大型数据库应用技术

05-SQL和PL/SQL

授课教师：王欢

准备知识

连接数据库

- 准备工作:
  - 1 安装Oracle软件
  - 2 配置好监听和服务
  - 3 确认两个服务已启动

    OracleServiceORCL 和 OracleOraDB12Home1TNSListener
- 打开数据库:
  - 1 Windows 命令行窗口（cmd）以DBA权限连接数据库：sqlplus / as sysdba
  - 2 查看pdb状态 show pdbs
  - 3 打开数据库alter pluggable database pdborcl open;

C:\Windows\system32\cmd.exe - sqlplus / as sysdba

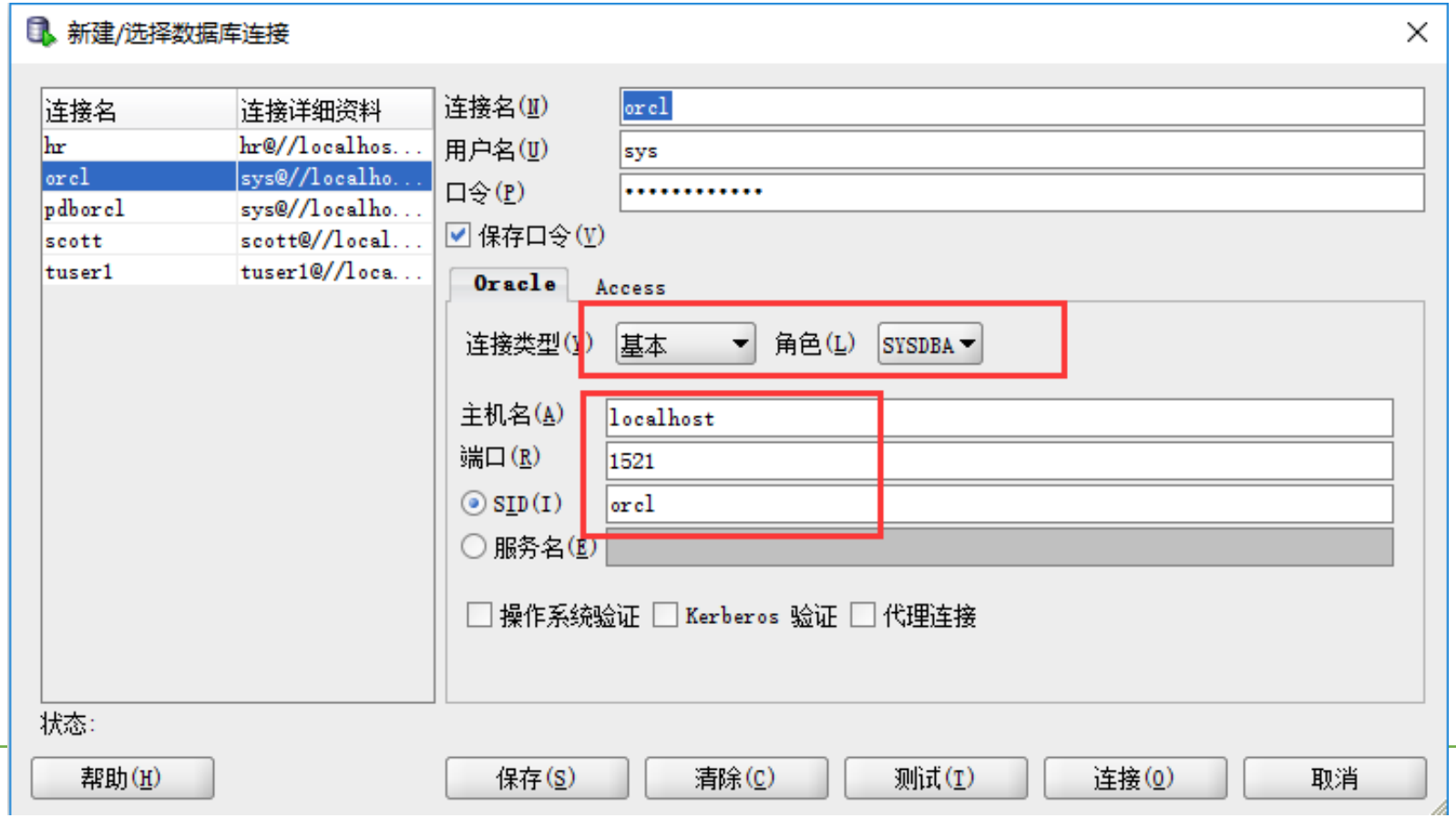
```
Microsoft Windows [版本 10.0.17134.1246]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\LENOVO>sqlplus / as sysdba

SQL*Plus: Release 12.1.0.2.0 Production on Wed Oct 7 09:38:51 2020

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> show pdbs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 PDBORCL                        MOUNTED
SQL> alter pluggable database pdborcl open;

Pluggable database altered.

SQL> select name, open_mode from v$pdbs;

NAME                           OPEN_MODE
------------------------------ ----------
PDB$SEED                       READ ONLY
PDBORCL                        READ WRITE
```

