SQL> set pagesize 30 linesize 60

SQL> set SERVEROUTPUT ON

为了显示日期时间的需要，先更改会话的日期时间格式如下：

SQL> ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YY HH24:MI:SS';

SQL> clear screen 清除屏幕缓存区 或者 SQL> CL SCR

一、安装 Oracle10g 注意事项：

1. 启动操作系统，并以 Administrator 身份登录。
2. 如果有任何其他 Oracle 服务，就应该先将其停止，特别是要将监听器服务停止。
3. 如果存在 ORACLE_HOME 环境变量，要将其删除。
4. 在安装前，记录下无力数据库服务器的计算机名称、IP 地址，以便在安装客户机过程中，定义网络服务时使用。
5. 在安装过程中，记录下每个步骤、提问及输入数据，尤其时用户名和口令。
6. 安装完成后，检查安装结果。

安装结束后的 URL 及其端口号等信息被记录在如下的文件中：

C:\oracle\product\10.1.0\db_1\install\portlist.int

如安装出现问题后，可以查看它的安装记录文件查看问题原因。记录文件在如下目录中：

SYSTEM_DRIVE:\ProgramFiles\Oracle\Inventory\logs

格式为：installActionsdate_time.log 如：installActions2003-05-14_09-00-56-am .log

安装完成后会在此 D:\oracle\product\10.2.0 路径下有个 admin 子目录，子目录下有以数据库名称命名的子目录。该子目录下的几个子目录分别用于保存后台进程跟踪文件（bdump）、内核放弃文件（cdump）、数据库创建文件（create）、初始化参数文件（pfile）和用户 SQL 跟踪文件（udump）。----以上是 C 盘系统文件安装处。

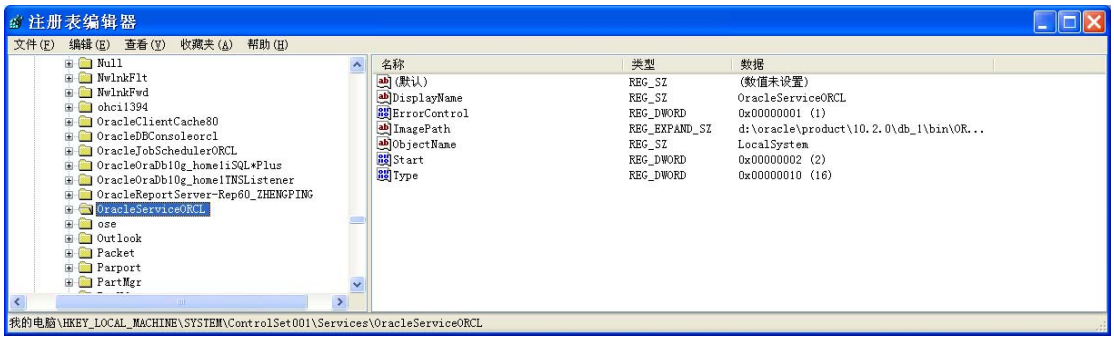
注意：下次安装时要把系统文件和数据库文件分开安装。

在 D:\oracle\product\10.2.0\oradata 数据文件存放处同样有一个以数据库名称命名的子目录，如 orcl。该数据库的控制文件（.ctl）、重做日志文件（.log）、数据文件（.dbf）等均存储在该目录的各个子目录中。

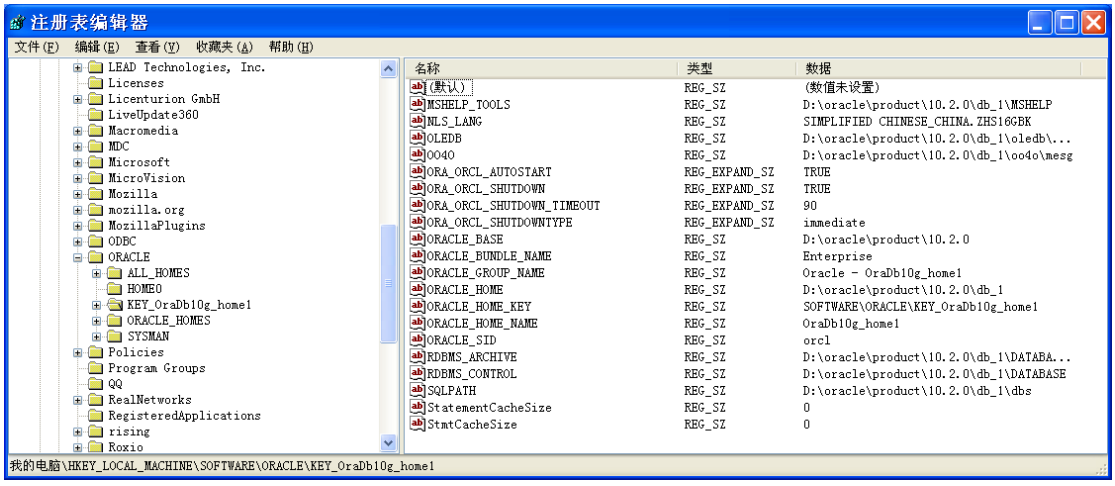
注册项

Oracle 系统的配置和服务信息，存储在操作系统中注册表数据库的几个子键中。

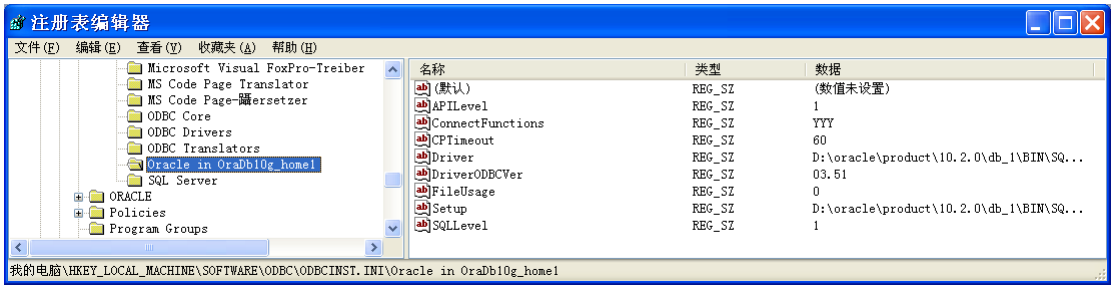
与服务有关的几个子键：



与安装和运行的环境、参数设置有关的几个子键：



与 ODBC 驱动程序有关的几个子键：



环境变量

为了能够在不提供路径的情况下直接运行或调用 Oracle 应用程序，安装完成后，应该在环境变量（系统变量）path 和 PERL5LLIB 等中添加相应的路径。

连接标识符可以把（数据库名）替换成如下标识（DESCRIPTION=（ADDRESS_LIST=（ADDRESS=（PROTOCOL=TCP）（HOST=计算机名）（PORT=1521））（CONNECT_DATA=（SERVICE_NAME=数据库名）））

第5章　SQL*Plus 工具

5．1、启动和退出 SQL*Plus 工具

注意：主机字符串就是数据库的网络服务名，如果省略，SQL*Plus 就会使用变量 ORACLE_SID 中定义的本地数据库。

退出：exit

5．2、输入、编辑、运行命令

5．2．1、输入、编辑命令

可以在 SQL*Plus 的命令提示符下，输入三种类型的命令：

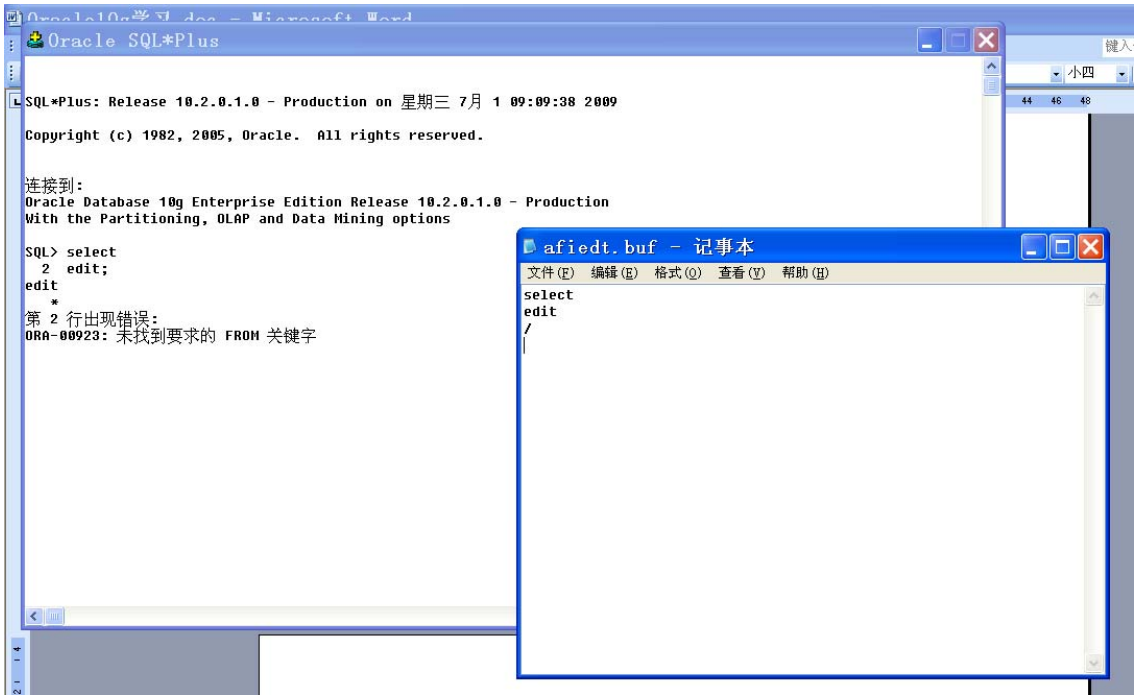
a、SQL 命令：用于操作数据库中的信息。

b、PL/SQL 块：用于操作数据库中的信息。

c、SQL*Plus 命令：用于编辑、保存、运行 SQL 命令、PL/SQL 块、格式化查询结果、自定义 SQL*Plus 环境等。

前两种命令可以访问数据库，而 SQL*Plus 命令则不能。当执行前两种命令时，会将命令暂时存放到 SQL 缓冲区中（在输入另一个命令之前一直存放在 SQL 缓冲区中。）而 SQL*Plus 命令不能被存放到 SQL 缓冲区中。

注意：EDIT 编辑器只有在 SQL 缓冲区有内容时才能启动，如下所示：



要退出编辑器先原则“保存”再选择“退出” 。然后在 SQL*Plus 命令提示符下输入“Run”命令（或/），按回车执行。编写的最后一个语句不可以用“;”号做结尾，否则无法运行。一定要以“/”做结尾。

如果要输入多条语句或命令，必须保证每条语句或命令都以一个（;）结束。

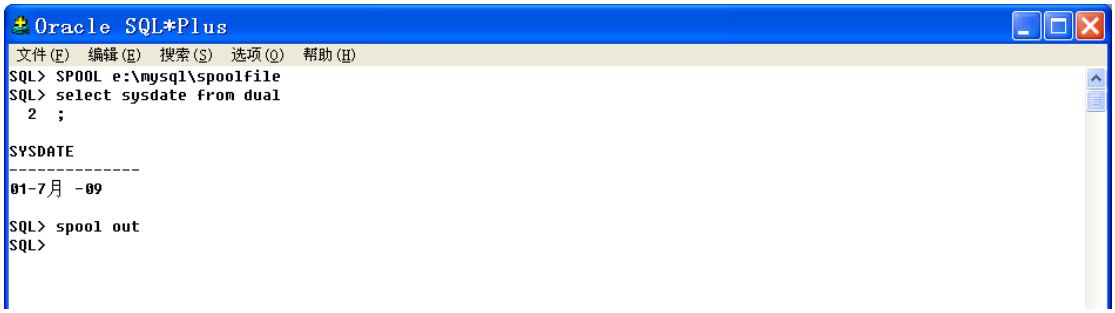
清除 SQL*Plus 的屏幕及屏幕缓冲区用：SQL>CLEAR SCREEN

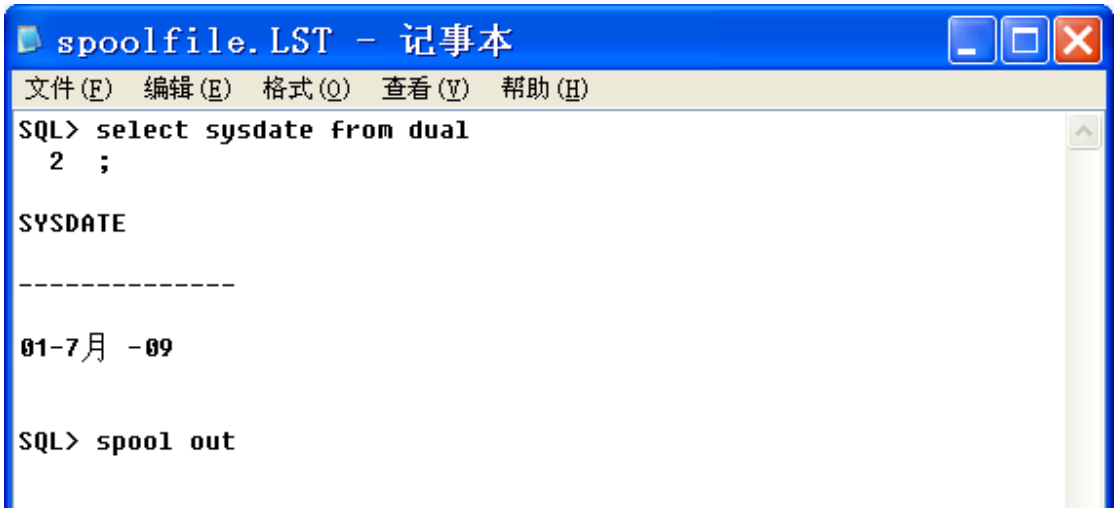
保存 SQL*Plus 命令：如果需要将命令运行情况及其结果发送到一个文件中保存起来。可使用 SPOOL filename 命令建立一个假脱机文件，如：SQL> SPOOL e:\mysql\spoolfile，就会在建立一个名为 

的假脱机文件。如果

filename 中不包含路径，则保存在*:\oracle\product\10.1.0\db_1\BIN 目录中；如果不包含扩展名，则使用“.lis” 。然后使用 SPOOL out 命令，将 SQL 语句及其结果发送到这个假托机文件中。如果不需要保存结果，可以使用 SPOOL off 关闭假托机。

例 ：



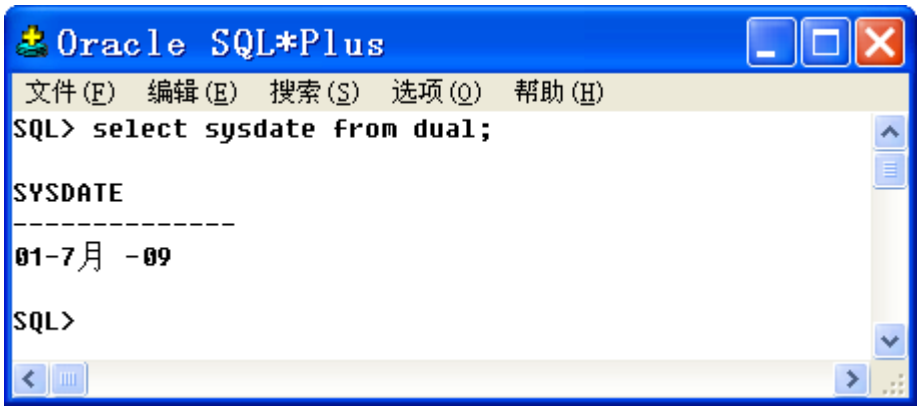


5．2．2、运行命令

在 SQL*Plus 中，可以使用三种方式运行 SQL 语句和 PL/SQL 块。分别为：命令行方式，SQL 缓冲区方式，脚本文件方式。

5．2．2．1、命令行方式

一、以命令行方式运行 SQL 命令，只需要在输入完 SQL 语句后，输入“;” ，或“/” ，按“回车”即可



二、以命令行方式运行 PL/SQL 命令，只需要在输入完块之后，在最后输入“/”，按回车即可。

三、EXECUTE（或 EXEC）。该命令能勾直接在 SQL*Plus 下执行单条 PL/SQL 语句，而不需要从缓冲区或脚本文件中执行，主要用于运行涉及一个函数或存储过程的 PL/SQL 语句。语法：EXEC[UTE] “SQL 语句”

5．2．2．2、SQL 缓冲区方式

一、RUN 命令显示并运行当前存储在 SQL 缓冲区中的 SQL 语句或 PL/SQL 块。

二、“/”命令只运行当前存储在 SQL 缓冲区中的 SQL 语句或 PL/SQL 块。

三、“LIST”命令可以列出 SQL 缓冲区中的内容，“APPEND”可以在 SQL 缓冲区的当前行（用*标记的行）后面添加新内容。

四、最后一行需要“/”，并且 SQL 语句最后不需要分号（;），否则不能正确运行。

5．2．2．3、脚本文件方式

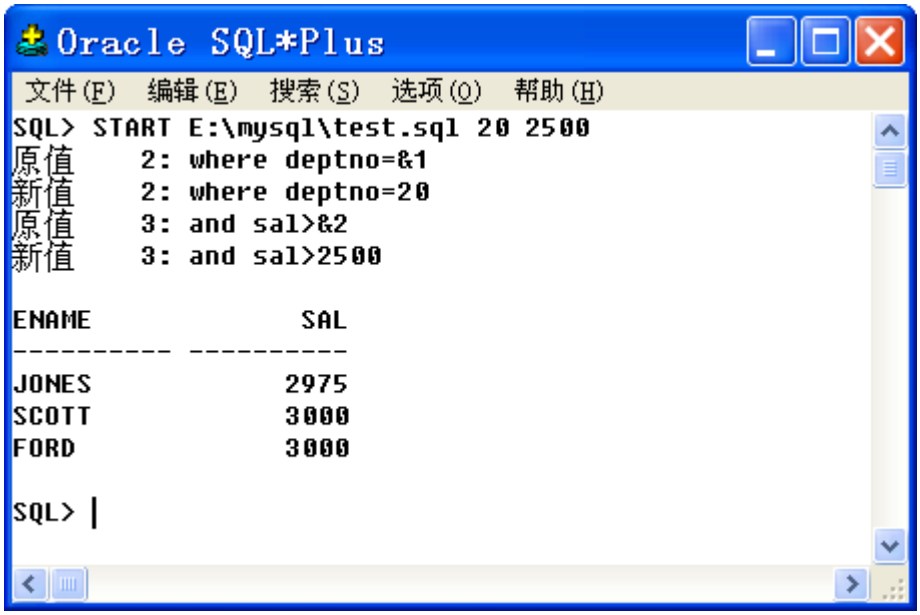
有两种运行脚本文件的方式：“START”命令和“@”命令。区别在于“@”命令机可以在 SQL*Plus 中运行可以在命令行中运行。

一、START 命令，语法为：START filename[.ext][arg1 arg2 arg3…..]，其中 filename[.ext]表示要运行的脚本文件，不写扩展名默认为“.sql” 。[arg1 arg2 arg3…..]表示希望传递给脚本文件的参数。脚本文件中的替换参数必须使用这样的声明：&1，&2，&3 等。（这里的 1，2，3 是有意义的位置标记，不能改成其他内容，否则就不会自动替代而会提示输入。）可以在一个脚本文件中放多个 SQL 语句，以便作为一个单位同事运行。此种方法常用于初始化数据库的表，和向表中插入一批初始数据。

例：脚本文件 E:\mysql\test.sql 的内容如下：

```
select ename,sal from emp
where deptno=&1
and sal>&2;
```

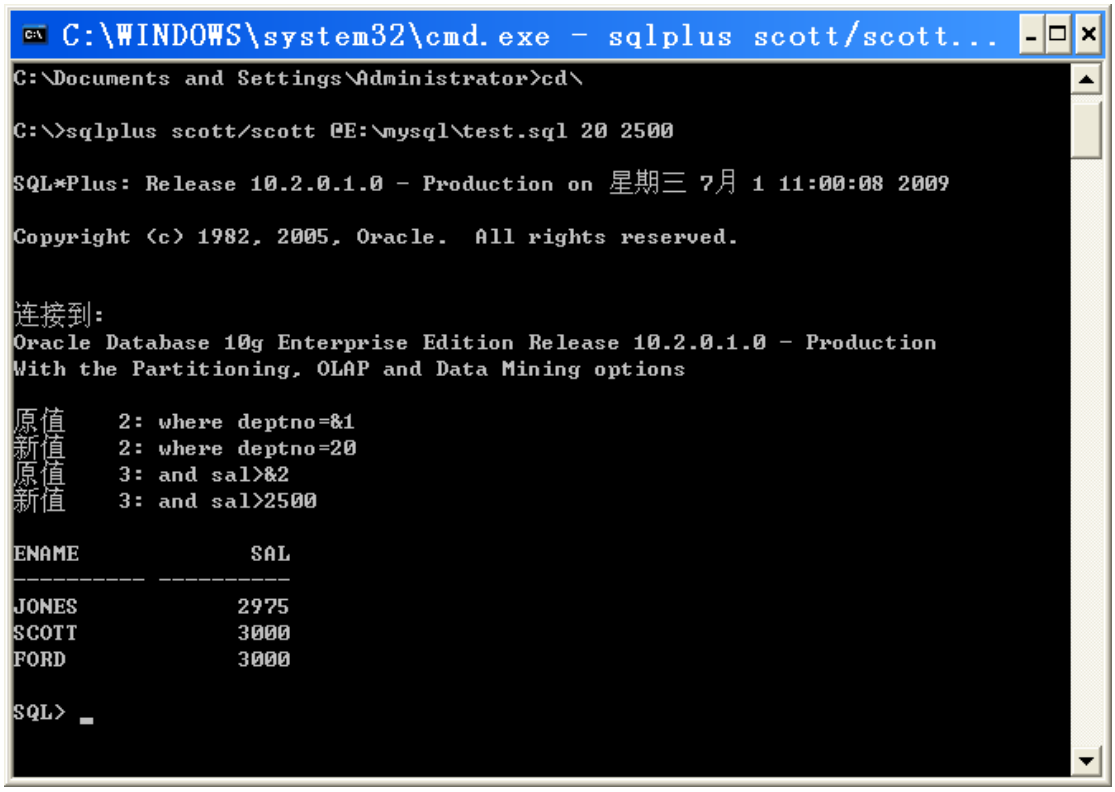
使用 START 命令运行的方式和结果如下：



二、@命令

例：在命令行使用“@”命令：

```
C:\>sqlplus scott/scott   @E:\mysql\test.sql 20 2500
```



5．3 连接命令

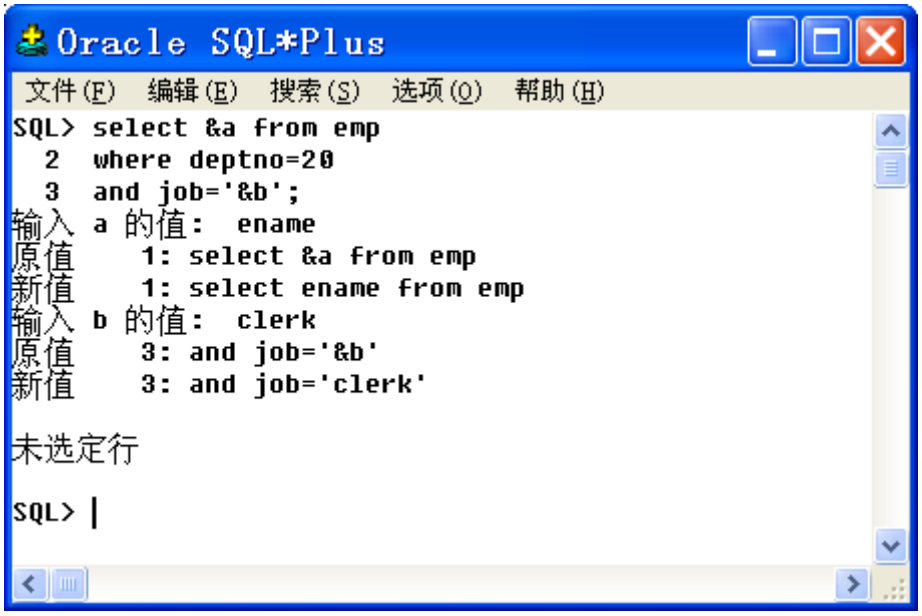
CONN[ECT]命令：此命令是先断开前一个连接，然后建立新的连接。

DISC[ONNECT]命令：此命令会断开当前连接，但不会退出 SQL*Plus。

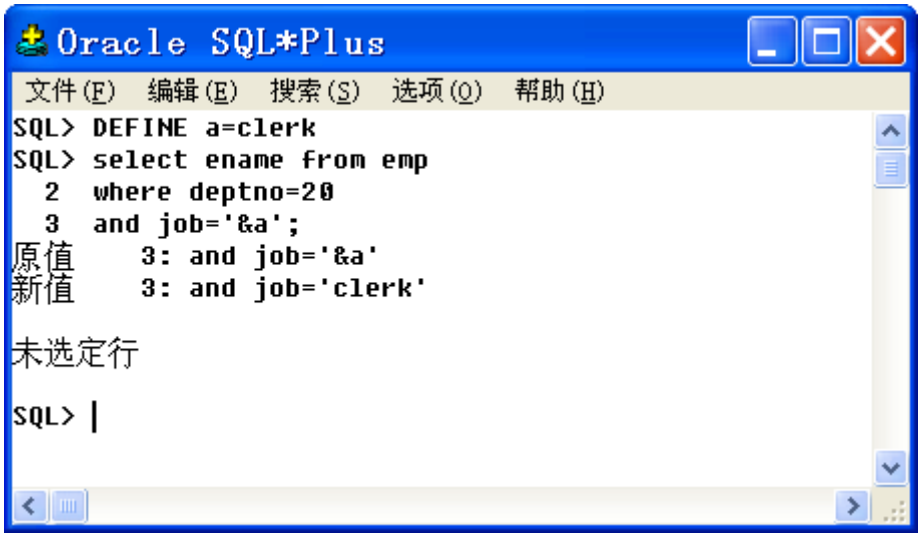
5．3 交互式命令

5．3．1 替换变量

使用替换变量：在使用替换变量输入字符或日期时，应保证在 SQL 命令中将这些变量用单引号括起来，否则，用户就需要在输入时用单引号将输入的数据括起来，以保证最后的 SQL 命令的格式是正确的。列名也可以被替代。

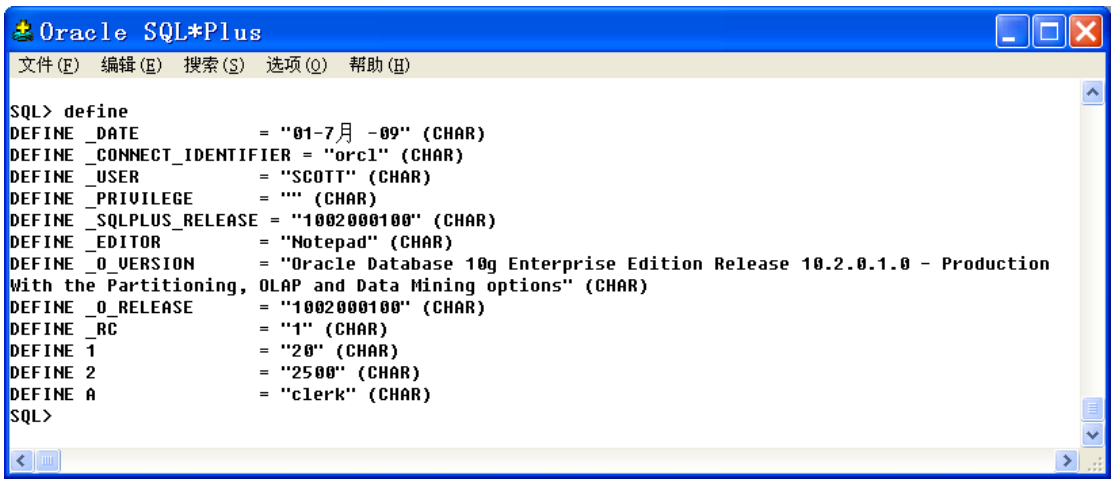


定义替换变量：可事先用 DEFINE 命令定义替换变量，这样运行时就不用输入替换变量。

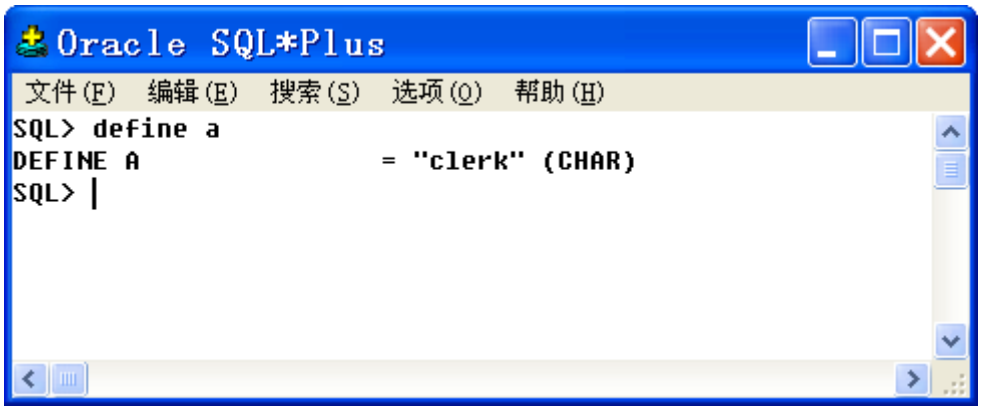


查看替换变量：可以使用 DEFINE 查看已经保留在当前 SQL*Plus 会话中的替换变量。（可以看到 a＝clerk 已经保留在当前 SQL*Plus 会话中了。）

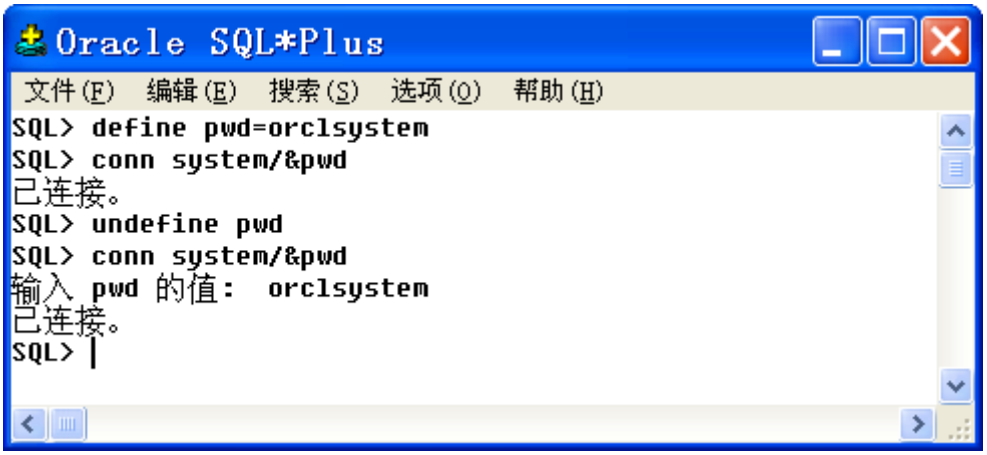
*：用 DEFINE 命令定义的替换变量总是采用 CHAR 数据类型，否则请使用 ACCEPT 命令来定义。



或直接查询替换变量 a 的值：



清除替换变量：UNDEFINE 命令清除 DEFINE 定义的替换变量。



5．3．2 与用户通信

可以使用 PROMPT 命令、PAUSE 命令、ACCEPT 命令与用户进行通信。

PROMPT 命令用于输出提示信息，以便使用户了解脚本文件的功能和运行情况。

PAUSE 命令用于暂停脚本文件的运行。

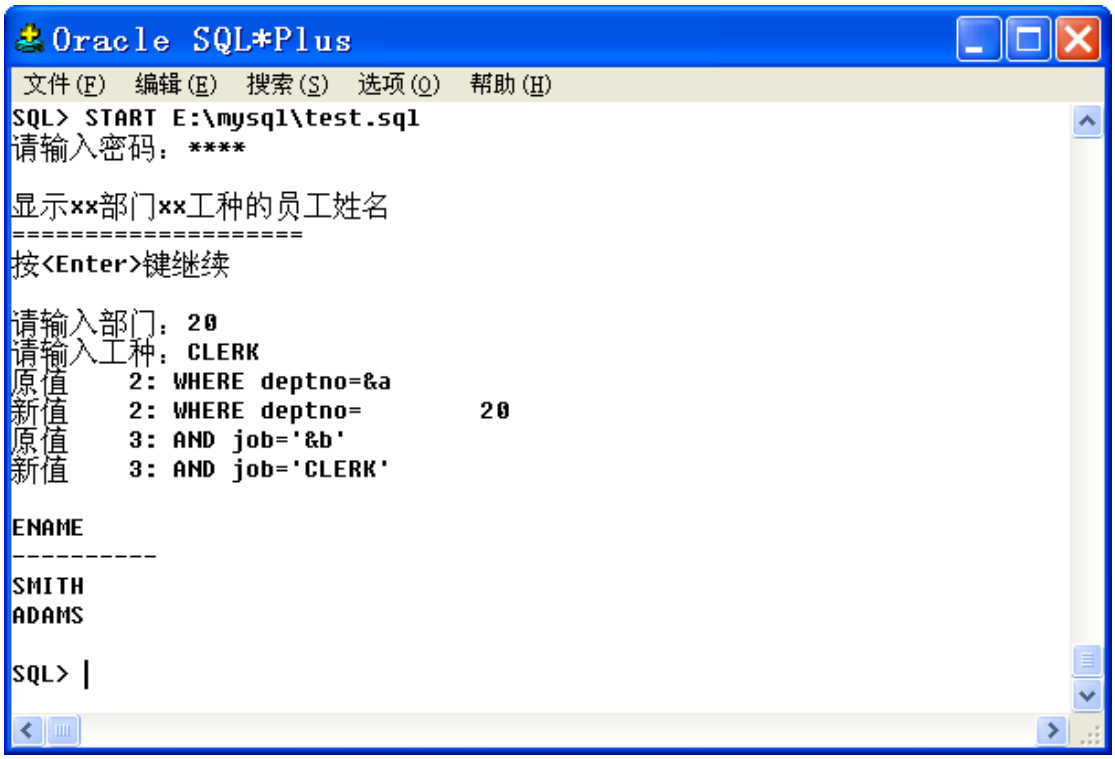
ACCEPT 命令可以让用户指定替换变量的类型（如 CHAR,NUMBER,DATE），用 PROMTP 选项指定提示信息，用 HIDE 选项隐藏输入，以便于用户输入替换变量。

例子：

下面是这 3 个命令的脚本文件。

```
ACCEPT pwd PROMPT '请输入密码：' HIDE
PROMPT
PROMPT 显示 xx 部门 xx 工种的员工姓名
PROMPT ====================
PROMPT 按<Enter>键继续
PAUSE
ACCEPT a NUMBER PROMPT '请输入部门：'
ACCEPT b CHAR PROMPT '请输入工种：'
SELECT ename FROM emp
WHERE deptno=&a
AND job='&b';
```

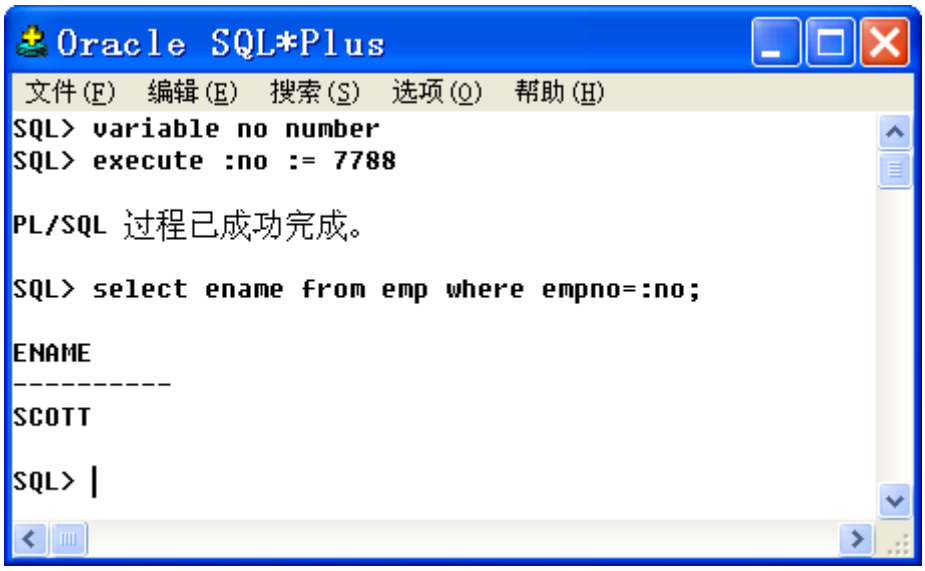
该脚本文件的运行结果是：



5．3．3 绑定变量

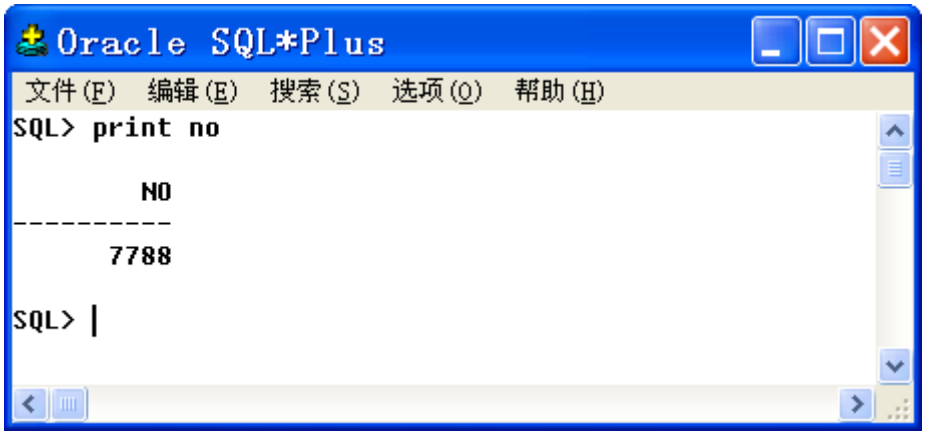
定义绑定变量：可以使用 VARIABLE 命令来定义绑定变量，当在 SQL 语句或 PL/SQL 块种使用绑定变量时，必须在绑定变量前加冒号（：）。当直接给绑定变量赋值时，需要使用 EXECUTE 命令（相当于调用 PL/SQL 存储过程）。

例：



输出绑定变量：使用 PRINT 命令输出绑定变量。

例：



5．4 自定义 SQL*Plus 环境

SHOW：显示当前 SQL*Plus 的环境变量的值

显示所有环境变量的值：SQL> SHOW all

显示当前的用户名称：SQL> SHOW user

同时显示 linesize 和 pagesize 两个环境变量的值：

SQL> SHOW linesize pagesize

SET：设置或修改管井变量的值。

同时设置 linesize 和 pagesize 两个环境变量的值：

SQL> SET linesize 100 pagesize 24

可以使用 HELP SET 命令来查看 SET 命令的功能和所有环境变量。

常用环境变量举例：

ARRAYSIZE：用于设置从数据库中一次提取的行数，默认为 15。该值越大，网络开销就越小，单占用内存将会增加。假定为 15，如果返回 50 行，则需要通过网络传送 4 次数据；假定为 25，则只需要传送 2 次。

SQL> show arraysize

arraysize 15

SQL> SET arraysize 25

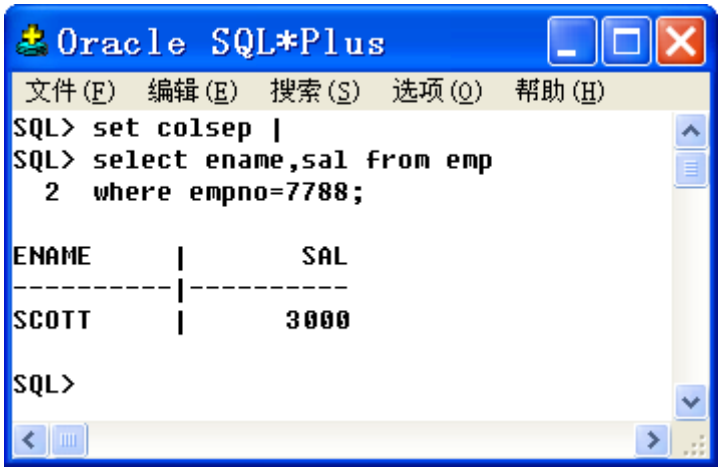
AUTOCOMMIT：用于设置是否自动提交，当设置为 ON 时，每次用户 DML 语句时都会自动提交；为 n 时，表示执行 n 个成功的 SQL 命令或 PL/SQL 块后自动提交。

