredate)
when sal>2000 then
into e2 values(ename,deptno,mgr)
select ename,sal,hiredate,deptno,mgr
from emp ;
```
First 的含义为：一行只能给一张表，即使两个表的条件都符合

第二部分 pl/sql 基础

匿名块的编写

- 块的结构和声明变量

模块的组成
DECLARE
变量声明部分，可以没有
Begin
逻辑处理执行部分，到 end 结束，必须有
EXCEPTION
错误处理部分，可以没有
End;
块是我们 PL/SQL 的基石，我们的程序都是通过块组成，没有名称的块叫匿名块，完成一定的功能。

为什么要使用 pl/sql
便于维护（模块化）
提高数据安全性和完整性（通过程序操作数据）
提高性能（编译好的）
简化代码（反复调用）

实验 31：书写一个最简单的块，运行并查看结果

该实验的目的是掌握简单的 pl/sql 语法.执行一个最简单的匿名块.

书写一个最简单的块，将字符串输出到屏幕。使用的是 sqlplus .
输出 Hello world
先设定 SQLPLUS 的环境变量，如果不指定默认值为不输出，设定后用 show 来验证。

```
set serveroutput on
show serveroutput

begin
dbms_output.put_line('------------------输出------------------');
dbms_output.put_line('hello world');
dbms_output.put_line('------------------结束------------------');
end;
/
```

每句话以分号结束，最后加上 /

```
set serveroutput on
begin
dbms_output.put_line('------------------输出------------------');
dbms_output.put_line('hello world');
dbms_output.put_line('------------------结束------------------');
end;
/
```

将上面存储为文件 c:\bk\out.txt

```
Sql>@ c:\sql\out.txt
```

@文件名称代表运行该文件，最好是指定绝对路径，如果只给文件名称，将使用相对路径。

- 使用变量

变量用来存储数据

操作存储的数据
可以重复应用
使维护工作简单

语法
identifier [CONSTANT] datatype [NOT NULL]
[:= | DEFAULT expr];
[ ]内为可选项
每行定义一个变量
在 declare 部分声明

实验 32：在块中操作变量

该实验的目的是掌握在 pl/sql 块中操作变量.

```
DECLARE
v_hiredate DATE;
v_deptno NUMBER(2) NOT NULL := 10;
v_location VARCHAR2(13) := 'Atlanta';
c_comm CONSTANT NUMBER := 1400;
v_valid BOOLEAN NOT NULL := TRUE;
```

Not null 一定要给初值
CONSTANT 也一定要给值
:= 为赋值，=为逻辑判断，判断是否相等。

变量的命名规则
在不同的模块中，变量可以重名
变量的名称不应该和模块中引用的列的名称相同
变量名称应该有一定的可读性

- %TYPE 属性

声明一个变量和某列数据类型相同
声明一个变量和另外一个变量数据类型一致
减小程序的无效的可能性，可以不知道列的数据类型，定义一个与之相同的变量。

```
v_name emp.ename%TYPE;
v_balance  NUMBER(7,2);
v_min_balance  v_balance%TYPE := 10;
```

在 begin end 间是执行部分，是一个模块的必须部分。

```
--是行注释
/*   */是多行注释
DECLARE
v_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
monthly salary input from the user */
v_sal := :g_monthly_sal * 12;
END; -- This is the end of the block
```

变量的作用范围
外部模块变量可以传到内部模块
内部模块的变量不会影响外部

实验 33：在块中操作表的数据

该实验的目的是掌握在 pl/sql 块中操作数据库中的表.select into 将表中的数据放入到变量

- 取表中的数据

```
declare
v1 emp.ename%type;
v2 emp.sal%type;
begin
select ename,sal into v1,v2
from emp where empno=7900;
dbms_output.put_line(v1);
dbms_output.put_line(v2);
end;
/
```

一定要有 into
一次只能操作一行，操作多行得用循环
变量类型和个数要匹配

Dml 语句和 sql 相同
使用隐式游标的属性来控制 dml，有四种隐式的游标。
SQL%ROWCOUNT
SQL%FOUND
SQL%NOTFOUND
SQL%ISOPEN

```
declare
v1 emp.deptno%type :=20;
v2 number;
begin
delete emp where deptno=v1;
v2:=sql%rowcount;
dbms_output.put_line('delete rows :');
dbms_output.put_line(v2);
rollback;
end;
/
```

实验 34：在块中的分支操作 if 语句

该实验的目的是掌握在 pl/sql 块中使用 if 语句进行分支操作.

- 分支

```
IF-THEN-END IF
IF-THEN-ELSE-END IF
IF-THEN-ELSIF-END IF

IF condition THEN
statements;
[ELSIF condition THEN
statements;]
[ELSE
statements;]
END IF;
```

分支就是树的结构，条件就是分支的选择，我们只能走到一个支干上，即使每个条件都符合，我们也只能操作一个支干的语句。

```
IF UPPER(v_last_name) ='SCOTT' THEN
v_mgr := 102;
END IF;

DECLARE
v1 DATE := to_date('12-11-1990','mm-dd-yyyy');
v2 BOOLEAN;
BEGIN
IF MONTHS_BETWEEN(SYSDATE,v1) > 5 THEN
v2 := TRUE;
dbms_output.put_line('true');
ELSE
v2 := FALSE;
dbms_output.put_line('false');
END IF;
end;
/
```

- Case 语句

```
DECLARE
v1 CHAR(1) := UPPER('&v1');
v2 VARCHAR2(20);
BEGIN
v2 :=CASE  v1
WHEN 'A' THEN 'Excellent'
WHEN 'B' THEN 'Very Good'
WHEN 'C' THEN 'Good'
ELSE 'No such grade'
END;
DBMS_OUTPUT.PUT_LINE ('级别为: '|| v1 || '是 ' || v2);
END;
/
```

Null 的逻辑运算真值表
True and null 结果为 null
Flase and null 结果为 flase

实验 35：在块中使用循环，三种循环模式

该实验的目的是掌握在 pl/sql 块中使用循环操作.

- 基本循环 loop

```
Drop table t1 purge;
create table t1 (c1 number(2));
Select * from t1;
```

建立实验表 t1，我们将想 t1 表中加入数据。Loop 循环必须含有退出的条件，而且该条件一定要每次循环都要变化，如果没有变化就是死循环，死循环的结果就是 cpu 总是 100%，你可以重新启动数据库来消除死循环。

```
Declare
V1 number(2) :=1;
Begin
Loop
Insert into t1 values(v1);
v1:=v1+1;
Exit When v1>10 ;
End loop;
```

```
End;
/
```

- While 循环，先判定条件，每次循环时条件都要变化，如果不变化就是死循环。

```
Declare
V1 number(2) :=1;
Begin
While v1<10 Loop
Insert into t1 values(v1);
v1:=v1+1;
End loop;
End;
/
```

- For 循环，pl/sql 中的最常见的循环，是和游标操作的绝配。方便而直观。

```
begin
for v1 in 1..9 loop
Insert into t1 values(v1);
end loop;
end;
/

begin
for v1 in REVERSE 1..9 loop
Insert into t1 values(v1);
end loop;
end;
/
```

For 循环特点
步长为 1
计数器不要声明，自动声明
对计数器只能引用。不能做赋值操作
计数器的数据类型和上下界的数据类型相同
计数器只能在循环体内引用
Oracle 数据库中有两个引擎,sql 引擎和 pl/sql 引擎,我们在 pl/sql 的模块中调用了 sql 语句,数据库就要在两个引擎中来回的切换,如果我们使用循环来处理 sql 语句的话,就会造成频繁的在两个引擎中进行切换.为了避免这样的情况发生,我们最好是将要传递的值放入到复合变量中,一次传递更多的数据,这个技术叫做**批量绑定**.

- 含有成员的变量

PL/SQL RECORDs
PL/SQL Collections
典型的应用，存储表的一行

自定义记录结构

```
TYPE type_name IS RECORD
(field_declaration[, field_declaration]...);--声明数据类型
Identifier  type_name;--声明变量
```

非常象高级语言的结构类型变量
内部成员叫做域（field）
使用时要指明 变量名.域名

实验 36：在块中自定义数据类型，使用复合变量

该实验的目的是掌握在 pl/sql 块中使用复合变量.

自定义数据类型

```
TYPE emp_record_type IS RECORD
(last_name VARCHAR2(25),
job_id VARCHAR2(10),
salary NUMBER(8,2));
emp_record   emp_record_type;
```

- %rowtype 记录结构

前缀为表的名称
内部域的属性为表中列的数据类型
域的名称为列的名称
便于存储表的一行

```
V1  emp%rowtype;
V2  dept%rowtype;
```

使用记录

```
set serveroutput on
declare
v1 dept%rowtype;
begin
select * into v1 from dept where rownum=1;
dbms_output.put_line(v1.deptno);
dbms_output.put_line(v1.dname);
dbms_output.put_line(v1.loc);
end;
/
```

- Pl/sql 表（集合），表面上看象数组，但不是，它更象一个带有主键的表，我们通过主键来访问数据。

含有两要素：
主键，数据类型为 BINARY_INTEGER
成员，可以为简单变量，也可以为记录复合变量

```
TYPE type_name IS TABLE OF
{column_type | variable%TYPE
| table.column%TYPE} [NOT NULL]
| table.%ROWTYPE
[INDEX BY BINARY_INTEGER];
identifier type_name;
```

```
DECLARE
TYPE t1 IS TABLE OF  emp.ename%TYPE
INDEX BY BINARY_INTEGER;
TYPE t2 IS TABLE OF DATE  INDEX BY BINARY_INTEGER;
v1 t1;
v2 t2;
BEGIN
v1(1) := 'CAMERON';
v2(8) := SYSDATE + 7;
select ename,hiredate into v1(7900),v2(7900) from emp where empno=7900;
dbms_output.put_line(v1(1)||' '||v1(7900));
dbms_output.put_line(v2(8)||' '||v2(7900));
END;
/
```

- 使用集合的属性来操作集合的数据

– NEXT
– TRIM
– DELETE
– EXISTS
– COUNT

```
- FIRST and LAST
- PRIOR

DECLARE
TYPE t2 IS TABLE OF dept%rowtype INDEX BY BINARY_INTEGER;
v2 t2;
n1 number(3);n2 number(3);n3 number(3);n4 number(3);n5 number(3);n6 number(3);
BEGIN
select * into v2(10) from dept where deptno=10;
select * into v2(20) from dept where deptno=20;
select * into v2(30) from dept where deptno=30;
select * into v2(40) from dept where deptno=40;
n1:=v2.first;--第一号元素
n2:=v2.last;--最后一个元素
n3:=v2.count;--共有多少个元素
n4:=v2.PRIOR(20);--20 号的前一个为几号
n5:=v2.NEXT(20);--20 号的后一个为几号
dbms_output.put_line(n1||'...'||n2||'...'||n3||'...'||n4||'...'||n5);
v2.delete(30);--将 30 号删除，不写几号为都删除
n6:=v2.count;
dbms_output.put_line(n6);
END;
/

使用属性
DECLARE
TYPE t1 IS TABLE OF emp.ename%TYPE
INDEX BY BINARY_INTEGER;
TYPE t2 IS TABLE OF DATE INDEX BY BINARY_INTEGER;
v1 t1;
v2 t2;

BEGIN
v1(1) := 'CAMERON';
v2(8) := SYSDATE + 7;
select ename,hiredate into v1(7900),v2(7900) from emp where empno=7900;
for i in 1..10000 loop
if v1.exists(i) then
dbms_output.put_line('v1('||i||')= '||v1(i));
end if;
if v2.exists(i) then
dbms_output.put_line('v2('||i||')= '||v2(i));
end if;
end loop;
END;
/

成员为复合变量，每个主键访问一行数据
DECLARE
TYPE t1 IS TABLE OF emp%rowtype
INDEX BY BINARY_INTEGER;
TYPE t2 IS TABLE OF dept%rowtype INDEX BY BINARY_INTEGER;
v1 t1;
v2 t2;
BEGIN
select * into v1(7900) from emp where empno=7900;
select * into v2(10) from dept where deptno=10;
dbms_output.put_line(v1(7900).empno||v1(7900).ename);
dbms_output.put_line(v2(10).dname);
```

```
END;
/
```

实验 37：在块中使用自定义游标

该实验的目的是掌握在 pl/sql 块中操作游标.

- Cursor 游标

在 declare 部分声明
在 begin 中操作
用来操作表的每一行
分为自定义和系统定义两类
系统定义 sql%rowcount 等

游标的操作
定义 declare
打开 open
抓取 fetch，每次只能抓取一行，使用循环可以处理表的每一行。
关闭 close

```
DECLARE
CURSOR c1 is select ename,sal from emp order by sal desc;
v1 c1%rowtype;
BEGIN
open c1;
fetch c1 into v1;
dbms_output.put_line(v1.ename||v1.sal);
close c1;
END;
/
```

游标的属性，前缀为游标的名称
%isopen，测试该游标是否打开，返回真或假
%rowcount，游标已经操作了多少行， 返回数值
%found，游标是否找到记录，返回真或假
%notfound，游标是否找到记录，返回真或假

- 游标属性

```
DECLARE
CURSOR c1 is select ename,sal from emp order by sal desc;
v1 c1%rowtype;
n1 number(2);
BEGIN
if not c1%isopen then
open c1;
end if;
fetch c1 into v1;
n1:=c1%rowcount;
dbms_output.put_line(v1.ename||' '||v1.sal||' '||n1);
close c1;
END;
/
```

循环控制

```
DECLARE
CURSOR c1 is select ename,sal from emp order by sal desc;
v1 c1%rowtype;
```

```
n1 number(2);
BEGIN
open c1;
loop fetch c1 into v1;
exit when c1%notfound;
dbms_output.put_line(v1.ename||' '||v1.sal);
n1:=c1%rowcount; end loop; close c1;
dbms_output.put_line(n1);
END;
/
```

For 循环

```
DECLARE
CURSOR c1 is select ename,sal from emp order by sal desc;
n1 number(2);
BEGIN
for v1 in c1 loop
dbms_output.put_line(v1.ename||' '||v1.sal);
n1:=c1%rowcount;
end loop;
dbms_output.put_line(n1);
END;
/
```

V1 的数据类型为 c1%rowtype,c1 自动 open,自动 fetch,自动 close，for 循环和游标的结合可以很方便的处理游标内的每一行。

带变量的游标，每次打开游标的时候需要给定变量。根据变量的不同，游标的内容将不同。一般用于多层循环中内层循环的游标控制。

```
DECLARE
CURSOR c1(n1 number) is select ename,sal from emp where empno=n1;
v1 c1%rowtype;
BEGIN
open c1(7900);
fetch c1 into v1;
dbms_output.put_line(v1.ename||' '||v1.sal);
close c1;
END;
/
```

准备实验环境，建立一个表，其中一个列是空的，我们要将空列的值赋予相对应的部门名称。

```
conn scott/tiger
drop table t1 purge;
create table t1 as select  ename,deptno from emp;
alter table t1 add(dname varchar2(18));
select * from t1;
```

For update 游标，所更改的行既是当前游标所指定的行。

```
DECLARE
CURSOR c1 is select * from t1 for update;
v1 dept.dname%type;
BEGIN
for n1 in c1 loop
select dname into v1 from dept where deptno=n1.deptno;
update t1 set dname=v1 WHERE CURRENT OF C1;
end loop;
END;
/
```

- Pl/sql 错误的处理

错误是指 begin end 运行模块中出现的错误
错误有三类
Oracle 预先定义了几十种错误
Oracle 没有定义的几万种错误
程序员自己报的错误

错误的产生
Oracle 自定义的错误会自动的发生
我们自己定义的错误要我们自己报错

错误的处理
可以在模块中处理
内部模块会向外部模块传递错误
模块内处理不了的错误会向外传递

模块内处理，在 exception 部分
```
Declare
--声明变量部分的错误不会被处理
Begin
--只有运行部分的错误才会被处理
…
Exception
--处理不了的错误会向外传递
..
End;
```

实验 38：在块中处理错误 exception

该实验的目的是掌握在 pl/sql 块中捕获并处理错误.

处理 oracle 预先定义的错误
```
set serveroutput on
declare
v1 emp.sal%type;
begin
select sal into v1 from emp;
exception
when TOO_MANY_ROWS then
dbms_output.put_line('more person ');
end;
/
```

预先定义的错误，抛砖引玉，教我们如何自定义几万个错误
获得数据库预先定义的错误描述和错误号码
```
SQL> select text from dba_source where name='STANDARD'
  2  AND ROWNUM<100 AND TEXT LIKE '%EXCEPTION_INIT%';

TEXT
------------------------------------------------------------
    pragma EXCEPTION_INIT(CURSOR_ALREADY_OPEN, '-6511');
    pragma EXCEPTION_INIT(DUP_VAL_ON_INDEX, '-0001');
    pragma EXCEPTION_INIT(TIMEOUT_ON_RESOURCE, '-0051');
    pragma EXCEPTION_INIT(INVALID_CURSOR, '-1001');
    pragma EXCEPTION_INIT(NOT_LOGGED_ON, '-1012');
    pragma EXCEPTION_INIT(LOGIN_DENIED, '-1017');
    pragma EXCEPTION_INIT(NO_DATA_FOUND, 100);
    pragma EXCEPTION_INIT(ZERO_DIVIDE, '-1476');
    pragma EXCEPTION_INIT(INVALID_NUMBER, '-1722');
```

```
    pragma EXCEPTION_INIT(TOO_MANY_ROWS, '-1422');
    pragma EXCEPTION_INIT(STORAGE_ERROR, '-6500');
    pragma EXCEPTION_INIT(PROGRAM_ERROR, '-6501');
    pragma EXCEPTION_INIT(VALUE_ERROR, '-6502');
    pragma EXCEPTION_INIT(ACCESS_INTO_NULL, '-6530');
    pragma EXCEPTION_INIT(COLLECTION_IS_NULL , '-6531');
    pragma EXCEPTION_INIT(SUBSCRIPT_OUTSIDE_LIMIT,'-6532');
    pragma EXCEPTION_INIT(SUBSCRIPT_BEYOND_COUNT ,'-6533');
    pragma EXCEPTION_INIT(ROWTYPE_MISMATCH, '-6504');
    PRAGMA EXCEPTION_INIT(SYS_INVALID_ROWID, '-1410');
    pragma EXCEPTION_INIT(SELF_IS_NULL, '-30625');
    pragma EXCEPTION_INIT(CASE_NOT_FOUND, '-6592');
    pragma EXCEPTION_INIT(USERENV_COMMITSCN_ERROR, '-1725');
    pragma EXCEPTION_INIT(NO_DATA_NEEDED, '-6548');
    pragma EXCEPTION_INIT(INVALID_USERENV_PARAMETER, -2003);
    pragma EXCEPTION_INIT(ICD_UNABLE_TO_COMPUTE, -6594);

25 rows selected.
```

处理非预定义错误

```
declare
ttt   exception;                                    --定义错误
pragma exception_init(ttt,-2292);  --绑定错误号码
begin
delete dept;                  --发生错误
dbms_output.put_line('ok');
exception
when TOO_MANY_ROWS then
dbms_output.put_line('more person ');
when NO_DATA_FOUND then dbms_output.put_line('no person ');
when ttt then  dbms_output.put_line(‘forign key  '); --捕获错误
end;
/
```

自定义错误号码

```
declare
my_errors EXCEPTION;  --定义错误
PRAGMA  EXCEPTION_INIT(my_errors, -20001);--分配错误号
BEGIN
raise_application_error(-20001,‘工资不能被改动');--手工发生
EXCEPTION
WHEN my_errors then    --捕获错误
dbms_output.put_line('工资不能被改动');
end;
/
```

捕获错误代码和错误描述

```
declare
ttt   exception;
pragma exception_init(ttt,-2292);
begin
update emp set deptno=90;
dbms_output.put_line('ok');
exception
when TOO_MANY_ROWS then
dbms_output.put_line('more person ');
when NO_DATA_FOUND then
dbms_output.put_line('no person ');
when ttt then
```

```
dbms_output.put_line('forign key  ');
when others then
dbms_output.put_line(sqlcode||';'||sqlerrm);
end;
/
```

错误的传递
错误会传递到调用它的模块或环境

编写程序

实验 39：触发器

该实验的目的是编写不同类型的触发器.

- 触发器

构建实验表

```
connect scott/tiger
drop table d purge;
drop table e  purge;
create table d as select * from dept;
create table e as select * from emp;
Select * from d;
Select * from e;
```

感受触发器

```
CREATE or replace TRIGGER d_update
AFTER delete or UPDATE OF deptno ON d
FOR EACH ROW
BEGIN  --当 D 表的部门号修改的时候 E 表的对应部门号也相应的修改
IF  (UPDATING AND :old.deptno != :new.deptno)
THEN UPDATE e
SET deptno = :new.deptno
WHERE deptno = :old.deptno;
END IF;
--当 D 表的某个部门号删除的时候，E 表的对应部门同时被删除
if deleting then
delete e where deptno=:old.deptno;
end if;
END;
/

--验证触发器的状态
select trigger_name,status from user_triggers;
--改变触发器的状态
--禁用某个触发器
ALTER TRIGGER  d_update  disable;
--禁用某个表上的所有触发器
alter table d disable all triggers;
--删除触发器
DROP TRIGGER d_update;
```

验证 d_update 的功能

```
update d set deptno=50 where deptno=30;
select * from e;
select * from d;
```

```
delete d where deptno=20;
select * from e;
select * from d;
Commit;
```

触发器的类型
行级触发 FOR EACH ROW
　　影响的每一行都会执行触发器
语句级触发
　　默认的模式，一句话才执行一次触发器
触发器不能嵌套，不能含有事物控制语句

何时触发
Before
在条件运行前，执行触发器

After
在条件运行后，执行触发器

INSTEAD OF
替代触发，作用在视图上

禁止对表 E 的 SAL 列进行修改

```
create or replace trigger e_update
before update of sal on e
begin
if updating then
raise_application_error(-20001,'工资不能被改动');
end if;
end;
/
```

保存老值和新的值

```
CONNECT SCOTT/TIGER
DROP TABLE T1;
CREATE TABLE T1 AS SELECT SAL OLD_VALUE,SAL NEW_VALUE FROM EMP WHERE 0=9;
CREATE OR REPLACE TRIGGER TRG1
BEFORE INSERT OR UPDATE OF sal ON emp
FOR EACH ROW
BEGIN
INSERT INTO T1 VALUES(:OLD.SAL,:NEW.SAL);
END;
/
SELECT * FROM T1;
update emp set sal=sal+1;
commit;
select * from t1;
```

建立一个不可**通过视图**来改基表的视图 V1

```
drop table e1;
create table e1 as select * from emp;
drop view v1;
create view v1 as
select distinct deptno from e1;
--试图修改 V1 时报错
update v1 set deptno=50 where deptno=10;
```

建立一个**替代触发器**，当修改 V1 的时候会自动的修改基表

```
create or replace trigger trigger_instead_of
instead of insert or update or delete on v1
```

```
for each row
begin
if updating then
update e1 set deptno=:new.deptno where deptno=:old.deptno;
end if;
end;
/
```

建立一个登录的审计触发器

```
conn scott/tiger
drop table login_table;
create table login_table(user_id varchar2(15),log_date date,action varchar2(15));

--on schema 方式为只记录当前的用户行为
CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON SCHEMA
BEGIN
INSERT INTO login_table(user_id, log_date, action)
VALUES (USER, SYSDATE, 'Logging on');
END;
/

CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON SCHEMA
BEGIN
INSERT INTO login_table(user_id, log_date, action)
VALUES (USER, SYSDATE, 'Logging off');
END;
/

conn scott/tiger
conn hr/hr
conn scott/tiger
select user_id, to_char(log_date,'yyyy/mm/dd:hh24:mi:ss') log_date, action from login_table;
--删除触发器
drop trigger logon_trig;
drop trigger logoff_trig;
```

使用触发器的目的
维护数据库的完整性
通过视图改基表
审计数据库的操作

实验 40：编写函数

该实验的目的是编写自己的函数,调用自己编写的函数.

- 函数

函数是有名称的 pl/sql 块
函数有返回值
在表达式中调用函数
存储在服务器端

```
CREATE OR REPLACE FUNCTION get_sal
     (v_id  IN   emp.empno%TYPE)
    RETURN NUMBER
    IS
      v_salary     emp.sal%TYPE :=0;
```

```
    BEGIN
      SELECT sal INTO  v_salary   FROM  emp  WHERE  empno = v_id;
     RETURN (v_salary);
  END get_sal;
/
```

验证对象
```
select object_name,object_type from user_objects;
```
查看原程序
```
select text from user_source;
```
调用函数
```
select get_sal(7839) from dual;
```
删除函数
```
DROP FUNCTION get_salary;
```

加密函数
将建立函数的文本存储为文件 c:\bk\1.txt
```
C:\bk> set nls_lang=american_america.us7ascii
C:\bk>wrap iname=c:\bk\1.txt oname=c:\bk\2.txt
SQL> @c:\bk\2.txt
select text from user_source;
```

建立索引用的函数
要使基于函数的索引被使用要先收集统计信息
DETERMINISTIC（确定性）要在函数的定义中指明
（意思为输入的值相同时函数的返回值也相同，如随机数发生函数,日期函数就不符合）
函数中不能含有集合函数
Or 运算时不会使用函数索引
```
QUERY_REWRITE_ENABLED=TRUE
QUERY_REWRITE_INTEGRITY=TRUSTED

create or replace function f_sal
(v1 in number)
return number deterministic
as
begin
if v1<1000 then return 1;
elsif v1<2000 then return 2;
else return 3;
end if;
end;
/
create index i1 on  emp(f_sal(sal));
select * from emp where f_sal(sal)=1;
```

实验 41：编写存储过程

该实验的目的是编写存储过程,调用存储过程.

- 存储过程

存储在服务器端
编译好的
可以在程序中调用
完成一定功能
可以没有返回值，也可以有多个返回值

参数导入型的存储过程(in)
```
CREATE OR REPLACE PROCEDURE p1
```

```
   (v_id in emp.empno%TYPE)
   IS
   BEGIN
     UPDATE emp
     SET    sal = sal +1
     WHERE  empno = v_id;
      commit;
   END p1;
/

--验证原程序
select text from user_source where NAME   ='P1';--大写
select OBJECT_NAME,OBJECT_TYPE,STATUS from user_objects
where OBJECT_TYPE='PROCEDURE';
EXECUTE P1 (7369); --单独运行过程
Begin  --在模块中调用
P1(7900);
P1 (7902);
P1 (7839);
end;
/

参数导入和导出型的存储过程(in/out)
CREATE OR REPLACE PROCEDURE query_emp
    (v_id          IN    emp.empno%TYPE,
     v_name      OUT     emp.ename%TYPE,
     v_salary      OUT   emp.sal%TYPE,
     v_comm      OUT     emp.comm%TYPE)
    IS
    BEGIN
      SELECT   ename, sal, comm
      INTO     v_name, v_salary, v_comm
      FROM     emp
     WHERE    empno = v_id;
   END query_emp;
/

传变量到调用的模块
declare
v1 emp.ename%TYPE;
v2 emp.sal%TYPE;
v3 emp.comm%TYPE;
begin
query_emp(7654,v1,v2,v3);
dbms_output.put_line(v1);
dbms_output.put_line(v2);
dbms_output.put_line(v3);
end;
/

参数导入和导出共用变量型的存储过程(inout)
CREATE OR REPLACE PROCEDURE format_phone
    (v_phone_no IN OUT VARCHAR2)
   IS
    BEGIN
     v_phone_no := '(' || SUBSTR(v_phone_no,1,3) ||
                   ')' || SUBSTR(v_phone_no,4,3) ||
                   '-' || SUBSTR(v_phone_no,7);
    END format_phone;
```

```
/

传变量到调用的模块
declare
v1 varchar2(20);
begin
v1:='010456789';
format_phone(v1);
dbms_output.put_line(v1);
end;
/
```

实验 42：编写包 package

该实验的目的是编写包,掌握包的特性.

- 包 package

```
将功能相近的函数或存储过程组织在一起
便于管理
包内的函数可以重名,提高程序的通用性
减少对象的名称占用问题
一个包内函数使用，整个包都调入内存
包内一个程序失效，整个包重新编译
由包头和包体组成

包头
不能加密
描述了包内的函数，存储过程的参数
可以独立存在
包体
可以加密
函数的实现
不能独立存在

建立包头
create or replace package PK87 is
Function F1 (NO NUMBER) return NUMBER;
Function F1 (NO EMP.ENAME%TYPE) return NUMBER;
PROCEDURE P1(V_NO NUMBER);
end PK87;
/

建立包体
create or replace package body PK87 is
-- Function and procedure implementations
FUNCTION F1
(NO IN NUMBER)
RETURN NUMBER
IS
v_salary emp.sal%TYPE :=0;
BEGIN
SELECT sal INTO v_salary FROM emp WHERE empNO = NO;
RETURN v_salary;
END F1;

FUNCTION F1
(NO EMP.ENAME%TYPE)
RETURN NUMBER
```

```
IS
v_salary emp.sal%TYPE :=0;
BEGIN
SELECT sal INTO v_salary FROM emp WHERE eNAME = NO;
RETURN v_salary;
END F1;

procedure P1(V_NO in NUMBER)
is
begin
   UPDATE EMP SET SAL=SAL+1 WHERE EMPNO=V_NO;
COMMIT;
end P1;
end PK87;
/
```

验证和调用包内函数

```
SELECT TEXT FROM USER_SOURCE WHERE NAME='PK87';

Desc pk87

SELECT PK87.F1(7900),PK87.F1('KING') FROM DUAL;

DECLARE
V1 EMP.SAL%TYPE;
V2 EMP.ENAME%TYPE;
BEGIN
V1:=PK87.F1(7900);
V2:=PK87.F1('KING');
pk87.P1(7902);
DBMS_OUTPUT.PUT_LINE(V1);
DBMS_OUTPUT.PUT_LINE(V2);
END;
/
```

函数和包等的相互依存关系

存在公共同意词，而后又存在相应的同名称的表

```
conn / as sysdba
drop public synonym eee;
create public synonym eee for hr.employees;

conn hr/hr
grant select on employees to scott;

conn scott/tiger
drop view v1;
create view v1 as select last_name from eee where rownum<5;
select * from v1;

create table eee as select * from hr.employees where rownum<10;
--对象失效
select object_name,status from user_objects;
select * from v1;
--自动得编译，有效了
select object_name,status from user_objects;
```

加密包体

```
1.set nls_lang=american_amrica.us7ascii
2.wrap iname=1.sql
```

sql>@1.plb
其中 1.sql 中是包体的原程序,加密以后你可以指定加密文件的名称,如果不指明就是默认为原文件名称,扩展名为 plb.在 oracle_home\rdbms\admin\目录下有很多的加密包体,那是给 dbms_****包的,我们在建立数据库的时候会调用其中的部分包.

第三部分数据库的体系结构

数据库产品
不同的产品存在于不同的 oracle_home，可以在同一台主机上存在 8i,9i,10g,ias 等多个数据库产品。
产品是安装的，大家都相同，放之四海皆准。
不同的是数据库，千差万别。
数据库是我们安装完产品后建立的
一套产品可以建立多个数据库，每个数据库是独立的。

Oracle 数据库产品
1993 oracle7
1997 oracle8
1999 oracle8i(815,816,8174)
2001 oracle9i(9.0,9.2)
2004 oracle10g(10.1,10.2)

一套产品，建立多个数据库
各个数据库间是独立的，每个数据库都有自己的全套相关文件，有各自的密码文件，参数文件，数据文件，控制文件和日志文件。

分布式数据库
单机数据库
主备方式的数据库，同一时刻只能一台主机可以读取数据库
RAC 集群数据库，多节点同时访问一个数据库

数据库由一些物理文件组成
我们的表存储在数据库中
数据库不能直接读取
我们通过实例（instance)来访问数据库

实例的维护

实例由内存和后台进程组成
实例是访问数据库的方法
初始化参数控制实例的行为
一个实例只能连接一个数据库

启动实例不需要数据库
产品安装好
有初始化参数文件
就可以启动实例
与是否存在数据库无关

实例内的内存叫 sga
System Global Area (SGA)
也可以理解为 shared global area
查看 SGA
进入高级帐号
```
Conn / as sysdba
 select * from v$sga;
Show sga
```

Sga 是全局共享的

大小由初始化参数文件控制

后台进程是实例和数据库的联系纽带
分为核心进程和非核心进程
当前后台进程的查看

```
select NAME,DESCRIPTION from v$bgprocess where paddr<>'00';
NAME                                     DESCRIPTION
---------------------------------------- ----------------------------
PMON                                     process cleanup
PSP0                                     process spawner 0
MMAN                                     Memory Manager
DBW0                                     db writer process 0
ARC0                                     Archival Process 0
ARC1                                     Archival Process 1
ARC2                                     Archival Process 2
ARC3                                     Archival Process 3
ARC4                                     Archival Process 4
ARC5                                     Archival Process 5
ARC6                                     Archival Process 6
ARC7                                     Archival Process 7
ARC8                                     Archival Process 8
ARC9                                     Archival Process 9
ARCa                                     Archival Process 10
LGWR                                     Redo etc.
CKPT                                     checkpoint
SMON                                     System Monitor Process
RECO                                     distributed recovery
CJQ0                                     Job Queue Coordinator
QMNC                                     AQ Coordinator
MMON                                     Manageability Monitor Process
MMNL                                     Manageability Monitor Process 2
```

核心进程，必须存在，有一个终止，所有数据库进程全部终止

非核心进程
完成数据库的额外功能
归档
调度作业
共享 server
Database writer (DBWn)数据库写进程
将数据库的变化写入到文件
最多 20 个
DBW0-DBW9　　DBWa-DBWj
应该和 cpu 的个数对应
由参数 DB_WRITER_PROCESSES 描述
因为 dbwr 是**哪里来**的数据写**回到哪里**，所以可以多个进程一起工作。

Log writer (LGWR)日志写进程
将日志缓冲写入到磁盘的日志文件
只有一个,因为日志写是顺序写，所以一个就可以了，因为是顺序写所以也**不能为多个**。

Checkpoint (CKPT)检查点进程
存盘点
触发 dbwn，写脏数据块
更新数据文件头，更新控制文件

System monitor (SMON)系统监测进程
实例崩溃时进行自动恢复
清除作废的排序临时段

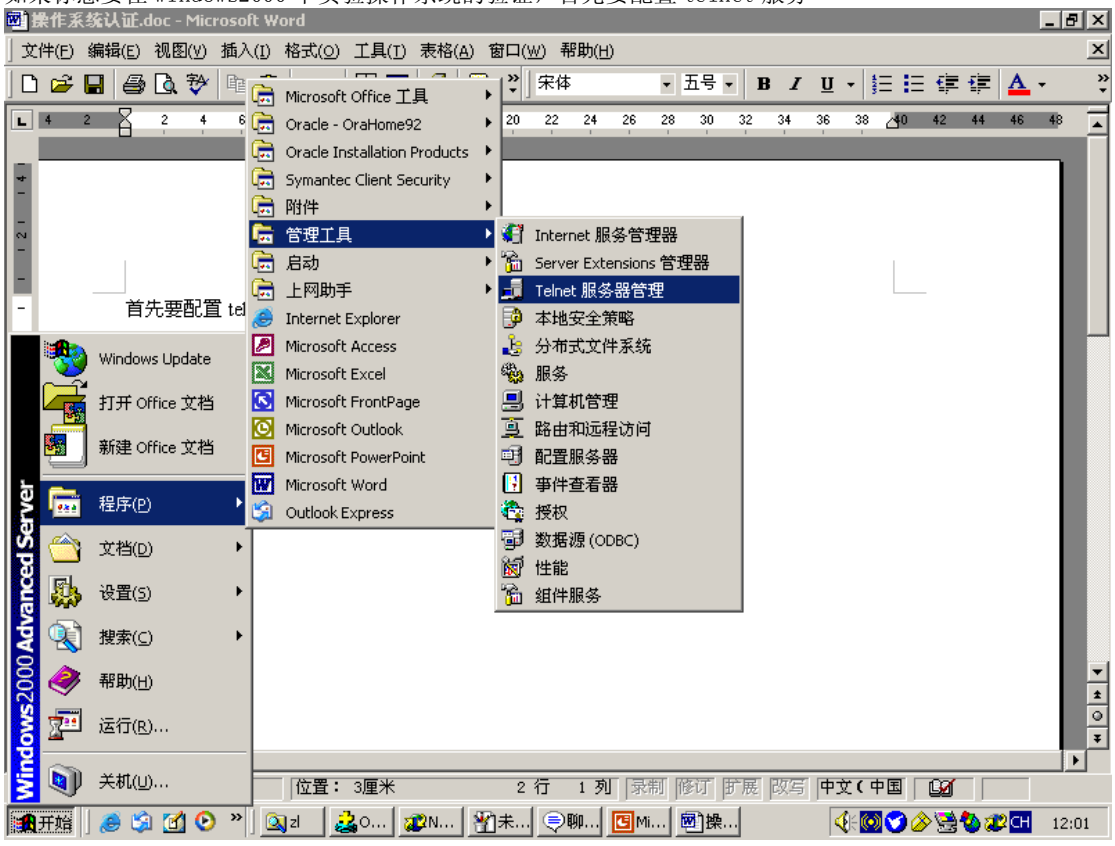
Process monitor (PMON)进程监测进程
清除死进程
重新启动部分进程
监听的自动注册

我们连接到数据库其实是连接到实例
这个过程叫建立一个会话

实验 43：数据库的最高帐号 sys 的操作系统认证模式

该实验的目的是进入数据库的最高帐号 sys.掌握操作系统认证的两个条件.
操作系统认证
因为数据库是在 OS 上的软件
能进入 ORACLE 帐号,就可以进入到数据库的最高帐号
Conn / as sysdba
Show user
无论数据库处于何种状态,sys 用户总可以进入到数据库
因为 sys 是外部操作系统认证的

如果你想要在 windows2000 下实验操作系统的验证，首先要配置 telnet 服务



Microsoft (R) Windows 2000 (TM) (内部版本号 2195)
Telnet Server Admin (Build 5.00.99201.1)

请在下列选项中选择一个:

0) 退出这个应用程序
1) 列出当前用户
2) 结束一个用户的会话 ...

3) 显示 / 更改注册表设置 ...
4) 开始服务
5) 停止服务

请键入一个选项的号码 [0 - 5] 以选择该选项: 3

请在下列选项中选择一个:

0) 退出这个菜单
1) AllowTrustedDomain
2) AltKeyMapping
3) DefaultDomain
4) DefaultShell
5) LoginScript
6) MaxFailedLogins
7) NTLM
8) TelnetPort
请键入一个选项的号码 [0 - 8] 以选择该选项: 7
NTLM 的当前值 = 2
您确实想更改这个值吗 ? [y/n]y
NTLM [当前值 = 2; 可接受的值 0、1 或 2] :1
您确实想将 NTLM 设置为 : 1 ? [y/n]y

只有当 Telnet 服务重新开始后设置才会生效

请在下列选项中选择一个:

0) 退出这个菜单
1) AllowTrustedDomain
2) AltKeyMapping
3) DefaultDomain
4) DefaultShell
5) LoginScript
6) MaxFailedLogins
7) NTLM
8) TelnetPort
请键入一个选项的号码 [0 - 8] 以选择该选项: 0
然后一直选零，退出配置。
以上是配置 windows2000 的 telnet 设置.如果是 xp 操作系统不用配置,配置的目的是可以选择我们自己的用户来 telnet 登录操作系统.这是第一步骤. 注意：服务中的 telnet 项要启动。

第二：建立一个自己的用户 z3，属于管理员组，而不属于 dba 组。
第三：进入 DOS，打 telnet 后**选择 n**,用户为你新建立的用户 z3。
第四：在自己用户下打 sqlplus /nolog,目的是以 z3 身份进入 sqlplus.

第五：conn / as sysdba 应该进不去。报权限不足.
第六：给 z3 的操作系统用户让它属于 oracle 管理员的组。Windows 下为 ora_dba,其它操作系统为 dba

请进入到 oracle_home\network\admin
纯文本方式打开 sqlnet.ora 文件
#SQLNET.AUTHENTICATION_SERVICES = (NTS)
注释掉该行后操作系统认证就不起作用了.其中第一列#为注释.
屏蔽掉 nt 的操作系统认证,仅对 windows 操作系统系列有用,其它操作系统没有用.
总结：操作系统认证的两条件。一、操作系统的用户要属于 dba 组；二、和数据库间的连接是安全的。

实验 44：数据库的最高帐号 sys 的密码文件认证模式

该实验的目的是使用密码文件的认证方式进入到最高 sys 帐号,如何建立和维护密码文件.
在远程,或者操作系统认证不可以使用的情况下,请使用密码文件来认证 sys 用户
在 windows 下
密码文件路径 oracle_home\database
密码文件名称 pwd+sid.ora
在 unix 下
密码文件路径 oracle_home/dbs
密码文件名称 pwd+sid
Sid 为实例名称,查看实例名称
Select instance_name from v$instance;
select 'pwd'||instance_name||'.ora' from v$instance;

密码文件必须存在,即使你以操作系统认证,因为参数 remote_login_passwordfile 默认的值是要使用密码文件的，除非你将 remote_login_passwordfile 的值改为 none,这样就禁止了密码文件的使用，你想进入到 sys 用户必须使用操作系统认证模式。
密码文件丢失必须重新建立
Orapwd 为 oracle 的命令，用于建立密码文件，命令的格式如下
Orapwd file=…. Password=….
密码文件中含有 sys 用户的密码

建立密码文件的步骤

1.确定实例的名称
2.确定密码文件的路径和名称
3.停止数据库,删除老的密码文件
4.在操作系统下运行
orapwd file=d:\oracle\102\database\pwdora10.ora password=hello
其中 ora10 为实例的名称，hello 为密码，是 sys 用户的密码
5.连接的 sys
Conn sys/hello as sysdba
显示为连接的空闲实例，因为数据库还没有启动。
但这并没有证明你使用了密码文件.
SQL> conn sys/addas as sysdba
Connected.
SQL> conn asfdsf/adaf as sysdba
Connected.
SQL> conn / as sysdba
Connected.
SQL> conn sys/hello as sysoper
Connected.
SQL> conn sys/adsssd as sysoper
ERROR:
ORA-01031: 权限不足
原因很简单,因为操作系统认证的**优先级高**于密码文件.所以你只要写 as sysdba 就可以进入老大,但 sysoper 不能使用操作系统来认证,它只能使用密码文件认证,上面的实验证明 hello 是正确的密码.

6.启动数据库 startup
建立密码文件要重新启动数据库,因为内存中保留有原来的密码.
初始化参数 remote_login_passwordfile =none 则数据库设置为禁止使用密码文件,只能使用操作系统认证登录到最高的老大用户.即使你以密码认证连接到数据库,也不能启动和停止数据库,报权限不足.

实验 45：数据库的两种初始化参数文件

该实验的目的是认识参数文件,两类参数文件的相互转换.如何修改参数.
初始化参数文件是描述实例的行为的文件,文件大小很小.

初始化参数文件在 windows 操作系统的 oracle_home\database 目录下,有纯文本和二进制两种形式.
初始化参数文件在其它操作系统中存在于 oracle_home/dbs 目录下.
纯文本参数文件,修改参数的时候直接编辑文件,再保存就可以了.
Initsid.ora
二进制参数文件,必须存在于服务器端.使用命令来修改.
Spfilesid.ora

Server parameter file

纯文本参数文件和二进制参数文件的差别
1。修改参数的方式不同
2。优先级不同
3。是否动态存储修改的参数
4。存在的位置不同
　　纯文本可以存在于客户端
　　二进制文件一定存在于 server 端
5.rman 可以备份二进制参数文件,不能备份纯文本参数文件.
验证现在数据库使用的参数文件类型,我们一定要知道我们使用的是什么类型的参数文件,涉及到我们如何修改参数的手段.
select distinct ISSPECIFIED from v$spparameter;
如果含有 true 就是使用二进制参数文件
如果只有 false 就是使用的纯文本参数文件
SQL> select distinct ISSPECIFIED from v$spparameter;

ISSPECIFIED

TRUE
FALSE
因为上面的选择有 true,所以这个数据库使用的是二进制参数文件.我们修改参数要使用命令而不是编辑文件,千万不要编辑二进制参数文件,你编辑以后会报 ora-00600 的错误.
SQL> select   ISSPECIFIED ,count(*) from v$spparameter group by ISSPECIFIED;

ISSPECIFIED   COUNT(*)
------------ ----------
TRUE   50
FALSE   213
上面的查询表示有 50 个参数存在于二进制参数文件,213 个参数为默认值.

两类参数文件的相互转换
Create pfile from spfile;
Create spfile from pfile;
上面的命令在连接的 sys 就可以使用,而不必启动数据库.当我们转换不了的时候,请将数据库停止,再转换,再重新启动数据库,再验证.

参数文件的优先级
Spfilesid.ora
Spfile.ora
Initsid.ora
SQL> col value for a40
SQL> select name,value from v$spparameter where ISSPECIFIED='TRUE';
查看二进制参数文件内的参数设置.
NAME   VALUE
------------------------------------ --------------------------------
processes   150
trace_enabled   FALSE
nls_language   SIMPLIFIED CHINESE
nls_territory   CHINA
sga_target   167772160
control_files   D:\ORACLE\ORADATA\ORA10\CONTROL01.

```
control_files                          D:\ORACLE\ORADATA\ORA10\CONTROL02.
control_files                          D:\ORACLE\ORADATA\ORA10\CONTROL03.
db_file_name_convert                   ORA10
db_file_name_convert                   ORA10
log_file_name_convert                  ORA10
log_file_name_convert                  ORA10
db_block_size                          8192
compatible                             10.2.0.1.0
log_archive_config                     DG_CONFIG=(ORA10,ORASB)
log_archive_dest_1                     location=c:\arc
log_archive_dest_2                     location=c:\bk\ MANDATORY
log_archive_dest_state_1               ENABLE
log_archive_dest_state_2               ALTERNATE
log_archive_max_processes              10
log_archive_format                     %t_%s_%r.arc
fal_client                             157
fal_server                             11
db_file_multiblock_read_count          16
db_recovery_file_dest                  c:\bk
db_recovery_file_dest_size             314572800
standby_file_management                AUTO
_allow_resetlogs_corruption            TRUE
log_checkpoints_to_alert               TRUE
undo_management                        AUTO
undo_tablespace                        UNDOTBS1
undo_retention                         1800
recyclebin                             OFF
O7_DICTIONARY_ACCESSIBILITY            TRUE
remote_login_passwordfile              EXCLUSIVE
audit_sys_operations                   FALSE
db_domain
service_names                          ORA10
local_listener
utl_file_dir                           c:\bk
job_queue_processes                    10
cursor_sharing                         SIMILAR
audit_file_dest                        d:\oracle/admin/ora10/adump
background_dump_dest                   d:\oracle/admin/ora10/bdump
user_dump_dest                         d:\oracle/admin/ora10/udump
core_dump_dest                         d:\oracle/admin/ora10/cdump
audit_trail                            NONE
db_name                                ORA10
open_cursors                           300
pga_aggregate_target                   25165824
_awr_flush_threshold_metrics           FALSE
```

我们修改参数有三个选项

```
SQL> show parameter pga_aggregate_target

NAME                                 TYPE                           VALUE
------------------------------------ ------------------------------ -----
pga_aggregate_target                 big integer                    24M
SQL> alter system set pga_aggregate_target=30m scope=memory;
```

只修改内存的值,不改变参数文件的设置,下回再次启动数据库时值还是老的,能修改的前提是该参数可以动态修改,如果是静态参数只能使用下面的方法.

```
System altered.

SQL> alter system set pga_aggregate_target=30m scope=spfile;
```

只修改二进制文件,而不修改内存,静态参数只能先改文件再重新启动数据库.

```
System altered.
```

SQL> alter system set pga_aggregate_target=30m scope=both;
同时修改二进制文件和内存,该参数必须是可以动态修改的.
System altered.

SQL> alter system set pga_aggregate_target=30m;
如果没有指明修改哪里,默认为参数文件和内存同时修改,默认就是 both.
System altered.
我们可以从参数文件中删除一个参数,当然你也可以先转化为纯文本再转化为二进制参数文件.
alter system reset trace_enabled scope=spfile sid='*';
select name,value from v$spparameter where ISSPECIFIED='TRUE' order by 1;
验证一下,果然少了一行,下回启动后该参数就按默认值来处理.
二进制参数文件在修改的时候有的时候会报错误,我们可以先该文本文件后再建立二进制参数文件.我估计是 bug,我们原谅它了,但确实对我们的学习造成一定的困惑.如果你认为参数配置的没有问题,但就是不让修改,那就先改纯文本,再变为二进制.总的来说二进制的参数文件好于纯文本,你到底选择哪种类型的参数文件都没有关系.

实验 46：启动数据库的三个台阶 nomount,mount,open

该实验的目的是细化启动数据库的三个步骤,彻底的明白还要等到学习完冷备份之后.
启动数据库到 nomount 状态的条件如下.如果你是非 windows 操作系统就没有注册表,而有环境变量.
服务中的 OracleService 必须启动
服务的名称和注册表中的 oracle_sid 相匹配
存在正确的密码文件和参数文件
有足够的内存
参数文件中描述的路径必须存在
数据库产品软件安装正确

conn sys/sys as sysdba
Shutdown abort;
Startup nomount;
select instance_name,status from v$instance;
启动数据库到第一个台阶 nomount 状态做了如下的工作.
1.读参数文件
2.分配内存
3.启动后台进程
4.初始化部分 v$视图

将数据库带到 mount 状态
select value from v$spparameter where name='control_files';
Alter database mount;
Mount 数据库的过程是读参数文件中描述的控制文件,校验控制文件的正确性,将控制文件的内容读入到内存,mount 是挂接的意思,是操作系统中的概念.一旦 mount 之后,就是将一个没有意义的实例和一个数据库发生了联系.因为实例是空壳.没有任何数据库和该实例发生关系,我们可以理解为实例是水泵,放到哪个水塘里就会抽取哪里的数据,实例是通用的.mount 的意思是将一个通用的水泵放入到指定的水塘.mount 是读控制文件,控制文件中有数据文件和日志文件的信息.
select instance_name,status from v$instance;

打开数据库
Alter database open;
读控制文件中描述的数据文件
验证数据文件的一致性,如果不一致,使用日志文件将数据库文件恢复到一致的状态.
数据库 open 后,普通用户才可以访问数据库
用户的表才为可见

只读方式 open 数据库

Alter database open read only;
select OPEN_MODE from v$database;
默认的 open 方式为 read write
想改 read only 为 read write 必须重新启动数据库
我们现在回想一下数据库启动的三个台阶,我们先读的是参数文件,参数文件可以有我们来编写.读完参数文件后又读了控制文件,控制文件描述了数据文件和日志文件的信息,如果控制文件丢失可以重新建立,最后是读数据文件.数据文件里才存放了我们的数据.数据库将启动分为三个台阶,目的是我们可以准确的知道哪里有问题,迅速的排除.有点象老鼠拖木钎,大头在后面.由最开始的一个 1k 的参数文件,最后到几个 t 的大型数据库.当我们只打 startup 而不加任何参数的时候.默认是到 open,等于 startup open;
SQL> startup
ORACLE instance started.

Total System Global Area 167772160 bytes
Fixed Size   1247900 bytes
Variable Size   75498852 bytes
Database Buffers   88080384 bytes
Redo Buffers   2945024 bytes
Database mounted.
Database opened.
我们从屏幕显示的结果可以清楚的看出,有三个台阶.
还有一个命令是 startup force 强制启动数据库,等于强制停止数据库再启动数据库.

实验 47：停止数据库的四种模式

该实验的目的是区分不同的停止数据库的方式.
四种停止数据库的方式各不相同,用于不同的情况,一般我们采用 shutdown immediate 方式停止数据库,下面是每种停止数据库方式的差别.
Shutdown NORMAL
Shutdown TRANSACTIONAL
Shutdown IMMEDIATE
Shutdown abort

Shutdown NORMAL
新的会话不接受
等待非活动的会话结束
等待事物结束
产生检查点
停止数据库

Shutdown TRANSACTIONAL
新的会话不接受
不等待非活动的会话结束
等待事物结束
产生检查点
停止数据库

Shutdown immediate
新的会话不接受
不等待非活动的会话结束
不等待事物结束
产生检查点
停止数据库

Shutdown abort
新的会话不接受
不等待非活动的会话结束
不等待事物结束

不产生检查点
停止数据库

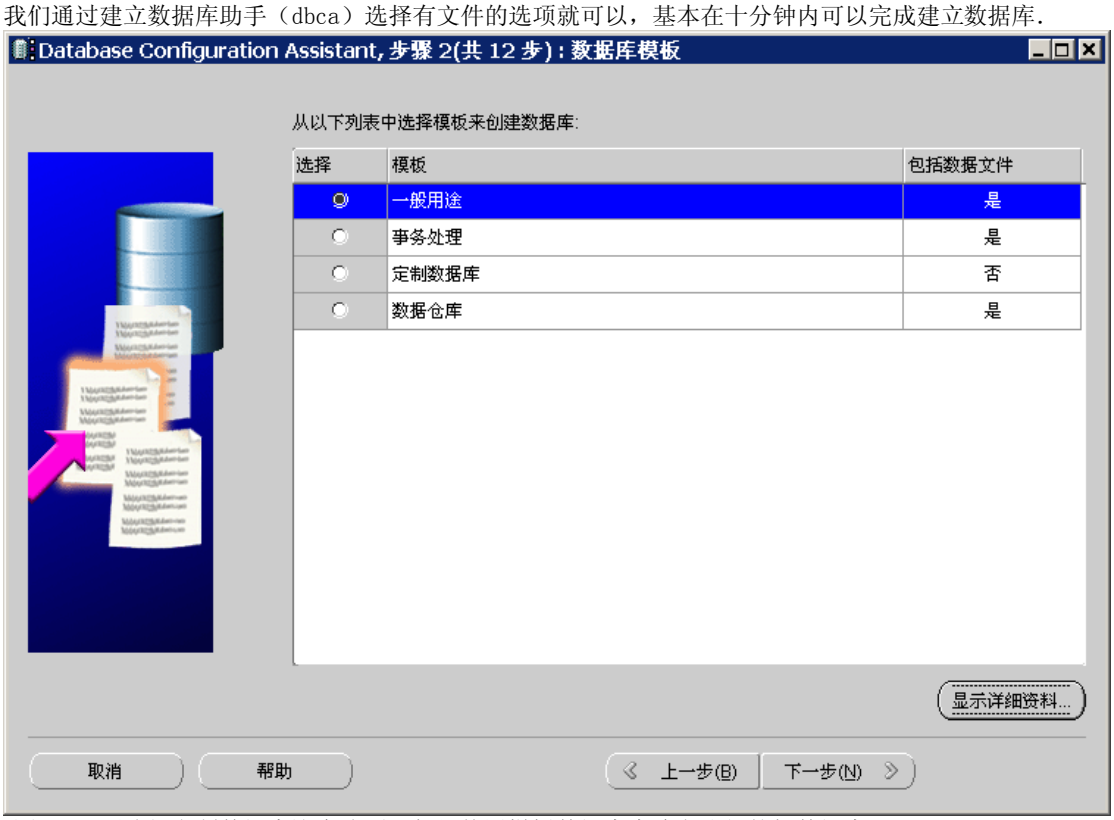
一致性 shutdown,产生检测点
Shutdown NORMAL
Shutdown TRANSACTIONAL
Shutdown IMMEDIATE
数据库再次启动的时候不要恢复

不一致性 shutdown,不产生检测点
Shutdown abort
Startup force
Instance 崩溃（停电）
数据库再次启动的时候需要恢复，自动的，透明的。

实验 48：建立数据库

该实验的目的是会使用向导和手工建立数据库.

数据库的建立可以通过样板库建立，也可以自己建立基本的数据库.
使用样板数据库来建立比较简单，主要是解压缩和客户化的过程. 每个数据库产品安装完成以后，都会有一份打好包的数据库，存在于%oracle_home%\assistants\dbca\templates 目录下.
我们通过建立数据库助手（dbca）选择有文件的选项就可以，基本在十分钟内可以完成建立数据库.



我们只要不选择定制数据库这个选项，都是使用样板数据库来建立我们的新数据库.
建立数据库的步骤如下：

```
mkdir D:\oracle\product\10.2.0\db_2\cfgtoollogs\dbca\km
mkdir D:\oracle\product\10.2.0\db_2\dbs
mkdir f:\oracle\admin\km\adump
mkdir f:\oracle\admin\km\bdump
mkdir f:\oracle\admin\km\cdump
```

```
mkdir f:\oracle\admin\km\dpdump
mkdir f:\oracle\admin\km\pfile
mkdir f:\oracle\admin\km\udump
mkdir f:\oracle\oradata\km
set ORACLE_SID=km
```
这句话的目的是设置环境变量
```
D:\oracle\product\10.2.0\db_2\bin\oradim.exe -new -sid KM -startmode manual -spfile
```
这句话的目的是建立服务中的服务项，并修改相对应的注册表.
```
D:\oracle\product\10.2.0\db_2\bin\oradim.exe -edit -sid KM -startmode auto -srvcstart system
```
这句话的目的是编辑服务项，修改其中的属性
```
D:\oracle\product\10.2.0\db_2\bin\sqlplus /nolog @f:\oracle\admin\km\scripts\km.sql
set verify off
PROMPT specify a password for sys as parameter 1;
DEFINE sysPassword = &1
PROMPT specify a password for system as parameter 2;
DEFINE systemPassword = &2
host                                            D:\oracle\product\10.2.0\db_2\bin\orapwd.exe
file=D:\oracle\product\10.2.0\db_2\database\PWDkm.ora password=&&sysPassword force=y
```
这句话的目的是建立密码文件
```
@f:\oracle\admin\km\scripts\CloneRmanRestore.sql
@f:\oracle\admin\km\scripts\cloneDBCreation.sql
@f:\oracle\admin\km\scripts\postScripts.sql
host        "echo        SPFILE='D:\oracle\product\10.2.0\db_2/dbs/spfilekm.ora'        >
D:\oracle\product\10.2.0\db_2\database\initkm.ora"
@f:\oracle\admin\km\scripts\postDBCreation.sql
```

建立新的自定义数据库可以使用向导的定制数据库这个选项，也可以完全自己书写脚本. 一般来说我们是使用向导产生建立数据库的脚本，然后再修改脚本，最后运行脚本
自定义建立数据库要完成如下的过程.
建立参数文件
建立密码文件
建立服务项
启动到 nomount
```
CREATE DATABASE "manual"
MAXINSTANCES 8
MAXLOGHISTORY 1
MAXLOGFILES 16
MAXLOGMEMBERS 3
MAXDATAFILES 100
DATAFILE 'd:\oracle\oradata\manual\system01.dbf' SIZE 300M REUSE AUTOEXTEND ON NEXT  10240K
MAXSIZE UNLIMITED
EXTENT MANAGEMENT LOCAL
SYSAUX DATAFILE 'd:\oracle\oradata\manual\sysaux01.dbf' SIZE 120M REUSE AUTOEXTEND ON NEXT
10240K MAXSIZE UNLIMITED
SMALLFILE DEFAULT TEMPORARY TABLESPACE TEMP TEMPFILE 'd:\oracle\oradata\manual\temp01.dbf' SIZE
20M REUSE AUTOEXTEND ON NEXT  640K MAXSIZE UNLIMITED
SMALLFILE UNDO TABLESPACE "UNDOTBS1" DATAFILE 'd:\oracle\oradata\manual\undotbs01.dbf' SIZE
200M REUSE AUTOEXTEND ON NEXT  5120K MAXSIZE UNLIMITED
CHARACTER SET ZHS16GBK
NATIONAL CHARACTER SET AL16UTF16
LOGFILE GROUP 1 ('d:\oracle\oradata\manual\redo01.log') SIZE 51200K,
GROUP 2 ('d:\oracle\oradata\manual\redo02.log') SIZE 51200K,
GROUP 3 ('d:\oracle\oradata\manual\redo03.log') SIZE 51200K
USER SYS IDENTIFIED BY "Password" USER SYSTEM IDENTIFIED BY "Password";
```
上面写的是一句话，这句话在 nomount 状态下运行. 这句话是建立数据库的核心语法，它建立了 system 表空间的文件，建立了回退表空间，建立了 sysaux 系统辅助表空间，建立了临时表空间，建立了控制文件，初始化了日志文件. 在系统表空间中放入了以$结尾的基表，然后将数据库启动到 open 状态.
这一句话结束以后数据库就诞生了. 可以使用了，没有字典，没有系统包的支持，使用起来很不方便，功

能也有限. 就象我们刚刚安装完操作系统，没有安装任何软件，你可以使用系统，但你不能打开 word 文件，不能听 mp3，你有的只是一个基本的操作系统，我们建立数据库的道理也是一样，刚建立的数据库还需要完善，需要建立数据字典，需要建立系统包，需要有其它表空间.
建立表空间
建立数据字典%oracle_home%\rdbms\admin\catalog.sql;
建立系统包%oracle_home%\rdbms\admin\catproc.sql;

实验 49：查找你想要的数据字典

该实验的目的是了解什么是数据字典,字典的来源和如何查找到我们关心的数据字典.
数据字典是 oracle 的核心
分为两大类
存在于 **system 表空间**
...$结尾的基本表
Dba_..,all_....,user_...视图

存在于**内存中**
X$.....的虚表
V$...的动态性能视图
数据字典是哪里来的呢?是我们建立数据库的时候运行脚本建立的.
%oracle_home%\rdbms\admin\catalog.sql;脚本当中含有建立数据字典的语句.而 v$的字典是数据库在启动实例的时候初始化的.

数据字典的使用
　　数据库自己使用字典获取信息
　　数据库自动维护
　　我们查看字典来获得数据库的有关信息

基本表,是字典得基本表,在建立 system 表空间的时候建立的.
```
select table_name,owner from dba_tables where table_name like '%$' and owner='SYS';
```
视图,是在建立数据库以后运行 catalog.sql;脚本建立的.
查看哪些字典中含有 TABLE 关键字,一定要大写.
```
select table_name from dict where table_name like '%TABLE%';
```

查看哪些字典中含有 VIEW 关键字,一定要大写.
```
select table_name from dict where table_name like '%VIEW%';
```

查看哪些字典中的一列含有 FILE#这一列,一定要大写.
```
select table_name from dict_COLUMNS WHERE COLUMN_NAME='FILE#';
```

查看所有的 x$和 v$的表的信息.
```
SELECT * FROM V$FIXED_TABLE;
```

三大类视图,****代表可以替换为某个单词.
Dba_*****
All_****
User_****
我们拿 tables 说明上面得含义.
其中 **user_tables** 是查看当前用户所拥有的表. **all_tables** 是查看当前用户可以访问的表. **dba_tables** 是查看当前整个数据库拥有的表,但是你得有权限,如果没有权限会报没有这个表.

控制文件

控制文件是二进制文件（不会超过 100m，一般是几 m 大小）

控制文件记录了数据库的结构和行为
在 mount 时候读
在数据库 open 时一直使用
丢失需要恢复
相关字典
Select * from v$controlfile;
select CONTROLFILE_SEQUENCE# from v$database;
select TYPE,RECORD_SIZE,RECORDS_TOTAL,RECORDS_USED from V$CONTROLFILE_RECORD_SECTION;
select value from V$spparameter where name='control_files';
控制文件的位置在参数文件中描述
control_files='file1','file2'
多个控制文件是镜像的关系
最多八个,最少一个

实验 50：减少控制文件的个数

该实验的目的是初步认识如何修改参数文件,如何减少控制文件.
减少控制文件，实验的目的，有一个控制文件损坏，我们要将损坏的控制文件剔除。
1.修改参数文件，并验证
2.停止数据库
3.启动数据库
4.验证,查看 v$controlfile
SQL> select * from v$controlfile;
验证现在内存中的控制文件个数

```
STATUS     NAME                                     IS_ BLOCK_SIZE FILE_SIZE_BLKS
---------- ---------------------------------------- --- ---------- --------------
           D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL     NO       16384            450
           D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL     NO       16384            450
```

修改二进制的初始化参数文件中的 control_files 选项

```
SQL> alter system set control_files=
  2  'D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL' scope=spfile;

System altered.
```

验证参数文件已经被修改

```
SQL> select value from v$spparameter where name='control_files';

VALUE
--------------------------------------------------------------------------------
D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL
```

验证内存中的值没有被修改，因为 control_files 是静态参数，想要改变必须重新启动数据库。

```
SQL> select * from v$controlfile;

STATUS     NAME                                     IS_ BLOCK_SIZE FILE_SIZE_BLKS
---------- ---------------------------------------- --- ---------- --------------
           D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL     NO       16384            450
           D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL     NO       16384            450
```

重新启动数据库，使修改的参数起作用

```
SQL> startup force;
ORACLE instance started.

