# 创建表空间

表空间？ ORACLE数据库的逻辑单元。 数据库---表空间 一个表空间可以与多个数据文件（物理结构）关联一个数据库下可以建立多个表空间，一个表空间可以建立多个用户、一个用户下可以建立多个表。

## 表空间

create tablespace itcast001

datafile 'c:\itcast001.dbf'

size 100m

autoextend on

next 10m

itcast 为表空间名称

datafile 指定表空间对应的数据文件 （Linux 看情况修改）

size 后定义的是表空间的初始大小

autoextend on 自动增长 ，当表空间存储都占满时，自动增长

next 后指定的是一次自动增长的大小。

用户

## 创建用户

创建用户

create user itcastuser

identified by itcast

default tablespace itcast001

identified by 后边是用户的密码

default tablespace 后边是表空间名称

oracle数据库与其它数据库产品的区别在于，表和其它的数据库对象都是存储在用户下的。

## 用户赋权限

新创建的用户没有任何权限，登陆后会提示

Oracle中已存在三个重要的角色：connect角色，resource角色，dba角色。

CONNECT角色： --是授予最终用户的典型权利，最基本的

ALTER SESSION --修改会话

CREATE CLUSTER --建立聚簇

CREATE DATABASE LINK --建立数据库链接

CREATE SEQUENCE --建立序列

CREATE SESSION --建立会话

CREATE SYNONYM --建立同义词

CREATE VIEW --建立视图

RESOURCE角色： --是授予开发人员的

CREATE CLUSTER --建立聚簇

CREATE PROCEDURE --建立过程

CREATE SEQUENCE --建立序列

CREATE TABLE --建表

CREATE TRIGGER --建立触发器

CREATE TYPE --建立类型

DBA角色：拥有全部特权，是系统最高权限，只有DBA才可以创建数据库结构，并且系统权限也需要DBA授出，且DBA用户可以操作全体用户的任意基表，包括删除

grant dba to itcastuser

进入system用户下给用户赋予dba权限，否则无法正常登陆

# 解锁scott 用户

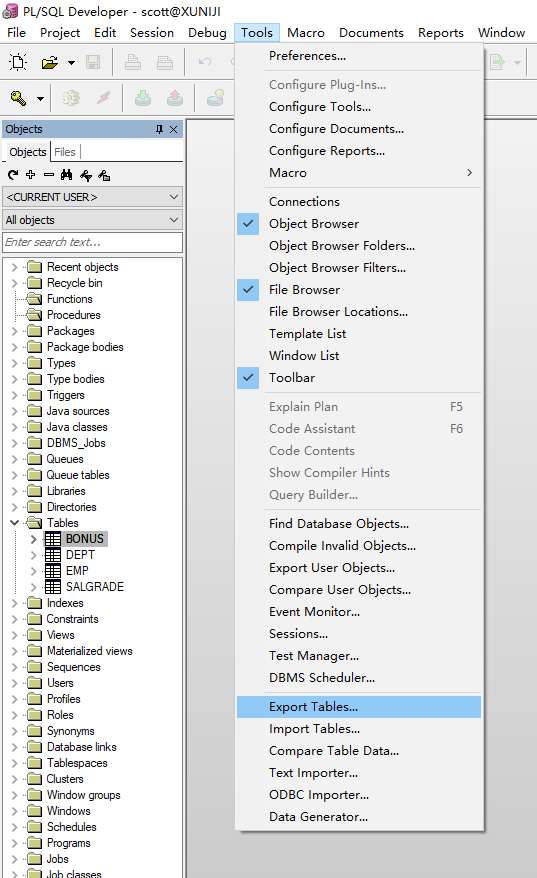
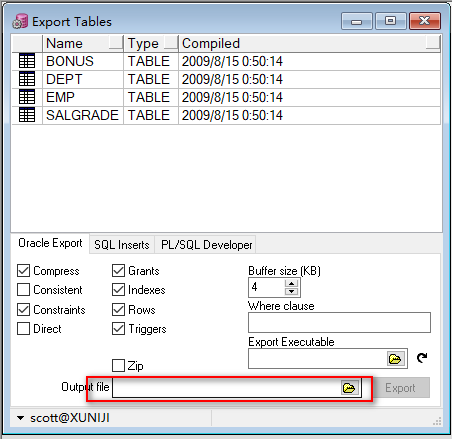
-------解锁scott用户

alter user scott account unlock;

alter user scott identified by tiger;