准备知识

连接数据库

- **连接数据库**：

1 打开 SQL Developer　　**2** 设置连接



内容提纲

- PL/SQL概述
- 数据类型
- 程序结构和语句
- 游标
- 存储过程、函数、触发器
- 程序包

PL/SQL编程

PL/SQL概述

PL/SQL（Procedural Language / SQL）是 Oracle 在标准 SQL 上扩展后的程序设计语言，是 Oracle 数据库特有的、支持应用开发的语言。

PL/SQL程序块的结构：

[DECLARE]——声明部分，可选

声明一些变量、常量、用户定义的数据类型以及游标等

BEGIN——执行部分，必须

主程序体，可以加入各种合法语句

[EXCEPTION]——异常处理部分，可选

异常处理程序，当程序中出现错误时执行这 一部分

END ；——主程序体结束

PL/SQL编程

PL/SQL概述

一个PL/SQL程序包含一个或多个块，这些块可以独立存在，也可以嵌套在另一个块中，PL/SQL程序有三种类型的块组成，分别为匿名块、过程和函数。

匿名块	过程	函数

[DECLARE]
 	
BEGIN
-statements
 	
EXCEPTION
 	
END;
	PROCEDURE name
	IS
	
	BEGIN
	-statements
	
	EXCEPTION
	
	END;
		FUNCTION name
		RETURN datatype
		IS
		BEGIN
		-statements
		
		EXCEPTION
		
		END;

PL/SQL编程

PL/SQL概述

- 匿名块是未命名的块，在运行时被执行，匿名块不存储在数据库中，如果要再次执行相同的块，则必须重写该块，并在执行时重新进行编译。由于块是匿名的，每次执行后该块就不再存在，开发者不能调用以前写过的匿名块。
- 过程和函数都称作子程序，它们是匿名块的有益补充，是存储在数据库中被命名的PL/SQL块。由于子程序已经命名，且存储在数据库中，开发者可以随时调用它们，可以将子程序声明为过程或函数，通常使用过程执行一些操作，而使用函数进行计算并返回一个值。
- 函数和过程类似，不同的是函数必须有返回值，而过程没有。

内容提纲

- PL/SQL概述
- 数据类型
- 程序结构和语句
- 游标
- 存储过程、函数、触发器
- 程序包

字符集

- 合法字符
  - 所有的大写和小写英文字母，数字0~9；
  - 符号 ()、+、-、*、/、<、>、=、!、~、;、:、.、’ 、@、%、,、” 、#、^、&、_、{、}、?、[、]。
- 运算符
  - 算术运算符：+（加）、-（减）、*（乘）、/（除）、**（指数）和 ||（连接字符）；
  - 关系运算符：比较运算符、BETWEEN … AND …、IN、LIKE、IS NULL / IS NOT NULL；
  - 逻辑运算符：AND、OR、NOT。
- 其它符号：列表分隔、项分离、字符串界定符、注释符等。

PL/SQL编程

标识符

- 标识符用于定义PL/SQL变量、常量、异常、游标名称、游标变量、参数、子程序名称和其他的程序单元名称等。
- 在PL/SQL程序中，标识符是以字母开头的，后边可以跟字母、数字、美元符号（$）、井号（#）或下划线（_），其最大长度为30个字符。
- 标识符中不能包含减号“-”和空格，不能使SQL保留字
  - 例如，X，v_empno，v_$等都是有效的标识符，而X-y，_temp则是非法的标识符。
- 注意：一般不要把变量名声明与表中字段名完全一样,如果这样可能得到不正确的结果。

数据类型

类型概览

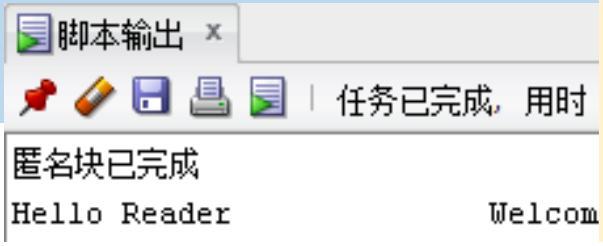
- 系统预定义类型
  - 标量型：标量数据类型的变量只有一个值，且内部没有分量。包括数值型，字符型，日期/时间型和布尔型。
  - 复合型：含有能够被单独操作的内部组件，包括 record，table 和 cursor 型。
  - LOB 型：专门用于存储大对象的数据，包括大文本、图像图像、视频剪辑等。分为内部 LOB 和外部 LOB。
  - 引用类型：引用数据类型是 PL/SQL 程序语言特有的数据类型，是用来引用数据库当中的某一行或者某个字段作为数据类型的声明。
- 用户自定义子类型：

```
SUBTYPE subtype_name IS base_type[ (constraint) ] [ NOT NULL ];
```