Total System Global Area  167772160 bytes
Fixed Size                  1247900 bytes
Variable Size              75498852 bytes
Database Buffers           88080384 bytes
Redo Buffers                2945024 bytes
Database mounted.
Database opened.
```

```
SQL> select * from v$controlfile;
验证内存被修改了
STATUS     NAME                                     IS_ BLOCK_SIZE FILE_SIZE_BLKS
---------- ---------------------------------------- --- ---------- --------------
           D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL     NO       16384            450

SQL> select value from v$spparameter where name='control_files';
验证参数文件中的值和内存中的值相同
VALUE
--------------------------------------------------------------------------------
D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL
```

如果你在启动的时候看到 ora-00205 错误，说明你修改的参数不正确，可能是路径写的不对或者在路径前面多写了空格，请重新修改为正确的值再重新启动数据库。

实验 51：增加控制文件的个数

实验的目的是增加控制文件的个数，1 到 8 个，保护控制文件。认识控制文件的一致性.什么是控制文件的版本.控制文件的结构.

增加控制文件

1.修改参数文件

2.停止数据库

3.复制控制文件

4.启动数据库

5.验证,查看 v$controlfile

修改二进制的初始化参数文件中的 control_files 选项

```
SQL> alter system set control_files=
  2  'D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL',
  3  'D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL'
  4   scope=spfile;

System altered.

SQL>  select value from v$spparameter where name='control_files';
验证参数文件已经被修改
VALUE
--------------------------------------------------------------------------------
D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL
D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL
SQL> select * from v$controlfile;
验证现在内存中的控制文件个数

STATUS     NAME                                     IS_ BLOCK_SIZE FILE_SIZE_BLKS
---------- ---------------------------------------- --- ---------- --------------
           D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL     NO       16384            450

SQL> startup force;
ORACLE instance started.
重新启动数据库，使修改的参数起作用
Total System Global Area  167772160 bytes
Fixed Size                  1247900 bytes
Variable Size              75498852 bytes
Database Buffers           88080384 bytes
Redo Buffers                2945024 bytes
ORA-00214: control file 'D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL' version 9499 inconsistent with
file
'D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL' version 9483
```

因为 CONTROL02.CTL 刚才脱离了数据库，没有参加修改，CONTROL01.CTL 已经变化了，二 CONTROL02.CTL 没有变化，所以时间戳不正确了。

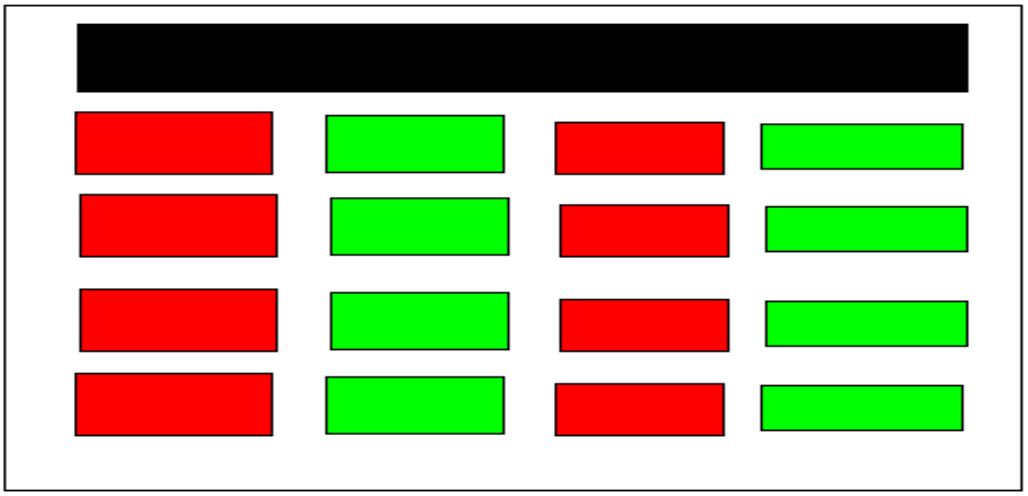
```
SQL> host copy D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL
```
使用操作系统的命令将老的控制文件覆盖
```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01507: database not mounted
```
因为我们处于数据库的 nomount 状态，想要 open 不能跨越 mount 台阶，所以必须先 mount 数据库。

```
SQL> alter database mount;
```
启动到 mount 状态
```
Database altered.

SQL> alter database open;
```
启动到 open 状态
```
Database altered.

SQL> select value from v$spparameter where name='control_files';
```
验证参数文件中 control_files 选项的值
```
VALUE
--------------------------------------------------------------------------------
D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL
D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL

SQL> select * from v$controlfile;
```
验证现在内存中的控制文件个数

```
STATUS     NAME                                     IS_ BLOCK_SIZE FILE_SIZE_BLKS
---------- ---------------------------------------- --- ---------- --------------
           D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL     NO       16384            450
           D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL     NO       16384            450
```
再次修改参数文件 control_files 的值为 3 个控制文件
```
SQL> alter system set control_files=
  2  'D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL',
  3  'D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL',
  4  'D:\ORACLE\ORADATA\ORA10\CONTROL03.CTL'
  5   scope=spfile;

System altered.

SQL> startup force;
ORACLE instance started.
```
重新启动数据库，使修改的参数起作用
```
Total System Global Area  167772160 bytes
Fixed Size                  1247900 bytes
Variable Size              75498852 bytes
Database Buffers           88080384 bytes
Redo Buffers                2945024 bytes
ORA-00205: error in identifying control file, check alert log for more info
```
因为不存在 CONTROL03.CTL 文件，所以数据库报错，没有找到指定的控制文件。

```
SQL> host copy D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL D:\ORACLE\ORADATA\ORA10\CONTROL03.CTL
```
将最新的控制文件拷贝到 CONTROL03.CTL 文件，使三个控制文件完全相同
```
SQL>  alter database mount;
```
启动到 mount 状态
```
Database altered.
```

```
SQL> alter database open;
```
启动到 open 状态
```
Database altered.

SQL> select value from v$spparameter where name='control_files';
```
验证参数文件中 control_files 选项的值
```
VALUE
--------------------------------------------------------------------------------
D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL
D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL
D:\ORACLE\ORADATA\ORA10\CONTROL03.CTL

SQL> select * from v$controlfile;
```
验证现在内存中的控制文件个数
```
STATUS     NAME                                     IS_ BLOCK_SIZE FILE_SIZE_BLKS
---------- ---------------------------------------- --- ---------- --------------
           D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL    NO       16384            450
           D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL    NO       16384            450
           D:\ORACLE\ORADATA\ORA10\CONTROL03.CTL    NO       16384            450
```
控制文件的结构,每个块大小为 db_block_size



黑色的为文件头,8 个数据块
红的为每个 8k
绿的为红的块的镜像

控制文件已经使用空间和剩余空间
```
select TYPE,RECORD_SIZE,RECORDS_TOTAL,RECORDS_USED from V$CONTROLFILE_RECORD_SECTION;
```
控制文件是预留空间

控制文件的估计大小
例子中 db_block_size=8k
```
select       sum(ceil(RECORD_SIZE*RECORDS_TOTAL/(8192-24))*2*8)+8*8             kb       from
V$CONTROLFILE_RECORD_SECTION;
```
8*8 为控制文件头,8 个块
8192-24,24 为块头
Ceil 取整,因为分配单位为块
2 倍,因为控制文件是内部镜像的
如何将控制文件的信息转储到跟追文件呢?
```
alter session set events 'immediate trace name controlf level 1';
```
level 1 代表只 dump 文件头的信息.
```
show parameter user_d
```
你的转储文件在 udump 目录下.根据日期和时间来查找,也可以根据进程号码来查找.以后再介绍.

```
*** 2007-09-25 21:52:45.467
DUMP OF CONTROL FILES, Seq # 10721 = 0x29e1
 V10 STYLE FILE HEADER:
     Compatibility Vsn = 169869568=0xa200100
     Db ID=568967312=0x21e9c090, Db Name='ORA10'
     Activation ID=0=0x0
     Control Seq=10721=0x29e1, File size=450=0x1c2
     File Number=0, Blksiz=16384, File Type=1 CONTROL
*** END OF DUMP ***
alter session set events 'immediate trace name controlf level 10';
```
level 10 代表只 dump 文件的所有信息.

日志文件

日志文件是二进制文件
它记录了数据文件的变化
Select * from v$logfile;
查看日志文件的位置等信息

```
SQL> Select * from v$logfile;

 GROUP# STATUS     TYPE            MEMBER                                  IS_
------- ---------- --------------- --------------------------------------- ---
      3            ONLINE          D:\ORACLE\ORADATA\ORA10\REDO03.LOG      NO
      2            ONLINE          D:\ORACLE\ORADATA\ORA10\REDO02.LOG      NO
      1            ONLINE          D:\ORACLE\ORADATA\ORA10\REDO01.LOG      NO
      4            STANDBY         D:\ORACLE\ORADATA\ORA10\REDO04.LOG      NO
      5            STANDBY         D:\ORACLE\ORADATA\ORA10\REDO05.LOG      NO
```

日志文件是物理存在的文件
它的组织模式是组
组是逻辑的组织方式
每个实例至少要两个组

```
      Select * from v$log;
      Select * from  v$log_history;
      查看日志组的信息
SQL> Select * from v$log;

 GROUP#  THREAD#  SEQUENCE#      BYTES    MEMBERS ARC STATUS     FIRST_CHANGE# FIRST_TIME
------- -------- ---------- ---------- ---------- --- ---------- ------------- ----------
      1        1        184    5242880          1 NO  CURRENT          3331887 26-JUN-07
      2        1        182    5242880          1 YES INACTIVE         3320448 25-JUN-07
      3        1        183    5242880          1 YES INACTIVE         3331659 26-JUN-07


SQL> select to_char(FIRST_TIME,'yyyy/mm/dd') day,count(*) from
  2  v$log_history
  3  group by to_char(FIRST_TIME,'yyyy/mm/dd');

DAY                  COUNT(*)
-------------------- ----------
2007/06/14                    4
2007/04/29                    1
2007/06/25                    1
2007/06/07                   34
2007/04/30                    8
2007/06/13                   84
```

2007/05/01 7
2007/05/02 1
2007/06/26 1
2007/05/28 25
2007/06/12 10
2007/06/23 7
该语句可以查看每天产生日志的多少，估计我们应用的日志量，可以估计归档的大小。

组和组间是平等的关系
实例同一时刻只能向一个组写入日志
一个组写满后，写下一个组
这个过程叫切换（switch）
自动切换：日志写满 oracle 会写下一个组
手工切换：alter system switch logfile;

日志组的切换要产生检查点(checkpoint)
检查点有增量检查和完全检查两种
完全检查
1。一致性 shutdown 数据库的时候。
2。Alter system checkpoint;
结果为：
所有的脏数据块都写入数据文件
改写文件的头
除了完全检查点以外的所有其它检查点都是增量检查点，增量检查是查找检查点列表，将某一个时间点做标记，该时间点前的脏块写入到数据文件，增量检查不一定马上执行，根据我们脏的块多少来决定，这就出现了检查点滞后的情况。
参数 log_checkpoints_to_alert 决定是否将检查点的信息写入报警日志。默认为假，不写日志。
我们可以将这个参数改为真，可以看到检查点的信息。
日志的部分信息如下

```
Fri Jul 06 12:30:52 2007
ALTER SYSTEM SET log_checkpoints_to_alert=TRUE SCOPE=BOTH;
Fri Jul 06 12:31:15 2007
Beginning log switch checkpoint up to RBA [0xb9.2.10], SCN: 3334816
Thread 1 advanced to log sequence 185
  Current log# 2 seq# 185 mem# 0: D:\ORACLE\ORADATA\ORA10\REDO02.LOG
Beginning log switch checkpoint up to RBA [0xba.2.10], SCN: 3334818
Thread 1 advanced to log sequence 186
  Current log# 3 seq# 186 mem# 0: D:\ORACLE\ORADATA\ORA10\REDO03.LOG
Thread 1 cannot allocate new log, sequence 187
Checkpoint not complete
  Current log# 3 seq# 186 mem# 0: D:\ORACLE\ORADATA\ORA10\REDO03.LOG
Fri Jul 06 12:31:21 2007
Completed checkpoint up to RBA [0xb9.2.10], SCN: 3334816
Fri Jul 06 12:31:22 2007
Beginning log switch checkpoint up to RBA [0xbb.2.10], SCN: 3334826
Thread 1 advanced to log sequence 187
  Current log# 1 seq# 187 mem# 0: D:\ORACLE\ORADATA\ORA10\REDO01.LOG
Fri Jul 06 12:31:27 2007
Completed checkpoint up to RBA [0xba.2.10], SCN: 3334818
```

System change number(scn)，数据库的更改号码，如果你不懂 SCN 就绝对不懂数据库，这句话一点都不夸张，因为数据库中的一切运转都离开不了 SCN，SCN 的地位在数据库中就向我们生活中的时间一样，你觉察不到，但又处处离不开。SCN 存在于数据块的块头，文件头，也可以建立特殊的表，使 SCN 存在于表的行

头，SCN 存在内存中，它是维护数据库的运行基本保证。备份和恢复更是根据 SCN 来决定我们要重做那些操作和交易。SCN 的发生机制在不同版本会不同，我们也不用去关心，我们可以理解为数据库的一切进程操作都要有一个时间的标志，这就是 SCN。Select ,update,delete,insert 数据库的一切操作都有 SCN。数据库内的任何操作都产生 scn。SCN 小的就是先操作的，SCN 大的就是后操作的，数据库使用 SCN 来维护因果关系。我们可以将 SCN 理解为数据库的内部时间。
Scn 的最大值为

```
SQL> select to_number('fffffffffffff','xxxxxxxxxxxxxxxxxxxxxxxxxxxxx') from dual;

TO_NUMBER('FFFFFFFFFFFF','XXXX
------------------------------
                    2.8147E+14
```

为什么最大值是'fffffffffffff'呢?

```
SQL> alter system dump datafile 1 block 2;

System altered.
```

我们随便将一个数据文件的块转存到 udump 的跟踪文件.

```
Start dump data blocks tsn: 0 file#: 1 minblk 2 maxblk 2
buffer tsn: 0 rdba: 0x00400002 (1/2)
scn: 0x0000.c666400e seq: 0x02 flg: 0x04 tail: 0x400e1d02
frmt: 0x02 chkval: 0x91eb type: 0x1d=KTFB Bitmapped File Space Header
Hex dump of block: st=0, typ_found=1
```

我们看到了吧.最大就是 12 位的十六进制数值.

数据文件头会保存一个特殊的 SCN
Stop scn 记录在数据文件头上。当数据库处在打开状态时，stop scn 被设成最大值 0xffff.ffffffff。在数据库正常关闭过程中，stop scn 被设置成当前系统的最大 scn 值。在数据库打开过程中，Oracle 会比较各文件的 stop scn 和 checkpoint scn，如果值不一致，表明数据库先前没有正常关闭，需要做恢复。

查看数据库当前 scn

```
select current_scn from V$database;(10g 才有)
select dbms_flashback.get_system_change_number() from dual; (9i 以后才有)
```

检查点的 scn，检查点是一个特殊的 SCN，小于该号码的块都已经存盘了，数据库的恢复只需要恢复该 SCN 号码以后的操作就可以了。
SCN 号码和物理的时间有对照表。SMON_SCN_TIME

```
select * from SMON_SCN_TIME;
```

这个表在每个版本的结果会不同，9I 的信息较少，10G 的信息更多一些。

```
select name,checkpoint_change# from v$datafile;
NAME                                     CHECKPOINT_CHANGE#
---------------------------------------- ------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF                3334818
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF               3334818
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF                3334818
D:\ORACLE\ORADATA\ORA10\USERS01.DBF                 3334818
D:\ORACLE\ORADATA\ORA10\TL.DBF                      3334818
D:\ORACLE\ORADATA\ORA10\TS1.1                       3334818
D:\ORACLE\ORADATA\ORA10\TS1.2                       3334818
```

日志记录的范围

```
SQL>  select GROUP#,sequence#,STATUS,FIRST_CHANGE#,
  2  to_char(FIRST_TIME,'yyyy/mm/dd:hh24:mi:ss') time from V$log;

 GROUP#  SEQUENCE# STATUS     FIRST_CHANGE# TIME
------- ---------- ---------- ------------- --------------------
      1        187 CURRENT          3334826 2007/07/06:12:31:22
      2        185 INACTIVE         3334816 2007/07/06:12:31:14
      3        186 INACTIVE         3334818 2007/07/06:12:31:16
```

185 号日志记录了 3334816 到 3334818 之间的数据库变化。

186 号日志记录了 3334818 到 3334826 之间的数据库变化。
187 号日志记录了 3334826 到最后的 SCN 之间的数据库变化。

如何将日志文件的信息转储到 dump 文件中.
ALTER SESSION SET EVENTS 'immediate trace name redohdr level n';

1 控制文件中的 redo log 信息
2 level 1 + 文件头信息
3 level 2 + 日志文件头信息

ALTER SYSTEM DUMP LOGFILE 'FileName';

实验 52：日志文件管理和 nologging 的实现

该实验的目的是验证我们学习的日志文件的原理,管理维护日志文件.如何减少日志的产生.用到了一些后面的语法,克服一下.
增加组
查看当前日志文件的路径
Select member from v$logfile;
增加组
alter database add logfile group 9 'D:\ORACLE\ORADATA\O10\REDO08.rr' size 4m;
如果不指定组号，数据库自动分配组号
验证
Select * from v$log;
Select * from v$logfile;

组内的日志文件叫做成员
同组内的成员是镜像关系，大小相等
使用成员的目的是安全
一个组内有一个成员可以使用，该组就可用
一般要把不同的成员放在不同的盘上
一个组内成员的最大成员数由控制文件决定。

增加成员到现有的组
alter database add logfile member 'D:\ORACLE\ORADATA\O10\REDO08.kk' to group 9;
增加成员不要指明大小，它和现有的日志大小相同，是镜像关系。要指明组。
Alter system switch logfile;
使成员的状态为正确
Select * from v$logfile;
查看成员信息

删除成员
alter database drop logfile member 'D:\ORACLE\ORADATA\O10\REDO08.kk';
如果删除不掉，可能是因为该组为当前组
手工切换
alter system switch logfile;
再删除就可以了，一个组内最少有一个成员。最后的成员是不能删除的。

日志文件改名称
1.Select * from v$log;
2.如果想修改的日志为当前组，请切换
Alter system switch logfile;
3.select * from v$logfile;
查看现有的文件名称
4.拷贝日志文件到新的名称
5.alter database rename file '..old' to '...new';

这句话是修改控制文件的指针，使控制文件知道日志文件已经处于新的位置了，所以新的位置文件一定得存在，不然会报错。

6.select * from v$logfile;

验证修改成功

清除组内的内容

Alter database clear logfile group 9;

如果清除不了

Alter system switch logfile;

清除等于删除老的，增加新的日志文件

删除组

该组应该为非当前，非活动。如果是请切换组

Alter database drop logfile group 9;

正常的 DML 总要产生日志,但当我们要大量的加载数据的时候我们希望尽快的完成任务,我们可以使用 nologging 的选项,该选项可以减少日志的产生,只产生少量的日志.当真的是这样吗?不是的,请看实验

```
SQL> conn scott/tiger
Connected.
SQL> drop table t1 purge;

Table dropped.

SQL> drop table t2 purge;

Table dropped.

SQL> create table t1 as select * from emp;

Table created.

SQL> insert into t1 select * from t1;

14 rows created.

SQL> /

28 rows created.
```

反复重复插入,直到 2000 行.构造一个**大表**.

```
SQL> /

1792 rows created.

SQL> commit;

Commit complete.
SQL> create table t2 as select * from t1 where 0=9;
```

建立**空表** t2

```
Table created.

SQL> set autotrace trace stat
SQL> insert into t2 select * from t1;
```

正常的 SQL 语句,没有禁止日志的产生

```
3584 rows created.

Statistics
----------------------------------------------------------
        298  recursive calls
        322  db block gets
```

```
        157  consistent gets
          1  physical reads
     193200  redo size  正常产生日志

SQL> rollback;

Rollback complete.

SQL> insert into t2 select * from t1 nologging;
禁止日志的产生模式插入

3584 rows created.

Statistics
----------------------------------------------------------
          5  recursive calls
        166  db block gets
        106  consistent gets
          0  physical reads
     181104  redo size   没有效果


SQL> rollback;

Rollback complete.

SQL> alter table t2 nologging;
将 T2 表改为 nologging 模式
Table altered.

SQL> insert into t2 select * from t1 nologging;

3584 rows created.

Statistics
----------------------------------------------------------
        175  recursive calls
        164  db block gets
        125  consistent gets
          0  physical reads
     181044  redo size   没有效果


SQL> rollback;

Rollback complete.

SQL> insert /*+ append */ into t2 select * from t1 nologging;

3584 rows created.
Statistics
----------------------------------------------------------
        136  recursive calls
        107  db block gets
        105  consistent gets
          0  physical reads
     189460  redo size  没有效果
```

为什么我们**无论**怎么改变都没有效果,很奇怪

```
SQL> conn / as sysdba
Connected.
SQL> select FORCE_LOGGING from v$database;

FORCE_
------
YES
```

原来数据库的运行模式为强制产生日志,这就难怪了.

```
SQL> alter database no FORCE LOGGING;

Database altered.

SQL> select FORCE_LOGGING from v$database;

FORCE_
------
NO

SQL> conn scott/tiger
Connected.
SQL> set autot  trace stat
SQL> insert into t2 select * from t1;

3584 rows created.

Statistics
----------------------------------------
        311  recursive calls
        166  db block gets
        141  consistent gets
          0  physical reads
     181052  redo size

SQL> insert into t2 select * from t1 nologging;

3584 rows created.

Statistics
----------------------------------------------
        327  recursive calls
        261  db block gets
        183  consistent gets
          0  physical reads
     188816  redo size  没有效果

SQL> insert /*+ append */ into t2 select * from t1;
```

并行插入**无** nologging 选项

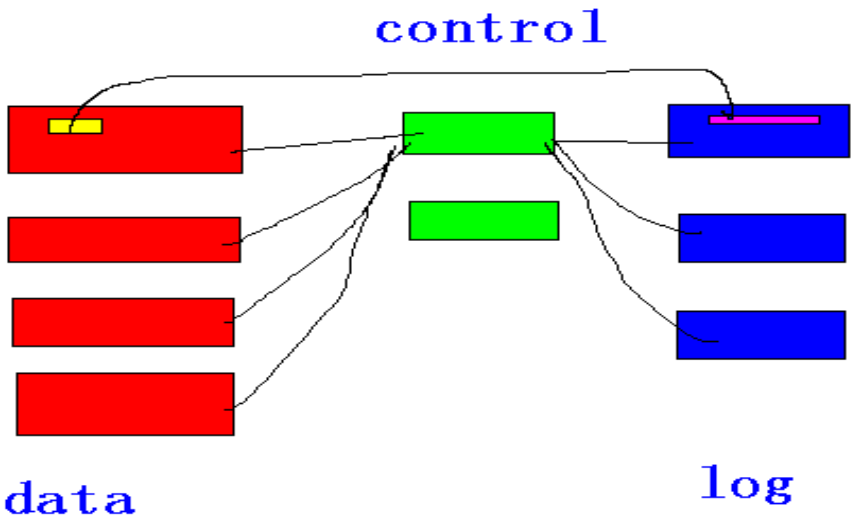
```
3584 rows created.


Statistics
-----------------------------------------------------------
          0  recursive calls
         34  db block gets
         31  consistent gets
          0  physical reads
        528  redo size  有效果
```

```
SQL> rollback;

Rollback complete.

SQL> insert /*+ append */ into t2 select * from t1 nologging;
并行插入加 nologging 选项
3584 rows created.


Statistics
----------------------------------------------------------
          0  recursive calls
         34  db block gets
         31  consistent gets
          0  physical reads
        484  redo size  效果最好,只有少量的日志产生
```

总结:只有数据库不是强制产生日志的状态下,并行插入才可以产生少量的日志.其它情况都要产生大量的日志,无论你在什么样的表空间,无论是否使用 nologging 选项.

数据文件

数据文件是数据的存放载体
数据文件存在于操作系统上,可以不是文件。设备也可以。
数据文件不能独立存在,得有组织
数据文件的逻辑组织形式为表空间 tablespace
一个表空间内可以含有多个数据文件
数据库内可以有多个表空间



红的是存放数据的数据文件
红色的变化存放在蓝色的日志文件
绿色为控制文件，存放红色和蓝色的结构和行为

实验 53：建立新的表空间

该实验的目的是初步认识数据文件和表空间.
查看表空间和数据文件的信息
select tablespace_name,file_name,ceil(bytes/1024/1024) mb

```
from dba_data_files order by 1;
TABLESPACE_NAME FILE_NAME                                          MB
--------------- ---------------------------------------- ----------
SYSAUX          D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF             290
SYSTEM          D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF             490
TL              D:\ORACLE\ORADATA\ORA10\TL.DBF                     2
UNDOTBS1        D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF             60
USERS           D:\ORACLE\ORADATA\ORA10\USERS01.DBF               18
```

建立新的表空间

```
create tablespace ts1 datafile 'D:\ORACLE\ORADATA\Ora10\ts1.1' size 2m;
```

验证 dba_data_files

```
select tablespace_name,file_name,ceil(bytes/1024/1024) mb
from dba_data_files order by 1;

TABLESPACE_NAME FILE_NAME                                          MB
--------------- ---------------------------------------- ----------
SYSAUX          D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF             290
SYSTEM          D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF             490
TL              D:\ORACLE\ORADATA\ORA10\TL.DBF                     2
TS1             D:\ORACLE\ORADATA\ORA10\TS1.1                      2
UNDOTBS1        D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF             60
USERS           D:\ORACLE\ORADATA\ORA10\USERS01.DBF               18
```

加入新的数据文件

```
alter tablespace ts1 add datafile 'D:\ORACLE\ORADATA\Ora10\ts1.2' size 2m;
```

数据文件只能加入，不能删除，除非将表空间删除。但在 10G 数据库版本可以删除

```
SQL> select tablespace_name,file_name,ceil(bytes/1024/1024) mb
  2  from dba_data_files order by 1;

TABLESPACE_NAME FILE_NAME                                          MB
--------------- ---------------------------------------- ----------
SYSAUX          D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF             290
SYSTEM          D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF             490
TL              D:\ORACLE\ORADATA\ORA10\TL.DBF                     2
TS1             D:\ORACLE\ORADATA\ORA10\TS1.1                      2
TS1             D:\ORACLE\ORADATA\ORA10\TS1.2                      2
UNDOTBS1        D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF             60
USERS           D:\ORACLE\ORADATA\ORA10\USERS01.DBF               18
SQL> alter tablespace ts1 drop datafile 'D:\ORACLE\ORADATA\ORA10\TS1.2';

Tablespace altered.

SQL> select tablespace_name,file_name,ceil(bytes/1024/1024) mb
  2  from dba_data_files order by 1;

TABLESPACE_NAME FILE_NAME                                          MB
--------------- ---------------------------------------- ----------
BIGTS           D:\ORACLE\ORADATA\ORA10\BIGTS.BIG                  2
SYSAUX          D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF             290
SYSTEM          D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF             490
TL              D:\ORACLE\ORADATA\ORA10\TL.DBF                     2
TS1             D:\ORACLE\ORADATA\ORA10\TS1.1                      2
UNDOTBS1        D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF             60
USERS           D:\ORACLE\ORADATA\ORA10\USERS01.DBF               18

7 rows selected.
```

改变数据文件的大小

可以加大，也可以缩小

Alter database datafile '.....' resize 100m;
100m 为最后的大小，不是加大 100m.一个文件的最小为文件头加最小的表，一个文件最大为该表空间的块大小乘 4m.

数据文件的自动扩展
select FILE_NAME,AUTOEXTENSIBLE,MAXBLOCKS,
INCREMENT_BY from dba_data_files;
改为自动扩展
alter database datafile 'D:\ORACLE\ORADATA\O10\TS1.1'
autoextend on next 1m maxsize 100m;
改为手工扩展
alter database datafile 'D:\ORACLE\ORADATA\O10\TS1.1'
Autoextend off;

表空间只读
Alter tablespace users read only;
验证
select TABLESPACE_NAME,STATUS from dba_tablespaces;
读写
Alter tablespace users read write;
只读表空间内的表不能 dml，但可以 drop .
因为 DROP 操作的是 system 表空间，SYSTEM 表空间不能设为只读。

Offline 表空间
 alter tablespace users offline;
验证
select TABLESPACE_NAME,STATUS from dba_tablespaces;
在线
alter tablespace users online;
只有完整的数据文件才可以 online,如果不完整请恢复。
恢复一致后再 online;
文件 online 时用户才可以访问

实验 54：更改表空间的名称，更改数据文件的名称

该实验的目的是管理表空间,了解什么是数据文件的一致性.
表空间改名称（10g 新特性）
System 和 sysaux 表空间不能改名称
要改的表空间必须 online,read write
版本 10 以上
Alter tablespace ts1 rename to ts2;

数据文件改名称
1。查看现有文件位置
2。Offline
3。复制到新的名称
4。Alter database rename file '...old' to '...new';
5。online
6。查看 dba_data_files 验证

SQL> select name,status from v$datafile;
查看数据文件得名称和状态

NAME	STATUS
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF	SYSTEM
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF	ONLINE
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF	ONLINE

```
D:\ORACLE\ORADATA\ORA10\USERS01.DBF        ONLINE
D:\ORACLE\ORADATA\ORA10\TL.DBF             ONLINE
D:\ORACLE\ORADATA\ORA10\TS1.1              ONLINE
D:\ORACLE\ORADATA\ORA10\TS1.2              ONLINE

SQL> alter database datafile 'D:\ORACLE\ORADATA\ORA10\TS1.1' offline;
使某个数据文件 offline,该表空间得其它数据文件 online;
Database altered.

SQL> select name,status from v$datafile;
查看数据文件得名称和状态

NAME                                       STATUS
------------------------------------------ ----------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF        SYSTEM
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF       ONLINE
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF        ONLINE
D:\ORACLE\ORADATA\ORA10\USERS01.DBF         ONLINE
D:\ORACLE\ORADATA\ORA10\TL.DBF              ONLINE
D:\ORACLE\ORADATA\ORA10\TS1.1               RECOVER
D:\ORACLE\ORADATA\ORA10\TS1.2               ONLINE

7 rows selected.

SQL> host copy D:\ORACLE\ORADATA\ORA10\TS1.1 D:\ORACLE\ORADATA\ORA10\TS1.dbf
操作系统的复制
SQL> alter database rename file 'D:\ORACLE\ORADATA\ORA10\TS1.1'
  2  to 'D:\ORACLE\ORADATA\ORA10\TS1.dbf';
修改控制文件的指针描述
Database altered.

SQL> alter database datafile 'D:\ORACLE\ORADATA\ORA10\TS1.dbf' online;
alter database datafile 'D:\ORACLE\ORADATA\ORA10\TS1.dbf' online
*
ERROR at line 1:
ORA-01113: file 6 needs media recovery
ORA-01110: data file 6: 'D:\ORACLE\ORADATA\ORA10\TS1.DBF'
在线不了，因为我们离线的是单个数据文件

SQL> recover datafile 'D:\ORACLE\ORADATA\ORA10\TS1.DBF';
Media recovery complete.
进行介质恢复
SQL> alter database datafile 'D:\ORACLE\ORADATA\ORA10\TS1.dbf' online;
使数据文件在线
Database altered.

SQL> select name,status from v$datafile;
查看数据文件得名称和状态
NAME                                       STATUS
------------------------------------------ ----------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF        SYSTEM
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF       ONLINE
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF        ONLINE
D:\ORACLE\ORADATA\ORA10\USERS01.DBF         ONLINE
D:\ORACLE\ORADATA\ORA10\TL.DBF              ONLINE
D:\ORACLE\ORADATA\ORA10\TS1.DBF             ONLINE
D:\ORACLE\ORADATA\ORA10\TS1.2               ONLINE

7 rows selected.
```

第二种改法是将整个表空间离线
SQL> alter tablespace ts1 offline;
离线 ts1 表空间，而不是单个数据文件
Tablespace altered.

SQL> select name,status from v$datafile;
查看数据文件得名称和状态
NAME   STATUS
-- -------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF   SYSTEM
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\USERS01.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\TL.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\TS1.DBF   OFFLINE
D:\ORACLE\ORADATA\ORA10\TS1.2   OFFLINE

SQL> host copy D:\ORACLE\ORADATA\ORA10\TS1.2 D:\ORACLE\ORADATA\ORA10\TS1_2.dbf
操作系统的复制

SQL> alter database rename file 'D:\ORACLE\ORADATA\ORA10\TS1.2'
  2 to 'D:\ORACLE\ORADATA\ORA10\TS1_2.dbf';
修改控制文件的指针描述

Database altered.

SQL> alter tablespace ts1 online;
整个表空间在线
Tablespace altered.

SQL> select name,status from v$datafile;
查看数据文件得名称和状态
NAME   STATUS
-- ----------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF   SYSTEM
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\USERS01.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\TL.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\TS1.DBF   ONLINE
D:\ORACLE\ORADATA\ORA10\TS1_2.DBF   ONLINE

7 rows selected.

删除数据文件
只能删除表空间，才能删除数据文件
Drop tablespace ts1 including contents and datafiles;
临时表空间的临时文件除外，你可以删除临时表空间内的临时文件。

我们如何得到文件头的转储文件呢?
ALTER SESSION SET EVENTS 'immediate trace name file_hdrs level 3';
1 控制文件中的文件头信息
2 level 1 + 文件头信息
3 level 2 + 数据文件头信息
这句话将所有数据文件的头都转储到 dump 文件中.

DUMP OF DATA FILES: 8 files in database

```
DATA FILE #1:
  (name #10) D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF
creation size=0 block size=8192 status=0xe head=10 tail=10 dup=1
 tablespace 0, index=1 krfil=1 prev_file=0
 unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
 Checkpoint cnt:624639092 scn: 0x0000.c669eaf1 09/25/2007 22:04:15
 Stop scn: 0xffff.ffffffff 09/25/2007 14:21:40
```

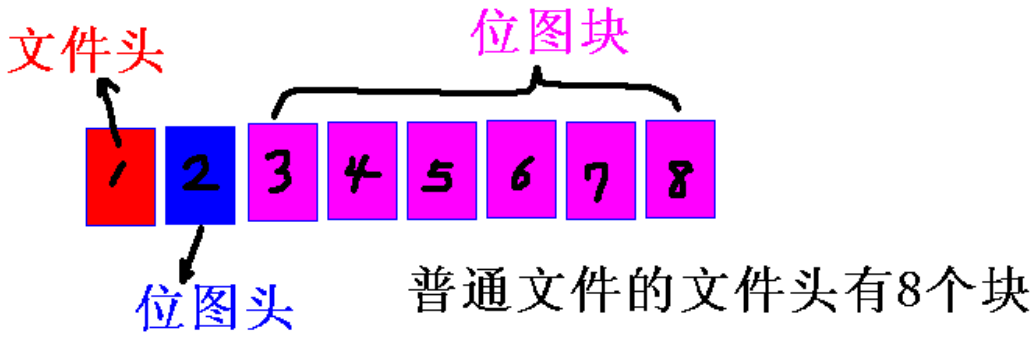
文件头最特殊,使用上面的语法 dump,其他的块使用如下语法.

```
SQL> alter system dump datafile 1 block min 2 block max 4;

System altered.
```

转储文件 1 的 2 到 4 数据块.

```
SQL>  alter system dump datafile 1 block 2;

System altered.
```

转储文件 1 的 2 号数据块.



我上面画的普通的数据文件的文件头的示例图.我们的数据存储在第 9 个块以后.

表空间

System 表空间
数据库内最重要的表空间
在建立数据库时,就诞生了
在数据库 open 的时候必须 online
该表空间含有数据字典的基表
含有包,函数,视图,存储过程的定义
原则上不存放用户的数据

Sysaux 表空间(system auxiliary 辅助)
10g 新引入的新的表空间
分担 system 表空间的压力
一些应用程序的存放数据空间
不能改名称
可以 offline,但部分数据库功能受影响

```
select * from V$SYSAUX_OCCUPANTS;
```

查看有那些应用程序使用了 sysaux 表空间

```
SELECT OCCUPANT_NAME, SCHEMA_NAME, MOVE_PROCEDURE
FROM V$SYSAUX_OCCUPANTS;
```

查看用哪些程序可以更改某些帐号的默认表空间

数据库默认数据表空间(10g 新特性)

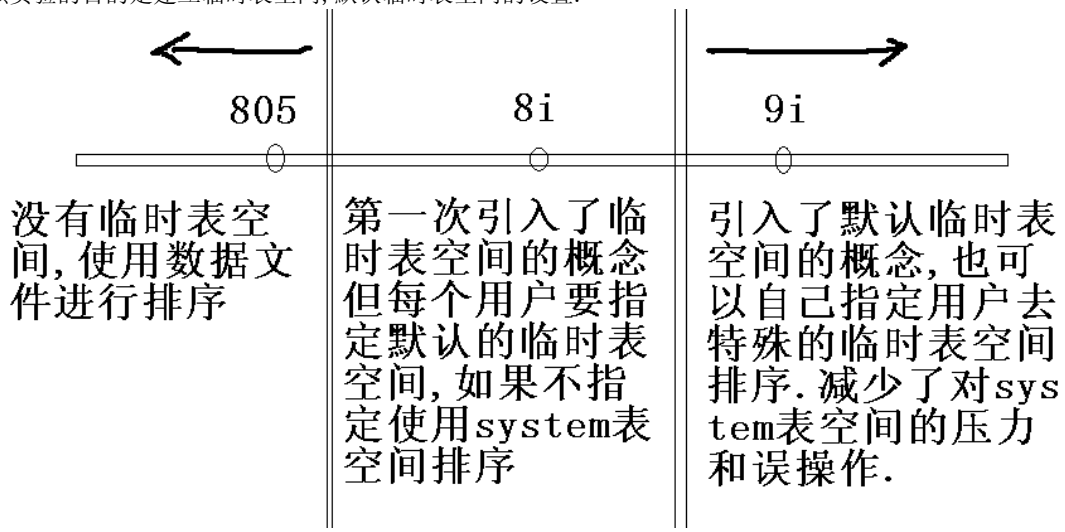
```
SELECT property_value FROM database_properties
```

WHERE property_name =
'DEFAULT_PERMANENT_TABLESPACE';
以前版本的默认表空间为 system,现在可以自己指定.
ALTER DATABASE DEFAULT TABLESPACE newusers;
默认数据表空间不能被删除，想将它删除请先指定别的表空间为默认数据表空间。

临时表空间
不存放永久的对象
用来排序或临时存放数据的
临时表空间的内部分配由 oracle 自动完成
重新启动数据库时该表空间都会重新分配
有排序需求时分配，SHUTDOWN 后回收
数据库内可以有多个临时表空间
select TABLESPACE_NAME,CONTENTS,LOGGING from dba_tablespaces order by 2;

实验 55：建立临时表空间

该实验的目的是建立临时表空间,默认临时表空间的设置.



建立临时表空间
```
CREATE TEMPORARY TABLESPACE temp2 TEMPFILE 'd:\oracle\oradata\ora10\temp02.dbf'
SIZE 20M REUSE
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M
```
查看
V$TEMPFILE 和 DBA_TEMP_FILES

```
SQL> select name,bytes from v$tempfile;
```
验证临时文件
```
NAME                                       BYTES
---------------------------------------- ----------
D:\ORACLE\ORADATA\ORA10\TEMP.TMP             10485760
D:\ORACLE\ORADATA\ORA10\TEMP02.DBF           20971520
SQL> select TABLESPACE_NAME,file_name from dba_temp_files;
```
验证临时文件和临时表空间的关系
```
TABLESPACE_NAME FILE_NAME
--------------- ----------------------------------------
TEMP            D:\ORACLE\ORADATA\ORA10\TEMP.TMP
TEMP2           D:\ORACLE\ORADATA\ORA10\TEMP02.DBF
```

```
SQL> alter tablespace TEMP2 add tempfile 'd:\oracle\oradata\ora10\temp03.dbf' size 3m;
```
增加一个临时文件到指定的临时表空间
```
Tablespace altered.
```

```
SQL> select TABLESPACE_NAME,file_name from dba_temp_files;
```
验证临时文件和临时表空间的关系

```
TABLESPACE_NAME FILE_NAME
--------------- ----------------------------------------
TEMP            D:\ORACLE\ORADATA\ORA10\TEMP.TMP
TEMP2           D:\ORACLE\ORADATA\ORA10\TEMP02.DBF
TEMP2           D:\ORACLE\ORADATA\ORA10\TEMP03.DBF

SQL>  alter tablespace TEMP2 drop tempfile 'd:\oracle\oradata\ora10\temp02.dbf';
```
删除一个临时文件
```
Tablespace altered.

SQL>  select TABLESPACE_NAME,file_name from dba_temp_files;
```
验证临时文件和临时表空间的关系
```
TABLESPACE_NAME FILE_NAME
--------------- ----------------------------------------
TEMP            D:\ORACLE\ORADATA\ORA10\TEMP.TMP
TEMP2           D:\ORACLE\ORADATA\ORA10\TEMP03.DBF
```

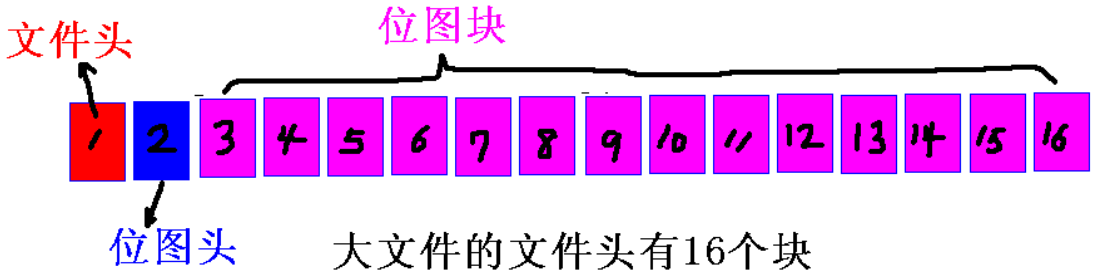
默认临时表空间(9i 新特性)
```
select PROPERTY_VALUE from database_properties
where PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';
```
每个用户可以设定自己的临时表空间
如果没有指定，就用默认临时表空间
默认临时表空间不能被 drop
```
alter database default temporary tablespace temp2;
```
在 8I 前没有默认临时表空间，这钟情况下，数据库使用系统表空间进行排序，所以 8i 对于用户来说使用起来较困难，你必须是对数据库有一定的了解才能更好的使用，在 9i 以后排序的问题受到了重视，临时表空间的地位上升了，有了默认的临时表空间，对于用户来说方便了许多。不会因为大量的排序而增加系统表空间的负担。

实验 56：大文件表空间和表空间的管理模式

该实验的目的是建立支持大文件类型的表空间,表空间的字典管理和本地管理.

Bigfile 大文件表空间（10g 新特性）
```
SQL> create bigfile tablespace bigts datafile 'D:\ORACLE\ORADATA\Ora10\bigts.big' size 2m;
```
建立大文件表空间
```
Tablespace created.

SQL> select TABLESPACE_NAME, BIGFILE from DBA_TABLESPACES;
```
验证表空间的文件属性
```
TABLESPACE_NAME BIG
--------------- ---
SYSTEM          NO
UNDOTBS1        NO
SYSAUX          NO
TEMP            NO
USERS           NO
TEMP2           NO
BIGTS           YES
TP1             NO
TL              NO
TS1             NO
```

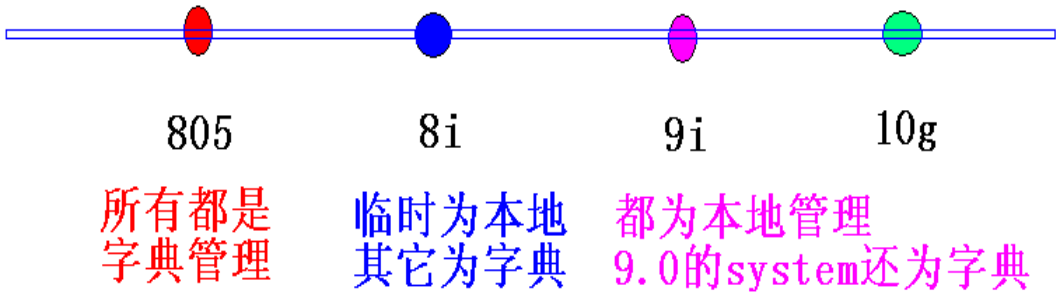
10 rows selected.该类表空间只能含有一个数据文件
该文件可以有 4g 个 oracle 块
普通的 smallfile 文件只能含有 4m 个块,一切都有根源,其根源是 rowid 的限制.
大文件表空间适用于存储大的连续数据，减少控制文件的文件个数，减少文件名称的占用。



我上面画的大数据文件的文件头的示例图.我们的数据存储在第 17 个块以后.

- **表空间的管理模式**

本地管理（local）
字典管理(dictionary)



数据库对每个表空间的剩余空间管理有两种办法，一种是使用数据字典，一种是使用数据文件头。
8i 以前的数据库只有一种管理办法，就是数据字典的管理。如果你想分配空间就必须操作数据字典，给系统表空间带来了较繁重的工作量。随着数据的增加，管理效率低，所以在 8i 诞生了本地管理。普通数据字典管理的文件头大小为一个数据块，而本地管理的表空间文件头有 8 个数据块。在这些多出的数据块中记录着位图，描述的是该数据文件有哪些空闲的空间可以存储数据。本地管理的目的就是解放数据字典，提高管理的效率，因为每次分配空间不操作数据字典，所以没有了数据字典的回退操作，因而少了一系列的操作。本地管理还可以自动的把连续的空闲空间合并，而数据字典管理的表空间需要手工合并。
因为 8i 是第一次引入本地管理的概念，其主要的目的是解决只读数据库的排序问题，所以 8i 的第一个应用是把排序表空间设为本地管理。在 9.0.1 的时候，除了 system 表空间以外的所有表空间都是本地管理。到了 9.2 以后，全部的表空间都是本地管理。9.2 版本以后你想要建立字典管理的表空间很困难，你要重新建立一个 system 表空间为数据字典管理的表空间，并将初始化参数 compatible 降低到 9.0 才可以。
请记住一句话，凡是 oracle 数据库引入的新特性，都为了解决原来不可克服的困难，而且以后都会上升为主导的技术模式。这也是我们学习数据库的难点，如果你一上来就学习 10g，以前的历史发展你都不熟悉，很难学的透彻。知古鉴今，你通过不同版本的对比，你才会明白 oracle 这么做的目的。

非标准块大小的表空间（9i 新特性）
华而不实，我们工作中很少这么做，这是一个技术上的突破，突破了标准块的限制。对我的管理增加了难度，除非性能太差，我们一般不必使用。
必须先设定非标内存大小，因为不同的块大小文件要进入到相应大小块的内存。

```
alter system set DB_4K_CACHE_SIZE=4m;
show parameter cache
```

指明块大小

```
create smallfile tablespace t4k datafile
'D:\ORACLE\ORADATA\O10\t4k.4' size 2m
blocksize 4k;
```

验证

select TABLESPACE_NAME,BLOCK_SIZE from DBA_TABLESPACES;
将来 t4k 表空间要使用内存就从 DB_4K_CACHE_SIZE=4m 中获得。

日志是否产生，我对一些操作不希望产生日志，以便获得更加好的性能，但你也同时失去了数据安全的保护，有一得，必有一失。
create smallfile tablespace tnolog datafile
'D:\ORACLE\ORADATA\O10\tnolog' size 2m NOLOGGING;

select TABLESPACE_NAME,LOGGING,FORCE_LOGGING from dba_tablespaces;
如果强制产生日志 FORCE_LOGGING 则会永远产生日志
即使你在表级指定了 NOLOGGING 的属性
SQL> select TABLESPACE_NAME,LOGGING,FORCE_LOGGING from dba_tablespaces;

TABLESPACE_NAME	LOGGING	FOR
SYSTEM	LOGGING	NO
UNDOTBS1	LOGGING	NO
SYSAUX	LOGGING	NO
TEMP	NOLOGGING	NO
USERS	LOGGING	NO
TEMP2	NOLOGGING	NO
BIGTS	LOGGING	NO
TP1	NOLOGGING	NO
TL	LOGGING	NO
TS1	LOGGING	NO

默认情况下只有临时表空间是不产生日志的，因为临时表空间不需要恢复。

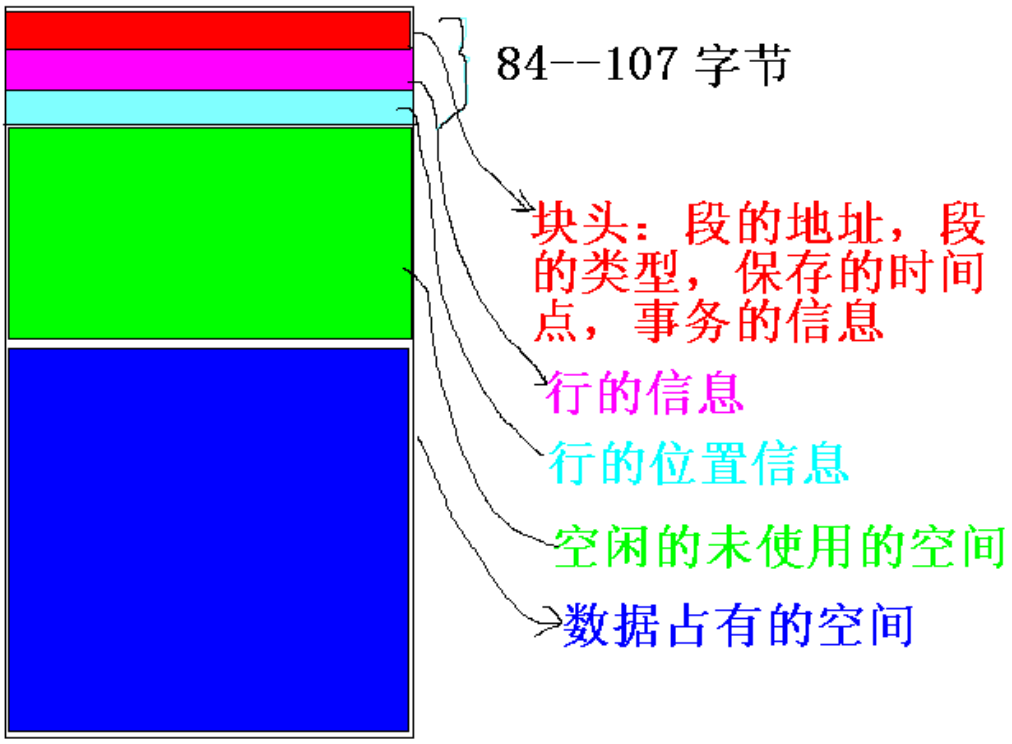
数据库的逻辑结构

Block（块）最基本的存储单元
Extent（范围）一次分配的连续的块
Segment（段）属于同一对象的范围组成一个段
Tablespace（表空间）数据文件的组织行为
```
SQL> select TABLESPACE_NAME,SEGMENT_NAME,EXTENT_ID,BLOCKS
  2  from dba_extents
  3  where SEGMENT_NAME='EMP';
```

TABLESPACE_NAME	SEGMENT_NAME	EXTENT_ID	BLOCKS
USERS	EMP	0	8

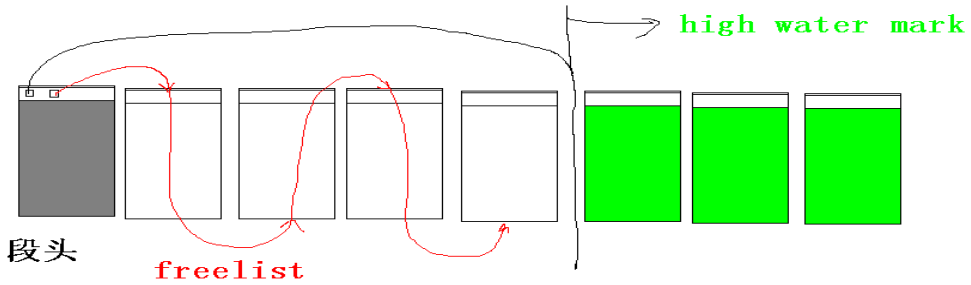
USERS 表空间中有一个 emp 表，emp 表是一个段。Emp 表由一个 0 号范围组成。该范围中有 8 个数据块。
这个字典完美的显示了数据库逻辑体系的四个层次。这四个层次都是逻辑的。

块是建筑数据库的基石。是数据库的最小 i/o 单位。
块的结构如下：



控制块存储的参数
在建表的时候指明，也可以后来修改

典型的表的存储示意图



每个块为 8k,段头我们不能存放数据，空闲列表指向第一个可用的数据块，再连接到下一个可用数据块。
高水位以上是崭新的数据块，一次数据都没有存放。

实验 57：建立表，描述表的存储属性

该实验的目的是理解表的存储结构.会使用到后面的一些概念.

建立指定存储参数的表

```
Conn scott/tiger
Drop table e3 purge;
create table e3 tablespace system pctfree 20 pctused 30
storage( freelists 2)
as select * from emp where 0=9;
```

我们建立了一个空表,为什么要建立到 system 表空间,因为 system 表空间是手工管理的,有空闲列表.

验证

```
SQL> select table_name,pct_free,pct_used,freelists from user_tables;

TABLE_NAME                     PCT_FREE    PCT_USED  FREELISTS
```

```
------------------------------ ---------- ---------- ----------
E3                                     20         30          2
DEPT                                   10
EMP                                    10
BONUS                                  10
SALGRADE                               10
```

为什么只有表 e3 的显示和其它表不一样,因为他们的表空间不同,e3 在 system 表空间,手工管理空闲的块,其它表为 users 表空间,102 以后的版本默认的表空间都为自动管理空闲块.自动管理空闲块是通过位图块来管理的,本质的不同,没有空闲列表.

```
SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
```

没有显示,因为我们没有分析表,我们查看的这些列是静态的,我们每次查看前一定要**先分析,再查看**.

```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.
```

完全分析表,获得该表的详细信息.

```
SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         0            7          0                  0
```

我们看到有 7 个块在高水位以上,因为有一个表头,占用了一个块,因为是空表,空闲列表中没有块.下面我们插入一行.

```
SQL> insert into e3 select * from emp where rownum=1;

1 row created.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.
```

每次查看前要先分析.

```
SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         1            6       8037                  1
```

AVG_SPACE 为使用块的平均空闲空间.我的数据库块为 8192,我们再插入数据,查看发生了什么.

```
SQL> insert into e3 select * from e3;

1 row created.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         1            6       7970                  1
```

下面我们不断的翻倍,分析,查看.

```
SQL> insert into e3 select * from e3;
```

```
2 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         1            6       7892                  1

SQL> insert into e3 select * from e3;

4 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         1            6       7736                  1

SQL> insert into e3 select * from e3;

8 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         1            6       7422                  1

SQL> insert into e3 select * from e3;

16 rows created.
```

```
SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- -------------------
         1            6       6766                   1

SQL> insert into e3 select * from e3;

32 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- -------------------
         1            6       5454                   1

SQL> insert into e3 select * from e3;

64 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- -------------------
         1            6       2830                   1

SQL> insert into e3 select * from e3;

128 rows created.

SQL> commit;
```

```
Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         2            5       2830                  1

SQL> insert into e3 select * from e3;

256 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         4            3       2830                  1

SQL> insert into e3 select * from e3;

512 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
         7            0       2080                  1

SQL> insert into e3 select * from e3;

1024 rows created.

SQL> commit;

Commit complete.
```

```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- -------------------
        15            0       2480                   2
```

Insert 插入操作的详细步骤
1。获得空闲列表
2。找到可以使用的数据块
3。插入到 pctfree 的位置停止插入
4。将该数据块从空闲列表中移走
5。通过指针找到下一个可以使用的数据块
6。继续插入，直到操作完成
步骤 3 留下的空间是给块内的行 update 使用的

Delete 删除操作的详细步骤
块内数据删除
当块内数据大于 pctused 的时候，该数据块不能插入，因为它不在空闲列表中
当块内数据少于 pctused 的时候，将该块加入到空闲列表，可以插入新数据了
我们接着上面的实验.进行删除操作.

```
SQL> update e3 set empno=rownum;

2048 rows updated.
```

为了便于删除,我做了一下修改表的值,为了想删除哪行就删除哪行.

```
SQL> delete e3 where empno<100;

99 rows deleted.
```

将前 99 行删除.这 99 行基本是在一个块当中.

```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- -------------------
        15            0       2739                   3
```

多出了一个空闲块.

```
SQL> delete e3 where mod(empno,10)=1;

195 rows deleted.
```

删除十分之一的行,每个块都删除几行.

```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL>  select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2   from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- -------------------
```

```
        15            0       3246                    3
```
AVG_SPACE 的值加大了,而 NUM_FREELIST_BLOCKS 没有变化.
```
SQL> delete e3 where mod(empno,10)=2;

195 rows deleted.
```
再删除十分之一的行
```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
        15            0       3753                  3

SQL> delete e3 where mod(empno,10)=3;

195 rows deleted.
```
再删除十分之一的行
```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
        15            0       4260                  3

SQL> delete e3 where mod(empno,10)=4;

195 rows deleted.
```
再删除十分之一的行
```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
        15            0       4767                  3

SQL> delete e3 where mod(empno,10)=5;

195 rows deleted.
```
再删除十分之一的行
```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
```

```
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
        15            0       5274                  3

SQL> delete e3 where mod(empno,10)=6;

195 rows deleted.
```
再删除十分之一的行
```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
        15            0       5781                  7

SQL> delete e3 where mod(empno,10)=7;

195 rows deleted.
```
再删除十分之一的行
```
SQL> ANALYZE TABLE E3 COMPUTE STATISTICS;

Table analyzed.