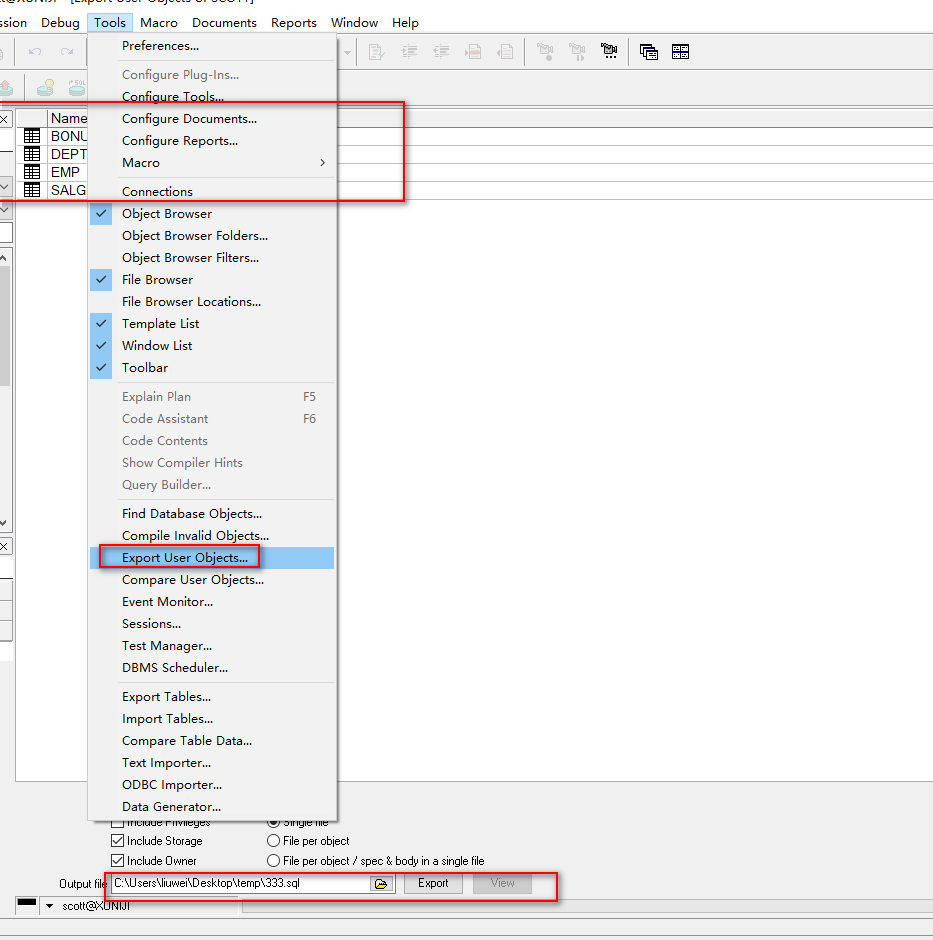
# 导出数据

选择tools – 导出表格

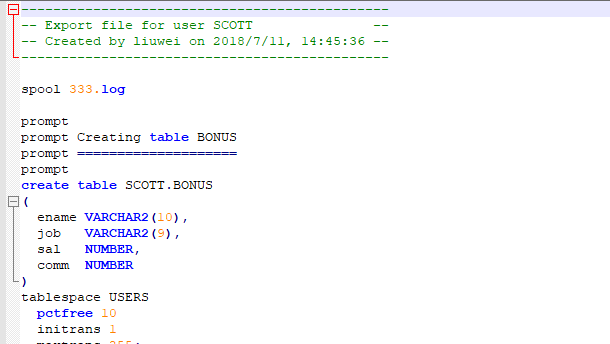
 

## 备份表结构

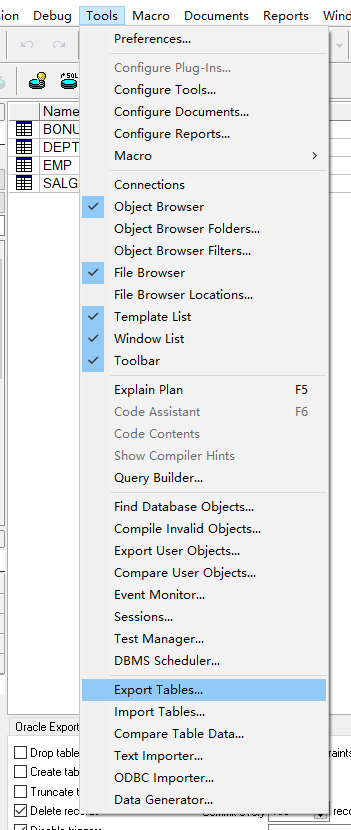
Tools – export user object



运行生成的文件，只有表结构

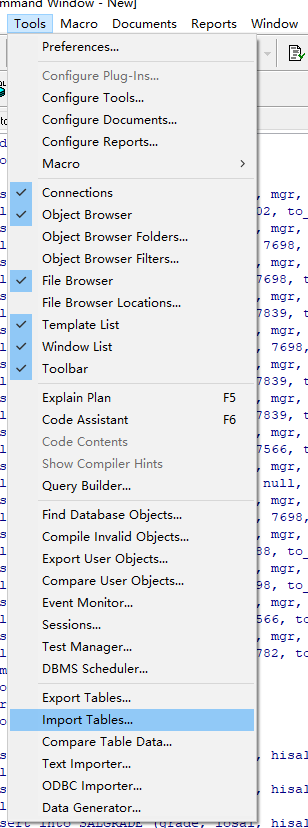


## 备份 数据





## 导入数据



# 第一个存储过程

## 第一个存储过程

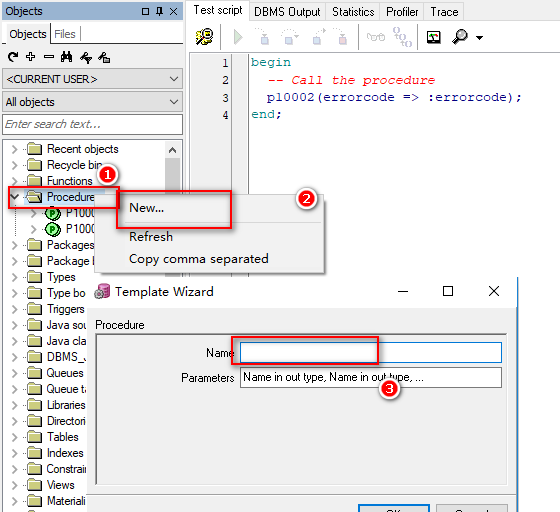
***在使用上述过程之前必须将 SQL \* PLUS 的环境参数 SERVEROUTPUT 设置为 ON, 否则将看不到输出结果: set serveroutput on***

在commond 窗口 或者 sql 窗口执行