SQL> show autocommit

autocommit OFF

SQL> set autocommit on

COLSEP：用于设置在选定列之间的分隔符，默认为空格。

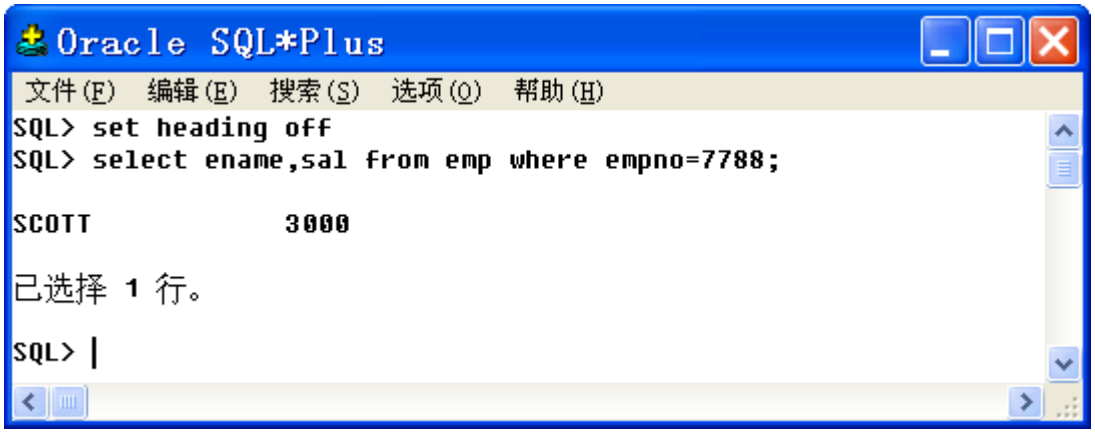


SQL> set colsep ″ ″。设置为空格

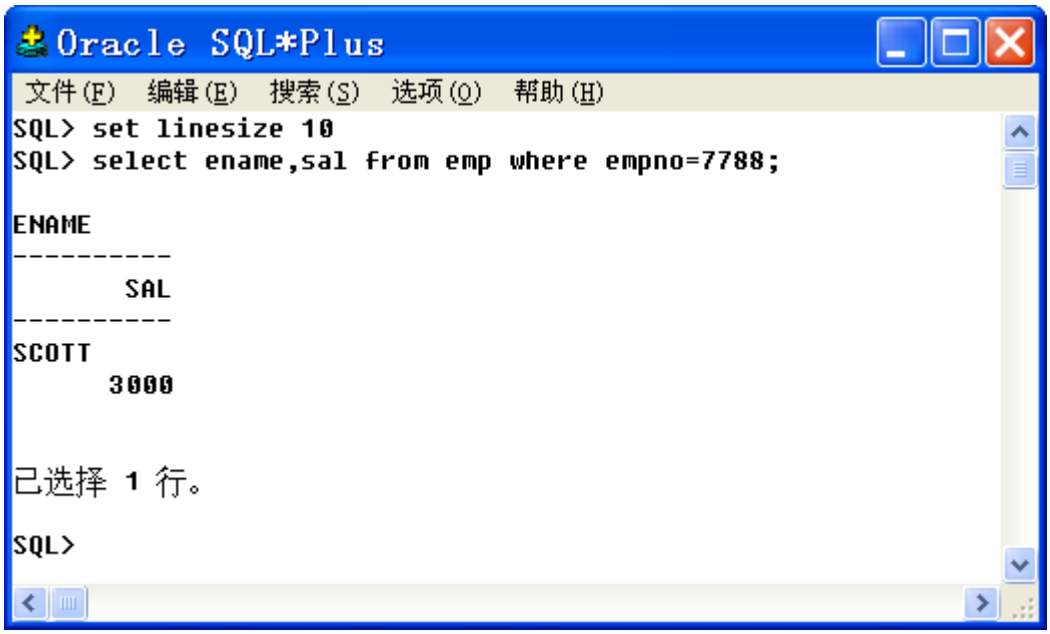
FEEDBACK：当一个查询选择至少 n 行记录时，就会显示返回的行数，默认值是 6。

如果要禁止显示行数，则将 FEEDBACK 设置为 OFF。

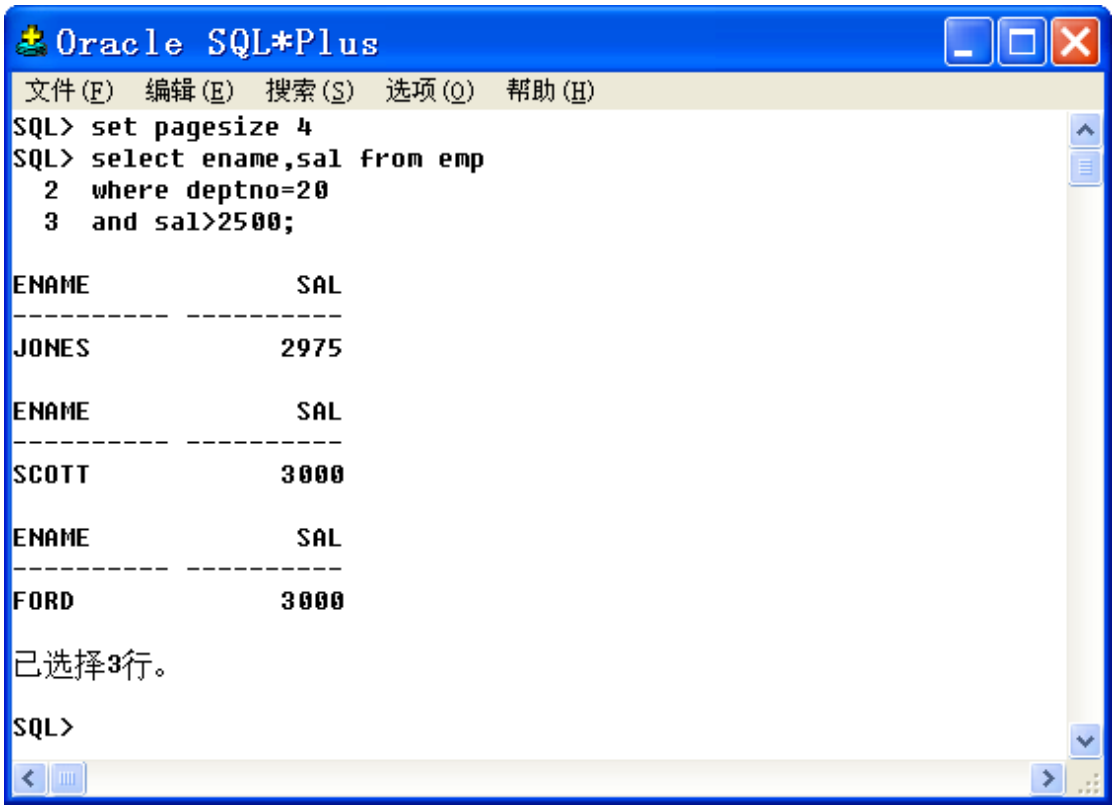
HEADING：表示是否显示列标题，默认设置为 ON。如果不想显示列标题，则设置为 OFF。



LINESIZE：代表行宽度，默认为 80，此时，如果行数据长度超过 80 个字符，那么在 SQL*Plus 中会换行显示数据，如果 LINESIZE 长度很大，则可以在一行中显示更多的数据。



PAGESIZE：表示每页所显示的行数，默认为 24。如果大于 24，则可以在一页中显示更多的数据。



SERVEROUTPUT：控制是否显示存储过程的输出，即 DBMS_OUTPUT.PUT_LINE 的输出。默认为 OFF，即当调用 DBMS_OUTPUT 时不会在 SQL*Plus 屏幕上显示输出结果。SQL> SET serveroutput on

SQL> exec dbms_output.put_line('hello')

hello

PL/SQL 过程已成功完成。

SQLPROMPT:设置 SQL*Plus 的命令提示符，默认值为 SQL>。

SQL> SET sqlprompt zhengpingSQL>

zhengpingSQL>

TIME:是否在 SQL*Plus 命令提示符前显示系统时间，默认值为 OFF

SQL>SET time on

14:32:30 SQL>

UNDERLINE:设置下划线字符，默认值为“_” 。

VERIFY:用于控制在交互式命令中，在替换变量之前和之后是否列出一个 SQL 语句的文本内容。默认为 NO。

在 Windows 下，可以执行“”－－>“环境”命令，弹出“环境”窗口。

可显示和设置环境变量。如果在该窗口的左边选择了一个环境变量，就会在窗口的中间显示该变量是否进行过自定义、当前值式什么，以及相关参数。

应该用 system 用户启动 SQL*Plus，在窗口中间对环境变量进行设置或恢复成默认值，单击“确定”按钮。

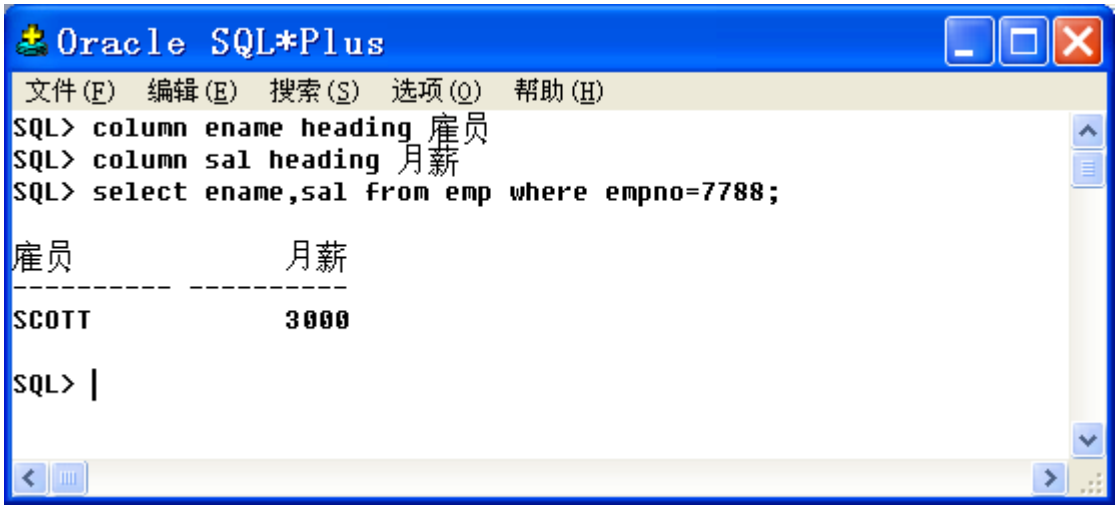
保存 SQL*Plus 环境：可以用 STORE SET filename 命令将自己定义好的 SQL*Plus 环境保存到一个脚本文件中，以便以后用 START 命令来运行。

```
SQL> store set e:\mysql\setfile.sql
已创建 file e:\mysql\setfile.sql
```

5．5 格式化查询结果

5．5．1 格式化列

一、修改列标题：可以使用 COLUMN 命令的 HEADING 子句来重新定义为更容易理解的列标题。在重新输入新的列标题或退出 SQL*Plus 之前，这些定义的列标题一直有效。还可以使用 COLUMN 命令的 JUSTIFY 子句来指定列标题的对齐格式（LEFT，CENTER，RIGHT）。

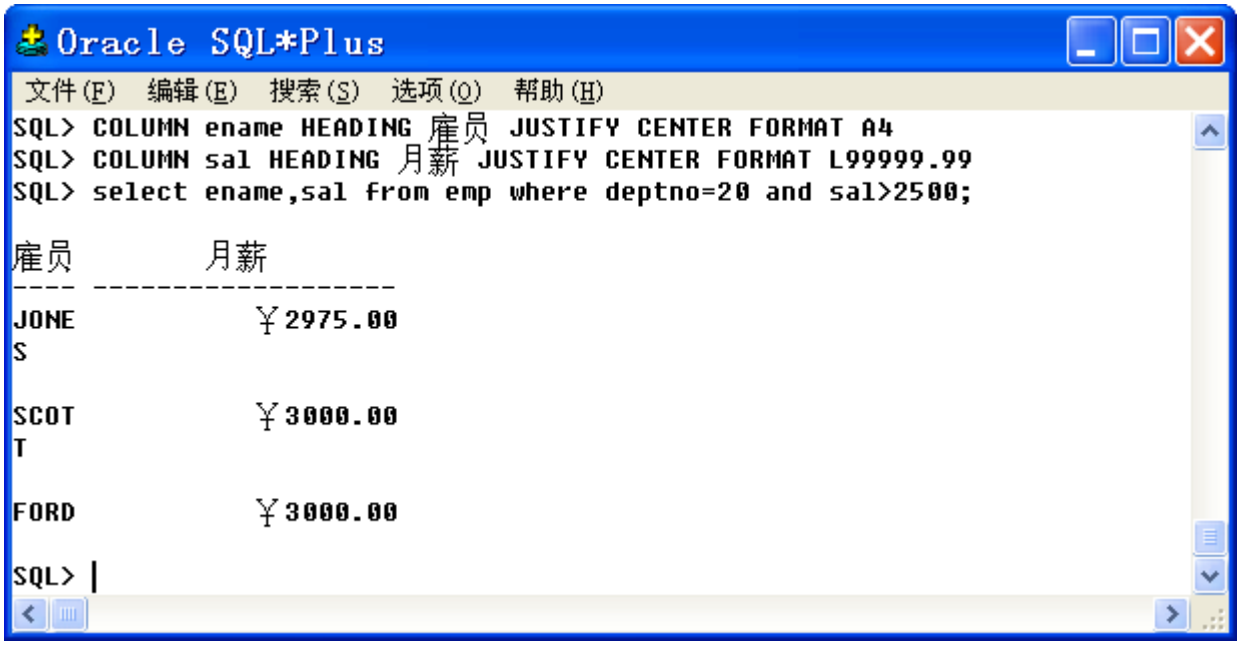


二、格式化列：使用 COLUMN 命令的 FORMAT 子句重新定义列的显示格式，格式模型包含的元素和用途，如下表：

格式模式	用途
An	设置非数值类型列（如 CHAR,DATE,LONG 等）的显示宽度。如果指定宽度小于列标题，则 SQL*Plus 将截断标题；如果为 LONG，CLOB，NCLOB 定义列的宽度则 SQL*Plus 使用 LONGCHUNKSIZE 定义的宽度
9	在数值类型列（如 NUMBER）上禁止显示前导 0
0	在数值类型列（如 NUMBER）上强制显示前导 0
$	在数值类型列（如 NUMBER）前显示美元符号
L	在数值类型列（如 NUMBER）前显示本地货币符号
.	指定数值类型列（如 NUMBER）的小数点位置
,	指定数值类型列（如 NUMBER）的千分隔符

如：

SQL> COLUMN ename HEADING 雇员 JUSTIFY CENTER FORMAT A4



三、列出和恢复列的格式

可以列出所有列的格式，执行如下命令

SQL> COLUMN

可以使用 CLEAR 命令恢复所有列的格式：

SQL> clear columns

columns 已清除

列出给定列的格式：

SQL> COLUMN sal

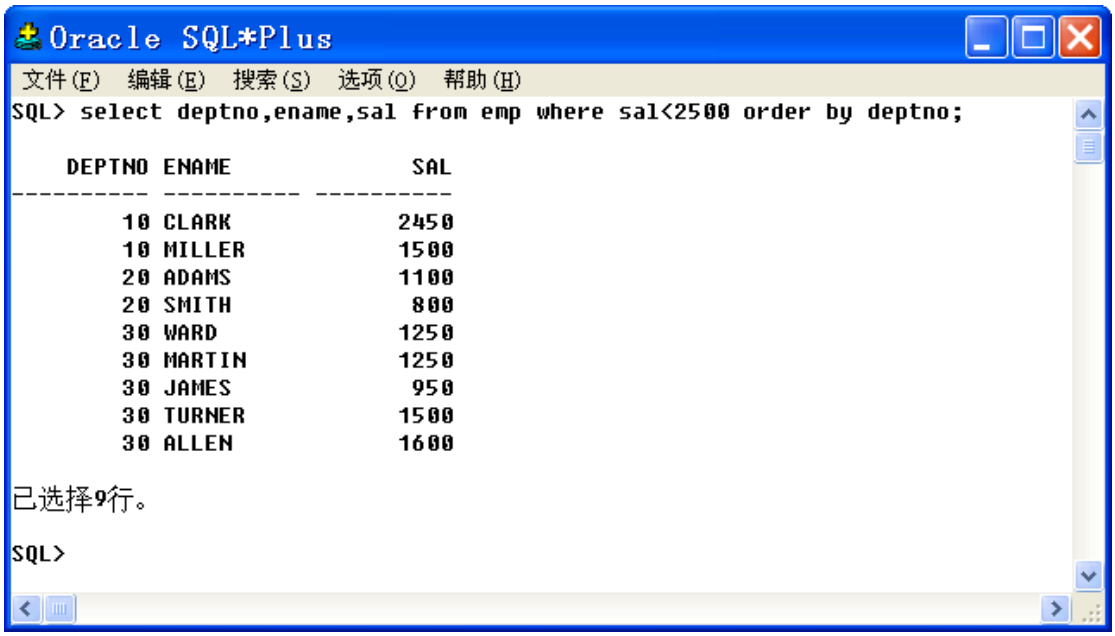
使用 COLUMN 命令的 CLEAR 选项恢复列的格式：

SQL> COLUMN sal CLEAR

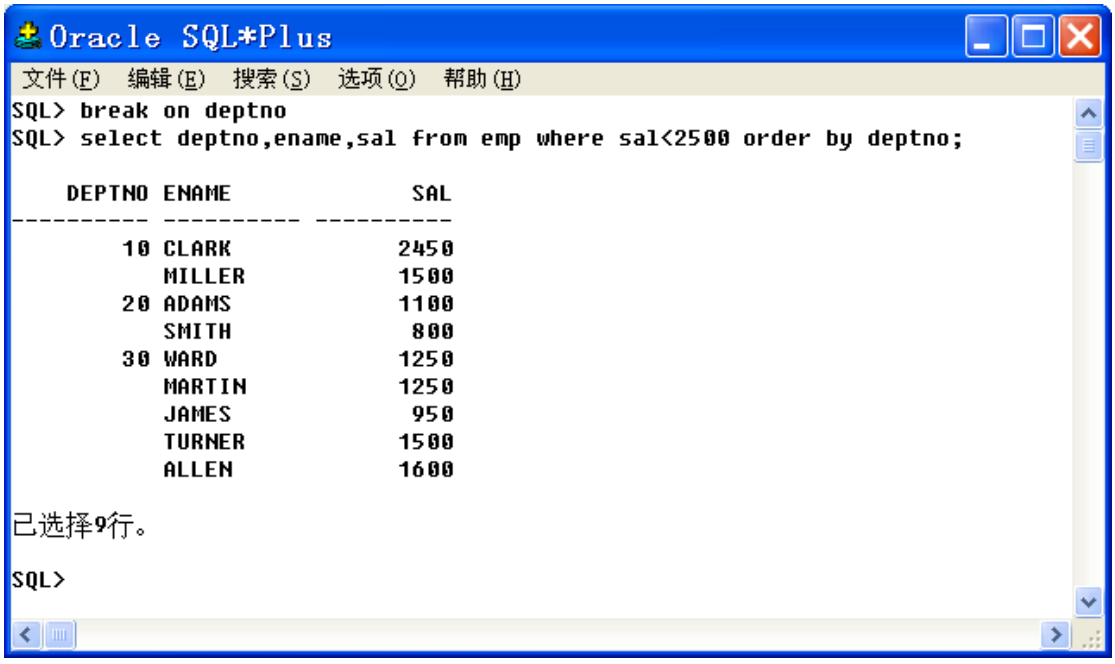
5．5．2 限制重复行和使用汇总行

BREAK 命令：限制列的重复数据。

未限制：



已限制：



在限制重复行的同时，还可以在各个分组之间插入 n 个空行：

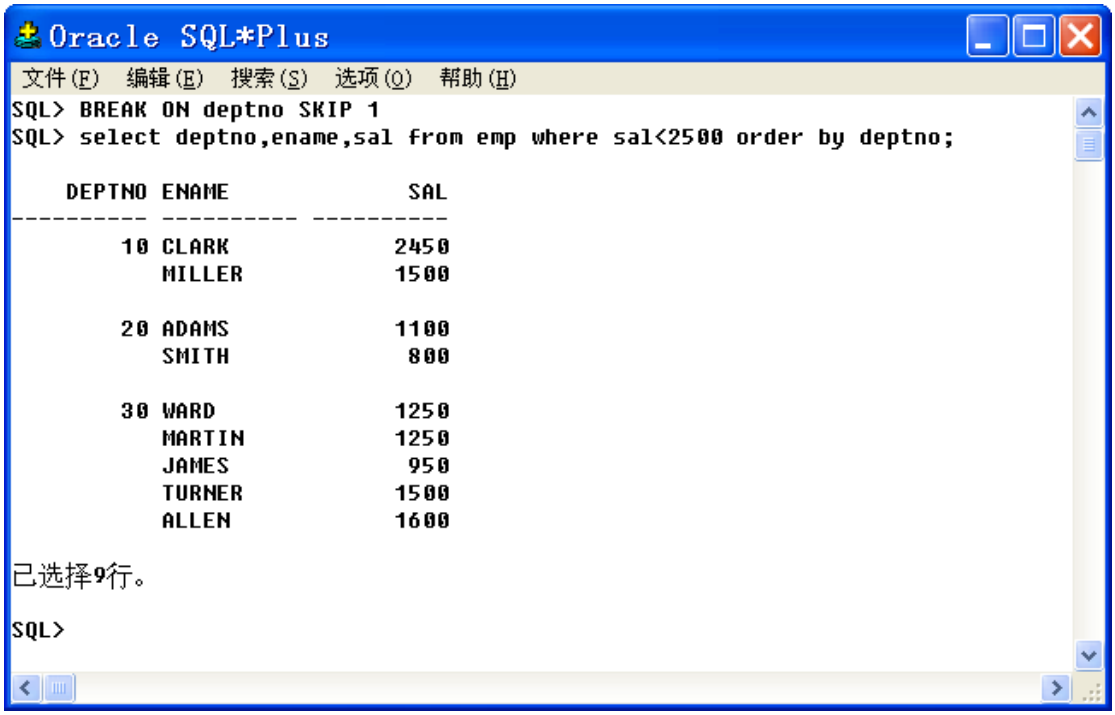
BREAK ON break_column SKIP n

甚至跳到新的一页：BREAK ON break_column SKIP PAGE

在每行之后插入 n 个空行：BREAK ON ROW SKIP n

在报表之后插入 n 个空行：BREAK ON REPORT SKIP n

例：在各个分组之间插入一个空行：



使用汇总行：与 BREAK 命令配合使用 COMPUTE 命令对各个分组进行各种汇总计算。

BREAK ON column SKIP n

COMPUTE function LABEL text

OF{expr|column|alias}...ON{ expr|column|alias|REPORT|ROW }

其中 function 是计算函数，常用的计算函数及功能如下表：

函数	功能
SUM	计算列值的和
MINMUM	计算列值的最小值
MAXMUM	计算列值的最大值
AVG	计算列值的平均值
STD	计算列值的标准方差
VARIANCE	计算列值的协方差
COUNT	计算列值的非 NULL 的个数
NUMBER	计算列值的行数

LABEL text 子句是汇总结果的描述文本，如省略，则使用进行计算的函数的名字。

OF{expr|column|alias}子句指定要进行计算的列或表达式

ON{ expr|column|alias|REPORT|ROW }子句指定分组的列或表达式，即当该列或表达式的值发生变化时，产生一个汇总行并计算汇总结果。

如果要在报告的后面计算汇总结果，使用下面的命令格式：

BREAK ON REPORT

COMPUTE function LABEL text OF {expr|column|alias}...ON REPORT

例：

按照如下方式实现按部门汇总 sal 列：

SQL> BREAK ON deptno

SQL> COMPUTE SUM LABEL 小计：OF sal comm ON deptno

SQL> select deptno,ename,sal,comm from emp where sal<2500 order by deptno;

查看、清除限制重复行和使用汇总行：

查看当前的 BREAK 和 COMPUTE 的设置情况：

SQL> BREAK

SQL> COMPUTE

清除限制重复行和使用汇总行：

SQL> CLEAR BREAK

SQL> CLEAR COMPUTE

设置页与报告的标题和脚注：

“页”值一屏信息，“报告”值一个查询的完整结果。

SQL*Plus 使用如下几个环境变量来控制页的大小；

NEWPAGE：每页开始和标题之间的空行数。

PAGESIZE：每页的行数。

LINESIZE：每行的字符数。

使用 SHOW 命令查看、使用 SET 命令设置这些环境变量。

设置标题和脚注：

SQL*Plus 提供了为每页和报表添加标题和脚注的命令。使用 TTITLE 和 BTITLE 命令设置页的标题和脚注，使用 REPHEADER 和 REPFOOTER 命令设置报告的标题和脚注。命令格式如下：

TTITLE position_clause char_value position_clause char_value ...

BTITLE position_clause char_value position_clause char_value ...

REPHEADER position_clause char_value position_clause char_value ...

REPFOOTER position_clause char_value position_clause char_value ...

其中：

position_clause：可以为标题和脚注指定位置，如下表：

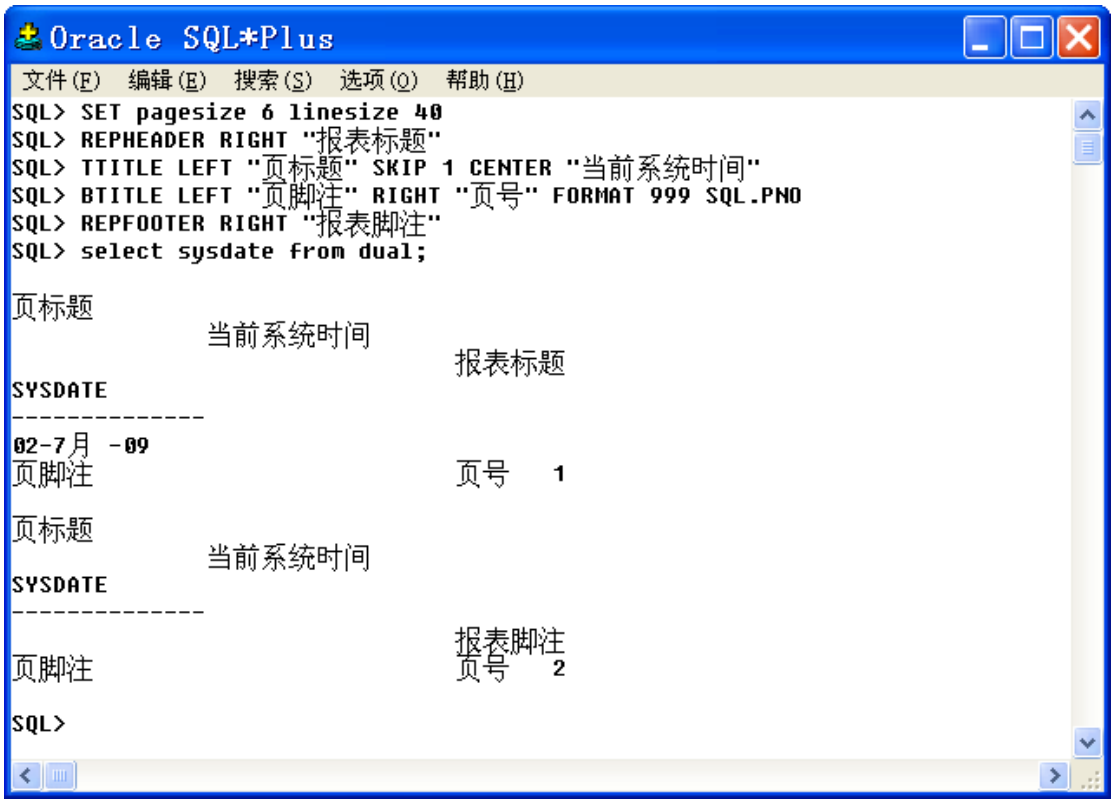
格式	功能
COL<n>	在第 n 列开始显示标题和脚注
SKIP<n>	跳过 n 行
TAB<n>	插入 n 个制表符（Tab）
LEFT	把标题和脚注在页中左对齐
RIGHT	把标题和脚注在页中右对齐
CENTER	把标题和脚注在页中居中

char_value：指定标题或脚注的文本内容。

例：

在页中居中显示标题，在脚注的右边插入页号（使用系统维护值 SQL.PNO）。

```
SQL> SET pagesize 6 linesize 40
SQL> REPHEADER RIGHT "报表标题"
SQL> TTITLE LEFT "页标题" SKIP 1 CENTER "当前系统时间"
SQL> BTITLE LEFT "页脚注" RIGHT "页号" FORMAT 999 SQL.PNO
SQL> REPFOOTER RIGHT "报表脚注"
SQL> select sysdate from dual;
```



查询、显示标题和脚注：

查询页与报表的标题和脚注：

SQL> TTITLE

SQL> BTITLE

SQL> REPHEADER

SQL> REPFOOTER

使用 off 或 on 来关闭或显示页与报表的标题和脚注：

SQL> TTITLE off

SQL> BTITLE off

SQL> REPHEADER off

SQL> REPFOOTER off

用脚本文件生成报表

脚本文件中用 REM[MARKS]输入单行注释，用/*…*/输入多行注释。

例：编辑脚本文件如下，然后在 SQL*Plus 环境中生成报表。脚本文件名为 test1.sql

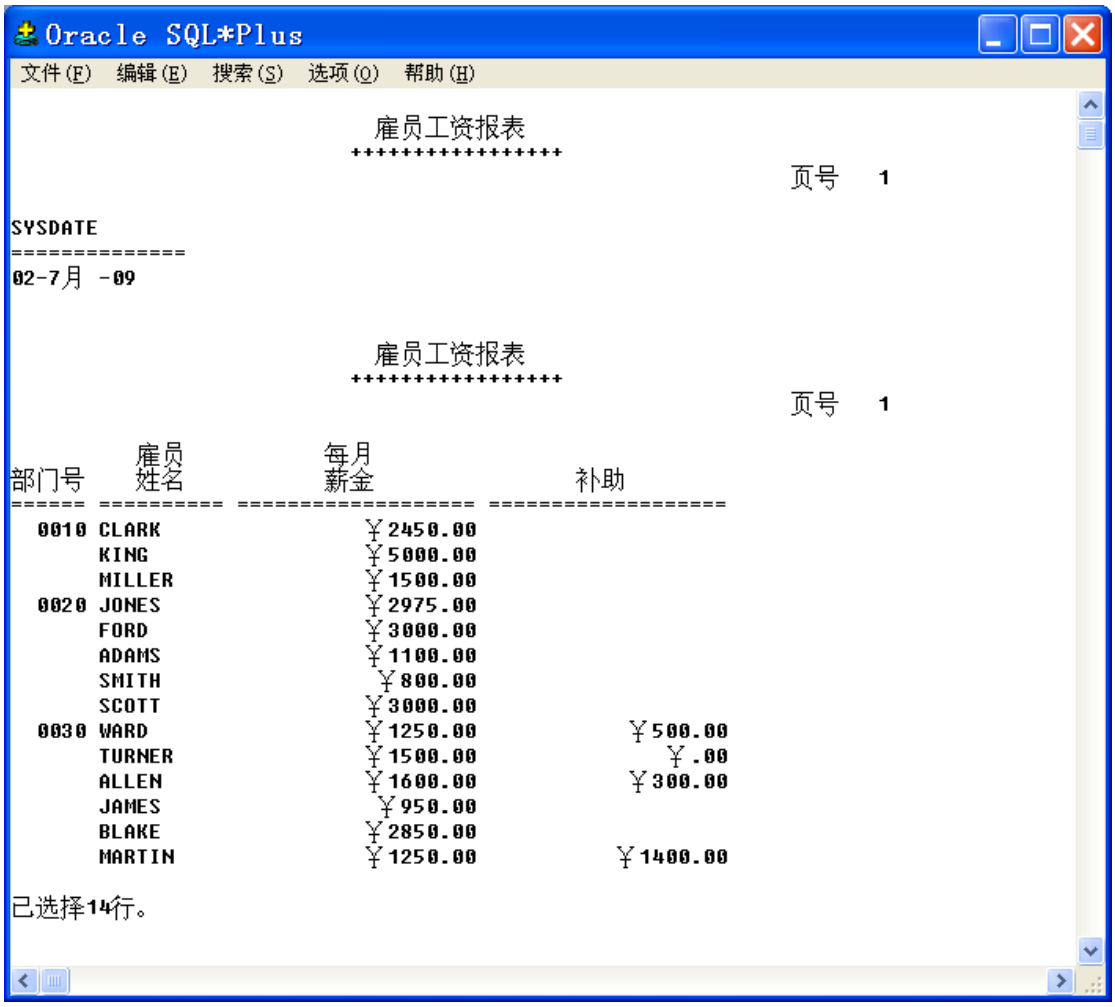
```
/*注释部分*/
REMARK mytest1.sql
REMARK 报表格式化举例
REMARK 2009－7－2
REMARK
/*设置环境变量*/
REMARK SET termout off
SET pagesize 55 linesize 70
SET underline =
/*格式化列*/
COLUMN deptno HEADING 部门号 JUSTIFY CENTER FORMAT 0000
COLUMN ename HEADING 雇员|姓名 JUSTIFY CENTER FORMAT A10
COLUMN sal HEADING 每月|薪金 JUSTIFY CENTER FORMAT L99999.99
COLUMN comm HEADING 补助 JUSTIFY CENTER FORMAT L99999.99
/*限制重复行和使用汇总行*/
BREAK ON deptno ON REPORT
COMPUTE SUM LABEL 小计：OF sal comm ON deptno
COMPUTE SUM LABEL 总计：OF sal comm ON REPORT
/*标题和脚注*/
REPHEADER CENTER "雇员工资报表" SKIP 1 -
      CENTER "+++++++++++++++++" SKIP 1 -
      RIGHT "页号" FORMAT 999 SQL.PNO SKIP 2
/*将输出保存到文件 E:\mysql\spoolfile1.list*/
SPOOL E:\mysql\spoolfile1.list
/*清除屏幕*/
CLEAR SCREEN
/*SQL 命令*/
```

```
SELECT deptno,ename,sal,comm from emp
ORDER BY deptno;
/*开始输出*/
SPOOL out
/*清除用户自定义的环境、列格式、输出*/
REMARK SET termout on
SET pagesize 24 linesize 80
SET underline -
CLEAR COLUMNS
CLEAR BREAKS
REPHEADER OFF
SPOOL off
```

运行该脚本文件如下：

```
SQL> START E:\mysql\test1.sql
```

如果不希望在屏幕上看到输出，可以添加 SET termout off 到脚本文件的开始出，添加 SET termout on 到脚本文件的结尾处。


Oracle SQL*Plus
文件(F) 编辑(E) 搜索(S) 选项(O) 帮助(H)
雇员工资报表
页号 1
SYSDATE
02-7月 -09
雇员工资报表
页号 1
部门号 雇员姓名 每月薪金 补助
0010 CLARK ￥2450.00
KING ￥5000.00
MILLER ￥1500.00
0020 JONES ￥2975.00
FORD ￥3000.00
ADAMS ￥1100.00
SMITH ￥800.00
SCOTT ￥3000.00
0030 WARD ￥1250.00 ￥500.00
TURNER ￥1500.00 ￥.00
ALLEN ￥1600.00 ￥300.00
JAMES ￥950.00
BLAKE ￥2850.00
MARTIN ￥1250.00 ￥1400.00
已选择14行。

第6章　SQL 语言基础

SQL（Structured Query Language，结构化查询语言）是一种在关系数据库中定义和操纵数据的标准语言。

本章只介绍数据操纵语言和事务控制语言。数据定义语言暂不讲。

SQL 语言共分为如下几大类：

1、数据定义语言（Data Definition Language，DDL）：用于定义、修改、删除数据库模式对象，进行权限管理等。DDL 语言包括创建、修改、删除或者重命名模式对象（create，alter，drop，rename）的语句，删除表中所有行但不删除表（truncate）的语句，管理权限（grant，revoke）的语句，审核数据库使用（audit,noaudit）的语句，以及在数据字典中添加说明（comment）的语句。使用 DDL 语言定义模式对象时，会将其定义保存在数据字典中。DDL 语言是自动提交的。

2、数据操纵语言（Data Manipulation Language，DML）：用于查询、生成、修改、删除数据库中的数据。DML 语言包含用于查询数据（select）、添加新行数据（insert）、修改线有行数据（Update）、删除现有行数据（delete）、合并数据（merge）的语句，以及查看一个 SQL 运行计划（explain plan）和锁定一个数据库表以限制访问（lock table）的语句。

3、事务控制（Transaction Control）：用于把一组 DML 语句组合起来形成一个事务并进行事务控制。使用这些语句可以把这些语句组合所做的修改保存起来（commit）或者撤销这些修改（rollback）。它包括在事务中设置一个保存点（savepoint）的语句，以实现可能出现的回溯操作，还包括设置事务属性（set transaction）的语句。

4、会话控制（Session Control）：用于控制一个会话（session，指从与数据库连接开始到断开之间的时间过程）的属性。它包括用于控制会话属性（alter session）的语句，以及切换角色（set role）的语句。

5、系统控制（System Control）：用于管理数据库的属性。只有一条语句：alter system。

6．1、查询数据

先设置页和行大小的环境变量：SQL> SET pagesize 39 linesize 120

显示用户下所有表和视图：SQL> select * from tab;

查询表结构（describe）：SQL> DESC dept;

取消重复行（distinct）：SQL>select DISTINCT deptno from emp;

使用表达式：

1、to_date

2、in

3、BETWEEN … AND…

4、Like

5、Order by [desc]

6、分组查询

7、GROUP BY

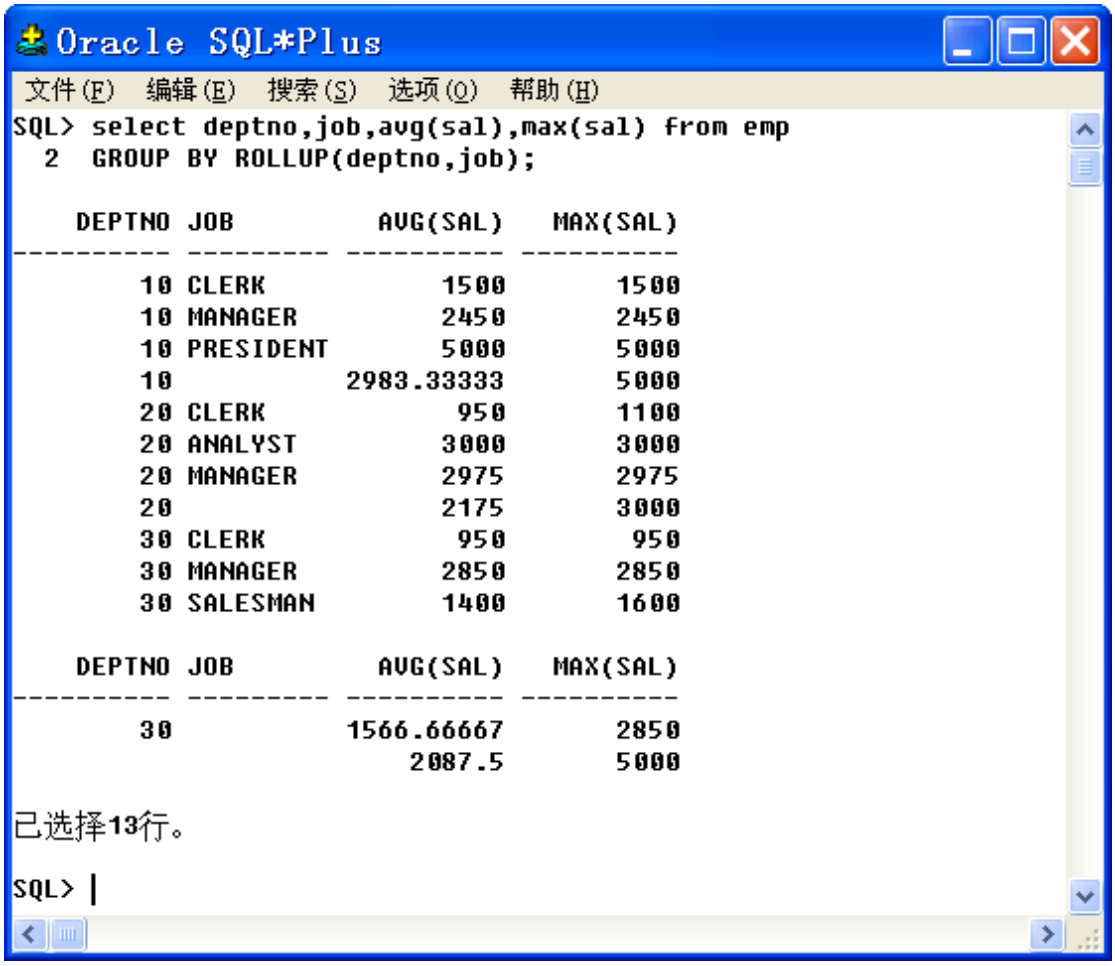
8、使用 ROLLUP 和 CUBE 限定词

当直接使用 GROUP BY 子句进行多列分组时，只能生成简单的数据统计结果。而在实际应用中，往往还希望生成横向、纵向的统计结果（两列分组时）。此时可以在 GROUP BY 子句中使用 ROLLUP 和 CUBE 限定词来生成超级组合(super aggregate)

ROLLUP 用于生成横向统计结果。

例：显示每个部门、每种岗位的平均工资和最高工资及其横向统计结果。

```
SQL> select deptno,job,avg(sal),max(sal) from emp
  GROUP BY ROLLUP(deptno,job);
```