SQL> select BLOCKS,EMPTY_BLOCKS,AVG_SPACE,NUM_FREELIST_BLOCKS
  2  from user_tables where table_name='E3';

    BLOCKS EMPTY_BLOCKS  AVG_SPACE NUM_FREELIST_BLOCKS
---------- ------------ ---------- ------------------
        15            0       6288                 15

SQL> select count(*) from e3;

  COUNT(*)
----------
       584
```
仔细分析上面的实验,对插入和删除的操作和空闲列表的关系就清楚了.

实验 58：数据库范围 extent 的管理

该实验的目的是理解数据库的逻辑体系.

什么是范围，一次分配的，连续的，oracle 块。
建立表时分配新的范围(extent)
```
conn system/manager
grant select any dictionary to scott;
```
使 scott 用户可以查看数据字典。
```
SQL> conn scott/tiger
Connected.
```

```
SQL> drop table t1 purge;
彻底的清除表 t1.
Table dropped.

SQL> create table t1 as select * from emp;
建立新的表 t1.
Table created.

SQL>
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
```

T1 表建立的时候会分配新的空间，这个空间叫做初始范围，即使是一个空表也有初始范围，初始范围和其它以后的范围都不同，因为初始范围中含有表头块，这个数据块中没有我们的数据。首先数据库要找到在哪个表空间中分配空间，如果我们在建立表的时候没有指明，那么就从默认表空间中分配。我们 scott 用户的默认表空间是 users,users 表空间的数据文件是四号文件。所以显示的文件号为 4。
在四号文件中的第 89 个块以后有连续 8 个空闲空间，为什么要分派 8 个数据块，而不是其它数量的数据块呢？这要由 users 表空间的属性来决定。这个属性是在我们建立表空间时决定的。

```
SQL> select TABLESPACE_NAME,INITIAL_EXTENT,NEXT_EXTENT,MIN_EXTENTS,MAX_EXTENTS
  2  from dba_tablespaces;

TABLESPACE_NAME INITIAL_EXTENT NEXT_EXTENT MIN_EXTENTS MAX_EXTENTS
--------------- -------------- ----------- ----------- -----------
SYSTEM                  65536                       1  2147483645
UNDOTBS1                65536                       1  2147483645
SYSAUX                  65536                       1  2147483645
TEMP                  1048576     1048576           1
USERS                   65536                       1  2147483645
TEMP2                 1048576     1048576           1
BIGTS                   65536                       1  2147483645
TP1                   1048576     1048576           1
TL                      65536                       1  2147483645
TS1                     65536                       1  2147483645
```

表中数据增长时分配新的范围(extent)

```
SQL> insert into t1 select * from t1;

14 rows created.
```

增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
```

我们的 t1 表没有分配新的空间，因为 8 个块可以存放所有的行。

```
SQL> insert into t1 select * from t1;

28 rows created.
```

增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
```

```
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                             0          4         89          8
```
我们的 t1 表没有分配新的空间，因为 8 个块可以存放所有的行。

```
SQL> insert into t1 select * from t1;

56 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                             0          4         89          8
```
我们的 t1 表没有分配新的空间，因为 8 个块可以存放所有的行。

```
SQL> insert into t1 select * from t1;

112 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                             0          4         89          8
```
我们的 t1 表没有分配新的空间，因为 8 个块可以存放所有的行。

```
SQL> insert into t1 select * from t1;

224 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                             0          4         89          8
```
我们的 t1 表没有分配新的空间，因为 8 个块可以存放所有的行。

```
SQL> insert into t1 select * from t1;

448 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';
```

```
SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID      BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
T1                            1          4        521          8
```
我们的 t1 表分配新的空间，因为 8 个块不能存放所有的行。必须要有新的空间来存放数据。
为什么是 521 呢？因为在 521 数据块前有其它表的数据。数据库在 521 以后找到了连续的 8 个数据块。

```
SQL> insert into t1 select * from t1;

896 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID      BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
T1                            1          4        521          8
```
我们的 t1 表没有分配新的空间，因为 16 个块可以存放所有的行。
```
SQL> insert into t1 select * from t1;

1792 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID      BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
T1                            1          4        521          8
T1                            2          4        537          8
T1                            3          4        545          8
```
我们的 t1 表分配新的空间，因为 16 个块不能存放所有的行。必须要有新的空间来存放数据。
为什么是 537 呢？因为在 537 数据块前有其它表的数据。数据库在 537 以后找到了连续的 8 个数据块。
空间还是不够，又分配了新的范围。在 545 数据块后又分配了 8 个块。

```
SQL> insert into t1 select * from t1;

3584 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID      BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
T1                            1          4        521          8
T1                            2          4        537          8
T1                            3          4        545          8
T1                            4          4        553          8
T1                            5          4        561          8
```

```
T1                                6          4        569          8
```
又分配了 3 个新的范围。
```
SQL> insert into t1 select * from t1;

7168 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID      BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
T1                            1          4        521          8
T1                            2          4        537          8
T1                            3          4        545          8
T1                            4          4        553          8
T1                            5          4        561          8
T1                            6          4        569          8
T1                            7          4        577          8
T1                            8          4        585          8
T1                            9          4        593          8
T1                           10          4        601          8
T1                           11          4        609          8
```
又分配了 4 个新的范围。

```
SQL> insert into t1 select * from t1;

14336 rows created.
```
增加一倍的数据。

```
SQL> select SEGMENT_NAME,EXTENT_ID,FILE_ID,
  2  BLOCK_ID,BLOCKS from dba_extents
  3  where OWNER='SCOTT' AND SEGMENT_NAME='T1';

SEGMENT_NAME          EXTENT_ID    FILE_ID   BLOCK_ID      BLOCKS
-------------------- ---------- ---------- ---------- ----------
T1                            0          4         89          8
T1                            1          4        521          8
T1                            2          4        537          8
T1                            3          4        545          8
T1                            4          4        553          8
T1                            5          4        561          8
T1                            6          4        569          8
T1                            7          4        577          8
T1                            8          4        585          8
T1                            9          4        593          8
T1                           10          4        601          8
T1                           11          4        609          8
T1                           12          4        617          8
T1                           13          4        625          8
T1                           14          4        633          8
T1                           15          4        641          8
T1                           16          4        137        128
```

为什么最后一个范围的大小是 128 个数据块，而不是 8 个了。因为数据库已经分配了 16 个范围，共分配了 1m 的空间，我们还要求分配新的空间，数据库认为这个表很大，所以就一次分了 128 个数据块。如果我们再增加表的大小，数据库会分配多个 1m,以后就 8m 为大小分配，再以后就 64m 大小分配。总之是一个原则，

我们现有的数据越大，未来的范围就越大。

我们手工分配范围
```
 alter table t1 allocate extent;
alter table t1 allocate extent(datafile 'D:\ORACLE\ORADATA\O10\USERS01.DBF' size 9k);
```

手工回收未使用的范围
```
alter table t1 deallocate unused;
```
查看结果
```
select SEGMENT_NAME,EXTENT_ID,FILE_ID,
BLOCK_ID,BLOCKS from dba_extents
where OWNER='SCOTT' AND SEGMENT_NAME='T1';
```

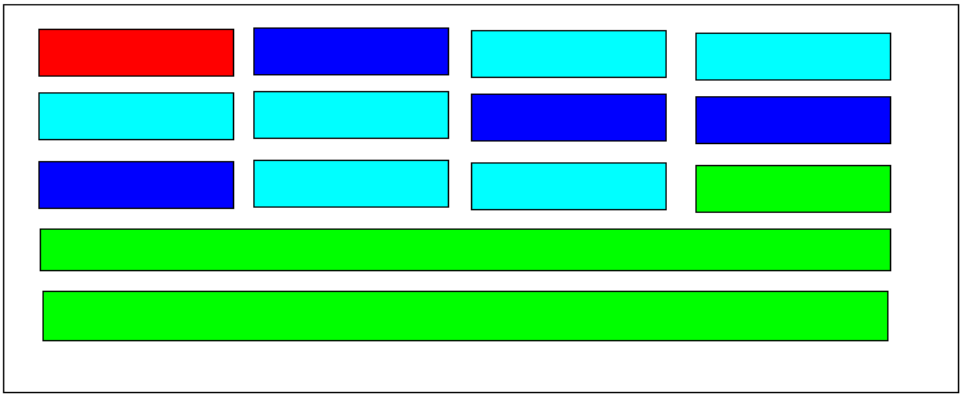
Truncate table
```
Truncate table t1;
```
Ddl 语言，自动提交
不能回退
回收范围
挪动高水位线
将所有的数据清除，保留表结构
将表缩的最小
保留表的约束和权限

Drop table
```
Drop table t1;
```
不释放空间
```
Purge table t1;
```
释放空间

```
Drop table t1 purge;
```
删除表同时清空回收站

delete t1;	truncate table t1;	drop table t1 purge;
dml语言	ddl语言	ddl语言
可以回退	不可以回退	不可以回退
手工提交	自动提交	自动提交
写大量日志	写很少日志	写很少日志
写大量回退	写很少回退	写很少回退
占用大量内存	占用很少内存	占用很少内存
不释放空间	释放空间	释放空间
不挪动高水位	挪动高水位	挪动高水位
可以闪回到历史	不能闪回到历史	不能闪回到历史
	保留最小的初始范围	彻底的删除

三种回收方式的不同层次



红的为初始范围　蓝的为存在数据的范围
浅蓝为曾经有数据，现在空闲的范围
绿的为从来未装过数据的新范围
手工回收，回收的是绿的范围
truncate table，回收的是除了红的以外的范围
drop table ，回收的是全部范围

分配空间
1。建立表
2。长大的时候
3。手工分配

回收空间
1。手工回收
2。truncate
3。drop

表空间内的空闲空间
数据库总的可分配空间

```
select tablespace_name,sum(ceil(bytes/1024/1024)) free_mb
from dba_free_space
group by tablespace_name;
```

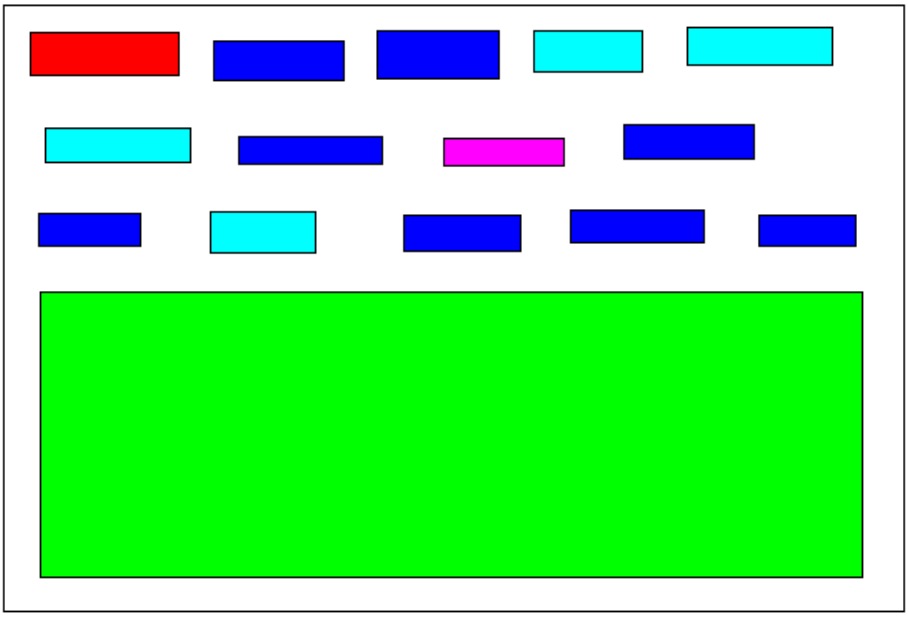
USERS 表空间可分配空间

```
select tablespace_name,ceil(bytes/1024/1024) free_mb
from dba_free_space where tablespace_name='USERS';
```

USERS 表空间可分配空间的详细分布

```
select FILE_ID,BLOCK_ID,BLOCKS from dba_free_space
where tablespace_name='USERS';
```

图为一个数据文件，红色为文件头（8 个块）
深蓝为分配给用户的范围；浅蓝为曾经分配，现在回收的范围；粉色为一次都未分配的碎片范围；绿的为崭新的数据文件部分，从来未被使用过的空间。



Dba_data_free 查看的范围为
蓝+粉+绿
RMAN 工具备份空间为
除了绿色以外的全都备份

自动的段空间管理（9i 新特性）
一种用位图块来管理空闲空间的办法
替代 freelist（空闲列表管理，8i 前的唯一方式）
更加高效
要在建立表空间的时候声明

默认表空间管理模式为自动

```
create tablespace ts3 datafile 'D:\ORACLE\ORADATA\010\TS3.dbf'
size 2m SEGMENT SPACE MANaGEMENT AUTO;
```

验证

```
select TABLESPACE_NAME,SEGMENT_SPACE_MANAGEMENT from  dba_tablespaces;
```

建立手工管理的表空间

```
create tablespace ts4 datafile 'D:\ORACLE\ORADATA\010\TS4.dbf'
size 2m SEGMENT SPACE MANaGEMENT manual;
```

undo 段的管理

实验 59：数据库自动回退段的管理

该实验的目的是理解回退的作用,维护自动管理模式的回退段.

回退段 9i 前叫 rollback
回退段 9i 后叫 undo
回退段存放事务影响过的数据
回退段有手工和自动两种管理方式

回退段的四大作用

交易的回退:没有提交的交易可以后悔.
交易的恢复:数据库崩溃的时候,将写入磁盘的不正确数据恢复到交易前.
读一致性:查询时结果集已经确定.
闪回数据:从回退段中构造历史的数据.

系统回退段
存在于 system 表空间
不受我们控制
只有 system 表空间内的对象才可以使用

```
select * from v$rollname;
USN NAME
---- ------------
   0 SYSTEM
   1 _SYSSMU1$
   2 _SYSSMU2$
   3 _SYSSMU3$
   4 _SYSSMU4$
   5 _SYSSMU5$
   6 _SYSSMU6$
   7 _SYSSMU7$
   8 _SYSSMU8$
   9 _SYSSMU9$
  10 _SYSSMU10$

show parameter undo
undo_management   AUTO
undo_tablespace       UNDOTBS1
```

建立 undo 表空间

```
create undo tablespace undo2 datafile 'D:\ORACLE\ORADATA\010\undo2.dbf' size 2m;
```

验证

```
select SEGMENT_NAME,OWNER,TABLESPACE_NAME,
STATUS from dba_rollback_segs;
```

切换

```
Alter system set undo_tablespace =undo2;
```

当前在线的回退段

```
Select * from v$rollname;
```

事务所使用的回退段

```
 update scott.emp set sal=sal+1;
select XIDUSN  from V$transaction;
 select * from v$rollname;
```

随机选择一个使用最少的回退段
如果有空间，会自动的建立新的回退段
数据库停止后，新分配的被回收

延迟回退
当有未提交的事务时切换回退
这时已经切换到新的回退表空间
但因为事务在未完成的时候不会更换回退段
所以原来的事务所在的回退段处于延迟状态

```
select usn,status from v$rollstat;
```

估算所需回退的总大小
1。先计算平均每秒产生的回退 v$undostat
2。show parameter undo 查看

```
            undo_retention         900
```

3。回退所需总空间为 1*2

删除回退表空间
1。先切换到其它回退表空间
2。等待事务结束
3。Drop tablespace undo2;

如何 dump 回退段的信息呢?
dump 回退段的头信息
ALTER SYSTEM DUMP UNDO_HEADER 'segment_name';

dump 回退段的交易信息
ALTER SYSTEM DUMP UNDO BLOCK 'segment_name' XID xidusn xidslot xidsqn;

实验 60：数据库手工回退段的管理

该实验的目的是理解数据库手工管理回退段管理的模式.

1.undo_management =manual;
2.startup force;
3.建立新的回退段
    create rollback segment r1 tablespace undo2;
4.online
    alter rollback segment r1 online;
5.查看   select * from v$rollname;
6.offline   alter rollback segment r1 offline;
7.删除   drop rollback segment r1;
手工管理很好,一切都由我们控制,多少,大小,指定事务用指定的回退.

实验 61：通过回退段闪回历史数据

该实验的目的是使用回退的四大作用之一,闪回历史的数据
                                    通过回退段来闪回老交易的数据
闪回到指定的 scn 点
--当前系统的 SCN 号
select DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER() from dual;
改变表的数据，提交
update scott.emp set sal=sal+1;
Commit;
闪回
execute DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_number (#####);
--结束闪回
execute dbms_flashback.disable();

闪回到指定的时间点
物理时间和数据库间的 SCN 的对照表，每五分钟采样
select to_char(TIME_DP,'yyyy/mm/dd:hh24:mi:ss'),SCN from SYS.SMON_SCN_TIME;

Execute dbms_flashback.enable_at_time(to_date('2004/11/24:16:20','yyyy/mm/dd:hh24:mi:ss'))

闪回---取值到游标----停止闪回----将游标中的值插入原表
declare
cursor c1 is select * from scott.e2 where empno=7369;
v_sal c1%rowtype;

```
begin
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_number (13346413);
open c1;
fetch c1 into v_sal;
dbms_flashback.disable();
update scott.e2 set sal=v_sal.sal where empno=v_sal.empno;
close c1;
end;
/
```

实验 62：闪回数据的查询方法，以及历史交易

该实验的目的是查询一张表的历史数据变化的痕迹.在回退段中查询.
闪回版本查询（10g 新特性）

```
conn scott/tiger
drop table t1 purge;
create table t1 as select ename,sal from emp where empno=7900;

update t1 set sal=1000;
commit;
update t1 set sal=2000;
commit;
```

产生一些交易

```
insert into t1 select ename,sal from emp where empno=7900;
commit;
insert into t1 select ename,sal from emp where empno=7902;
commit;
insert into t1 select ename,sal from emp where empno=7839;
commit;
update t1 set sal=3000 where empno=7902;
commit;

col VERSIONS_STARTTIME for a25
col VERSIONS_ENDTIME for a25

select versions_starttime, versions_endtime, versions_xid,
versions_operation, sal
from t1
 versions between timestamp minvalue and maxvalue
order by VERSIONS_STARTTIME;
```

取一个时间段的版本

```
select versions_starttime, versions_endtime, versions_xid, versions_operation, rate from rates
versions between timestamp to_date('12/11/2003 15:57:52', 'mm/dd/yyyy hh24:mi:ss') and maxvalue
order by VERSIONS_STARTTIMe ;
```

查看原 sql 语句

```
SELECT UNDO_SQL FROM FLASHBACK_TRANSACTION_QUERY WHERE XID = '08001100C7010000';
```

多次 drop 后，建立同名称表

```
conn scott/tiger
drop table t1 purge;
create table t1 as select ename,sal from emp where empno=7900;
drop table t1;
create table t1 as select ename,sal from emp where empno=7902;
```

```
drop table t1;
create table t1 as select ename,sal from emp where empno=7839;

show recycl
FLASHBACK TABLE T1 TO BEFORE DROP RENAME TO T8;
FLASHBACK TABLE T1 TO BEFORE DROP RENAME TO T9;
```

知识点
在回退段中获取以前的数据
在回收站里恢复表并更改名称
1。手工指定到闪回的 scn
2.利用伪列，进行查询
3。将回收站的表，复原并更改名称

表—存储数据的最基本单元

建立表的原则
设计表的时候表的名称，列名称，索引，约束等名称要统一
表和列要加注释
将会出现 null 值的列放在最后，最后连续的 null 都不占空间
定义约束，维护数据的完整性
尽量使用 cluster 类型的表，使存储最少，sql 语句最优

实验 63：rowid 的含义,位图块和空闲列表对比

该实验的目的是通过 rowid 来获得表的存储信息.

- **Rowid 行标识**

```
select rowid,ename from emp;
AAAMfP      AAE  AAAAAg     AAA
对象代码   文件    块         行
```

为了能描述更多的行,rowid 是 64 进制的,由 0-9，a-z,A-Z，+，/ 组成

ROWID 是伪列，计算出来的，索引使用它,表中并没有存储 rowid,每一行的 rowid 是根据该行的物理位置计算出来的,我们得到一个 rowid 可以获得该行的所在的物理位置,可以快速定位该行.我们也可以将指定的数构造出一个 rowid,

分解 ROWID

```
SQL> conn scott/tiger
Connected.
SQL> SELECT ENAME,ROWID,
  2  DBMS_ROWID.ROWID_OBJECT(ROWID) OBJECT#,
  3  DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID) FILE#,
  4  DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) BLOCK#,
  5  DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)  ROW# FROM EMP;

ENAME      ROWID                 OBJECT#      FILE#     BLOCK#       ROW#
---------- ------------------ ---------- ---------- ---------- ----------
SMITH      AAANNhAAEAAAAAgAAA      54113          4         32          0
ALLEN      AAANNhAAEAAAAAgAAB      54113          4         32          1
WARD       AAANNhAAEAAAAAgAAC      54113          4         32          2
JONES      AAANNhAAEAAAAAgAAD      54113          4         32          3
MARTIN     AAANNhAAEAAAAAgAAE      54113          4         32          4
BLAKE      AAANNhAAEAAAAAgAAF      54113          4         32          5
CLARK      AAANNhAAEAAAAAgAAG      54113          4         32          6
SCOTT      AAANNhAAEAAAAAgAAH      54113          4         32          7
```

```
KING        AAANNhAAEAAAAAgAAI      54113          4          32          8
TURNER      AAANNhAAEAAAAAgAAJ      54113          4          32          9
ADAMS       AAANNhAAEAAAAAgAAK      54113          4          32         10
JAMES       AAANNhAAEAAAAAgAAL      54113          4          32         11
FORD        AAANNhAAEAAAAAgAAM      54113          4          32         12
MILLER      AAANNhAAEAAAAAgAAN      54113          4          32         13
```

构造 rowid

```
SQL> select dbms_rowid.rowid_create(1,54113,4,32,7) from dual;
```

参数 1 代表是新版本的 rowid 格式,64 进制的.18 位

```
DBMS_ROWID.ROWID_CREATE(1,5411
----------------------------------
AAANNhAAEAAAAAgAAH

SQL> select dbms_rowid.rowid_create(0,54113,4,32,7) from dual;
```

参数 0 代表是老版本的 rowid 格式,16 进制的.16 位,oracle8 以前的 rowid 是以老版本描述的.

```
DBMS_ROWID.ROWID_CREATE(0,5411
----------------------------------
00000020.0007.0004
```

八位的块,四位的行,四位的文件,将 20 转为 10 进制是 32,上面的 rowid 代表着 4 号文件的,第 32 个块的,第 7 行.我们将最大的 8 位 16 进制转化为十进制.刚好为 4m.所以一个数据文件大小的上限为 4m 个 oracle 块.

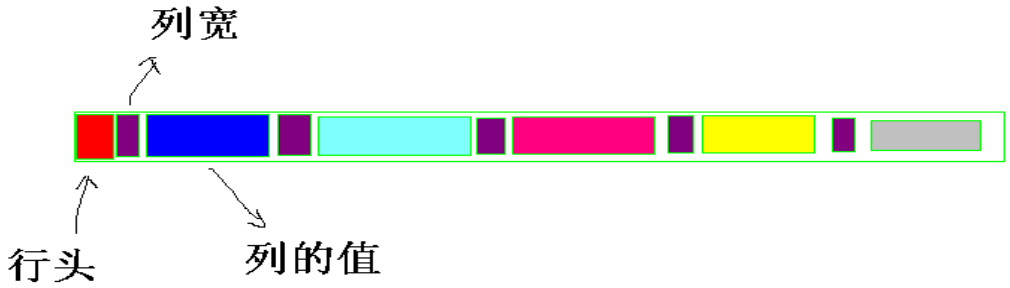
```
SQL> select to_number('ffffffff','xxxxxxxxxxxx') from dual;

TO_NUMBER('FFFFFFFF','XXXXXXXX
------------------------------
                    4294967295
```

我们通过 rowid 可以判断行是如何分布在每个数据块当中的.

```
SQL> drop table t1 purge;

Table dropped.

SQL> create table t1 as select * from emp;

Table created.

SQL> insert into t1 select * from t1;

14 rows created.

SQL> /

28 rows created.

SQL> /

56 rows created.

SQL> /

112 rows created.

SQL> /

224 rows created.

SQL> commit;
```

```
Commit complete.
```
我们建立了一个 400 多行的实验表 t1.
```
SQL> SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) BLOCK#,COUNT(*)
  2  FROM T1
  3  GROUP BY DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID);

    BLOCK#   COUNT(*)
---------- ----------
        95        168
        94         42
        96         56
        93        168
        92         14
```
我们看到最多可以存放 168 行,根据每行的行长不同,块中存放的行数也不同,行长就可以存储少一些,行短就存储多一些.

行是紧密的码放在块中，行头存放锁的信息



验证每个数据块可以存储的最大行数
```
CONN SCOTT/TIGER
SQL> DROP TABLE T4 PURGE;

Table dropped.

SQL> CREATE TABLE T4 (c varchar2(1)) pctfree 0;

Table created.

SQL> INSERT INTO T4 VALUES(null);

1 row created.

SQL> INSERT INTO T4 SELECT * FROM T4;

1 row created.

SQL> /

2 rows created.
SQL> /

1024 rows created.

SQL> commit;

Commit complete.
```
--到 2000 行,我们建立了一个表 t4,该表的每一行都存储的 null 值,可以说是最短的行了.
pctfree 0 的含义是每个块都放满数据不留空间.

```
SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) BLOCK#,COUNT(*)
FROM T4 GROUP BY DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID);
BLOCK#   COUNT(*)
------ ----------
   100        582
   104        733
   103        733
```

我们看到每个块即使存储空行,也只能存储 733 行.

将数据块存储到平面文件

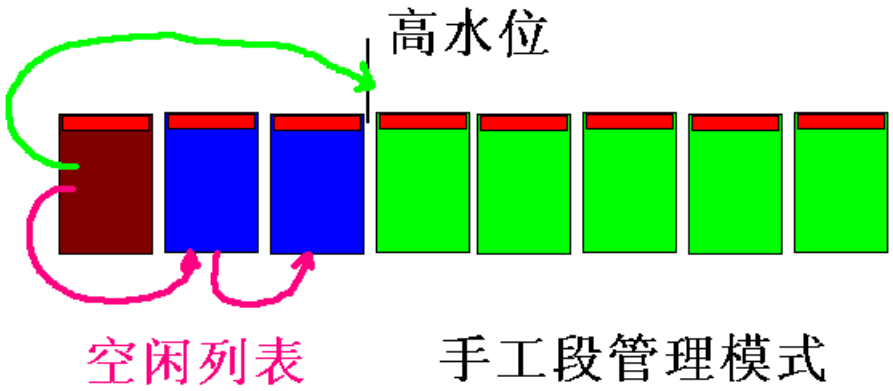
```
alter system  dump datafile 4 block 32;
show parameter  user_dump_dest
```

- **位图块和空闲列表对比**

当表空间使用手工段管理的时候,段的块管理模式是空闲列表模式. 当表空间使用自动段管理的时候,段的块管理模式是位图块模式.

```
SQL> select TABLESPACE_NAME,SEGMENT_SPACE_MANAGEMENT from dba_tablespaces
  2  order by 2;

TABLESPACE_NAME SEGMENT_SPAC
--------------- ------------
T2M             AUTO
TL              AUTO
USERS           AUTO
BIGTS           AUTO
SYSAUX          AUTO
TP1             MANUAL
TEMP1           MANUAL
TEMP            MANUAL
UNDOTBS1        MANUAL
SYSTEM          MANUAL
TEMP2           MANUAL
```

上半部分为手工段管理模式,下半部分为位图块管理模式,管理的是段内的块.
手工段管理模式是一直存在的模式, 位图块管理模式是 9i 的新特性,在 10.2 版以后,建立的表空间默认就是自动段管理.所以我们可以看出,自动段管理有性能的优势.
空闲列表存在于第一个块,我们怎么知道的呢?通过 rowid 可以发现,在手工管理的表中,数据总是从第二个块开始存放.



当我们使用自动段管理模式的时候,情况又如何呢?我们通过 ROWID 可以看到详细的行分布情况,我们会发现每次数据都是从第四个块开始存放,那么前三个数据块干什么用了呢?是巧合还是必然.我看了很多个表,都是一样的,没有一个是例外.我们就将数据块转存到 dump 文件,仔细研究一下.

```
ALTER SYSTEM DUMP datafile  4 block 9;
ALTER SYSTEM DUMP datafile  4 block 10;
ALTER SYSTEM DUMP datafile  4 block 11;
```

4 号文件在默认建立数据库的时候是 users 表空间,9 到 15 之间的数据块存储的是 dept 表.
9 号数据块的部分内容:

```
Start dump data blocks tsn: 4 file#: 4 minblk 9 maxblk 9
buffer tsn: 4 rdba: 0x01000009 (4/9)
scn: 0x0000.c6682f1c seq: 0x02 flg: 0x04 tail: 0x2f1c2002
frmt: 0x02 chkval: 0xd57e type: 0x20=FIRST LEVEL BITMAP BLOCK
```

该块为一级位图块.

10 号数据块的部分内容:

```
Start dump data blocks tsn: 4 file#: 4 minblk 10 maxblk 10
buffer tsn: 4 rdba: 0x0100000a (4/10)
scn: 0x0000.c6682eec seq: 0x02 flg: 0x04 tail: 0x2eec2102
frmt: 0x02 chkval: 0x9528 type: 0x21=SECOND LEVEL BITMAP BLOCK
```

该块为二级位图块.

11 号数据块的部分内容:

```
Start dump data blocks tsn: 4 file#: 4 minblk 11 maxblk 11
buffer tsn: 4 rdba: 0x0100000b (4/11)
scn: 0x0000.c6682f1c seq: 0x02 flg: 0x04 tail: 0x2f1c2302
frmt: 0x02 chkval: 0xb88b type: 0x23=PAGETABLE SEGMENT HEADER
Highwater::  0x01000011  ext#: 0      blk#: 8      ext size: 8
```

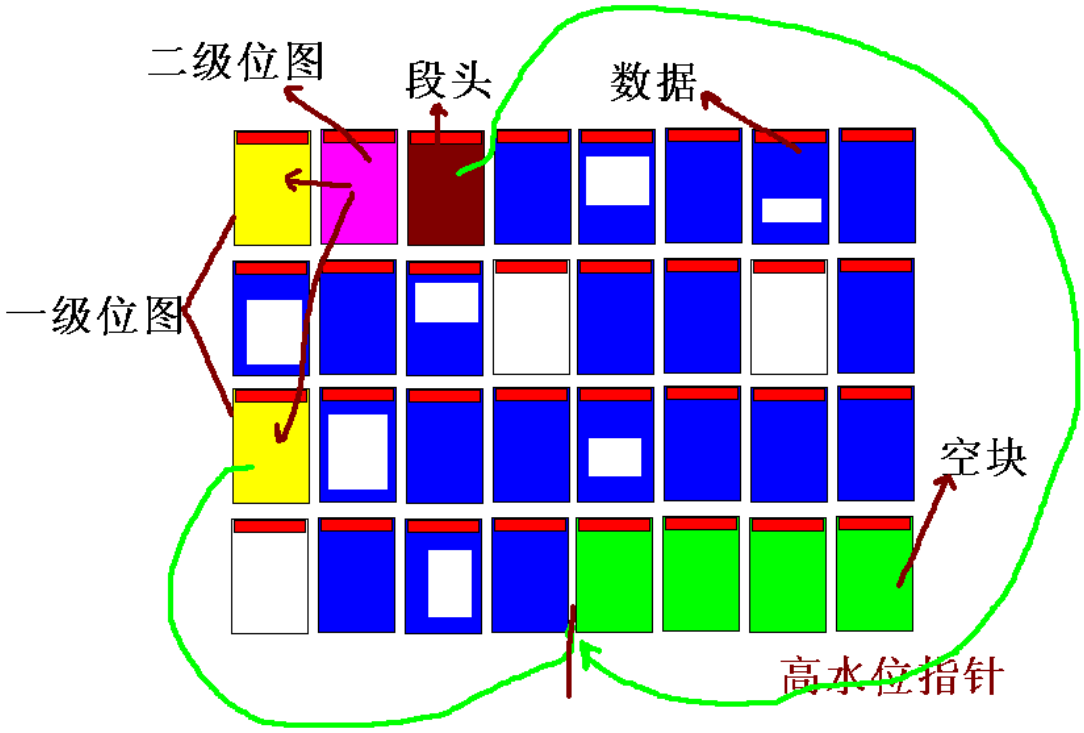
该块为段头块.存储了高水位标记.

```
SQL> select OWNER,SEGMENT_NAME,SEGMENT_TYPE,HEADER_BLOCK
    from dba_segments where SEGMENT_NAME='DEPT';

OWNER      SEGMENT_NAME          SEGMENT_TYPE HEADER_BLOCK
---------- --------------------- ------------ ------------
SCOTT      DEPT                  TABLE                  11
```

这句话表明段头在 11 块上.

```
SQL> SELECT SEGMENT_NAME,EXTENT_ID,BLOCK_ID,BLOCKS FROM DBA_EXTENTS
     where SEGMENT_NAME='DEPT';

SEGMENT_NAME           EXTENT_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ----------
DEPT                          0          9          8
```

这句话表明这个表存储空间的分布为 9 到 15 共 8 个数据块.
一切都真相大白了.



位图块为什么要分级别,很简单,位图块的使用和索引的原理相同,一定要有根.所有的位图块是一个**树状的结构**.这样数据库才会迅速的找到所有的位图块.迅速的分配空间.自动段的管理**高水位**既存在于最后一个一级位图块中,**又**存在于段头中.一级位图块会有很多个,因为分配空间由位图块决定,所以在位图块中要有高水位的信息.
想验证上面的图形,用到 rowid 来定位行,alter system dump datafile ## block ##;来获得块的信息.
一个位图块最多存储 64 个块的信息,每个 128 个块的范围的头两个块都是位图块.当然我们的实验是 8k 大小的块,我估计 16k 大小的块会存在一些变化,这些都无所谓了.当**范围大小为 8 个块**的时候,每两个范围使用一个位图块,每个位图块中描述 16 个块的信息.

下面显示的是一个二级位图块和两个一级位图块的转存 dump 文件.
我们建立一个五万行的大表 t1,再删除部分的数据.

```
SQL>  SELECT SEGMENT_NAME,EXTENT_ID,BLOCK_ID,BLOCKS FROM DBA_EXTENTS
  2     where SEGMENT_NAME='T1';
```

SEGMENT_NAME	EXTENT_ID	BLOCK_ID	BLOCKS
T1	0	57	8
T1	1	65	8
T1	2	105	8
T1	3	113	8
T1	4	121	8
T1	5	521	8
T1	6	537	8
T1	7	545	8
T1	8	553	8
T1	9	561	8
T1	10	569	8
T1	11	577	8
T1	12	585	8
T1	13	593	8
T1	14	601	8
T1	15	609	8
T1	16	137	128
T1	17	265	128

从范围的分布我们可以看出,58 号块为二级位图块,是所有一级位图块的根.而 266 为最后一个一级位图块,其中含有高水位的标记.其中红颜色的为一级位图块.59 号块为段头.

58 号块

```
Start dump data blocks tsn: 4 file#: 4 minblk 58 maxblk 58
buffer tsn: 4 rdba: 0x0100003a (4/58)
scn: 0x0000.c66bfc3b seq: 0x01 flg: 0x04 tail: 0xfc3b2101
frmt: 0x02 chkval: 0x95de type: 0x21=SECOND LEVEL BITMAP BLOCK
Hex dump of block: st=0, typ_found=1
Dump of memory from 0x07C62200 to 0x07C64200
7C62200 0000A221 0100003A C66BFC3B 04010000  [!...:...;.k.....]
7C62210 000095DE 00000000 00000000 00000000  [................]
7C62220 00000000 00000000 00000000 00000000  [................]
        Repeat 1 times
7C62240 00000000 00000000 00000000 0100003B  [............;...]
7C62250 0000000C 0000000C 00000000 00000000  [................]
7C62260 00000000 00000000 0000D6A2 00000001  [................]
7C62270 00000000 01000039 00010003 01000069  [....9.......i...]
7C62280 00010003 01000079 00010003 01000219  [....y...........]
7C62290 00010003 01000229 00010003 01000239  [....).......9...]
7C622A0 00010003 01000249 00010003 01000259  [....I.......Y...]
7C622B0 00010003 01000089 00010003 0100008A  [................]
7C622C0 00010003 01000109 00010003 0100010A  [................]
7C622D0 00010005 00000000 00000000 00000000  [................]
7C622E0 00000000 00000000 00000000 00000000  [................]
        Repeat 496 times
7C641F0 00000000 00000000 00000000 FC3B2101  [............!;.]
Dump of Second Level Bitmap Block
   number: 12      nfree: 12      ffree: 0      pdba:      0x0100003b
   Inc #: 0 Objd: 54946
  opcode:0
 xid:
  L1 Ranges :
  ------------------------------------------------------
   0x01000039  Free: 3 Inst: 1      注释:转化为十进制为 57
   0x01000069  Free: 3 Inst: 1      注释:转化为十进制为 105
   0x01000079  Free: 1 Inst: 1      注释:转化为十进制为 121
   0x01000219  Free: 1 Inst: 1      注释:转化为十进制为 537
   0x01000229  Free: 1 Inst: 1      注释:转化为十进制为 553
   0x01000239  Free: 3 Inst: 1      注释:转化为十进制为 569
   0x01000249  Free: 3 Inst: 1      注释:转化为十进制为 585
   0x01000259  Free: 3 Inst: 1      注释:转化为十进制为 601
   0x01000089  Free: 3 Inst: 1      注释:转化为十进制为 137
   0x0100008a  Free: 3 Inst: 1      注释:转化为十进制为 138
   0x01000109  Free: 3 Inst: 1      注释:转化为十进制为 265
   0x0100010a  Free: 5 Inst: 1      注释:转化为十进制为 266
  我们这里可以看出总共有 12 个一级位图块.这是所有位图块的根块.
其中位图块的分布正好和我们推理的一致.
其中 free 1 代表有 0%的空间可以使用
,free 2 代表有 0%-25%的空间可以使用,
,free 3 代表有 25%-50%的空间可以使用,
,free 4 代表有 50%-75%的空间可以使用,
free 5 代表有 100%的空间可以使用.

  ------------------------------------------------------
End dump data blocks tsn: 4 file#: 4 minblk 58 maxblk 58
*****************************************************************************
```

265 号块

```
Start dump data blocks tsn: 4 file#: 4 minblk 265 maxblk 265
buffer tsn: 4 rdba: 0x01000109 (4/265)
scn: 0x0000.c66bfc7b seq: 0x01 flg: 0x04 tail: 0xfc7b2001
frmt: 0x02 chkval: 0x97c1 type: 0x20=FIRST LEVEL BITMAP BLOCK
Hex dump of block: st=0, typ_found=1
Dump of memory from 0x07C62200 to 0x07C64200
7C62200 0000A220 01000109 C66BFC7B 04010000  [ .......{.k.....]
7C62210 000097C1 00000000 00000000 00000000  [................]
7C62220 00000000 00000000 00000000 00000000  [................]
        Repeat 1 times
7C62240 00000000 00000000 00000000 00000004  [................]
7C62250 FFFFFFFF 00000000 00000002 00000040  [............@...]
7C62260 00010001 00000000 0000003E 00000000  [........>.......]
7C62270 00000000 00000002 4725C48F 4725C48F  [..........%G..%G]
7C62280 00000000 00000000 00000000 00000000  [................]
7C62290 0100003A 0000000A 00000000 00000000  [:...............]
7C622A0 00000000 00000000 00000000 00000000  [................]
        Repeat 1 times
7C622C0 0000D6A2 00000000 00000000 01000109  [................]
7C622D0 00000040 00000000 00000000 00000000  [@...............]
7C622E0 00000000 00000000 00000000 00000000  [................]
        Repeat 9 times
7C62380 00000000 00000000 00000000 33333311  [............333]
7C62390 33333333 33333333 33333333 33333333  [3333333333333333]
7C623A0 33333333 33333333 33333333 00000000  [333333333333....]
7C623B0 00000000 00000000 00000000 00000000  [................]
        Repeat 483 times
7C641F0 00000000 00000000 00000000 FC7B2001  [............ {.]
Dump of First Level Bitmap Block
 --------------------------------
   nbits : 4 nranges: 1         parent dba:  0x0100003a   poffset: 10
   unformatted: 0       total: 64        first useful block: 2
   owning instance : 1
   instance ownership changed at 10/29/2007 19:31:27
   Last successful Search 10/29/2007 19:31:27
   Freeness Status:  nf1 0      nf2 62     nf3 0      nf4 0

   Extent Map Block Offset: 4294967295
   First free datablock : 2
   Bitmap block lock opcode 0
   Locker xid:     :  0x0000.000.00000000
   Inc #: 0 Objd: 54946
  ------------------------------------------------------
  DBA Ranges :
  ------------------------------------------------------
   0x01000109  Length: 64     Offset: 0

   0:Metadata   1:Metadata   2:25-50% free   3:25-50% free
   4:25-50% free   5:25-50% free   6:25-50% free   7:25-50% free
   8:25-50% free   9:25-50% free   10:25-50% free   11:25-50% free
   12:25-50% free   13:25-50% free   14:25-50% free   15:25-50% free
   16:25-50% free   17:25-50% free   18:25-50% free   19:25-50% free
   20:25-50% free   21:25-50% free   22:25-50% free   23:25-50% free
   24:25-50% free   25:25-50% free   26:25-50% free   27:25-50% free
   28:25-50% free   29:25-50% free   30:25-50% free   31:25-50% free
   32:25-50% free   33:25-50% free   34:25-50% free   35:25-50% free
   36:25-50% free   37:25-50% free   38:25-50% free   39:25-50% free
   40:25-50% free   41:25-50% free   42:25-50% free   43:25-50% free
```

```
   44:25-50% free   45:25-50% free   46:25-50% free   47:25-50% free
   48:25-50% free   49:25-50% free   50:25-50% free   51:25-50% free
   52:25-50% free   53:25-50% free   54:25-50% free   55:25-50% free
   56:25-50% free   57:25-50% free   58:25-50% free   59:25-50% free
   60:25-50% free   61:25-50% free   62:25-50% free   63:25-50% free
  ------------------------------------------------------
End dump data blocks tsn: 4 file#: 4 minblk 265 maxblk 265

*************************************************************************
```

266 号块,最后一个一级位图块

```
Start dump data blocks tsn: 4 file#: 4 minblk 266 maxblk 266
buffer tsn: 4 rdba: 0x0100010a (4/266)
scn: 0x0000.c66bfc9f seq: 0x01 flg: 0x04 tail: 0xfc9f2001
frmt: 0x02 chkval: 0xf65b type: 0x20=FIRST LEVEL BITMAP BLOCK
Hex dump of block: st=0, typ_found=1
Dump of memory from 0x07C62200 to 0x07C64200
7C62200 0000A220 0100010A C66BFC9F 04010000  [ .........k.....]
7C62210 0000F65B 00000000 00000000 00000000  [[..............]
7C62220 00000000 00000000 00000000 00000000  [................]
        Repeat 1 times
7C62240 00000000 00000000 00000000 00000004  [................]
7C62250 FFFFFFFF 00000000 00000000 00000040  [............@...]
7C62260 00010001 00000000 00000023 00000000  [........#.......]
7C62270 0000001D 00000000 4725C48F 4725C48F  [..........%G..%G]
7C62280 00000000 00000000 00000000 00000000  [................]
7C62290 0100003A 0000000B 00000011 00000080  [:...............]
7C622A0 00000080 01000189 00000000 00000011  [................]
7C622B0 00000000 00000172 00000000 00000001  [....r...........]
7C622C0 0000D6A2 00000000 00000000 01000149  [............I...]
7C622D0 00000040 00000000 00000000 00000000  [@...............]
7C622E0 00000000 00000000 00000000 00000000  [................]
        Repeat 9 times
7C62380 00000000 00000000 00000000 53355335  [............5S5S]
7C62390 33355335 33353335 33553355 33553355  [5S535353U3U3U3U3]
7C623A0 33553355 33553355 33553355 00000000  [U3U3U3U3U3U3....]
7C623B0 00000000 00000000 00000000 00000000  [................]
        Repeat 483 times
7C641F0 00000000 00000000 00000000 FC9F2001  [............ ..]
Dump of First Level Bitmap Block
 --------------------------------
  nbits : 4 nranges: 1         parent dba:  0x0100003a   poffset: 11
  unformatted: 0       total: 64        first useful block: 0
  owning instance : 1
  instance ownership changed at 10/29/2007 19:31:27
  Last successful Search 10/29/2007 19:31:27
  Freeness Status:  nf1 0      nf2 35     nf3 0      nf4 29

  Extent Map Block Offset: 4294967295
  First free datablock : 0
  Bitmap block lock opcode 0
  Locker xid:     :  0x0000.000.00000000
  Inc #: 0 Objd: 54946
 HWM Flag: HWM Set
    Highwater::  0x01000189  ext#: 17     blk#: 128    ext size: 128
 #blocks in seg. hdr's freelists: 0
 #blocks below: 370
 mapblk  0x00000000  offset: 17
```

```
-------------------------------------------------------
DBA Ranges :
-------------------------------------------------------
 0x01000149  Length: 64     Offset: 0

 0:full   1:75-100% free   2:75-100% free   3:25-50% free
 4:25-50% free   5:75-100% free   6:75-100% free   7:25-50% free
 8:25-50% free   9:50-75% free   10:75-100% free   11:25-50% free
 12:25-50% free   13:75-100% free   14:25-50% free   15:0-25% free
 16:25-50% free   17:75-100% free   18:25-50% free   19:25-50% free
 20:25-50% free   21:75-100% free   22:25-50% free   23:25-50% free
 24:75-100% free   25:75-100% free   26:25-50% free   27:25-50% free
 28:75-100% free   29:75-100% free   30:25-50% free   31:25-50% free
 32:75-100% free   33:75-100% free   34:25-50% free   35:25-50% free
 36:75-100% free   37:75-100% free   38:25-50% free   39:25-50% free
 40:75-100% free   41:75-100% free   42:25-50% free   43:25-50% free
 44:75-100% free   45:75-100% free   46:25-50% free   47:25-50% free
 48:75-100% free   49:75-100% free   50:25-50% free   51:25-50% free
 52:75-100% free   53:75-100% free   54:25-50% free   55:25-50% free
 56:75-100% free   57:75-100% free   58:25-50% free   59:25-50% free
 60:75-100% free   61:75-100% free   62:25-50% free   63:25-50% free
-------------------------------------------------------
End dump data blocks tsn: 4 file#: 4 minblk 266 maxblk 266
```

当数据块被插入满数据以后,只有下降到 25%-50%可以使用的时候,才变位图块,当我们看到块全为 full 的时候,我们删除数据,每删 20 行 dump 一下一级位图块.我们就会看到各种标记的空闲块.

实验 64：临时表的使用

该实验的目的是使用临时表,阶段性的保存数据,每个会话是独立的.

临时表

```
drop table tmp1;
```

- **会话内保留行的临时表**

```
CREATE GLOBAL TEMPORARY TABLE tmp1
ON COMMIT PRESERVE ROWS
AS SELECT * FROM emp;
```

TMP1 是会话级的临时表，在一个会话期间内，表的数据都会存在，存在于排序段，每个会话是隔离的，换句话说每个会话只能见到自己的数据，即使提交了别的会话也看不到，因为临时表存在于排序段中，而排序段是会话所专有的。每个会话只能见到自己的数据。

当我们多个会话同时使用临时表的时候，我们会发现有多个排序段在活动。每个会话只是使用临时表在系统表空间中的定义，所以我们 DROP 的时候不会去回收站，而是直接从字典中删除。

```
select table_name,LOGGING,TEMPORARY,DURATION from user_tables;
     TMP2            NO  Y    SYS$TRANSACTION
     TMP1            NO  Y    SYS$SESSION
```

这句话验证表是否为临时表，以及临时表的生命周期。

```
SELECT TABLESPACE_NAME,CURRENT_USERS  FROM  V$SORT_SEGMENT;
```

这句话验证有多少个用户在使用排序段。

- **事物内保存行的临时表**

```
drop table tmp2;
CREATE GLOBAL TEMPORARY TABLE tmp2
AS SELECT * FROM emp;

SELECT * FROM TMP2;
INSERT INTO TMP2 SELECT * FROM EMP;
SELECT * FROM TMP2;
COMMIT;
```

SELECT * FROM TMP2;
表 TMP2 是事务级的，当事务结束的时候表的数据会自动的删除。

实验 65：压缩存储数据和在线回缩高水位

该实验的目的是使用压缩的方法存储表中的数据,使表的占用空间少,从而提高内存的使用率.

- 关于 null 值的存储

我们在设计表的时候要将可能为 null 值的列放在最后,因为数据库不存储最后连续的 null 值.
我们建立两张相同的表 t1,t2,插入数据.t2 表的前两列插入数据,后面的所有列都为 null; t1 表的第一列插入数据,最后面列插入数据,中间的所有列都为 null.当然我们插入的为一样的数据.
重复自身插入自身,达到一万行左右,分析表,查看平均行长.

```
SQL> select * from t1 where rownum=1;

EMPNO      ENAME      JOB              MGR HIREDATE            SAL  COMM DEPTNO
---------- ---------- ----------------- ---------- ----------- ---------- ----- -------
A                                                                          A
```

中间有 6 列为 null 值.

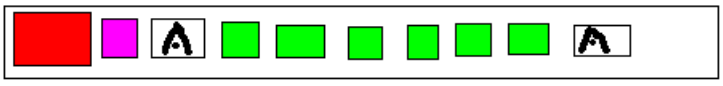
```
SQL> select * from t2 where rownum=1;

EMPNO      ENAME      JOB              MGR HIREDATE            SAL  COMM DEPTNO
---------- ---------- ----------------- ---------- ----------- ---------- ----- -------
A                  A
```

最后的连续 6 列为 null 值.

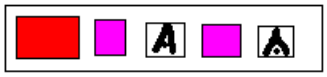
```
SQL> ANALYZE TABLE T1 COMPUTE STATISTICS;

Table analyzed.
SQL> ANALYZE TABLE T2 COMPUTE STATISTICS;

Table analyzed.
```

我们收集了两张表的统计信息.

```
SQL> select BLOCKS,NUM_ROWS,AVG_SPACE,AVG_ROW_LEN from tabs where table_name='T1';

    BLOCKS   NUM_ROWS  AVG_SPACE AVG_ROW_LEN
---------- ---------- ---------- -----------
        20       8192       1927          13

SQL> select BLOCKS,NUM_ROWS,AVG_SPACE,AVG_ROW_LEN from tabs where table_name='T2';

    BLOCKS   NUM_ROWS  AVG_SPACE AVG_ROW_LEN
---------- ---------- ---------- -----------
        13       8192       1138           7
```



t1表的存储,平均行长为13



t2表的存储,平均行长为7

```
SQL> select SEGMENT_NAME,HEADER_FILE,HEADER_BLOCK from dba_segments
```

```
  2  where owner='SCOTT' and SEGMENT_NAME in('T1','T2');

SEGMENT_NAME         HEADER_FILE HEADER_BLOCK
-------------------- ----------- ------------
T1                             4           59
T2                             4          115
```

我们将 60 数据块和 116 数据块转存到 dump 文件.
T1 表的转存文件的局部,两个 A 是分开存储的. 中间多了 6 个 null 的列头.

```
51B3EC0 FFFFFF41 01FFFFFF 08012C41 FFFF4101  [A.......A,...A..]
tab 0, row 0, @0x1a1c
tl: 13 fb: --H-FL-- lb: 0x1  cc: 8
col  0: [ 1]  41
col  1: *NULL*
col  2: *NULL*
col  3: *NULL*
col  4: *NULL*
col  5: *NULL*
col  6: *NULL*
col  7: [ 1]  41
```

T2 表的转存文件的局部,两个 A 是连续存储的.

```
7C63A10 01410102 02012C41 41014101 0102012C  [..A.A,...A.A,...]
tab 0, row 0, @0x1add
tl: 7 fb: --H-FL-- lb: 0x1  cc: 2
col  0: [ 1]  41
col  1: [ 1]  41
```

以上实验证明了数据库不存储最后的连续的 null 值,但中间的 null 值要留有列的标记位.虽然差别有限,但体现了你对数据库的存储的理解.

- **压缩表的数据(10g 的新特性)**

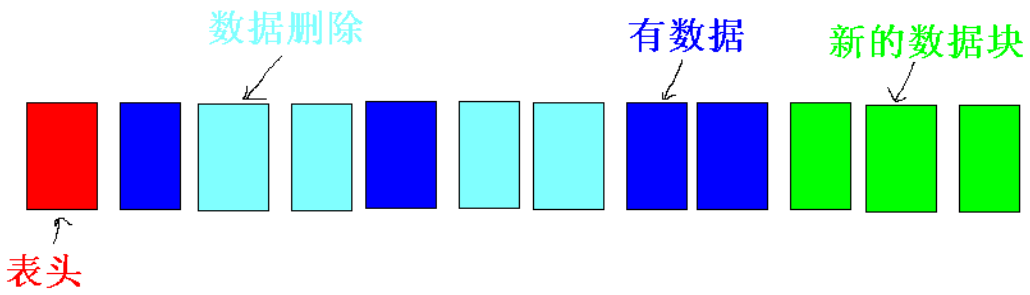
```
conn scott/tiger
drop table t1 purge;
create table t1 as select * from emp;
insert into t1 select * from t1;
--到 10000 行
drop table t2 purge;
create table t2 compress
as select * from t1 order by ename;

select segment_name,blocks from user_segments where segment_name in('T1','T2');
SEGMENT_NAME                   BLOCKS
-------------------            ----------
T1                             256
T2                             40
```

我们看到，压缩的很厉害，原来每个数据块存放 168 行，现在每个数据块存放 720 行，接近一个数据块可以存放行的极限，因为一个 8K 的数据块最多可以存放 733 行的数据。压缩存储的原理是每个块内相同的数据只存放一次，所以我们在压缩表的时候最好要排序。我们把大的静态表压缩存储，这样既可以节约存储空间又提高了 I/O 的效率，还节约了内存的使用，但是经常 update 的表我们不要压缩存储。会下降 DML 的性能。

- **移动表空间**

```
Alter table t1 move tablespace users;
```

释放多余的空间



释放浅蓝和绿色的数据块
当表运行一段时间后，表中的数据会被删除，导致了高水位下有空的数据块，或者不满的数据块，数据库在处理全表扫描的时候总是读高水位下所有的数据块。降低了全表扫描的效率，我们将表挪动表空间后，数据会紧密的码放，释放多余的空间，回收了高水位线。提高了全表扫描的性能，节约了存储空间。在 9I 前，挪动表以后会使索引无效，在 10G 版本中会自动重新建立索引。
移动表空间的动作比较大,我们在 10g 中提供了一个新特性—在线回缩表的高水位.

- **在线回缩表的高水位**

```
conn scott/tiger
alter table emp1 enable row movement;
-- 启动回缩特性
insert into emp1 select * from emp1;
/
/
/
commit;
-- 增加到 14000 行

delete emp1 where deptno=30;
commit;
-- 删除一半的数据
analyze table emp1 compute statistics;
-- 分析表的结构
select NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE from tabs where table_name='EMP1';
-- 查询高水位
SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) BLOCK#,COUNT(*) FROM EMP1
GROUP BY DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID);
-- 查询快内行的分布
SELECT EXTENT_ID,BLOCK_ID,BLOCKS FROM DBA_EXTENTS WHERE SEGMENT_NAME='EMP1';
-- 将数据挪动到表的前端，但不回缩高水位
alter table EMP1 shrink space compact;
-- 回缩高水位
 alter table EMP1 shrink space ;
```

实验 66：删除表中指定列操作

该实验的目的是修改表结构,删除多余的列
删除表的列

```
alter table t1 drop column sal checkpoint 1000;
```

Drop 过程中表的状态为 INVALID
如果过程中停电
启动数据库后

```
ALTER TABLE t1     DROP COLUMNS CONTINUE;
```

每次只能删除一列

Unused 列

```
conn scott/tiger
drop table t1 purge;
create table t1 as select* from emp ;
ALTER TABLE t1     set unused COLUMN hiredate;
ALTER TABLE t1     set unused COLUMN comm;
ALTER TABLE t1     set unused COLUMN mgr;
SELECT * FROM USER_UNUSED_COL_TABS;

查看被禁用的列的位置
SELECT COL#,NAME FROM SYS.COL$
WHERE OBJ#=(SELECT OBJECT_ID FROM DBA_OBJECTS WHERE OBJECT_NAME='T1');

删除所有被禁用的列
ALTER TABLE t1     DROP  unused COLUMNS CHECKPOINT 1000;
```

实验 67：使用 sqlldr 加载外部的数据

该实验的目的是使用 oracle 提供给我们的小工具.
Sqlldr 将平面文件到入到表
建立表 d1,建立文本文件 c:\bk\1.txt
其内容如下:是一个纯文本的文件,就是**数据源**.

```
10 ACCOUNTING                  NEW YORK
20 RESEARCH                    DALLAS
30 SALES                       CHICAGO
40 OPERATIONS                  BOSTON
```

控制流程的文件
‘c:\bk\c.txt‘
其内容如下:是一个纯文本的文件

```
LOAD DATA
INFILE ‘c:\bk\1.txt’
INTO TABLE D1
FIELDS TERMINATED BY ’,’ OPTIONALLY ENCLOSED BY ’″’
(DEPTNO, DNAME, LOC)
```

在操作系统下运行

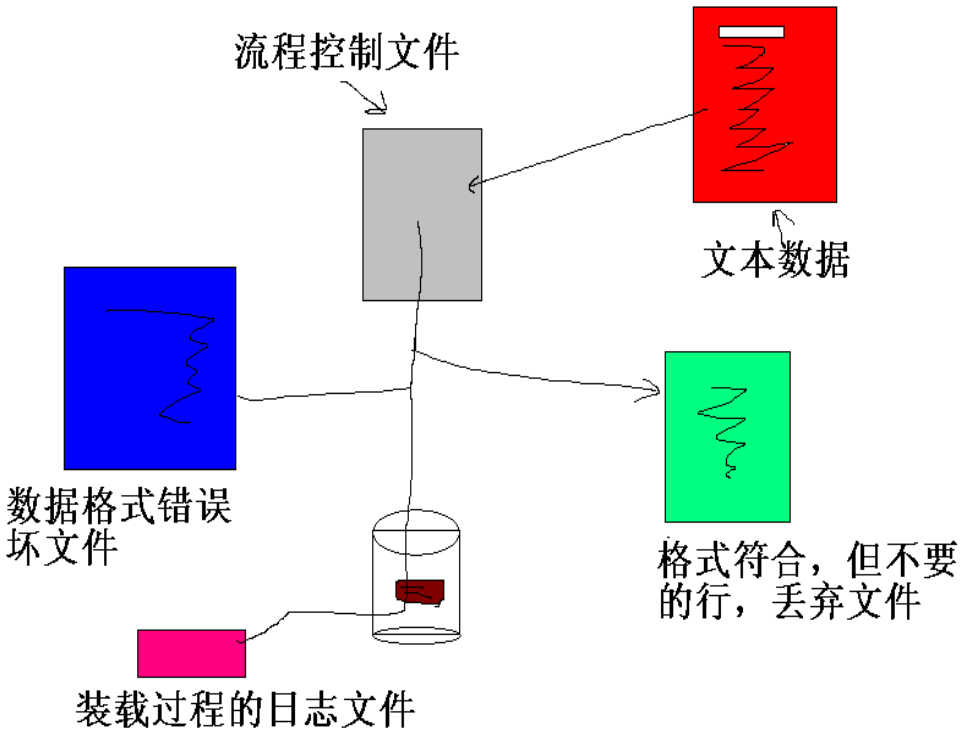
```
C:>sqlldr    scott/tiger    control=c:\bk\c.txt    discard=c:\bk\dis.txt    bad=c:\bk\bad.txt
log=c:\bk\log.txt
```

实验二:
下面我们将如下的纯文本加载的数据库中,分割符号为逗号.