set serveroutput on

选择 process -- new -- name

创建一个存储过程



默认的代码

create or replace procedure P10003(Name in out type, Name in out type, ...) is

begin

end P10003;

红色部分需要修改相应的，入参 和 出参数

修改后的代码 Hello world

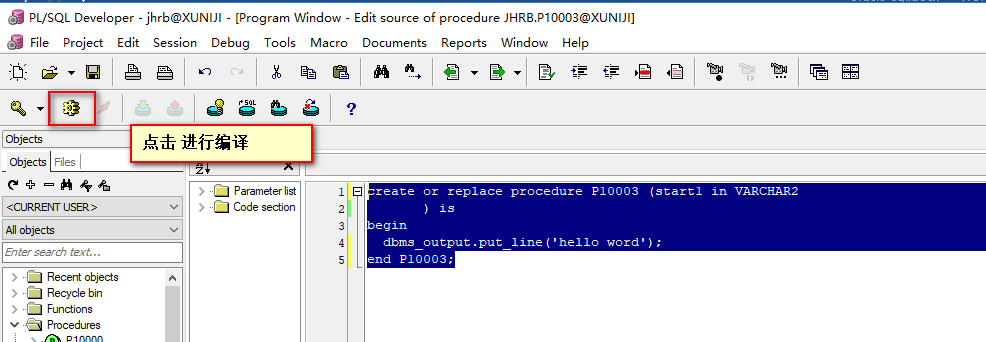
create or replace procedure P10003 as

begin

dbms\_output.put\_line('hello word');

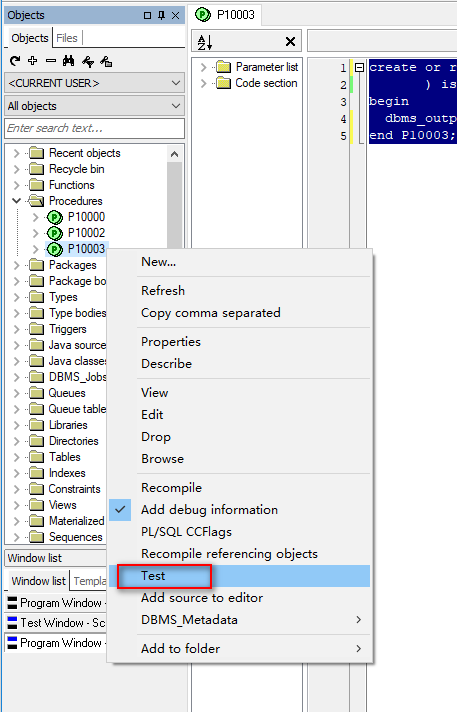
end P10003;