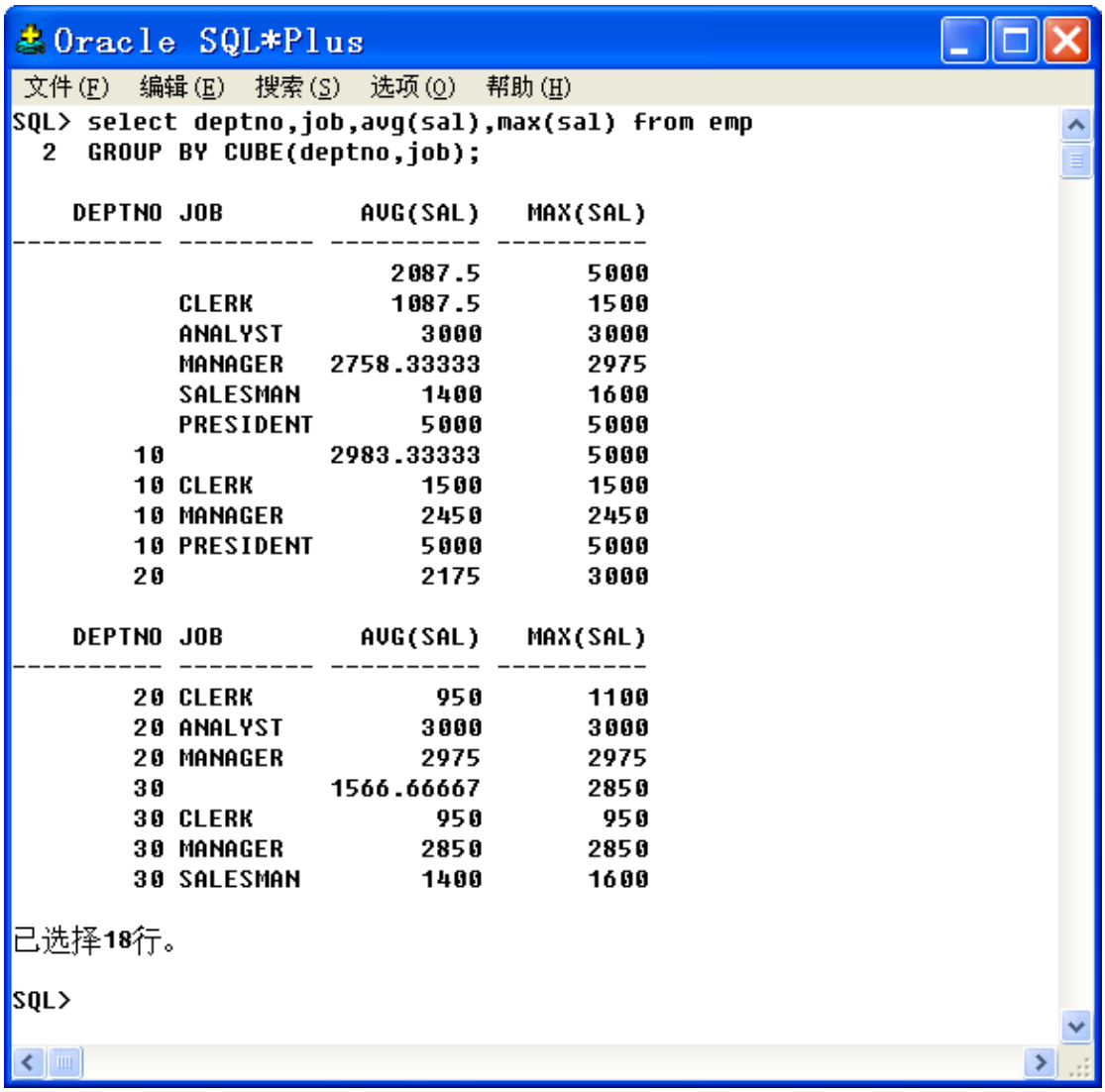
其结果形式如下表

部门|岗位	CLERK	MANAGER	PRESIDENT	ANALYST	SALESMAN	横向统计
10	1300	2450	5000			2916.66667
20	950	2975		3000		2175
30	950	2850			1400	1566.66667
合计						2073.21429

CUBE 用于生成纵向统计结果。

列：显示每个部门、每种岗位的平均工资和最高工资及其纵向统计结果。

```
SQL> select deptno,job,avg(sal),max(sal) from emp
     GROUP BY CUBE(deptno,job);
```



其结果形式如下表：

部门|岗位	CLERK	MANAGER	PRESIDENT	ANALYST	SALESMAN	横向统计
10	1300	2450	5000			2916.66667
20	950	2975		3000		2175
30	950	2850			1400	1566.66667
合计	1037.5	2758.3333	5000	3000	1400	2073.21429

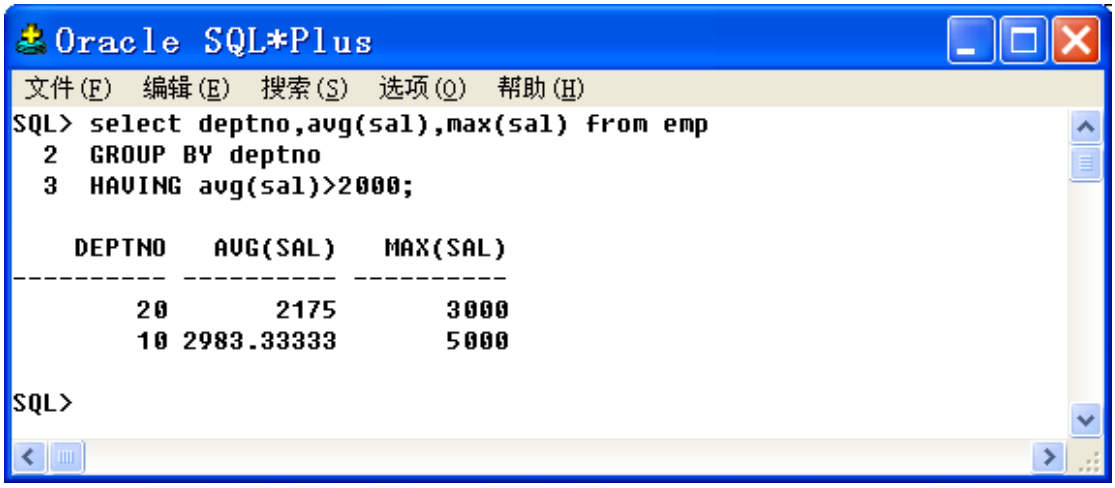
9、HAVING 子句：限制（或过滤）分组处理后的结果显示，并且 HAVING 子句必须跟在 GROUP BY 子句之后。

例：显示平均工资高于 2000 的部门编号、平均工资和最高工资。

```
SQL> select deptno,avg(sal),max(sal) from emp
```

GROUP BY deptno

HAVING avg(sal)>2000;



限制分组显示结果时，必须要使用 HAVING 子句，而不能在 WHERE 子句中使用组处理函数来限制分组显示结果（即，不能在 WHERE 子句中使用组处理函数）。如果将上面的 HAVING avg(sal)>2000 改写成 WHERE avg(sal)>2000 就会出错。

因为 WHERE 在进行组合处理之前先过滤数据行，而 HAVING 则是在组合处理之后再过滤数据组。数据库在从表中查询数据时并不知道组处理函数时什么，组处理要在所有行都查询出来并分好组之后才进行的。

10、 相等连接查询（＝）

11、 合并查询

UNION：获得两个结果集的并集，并自动去掉重复行，而且会以第一列的记过进行排序。

UNION ALL：获得两个结果集的并集，但不会自动去掉重复行，并且不会对结果进行排序。

INTERSECT：获得两个结果集的交集，只会显示同时存在于两个结果集中的数据，并且会按第一列的结果进行排序。

MINUS：获得两个结果集的差集，只会显示在第一个结果集中存在、但在第二个结果集中不存在的数据，并且会按第一列的结果进行排序。

12、 子查询

在 SELECT 语句和 DML 语句中的子查询不能使用 ORDER BY 子句，它只能用在

最外面的父查询中。但是在 DDL 语句中的子查询可以使用 ORDER BY 子句。

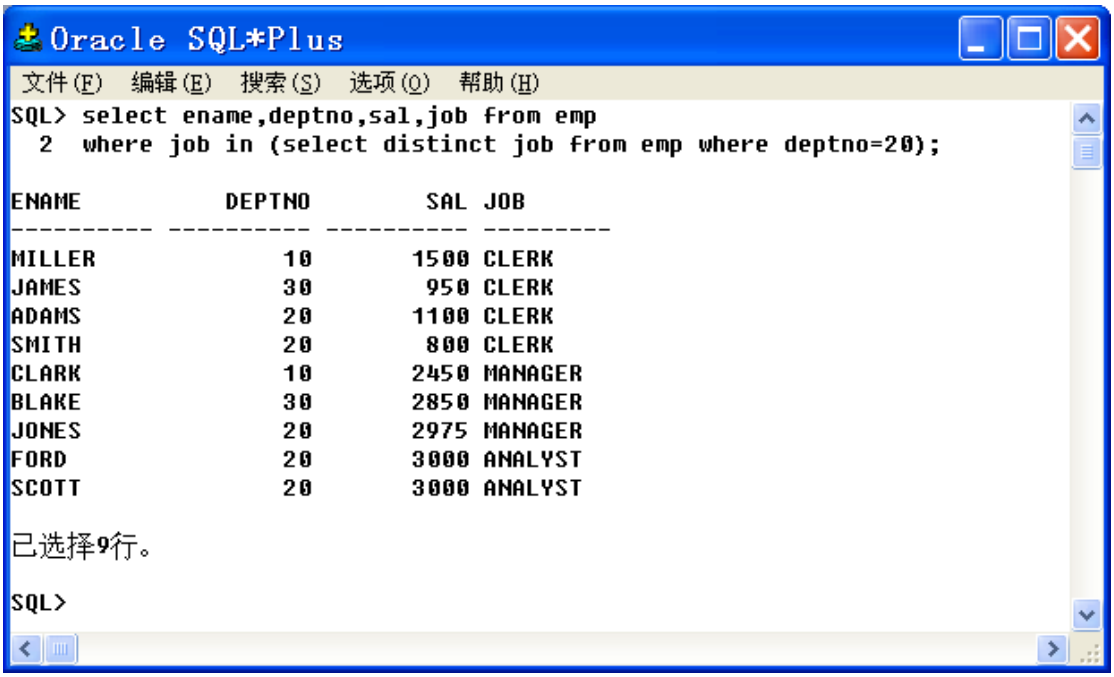
1)、单行子查询

2)、多行子查询：

多行子查询可以返回多行数据。在 WHERE 子句中使用多行子查询时，必须使用多行运算符（IN，NOT IN，EXISTS，NOT EXISTS，ALL，ANY）。如果不能确认子查询会返回多少行，则在子查询中使用多行运算符比使用单行运算符更安全。

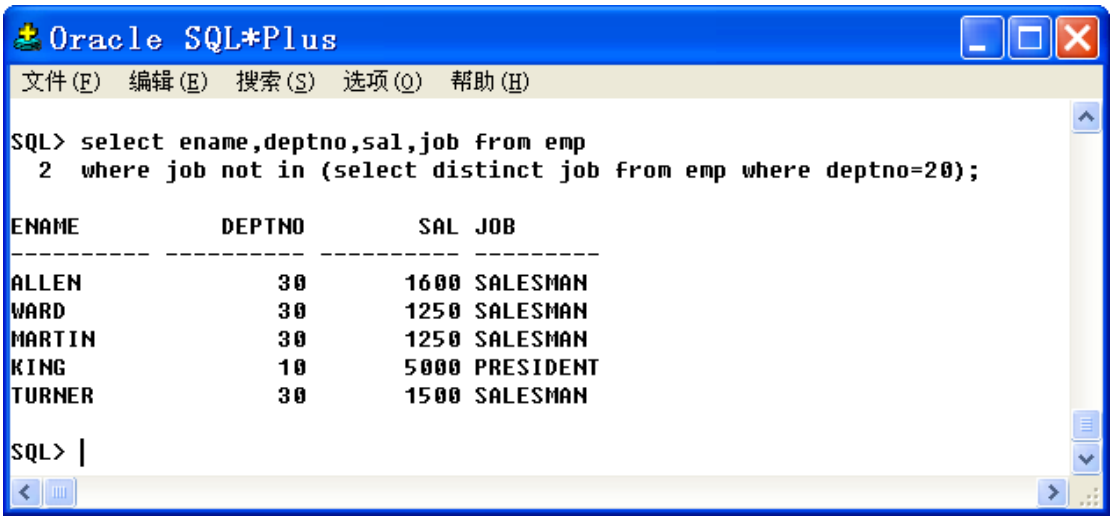
IN：显示与部门编号为 20 的岗位相同的雇员信息：

```
SQL> select ename,deptno,sal,job from emp
  2  where job in (select distinct job from emp where deptno=20);
```



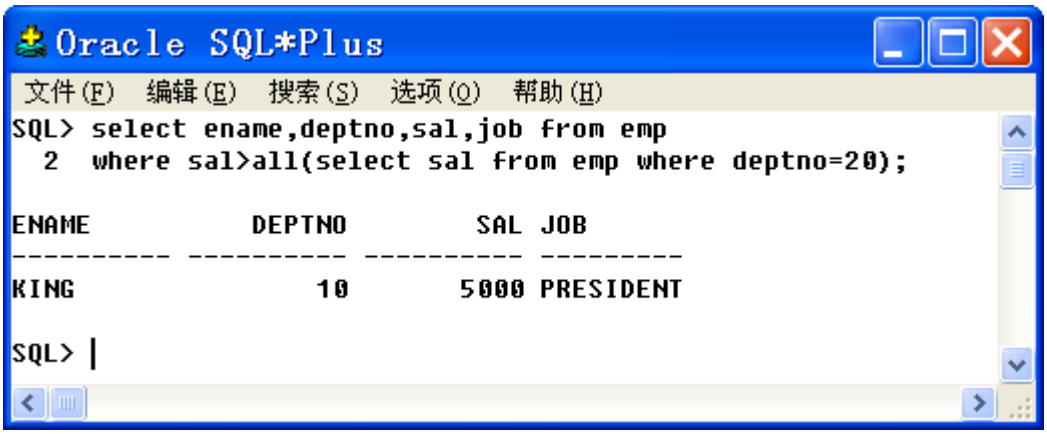
NOT IN：显示不与部门编号为 20 的岗位相同的雇员信息：

```
SQL> select ename,deptno,sal,job from emp
  2  where job not in (select distinct job from emp where deptno=20);
```



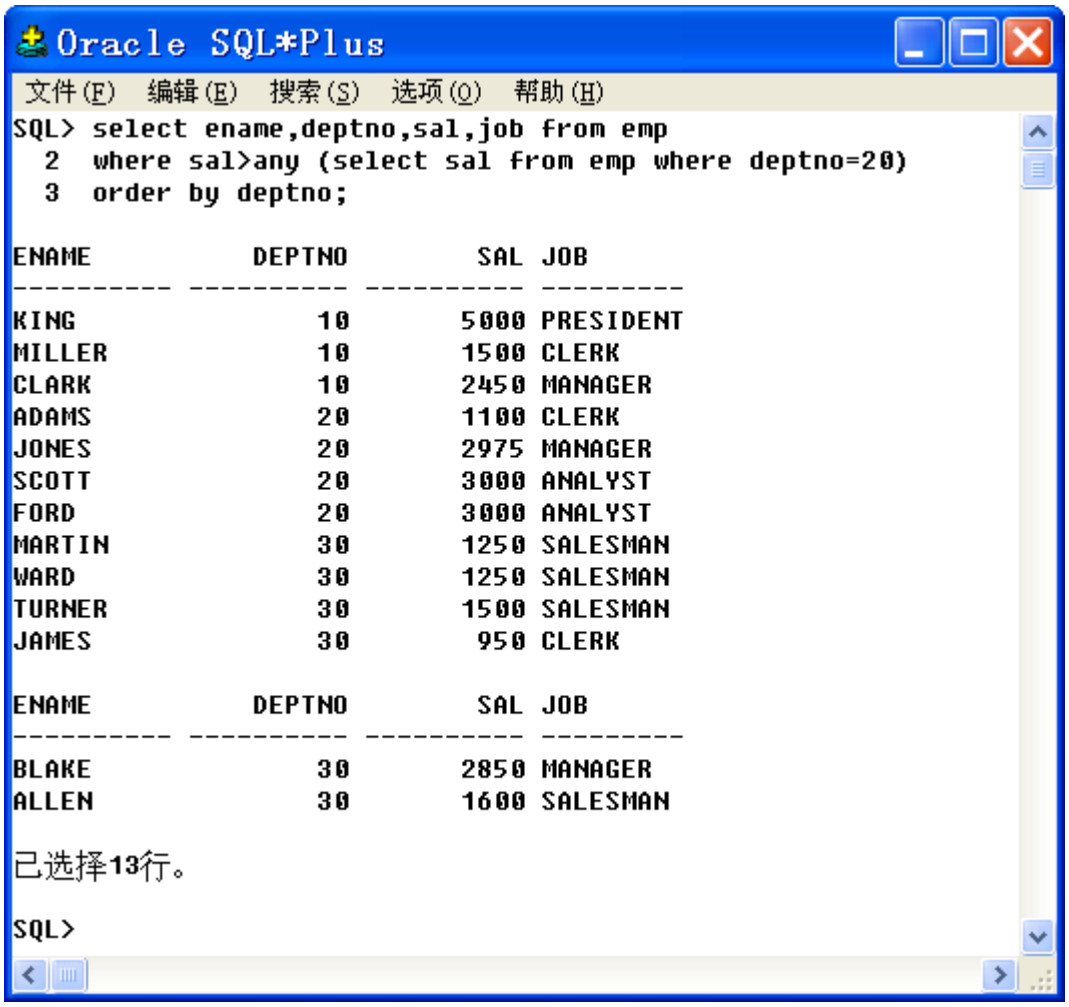
ALL：显示高于部门编号为 20 的所有雇员的工资的雇员信息（肯定不包括部门编号为 20 的所有雇员）：

```
SQL> select ename,deptno,sal,job from emp
  2  where sal>all(select sal from emp where deptno=20);
```



ANY：显示高于部门编号为 20 的任意雇员的工资的雇员信息（可能包括部门编号为 20 的某些雇员）：

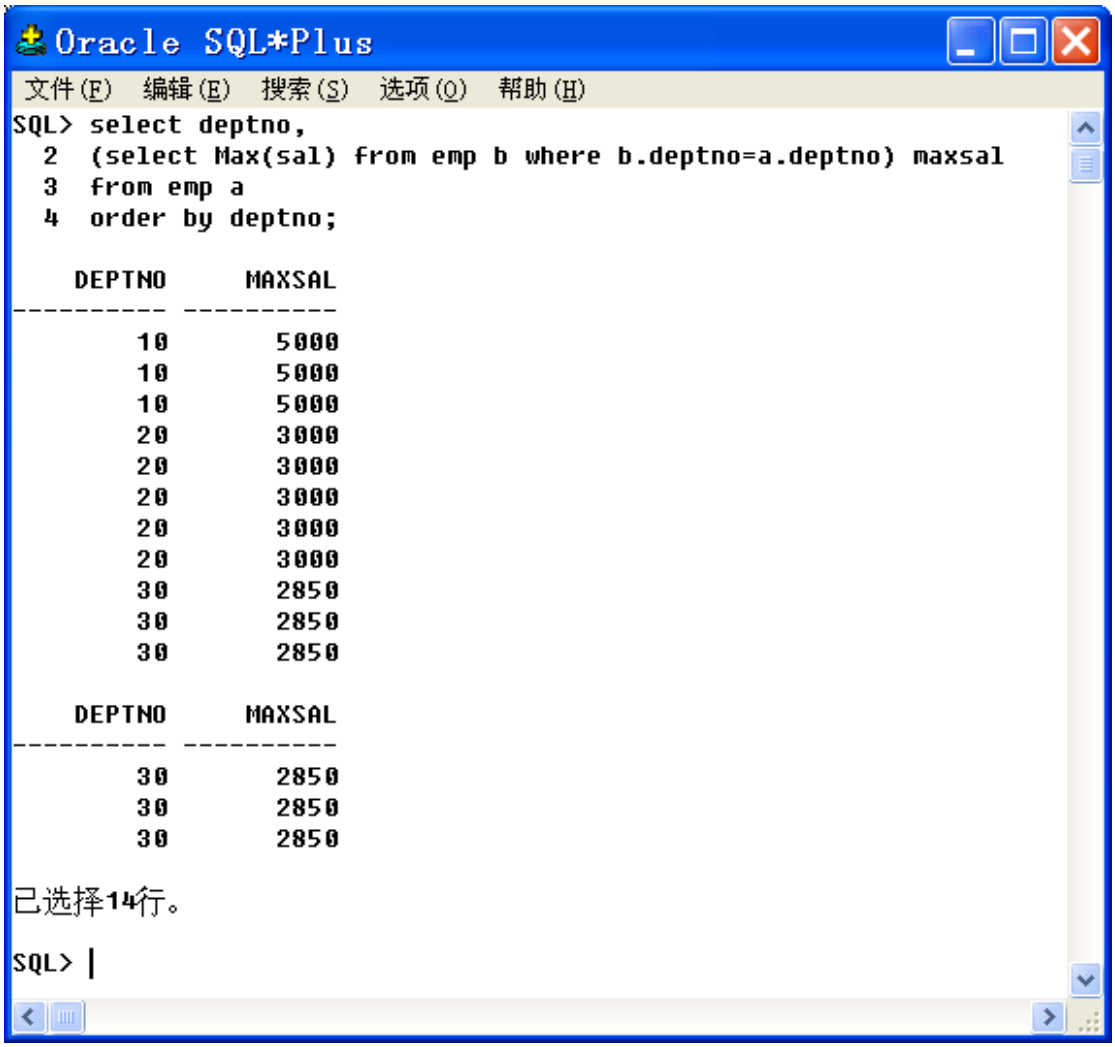
```
SQL> select ename,deptno,sal,job from emp
  2  where sal>any (select sal from emp where deptno=20)
  3  order by deptno;
```



3)、相关子查询：引用了父查询中某些表或某些列的子查询（但父查询不能引用子查询中的表或列）。父查询可以时一条 SELECT，UPDATE，DELETE 语句。

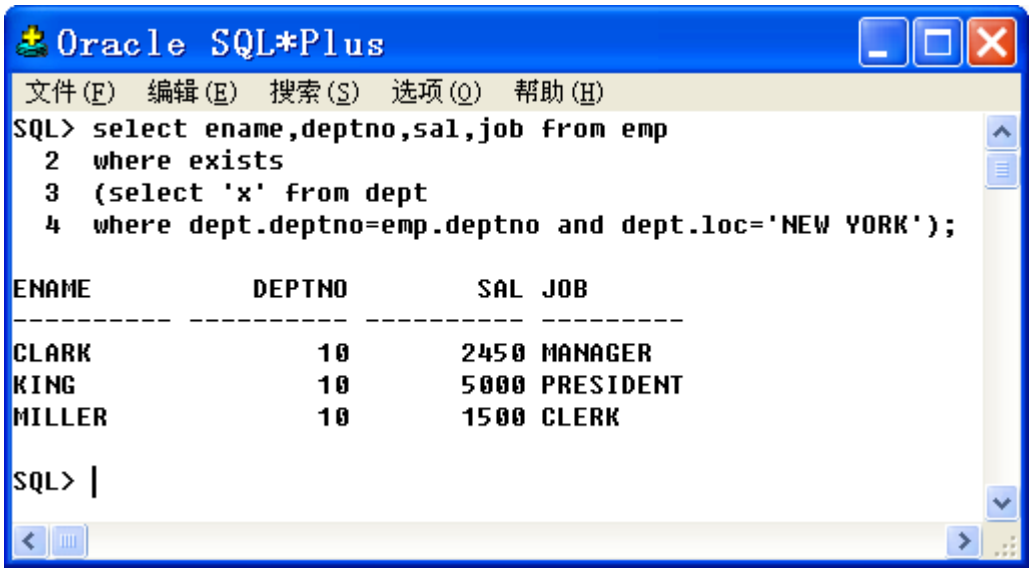
例：显示每个部门的最高工资的雇员信息。

```
SQL> select deptno,
  2  (select Max(sal) from emp b where b.deptno=a.deptno) maxsal
  3  from emp a
  4  order by deptno;
```



EXISTS：显示工作在 NEW YORK 的雇员信息。EXISTS 会根据具体情况，产生相关行是否存在的逻辑值（true 或 false），使得子查询的选择列没有实际意义而被糊了，因此可以使用一个虚拟的列名（’x’或*）。

```
SQL> select ename,deptno,sal,job from emp
  2  where exists
  3  (select 'x' from dept
  4  where dept.deptno=emp.deptno and dept.loc='NEW YORK');
```



4)、标量子查询：只返回单行单列数据。标量子查询可以用在大多数使用一个列名或者表达式的地方，如作为一个参数用在一个单行函数、INSERT 语句的 VALUES 子句、ORDER BY 子句、WHERE 子句、SELECT 的选择列中。但不能用在 GROUP BY 子句和 HAVING 子句中。

例：显示每个部门的最高工资的雇员信息：

```
SQL> select distinct deptno,
  2  (select Max(sal) from emp b where b.deptno=a.deptno) maxsal
  3  from emp a
  4  order by deptno;

    DEPTNO      MAXSAL
---------- ----------
        10        5000
        20        3000
        30        2850
```

5)、多列子查询：前面的单行子查询、多行子查询都是指返回单列数据。而多列子查询是指返回多列数据的子查询。当多列子查询返回单行数据时，在 WHERE 子句中可以使用单行操作符（如＝），当返回多行数据时，则必须使用多行操作符（如 IN）。多行子查询一般用于比较多个列的数据，集可以成对比较（要求比较的多个列的数据必须同时匹配），也可以非成对比较（只需要有一个列匹配就

行)。

例:显示与 SMITH 部门和岗位完全相同的所有雇员信息,以说明成对比较的意思。

```
SQL> select ename,deptno,sal,job from emp
  2  where (deptno,job)=
  3  (select deptno,job from emp where ename='SMITH');

ENAME          DEPTNO        SAL JOB
---------- ---------- ---------- ---------
SMITH              20        800 CLERK
ADAMS              20       1100 CLERK
```

例:显示岗位或者管理员匹配与部门编号为 20 的所有雇员的信息,以说明非成对比较的意思:

```
SQL> select ename,deptno,sal,job,mgr from emp
  2  where job in(select job from emp where deptno=20)
  3  and mgr in(select mgr from emp where deptno=20)
  4  order by deptno;

ENAME          DEPTNO        SAL JOB              MGR
---------- ---------- ---------- --------- ----------
CLARK              10       2450 MANAGER         7839
JONES              20       2975 MANAGER         7839
ADAMS              20       1100 CLERK           7788
SCOTT              20       3000 ANALYST         7566
FORD               20       3000 ANALYST         7566
SMITH              20        800 CLERK           7902
BLAKE              30       2850 MANAGER         7839

已选择 7 行。
```

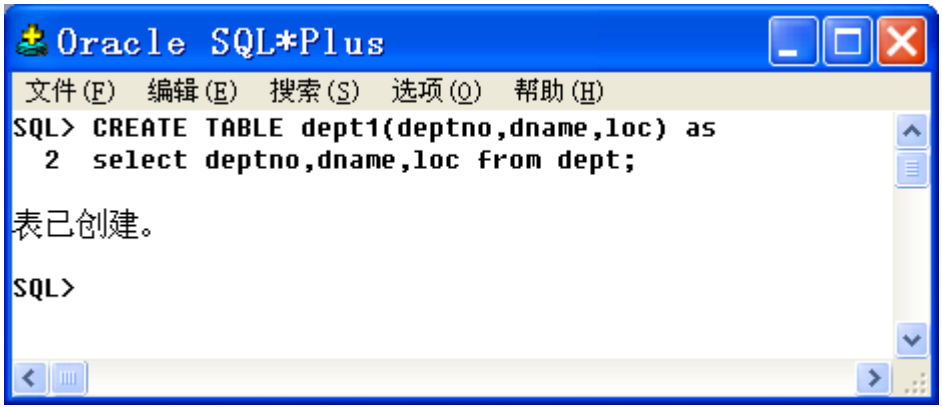
6．2、在 DDL 语句中使用子查询

一、在 CREATE TABLE 语句中使用子查询

通过在 CREATE TABLE 语句中使用子查询，可以在创建新表的同时插入表中的数据。

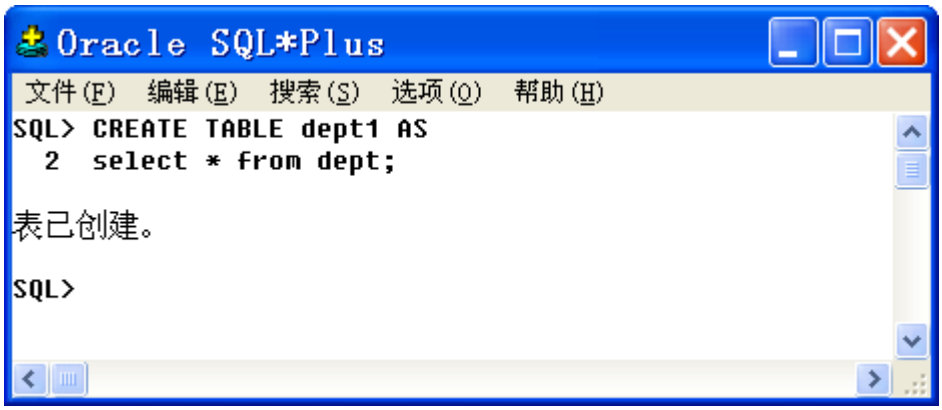
例：按 dept 表建立一个新表 dept1，并将 dept 表中的数据插入到 dept1 表中。

```
SQL> CREATE TABLE dept1(deptno,dname,loc) as
  2  select deptno,dname,loc from dept;
```



如果时创建所有的列，则可以用*简化该语句。下面仍然时建立 dept1 表，只不过将 dept 表中的数据全部插入到 dept1 表中。

```
SQL> CREATE TABLE dept1 AS
  2  select * from dept;
```



二、 在 CREATE VIEW 中使用子查询

在创建视图时，必须指定视图所对应的子查询。下面的例子创建视图 dept_20，功能时查询部门编号为 20 的雇员信息。

```
SQL> CREATE OR REPLACE VIEW dept_20 AS
```

```
2  select * from emp where deptno=20 order by empno;
```

6．3、事务控制

事务（transaction）涉及到数据的并发访问，用于确保数据库中数据的一致性。它是用户定义的一组操作序列，有一组相关的 SQL 语句组成，这些 SQL 语句只限于 DML 命令，而 DDL 和 DCL 语句时不能被回退的（事务对其不起作用，它们会自动提交事务）。事务的基本语句包括：设置事务、提交事务、设置保存点、回退部分或全部事务。

一、设置事务：

设置事务实际上是对即将开始的事务的性质进行一种控制。

1)、SET TRANSACTION ISOLATION LEVEL READ COMMITTED

用于设置语句级一致事务或读已提交事务，这是默认的事务。

2)、SET TRANSACTION READ ONLY

设置只读事务，在只读事务中不允许执行 DML 语句更改数据，因此只读事务只能使用下列语句：

SELECT（不包括 FOR UPDATE 子句）

LOCK TABLE

SET ROLE

ALTER SYSTEM

ALTER SESSION

3)、SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

设置事务级一致性事务或顺序事务。顺序事务与只读事务的区别在于，在顺序事务中允许执行 DML 语句。

4)、SET TRANSACTION USE ROLLBACK SEGMENT rollback_segment_1

设置该事务使用一个回退段 rollback_segment_1

在默认情况下，Oracle 使用一个 round-robin 算法来向事务分配回退段。因此一个很大的事务可能被分配到任何回退段，并使该回退段的大小明显增加。这种动态空间管理对于系统性能和磁盘空间的使用都会造成负面影响，甚至使用回

退段的空间需求超过该回退段所在的表空间的大小，而使数据库操作失败。为了避免随即地向任何回退段分配较大事务的情况，可以使用此命令。

二、提交事务（commit）：

提交事务，即通知 DBMS 将该事务对数据库所做的修改（在内存的数据库缓存中）全部保存到操作系统文件中（至少要先被保存到重做日志文件中，然后再保存到数据文件中），使其生效，实现该事务提交的成功（只要保存到重做日志文件中，该提交就算成功了）。

如下情况会自动提交：

*执行 DDL 语句（如 create table，alter table，alter system）后。

*执行 DCL 语句（如 grant，revoke）后。

*退出 SQL*Plus 后。

三、设置保存点（savepoint）：SAVEPOINT spname

四、回退全部事务（rollback）：ROLLBACK

五、回退部分事务（rollback）：ROLLBACK TO spname

6．3、SQL 函数

补充：

1、单行转换函数：

TO_TIMESTAMP_TZ(c[,fmt])：将符合 fmt 指定的特定日期格式的字符串 c 转换成 TIMESTAMP WITH TIME ZONE 类型的数据。

例：

```
SQL> select to_timestamp_tz('2003-04-05','YYYY-MM-DD') from dual;

TO_TIMESTAMP_TZ('2003-04-05','YYYY-MM-DD')
---------------------------------------------------------------------------
------
05-4 月 -03 12.00.00.000000000 上午 +08:00
```

TO_YMINTERVAL(c)：将符合特定格式的字符串 c 转换成 INTERVAL YEAR TO MONTH 类型的数据。

例：

```
SQL> select sysdate,
  2  sysdate+to_yminterval('01-03') "+15 months",
  3  sysdate+to_yminterval('00-03') "+3 months" from dual;

SYSDATE         +15 months      +3 months
--------------- --------------- ---------------
03-7 月 -09      03-10 月-10      03-10 月-09
```

第 7 章 PL/ SQL 语言基础

PL/SQL 有很多大量的内置程序包，如：DBMS_OUTPUT 程序包、UTL_FILE 程序包、TEXT_TO 程序包能够实现输入输出操作（包括屏幕读写、文件读写）。DBMS_ALTER 程序包能够实现数据库报警，DBMS_PIPE 可以管理数据库管道，他们都允许会话间的通信。DBMS_AQ 程序包能够实现高级队列的消息发送和接收。DBMS_AQADM 程序包能够管理和配置高级队列，如创建队列表、创建对了、授予访问队列的权限。DBMS_JOB 程序包能够实现任务调度服务和数据库作业管理。DBMS_LOB 程序包能够实现对大对象的操作。

再准备编写 PL/SQL 前，必须首先明确如下内容：

*如果要创建数据库级触发器，必须具有 ADMINISTER DATABASE TRIGGER 权限。

*确认 SYSTEM 表空间有足够的空间存放程序。

块的类型：匿名块，命名块，子程序，触发器（除对表和视图操作外，还可以对数据库级操作，如关闭、启动、登陆、退出数据库、创建对象、修改对象、删除对象等（称为系统触发器））。

7．1、常量、变量、格式、定义等

再 PL/SQL 中，每行只能声明一个常量或变量。

声明各种类型变量方法如下：

声明	说明
X Date	
X Char:= ‘0’	
X Constant Number(8.2):=5000.00	声明一个 NUMBER 数据类型的常量，并赋初值。
X Constant Number(8.5):=3.14159*1**2	声明一个 NUMBER 数据类型的常量，并用表达式给它赋初值。

数据类型的定义：

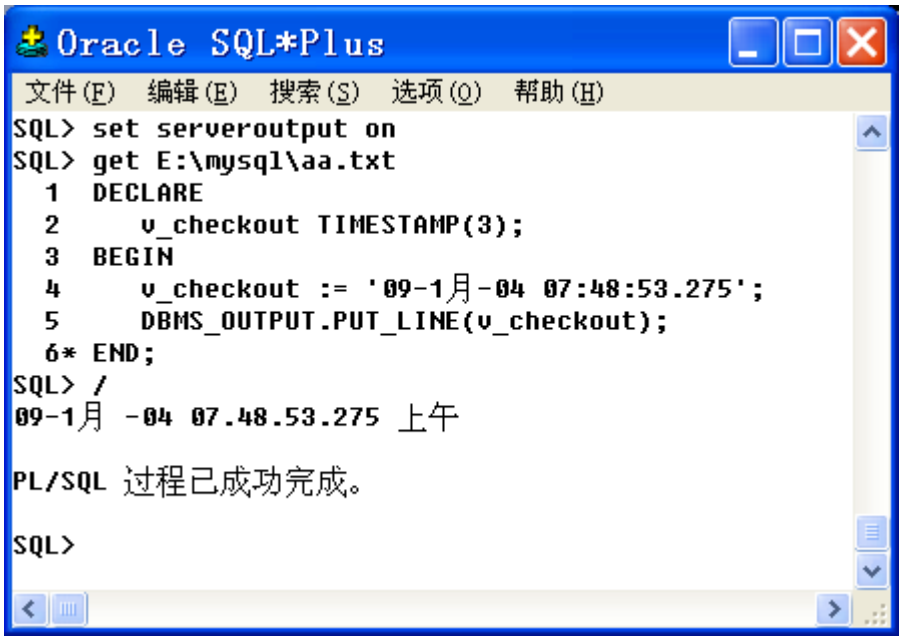
BOOLEAN：用于定义布尔（逻辑）变量。其取值范围是 TRUE，FALSE，NULL。该数据类型是 PL/SQL 数据类型，列表不能采用该数据类型，可以应用 CHAR 或 CHAR(1)来代替，不能将 BOOLEAN 变量的值插入到表列中，也不能将表列的值选取到布尔变量中。布尔变量一般用于 PL/SQL 的控制结构中。

TIMESTAMP(s)：是 Date 类型的扩展，还包括了上午、下午。其中 s 是可选的，它表示秒部分的小数位位数，取值范围是 0～9。默认的 TIMESTAMP 格式是由初始化参数 NLS_TIMESTAMP_FORMAT 来设置的。

例：

```
SQL> set serveroutput on
SQL> get E:\mysql\aa.txt
  1  DECLARE
  2     v_checkout TIMESTAMP(3);
  3  BEGIN
  4     v_checkout := '09-1 月-04 07:48:53.275';
  5     DBMS_OUTPUT.PUT_LINE(v_checkout);
  6* END;
SQL> /
```

09-1 月 -04 07.48.53.275 上午



INTERVAL YEAR(y) TO MONTH：用于存储和操作年和月之间的时间间隔。其中 y 是可选的，表示年部分的位数，其取值范围是 0～4，默认为。

例：定义一个变量，并对其初始化。

```
SQL> declare
  2     v_lifetime interval year(3) to month;
  3  begin
  4     v_lifetime := interval '101-3' year to month;
  5     dbms_OUTPUT.PUT_LINE('1:'||v_lifetime);
  6  end;
  7  /
1:+101-03
PL/SQL 过程已成功完成。
```

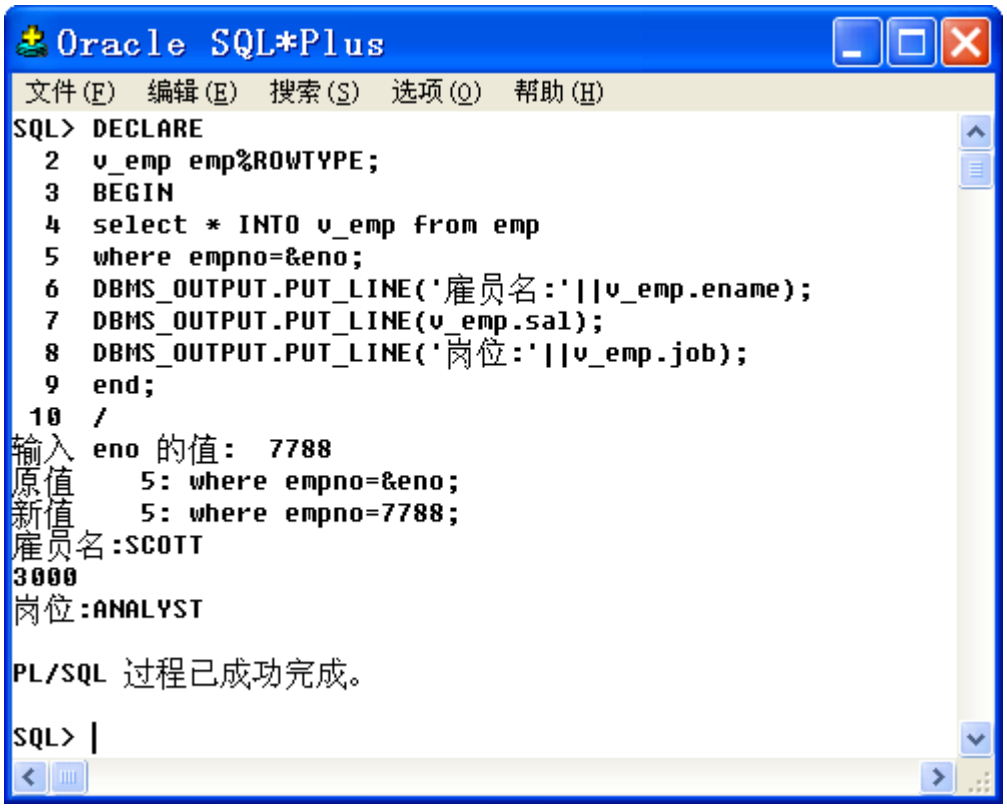
INTERVAL DAY(d) TO SECOND(s) SECOND：用于存储和操纵天数、小时、分钟和秒之间的时间间隔。其中 d 和 s 是可选的，d 表示天部分用几位数字表示，s 表示秒部分的位数，取值范围是 0～9，默认值分别是 2 和 6。

/*================下面为复合类型数据变量==================*/

%ROWTYPE:为了使一个变量的数据类型与一个参照表中记录的数据类型相对应、相一致。当被参照的那个表的列及其数据类型改变了之后，这个新定义的变量的数据类型也自动改变了，易于保持一致。当不能确切知道被参照的那个表的结构及其数据类型时，就只能采用这种方法定义变量的数据类型。如果一个表由较多了列，使用此种定义方法比使用%TYPE 来定义记录要简洁的多，且不易出错。

例：

```
SQL> DECLARE
  2  v_emp emp%ROWTYPE;
  3  BEGIN
  4  select * INTO v_emp from emp
  5  where empno=&eno;
  6  DBMS_OUTPUT.PUT_LINE('雇员名:'||v_emp.ename);
  7  DBMS_OUTPUT.PUT_LINE(v_emp.sal);
  8  DBMS_OUTPUT.PUT_LINE('岗位:'||v_emp.job);
  9  end;
 10  /
输入 eno 的值:  7788
原值    5: where empno=&eno;
新值    5: where empno=7788;
雇员名:SCOTT
3000
岗位:ANALYST

PL/SQL 过程已成功完成。
```



RECORD：定义记录类型。类似于 C 语言中的“结构”类型，PL/SQL 提供了几个相关的、分离的基本数据类型的变量组成一个整体的方法。在使用记录类型变量时，需要在声明部分先定义记录的组成和记录变量，然后在执行部分引用该记录变量本身或其中的成员。