```
7369,SMITH      ,CLERK             ,       7902,17-DEC-80   ,        810,      ,     20
7499,ALLEN      ,SALESMAN          ,       7698,20-FEB-81   ,       1610,   300,     30
7521,WARD       ,SALESMAN          ,       7698,22-FEB-81   ,       1260,   500,     30
7566,JONES      ,MANAGER           ,       7839,02-APR-81   ,       2985,      ,     20
7654,MARTIN     ,SALESMAN          ,       7698,28-SEP-81   ,       1260,  1400,     30
7698,BLAKE      ,MANAGER           ,       7839,01-MAY-81   ,       2860,      ,     30
7782,CLARK      ,MANAGER           ,       7839,09-JUN-81   ,       2460,      ,     10
7788,SCOTT      ,ANALYST           ,       7566,19-APR-87   ,       3010,      ,     20
7839,KING       ,PRESIDENT         ,           ,17-NOV-81   ,       5010,      ,     10
7844,TURNER     ,SALESMAN          ,       7698,08-SEP-81   ,       1510,     0,     30
7876,ADAMS      ,CLERK             ,       7788,23-MAY-87   ,       1110,      ,     20
7900,JAMES      ,CLERK             ,       7698,03-DEC-81   ,        960,      ,     30
7902,FORD       ,ANALYST           ,       7566,03-DEC-81   ,       3010,      ,     20
```

7934,MILLER ,CLERK , 7782,23-JAN-82 , 1310, , 10

控制文件如下:
LOAD DATA
INFILE ‘c:\bk\1.txt’
APPEND
into TABLE e1
when deptno=’30’
FIELDS TERMINATED BY ’,’ OPTIONALLY ENCLOSED BY ’"’
(EMPNO ,ENAME ,JOB ,MGR ,HIREDATE date "dd-mon-rr",SAL,COMM ,DEPTNO);



实验 68：使用 utl_file 包来将表的数据存储到外部文件

该实验的目的是将表的数据按照我们的格式写入到操作系统的文件,利用了数据库提供的包.
将表中的数据保存在文本文件
在初始化参数文件中加入下面一行
utl_file_dir=c:\bk
重新启动数据库
Show parameter utl 来验证该参数已经修改为 c:\bk

案例 1:数据写入到文本中
declare
v_filehandle UTL_FILE.FILE_TYPE;
begin
v_filehandle:=utl_file.fopen(‘c:\bk’,’output.txt’,’w’);
UTL_FILE.PUTF (v_filehandle,’SALARY REPORT: GENERATED ON%s\n’, SYSDATE);
UTL_FILE.NEW_LINE (v_filehandle);
UTL_FILE.PUTF (v_filehandle, ’%s\n’,’hello ’);
UTL_FILE.PUTF (v_filehandle, ’DEPARTMENT: %s\n’,’world ’);
UTL_FILE.PUTF(v_filehandle, ’aaaa%sbbb%sccc%sddd%seee%s’,’1’,’2’,’3’,’4’,’5’);
UTL_FILE.FCLOSE (v_filehandle);

```
end;
/
```

其中/n 为换行
%s 为替代字符，将来会被后面的 1 到 5 个参数替代，默认值为 NULL
NEW_LINE 过程建立一个新的空行

案例 2:将 DEPT 表的数据导入到文本中

```
declare
v_filehandle UTL_FILE.FILE_TYPE;
begin
v_filehandle:=utl_file.fopen('c:\bk','output.txt','w');
UTL_FILE.PUTF (v_filehandle,'表 DEPT 的文本数据，导出时间为：%s\n', SYSDATE);
UTL_FILE.NEW_LINE (v_filehandle);
for i in(select * from dept) loop
UTL_FILE.PUTF (v_filehandle, '%s ,%s, %s\n',i.deptno,i.dname,i.loc);
end loop;
UTL_FILE.FCLOSE (v_filehandle);
end;
/
```

案例 3:将 EMP 表的数据导入到文本中

```
declare
v_filehandle UTL_FILE.FILE_TYPE;
begin
v_filehandle:=utl_file.fopen('c:\bk','output.txt','w');
UTL_FILE.PUTF (v_filehandle,'表 EMP 的文本数据，导出时间为：%s\n', SYSDATE);
UTL_FILE.NEW_LINE (v_filehandle);
for i in(select * from EMP) loop
UTL_FILE.PUTF (v_filehandle, '%s| %s |%s| %s| %s
|',i.EMPNO,i.ENAME,i.JOB,NVL(i.MGR,-1),i.HIREDATE);
UTL_FILE.PUTF (v_filehandle, '%s| %s |%s\n',i.SAL,NVL(i.COMM,-1),i.DEPTNO);
end loop;
UTL_FILE.FCLOSE (v_filehandle);
end;
/
```

实验 69：使用外部表

该实验的目的是直接读取操作系统的文件,当做表来查询.

外部表的建立（sqlldr 的应用）

```
CONN SYSTEM/MANAGER
GRANT CREATE ANY DIRECTORY TO SCOTT;
CONN SCOTT/TIGER

CREATE DIRECTORY DBK  AS 'D:\BK';

CREATE TABLE oldDEPT (
  DEPTno NUMBER, Dname CHAR(20), LOC CHAR(20))
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_LOADER
  DEFAULT DIRECTORY DBK
  ACCESS PARAMETERS
  (RECORDS DELIMITED BY NEWLINE
  BADFILE 'bad_emp'
  LOGFILE 'log_emp'
  FIELDS TERMINATED BY ' '
```

```
  (DEPTno CHAR,
  Dname CHAR,
  LOC CHAR))
  LOCATION ('DEPT.txt'))
  PARALLEL 5
  REJECT LIMIT 200;
```

D:\bk\dept.txt 的内容为

```
10 ACCOUNTING NEWYORK
20 RESEARCH DALLAS
30 SALES CHICAGO
40 OPERATIONS BOSTON
```

Select * from oldDEPT;直接获取文本文件的内容

实验 70：处理挂起的事务

该实验的目的是暂停失败的事务,不要回退,处理故障后继续的运行原语句.

事务挂期后的处理（9i 的新特性）

事务当缺少某些资源不能运行的时候数据库会有两种处理方法,一是自动的回退了,当我们运行大的事务时,回退是很大的工作量.二是数据库可以将事务挂起,等待新的资源到来,继续进行处理没有完成的事务.我们可以设置等待的时间,可以检测等待的资源.

```
conn SYSTEM/MANAGER

drop tablespace t2m including contents and datafiles;
CREATE TABLESPACE T2M DATAFILE ' D:\ORACLE\ORADATA\ORA10\T2M.DBF' SIZE 1M AUTOEXTEND OFF;
```

建立一个很小的非自动扩展的表空间.在其中建立一个表 e.

```
CONN SCOTT/TIGER
drop table e purge;
CREATE TABLE E TABLESPACE T2M AS SELECT * FROM EMP;
INSERT INTO E SELECT * FROM E;
```

将表 e 不停的翻倍.直到报错,没有空间了.当前的事务会自动的回退最后一句话.

启动挂起的特性,等待 1 小时.

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600;

SELECT DBMS_RESUMABLE.GET_SESSION_TIMEOUT(159) FROM DUAL;
INSERT INTO E SELECT * FROM E;
```

将表 e 不停的翻倍.直到挂起不动了.

```
CONN SYSTEM/MANAGER
```

检测被挂起的事务.

```
SELECT * FROM DBA_RESUMABLE;
```

分配给表空间更多的空间,让事务进行.

```
ALTER DATABASE DATAFILE 6 RESIZE 3M;
```

下面就是上面的实验:

```
SQL> conn system/manager
Connected.
SQL> select name from v$datafile where rownum=1;

NAME
----------------------------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF
```

这句话的目的是定位路径.

```
SQL> drop tablespace t2m including contents and datafiles;
drop tablespace t2m including contents and datafiles
```

```
*
ERROR at line 1:
ORA-00959: tablespace 'T2M' does not exist


SQL> CREATE TABLESPACE T2M DATAFILE 'D:\ORACLE\ORADATA\ORA10\T2M.DBF'
  2  SIZE 1M AUTOEXTEND OFF;

Tablespace created.

SQL> CONN SCOTT/TIGER
Connected.
SQL> drop table e purge;
drop table e purge
           *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> CREATE TABLE E TABLESPACE T2M AS SELECT * FROM EMP;

Table created.

SQL> INSERT INTO E SELECT * FROM E;

14 rows created.

SQL> /

28 rows created.

SQL> /

56 rows created.

SQL> /

112 rows created.

SQL> /

224 rows created.

SQL> /

448 rows created.

SQL> /

896 rows created.

SQL> /

1792 rows created.

SQL> /

3584 rows created.
```

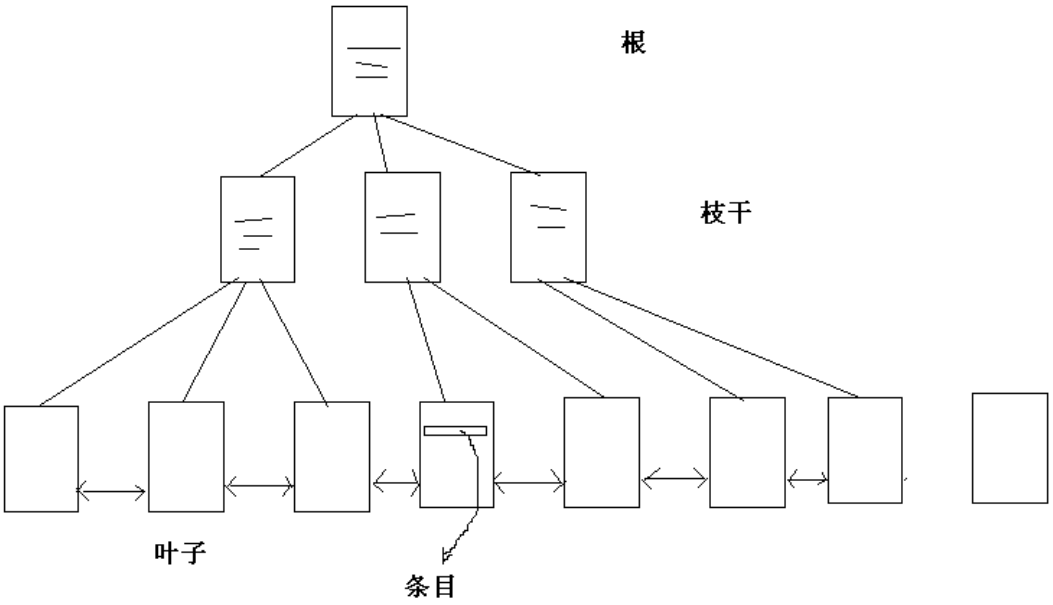
```
SQL> /

7168 rows created.

SQL> /
INSERT INTO E SELECT * FROM E
*
ERROR at line 1:
ORA-01653: unable to extend table SCOTT.E by 8 in tablespace T2M
只回退当前的语句,其他插入还在,等待我们结束事务.

SQL> select xidusn from v$transaction;

    XIDUSN
----------
         8

SQL> ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600;
起用挂起的特性
Session altered.

SQL> select sid from v$mystat where rownum=1;
查看当前的会话
       SID
----------
       159

SQL> SELECT DBMS_RESUMABLE.GET_SESSION_TIMEOUT(159) FROM DUAL;

DBMS_RESUMABLE.GET_SESSION_TIM
------------------------------
                          3600

SQL> INSERT INTO E SELECT * FROM E;
现在挂起不动了.等待资源.我们新开一个会话.
SQL> conn system/manager
Connected.
SQL> select event from V$session_wait where sid=159;
看 159 会话在等待什么.
EVENT
------------------------------
statement suspended, wait err
or to be cleared


SQL> SELECT * FROM DBA_RESUMABLE;

   USER_ID SESSION_ID INSTANCE_ID COORD_INSTANCE_ID COORD_SESSION_ID STATUS        TIMEOUT
---------- ---------- ----------- ----------------- ----------------- ---------- ----------
START_TIME                          SUSPEND_TIME
----------------------------------- -----------------------------------
RESUME_TIME                         NAME
----------------------------------- ---------------------------------------
SQL_TEXT
------------------------------------------------------------------------------------------------------------
--------
ERROR_NUMBER
------------
ERROR_PARAMETER1
```

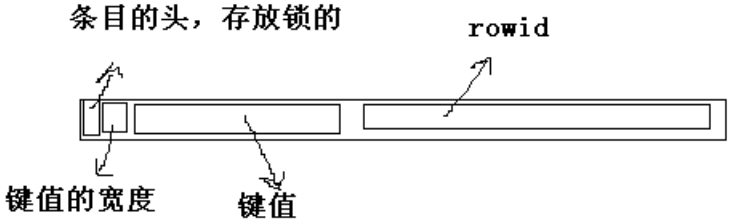
```
------------------------------------------------------------------------------------------------
--------
ERROR_PARAMETER2
------------------------------------------------------------------------------------------------
--------
ERROR_PARAMETER3
------------------------------------------------------------------------------------------------
--------
ERROR_PARAMETER4
------------------------------------------------------------------------------------------------
--------
ERROR_PARAMETER5
------------------------------------------------------------------------------------------------
--------
ERROR_MSG
------------------------------------------------------------------------------------------------
--------
        71        159           1                                    SUSPENDED         3600
09/21/07 09:19:58                       09/21/07 09:22:47
                                        User SCOTT(71), Session 159, Instance 1
INSERT INTO E SELECT * FROM E
        1653
SCOTT
E    /*操作的表*/
8
T2M

ORA-01653: unable to extend table SCOTT.E by 8 in tablespace T2M
/*为什么挂起*/

SQL> select file_id from dba_data_files where tablespace_name='T2M';

   FILE_ID
----------
         6

SQL> ALTER DATABASE DATAFILE 6 RESIZE 3M;

Database altered.
再回到第一个会话.我们看到
14336 rows created.

SQL> commit;

Commit complete.
```

索引

索引概论
表的数据是无序的，所以叫堆表（**heap table**）,意思为随机存储数据。因为数据是随机存储的，所以在查询的时候需要**全表扫描**。索引就是将无序的数据**有序**化，这样就可以在查询数据的时候减少数据块的读取，实现快速定位数据。对大表的排序是非常消耗资源的，索引是事先排好序，这样就可以在需要排序的时候使用索引就可以避免排序。索引对数据库的影响是巨大的，但索引不是万能的，数据库对索引的使用是有选择的，我们可以强制使用索引，也可以强制不使用索引。
一般的情况下数据库会自动的判断是否使用索引，除非你明确的在 SQL 语句中指定。
所有索引的原形都是**树状结构**，由根、枝干和叶子组成。根和枝干中存放键值范围的导引指针，叶子中存

放的是条目，条目中存放的是索引的键值和该数据行的ROWID。索引的叶子间通过指针横向的联系在一起，前一个叶子指向下一片叶子，这样的目的是数据库在找到一个叶子后就可以查找相临近的叶子，而不必再次去查找根和枝干的数据块。





索引和表一样是段级单位，和表和回退段是平级单位。
查看索引的属性。

```
SQL> select INDEX_NAME,INDEX_TYPE,TABLE_NAME,UNIQUENESS from user_indexes;
INDEX_NAME                     INDEX_TYPE           TABLE_NAME                     UNIQUENES
------------------------------ -------------------- ------------------------------ ---------
PK_EMP1                        NORMAL               MV1                            UNIQUE
PK_DEPT                        NORMAL               DEPT                           UNIQUE
PK_EMP                         NORMAL               EMP                            UNIQUE
```

查看索引在哪个表和哪个列上。

```
SQL> select INDEX_NAME,TABLE_NAME,COLUMN_NAME from user_ind_columns order by 2,3;

INDEX_NAME                     TABLE_NAME      COLUMN_NAME
------------------------------ --------------- --------------------
PK_DEPT                        DEPT            DEPTNO
PK_EMP                         EMP             EMPNO
PK_EMP1                        MV1             EMPNO
PK_EMP2                        MV_EMP          EMPNO
```

索引的建立
索引的建立有两种模式：隐式建立和显式建立
隐式建立，当我们在表上建立主键和唯一性约束的时候，数据库会自动的建立同名称的索引，如果你删除或者禁用约束，数据库会自动的删除该同名称的索引，因为这两个约束是通过索引来实现的。
显式建立，我们在需要的列上建立索引以提高数据库的性能，一个表上可以建立很多个索引。因为列的排序组合会有很多种模式，每个列上可以有很多个函数索引，所以理论上一张表的索引数目是无限的，但请记住我们只建立查询时经常使用的索引，因为索引要占空间，而且需要数据库维护。

索引的维护
数据库会自动的维护索引，我们手工可以重新建立和合并指定的索引
当索引已经建立，我们 update 索引的键值的时候，数据库会删除老的条目，添加新的条目，因为索引是有

序的，条目不能直接在原地修改到新的键值，新的键值必须去它应该存在的叶子。

实验 71：查看索引的内部信息

该实验的目的是理解索引的存储结构.

```
SQL> conn scott/tiger
Connected.
SQL> drop table t1 purge;
Table dropped.
SQL> create table t1 as select * from emp;
Table created.
SQL> create index i_t1_empno on t1(empno);
Index created.
SQL> analyze index i_t1_empno validate structure;
Index analyzed.
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
      高度  索引总块数    枝干块数    叶子块数  叶子内行数    叶子中被删除的行数
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS       DEL_LF_ROWS
---------- ---------- ---------- ---------- ----------       -----------
         1          8          0          1           14                 0
```

现在更新 EMPNO，再次查看，查看前一定要先分析，不然数据不会被重新收集。

```
SQL> update t1 set empno=7777 where empno=7900;
1 row updated.
SQL> commit;
Commit complete.
SQL> analyze index i_t1_empno validate structure;
Index analyzed.
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- -----------
    1          8          0          1           15           1
```

表中虽然只有 14 行，但索引的叶子中却有 15 行，其中的一行是 7900 被删除时留下的。

再次更新表

```
SQL> update t1 set empno=8888 where empno=7902;
1 row updated.
SQL> analyze index i_t1_empno validate structure;
Index analyzed.
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- -----------
      1          8          0          1           16           2
```

索引的叶子中有 16 行，新增加的一行是 7902 被删除时留下的。

索引会重新使用被删除的条目所留下的空洞，但这是在不改变索引的本质，索引是有序的情况下重用空洞。

我们现在向 T1 表中插入数据。直到 T1 的行达到 559 行。

```
SQL> select count(*) from t1;
  COUNT(*)
----------
       559
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- -----------
         1          8          0          1        559           0
```

我们看到 LF_BLKS 为 1，说明现在索引的所有条目都存在于一个叶子中。

再插入一行，看看发生了什么。

```
SQL> insert into t1 select * from t1 where rownum=1;
1 row created.
SQL> analyze index i_t1_empno validate structure;
```

```
Index analyzed.
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
  HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- ----------
         2          8          1          2        560          0
```

我的数据库中的块为 8K，当达到 560 行的时候，一个叶子已经容纳不下所有的条目。索引会长一层，HEIGHT 达到 2，叶子数 LF_BLKS 为 2，现在我们的索引为一个根，两片叶子，根中存放了叶子的指针。

如果我们继续插入，当条目达到十几万后，索引又会长一层，达到三层索引结构，向我们图中画的索引结构一样。索引由根，枝干和叶子组成。

以上索引的变化是数据库自己来维护的，对我们来说是透明的，你不用操心，我们通过这个实验来明白所以是如何变化的。

我们看看手工如何来维护索引。

合并索引（coalesce）。

我们向 T1 表插入到 1120 行。

```
SQL>  insert into t1 select * from t1;
560 rows created.
SQL> analyze index i_t1_empno validate structure;
Index analyzed.
SQL>  select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- ----------
         2          8          1          3       1120          0
```

再删除 600 行

```
SQL> delete t1 where rownum<=600;
600 rows deleted.
SQL> analyze index i_t1_empno validate structure;
Index analyzed.
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- ----------
         2          8          1          3       1120        600
```

我们发现有 600 个空洞。

```
SQL> alter index i_t1_empno coalesce;
Index altered.
SQL> analyze index i_t1_empno validate structure;
Index analyzed.
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- ----------
         2          8          1          2        520          0
```

经过合并以后，空洞消失了，叶子的块数降低了，说明叶子中的条目整合了。更加的紧密的码放在一起。

索引的重建（rebuild）

```
SQL> alter index i_t1_empno rebuild;
Index altered.
SQL> analyze index i_t1_empno validate structure;
Index analyzed.
SQL> select HEIGHT,BLOCKS,BR_BLKS,LF_BLKS,LF_ROWS,DEL_LF_ROWS from index_stats;
    HEIGHT     BLOCKS    BR_BLKS    LF_BLKS    LF_ROWS DEL_LF_ROWS
---------- ---------- ---------- ---------- ---------- ----------
         1          8          0          1        520          0
```

索引的重建就是将老的抛弃，跟新的一样，这样就使索引的结构发生了变化，由原来的 2 层结构降为一层结构。那合并和重建有什么区别呢？

合并不释放段所拥有的空间，不处理正在变化的行，也就是说有事务的时候也可以合并。合并只是合并枝干内的叶子，如果叶子属于不同的枝干则分别独立合并，记住合并不会改变索引的结构。不会改变改变索引的表空间和索引类型。

重建只能在没有事务的情况下进行，如果有未提交的事务，就会报错。

```
SQL> alter index i_t1_empno rebuild;
alter index i_t1_empno rebuild
*
```

```
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified
```
我们加上 online 选项。
```
SQL> alter index i_t1_empno rebuild online;
```
我们发现会话挂起了，说明在等待资源。我们在另一个会话里查询等待会话的等待事件。
```
SQL> select event from v$session_wait where sid=159;
EVENT
----------------------------
enq: TM - contention

SQL> select SID,REQUEST from v$lock where request<>0;
       SID    REQUEST
---------- ----------
       159          4
```
因为 159 会话没有获得足够的锁资源而等待。
因为索引重建要求所有要索引的行都要就位，所以必须等待事务结束。索引重建会改变索引的结构，释放多余的空间，可以改变索引的表空间和索引类型。
我们在工作中选择合并还是重建就看你的需求了，合并要求的资源少。重建要求的资源多。

实验 72：监控索引的使用状态

该实验的目的是找到没有被使用的索引.
索引的监视
我们建立索引的目的就是要使用索引来提高效率，所以如果我们建立的索引没有被使用，我们就应该删除从来没有被使用过的索引。那些索引被使用，我们可以通过对索引的监视来发现。
```
SQL> alter index pk_emp monitoring usage;
Index altered.
SQL> select * from v$object_usage;
索引名称       表名称          被监视否    使用过否   开始监视时间              结束监视时间
INDEX_NAME TABLE_NAME        MONITO      USED      START_MONITORING       END_MONITORING
---------- --------------- ------ ------ ------------------------ -------------------
PK_EMP       EMP             YES         NO        10/13/2006 23:20:04
```
如果你想监视当前用户下的所有索引。你得事先产生一个脚本。
```
SQL> select 'alter index '||index_name||' monitoring usage;' from user_indexes;
'ALTERINDEX'||INDEX_NAME||'MONITORINGUSAGE;'
--------------------------------------------------------------------------------
alter index PK_EMP monitoring usage;
alter index I_T1_EMPNO monitoring usage;
alter index PK_DEPT monitoring usage;
```
我们运行查询得到得结果，就会监视当前用户下的所有索引。
在程序运行一段时间后，查看"USED"列，就可以找到那些从来未被使用得索引。我们要将未使用得索引删除，当然主键和唯一键得索引你得仔细考虑是否可以删除。
我们在监视得过程中不能重新启动数据库，因为 v$的视图会被重新建立，丢失原来的监视。
```
SQL> select 'alter index '||index_name||' nomonitoring usage;' from user_indexes;
'ALTERINDEX'||INDEX_NAME||'NOMONITORINGUSAGE;'
--------------------------------------------------------------------------------
alter index PK_EMP nomonitoring usage;
alter index I_T1_EMPNO nomonitoring usage;
alter index PK_DEPT nomonitoring usage;
```
加个 NO 就会取消监视。

索引的类别
b-tree 索引,最常见的索引类型，一切索引的原形。

下面的不同索引类型在 SQL 优化中介绍
位图索引
函数索引

联合索引
分区本地索引

约束的管理

延迟约束
```
select table_name,CONSTRAINT_NAME,DEFERRABLE,DEFERRED from user_constraints;
```
运行每句话都判定约束为立即约束
在事务结束的时候统一判定叫延迟约束
延迟约束可以容纳一段时间的非法数据
建立约束默认为立即约束

实验 73：改变约束的状态

该实验的目的是理解索引和约束的关系,理解约束的不同状态
建立一个初始状态为延迟的可延迟约束
```
drop table e purge;
create table e as select * from emp;
alter table e add constraint u_empno UNIQUE (EMPNO) INITIALLY DEFERRED DEFERRABLE;
UPDATE E  SET EMPNO=7900 WHERE EMPNO=7902;
```
这时 E 表内有相同 EMPNO 的员工
如果提交，事务自动回退

如果一个约束是可以延迟的
当前会话的属性决定约束是否延迟
```
select table_name,CONSTRAINT_NAME,DEFERRABLE,DEFERRED from user_constraints;
ALTER SESSION SET CONSTRAINT=IMMEDIATE;
UPDATE E  SET EMPNO=7900 WHERE EMPNO=7902;
ALTER SESSION SET CONSTRAINT=DEFERRED;
--可以延迟的约束就会延迟
alter session set constraints =default;
--约束初始状态是什么就是什么
```

如果主键或者唯一约束可以延迟
因为这两个约束都要索引来维护唯一性
可以延迟就要容纳部分错误的数据
所以应该为非唯一索引

约束的状态
默认的约束状态为启用有效
```
select table_name,CONSTRAINT_NAME,status, VALIDATED  from user_constraints;
```
我们可以修改约束的状态
```
alter table emp disable novalidate constraint FK_DEPTNO;
alter table emp disable validate constraint FK_DEPTNO;
alter table emp enable novalidate constraint FK_DEPTNO;
alter table emp enable validate constraint FK_DEPTNO;
```
兰色的可以不写

禁用约束
```
alter table emp disable novalidate constraint FK_DEPTNO;
```
如果约束有索引，自动删除
约束存在定义中，不起作用
在批量加载数据的时候，先禁用约束
提高加载的效率

启用约束
如果需要，自动建立索引
数据必须满足约束的条件

实验 74：找到违反约束条件的行

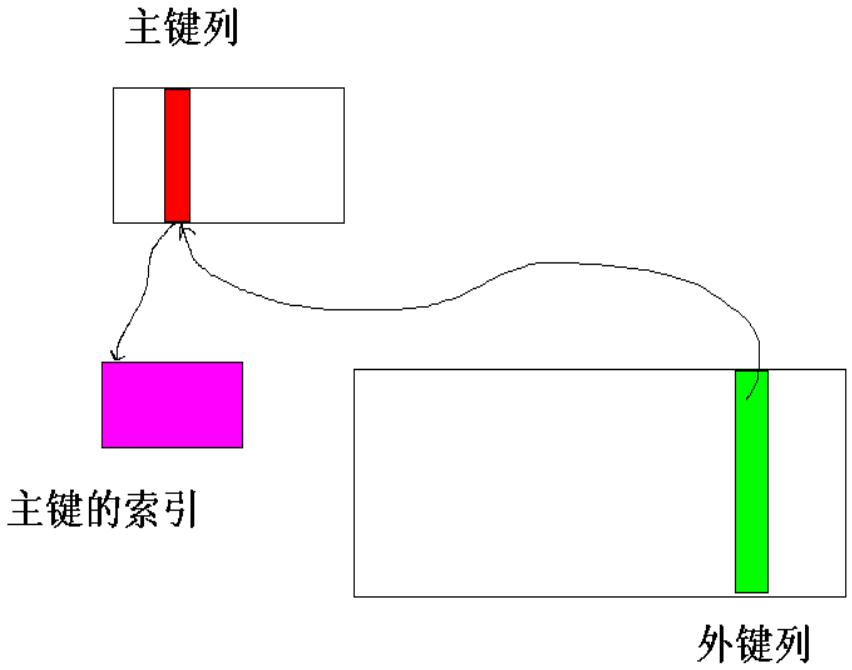
该实验的目的是启用约束的时候失败,找到引起约束失败的行
处理违反约束条件的行
1。建立一个 DISABLE 的约束
2。建立 EXPTIONS 表
3。启动约束
4。查看 exptions 表
5。处理违反约束条件的行
6。再次启用约束

```
Conn scott/tiger
@%ORACLE_HOME%\rdbms\admin\utlexpt1.sql
drop table t1 purge;
create table t1 as select * from emp;
alter table t1 add constraint t1_pk primary key (empno) disable ;
--添加重复的行
update t1 set empno=7900 where empno=7902;
commit;
--将违反约束的行放入到指定表中
alter table t1 enable constraint t1_pk exceptions into exceptions;
```

启用一个无效的约束

```
drop table d purge;
create table d tablespace users as select * from dept;
alter table d add constraint pk_d primary key  (deptNO) ;
select index_name,UNIQUENESS     from user_indexes;
alter table d disable constraint pk_d ;
select index_name,UNIQUENESS     from user_indexes;
update d set deptno=20 where deptno=10;
commit;--使表内有重复的行
create index i5 on d(deptno);--建立非唯一索引
alter table d enable novalidate constraint pk_d using index i5;
insert into d deptno values(30,null,null); --失败
```

外键和索引的关系
当修改绿的值时，粉的必须可以访问，与红色无关



Profile 配置

实验 75：管理密码的安全配置

该实验的目的是使密码更加安全,练习 profile 的配置
建立用户的考虑项目
名称要规范，符合命名规则
密码是否过期
默认临时表空间
默认永久表空间
权限和角色
空间使用配额

建立帐号 U1
```
Create user u1 identified by u1;
```
给 U1 授权
```
Grant create session ,create table to u1;
```
连接到 U1
```
Conn u1/u1
```
查看 U1 的帐号属性
```
Select * from user_users;
```

```
Conn system/manager
```
限定配额
```
Alter user u1 quota 1m on users;
Create table u1.t1 as select * from scott.emp;
```
查看配额的使用情况
```
Select * from dba_ts_quotas;
```

配额为零，不再分配新的空间，现有的可以继续使用

```
Alter user u1 quota 0 on users;
```

建立配置文件 P1

```
create profile p1 limit FAILED_LOGIN_ATTEMPTS 2;
```

验证 dba_profiles

```
select * from dba_profiles order by 1,3;
```

赋予用户

```
alter user u1 profile p1;
```

验证 dba_users

```
select  USERNAME,PROFILE    from  dba_users;
```

检验 p1 效果
故意以错误密码登陆 U1，连续三回
U1 帐号被锁

```
select USERNAME,ACCOUNT_STATUS from dba_users where username='U1';
```

高级帐号解锁

```
conn system/manager
alter user u1 account unlock;
```

修改 p1 的其它限制
PASSWORD_VERIFY_FUNCTION
PASSWORD_REUSE_MAX
FAILED_LOGIN_ATTEMPTS
PASSWORD_LOCK_TIME
PASSWORD_LIFE_TIME 30
PASSWORD_GRACE_TIME 3
PASSWORD_REUSE_TIME
以上都是限制密码的，总起作用
带有 TIME 单位为天

实验 76：限制会话的资源配置

该实验的目的是控制每个会话的资源的使用,如 cpu,连接时间,连接的会话个数等.
限制资源
LOGICAL_READS_PER_CALL
LOGICAL_READS_PER_SESSION
CPU_PER_CALL
CPU_PER_SESSION
PRIVATE_SGA
COMPOSITE_LIMIT
IDLE_TIME
CONNECT_TIME
SESSIONS_PER_USER

限制资源起作用的条件

```
show parameter limit
alter system set resource_limit=true;
alter profile p1 limit SESSIONS_PER_USER 1;
```

删除配置文件 p1

```
CONN SYSTEM/MANAGER
DROP PROFILE P1 CASCADE;
```

Defalut 配置文件自动赋予 u1 帐号

删除用户

```
Drop user u1 cascade;
```

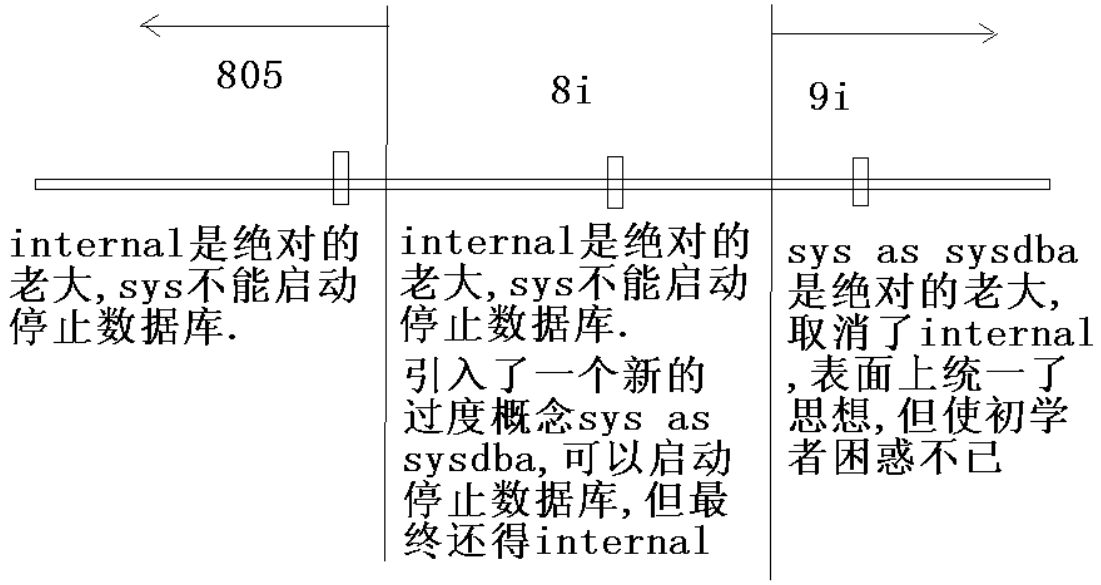
权限管理

实验 77：维护系统权限

该实验的目的是理解数据库的系统权限
你能做什么
Grant 授权
Revoke 回收

查看当前用户拥有的系统权限
Select * from session_privs;
查看所有用户和角色的系统权限授予情况
select grantee,COUNT(*) FROM dba_sys_privs
GROUP BY grantee ORDER BY 2;

show parameter o7
O7_DICTIONARY_ACCESSIBILITY
Oracle7 版本的 any 是含有 sys 的对象的
Oracle7 以后的版本的 any 不含有 sys 的对象的



该参数为 true，表示 any 的范围包括 sys 的对象,可以使用 conn sys/pass 登录而不加 as sysdba 选项,进入的是数据库的 sys,是数据库的拥有者.而不是操作系统的 sys,conn / as sysdba 等于原来的 internal.
该参数为 false，表示 any 的范围剔除 sys 的对象

Select any table
Drop any table 等系统权限

建立实验帐号
conn system/manager
drop user u1 cascade;
drop user u2 cascade;

create user u1 identified by u1;
create user u2 identified by u2;

```
grant create session to u1,u2;

grant create session to u1,u2;
--连带管理的权限
grant select any table to u1 with admin option;
--权限转授
conn u1/u1
grant select any table to u2;
conn system/manager
select * from dba_sys_privs where grantee in('U1','U2');
--转授人权限被回收
REVOKE SELECT ANY TABLE FROM U1;
--验证 U2 的权限，没有被级连回收
select * from dba_sys_privs where grantee in('U1','U2');
```

实验 78：维护对象权限

该实验的目的是理解数据库的对象权限

对象权限

你在某个指定的对象上有什么权限
你在 emp 表上是否有
Select
Update
Insert
Delete
Alter
index

表级的对象权限

```
conn scott/tiger
grant select on  emp to u1;
select * from user_tab_privs_made;
conn u1/u1
select * from user_tab_privs_recd;
```

列上的对象权限

```
conn scott/tiger
grant update(sal) on scott.emp to u1;
select * from user_col_privs_made;
conn u1/u1
select * from user_col_privs_recd;
```

对象权限的级连回收问题

```
conn scott/tiger
grant delete on scott.emp to u1 with grant option;
select * from user_tab_privs_made;
conn u1/u1
grant delete on scott.emp to u2;
select * from user_tab_privs_recd;
select * from user_tab_privs_made;
conn u2/u2
select * from user_tab_privs_recd;
conn scott/tiger
revoke delete on scott.emp from u1;
```

权限很有意思,有的时候必须直接赋予权限,而不能通过角色来获得.当你查看有权限,而还不能执行某些操作的时候,请考虑一下,是否是角色转来的权限.

实验 79：维护角色

该实验的目的是维护数据库的角色

角色

角色是权限的集合
角色可以嵌套
角色简化了管理
角色不会被级连回收
每个用户可以有多个角色

```
SQL> show parameter role

NAME                                 TYPE                              VALUE
------------------------------------ --------------------------------- -----
max_enabled_roles                    integer                           150
```

该参数决定了数据库内的最大角色数,9i 前默认是 30,所以你在导入数据库的时候要注意,先把它改大,不然可能有的角色建立失败而导致一系列的错误.
查看现有的角色

```
conn system/manager
--数据库内所有的角色
select * from dba_roles;
--角色的嵌套关系和所授予的用户
select * from dba_role_privs order by 1;
```

预先定义的角色

```
CONN SYS/SYS AS SYSDBA
SELECT * FROM ROLE_SYS_PRIVS WHERE ROLE IN('CONNECT','RESOURCE');
SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE IN('CONNECT','RESOURCE');
```

Connect 角色在 10g 以后只有 create session 的权限.原来以前的版本有好多其它的权限都被撤消了.
Resource 角色被授予后用户自动有 UNLIMITED TABLESPACE 的系统权限.

自己定义角色

```
drop role r1;
create role r1;
grant create table to r1;
grant select on scott.emp to r1;
select * from dba_sys_privs where grantee='R1';
SELECT * FROM ROLE_TAB_PRIVS WHERE ROLE='R1';
GRANT R1 TO U1;
CONN U1/U1
SELECT * FROM SESSION_ROLES;
```

用户默认角色

```
CONN SYSTEM/MANAGER
create role r2;
grant create ANY INDEX to r2;

GRANT R2 TO U1;
select * from dba_role_privs WHERE GRANTEE='U1';
--不直接指明，所有的角色都为默认角色
Conn u1/u1
Select * from session_roles;
```

指定某个角色为默认角色

```
CONN SYSTEM/MANAGER
ALTER USER U1 DEFAULT ROLE NONE;
ALTER USER U1 DEFAULT ROLE r1;
```

```
Conn u1/u1
Select * from session_roles;
```

角色的切换

```
Conn u1/u1
Set role all;
Set role r1;
Set role r2;
Select * from session_roles;
```

角色的密码验证
在角色切换的时候，需要指定密码

```
CONN SYSTEM/MANAGER
ALTER ROLE R2 IDENTIFIED BY R2;
ALTER USER U1 DEFAULT ROLE R1;
```

我们要把非默认的角色保护起来

```
Conn u1/u1
Set role r2 identified by r2;
```

建立角色的原则
按照应用程序建立一级角色
再按照使用程序的人建立二级的角色
将角色赋予用户
有些情况下通过角色来获得的权限是受限制的，请直接授与对象权限，而不是通过角色。

实验 80：审计

该实验的目的是监控可疑的数据库操作.

数据库的审计

```
conn sys/manager as sysdba
startup force
show parameter audit
ALTER SESSION SET NLS_DATE_FORMAT='YYYY/MM/DD:HH24:MI:SS';
select * from sys.aud$;
AUDIT DELETE ON scott.emp1 BY ACCESS WHENEVER SUCCESSFUL;
SELECT * FROM DBA_OBJ_AUDIT_OPTS WHERE OWNER = 'SCOTT' AND OBJECT_NAME LIKE 'EMP%';

conn scott/tiger
delete emp1;
commit;
conn system/manager
SELECT * FROM DBA_AUDIT_OBJECT;
select * from sys.aud$;


noAUDIT DELETE ON scott.emp1;
SELECT * FROM DBA_OBJ_AUDIT_OPTS WHERE OWNER = 'SCOTT' AND OBJECT_NAME LIKE 'EMP%';


AUDIT TABLE;
SELECT * FROM DBA_STMT_AUDIT_OPTS;
SELECT * FROM DBA_AUDIT_TRAIL;
NOAUDIT TABLE;

AUDIT SELECT ON SCOTT.EMP;
SELECT * FROM DBA_OBJ_AUDIT_OPTS WHERE OWNER = 'SCOTT' AND OBJECT_NAME LIKE 'EMP%';
SELECT * FROM DBA_AUDIT_OBJECT;
```

```
NOAUDIT SELECT ON SCOTT.EMP;

--操作系统审计 文档 E:\generic_102doc\network.102\b14266.dbf
AUDIT_TRAIL=XML
audit_file_dest=D:\ORACLE\ADMIN\ORA10\ADUMP
audit_sys_operations    =true
select * from V$XML_AUDIT_TRAIL;
```

数据库字符集

建立数据库时指定的
必须是 ascii 的完全超集
数据库的 char,varchar,long,clob 的编码
国家语言字符集
建立数据库时指定的
必须是统一编码 （ AL16UTF16 或 UTF8）
数据库的 nchar,nvarchar,nclob 的编码
两种字符集都不容易更改

查看支持的字符集

```
select * from v$nls_valid_values where PARAMETER='CHARACTERSET' order by 2;
ZHS16CGB231280
ZHS16CGB231280FIXED
ZHS16DBCS
ZHS16DBCSFIXED
ZHS16GBKFIXED
ZHS16MACCGB231280
```

ZHS32GB18030 和 ZHS16GBK 没有子集的关系,他们有共同的部分,也有不能的部分,所以我们在测试的时候多选一些汉字,多选怪字.才能发现两种字符集的不同.
区域+位数+iso 标准

字符集的选择
单字节的数据库字符集
AL16UTF16 国家语言字符集
在设计表的时候用 nchar,nvarchar2 的数据类型来存储汉语
查看当前数据库的字符集

```
Select * from nls_database_parameters;
PARAMETER                VALUE
------------------------ ------------------------------
NLS_NCHAR_CHARACTERSET   AL16UTF16     国家语言字符集
NLS_LANGUAGE             AMERICAN
NLS_TERRITORY            AMERICA
NLS_CURRENCY             $
NLS_ISO_CURRENCY         AMERICA
NLS_NUMERIC_CHARACTERS   .,
NLS_CHARACTERSET         ZHS16GBK      数据库字符集
NLS_CALENDAR             GREGORIAN
NLS_DATE_FORMAT          DD-MON-RR
NLS_DATE_LANGUAGE        AMERICAN
NLS_SORT                 BINARY
NLS_TIME_FORMAT          HH.MI.SSXFF AM
NLS_TIMESTAMP_FORMAT     DD-MON-RR HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT       HH.MI.SSXFF AM TZR
NLS_TIMESTAMP_TZ_FORMAT  DD-MON-RR HH.MI.SSXFF AM TZR
NLS_DUAL_CURRENCY        $
NLS_COMP                 BINARY
```

```
NLS_LENGTH_SEMANTICS      BYTE
NLS_NCHAR_CONV_EXCP       FALSE
NLS_RDBMS_VERSION         9.2.0.8.0

SQL> conn scott/tiger
Connected.
SQL> drop table t1;

Table dropped.

SQL> create table t1 (c1 varchar2(8),c2 nvarchar2(8));
两列分别是以数据库字符集和国家语言字符集存储的.
Table created.

SQL> insert into t1 values('a','a');
存储两个 a.
1 row created.

SQL> commit;

Commit complete.

SQL> select length(c1),length(c2),lengthb(c1),lengthb(c2) from t1;
我们看到两个 a 不是以一种字符集存储的,一个为 1 位,一个为 2 位.
LENGTH(C1) LENGTH(C2) LENGTHB(C1) LENGTHB(C2)
---------- ---------- ----------- -----------
         1          1           1           2
```

实验 81：配置国家语言支持

```
该实验的目的是理解数据库的不语言环境的影响.
客户端的语言环境设置
alter session set nls_language=american;
决定提示信息的语言类型
日期的语言
Select hiredate from emp;

alter session set nls_language='simplified chinese';
合法的语言名称
select * from v$nls_valid_values where PARAMETER='LANGUAGE' order by 2;

alter session set NLS_TERRITORY =america;
决定货币的格式
select to_char(sal,'l999999') from emp;

alter session set NLS_TERRITORY =china;

合法的地域名称
select * from v$nls_valid_values where PARAMETER='TERRITORY' order by 2;

合法的排序名称
select * from v$nls_valid_values where PARAMETER='SORT' order by 2;

alter session set NLS_sort =SCHINESE_PINYIN_M;
alter session set NLS_sort =SCHINESE_STROKE_M;
alter session set NLS_sort =SCHINESE_RADICAL_M;
```

决定排序和建立索引的顺序

```
conn scott/tiger
drop table t1;
create table t1(c varchar2(4));
insert into t1 values('啊');
insert into t1 values('一');
insert into t1 values('人');
insert into t1 values('木');
insert into t1 values('目');
insert into t1 values('藏');
insert into t1 values('三');
commit;

--当前会话的排序模式
select VALUE from nls_session_parameters where PARAMETER='NLS_SORT';
select * from t1 order by 1;

--修改排序模式
alter session set NLS_SORT='SCHINESE_PINYIN_M';
select * from t1 order by 1;

alter session set NLS_SORT='SCHINESE_STROKE_M';
select * from t1 order by 1;

alter session set NLS_SORT='GBK';
select * from t1 order by 1;

alter session set NLS_SORT='SCHINESE_RADICAL_M';
select * from t1 order by 1;

alter session set NLS_sort =BINARY;
select * from t1 order by 1;


alter session set NLS_date_format ='yyyy/mm/dd:hh24:mi:ss';
```

日期的显示格式

```
 select sysdate from dual;
```

会话级别如果没有设置语言环境
那么就以程序的环境变量为默认值
Nls_lang=语言_地域.字符集
可以设置在注册表中，也可以放在环境变量中

```
Set nls_lang=american_america.us7ascii
```

查看关于语言变量的字典

```
col value for a30
--数据库的信息
select * from Nls_database_parameters;
--实例的信息
select * from Nls_instance_parameters;
--当前会话的信息
select * from Nls_session_parameters;
```

修改数据库的字符集

```
alter database “orcl” character set ZHS16CGB231280;
```

修改国家语言字符集

```
alter database “orcl” national character set ZHS16CGB231280;
```

我配置国家语言支持的时候要考虑四种字符集的设置.主机的操作系统字符集,主机的数据库字符集,客户端的环境设置,客户端的操作系统字符集.百密一疏,从长计议.

元数据

实验 82：提取元数据 dbms_metedata

该实验的目的是使用 dbms_metadata 包来学习 oracle 的语法.
元数据的提取，exp 中有但不好用，expdp 中新加了这个特性，较好用。
元数据,TABLE,INDEX 一定要大写,目的是为了获取数据库建立对象的 DDL 命令集
我们通过取元数据的**目的**是备份产生对象的**脚本**和学习创建时的**语法和参数**使用选项.

```
Conn scott/tiger
Set long 10000
select dbms_metadata.get_ddl('TABLE','EMP') FROM DUAL;
select dbms_metadata.get_ddl('INDEX','PK_EMP') FROM DUAL;
```

```
SQL> Conn scott/tiger
Connected.
SQL> Set long 10000
```

取**表**的元数据

```
SQL> select dbms_metadata.get_ddl('TABLE','EMP') FROM DUAL;

DBMS_METADATA.GET_DDL('TABLE',
--------------------------------------------------------------------------------

  CREATE TABLE "SCOTT"."EMP"
   (    "EMPNO" NUMBER(4,0),
        "ENAME" VARCHAR2(10),
        "JOB" VARCHAR2(9),
        "MGR" NUMBER(4,0),
        "HIREDATE" DATE,
        "SAL" NUMBER(7,2),
        "COMM" NUMBER(7,2),
        "DEPTNO" NUMBER(2,0),
         CONSTRAINT "PK_EMP" PRIMARY KEY ("EMPNO")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "USERS"  ENABLE,
         CONSTRAINT "FK_DEPTNO" FOREIGN KEY ("DEPTNO")
          REFERENCES "SCOTT"."DEPT" ("DEPTNO") ENABLE
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "USERS"
```

取**索引**的元数据

```
SQL> select dbms_metadata.get_ddl('INDEX','PK_EMP') FROM DUAL;
```

```
DBMS_METADATA.GET_DDL('INDEX',
--------------------------------------------------------------------------------

  CREATE UNIQUE INDEX "SCOTT"."PK_EMP" ON "SCOTT"."EMP" ("EMPNO")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "USERS"
```

取**序列**的元数据

```
SQL> CONN SYSTEM/MANAGER
Connected.
SQL> select dbms_metadata.get_ddl('SEQUENCE','SYSTEM_GRANT','SYS') FROM DUAL;

DBMS_METADATA.GET_DDL('SEQUENC
--------------------------------------------------------------------------------

   CREATE SEQUENCE  "SYS"."SYSTEM_GRANT"  MINVALUE 1 MAXVALUE 9999999999999999999999999999
INCREMENT
2 CACHE 20 ORDER  NOCYCLE
```

取**表空间**的元数据

```
SQL> select dbms_metadata.get_ddl('TABLESPACE','UNDOTBS1') FROM DUAL;

DBMS_METADATA.GET_DDL('TABLESP
--------------------------------------------------------------------------------

  CREATE UNDO TABLESPACE "UNDOTBS1" DATAFILE
  'D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF' SIZE 26214400
  AUTOEXTEND ON NEXT 5242880 MAXSIZE 32767M
  BLOCKSIZE 8192
  EXTENT MANAGEMENT LOCAL AUTOALLOCATE
   ALTER DATABASE DATAFILE
  'D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF' RESIZE 62914560
```

取出的元数据都没有分号和斜杠,这样就不能批量运行,我们在取完元数据的语法后加上.
我写的语法中含有多个单引,因为单引是特殊字符,需要转义.我们也可以使用 chr(39)来替代.

```
SQL> select chr(39) from dual;

CHR(
----
'
```

```
-------------------------构建脚本--------------
set pagesize 1000
```

加上斜杠

```
select
'select  dbms_metadata.get_ddl('||''''||'TABLE'||''''||','||''''||table_name||''''||')  from
dual;'
||chr(10)
||'select '||''''||'/'||''''|| ' from dual;' from user_tables;
```

加上分号

```
SELECT                                                                          'SELECT
DBMS_METADATA.GET_DDL(''INDEX'','''||INDEX_NAME||''','''SCOTT''')'||'|'||'|'||''';'''||'  FROM
DUAL'||';' FROM DBA_INDEXES WHERE TABLESPACE_NAME='USERS';

SELECT                                                                          'SELECT
DBMS_METADATA.GET_DDL(''TABLE'','''||TABLE_NAME||''','''SCOTT''')'||'|'||'|'||''';'''||'  FROM
```

```
DUAL'||';' FROM DBA_INDEXES WHERE TABLESPACE_NAME='USERS';
```

将产生的语句保存成文件 1.txt

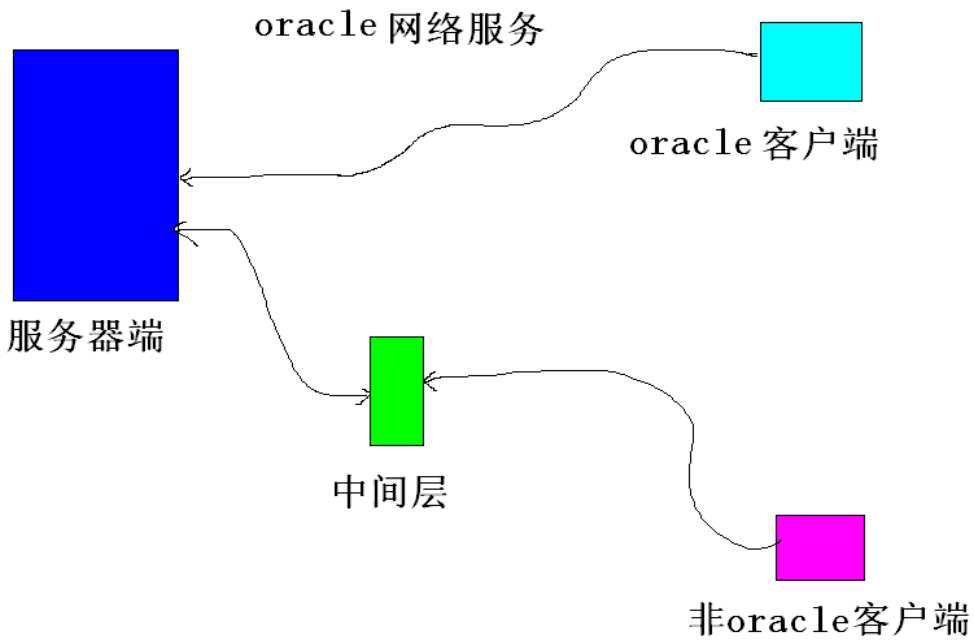
```
select dbms_metadata.get_ddl('TABLE','EMP') from dual;
select '/' from dual;

select dbms_metadata.get_ddl('TABLE','DEPT') from dual;
select '/' from dual;

select dbms_metadata.get_ddl('TABLE','BONUS') from dual;
select '/' from dual;

select dbms_metadata.get_ddl('TABLE','SALGRADE') from dual;
select '/' from dual;

SELECT DBMS_METADATA.GET_DDL('INDEX','IT4','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('INDEX','I_F_SAL','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('INDEX','PK_EMP1','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('INDEX','PK_EMP','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('INDEX','PK_DEPT','SCOTT')||';' FROM DUAL;

SELECT DBMS_METADATA.GET_DDL('TABLE','EMP','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('TABLE','EMP','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('TABLE','MV1','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('TABLE','EMP','SCOTT')||';' FROM DUAL;
SELECT DBMS_METADATA.GET_DDL('TABLE','DEPT','SCOTT')||';' FROM DUAL;
```

再设定环境

```
set heading off
SET ECHO OFF
```

运行上面的 1.txt,我们会产生一个脚本 2.txt,2.txt 就是我们建立对象的脚本,将来可以使用 2.txt 来建立新数据库中的表或者索引.

第四部分数据库的网络配置



网络的配置好象简单,其实网络还是很复杂的.
服务器端的配置

实验 83：配置监听

该实验的目的是配置服务器端的网络设置.
监听(listener)
配置文件 oracle_home\network\admin\listener.ora
要素
监听的名称（默认为 listener,最好不改）
监听主机的信息（主机名称或 ip,协议，端口号）
监听数据库的信息（数据库名称，oracle_home,实例名称）

只能监听本地的主机
不能监听远程的主机
一个监听可以监听多个数据库
一个数据库可以被多个监听监听

```
C:\>lsnrctl

LSNRCTL for 32-bit Windows: Version 9.2.0.8.0 - Production on 21-SEP-2007 21:57:47

Copyright (c) 1991, 2006, Oracle Corporation.  All rights reserved.

Welcome to LSNRCTL, type "help" for information.

LSNRCTL> start
Starting tnslsnr: please wait...

TNSLSNR for 32-bit Windows: Version 9.2.0.8.0 - Production
Log messages written to f:\oracle\92\network\log\listener.log
```

```
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=zhanglie)(PORT=1521)))

Connecting to (ADDRESS=(PROTOCOL=tcp)(PORT=1521))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for 32-bit Windows: Version 9.2.0.8.0 - Production
Start Date                21-SEP-2007 21:57:51
Uptime                    0 days 0 hr. 0 min. 2 sec
Trace Level               off
Security                  OFF
SNMP                      OFF
Listener Log File         f:\oracle\92\network\log\listener.log
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=zhanglie)(PORT=1521)))
The listener supports no services
The command completed successfully
LSNRCTL> status
Connecting to (ADDRESS=(PROTOCOL=tcp)(PORT=1521))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for 32-bit Windows: Version 9.2.0.8.0 - Production
Start Date                21-SEP-2007 21:57:51
Uptime                    0 days 0 hr. 0 min. 15 sec
Trace Level               off
Security                  OFF
SNMP                      OFF
Listener Log File         f:\oracle\92\network\log\listener.log
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=zhanglie)(PORT=1521)))
Services Summary...
Service "ora9" has 1 instance(s).
  Instance "ora9", status READY, has 2 handler(s) for this service...
The command completed successfully
LSNRCTL> stop
Connecting to (ADDRESS=(PROTOCOL=tcp)(PORT=1521))
The command completed successfully
LSNRCTL>exit
```

在 windows 下,第一次启动监听会产生一个服务, OracleOraDb10g_home1TNSListener
如果你配置得监听不是默认得名称,默认名称是 listener,假如你得监听名称是 ok.
我们启动监听,查看状态和停止监听得时候都要指明监听得名称
Lsnrctl start ok
Lsnrctl status ok
Lsnrctl stop ok
对应在服务中的服务就是 OracleOraDb10g_home1TNSok
当我们的监听启动失败,但配置都正确情况下,可能是 OracleOraDb10g_home1TNSListener
服务有问题了,请在注册表中删除所有含有关于监听的选项,使得服务不存在,重新配置再启动就可以了,因为是首次启动,会报错误,请把服务停止再启动就正常了,软件吗!有 bug 是正常的,我们原谅它了.
初始化参数 local_listener 很有作用,往往被我们忽视了.它的作用是注册一些服务到指定的监听,尤其我们使用的是浮动 ip,而不是本地的 ip 地址的时候.

实验 84：客户端的网络配置

该实验的目的是配置 tnsnames.ora 文件,进行客户端的配置.
客户端的配置
本地解析文件

配置文件 oracle_home\network\admin\tnsnames.ora
要素
网络服务名称（最好代表一定的含义如 ip 等）
远端主机的信息（主机名称或 ip,协议，端口号）
远端数据库的信息（数据库名称，oracle_home,实例名称）

服务器端的配置步骤
本地连接的 scott/tiger 帐号，如果可以，说明数据库已经 open.
配置或检验 listener.ora 文件
Lsnrctl status 查看监听的状态
本地配置网络服务名称，如 kk
Conn scott/tiger@kk
如果可以连接，说明监听完好

客户端配置步骤
配置 tnsnames.ora,假如网络服务名称为 qq
Ping 主机
Tnsping qq 8
Sqlplus 下 conn scott/tiger@qq
注意防火墙等，最好关闭

客户端 conn scott/tiger@qq 的步骤
1.首先在客户端的 sqlnet.ora 中 NAMES.DIRECTORY_PATH= (TNSNAMES, ONAMES, HOSTNAME)的选项来决定 qq 以什么样的方式来解析,如果没有上面的选项,那么就是上面的顺序,因为上面的解析顺序是默认值,我们通过调整参数的顺序来决定哪个优先使用,默认情况下优先使用 tnsnames 透明网络服务来解析,也就是在 tnsnames.ora 中查找是否有 qq 这个字符串.假如没有找到,就去 onames,oracle 的命名解析服务器来查找,如果还没有找到,就认为 qq 是一个主机的名称,当然 qq 得在 hosts 文件中可以解析.如果都没有找到 qq 是什么意思,就报错,无法解析服务名.
2.找到 qq 的含义后,就去相对应得主机,找 1521 端口.
3.监听判定你描述得实例名称有效,并且有 scott 用户,密码为 tiger
4.监听进程启动一个服务进程,如果使用共享模式就返回给客户端调度进程得信息.
5.监听进程将服务进程得信息传递给客户端
6.客户端重新连接到服务进程
7.监听得使命完成，等待下一个新得连接得请求

如果我们想在 oracle 的存储过程中**调用外部的 c 函数**.请配置 ipc 协议.并且在 tnsnames.ora 中建立一个名称.我们连接还是要走 tcp 协议,ipc 协议我们不能使用,数据库内部使用的.

Tnsnames.ora

```
EXTPROC_CONNECTION_DATA =
   (DESCRIPTION =
     (ADDRESS_LIST =
       (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
     )
     (CONNECT_DATA =
       (SID = PLSExtProc)
       (PRESENTATION = RO)
     )
   )

ME =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = zhanglie)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ora9)
    )
  )
```

这个解析名称是固定的,我们不能使用它来进行网络连接,它是内部使用的.

Listener.ora

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = zhanglie)(PORT = 1521))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = ora9)
      (ORACLE_HOME = F:\oracle\92)
      (SID_NAME = ora9)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ENVS = EXTPROC_DLLS=ANY)
      (ORACLE_HOME = F:\oracle\92)
      (PROGRAM = ExtProc)
    )
  )
```

Envs 的含义是使得动态连接库可以存在任何目录,而不是在特定的目录.

```
conn system/manager@me
```

以 tcp 协议连接到数据库.

```
create or replace library a_b as 'F:\oracle\92\plsql\TextOut.dll';
/
```

其中 **TextOut.dll** 是一个动态连接库,里面含有函数 c_tax,这是一个 C 语言的函数.返回两个值的和

```
select * from user_libraries;

GRANT EXECUTE ON a_b to public;

CREATE or replace FUNCTION tax_amt (
 x BINARY_INTEGER,
 y BINARY_INTEGER)
 RETURN BINARY_INTEGER
  AS LANGUAGE C
  LIBRARY A_B
  NAME "c_tax";
/
```

tax_amt 是数据库的函数,它调用了外部的动态连接库.验证函数的使用.

```
select tax_amt(1,2) from dual;
```

实验 85：数据库共享连接的配置

该实验的目的是使用数据库连接来访问远程的数据库中的表.
数据库的客户端发出请求,有两种模式连接到服务器.一种为共享模式,一种为专有模式.
到底走哪种连接模式,由服务器的配置和客户端的配置两方面决定.数据库管理员要明确使用专有连接连接到数据库,因为共享连接不能停止数据库.我们配置的原则是批处理走专有,小的事物走共享.共享连接的概念很好,可惜BUG太多,容易资源锁死,所以大部分用户放弃了ORACLE提供的共享连接,而买昂贵的第三方中

间件产品就是这个道理.oracle 只有数据库是很好的产品,其它的都不出色.请选择产品的时候仔细考虑.
服务器的配置有两个因素决定了连接的模式.
初始化参数: dispatchers, shared_servers, local_listener
监听的配置:一定要监听 ip 地址,因为有浮动 ip.
客户端的配置:tnsnames.ora 文件的设置

```
ls =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.1)(PORT = 1521))
    ))
#给 local_listener 参数使用的.

dc =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.1)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = emisdw)
      (SERVER = DEDICATED)
    )
  )
#明确指明必须走专有连接.
SH =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.1)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = emisdw)
      (SERVER = SHARED)
    )
  )
#明确指明必须走共享连接.