执行的主体就 一行代码，执行hello world



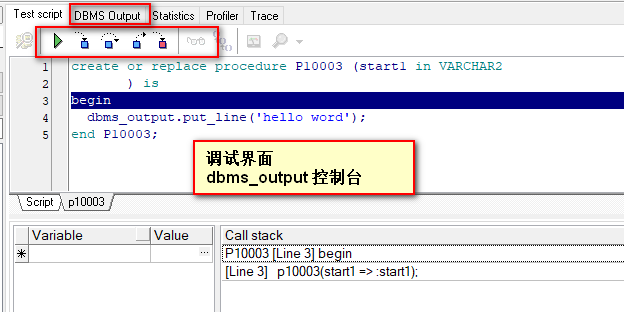
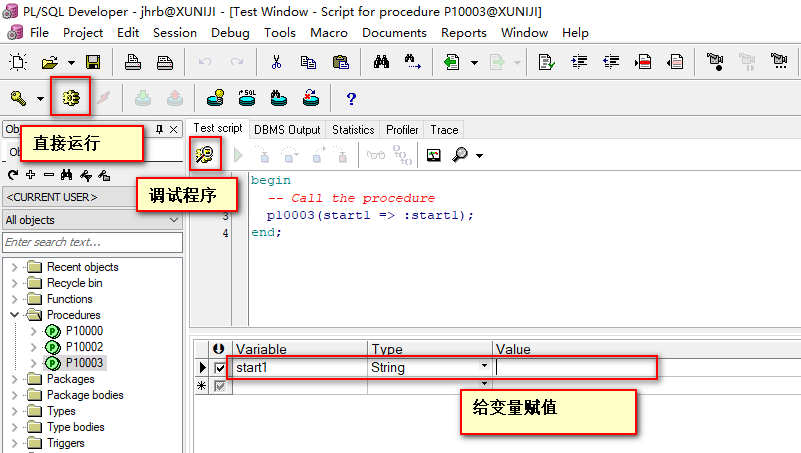
执行第一个程序

选择这个存储过程 -- 右键 – test

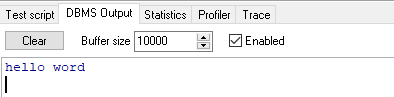


## 启动这个程序

可以进行调试，可以直接运行



## 运行结果



# 存储过程

变量命名在PL/SQL中有特别的讲究，建议在系统的设计阶段就要求所有编程人员共同遵守一定的要求，使得整个系统的文档在规范上达到要求。下面是**建议的命名方法**：

|  |  |  |
| --- | --- | --- |
| 标识符 | 命名规则 | 例子 |
| 程序变量 | **V\_**name | V\_name |
| 程序常量 | **C\_**Name | C\_company\_name |
| 游标变量 | Name**\_cursor** | Emp\_cursor |
| 异常标识 | **E\_**name | E\_too\_many |
| 表类型 | Name**\_table\_type** | Emp\_record\_type |
| 表 | Name**\_table** | Emp |
| 记录类型 | Name**\_record** | Emp\_record |
| SQL\*Plus 替代变量 | **P\_**name | P\_sal |
| 绑定变量 | **G**\_name | G\_year\_sal |

在ORACLE8i中可以使用的变量类型有：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 子类 | 说 明 | 范 围 | ORACLE限制 |
| CHAR | Character  String  Rowid  Nchar | 定长字符串  民族语言字符集 | 0🡪32767  可选,确省=1 | 2000 |
| VARCHAR2 | Varchar, String  NVARCHAR2 | 可变字符串  民族语言字符集 | 0🡪32767  4000 | 4000 |
| BINARY\_INTEGER |  | 带符号整数,为整数计算优化性能 |  |  |
| NUMBER(p,s) | Dec  Double precision  Integer  Int  Numeric  Real  Small int | 小数, NUMBER 的子类型  高精度实数  整数, NUMBER 的子类型  整数, NUMBER 的子类型  与NUMBER等价  与NUMBER等价  整数, 比 integer 小 |  |  |
| LONG |  | 变长字符串 | 0->2147483647 | 32,767字节 |
| DATE |  | 日期型 | 公元前4712年1月1日至公元后4712年12月31日 |  |
| BOOLEAN |  | 布尔型 | TRUE, FALSE,NULL | 不使用 |
| ROWID |  | 存放数据库行号 |  |  |
| UROWID |  | 通用行标识符，字符类型 |  |  |

## 声明变量

通过关键字 声明变量

### 声明一个变量

declare

--声明一个变量

v\_name varchar2(25);