SUBTYPE *subtype_name* IS *base_type*[(constraint)] [NOT NULL];

用户定义子类型示例

```
SET serveroutput ON - -显示服务器输出信息
DECLARE
  SUBTYPE name IS    CHAR(20);                  --定义了一个name类型
  SUBTYPE MESSAGE IS VARCHAR2(100); --定义了一个MESSAGE类型
  salutation name;                              --定义了一个name类型的变量
  greetings MESSAGE;                        --定义了一个MESSAGE类型的变量
BEGIN
  salutation := 'Reader ';
  greetings  := 'Welcome to the World
  dbms_output.put_line('Hello ' || salu
END;
```



注意

若要在SQL*Plus环境中看到DBMS_OUTPUT.PUT_LINE方法的输出结果，必须将环境变量SERVEROUTPUT设置为ON。

SET SERVEROUTPUT ON

数据类型

标量型——数值类型和子类型

序号	类型	描述
1	PLS_INTEGER	带符号整数：-2,147,483,648至2,147,483,647，以32位表示
2	BINARY_INTEGER	带符号整数：-2,147,483,648至2,147,483,647，以32位表示
3	BINARY_FLOAT	单精度IEEE 754格式浮点数
4	BINARY_DOUBLE	双精度IEEE 754格式浮点数
5	NUMBER(prec, scale)	在1E-130到(但不包括)1.0E126范围内的绝对值的定点或浮点数。NUMBER变量也可以表示0
6	DEC(prec, scale)	ANSI特定定点类型，最大精度为38位十进制数字
7	DECIMAL(prec, scale)	IBM具体定点类型，最大精度为38位十进制数字
8	NUMERIC(prec, seale)	浮点型，最大精度为38位十进制数
9	DOUBLE PRECISION	ANSI特定浮点类型，最大精度为126位二进制数字
10	FLOAT	ANSI和IBM特定浮点类型，最大精度为126位二进制数字
11	INT	ANSI特定整数类型，最大精度为38位十进制数
12	INTEGER	ANSI和IBM特定整数类型，最大精度为38位十进制数
13	SMALLINT	ANSI和IBM特定整数类型，最大精度为38位十进制数
14	REAL	浮点型，最大精度为63位二进制数字(约十八位数)

数据类型

标量型——字符类型和子类型

序号	类型	描述
1	CHAR	固定长度字符串，最大大小为32,767字节
2	VARCHAR2	最大大小为32,767字节的可变长度字符串
3	RAW	最大大小为32,767字节的可变长度二进制或字节字符串，不由PL/SQL解释
4	NCHAR	固定长度的国家字符串，最大大小为32,767字节
5	NVARCHAR2	可变长度的国家字符串，最大大小为32,767字节
6	LONG	最大长度为32,760字节的可变长度字符串
7	LONG RAW	最大大小为32,760字节的可变长度二进制或字节字符串，不由PL/SQL解释
8	ROWID	物理行标识符，普通表中的行的地址
9	UROWID	通用行标识符(物理，逻辑或外部行标识符)

数据类型

标量型——日期/时间类型

序号	类型	描述
1	DATE	7字节的定宽日期/时间数据类型。包含7个属性，包括：世纪、世纪中的年、月份、月份中的天、小时、分钟和秒。 示例：2005-12-05 12:30:43 存储的是 120, 105, 12, 5, 12, 31, 44。 解释：世纪和年份分别加100，分钟和秒钟分别加1。
2	TIMESTAMP	与DATE的区别是不仅可以保存日期和时间，还能保存小数秒，小数位数可以指定为0-9，默认为6位，所以最高精度可以到ns(纳秒)，数据库内部用7或者11个字节存储，如果精度为0，则用7字节存储，与DATE类型功能相同，如果精度大于0则用11字节存储。
3	INTERVAL	INTERVAL YEAR TO MONTH：单位为年和月的时间间隔； INTERVAL DAY TO SECOND：单位为天和秒的时间间隔；

INTERVAL YEAR [(year_precision)] TO MONTH
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]
year_precision、day_precision、fractional_seconds_precision：精度位数，默认年和日是两位数，小数秒是六位数。
示例：INTERVAL '123 2:25:45.1247' DAY(3) TO SECOND(4)
说明：时间间隔为123天2小时25分45.1247秒；天的精度是3位数字，小数秒精度是4位数字。