DEF =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.1)(PORT = 1521))
    )
    (CONNECT_DATA =

      (SID = emisdw)
       )
  )
```

没有明确指明必须走什么连接.那么监听服务有共享信息就走共享,没有监听到共享信息就走专有.

初始化参数的设置:

```
DISPATCHERS='(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.1.191))(DISPATCHERS=2)'
DISPATCHERS='(ADDRESS=(PROTOCOL=tcp)(HOST=zhanglie)(PORT=5000))(DISPATCHERS=1)'
```

我们设置二进制参数文件的时候可能报错,那我们就设纯文本的参数文件,在转化为二进制参数文件,文档描述了这个参数为动态的,可以直接修改,但很多情况下报错,没有关系,你改完参数文件后重新启动数据库,如果我没有指明 host 的选项,那么默认的走主机名称,我们也可以直接指明 ip 地址,尤其在有浮动 IP 地址的时候.如果我们不指定 PORT,数据库会自己指定端口号,我们也可以直接在参数中描述.

```
shared_servers=2
local_listener=LS
```

其中 LS 必须在 tnsnames.ora 中有正确的描述,见上面的配置. local_listener 的作用是将应用注册到指定的监听中,监听的注册是通过 PMON 进程完成的,所以我有的时候要等待一分钟才能注册,因为 PMON 是轮寻进程.alter system register;立即注册调度进程到监听中.如果监听中没有 d000 的信息,那么走共享连接

的就不能连接,只能走专有连接.

Lsnrctl services
看到有
"D000" established:0 refused:0 current:0 max:1002 state:ready
DISPATCHER <machine: ZHANGLIE, pid: 2576>
(ADDRESS=(PROTOCOL=tcp)(HOST=zhanglie)(PORT=5000))

```
SQL> select SERVER,count(*) from  v$session group by server;

SERVER      COUNT(*)
--------- ----------
DEDICATED         11
NONE               4
```

显示为 none 的就是走的共享.

```
set lines 100
set pages 100
col MACHINE for a30
select MACHINE,server,username,sid,serial# from v$session order by 1,2,3;
```
该语句查看每个会话的连接模式

实验 86：数据库 dblink

该实验的目的是使用数据库连接来访问远程的数据库中的表.

当我们想查询远程的数据库的表,同时和本地数据库的表进行联合操作的时候,我们需要数据库连接(database link).
我们先配本地的 tnsnames.ora,使得可以登录到远程的数据库,比如在 tnsnames.ora 中描述了 111 这个字符串.我们测试一下,conn scott/tiger@111 可以连接.
Conn / as sysdba 连接到本地的数据库.
CREATE PUBLIC DATABASE LINK dh connect to system identified by manager using '111';
我们建立了一个共有的数据库连接,名称是 dh,连接到 111 所描述的数据库中的 system 用户,密码为 manager.
SELECT * FROM DBA_DB_LINKS;
select * from v$instance@dh
--关闭数据库连接
ALTER SESSION CLOSE DATABASE LINK dh;

drop PUBLIC DATABASE LINK dh ;

global_names = TRUE 时连接名称必须和数据库的全局名称相同
网络连接写 SID=，而不要写 serives=

第五部分数据库的备份和恢复

Exp 导出和 imp 导入

EXP 的概念
1、EXP 是 ORACLE 的小工具。用来操作数据库中的数据。
2、EXP 只备份数据，和物理结构无关。
3、数据库必须在 OPEN 下，才可以使用 EXP 和 IMP。
4、导出的是一个二进制文件

使用 EXP 的目的
1、数据库的迁移
2、归档历史的数据
3、重新组织表
4、转移数据给其它数据库
5、物理备份的辅助

EXP 和 IMP 的使用方式
1、交互模式
2、命令行模式
3、参数文件模式
4。图形向导模式

实验 87：交互模式导出和导入数据

该实验的目的是初步认识逻辑备份
交互模式
在操作系统下键入：exp
然后在提示下完成导出
缺点：
参数提示不全
下次运行不能自动重复，必须手工输入
实验：在交互模式下导出 EMP 表

```
C:\bk>exp

Export: Release 9.2.0.8.0 - Production on Sat Sep 22 09:36:28 2007

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Username: scott/tiger

Connected to: Oracle9i Enterprise Edition Release 9.2.0.8.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.8.0 - Production
Enter array fetch buffer size: 4096 > 8192

Export file: EXPDAT.DMP > c:\bk\1.dmp

(2)U(sers), or (3)T(ables): (2)U > t

Export table data (yes/no): yes >
```

```
Compress extents (yes/no): yes >
```
回车就是默认
```
Export done in ZHS16GBK character set and AL16UTF16 NCHAR character set

About to export specified tables via Conventional Path ...
Table(T) or Partition(T:P) to be exported: (RETURN to quit) > t1

. . exporting table                             T1          1 rows exported
Table(T) or Partition(T:P) to be exported: (RETURN to quit) > dept

. . exporting table                           DEPT          4 rows exported
Table(T) or Partition(T:P) to be exported: (RETURN to quit) > emp

. . exporting table                            EMP         14 rows exported
Table(T) or Partition(T:P) to be exported: (RETURN to quit) >

Export terminated successfully without warnings.
```

实验 88：命令行模式导出和导入数据

该实验的目的是使用命令行模式进行逻辑备份
命令行模式
用显式的模式书写所需要的参数值
缺点：
当参数太多的时候书写不便，可读性差。特殊字符需要转义
实验：用命令行模式导出 EMP 表
```
C:\bk>exp scott/tiger file=c:\bk\1.dmp tables=emp,dept,t1 log=c:\bk\1.log

Export: Release 9.2.0.8.0 - Production on Sat Sep 22 09:50:17 2007

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to: Oracle9i Enterprise Edition Release 9.2.0.8.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.8.0 - Production
Export done in ZHS16GBK character set and AL16UTF16 NCHAR character set

About to export specified tables via Conventional Path ...
. . exporting table                            EMP         14 rows exported
. . exporting table                           DEPT          4 rows exported
. . exporting table                             T1          1 rows exported
Export terminated successfully without warnings.
```

实验 89：参数文件模式导出和导入数据

该实验的目的是使用参数文件进行逻辑备份
参数文件模式
强烈推荐的模式
书写方便，可读性好，可以反复调用
```
Exp parfile=d:\bk\1.txt
```
实验：用参数文件模式导出 EMP 表
参数文件中的**注释是#**
c:\bk\e.txt 文件的内容如下

```
userid=scott/tiger
buffer=100000
log=c:\bk\exp.log
file=c:\bk\exp_users.dmp
feedback=10000
tables=emp

C:\bk>exp parfile=c:\bk\e.txt
```

图形模式
用 OEM 的图形向导来建立导出的参数文件，必须正确的配置**首选项**的参数。主要有操作系统的密码和数据库的密码**两个配置**。其中操作系统的帐号必须有**批处理作业的权限**。

导出的内容
1、导出表
2、导出用户
3、全数据库导出
4、导出表空间的定义

实验 90：导出和导入表的操作

该实验的目的是通过对表的备份和恢复,掌握数据库的逻辑备份
1.表的备份和恢复
导出表 t1,将 t1 表从数据库中删除 drop table t1;导入表 t1,验证表已经恢复

2.将 scott 用户的表**导入到 u1 用户**
建立 u1 用户,并赋予权限

```
conn / as sysdba
drop user u1 cascade;
create user u1 identified by u1;
grant connect ,resource to u1;
alter user u1 default tablespace users;
```

在 scott 用户下建立实验表 t81

```
conn scott/tiger
drop table t81;
create table t81 as select * from emp;
update t81 set sal=81;
commit;
```

书写导出参数文件
c:\bk\e81.txt 文件的内容如下

```
userid=scott/tiger
buffer=100000
log=c:\bk\exp.log
file=c:\bk\t81.dmp
feedback=10000
tables=t81
```

导出 t81 表

```
exp parfile=c:\bk\e81.txt
```

书写导入参数文件
c:\bk\i81.txt 文件的内容如下

```
userid=system/manager
buffer=100000
```

```
log=c:\bk\exp.log
file=c:\bk\t81.dmp
feedback=10000
tables=t81
touser=u1

imp parfile=c:\bk\i81.txt

conn u1/u1
select * from t81;
```

3.将数据库一的 scott 用户的表导入到**数据库二**的 u1 用户
发生字符集的转换,先由数据库一的字符集转为导出客户端的字符集.
再由导出的字符集转换为导入的数据库二的字符集.最好都设置为相同的
字符集.如果一定要转换,请由小的字符集转到比它大的字符集.

```
userid=system/manager@remote_db
```

4.导出一张表的**部分符合条件的行**.用来备份历史数据和避开坏的数据块.
书写导出参数文件
c:\bk\e81.txt 文件的内容如下

```
userid=scott/tiger
buffer=100000
log=c:\bk\exp.log
file=c:\bk\t81.dmp
feedback=10000
tables=t81
query='where deptno=30'
```

建立大的表,导出一张表除了某个块的所有行.

```
conn scott/tiger
drop table t1;
create table t1 as select * from all_objects;
select dbms_rowid.rowid_block_number(rowid) block#,count(*) from
t1 group by  dbms_rowid.rowid_block_number(rowid) ;
 BLOCK#   COUNT(*)
------- ----------
     68         91
     69         86
     70         85
     71         88
```

假如我想导出除了 71 块以外的行.
书写导出参数文件
c:\bk\e1.txt 文件的内容如下

```
userid=scott/tiger
buffer=100000
log=c:\bk\exp.log
file=c:\bk\t1.dmp
feedback=10000
tables=t1
query='where dbms_rowid.rowid_block_number(rowid) <>71'
```

5.追加导入数据
当建立表失败的时候,不终止导入,接着将数据追加到表中,当然表的结构要相同.如果
有约束的情况下不能违反约束
c:\bk\i1.txt 文件的内容如下

```
userid=scott/tiger
```

```
buffer=100000
log=c:\bk\exp.log
file=c:\bk\t1.dmp
feedback=10000
tables=t1
ignore =y
```

总结:导出表的时候是将表的所有信息都导出,包含索引的定义,对象权限,约束等.
导入的时候会找原来的表空间建立表,如果原来的表空间不存在,就建立表在导入用户的
默认表空间.所以如果原来表在 system 就很难处理,因为每次都先回 system 表空间,每个数据库
都有 system 表空间.这时我们要先将表挪动到其他表空间后再导出和导入.
和导出导入速度最相关的参数是 buffer,设置正确的大小.一般我们在内存大的情况下设置 100m 或者更大.
内存小的情况下设置为 10m 左右.feedback 是进度条,一般设置为最大表的百分之一.千万不要
设置为 1,比如 feedback=10000 的含义是完成 10000 行以一个点来显示.

实验 91：导出和导入用户操作

该实验的目的是逻辑备份用户
导出用户
书写导出参数文件
c:\bk\e1.txt 文件的内容如下

```
userid=scott/tiger
buffer=100000
log=c:\bk\exp.log
file=c:\bk\t1.dmp
feedback=10000
```

参数文件中不说导出什么内容,默认的是导出当前用户的所有数据.

同时导出多个用户
c:\bk\e1.txt 文件的内容如下

```
userid=system/manager
buffer=100000
log=c:\bk\exp.log
file=c:\bk\t1.dmp
feedback=10000
owner=HR,scott sh
#owner 列表可以是以空格分割,逗号分割,或者回车分割
```

导入用户,建立用户并赋予权限,再进行导入.

```
conn / as sysdba
drop user scott cascade;
grant connect ,resource to scott identified by tiger;
alter user scott default tablespace users;
```

c:\bk\i1.txt 文件的内容如下

```
userid=system/manager
buffer=100000
log=c:\bk\exp.log
file=c:\bk\t1.dmp
feedback=10000
fromuser=scott
touser=scott
```

实验 92：导出和导入全数据库操作

该实验的目的是逻辑备份全数据库
全库导出
userid=system/manager
buffer=100000
file=d:\bk\1.dmp
log=d:\bk\1.log
feedback=10000
Full=y
导出除 SYS 用户以外的所有其它用户的数据，以及索引，约束和同义词等。

实验 93：导出和导入表空间操作

该实验的目的是逻辑备份表空间内的数据,不含有存储过程.
导出表空间
userid=system/manager
buffer=100000
file=d:\bk\1.dmp
log=d:\bk\1.log
feedback=1
Tablespaces=users
导出 users 表空间内的所有表，以及这些表的约束和触发器。

数据泵(data pump)10g 新特性
只能将数据存储在服务器端
不支持物理路径
直接用 api 来加载和卸载 数据
性能要比 exp/imp 快的多
导出的控制更强，如只导出函数，存储过程等
监控信息更加丰富

实验 94：数据泵

该实验的目的是使用 10g 的新特性,数据泵.
导出 t1
Sqlplus>
conn system/manager
create directory dpdata1 as 'c:\bk';
grant read, write on directory dpdata1 to scott;
Dos>expdp help=y
expdp scott/tiger tables=t1 directory=DPDATA1
dumpfile=pump_t1.dmp job_name=CASES_EXPORT

导入 t1
impdp scott/tiger tables=t1 directory=DPDATA1 dumpfile= pump_t1.dmp job_name=CASES_EXPORT

监控作业
DBA_DATAPUMP_JOBS
DBA_DATAPUMP_SESSIONS
V$SESSION
V$SESSION_LONGOPS

导入和导出过程中
Control-C 进入交互模式
CONTINUE_CLIENT 继续工作

冷备份

停止数据库后做的备份
所有的数据库都可以冷备份
冷备份不能备局部，必须备份整体
没有增量备份策略
需要的空间较大
概念简单，执行简单

冷备份的执行步骤
1。一致性停止数据库（shutdown immediate）
2。备份数据文件，控制文件，日志文件
　　密码文件，参数文件，临时文件（可选）
3。启动数据库

书写冷备份脚本

```
select 'copy '||name||' d:\bk' from v$datafile
union all
select 'copy '||name||' d:\bk' from v$controlfile
union all
select 'copy '||name||' d:\bk' from v$tempfile
union all
select 'copy '||member||' d:\bk' from v$logfile;


copy D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF d:\bk
copy D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF d:\bk
copy D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF d:\bk
copy D:\ORACLE\ORADATA\ORA10\USERS01.DBF d:\bk
copy D:\ORACLE\ORADATA\ORA10\TL.DBF d:\bk
copy D:\ORACLE\ORADATA\ORA10\T2M.DBF d:\bk
copy D:\ORACLE\ORADATA\ORA10\BIGTS.BIG d:\bk
copy D:\ORACLE\ORADATA\ORA10\CONTROL01.CTL d:\bk
copy D:\ORACLE\ORADATA\ORA10\CONTROL02.CTL d:\bk
copy D:\ORACLE\ORADATA\ORA10\CONTROL03.CTL d:\bk
copy D:\ORACLE\ORADATA\ORA10\TEMP.TMP d:\bk
copy D:\ORACLE\ORADATA\ORA10\TEMP03.DBF d:\bk
copy D:\ORACLE\ORADATA\ORA10\REDO03.LOG d:\bk
copy D:\ORACLE\ORADATA\ORA10\REDO02.LOG d:\bk
copy D:\ORACLE\ORADATA\ORA10\REDO01.LOG d:\bk
```

当然这是脚本的局部,前面得加停止数据库,后面加启动数据库,另外加上参数文件和密码文件得备份命令.
运行脚本
查看日志
1。数据库停止了
2。复制成功了
3。数据库启动了

优点
冷备份是最可靠的备份
在不完全恢复前和后都最好做冷备份
缺点

必须停止数据库
空间占用大,冷备份是一个整体,不能备份局部.所以使用的时候有一定的局限性,当然你是归档数据库可以备份局部,但一般我们不这么做,因为归档数据库可以热备份,不必停止数据库.

冷备份的恢复
停止数据库，将备份复制回原来的目录就可以
将数据库带回到备份的时间点
如果想恢复全部的交易
请应用归档的日志

实验 95：将冷备份恢复到其它目录

该实验的目的是理解什么是 mount,如何修改文件的路径.
练习
将冷备份的数据库恢复到其它磁盘
1。将冷备份的数据库恢复到不同的磁盘
2。修改参数文件中 control_files 的值，指到新的文件
3。启动数据库到 mount 状态
4。改文件的名称到新位置
5。Alter database open;

实验 96：修改实例的名称

该实验的目的是理解什么是 nomount,注册表和服务的配置.
实例的名称可以更改
1。修改 instance_name 参数的值，改为新的名称###
2。建立新的服务项
    oradim –new –sid ###
3.修改注册表中 oracle_sid=###
4.重新进入 sqlplus
5.启动数据库
6。验证 select instance_name from v$instance;
7.修改监听和网络服务名称的配置。

通过修改实例的名称
更加深入理解实例的概念
实例是访问数据库的方法
实例由内存加后台进程组成

条件
有冷备份
目的
将该备份恢复到其它主机上
实际应用
准备备用数据库
重大灾难导致主机不可再用

实验 97：将冷备份恢复到其它主机

该实验的目的是彻底的理解冷备份,冷备份是放之四海皆准,可以跨越平台,但得重新建立控制文件.理解 dump 目录得重要性.
1。新主机已经安装了同版本的数据库产品
2。操作系统一致
3。将备份恢复到任意目录

4。将参数文件和密码文件恢复到 oracle_home\database
　　建立参数文件中所描述的路径，或改为新路径
5。修改注册表 oracle_sid
6。建立服务项 oradim –new –sid ###
7.修改参数文件的 control_files 到新的目标
8。Startup mount
9.更改文件名称 alter database rename file '...' to '...';
10.alter database open;

实验目的
理解产品和数据库的关系

数据库的运行基本模式
非归档数据库（默认模式）
　日志切换后不复制到其它位置，下次使用时将老信息覆盖。如果日志被覆盖，以前的备份就只能恢复到日志被覆盖前的备份时间点。
归档模式
　日志切换后要复制到其它位置。

数据库处于哪种模式由控制文件决定

实验 98：将数据库改为归档数据库

该实验的目的是理解什么是归档,极其归档数据库得配置
　　改为归档数据库
修改参数
log_archive_format　= ARC%S_%R.%T
log_archive_dest_1 ='location=c:\arc'
停止数据库（一定要一致性停数据库）
启动到 mount 状态
Alter database archivelog;
Alter database open;
验证 archive log list;

归档数据库的维护
Alter system switch logfile;
查看归档目录
select * from v$archive_processes;
 select * from v$archived_log;
Select * from v$log;

如果归档进程报错
archive log stop;
archive log start;

热备份

数据库 open 下的备份
数据库必须处于归档状态
可以备份局部
没有增量备份策略

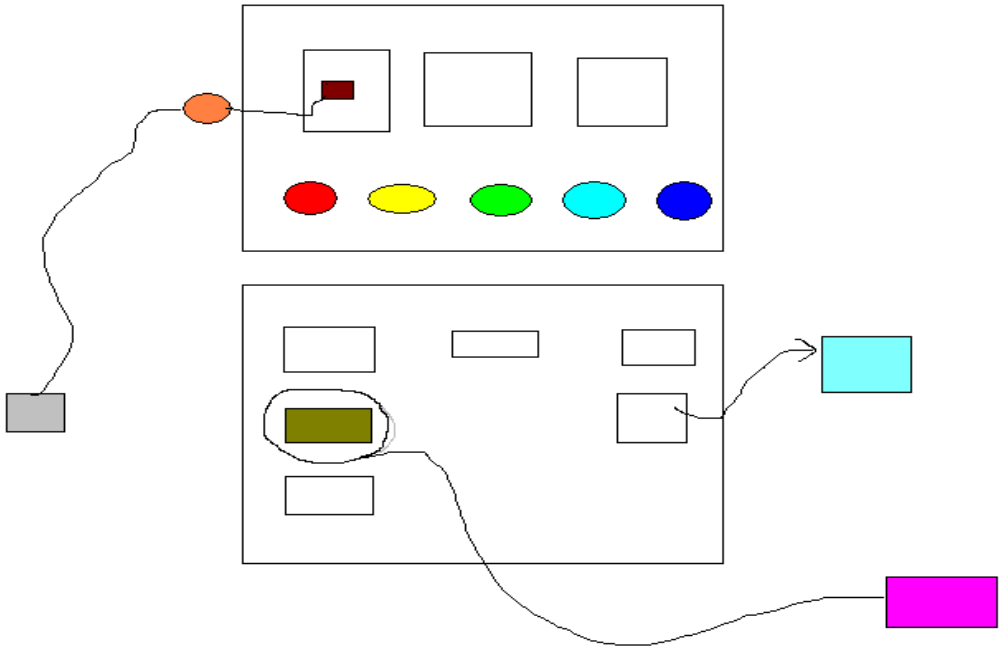
备份内容
数据文件

控制文件
归档日志文件
密码文件
参数文件
不能备份在线的日志文件

数据文件的备份

Alter tablespace users begin backup;
Host copy ####### ********
Alter tablespace users end backup;
除了临时表空间外，所有表空间都要做一遍。
Select * from v$backup;

交易一直存在，当你复制的时候，文件已经变化了



Alter tablespace users begin backup;
1.将该表空间的文件**单独存盘**。
2。将该表空间的**文件头冷冻**
3。日志的产生**加入**了变化块的原来拷贝
4.数据**文件体**不影响，因为文件头中没有我们的数据，所以交易可以继续
5。恢复的时候需要归档文件的支持

Alter tablespace users end backup;
将文件**头解冻**
将控制文件中最新的存盘时间 SCN 写入文件头
一句话,热备份的文件是一个**无效的垃圾文件**,需要**日志的配合**才能恢复,所以**归档数据库**是热备份的前提条件.热备份的文件中**只有一个**数据块是保真的,就是被冷冻的数据文件头的**第一个块**,文件头有 8 个块,数据库只会冷冻一个,因为数据库只是需要一个 scn **坐标**而已.其余的 7 个数据块含有范围的信息,是会改变的,不能冷冻.我们备份的垃圾文件的数据块有两类,一是 **scn 小于头**的,另一类是 **scn 大于头**的块.凡是 scn 大于文件头的块都有一个该块的原形存在于日志文件中.scn 是数据库运行的不二法则,相当于现实世界中的时间,你可能没有觉察时间的存在,但任何事情都和时间相关.你使用 oracle 很多年,但你也可能不知道 scn.不懂 scn 就不会懂得 oracle.oracle 的**一切运行都有 scn 参与**.scn 数据库的时间.

实验 99：热备份数据文件

该实验的目的是理解什么是热备份,掌握热备份的每个步骤都发生了什么.

```
SQL> conn / as sysdba
Connected.
SQL> select name,checkpoint_change# from v$datafile;

NAME                                     CHECKPOINT_CHANGE#
---------------------------------------- ------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF               3328702549
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF              3328702549
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF               3328702549
D:\ORACLE\ORADATA\ORA10\USERS01.DBF                3328702549
D:\ORACLE\ORADATA\ORA10\TL.DBF                     3328702549
D:\ORACLE\ORADATA\ORA10\T2M.DBF                    3328702549
D:\ORACLE\ORADATA\ORA10\BIGTS.BIG                  3328702549
```

所有的号码**都是一致的**

```
7 rows selected.

SQL> select * from v$backup;

     FILE# STATUS        CHANGE# TIME
---------- ---------- ---------- ------------
         1 NOT ACTIVE          0
         2 NOT ACTIVE     558098 29-MAR-06
         3 NOT ACTIVE          0
         4 NOT ACTIVE 3328562022 02-SEP-07
         5 NOT ACTIVE          0
         6 NOT ACTIVE          0
         8 NOT ACTIVE          0
```

没有处于备份活动状态的文件,这些文件都是正常的.**非活动备份**状态.

```
7 rows selected.
SQL> alter tablespace USERS begin backup;

Tablespace altered.

SQL> select * from v$backup;

     FILE# STATUS        CHANGE# TIME
---------- ---------- ---------- ------------
         1 NOT ACTIVE          0
         2 NOT ACTIVE     558098 29-MAR-06
         3 NOT ACTIVE          0
         4 ACTIVE     3328706439 22-SEP-07
         5 NOT ACTIVE          0
         6 NOT ACTIVE          0
         8 NOT ACTIVE          0

7 rows selected.

SQL> select name,checkpoint_change# from v$datafile;

NAME                                     CHECKPOINT_CHANGE#
---------------------------------------- ------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF               3328702549
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF              3328702549
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF               3328702549
D:\ORACLE\ORADATA\ORA10\USERS01.DBF                3328706439
```

```
D:\ORACLE\ORADATA\ORA10\TL.DBF                          3328702549
D:\ORACLE\ORADATA\ORA10\T2M.DBF                         3328702549
D:\ORACLE\ORADATA\ORA10\BIGTS.BIG                       3328702549
```
User01.dbf 最大号,因为它单独存盘了.
```
7 rows selected.
SQL> alter system checkpoint;
```
强制产生完全存盘
```
System altered.

SQL> select name,checkpoint_change# from v$datafile;

NAME                                               CHECKPOINT_CHANGE#
-------------------------------------------------- ------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF                       3328706466
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF                      3328706466
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF                       3328706466
D:\ORACLE\ORADATA\ORA10\USERS01.DBF                        3328706439
D:\ORACLE\ORADATA\ORA10\TL.DBF                             3328706466
D:\ORACLE\ORADATA\ORA10\T2M.DBF                            3328706466
D:\ORACLE\ORADATA\ORA10\BIGTS.BIG                          3328706466
```
User01.dbf 的 scn 号没有变化,因为它存盘了,但是没有写文件头,文件头被冷冻了.其它文件头都写入了最新的 scn,使得 User01.dbf 的 scn 号变为最小了.

```
7 rows selected.
SQL>host copy D:\ORACLE\ORADATA\ORA10\USERS01.DBF d:\bk
```
拷贝这个文件,但不保证完整性,因为有交易存在,所以拷贝的是一个垃圾文件.
```
SQL>alter tablespace USERS end backup;
Tablespace altered.
```
结束备份
```
SQL> select * from v$backup;

     FILE# STATUS        CHANGE# TIME
---------- ---------- ---------- ------------
         1 NOT ACTIVE          0
         2 NOT ACTIVE     558098 29-MAR-06
         3 NOT ACTIVE          0
         4 NOT ACTIVE 3328706439 22-SEP-07
         5 NOT ACTIVE          0
         6 NOT ACTIVE          0
         8 NOT ACTIVE          0

7 rows selected.

SQL> select name,checkpoint_change# from v$datafile;

NAME                                               CHECKPOINT_CHANGE#
-------------------------------------------------- ------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF                       3328706466
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF                      3328706466
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF                       3328706466
D:\ORACLE\ORADATA\ORA10\USERS01.DBF                        3328706466
D:\ORACLE\ORADATA\ORA10\TL.DBF                             3328706466
D:\ORACLE\ORADATA\ORA10\T2M.DBF                            3328706466
D:\ORACLE\ORADATA\ORA10\BIGTS.BIG                          3328706466
```
所有文件头又一致了,文件头解冻了,一切正常了.
```
7 rows selected.
```

我们书写一个脚本来完成备份的工作.
```
select 'alter tablespace '||tablespace_name|| ' begin backup;'
```

```
||chr(10)
||'host copy '||file_name||' d:\bk'
||chr(10)
||'alter tablespace '||tablespace_name|| ' end backup;'
from dba_data_files order by tablespace_name;
```

实验 100：热备份控制文件

该实验的目的是对控制文件进行备份

控制文件的备份

备份建立控制文件的脚本到 udump 目录

```
Alter database backup controlfile to trace;
```

将当前的控制文件备份到文件

```
Alter database backup controlfile to ‘c:\bk\control.bak’;
```

编写热备份脚本

```
select 'alter tablespace '||tablespace_name||' begin backup;'
||chr(10)
||'host copy '||file_name||' c:\bk'
||chr(10)
||'alter tablespace '||tablespace_name||' end backup;'
from dba_data_files order by tablespace_name;

Alter database backup controlfile to ‘c:\bk\control.bak’;
```

控制文件的三种备份

1。停数据库，复制控制文件

2。热备份建立控制文件的脚本到 udump 目录

```
Alter database backup controlfile to trace;
```

3。热备份将当前的控制文件备份到文件

```
Alter database backup controlfile to ‘c:\bk\control.bak’;
```

控制文件恢复（1）

用最新的控制文件复制，然后修改成需要的名称

例如：

增加新的控制文件

实验 101：改变控制文件大大小

该实验的目的是建立新的控制文件

控制文件恢复（2）

重新建立新的控制文件。可以修改大小

使用 noresetlogs 选项

```
CREATE CONTROLFILE REUSE DATABASE "O10" NORESETLOGS  NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 292
```

```
LOGFILE
  GROUP 1 'D:\ORACLE\ORADATA\O10\REDO01.LOG'  SIZE 50M,
  GROUP 2 'D:\ORACLE\ORADATA\O10\REDO02.LOG'  SIZE 50M
DATAFILE
  'D:\ORACLE\ORADATA\O10\SYSTEM01.DBF',
  ..................
CHARACTER SET ZHS16GBK;
```

实验 102：改变数据库的名称

该实验的目的是修改数据库的名称,虽然有命令来修改数据库的名称,但没有我们手工修改理解的深刻.
控制文件恢复（3）
重新建立新的控制文件。可以修改数据库的名称
使用 resetlogs 选项,要修改 db_name 参数,这个参数需要在 nomount 下修改.

```
CREATE CONTROLFILE REUSE DATABASE "O10"
set database new   RESETLOGS  NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 292
LOGFILE。。。。。。
```

实验 103：使用老的控制文件进行数据库恢复

该实验的目的是使用老控制文件来恢复数据库,一句话,老控制文件不知道数据库的终点.
控制文件恢复（4）
用备份的控制文件复制
数据库启动到 mount

```
Recover database using backup controlfile;
```

会失败,因为老的控制文件不知道最后一个日志文件的号码,所以在恢复的最后会找一个归档文件,这个文件并没有存在,因为当前的日志文件总是没有归档的.

```
Select member from v$logfile;
```

查找到所有的在线日志文件,挨个去实验,失败后再次恢复,直到成功.

```
Alter database open resetlogs;
Select * from v$log;
```

Resetlogs 后一定要执行全备份，老备份失效了。

数据文件失败的恢复

1。非系统表空间恢复
2。System 或有活动事务的 undo 段失败的恢复
3。索引或 temp 表空间的恢复
4。无备份的表空间的恢复

实验 104：系统表空间损坏的恢复

该实验的目的是局部恢复数据文件,系统表空间损坏必须在 mount 下恢复.
非系统表空间恢复
可以 offline 的表空间，数据库不必停，open 下恢复
1。验证状态 select name,staus from v$datafile;
2。Alter database datafile '###' Offline;
3.从备份中将要恢复的文件复原。

4。Recover datafile ####;
5.alter database datafile '###' online;
6.查看交易的数据是否存在

实验 105：非系统表空间损坏的恢复

该实验的目的是在不停数据库的情况下恢复表空间
系统表空间或 undo 表空间恢复
不可以 offline 的表空间
数据库必须停，启动到 mount 下恢复
1。从备份中将要恢复的文件复原。
2。 Recover datafile ####;
3.alter database open;
4。查看交易的数据是否存在.

```
SQL>
SQL> --开始备份
SQL> select 'alter tablespace '||tablespace_name|| ' begin backup;'
  2  ||chr(10)
  3  ||'host copy '||file_name||' d:\bk /y'
  4  ||chr(10)
  5  ||'alter tablespace '||tablespace_name|| ' end backup;'
  6  from dba_data_files where tablespace_name='USERS';

'ALTERTABLESPACE'||TABLESPACE_NAME||'BEGINBACKUP;'||CHR(10)||'HOSTCOPY'||FILE_NAME||'D:\BK/Y
'||CHR(1
--------------------------------------------------------------------------------
alter tablespace USERS begin backup;
host copy F:\ORACLE\ORADATA\ORA9\USERS01.DBF d:\bk /y
alter tablespace USERS end backup;


SQL> --运行命令
SQL> alter tablespace USERS begin backup;

表空间已更改。

SQL> host copy F:\ORACLE\ORADATA\ORA9\USERS01.DBF d:\bk /y

SQL> alter tablespace USERS end backup;

表空间已更改。

SQL>
SQL> --验证备份
SQL> select * from V$backup;

     FILE# STATUS        CHANGE# TIME
---------- ---------- ---------- ----------
         1 NOT ACTIVE          0
         2 NOT ACTIVE          0
         3 NOT ACTIVE          0
         4 NOT ACTIVE          0
         5 NOT ACTIVE          0
         6 NOT ACTIVE          0
         7 NOT ACTIVE          0
         8 NOT ACTIVE          0
         9 NOT ACTIVE    1936044 09-1 月 -07
```

```
        10 NOT ACTIVE          0
        11 NOT ACTIVE          0
        12 NOT ACTIVE          0

已选择 12 行。

SQL> --开始交易
SQL> select * from scott.emp;

     EMPNO ENAME      JOB              MGR HIREDATE          SAL  COMM DEPTNO
---------- ---------- --------- ---------- ---------- ---------- ----- ------
      7369 SMITH      CLERK           7902 17-12 月-80       1675            20
      7499 ALLEN      SALESMAN        7698 20-2 月 -81       2474   300      30
      7521 WARD       SALESMAN        7698 22-2 月 -81       2124   500      30
      7566 JONES      MANAGER         7839 02-4 月 -81       3849            20
      7654 MARTIN     SALESMAN        7698 28-9 月 -81       2124  1400      30
      7698 BLAKE      MANAGER         7839 01-5 月 -81       3724            30
      7782 CLARK      MANAGER         7839 09-6 月 -81       3324            10
      7839 KING       PRESIDENT            17-11 月-81       5874            10
      7844 TURNER     SALESMAN        7698 08-9 月 -81       2374     0      30
      7900 JAMES      CLERK           7698 03-12 月-81       1824            30
      7902 FORD       ANALYST         7566 03-12 月-81       3874            20
      7934 MILLER     CLERK           7782 23-1 月 -82       2174            10

已选择 12 行。

SQL> update scott.emp set sal=sal+1;

已更新 12 行。

SQL> commit;

提交完成。

SQL> alter system switch logfile;

系统已更改。

SQL> update scott.emp set sal=sal+1;

已更新 12 行。

SQL> commit;

提交完成。

SQL> alter system switch logfile;

系统已更改。

SQL> update scott.emp set sal=sal+1;

已更新 12 行。

SQL> commit;

提交完成。

SQL> alter system switch logfile;
```

系统已更改。

```
SQL> update scott.emp set sal=sal+1;
```

已更新 12 行。

```
SQL> commit;
```

提交完成。

```
SQL> alter system switch logfile;
```

系统已更改。

```
SQL> --查看最后的金额
SQL> select * from scott.emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12 月-80	1679		20
7499	ALLEN	SALESMAN	7698	20-2 月 -81	2478	300	30
7521	WARD	SALESMAN	7698	22-2 月 -81	2128	500	30
7566	JONES	MANAGER	7839	02-4 月 -81	3853		20
7654	MARTIN	SALESMAN	7698	28-9 月 -81	2128	1400	30
7698	BLAKE	MANAGER	7839	01-5 月 -81	3728		30
7782	CLARK	MANAGER	7839	09-6 月 -81	3328		10
7839	KING	PRESIDENT		17-11 月-81	5878		10
7844	TURNER	SALESMAN	7698	08-9 月 -81	2378	0	30
7900	JAMES	CLERK	7698	03-12 月-81	1828		30
7902	FORD	ANALYST	7566	03-12 月-81	3878		20
7934	MILLER	CLERK	7782	23-1 月 -82	2178		10

已选择 12 行。

```
SQL> --破坏数据文件
SQL> host copy d:\bk\1.txt F:\ORACLE\ORADATA\ORA9\USERS01.DBF

SQL>
SQL> --查看内存中的影象
SQL>  select * from scott.emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12 月-80	1679		20
7499	ALLEN	SALESMAN	7698	20-2 月 -81	2478	300	30
7521	WARD	SALESMAN	7698	22-2 月 -81	2128	500	30
7566	JONES	MANAGER	7839	02-4 月 -81	3853		20
7654	MARTIN	SALESMAN	7698	28-9 月 -81	2128	1400	30
7698	BLAKE	MANAGER	7839	01-5 月 -81	3728		30
7782	CLARK	MANAGER	7839	09-6 月 -81	3328		10
7839	KING	PRESIDENT		17-11 月-81	5878		10
7844	TURNER	SALESMAN	7698	08-9 月 -81	2378	0	30
7900	JAMES	CLERK	7698	03-12 月-81	1828		30
7902	FORD	ANALYST	7566	03-12 月-81	3878		20
7934	MILLER	CLERK	7782	23-1 月 -82	2178		10

已选择 12 行。

```
SQL> --切换会使损坏的表空间离线,但9i以前的数据库会有两种结果，1。崩溃。2。offline
SQL> alter system switch logfile;

系统已更改。

SQL> /

系统已更改。

SQL> /

系统已更改。

SQL>  select * from scott.emp;
 select * from scott.emp
                     *
ERROR 位于第 1 行:
ORA-00376: 此时无法读取文件 9
ORA-01110: 数据文件 9: 'F:\ORACLE\ORADATA\ORA9\USERS01.DBF'


SQL> --如果崩溃，请startup,alter database datafile ### offline; alter database open;
SQL> --企图online
SQL> alter tablespace users online;
alter tablespace users online
*
ERROR 位于第 1 行:
ORA-01157: 无法标识/锁定数据文件 9 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 9: 'F:\ORACLE\ORADATA\ORA9\USERS01.DBF'
因为该文件不是正确的数据文件.

SQL> --恢复备份后，再online
SQL> host copy d:\bk\USERS01.DBF F:\ORACLE\ORADATA\ORA9\USERS01.DBF

SQL> alter tablespace users online;
alter tablespace users online
*
ERROR 位于第 1 行:
ORA-01113: 文件 9 需要介质恢复
ORA-01110: 数据文件 9: 'F:\ORACLE\ORADATA\ORA9\USERS01.DBF'
因为该文件是正确的数据文件.但和控制文件中记录的时间有差距


SQL> select * from V$recover_file;

     FILE# ONLINE  ONLINE_ ERROR
---------- ------- ------- ----------------------------------------------------------------
   CHANGE# TIME
---------- ----------
         9 OFFLINE OFFLINE
   1936044 09-1月 -07


SQL> select * from V$recovery_log;

 THREAD#  SEQUENCE# TIME       ARCHIVE_NAME
-------- ---------- ---------- ----------------------------------------
       1          9 09-1月 -07 C:\ARC\ORA9_9.ARC
       1         10 09-1月 -07 C:\ARC\ORA9_10.ARC
```

```
         1          11 09-1月 -07 C:\ARC\ORA9_11.ARC
         1          12 09-1月 -07 C:\ARC\ORA9_12.ARC
         1          13 09-1月 -07 C:\ARC\ORA9_13.ARC

SQL> recover datafile 9;
ORA-00279: 更改 1936044 (在 01/09/2007 13:50:41 生成) 对于线程 1 是必需的
ORA-00289: 建议: C:\ARC\ORA9_9.ARC
ORA-00280: 更改 1936044 对于线程 1 是按序列 # 9 进行的


指定日志: {<RET>=suggested | filename | AUTO | CANCEL}

ORA-00279: 更改 1936215 (在 01/09/2007 13:51:34 生成) 对于线程 1 是必需的
ORA-00289: 建议: C:\ARC\ORA9_10.ARC
ORA-00280: 更改 1936215 对于线程 1 是按序列 # 10 进行的
ORA-00278: 此恢复不再需要日志文件 'C:\ARC\ORA9_9.ARC'


指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
auto
ORA-00279: 更改 1936221 (在 01/09/2007 13:51:36 生成) 对于线程 1 是必需的
ORA-00289: 建议: C:\ARC\ORA9_11.ARC
ORA-00280: 更改 1936221 对于线程 1 是按序列 # 11 进行的
ORA-00278: 此恢复不再需要日志文件 'C:\ARC\ORA9_10.ARC'


ORA-00279: 更改 1936228 (在 01/09/2007 13:51:43 生成) 对于线程 1 是必需的
ORA-00289: 建议: C:\ARC\ORA9_12.ARC
ORA-00280: 更改 1936228 对于线程 1 是按序列 # 12 进行的
ORA-00278: 此恢复不再需要日志文件 'C:\ARC\ORA9_11.ARC'


ORA-00279: 更改 1936234 (在 01/09/2007 13:51:45 生成) 对于线程 1 是必需的
ORA-00289: 建议: C:\ARC\ORA9_13.ARC
ORA-00280: 更改 1936234 对于线程 1 是按序列 # 13 进行的
ORA-00278: 此恢复不再需要日志文件 'C:\ARC\ORA9_12.ARC'


已应用的日志。
完成介质恢复。
SQL> alter tablespace users online;

表空间已更改。

SQL> select * from scott.emp;

     EMPNO ENAME      JOB              MGR HIREDATE          SAL  COMM DEPTNO
---------- ---------- --------- ---------- ---------- ---------- ----- ------
      7369 SMITH      CLERK           7902 17-12月-80       1679             20
      7499 ALLEN      SALESMAN        7698 20-2月 -81       2478   300       30
      7521 WARD       SALESMAN        7698 22-2月 -81       2128   500       30
      7566 JONES      MANAGER         7839 02-4月 -81       3853             20
      7654 MARTIN     SALESMAN        7698 28-9月 -81       2128  1400       30
      7698 BLAKE      MANAGER         7839 01-5月 -81       3728             30
      7782 CLARK      MANAGER         7839 09-6月 -81       3328             10
      7839 KING       PRESIDENT            17-11月-81       5878             10
      7844 TURNER     SALESMAN        7698 08-9月 -81       2378     0       30
      7900 JAMES      CLERK           7698 03-12月-81       1828             30
      7902 FORD       ANALYST         7566 03-12月-81       3878             20
```

```
      7934 MILLER     CLERK           7782 23-1 月 -82       2178                10
```

已选择 12 行。

实验 106：索引表空间损坏的恢复

该实验的目的是理解什么是索引元数据的应用.索引和表分别存储的好处.
索引或 temp 表空间的恢复
因为这两类空间存放的数据都可以重建
1。定位该表空间内的索引
select index_name from dba_indexes where tablespace_name='INDX';
2。删除相应的索引
select 'drop index '||index_name||';' from dba_indexes where tablespace_name='INDX';
3。建立新的表空间
4。重新建立被删除的索引
5。如果为 temp，重新建立，修改默认表空间
SQL> SELECT TABLESPACE_NAME,FILE_NAME,BYTES/1024/1024 MB FROM DBA_DATA_FILES;

```
TABLESPACE_NAME FILE_NAME                                       MB
--------------- ------------------------------------ ----------
QQ              F:\ORACLE\ORADATA\ORA9\QQ.DBF                    1
XDB             F:\ORACLE\ORADATA\ORA9\XDB01.DBF            46.875
USERS           F:\ORACLE\ORADATA\ORA9\USERS01.DBF              25
TOOLS           F:\ORACLE\ORADATA\ORA9\TOOLS01.DBF          38.125
ODM             F:\ORACLE\ORADATA\ORA9\ODM01.DBF                 20
INDX            F:\ORACLE\ORADATA\ORA9\INDX.DBF                   2
EXAMPLE         F:\ORACLE\ORADATA\ORA9\EXAMPLE01.DBF        149.375
DRSYS           F:\ORACLE\ORADATA\ORA9\DRSYS01.DBF               20
CWMLITE         F:\ORACLE\ORADATA\ORA9\CWMLITE01.DBF             20
UNDOTBS1        F:\ORACLE\ORADATA\ORA9\UNDOTBS01.DBF         16.375
SYSTEM          F:\ORACLE\ORADATA\ORA9\SYSTEM01.DBF             420

11 rows selected.
```

查看当前数据库拥有的表空间
建立两个索引在 indx 表空间

```
create index scott.i1 on scott.emp(sal) tablespace indx;
create index scott.i2 on scott.emp(ename) tablespace indx;

SQL> select segment_name,segment_type,owner from dba_segments where tablespace_name='INDX';

SEGMENT_NAME         SEGMENT_TYPE OWNER
-------------------- ------------ ----------
I1                   INDEX        SCOTT
I2                   INDEX        SCOTT
```

破坏 indx 表空间的数据文件.

```
SQL> HOST COPY C:\BK\1.TXT F:\ORACLE\ORADATA\ORA9\INDX.DBF

SQL> --提取元数据
```

文件坏了也可以找到元数据,因为元数据存在于 system 表空间的字典中,而没有存在 indx 表空间.

```
SQL> SET LONG 10000
```

设定环境,是列为 long 类型的数据显示前 10000 个字符,默认值为 80 个字符.

```
SQL> select dbms_metadata.get_ddl('INDEX','I1','SCOTT') FROM DUAL;
```

```
DBMS_METADATA.GET_DDL('INDEX','I1','SCOTT')
--------------------------------------------------------------------------------

  CREATE INDEX "SCOTT"."I1" ON "SCOTT"."EMP" ("SAL")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "INDX"



SQL> select dbms_metadata.get_ddl('INDEX','I2','SCOTT') FROM DUAL;

DBMS_METADATA.GET_DDL('INDEX','I2','SCOTT')
--------------------------------------------------------------------------------

  CREATE INDEX "SCOTT"."I2" ON "SCOTT"."EMP" ("ENAME")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "INDX"



SQL> --删除损坏的索引表空间

SQL> alter database datafile 'F:\ORACLE\ORADATA\ORA9\INDX.DBF' offline ;

Database altered.

SQL> drop tablespace indx including contents ;

Tablespace dropped.

SQL> --建立新的表空间

SQL>  create tablespace indx datafile 'F:\ORACLE\ORADATA\ORA9\INDX.DBF' size 2m reuse;

Tablespace created.
```

运行刚才提取出的元数据,重新建立索引,记住一定要先提取元数据再删除损坏的表空间,因为表空间删除以后,元数据也被删除了.

```
SQL> CREATE INDEX "SCOTT"."I1" ON "SCOTT"."EMP" ("SAL")
  2  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  3  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  4  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  5  TABLESPACE "INDX"
  6  ;

Index created.

SQL> CREATE INDEX "SCOTT"."I2" ON "SCOTT"."EMP" ("ENAME")
  2  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  3  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  4  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  5  TABLESPACE "INDX"
  6  ;
```

Index created.

实验 107：临时表空间损坏的恢复

该实验的目的是理解什么是临时表空间,临时表空间损坏的修复.
临时表空间是用来排序的,临时存储数据,数据库启动的时候不检查临时表空间是否存在.
对临时表空间出现问题的修复就三句话.
建立新的,改为默认,删除老的.
临时表空间丢失,损坏,过小,都可以使用上面三句话来处理.

实验 108：无备份表空间损坏的恢复

该实验的目的是使用归档恢复建立表空间以来的数据.
无备份的表空间的恢复
1。该表空间建立以来的归档日志都存在
2。Alter database create datafile ‘old..’ as ‘new..’;
3.recover datafile ‘…..’;
4.alter tablespace ### online;
5.验证交易存在否

```
SQL> --建立新的表空间
SQL> select name from v$datafile;
该语句的命令是查看当前数据文件所在的路径
NAME
----------------------------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF
D:\ORACLE\ORADATA\ORA10\USERS01.DBF

SQL> create tablespace wb datafile 'D:\ORACLE\ORADATA\ORA10\wb.qq' size 3m;

表空间已创建。

SQL> alter system switch logfile;

系统已更改。

SQL> drop table scott.t1 purge;
drop table scott.t1 purge
                *
第 1 行出现错误:
ORA-00942: 表或视图不存在


SQL> create table scott.t1 tablespace wb as select * from scott.emp;

表已创建。
在新的表空间中建立实验表 t1
SQL> alter system switch logfile;

系统已更改。

SQL> update scott.t1 set sal=1000;
```

已更新 14 行。

SQL> commit;

提交完成。

SQL> alter system switch logfile;

系统已更改。

SQL> update scott.t1 set sal=2000;

已更新 14 行。

SQL> commit;

提交完成。

SQL> alter system switch logfile;

系统已更改。

SQL> update scott.t1 set sal=3000;

已更新 14 行。

SQL> commit;

提交完成。
T1 表建立以后做了三笔交易,我每次都切换日志的目的是造成时间的差,产生归档的日志,使实验和现实的情况更加的接近
SQL> select tablespace_name from dba_tables where table_name='T1' AND OWNER='SCOTT';

TABLESPACE_NAME

WB
验证 t1 表存在于新建立的表空间中
SQL> select name from v$datafile;

NAME

D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF
D:\ORACLE\ORADATA\ORA10\USERS01.DBF
D:\ORACLE\ORADATA\ORA10\WB.QQ

SQL> host copy c:\bk\hot.txt D:\ORACLE\ORADATA\ORA10\WB.QQ
随便找一个文件,替代当前的 wb.qq 数据文件,使之被人为的破坏.
SQL> select * from scott.t1;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-12 月-80	**3000**		20
7499	ALLEN	SALESMAN	7698	20-2 月 -81	3000	300	30
7521	WARD	SALESMAN	7698	22-2 月 -81	3000	500	30
7566	JONES	MANAGER	7839	02-4 月 -81	3000		20
7654	MARTIN	SALESMAN	7698	28-9 月 -81	3000	1400	30
7698	BLAKE	MANAGER	7839	01-5 月 -81	3000		30

```
      7782 CLARK      MANAGER         7839 09-6 月 -81           3000          10
      7788 SCOTT      ANALYST         7566 19-4 月 -87           3000          20
      7839 KING       PRESIDENT            17-11 月-81           3000          10
      7844 TURNER     SALESMAN        7698 08-9 月 -81           3000       0  30
      7876 ADAMS      CLERK           7788 23-5 月 -87           3000          20
      7900 JAMES      CLERK           7698 03-12 月-81           3000          30
      7902 FORD       ANALYST         7566 03-12 月-81           3000          20
      7934 MILLER     CLERK           7782 23-1 月 -82           3000          10

已选择 14 行。
```
文件已经破坏还能看到数据,看的是内存的信息
```
SQL> alter system checkpoint;

系统已更改。
```
强制产生存盘操作,刷洗了内存
```
SQL> select * from scott.t1;
select * from scott.t1
                    *
第 1 行出现错误:
ORA-00376: 此时无法读取文件 5
ORA-01110: 数据文件 5: 'D:\ORACLE\ORADATA\ORA10\WB.QQ'
```
这回看不到 t1 了.因为内存中不存在 t1 表了,物理文件也损坏了.
```
SQL>     alter     database     create     datafile     'D:\ORACLE\ORADATA\ORA10\WB.QQ'     as
'D:\ORACLE\ORADATA\ORA10\WB.dbf';

数据库已更改。
```
根据控制文件中记录的信息,建立一个空文件,大小和原来的文件相同,同时改了文件的名称
```
SQL> select name from V$datafile;

NAME
----------------------------------------
D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF
D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF
D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF
D:\ORACLE\ORADATA\ORA10\USERS01.DBF
D:\ORACLE\ORADATA\ORA10\WB.DBF

SQL> alter tablespace wb online;
alter tablespace wb online
*
第 1 行出现错误:
ORA-01113: 文件 5 需要介质恢复
ORA-01110: 数据文件 5: 'D:\ORACLE\ORADATA\ORA10\WB.DBF'


SQL> select * from V$recovery_log;

 THREAD#  SEQUENCE# TIME           ARCHIVE_NAME
-------- ---------- -------------- ----------------------------------------
       1         20 29-3 月 -06    C:\ARC\ARC00020_0586360856.001
       1         21 29-3 月 -06    C:\ARC\ARC00021_0586360856.001
```
查看恢复该文件要哪些日志文件.
```
SQL> select * from V$log;

 GROUP#  THREAD#  SEQUENCE#      BYTES    MEMBERS ARCHIV STATUS     FIRST_CHANGE# FIRST_TIME
------- -------- ---------- ---------- ---------- ------ ---------- ------------- --------------
      1        1         23    5242880          1 YES    INACTIVE          556623 29-3 月 -06
      2        1         24    5242880          1 NO     CURRENT           556631 29-3 月 -06
```

```
      3         1          22     5242880          1 YES     INACTIVE          556613 29-3 月 -06

SQL> recover datafile 5;
ORA-00279: 更改 556483 (在 03/29/2006 16:02:18 生成) 对于线程 1 是必需的
ORA-00289: 建议: C:\ARC\ARC00020_0586360856.001
ORA-00280: 更改 556483 (用于线程 1) 在序列 #20 中


指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
auto
ORA-00279: 更改 556511 (在 03/29/2006 16:02:34 生成) 对于线程 1 是必需的
ORA-00289: 建议: C:\ARC\ARC00021_0586360856.001
ORA-00280: 更改 556511 (用于线程 1) 在序列 #21 中
ORA-00278: 此恢复不再需要日志文件 'C:\ARC\ARC00020_0586360856.001'


已应用的日志。
完成介质恢复。
SQL> alter tablespace wb online;

表空间已更改。

SQL> select * from scott.t1;

     EMPNO ENAME      JOB              MGR HIREDATE              SAL  COMM DEPTNO
---------- ---------- --------- ---------- -------------- ---------- ----- ------
      7369 SMITH      CLERK           7902 17-12 月-80          3000              20
      7499 ALLEN      SALESMAN        7698 20-2 月 -81           3000   300      30
      7521 WARD       SALESMAN        7698 22-2 月 -81           3000   500      30
      7566 JONES      MANAGER         7839 02-4 月 -81           3000              20
      7654 MARTIN     SALESMAN        7698 28-9 月 -81           3000  1400      30
      7698 BLAKE      MANAGER         7839 01-5 月 -81           3000              30
      7782 CLARK      MANAGER         7839 09-6 月 -81           3000              10
      7788 SCOTT      ANALYST         7566 19-4 月 -87           3000              20
      7839 KING       PRESIDENT            17-11 月-81           3000              10
      7844 TURNER     SALESMAN        7698 08-9 月 -81           3000      0       30
      7876 ADAMS      CLERK           7788 23-5 月 -87           3000              20
      7900 JAMES      CLERK           7698 03-12 月-81           3000              30
      7902 FORD       ANALYST         7566 03-12 月-81           3000              20
      7934 MILLER     CLERK           7782 23-1 月 -82           3000              10

已选择 14 行。
该表空间不但恢复了,而且所有的交易都存在.
                                                    不完全恢复
有意的将数据库恢复到以前的某个时间点避免错误
日志丢失被迫做了不完全恢复

案例描述
有完整的备份
归档的数据库
在某一个时间点 drop table scott.emp purge;
Emp 表必须恢复
```

实验 109：日志挖掘

该实验的目的是查看日志内的语句.

日志挖掘

1。进入最高用户
2。指定挖掘队列
EXECUTE DBMS_LOGMNR.ADD_LOGFILE('D:\ORACLE\ORADATA\O10\REDO03.LOG');
3。开始挖掘
EXECUTE dbms_logmnr.start_logmnr(OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG);
4。查询结果
SELECT SCN,sql_redo FROM V$LOGMNR_CONTENTS WHERE upper(seg_name) = 'T1';
5。结束挖掘 EXECUTE DBMS_LOGMNR.END_LOGMNR;

实验 110：不完全恢复，删除表的恢复

该实验的目的是理解 scn 是顺序增加的.
案例描述:我们每天晚 20:00 做全备份,数据库为归档模式,上午 10:00 点重要的表被误删除,该表必须恢复.我们将含有该表的表空间单独恢复可以吗?不行,因为数据文件的独立恢复必须是应用全部的日志.如果只使用了部分,该表空间不能 online,因为数据文件的时间戳不一致.我们要想恢复被删除的表,理论上是将数据库后退,后退到 drop 发生之前.数据库的本质是不能将 scn 号后退的,10g 数据库使用了闪回数据库的特性,但有严格的前提条件.我们这里使用的办法是任何情况都可以使用的,没有前提条件的.
数据库恢复的基本法则:一.scn 都是由小向大增加,不能后退.二.控制文件要正确描述数据库的结构和行为.

1。通过日志挖掘找到失败的 scn 号,####
2。全备份数据库（再次恢复的基石）
3。停止数据库
4。恢复所有的 datafile,仅恢复 datafile(千万不要恢复其它）
5。Startup mount;
6.recover database until change ####;
7.alter database open resetlogs;
8.验证丢掉的表是否恢复。
9。全备份数据库（未来恢复的基石）

实验 111：不完全恢复，删除表空间的恢复

该实验的目的是理解恢复数据库的时候,控制文件一定要正确描述数据库的结构和行为.
案例描述
有完整的备份
归档的数据库
在某一个时间点
drop tablespace ts1 including contents and datafiles;
该表空间必须恢复
我们挖掘会找到很多 scn 中有 drop 操作,我们选择最小的那个,因为数据库是先删除该表空间内的表,最后再删除表空间的.如果你选择了 drop tablespace 那句话的 scn,将将恢复一个空的表空间,没有意义,所以要使用最小的 scn 号来恢复.
1。通过日志挖掘找到失败的 scn 号,####
2。全备份数据库（再次恢复的基石）
3。停止数据库
4。恢复所有的 datafile,和 controlfile(千万不要恢复日志）
5。Startup mount;
6.recover database until change #### using backup controlfile;
7.alter database open resetlogs;
8.验证丢掉的表是否恢复。
9。全备份数据库（未来恢复的基石）

实验 112：不完全恢复，当前日志损坏的恢复

该实验的目的是处理各种情况下的日志错误,如何使用带有下划的隐含参数.

日志文件丢失

组内某个成员丢失，但还有其它成员可以使用
非当前组丢失
当前的组丢失

组内某个成员丢失，但还有其它成员可以使用
1。通过报警日志文件找到丢失的成员
2。删除丢失的成员
3。重新建立丢失的成员

非当前组丢失
如果及时发现，clear 该组，或者删除，在建立
如果没有发现，将变为当前日志丢失

当前的组丢失
数据库崩溃，因为 lgwr 进程崩溃，导致数据库崩溃
1。全备份数据库（再次恢复的基石）
2。恢复所有的 datafile（千万不要恢复其它文件）
2。 Startup mount;
3。 recover database until cancel;
4。提供归档日志
5。数据库要丢失的日志时，键入 cancel
6. alter database open resetlogs;
7.全备份数据库（未来恢复的基石）
结果丢失了当前组所记录的交易，因为当前组没归档

案例描述
有完整的备份
归档的数据库
在某一个时间点磁盘崩溃
数据库必须恢复
某个归档日志丢失

1。全备份数据库（再次恢复的基石）
2。恢复所有的 datafile,和 controlfile(日志文件不要了)
3。 Startup mount;
4。 recover database until cancel using backup controlfile;
5。提供归档日志
6。数据库要丢失的日志时，键入 cancel
7。alter database open resetlogs;
8。全备份数据库（未来恢复的基石）

如果没有备份情况下当前日志组损坏将如何?

```
SQL> conn / as sysdba
Connected.
SQL> select * from v$log;

    GROUP#    THREAD#   SEQUENCE#      BYTES    MEMBERS ARCHIV STATUS     FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- ------ ---------- ------------- -----------
         1          1        256    5242880          1 YES    INACTIVE       3328619657 12-SEP-07
         2          1        257    5242880          1 YES    INACTIVE       3328619901 12-SEP-07
         3          1        258    5242880          1 NO     CURRENT        3328624139 14-SEP-07

SQL> select * from v$logfile;
```

```
 GROUP# STATUS     TYPE                          MEMBER                                  IS_REC
------- ---------- ----------------------------- --------------------------------------- ------
      3            ONLINE                        D:\ORACLE\ORADATA\ORA10\REDO03.LOG       NO
      2            ONLINE                        D:\ORACLE\ORADATA\ORA10\REDO02.LOG       NO
      1            ONLINE                        D:\ORACLE\ORADATA\ORA10\REDO01.LOG       NO

SQL> host copy c:\bk\1.txt D:\ORACLE\ORADATA\ORA10\REDO03.LOG
破坏当前的日志文件,再进行切换
SQL> alter system switch logfile;
alter system switch logfile
*
ERROR at line 1:
ORA-00316: log  of thread , type  in header is not log file
实例崩溃了.因为 lgwr 死了,它是核心进程,一个核心进程死亡实例就会崩溃
SQL> conn / as sysdba
Connected to an idle instance.
SQL> startup
ORACLE instance started.

Total System Global Area  167772160 bytes
Fixed Size                  1247900 bytes
Variable Size              75498852 bytes
Database Buffers           88080384 bytes
Redo Buffers                2945024 bytes
Database mounted.
ORA-00313: open failed for members of log group 3 of thread 1
ORA-00312: online log 3 thread 1: 'D:\ORACLE\ORADATA\ORA10\REDO03.LOG'
ORA-27046: file size is not a multiple of logical block size
OSD-04012: file size mismatch (OS 6374)
我们想启动数据库,但是失败了.因为我们现在的文件根本不是一个日志文件.

SQL>  alter system set _allow_resetlogs_corruption=true  scope=spfile;
 alter system set _allow_resetlogs_corruption=true  scope=spfile
                  *
ERROR at line 1:
ORA-00911: invalid character
修改参数失败了.

SQL>  alter system set "_allow_resetlogs_corruption"=true  scope=spfile;
加上双引号,修改成功
System altered.