--通过 select ... into ... 语句为变量赋值

单个变量

这样查询dept 表格中一个数据 举例：

create or replace procedure P10005(ERRORCODE in VARCHAR2 *---返回的结果0成功,1失败*

) is

*--声明一个变量*

v\_name VARCHAR2(30);

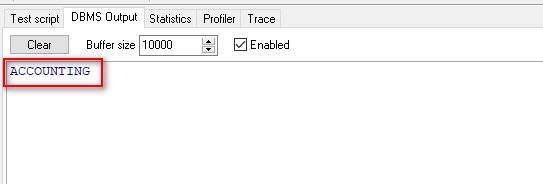
begin

select DNAME into v\_name from dept where deptno = 10;

dbms\_output.put\_line(v\_name);

end P10005;

运行结果



### 声明多个变量

declare

--声明变量

v\_name varchar2(25);

v\_email varchar2(25);

v\_salary number(8, 2);

v\_job\_id varchar2(10);

查询多个变量

create or replace procedure P10004(ERRORCODE in VARCHAR2 *---返回的结果0成功,1失败*

) is

*--声明一个变量*

v\_name VARCHAR2(30);

v\_loc varchar2(30);

begin

select DNAME, loc into v\_name, v\_loc from dept where deptno = 10;

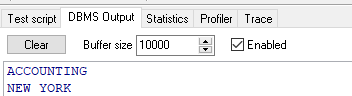
dbms\_output.put\_line(v\_name);

dbms\_output.put\_line(v\_loc);

end P10004;

这样打印的结果是

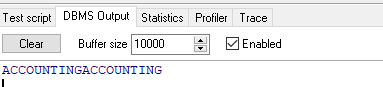
分成两行打印，如果想打印成一行



可以修改成 || 进行连接。

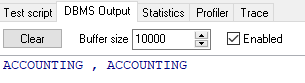
dbms\_output.put\_line(v\_name || v\_name );

执行结果：



连在一起了，因此 需要加上 ， 空格 等进行分开

这次运行结果，就好看多了



## 使用%TYPE

表dept。 中有 deptno 字段。

如果有一个变量 v\_deptno 保存的就是 deptno 的值，数据类型和 deptno 是一样的

我就可以 数据 类型 通过 %type 取出 deptno 的type

SQL> desc dept

Name Type Nullable Default Comments

------ ------------ -------- ------- --------

DEPTNO NUMBER(2)

DNAME VARCHAR2(14) Y

LOC VARCHAR2(13) Y

一个变量的存储过程可以 写成了

v\_name VARCHAR2(30);

就可以修改成 表名 + 点 + 字段名 + %type

v\_name dept.dname%type;

优点很明显。 数据类型 和表的一直，不必知道表的数据类型 。 数据库如果改变，他也能跟着改变

## 记录类型

提供%ROWTYPE操作符, 返回一个记录类型, 其数据类型和数据库表的数据结构相一致

%type 是 表中的一个字段

%rowtype 就是整张表

记录类型的命名规范

|  |  |  |
| --- | --- | --- |
| 记录类型 | Name**\_record** | Emp\_record |

举例：

create or replace procedure P10007(ERRORCODE in VARCHAR2 *---返回的结果0成功,1失败*

) is

*--声明一个变量*

dept\_record dept%rowtype;

begin

select DNAME, loc into dept\_record.dname, dept\_record.loc from dept where deptno = 10;

dbms\_output.put\_line(dept\_record.dname || ' , ' || dept\_record.loc);

end P10007;

声明变量

dept\_record dept%rowtype;

记录类型名\_record 表名 %rowtype

取值的时候 通过 记录类型名\_record 点 获取值 举例： dept\_record.dname

## 运算符

### 关系运算符

|  |  |
| --- | --- |
| 运算符 | **意义** |
| = | 等于 |
| <> , != , ~= , ^= | 不等于 |
| < | 小于 |
| > | 大于 |
| <= | 小于或等于 |
| >= | 大于或等于 |

### 一般运算符

|  |  |
| --- | --- |
| 运算符 | **意义** |
| + | 加号 |
| - | 减号 |
| \* | 乘号 |
| / | 除号 |
| **:=** | **赋值号** |
| **=>** | **关系号** |
| **..** | **范围运算符** |
| || | 字符连接符 |

### 逻辑运算符

|  |  |
| --- | --- |
| 运算符 | **意义** |
| IS NULL | 是空值 |
| BETWEEN AND | 介于两者之间 |
| IN | 在一列值中间 |
| AND | 逻辑与 |
| OR | 逻辑或 |
| NOT | 取返,如IS NOT NULL, NOT IN |