数据类型

复合型

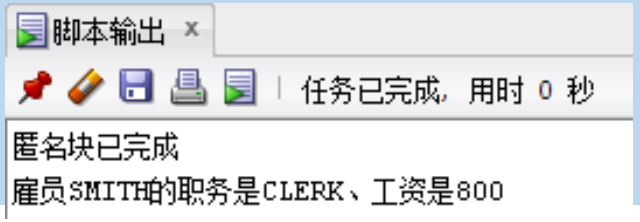
序号	类型	描述
1	RECORD	记录类型，使用该类型的变量可以存储由多个列值组成的一行数据。在声明记录类型变量之前，首先需要定义记录类型，然后才可以声明记录类型的变量。 TYPE record类型名称 is record(属性 数据类型, 属性 数据类型 …);
2	TABLE	使用记录类型变量只能保存一行数据，这限制了SELECT语句的返回行数，如果SELECT语句返回多行就会错。Oracle提供了另外一种自定义类型，也就是表类型，它是对记录类型的扩展，允许处理多行数据，类似于表。 TYPE table类型名称 IS TABLE OF data_type [NOT NULL] INDEX BY BINARY_INTEGER ;
3	CURSOR	游标 后续详细介绍。

RECORD类型使用示例

【例】SCOTT 模式中，声明一个记录类型，然后使用该类型的变量存储 emp 表中的一条记录信息并输出。

```
SET serveroutput ON
DECLARE
type emp_type IS record              --声明record类型emp_type
  (
    var_ename  VARCHAR2(20),    --定义字段/成员变量
    var_job        VARCHAR2(20),
    var_sal        NUMBER );
  empinfo  emp_type;
BEGIN
  SELECT ename, job, sal INTO empinfo FROM emp WHERE empno=7369;
  /*输出雇员信息*/
  dbms_output.put_line('雇员'||empinfo.var_ename||'的职务是
'||empinfo.var_job|| '、工资是'||empinfo.var_sal);
END;
```

PL/SQL是面向过程语言和SQL语言的结合，可使用SELECT-INTO语句将SQL语言查询结果存入变量。



数据类型

LOB 型

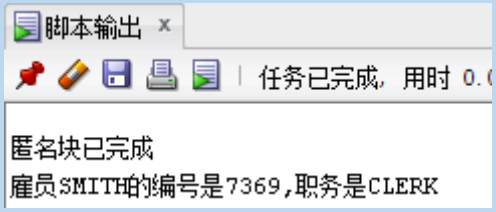
序号	类型	描述
1	BFILE	外部LOB类型，二进制文件LOB。与其说是一个数据库存储实体，不如说是一个指针。带BFILE列的数据库存储的只是某个文件的一个指针。这个文件在数据库之外维护，不是数据库的一部分。BFILE提供了对文件内容的只读访问。
2	BLOB	二进制LOB。用于存储大量的二进制信息，如文档、图像和任何其他数据。它不会执行字符集转换。应用向BLOB写入什么位和字节，其就会返回什么位和字节。
3	CLOB	字符LOB。这种类型用于存储大量的文本信息，如xml或者只是纯文本。这个数据类型需要进行字符集转换，也就是说，在获取时，这个字段中的字符会从数据库的字符集转换为客户的字符集，而在修改时会总客户的字符集转换为数据库的字符集。
4	NCLOB	存储在这一列中的数据所采用的字符集是数据库的国家字符集，而不是数据库的默认字符集。

数据类型

引用型

序号	类型	描述
1	%TYPE	引用数据表的某列的类型作为某变量的数据类型，也可以直接引用PL/SQL程序中某个变量作为新变量的数据类型。 示例：pename emp.ename%type; 解释：创建一个变量pename，其类型与emp表ename列类型相同；
2	%ROWTYPE	引用数据库表中的一行作为数据类型，即 RECORD 类型表示一条数据记录。 示例：rowVar_emp emp%rowtype; 解释：创建一个变量rowVar_emp（Record类型），其能够存储emp表的一行数据；

```
DECLARE
  rowVar_emp emp%rowtype;
BEGIN
  SELECT * INTO rowVar_emp FROM emp WHERE empno=7369;
  dbms_output.put_line('雇员'||rowVar_emp.ename||'的编号是'||rowVar_emp.empno||',职务是'||rowVar_emp.job);
END;
```



口数据类型

- %TYPE类型：使用 %TYPE 关键字可以声明一个与指定列名称相同的数据类型，它通常紧跟在指定列名的后面。

变量名　表名.字段名% %type

v_job　emp.job %type

使用%type的优点：

- 所引用的数据库列的数据类型可以不必知道；
- 所引用的数据库列的数据类型可以实时改变。

□数据类型

- %ROWTYPE 类型：可以根据数据表中行的结构定义一种特殊的数据类型，用来存储从数据表中检索到的一行数据。

```
记录变量名    表名%rowtype
rowVar_emp   emp %rowtype
```