SQL> shutdown abort;
ORACLE instance shut down.
SQL> startup mount;
ORACLE instance started.
重新启动实例使修改的参数生效
Total System Global Area  167772160 bytes
Fixed Size                  1247900 bytes
Variable Size              75498852 bytes
Database Buffers           88080384 bytes
Redo Buffers                2945024 bytes
Database mounted.
SQL> show parameter allow

NAME                                 TYPE                          VALUE
------------------------------------ ----------------------------- -----
_allow_resetlogs_corruption          boolean                       TRUE
```

```
SQL> alter database open resetlogs;
alter database open resetlogs
*
ERROR at line 1:
ORA-01139: RESETLOGS option only valid after an incomplete database recovery
```
我们想以 resetlogs 模式打开数据库,让数据库重新建立日志,但失败了.

我们做一个假恢复,欺骗数据库.走个形式,因为我们没有备份,不可能真恢复
```
SQL> recover database until cancel;
ORA-00279: change 3328624139 generated at 09/14/2007 20:00:38 needed for thread 1
ORA-00289: suggestion : C:\ARC\1_258_621191202.ARC
ORA-00280: change 3328624139 for thread 1 is in sequence #258


Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
cancel
ORA-01547: warning: RECOVER succeeded but OPEN RESETLOGS would get error below
ORA-01194: file 1 needs more recovery to be consistent
ORA-01110: data file 1: 'D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF'


ORA-01112: media recovery not started
```
数据库相信了,可以了,但打开的时候又崩溃了.
```
SQL> alter database open resetlogs;
alter database open resetlogs
*
ERROR at line 1:
ORA-01092: ORACLE instance terminated. Disconnection forced


SQL> conn / as sysdba
Connected to an idle instance.
SQL> startup
ORACLE instance started.

Total System Global Area  167772160 bytes
Fixed Size                  1247900 bytes
Variable Size              75498852 bytes
Database Buffers           88080384 bytes
Redo Buffers                2945024 bytes
Database mounted.
Database opened.
```
数据库好了!
```
SQL> select * from scott.emp;

     EMPNO ENAME      JOB                 MGR HIREDATE            SAL  COMM DEPTNO
---------- ---------- ------------------ ---------- ------------ ---------- ----- ------
      7369 SMITH      CLERK                7902 17-DEC-80           913             20
      7499 ALLEN      SALESMAN             7698 20-FEB-81          1712   300       30
      7521 WARD       SALESMAN             7698 22-FEB-81          1362   500       30
      7566 JONES      MANAGER              7839 02-APR-81          3087             20
      7654 MARTIN     SALESMAN             7698 28-SEP-81          1362  1400       30
      7698 BLAKE      MANAGER              7839 01-MAY-81          2962             30
      7782 CLARK      MANAGER              7839 09-JUN-81          2562             10
      7788 SCOTT      ANALYST              7566 19-APR-87          3112             20
      7839 KING       PRESIDENT                 17-NOV-81          5112             10
      7844 TURNER     SALESMAN             7698 08-SEP-81          1612     0       30
      7876 ADAMS      CLERK                7788 23-MAY-87          1212             20
```

7900 JAMES	CLERK	7698 03-DEC-81	1062	30
7902 FORD	ANALYST	7566 03-DEC-81	3112	20
7934 MILLER	CLERK	7782 23-JAN-82	1412	10

14 rows selected.

实验 113：不完全恢复，resetlogs 后的再次恢复

该实验的目的是理解控制文件决定了恢复的一切操作
案例描述
有完整的备份
归档的数据库
在某一个时间点 drop table scott.emp purge;
Emp 表必须恢复
已经不完全恢复成功，但没有备份
继续交易
磁盘崩溃
数据库必须恢复所有交易

1。通过 bdump 下的报警日志找到上次恢复的 scn 号,####
2。全备份数据库（再次恢复的基石）
3。停止数据库
4。恢复所有的 datafile,和 controlfile(千万不要恢复其它）
5。Startup mount;
6.recover database until change #### using backup controlfile;
7.将 2 步骤中备份的控制文件恢复
8. Recover database;
9.alter database open ;
10.验证丢掉的表是否恢复,以后的交易是否存在。
11。全备份数据库（未来恢复的基石）

实验 114：表空间的传送

该实验的目的是理解什么是表空间对象的定义,如何使用 sys 用户进行逻辑的备份
表空间传送(TSPITR)
1.使用备份建立新的数据库
2。将新的数据库恢复到失败的时间点。
3。假如你想恢复 users 表空间。
4。Alter tablespace users read only;
5.exp ‘sys/sys as sysdba’ tablespaces=users TRANSPORT_TABLESPACE=y file=c:\bk\user.dmp
6.在生产数据库将 users 表空间删除
7。将 5 的 user.dmp 和数据文件复制到生产数据库
8。 imp ‘sys/sys as sysdba’ tablespaces=users TRANSPORT_TABLESPACE=y file=c:\bk\user.dmp
Datafiles=‘c:\bk\users01.dbf’
9.生产数据库 alter tablespace users read write;

知识点
数据库的 scn 只能增长，不能后退
如果想后退，请从更低的 scn 增长到想到的 scn
在恢复的时候，控制文件一定要正确描述数据库的状态
一定要以 resetlogs 方式 open 数据库
Resetlogs 后一定要备份

实验 115：整个数据库的闪回

该实验的目的是掌握 10g 的新特性,有一定的作用,工作中华而不实,有很大的局限性.

使用快速恢复区进行反算

下列情况不能闪回
1。控制文件重新建立，或者使用老的控制文件
2。表空间被 drop 的情况
3。数据库文件被回缩了
4。数据库 resetlogs 以后

```
--设定恢复文件的目录
alter system set db_recovery_file_dest='c:\bk' scope=spfile;
alter system set DB_RECOVERY_FILE_DEST_SIZE=300m scope=spfile;
show parameter DB_RECOVERY_FILE_DEST

--设置恢复的目标时间，大概的值，单位为分钟
show parameter DB_FLASHBACK_RETENTION_TARGET

--启动数据库到 mount 下，如果为 rac,请先改为独享模式。
shutdown immediate;
startup mount;
select name from V$bgprocess where paddr<>'00' order by 1;

ALTER DATABASE FLASHBACK ON;
select name from V$bgprocess where paddr<>'00' order by 1;
--多了 RVWR 后台进程
SELECT flashback_on FROM v$database;

alter database open;


-----------
SELECT estimated_flashback_size,flashback_size FROM V$FLASHBACK_DATABASE_LOG;

SELECT oldest_flashback_scn,oldest_flashback_time FROM V$FLASHBACK_DATABASE_LOG;

SELECT * FROM V$FLASHBACK_DATABASE_STAT;
select dbms_flashback.get_system_change_number() from dual;

SELECT name, space_limit AS quota,space_used AS used,space_reclaimable AS reclaimable,
number_of_files AS files FROM v$recovery_file_dest ;

--闪回数据库必须在 mount 下，必须 resetlogs 方式 open 数据库
FLASHBACK DATABASE TO SCN  23565;
FLASHBACK DATABASE TO SEQUENCE=223 THREAD=1;
FLASHBACK DATABASE TO TIMESTAMP(SYSDATE-1/24);
FLASHBACK DATABASE TO TIME =TO_DATE('2004-05-27 16:00:00', 'YYYY-MM-DD HH24:MI:SS');

--部分表空间排除在闪回之列，在 offline 下修改.默认都处于闪回状态
ALTER TABLESPACE <ts_name> FLASHBACK off;
SELECT name, flashback_on FROM v$tablespace;
```

Rman 备份和恢复

实验 116：rman 的连接，report 和 list 命令

该实验的目的是初步认识 rman.运行简单的 rman 命令
Rman 的使用
连接到 rman 必须以最高用户
在操作系统下运行
rman target sys/sys nocatalog
report schema;
命令以分号结束
Report 告诉我们应该做什么
report need backup days 3;
告诉我三天以上没有备份的文件

List，告诉我门已经有什么了
list copy of datafile 9;
list backup of datafile 9;
这是一对命令,一个告诉我们应该做什么,一个告诉我们已经存在了什么.

```
RMAN> connect target /

connected to target database: ORA10 (DBID=568967312)

RMAN> report schema;

using target database control file instead of recovery catalog
Report of database schema

List of Permanent Datafiles
===========================
File Size(MB) Tablespace           RB segs Datafile Name
---- -------- -------------------- ------- ------------------------
1    490      SYSTEM               ***     D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF
2    60       UNDOTBS1             ***     D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF
3    290      SYSAUX               ***     D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF
4    26       USERS                ***     D:\ORACLE\ORADATA\ORA10\USERS01.DBF
5    0        TL                   ***     D:\ORACLE\ORADATA\ORA10\TL.DBF
6    3        T2M                  ***     D:\ORACLE\ORADATA\ORA10\T2M.DBF
8    2        BIGTS                ***     D:\ORACLE\ORADATA\ORA10\BIGTS.BIG

List of Temporary Files
=======================
File Size(MB) Tablespace           Maxsize(MB) Tempfile Name
---- -------- -------------------- ----------- --------------------
1    20       TEMP1                32767       D:\ORACLE\ORADATA\ORA10\TEMP1.DBF

RMAN> report need backup days 3;

Report of files whose recovery needs more than 3 days of archived logs
File Days  Name
---- ----- -----------------------------------------------
1    756   D:\ORACLE\ORADATA\ORA10\SYSTEM01.DBF
2    756   D:\ORACLE\ORADATA\ORA10\UNDOTBS01.DBF
3    756   D:\ORACLE\ORADATA\ORA10\SYSAUX01.DBF
5    104   D:\ORACLE\ORADATA\ORA10\TL.DBF
6    4     D:\ORACLE\ORADATA\ORA10\T2M.DBF
```

8 80 D:\ORACLE\ORADATA\ORA10\BIGTS.BIG
这里没有显示 4 号文件,因为我们三天之内已经备份过该文件了,所以不显示了.rman 不是神仙,它从哪里获得的数据信息呢?控制文件中含有 rman 的备份信息,所以使用 rman 进行备份和恢复时数据库一定要在 mount 以上的状态,如果你重新建立了控制文件,当然备份的信息就完全丢失了.

实验 117：rman 的 copy 命令

该实验的目的是使用 rman 进行 copy 的操作.
Copy 命令备份文件的所有块,包含崭新的没有装过数据的块.
copy datafile 'D:\ORACLE\ORADATA\O10\USERS01.DBF' to 'c:\bk\u1.cp';
List copy of datafile 4;
等同于热备份,所以 rman 使用 copy 命令产生的备份可以使用手工来恢复.

实验 118：rman 的 backup 命令

该实验的目的是使用 rman 进行 backup 的操作.
Backup
backup datafile 4 format 'c:\bk\u1.%s';
List backup of datafile 4;
只备份使用过的数据块,**外加一个文件头**,这个头里描述了该文件的信息.所以使用 backup 命令备份的数据只有使用 backup 来恢复.

Set
逻辑的，由备份片组成
%s 代表备份**集**
Piece
物理的文件，大小可以限制，不指明每个集合有一个片
%p 代表备份**片**
%d 代表数据**库**的名称
%u 代表**唯一**标识,这里是区分大小写的.

RMAN> **backup** datafile 4 **format** 'c:\bk\f4_%u';

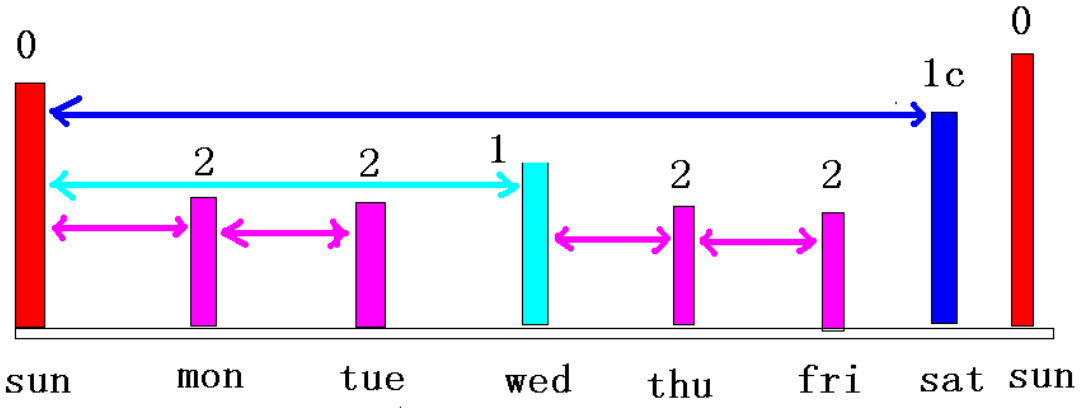
```
Starting backup at 25-SEP-07
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=146 devtype=DISK
channel ORA_DISK_1: starting full datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel ORA_DISK_1: starting piece 1 at 25-SEP-07
channel ORA_DISK_1: finished piece 1 at 25-SEP-07
piece handle=C:\BK\F4_01ISR753 tag=TAG20070925T153250 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:12
Finished backup at 25-SEP-07
```

实验 119：rman 的 backup 备份增量级别

该实验的目的是使用 rman 进行 backup 的操作进行增量备份,理解什么是增量级别.

Backup 有增量,copy 没有增量.
0，所有的使用的数据块，是基石
1—4 增量级别,备份<=n 以来的变化

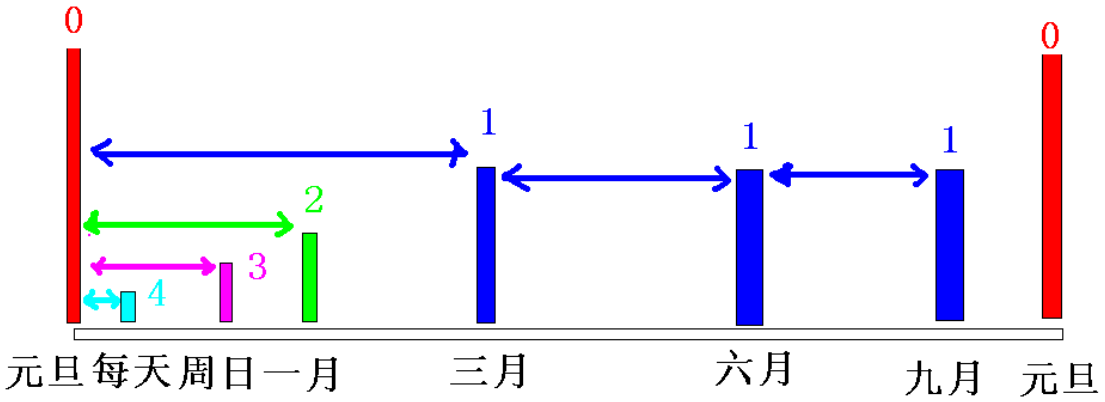
1c---4c 累积增量,备份<=n-1 以来的变化
通过增量的级别来指定完备的备份策略



请看这个案例,每周日晚做 0 级备份,就是备份所有使用过的数据块.

周一做 2 级增量,备份小于等于 2 以来备份后发生变化的块,前面有个 0 级备份.所以备份当天的变化.

周二做 2 级增量,备份小于等于 2 以来备份后发生变化的块,前面有个 2 级备份.所以备份当天的变化.
周三做 1 级增量,备份小于等于 1 以来备份后发生变化的块,前面有个 0 级备份.所以备份三天的变化.
周四做 2 级增量,备份小于等于 2 以来备份后发生变化的块,前面有个 1 级备份.所以备份当天的变化.
周五做 2 级增量,备份小于等于 2 以来备份后发生变化的块,前面有个 2 级备份.所以备份当天的变化.
周六做 1c 级增量,备份小于等于 1-1=0 以来备份后发生变化的块,前面有个零级备份.所以备份六天的变化.
周日做 0 级增量, 备份所有使用过的数据块.



请看这个案例,每年元旦做 0 级备份,就是备份所有使用过的数据块.
每天做 4 级增量,备份小于等于 4 以来备份后发生变化的块,任何级都小于等于 4.所以备份当天的变化.
每周做 3 级增量,备份小于等于 3 以来备份后发生变化的块,前面有个 3 级备份.所以备份一周的变化.
每月做 2 级增量,备份小于等于 2 以来备份后发生变化的块,前面有个 2 级备份.所以备份当月的变化.
每季度做 1 级增量,备份小于等于 1 以来备份后发生变化的块,前面有个 1 级备份.所以备份三个月的变化.
每年元旦做 0 级备份,就是备份所有使用过的数据块.
增量备份

```
RMAN>  backup incremental level 0 datafile 4 format 'c:\bk\%d_%s_%p';

Starting backup at 25-SEP-07
using channel ORA_DISK_1
channel ORA_DISK_1: starting incremental level 0 datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel ORA_DISK_1: starting piece 1 at 25-SEP-07
channel ORA_DISK_1: finished piece 1 at 25-SEP-07
piece handle=C:\BK\ORA10_2_1 tag=TAG20070925T161504 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:08
```

```
Finished backup at 25-SEP-07
```

建立表,插入数据.

```
sql>create table scott.t1 as select * from scott.emp;
insert into scott.t1 select * from t1;
commit;
```

使备份以来数据块发生改变

```
RMAN> backup incremental level 1 datafile 4 format 'c:\bk\%d_%s_%p';

Starting backup at 25-SEP-07
using channel ORA_DISK_1
channel ORA_DISK_1: starting incremental level 1 datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel ORA_DISK_1: starting piece 1 at 25-SEP-07
channel ORA_DISK_1: finished piece 1 at 25-SEP-07
piece handle=C:\BK\ORA10_3_1 tag=TAG20070925T161904 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:03
Finished backup at 25-SEP-07
RMAN> backup cumulative incremental level 1 datafile 4 format 'c:\bk\%d_%s_%p';

Starting backup at 25-SEP-07
using channel ORA_DISK_1
channel ORA_DISK_1: starting incremental level 1 datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel ORA_DISK_1: starting piece 1 at 25-SEP-07
channel ORA_DISK_1: finished piece 1 at 25-SEP-07
piece handle=C:\BK\ORA10_4_1 tag=TAG20070925T162254 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:03
Finished backup at 25-SEP-07

RMAN> backup  incremental level 1 cumulative datafile  4 format 'c:\bk\%d_%s_%p';

Starting backup at 25-SEP-07
using channel ORA_DISK_1
channel ORA_DISK_1: starting incremental level 1 datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel ORA_DISK_1: starting piece 1 at 25-SEP-07
channel ORA_DISK_1: finished piece 1 at 25-SEP-07
piece handle=C:\BK\ORA10_5_1 tag=TAG20070925T162832 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:03
Finished backup at 25-SEP-07

RMAN> backup incremental level 2 datafile 4 format 'c:\bk\%d_%s_%p';

Starting backup at 25-SEP-07
using channel ORA_DISK_1
channel ORA_DISK_1: starting incremental level 2 datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel ORA_DISK_1: starting piece 1 at 25-SEP-07
channel ORA_DISK_1: finished piece 1 at 25-SEP-07
piece handle=C:\BK\ORA10_7_1 tag=TAG20070925T192410 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:18
Finished backup at 25-SEP-07
```

```
RMAN> backup incremental level 0 datafile 4 format 'c:\bk\%d_%s_%p';

Starting backup at 25-SEP-07
using channel ORA_DISK_1
channel ORA_DISK_1: starting incremental level 0 datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel ORA_DISK_1: starting piece 1 at 25-SEP-07
channel ORA_DISK_1: finished piece 1 at 25-SEP-07
piece handle=C:\BK\ORA10_8_1 tag=TAG20070925T192903 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:17
Finished backup at 25-SEP-07

RMAN> list backup of datafile 4;

List of Backup Sets
===================

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
2       Incr 0  25.51M     DISK        00:00:05     25-SEP-07
        Piece Name: C:\BK\ORA10_2_1

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
3       Incr 1  296.00K    DISK        00:00:02     25-SEP-07
        Piece Name: C:\BK\ORA10_3_1

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
4       Incr 1  296.00K    DISK        00:00:03     25-SEP-07
        Piece Name: C:\BK\ORA10_4_1
4 我做的是 1c,一级累积增量
BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
5       Incr 1  296.00K    DISK        00:00:03     25-SEP-07
        Piece Name: C:\BK\ORA10_5_1
5 我做的是 1c,一级累积增量

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
7       Incr 2  32.00K     DISK        00:00:10     25-SEP-07
        Piece Name: C:\BK\ORA10_7_1

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
8       Incr 0  25.51M     DISK        00:00:11     25-SEP-07
        Piece Name: C:\BK\ORA10_8_1
```

我们看到 1 和 1c 在列表中我们没有区分,实际数据库也不区分,在备份时的语法不同. 备份集不区分累积增量和普通增量. 但从大小可以看到 3,4,5 的备份集大小相同,如果不是累积将较小.

实验 120：rman 的 backup 备份片大小的限制

该实验的目的是使用 rman 进行 backup 的操作. 限制每个备份片的大小.

限制通道的大小,什么是通道呢?进程!告诉数据库使用一个进程 d1 做备份,我们通过限制进程的写信息来限制每个备份片的大小.备份集是逻辑的,有多个备份片组成.备份片是物理的.没有明确说明默认情况下为一个备份集对应一个物理的备份片,一个备份集中可以包含多个备份的数据文件.你可以理解为备份集是压缩包的名称,一个压缩包里含有多个被压缩的文件,我们又将压缩包分解为多个小压缩文件,当我们解压缩的时候需要所有的小压缩包才能解压缩文件.我们一般限制备份片大小的目的是使得备份片可以存放在一个磁带上.下面的命令是限制每个备份片大小为 10m,我们上面的实验看到正常备份有 25 m,所以会被切割为三个备份片,前两个每个为 10m,最后一个为 5m.

```
RUN {
ALLOCATE CHANNEL d1 TYPE disk;
set limit channel d1 kbytes=10000;
backup datafile 4 FORMAT 'c:\bk\%s_%p' ;}

RMAN> RUN {
2> ALLOCATE CHANNEL d1 TYPE disk;
3> set limit channel d1 kbytes=10000;
4> backup datafile 4 FORMAT 'c:\bk\%s_%p' ;}

using target database control file instead of recovery catalog
allocated channel: d1
channel d1: sid=146 devtype=DISK


Starting backup at 26-SEP-07
channel d1: starting full datafile backupset
channel d1: specifying datafile(s) in backupset
input datafile fno=00004 name=D:\ORACLE\ORADATA\ORA10\USERS01.DBF
channel d1: starting piece 1 at 26-SEP-07
channel d1: finished piece 1 at 26-SEP-07
piece handle=C:\BK\9_1 tag=TAG20070926T100615 comment=NONE
channel d1: starting piece 2 at 26-SEP-07
channel d1: finished piece 2 at 26-SEP-07
piece handle=C:\BK\9_2 tag=TAG20070926T100615 comment=NONE
channel d1: starting piece 3 at 26-SEP-07
channel d1: finished piece 3 at 26-SEP-07
piece handle=C:\BK\9_3 tag=TAG20070926T100615 comment=NONE
channel d1: backup set complete, elapsed time: 00:00:14
Finished backup at 26-SEP-07
released channel: d1

RMAN>list backup of datafile 4;

BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
9       Full    25.55M     DISK        00:00:12     26-SEP-07
  List of Datafiles in backup set 9
  File LV Type Ckp SCN    Ckp Time  Name
  ---- -- ---- ---------- --------- ----
  4       Full 3328858092 26-SEP-07 D:\ORACLE\ORADATA\ORA10\USERS01.DBF

  Backup Set Copy #1 of backup set 9
  Device Type Elapsed Time Completion Time Compressed Tag
  ----------- ------------ --------------- ---------- ---
  DISK        00:00:12     26-SEP-07       NO         TAG20070926T100615

    List of Backup Pieces for backup set 9 Copy #1
    BP Key  Pc# Status      Piece Name
    ------- --- ----------- ----------
    9       1   AVAILABLE   C:\BK\9_1
    10      2   AVAILABLE   C:\BK\9_2
```

```
11       3   AVAILABLE   C:\BK\9_3
```

第二种方法是设置系统的默认配置
CONFIGURE CHANNEL DEVICE TYPE disk MAXPIECESIZE 1G;
这句话说明如果不说大小,就是每个备份片的大小为 1g.

实验 121：rman 的 backup 备份数据文件

该实验的目的是使用 rman 进行 backup 的操作.备份全数据库

我们使用 report schema;命令来查看有哪些文件要备份.
Backup database 就会备份所有的数据文件,我们也可以指定多少个数据文件放在一个备份集中.如果不指定会将所有的文件放入到一个备份集中.这样如果数据库很大就不太好.

```
backup database incremental level 0  format 'c:\bk\%d_%s_%p' filesperset 3;
```

实验 122：rman 的 backup 备份控制文件

该实验的目的是使用 rman 进行 backup 的操作.备份控制文件

备份控制文件

```
copy  current controlfile to 'd:\bk\c1.ctl';
backup  current controlfile format 'd:\bk\c1.%s';

list copy of controlfile;
list backup of controlfile;
```

实验 123：rman 的 backup 备份归档日志文件

该实验的目的是使用 rman 进行 backup 的操作.备份归档日志文件.

备份归档日志文件

```
list archivelog all;
list copy of archivelog all;
list backup of archivelog all;
list copy of archivelog sequence between 264 and 265 thread=1 ;

copy archivelog 'F:\oracle\oradata\zl9\arch\ARC264.LOG' to 'd:\bk\a.cp';
BACKUP ARCHIVELOG ALL DELETE INPUT  format 'd:\bk\arc%s.bk';
backup archivelog sequence between 264 and 265 thread=1 format 'd:\bk\arc%s.bk';
backup  archivelog  sequence  between  50  and  52  thread=1  like  '%0586360856%'  format
'c:\bk\arc%s.bk';

BACKUP ARCHIVELOG ALL DELETE INPUT  format 'd:\bk\arc%s.bk';
```

如果上面的语句有问题，请运行下面语句来标定控制文件中归档日志的状态。

```
change archivelog all crosscheck;
delete archivelog all;

select SEQUENCE#,APPLIED,DELETED,STATUS,BACKUP_COUNT from v$archived_log
```

实验 124：rman 的 backup 备份二进制参数文件

该实验的目的是使用 rman 进行 backup 的操作.备份二进制参数文件.

备份二进制参数文件
```
backup spfile format 'd:\bk\spfile.%s';
list backup of spfile;
```

实验 125：rman 的恢复目录的配置

该实验的目的是使用 rman 的 catalog 来存储备份的信息.

```
1.create tablespace rman datafile '.......\rman.dbf' size 20m;
2.create user Rman identified by Rman default tablespace rman;
3.grant recovery_catalog_owner, connect, resource to Rman;
4.DOS>  RMAN CATALOG RMAN/RMAN
5.RMAN>create catalog ;exit;
6.dos>RMAN TARGET SYS/SYS@3 catalog rman/rman
7.rman>register database;

CREATE script b0 {
backup
incremental level 0
format 'D:\bk\%d_%s_%p'
filesperset 3
(database include current controlfile);
sql "alter system archive log current";
BACKUP ARCHIVELOG ALL DELETE INPUT  format 'd:\bk\arc%s.bk';
}
```

执行备份
```
run {execute script b0;}
run {execute script b1;}
run {execute script b2;}
```

数据文件的恢复
```
Resotre datafile 4;
Recover datafile 4;
```

实验 126：rman 的数据文件的恢复

该实验的目的是使用 rman 进行数据文件的操作.原理是和手工处理是一样的,换了个工具而已.

将文件恢复到其它目录
```
RUN{
SET NEWNAME FOR datafile 'D:\ORACLE\ORADATA\USER63\TOOLS01.DBF' to 'e:\bk\TOOLS01.DBF';
RESTORE (tablespace tools);
SWITCH datafile 8;
RECOVER TABLESPACE tools;
SQL "alter tablespace tools online";
}
```

数据库的完全恢复
```
Restore database;
```

```
Recover database;
```

实验 127：rman 的数据块完全恢复

该实验的目的是使用 rman 的备份进行数据库块级别的操作.

我对数据块的损坏可以使用 rman 来恢复,这是我手工备份所办不到的.我们手工备份的恢复必须要恢复整个数据文件来处理坏的数据块.当我们恢复整个文件的时候要有该文件备份以来的所有完整的归档日志文件,如果中有归档断档就不能恢复.而基于数据块的 rman 恢复则有可能恢复,如果你丢失的归档中要没有关于这个数据块的操作,就可以恢复.而且基于数据库块的恢复要更少的 io.速度更快.

```
CONN / AS SYSDBA
--建立实验的表空间 hg
drop tablespace hg including contents and datafiles;
CREATE TABLESPACE HG DATAFILE 'F:\ORACLE\ORADATA\ORA9\HG.DBF'  SIZE 128K reuse;
--建立表，装满该表空间
DROP TABLE SCOTT.T1 ;
CREATE TABLE SCOTT.T1 TABLESPACE HG AS SELECT * FROM SCOTT.EMP;
INSERT INTO SCOTT.T1 SELECT * FROM SCOTT.T1;
/
/
/
/
/
INSERT INTO SCOTT.T1 SELECT * FROM SCOTT.T1 where rownum<100;
/
--装满数据

COMMIT;
--rman 进行备份
HOST RMAN TARGET /
BACKUP DATAFILE 'F:\ORACLE\ORADATA\ORA9\HG.DBF' FORMAT 'D:\BK\%u';
EXIT
------回到 sqlplus--------------------

--停止数据库
shutdown immediate;

--二进制编辑器编辑该数据文件，破坏几个行的数据.
ULTRAEDIT  F:\ORACLE\ORADATA\ORA9\HG.DBF  修改一个数据库块，靠后的数据块。

--启动数据库
startup

--验证有坏数据块
select count(*) from scott.t1;
select * from scott.t1;
ORA-01578: ORACLE data block corrupted (file # 12, block # 12)
ORA-01110: data file 12: 'F:\ORACLE\ORADATA\ORA9\HG.DBF'

select * from V$DATABASE_BLOCK_CORRUPTION ;

--rman 进行块级别的恢复
HOST RMAN TARGET /
BLOCKRECOVER DATAFILE 12 BLOCK 12;
exit
------回到 sqlplus--------------------
```

```
select count(*) from scott.t1;
select * from scott.t1;
```

实验 128：rman 的数据库不完全恢复

该实验的目的是使用 rman 进行不完全恢复,原理相同,使用 rman 来操作.

```
数据库的不完全恢复
数据库得处于 mount 状态
run {
 allocate channel c1 type DISK;
 allocate channel c2 type DISK;
 set until SCN = ****** ;
 restore database;
 recover database;
alter database open resetlogs; }
```

实验 129：rman 的数据库副本管理

该实验的目的是理解控制文件记录的内容,极其 resetlogs 的影响.

```
恢复数据库到改变副本以前的数据库备份点
--列出共有多少副本
list incarnation of database;
reset database;    --重置副本
--启动到 NOMOUNT 状态
STARTUP FORCE NOMOUNT;
--重置到指定的数据库副本
RESET DATABASE TO INCARNATION ##;
RESTORE CONTROLFILE;
STARTUP FORCE MOUNT;
RESTORE DATABASE UNTIL SCN *****;
RECOVER DATABASE UNTIL SCN *****;
```

实验 130：rman 的备份管理

该实验的目的是管理控制文件的记录和实际存在的备份的物理联系.

```
其它文件恢复
RESTORE CONTROLFILE;
restore archivelog sequence between 264 and 265 thread=1 ;

delete force copy;

CROSSCHECK BACKUP;
CROSSCHECK COPY;

DELETE  EXPIRED  archivelog all;
DELETE  EXPIRED backup;

catalog datafilecopy 'd:\bk\users01.dbf';
```

备份策略的选择

数据量
归档否
是否是容易加载，而比恢复更块
是否有存储软件
是否为裸设备

数据量
如果数据量小，每天全备份
如果数据量大，要有 rman 的备份策略

归档否
非归档数据库：只有冷备份
归档：既可以冷备份，也可以热备份

是否是容易加载，而比恢复更块
如果数据是每天统一灌入，以后就是查询
可以每天冷备份，失败后先将数据库恢复到备份点，再重新灌入

是否有存储软件
如果有其它存储软件
最好归档，rman ,增量

是否为裸设备
如果是裸设备
Rman
或逻辑备份
还有 ocopy
ocopy from_file [to_file [a | size_1 [size_n]]]
ocopy -b from_file to_drive
ocopy -r from_drive to_dir

第六部分数据库的优化

优化数据库
实例的优化
数据库的优化
SQL 的优化

优化数据库的步骤
1。采集数据库的信息
2。修改数据库的配置
3。再次采集

采集数据

平面文件
bdump
Udump
Cdump

查找当前会话的进程代码

```
 select spid from v$process
 where addr=(select paddr from v$session where sid=(select distinct sid from v$mystat));
```

查找当前会话的跟踪文件

```
select p.value||'\'||i.instance_name||'_ora_'||spid||'.trc' as "trace_file_name"
from v$parameter p ,v$process pro, v$session s,
 (select sid from v$mystat where rownum=1) m,
 v$instance i
where lower(p.name)='user_dump_dest'
and pro.addr=s.paddr
and m.sid=s.sid;
```

数据字典
V$sysstat
V$system_event
V$session_wait

实验 131：优化工具 utlbstat/utlestat 的使用

该实验的目的是使用数据库提供的脚本来采集我们的优化信息.

比较两个时间点的统计值

收集一段的数据库统计信息
@%oracle_home%\rdbms\admin\utlbstat.sql
处理交易
。。。
@%oracle_home%\rdbms\admin\utlestat.sql

实验 132：优化工具 spreport 的使用

该实验的目的是使用数据库提供的脚本采集更加丰富的信息.
自动收集统计信息

```
@%ORACLE_HOME%\rdbms\admin\spcreate.sql

conn perfstat/oracle
statspack.snap

@%ORACLE_HOME%\rdbms\admin\spreport.sql
@%ORACLE_HOME%\rdbms\admin\spauto.sql
@%ORACLE_HOME%\rdbms\admin\spdrop.sql
```

实验 133：系统包 dbms_job 维护作业

该实验的目的是维护 job.
作业有**四要素**:
作业号:系统自动发生的
做什么:我们来指定,以分号分隔语句,可以是存储过程.
什么时间开始做:日期类型
作业的**间隔时间**:字符型
下面留一个简单得到作业,体现了上面的四要素.

```
VARIABLE jobno NUMBER
BEGIN
DBMS_JOB.SUBMIT(:jobno,
'update scott.emp set sal=sal+1;',
SYSDATE ,
'SYSDATE + 1/24/60');
COMMIT;
END;
/

col error clear
select * from v$bgprocess where paddr<>'00';
--CJQ0  (Job Queue Coordinator) 作业队列调度后台进程

--同时最大的作业数
show parameter JOB_QUEUE_PROCESSES
ALTER SYSTEM set job_queue_processes=20;
show parameter JOB_QUEUE_PROCESSES

--以老大 sys 留作业
--提交作业，作业不能调用作业，作业描述中的单引要用双单引
VARIABLE jobno NUMBER
BEGIN
DBMS_JOB.SUBMIT(:jobno,
'update scott.emp set sal=sal+1;',
SYSDATE , 'SYSDATE + 1/24/60');
COMMIT;
END;
/

--验证作业
PRINT jobno
SELECT * from dba_jobs;
```

```
select JOB,WHAT,LAST_DATE,NEXT_DATE from dba_jobs;
select job,LAST_DATE,LAST_SEC,NEXT_DATE, NEXT_SEC,INTERVAL, BROKEN,FAILURES from dba_jobs;

--作业号由 SYS.JOBSEQ 序列产生
select * from dba_sequences where SEQUENCE_NAME='JOBSEQ';

--查看作业进程
select * from v$process where PROGRAM like 'ORACLE.EXE (J%';

select * from DBA_JOBS_RUNNING;

--修改作业
--------------------------------------------------------------------------------------------
--调度时间
select JOB,WHAT from dba_jobs;
select job,LAST_DATE,LAST_SEC,NEXT_DATE, NEXT_SEC,INTERVAL,FAILURES from dba_jobs;
BEGIN
DBMS_JOB.CHANGE(24, NULL, NULL, 'SYSDATE+3');
END;
/
--修改下次启用时间
BEGIN
DBMS_JOB.NEXT_DATE(24, SYSDATE + 1/24/60);
END;
/
--作业运行后不再启用
BEGIN
DBMS_JOB.INTERVAL(24, 'NULL');
END;
/
--作业的内容
BEGIN
DBMS_JOB.WHAT(24,'update scott.emp set sal=sal+1;update scott.emp set sal=sal+5;');
end;
/
--------------------------------------------------------------------------------------------
--中断作业
BEGIN
DBMS_JOB.BROKEN(27, TRUE);
END;
/
select job,LAST_DATE,LAST_SEC,NEXT_DATE, NEXT_SEC,INTERVAL, BROKEN,FAILURES from dba_jobs;

--启用作业
BEGIN
DBMS_JOB.BROKEN(24, FALSE, NEXT_DAY(SYSDATE, 'MONDAY'));
END;
/
--------------------------------------------------------------------------------------------
select job,LAST_DATE,LAST_SEC,NEXT_DATE, NEXT_SEC,INTERVAL, BROKEN,FAILURES from dba_jobs;
--运行作业
BEGIN
DBMS_JOB.RUN(28);
END;
/

--取消作业
select JOB,WHAT from dba_jobs;
execute dbms_job.remove(27);
```

--错误信息写在 bdump,cdump 下
show parameter dump

相关的字典
select * from v$lock;
select * from v$session;
select * from dba_jobs;
select * from DBA_JOBS_RUNNING;
select * from v$bgprocess where paddr<>'00';
select * from v$process where PROGRAM like 'ORACLE.EXE (J%';
select * from dba_sequences where SEQUENCE_NAME='JOBSEQ';

Shared_pool

存储很多信息
最重要的为 Library cache 和 Dictionary Cache
Shared_pool 的没有命中比 db_cache 的没有命中更加不好
Dictionary Cache
用户名称，段名称，profile,序列，表空间信息等
Library cache
曾经使用的 SQL，PL/SQL 的语句和执行计划

共享 sql 语句的条件
1。内存中有相同的文本串，共享。
2。Hash 值不在内存，执行分析
3。Hash 值相同，比较两个语句的文本是否相同。
4。对象的 owner 必须相同，不同帐号的同名表不能共享。
5。使用绑定变量时，变量名称必须相同
6。运行语句的环境相同，比如优化模式等参数

实验 134：sql 语句在 shared_pool 中的查询

该实验的目的是理解 SQL 语句如何存储在内存中
使用绑定变量
select * from emp where empno=7900;
select * from emp where empno=7902;
以上两句话不会共享。
var v1 number
begin
:v1:=7900;
end;
/
select * from emp where empno=:v1;
select sql_text from v$sqlarea where
 sql_text = 'select * from emp where empno=:v1';

开发的建议
统一绑定变量，sql,pl/sql 的命名习惯，空格的个数
尽量调用 pl/sql 的函数和存储过程

查看空余的共享池
SELECT * FROM V$SGASTAT
WHERE NAME = 'free memory'

AND POOL = 'shared pool';
如果有，说明共享池不必优化

实验 135：shared_pool 的 sql 命中率

该实验的目的是 SQL 命中给我们带来的好处
命中 SQL 语句的目的是为了共享以前的劳动成果.已经硬分析过的语句再下次运行的时候就不必硬分析了,软分析就可以了.
运行一个语句的过程:
1. 将 SQL 语句经过 hash 算法后得到一个值 hash_value
2. 如果该值在内存中存在,那么叫命中执行软分析
3. 如果该值不存在,执行硬分析
4. 进行语法分析,看语法是否有错误
5. 语意分析,看权限是否符合
6. 如果有视图,将视图的定义取出
7. 进行 SQL 语句的自动改写,如将子查询改写为连接
8. 优选最佳的执行计划
9. 变量的帮定
10. 运行执行计划
11. 将结果返回给用户
如果是软分析,直接运行 9 以后的步骤.从上面的过程看出,软分析比硬分析节约了很多的开销.

共享池的命中率
```
SELECT NAMESPACE, PINS, PINHITS, RELOADS, INVALIDATIONS
FROM V$LIBRARYCACHE ORDER BY NAMESPACE;
```
实例启动以来的命中率
```
select SUM(PINHITS)/SUM(PINS) from V$LIBRARYCACHE;
```
如果取一定的时间间隔，更加有代表意义
8：00 查看一下 V$LIBRARYCACHE
10：00 再次查看一下 V$LIBRARYCACHE
求出差值后在求命中率

V$SHARED_POOL_ADVICE
估算 10%---200%的现在配置的大小对数据库的影响有多大
如果 SQL 的命中率小于 90%,我们就要优化.优化的手段有:
1. 加大 shared_pool_size 的大小,但也不要太大,太大会增加管理的额外费用.
2. 编写程序的时候使用变量传入,而不是使用常量.
3. 将大的包定在内存中
4. 修改初始化参数 cursor_sharing
下面的实验验证了该参数的三个不同的选项的差别.
```
SQL> conn scott/tiger
Connected.
SQL> drop table t1;
drop table t1
           *
ERROR at line 1:
ORA-00942: table or view does not exist
```
删除 t1,如果存在删除,不存在预防

```
SQL> create table t1 as select * from emp;

Table created.
```
建立实验表 t1
```
SQL> insert into t1 select * from t1;

14 rows created.
```
自身插入自身,使表中的数据翻倍

```
SQL> /
SQL> /

57344 rows created.
```
重复运行,直到插入 5 万行左右,这时的表 t1 中有 10 万左右的行.
```
SQL> commit;

Commit complete.

SQL> update t1 set empno=1000;

114688 rows updated.
```
将所有行的 empno 都改为 1000
```
SQL> commit;

Commit complete.

SQL> update t1 set empno=2000 where rownum=1;

1 row updated.
```
将 t1 表的第一行的 empno 改为 2000
```
SQL> commit;
```
我们构造了一个列值的**分布不均匀**的大表.一行为 2000,其它行都为 1000.
```
Commit complete.
SQL> create index i_t1 on t1(empno);

Index created.
```
建立列 empno 的索引
```
SQL> analyze table t1 compute statistics ;

Table analyzed.
```
分析表,告诉数据库表的大小
```
SQL> analyze table t1 compute statistics for columns empno;

Table analyzed.
```
分析列,告诉数据库 empno 列的数据分布是不均匀的,只有一行为 2000,其它所有行为 1000.

验证精确匹配的效果
```
SQL> show parameter cursor_sharing
```
验证 cursor_sharing 参数的值为精确匹配(EXACT)
```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
cursor_sharing                       string      EXACT
SQL> select * from t1 where empno=1000;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=68 Card=114687 Bytes =3555297)
      1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=68 Card=114687 Bytes=3555297)
```
执行计划为全表扫描,因为要查找大部分的数据,不会使用到索引
```
SQL> select * from t1 where empno=2000;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=31)
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'T1' (Cost=2 Card=1 Bytes=31)
   2    1     INDEX (RANGE SCAN) OF 'I_T1' (NON-UNIQUE) (Cost=1 Card=1 )
```
执行计划为索引扫描,因为要查找一行的数据,会使用到索引

验证近似匹配的效果

```
SQL> conn / as sysdba
Connected.
SQL> alter system set cursor_sharing  =SIMILAR scope=spfile;

System altered.
修改参数,因为是静态参数,所以只能先修改参数文件,而不能直接修改内存
SQL> startup force;
ORACLE instance started.
想让参数起到作用,重新启动数据库
Total System Global Area  168893796 bytes
Fixed Size                   453988 bytes
Variable Size             100663296 bytes
Database Buffers           67108864 bytes
Redo Buffers                 667648 bytes
Database mounted.
Database opened.
SQL> show parameter cursor_sharing
验证 cursor_sharing 参数的值为近似匹配(SIMILAR)

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
cursor_sharing                       string      SIMILAR
SQL> conn scott/tiger
Connected.
SQL> set autot traceonly explain
SQL> select * from t1 where empno=1000;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=68 Card=114687 Bytes =3555297)
            1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=68 Card=114687 Bytes=3555297)

SQL> select * from t1 where empno=2000;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=31)
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'T1' (Cost=2 Card=1 Bytes =31)
      2    1     INDEX (RANGE SCAN) OF 'I_T1' (NON-UNIQUE) (Cost=1 Card=1 )
```

验证强制匹配的效果

```
SQL> conn / as sysdba
Connected.
SQL> alter system set cursor_sharing  =force scope=spfile;

System altered.

SQL> startup force;
ORACLE instance started.

Total System Global Area  168893796 bytes
Fixed Size                   453988 bytes
Variable Size             100663296 bytes
Database Buffers           67108864 bytes
Redo Buffers                 667648 bytes
```

```
Database mounted.
Database opened.
SQL> show parameter cursor_sharing
```
验证 cursor_sharing 参数的值为强制匹配(force)

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
cursor_sharing                       string      force
SQL>conn scott/tiger
SQL> set autot traceonly explain
SQL> select * from t1 where empno=1000;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=68 Card=57344 Bytes=  1777664)
     1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=68 Card=57344 Bytes=1777664)


SQL> select * from t1 where empno=2000;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=68 Card=57344 Bytes=1777664)
     1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=68 Card=57344 Bytes=1777664)
```
错误的执行计划

实验的总论:
强制匹配将 where 条件都用变量来处理,提高了 SQL 的命中率,但不能区分列值的数据敏感性,会导致部分 sql 语句的执行计划不是正确的.
近似匹配将 where 条件都用变量来处理,提高了 SQL 的命中率,但可以区分列值的数据敏感性,既保证了语句的复用,提高的命中率,又可以区分列的条件差异.但 oralce 有的时候会有 bug,导致美好的东西变成了泡影.所以我们改了以后一定观察一下性能.
精确匹配将原语句不处理,降低了 SQL 的命中率,但保证执行计划都是正确的.精确匹配为默认值.

实验 136：数据字典的命中率查询

该实验的目的是判定数据字典的命中率
Dictionary Cache 的统计

```
column parameter format a21
column pct_succ_gets format 999.9
column updates format 999,999,999
SELECT parameter
, sum(gets)
, sum(getmisses)
, 100*sum(gets - getmisses) / sum(gets) pct_succ_gets
, sum(modifications) updates
FROM V$ROWCACHE WHERE gets > 0
GROUP BY parameter;
```

实验 137：shared_pool 保留区的判断

该实验的目的是改变 SHARED_POOL_RESERVED 区的大小.
SHARED_POOL_RESERVED_SIZE
共享池的保留取，当内存需求大于 4400 字节时使用

默认的大小为共享池的 5%，不能大于 50%
select REQUESTS,REQUEST_MISSES from V$SHARED_POOL_RESERVED;
show parameter shared_pool_reserved_size

如果 REQUEST_MISSES 持续增大，说明小了
如果 REQUEST_MISSES 总为零，说明大了

CURSOR_SHARING 参数
1。应用中有大量的相似语句
2。由于 library cache 的没有命中造成响应速度下降
语句完全相同才共享内存（默认）
CURSOR_SHARING=EXACT
部分相同有可能共享
CURSOR_SHARING=SIMILAR
部分相同就共享（有潜在的问题）
CURSOR_SHARING=FORCE

最好保持该参数为 EXACT，而是通过修改程序的办法来共享 sql

其它内存优化

实验 138：db_cache 命中率和 db_cache 的细化管理

该实验的目的是理解表的访问流程,知道数据命中的好处.
查看 db_buffer 的命中率
```
SELECT NAME, PHYSICAL_READS, DB_BLOCK_GETS, CONSISTENT_GETS,
1 - (PHYSICAL_READS / (DB_BLOCK_GETS + CONSISTENT_GETS)) "Hit Ratio"
FROM V$BUFFER_POOL_STATISTICS;
```

sga_target 如果设为非零
下面参数将自动设置，即使你设了也没用
db_cache_size

查看当前时间点存储在内存的非系统数据块
```
SELECT o.OBJECT_NAME, COUNT(*) NUMBER_OF_BLOCKS
FROM DBA_OBJECTS o, V$BH bh
WHERE o.DATA_OBJECT_ID = bh.OBJD
AND o.OWNER != 'SYS'
GROUP BY o.OBJECT_NAME
ORDER BY COUNT(*);
```

估算 db_cache 放大或减小后对 I/o 的影响
```
COLUMN size_for_estimate FORMAT 999,999,999,999 heading 'Cache Size (MB)'
COLUMN buffers_for_estimate FORMAT 999,999,999 heading 'Buffers'
COLUMN estd_physical_read_factor FORMAT 999.90 heading 'Estd Phys|Read Factor'
COLUMN estd_physical_reads FORMAT 999,999,999 heading 'Estd Phys| Reads'

SELECT size_for_estimate, buffers_for_estimate, estd_physical_read_factor, estd_physical_reads
FROM V$DB_CACHE_ADVICE
WHERE name = 'DEFAULT'
AND block_size = (SELECT value FROM V$PARAMETER WHERE name = 'db_block_size')
AND advice_status = 'ON';
```

多元化使用内存
```
alter system set db_keep_cache_size=20m;
alter system set db_recycle_cache_size=10m;
```

```
 SELECT  * FROM  v$buffer_pool;


CREATE INDEX cust_idx ...
  STORAGE (BUFFER_POOL KEEP ...);

ALTER TABLE customer
  STORAGE (BUFFER_POOL RECYCLE);

ALTER INDEX cust_name_idx
  STORAGE (BUFFER_POOL KEEP);
```

实验 139： v$latch 的使用

该实验的目的是理解 latch 的机制
Latch 是闩锁,保护内存的,保证在同一个时间点只有一个进程在操作指定的内存,尤其在多 cpu 的系统中,虽然有多个处理器,但他们是排队的,因为只有获得了 latch 才能操作,没有获得 latch 的或者排队或者去干下一个任务,总之它不能干当前的任务.数据库原则上是不用我调节 latch 的参数的.

```
SQL> col name for a45
SQL> col value for a15
SQL> col isdefault for a8
SQL> col ismod for a8
SQL> col isadj for a8
SQL> select
  2    x.ksppinm  name,
  3    y.ksppstvl  value,
  4    y.ksppstdf  isdefault,
  5    decode(bitand(y.ksppstvf,7),1,'MODIFIED',4,'SYSTEM_MOD','FALSE')  ismod,
  6    decode(bitand(y.ksppstvf,2),2,'TRUE','FALSE')  isadj
  7  from
  8    sys.x$ksppi x,
  9    sys.x$ksppcv y
 10  where
 11    x.inst_id = userenv('Instance') and
 12    y.inst_id = userenv('Instance') and
 13    x.indx = y.indx and x.ksppinm like '%latch%'
 14  order by
 15    translate(x.ksppinm, ' _', ' ');

NAME                                       VALUE           ISDEFAUL ISMOD    ISADJ
------------------------------------------ --------------- -------- -------- --------
_db_block_hash_latches                     1024            TRUE     FALSE    FALSE
_db_block_lru_latches                      8               TRUE     FALSE    FALSE
_disable_latch_free_SCN_writes_via_32cas   FALSE           TRUE     FALSE    FALSE
_disable_latch_free_SCN_writes_via_64cas   FALSE           TRUE     FALSE    FALSE
_enable_reliable_latch_waits               TRUE            TRUE     FALSE    FALSE
_enqueue_hash_chain_latches                1               TRUE     FALSE    FALSE
_flashback_copy_latches                    10              TRUE     FALSE    FALSE
_gc_latches                                8               TRUE     FALSE    FALSE
_gcs_latches                               0               TRUE     FALSE    FALSE
_kgl_latch_count                           0               TRUE     FALSE    FALSE
_kgx_latches                               512             TRUE     FALSE    FALSE
_ktc_latches                               0               TRUE     FALSE    FALSE
_ktu_latches                               0               TRUE     FALSE    FALSE
_latch_class_0                                             TRUE     FALSE    FALSE
```

_latch_class_1		TRUE	FALSE	FALSE
_latch_class_2		TRUE	FALSE	FALSE
_latch_class_3		TRUE	FALSE	FALSE
_latch_class_4		TRUE	FALSE	FALSE
_latch_class_5		TRUE	FALSE	FALSE
_latch_class_6		TRUE	FALSE	FALSE
_latch_class_7		TRUE	FALSE	FALSE
_latch_classes		TRUE	FALSE	FALSE
_latch_miss_stat_sid	0	TRUE	FALSE	FALSE
_latch_recovery_alignment	998	TRUE	FALSE	FALSE
_lm_drm_xlatch	0	TRUE	FALSE	FALSE
_lm_num_pcmhv_latches	0	TRUE	FALSE	FALSE
_lm_num_pt_latches	128	TRUE	FALSE	FALSE
_max_sleep_holding_latch	4	TRUE	FALSE	FALSE
_num_longop_child_latches	1	TRUE	FALSE	FALSE
_session_idle_bit_latches	0	TRUE	FALSE	FALSE
_ultrafast_latch_statistics	TRUE	TRUE	FALSE	FALSE

我们看到有很多关于 latch 的隐含参数.但没有正式的关于 latch 的参数.
SQL> select * from v$parameter where name like '%latch%';

no rows selected
latch 有问题证明了内存有问题,内存有问题证明了 SQL 语句有问题.我们通过调整 SQL 的运行模式,改变 LATCH 的问题.
1.select sid,event from v$session_wait;
可以看到大量的 latch free 事件。如果没有,证明当前没有因为 latch 的等待事件
2。 P1－表示 Latch 地址，也就是进程正在等待的 latch 地址。
P2－表示 Latch 编号，对应于视图 V$LATCHNAME 中的 latch#。
select * from v$latchname where latch# = number;
P3－表示为了获得该 latch 而尝试的次数。
3。Cache buffers chains latch:
Data buffer chains—热点块,找到 misses>10000 次的

```
select CHILD# "cCHILD",ADDR "sADDR", GETS "sGETS", MISSES "sMISSES", SLEEPS "sSLEEPS"
from v$latch_children
where name = 'cache buffers chains'
and misses>10000
order by 4, 1, 2, 3;

--找到段的名称，千万别运行，老大的查询，相当慢
select /*+ RULE */
e.owner ||'.'|| e.segment_name segment_name,
e.extent_id extent#,
x.dbablk - e.block_id + 1 block#,x.tch,l.child#
from sys.v$latch_children l,sys.x$bh x,sys.dba_extents e
where
x.hladdr = 'SADDR' and
e.file_id = x.file# and
x.hladdr = l.addr and
x.dbablk between e.block_id and
e.block_id + e.blocks -1
order by x.tch desc ;

--library cache latch 的诊断
--查看 latch 信息：假定我们关心有关 library 的 latch.
select name,gets,misses,sleeps
from v$latch
where name like 'library%';
--查看 latch 操作系统进程号
```

```
select a.name,pid from v$latch a , V$latchholder b
where a.addr=b.laddr
and a.name = 'library cache%';
--查看关于 latch free 的等待的总数.
select count(*) number_of_waiters
from v$session_wait w, v$latch l
where w.wait_time = 0
and w.event = 'latch free'
and w.p2 = l.latch#
and l.name like 'library%';
```

实验 140：log_buffer 的优化

该实验的目的是优化日志缓冲区.
日志缓冲区
一个内存的运行原理决定了它优化模式. 日志缓冲区中存放的是数据库的变化的流水.如果数据库变化很快,日志的流量就很大.如果同时的业务很多,同时写日志的进程就会竞争.

```
show sga  显示日志缓冲区的大小
顺序写，循环写
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME = 'redo buffer allocation retries';
select * from v$system_event
where EVENT='log buffer space';
```
日志缓冲的优化主要有三个办法,

实验 141：pga 的优化

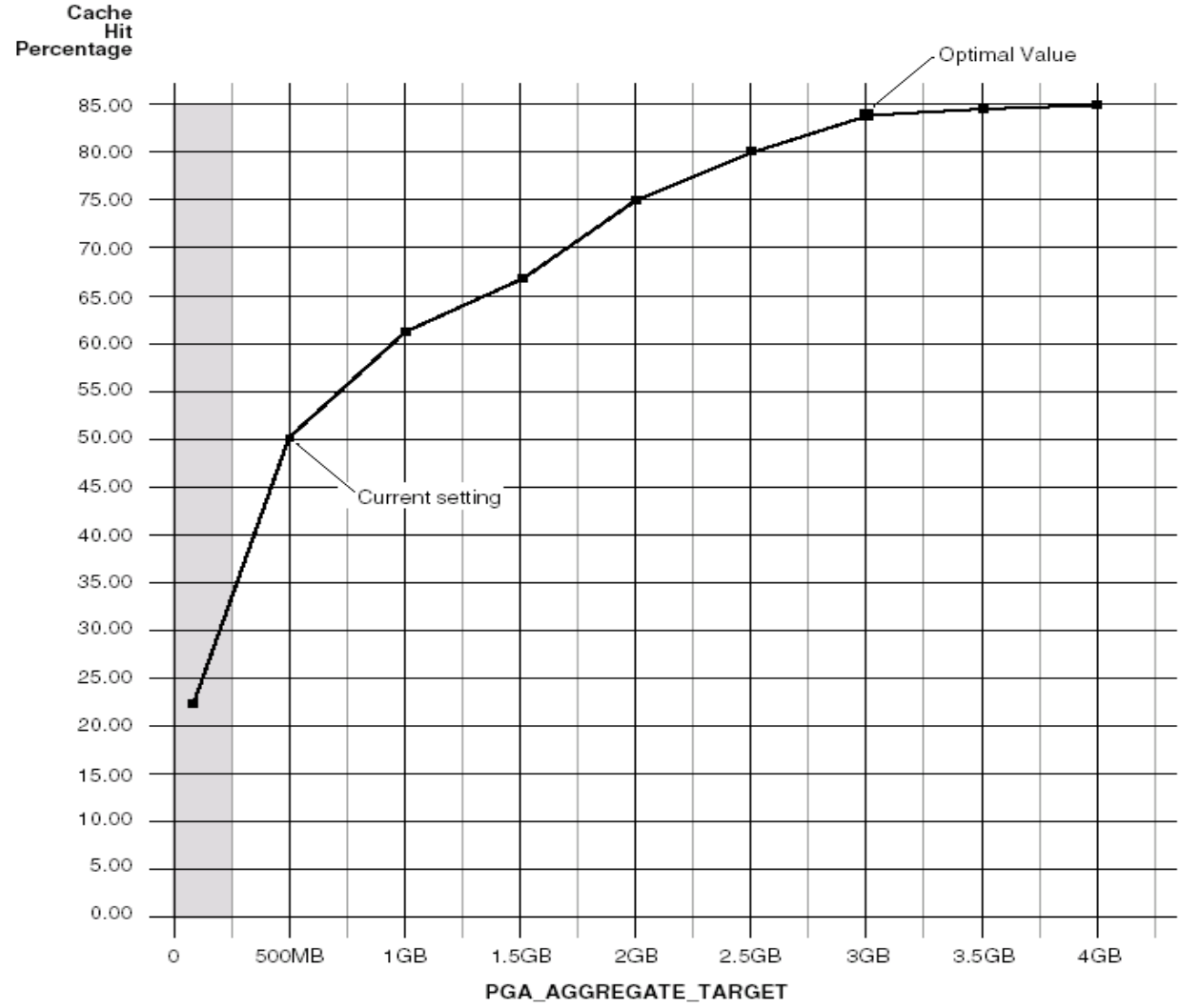
该实验的目的是理解排序操作对数据库的巨大影响.
Pga
由参数 pga_aggregate_target 决定
该参数会屏蔽*_AREA_SIZE 的设置

```
optimal size  ，要达到 90%，不够请加大 pga
one-pass size  （10%以内）22m 内存就可以排 1g 数据
Multi-pass  最好没有，极坏的负面影响

查看最优，一次过，多次过的次数
SELECT optimal_count, round(optimal_count*100/total, 2) optimal_perc,
onepass_count, round(onepass_count*100/total, 2) onepass_perc,
multipass_count, round(multipass_count*100/total, 2) multipass_perc
FROM
(SELECT decode(sum(total_executions), 0, 1, sum(total_executions)) total,
sum(OPTIMAL_EXECUTIONS) optimal_count,
sum(ONEPASS_EXECUTIONS) onepass_count,
sum(MULTIPASSES_EXECUTIONS) multipass_count
FROM v$sql_workarea_histogram
WHERE low_optimal_size > 64*1024);
```

Pga 和 sga 的分配
系统刚上线，不知道负载的情况
Oltp pga:sga=1:4

Dss　　pga:sga=1:1

```
SELECT round(PGA_TARGET_FOR_ESTIMATE/1024/1024) target_mb,
ESTD_PGA_CACHE_HIT_PERCENTAGE cache_hit_perc,
ESTD_OVERALLOC_COUNT
FROM V$PGA_TARGET_ADVICE;
```




```
SELECT name profile, cnt, decode(total, 0, 0, round(cnt*100/total)) percentage
FROM (SELECT name, value cnt, (sum(value) over ()) total
FROM V$SYSSTAT
WHERE name like 'workarea exec%');
```

DB_BLOCK_SIZE　　基本块大小
db_file_multiblock_read_count　　顺序读时一次读最多的块数

以随机读写为主的系统：条带>=2* DB_BLOCK_SIZE
以顺序读写为主的系统：条带>=2* DB_BLOCK_SIZE*块数

日志文件
最好单独放在独立的盘上
最好不要放在 raid5 的阵列上
归档日志最好和联机日志放在不同的设备上
同组的成员要放到不同的设备上

不同的存储格式

实验 142：OMF 管理的文件

该实验的目的是使用 omf 建立表空间.
Oracle-Managed File (omf)
SHOW PARAMETER db_create_
指定数据文件和日志文件的目录
可以动态修改
文件名称自动建立
删除的时候自动删除
对第三方程序特别方便，建立表空间的时候不必指明物理文件名称

建立 omf 管理的表空间

```
alter system set db_create_file_dest='D:\00';
create tablespace zl3 datafile size 1m;
select NAME from v$datafile;

ALTER SYSTEM SET db_create_online_log_dest_1 ='D:\01';
ALTER SYSTEM SET db_create_online_log_dest_2 ='D:\02';
SHOW PARAMETER db_create_
ALTER DATABASE ADD LOGFILE SIZE 2M;
SELECT MEMBER FROM V$LOGFILE;
```

热点的文件和表
V$filestat 文件的统计
V$SEGMENT_STATISTICS 表的 IO 统计

```
SELECT STATISTIC_NAME,STATISTIC#,VALUE from V$SEGMENT_STATISTICS where OBJECT_NAME='EMP';
```

表空间的规划
物以类聚
系统和非系统的分开
永久的和临时的分开
大的和小的分开
静态和动态的分开
数据和索引分开

自动回退管理性能高
空间足够大
监测
V$undostat
V$rollstat

日志大小应该容纳 20 分钟的业务
理论上大好
100m---2g
FAST_START_MTTR_TARGET 参数控制 checkpoint 的频率
V$log
V$logfile
V$LOG_HISTORY

实验 143：处理行迁移

该实验的目的是详细理解表的存储.
行迁移的形成
由于 update 造成的。
当行长长的时候，本数据块没有足够的空闲空间。
导致该行被迫存储到其它数据块，在原数据块保留访问的指针。
当数据库访问该行时，要进行二次 io.
下降数据库的性能

查看链接的行数
```
conn scott/tiger

ANALYZE TABLE t1 COMPUTE STATISTICS;

select NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE,CHAIN_CNT,AVG_ROW_LEN
from user_tables where table_name='T1';
```

找到链接的行
```
@%oracle_home%\rdbms\admin\utlchain.sql
Desc CHAINED_ROWS
ANALYZE TABLE t1 list chained rows;
select * from t1 where rowid in (select HEAD_ROWID from CHAINED_ROWS);
```

消除迁移的行
1。ANALYZE 语句定位迁移的行。
2。建立新的表 t，该表含有迁移的行。
3。删除迁移的行。
4。将表 t 的数据插入到原表。

```
SQL> conn scott/tiger
Connected.
SQL> drop table t1 purge;

Table dropped.

SQL> create table t1 (name varchar2(30));

Table created.
SQL> alter table t1 pctfree 0;

Table altered.
SQL> insert into t1 values('aaaa5');

1 row created.
SQL> insert into t1 select * from t1;

2 rows created.
.
.
.
```
一直重复插入,直到 t1 表达到 8000 行就可以了.
```
SQL> commit;

Commit complete.

SQL> ANALYZE TABLE t1 COMPUTE STATISTICS;
```

```
Table analyzed.

SQL> select NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE,CHAIN_CNT,AVG_ROW_LEN
  2  from user_tables where table_name='T1';

  NUM_ROWS     BLOCKS EMPTY_BLOCKS  AVG_SPACE  CHAIN_CNT AVG_ROW_LEN
---------- ---------- ------------ ---------- ---------- -----------
      8192         13            3       1139          0           9
```

我们看到没有迁移的行,平均行长为 9 个字节.因为行头有 4 个字节.

```
SQL> update t1 set name='aaaaaaaaaaaaaaaaaaaaaaa26';

8192 rows updated.

SQL> ANALYZE TABLE t1 COMPUTE STATISTICS;

Table analyzed.

SQL> select NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE,CHAIN_CNT,AVG_ROW_LEN
  2  from user_tables where table_name='T1';

  NUM_ROWS     BLOCKS EMPTY_BLOCKS  AVG_SPACE  CHAIN_CNT AVG_ROW_LEN
---------- ---------- ------------ ---------- ---------- -----------
      8192         80            8        595       8192          36
```

我们看到所有的行都迁移了,平均行长为 36.
我做的例子比较极端,一行不剩,全迁移了.

```
SQL> alter table t1 move tablespace users;

Table altered.

SQL> ANALYZE TABLE t1 COMPUTE STATISTICS;

Table analyzed.

SQL> select NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE,CHAIN_CNT,AVG_ROW_LEN
  2  from user_tables where table_name='T1';

  NUM_ROWS     BLOCKS EMPTY_BLOCKS  AVG_SPACE  CHAIN_CNT AVG_ROW_LEN
---------- ---------- ------------ ---------- ---------- -----------
      8192         38            2        102          0          30
```

我们消除了迁移的行,表由 80 个块,下降到 38 个块,平均行长由 36 下降为 30.

```
SQL> update t1 set name='aaaaaaaaaaaaaaaaaaaaaaaaaaa30';

8192 rows updated.

SQL> ANALYZE TABLE t1 COMPUTE STATISTICS;

Table analyzed.

SQL> select NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE,CHAIN_CNT,AVG_ROW_LEN
  2  from user_tables where table_name='T1';

  NUM_ROWS     BLOCKS EMPTY_BLOCKS  AVG_SPACE  CHAIN_CNT AVG_ROW_LEN
---------- ---------- ------------ ---------- ---------- -----------
      8192         55            1       1107       1280          35
```

我们看到又产生了行迁移.但没有那么多,只迁移了 1280 行.

```
SQL> @%oracle_home%\rdbms\admin\utlchain.sql
```

```
Table created.

SQL> desc CHAINED_ROWS
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- ------------
 OWNER_NAME                                                         VARCHAR2(30)
 TABLE_NAME                                                         VARCHAR2(30)
 CLUSTER_NAME                                                       VARCHAR2(30)
 PARTITION_NAME                                                     VARCHAR2(30)
 SUBPARTITION_NAME                                                  VARCHAR2(30)
 HEAD_ROWID                                                         ROWID
 ANALYZE_TIMESTAMP                                                  DATE
```

我们通过脚本建立了一个表 CHAINED_ROWS

```
SQL> ANALYZE TABLE t1 list chained rows;

Table analyzed.

SQL> select count(*) from CHAINED_ROWS;

  COUNT(*)
----------
      1280

SQL> create table t1_chain as select * from t1
  2  where rowid in(select HEAD_ROWID from CHAINED_ROWS);
```

t1_chain 表中临时存储被迁移的行.

```
Table created.

SQL> delete t1 where rowid in(select HEAD_ROWID from CHAINED_ROWS);
```

删除所有迁移的行

```
1280 rows deleted.

SQL> insert into t1 select * from t1_chain;
```

再将被删除的行插入

```
1280 rows created.

SQL> commit;

Commit complete.

SQL> ANALYZE TABLE t1 COMPUTE STATISTICS;

Table analyzed.

SQL> select NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE,CHAIN_CNT,AVG_ROW_LEN
  2  from user_tables where table_name='T1';

  NUM_ROWS     BLOCKS EMPTY_BLOCKS  AVG_SPACE  CHAIN_CNT AVG_ROW_LEN
---------- ---------- ------------ ---------- ---------- -----------
      8192         55            1       1439          0          34
```

外科手术式的消除迁移的行,因为 insert 不产生迁移,update 才会发生迁移.