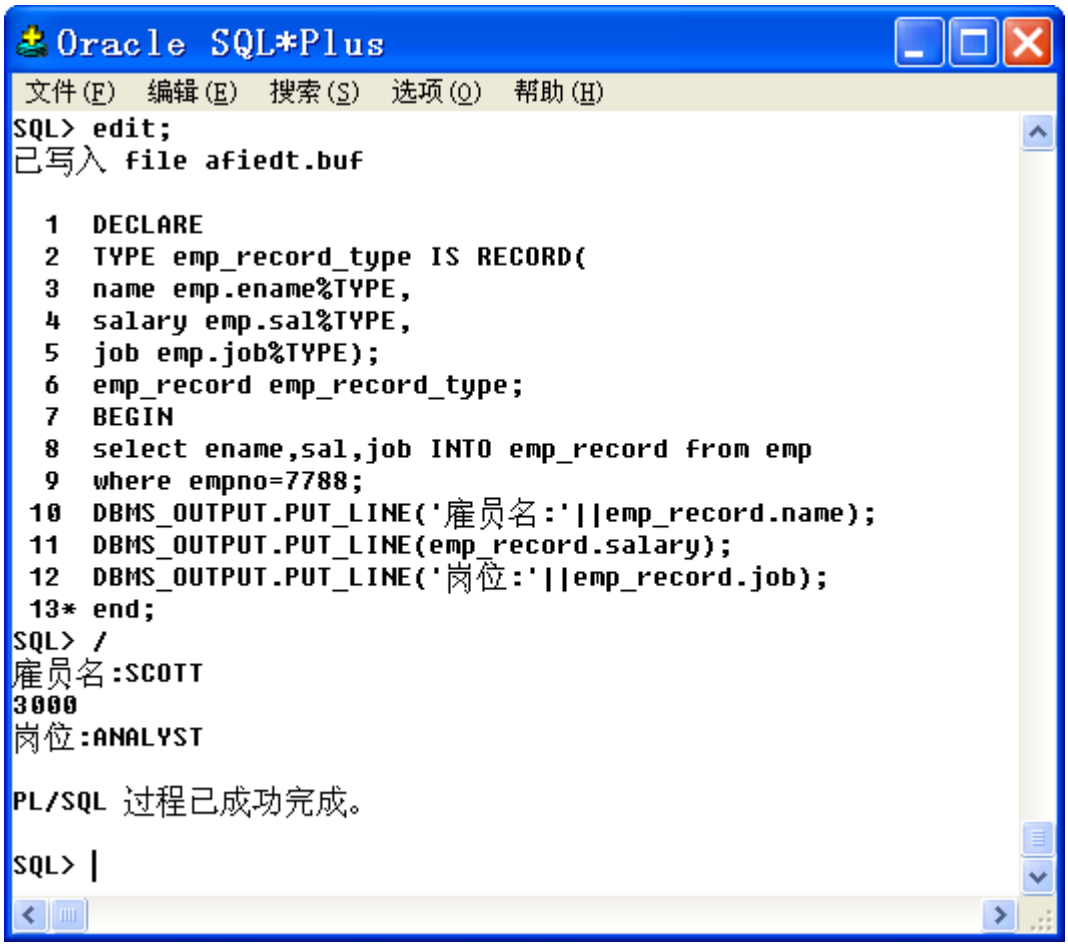
语法如下：

```
TYPE record_type IS RECORD(
V1 data_type1,
v2 data_type2,
vn data_typen);
```

例：

```
SQL> edit;
已写入 file afiedt.buf

  1  DECLARE
  2  TYPE emp_record_type IS RECORD(
```

```
  3  name emp.ename%TYPE,
  4  salary emp.sal%TYPE,
  5  job emp.job%TYPE);
  6  emp_record emp_record_type;
  7  BEGIN
  8  select ename,sal,job INTO emp_record from emp
  9  where empno=7788;
 10  DBMS_OUTPUT.PUT_LINE('雇员名:'||emp_record.name);
 11  DBMS_OUTPUT.PUT_LINE(emp_record.salary);
 12  DBMS_OUTPUT.PUT_LINE('岗位:'||emp_record.job);
 13* end;
SQL> /
雇员名:SCOTT
3000
岗位:ANALYST
```



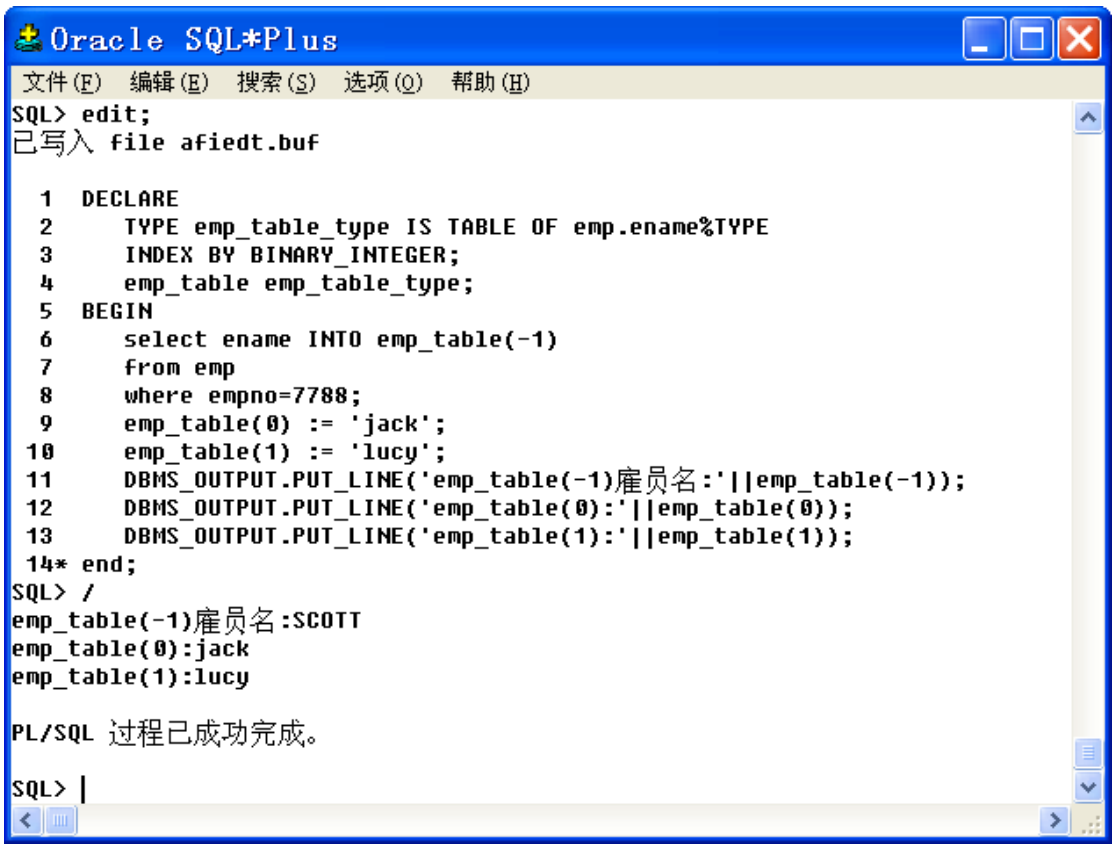
TABLE：定义表类型。类似于 C 语言中的数组，PL/SQL 提供了将几个相同的基本数据类型的变量组成一维表、用下标来访问的方法，在使用表类型变量时，需要在声明部分先定义表、表变量，然后在执行部分引用该记录变量本身或其中的成员。语法如下：

```
TYPE table_type IS TALBE OF data_type INDEX BY BINARY_INTEGER;
```

例：

```
SQL> edit;
已写入 file afiedt.buf

  1  DECLARE
  2     TYPE emp_table_type IS TABLE OF emp.ename%TYPE
  3     INDEX BY BINARY_INTEGER;
  4     emp_table emp_table_type;
  5  BEGIN
```

```
  6      select ename INTO emp_table(-1)
  7      from emp
  8      where empno=7788;
  9      emp_table(0) := 'jack';
 10      emp_table(1) := 'lucy';
 11      DBMS_OUTPUT.PUT_LINE('emp_table(-1)雇员名:'||emp_table(-1));
 12      DBMS_OUTPUT.PUT_LINE('emp_table(0):'||emp_table(0));
 13      DBMS_OUTPUT.PUT_LINE('emp_table(1):'||emp_table(1));
 14* end;
SQL> /
emp_table(-1)雇员名:SCOTT
emp_table(0):jack
emp_table(1):lucy

PL/SQL 过程已成功完成。
```



7．2、流程控制

一、条件控制：

1)、IF...THEN...END IF 结构

```
IF 条件表达式 THEN
    语句段
END IF;
```

2)、IF...THEN...ELSE...END 结构

```
IF 条件表达式 THEN
    语句段 1
ELSE
    语句段 2
END IF;
```

3)、CASE 结构

```
CASE
    WHEN 条件表达式 1 THEN
        语句段 1
    WHEN 条件表达式 2 THEN
        语句段 2
    ..............
    ELSE
        语句段 n
END CASE;
```

CASE 结构的另一种语法结构如下所示：

```
CASE 条件表达式
    WHEN 条件表达式结果 1 THEN
        语句段 1
    WHEN 条件表达式结果 2 THEN
        语句段 2
    ..............
```

ELSE

语句段 n

END CASE;

7．3、循环控制

1)、LOOP...END LOOP

[<<循环标签>>]

LOOP

语句段

END LOOP [循环标签];

LOOP 循环语句自身没有循环条件测试，即程序一旦进入循环体，将重复执行其中的语句，所以在循环体中只能使用 EXIT 语句来结束循环。EXIT 语句由如下两种语法格式：

EXIT [循环标签]

EXIT [循环标签] WHEN 条件表达式

第一种 EXIT 语句使程序立即跳出循环体，所以一般它需要和条件语句配合使用，才能控制何时跳出循环。

第二种 EXIT 语句只有当 WHEN 子句中的表达式的结果为 TRUE 时，才控制跳出循环。

当在 EXIT 语句中指定循环标签时，它将控制程序跳出循环标签指定的循环体。当程序出现循环嵌套时，在 EXIT 语句中指定循环标签就显得特别有用，如果不指定循环标签，EXIT 语句就使程序跳出当前的循环体。

注意：*EXIT 语句只能够退出循环体，不能够退出 PL/SQL 块，退出 PL/SQL 块应该使用 RETURN 语句。

2)、WHILE...LOOP...END LOOP

[<<循环标签>>]

WHILE 条件表达式 LOOP

语句段

END LOOP [循环标签];

3)、FOR...LOOP...END LOOP 结构

[<<循环标签>>]

FOR 循环变量 IN [REVERSE] 初始表达式...终值表达式 LOOP

语句段

END LOOP [循环标签];

FOR 循环的执行流程如下：

Step1 先计算初值表达式和终值表达式的值。

Step2 将初值表达式（当使用了 REVERSE 是终值表达式）的值赋予循环变量。

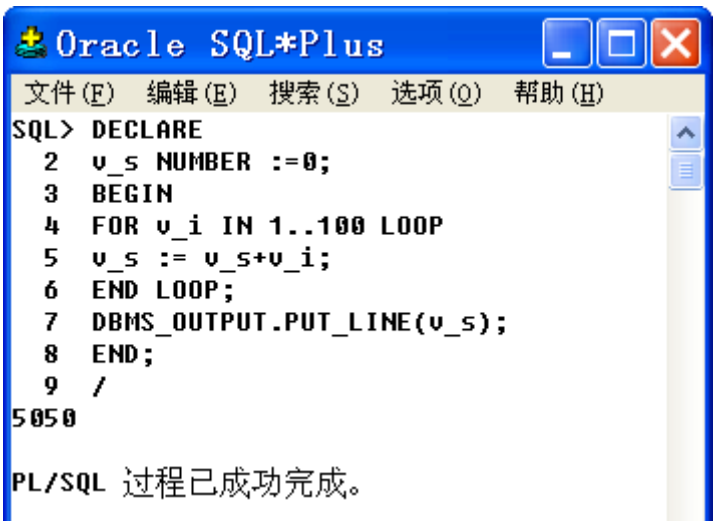
Step3 开始执行第一次循环条件测试，如果循环变量的值小于等于（或大于等于）终值表达式（初值表达式）时，执行循环。

Step4 循环变量的值自动加（减）1，进入下次循环。

PL/SQL 支持隐含声明循环变量，即，用户不需要声明一个循环变量。隐含声明的循环变量为整数，所以此时初值表达式和终值表达式的值必须为整数。隐含声明的循环变量，只在循环体内有效。不能对这个隐含声明的循环变量进行赋值，循环开始后，由初值/终值表达式确定范围，范围在循环体内不能改变。

例：

```
SQL> DECLARE
  2    v_s NUMBER :=0;
  3  BEGIN
  4    FOR v_i IN 1..100 LOOP
  5        v_s := v_s+v_i;
  6    END LOOP;
  7    DBMS_OUTPUT.PUT_LINE(v_s);
  8  END;
  9  /
5050

PL/SQL 过程已成功完成。
```



4)、循环的嵌套

循环的嵌套指一个循环体内可以包含另一个完成的循环体。例如：

```
WHILE  条件 1    LOOP
    语句段 11

    WHILE  条件 2    LOOP
        语句段 2
    END LOOP;

    语句段  12
END LOOP;
```

```
LOOP
    语句段 11

    WHILE  条件 1    LOOP
        语句段  2
    END LOOP;

    语句段 12
    EXIT WHEN  条件 2
END LOOP;
```

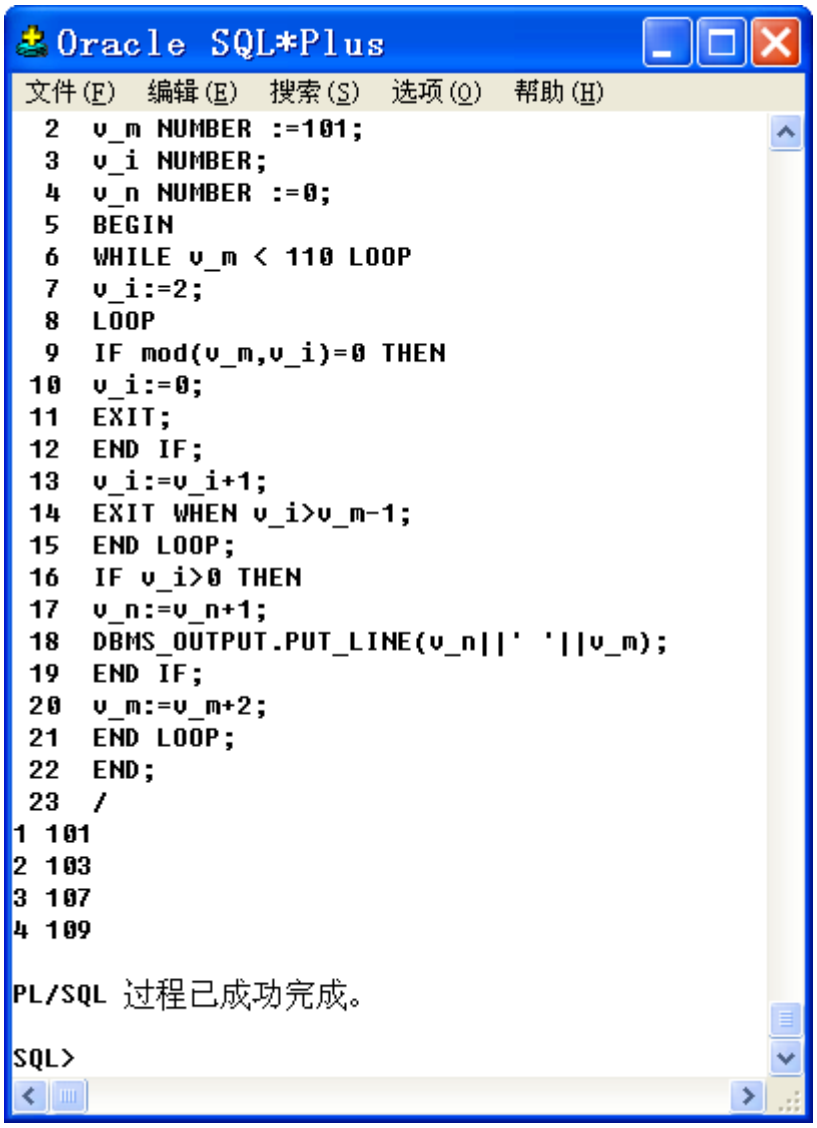
```
FOR  计数器  IN  上届…下届
LOOP
    语句段 11

        LOOP
            语句段 2
            EXIT WHEN  条件 1
        END LOOP;

    语句段 12
END LOOP;
```

例：计算并显示 100～110 之间的素数。其算法是：对于一个整数 m 而言，把 m 分别用 2 到（m－1）的整数除，如果能被其中任何一个数整除，则表示 m 不是素数。显然素数肯定是奇数。

```
SQL> DECLARE
  2     v_m NUMBER :=101;
  3     v_i NUMBER;
  4     v_n NUMBER :=0;
  5  BEGIN
  6     WHILE v_m < 110 LOOP
  7        v_i:=2;
  8        LOOP
  9           IF mod(v_m,v_i)=0 THEN
 10              v_i:=0;
 11              EXIT;
 12           END IF;
 13           v_i:=v_i+1;
 14           EXIT WHEN v_i>v_m-1;
 15        END LOOP;
 16        IF v_i>0 THEN
 17           v_n:=v_n+1;
 18           DBMS_OUTPUT.PUT_LINE(v_n||' '||v_m);
 19        END IF;
 20        v_m:=v_m+2;
 21     END LOOP;
 22  END;
 23  /
1 101
2 103
3 107
```

4 109

PL/SQL 过程已成功完成。



5)、顺序控制

1)、GOTO 语句

2)、NULL 语句：过程中可以使用 NULL 语句执行一个空语句。

7．3、异常处理

一个 PL/SQL 程序的错误可以分为两种：编译时刻错误和运行时刻错误。编译时刻错误，可以查询数据字典 USER_ERRORS，ALL_ERRORS，DBA_ERRORS 来读取错误信息。如：SQL> select * from user_errors;

在 PL/SQL 块中捕获和处理异常的语法如下：

```
EXCEPTION
    WHEN 异常错误名称 1 [OR 异常错误名称 2......] THEN
        语句段 1
    WHEN 异常错误名称 3 [OR 异常错误名称 4......] THEN
        语句段 2
        ......
    WHEN OTHERS THEN
        语句段 3
END;
```

如上所示，可以用 WHEN 子句按异常错误名称捕获各种异常错误，如果还有其他没有预计到的异常错误，则可以使用 WHEN OTHERS 子句。当异常错误发生在声明部分或异常处理部分时，当前块内的异常处理部分无法捕获它们。

PL/SQL 在程序包 STANDARD 中包含了一些预定义异常。

异常错误名称	错误代码	描述
ACCESS_INTO_NULL	-6530	当开发对象类型应用时，在引用对象属性之前，必须首先初始化对象。如果试图给一个没有初始化的对象属性赋值，就会引发该异常错误。
CASE_NOT_FOUND	-6592	在 CASE 语句中没有包含必需的 WHERE 子句，并且没有包含 ELSE 子句。
CURSOR_ALREAD_OPEN	-6511	试图打开一个已经打开的游标。一个游标在被重新打开之前必须被关闭。一个游标 FOR 循环会自动打开所涉及的游标，所以在游标循环中不能打开游标

DUP_VAL_ON_INDEX	-1	试图在一个具有惟一约束的数据库列中存储重复的值。
INVALID_CURSOR	-1001	试图执行一个无效的游标操作。
INVALID_NUMBER	-1722	将字符串转换为数字时失败，在过程性语句中，将引发 VALUE_ERROR 错误
LOGIN_DENIED	-1017	用无效的用户名或口令登陆 Oracle
NO_DATA_FOUND	+100	SELECT INTO 语句没有返回任何数据行，或者程序引用一个嵌套表中已经被删除的元素，或索引表中一个没有被初始化的元素
NOT_LOGGED_ON	-01012	在没有登陆 Oracle 数据库的情况下，访问数据库
PROGRAM_ERROR	-6501	Oracle 内在错误，通常由 PL/SQL 本身造成
STORAGE_ERROR	-6500	PL/SQL 程序在运行时内存不够或者内存由问题
TIMEOUT_ON_RESOURCE	-51	Oracle 在等待资源时发生超时现象
TOO_MANY_ROWS	-1422	SELECT INTO 语句返回了多行数据
VALUE_ERROR	-6502	发生了一个算法、转换、截断或者大小约束错误。如果在一个 SQL 语句中发生这些错误，则会引发 INVALID_ERROR 错误
ZERO_DIVIDE	-1476	发生被 0 除的错误
	－2291	违反完整性约束条件
	-79	无效变量

非预定义异常错误。：

步骤 1、在声明部分，用 EXCEPTION 类型定义异常错误的名称。

注意：异常和变量的声明相似，但它不是变量，不能出现在赋值语句中。异常和变量的作用于和规则时相同的。

步骤 2、在声明部分，用 EXCEPTION_INIT 编译指令建立该异常错误名称与某个 Oracle 错误之间的联系。因此当该 Oracle 错误发生的时候，就能够自动引发该

异常错误了，而不需要 RAISE 语句来引发。

步骤 3、在异常处理部分，处理异常错误。

EXCEPTION_INIT 的语法格式是：

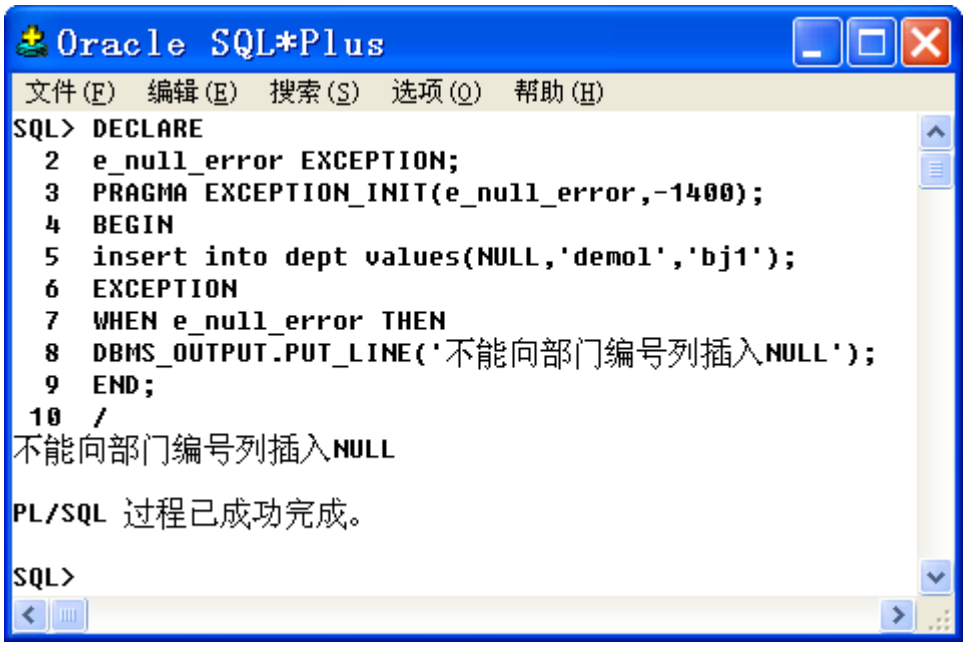
PRAGMA EXCEPTION_INIT(exception_name，oracle_error_code)

其中，exception_name 是定义的异常错误的名称，oracle_error_code 是 Oracle 的错误代码。

例：为错误代码为－1400 的 Oracle 错误进行异常处理。

```
SQL> DECLARE
  2    e_null_error EXCEPTION;
  3    PRAGMA EXCEPTION_INIT(e_null_error,-1400);
  4  BEGIN
  5    insert into dept values(NULL,'demol','bj1');
  6  EXCEPTION
  7    WHEN e_null_error THEN
  8        DBMS_OUTPUT.PUT_LINE('不能向部门编号列插入 NULL');
  9  END;
 10  /
```

不能向部门编号列插入 NULL



另一种处理非预定义异常错误的方法是，在异常处理部分最后的 WHEN OTHERS 子句中使用内置异常错误函数 SQLCODE 和 SQLERRM。

SQLCODE 函数没有参数，它返回 Oracle 错误代码。

有几种特殊情况：用户自定义异常返回+1，没有异常引发返回 0，未找到数据返回+100。SQLERRM（oracle_error_code）函数按照输入的 Oracle 错误代码 oracle_error_code,返回其对应的 Oracle 错误文本。当省略 oracle_error_code 时，SQLERRM 函数返回 SQLCODE 当前值对应的错误消息文本。

有几种特殊情况：当 oracle_error_code 为+1 时返回“User-Defined Exception”，当 oracle_error_code 为 0 时返回“ORA-0000:normal,successful completion”，当 oracle_error_code 为+100 时返回“ORA-01403:no data found”，当 oracle_error_code 为正数时返回“non-Oracle exception”。

注意：SQLCODE 和 SQLERRM 函数均不能直接用在 SQL 语句中，如果需要在 SQL 语句中使用，需要先将其值赋予变量，然后再引用。

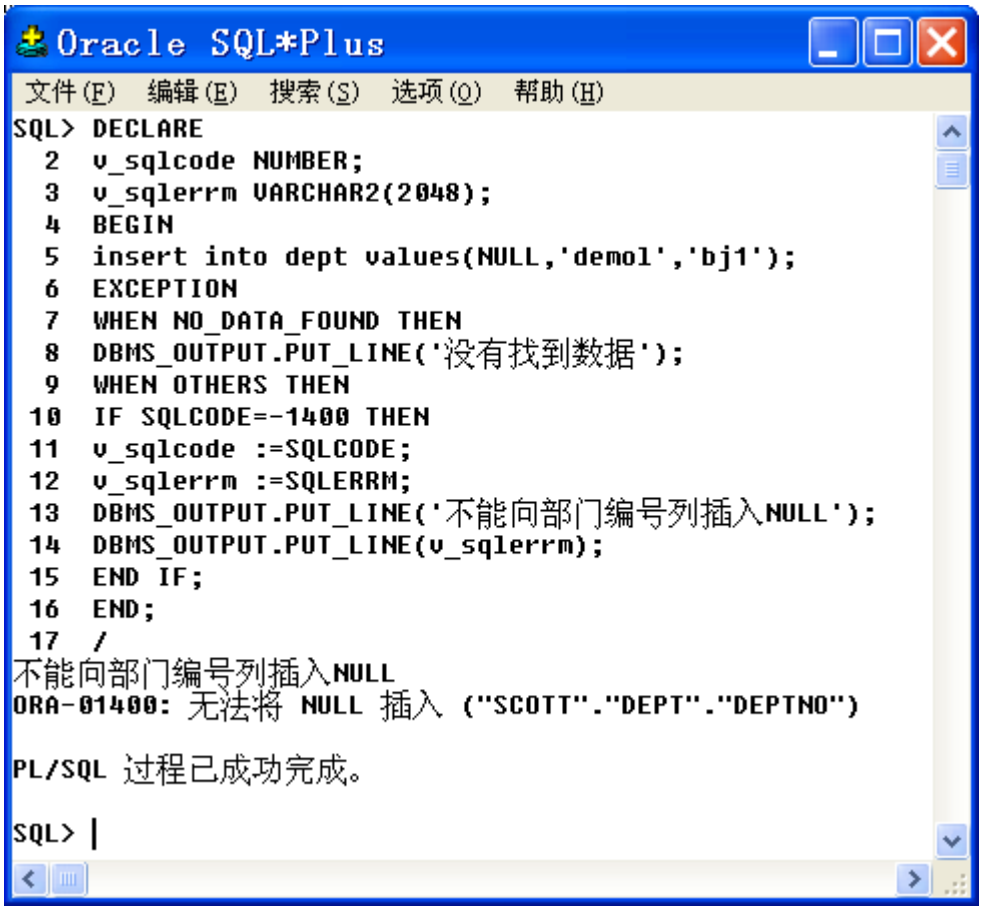
例：

```
SQL> DECLARE
  2    v_sqlcode NUMBER;
  3    v_sqlerrm VARCHAR2(2048);
  4  BEGIN
  5    insert into dept values(NULL,'demol','bj1');
  6  EXCEPTION
  7    WHEN NO_DATA_FOUND THEN
  8        DBMS_OUTPUT.PUT_LINE('没有找到数据');
  9  WHEN OTHERS THEN
 10        IF SQLCODE=-1400 THEN
 11            v_sqlcode :=SQLCODE;
 12            v_sqlerrm :=SQLERRM;
 13            DBMS_OUTPUT.PUT_LINE('不能向部门编号列插入 NULL');
 14            DBMS_OUTPUT.PUT_LINE(v_sqlerrm);
 15  END IF;
```

```
 16  END;
 17  /
```

不能向部门编号列插入 NULL

ORA-01400: 无法将 NULL 插入 ("SCOTT"."DEPT"."DEPTNO")

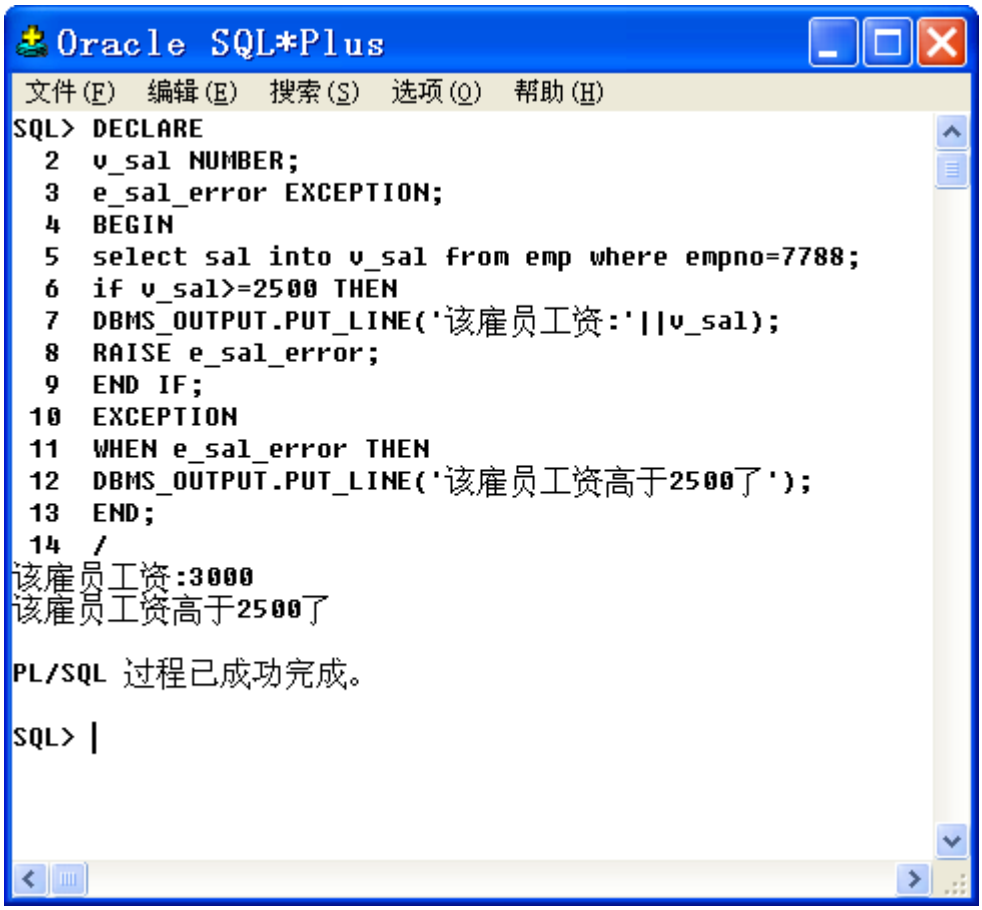


自定义异常：它并不一定是个错误，而是程序员根据编程和调试的需要而为特殊情况所定义的异常。自定义异常必须要声明，并且必须要用 RAISE 语句显示引发。

例：

```
SQL> DECLARE
  2    v_sal NUMBER;
  3    e_sal_error EXCEPTION;
  4  BEGIN
  5    select sal into v_sal from emp where empno=7788;
  6    IF v_sal>=2500 THEN
```

```
  7         DBMS_OUTPUT.PUT_LINE('该雇员工资:'||v_sal);
  8         RAISE e_sal_error;
  9     END IF;
 10  EXCEPTION
 11     WHEN e_sal_error THEN
 12         DBMS_OUTPUT.PUT_LINE('该雇员工资高于 2500 了');
 13  END;
 14  /
该雇员工资:3000
该雇员工资高于 2500 了

PL/SQL 过程已成功完成。
```



创建过程之后，Oracle 会将过程及其执行代码放到数据字典中。通过查询 USER_SOURCE，可以显示当前用户的所有过程及其源代码：

```
SQL> SELECT text FROM user_source WHERE name='QUERY_EMP';
```

函数：用于计算和返回特定的数据，语法如下：

```
CREATE [OR REPLACE] FUNCTION function_name
    ([arg1 [IN | OUT | IN OUT]] arg_type1,
    ([arg2 [IN | OUT | IN OUT]] arg_type2,
    ......
    ([argn [IN | OUT | IN OUT]] arg_typen)
RETURN ruturn_type
IS|AS
    声明部分
BEGIN
    执行部分
EXCEPTION
    异常处理部分
END function_name
```

必须有一个 RETURN 语句，原则上函数只能返回一个值，可以可以通过使用 OUT 形参来返回多值。

例：按照输入的部门编号查询指定部门的工资总和，如果没有该部门，则进行异常处理。同时还可以用 OUT 形参获得该部门的人数。

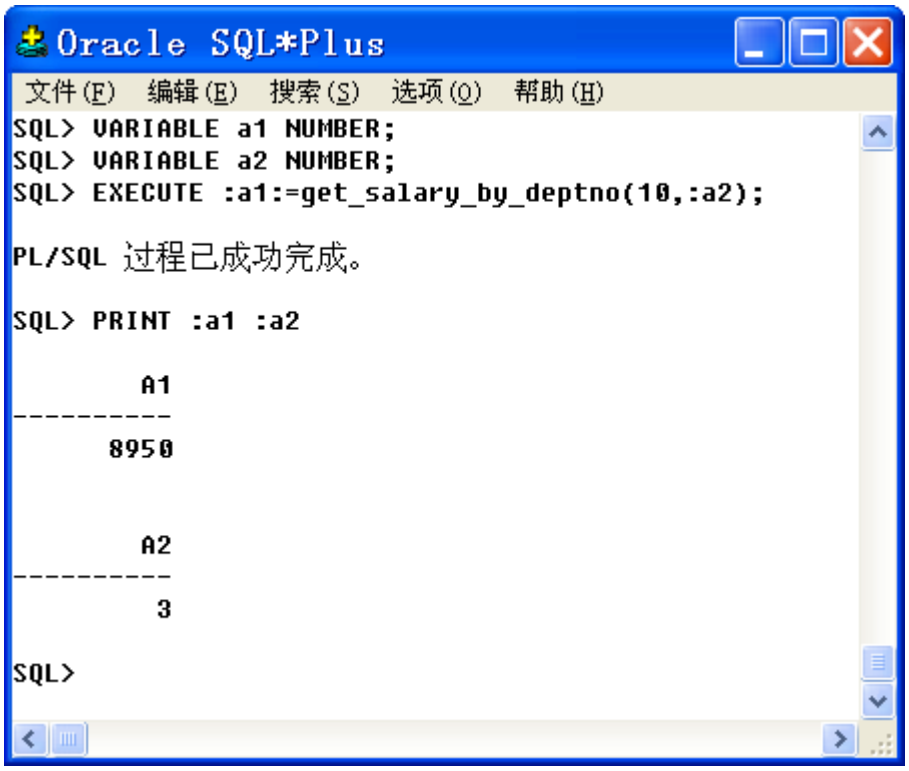
```
CREATE OR REPLACE FUNCTION get_salary_by_deptno(v_dept_no IN
emp.deptno%TYPE, --输入部门号
                                                v_emp_cnt OUT NUMBER
--输出部门人数
                                                ) RETURN NUMBER IS
  v_sum NUMBER(10, 2); --指定返回指定部门的工资综合
BEGIN
  SELECT sum(sal), count(*)
    INTO v_sum, v_emp_cnt
    FROM emp
   WHERE deptno = v_dept_no;
  RETURN v_sum;
```

```
END get_salary_by_deptno;
```

在不同的环境中对函数的调用方式时不同的。形参和实参的传递关系可以采用按位置传递，按名称传递和组合传递。

下面在 SQL*Plus 中，按位置传递调用上面的函数，按位置传递需要严格按照形参的顺序一次安排实参。

```
SQL> VARIABLE a1 NUMBER;
SQL> VARIABLE a2 NUMBER;
SQL> EXECUTE :a1:=get_salary_by_deptno(10,:a2);

PL/SQL 过程已成功完成。

SQL> PRINT :a1 :a2

        A1
----------
      8950


        A2
----------
         3
```



创建函数之后，Oracle 会将函数及其执行代码放到数据字典中。通过查询 USER_SOURCE，可以显示当前用户的所有函数及其源代码：

SQL> SELECT text FROM user_source WHERE name=’ GET_SALARY_BY_DEPTNO ’;在条件子句中函数名和过程名一定要大写。

自定义错误代码及其消息文本：调用 Oracle 提供的 DBMS_STANDARD 程序包中的 RAISE_APPLICATION_ERROR 过程，可以自定义错误代码及其消息文本。语法如下：

RAISE_APPLICATION_ERROR(error_code,message[,TRUE|FALSE])

其中：

error_code 是自己安排的错误代码（在-20000～-20999 之间）。

Message 为自己安排的对应的消息文本（最长为 2048 个字节）。第三个参数是可选的，如果为 TRUE，则将当前 message 添加到错误消息栈中；如果为 FALSE（默认值），则用当前的 message 代替原来的错误消息栈中的消息。在调用程序中，可以像对待非预定义异常错误那样来捕获和处理 RAISE_APPLICATION_ERROR 返回的错误代码及其消息文本。

例：

在雇员没有补助的情况下，提供一个自定义错误代码-20001 及其消息文本。

```
CREATE OR REPLACE PROCEDURE query_comm_if_null(v_no IN emp.empno%TYPE)
IS
  v_comm emp.comm%TYPE;
BEGIN
  SELECT comm INTO v_comm FROM emp WHERE empno = v_no;
  IF v_comm IS NULL OR v_comm = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, '该雇员无补助');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('没有该雇员' || v_no);
END query_comm_if_null;
```

下面（利用上面的例子）是当雇员没有补助时，按非预定义异常错误进行处理（自定义的错误代码是-20001，与上例对应）。这个例子也是关于异常的传播问题的例子。

```
SQL> DECLARE
  2     e_comm_null EXCEPTION;
  3     PRAGMA EXCEPTION_INIT(e_comm_null,-20001);
  4  BEGIN
  5     DBMS_OUTPUT.PUT_LINE('雇员编号:5678');
  6     <<block_1>>
  7     BEGIN
  8        query_comm_if_null(5678);
  9     END block_1;
 10     DBMS_OUTPUT.PUT_LINE('雇员编号:7788');
 11     <<block_2>>
 12     BEGIN
 13        query_comm_if_null(7788);
 14     EXCEPTION
 15        WHEN e_comm_null THEN
```

```
 16              DBMS_OUTPUT.PUT_LINE('提醒财务部门给该雇员加补助');
 17      END block_2;
 18  END;
 19  /
```

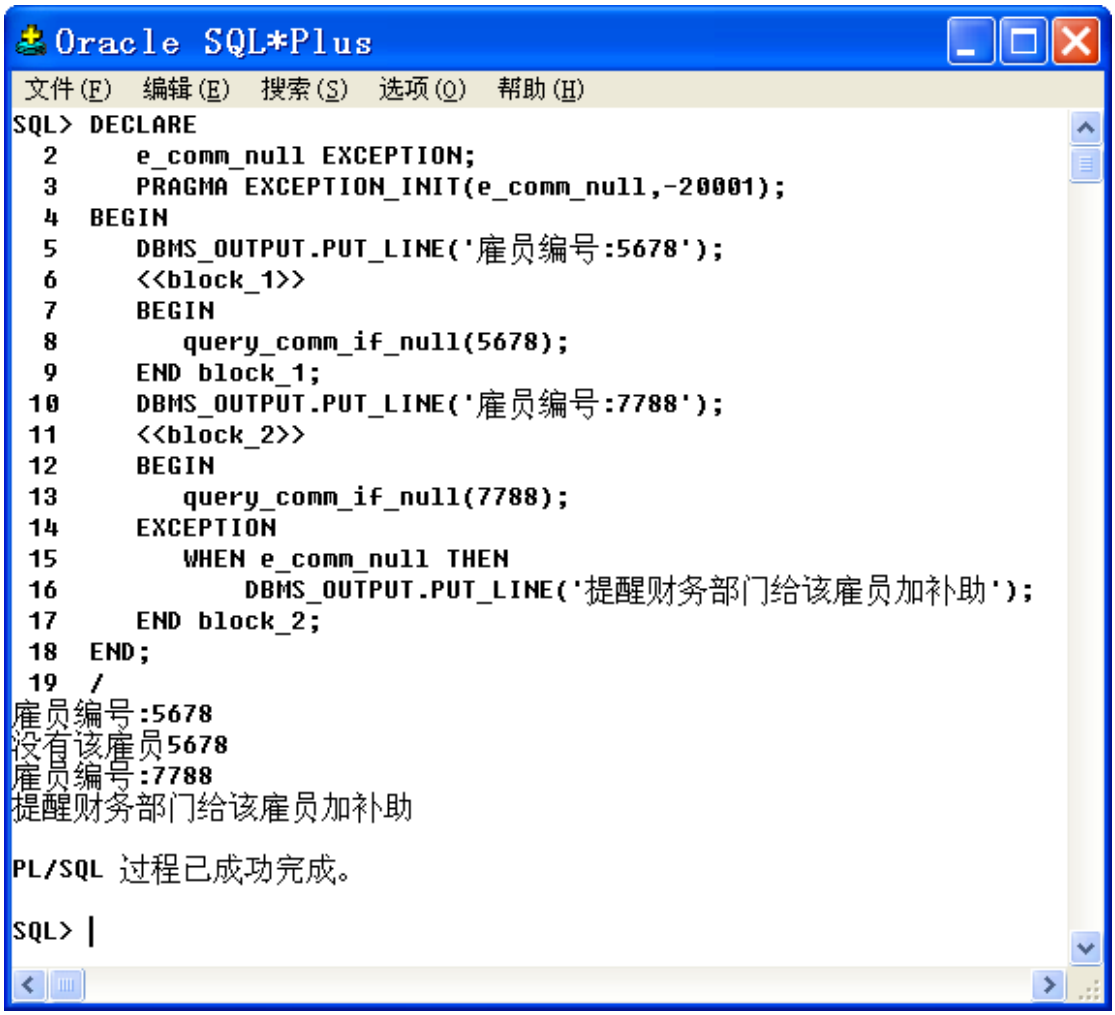
雇员编号:5678

没有该雇员 5678

雇员编号:7788

提醒财务部门给该雇员加补助

PL/SQL 过程已成功完成。



7．4、游标

在每个用户会话中，可以同时打开多个游标，其数量由数据库初始化参数文件中的 OPEN_CURSORS 参数定义。

一、声明游标：

即声明该游标所对应的 select 语句，可以带参数，也可以由返回值。语法格式如下：

```
CURSOR cursor_name[(arg1 arg1_datatype [,arg2 arg2_datatype]
.....)]
    [RETURN return_datatype]
IS
    Select_statement;
```