### 注释

在PL/SQL里，可以使用两种符号来写注释，即：

 使用双 ‘-‘ ( 减号) 加注释

PL/SQL允许用 – 来写注释，它的作用范围是只能在一行有效。如：

V\_Sal NUMBER(12,2); -- 工资变量。

 使用 /\* \*/ 来加一行或多行注释，如：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 文件名： department\_salary.sql \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 插入一条数据

数据更改后，需要执行commit，才能写入数据库中

CREATE OR REPLACE PROCEDURE P10008(V\_DEPTNO IN DEPT.DEPTNO%TYPE,

V\_NAME IN DEPT.DNAME%TYPE,

V\_LOC IN DEPT.LOC%TYPE) IS

BEGIN

*-- 插入一条数据*

INSERT INTO DEPT (DEPTNO, DNAME, LOC) VALUES (V\_DEPTNO, V\_NAME, V\_LOC);

commit;

END P10008;

### 删除一条数据

删除一条数据

CREATE OR REPLACE PROCEDURE P10009(V\_DEPTNO IN DEPT.DEPTNO%TYPE

) IS

BEGIN

*-- 删除一条数据*

delete from dept where deptno = V\_DEPTNO;

commit;

END P10009;

# 控制流程语句

* 控制语句: IF 语句
* 循环语句: LOOP语句, EXIT语句
* 顺序语句: GOTO语句, NULL语句

## 控制语句 -- 判断语句

### If 语句

使用 IF ... THEN ... ELSIF ... THEN ... END IF;

通过员工号码。查询员工的工资

判断工资档次

CREATE OR REPLACE PROCEDURE P10010(V\_EMPNO IN EMP.EMPNO%TYPE) IS

v\_sal emp.sal%type;

BEGIN

*-- 查询员工的工资*

SELECT SAL INTO V\_SAL FROM EMP WHERE EMPNO = V\_EMPNO;

DBMS\_OUTPUT.PUT\_LINE(V\_SAL);

*-- 进行工资的判断*

*-- 1000 以下，输出 低薪*

*-- 1000 - 2000 中薪*

*-- 2000 以上 高薪*

IF V\_SAL < 1000 THEN

DBMS\_OUTPUT.PUT\_LINE('低薪');

ELSIF V\_SAL < 2000 THEN

DBMS\_OUTPUT.PUT\_LINE('中薪');

ELSE

DBMS\_OUTPUT.PUT\_LINE('高薪');

end if;

END P10010;

注意：

else if 🡪 elsif 没有e

**ELSIF 不能写成 ELSEIF**

## case 语句

case when … then 语句

CREATE OR REPLACE PROCEDURE P10011(V\_EMPNO IN EMP.EMPNO%TYPE) IS

V\_DEPTNO EMP.DEPTNO%TYPE;

V\_DETPNAME VARCHAR2(30);

BEGIN

*-- 查询员工的工资*

SELECT DEPTNO INTO V\_DEPTNO FROM EMP WHERE EMPNO = V\_EMPNO;

DBMS\_OUTPUT.PUT\_LINE(V\_DEPTNO);

V\_DETPNAME := CASE V\_DEPTNO

WHEN 10 THEN

'高级'

WHEN 20 THEN

'中级'

ELSE

'其他'

END;

DBMS\_OUTPUT.PUT\_LINE(V\_DETPNAME);

END P10011;

## 循环语句

### 简单循环

#### Loop 循环

LOOP

要执行的语句;

EXIT WHEN <条件语句> /\*条件满足，退出循环语句\*/

END LOOP;

控制台输出 1-100

CREATE OR REPLACE PROCEDURE P10012 as

v\_count number(3) := 1;

BEGIN

loop

dbms\_output.put\_line(v\_count);

v\_count := v\_count + 1;

EXIT WHEN v\_count > 100;

end loop;

dbms\_output.put\_line('over');

END P10012;

#### While 循环

结构：

**WHILE** <布尔表达式> **LOOP**

要执行的语句;

**END LOOP;**

控制台输出 1 -100

CREATE OR REPLACE PROCEDURE P10013 AS

V\_COUNT NUMBER(3) := 1;

BEGIN

WHILE V\_COUNT <= 100 LOOP

DBMS\_OUTPUT.PUT\_LINE(V\_COUNT);

V\_COUNT := V\_COUNT + 1;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('over');

END P10013;

这两种循环 一种是 条件 成立的时候 结束，一种是条件为真的时候执行

#### 数字循环

**FOR** 循环计数器 **IN** [ REVERSE ] 下限 .. 上限 **LOOP**

要执行的语句;

**END LOOP;**

CREATE OR REPLACE PROCEDURE P10014 AS

BEGIN

*-- 循环 从 1 到 100*

FOR I IN 1 .. 100 LOOP

DBMS\_OUTPUT.PUT\_LINE(I);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('over');