消除行迁移的其它手段:
Exp/imp
Alter table move tablespace;

自动加锁：数据库在事务过程中，为了维护数据库的安全，自动加锁。

手工加锁：我们为了某些业务的需要，手工加锁。

解锁：事务结实，锁消除
锁的模式
0 - none
1 - null (NULL)
2 - row-S (SS)
3 - row-X (SX)
4 - share (S)
5 - S/Row-X (SSX)
6 - exclusive (X)

查看锁的信息
v$lock
 v$locked_object
DBA_BLOCKERS
DBA_WAITERS

DBA_DML_LOCKS
 COL OWNER FOR A8
 COL NAME   FOR A22
 COL TYPE FOR A20
DBA_DDL_LOCKS

DML 事务要两把锁
表：共享
行：独占
```
Update scott.emp set sal=sal+1;
Select * from v$Lock;
```

锁的模式
```
lock table emp in  ROW SHARE  mode; --2 号锁
Update scott.emp set sal=sal+1;--3 号锁
 lock table emp in share mode;--4 号锁
lock table emp in SHARE ROW EXCLUSIVE  mode; --5 号锁
 lock table emp in   EXCLUSIVE  mode; --6 号锁
```

实验 144：lock 的信息查询

该实验的目的是理解 lock 保护事务.
锁等待
锁是队列机制
先来的事务占有锁，事务结束后锁释放。
后来的事务排队
死锁
Oracle 自动检测死锁，将发现死锁的交易回退。
将死锁的信息写入 bdump 下的报警日志
```
conn scott/tiger
update emp set sal=sal+1;
```
产生一个交易,不要提交
```
select SID,TYPE,ID1,ID2,LMODE from v$lock where CTIME<1000;
```
查看加锁时间少于 1000 秒的锁信息
```
SID TYPE              ID1        ID2      LMODE
--- ---------- ---------- ---------- ----------
```

```
 17 TX              458789       7305          6
 17 TM               33218          0          3
```
Tx 是锁行的,tm 是锁表的.tm 的 33218 代表是对象的代码,在 dba_objects.object_id 列可以查到.
对于 TX 类型的 lock，ID1 表示 XIDUSN 和 XIDSLOT，ID2 为 XIDSQN
458789 可以分解为事务的信息.
ID1 的高 16 位为 XIDUSN，低 16 位为 XIDSLOT

```
select XIDUSN,XIDSLOT,XIDSQN from v$transaction;

XIDUSN    XIDSLOT     XIDSQN
------ ---------- ----------
     7         37       7305

SQL> select trunc(458789/power(2,16)) from dual;

TRUNC(458789/POWER(2,16))
-------------------------
                        7

SQL> select bitand(458789,to_number('ffff','xxxx')) from dual;

BITAND(458789,TO_NUMBER('FFFF','XXXX'))+0
-----------------------------------------
                                       37
```
求后 16 位的十进制的值

power(m,n) 求 m 的 n 次幂
bitand(m,n) 按位与运算,有点象子网掩码的意思

```
SQL> select bitand(1,1) from dual;

BITAND(1,1)
-----------
          1
1 and 1

SQL> select bitand(2,1) from dual;
10 and 01
BITAND(2,1)
-----------
          0


SQL> select 7*power(2,16)+37 from dual;

7*POWER(2,16)+37
----------------
         458789
```

在 10G 数据库中可以通过事物的 xid 找到原 SQL 语句.
```
SELECT UNDO_SQL FROM FLASHBACK_TRANSACTION_QUERY WHERE XID = '0200280094040000';

SQL> conn / as sysdba
Connected.
SQL> grant select any dictionary to scott;

Grant succeeded.
```

```
SQL> conn scott/tiger
Connected.
SQL> select sid from v$mystat where rownum=1;

       SID
----------
       159

SQL> select * from v$lock where sid=159;

no rows selected

SQL> update emp set sal=sal+1;

14 rows updated.
SQL> col type for a4
SQL> col  REQUEST for 9
SQL> col LMODE for 9
SQL> col CTIME for 9999
SQL> col BLOCK for 99
SQL> select * from v$lock where sid=159;

ADDR     KADDR           SID TYPE        ID1        ID2 LMODE REQUEST CTIME BLOCK
-------- -------- ---------- ---- ---------- ---------- ----- ------- ----- -----
189C4074 189C408C        159 TM        54638          0     3       0   102     0
18A18084 18A181A0        159 TX        65540       1216     6       0   102     0
SQL> drop table t1 purge;

Table dropped.

SQL> create table t1 as select * from emp;
建立实验表 t1;
Table created.

SQL> select * from v$lock where sid=159;
没有锁了,因为建立和 drop 表是 ddl 语句,自动提交,锁的生命周期伴随着事务的完结而结束.
no rows selected
我们再开一个新的会话,以 scott 用户登录.查看两个 scott 用户的 sid.
SQL> select sid from v$session where username='SCOTT';

       SID
----------
       146
       159
我们再连接一个高级用户,现在我们有三个窗口,我们假定 3 号为高级用户,负责查看锁的信息的,1 号和 2
号是以 scott 用户登录的,用来做交易的.其中 1 号会话的 sid 为 159, 其中 2 号会话的 sid 为 146,
1 号窗口改 emp 表,update emp set sal=sal+1;
2 号窗口改 t1 表,update t1 set sal=sal+1;
3 号窗口查看 v$lock
SQL> select * from v$lock where sid in(159,146);

ADDR     KADDR           SID TYPE        ID1        ID2 LMODE REQUEST CTIME BLOCK
-------- -------- ---------- ---- ---------- ---------- ----- ------- ----- -----
189C4074 189C408C        159 TM        54638          0     3       0    18     0
189C4138 189C4150        146 TM        54726          0     3       0     6     0
18A008C4 18A009E0        146 TX       458799       1172     6       0     6     0
18A185A8 18A186C4        159 TX       327722       1172     6       0    18     0
相安无事,因为每个人改的对象不同.各自加了锁再各自影响的表和行上.
```

现在我们在 1 号窗口改 t1 表,update t1 set sal=sal+1;改不了,因为锁被 2 号窗口把持.1 号会话的状态为等待,屏幕不动了.

```
SQL> select * from v$lock where sid in(159,146) order by sid;

ADDR     KADDR           SID TYPE        ID1        ID2 LMODE REQUEST CTIME BLOCK
-------- -------- ---------- ---- ---------- ---------- ----- ------- ----- -----
189C4138 189C4150        146 TM        54726          0     3       0   221     0
18A008C4 18A009E0        146 TX       458799       1172     6       0   221     1
18A185A8 18A186C4        159 TX       327722       1172     6       0   233     0
189C41FC 189C4214        159 TM        54726          0     3       0    13     0
189C4074 189C408C        159 TM        54638          0     3       0   233     0
19434394 194343A8        159 TX       458799       1172     0       6    13     0
```

Block 为 1 的含义是它堵塞了其它会话,因为 2 号先在行上加了 6 号独占锁,1 号会话不能获得,只能排队了.
REQUEST 为 6 的含义是该会话正在申请 6 号锁,而没有获得.我们通过 id1 这列可以发现他们要使用同一个资源,所以 146 号会话堵塞了 159 号会话.
接下来我们在 2 号会话中运行 update emp set sal=sal+1;然后迅速回到 3 号会话进行查看,看看发现了什么?连续看两回.

```
ADDR     KADDR           SID TYPE        ID1        ID2 LMODE REQUEST CTIME BLOCK
-------- -------- ---------- ---- ---------- ---------- ----- ------- ----- -----
189C4138 189C4150        146 TM        54726          0     3       0   545     0
18A008C4 18A009E0        146 TX       458799       1172     6       0   545     1
189C42C0 189C42D8        146 TM        54638          0     3       0     6     0
1943444C 19434460        146 TX       327722       1172     0       6     6     0
18A185A8 18A186C4        159 TX       327722       1172     6       0   557     1
189C4074 189C408C        159 TM        54638          0     3       0   557     0
189C41FC 189C4214        159 TM        54726          0     3       0   337     0
19434394 194343A8        159 TX       458799       1172     0       6   334     0

8 rows selected.
```

我们在瞬间会看到有两个 block 为 1,两个 request 为 6,再仔细看,他们相互锁着,死循环了.

```
SQL> /

ADDR     KADDR           SID TYPE        ID1        ID2 LMODE REQUEST CTIME BLOCK
-------- -------- ---------- ---- ---------- ---------- ----- ------- ----- -----
1943444C 19434460        146 TX       327722       1172     0       6     6     0
189C4138 189C4150        146 TM        54726          0     3       0   545     0
18A008C4 18A009E0        146 TX       458799       1172     6       0   545     0
189C42C0 189C42D8        146 TM        54638          0     3       0     6     0
18A185A8 18A186C4        159 TX       327722       1172     6       0   557     1
189C4074 189C408C        159 TM        54638          0     3       0   557     0

6 rows selected.
```

当我们再次查看的时候,一个会话锁没有了,只剩下一个堵塞和一个请求了.原来数据库检测到死锁,回退了一个语句,到底退哪个?谁先判定就先退谁的最后一句话,有的时候同时都退.这种情况很少,一般是退第一个会话的第二条语句.我们会看到 ORA-00060: deadlock detected while waiting for resource.
在 bdump 和 udump 中有语句的信息.
锁是排队机制的,一个等待一个.根据 ctime 来判断.单位为秒.

SQL 语句的优化

Sql 语句的执行步骤

分析→绑定→运行→抓取(**parsing, binding, executing, and fetching**)
只有查询语句才运行抓取的阶段,如果是 DDL,DML 就没有抓取阶段.
FETCH—(RE)BIND—EXECUTE—FETCH
上面的过程可以是反复进行的,抓取,再绑定,运行,抓取,直到完成.

分析阶段做如下的过程:
1。检查该语句是否存在于 shared_pool 内存中,如果在叫命中.
2。语法分析,检测是否存在 SQL 的语法错误
3。语意分析,检测同意词和权限.
4。视图替代,融合视图和子查询
5。选出最优的执行计划
绑定阶段:
扫描所有的绑定变量
将实际的值替代到变量中.
运行阶段:
运行优选出来的执行计划
进行 i/o 和排序操作(特指 dml 语句的排序)
抓取阶段:
返回行
进行排序

实验 145：explain 列出执行计划

该实验的目的是会使用 explain 语句查看 SQL 的执行计划.
Explain plan 方式列出执行计划.
Conn scott/tiger
@%ORACLE_HOME%/rdbms/admin/utlxplan.sql
建立了表 plan_table,这个表用于存放 SQL 语句的运行计划
SQL> desc plan_table;

Name	Null?	Type
STATEMENT_ID		VARCHAR2(30)
PLAN_ID		NUMBER
TIMESTAMP		DATE
REMARKS		VARCHAR2(4000)
OPERATION		VARCHAR2(30)
OPTIONS		VARCHAR2(255)
OBJECT_NODE		VARCHAR2(128)
OBJECT_OWNER		VARCHAR2(30)
OBJECT_NAME		VARCHAR2(30)
OBJECT_ALIAS		VARCHAR2(65)
OBJECT_INSTANCE		NUMBER(38)
OBJECT_TYPE		VARCHAR2(30)
OPTIMIZER		VARCHAR2(255)
SEARCH_COLUMNS		NUMBER
ID		NUMBER(38)
PARENT_ID		NUMBER(38)
DEPTH		NUMBER(38)
POSITION		NUMBER(38)
COST		NUMBER(38)
CARDINALITY		NUMBER(38)
BYTES		NUMBER(38)
OTHER_TAG		VARCHAR2(255)
PARTITION_START		VARCHAR2(255)
PARTITION_STOP		VARCHAR2(255)
PARTITION_ID		NUMBER(38)
OTHER		LONG
OTHER_XML		CLOB
DISTRIBUTION		VARCHAR2(30)
CPU_COST		NUMBER(38)

```
 IO_COST                                                   NUMBER(38)
 TEMP_SPACE                                                NUMBER(38)
 ACCESS_PREDICATES                                         VARCHAR2(4000)
 FILTER_PREDICATES                                         VARCHAR2(4000)
 PROJECTION                                                VARCHAR2(4000)
 TIME                                                      NUMBER(38)
 QBLOCK_NAME                                               VARCHAR2(30)
```

```
--产生计划
Explain plan for select ename from emp where empno=7900;
```

这句话的含义是将执行计划存储到 plan_table 表中.

```
--查看计划
SQL> col options for a20
SQL> col OPERATION for a30
SQL> select ID,PARENT_ID,OBJECT_NAME,OPTIONS,OPERATION from plan_table order by 1;

        ID  PARENT_ID OBJECT_NAME                    OPTIONS              OPERATION
---------- ---------- ------------------------------ -------------------- ------------------------------
         0                                                                  SELECT STATEMENT
         1          0 EMP                            BY INDEX ROWID       TABLE ACCESS
         2          1 PK_EMP                         UNIQUE SCAN          INDEX
```

上面是一个比较简单的执行计划,我们应该如何看懂这个执行计划呢?我们看 id,parent_id.请记住执行计划永远是一个二叉树.底下的结果返回给父 id.

我们再来看这个计划,这个树的最底层是 id=2 的语句.先运行索引 PK_EMP 的唯一定位扫描,因为是主键,不存在重复值的问题.将索引得到的 rowid 返回 id=1 的语句,该语句调用了查询索引得到的 rowid,查找到与之相对应的行.

因为我们得到的计划是一个树状结构,所以可以使用层次结构查询,通过伪列 level 来使查询的计划看起来有层次感,越靠右的语句越先运行.

```
SELECT LPAD(' ',LEVEL*2)||' '||OPERATION||' '||OPTIONS||' | '||OBJECT_NAME AS "SELECT QUERY"
FROM plan_table  START WITH ID=0
CONNECT BY PRIOR ID=PARENT_ID;

SELECT QUERY
----------------------------------------
    SELECT STATEMENT  |
      TABLE ACCESS BY INDEX ROWID | EMP
        INDEX UNIQUE SCAN | PK_EMP
```

我们完全可以将执行计划保存到我们指定的表中.

```
SQL> create table MY_PLAN_TABLE (
  2          statement_id       varchar2(30),
  3          plan_id            number,
  4          timestamp          date,
  5          remarks            varchar2(4000),
  6          operation          varchar2(30),
  7          options            varchar2(255),
  8          object_node        varchar2(128),
  9          object_owner       varchar2(30),
 10          object_name        varchar2(30),
 11          object_alias       varchar2(65),
 12          object_instance    numeric,
 13          object_type        varchar2(30),
 14          optimizer          varchar2(255),
 15          search_columns     number,
 16          id                 numeric,
 17          parent_id          numeric,
 18          depth              numeric,
 19          position           numeric,
```

```
 20          cost                numeric,
 21          cardinality         numeric,
 22          bytes               numeric,
 23          other_tag           varchar2(255),
 24          partition_start     varchar2(255),
 25          partition_stop      varchar2(255),
 26          partition_id        numeric,
 27          other               long,
 28          distribution        varchar2(30),
 29          cpu_cost            numeric,
 30          io_cost             numeric,
 31          temp_space          numeric,
 32          access_predicates   varchar2(4000),
 33          filter_predicates   varchar2(4000),
 34          projection          varchar2(4000),
 35          time                numeric,
 36          qblock_name         varchar2(30),
 37          other_xml           clob
 38  );

Table created.


EXPLAIN PLAN SET STATEMENT_ID = 'st1'  into my_plan_table
FOR SELECT ename FROM emp;
```

我们上面增加了两个选项,一个指定了语句的标识.好在一张表中存储多个执行计划,再使用 where 语句来区分开来;into 语句的目的是把执行计划存储到我们指定的表中,但表的结构要相同.

```
EXPLAIN PLAN SET STATEMENT_ID = 'st2' into my_plan_table
FOR SELECT dname FROM dept;
```

调用数据库提供的脚本来自动美化输出最后的计划

```
SQL>@%oracle_home%\rdbms\admin\UTLXPLS.SQL
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2949544139

--------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    10 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    10 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | PK_EMP  |     1 |       |     0   (0)| 00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("EMPNO"=7900)

SQL> select plan_table_output from table(dbms_xplan.display('plan_table',null,'serial'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2949544139

--------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
```

```
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |     1 |    10 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    10 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_EMP  |     1 |       |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("EMPNO"=7900)
```

Plan_table 这个表可以随时的删除,清空,重新建立.10G 版本的数据库有了一些的改进,plan_table 是一个公共的同意词,代表的是 sys.PLAN_TABLE$表.我们最好不使用公共的 plan_table,使用在自己用户下建立的 plan_table,这样的好处是不给 system 表空间带来压力,10g 的好多地方象糊弄傻子似的.都是狗尾续貂的感觉.越来越没有审美了,追求一些虚浮的东西.

实验 146：跟踪 sql 语句的使用

该实验的目的是使用 SQL 跟踪,查看 udump 的 trc 文件.
查看下列**参数**
TIMED_STATISTICS 可以统计关于时间的信息.默认为 TRUE
MAX_DUMP_FILE_SIZE 可以限制跟踪文件的大小,默认为 UNLIMITED,不限制大小.

```
SQL> show parameter user_d

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
user_dump_dest                       string      D:\ORACLE\ADMIN\ORA10\UDUMP
```

我们可以动态的修改存储跟踪文件的位置.
alter system set USER_DUMP_DEST='c:\bk';
我一般不去修改存储跟踪文件的目录,没有必要,除非你的跟踪文件想存储到其它目录中.
10G 版本的数据库权限设计的严谨了,scott 这个用户没有修改会话的权限.
为了查到更加多的信息,我们赋予下面的权限.

```
SQL> conn / as sysdba
Connected.
SQL> grant alter session,SELECT ANY DICTIONARY to scott;

Grant succeeded.

SQL> conn scott/tiger
Connected.
SQL> select * from session_privs;

PRIVILEGE
----------------------------------------
CREATE SESSION
ALTER SESSION
UNLIMITED TABLESPACE
CREATE TABLE
CREATE CLUSTER
CREATE SEQUENCE
CREATE PROCEDURE
CREATE TRIGGER
CREATE TYPE
CREATE OPERATOR
```

```
CREATE INDEXTYPE
SELECT ANY DICTIONARY

Alter session set sql_trace=true;
Select * from emp where empno=7900;
Select * from dept;
Select empno from emp where empno=1000;
Alter session set sql_trace=false;
```

True 和 false 语句之间的 sql 运行的信息都会存储到一个跟踪文件中.
文件的名称和该会话的**操作系统进程号**相关联.
我们的 udump 目录下存放了好多的跟踪文件,到底哪个文件是我们的呢?
我们下面就一步步的去查找.

```
SQL> select sid from v$mystat where rownum=1;

       SID
----------
       159
```

查找当前会话的 sid.

```
SQL> select paddr from v$session where sid=159;

PADDR
--------
19E4C00C
```

查找该会话的程序地址

```
SQL> select spid from v$process where addr='19E4C00C';

SPID
------------------------
2228
```

查找该会话服务进程的操作系统号码

```
SQL> select value from v$parameter where name='user_dump_dest';

VALUE
-------------------------------------------------------------
D:\ORACLE\ADMIN\ORA10\UDUMP
```

查找存储跟踪文件的目录.

```
SQL> select instance_name from v$instance;

INSTANCE_NAME
--------------------------------
ora10
```

查找实例的名称
跟踪文件有固定的名称和位置.文件的名称为 sid_ora_spid.trc,哪我们的当前跟踪文件为 Ora10_ora_2228.trc 我们到 D:\ORACLE\ADMIN\ORA10\UDUMP 目录下果然有这个文件.

我们上面写的是分步骤查找的,我也可以写一个联合查询.

```
SQL> select p.value||'\'||i.instance_name||'_ora_'||spid||'.trc' as "trace_file_name"
  2  from v$parameter p ,v$process pro, v$session s,
  3   (select sid from v$mystat where rownum=1) m,
  4   v$instance i
  5  where lower(p.name)='user_dump_dest'
  6  and pro.addr=s.paddr
  7  and m.sid=s.sid;

trace_file_name
-------------------------------------------------------------------------------
D:\ORACLE\ADMIN\ORA10\UDUMP\ora10_ora_2228.trc
```

我们现在打开这个文件,以文本方式打开,我们截取部分看以下

```
====================
PARSING IN CURSOR #2 len=18 dep=0 uid=72 oct=3 lid=72 tim=44186670798 hv=3911648221 ad='166d3298'
Select * from dept
END OF STMT
PARSE #2:c=0,e=4699,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=44186670785
EXEC #2:c=0,e=102,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=44186684474
FETCH #2:c=0,e=227,p=0,cr=7,cu=0,mis=0,r=1,dep=0,og=1,tim=44186687634
FETCH #2:c=0,e=78,p=0,cr=1,cu=0,mis=0,r=3,dep=0,og=1,tim=44186691258
STAT #2 id=1 cnt=4 pid=0 pos=1 obj=54636 op='TABLE ACCESS FULL DEPT (cr=8 pr=0 pw=0 time=193 us)'
```

很难看明白.

数据库为我们提供了一个工具来格式化产生的跟踪文件.

```
tkprof D:\ORACLE\ADMIN\ORA10\UDUMP\ora10_ora_2228.trc  c:\bk\1.txt sys=no
```

sys=no 的含义是只查看用户的语句,sys 自己分析调用的不看.我们现在查看 1.txt 文件的部分.

```
TKPROF: Release 10.2.0.1.0 - Production on Thu Oct 11 22:24:53 2007

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Trace file: D:\ORACLE\ADMIN\ORA10\UDUMP\ora10_ora_2824.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************

Alter session set sql_trace=true


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      1      0.00       0.01          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        1      0.00       0.01          0          0          0           0

Misses in library cache during parse: 0
Misses in library cache during execute: 1
Optimizer mode: ALL_ROWS
Parsing user id: 72
********************************************************************************

Select *
from
 emp where empno=:"SYS_B_0"


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
```

```
Execute      1      0.00       0.03          0          0          0           0
Fetch        2      0.00       0.00          0          2          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.00       0.04          0          2          0           1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 72

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID EMP (cr=2 pr=0 pw=0 time=90 us)
      1   INDEX UNIQUE SCAN PK_EMP (cr=1 pr=0 pw=0 time=43 us)(object id 54639)

********************************************************************************

Select *
from
 dept


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          8          0           4
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.00       0.00          0          8          0           4

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 72

Rows     Row Source Operation
-------  ---------------------------------------------------
      4  TABLE ACCESS FULL DEPT (cr=8 pr=0 pw=0 time=193 us)
********************************************************************************
```

其中******号所间隔的是一句话,包含三部分内容,原语法,运行的统计,执行计划.

我们如果是高级用户,可以跟踪指定的会话

```
Conn sys/sys as sysdba
execute DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(147, 33, true);

execute DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(147, 33, false);
```

如果我们在程序中,可以使用下面的语句来跟踪

execute dbms_session.set_sql_trace(true);(程序中)

实验 147：AUTOTRACE 的使用

该实验的目的是使用 sqlplus 的特性,查看每句话的计划和统计信息.

```
SQL>Conn sys/sys as sysdba
SQL>@%oracle_home%\sqlplus\admin\plustrce.sql
```

我们运行脚本产生一个角色 plustrace.

```
SQL>Grant plustrace to scott;
SQL>Conn scott/tiger
SQL>Set autotrace on
SQL> select * from emp where empno=7900;

     EMPNO ENAME      JOB                 MGR HIREDATE            SAL  COMM DEPTNO
---------- ---------- ----------------- ---------- ----------- ---------- ----- ------
      7900 JAMES      CLERK                7698 03-DEC-81           960           30


Execution Plan
----------------------------------------------------------
Plan hash value: 2949544139

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    38 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    38 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | PK_EMP  |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("EMPNO"=7900)


Statistics
----------------------------------------------------------
        205  recursive calls
          0  db block gets
         39  consistent gets
          0  physical reads
          0  redo size
        730  bytes sent via SQL*Net to client
        374  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          6  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

我们看到的结果有**三部分**,查询结果,执行计划和统计信息.

如果我们不想看结果,只看计划和统计信息**两部分**

```
SQL> set autotrace traceonly
SQL> Select empno from emp where empno=1000;

no rows selected


Execution Plan
----------------------------------------------------------
Plan hash value: 56244932

--------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     1 |     4 |     0   (0)| 00:00:01 |
|*  1 |  INDEX UNIQUE SCAN| PK_EMP  |     1 |     4 |     0   (0)| 00:00:01 |
```

```
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("EMPNO"=1000)


Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          1  consistent gets
          0  physical reads
          0  redo size
        274  bytes sent via SQL*Net to client
        374  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

下面的设置只看执行计划

```
SQL> set autotrace traceonly explain
SQL> Select empno from emp where empno=1000;

Execution Plan
----------------------------------------------------------
Plan hash value: 56244932

--------------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |     1 |     4 |     0   (0)| 00:00:01 |
|*  1 |  INDEX UNIQUE SCAN| PK_EMP |     1 |     4 |     0   (0)| 00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("EMPNO"=1000)
```

下面的设置只看统计

```
SQL> set autotrace traceonly stat
SQL> Select empno from emp where empno=1000;

no rows selected


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          1  consistent gets
          0  physical reads
          0  redo size
        274  bytes sent via SQL*Net to client
        374  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
```

```
          0  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

下面的设置关闭自动跟踪,回到正常.
SQL> **set autotrace off**

优化 SQL 语句的过程:
1. 找到消耗资源高的 SQL,或者运行时间大于 6 秒的 SQL.
2. 列出该语句的执行计划
3. 改变执行计划
4. 运行新的语句进行测试.

我们的如何查看执行计划呢?
我们主要关注四项:
1. COST 值的高低,一般说来 COST 越高的语句运行越慢.
2. 连接的模式,如果连接模式不正确会引起灾难性的后果,比如大表之间的嵌套循环就会有问题.
3. 是否使用了索引,索引和全表扫描是死对头,对立的,二者只能取其一.
4. 排序的使用,我们可以通过索引来避免排序.

我们如何改变执行计划呢?
1. 改变数据库的版本,版本越高 sql 的引擎越智能.执行计划越优秀.
2. 改变数据库的初始化参数,比如内存,IO,索引等参数
3. 收集统计信息
4. 增加或减少索引
5. 改写原来的 SQL
6. 使用强制 HINT 来指定执行计划.

实验 148：定位高消耗资源语句

该实验的目的是找到对数据库有很大影响的 SQL 语句.
定位资源使用高的 SQL
```
 select max(DISK_READS),max(EXECUTIONS),max(BUFFER_GETS),MAX(SORTS) FROM v$sqlstats;
Select sql_text from v$sqlstats where BUFFER_GETS>####;
```
sql_text 只显示前 1000 个字符，如果想看全的 sql ,查 V$SQLTEXT

列出该表的执行计划

书写高效的 SQL 语句的原则:
在 where 中用 = 关系运算时, 避免用函数在关系运算中，除非你使用函数建立索引
```
      Where ename='king'
      Where  upper(ename)='KING'
      尽量不要隐式转化数据类型，数据类型一定要匹配
```

尽量将一句 SQL 分成多个语句完成。With 语句
条件确定的子查询用 in
条件在父查询中判断的用 exists

使用视图的注意事项
复杂视图的连接要小心，尤其有外键的时候
当查询的是视图中引用的部分表的时候，请不要使用视图，或者建立新的更小的表

存储中间结果
对查询中可能多次调用的结果集，请保存
考虑使用物化视图
将复杂的不能优化的查询分阶段完成。

尽量减少访问数据的次数
使用 case 语句
使用高级分组 rollup,cube
使用存储过程
使用 RETURNING 子句
DELETE FROM employees WHERE job_id = 'SA_REP' AND hire_date + TO_YMINTERVAL('01-00') < SYSDATE
RETURNING salary INTO :bnd1;
避免再次访问原来的表来获得数据,减少了工作量.

show parameter optimizer_mode
基于 cost 的优化
要使 SQL 语句的执行计划最优化，数据库必须知道关于数据库的详细统计数据
表（行数，块数，平均行长,hwm）
列（不同的值数，null 值的行数，柱状图）
索引（叶子数，索引深度，索引因子）
系统（IO 的性能和应用，CPU 的性能和应用）
Cost 是一个相对的值,和主机的环境有很大的关系,不同的数据库 cost 没有可比性.

代价最高的 SQL,不论运行的次数多少, OPTIMIZER_COST 不累加
```
select OPTIMIZER_COST,EXECUTIONS,sql_text from v$sqlarea
where OPTIMIZER_COST>
(select max(OPTIMIZER_COST)/5 from v$sqlarea);

OPTIMIZER_COST,EXECUTIONS
--------------,----------
SQL_TEXT
------------------------------------------------------------------------------------------
--------
          3399,          3
select count(*) from dba_extents

          2049,          1
select max(blocks) from dba_segments

          3087,          1
select                                                  o.owner#,o.obj#,decode(o.linkname,null,
decode(u.name,null,'SYS',u.name),o.remoteowner),  o.na
me,o.linkname,o.namespace,o.subname  from  user$  u,  obj$  o  where  u.user#(+)=o.owner#  and
o.type#=:1 an
d not exists (select p_obj# from dependency$ where p_obj# = o.obj#) order by o.obj# for update

          2050,          1
select segment_name from dba_segments where blocks=:"SYS_B_0"
```

IO 最高的 SQL, DISK_READS 为总的个数,需要除已执行次数
```
select round(DISK_READS/EXECUTIONS) ,DISK_READS,EXECUTIONS,sql_text
from v$sqlarea
where round(DISK_READS/EXECUTIONS)>
(select max(round((DISK_READS/EXECUTIONS)/5)) from v$sqlarea
where EXECUTIONS>0)
and EXECUTIONS>0 and DISK_READS>100
order by 1;

ROUND(DISK_READS/EXECUTIONS),DISK_READS,EXECUTIONS
---------------------------,----------,----------
SQL_TEXT
```

```
--------------------------------------------------
                        3988,      11963,          3
select count(*) from dba_extents
```

处理行最多的 SQL, ROWS_PROCESSED 为总的行数,需要除已执行次数

```
select round(ROWS_PROCESSED/EXECUTIONS) ,ROWS_PROCESSED,EXECUTIONS,sql_text
from v$sqlarea
where round(ROWS_PROCESSED/EXECUTIONS)>
(select max(round((ROWS_PROCESSED/EXECUTIONS)/5)) from v$sqlarea
where EXECUTIONS>0)
and EXECUTIONS>0 and ROWS_PROCESSED>1000
order by 1;

ROUND(ROWS_PROCESSED/EXECUTIONS),ROWS_PROCESSED,EXECUTIONS
---------------------------------,--------------,----------
SQL_TEXT
------------------------------------------------------------
                             8192,          16384,          2
select * from t1
```

实验 149：收集数据库的统计信息

该实验的目的是收据数据库的统计信息.
统计信息可以收集,删除和给假的仿真信息.
统计的数据存储在数据字典中

```
DBA_TABLES                          DBA_OBJECT_TABLES
DBA_TAB_STATISTICS    DBA_TAB_COL_STATISTICS
DBA_TAB_HISTOGRAMS    DBA_INDEXES
DBA_IND_STATISTICS        DBA_CLUSTERS
DBA_TAB_PARTITIONS    DBA_TAB_SUBPARTITIONS
DBA_IND_PARTITIONS    DBA_IND_SUBPARTITIONS
DBA_PART_COL_STATISTICS
DBA_PART_HISTOGRAMS
DBA_SUBPART_COL_STATISTICS
DBA_SUBPART_HISTOGRAMS
```

以上视图可以查看数据库的统计信息
我们可以使用 analyze 语句,也可以使用 dbms_stats 包来收集

```
conn system/manager
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('SCOTT',DBMS_STATS.AUTO_SAMPLE_SIZE);
```

自己决定抽样统计的权重

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('SCOTT',DBMS_STATS.AUTO_SAMPLE_SIZE,CASCADE=>true);
```

同时统计相关的索引信息.

统计表的信息和索引的信息.我们可以写脚本统计当前用户的所有表和索引,也可以使用动态 SQL 来完成.

```
Conn scott/tiger
ANALYZE TABLE EMP ESTIMATE STATISTICS;
ANALYZE TABLE EMP COMPUTE STATISTICS;

ANALYZE index pk_emp ESTIMATE STATISTICS;
ANALYZE index pk_emp COMPUTE STATISTICS;
```

产生脚本,然后运行脚本.

```
SQL> select 'analyze '||object_type||' '||object_name||' ESTIMATE STATISTICS
  2  from user_objects where object_type in('TABLE','INDEX')
  3  ORDER BY 1;

'ANALYZE'||OBJECT_TYPE||''||
--------------------------------------------------------------------------------
analyze INDEX IT4 ESTIMATE STATISTICS;
analyze INDEX I_F_SAL ESTIMATE STATISTICS;
analyze INDEX PK_DEPT ESTIMATE STATISTICS;
analyze INDEX PK_EMP ESTIMATE STATISTICS;
analyze INDEX PK_EMP1 ESTIMATE STATISTICS;
analyze TABLE BONUS ESTIMATE STATISTICS;
analyze TABLE DEPT ESTIMATE STATISTICS;
analyze TABLE E ESTIMATE STATISTICS;
analyze TABLE EMP ESTIMATE STATISTICS;
analyze TABLE EMP1 ESTIMATE STATISTICS;
analyze TABLE MV1 ESTIMATE STATISTICS;
analyze TABLE SALGRADE ESTIMATE STATISTICS;
analyze TABLE T1 ESTIMATE STATISTICS;
analyze TABLE T1_1 ESTIMATE STATISTICS;
analyze TABLE T1_4 ESTIMATE STATISTICS;
analyze TABLE T2 ESTIMATE STATISTICS;
analyze TABLE T201 ESTIMATE STATISTICS;
analyze TABLE T202 ESTIMATE STATISTICS;
analyze TABLE T205 ESTIMATE STATISTICS;
analyze TABLE T213 ESTIMATE STATISTICS;
analyze TABLE T236 ESTIMATE STATISTICS;
analyze TABLE T239 ESTIMATE STATISTICS;
analyze TABLE T3 ESTIMATE STATISTICS;
analyze TABLE T4 ESTIMATE STATISTICS;
analyze TABLE T5 ESTIMATE STATISTICS;
analyze TABLE T6 ESTIMATE STATISTICS;
analyze TABLE TABLEZHAO ESTIMATE STATISTICS;
analyze TABLE TFXJ ESTIMATE STATISTICS;
analyze TABLE THVL1 ESTIMATE STATISTICS;
analyze TABLE TMP1 ESTIMATE STATISTICS;
```

使用动态 SQL 语句.

```
set serveroutput on
declare
tx varchar2(500);
BEGIN
for i in (select 'analyze '||object_type||' '||
object_name||' ESTIMATE STATISTICS' as sql_text
from user_objects where object_type in('TABLE','INDEX')
ORDER BY 1)  loop
tx:=i.sql_text;
EXECUTE IMMEDIATE tx;
end loop;
END;
/
```

实验 150：收集列的统计信息

该实验的目的是收集列的统计信息
列的柱状图统计（准备环境）

```
drop table t1 purge;
create table t1 as select * from emp;
insert into t1 select * from t1;
--10000rows
alter table t1 modify (empno number(8));
update t1 set empno=rownum;
commit;
```

删除统计信息

```
BEGIN
DBMS_STATS.DELETE_TABLE_STATS (OWNNAME => 'SCOTT', TABNAME => 'T1');
END;
/
```

收集统计信息

```
BEGIN
DBMS_STATS.GATHER_table_STATS
 (OWNNAME => 'SCOTT', TABNAME => 'T1',
METHOD_OPT => 'FOR COLUMNS SIZE 100 empno');
END;
/
```

收集整个帐号的信息，包含索引的信息
取 20%的数据块

```
 execute dbms_stats.GATHER_SCHEMA_STATS('TEACH',20,TRUE,CASCADE=>TRUE);
```

查看统计信息

```
SELECT column_name, num_distinct, num_buckets, histogram
FROM USER_TAB_COL_STATISTICS
WHERE table_name = 'T1' AND column_name = 'EMPNO';

SELECT endpoint_number, endpoint_value
FROM USER_HISTOGRAMS
WHERE table_name = 'T1' and column_name = 'EMPNO'
ORDER BY endpoint_number;
```

请将列的值改为如下

```
update t1 set empno=9000 where empno<9000;
Commit;
```

再次收集，再次查看统计信息
柱状图有两种。
1。Height-Balanced 当不同的键值高于分割的桶数
2。Frequency 当不同的键值低于分割的桶数

实验 151：自动收集统计信息

该实验的目的是了解 10g 的新特性来定时收集统计信息.
自动的收集统计信息

```
SELECT * FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME = 'GATHER_STATS_JOB';
BEGIN
DBMS_SCHEDULER. ENABLE ('GATHER_STATS_JOB');
END;
/
```

```
--禁止自动的收集统计信息
BEGIN
DBMS_SCHEDULER. DISABLE ('GATHER_STATS_JOB');
END;
/
```

禁止个别表自动收集统计信息
当一个表变化比较快，自动收集信息可能不会满足高性能 SQL 的需要。

```
BEGIN
DBMS_STATS.DELETE_TABLE_STATS('OE','ORDERS');
DBMS_STATS.LOCK_TABLE_STATS('OE','ORDERS');
END;
/
```

1。将原有表的统计信息删除
2。锁定表，禁止自动分析

手工收集统计信息

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('SCOTT',DBMS_STATS.AUTO_SAMPLE_SIZE);
GATHER_INDEX_STATS                  索引
GATHER_TABLE_STATS                  表，列和索引
GATHER_SCHEMA_STATS                帐号
GATHER_DICTIONARY_STATS      字典
GATHER_DATABASE_STATS           全数据库
```

何时进行手工收集
1。如果数据是不断增加的，每周，或每月进行统计就可
2。如果是批量加载的，在加载后就应该收集
3。如果分区表的某个区域数据变化，可以单独收集该区域的统计信息，而不是收集全表的信息

影响执行计划的参数

```
OPTIMIZER_FEATURES_ENABLE=10.1.0
optimizer_mode=all_rows
PGA_AGGREGATE_TARGET
DB_FILE_MULTIBLOCK_READ_COUNT
CURSOR_SHARING
```

最优计划的选择方式
1。列出执行计划集
2。估算每个计划的代价（cpu,io,网络）
3。选择代价(cost)最小的执行计划

数据库的不同访问模式

实验 152：全表扫描的优化

该实验的目的是理解全表扫描的操作
全表扫描
将高水位以下的数据块都读一遍
DB_FILE_MULTIBLOCK_READ_COUNT 参数决定扫描的速度,该参数的值乘以块的大小应该小于操作系统的最大 io,一句话,操作系统一次 io 应该是大于等于它们的乘积.
因为范围是连续的块，所以全表扫描会连续的读，效率很高。
何时数据库使用全表扫描
1。表小。
2。索引缺少,条件判定列上没有索引.
3。使用 hint,强制使用全表扫描
4。读的数据比重大。一般超过 10%的数据要读取就会选择全表扫描.

5。并行查询,和索引是对立的,两者必须选择一个.
我们优化全表扫描的手段:
1.回收高水位线
2.DB_FILE_MULTIBLOCK_READ_COUNT 加大
3.使每个块装的数据更多,减少 pctfree
4.使用并行查询

```
SQL> conn scott/tiger
Connected.
SQL> drop table t1;

Table dropped.

SQL> create table t1 as select * from emp;

Table created.

SQL> insert into t1 select * from t1;

14 rows created.

SQL> /
```

连续的使表翻倍,达到 10 万行左右

```
57344 rows created.

SQL> commit;

Commit complete.
SQL> analyze table t1 estimate statistics;
```

收集统计信息

```
Table analyzed.

SQL> select * from t1;
```

写一个无条件的查询

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=68 Card=115182 Bytes  =3685824)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=68 Card=115182 Bytes=3685824)
```

执行计划为全表扫描,代价为 68
使用并行查询的强制

```
SQL> select /*+ full(t1) parallel(t1,4) */ * from t1;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=17 Card=115182 Bytes =3685824)
   1    0   TABLE ACCESS* (FULL) OF 'T1' (Cost=17 Card=115182 Bytes=36
```

执行计划为全表扫描,代价为 17

```
SQL> alter table t1 pctfree 0;
Table altered.
```

将每个数据块都装满

```
SQL> alter table t1 move tablespace users;

Table altered.

SQL> select * from t1;
```

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=63 Card=118034 Bytes =3777088)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=63 Card=118034 Bytes=3777088)
```
执行计划为全表扫描,代价为 63,比第一次要小.因为扫描的块少了.

```
SQL> conn / as sysdba
Connected.

SQL> alter system set db_file_multiblock_read_count=8;
```
修改参数配置,默认为 16 个块
```
System altered.
SQL> conn scott/tiger
Connected.
SQL> set autotrace traceonly explain
SQL> select * from t1;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=98 Card=118034 Bytes=3777088)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=98 Card=118034 Bytes=3777088)
```
执行计划为全表扫描,代价为 98,因为一次读的块少了,代价增加
```
SQL> conn / as sysdba
Connected.
SQL> alter system set db_file_multiblock_read_count=32;

System altered.

SQL> conn scott/tiger
Connected.
SQL> set autotrace traceonly explain
SQL> select * from t1;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=40 Card=118034 Bytes=3777088)
   1    0   TABLE ACCESS (FULL) OF 'T1' (Cost=40 Card=118034 Bytes=3777088)
```
执行计划为全表扫描,代价为 40,因为一次读的块多了,代价减少

实验 153：索引的五种使用模式

该实验的目的是深刻体会索引对数据库的巨大影响.
索引在数据库中是很重要的.没有索引的数据库是不可想象的,我们普通的表是无序的,也叫做堆表(heap table),一句话概括索引,索引是有序的结构,通过索引可以快速定位我们要找的行,避免全表扫描.索引的访问模式有五种.
1。INDEX UNIQUE SCAN 效率最高，主键或唯一索引
2。INDEX FAST FULL SCAN 读的最块，可以并行访问索引，但输出不按顺序
3。INDEX FULL SCAN 有顺序的输出，不能并行读索引
4。INDEX RANGE SCAN 给定的区间查询
5。INDEX SKIP SCAN 联合索引，不同值越少的列，越要放在前面

```
SQL> conn scott/tiger
Connected.
SQL> drop table t1 purge;

Table dropped.
```

```
SQL> create table t1 as select * from dba_objects;

Table created.

SQL> analyze table t1 compute statistics;
收集表 t1 的统计信息
Table analyzed.
SQL> create unique index i2t1 on t1(object_id);

Index created.
SQL>set autot traceonly explain
1。INDEX UNIQUE SCAN    效率最高，主键或唯一索引

SQL> select * from t1 where object_id=9999;

Execution Plan
----------------------------------------------------------
Plan hash value: 1026981322

--------------------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |     1 |    87 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T1   |     1 |    87 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | I2T1 |     1 |       |     1   (0)| 00:00:01 |
```

执行计划为唯一定位,最快.

```
2。INDEX FAST FULL SCAN  读的最块，可以并行访问索引，但输出不按顺序

SQL> select object_id from t1 ;

Execution Plan
----------------------------------------------------------
Plan hash value: 3617692013

-------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      | 49859 |   194K|   337   (1)| 00:00:07 |
|   1 |  TABLE ACCESS FULL| T1   | 49859 |   194K|   337   (1)| 00:00:07 |
```

为什么没有使用索引,而进行了全表扫描.因为 object_id 可能有 null 值.因为 null 不入普通索引.我们进行全索引的扫描就会得到错误的结果.这是全表扫描是正确的.虽然我们的的查询仅包含了索引中的值.
我们如果有非空约束就会极大的提高性能.

```
SQL> delete t1 where object_id is null;
SQL> alter table t1 modify (OBJECT_ID not null);
SQL> select object_id from t1 ;

Execution Plan
----------------------------------------------------------
Plan hash value: 2003301201

-------------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      | 49859 |   194K|    51   (2)| 00:00:02 |
|   1 |  INDEX FAST FULL SCAN| I2T1 | 49859 |   194K|    51   (2)| 00:00:02 |
```

计划仅扫描了索引,代价为 51.因为所有的行都在索引中了,使用索引不会造成错误的结果.因为我们的输出没有要求有序,所以数据库将高水位下所有的索引块都读一遍就可以了,这就叫索引的快速全扫描.

3。INDEX FULL SCAN　　有顺序的输出，不能并行读索引
SQL> select object_id from t1 **order by object_id ;**

Execution Plan

Plan hash value: 1111347323

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		49859	194K	**106** (2)	00:00:03
1	**INDEX FULL SCAN**	I2T1	49859	194K	106 (2)	00:00:03

执行计划为全扫描索引,含义是按叶子的大小顺序来读索引,因为我们要求输出是有序的.
代价为 106,高于快速全扫描,因为我们不是将高水位的块连续读,而是按照叶子的顺序读.正因为是按照叶子的顺序读,所以不能并行操作.
4。INDEX RANGE SCAN　　给定的区间查询
SQL> select * from t1 where object_id between 300 and 400;

Execution Plan

Plan hash value: 1490405106

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		93	8091	**4** (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T1	93	8091	4 (0)	00:00:01
* 2	**INDEX RANGE SCAN**	I2T1	93		2 (0)	00:00:01

当我们的索引为非唯一,或者我们的索引唯一但查询的条件为一个范围的时候数据库会选择范围定位.
代价的大小取决于你所查询行的多少.

5。INDEX SKIP SCAN　　联合索引，不同值越少的列，越要放在前面
在下一个实验**联合索引**中有详细的描述.

数据库的主键,唯一约束和外键的使用也要索引的参与.
SQL> drop table t2 purge;

Table dropped.

SQL> create table t2 as select distinct owner from dba_objects;
建立一个含有 owner 的表.
Table created.
SQL> alter table t2 add constraint pk_t2 primary key (owner);
建立主键
Table altered.

SQL> alter table t1 add constraint fk_t1 foreign key (owner)
     references t2(owner) on delete cascade;;
建立一个级联删除的外键
Table altered.

SQL>**DELETE T2** WHERE OWNER='SYS';
查看计划,我们发现
Rows　　Row Source Operation
------- ---

```
      1  DELETE  T2 (cr=726 pr=0 pw=0 time=16740731 us)
      1   INDEX UNIQUE SCAN PK_T2 (cr=1 pr=0 pw=0 time=52 us)(object id 54503)

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  DELETE  T1 (cr=725 pr=0 pw=0 time=16714571 us)
  22984   TABLE ACCESS FULL T1 (cr=690 pr=0 pw=0 time=160995 us)
```

在删除 t2 的同时,要全表扫描 t1 表,因为我们建立了外键.如果在外键上有索引,那么就很可能走索引,会极大的提高数据库的性能.

索引使用总论:
能用唯一索引，一定用**唯一**索引
能加非空，就加**非空**约束
一定要**统计**表的信息，索引的信息，柱状图的信息。
联合索引的**顺序**不同，影响索引的选择，尽量将不同值少的列放在前面
只有做到以上四点，数据库才会正确的选择执行计划。
索引是在不修改代码的情况下提高性能的重要手段.索引也是约束的维护纽带.在**外键**上最好建立索引.
参数 optimizer_index_cost_adj 定义了索引的权重,该值越大,数据库认为使用索引的成本越高,默认值为 100,如果设置为 50,那么数据库认为使用索引的代价比它计算出来的少一半,如果你设置为 1000,那么认为你使用索引的成本为计算出来的 10 倍,该值最大为 10000,最小为 1.

```
SQL> conn scott/tiger
Connected.
SQL> set autot on
SQL> alter session set optimizer_index_cost_adj  = 100;
SQL> select * from emp order by empno;
```

这句话可以走索引,也可以不走索引.默认为 100 的情况.

```
Execution Plan
----------------------------------------------------------
Plan hash value: 4170700152

--------------------------------------------------------------------------------
| Id  | Operation                   | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |    14 |   532 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |    14 |   532 |     2   (0)| 00:00:01 |
|   2 |   INDEX FULL SCAN           | PK_EMP |    14 |       |     1   (0)| 00:00:01 |
--------------------------------------------------------------------------------
```

走了索引

```
SQL> select * from emp where empno between 1 and 1000;

no rows selected


Execution Plan
----------------------------------------------------------
Plan hash value: 169057108

--------------------------------------------------------------------------------
| Id  | Operation                   | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |     1 |    38 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |    38 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | PK_EMP |     1 |       |     2   (0)| 00:00:01 |
```

我们做了一个区间的范围查询,走了索引,进行了索引的范围查询.

```
SQL> alter session set optimizer_index_cost_adj=1000;
```

```
Session altered.
```
将该参数的值改为 1000,数据库评估索引的时候认为成本很高.
```
SQL> select * from emp order by empno;

Execution Plan
----------------------------------------------------------
Plan hash value: 150391907

--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    14 |   532 |     5  (20)| 00:00:01 |
|   1 |  SORT ORDER BY     |      |    14 |   532 |     5  (20)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| EMP  |    14 |   532 |     4   (0)| 00:00:01 |
--------------------------------------------------------------------------------
```
所以改为全表扫描了.
```
SQL> select * from emp where empno between 1 and 1000;

no rows selected


Execution Plan
----------------------------------------------------------
Plan hash value: 3956160932

--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |    38 |     4   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    38 |     4   (0)| 00:00:01 |
--------------------------------------------------------------------------------
```
认为索引的成本高,改为全表扫描了.

实验 154：连接的三种模式

该实验的目的是了解表的连接模式
连接（join)
如果有主键的列连接，将带主键和唯一键约束的表放在连接的第一个位置，再考虑其它表连接
如果有外键连接，则将该表放在连接的最后。

Nested Loop Joins（嵌套循环连接）
外部表的每一行都和内部表的所有行连接。
当表的行较少的时候，数据库会选择这种连接。
提示：USE_NL(table1 table2)
```
SQL> CONN SCOTT/TIGER
Connected.
SQL> set autot traceonly explain
SQL> select ename,loc from emp,dept
  2  where emp.deptno=dept.deptno ;

Execution Plan
----------------------------------------------------------
Plan hash value: 351108634

--------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
```

```
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |    14 |   280 |     5   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |          |    14 |   280 |     5   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL          | EMP      |    14 |   126 |     4   (0)| 00:00:01 |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPT     |     1 |    11 |     1   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN         | PK_DEPT  |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Hash Joins
适用于大数据量的连接。
将两个表中较小的表的连接列建立一个 hash 表，将 hash 表放入到内存中。
什么时候用 HASH 连接?大量数据要连接,但要想使 hash 连接起到作用,必须有等值的条件.
使用 hint:USE_HASH

```
SQL> drop table t1 purge;

Table dropped.

SQL> drop table t2 purge;

Table dropped.

SQL> create table t1 as select * from dept;

Table created.

SQL> create table t2 as select * from emp;

Table created.

SQL> select ename,loc from t1,t2
  2  where t1.deptno=t2.deptno;

Execution Plan
----------------------------------------------------------
Plan hash value: 1838229974

-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    14 |   574 |     9  (12)| 00:00:01 |
|*  1 |  HASH JOIN         |      |    14 |   574 |     9  (12)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| T1   |     4 |    84 |     4   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| T2   |    14 |   280 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

排序融合连接
HASH 连接在大部分时候都比排序连接性能好。
但如果不是等值条件的时候,条件是>,>=,<,<=的时候,不能使用 hash 连接,使用排序连接和嵌套循环连接。
再有当连接的结果要排好序的时候，也可以选择排序融合连接。

```
SQL> select ename,grade from emp,salgrade
  2   where sal between LOSAL and hisal;
  Execution Plan
  ----------------------------------------------------------
     0      SELECT STATEMENT Optimizer=CHOOSE (Cost=7 Card=4 Bytes=64)
     1    0   MERGE JOIN (Cost=7 Card=4 Bytes=64)
     2    1     SORT (JOIN) (Cost=4 Card=5 Bytes=40)
     3    2       TABLE ACCESS (FULL) OF 'SALGRADE' (Cost=2 Card=5 Bytes=40)
     4    1     FILTER
     5    4       SORT (JOIN)
```

```
   6    5         TABLE ACCESS (FULL) OF 'EMP' (Cost=2 Card=14 Bytes=112)
```

实验 155：联合索引的建立

该实验的目的是使用联合索引提高性能.
联合索引
当表达式中含有多列
如果你想查的列全部在索引中，就不用访问基表了，在索引中直接取数据，减少了 io
注意列的顺序,我们一般把重复的值多的列放在前.

```
SQL> conn / as sysdba
Connected.
SQL> grant select any dictionary to scott;
```

赋予 scott 查询字典的权限

```
Grant succeeded.

SQL> conn scott/tiger
Connected.
SQL> drop table t1 purge;

Table dropped.

SQL> create table t1 as select * from dba_objects;
```

建立一个大表

```
Table created.
SQL> select count(distinct owner),count(distinct object_type),
  2  count(distinct object_name) from t1;
```

显示列的不同键值的个数

```
COUNT(DISTINCTOWNER) COUNT(DISTINCTOBJECT_TYPE) COUNT(DISTINCTOBJECT_NAME)
-------------------- -------------------------- -------------------------
                  18                         41                     29844
```

建立一个联合索引

```
SQL> create index i_l3 on t1(owner,object_type,object_name);

Index created.
SQL> set autot traceonly explain
```

启动自动跟踪

```
SQL> select owner,object_type,object_name from t1 where object_name='DBA_TABLES';

Execution Plan
----------------------------------------------------------
Plan hash value: 1231462060

--------------------------------------------------------------------------------
| Id  | Operation             | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |     8 |   752 |   175   (2)| 00:00:04 |
|*  1 |  INDEX FAST FULL SCAN | I_L3 |     8 |   752 |   175   (2)| 00:00:04 |
--------------------------------------------------------------------------------
```

为什么没有查找表?因为你要查询的列都在索引中有了,不必找表了.

```
SQL> select owner,object_type,object_name from t1 where object_type='WINDOW';

Execution Plan
----------------------------------------------------------
Plan hash value: 1231462060

--------------------------------------------------------------------------------
```

```
| Id  | Operation             | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |     8 |   752 |   175   (2)| 00:00:04 |
|*  1 |  INDEX FAST FULL SCAN | I_L3 |     8 |   752 |   175   (2)| 00:00:04 |
-------------------------------------------------------------------------------
```

现在我们做一个查询,条件为索引的第二列.也是全扫描索引.

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("OBJECT_TYPE"='WINDOW')

Note
-----
   - dynamic sampling used for this statement

SQL> analyze table t1 compute statistics for all indexed columns;
```

我们对每个列的数据分布情况进行统计.告诉数据库每列的键值的数据分布的均衡情况.

```
Table analyzed.

SQL> select owner,object_type,object_name from t1 where object_type='WINDOW';

Execution Plan
----------------------------------------------------------
Plan hash value: 3872484102

-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     2 |    74 |    20   (0)| 00:00:01 |
|*  1 |  INDEX SKIP SCAN  | I_L3 |     2 |    74 |    20   (0)| 00:00:01 |
```

执行计划为索引的跳远扫描.代价为 20,比上面小的多.因为数据库知道了 owner 的数量不多.数据库将上面的查询转化为 or 运算.请自己仔细想一下.数据库挺高!