其中：

* cursor_name 为声明的游标。

* arg1，arg2，...是可选的，表示游标的输入参数（只能是输入参数）。

* arg1_datatype，arg2_datatype，...表示为对应的数据类型。

注意：此处的数据类型后面不能带参数，即精度、范围等。如 NUMBER(12,2)只能写成 NUMBER。

* RETURN return_datatype 是可选的，为游标的返回数据类型。如果选择，则应该严格与 Select_statement 中的选择列表在次序和数据类型上保持匹配。一般是记录数据类型或带%ROWTYPE 的数据类型。

* Select_statement 为游标的查询语句，它决定游标中的数据结构和行数。

注意：声明游标之后，还可以基于游标用%ROWTYPE 定义记录变量，然后该记录变量便具有了 Select_statement 中选择列表的列名或别名（如果有表达式列，一定要取别名）。这种方法可以简化游标的数据处理，语法如下：

```
record_variable cursor_name%ROWTYPE
```

二、打开游标：

为了执行游标中的 Select_statement 语句，查询并得到一个结果集，需要打开游标。打开游标后得到的结果集是静态的，所有对数据操作的 SQL 语

句对该结果集都没有影响，直到该游标关闭后再打开，这些影响才会在结果集中反映出来，语法格式如下：

OPEN cursor_name[(arg1 arg1_datatype [,arg2 arg2_datatype]......)];

三、提取游标：

打开游标后，游标的指针指向结果集的第一行。如果要提取结果集中的数据，就需要提取游标。语法格式如下：

FETCH cursor_name INTO {variable_list | record_ variable};

其中，variable_list 和 record_ variable 分别为已经声明的变量列表和记录变量。

使用 variable_list 时，FETCH 语句将结果集中的一个数据行的各列值赋予不同的变量。使用 record_ variable 时，FETCH 语句将结果集中的一个数据行的各列值赋予记录变量中的不同元素。

FETCH 语句执行时，每次返回一个数据行，然后将游标指针移动到下一个数据行。当检索到最后一行数据时，如果再次执行 FETCH 语句，操作将失败，并时游标属性%NOTFOUND 置为 TRUE。所以每次执行完 FETCH 语句后，检查游标属性%NOTFOUND 就可以判断 FETCH 语句是否执行成功，以便确定是否给变量赋了值。

四、关闭游标：

当提取和处理完游标后，应该及时关闭游标，以释放它所占用的系统资源。其语法格式为：

CLOSE cursor_name

游标关闭后，可以再次打开。如果游标关闭后对它执行打开以外的操作，将产生 INVALID_CURSOR 异常错误。

例：先声明 4 各游标：c1 不带参数，c2 带参数，c3 带参数并带返回值类型，c4 带参数；然后打开游标；用 LOOP 循环提取游标，并处理结果集；关闭游标，并显示游标参数的传递方法和基于游标定义记录变量的方法。

```
DECLARE
  TYPE DeptRecord IS RECORD
       (deptno  dept.deptno%TYPE,
        dname   dept.dname%TYPE,
        loc     dept.loc%TYPE);
```

```
  v_deptrecord DeptRecord;
  v_dept_name  dept.dname%TYPE;
  v_dept_loc   dept.loc%TYPE;
  CURSOR c1                        --声明游标
  IS
         Select dname,loc from dept where deptno<=10;
  CURSOR c2(v_dept_no NUMBER)   --有参数游标
  IS
         Select dname,loc from dept where deptno<=v_dept_no;
  CURSOR c3(v_dept_no NUMBER)   --有返回数据类型游标
         RETURN DeptRecord
  IS
         Select deptno,dname,loc from dept where deptno<=v_dept_no;
  CURSOR c4(v_dept_no NUMBER)
  IS
         Select deptno,dname,loc from dept where deptno<=v_dept_no;
  v_dept_rec c4%ROWTYPE;         --基于游标定义记录变量，比DeptRecord方便
BEGIN
  OPEN c1;                         --打开游标
  LOOP
         FETCH c1 INTO v_dept_name,v_dept_loc;     --提取游标
         IF c1%FOUND THEN
            DBMS_OUTPUT.PUT_LINE('c1:'||v_dept_name||' '||v_dept_loc);
         ELSE
            DBMS_OUTPUT.PUT_LINE('c1:'||'已经处理完结果集了');
            EXIT;
         END IF;
  END LOOP;
  OPEN c2(20);
  LOOP
         FETCH c2 INTO v_dept_name,v_dept_loc;
         EXIT WHEN c2%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('c2:'||v_dept_name||' '||v_dept_loc);
  END LOOP;
  OPEN c3(v_dept_no=>30);
  LOOP
         FETCH c3 INTO v_deptrecord;
         EXIT WHEN c3%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('c3:'||v_deptrecord.deptno||' '||v_deptrecord.dname);
  END LOOP;
  OPEN c4(40);
  LOOP
         FETCH c4 INTO v_dept_rec;
         EXIT WHEN c4%NOTFOUND;
```

```
        DBMS_OUTPUT.PUT_LINE('c4:'||v_dept_rec.deptno||' '||v_dept_rec.dname);
  END LOOP;
  CLOSE c1;       --关闭游标
  CLOSE c2;
  CLOSE c3;
  CLOSE c4;
END;/
```

```
c1:ACCOUNTING NEW YORK
c1:已经处理完结果集了
c2:ACCOUNTING NEW YORK
c2:RESEARCH DALLAS
c3:10 ACCOUNTING
c3:20 RESEARCH
c3:30 SALES
c4:10 ACCOUNTING
c4:20 RESEARCH
c4:30 SALES
c4:40 OPERATIONS

PL/SQL 过程已成功完成。
```

在该例子中使用的 dept 表的数据如下：

```
SQL> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 testing        Dubai
```

SQL>

游标属性：

无论时显式游标还是隐式游标，均有%ISOPEN，%FOUND，%NOTFOUND，%ROWCOUNT 四种属性。它们描述与游标操作相关的 DML 语句的执行情况。游标属性只能用在 PL/SQL 过程语句内，而不能用在 SQL 语句内。

显式游标的属性如下表：

属性	描述
cursor_name%ISOPEN	游标打开为 TRUE，否则为 FALSE
cursor_name%FOUND	如果最近一次提取游标操作 FETCH 成功为 TRUE，否则为 FALSE。当在游标打开之后、提取之前访问%FOUND 时，返回 NULL;当在游标打开之前访问%FOUND 时，将引起 INVALID_CURSOR 异常错误。
cursor_name%NOTFOUND	如果最近一次提取游标操作 FETCH 失败为 TRUE，否则为 FALSE。当在游标打开之后、提取之前访问%NOTFOUND 时，返回 NULL;当在游标打开之前访问%FOUND 时，将引起 INVALID_CURSOR 异常错误。
cursor_name%ROWCOUNT	返回最近一次提取到的数据行的序号。当在游标打开之后、提取之前访问%ROWCOUNT 时，返回 0，当在游标打开之前访问%ROWCOUNT 时，将引起 INVALID_CURSOR 异常错误。

隐式游标：

Oracle 为每一个不属于显示游标的 DML 语句都创建一个隐式游标。隐式游标的名称是 SQL。不能对 SQL 游标显示地执行 OPEN，FETCH，CLOSE 语句。当使用 SELECT 语句时，SQL 游标一次只能返回一行或没有返回行(对应于 NO_DATA_FOUND 异常)。如果返回多行，就会产生 TOO_MANY_ROWS 异常，这时就应该使用显示游标来处理了。当时用 INSERT，UPDATE，DELETE 时，SQL 游标可以处理多行。

隐式游标的属性如下表：

属性	值	select	insert	update	delete
SQL%ISOPEN		FALSE	FALSE	FALSE	FALSE
SQL%FOUND	TRUE	有结果		成功	成功
SQL%FOUND	FALSE	没结果		失败	失败
SQL%NOTFOUND	TRUE	没结果		失败	失败
SQL%NOTFOUND	FALSE	有结果		成功	成功
SQL%ROWCOUNT		返回的行数只能为 1	插入的行数	修改的行数	删除的行数

游标 FOR 循环：可以隐含地实现 OPEN，FETCH，CLOSE 游标以及循环处理结果集的功能。其步骤时：当进入循环时，自动打开一个已经声明的游标，并提取第一行游标数据；当处理完当前所提取的数据而进入下一次循环时，自动提取下一行游标；当提取完结果集中的所有数据行后结束循环，并自动关闭游标。除此之外，当在游标 FOR 循环语句中调用 EXIT 或 GOTO 语句，或者由于发生异常错误等原因而导致跳出循环时 PL/SQL 均能够自动关闭游标。其语法格式如下：

```
FOR index_variable IN cursor_name[(value1 [,value2].....)] LOOP
    语句段
END LOOP;
```

其中：

- cursor_name 为已经声明的游标。
- value1,value2，...是应用程序传递给游标的参数。
- index_variable 是游标 FOR 循环隐含声明的索引变量，该索引变量为记录变量，其结构与游标查询语句返回的结果集的结构相同。在程序中可以通过引用该索引变量中的元素来读取所提取的游标数据。索引变量中的元素名称与游标查询语句选择列表中所指定的列名相同。如果在游标查询语句的选择列中有计算列，则必须为这些计算列指定别名，然后才能通过游标 FOR 循环的索引变量来访问这些列数据。

例：使用游标 FOR 循环来查询显示多行记录数据集，包括了向游标传递参数。

```
DECLARE
  CURSOR c1(v_dept_no NUMBER DEFAULT 10)              --有参数游标，默认为10
  IS
        select deptno,dname from dept where deptno<=v_dept_no;
  BEGIN
        DBMS_OUTPUT.PUT_LINE('给v_dept_no传递参数30');
        FOR c1_rec IN c1(30) LOOP
            DBMS_OUTPUT.PUT_LINE('c1:'||c1_rec.deptno||' '||c1_rec.dname);
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('使用v_dept_no默认参数10');
        FOR c1_rec IN c1 LOOP
            DBMS_OUTPUT.PUT_LINE('c1:'||c1_rec.deptno||' '||c1_rec.dname);
        END LOOP;
  END;
```

给 v_dept_no 传递参数 30

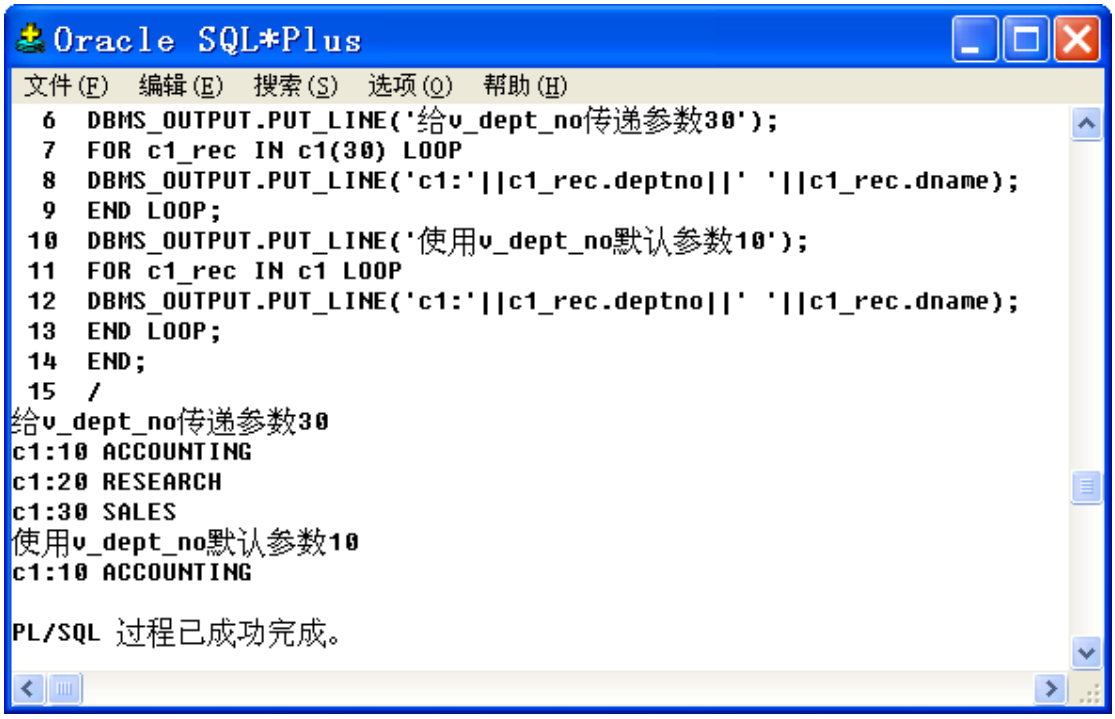
c1:10 ACCOUNTING

c1:20 RESEARCH

c1:30 SALES

使用 v_dept_no 默认参数 10

c1:10 ACCOUNTING



使用游标更新或删除数据：

通过使用显示游标，不仅可以一行一行地处理 SELECT 语句的结果集，而且还可以更新或删除当前游标行的数据。这时，要求游标查询语句中必须使用 FOR UPDATE 选项，以便在打开游标时锁定游标结果集在数据库表中对应的数据行。其语法格式如下：

SELECT column_list FROM table_list FOR UPDATE

使用 FOR UPDAE 打开游标之后，就可以在 UPDATE 和 DELETE 语句中使用 WHERE CURRENT OF 子句，修改或删除游标结果集中当前行所对应的数据库表中的数据行。其语法格式如下：

WHERE CURRENT OF cursor_name

例：将 emp 表中部门编号大于等于 30 的雇员的补助调整到 800 元。

```
DECLARE
  v_emp_rec emp%ROWTYPE;
  CURSOR c1
  IS
         select * from emp FOR UPDATE;
  BEGIN
         OPEN c1;
         LOOP
              FETCH c1 INTO v_emp_rec;
              EXIT WHEN c1%NOTFOUND;
              IF v_emp_rec.deptno>=30 THEN
                 update emp set comm = 800      --修改数据
                 where CURRENT OF c1;
              END IF;
          END LOOP;
          COMMIT;                    --提交已经修改的数据
          CLOSE c1;
  END;
```

如果将其中的 UPDATE 语句更改为：

DELETE FORM EMP WHERE CURRENT OF c1;就会将 emp 表中部门编号大于 30 的记录删除。

7．5、程序包

7.5.1、程序包说明

包说明是包与应用程序之间的接口。它用于定义包的共有组件，如变量、常量、过程、函数、游标等。在包说明中所定义的公有组件不仅可以在包内使用，还可以由其他过程和函数使用。需要注意，建立包说明时，为了实现信息隐藏，不应该将所有组建都放在包说明处定义，而只应该定义共有组件。

创建包的语法格式为：

```
CREATE [OR REPLACE] PACKAGE package_name
IS|AS
[PRAGMA SERIALLY_REUSABLE;]
    公有数据类型定义
    公有变量声明
    公有常量声明
    公有异常错误声明
    公有游标声明
    公有函数声明
    公有过程声明
END package_name;
```

其中，PRAGMA SERIALLY_REUSABLE 是可选的，决定所创建的包是否可以被连续调用。如果省略，则包的运行状态被放在用户全局区；如果使用，则包的运行状态被放在系统全局区。这样每次调用包之后，该包的运行状态就会被释放。可以被连续调用，而不会受以前的运行状态的影响。建议添加此选项。

注意：共有过程、函数的声明要放在其他声明之后。

例：创建一个包说明 my_pkg，其中包含一个记录标量 v_deptrec。两个用于保存内置异常错误函数 SQLCODE 和 SQLERRM 返回值的变量、两个函数、两个过程。记录变量 v_deptrec 的结构与 dept 表的结构相同：

```
CREATE OR REPLACE PACKAGE my_pkg
IS
PRAGMA SERIALLY_REUSABLE;
        v_deptrec dept%ROWTYPE;
        v_sqlcode NUMBER;
```

```
        v_sqlerrm VARCHAR2(2048);
        FUNCTION add_dept(v_deptno   NUMBER,
                          v_deptname VARCHAR2,
                          v_deptloc  VARCHAR2)
           RETURN NUMBER;
        FUNCTION remove_dept(v_deptno NUMBER)
           RETURN NUMBER;
        PROCEDURE query_dept(v_deptno NUMBER);
        PROCEDURE read_dept;
END my_pkg;
```

7.5.2、程序包体

包体是包的具体实现。在其中可以定义私有组件，可以定义公有游标，实现在包说明中说明的过程、函数。建立包体时，需要注意：

- 包体只能在包说明被编译后才进行编译。
- 在包说明中的过程、函数名称必须严格地与包体中实现部分的过程、函数名称相匹配。
- 在包体中声明的数据类型、变量、常量都是私有的，只能在包体中使用。但在包体中可以使用在包说明中声明的数据类型、常量、变量。
- 在保体的执行部分可以对包说明中声明的变量进行初始化。因为包不能传递参数，所以只能通过这种方法进行初始化。包的初始化只在第一此调用包的时候运行一次。

创建包体的语法格式为：

```
CREATE [OR REPLACE] PACKAGE BODY package_name
IS|AS
[PRAGMA SERIALLY_REUSABLE;]
    私有数据类型定义
    私有变量声明
    私有常量声明
    私有异常错误声明
    私有函数声明和定义
    私有过程声明和定义
    公有游标定义
```

公有函数定义

公有过程定义

BEGIN

执行部分（初始化部分）

END package_name;

其中：

package_name 是要创建的包体的名称。在数据库中的一个用户创建的包体名称都是唯一的。

PRAGMA SERIALLY_REUSABLE 是可选的，当在包的说明中包含 PRAGMA SERIALLY_REUSABLE 时，也需要在包体中包含 PRAGMA SERIALLY_REUSABLE。

在创建包说明和包体的时候，容易出现错误。如果要查看编译错误，可以使用如下命令：

SQL> SHOW errors;

例：创建一个包体，实现上面例子中所创建的包说明，并在包体中声明私有变量、私有游标和私有函数。

```
CREATE OR REPLACE PACKAGE BODY my_pkg
IS
PRAGMA SERIALLY_REUSABLE;
        --私有变量声明
        v_flag NUMBER;
        --私有游标声明
        CURSOR mycursor IS
             select deptno,dname from dept;
        --私有函数声明和定义
        FUNCTION check_dept(v_deptno NUMBER)
             RETURN NUMBER
        IS
        BEGIN
             select count(*) INTO v_flag from dept
             where deptno = v_deptno;
             IF v_flag > 0 THEN
                v_flag := 1;
             END IF;
             RETURN v_flag;
        END check_dept;
        --公有函数定义add_dept
        FUNCTION add_dept(v_deptno   NUMBER,
```

```
                    v_deptname VARCHAR2,
                    v_deptloc  VARCHAR2)
     RETURN NUMBER
IS
BEGIN
     IF check_dept(v_deptno) = 0 THEN
        insert into dept
        values(v_deptno,v_deptname,v_deptloc);
        RETURN 1;
     ELSE
        RETURN 0;
     END IF;
EXCEPTION
     WHEN OTHERS THEN
          v_sqlcode := SQLCODE; --捕获异常错误消息
          v_sqlerrm := SQLERRM;
          RETURN -1;
END add_dept;
 --公有函数定义remove_dept
FUNCTION remove_dept(v_deptno NUMBER)
     RETURN NUMBER
IS
BEGIN
     IF check_dept(v_deptno) = 1 THEN
        delete from dept where deptno=v_deptno;
        RETURN 1;
     ELSE
        RETURN 0;
     END IF;
EXCEPTION
     WHEN OTHERS THEN
          v_sqlcode := SQLCODE; --捕获异常错误消息
          v_sqlerrm := SQLERRM;
          RETURN -1;
END remove_dept;
--公有过程定义query_dept
PROCEDURE query_dept(v_deptno NUMBER)
IS
BEGIN
     IF check_dept(v_deptno) = 1 THEN
        select * INTO v_deptrec from dept
        where deptno=v_deptno;
     END IF;
EXCEPTION
```

```
            WHEN OTHERS THEN
                 v_sqlcode := SQLCODE; --捕获异常错误消息
                 v_sqlerrm := SQLERRM;
        END query_dept;
        --公有过程定义
        PROCEDURE read_dept
        IS
             v_deptno NUMBER;
             v_dname VARCHAR2(14);
        BEGIN
             --用游标式FOR循环处理游标结果的每一行
             FOR c_mycursor IN mycursor LOOP
                 v_deptno := c_mycursor.deptno;
                 v_dname := c_mycursor.dname;
                 DBMS_OUTPUT.PUT_LINE(v_deptno||' '||v_dname);
             END LOOP;
        END read_dept;
BEGIN --保体初始化部分，对公有变量进行初始化
      v_sqlcode := NULL;
      v_sqlerrm := '初始化消息文本';
END my_pkg;
```

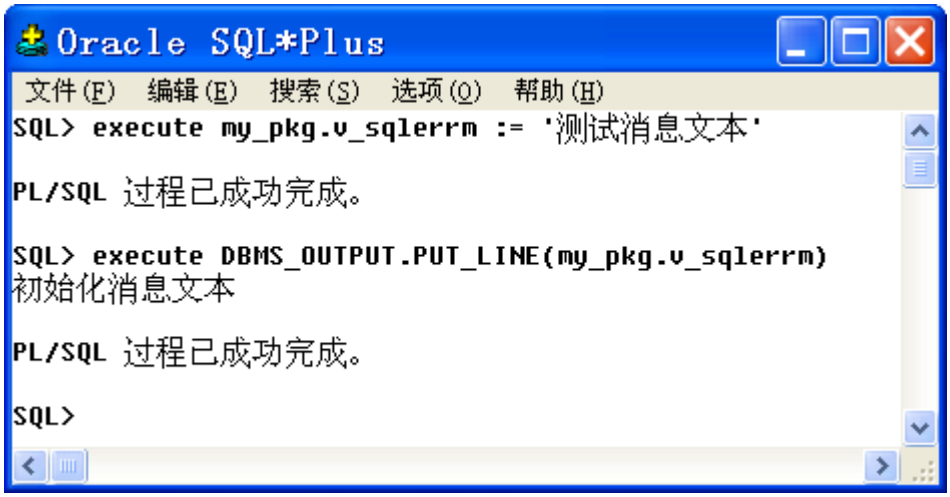
7.5.2、程序包的调用、查看和删除

一、调用包

在创建了包说明和相应的包体之后，就可以调用该包的各个组建了，对包内公有组件的调用格式要用包名并加“.”作为限定词，即包名.组件名称。下面用EXECUTE 命令来调用包中的各个组件。

1)、调用包的公有变量

```
SQL> execute my_pkg.v_sqlerrm := '测试消息文本'
PL/SQL 过程已成功完成。
SQL> execute DBMS_OUTPUT.PUT_LINE(my_pkg.v_sqlerrm)
初始化消息文本
```

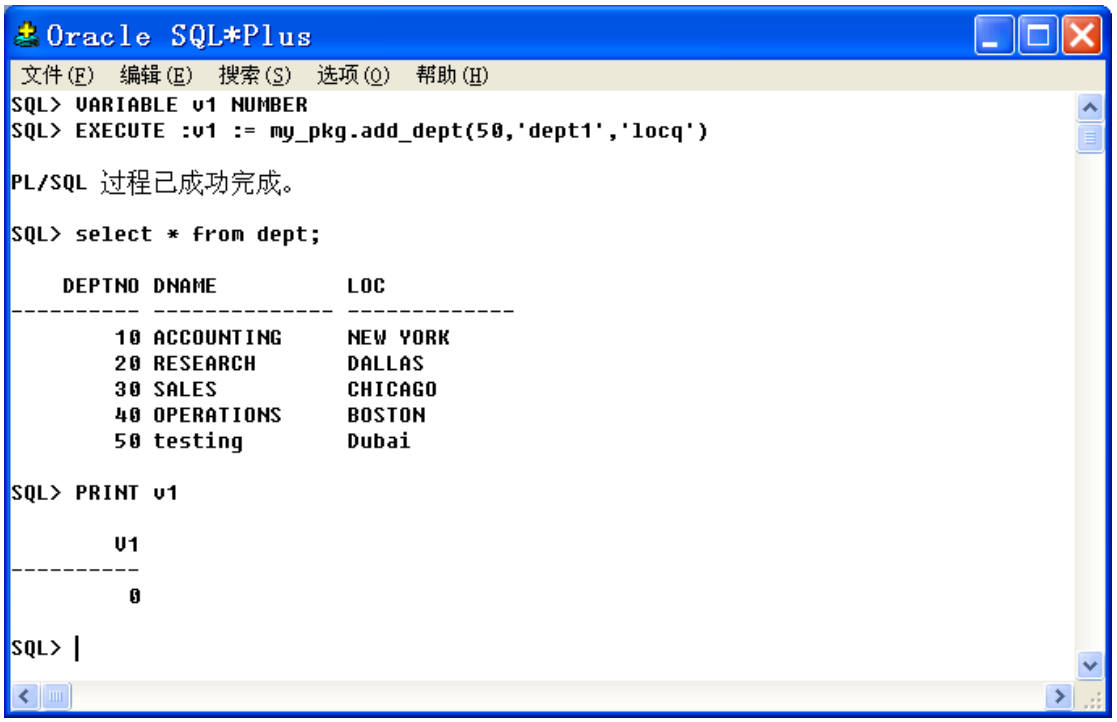


2)、调用包的公有函数

例:

```
SQL> VARIABLE v1 NUMBER
SQL> EXECUTE :v1 := my_pkg.add_dept(50,'dept1','locq')

PL/SQL 过程已成功完成。

SQL> select * from dept;

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

SQL> PRINT v1

        V1
```

0



3)、调用包的公有过程

例：

SQL> EXECUTE my_pkg.read_dept

10 ACCOUNTING

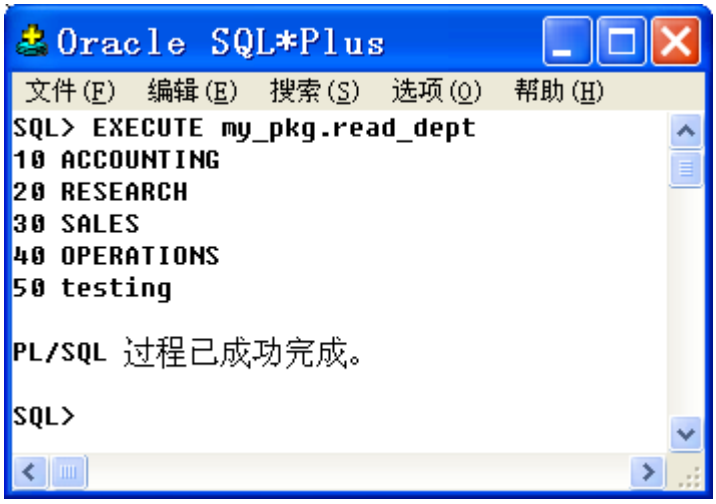
20 RESEARCH

30 SALES

40 OPERATIONS

50 testing

PL/SQL 过程已成功完成。



三、 查看包源代码

创建程序包之后，Oracle 会将程序包及其执行代码放到数据字典中。通过查询 USER_SOURCE，可以显示当前用户的所有程序包及其源代码。如下例（注意：函数名要大写）

SQL> select text from user_source where name ='MY_PKG';

四、 删除包

当不再需要某个程序包时，可以将其删除。如果只删除包体，则可以使用命令：

DROP PACKAGE BODY package_name;

如果要同时删除包说明，可以使用命令：

DROP PACKAGE package_name;

7．6、触发器

触发器主要由以下几个部分组成：

*、触发事件：引起触发器被触发的事件。如 DML 语句（如 INSERT，UPDATE，DELETE 语句对表或视图执行数据处理操作）DDL 语句（如 CREATE，ALTER，DROP 语句在数据库中创建、修改、删除模式对象）。数据库系统事件（如系统启动、退出或异常退出）和用户事件（如登录或退出数据库）。

*、触发条件：由 WHEN 子句指定的一个逻辑表达式。只有当该表达式的值为

TRUE 时，遇到触发事件才会自动执行触发器，否则即便遇到触发事件也不会执行触发器。

*、触发对象：包括表、试图、模式和数据库。

*、触发操作：触发器所要执行的过程。

编写触发器时，要注意以下几点：

1)、触发器不接受参数。

2)、一个表上最多可以由 12 个触发器，但同一事件、同一事件、同一类型的触发器只能由一个。还需要注意，各个触发器之间不能由矛盾。

3)、在一个表上的触发器越多，对在该表上的 DML 操作的性能影响就越大。

4)、触发器最大为 32KB，如果确实需要，可以先建立过程，然后在触发器中用 CALL 语句调用。

5)、在 DML 触发器中只能使用 DML 语句（SELECT，INSERT，UPDATE，DELETE）。

6)、在系统触发器中只能包含 DDL 语句（CREAT，ALTER，DROP）。

7)、触发器中不能包含事务控制语句（COMMIT，ROLLBACK，SAVEPOINT）。因为触发器时触发语句的一部分，触发语句被提交或回退时，触发器也就被提交或回退了。

8)、在触发器主体中调用的任何过程、函数都不能使用事务控制语句。

9)、在触发器主体中不能声明任何 long 和 blob 变量。新值 new、旧值 old 也不能指向表中的任何 long 和 blob 列。

10)、不同类型的触发器（如 DML 触发器、INSTEAD OF 触发器、系统触发器）的语法格式和作用都有较大区别。

7.6.1、DML 触发器

DML 触发器是定义在一个表和简单视图（针对一个表的视图）上的触发器，建立了 DML 触发器之后，如果发生了 DML 操作（INSERT，UPDATE，DELETE），则会自动执行相应的触发器。

DML 触发器的基本要点是：

*、触发时机：指定触发器的触发时间。如果指定为 BEFORE，则表示在执行 DML 操作之前触发：如果指定 AFTER，则表示在执行 DML 操作之后触发。

*、触发事件：引起触发器被触发的事件，即 DML 操作（INSERT，UPDATE，

DELETE)。即可以是单个触发事件，也可以是多个触发事件的组合（只能是 OR 逻辑组合)。

*、条件谓词：当在触发器中包含多个触发事件（INSERT，UPDATE，DELETE）的组合时，为了分别针对不同的事件进行不同的处理，需要使用 Oracle 提供的如下条件谓词：

1、INSERTING：当触发事件是 INSERT 时，取值为 TRUE，否则为 FALSE。

2、UPDATING[(column_x)]：当触发事件时 UPDATE 时，如果修改了 column_x 列，则取值为 TRUE，否则为 FALSE。其中 column_x 是可选的。

UPDATING 与(column_x)之间有空格。

3、DELETING：当触发事件是 DELETE 时，取值为 TRUE，否则为 FALSE。

这三个谓词只能用在触发器的 PL/SQL 块中，而不能用在触发器所调用的子程序中。

*、触发对象：指定触发器是创建在哪个表或视图上。当针对这些表或视图执行 DML 操作时，才触发这些触发器。

*、触发类型：指定触发事件发生后，需要执行几次触发器。即，是语句级触发器还是行级触发器，默认是语句级触发器。语句级触发器将整个 DML 语句操作作为触发事件，当符合触发条件时，只会执行一次触发器；当符合触发条件时，行级触发器对于 DML 语句所影响的每一行都会执行一次触发。

*、触发条件：由 WHEN 子句指定的一个逻辑表达式。只有当该表达式的值为 TRUE 时，遇到触发事件才会自动执行触发器，使其执行触发操作，否则即便遇到触发事件也不会执行触发器。

只允许在行级触发器上指定触发条件。

*、触发顺序：因为在一个表或视图上可以创建多个触发器，在触发时机（BEFORE 和 AFTER）和触发类型（语句级触发器、行级触发器）的组合下，这些触发器有一个执行的顺序，同一个触发器在多次执行时也有一个执行顺序。对于多行数据而言，语句级触发器只被执行一次，而行级触发器在每行上都被执行一次。

*、功能特点：为了确保数据库满足特定的商业规则或企业逻辑，可以使用约束、触发器、子程序（过程和函数)。其中，约束的性能最好，实现最简单，

所以首选约束。对于约束不能实现的规则或逻辑，可以选择触发器。对于触发器也不能实现的规则或逻辑，则应该选择子程序。

DML 触发器可以用于实现数据库操作的安全保护、数据审计、数据完整性、参照完整性和数据复制等功能。

7.6.2、语句级触发器

语句级触发器是指当执行 DML 操作时，以语句为单位执行的触发器。格式为：

CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER}

{INSERT | UPDATE | DELETE}

[OR { INSERT | UPDATE | DELETE }...]

ON table_name

PL/SQL_block | CALL procedure_name;

其中：

*、BEFORE 和 AFTER 指定触发时机。

*、INSERT，UPDATE，DELETE 指定触发事件。

*、ON table_name 指定触发对象，即该触发器被创建在哪个表或视图上。

*、PL/SQL_block 和 CALL procedure_name 指定触发器被触发后要执行的 PL/SQL 程序块或使用 CALL 调用的子程序。PL/SQL_block 包括可选的声明部分、必需的执行部分和可选的异常处理部分。

在语句级触发器中不能对列值进行访问和操作。

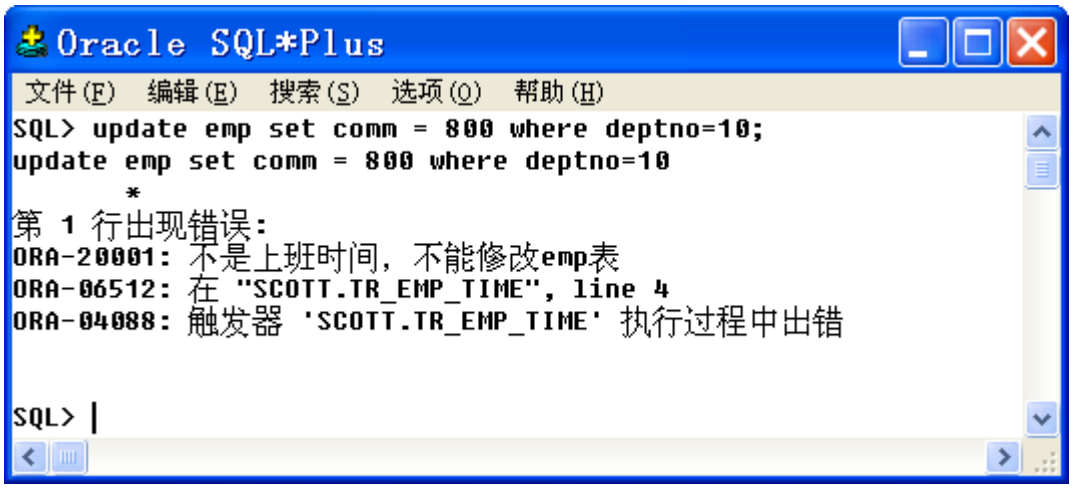
利用语句级 DML 触发器可以实现数据操作的安全保护，如限制对某个表的修改时间。

例：限制对 emp 表的修改时间范围，即不允许在非工作时间修改 emp 表。应用 BEFORE 触发该触发器。

```
CREATE OR REPLACE TRIGGER tr_emp_time
BEFORE insert
    OR update
    OR delete
ON emp
BEGIN
   IF (TO_CHAR(sysdate,'DAY') IN ('星期六','星期日'))
```

```
    OR (TO_CHAR(sysdate,'HH24') NOT BETWEEN 8 AND 12) THEN
    RAISE_APPLICATION_ERROR(-20001,'不是上班时间，不能修改emp表');
  END IF;
END;
```

当创建了该触发器后，如果在星期六、星期日或不在 8 点到 12 点之间修改 emp 表，就会显示如下错误提示：



7.6.3、行级触发器：指当执行 DML 操作时，以数据行为单位执行的触发器，即每一行都执行一次触发器。语法格式如下：

```
CREATE [OR REPLACE] TRIGGER trigger_name
    {BEFORE | AFTER}
    {INSERT | DELETE | UPDATE [OF column1 [, column2, ...]]}
    [OR { INSERT | DELETE | UPDATE [OF column3 [, column4, ...]]}...]
    ON table_name
    FOR EACH ROW
    [WHEN condition]
    PL/SQL_block | CALL procedure_name;
```

其中：

*、BEFORE 和 AFTER 指定触发时机。

*、INSERT，UPDATE，DELETE 指定触发事件。

*、OF column1 [，column2，...]用于当触发事件是 UPDATE 时，指定被修改的行，以便在条件谓词 UPDATING [(colum_x)]中指定。

*、ON table_name 指定触发对象，即该触发器被创建在哪个表或视图上。

*、FOR EACH ROW 表示该触发器时行级触发器。

*、WHEN condition 子句时可选的，用于指定触发条件，即只有满足触发条件的行才执行触发器。

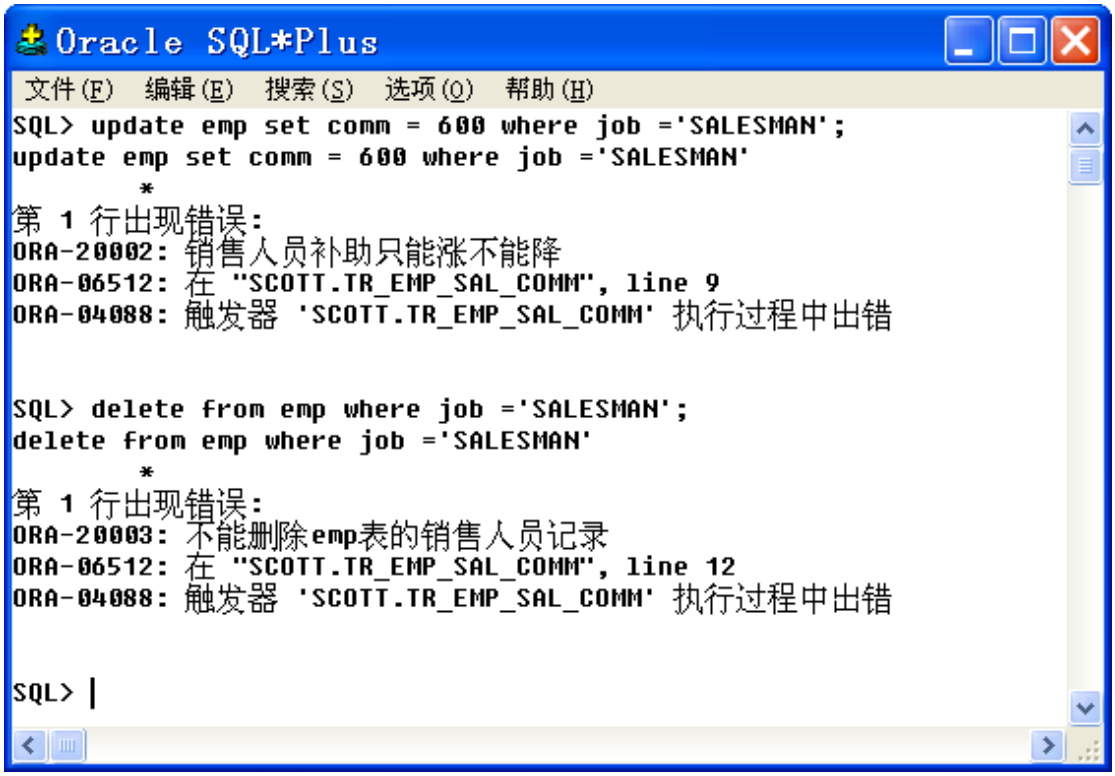
*、PL/SQL_block 和 CALL procedure_name 指定触发器被触发后要执行的 PL/SQL 程序块或用 CALL 调用的子程序。PL/SQL_block 包括可选的声明部分、必需的执行部分和可选的异常处理部分。

在行级触发器中，可以对列值进行访问。在列名前加上“OLD.”限定词就表示变化前的值，在列名前加上“NEW.”限定词就表示变化后的值。在 PL/SQL 或 SQL 语句中应用时，前面还要加“:”，但在 WHEN condition 子句中不用加“:”。

例：行级触发器，使用了条件谓词分别处理不同的触发事件，使用 WHEN condition 子句限定了只对岗位时 SALESMAN 的记录进行触发器操作。

```
CREATE OR REPLACE TRIGGER tr_emp_sal_comm
BEFORE update OF sal,comm
       OR DELETE
ON emp
FOR EACH ROW
when (old.job='SALESMAN')
BEGIN
     CASE
         WHEN UPDATING ('sal') THEN
              IF :NEW.sal < :old.sal THEN
                 RAISE_APPLICATION_ERROR(-20001,'销售人员工资只能涨不能降');
              END IF;
         WHEN UPDATING ('comm') THEN
              IF :NEW.comm < :old.comm THEN
                 RAISE_APPLICATION_ERROR(-20002,'销售人员补助只能涨不能降');
              END IF;
         WHEN DELETING THEN
              RAISE_APPLICATION_ERROR(-20003,'不能删除emp表的销售人员记录');
     END CASE;
END;
```

当创建了该触发器之后，如果修改 emp 表中小时候人员的工资和补助，或删除销售人员的记录，就会显示如下错误。



如果两个表之间的数据具有主从关系，即主表的某列数据在从表中对应多行记录，如 dept 表中的部门编号 deptno 被 emp 表的多个属于一个部门的雇员所使用，则在修改 dept 表中的 deptno 时，也应该级联地、自动地修改 emp 表中原来属于该部门的雇员的 deptno。这可以在 dept 表上创建一个触发器来实现。

例:利用行级触发器,实现级联更新。即在修改了 dept 表的 deptno 之后(AFTER)，级联地、自动地修改 emp 表中原来属于该部门的雇员的 deptno。

```
CREATE OR REPLACE TRIGGER tr_dept_emp
AFTER update OF deptno
ON dept
FOR EACH ROW
BEGIN
    UPDATE emp SET deptno = :new.deptno
    WHERE deptno = :old.deptno;
END;
```