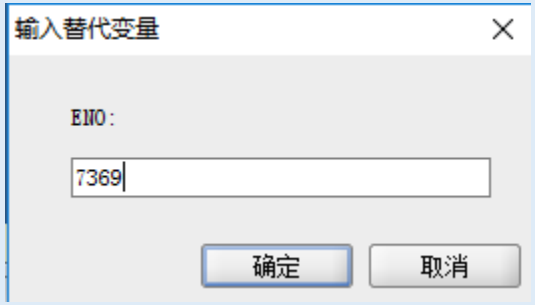
使用%ROWTYPE特性的优点：

- 所引用的数据库中列的个数和数据类型可以不必知道；
- 所引用的数据库中列的个数和数据类型可以实时改变。

%ROWTYPE类型

```
DECLARE
  v_emp emp%rowtype;
BEGIN
  select * into v_emp from emp where empno=&eno;
  dbms_output.put_line('员工名字:'||v_emp.ename);
  dbms_output.put_line('员工工资:'||v_emp.sal);
  dbms_output.put_line('员工职位'||v_emp.job);
END;
```

PL/SQL是面向过程语言和SQL语言的结合，可使用SELECT-INTO语句将SQL语言查询结果存入变量。




```
匿名块已完成
员工名字:SMITH
员工工资:800
员工职位CLERK
```

数据类型

变量和常量

- 变量是指其值在程序运行过程中可以改变的数据存储结构，定义变量必须的元素就是变量名和数据类型，另外还有可选择的初始值。

<变量名> <数据类型> [:= <初始值>];

  - PL/SQL定义了一个未初始化变量应该存放的内容，被赋值为NULL。

- 常量是指其值在程序运行过程中不可改变的数据存储结构，定义常量必须的元素包括常量名、数据类型、常量值和 constant 关键字。

<常量名> constant <数据类型> := <常量值>;

```
Birthday DATE ;
emp_count SMALLINT  :=0;
credit_limit CONSTANT REAL:=5000.00;
pi REAL:=3.14159;
radius REAL:=1;
area REAL:= pi*radius**2;
```

变量和常量示例

内容提纲

- PL/SQL概述
- 数据类型
- 程序结构和语句
- 游标
- 存储过程、函数、触发器
- 程序包

程序结构和语句

程序块结构

[DECLARE]——声明部分，可选

声明块中使用的变量、常量、用户定义数据类型以及游标等，在块中声明的内容只能在当前块中使用。

BEGIN——执行部分，必须

主程序体，可以加入各种合法语句

[EXCEPTION]——异常处理部分，可选

异常处理程序，当程序中出现异常时执行这一部分，如果程序进入了异常处理部分，这部分语句执行完毕，整个块结束，并不返回出现异常处。

END; ——主程序体结束

程序结构和语句

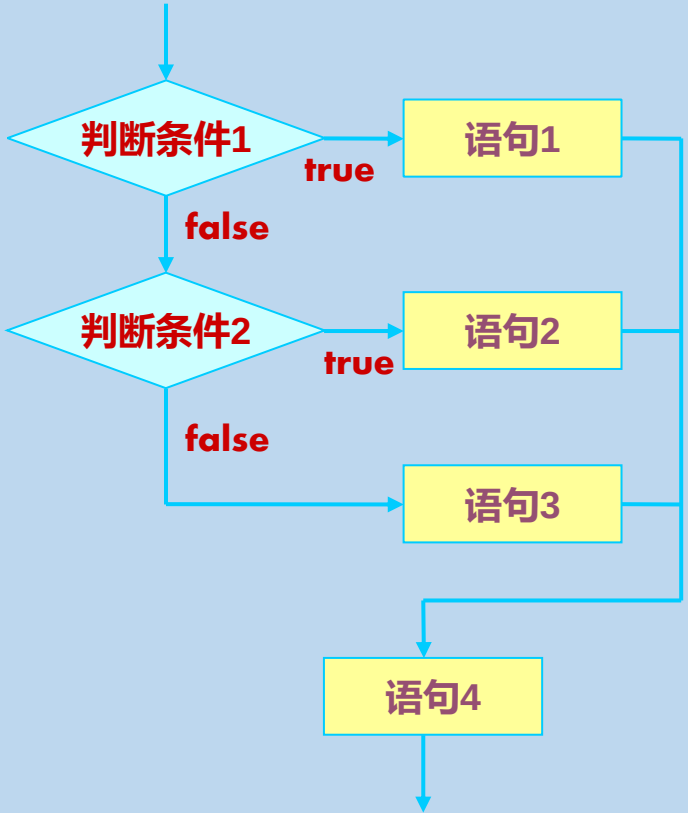
结构控制语句

控制语句	意义说明
if...then	判断 if 正确则执行 then
if...then...else	判断 if 正确则执行 then，否则执行 else
if...then...elsif	嵌套式判断
case	有逻辑地从数值中做出选择
loop...exit...end	循环控制，用判断语句执行 exit
loop...exit when...end	同上，当 when 为真时执行 exit
while...loop...end	当 while 为真时循环
for...in...loop...end	已知循环次数的循环
goto	无条件转向控制