实验 156：基于函数索引的建立

该实验的目的是使用函数索引提高查询性能.
基于函数的索引
如果条件中引用的是函数，就要建立基于函数的索引。
如果是自定义函数，则要指明返回值是确定的。
要想在 SQL 语句中使我们建立的函数索引起作用,我们还需要修改一些初始化参数.
QUERY_REWRITE_INTEGRITY = TRUSTED (9i 前需要)
QUERY_REWRITE_ENABLED = TRUE
COMPATIBLE > 8.1.0.0.0
有收集统计信息的表，函数索引才可以使用。

```
SQL> conn scott/tiger
Connected.
```

我们建立一个有确定返回值的函数

```
SQL> create or replace function f_sal
  2  (v1 in number)
  3  return number deterministic
  4  as
  5  begin
  6  if v1<1000 then return 1;
  7  elsif v1<2000 then return 2;
  8  else return 3;
```

```
  9  end if;
 10  end;
 11  /
```
建立一个基于函数的索引
```
SQL> create index i_f_sal on emp(f_sal(sal));
```
打开自动跟踪
```
SQL> set autot traceonly explain
SQL> select * from emp where f_sal(sal)=2;

Execution Plan
----------------------------------------------------------

Plan hash value: 4263848096

-------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |     1 |    38 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    38 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | I_F_SAL |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```
我们查看执行计划,我新建立的索引被使用了,函数索引在有的时候会极大的提高速度.

实验 157：位图索引的建立

该实验的目的是使用位图索引提高查询性能.
位图索引（bitmap）
列的不同键值数很少，一般少于 1000 个。不过不是绝对的,超过一千也没有问题.
Where 子句中含有多个表达式,位图索引对 and,or 的操作是绝活.位图索引包含了所有的行,即使是空行,也会在索引当中体现.所以有了位图索引,我们执行 count(*)的操作会直接读位图索引,极快.
```
SQL> conn / as sysdba
Connected.
SQL> drop table scott.t1 purge;

Table dropped.
SQL> create table scott.t1
     as select OWNER,OBJECT_NAME,OBJECT_ID,OBJECT_TYPE,CREATED from dba_objects;

Table created.建立一个大表
SQL> set autot traceonly explain
SQL> select count(*) from scott.t1;

Execution Plan
----------------------------------------------------------

Plan hash value: 3724264953

--------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
--------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |   195   (2)| 00:00:04 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 48958 |   195   (2)| 00:00:04 |
--------------------------------------------------------------------
```
执行计划为全表扫描,代价为 195

```
SQL> create bitmap index scott.i_bit1 on scott.t1(OWNER);

Index created.
```
我们再建立一个基于对象拥有者的位图索引.

```
SQL> select count(*) from scott.t1;

Execution Plan
----------------------------------------------------------
Plan hash value: 3966455870

-------------------------------------------------------------------------------
| Id  | Operation                     | Name   | Rows  | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |        |     1 |     5   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE               |        |     1 |            |          |
|   2 |   BITMAP CONVERSION COUNT     |        | 48958 |     5   (0)| 00:00:01 |
|   3 |    BITMAP INDEX FAST FULL SCAN| I_BIT1 |       |            |          |
```

执行计划为全索引扫描,代价为 5,差别是巨大的

```
SQL> create bitmap index scott.i_bit2 on scott.t1(object_type);

Index created.
```

我们再建立一个基于对象类型的位图索引.

```
SQL> select * from SCOTT.t1 where owner='SCOTT' AND OBJECT_TYPE='TABLE';

Execution Plan
----------------------------------------------------------
Plan hash value: 1363133049

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |        |    39 |  4524 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID  | T1     |    39 |  4524 |     3   (0)| 00:00:01 |
|   2 |   BITMAP CONVERSION TO ROWIDS |        |       |       |            |          |
|   3 |    BITMAP AND                 |        |       |       |            |          |
|*  4 |     BITMAP INDEX SINGLE VALUE | I_BIT1 |       |       |            |          |
|*  5 |     BITMAP INDEX SINGLE VALUE | I_BIT2 |       |       |            |          |
-------------------------------------------------------------------------------------------
```

执行计划为位图索引的 and 运算,代价为 3.相当的快.

```
SQL> select * from SCOTT.t1 where owner='SCOTT' or OBJECT_TYPE='TABLE';

Execution Plan
----------------------------------------------------------
Plan hash value: 2884896420

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |        |  1708 |   193K|   115   (0)| 00:00:03 |
|   1 |  TABLE ACCESS BY INDEX ROWID  | T1     |  1708 |   193K|   115   (0)| 00:00:03 |
|   2 |   BITMAP CONVERSION TO ROWIDS |        |       |       |            |          |
|   3 |    BITMAP OR                  |        |       |       |            |          |
|*  4 |     BITMAP INDEX SINGLE VALUE | I_BIT1 |       |       |            |          |
|*  5 |     BITMAP INDEX SINGLE VALUE | I_BIT2 |       |       |            |          |
```

执行计划为位图索引的 or 运算,代价为 115.不理想.因为是表的对象太多了.

```
SQL> select * from SCOTT.t1;
```

我要执行全表扫描

```
Execution Plan
----------------------------------------------------------

Plan hash value: 3617692013

--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      | 48958 |  5546K|   196   (2)| 00:00:04 |
|   1 |  TABLE ACCESS FULL| T1   | 48958 |  5546K|   196   (2)| 00:00:04 |
```

执行计划为全表扫描,代价为 196.我们看到上面的 115 还是比全表扫描来的快.
位图索引存储的是 rowid 的区间和该区间内的 rowid 的位图,没有真实的存储每个 rowid.所以必须含有每一行,而且存储的空间很小,比正常的索引小的多.正是位图索引很小,所以读取位图索引的速度就快,需要的内存就少.处理起来更快.位图索引的唯一缺点就是维护的代价高,我们更改一行的时候,需要重新建立两个位图,但也不想你想象的那么贵.实际工作中还是很快的.
我们再建立一个普通的索引来比较索引的大小.

```
SQL> create index scott.i_3 on scott.t1(OBJECT_NAME);

Index created.

SQL> set autot off
SQL> select segment_name,segment_TYPE,blocks from dba_segments
  2  where owner='SCOTT' AND SEGMENT_NAME IN('T1','I_3','I_BIT1','I_BIT2');

SEGMENT_NAME         SEGMENT_TYPE     BLOCKS
-------------------- ------------ ----------
T1                   TABLE               512
I_BIT1               INDEX                 8
I_BIT2               INDEX                 8
I_3                  INDEX               384
```

表最大,512 个块;普通索引也很大 384 个块;位图索引很小,8 个块.
总结一下:位图索引有巨大的性能的优势,但在变化比较频繁的表中维护的开销还是很大的.越大的表建立位图索引越好.

实验 158：反键索引的建立

该实验的目的是理解什么是反键索引,何时建立反键索引.
索引是有序的组织,所以相近的数据基本会存在有同一个叶子中.我们在 rac 的环境中,多个实例同时维护一个叶子就会产生竞争.因为 rac 的内存管理有仲裁,有更复杂的锁管理.当不同的实例维护一个块的时候有较大的开销.怎么把连续的数据让它不连续呢?反键索引. 反键索引是把建立索引的键值前后颠倒后在编排入索引.比如 8001,8002,8005,8006 在普通索引中会在一个叶子中出现,但反键索引是编排的 1008,2008,5008,6008 进入的索引.索引是有序的,所以上述的值不可能在同一个叶子.rac 中不同的实例会维护不同的叶子,没有了竞争.但反键索引也有自身的问题,那就是在范围查询的时候不会使用索引.

```
SQL> create index scott.it4 on scott.emp(sal) reverse;

Index created.
```

建立一个反键索引

```
SQL> conn scott/tiger
Connected.
SQL> set autot traceonly explain
SQL> select * from emp where sal between 1000 and 2000;
```

做一个范围的查询

```
Execution Plan
----------------------------------------------------------

Plan hash value: 3956160932
```

```
-----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     5 |   190 |     4   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     5 |   190 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------
```

执行计划为全表扫描,代价为 4.

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("SAL"<=2000 AND "SAL">=1000)

SQL> drop index it4;
```

删除反键索引

```
Index dropped.

SQL> create index scott.it4 on scott.emp(sal);
```

建立普通索引

```
Index created.

SQL> select * from emp where sal between 1000 and 2000;
```

做一个范围的查询

```
Execution Plan
----------------------------------------------------------
Plan hash value: 3868271256

-------------------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |     5 |   190 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP  |     5 |   190 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IT4  |     5 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

执行计划为索引的范围扫描,代价为 2.
总结:反键索引的目的为了避免 rac 的块竞争.尤其在顺序插入的时候.

实验 159：索引组织表的建立

该实验的目的是建立索引组织表来提高数据库性能.
索引组织表（iot）

```
connect SCOTT/TIGER
create table sales
 (office_cd number(3)
 ,qtr_end date
 ,revenue number(10,2)
 ,review varchar2(1000)
 ,constraint sales_pk
 PRIMARY KEY (office_cd,qtr_end)
 )
 ORGANIZATION INDEX tablespace USERS
 PCTTHRESHOLD 20
 INCLUDING revenue
 OVERFLOW TABLESPACE TOOLS;
```

验证索引组织表

```
SELECT * FROM DBA_SEGMENTS WHERE
SEGMENT_NAME IN('SALES_PK','SYS_IOT_OVER_39247') AND OWNER='SCOTT';
```

```
select table_name,tablespace_name,iot_name,iot_type
 from DBA_TABLES WHERE IOT_TYPE LIKE '%IOT%' AND OWNER='SCOTT';

select index_name,index_type,tablespace_name,table_name
 from DBA_INDEXES WHERE OWNER='SCOTT' AND TABLE_NAME='SALES';
```

实验 160：cluster 表的建立

该实验的目的是 Cluster 的建立
集簇的优点有二:节约一定的存储空间,联合查询的时候会提高效率.

```
CONN SCOTT/TIGER
CREATE CLUSTER clu1 (deptno NUMBER(2));
CREATE INDEX idx_clu1 ON CLUSTER clu1;
CREATE TABLE emp_c1
   CLUSTER clu1 (deptno)
   AS SELECT * FROM emp ;
CREATE TABLE dept_c1
   CLUSTER clu1 (deptno)
   AS SELECT * FROM dept;
SELECT * FROM DBA_EXTENTS where SEGMENT_NAME like '%CLU1';
SELECT * FROM DBA_TABLES WHERE TABLE_NAME like 'DEPT_%';
```

实验 161：物化视图的建立

该实验的目的是建立何维护物化视图.
物化视图
要有存储空间
用来保存中间结果
使 SQL 语句查询重写

建立物化视图

```
SQL> conn / as sysdba
Connected.
SQL> drop user scott cascade;

User dropped.

SQL> @%oracle_home%\rdbms\admin\scott
SQL> Conn / as sysdba
grant CREATE MATERIALIZED VIEW to scott;
grant execute on dbms_mview to scott;
--建立物化视图
Conn scott/tiger
create  MATERIALIZED VIEW mv1 as select * from emp;
--查看物化视图信息
SQL> select MVIEW_NAME,QUERY from user_mviews;

MVIEW_NAME
------------------------------
QUERY
--------------------------------------------------------------------------------
MV1
```

SELECT "EMP"."EMPNO" "EMPNO","EMP"."ENAME" "ENAME","EMP"."JOB" "JOB","EMP"."MGR"
因为 query 数据类型为 long,默认显示前 80 个字节。下面语句设为显示前 1000 个字符。
SQL> set long 1000
SQL> select MVIEW_NAME,QUERY from user_mviews;
再次查询，得到全的视图定义
MVIEW_NAME

QUERY

MV1
SELECT "EMP"."EMPNO" "EMPNO","EMP"."ENAME" "ENAME","EMP"."JOB" "JOB","EMP"."MGR"
 "MGR","EMP"."HIREDATE" "HIREDATE","EMP"."SAL" "SAL","EMP"."COMM" "COMM","EMP"."
DEPTNO" "DEPTNO" FROM "EMP" "EMP"

--刷新物化视图
 execute dbms_mview.refresh('mv1','complete');

建立预定义的物化视图，避免数据类型的不同
可以在表上建立索引
conn scott/tiger
drop table mv2 purge;
create table mv2 as select * from emp where 0=9;

create MATERIALIZED VIEW mv2
 ON PREBUILT TABLE
as select * from emp;

刷新的模式
完全
快速（需要日志）

COMMIT 刷新模式
CREATE SNAPSHOT LOG ON emp;
 CREATE MATERIALIZED VIEW mv_emp
  REFRESH FAST
  ON COMMIT
  AS select * from emp;

指定刷新的时间间隔
  CREATE MATERIALIZED VIEW mv_emp2
  REFRESH FAST
  START WITH SYSDATE NEXT SYSDATE + 1/24/60
  WITH PRIMARY KEY
  AS select * from emp;
Select * from user_jobs;
execute dbms_mview.refresh('mv_emp2','complete');
 execute dbms_mview.refresh('mv_emp2','fast');

show parameter global_names
False 的意思为：数据库连接可以和远程数据库的名称不同。
true 的意思为：数据库连接必须和远程数据库的名称相同。

--建立数据库连接
CREATE DATABASE LINK ####
  CONNECT TO scott IDENTIFIED BY tiger USING ‘网络连接标识符';
--测试数据库连接
Select * from emp@####;
建立远程的物化视图。测试刷新

实验 162：查询重写

该实验的目的是使用物化视图来提高查寻性能.
查询重写的含义是，如果数据库知道有一个已经查询好的结果，它会使用查询好的结果，而不是重新查询。但这需要条件。

```
SQL> conn scott/tiger
Connected.
SQL> DROP MATERIALIZED VIEW mv1;
DROP MATERIALIZED VIEW mv1
*
ERROR at line 1:
ORA-12003: materialized view "SCOTT"."MV1" does not exist
```

该语句是预防视图的名称重复，mv1 物化视图如果不存在报错，如果存在彻底删除。

```
SQL> create MATERIALIZED VIEW SCOTT.mv1
  2             tablespace users
  3             BUILD IMMEDIATE
  4             REFRESH COMPLETE
  5             ENABLE QUERY REWRITE
  6  AS select e.ename,d.loc from scott.emp e,scott.dept d where e.deptno=d.deptno;

ERROR at line 6:
ORA-01031: insufficient privileges
```

权限不够

```
SQL> conn / as sysdba
Connected.
SQL> grant create  MATERIALIZED VIEW to scott;
SQL> grant EXECUTE ANY PROCEDURE to scott;

Grant succeeded.
SQL> conn scott/tiger
Connected.
SQL> create MATERIALIZED VIEW SCOTT.mv1
  2             tablespace users
  3             BUILD IMMEDIATE
  4             REFRESH COMPLETE
  5             ENABLE QUERY REWRITE
  6  AS select e.ename,d.loc from scott.emp e,scott.dept d where e.deptno=d.deptno;

Materialized view created.
SQL> set autot traceonly explain
```

打开自动跟追，指看执行计划

```
SQL> select e.ename,d.loc from scott.emp e,scott.dept d where e.deptno=d.deptno;

Execution Plan
----------------------------------------------------------
Plan hash value: 2958490228

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |    12 |   180 |     4   (0)| 00:00:01 |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| MV1  |    12 |   180 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

执行计划走了捷径

```
SQL> ALTER SESSION SET query_rewrite_enabled   =false;
```

```
Session altered.
```
禁止查询重写
```
SQL> select e.ename,d.loc from scott.emp e,scott.dept d where e.deptno=d.deptno;

Execution Plan
----------------------------------------------------------
Plan hash value: 351108634

--------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |    12 |   240 |     5   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                 |         |    12 |   240 |     5   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL           | EMP     |    12 |   108 |     4   (0)| 00:00:01 |
|   3 |   TABLE ACCESS BY INDEX ROWID | DEPT    |     1 |    11 |     1   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN          | PK_DEPT |     1 |       |     0   (0)| 00:00:01 |
--------------------------------------------------------------------------------------
```
数据库就会跟没有视图一样，按照语句的本意进行查询。
查询重写一般在数据仓库中使用较多。

实验 163：最后的 sql 优化办法，使用 hints

该实验的目的是使用提示来提高数据库性能.
Hints 是我们优化的最后一个手段。书写时一定要跟在第一个单词后面。作用是强制该语句以我们指定的方式运行，作用范围是当前语句，对后面的语句不影响。
提示的写法有两种：/*+ 提示 */, --+ 提示
```
SQL> conn scott/tiger
Connected.
SQL> set autot traceonly explain
```
打开自动跟追，只看执行计划
使用提示，强制使用 hash 连接
```
SQL> select --+ use_hash(e d)
  2  ename,loc
  3  from emp e ,dept d
  4  where e.deptno=d.deptno
  5  and e.deptno=d.deptno;

Execution Plan
----------------------------------------------------------
Plan hash value: 615168685

------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    12 |   240 |     9  (12)| 00:00:01 |
|*  1 |  HASH JOIN         |      |    12 |   240 |     9  (12)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPT |     4 |    44 |     4   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| EMP  |    12 |   108 |     4   (0)| 00:00:01 |
------------------------------------------------------------------------------
```
使用提示，强制使用 merge 连接
```
SQL> select --+ use_merge(e d)
  2  ename,loc
  3  from emp e ,dept d
  4  where e.deptno=d.deptno;

Execution Plan
----------------------------------------------------------
```

```
Plan hash value: 844388907

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |    12 |   240 |     7  (15)| 00:00:01 |
|   1 |  MERGE JOIN                   |         |    12 |   240 |     7  (15)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     4 |    44 |     2   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN            | PK_DEPT |     4 |       |     1   (0)| 00:00:01 |
|*  4 |   SORT JOIN                   |         |    12 |   108 |     5  (20)| 00:00:01 |
|   5 |    TABLE ACCESS FULL          | EMP     |    12 |   108 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

使用提示，强制使用 nest loop 连接

```
SQL> select --+ use_nl(e d)
  2  ename,loc
  3  from emp e ,dept d
  4  where e.deptno=d.deptno;

Execution Plan
----------------------------------------------------------
Plan hash value: 351108634

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |    12 |   240 |     5   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                 |         |    12 |   240 |     5   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL           | EMP     |    12 |   108 |     4   (0)| 00:00:01 |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    11 |     1   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN          | PK_DEPT |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

强制使用全表扫描而不使用主键

```
SQL> select /*+ full(emp) */ * from emp order by 1;

Execution Plan
----------------------------------------------------------
Plan hash value: 150391907

------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    12 |   444 |     5  (20)| 00:00:01 |
|   1 |  SORT ORDER BY     |      |    12 |   444 |     5  (20)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| EMP  |    12 |   444 |     4   (0)| 00:00:01 |
------------------------------------------------------------------------------
```

强制使用主键而不使用全表扫描

```
SQL> select /*+ index(emp pk_emp) */ * from emp;

Execution Plan
----------------------------------------------------------
Plan hash value: 4170700152

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |        |    12 |   444 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |    12 |   444 |     2   (0)| 00:00:01 |
|   2 |   INDEX FULL SCAN             | PK_EMP |    12 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

强制使用并行查询，提高全表扫描的效率

```
SQL> select /*+ full(emp) parallel(emp,4) */ * from emp;

Execution Plan
----------------------------------------------------------
Plan hash value: 2873591275

-------------------------------------------------------------------------------------------------

| Id  | Operation     | Name     | Rows  | Bytes | Cost (%CPU)| Time     |  TQ  |IN-OUT| PQ Distrib

-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |     |    12 |   444 |     2   (0)| 00:00:01 |      |      |      |
|   1 |  PX COORDINATOR      |     |       |       |            |          |      |      |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |12|444 | 2 (0)| 00:00:01 |  Q1,00 | P->S | QC (RAND)
|   3 |    PX BLOCK ITERATOR |  |  12 |   444 |  2   (0)| 00:00:01 |  Q1,00 | PCWC |       |
|   4 |     TABLE ACCESS FULL| EMP  |  12 | 444 |  2 (0)| 00:00:01 |  Q1,00 | PCWP |       |
```

我们的第一个语句为 select * from emp;介绍大家使用简单的 sql 语句，我们的最后一个语句也是这句话，但你对这句话的理解肯定不一样了，你现在更多的考虑是如何写一个高效的 sql 语句，该语句是如何运行的，它需要多少资源，如果你理解了，我的目的就达到了，你已经 oracle 数据库入门了，通过 163 个小实验，你理解了 oracle 数据库的原理。想自我提高就很快了。谢谢你读我的书。以 select * from emp;开始，又以 select * from emp;结束！

不知何处来，焉知去何处。

周而又复始，似水无痕踪。

附录:

数据库的健康检查

检查日期：2007/09/297
数据库已经启动的时间：2007/04/25
主机名称：ST-HP-02
操作系统：hp-ux
应用类型：联机交易
备份软件：无
数据库版本：8.1.7.4.0
数据库的补丁：最高的 4 号补丁
数据库的运行方式：单机加磁盘阵列，另外一台主机备用
数据文件的类型：文件系统
字符集：ZHS16CGB231280/US7ASCII
归档模式：归档模式
Sid 名称：oramain
数据库名称：oramain
服务名称： oramain
Oracle_home: /u01/app/oracle/product/8.1.7
Udump: /u01/app/oracle/admin/oramain/udump
Bdump: /u01/app/oracle/admin/oramain/bdump
Cdump: /u01/app/oracle/admin/oramain/cdump
数据库备份策略：EXP 逻辑备份+物理备份

控制文件的位置：

```
select * from V$controlfile;
NAME
----------------------
/oramain/control01.ctl
/oramain/control02.ctl
```

控制文件的空间使用情况：

```
select TYPE, RECORDS_TOTAL,RECORDS_USED from V$controlfile_record_section;
TYPE                 RECORDS_TOTAL RECORDS_USED
-------------------- ------------- ------------
DATABASE                         1            1
CKPT PROGRESS                   32            0
REDO THREAD                     32            1
REDO LOG                        64            4
DATAFILE                       254           12
FILENAME                       383           16
TABLESPACE                     254            6
RESERVED1                      254            0
RESERVED2                        1            0
LOG HISTORY                   3635         2085
OFFLINE RANGE                  292            0
ARCHIVED LOG                  3221          914
BACKUP SET                     409            0
BACKUP PIECE                   511            0
BACKUP DATAFILE                564            0
BACKUP REDOLOG                 215            0
DATAFILE COPY                  520            3
BACKUP CORRUPTION              371            0
COPY CORRUPTION                409            0
DELETED OBJECT                3272            0
```

```
PROXY COPY                     652            0
RESERVED4                    16360            0
```

日志文件的位置：

```
select group#,member from V$logfile order by 1;
GROUP# MEMBER
------ --------------------
     1 /oramain/redo01.log
     2 /oramain/redo02.log
     3 /oramain/redo03.log
     4 /oramain/redo04.log
```

日志文件的大小：100m

临时文件的位置和大小：

```
select * from V$tempfile;
```

使用的是数据文件做临时文件使用

TEMP 为生产用户的临时表空间。

数据文件的空间分配情况：

```
col ts       for a10;
col EXM      for a10;
col ATPYE    for a10;
col CONTENTS for a10;
select TABLESPACE_NAME ts,STATUS,CONTENTS,EXTENT_MANAGEMENT exm,
ALLOCATION_TYPE atpye
from dba_tablespaces order by 3;
TS         STATUS     CONTENTS   EXM        ATPYE
---------- ---------- ---------- ---------- -----
SYSTEM     ONLINE     PERMANENT  DICTIONARY USER
RBS        ONLINE     PERMANENT  DICTIONARY USER
USERS01    ONLINE     PERMANENT  DICTIONARY USER
SPSTAT     ONLINE     PERMANENT  DICTIONARY USER
INDX       ONLINE     PERMANENT  DICTIONARY USER
TEMP       ONLINE     TEMPORARY  DICTIONARY USER
```

数据库数据文件的分布

```
select tablespace_name,file_name,ceil(BYTES/1024/1024) tmb
from dba_data_files order by 1,2;
TABLESPACE_NAME FILE_NAME                                      TMB
--------------- ------------------------------------ ----------
INDX            /oramain/indx.dbf                             500
RBS             /oramain/rbs.dbf                              300
RBS             /oramain/rbs1.dbf                             504
SPSTAT          /oramain/spstat.dbf                           300
SYSTEM          /oramain/system01.dbf                         275
TEMP            /oramain/tmp.dbf                             1000
USERS01         /oramain/USER.DBF                            2000
USERS01         /oramain/USER_02.dbf                         2000
USERS01         /oramain/USER_03.dbf                         2000
USERS01         /oramain/USER_04.dbf                         2000
USERS01         /oramain/user_05.dbf                         2000
USERS01         /oramain/user_06.dbf                         2000
```

表空间的总容量和总空闲：

```
select t.tablespace_name,tmb ,fmb from
(select tablespace_name,round(sum(bytes/1024/1024)) fmb from
dba_free_space group by tablespace_name) f,
(select tablespace_name,round(sum(bytes/1024/1024)) tmb from
```

```
dba_data_files group by tablespace_name) t
where t.tablespace_name=f.tablespace_name order by tmb;
TABLESPACE_NAME         TMB        FMB
--------------- ---------- ----------
SYSTEM                  275        228
SPSTAT                  300         98
INDX                    500        500
RBS                     804        106
TEMP                   1000         68
USERS01               12000       6358
```

回退段的信息：

```
USN NAME
--- ------
  0 SYSTEM
  2 R01
  3 R02
  4 R03
  5 R04
```

大表的信息：

```
select SEGMENT_NAME,TABLESPACE_NAME,BLOCKS,EXTENTS
from dba_segments
where (EXTENTS>50 or BLOCKS>10000) and
SEGMENT_TYPE='TABLE' AND tablespace_name='USERS01'
order by BLOCKS desc;
SEGMENT_NAME          TABLESPACE_NAME      BLOCKS     EXTENTS
-------------------- --------------- ---------- ----------
TBMIMAGE              USERS01              87578        2526
TBALTLEND             USERS01              66089        1918
TBMLTBOOKSTORE        USERS01              20895         597
TBAGNMARCMAP          USERS01               8925         255
TBAGNLTMARC           USERS01               8925         255
TBMACCOUNTRECORD      USERS01               7432         215
TBMREADER             USERS01               3989         115
TBGREADERDISOBEY      USERS01               3566         103
TBMREADERCARD         USERS01               1971          57
```

大索引的信息：

```
select SEGMENT_NAME,TABLESPACE_NAME,BLOCKS,EXTENTS
from dba_segments
where (EXTENTS>50 or BLOCKS>10000) and
SEGMENT_TYPE='INDEX' AND tablespace_name='USERS01'
order by BLOCKS desc;
SEGMENT_NAME                  TABLESPACE_NAME      BLOCKS     EXTENTS
----------------------------- --------------- ---------- ----------
IX_TBALTLEND                  USERS01              33684         975
PK_TBMLTBOOKSTORE             USERS01              18305         523
IX_TBALTLEND_READERID         USERS01              17966         520
IX_TBALTLEND_LENDTM           USERS01              16860         489
IX_TBALTLEND_RETURNTIME       USERS01              15559         452
PK_TBALTLEND                  USERS01              10522         305
IX_TBMLTBOOKSTORE_MARCID      USERS01               9205         263
IX_TBAGNMARCMAP_INDEX         USERS01               4100         118
IND_TCR_READERID              USERS01               3921         114
PK_TBAGNMARCMAP               USERS01               3885         111
IX_TBMACCOUNTRECORD_DATE      USERS01               2802          81
```

PK_TBMACCOUNTRECORD USERS01 2283 66

数据库的统计：2007/04/25---2007/09/29 时间段的数据库信息

```
select NAME,value from  v$sysstat where value>1000 order by 2;
NAME                                                             VALUE
---------------------------------------------------------------- ----------
transaction rollbacks                                              1264
cleanouts and rollbacks - consistent read                          1327
gets

current blocks converted for CR                                    1405
enqueue timeouts                                                   1829
write clones created in foreground                                 2103
sorts (disk)                                                       4386
redo buffer allocation retries                                     5436
parse count (hard)                                                 6055
pinned buffers inspected                                          13981
table scans (long tables)                                         15061
DBWR transaction table writes                                     20633
cleanouts only - consistent read gets                             20961
summed dirty queue length                                         21030
immediate (CR) block cleanout applications                        22288
leaf node splits                                                  24047
cursor authentications                                            27673
switch current to new buffer                                      28453
CR blocks created                                                 36244
rollbacks only - consistent read gets                             36329
data blocks consistent reads - undo record                       141384
s applied

consistent changes                                               141864
calls to kcmgcs                                                  201689
DBWR undo block writes                                           372771
dirty buffers inspected                                          486523
free buffer inspected                                            501755
cluster key scans                                                508055
DBWR lru scans                                                   627587
DBWR make free requests                                          652720
rollback changes - undo records applied                         1000038
immediate (CURRENT) block cleanout applica                      1262174
tions

sorts (memory)                                                  1394998
physical writes direct                                          1978249
cluster key scan block gets                                     2121803
logons cumulative                                               2763157
total file opens                                                2912314
table scans (short tables)                                      4696847
DBWR checkpoint buffers written                                 7446778
physical writes non checkpoint                                  8307079
user commits                                                    9331378
redo synch writes                                               9500855
redo writes                                                     9517241
calls to kcmgas                                                 9539161
physical writes                                                10942922
DBWR free buffers found                                        12251082
messages sent                                                  12429346
messages received                                              12429347
```

```
DBWR summed scan depth                          13283198
DBWR buffers scanned                            13283198
background timeouts                             14113752
prefetched blocks                               20095713
redo blocks written                             21450261
deferred (CURRENT) block cleanout applicat      26275217
ions

enqueue releases                                31099737
enqueue requests                                31101607
commit cleanouts successfully completed         35046898
commit cleanouts                                35047516
physical reads direct                           38113965
execute count                                   45515660
hot buffers moved to head of LRU                48477582
redo wastage                                    49641056
redo entries                                    51103010
parse count (total)                             63806469
opened cursors cumulative                       64682731
calls to get snapshot scn: kcmgss               65004624
table scan blocks gotten                        66382256
SQL*Net roundtrips to/from client               67922207
free buffer requested                           72702076
recursive calls                                 90135086
db block changes                               100969932
user calls                                     109957799
physical reads                                 110353676
db block gets                                  123273492
rows fetched via callback                      163044741
table fetch continued row                      182221810
sorts (rows)                                   358894833
no work - consistent read gets                1013310407
buffer is not pinned count                    1452990699
consistent gets                               2004000946
session logical reads                         2127274160
table fetch by rowid                          7020686779
table scan rows gotten                        7758737146
bytes received via SQL*Net from client        9328862178
buffer is pinned count                        1.3691E+10
redo size                                     1.7275E+10
session uga memory                            1.8702E+10
bytes sent via SQL*Net to client              9.0697E+10
session uga memory max                        1.1885E+11
session pga memory                            4.3680E+11
session pga memory max                        4.3732E+11
```

数据库前几位的等待事件：2007/04/25---2007/09/29 时间段的事件

```
select event,TOTAL_WAITS from V$system_event where TOTAL_WAITS>1000 order by 2;
EVENT                          TOTAL_WAITS
------------------------------ -----------
file identify                         1073
LGWR wait for redo copy               2632
log buffer space                      5200
SQL*Net break/reset to client         6075
control file sequential read          8298
latch free                           37465
enqueue                              42448
smon timer                           44011
```

```
SQL*Net more data from client       53946
buffer busy waits                  330252
log file sequential read           334831
direct path write                  470155
db file parallel write            1557128
file open                         2912326
control file parallel write       4387867
pmon timer                        4389291
db file scattered read            6318594
log file sync                     9496888
log file parallel write           9517255
rdbms ipc message                24726748
SQL*Net more data to client      35558976
direct path read                 37353876
db file sequential read          45825431
SQL*Net message from client      70923944
SQL*Net message to client        70924017
```

数据库内的运行量大的 SQL 语句：

```
select max(EXECUTIONS),max(DISK_READS),max(BUFFER_GETS), max(ROWS_PROCESSED) from v$sqlarea;
MAX(EXECUTIONS) MAX(DISK_READS) MAX(BUFFER_GETS) MAX(ROWS_PROCESSED)
--------------- --------------- ---------------- -------------------
        2409607        16152692        206485289           116785883
```

运行次数多的语句

```
SQL> select sql_text ,EXECUTIONS from v$sqlarea where EXECUTIONS>1409607    order by 2;

SQL_TEXT
------------------------------------------------------------------------
EXECUTIONS
----------
SELECT MainMarcID FROM tbAGNMarcMap WHERE SubMarcID=:"SYS_B_0" AND MemberUnitID=:"SYS_B_1"
   2164859

UPDATE                                          tbALTLend                                               SET
ReturnTime=TO_DATE(:"SYS_B_0",:"SYS_B_1"),ISReturn=:"SYS_B_2",ReturnOPID=:"SYS_
B_3",ReturnMember=:"SYS_B_4",ReturnStation=:"SYS_B_5" WHERE ID=:"SYS_B_6"
   2306078

SELECT Debt FROM tbMReaderAccount WHERE ReaderID=:"SYS_B_0"
   2306488

UPDATE tbMreader SET CurLend=CurLend-:"SYS_B_0" WHERE CurLend>:"SYS_B_1" AND ID=:"SYS_B_2"
   2306497

UPDATE TBMREADER SET CURLEND=CURLEND + 1  WHERE ID = :b1
   2357357

UPDATE tbMLTBookStore SET LTStatus=:"SYS_B_0" WHERE Barcode=:"SYS_B_1"
   2357460

DELETE FROM TBALTLEND WHERE BARCODE = :b1  AND ISRETURN = 0  AND ID != :b2
   2357658

SELECT SEQ_tbALTLend_ID.NEXTVAL FROM DUAL
   2357800
```

```
SELECT
e.Barcode,d.Name,b.Title,a.LendTM,a.DueTM,a.ID,a.ReaderID,a.LTType,a.LendMember,c.Place FROM
tbALTLend a,tbAGNLTMarc b,tbMLTBookstore c,tbMreader d,tbMreaderCard e WHERE
a.Barcode=c.barcode AND
 c.Marcid=b.marcid AND a.ReaderID=d.ID AND d.CardCode=e.ID AND IsReturn=:"SYS_B_0" AND
a.Barcode=:"S
YS_B_1"
   2409627
```

处理行数多的 sql 语句

```
select sql_text ,ROWS_PROCESSED from v$sqlarea where
ROWS_PROCESSED>(select max(ROWS_PROCESSED)/10 from v$sqlarea) order by 2;

select
ID,READERID,BARCODE,LENDTM,DUETM,PRESSDATE,PRESSTIME,RENEWTIME,LTTYPE,LENDMEMBER,LENDOPID,RE
T
URNTIME,RETURNMEMBER,RETURNOPID,ISRETURN,LENDSTATION,RETURNSTATION,LENDID from
"XA_USER"."TBALTLEND"
      15152841

select privilege#,level from sysauth$ connect by grantee#=prior privilege# and privilege#>0 start
wi
th (grantee#=:1 or grantee#=1) and privilege#>0
      17053451

SELECT /*+NESTED_TABLE_GET_REFS+*/ "XA_USER"."TBALTLEND".* FROM "XA_USER"."TBALTLEND"
     116785883
/*+NESTED_TABLE_GET_REFS+*/代表 exp/imp 的语法,和我们的应用没有关系.
```

Io 多的语句

```
select sql_text ,DISK_READS from v$sqlarea where DISK_READS>6152692  order by 2;

SELECT
e.Barcode,d.Name,b.Title,a.LendTM,a.DueTM,a.ID,a.ReaderID,a.LTType,a.LendMember,c.Place FROM
tbALTLend a,tbAGNLTMarc b,tbMLTBookstore c,tbMreader d,tbMreaderCard e WHERE
a.Barcode=c.barcode AND
 c.Marcid=b.marcid AND a.ReaderID=d.ID AND d.CardCode=e.ID AND IsReturn=:"SYS_B_0" AND
a.Barcode=:"S
YS_B_1"
   8403619

select b.memo,count(*) from tbMreader a,tbcGreadercardtype b,tbMreadercard c where
c.LibcardType=b.
id and a.CardCode=c.ID and c.MemberUnitID=:"SYS_B_0" and a.OpID=:"SYS_B_1" and (RegisterDate
between
 TO_DATE(:"SYS_B_2",:"SYS_B_3") and TO_DATE(:"SYS_B_4",:"SYS_B_5") ) group by b.Memo
   9018231

select b.memo,count(*) from tbMreader a,tbcGreadercardtype b,tbMreadercard c where
c.LibcardType=b.
id and a.CardCode=c.ID and c.MemberUnitID=:"SYS_B_0" and (RegisterDate between
TO_DATE(:"SYS_B_1",:"
SYS_B_2") and TO_DATE(:"SYS_B_3",:"SYS_B_4") ) group by b.Memo
  12260412

select Name,tbMreadercard.BarCode,Reason,count(*),tbGreaderdisobey.OpID from
tbGreaderdisobey,tbMrea
```

```
der,tbMreadercard where (tbMreaderCard.MemberUnitID=:"SYS_B_0" or tbMreaderCard.MemberUnitID
in(sele
ct ID from tbMAllMember where SuperiorID=:"SYS_B_1")) and
tbMreader.ID=tbGreaderdisobey.ReaderID an
d tbMreader.CardCode=tbMreadercard.ID and (Status=:"SYS_B_2" or Status=:"SYS_B_3" or
Status=:"SYS_B_
4") and DisobeyDate between TO_DATE(:"SYS_B_5",:"SYS_B_6") and TO_DATE(:"SYS_B_7",:"SYS_B_8")
group
by Name,tbMreadercard.BarCode,Reason,tbGreaderdisobey.OpID
  16152692
```

索引的使用：有收集

数据库统计信息的收集：有

```
select table_name,num_rows from dba_tables where owner='XA_USER' order by 2;
```

TABLE_NAME	NUM_ROWS
TBAGNTTINDEX	0
TBAGNTTMARC	0
TBAGNTT_AUTHORINDEX	0
TBAGNTT_LANGUAGEINDEX	0
TBAGNTT_ISBNINDEX	0
TBAGNTT_CLASSINDEX	0
TBAGNTT_PUBLISHERINDEX	0
TBAGNTT_TOPICINDEX	0
TBAGNTT_TITLEINDEX	0
TBUSERRIGHT	0
TBCGREADERHEADSHIP	1
TBCGREADERTYPE	1
TBCGREADERLEVEL	1
TBCYYMEMBER	1
TBUSER	1
TBLENDCHECK	1
TBALTPREENGAGEFORMA	3
TBCALTPREENGAGEMEMBER	3
TBALTPREENGAGE	4
TBCGREADERTITLE	4
TBCALTPARAM	5
TBCGREADERIDTYPE	5
TBALTPREENGAGEFORMB	7
TBCGREADERDEGREE	8
PLAN_TABLE	9
TBCGREADERWORK	10
TBALTDISTRIBUTE	11
TBCGPAYITEM	12
TBCGCHARGEACTION	13
TBCGCHARGEACTIONDETAIL	13
TBCGREADERCARDTYPE	13
TBALTDISTRIBUTEFORMA	14
TBALTPREENGAGEHISTORY	16
TBCDISTRIBUTER	23
TBAGNTTINDEXPATH	48
TBALTDISTRIBUTEFORMB	63
TBALTDISTRIBUTEHISTORY	66
TBCSTOREPLACE	180
TBMALLMEMBER	300
TEMPCARD	340
TBMARCCHECK	430
TBALTDELIVERDETAIL	450
TBMREADERCHECK	1330

TBALTDELIVER	2070
TBALTDELIVERDETAILHISTORY	3560
T1	4570
TBRICHCHECK	7100
TBGCARDRECORD	43910
TBMIMAGE	332270
TBMREADERACCOUNT	345310
TBMREADER	347210
TBMREADERCARD	398760
TBGREADERDISOBEY	1344320
TBAGNLTMARC	1497200
TBMACCOUNTRECORD	2024760
TBAGNMARCMAP	2150710
TBMLTBOOKSTORE	5758690
TBALTLEND	6311920

数据库的参数设置：

```
select NAME,VALUE from v$parameter where value is not null order by 1;
```

NAME	VALUE
07_DICTIONARY_ACCESSIBILITY	TRUE
always_anti_join	NESTED_LOOPS
always_semi_join	STANDARD
aq_tm_processes	0
audit_file_dest	?/rdbms/audit
audit_trail	NONE
background_core_dump	partial
background_dump_dest	/u01/app/oracle/admin/oramain/bdump
backup_tape_io_slaves	FALSE
bitmap_merge_area_size	1048576
blank_trimming	FALSE
commit_point_strength	1
compatible	8.1.6
control_file_record_keep_time	7
control_files	/oramain/control01.ctl, /oramain/contro 02.ctl
core_dump_dest	/u01/app/oracle/admin/oramain/cdump
cpu_count	4
create_bitmap_area_size	8388608
cursor_sharing	force
cursor_space_for_time	FALSE
db_block_buffers	50480
db_block_checking	FALSE
db_block_checksum	FALSE
db_block_lru_latches	2
db_block_max_dirty_target	50480
db_block_size	16384
db_file_direct_io_count	64
db_file_multiblock_read_count	8
db_files	400
db_name	oramain
db_writer_processes	1
dblink_encrypt_login	FALSE
dbwr_io_slaves	0
disk_asynch_io	TRUE
distributed_transactions	10
dml_locks	748

enqueue_resources	1168
fast_start_io_target	50480
fast_start_parallel_rollback	LOW
gc_defer_time	10
gc_releasable_locks	0
gc_rollback_locks	0-1024=32!8REACH
global_names	FALSE
hash_area_size	131072
hash_join_enabled	TRUE
hash_multiblock_io_count	0
hi_shared_memory_address	0
hs_autoregister	TRUE
instance_name	oramain
instance_number	0
java_max_sessionspace_size	0
java_pool_size	150M
java_soft_sessionspace_limit	0
job_queue_interval	60
job_queue_processes	4
large_pool_size	614400
license_max_sessions	0
license_max_users	0
license_sessions_warning	0
lm_locks	12000
lm_ress	6000
lock_sga	FALSE
log_archive_dest_1	location=/disk/arc
log_archive_dest_state_1	enable
log_archive_dest_state_2	enable
log_archive_dest_state_3	enable
log_archive_dest_state_4	enable
log_archive_dest_state_5	enable
log_archive_format	%s.arc
log_archive_max_processes	1
log_archive_min_succeed_dest	1
log_archive_start	TRUE
log_archive_trace	0
log_buffer	163840
log_checkpoint_interval	10000
log_checkpoint_timeout	1800
log_checkpoints_to_alert	FALSE
max_commit_propagation_delay	700
max_dump_file_size	UNLIMITED
max_enabled_roles	30
max_rollback_segments	37
mts_circuits	0
mts_max_dispatchers	5
mts_max_servers	20
mts_multiple_listeners	FALSE
mts_servers	0
mts_service	oramain
mts_sessions	0
nls_language	AMERICAN
nls_territory	AMERICA
object_cache_max_size_percent	10
object_cache_optimal_size	102400
open_cursors	300
open_links	4
open_links_per_instance	4

NAME	VALUE
optimizer_features_enable	8.1.7
optimizer_index_caching	0
optimizer_index_cost_adj	100
optimizer_max_permutations	80000
optimizer_mode	CHOOSE
optimizer_percent_parallel	0
oracle_trace_collection_path	?/otrace/admin/cdf
oracle_trace_collection_size	5242880
oracle_trace_enable	FALSE
oracle_trace_facility_name	oracled
oracle_trace_facility_path	?/otrace/admin/fdf
os_roles	FALSE
parallel_adaptive_multi_user	FALSE
parallel_automatic_tuning	FALSE
parallel_broadcast_enabled	FALSE
parallel_execution_message_size	2152
parallel_max_servers	5
parallel_min_percent	0
parallel_min_servers	0
parallel_server	FALSE
parallel_server_instances	1
parallel_threads_per_cpu	2
partition_view_enabled	FALSE
plsql_v2_compatibility	FALSE
pre_page_sga	FALSE
processes	150
query_rewrite_enabled	FALSE
query_rewrite_integrity	enforced
read_only_open_delayed	FALSE
recovery_parallelism	0
remote_dependencies_mode	TIMESTAMP
remote_login_passwordfile	NONE
remote_os_authent	FALSE
remote_os_roles	FALSE
replication_dependency_tracking	TRUE
resource_limit	FALSE
rollback_segments	r01, r02, r03, r04
row_locking	always
serial_reuse	DISABLE
serializable	FALSE
service_names	oramain
session_cached_cursors	0
session_max_open_files	10
sessions	170
shadow_core_dump	partial
shared_memory_address	0
shared_pool_reserved_size	30728640
shared_pool_size	614572800
sort_area_retained_size	65536
sort_area_size	65536
sort_multiblock_read_count	2
sql92_security	FALSE
sql_trace	FALSE
sql_version	NATIVE
standby_archive_dest	?/dbs/arch
star_transformation_enabled	FALSE

```
tape_asynch_io                          TRUE
text_enable                             FALSE
thread                                  0
timed_os_statistics                     0
timed_statistics                        FALSE
transaction_auditing                    TRUE
transactions                            187
transactions_per_rollback_segment       5
use_indirect_data_buffers               FALSE
user_dump_dest                          /u01/app/oracle/admin/oramain/udump
```

基本块的大小：16384

内存的分配：

```
Select * from v$sga;
SHOW PARAMETER PGA;
NAME                                         VALUE
---------------------------------------- ----------
Fixed Size                                   104936
Variable Size                             780750848
Database Buffers                          827064320
Redo Buffers                                 172032
```

排序的统计：

```
Select name,value from v$sysstat where name like '%sort%';
NAME                                         VALUE
---------------------------------------- ----------
sorts (memory)                              1395056
sorts (disk)                                   4392
sorts (rows)                              358987739
```

数据内存的命中率：

```
SELECT 1 - (phy.value -lob.value -dir.value) / ses.value
  "CACHE HIT RATIO"  FROM   v$sysstat ses, v$sysstat lob,
v$sysstat dir, v$sysstat phy
    WHERE  ses.name = 'session logical reads'
    AND    dir.name = 'physical reads direct'
    AND    lob.name = 'physical reads direct (lob)'
    AND    phy.name = 'physical reads';


CACHE HIT RATIO
---------------
     .966038933
```

96.6%很好，不必调整了。

SQL 语句的内存命中率：

```
select NAMESPACE,GETHITRATIO from V$LIBRARYCACHE;
NAMESPACE                      GETHITRATIO
------------------------------ -----------
SQL AREA                        .999891852
TABLE/PROCEDURE                  .99942524
BODY                            .999833797
TRIGGER                         .999998464
INDEX                           .805940594
CLUSTER                         .994515539
```

```
OBJECT                                  1
PIPE                                    1

SQL> select * from V$sgastat;

POOL                    NAME                                       BYTES
----------------------- ---------------------------------------- ----------
                        fixed_sga                                  104936
                        db_block_buffers                        827064320
                        log_buffer                                 163840
shared pool             free memory                             544941568
shared pool             miscellaneous                             1200552
shared pool             processes                                  160800
shared pool             transactions                               284240
shared pool             PL/SQL SOURCE                                7296
shared pool             fixed allocation callback                    1904
shared pool             table definiti                               9768
shared pool             trigger source                               1712
shared pool             PL/SQL DIANA                               911112
shared pool             trigger inform                                448
shared pool             table columns                               32000
shared pool             db_handles                                 132000
shared pool             db_files                                   160976
shared pool             sessions                                   413440
shared pool             State objects                              362400
shared pool             db_block_hash_buckets                     1820496
shared pool             PL/SQL MPCODE                              672832
shared pool             db_block_buffers                         10499840
shared pool             errors                                     302712
shared pool             enqueue_resources                          121472
shared pool             DML locks                                  125664
shared pool             dictionary cache                          3179632
shared pool             SYSTEM PARAMETERS                          105352
shared pool             message pool freequeue                     191192
shared pool             character set memory                        96912
shared pool             KGFF heap                                  10016
shared pool             library cache                           22771952
shared pool             ktlbk state objects                        100232
shared pool             sql area                                38599928
shared pool             PLS non-lib hp                              2136
shared pool             KQLS heap                                2273320
shared pool             trigger defini                             12608
shared pool             KGK heap                                   14096
shared pool             event statistics per sess                 590240
large pool              free memory                               614400
java pool               free memory                             150003712
```

IO 的信息：

```
select name,PHYRDS,PHYWRTS from v$filestat f,v$datafile d
where f.file#=d.file# order by PHYRDS;
NAME                                       PHYRDS    PHYWRTS
---------------------------------------- ---------- ----------
/oramain/indx.dbf                             213        211
/oramain/rbs1.dbf                             352     215328
/oramain/rbs.dbf                              411     178499
/oramain/spstat.dbf                          1324        708
/oramain/system01.dbf                       89630       9680
/oramain/USER_02.dbf                      6153709    1344648
/oramain/USER_03.dbf                      6246731    1287246
```

/oramain/USER.DBF	6352329	1314316
/oramain/user_06.dbf	11027918	1504223
/oramain/user_05.dbf	11074179	1486826
/oramain/USER_04.dbf	11205833	1625337
/oramain/tmp.dbf	38033161	1982266

监听的配置信息：

1521 端口

LATCH 的应用状况评估和建议

```
select NAME,GETS,MISSES from V$latch where MISSES>100;
```

NAME	GETS	MISSES
session allocation	14170123	2001
session idle bit	223251524	602
messages	65520522	85456
enqueues	74130778	744
enqueue hash chains	62256193	1125
cache buffers lru chain	52132714	8523
checkpoint queue latch	73539589	4297
cache buffers chains	3725905067	**1169997**
redo allocation	73298884	972
redo writing	66673887	260381
dml lock allocation	36397396	406
list of block allocation	19079664	106
transaction allocation	29403134	256
undo global data	29437141	844
row cache objects	61870822	695
shared pool	139247703	7738
library cache	918542876	**141408**

```
SQL> col PARENT_NAME for a35
SELECT PARENT_NAME,SUM(LONGHOLD_COUNT) FROM V$LATCH_MISSES
GROUP BY PARENT_NAME
HAVING SUM(LONGHOLD_COUNT)>0;
```

PARENT_NAME	SUM(LONGHOLD_COUNT)
NLS data objects	1
Token Manager	1
cache buffers chains	**2679**
cache buffers lru chain	161
channel handle pool latch	6
channel operations parent latch	7
checkpoint queue latch	1915
cost function	7
dml lock allocation	6
enqueue hash chains	53
enqueues	14
library cache	**9554**
list of block allocation	3
messages	56
mostly latch-free SCN	2
redo allocation	22
redo writing	338
sequence cache	1
session allocation	102

```
session idle bit                                    3
shared pool                                      1064
transaction allocation                              9
undo global data                                   17
```

关于长查询的对象

```
col  MESSAGE for a80
select MESSAGE,count(*) from v$session_longops
group by MESSAGE;
```

由于缺少索引而造成了不表要的数据库的全表查询，建议建立适当的索引。

操作系统的信息

```
System: ST-HP-02                                        Sat Sep 29 10:10:53 2007
Load averages: 0.02, 0.04, 0.04
124 processes: 122 sleeping, 0% 100.0g
Cpu state7    2.0%    0.0%    0.6%   97.4
CPU    LOAD    US2%    0.0%    0.2%   99.6E   BLOCK   SWAIT    INTR    SSYS
 0     0.00    0.0%    0.0%    0.0% 100.0%    0.0%    0.0%    0.0%    0.0%
 1     0.02    4.8%    0.0%    0.8%   94.4%    0.0%    0.0%    0.0%    0.0%    1:03
 2     0.00    0.8%    0.0%    0.8%   98.4%    0.0%    0.0%    0.0%    0.0%
128 processes: 126  0.0%    0.2%   99.6%    0.0%    0.0%    0.0%    0.0%       8
---    ----2980K (186496K) real, 436748K (411716K) virtual, 1210644
127 processes: 125  0.0%    0.0% 100.0%    0.0%    0.0%    0.0%    0.0%

Memory: 215955.4%    0.0%    1.0%   93.6K (47760K) virtual, 12177007   1.17 Page# 1/7
 1    ?     13193 oracle    154 20 32832K   1784K sleep     0:00   2.44   0.44
 1    ?     13191 oracle    154 20 32832K   1784K sleep     0:00   1.54   0.40 COMMAND
 2    ?     13172 oracle    149 20 32960K   1784K sleep     0:00   0.74   0.36 vxfsd
 2    ?     13054 oracle    154 20 32832K   1784K sleep     0:00   0.65   0.29 oracleoramain
 1    ?     13170 oracle    154 20 32832K   1784K sleep     0:00   0.44   0.24 oracleoramain
 1    ?    348284 oracle    154 20 32832K   1784K sleep     0:00   0.40   0.2349 oracleoramain
 1    ?     13166 oracle    154 20 32832K   1784K sleep     0:00   0.40   0.23 oracleoramain
 2    ?     13168 oracle    154 20 32832K   1784K sleep     0:00   0.42   0.23 oracleoramain
 0    ?     17017 root       20 20  9744K   8516K sleep   515:32   0.23   0.23 cmcld
 1    ? 292 root 154 20     32K    132K sleep   571:13   0.23   0.23.69   0.51 cmcld
 2 pts/ta 13170 root     178 20    664K    452K run       0:00   0.41   0.21 top
 1 pts/ta 13130 root     178 20    664K    468K run       0:00   0.24   0.19 top
 2    ?     13147 oracle    154 20 32832K   1784K sleep     0:00   0.27   0.18 oracleoramain
 2    ?      1314 root     154 20   2516K   1408K sleep   350:02   0.16   0.16 ARMServer
 0    ?     19710 root     158 20    276K    220K sleep   361:49   0.15   0.15 sh
 1    ?     13129 oracle    154 20 32832K   1784K sleep     0:00   0.12   0.09 oracleoramain
 0 pts/ta 13096 root     158 20    556K    220K sleep     0:00   0.07   0.06 sh
 1    ?     13147 oracle    154 20 32832K   1784K sleep     0:00   0.65   0.29 oracleoramain
 2    ?       292 root     154 20     32K    132K sleep   571:13   0.29   0.29 syncer
 3    ?     19706 oracle    154 20  8988K   2424K sleep   121:07   0.29   0.29 tnslsnr
 0    ?     17017 root       20 20  9744K   8516K sleep   515:32   0.24   0.24 cmcld
 2 pts/ta 13130 root     178 20    664K    468K run       0:00   0.29   0.19 top
 1    ?     13129 oracle    154 20 32832K   1784K sleep     0:00   0.23   0.16 oracleoramain
 2    ?      1314 root     154 20   2516K   1408K sleep   350:02   0.13   0.13 ARMServer
 0    ?     19710 root     158 20    276K    220K sleep   361:49   0.13   0.13 sh
 0 pts/ta 13096 root     158 20    556K    220K sleep     0:00   0.13   0.10 sh

LSNRCTL> status
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.50)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for HPUX: Version 8.1.7.4.0 - Production
Start Date                25-APR-2007 21:36:53
```

```
Uptime                    156 days 11 hr. 50 min. 20 sec
Trace Level               off
Security                  OFF
SNMP                      OFF
Listener Parameter File   /u01/app/oracle/product/8.1.7/network/admin/listener.ora
Listener Log File         /u01/app/oracle/product/8.1.7/network/log/listener.log
Services Summary...
  PLSExtProc            has 1 service handler(s)
  XA2           has 1 service handler(s)
  oradb2                has 1 service handler(s)
  oramain               has 1 service handler(s)
  oramain               has 1 service handler(s)
  wd2000                has 1 service handler(s)
The command completed successfully
LSNRCTL> status
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.50)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for HPUX: Version 8.1.7.4.0 - Production
Start Date                25-APR-2007 21:36:53
Uptime                    156 days 11 hr. 50 min. 20 sec
Trace Level               off
Security                  OFF
SNMP                      OFF
Listener Parameter File   /u01/app/oracle/product/8.1.7/network/admin/listener.ora
Listener Log File         /u01/app/oracle/product/8.1.7/network/log/listener.log
Services Summary...
  PLSExtProc            has 1 service handler(s)
  XA2           has 1 service handler(s)
  oradb2                has 1 service handler(s)
  oramain               has 1 service handler(s)
  oramain               has 1 service handler(s)
  wd2000                has 1 service handler(s)
The command completed successfully
LSNRCTL> services
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.50)(PORT=1521)))
Services Summary...
  PLSExtProc            has 1 service handler(s)
    DEDICATED SERVER established:0 refused:0
      LOCAL SERVER
  XA2           has 1 service handler(s)
    DEDICATED SERVER established:0 refused:0
      LOCAL SERVER
  oradb2                has 1 service handler(s)
    DEDICATED SERVER established:0 refused:0
      LOCAL SERVER
  oramain               has 1 service handler(s)
    DEDICATED SERVER established:0 refused:0
      LOCAL SERVER
  oramain               has 1 service handler(s)
    DEDICATED SERVER established:1003868 refused:574
      LOCAL SERVER
  wd2000                has 1 service handler(s)
    DEDICATED SERVER established:0 refused:0
      LOCAL SERVER
The command completed successfully
LSNRCTL>
```

```
$ bdf
Filesystem          kbytes    used   avail %used Mounted on
/dev/vg00/lvol3     143360   51502   86168   37% /
/dev/vg00/lvol1      83733   46663   28696   62% /stand
/dev/vg00/lvol8    1064960  934729  122850   88% /var
/dev/vg00/lvol7    1179648  721833  429264   63% /usr
/dev/vg00/lvol9    9932800 4647103 4955362   48% /u01
/dev/vg00/lvol4    1536000 1043042  462675   69% /tmp
/dev/vg00/lvol6    1024000  377431  606201   38% /opt
/dev/vg00/lvol5     512000    4865  475496    1% /home
/dev/vgbk/lvbk     134209536 35172712 98263120   26% /disk
/dev/vg01/lv_oramain   30679040 15666296 14778236   51% /oramain
```

系统的健康总结：
数据在 30 天内增长了 120m.上回检查是 50 天增长 200m,我们系统每天 4m 的数据增加,现有的空间很富裕.
现在系统很稳定.

系统的优化处理

```
SQL> show parameter sort_

NAME                                 TYPE                 VALUE
------------------------------------ -------------------- -----
sort_area_retained_size              integer              65536
sort_area_size                       integer              65536
sort_multiblock_read_count           integer              2
SQL> show parameter area_size

NAME                                 TYPE                 VALUE
------------------------------------ -------------------- -------
bitmap_merge_area_size               integer              1048576
create_bitmap_area_size              integer              8388608
hash_area_size                       integer              131072
sort_area_size                       integer              65536
```

从我的角度出发,应该修改下面的参数.
sort_area_size =512k
hash_area_size =1024k

我们应该把高消耗资源的语句的执行计划列出来,仔细分析,改进 SQL 的性能.

数据库的安装

1. 获得数据库产品的安装介质,可以下载
2. 配置操作系统的环境,打所要求的补丁,修改操作系统内核参数
3. 建立 oracle 用户,配置环境变量
4. 测试图形终端可以显示
5. 运行./runInstaller
6. 建立数据库

我写的简单,其实不大容易,仔细按文档操作.见多识广就好了,这就是经验.

打补丁

打补丁说大不大,说小不小.一句话,先把备份做好,打补丁要覆盖原来的oracle_home.和安装一编产品差不多,但不建立数据库.要将老数据库通过脚本升级到新的数据库.每个补丁都有 readme,写的很详细.

数据库的主备模式

1. 最好是产品安装到本地,数据库建立在磁盘阵列上.先安装一台.
2. 监听要配置浮动 ip
3. 在一台主机建立数据库
4. 安装第二台主机,只安装产品,不要建立数据库
5. 配置网络
6. 将主机一的初始化参数文件和密码文件拷贝到本地
7. 主机二建立参数文件中描述的路径
8. 将主机一的/etc/oratab 复制到本地
9. 测试双机的切换

双机 rac 介绍

双机就是两个实例同时访问一个数据库.很有难度,oracle 做了十余年都没有做好,想法是好的,实际有很多的 bug,性能没有你想象的提高那么.从 8 的 ops 做到了 10g 的 rac,逐步成熟了.
双机并行的前提是操作系统的并行,主要是同时读写一个设备的问题,可以使用操作系统的软件集群,也可以使用 oracle 的软件.玩集群的前提是单机数据库搞明白,水到渠成.

迁移生产数据库到新的环境

1. 安装新的版本数据库,一般来说平台版本不一定相同
2. 建立新的数据库,注意字符集,最好要匹配
3. 安装最新的补丁集,运行脚本将数据库升级到最新补丁数据库
4. 配置新的数据库,建立表空间,修改参数,配置网络
5. 将老的数据库 exp
6. 在新数据库 imp
7. 测试业务
8. 制定备份策略

以上说的是相对较小的数据库,导出的结果不大的情况,以我的经验一般每小时可以导入 3g 的数据.
如果你的数据库很大,可以使用物理迁移.
比如你有 100g 的数据文件,在 windows 平台,817 的数据库,现在想迁移到 linux 操作系统.920 数据库.
你可以将冷备份拷贝到 linux 系统.重新建立控制文件,再使用迁移助手将老数据库迁移到新数据库.