当创建了该触发器之后，如果要修改 dept 表中的部门编号 deptno，就会级联修改 emp 表的相应雇员的部门编号 deptno。

7.6.4、INSTEAD OF 触发器

INSTEAD OF 触发器时定义在复杂视图上的触发器。复杂视图是指具有如下特征

的视图：

*、具有集合操作符（UNION，UNION ALL，INTERSECT，MINUS）

*、具有分组函数（MIN，MAX，SUM，AVG，COUNT 等）

*、具有 GROUP BY，CONNECT BY，START WITH 等子句

*、具有 DISTINCT 关键字

*、具有夺标的连接查询

对于复杂视图，不允许直接执行 DML 操作。但通过创建 INSTEAD OF 触发器，就可以在其上执行 DML 操作了，即 INSTEAD OF 触发器在后台对该 DML 操作进行分别地、分步地处理，实现其操作。

创建 INSTEAD OF 触发器需要注意以下几点：

*、只能创建在视图上，并且该视图没有指定 WITH CHECK OPTION 选项。

*、不能指定 BEFORE/AFTER 选项

*、必须指定 FOR EACH ROW 选项，即 INSTEAD OF 触发器只能在行级上触发，或是行级触发器。

*、没有必要在针对一个表的视图上创建 INSTEAD OF 触发器，只要创建 DML 触发器就可以了。

创建 INSTEAD OF 触发器的语法格式为：

```
CREATE [OR REPLACE] TRIGGER trigger_name
INSTEAD OF
{INSERT | DELETE | UPDATE [OF column1 [, column2, ...]]}
[OR {INSERT | DELETE | UPDATE [OF column3 [, column4, ...]]}]
ON table_name
FOR EACH ROW
[WHEN condition]
PL/SQL_block | CALL procedure_name;
```

例：首先创建一个复杂视图 view_dept_emp，然后在其上创建一个 INSTEAD OF 触发器 tr_i_o_dept_emp，处理向该视图插入（INSERT）记录的 DML 操作。

步骤如下：

Step1:创建复杂视图 view_dept_emp。视图是一个由 SELECT 语句选择出的列表

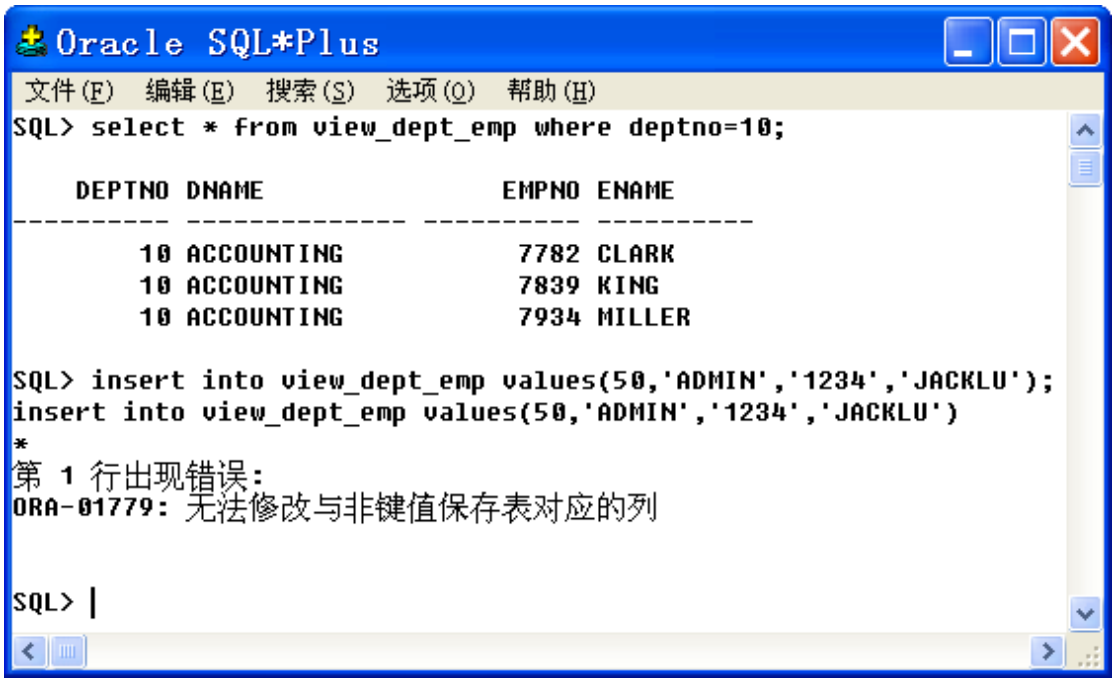
示的逻辑表，本身没有数据。当查询视图时，其数据实际上时从基础表中获取。

为了同时查询到部门与雇员的信息，可以建立一个多表的连接插接查询视图。如：

```
CREATE OR REPLACE VIEW view_dept_emp
AS
select a.deptno, a.dname, b.empno, b.ename
from dept a, emp b
where a.deptno = b.deptno
```

当创建了视图后就可以进行 SELECT 查询了，但不可以进行 INSERT 操作。

如：

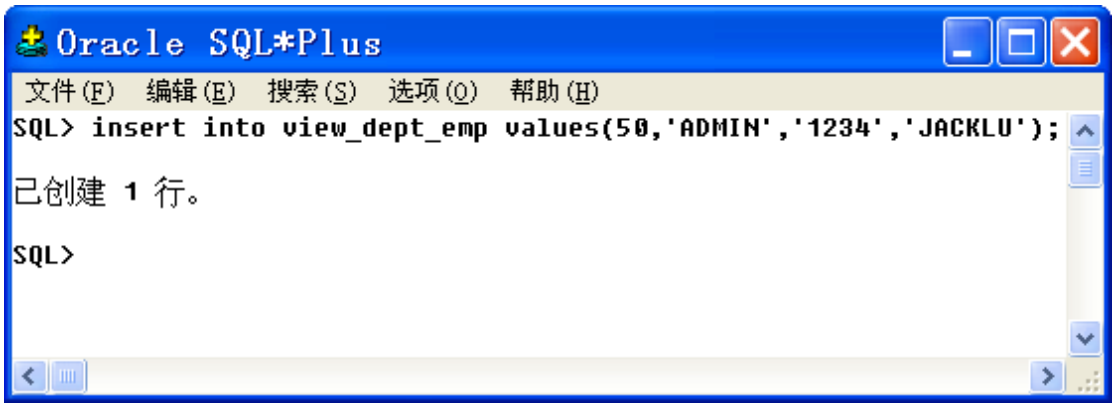


Step2:针对 INSERT 操作创建 INSTEAD OF 触发器 tr_i_o_dept_emp。

```
CREATE OR REPLACE TRIGGER tr_i_o_dept_emp
INSTEAD OF INSERT
ON view_dept_emp
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    select count(*) INTO v_count from dept
    where deptno = :new.deptno;
    IF v_count = 0 THEN
       insert INTO dept(deptno,dname)
       values (:new.deptno,:new.dname);
    END IF;
```

```
    select count(*) INTO v_count from emp
    where empno = :new.empno;
    IF v_count = 0 THEN
       insert INTO emp(empno,ename,deptno)
       values (:new.empno,:new.ename,:new.deptno);
    END IF;
END;
```

当创建了该触发器后，就可以在复杂视图 view_dept_emp 上进行前面的 INSERT 操作了，如：



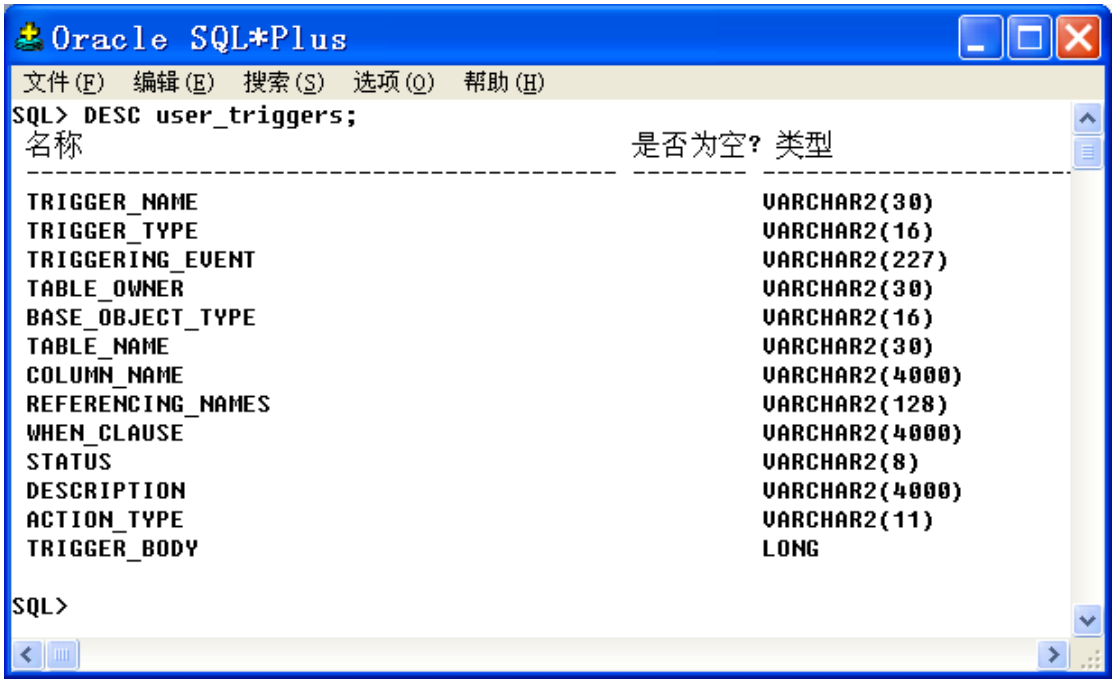
7.6.5、触发器的管理

一、查询触发器：

可以使用数据字典中的 USER_TRIGERS，ALL_TRIGERS，DBA_TRIGERS 视图来查询触发器的定义及其状态信息。

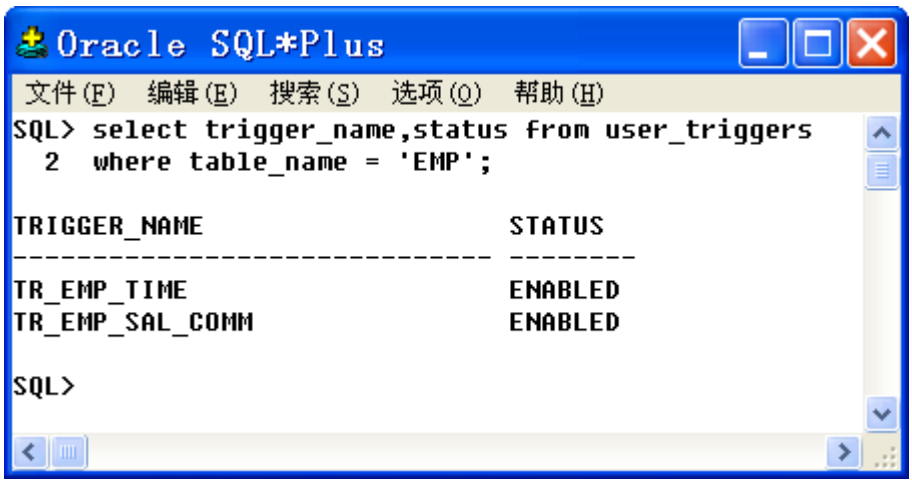
USER_TRIGERS 视图的字段如下所示：

```
SQL> DESC user_triggers;
```



例：查询当前用户在 emp 表上创建的所有触发器。

```
SQL> select trigger_name,status from user_triggers
  2  where table_name = 'EMP';

TRIGGER_NAME                   STATUS
------------------------------ --------
TR_EMP_TIME                    ENABLED
TR_EMP_SAL_COMM                ENABLED
```



二、禁止触发器

为了改善性能，并且在大量转载数据时避免触发相应的触发器（如，进行完整性效验、约束效验等），应该禁止触发器，使其暂时实效。

例：使 tr_emp_time 失效

```
SQL> ALTER TRIGGER tr_emp_time DISABLE;
```

三、激活触发器

```
SQL> ALTER TRIGGER tr_emp_time ENABLE;
```

四、禁止或激活表上的所有触发器

```
SQL> ALTER TABLE dept DISABLE ALL TRIGGERS;

SQL> ALTER TABLE dept ENABLE ALL TRIGGERS;
```

五、删除触发器

```
SQL> DROP TRIGGERS tr_emp_time;
```

第二大部分：ORACLE 10g 体系结构与存储管理

第 8 章 PL/ SQL 语言基础

从体系结构上降，完整的 Oracle 数据库包括数据库 DB、数据库管理系统 DBMS 两大部分。这两个部分分别对应的使存储网络结构和软件结构。

体系结构是从某一个角度，来分析与考察 Oracle 数据库的组成、工作原理、工作过程，它包括数据载数据库中的组织与管理机制、进程的分工协作机制，以及各个组成部分的必要性、功能、和她们之间的联系。

8．1、体系结构概述

可以从网络结构、工作原理两方面来说明 Oracle 数据库的体系结构。

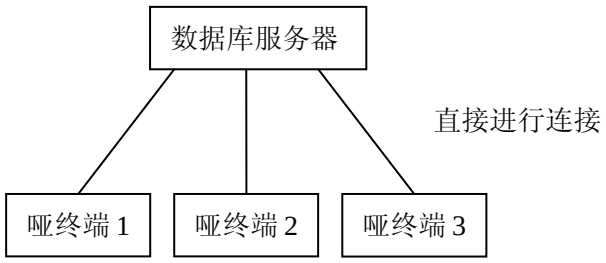
8.1.1、网络结构

1.单层结构 2.双层结构 3.n层结构

Oracle 数据库发展到今天，其网络结构的发展经历了如下三个过程（目前 Oracle 数据库能同时提供对这三种体系结构的支持）

8.1.1.1、单层结构（single-tier architecture）

自 Oracle 数据库诞生以来，此结构就一直存在，并且这一体系结构仍然被用于许多重要的大型应用领域，比如，飞机票（火车票）的订票系统、海关的出入境货物管理系统等。此结构最简单，结构图如下：



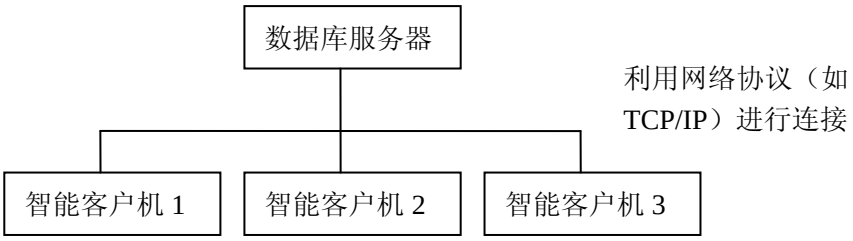
此结构的特点是使用基于字符的非图形终端设备直接串行的连接到 Oracle 数据库。所有的智能和处理都发生载大型机上。单层结构的配置和管理较方便，不存在网络协议问题，也不存在多操作系统的复杂性问题。

此结构可缩放性和灵活性方面有些受限制，大型机的性能决定了整个系统的性能。

8.1.1.2、双层结构（two-tier architecture）

此结构是由于 PC 机的出现而流行起来的。此结构在一些小规模的应用系统中得到广泛使用。此结构的一些特点明显地胜过单层结构，如客户机具有图形用户界面，易于理解、学习、操作；客户机具有智能，可以进行处理，减轻了对服务器性能的需求。

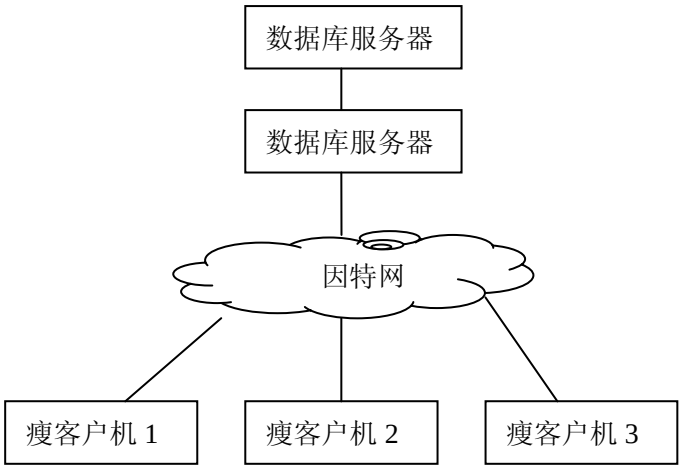
双层结构也被称为客户机/服务器（client/server）结构，结构如下图：



此结构不适应较大规模的较复杂的系统。软件、硬件的冗余较大，每个客户机都需要安装软件，硬件成本比哑终端高。难以处理网络之间的协议异构、硬件异构和操作系统异构的问题，显然其可缩放性和灵活性方面也是受限制的。

8.1.1.3、n 层结构（n-tier architecture）

n 层结构是双层结构之后发展起来的一种结构。它在客户机和数据库服务器之间引进了中间件（Middleware），如应用服务器或 Web 服务器。利用因特网的 n 层结构如下图：



在 n 层结构中，中间件的作用包括如下几个方面：

*、为适应不同网路协议，进行数据转换。

*、在客户机和服务器之间充当防火墙，控制客户机对服务器的访问。

*、实现客户机、服务器中的业务逻辑。

*、执行事务和监视客户机与服务器之间的活动，以便在多个服务器之间平衡负载。

*、在现有系统和新增系统之间充当网关。

此结构饱表示、业务逻辑、路由选择、数据库处理等任务分别放在瘦客户机、应用服务器、数据库服务器等多台计算机上，这说明此结构能适应大规模、较复杂的系统，具有可缩放性。

此结构的客户端浏览器给用户提供了一个一致的表示界面，其操作方式是一致的，可以减少对用户的培训。应用软件只需要安装在服务器端，而不需要安装在客户机端。

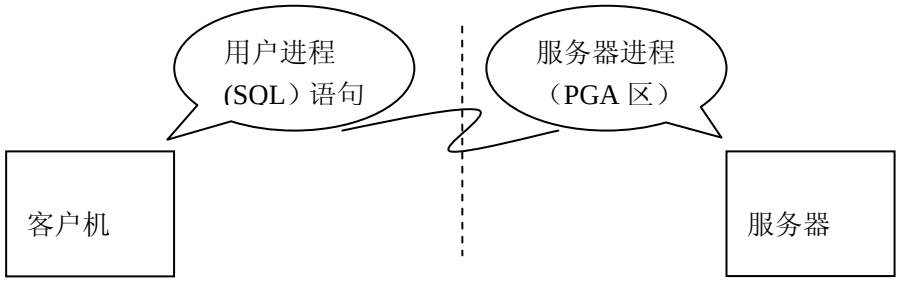
8.1.2、工作原理

Oracle 数据库的工作原理，包括数据库系统处理过程和体系结构两个方面。

8.1.2.1、数据库系统处理过程

要使用数据，必须连接到数据库。当用户运行一个应用程序（如 SQL*PLUS）时，实际上是在客户机启动一个用户进程，并将连接请求通过网络发送到服务器。

服务器上的数据库会为该用户进程派生一个对应的服务进程，其数据库系统处理过程如下图：



处理过程可以简单地描述为：

1、用户在其计算机上运行基于 Oracle 的应用程序，即启动用户进程。

2、在客户机、服务器之间建立连接（CONNECT）。

3、在建立连接的基础上，为用户建立会话（SESSION），并为该会话创建一个 PGA 区（Program Global Area，程序全局区）以存储与该会话相关的信息。在同一个连接中，不同的用户由不同的会话。

4、启动服务进程，由该服务进程负责执行该会话的各项任务。

5、用户进程发送 SQL 语句（如 SELECT，UPDATE，COMMIT）等。

6、服务器进程解析、编译、执行 SQL 语句，然后将结果写入数据库并返回给用户进程。

7、用户进程接收返回的 SQL 执行结果。

8、在应用程序中显示 SQL 执行结果。

8.1.2.2、总体结构

从作用和工作原理来看，可以将总体结构分成三大部分。

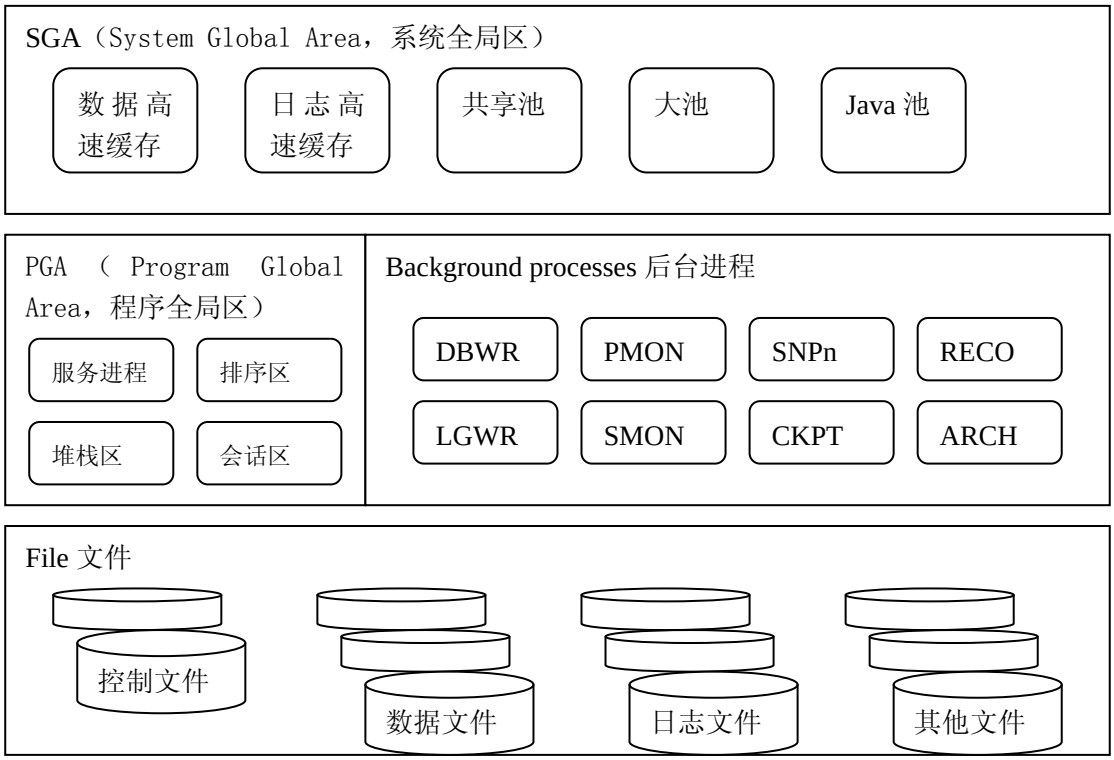
*、内存结构：包括 SGA 和 PGA。使用内存最多的是 SGA，同时也是影响数据库性能的最大参数。

*、进程结构：包括前台进程、后台进程。前台进程是指服务进程和用户进程。前台进程是根据实际需要而运行的，并在需要结束后立刻结束。后台进程是指在 Oracle 数据库启动后，自动启动的几个操作进程。

*、存储结构：包括控制文件、数据文件、日志文件等操作系统文件。

Oracle 数据库的例程由 SGA（System Global Area）和后台进程组成。

如下图所示：



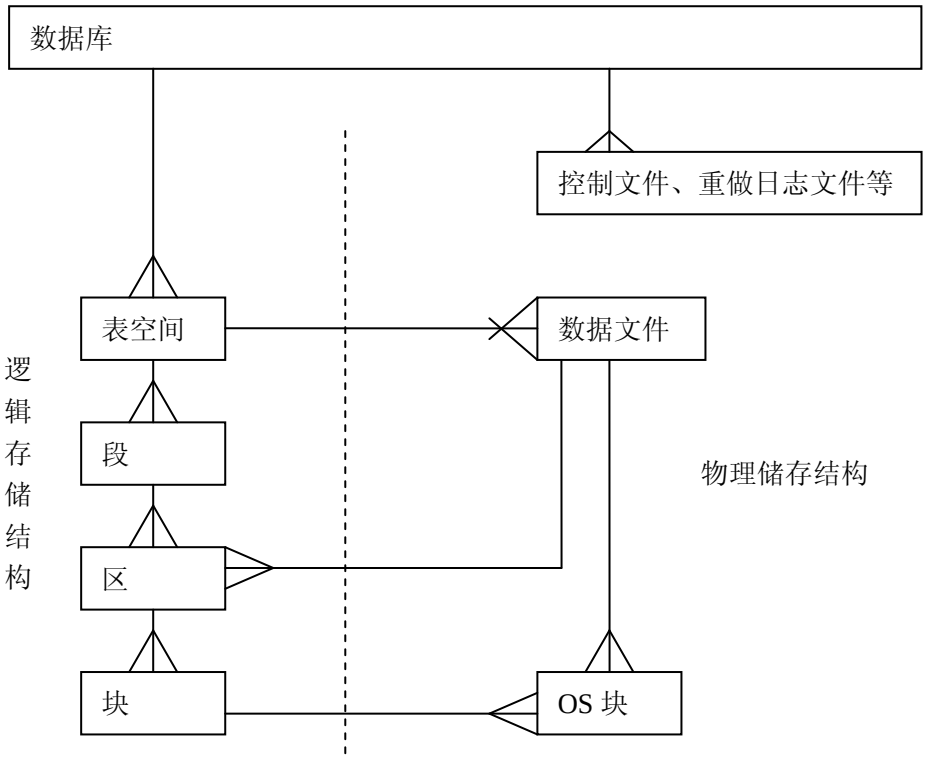
8．2、存储结构

Oracle 数据库的存储结构分为逻辑存储结构和物理存储结构，这两种存储结构既相互独立又相互联系。

逻辑存储结构主要描述 Oracle 数据库的内部存储结构，即从技术概念上描述 Oracle 数据库中如何组织、管理数据。因此，逻辑存储结构是和操作系统平台无关的，是由 Oracle 数据库创建和管理的。

物理存储结构主要描述 Oracle 数据库的外部存储结构，即在操作系统中如何组织、管理数据。因此物理存储结构是和操作系统平台有关的。物理存储结构是逻辑存储结构在物理上的、可见的、可操作的、具体的实现形式。物理存储结构对应的操作系统文件存储在磁盘上。

Oracle 数据库的存储结构，以及这两种存储结构之间的关系如图：



从物理上看，数据库是由控制文件、数据文件、重做日志文件等操作系统文件组成的；从逻辑上看，数据库是由系统表空间、用户表空间等表空间组成的。表空间是最大的逻辑单位，块是最小的逻辑单位。逻辑存储结构中的块最后对应到操作系统中的块。

8.2.1、逻辑存储结构

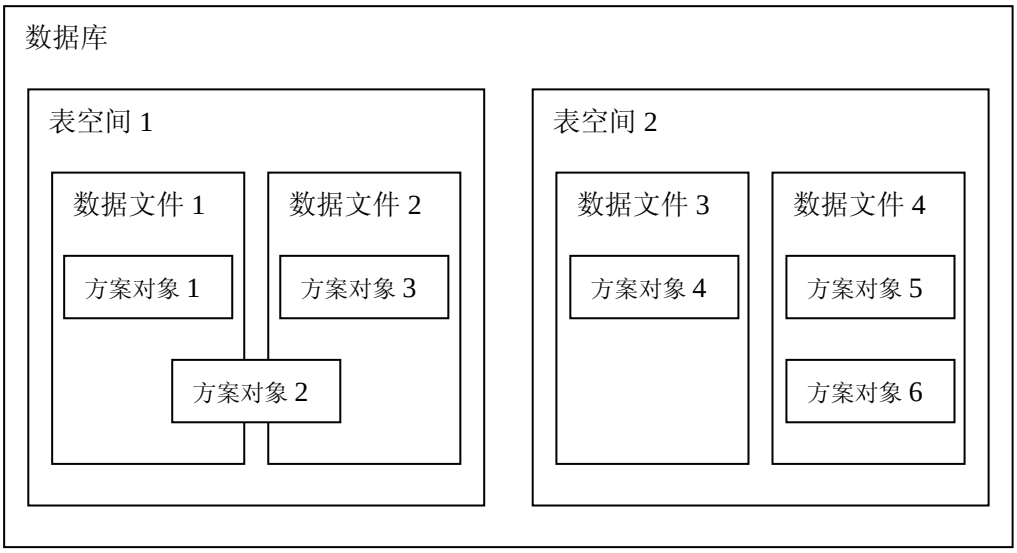
逻辑存储结构，主要描述 Oracle 数据库的内部存储结构，即从技术概念上描述在 Oracle 数据库中如何组织、管理数据。因此，在操作系统中无法找到逻辑存储结构，但通过查询 Oracle 数据库的数据字典，可以找到逻辑存储结构的描述。

逻辑存储结构包括表空间、段、区、数据块。简单的说，逻辑存储结构之间的关系是：多个数据块组成区，多个区组成段，多个段组成表空间，多个表空间组成逻辑数据库

方案对象也是一种逻辑结构，但它是数据组织的逻辑结构而不是数据存储的逻辑结构。

8.2.1.1、表空间

表空间是最大的逻辑单位，任何方案对象（如表、索引）都被存储在表空间的数据文件中。表空间分为系统表空间和非系统表空间两类。系统表空间包括 SYSTEM 表空间和 SYSAUX 表空间，其余的表空间就是非系统表空间。（系统表空间在所有数据库中都是必需的）下标表示了数据库、表空间、方案对象之间的关系：



安装完成后，自动创建的表空间名称及其说明如下表：

名称	说明
EXAMPLE	实例表空间，存放例子数据库的方案对象信息及其培训资料
SYSAUX	辅助系统表空间，用于减少系统表空间的负荷，提高系统的作业效率，是 Oracle10g 新增加的表空间
SYSTEM	系统表空间，存放关于表空间的名称、控制文件、数据文件等管理信息，它属于 SYS，SYSTEM 方案，仅被 SYS 和 SYSTEM 或其他具有足够权限的用户使用。即使是 SYS 和 SYSTEM 用户也不能删除或重命名此表空间
TEMP	临时表空间，存放临时表和临时数据，用于排序。每个数据库都应该由一个（或创建一个）临时表空间，以便在创建用户时将其分配给用户，否则将会使用 TEMP 表空间
UNDOTBS1	重做表空间，存放数据库的有关重做的相关信息和数据

USERS	用户表空间，存放永久性用户对象和私有信息，因此也被称为数据表空间。每个数据库都应该由一个（或创建一个）用户表空间，以便在创建用户时将其分配给用户，否则将会使用 SYSTEM 表空间。一般地，系统用户使用 SYSTEM 表空间，非系统用户使用 USERS 表空间

一、SYSTEM 表空间

SYSTEM 表空间存放的信息如下：

*、表空间名称、控制文件、数据字典、数据文件等管理信息。

*、方案对象（如表、索引、同义词、序列）的定义信息。当在数据库中创建一个新对象时，对象的实际数据可以存储在其他表空间中，但对象的定义信息却应该保存在 SYSTEM 表空间中。

*、所有 PL/SQL 程序的源代码和解析代码，包括存储过程、函数、包等。在需要大量 PL/SQL 程序的数据库中，应当设置足够大的 SYSTEM 表空间。

*、SYSTEM 撤销段。

二、SYSAUX 表空间

SYSAUX 表空间用来存储与 Oracle 供给特性有关的方案对象，比如空间数据选项、XMLDB（extensible Markup Language DataBase）或 Intermedia，还被用于减少系统表空间的负荷，提高系统的作业效率。

一个小型的数据库只需要 SYSTEM，SYSAUX，TEMP 表空间便可以进行所有的工作了。但，Oracle 建议使用多个表空间。SYSTEM 和 SYSAUX 表空间只存放系统信息（数据字典），不存放非系统信息（用户的私有信息）。将不同类型的数据部署到不同的表空间（如将表存放在数据表空间，将索引存放在索引表空间，将临时数据存放在临时表空间），一方面可以提高数据的访问性能，另一方面便于数据的管理、备份、恢复等操作。

8.2.1.2、段（segment）

段用于存储表空间中某一种特定的具有独立存储结构的对象的所有数据，它由一个或多个区组成。按照段中所存储数据的特征和用途的不同，可以将段分成几种类型。

一、数据段（表段）

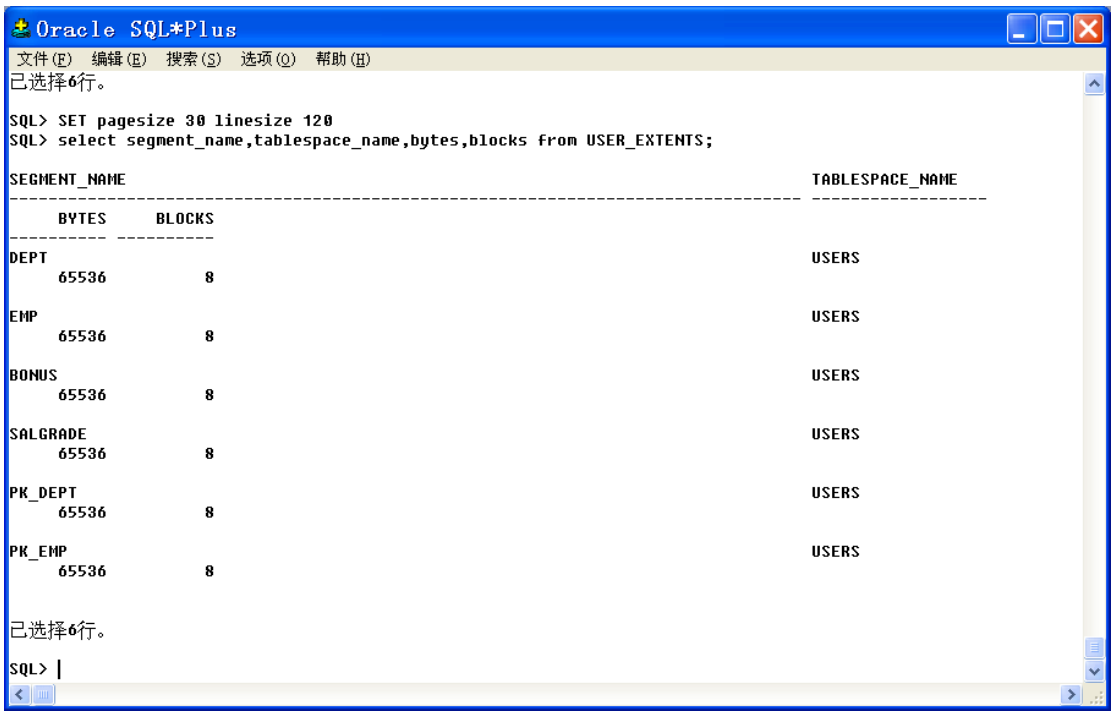
数据段存储表中的所有数据。当某个用户创建表时，就会在该用户的默认表空间中为该表分配一个与表名相同的数据段，以便将来存储该表的所有数据。如果创建的时分区表，则为每个分区分配一个数据段。显然，在一个表空间中创建了几个表，该表空间中就有几个数据段。

数据段随着数据的增加而逐渐变大。段的增大过程是通过增加区的个数而实现的。每次增加一个区，每个区的大小是块的整数倍。

例：查询 SCOTT 使用的数据段存储在哪个表空间、有多大、有几个区，可以用如下语句查询数据字典 USER_EXTENTS。

```
SQL> select segment_name,tablespace_name,bytes,blocks from USER_EXTENTS;
```

其中 segment_name 是数据段名；tablespace_name 是存储该数据段的表空间名；bytes 是数据段的大小；blocks 是块的个数。



二、索引段

索引段存储索引的所有数据。当用户用 CREATE INDEX 语句创建索引，或在定义约束（如主键）而自动创建索引时，就会在该用户的默认表空间中为该索引分配一个与索引名相同的索引段，以便将来存储该索引的所有数据。如果创建的是分区索引，则为每个分区索引分配一个索引段。

例：查询 SCOTT 使用的索引信息，可以用如下语句查询数据字典 USER_INDEXES.

```
SQL> select index_name,table_owner,table_name,tablespace_name from user_indexes;
```

三、临时段

临时段存储排序操作所产生的临时数据。当用户使用 ORDER BY 语句进行排序或汇总时，在该用户的临时表空间中自动创建一个临时段，排序结束，临时段自动消除。

在 Oracle 中，临时表空间一般通用，所有用户的默认临时表空间都是 TEMP 表空间。当然，可以在创建用户时，或创建用户之后，指定临时表空间。

注意：由于在创建临时段时存在空间的分配、回收、再分配的特点，容易造成不连续的磁盘碎片，因此，为了优化系统性能，Oracle 建议使用专用的临时表空间作为用户的临时表空间。

四、回退段

回退段存储数据修改之前的位置和值。它的原理与实现是一项十分复杂的技术，已经面临淘汰。自 Oracle9i 以来，增加了 UNDO 表空间，并增加了自动撤销管理功能来代替回退段的功能。（建议 DBA 使用自动撤销管理功能而不要使用回退段功能。）

例：查询是否使用了自动撤销管理功能：

```
SQL> conn sys/orclsys@orcl as sysdba
已连接。
SQL> select value from v$parameter where name='undo_management';

VALUE
--------------------------------------------------------------------
AUTO
```

上面的查询结果为 AUTO，表示使用的是自动撤销管理功能。如果要使用回退段功能，需要先将 undo_management 参数设置为 manual，并重新启动数据库。

8.2.1.3、区（extent）

区是由物理上连续存放的块构成的。区是 Oracle 存储分配的最小单位，由

一个或多个块组成区，由一个或多个区组成段。

当在数据库中创建带有实际存储结构的方案对象(如表、索引、簇)时，Oracle 将该方案对象分配若干个区，以便组成一个对应的段来为该方案对象提供初始的存储空间。当段中已的区都写满后，Oracle 就为该段分配一个新的区，以便容纳更多的数据。

可以通过在 CREATE TABLE 语句的 STORAGE 子句中设置 3 个存储参数来指定这个表的数据段的存储区大小、第 1 个后续区大小和后续区增加的比例。

例如：

STORAGE (INITIAL 128K

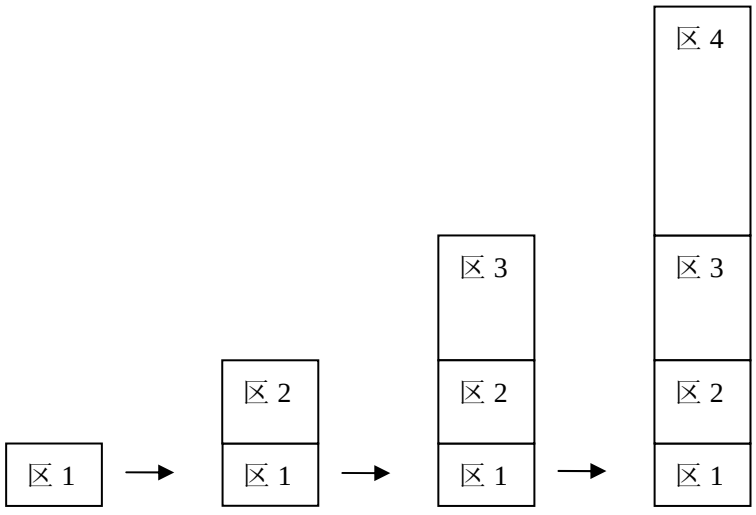
NEXIT 40K

PCTINCREASE 50)

该 STORAGE 子句表示，创建该表时，首先分配第 1 个 128KB 的初始区（INITIAL），用完后分配第 2 个 40KB 的后续区（NEXT），以后每个区的增长比例为 50（PCTINCREASE），即第 3 个后续区的大小为（1280＋40*50%）=148KB,第 4 个后续区为（148+148*50%）=212KB，依次类推。其计算公式是：

$$SIZEn = NEXT * (1 + PCTINCREASE/100)^{(n-2)}$$

其中，n 表示第 n 个区（除第 1 个区和第 2 个区之外），如下图：



8.2.1.4、块（block）

块是最小的数据管理单位，也是执行输入输出操作时最小的单位。相对应地，操作系统执行输入输出操作的最小单位是操作系统块。

块的大小是操作系统块大小的整数倍。以 Windows 2000 为例，操作系统块的大小是 4KB，所以块的大小可以是 4KB，8KB，16KB，等等。

如果块的大小是 4KB，EMP 表每行的数据占 100 个字节。如果某个查询语句只返回 1 行数据，那么，在将数据读入到数据高速缓存时，读取的数据量是 4KB 而不是 100 个字节。

1控制文件 2数据文件 3重做日志文件4其他文件（参数文件、口令文件、后台进程跟踪文件、服务进程跟踪文件 ）

8.2.2、物理存储结构

物理存储结构是现实的数据存储单元，对应于操作系统文件，比逻辑存储结构更容易理解。Oracle 数据库就是由驻留在服务器的磁盘上的这些操作系统文件所组成的。这些文件有控制文件、数据文件、重做日志文件。跟 Oracle 数据库有关，但从技术上看，不属于 Oracle 数据库的附件文件有密码文件(PWD.ORA)、参数文件（SPFILE.ORA）和归档重做日志文件。

8.2.2.1、控制文件（control file）

此文件是一个很小的（通常是数据库中最小的）文件，大小一般在 1MB 到 5MB 之间，为二进制文件。但它是数据库中的关键性文件，它对数据库的成功启动和正常运行都是至关重要的，因为它存储了在其他地方无法获得的关键信息，这些信息包括：

*、数据库的名称

*、数据文件和重做日志文件的名称、位置和大小

*、、发生磁盘故障或用户错误时，用于恢复数据库的信息

注意：每个数据库必须，而且只需要有一个控制文件

在装载（mount）数据库时，Oracle 将读取控制文件中的信息，以便判断数据库的状态，获得数据库的物理结构的信息物理文件的使用权。因此，控制文件对于数据库的成功装载，以及其后的打开都是非常重要的。只有控制文件是正常的，才能装载、打开数据库。

在数据库运行的过程中，每当出现数据库检查点（checkpoint）或修改数据库结构之后，Oracle（只能由 Oracle 本身）就会修改控制文件的内容。DBA 可以通过 OEM 工具修改控制文件中的部分内容（如是否归档），但 DBA 和用户都不应该人为地修改控制文件中的内容，否则会破坏控制文件。