程序结构和语句

结构控制语句——if…then…elsif 语句

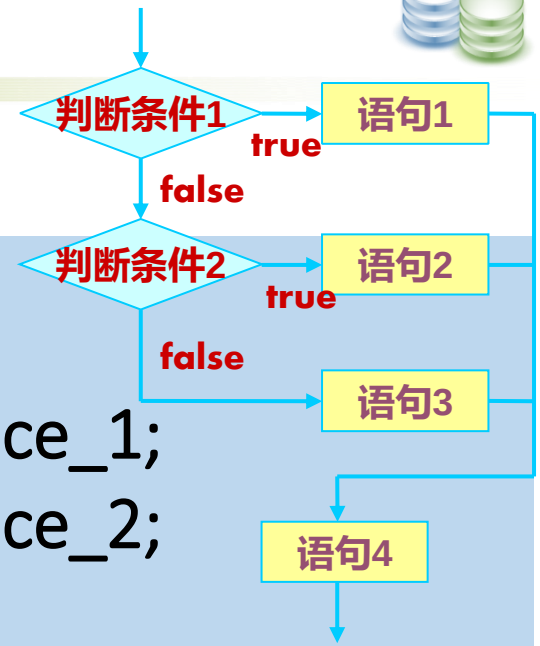
```
if < condition_expression1 > then
plsql_sentence_1;
elsif < condition_expression2 > then
plsql_sentence_2;
…
else
plsql_sentence_n;
end if;
```



注意：由于PL/SQL中的逻辑运算结果有TRUE，FALSE和NULL三种，因此在进行选择条件判断时，要考虑条件为NULL的情况。

程序结构和语句

结构控制语句——case 语句



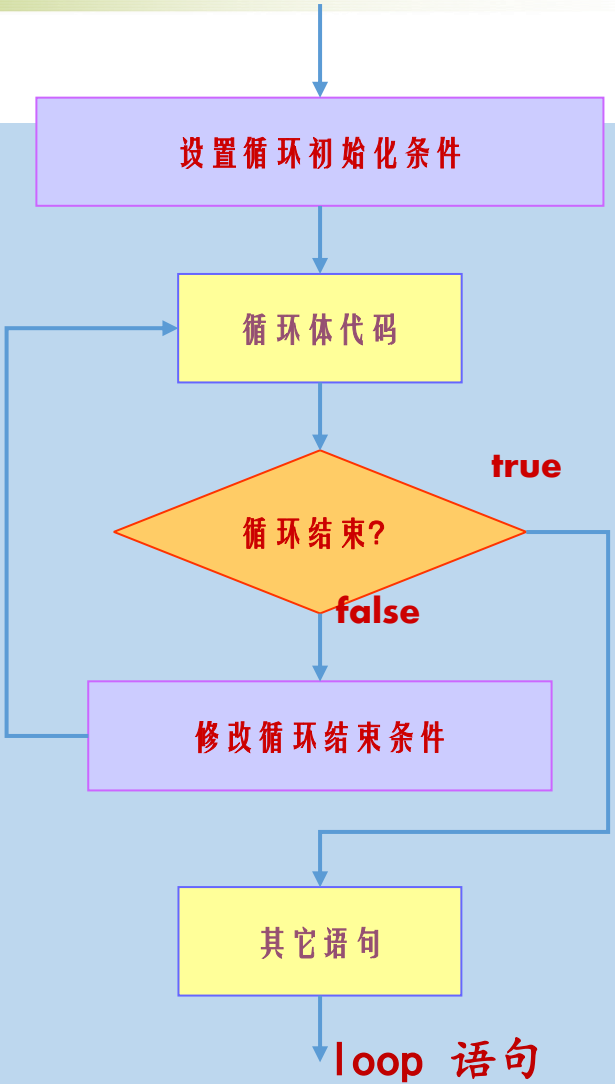

```
case < selector>
    when <expression_1> then plsql_sentence_1;
    when <expression_2> then plsql_sentence_2;
    …
    when <expression_n> then plsql_sentence_n;
    [else plsql_sentence;]
end case;
```