END P10014;

# 游标

对于不同的SQL语句，游标的使用情况不同：

|  |  |
| --- | --- |
| SQL语句 | 游标 |
| 非查询语句 | 隐式的 |
| 结果是单行的查询语句 | 隐式的或显示的 |
| 结果是多行的查询语句 | 显示的 |

## 显式游标处理

**属性**

**%FOUND 布尔型属性，当最近一次读记录时成功返回,则值为TRUE；**

%NOTFOUND 布尔型属性，与%FOUND相反；

%ISOPEN 布尔型属性，当游标已打开时返回 TRUE；

%ROWCOUNT 数字型属性，返回已从游标中读取的记录数。

**定义游标**

就是定义一个游标名，以及与其相对应的SELECT 语句。

格式：

CURSOR C\_CURSOR IS *--声明游标指向select的结果集*

SELECT \* FROM EMP;

**打开游标**

OPEN C\_CURSOR; *--打开游标*

**PL/SQL 程序不能用OPEN 语句重复打开一个游标。**

**提取游标数据**

就是检索结果集合中的数据行，放入指定的输出变量中。

格式：

FETCH C\_CURSOR

INTO V\_EMP; *--取出游标当前执向的值存入v\_emp，每fetch一次，游标指向下一条记录*

### 简单例子

通过使用游标 控制台输出 所有员工表的 名字

CREATE OR REPLACE PROCEDURE P10016 AS

CURSOR C\_CURSOR IS *--声明游标指向select的结果集*

SELECT \* FROM EMP;

V\_EMP C\_CURSOR%ROWTYPE;

BEGIN

OPEN C\_CURSOR; *--打开游标*

LOOP

FETCH C\_CURSOR

INTO V\_EMP; *--取出游标当前执向的值存入v\_emp，每fetch一次，游标指向下一条记录*

EXIT WHEN(C\_CURSOR%NOTFOUND); *--找不到就退出*

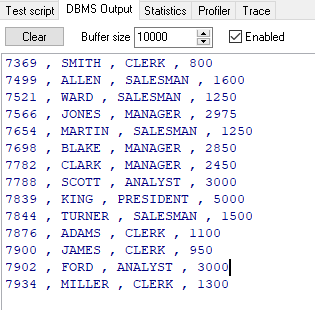
DBMS\_OUTPUT.PUT\_LINE( V\_EMP.EMPNO || ' , ' || v\_emp.ENAME || ' , ' || v\_emp.JOB || ' , ' || v\_emp.SAL);

END LOOP;

CLOSE C\_CURSOR; *--关闭游标*

END P10016;

输出结果：



### 项目中使用

在项目中使用，游标是参数

通过open MESSAGELIST FOR V\_SQL; 来打开游标 进行获取内容

CREATE OR REPLACE PROCEDURE P10017 (MESSAGELIST OUT SYS\_REFCURSOR) *--返回通道的列表*

is

V\_SQL VARCHAR2(2000); *-- 用于保存sql 语句*

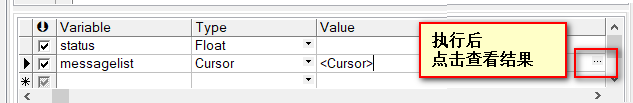
BEGIN

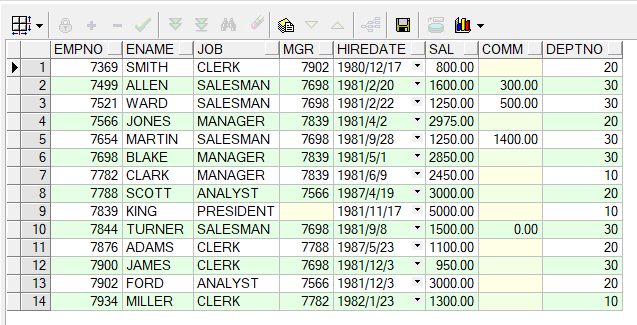
V\_SQL := 'select \* from emp';

open MESSAGELIST FOR V\_SQL;

END P10017;

查看结果





### 添加判断条件

可以通过if 语句 判断

比如说 参数 status 为

1 查询部门是 10

2 查询部门是 20

这样代码 可以修改成

CREATE OR REPLACE PROCEDURE P10017(STATUS IN NUMBER, MESSAGELIST OUT SYS\_REFCURSOR) *--返回通道的列表*

IS

V\_SQL VARCHAR2(2000); *-- 用于保存sql 语句*

BEGIN

V\_SQL := 'select \* from emp';

*-- 添加查询的条件*

IF STATUS = 1 THEN

V\_SQL := V\_SQL || 'where DEPTNO = 10';