注意：如果控制文件丢失或被破坏了，那对数据库来说将是不可挽救的损失，所以，应该定期对数据库的控制文件进行备份，并将备份保存在不同的硬盘上。另外，为了安全原因，可以创建多个控制文件（最好将它们放置到不同的磁盘上），互为镜像进行复用，以便某个控制文件损坏之后，还可以利用其他控制文件进行工作。

在 Oracle 10g 安装完毕后，自动创建的 3 个控制文件如下：

C:\oracle\product\10.1.0\oradata\oamis\CONTROL01.CTL

C:\oracle\product\10.1.0\oradata\oamis\CONTROL02.CTL

C:\oracle\product\10.1.0\oradata\oamis\CONTROL03.CTL

8.2.2.2、数据文件（data file）

数据文件是实际存储插入到数据库表中的实际数据的操作系统文件。数据文件的大小与它们所存储的数据量的大小直接相关，会自动增大（但即便删除数据后也不会减少）。

在创建表空间时，Oracle 会同时为该表空间创建第一个数据文件。如果这个数据文件很大，这个创建过程会用较长时间。新创建的数据文件不包含任何数据，只是一个准备存储数据的空容器。

随着不断地在表空间中创建、插入、更新数据，表空间所对应的所有数据文件的物理存储空间将被用完。这时，就需要为该表空间增加新的存储空间，或者时创建新的数据文件，或者是调整现有的数据文件的存储空间大小参数。

除 SYSTEM 表空间之外，任何表空间都可以由联机状态切换为脱机状态。当表空间进入脱机状态后，组成该表空间的数据文件也就进入脱机状态了。也可以将表空间中的某一个数据文件单独地设置为脱机状态，以便进行数据库的备份或恢复。

注意：正在使用中的数据文件是不能复制的。

安装完 Oracle 后，自动创建的 6 个表空间数据文件如下：

C:\oracle\product\10.1.0\oradata\oamis\EXAMPLE01.DBF

C:\oracle\product\10.1.0\oradata\oamis\SYSAUX01.DBF

C:\oracle\product\10.1.0\oradata\oamis\SYSTEM01.DBF

C:\oracle\product\10.1.0\oradata\oamis\TEMP01.DBF

C:\oracle\product\10.1.0\oradata\oamis\UNDOTBS01.DBF

C:\oracle\product\10.1.0\oradata\oamis\USERS01.DBF

8.2.2.3、重做日志文件（redo file）

当用户对数据进行修改时，Oracle 实际上时先在内存中进行修改，过一段时间后，再集中将内存中的修改结果成批地写入上面的数据文件中。

Oracle 采用这种方法，主要时出于性能上面的考虑。因为，对数据操作而言，硬盘的比内存的速度要慢上万倍。

但如果在将内存中的修改结果写入数据文件之前发生故障，那么这些结果就会丢失。这时就需要一种机制，能时刻把这些修改结果保存下来，以便在故障发生之后，还能重现当时的数据操作，进行数据库的恢复。

Oracle 是用重做日志文件来随时保存这些修改结果的，即 Oracle 随时将内存中的修改结果保存到重做日志文件中，“随时”表示在将修改结果写入数据文件之前，分几次写入重做日志文件。因此，即使发生故障导致数据库崩溃，Oracle 也可以利用重做日志文件中的信息来恢复丢失的数据。只要某项操作的重做信息没有丢失，就可以利用重做信息来重现该操作。

每个数据库至少需要两个重做日志文件，因为 Oracle 是以循环方式来使用重做日志文件的。当第 1 个重做日志文件被写满之后，后台进程 LGWR（日志写进程）开始写第 2 个重做日志文件，当第 2 个重做日志文件写满后，又开始写入第 1 个重做日志文件，依次类推。

安装完 Oracle 后，自动创建的 3 个重做日志文件如下：

C:\oracle\product\10.1.0\oradata\oamis\REDO01.DBF

C:\oracle\product\10.1.0\oradata\oamis\REDO02.DBF

C:\oracle\product\10.1.0\oradata\oamis\REDO03.DBF

8.2.2.4、其他文件

其他文件包括参数文件、口令文件、归档日志文件和后台进程跟踪文件等。

一、参数文件（）

参数文件也被称为初始化参数文件，用于存储 SGA、可选的 Oracle 特性和后台进程的配置参数，分为文本参数文件（pfile）和服务器参数文件（spfile）。可以使用其中之一来配置例程和数据选项。文本参数文件可以使用文本编辑器进

行编辑。服务器参数文件是二进制文件，不能直接用文本编辑器进行编辑。

参数文件类似于 Microsoft DOS 系统的 autoexce.bat 和 config.sys 文件。当数据库启动，并在创建例程或读取控制文件之前，会先读取参数文件，并按其中的参数进行例程的配置。

C:\oracle\product\10.1.0\db_1\database\SPFILEOAMISSID.ORA

二、口令文件（password file）

口令文件是个二进制文件，用于验证特权用户。特权用户是指具有 SYSOPER 或 SYSDBA 权限的特殊数据库用户。这些用户可以启动例程、关闭例程、创建数据库、执行备份恢复等操作。创建 Oracle 数据库，默认的特权用户是 SYS。

三、归档日志文件

非活动的重做日志文件的备份。通过使用归档日志文件，可以保留所有重做历史记录。

四、后台进程跟踪文件

记录后台进程的警告或错误信息。每个后台进程都有相应的跟踪文件。

五、服务进程跟踪文件

记录服务进程的相关信息，用于跟踪 SQL 语句、诊断 SQL 语句的性能，并实施相应的性能调整。

8．3、软件结构

Oracle 数据库服务器主要由两部分组成：物理数据库和数据库管理系统。物理数据库是保存数据的物理存储设备。数据库管理系统是用户与物理数据库之间的一个中间层，是软件层。这个软件层具有一定的结构。

8.3.1、软件结构

Oracle 数据库的软件结构又被称为例程结构。在启动数据库时，Oracle 首先要在内存中获取、划分、保留各种用途的区域（表现一定的结构），运行各种用途的后台进程，即创建一个例程（instance），然后再由该例程装载（mount）、打开（open）数据库，最后由这个例程来访问和控制数据库的各种物理结构。

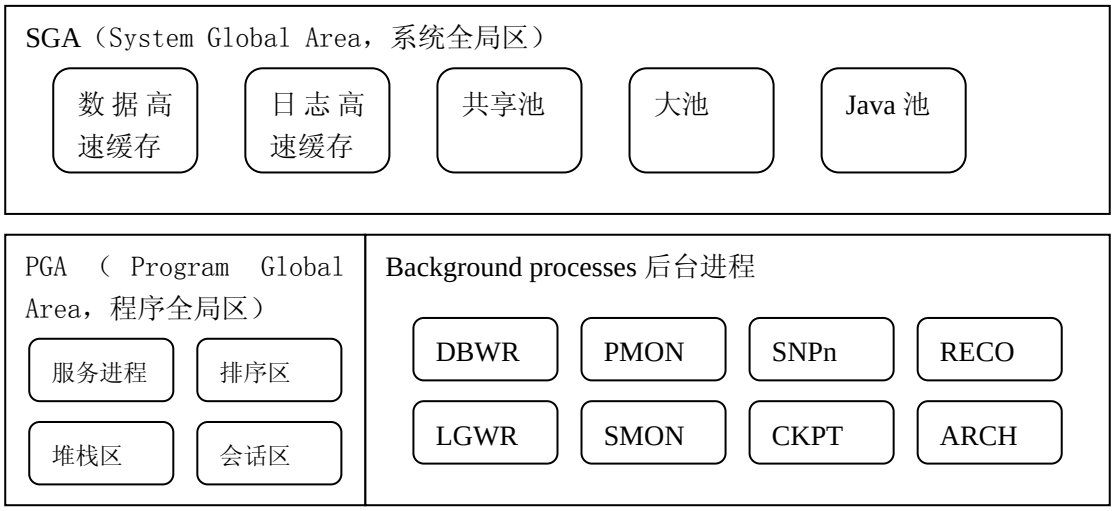
当用户连接到数据库并使用数据库时，实际上是连接到该数据库的例程，通

过例程来连接、使用数据库。所以例程是用户和数据库之间的一个中间层。

例程和数据库是有很大区别的。这里的数据库主要是指用于存储数据的物理结构，总是实际存在的；例程是由操作系统的内存结构和一系列进程所组成的，可以启动和关闭。

一台计算机上可以创建多个 Oracle 数据库，当同时要使用这些数据库时，就要创建多个例程。为了不使这些例程相混搅，每个例程都要用称为 SID（SystemIDentify，系统标识符）的符号来区分，即创建这些数据库时填写的数据库 SID。

软件结构由内存结构和进程结构组成，如下图：



8.3.2、内存结构

从计算机的体系结构和各部分功能来说，内存是用来保存指令代码和缓存数据的。要运行一个软件程序，必须先要在内存中为其指领代码和缓存数据申请、划分一个区域，再将其从磁盘上读入、放置到内存，然后才能执行。Oracle DBMS 是一个应用程序，所以它的执行也不例外，需要放置到内存中才能执行。

内存结构是 Oracle 数据库体系结构中最为重要的一部分，内存也是影响数据库性能的第一因素。内存的大小、速度直接影响数据库的运行速度。特别是当用户数增加时，如果内存不足，例程分配不到足够的内存，就会使有些用户连接不到数据库，或连接、查询的速度明显下降。

按照对内存的使用方法的不同，Oracle 数据库的内存可以分为 SGA（System Global Area，系统全局区）、PGA。

8.3.2.1、SGA（System Global Area,系统全局区）

SGA 区时例程内存结构的主要组成部分，每个例程都只有一个 SGA 区。当多个用户同时连接到一个例程时，所有的用户进程、服务进程都可以共享使用 SGA 区。它是不同用户进程与服务进程进行通信的中心，数据库的各种操作主要都在 SGA 区中进行，所以将其称为系统全局区。DBA 应该对 SGA 区的组成和原理有所了解。

注意：创建例程时，Oracle 为 SGA 区分配内存；终止例程时，释放 SGA 区占用的内存。

按照存放信息的类型不同，SGA 区可以分为几个部分。

一、数据高速缓存

数据高速缓存保存的是最近从数据文件中读取的数据块，其中的数据可以被所有用户共享。

当被访问的数据只在数据文件中时，Oracle 将读取磁盘上的数据文件，然后将其放入数据高速缓存中，再对数据进行处理；如果被访问的数据已经位于数据高速缓存中时，Oracle 将直接使用数据高速缓存中的数据，而不必再读取磁盘中的数据文件了。由于读取内存的速度比读取磁盘快很多倍，所以这种机制能提高数据库的整体效率。

数据高速缓存由许多大小相等的缓存块组成。这些缓存块分为三类：

*、脏缓存块：此缓存块中保存的是已经被修改过的缓存块。当一条 SQL 语句对某个缓存块中的数据进行修改后，这个缓存块就被标记为脏缓存块。它们迟早要被 DBWR 进程写入对应的硬盘中的数据文件，以便永久性地保留修改的结果。

*、空闲缓存块：此缓存块中没有数据，它在等待被写入数据。当 Oracle 从数据文件中读取数据后，将会寻找空闲缓存块，以便将数据写入其中。

*、命中缓存块：此缓存块保存的是最近正在被访问的缓存块。命中缓存块将始终被保留在数据高速缓存中，不会被写入数据文件。

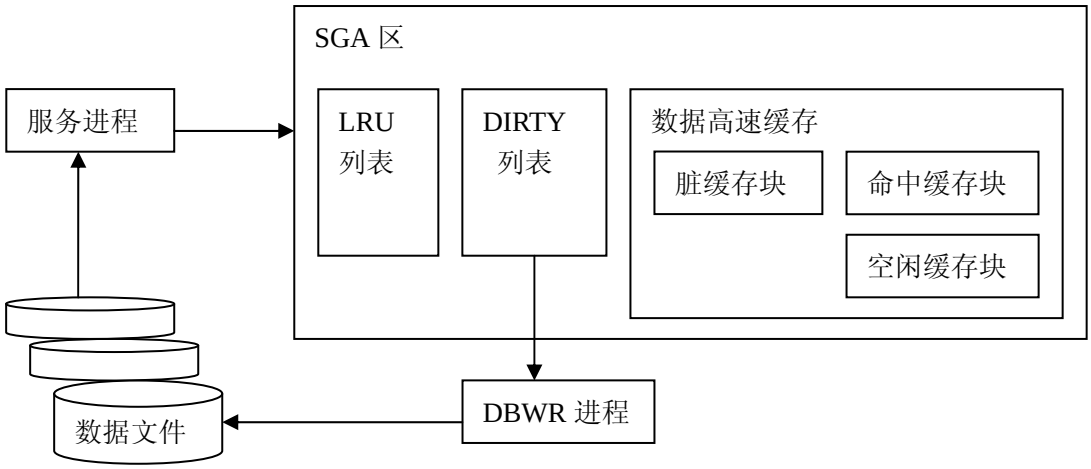
Oracle 通过两个列表来管理上述缓存块“

*、DIRTY 列表保存已经被修改但还没写入数据文件的脏缓存块。

*、LRU（Least Recently Used）列表保存所有空闲缓存块、命中缓存块，

以及还没由被移入 DIRTY 列表中的脏缓存块。可以将 LRU 列表看成是一个队列，当数据高速缓存中某个缓存块被访问后，这个缓存块就会被移动到 LRU 列表的头部，而其他缓存块就会向 LRU 列表的尾部移动。放在尾部的缓存块最先被移出 LRU 列表。

数据高速缓存的工作原理如下图：



Oracle 在将数据文件中的数据块复制到数据高速缓存中之前，必须先在数据高速缓存中找到空闲缓存块以便容纳该数据块。所以 Oracle 将从 LRU 列表的尾部开始搜索，直到找到所需要的空闲缓存块为止。

在搜索 LRU 列表时，如果先搜索到的是脏缓存块，就将其移入 DIRTY 列表中，然后继续搜索；如果搜索到的是空闲缓存块，就将数据库写入其中，然后再将该缓存块移动到 LRU 列表的头部。

如果能够搜索到足够的空闲缓存块，即能将所有数据块都写入到对应的空闲缓存块中，则该搜索写入过程结束。

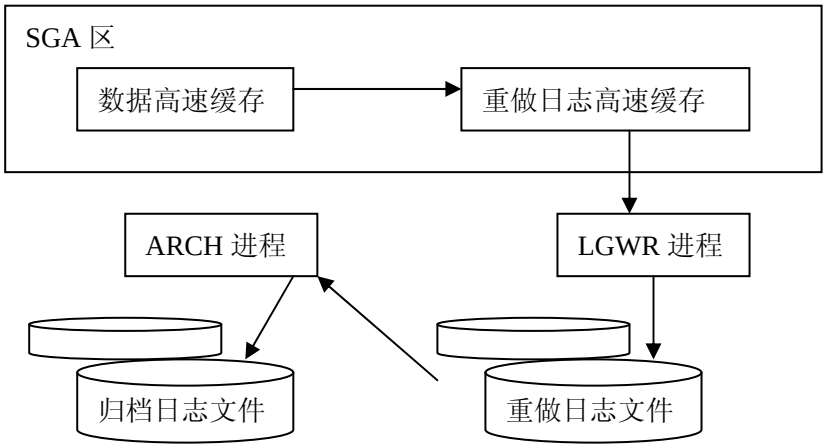
如果没有搜索到足够的空闲缓存块，即不能将所有数据块都写入到对应的空闲缓存块中，则 Oracle 将先停止对 LRU 列表的搜索，激活 DBWR（数据库写）进程，开始将 DIRTY 列表中的脏缓存块写入数据文件。已经被写入数据文件的脏缓存块将变成空闲缓存块，并被放入 LRU 列表中。执行完这项工作后，再重新开始搜索，这样就可以找到足够的空闲缓存块了。

二、重做日志高速缓存

当执行 INSERT、UPDATE、DELETE 语句对表进行修改时，或执行 CREATE，ALTER，

DROP 等语句创建方案对象时，Oracle 都会为这些操作生成重做记录。重做日志高速缓存就是用于存储重做记录的缓存。

为了加快访问的速度和工作效率，重做记录并不直接写入磁盘的重做日志文件中，而是首先被写入重做日志高速缓存，当重做日志高速缓存中的重做记录达到一定数量后，再由 LGWR（日志写）进程将其写入重做日志文件中（即，Oracle 总是“先日志后文件”或“先内存后磁盘”）。当出现重做日志文件切换时，由 ARCH（归档进程）将重做日志文件中的数据写入归档日志文件中，以作为备份，如下图所示：



重做日志高速缓存的大小由 LOG_BUFFER 初始化参数指定。可以在数据库运行期间修改该参数。重做日志高速缓存的大小对数据库的性能有很大影响。选择较大的值可以减少对重做日志文件的磁盘操作次数，比较适合长时间运行的、会产生大量重做记录的事务。

例：使用 SHOW 命令，可以查询重做日志缓存的大小。

```
SQL> conn sys/orclsys@orcl as sysdba
已连接。
SQL> show PARAMETER log_buffer;

NAME                                 TYPE        VALUE
------------------------------------ ----------- -----------
log_buffer                           integer     2899456
```

例：通过数据字典 V$SYSSTAT，可以查询用户进程等待重做日志缓存的

次数。

```
SQL> select name,value from V$SYSSTAT
  2  where name='redo buffer allocation retries';

NAME                                     VALUE
---------------------------------------- ----------
redo buffer allocation retries           0
```

三、共享池

共享池保存了最近执行的 SQL 语句、PL/SQL 程序和数据字典信息，是对 SQL 语句和 PL/SQL 程序进行语法分析、编译、执行的内存区。它主要由数据字典缓存（dictionary cache）、库缓存（library cache）组成。

*、数据字典缓存：在 Oracle 数据库的运行过程中，Oracle 会频繁地对数据字典中的表、视图进行访问，以便确定操作的数据库对象是否存在、是否具有合适的权限等信息。

为了提高访问的效率，Oracle 在共享池的数据字典缓存中保存了最常使用的数据字典信息，如数据库用户的账户、数据库的结构信息等。

在数据字典缓存中保存的是一条一条的记录（就像是内存中的小数据库一样），而其他缓存区中保存的是数据块。

*、库缓存：Oracle DBMS 在执行用户进程提交的各种 SQL 语句、PL/SQL 程序之前，先要对其进行语法上的解析、对象上的确认、权限上的判断、操作上的优化等一系列操作，并生成执行计划。这一系列操作会占用一定的系统资源。

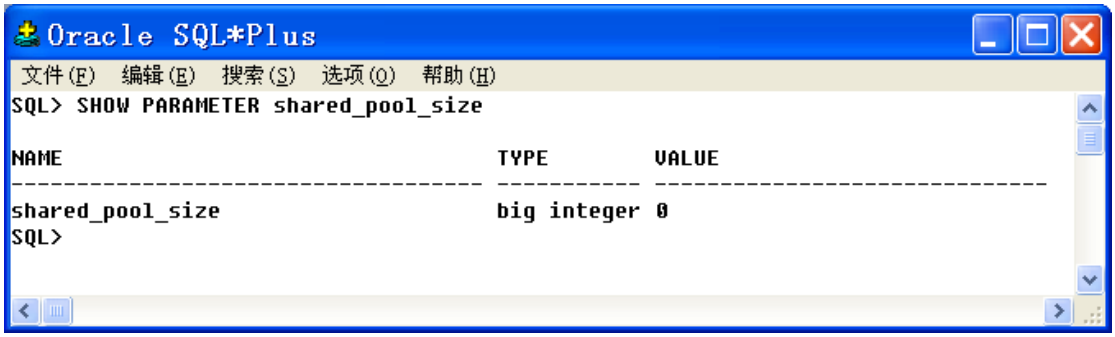
如果多次执行相同的 SQL 语句、PL/SQL 程序代码，都要进行这一系列操作的话，就会浪费系统资源。库缓存的目的就是用于保存最近解析过的 SQL 语句和 PL/SQL 程序。这样，Oracle 在执行一条 SQL 语句、一段 PL/SQL 程序之前，首先在库缓存中进行搜索，查看它们是否已经被解析过。如果有，Oracle 就利用库缓存中的解析结果和执行计划来执行，而不必在重复对它们进行解析了。这样就会明显地提高执行速度。

共享池的大小由 SHARED_POOL_SIZE 初始化参数指定。可以在数据库运行期

间修改该参数。共享池小，则运行 SQL 语句、PL/SQL 程序所占用的时间会长，而影响数据库的性能。

例：使用 SHOW 命令，可以查询重做日志缓存的大小。

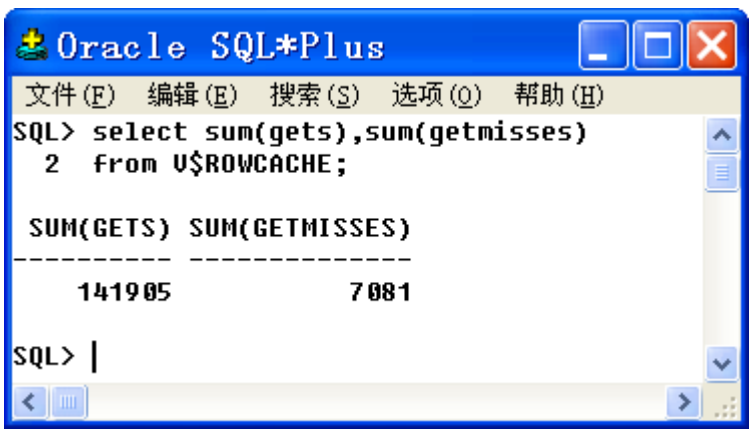
```
SQL> SHOW PARAMETER shared_pool_size;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------
shared_pool_size                     big integer 0
```



例：通过数据字典 V$ROWCACHE，可以查询共享池中数据字典缓存的成功与失败的次数。

```
SQL> select sum(gets),sum(getmisses) from V$ROWCACHE;

 SUM(GETS) SUM(GETMISSES)
---------- --------------
    141905           7081
```

其中 gets 表示读取某一类数据字典时成功的次数，getmisses 表示读取某一类数据字典时失败的次数。



四、大池

大池用于为需要大内存的操作提供相对独立的内存空间，以便提高这些操作的性能。大池是一个可选的内存结构。DBA 可以根据实际需要来决定是否在 SGA 区中创建大池。

需要大量内存的操作包括：

*、数据库备份和恢复，如使用 RMAN 在磁带设备上执行备份、转储、恢复等操作。

*、具有大量排序操作的 SQL 语句。

*、并行化的数据库操作。

注意：如果没有在 SGA 区中创建大池，上述操作所需的内存空间将占据共享池的内存。由于这些操作所占据的内存比较多，会导致影响到共享池的使用效率。这时候，就应该考虑在 SGA 区中创建大池，在大池中为这些操作分配内存。

大池的大小由初始化参数 LARGE_POOL_SIZE 确定。在 Oracle 9i 之前，为了改变大池的大小，需要修改参数文件，然后重新启动数据库。Oracle 9i 之后，DBA 可以使用 ALTER SYSTEM 语句来动态改变大池的大小，如：

```
ALTER SYSTEM SET LARGE_POOL_SIZE = 20M;
```

五、Java 池

在 Oracle 8i 之后，Oracle 增加了对 Java 语言的支持，所以提供了 Java 池，用于存放 Java 代码、Java 语句的语法分析表、Java 语句的执行方案和进行 Java 程序开发。

Java 池的大小由初始化参数 JAVA_POOL_SIZE 确定，一般不小于 20MB，以便安装 Java 虚拟机。

8.3.2.2、PGA（Program Global Area,程序全局区）

PGA 区是在用户进程连接到数据库，并创建一个对应的会话时，由 Oracle 为服务进程分配的，专门用于当前用户会话的内存区。这个内存区时非共享的，只有服务进程本身才能访问它自己的 PGA 区，而 SGA 区则是所有服务进程都可以共享的内存区。

PGA 区的大小由操作系统决定，并且分配后保持不变。当会话终止时，Oracle

会自动释放 PGA 区所占用的内存区。

按照存放信息的类型不同，PGA 区可以分为如下几个部分。

一、排序区

排序区用于存放排序操作所产生的临时数据，它是影响 PGA 区大小的主要因素，其大小由初始化参数 SORT_AREA_SIZE 定义。

在执行包括 order by 或 group by 等包含排序操作的 SQL 语句时，用户处理的数据都要按照某种属性进行排序。为了提高数据的访问和排序的性能，Oracle 利用内存比磁盘要快得多的事实，将准备排序的数据先临时存储到排序区中，并在排序区中进行排序，然后排序后的数据返回给用户。

二、会话区

保存会话所具有的权限、角色、性能统计信息。

三、游标区

当运行使用游标(cursors)的语句时，如 PL/SQL 或 OCI 程序中的语句，Oracle 会在共享池中为该语句分配上下文区，游标实际上是指向该上下文区的指针。

游标区在打开游标时创建，关闭游标时释放。因此在编写使用游标的程序时，应尽量避免反复地打开和关闭游标。

通过设置初始化参数 OPEN_CURSORS，可以限制用户能够同时打开的游标数目。

四、堆栈区

保存会话中的绑定变量（bind variable）、会话变量（session variable）以及 SQL 语句运行时的内存结构等信息。

例如，当运行 select * from emp where empno=:a;语句时，a 是绑定变量，提示用户输入，然后该信息将被保存在堆栈区中，以便在同一个会话中运行其他语句，例如：select * from emp where empno=:a and deptno=:b;此时可以再次使用该绑定变量。

8.3.3、进程结构

进程是操作系统中的一个概念，是一个可以独立调度的活动，用于完成指定的任务。进程与程序的区别是：

*、进程是动态的概念，即动态地创建，完成任务后立即消亡；程序是静态的实体，可以复制、编辑。

*、进程强调执行过程，程序仅仅是指令的有序集合。

*、进程在内存中，程序在外存中。

在 Oracle 中由两类进程，分别是用户进程和 Oracle 进程。

8.3.3.1、用户进程

当用户执行一个基于 Oracle 数据库的应用程序时，就会创建一个客户端的 Oracle 用户进程，以便来执行相应的任务。

与用户进程相关的两个概念是：连接和会话。

*、连接：连接是用户进程与数据库实例之间的路径。这个路径是通过硬件线路、操作系统的进程间通信机制（IPC）和各种网络协议实现的。

*、会话：会话是用户与数据库之间的路径。当用户启动一个基于 Oracle 数据库的应用程序，输入了正确的用户名、口令而登录到数据库之后，Oracle 就为该用户创建了一个会话。该会话在用户使用数据库期间一直存在，直到该用户退出数据库为止。

会话是通过连接来实现的。一个用户可以启动多个应用程序，即一个用户可以有多个会话。

8.3.3.2、Oracle 进程

Oracle 进程包括服务进程和后台进程。

一、服务进程

服务进程是为了给客户端的用户进程提供服务，Oracle 会在服务器端创建相应的服务进程。

用户进程必须通过服务进程才能访问数据库。服务进程分为专用服务进程（只为一个用户进程提供服务）和共享服务进程（为多个用户进程提供服务）。

服务进程主要完成如下任务：

*、解析并执行用户所提交的 SQL 语句。

*、搜索 SGA 区的数据库缓存，决定是否读取数据文件。如果数据块不在 SGA

区的数据库缓存中，则将其读入。

*、将查询或执行后形成的数据返回给用户。

二、后台进程

服务进程是由后台进程提供支持的。

在同一时刻，Oracle 可以处理上百个并发的请求，进行复杂的数据操作。Oracle 将管理和操作数据库这样一个复杂的大任务进行了划分，分别由几个相互独立的、各司其职的后台进程来完成。它们分工合作，共同完成这个大任务。

后台进程主要完成如下任务：

*、在内存和外存之间进行 I/O 操作。

*、监视各个进程的状态。

*、协调各个进程的任务。

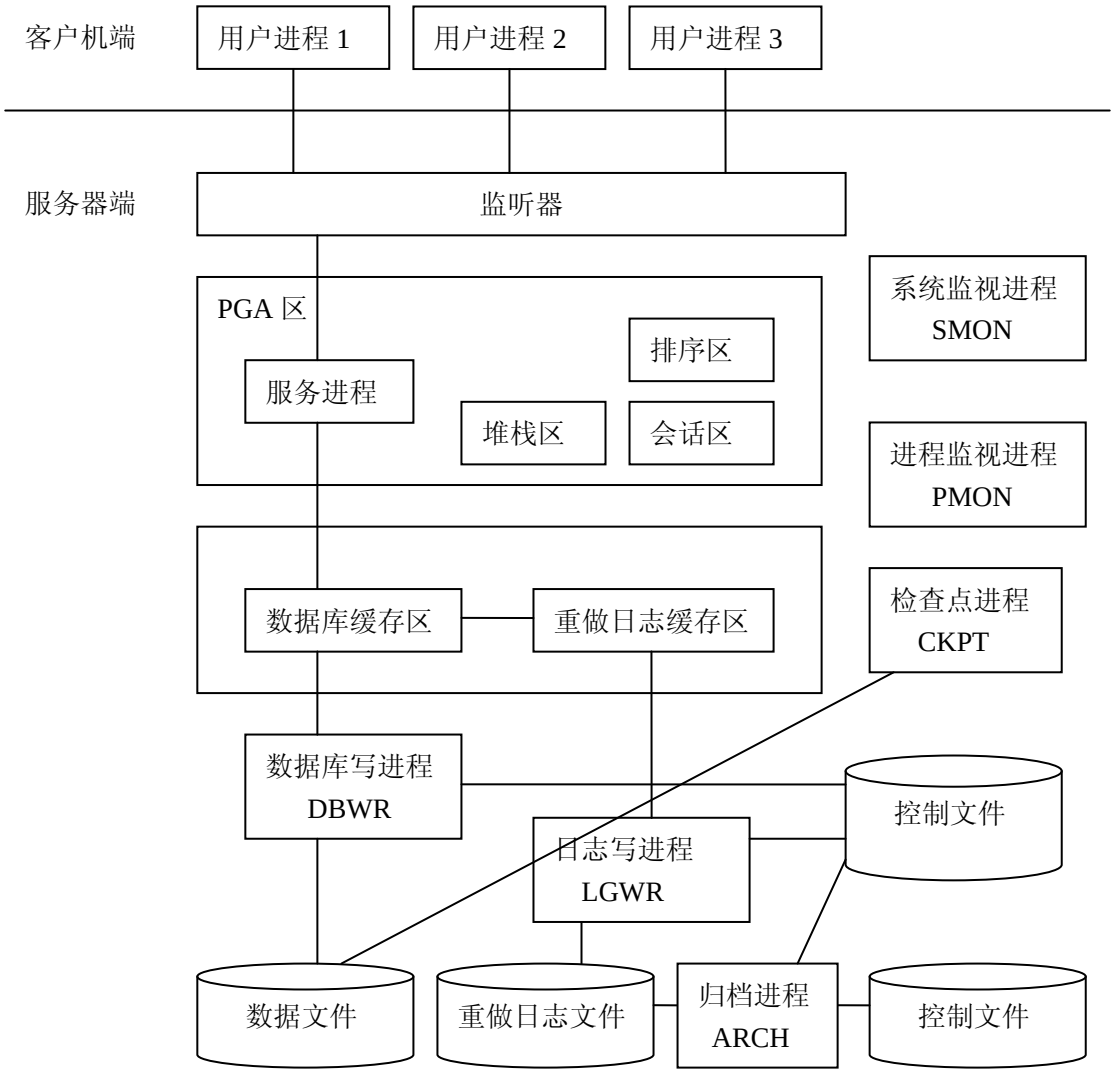
*、维护系统的性能。

*、保证系统的可靠性。

主要的后台进程有如下几个，其中前面 5 个后台进程是必需的，在默认情况下创建例程时只会启动这 5 个后台进程。另外几个时在分布式环境、多线程环境中使用的。

DBWR（数据库写进程）	LGWR（日志写进程）
CKPT（检查点进程）	SMON（系统监视进程）
PMON（进程监视进程）	ARCH（归档进程）
RECO（恢复进程）	LCKn（锁进程）
Dnnn（调度进程）	SNP（作业进程）

用户进程、Oracle 进程、物理存储文件之间的关系如图：



8.3.4、后台进程

在 Oracle 数据库中，为了使系统性能最优，并支持协调服务进程，提供了后台进程（background process）。这些后台进程存在于操作系统中，在启动例程时自动启动。只要数据库还在运行，后台进程就一直存在。常用的后台进程主要包括 DBWR、LGWR、CKPT、SMON、PMON、ARCH、RECO、LCKn、Dnnn、SNP 等。

每个后台进程在数据库运行中执行不同的任务。

数据库启动后，可以查询数据字典视图 V$BGPROCESS 来检查数据库中启动的后台进程个数及名称。

8.3.4.1、DBWR（database writer，数据库写进程）

DBWR 进程的作用是：

*、管理数据高速缓存区，以便服务进程总能找到空闲缓存块，用于保存从数据文件中读取的数据块。

*、在满足一定条件时，将 DIRTY 列表中的最近未被访问的脏缓存块成批地写入数据文件，以便获得更多的空闲缓存块。

*、使用 LRU 算法将最近正在使用的缓存块（命中缓存块）继续保留在 LRU 列表中，以免重新读取数据文件才能获取这些缓存块中的数据。

*、通过延迟写来优化磁盘读写操作。

启动 DBWR 进程的条件是：

*、当 DIRTY 列表中的脏缓存块达到一定数量（由初始化参数 DB_BLOCK_WRITE_BATCH 指定）。

*、当服务进程在 LRU 列表中查找了一定数量（由初始化参数 DB_BLOCK_MAX_SCAN 指定）的缓存块，但还没有查找到空闲缓冲块。

*、DBWR 进程出现超时（TIME_OUT），即大约 3 秒未启动 DBWR 进程。

*、当出现检查点，LGWR 进程通知 DBWR 进程写操作。

DBWR 进程启动的时间与用户提交事务的时间完全无关。也就是说，在用户执行 COMMIT 语句提交事务之后，与该事务相关的脏缓存块并不是立即被写入数据文件中。

注意：DBWR 进程的行为对整个 Oracle 系统的性能有很大影响。如果 DBWR 进程过于频繁地启动，将降低整个系统的 I/O 性能。但是如果 DBWR 进程间隔很久才启动一次，也会在数据库恢复方面带来不利影响。

Oracle 在创建例程时启动一个 DBWR 进程。如果数据库中的数据更改操作十分频繁，DBA 可以启动更多的 DBWR 进程提高写入能力。Oracle 最多允许启动 20 个 DBWR 进程(DBW0～DBW20)。但是 DBWR 进程的数目不应当超过系统 CPU 的数目，因为每个处理器同时只能运行一个 DBWR 进程。

8.3.4.2、LGWR（log writer，日志写进程）

LGWR 进程是负责管理重做日志高速缓存区的一个后台进程，用于将重做记录

从重做日志高速缓冲区写入重做日志文件。每个例程只有一个 LGWR 进程。

数据库在运行时，如果对数据进行了修改，则产生重做记录。重做记录首先被保存在重做高速日志缓存区中。当满足一定条件时，LGWR 进程就将重做记录写入到重做日志文件。当发生重做日志切换时，ARCH 进程再将重做日志文件中的数据写入到归档日志文件中。

重做日志高速缓存是一个循环结构，再 LGWR 进程将缓存中的数据写入重做日志文件的同时，服务进程还能够继续向缓存中写入新的数据。LGWR 进程将缓存中的数据写入重做日志文件之后，相应的缓存内容将被清空。LGWR 进程必须即使完成重做日志文件的写入，以便能够保证重做日志高速缓存中始终有足够的空闲空间来存储重做记录。

启动 LGWR 进程的条件是：

*、用户通过 COMMIT 语句提交当前事务。

*、重做日志高速缓存被写满三分之一。

*、DBWR 进程需要为检查点（checkpoint）清除脏缓存块，即将脏缓存块写入数据文件。

*、LGWR 进程出现超时（TIME_OUT），即大约 3 秒未启动 LGWR 进程。

假设一个事务在执行过程中对数据库进行了修改，它将在数据高速缓存中产生脏缓存块，同时还会在重做日志高速缓存中产生重做记录。在 DBWR 进程将该事务所产生的脏缓存块写入数据文件中之前，该事务所产生的重做记录必须已经被 LGWR 进程写入重做日志文件。这样才能够避免产生损害数据库完整性的情况。比如，数据库突然崩溃，或事务的修改已经被部分记录在数据文件中，但是由于没有该事务的重做记录，就无法完全恢复或完全回退该事务。如果 DBWR 进程发现与要写入的脏缓存块相关的重做记录仍然处于重做日志高速缓存中，它将通知服务进程启动 LGWR 进程，以便将这些重做记录写入重做日志文件。直到重做记录全部被写入后，DBWR 进程才开始将脏缓存块写入数据文件。只要重做记录被完全写入重做日志文件中了，即便数据库突然崩溃了，事务对数据库所做的修改也不会丢失，可以通过重做日志文件进行自动恢复。因此 LGWR 进程要先于 DBWR 进程启动。

LGWR 进程的行为比 DBWR 进程简单，因此与 LGWR 进程本身相关的初始化参数

比较少，但是 LGWR 进程除了要将重做日志高速缓存中的内容写入重做日志文件外，还有另外一个任务，就是如果例程没有启动 CKPT 进程，则由 LGWR 进程来完成检查点执行任务。在这种情况下，需要理解一些与检查点相关的初始化参数。

检查点导致 LGWR 进程和 DBWR 进程都要花 I/O 时间。检查点间隔时间越短，发生数据库故障时需要的恢复时间就越短，同时减少了必须执行每一个检查点所需的工作。当用户决定正确的检查点间隔时，必须权衡所有这些因素。

LOG_CHECKPOINT_INTERVAL 和 LOG_CHECKPOINT_TIMEOUT 是有关检查点间隔的两个参数，也是触发数据库检查点所必需的时间或条件。
LOG_CHECKPOINT_INTERVAL 指定当一定数量的操作系统数据块（不是 Oracle 数据块）被写入重做日志文件时，将触发一个检查点。LOG_CHECKPOINT_TIMEOUT 指定检查点之间的时间间隔（以秒为单位）。

注意：必须小心使用 LOG_CHECKPOINT_INTERVAL 和 LOG_CHECKPOINT_TIMEOUT 两个参数。

因为一个重做日志文件写满时，将触发一个检查点，所以应该使 LOG_CHECKPOINT_INTERVAL 设置的操作系统块数与重做日志文件的大小相匹配。

注意：不要设置不必要的检查点，或迫使不需要的检查点发生。例如，如果一个重做日志文件大小为 5MB，而 LOG_CHECKPOINT_INTERVAL 设置为 1152MB，那么当有 4.5MB（假设操作系统数据块的大小是 4KB，1152*4KB＝4.5MB）的数据写入重做日志文件时，将导致发生一个检查点。然后，当重做日志文件写满时（仅在又写入 0.5MB 的数据后），又发生一个检查点。事实上，这两个检查点将相继发生。

用户还可以通过相应地设置重做日志文件的大小来控制检查点发生的频繁程度。如果设置重做日志文件大小使得每小时发生一次日志切换，那么重做日志文件切换后在一小时之内，用户将只有一个检查点。但是，如果重做日志文件大小设置为每 5 分钟发生一个检查点，就将浪费大量的进程活动和 I/O 次数以执行相关的检查点。

8.3.4.3、CKPT（checkpoint，检查点进程）

检查点时一个事件。当该事件发生时，数据高速缓存中的脏缓存块将被写入

数据文件，同时 Oracle 将对控制文件和数据文件的文件头的同步序号进行修改，记录下当前数据库的结构和状态，以保证数据的同步。通常情况下，检查点发生在重做日志文件切换时。

在执行了一个检查点后，Oracle 知道所有已提交事务对数据库所做的更改已经全部被写入到硬盘中了。此时数据库处于一个完整状态。在发生数据库崩溃后，只需要将数据库恢复到上一个检查点执行时刻即可。因此，缩短检查点执行的间隔，可以缩短数据库恢复时所需要的时间。

注意：DBA 需要根据实际应用的情况，为检查点选择一个合适的执行间隔。如果检查点执行间隔太短，将会产生过多的硬盘 I/O 操作；如果检查点执行间隔太长，数据库的恢复将耗费太多时间。

CKPT 进程的作用就是执行检查点：

*、更新控制文件与数据文件，使其同步。

*、触发 DBWR 进程，使其将脏缓存块写入数据文件。

Oracle 在不同的时刻执行不同级别的检查点，Oracle 有如下三种检查点：

*、数据库检查点：在每一次重做日志文件切换时，执行数据库检查点。此时 DBWR 进程将数据高速缓存中所有的脏缓存块写入数据文件中。

*、表空间检查点：在将一个表空间设置为脱机状态时，执行一个表空间检查点。此时 DBWR 进程只会把数据高速缓存中的与该表空间相关的脏缓存块写入数据文件。

*、时间检查点：即每间隔多长时间执行一次检查点。

与检查点相关的初始化参数如下：

*、LOG_CHECKPOINT_TIMEOUT：用于指定检查点执行的最大时间间隔（以秒为单位）。比如将该参数设置为 1800，那么至少每隔 30 分钟执行一次检查点。如果将该参数设置为 0，将禁用基于时间的检查点。

*、LOG_CHECKPOINT_INTERVAL：用于指定在出现检查点之前，必须写入重做日志文件中的操作系统块（而不是数据库块）的数量。无论该参数设置为什么值，在切换重做日志文件时都会出现检查点。

*、LOG_CHECKPOINT_TO_ALERT：用于设置是否将检查点信息记录到警告日志文件中。通过将检查点信息记录在警告日志文件中。DBA 可以确定检查点

是否按所需频率出现。

通常 DBA 只能通过调整 LOG_CHECKPOINT_TIMEOUT 和 LOG_CHECKPOINT_INTERVAL 参数来改变检查点执行的间隔。在调整检查点执行间隔时要小心，不要让检查点间隔时间过长，但是也不要添加不必要的检查点。

8.3.4.4、SMON（system monitor，系统监视进程）

SMON 进程的作用是：

*、在例程启动时负责对数据库进行恢复。

*、清理不再使用的临时段

*、将各个表空间的空闲空间碎片合并在一起，使之更容易分配。

如果上一次数据库时非正常关闭的，则当下一次启动例程时，SMON 进程会自动读取重做日志文件，对数据库进行恢复，即执行将已提交的事务写入数据文件、回退未提交的事务等操作。