注意：在CASE语句中，当第一个WHEN条件为真时，执行其后的操作，操作完后结束CASE语句。其他的WHEN条件不再判断，其后的操作也不执行

程序结构和语句

结构控制语句——loop 语句

```
Loop
    循环执行的语句块;
    exit when 循环结束条件;
    循环结束条件修改;
end loop;
```

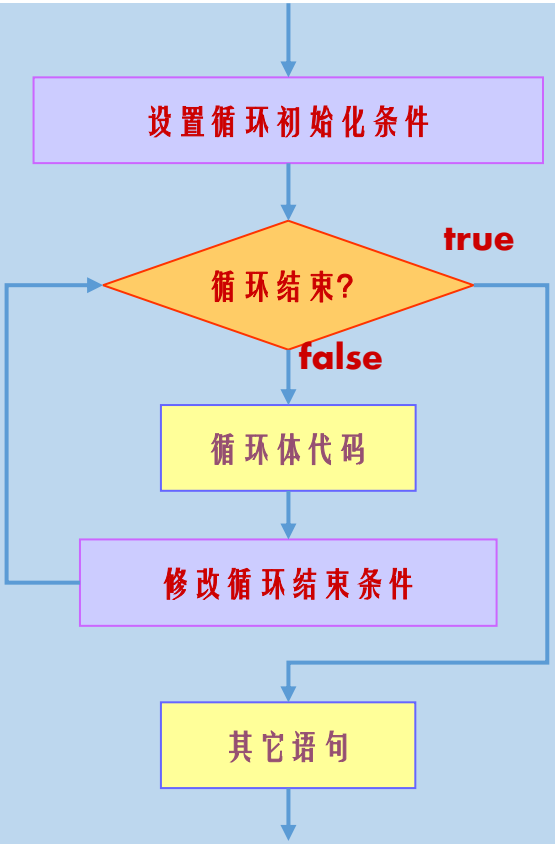


注意： 在循环体中一定要包含EXIT语句，否则程序进入死循环

程序结构和语句

结构控制语句——while…loop 语句

```
while(循环结束条件) loop

    循环执行的语句块;

    循环结束条件修改;

end loop;
```



while…loop 语句

程序结构和语句

结构控制语句示例

【例1】loop 语句

```
SET serveroutput ON
DECLARE
  numbers integer default 1;
BEGIN
LOOP
    EXIT WHEN numbers>3;
   dbms_output.put_line('hello');
   numbers:=numbers+1;
  END LOOP;
END;
```

```
匿名块已完成
hello
hello
hello
```

程序结构和语句

结构控制语句示例

【例2】while…loop 语句　求前100个自然数的和

```
SET serveroutput ON
DECLARE
  sum_i INT:= 0;
  i     INT:= 0;
BEGIN
  WHILE i<=99
  LOOP
    i:=i +1;
    sum_i:= sum_i+i;
  END LOOP;
  dbms_output.put_line('前100个自然数的和是：'||sum_i);
END;
```

```
匿名块已完成
前100个自然数的和是：5050
```

程序结构和语句

□ 结构控制语句 ——for 语句

```
for variable_ counter_name in [reverse] lower_limit..upper_limit

loop

    plsql_sentence;

end loop;
```

注意：

- 循环变量不需要显式定义，系统隐含地将它声明为BINARY_INTEGER变量；
- in确定循环变量的下界和上界，下界和上界之间是两个点".."
- 系统默认，循环变量自动从下界往上界递增计数，如果使用REVERSE关键字，则表示循环变量从上界向下界递减计数；
- 循环变量只能在循环体中使用，不能在循环体外使用。

程序结构和语句

□ 结构控制语句示例——for 语句

【例3】for 语句 求前100个自然数中偶数之和

```
SET serveroutput ON
DECLARE
  sum_i INT:= 0;    --定义整数变量，存储整数和
BEGIN
  FOR i IN reverse 1..100
  LOOP                --遍历前100个自然数
    IF mod(i,2)=0 THEN    --判断是否为偶数
      sum_i   :=sum_i+i;    --计算偶数和
    END IF;
  END LOOP;
  dbms_output.put_line('前100个自然数中偶数之和是:
'||sum_i);
END;
```

```
匿名块已完成
前100个自然数中偶数之和是: 2550
```

PL/SQL编程练习

for 语句 求前100个自然数中偶数之和

```
set serveroutput on
declare
      sum_i int:= 0;
begin
      for i in 1..100 loop
         if mod(i,2)=0 then
            sum_i:=sum_i+i;
         end if;
      end loop;
      dbms_output.put_line('前100个自然数中偶数之和是：'||sum_i);
end;
```