ELSIF STATUS = 2 THEN

V\_SQL := V\_SQL || 'where DEPTNO = 20';

END IF;

OPEN MESSAGELIST FOR V\_SQL;

END P10017;

## 隐式游标

概述：

显式游标主要是用于对查询语句的处理，尤其是在查询结果为多条记录的情况下；而对于非查询语句，如修改、删除操作，则由ORACLE 系统自动地为这些操作设置游标并创建其工作区，这些**由系统隐含创建的游标称为隐式游标**，**隐式游标的名字为SQL**，这是由ORACLE 系统定义的。对于隐式游标的操作，如定义、打开、取值及关闭操作，都由ORACLE 系统自动地完成，无需用户进行处理。**用户只能通过隐式游标的相关属性，来完成相应的操作**。在隐式游标的工作区中，所存放的数据是与用户自定义的显示游标无关的、最新处理的一条SQL 语句所包含的数据。

格式调用为： SQL%

**隐式游标属性**

SQL%FOUND 布尔型属性,当最近一次读记录时成功返回，则值为TRUE；

SQL%NOTFOUND 布尔型属性,与%FOUND相反；

SQL %ROWCOUNT 数字型属性, 返回已从游标中读取得记录数；

SQL %ISOPEN 布尔型属性, 取值总是FALSE。SQL命令执行完毕立即关闭隐式游标。

通过 id 和 name 两个参数进行更新

如果有数据，控制台相应的输出

CREATE OR REPLACE PROCEDURE P10008(V\_NAME DEPT.DNAME%TYPE, V\_NUMBER DEPT.DEPTNO%TYPE) IS

BEGIN

*-- 进行更新数据*

UPDATE DEPT SET DNAME = V\_NAME WHERE DEPTNO = V\_NUMBER;

*-- 隐形游标的判断*

IF SQL%NOTFOUND THEN

DBMS\_OUTPUT.PUT\_LINE('没有更新数据');

ELSE

DBMS\_OUTPUT.PUT\_LINE('更新了数据');

COMMIT;

END IF;

END P10008;

# 异常错误

预定义说明的部分 ORACLE 异常错误

|  |  |  |
| --- | --- | --- |
| 错误号 | 异常错误信息名称 | 说明 |
| ORA-0001 | Dup\_val\_on\_index | 试图破坏一个唯一性限制 |
| ORA-0051 | Timeout-on-resource | 在等待资源时发生超时 |
| ORA-0061 | Transaction-backed-out | 由于发生死锁事务被撤消 |
| ORA-1001 | Invalid-CURSOR | 试图使用一个无效的游标 |
| ORA-1012 | Not-logged-on | 没有连接到ORACLE |
| ORA-1017 | Login-denied | 无效的用户名/口令 |
| ORA-1403 | No\_data\_found | SELECT INTO没有找到数据 |
| ORA-1422 | Too\_many\_rows | SELECT INTO 返回多行 |
| ORA-1476 | Zero-divide | 试图被零除 |
| ORA-1722 | Invalid-NUMBER | 转换一个数字失败 |
| ORA-6500 | Storage-error | 内存不够引发的内部错误 |
| ORA-6501 | Program-error | 内部错误 |
| ORA-6502 | Value-error | 转换或截断错误 |
| ORA-6504 | Rowtype-mismatch | 宿主游标变量与 PL/SQL变量有不兼容行类型 |
| ORA-6511 | CURSOR-already-OPEN | 试图打开一个已存在的游标 |
| ORA-6530 | Access-INTO-null | 试图为null 对象的属性赋值 |
| ORA-6531 | Collection-is-null | 试图将Exists 以外的集合( collection)方法应用于一个null pl/sql 表上或varray上 |
| ORA-6532 | Subscript-outside-limit | 对嵌套或varray索引得引用超出声明范围以外 |
| ORA-6533 | Subscript-beyond-count | 对嵌套或varray 索引得引用大于集合中元素的个数. |

对这种异常情况的处理，只需在PL/SQL块的异常处理部分，直接引用相应的**异常情况名**，并对其完成相应的异常错误处理即可。

查询一条数据，如果这个数据这个数据

在异常中处理

CREATE OR REPLACE PROCEDURE P10009(V\_DEPTNO DEPT.DEPTNO%TYPE) IS

*-- 定义一个记录类型 保存数据*

DEPT\_RECORD DEPT%ROWTYPE;

BEGIN

*-- 进行查询*

SELECT \* INTO DEPT\_RECORD FROM DEPT WHERE DEPTNO = V\_DEPTNO;

EXCEPTION

*-- 没有找到数据 处理*

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('没有找到数据');

END P10009;