SMON 进程除了在例程启动时执行一次外，在例程运行期间，也会被定期地唤醒，以便检查是否有工作需要它来做。如果其他任何进程需要使用到 SMON 进程的功能，它们将随时唤醒 SMON 进程。

SMON 进程担任一个比较小但又非常重要的角色。

注意：如果某个表空间的存储参数 PCTINCREASE 被设置为 0，SMON 进程就不会对该表空间中的空闲碎片进行合并操作。因此，如果需要让 SMON 进程自动合并空闲碎片，就不能将 PCTINCREASE 存储参数设置为 0。

8.3.4.5、PMON（process monitor，进程监视进程）

PMON 进程的作用是：

*、恢复中断或失败的用户进程、服务进程。

*、清除非正常中断的进程流下来的孤儿会话。

*、回退未提交事务。

*、释放进程所占用的各种资源

*、监视服务进程和调度进程，如果它们失败，则自动重新启动它们。

由于种种原因，对 Oracle 数据库的连接可能会发生崩溃、挂起或其他非正

常终止现象。常见的情况是用户可能关闭他们的客户端程序，但是却没有从数据库中退出，或者是由于网络的突然中断而造成一个数据库连接的异常中止。在这些情况下，Oracle 将通过 PMON 进程来启动、清除中断或失败的用户进程，包括清除非正常中断的用户进程留下的孤儿会话，回退未提交的事务，释放会话所占用的锁、SGA 区、PGA 区等资源。

此外，PMON 进程还会定期地检查调度程序和服务进程的状态，如果它们失败，就会尝试重新启动它们，并释放它们所占用的各种资源。

与 SMON 进程类似，PMON 进程在例程运行期间会被定期地唤醒，检查是否有工作需要它来做。如果任何其他进程需要使用到 PMON 进程的功能，它们将随时唤醒 PMON 进程。

8.3.4.6、ARCH（archive，归档进程）

归档进程 ARCH 负责在重做日志文件切换后将已经写满的重做日志文件复制到归档日志文件中，以防止循环写入重做日志文件时将其覆盖。

注意：只有数据库运行在归档模式（ACHIVELOG）时，ARCH 进程才能被启动。

要启动 ARCH 进程，需要将初始化参数 ARCHIVE_LOG_START 设置为 TRUE。ARCH 进程启动后，数据库将具有自动归档功能。但即使数据库运行在归档模式下，如果 ARCHIVE_LOG_START 参数设置为 FALSE，ARCH 进程也不会被启动。这时，当重做日志文件全部被写满后，数据库将被挂起，等待 DBA 进行手工归档。

默认情况下，一个例程只会启动一个归档进程 ARCH。当 ARCH 进程正在归档一个重做日志文件时，任何其他进程都不能访问这个重做日志文件。

因为重做日志文件是被循环使用的，所以，如果 LGWR 进程要使用的下一个重做日志文件正在被归档，则数据库将被挂起，直到该重做日志文件归档完毕为止。因此，为了加快重做日志文件的归档速度，避免发生等待，LGWR 进程会根据需要自动启动更多的归档进程。Oracle 最多可以启动 10 个归档进程（ARC0～ARC9）。

如果 DBA 能够预测到归档任务将十分繁重，可以将初始化参数 LOG_ARCHIVE_MAX_PROCESSES 设置为大于 1 的数值。这样在创建例程时就会启动多个归档进程。通常 DBA 并不需要设置 LOG_ARCHIVE_MAX_PROCESSES 参数，LGWR

进程会根据归档任务的需要自动启动适当数量的归档进程。

8．4、数据字典

数据字典是 Oracle 数据库的最重要的组成部分。它提供了数据库的系统信息，以及例程的性能信息。

8.4.1、数据字典的概念

数据字典由一系列只读的数据字典表和数据字典视图组成。数据字典表中记录了数据库的系统信息（如方案对象的信息）、例程运行的性能信息（如例程的状态、SGA 区的信息）。数据字典表的所有者为 SYS 用户，其数据字典表和数据字典视图都被保存在 SYSTEM 表空间中。所以，为了性能和安全的原因，Oracle 建议不要在 SYSTEM 表空间中创建其他方案对象。

数据字典表主要保存有如下信息：

*、各种方案对象的定义信息，如表、视图、索引、同义词、绪论、存储过程、函数、包、触发器和各种对象。

*、存储空间的分配信息，如为某个对象分配了多少存储空间，该对象使用了多少存储空间。

*、安全信息，如账户、权限、角色、完整性约束信息。

*、例程运行时的性能和统计信息

*、其他数据库本身的基本信息。

数据字典的主要用途是：

*、Oracle 通过查询数据字典表或数据字典视图来获取有关用户、方案对象、对象的定义信息，以及其他存储结构的信息，以便确认权限、方案对象的存在性和正确性。

*、在每次执行 DDL 语句修改方案对象和对象后，Oracle 都在数据字典中记录下所做的修改。

*、用户可以从数据字典的只读视图中，获取各种与方案对象和对象有关的信息。

*、DBA 可以从数据字典的动态性能视图中，监视例程的运行状态，为性能调

整提供依据。

8.4.2、数据字典的组成

为了方便使用，数据字典中的信息通过表和视图的方式组织。数据字典的组成包括数据字典表和数据字典视图。

8.4.2.1、数据字典表

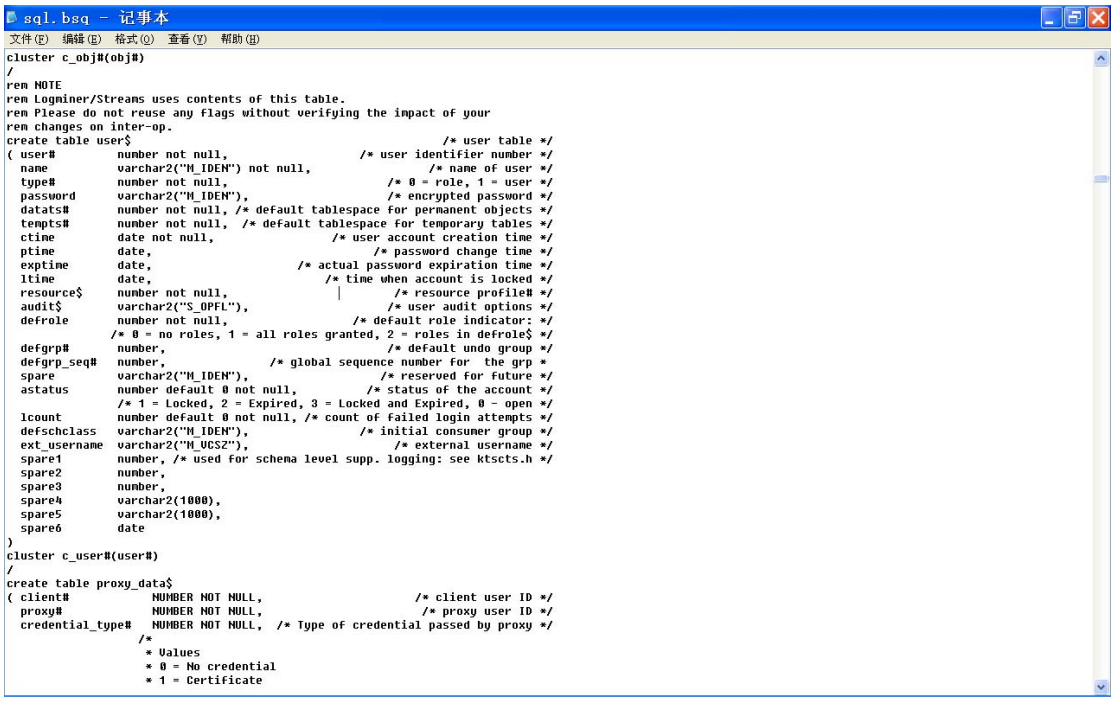
数据字典中的所有信息实际上都是保存在数据字典表中的。数据字典表中存储的信息通常都是经过加密处理的。

数据字典表属于 SYS 用户，通过在创建数据库时自动运行 SQL.BSQ 脚本来创建数据字典表。大部分数据字典表的名称中都包含$等这样的特殊符号。

SQL.BSQ 脚本可以在如下目录找到：

D:\oracle\product\10.2.0\db_1\RDBMS\ADMIN\sql.bsq

SQL.BSQ 脚本中的部分内容如下。它们创建用于管理用户、管理序列的数据字典表。




```
cluster c_obj#(obj#)
/
rem NOTE
rem Logminer/Streams uses contents of this table.
rem Please do not reuse any flags without verifying the impact of your
rem changes on inter-op.
create table user$                                          /* user table */
( user#         number not null,                  /* user identifier number */
  name          varchar2("M_IDEN") not null,              /* name of user */
  type#         number not null,                    /* 0 = role, 1 = user */
  password      varchar2("M_IDEN"),                 /* encrypted password */
  datats#       number not null, /* default tablespace for permanent objects */
  tempts#       number not null,  /* default tablespace for temporary tables */
  ctime         date not null,                /* user account creation time */
  ptime         date,                              /* password change time */
  exptime       date,                    /* actual password expiration time */
  ltime         date,                       /* time when account is locked */
  resource$     number not null,                    /* resource profile# */
  audit$        varchar2("S_OPFL"),                 /* user audit options */
  defrole       number not null,                 /* default role indicator: */
                /* 0 = no roles, 1 = all roles granted, 2 = roles in defrole$ */
  defgrp#       number,                             /* default undo group */
  defgrp_seq#   number,               /* global sequence number for  the grp *
  spare         varchar2("M_IDEN"),               /* reserved for future */
  astatus       number default 0 not null,          /* status of the account */
                /* 1 = Locked, 2 = Expired, 3 = Locked and Expired, 0 - open */
  lcount        number default 0 not null, /* count of failed login attempts */
  defschclass   varchar2("M_IDEN"),              /* initial consumer group */
  ext_username  varchar2("M_VCSZ"),                 /* external username */
  spare1        number, /* used for schema level supp. logging: see ktscts.h */
  spare2        number,
  spare3        number,
  spare4        varchar2(1000),
  spare5        varchar2(1000),
  spare6        date
)
cluster c_user#(user#)
/
create table proxy_data$
( client#           NUMBER NOT NULL,                    /* client user ID */
  proxy#            NUMBER NOT NULL,                     /* proxy user ID */
  credential_type#  NUMBER NOT NULL,  /* Type of credential passed by proxy */
                 /*
                  * Values
                  * 0 = No credential
                  * 1 = Certificate
```

注意：只有 Oracle 才负责对数据字典表进行管理和维护，任何用户（包括 DBA），都不能对数据字典中的内容进行修改。用户只能在数据字典上执行查询（SELECT）操作。

当执行 CREATE 操作（如 CREATE TABLE）时，Oracle 会在数据字典表中隐含地执行 INSERT 操作；当执行 ALTER 操作（如 ALTER TABLE）时，Oracle 会在数据字典表中执行 UPDATE 操作；当执行 DROP 操作（如 DROP TABLE）时，Oracle 会在数据字典表中执行 DELETE 操作。

8.4.2.2、数据字典视图

数据字典表中的信息经过解密和其他一些加工处理后，以数据字典视图的方式显示给用户。这些数据字典视图将各种信息分权限、分类存放，十分便于用户使用。大多数用户都可以通过数据字典视图查询到所需要的与数据库相关的系统信息。

数据字典视图是公用同义词的良好示例。在创建数据库时，通过自动运行 catalog.sql 脚本来创建数据字典视图后，创建公用同义词和授权。

catalog.sql 脚本可以在如下目录找到：

D:\oracle\product\10.2.0\db_1\RDBMS\ADMIN\catalog.sql

当用户编写引用数据字典视图 USER_DB_LINKS 的 SQL 代码时，不需要写成 SYS.USER_DB_LINKS，而是可以简单地写成 USER_DB_LINKS。此时实际上引用的是公用同义词 USER_DB_LINKS，它接着解析为 SYS.USER_DB_LINKS。

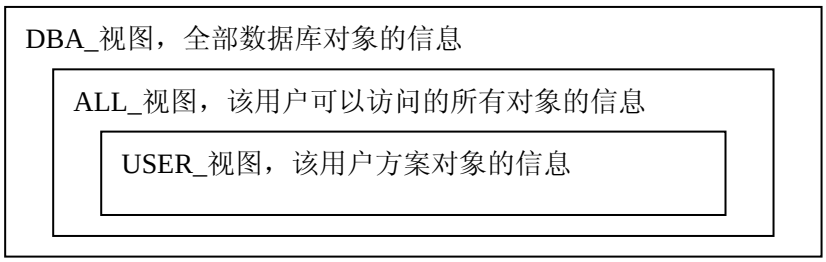
8.4.3、数据字典的使用

对于数据库用户来说，数据字典中的数据字典视图好比一本当前 Oracle 数据库的参考手册，可以通过 SELECT 语句来查询这些数据字典视图，以得到该数据库的有关信息。

数据字典视图分为三类，如下图所示，每类的名称都有不同的前缀。按照信息的划分，不同权限的用户只能查询不同的数据字典视图。

名称前缀	用途
USER_	用户视图
ALL_	扩展用户视图
DBA_	数据库管理员视图

数据字典视图之间的关系如下图：



注意：只有数据库处于 OPEN 状态时，才能访问数据字典视图。

8.4.3.1、USER 视图

USER 视图中包含了该用户方案对象的信息，USER 视图可以被看做 ALL 视图的子集，每个用户都可以查询 USER 视图。

例：一个用户可以通过如下语句来查询它所创建的所有方案对象的信息。

```
SQL> select object_name,object_type from USER_OBJECTS;
```

8.4.3.2、ALL 视图

ALL 视图是 USER 视图的扩展。在 ALL 视图中包含了该用户可以访问的所有对象的信息，包括该用户自己的方案对象，也包括被授权可以访问的其他用户的方案对象。

例：用户可以通过如下语句来查询他所能访问的用户 jack 的方案对象的信息。

```
SQL> select owner,object_name,object_type from ALL_OBJECTS where owner='jack';
```

8.4.3.2、DBA 视图

DBA 视图包含了全部数据库对象的信息。例如，当 SYSTEM 用户查询数据字典视图 DBA_TABLES 时，不仅会返回 SYSTEM 用户的所有表，还会返回 SYS 用户、SCOTT 用户等其他用户的表。因此，应该只有 DBA 角色的用户才能访问 DBA 视图。被授予 SELECT ANY DICTIONARY 系统权限的用户也能够访问 DBA 视图。

例：DBA 视图没有同义词（SYNONYMS）。因此，在使用 DBA 视图时，如果不是 SYS 用户，则应该在 DBA 视图前加上 SYS 前缀。

SQL> select owner,object_name,object_type from sys.dba_objects where owner='jack';

8.4.4、动态性能表和动态性能视图

在例程的运行过程中，Oracle 会在数据字典中维护一系列虚拟的表，在其中记录与数据库活动相关的性能统计信息，这些表被称为动态性能表。

这些虚拟的表不是固定的表。即在例程启动时被创建，并将向其中添加信息，而当例程消亡时，这些表也就被删除了。

动态性能视图属于 SYS 用户。Oracle 自动在动态性能表上创建了一些视图，即动态性能视图（目前有 340 多个）。所有动态性能视图都以 V_$开头。Oracle 为这些视图创建了公用同义词。这些同义词都以 V_$开头，因此动态性能视图也被称为“V$视图”。

例如，V$DATAFILE 视图包含了有关数据文件的统计信息；V$FIXED_TABLE 视图包含了有关所有动态性能表和动态性能视图的信息；V$SYSSTAT 视图包含了关于当前例程的性能统计信息。

注意：动态性能视图不能被大多数用户访问，具有 DBA 角色的用户可以访问动态性能视图。当数据处于不同的状态时，可以被访问的动态性能视图有所不同。

8.4.4.1、NOMOUNT 状态

启动例程后，Oracle 会打开参数文件，分配 SGA 区并启动各个后台进程。但当还没有加载数据库时，例程处于 NOMOUNT 状态时，只能访问从 SGA 区获得信息的动态性能视图。例如：

V$PARAMETER	V$SGA
V$OPTION	V$PROCESS
V$SESSION	V$VERSION
V$INSTANCE	

8.4.4.2、MOUNT 状态

当加载数据库时，Oracle 会根据参数文件中指定的控制文件打开控制文件，

使数据库处于 MOUNT 状态。此时，不仅可以访问从 SGA 区获得信息的动态性能视图，还可以访问从控制文件中获得信息的动态性能视图。例如：

V$THREAD	V$CONTROLEFILE
V$DATABASE	V$DATAFILE
V$DATAFILE_HEADER	V$LOGFILE

8.4.4.3、OPEN 状态

当打开数据库时，Oracle 会根据控制文件中的信息，打开所有数据文件和重做日志文件，使数据库处于 OPEN 状态。此时，不仅可以访问从 SGA 区获得信息的动态性能视图，还可以访问从控制文件中获得信息的动态性能视图，还可以访问与 Oracle 性能相关的动态性能视图。例如：

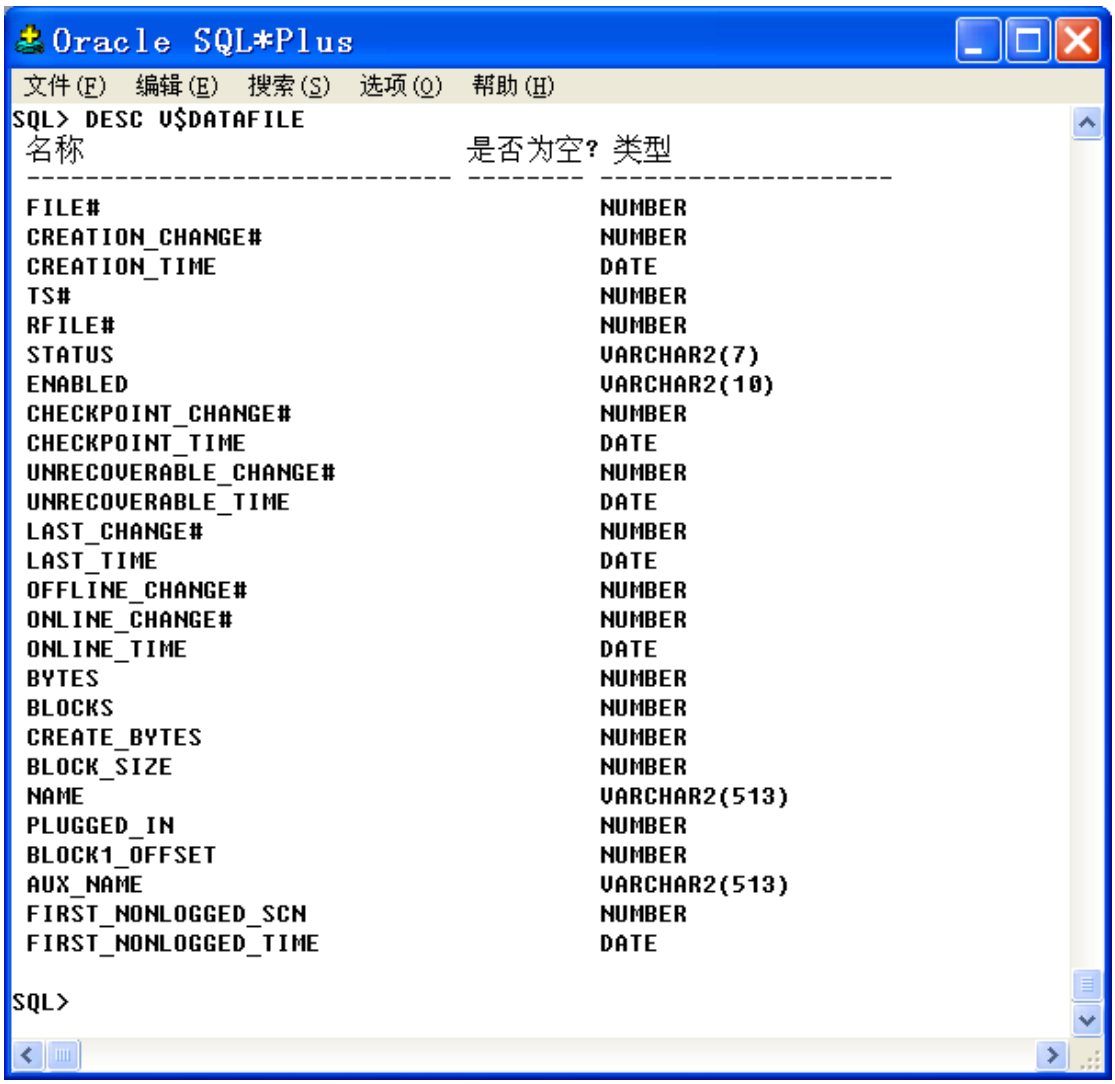
V$FILESTAT	V$SESSION_WAIT
V$WAITSTAT	

8.4.5、查询数据字典视图的信息

各类数据字典视图总共有几百甚至几千个，要记住它们的名称、结构、功能几乎是不可能的，但可以使用命令行方式。OEM 方式查询出这些信息。

例：使用查询语句查询 V$DATAFILE 视图结构，了解通过该视图可以查询到的信息。

```
SQL> DESC V$DATAFILE
```


Oracle SQL*Plus
文件(F) 编辑(E) 搜索(S) 选项(O) 帮助(H)
SQL> DESC V$DATAFILE
名称 是否为空? 类型
FILE# NUMBER
CREATION_CHANGE# NUMBER
CREATION_TIME DATE
TS# NUMBER
RFILE# NUMBER
STATUS VARCHAR2(7)
ENABLED VARCHAR2(10)
CHECKPOINT_CHANGE# NUMBER
CHECKPOINT_TIME DATE
UNRECOVERABLE_CHANGE# NUMBER
UNRECOVERABLE_TIME DATE
LAST_CHANGE# NUMBER
LAST_TIME DATE
OFFLINE_CHANGE# NUMBER
ONLINE_CHANGE# NUMBER
ONLINE_TIME DATE
BYTES NUMBER
BLOCKS NUMBER
CREATE_BYTES NUMBER
BLOCK_SIZE NUMBER
NAME VARCHAR2(513)
PLUGGED_IN NUMBER
BLOCK1_OFFSET NUMBER
AUX_NAME VARCHAR2(513)
FIRST_NONLOGGED_SCN NUMBER
FIRST_NONLOGGED_TIME DATE
SQL>

第 9 章　管理控制文件

控制文件包含有数据库的结构信息（即构成数据库的数据文件、重做日志文件）。主要包含如下几项内容：

- 控制文件所属的数据库名称，一个控制文件只能属于一个数据库。
- 相关的数据文件和重做日志文件的名称、位置、联机/脱机状态信息。
- 数据库创建的时间信息
- 当前重做日志的序号（log sequence number）。它是一个在重做日志文件切换时被递增和记录的唯一性标识号。
- 当前的检验点信息
- Recovery Manager（RMAN，恢复管理器）的备份信息。RMAN 是 DBA 用来备份恢复数据库的实用工具。

默认情况下，在创建数据库时至少要创建一个控制文本副本。如在 Windows 操作系统中，会创建 3 个副本。执行针对控制文件的管理工作，需要具有 ALTER DATABASE 系统权限。使用数据库的初始化参数 CONTROL_FILES 可以指定控制文件名。例程启动时将据此识别和打开所有列出的控制文件。在例程运行过程中将写入并维护这些控制文件。

9．1、管理控制文件的准则

9.1.1、多路复用控制文件

为了提高数据库的可靠性，需要备份控制文件。每个数据库最好应该至少有两个控制文件，并且应该分别存储在不同的磁盘上，进行多路复用（multiplexing）。初始化参数 CONTROL_FILES 列出所有多路复用的控制文件名。Oracle 会同时修改所有的控制文件，但只读取其中第一个控制文件中的信息。在整个数据库运行期间，如果任何一个控制文件变为不可用，那么例程就不能再继续运行，而且会中止这个例程。

例：下面以多路复用 OAMIS 数据库的控制文件为例，介绍使用 SPFILE 初始化参数文件进行多路复用控制文件的方法或步骤。

Step1、修改初始化参数 control_files。－因为 Oracle 是通过初始化参数 control_files 定位并打开控制文件的，所以，为了多路复用控制文件，必须利用 ALTER SYSTEM 语句修改该初始化参数，以便添加新的控制文件的名称。如以 SYS 用户身份登陆。执行如下语句：

其中，前面 3 个控制文件是原有的控制文件，而 e:\control04.ctl 是新添加的、放在不同磁盘上的控制文件，目前还不存在。

Step2、退出 SQL*PLUS。

Step3、关闭数据库。

Step4、停止所有与 Oracle 有关的服务。

Step5、复制现有的控制文件。为了确保所有控制文件互为镜像，完全相同，必须再关闭数据库的情况下，复制一个现有的控制文件，如 control01.ctl，形成新的控制文件 e:\control04.ctl。

Step6、启动数据库，启动所有服务。

9.1.2、管理控制文件的大小

控制文件的大小主要决定与在创建数据库时，CREATE DATABASE 指定的几个 MAX 子句的值，增大这些参数的值将增大相关数据库的控制文件的大小。如：

MAXDATAFILES：指定最大数据文件的个数。

MAXLOGFILES：指定最大重做日志文件的个数。

MAXLOGMEMBERS：指定重做日志文件中每个组的成员个数。

MAXLOGHISTROY：指定控制文件可记载的重做日志历史的最大个数。

MAXINSTANCES：指定可以同时访问数据库的最大例程的个数。

9．2、备份/恢复控制文件

为了提高数据库的可靠性，减少由于丢失控制文件（如磁盘损坏等原因）而造成的灾难性后果，DBA 除了要多路复用控制文件外，还应该定期备份控制文件。尤其是在改变了数据库的物理结构之后，需要重新备份控制文件。

备份了控制文件后，如果保存控制文件的磁盘坏了，只需要在初始化文件中将 control_files 参数设置成指向备份的控制文件，然后就可以重新启动数据库了；如果磁盘没有损坏，则只需要将备份的控制文件复制到被损坏的控制文件的位置上，并将文件名称改成原来的控制文件名称，然后就可以重新启动数据库了。

备份控制文件需要使用 ALTER DATABASE BACKUP CONTROLFILE 语句，有两种方法可以进行控制文件的备份操作。

9.2.1、将控制文件备份为二进制文件

将控制文件备份为二进制文件，实际上是在数据库运行期间原封不动地复制当前的控制文件。例如，使用如下语句将控制文件备份为二进制文件：

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO 'e:\control.bkp';
```

此时，将在 e 盘下产生一个备份的控制文件 control.bkp。

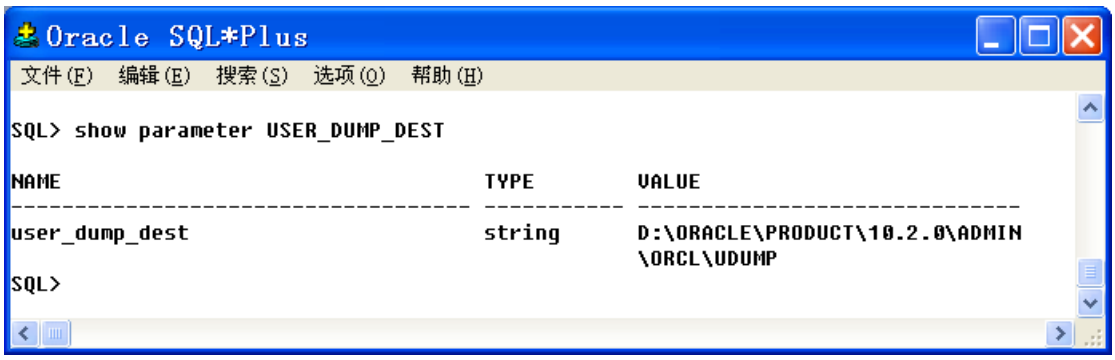
9.2.2、将控制文件备份为文本文件（又称为跟踪备份(backup to trace)）

例如，使用如下语句将控制文件备份为文本文件：

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

跟踪备份被存在由初始化参数 USER_DUMP_DEST 指定的目录中。

例：使用下面的语句实现跟踪备份被存放在由初始化参数 USER_DUMP_DEST 指定的目录中。



跟踪备份的文件名称的格式是 sid_ora_pid.trc。其中，sid 是创建该跟踪备份的用户的会话 ID；pid 是进程 ID。

如：D:\oracle\product\10.2.0\admin\orcl\udump\orcl_ora_3548.trc，可以在该目录中通过按修改日期的方式找到该文件。

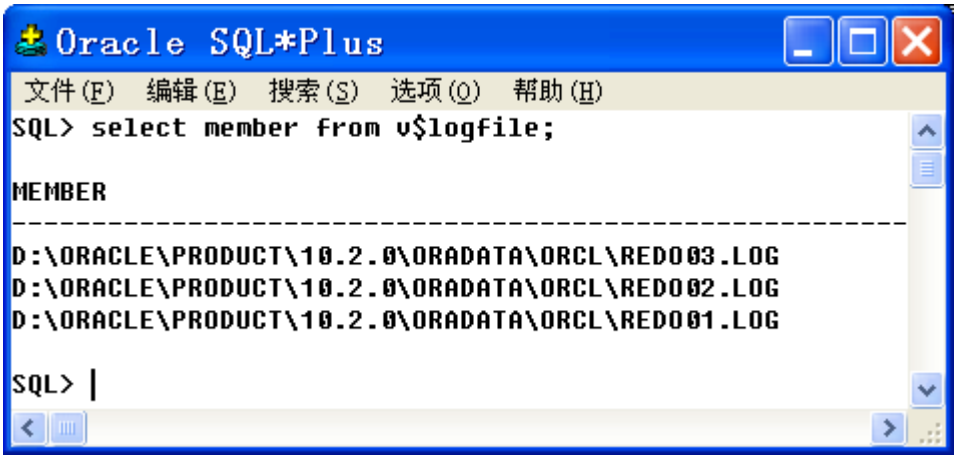
下面是这条命令输出的跟踪备份的内容。显然，该跟踪备份是一个 SQL 脚本，可以利用它来重建新控制文件。

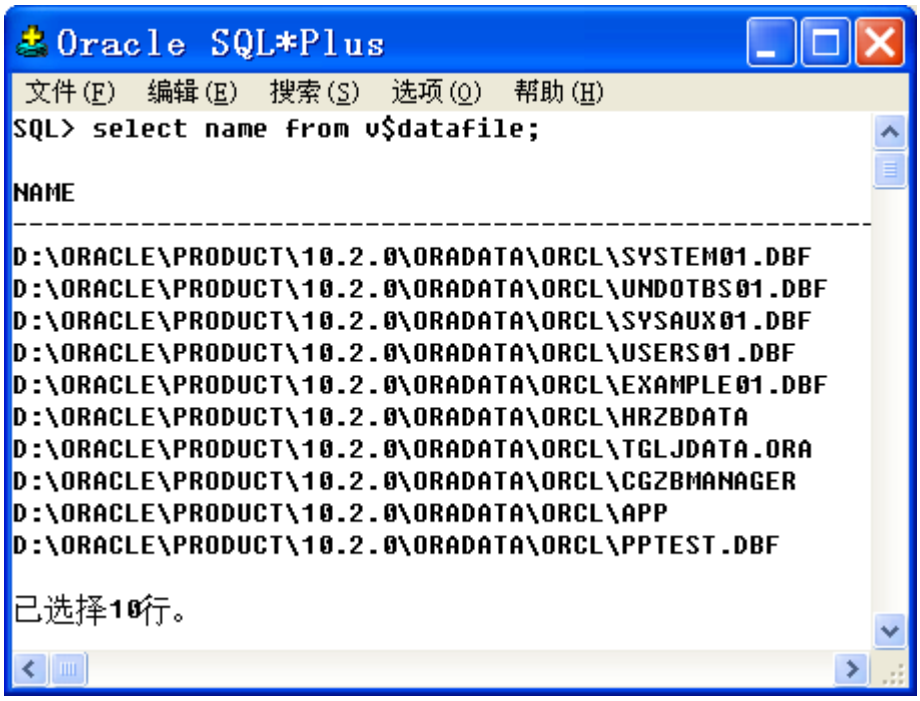
9．3、创建控制文件

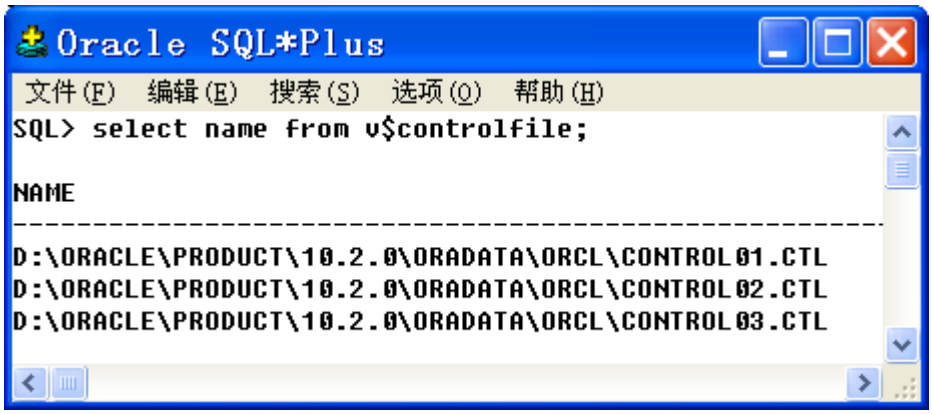
创建控制文件有两种方法，1、在创建数据库时创建初始控制文件，2、在创建数据库后创建新的控制文件。

第二种方法可以使用 CREATE CONTROLFILE 语句为数据库创建一个新的控制文件。步骤如下：

Step1、制作一个包含数据库的所有数据文件和重做日志文件的列表。执行如下语句根据产生的结果制定此列表。







Step2、关闭数据库：尽可能采用正常模式关闭数据。使用 IMMEDIATE 或 ABORT 选项作为最后的关闭手段：SQL> shutdown immediate

Step3、将数据库中的所有数据文件、重做日志文件、SPFILE 参数文件备份到其他地方。

Step4、启动一个新的例程，但是不要装载或打开数据库。如下：

SQL> STARTUP NOMOUNT

Step5、使用 CREATE CONTROLFILE 语句

Step6、在离线存储设备上存储新控制文件的备份。

Step7、编辑 CONTROL_FILES 初始化参数。

Step8、打开数据库

第 10 章　管理表空间

名词解释：

撤销（undo）表空间

临时（temporary）表空间

永久（permanent）表空间

10．1、表空间概述

表空间可分为如下几个类型：

1、系统表空间。

2、临时表空间。

3、撤销表空间。

4、大文件表空间与小文件表空间。

大文件表空间是 Oracle 10g 新引进的概念，小文件表空间即常规意义上创建的表空间。大文件表空间是为超大型数据库而设计的，只能放置一个数据文件（或临时文件）。必须在创建时指定 bigfile 关键字。

10.1.1、表空间的区、段管理方式

Oracle 10g 的表空间按照区和段进行管理，下面分别介绍区管理方式和段管理方式。

10.1.1.1、区管理方式

针对区的分配方式的不同，表空间有两种管理方式。

一、数据字典管理方式（dictionary-managed tablespace）

数据字典管理方式是传统的管理方式。在数据字典管理方式下，使用数据字典来管理存储空间的分配。

当在表空间中分配新的区，或回收已分配的区时，Oracle 将对数据字典中的相关基础表进行更新。由于对表的更新操作会产生回退信息和重做信息，因此，在分配区或回收区时，会在数据库中产生回退信息和重做信息。这是数据字典管理方式的特点，也是它的缺点。

在创建数据字典管理方式的表空间时，可以在 DEFAULT STORAGE 子句中设置 INITIAL，NEXT，MINEXTENTS，MAXEXTENTS，PCTINCREASE 等参数来为区设置存储管理方式，以指定表空间中区的默认分配方式。这样，如果这种表空间中创建表、索引等方案对象时，没有使用 DEFAULT STORAGE 子句，它们就会自动继承表空间的存储参数设置。

注意：如果其他表空间要采用数据字典管理方式，则要求 SYSTEM 表空间也必须采用数据字典管理方式。

二、本地管理方式（local-managed tablespace）

本地管理方式是一种新的管理方式。Oracle 10g 强烈建议使用本地管理 方式代替数据字典管理方式。从 Oracle 9i 开始，创建爱你表空间时默认地使用本地管理方式。因此，如果要创建字典管理方式的表空间，必须在 CREATE TABESPACE 语句中显示地使用 EXTENT MANAGEMENT DICTIONARY 子句进行声明。

在本地管理方式下，表空间中区的分配与回收的管理信息都被存储在表空间的数据文件中，而与数据字典无关。表空间会在每个数据文件中维护一个位图（bitmap）结构，用于记录表空间爱你中所有区的分配情况。

当在表空间中分配新的区，或回收已分配的区时，Oracle 将对数据文件中的位图进行更新。这种更新不是对表的更新操作，所以不会产生回退信息和重做信息。

与数据字典管理方式，本地管理方式具有如下好处：

*、因为空间的分配和回收不需要对数据库进行访问（只是简单地改变数据文件中的位图），所以能够提高空间存储管理的速度和并发性。

*、能够避免在数据字典管理方式中的空间分配期间可能出现的递归现象，所以提高了空间存储管理的性能。（在数据字典管理方式下，如果对某个表进行了更新，这时会产生存储管理操作，而该存储管理操作肯定会产生回退信息和重做信息，这会对回退段和重做日志文件进行读写，从而又产生存储管理操作，形成了递归现象）。

*、允许将数据库作为只读的备用数据库（standby database）。由于不会产生回退信息和重做信息，所以不会向数据库中写信息，这样就可以将整个数据库设置为只读状态。这种数据库可以作为备用数据库。（在数据字典管理方式下，

如果查询操作包含排序，则数据库需要为排序分配临时段，这会引起存储空间的分配操作，也就会对数据字典进行更新。所以，数据库就必须处于读写状态，而不能处于只读状态）

*、简化了空间分配，因为当指定了 AUTOALLOCATE 子句时，Oracle 会自动选择合适的区大小，不需要用户进行任何干预。

*、减少用户对数据字典的依赖，因为必要的信息都会被存储在数据文件的位图中，而不是保存在数据字典中。

*、不存在磁盘碎片问题，使用位图的方法去查询空闲空间，相邻的空闲块被视为一个大的空闲块，从设计上保证自动合并磁盘碎片，碎片产生后有系统自动消除。

*、DBMS_SPACE_ADMIN 包对本地管理的表空间提供维护过程。

在本地管理方式下，可以用如下两个关键字来只提高表空间的区的分配方式：

*、UNIFORM：统一分配，指定表空间中所有区的大小都相同。

*、AUTOALLOCATE：自动分配，指定由 Oracle 来自动管理区的大小，这时默认设置。

在 AUTOALLOCATE 区的分配方式下，区的大小随表的大小自动地动态改变，它们之间对应关系如下表所示，这一关系由系统自动维持。所以使用 AUTOALLOCATE 关键字是最佳的选择。因为，在这种表空间中创建所有表、索引等方案对象时，都不用使用 STORAGE 子句来设置 INITIAL，NEXT，MINEXTENTS，MAXEXTENTS，PCTINCREASE 等参数来为区设置存储管理方式，即使设置了，也会被忽略。

表大小	区大小
64 KB	64 KB
1 MB	1 MB
64 MB	8 MB
1000 MB	64 MB

10.1.1.2、段管理方式

在本地管理方式的表空间中，除了可以用 UNIFORM 和 AUTOALLOCATE 来指定

区的分配方式之外，还可以指定段的管理方式。段管理方式主要是指 Oracle 用来管理段中已用数据块和空闲数据块的机制。

在本地管理方式下，可以用如下两个关键字来指定表空间的段管理方式：

MANUAL（手工)方式：这时，在 Oracle 将使用可用列表（free list）来管理段的空闲数据块，这时默认的设置。

AUTO（自动）方式：Oracle 将用位图来管理段的已用/空闲数据块。

10.1.2、表空间的状态

10.1.2.1、读写状态

包含两种状态：

一、读写（Read-Write）

二、只读（Read-Only)：如将表空间设置为只读状态，则任何用户（包括 DBA）都无法向表空间写入数据。此种限制与权限无关。

10.1.2.1、脱机状态

SYSTEM 不能被设置为脱机表空间。共有如下四种脱机模式：

1、正常（Normal）

2、临时（Temporary）

3、立即（Immediate）

4、用于恢复（For Recovery）

10．2、管理表空间的准则

表空间的管理主要包括创建、修改、删除表空间，选择修改表空间的区、段管理方式。

10.2.1、创建多个（非 SYSTEM）表空间

10.2.1.1、设置表空间的默认存储参数

10.2.1.2、为用户设置表空间配额

能够在某个表空间中创建方案对象的用户必须满足两个条件：一是具有 CREATE TABLE 等创建方案对象的系统权限，二是在该方案对象使用的表空间中具有配额。

“配额”是指用户在某个指定的表空间中允许使用的存储空间的大小。当用户创建表、索引、簇等具有独立段结构的数据库方案对象时，都必须在表空间中为这些方案对象分配存储空间。一旦该用户用完了在某个空间中为他分配的配额，他将不能再在这个表空间中创建方案对象了。

如果 DBA 需要将资源限制作为自己安全策略中的一部分，那么可以考虑为每个用户都设置表空间配额。

10．3、创建表空间

用户必须拥有 CREATE TABLESAPCE 系统权限才能创建表空间。临时表空间和撤销表空间都是特殊表空间。它们与一般表空间不同的是

10.3.1、创建（永久）表空间

CREATE TABLESPACE

10.3.2、创建大文件表空间

CREATE BIGFILE TABLESPACE

10.3.3、创建临时表空间

CREATE TEMPORARY TABLESPACE

10.3.3.1、创建临时表空间

10.3.3.1、建立大文件临时表空间

10.3.4、创建撤销表空间

CREATE UNDO TABLESPACE

10．4、修改表空间

10.4.1、扩展表空间

10.4.1.1、添加数据文件

10.4.1.2、改变数据文件大小

10.4.1.3、允许数据文件自动扩展

10.4.2、修改属性、状态

10.4.2.1、修改表空间的可用性

10.4.2.2、修改表空间的读写性

10.4.2.3、修改表空间的名称

10.4.2.4、设置默认表空间

10．5、删除表空间

10．6、查询表空间信息

10．7、在 OEM 中管理表空间