【练习】for 语句 求10的阶乘

```
set serveroutput on
declare

begin




end;
```

PL/SQL编程练习

for 语句 求前100个自然数中偶数之和

```
set serveroutput on
declare
      sum_i int:= 0;
begin
      for i in 1..100 loop
         if mod(i,2)=0 then
            sum_i:=sum_i+i;
         end if;
      end loop;
      dbms_output.put_line('前100个自然数中偶数之和是：'||sum_i);
end;
```

【练习】for 语句 求某数的阶

```
set serveroutput on
declare
      v_num int:= &num;
      v_pro int:=1;
begin
       if v_num =0 then
            v_pro:=1;
        else
            for i in 1.. v_num loop
                 v_pro:= v_pro*i;
            end loop;
         end if;
       dbms_output.put_line(v_pro);
end;
```

程序结构和语句

□ 结构控制语句——goto 语句

语法：

《标号》

…

GOTO 标号；

说明：

- 块内可以跳转，内层块可以跳到外层块，但外层块不能跳到内层。
- IF语句不能跳入。不能从循环体外跳入循环体内。不能从子程序外部跳到子程序中。
- 由于goto语句的缺点，建议尽量少用甚至不用goto语句。

程序结构和语句

结构控制语句示例——goto 语句

【例4】goto 语句 求前100个整数之和

```
set serveroutput on
declare
v_i number:=0;
v_s number:=0;
begin
    <<label_1>>      v_i:=v_i+1;
    if v_i<=100 then
      v_s:=v_s+v_i;
      goto label_1;
    end if;
    dbms_output.put_line(v_s);
end;
```

```
匿名块已完成
5050
```

程序结构和语句

异常

语句执行过程中，会因为各种原因导致不能正常执行，并可能造成更大错误甚至系统崩溃，所以必须采取措施防止这种情况，PL/SQL 提供了异常（Exception）处理方法。

- **预定义异常**：Oracle 能够检测出并确定类型的异常，用户可以使用预定义的异常名称进行异常处理。

预定义异常的处理，语句写在程序块的 EXCEPTION 部分。

WHEN *exception_name* THEN 处理语句

- **自定义异常**：Oracle 无法检测出或确定类型的异常，需要用户在程序中自行定义，由 Oracle 根据定义引发。

程序结构和语句

常见预定义异常

序号	异常名称	异常码	描述
1	DUP_VAL_ON_INDEX	ORA-00001	试图向唯一索引列插入重复值
2	INVALID_CURSOR	ORA-01001	试图进行非法游标操作。
3	INVALID_NUMBER	ORA-01722	试图将字符串转换为数字
4	NO_DATA_FOUND	ORA-01403	SELECT INTO 语句中没有返回任何记录。
5	TOO_MANY_ROWS	ORA-01422	SELECT INTO 语句中返回多于 1 条记录。
6	ZERO_DIVIDE	ORA-01476	试图用 0 作为除数。
7	CURSOR_ALREADY_OPEN	ORA-06511	试图打开一个已经打开的游标
8	STORAGE_ERROR	ORA-06500	运行 PL/SQL 时，超出内存空间
9	ACCESS_INTO_NULL	ORA-06530	试图访问未初始化对象的时候出现

异常-预定义异常

预定义异常的处理，语句写在程序块的 EXCEPTION 部分。当遇到预先定义的错误时，错误被相应的WHEN-THEN语句捕捉，THEN后的语句代码将执行，对错误进行处理。

【例】 处理ZERO_DIVIDE异常。

```
DECLARE
  v_zero NUMBER:=0;
  v_result NUMBER;
BEGIN
  v_result:=100/v_zero;   /*100除以v_zero，即100/0，产生除数为零异常*/
EXCEPTION                 /*异常处理部分*/
    WHEN ZERO_DIVIDE THEN  DBMS_OUTPUT.PUT_LINE('除数为0异常');
END;
```

【例】预定义异常示例

```
set serveroutput on
declare
    var_empno number;              --定义变量，存储雇员编号
    var_ename varchar2(50);                --定义变量，存储雇员名称
begin
    select empno,ename into var_empno,var_ename
    from emp
    where deptno=10;               --检索部门编号为10的雇员信息
    if sql%found then              --若检索成功，则输出雇员信息
      dbms_output.put_line('雇员编号：'||var_empno||'；雇员名称
'||var_ename);
    end if;
exception                                  --捕获异常
    when too_many_rows then   --若SELECT INTO语句的返回记录超过一行
      dbms_output.put_line('返回记录超过一行');
    when no_data_found then    --若SELECT INTO语句的返回记录为0行
      dbms_output.put_line('无数据记录');
end;
```

```
匿名块已完成
返回记录超过一行
```

预定义异常处理示例

【例】在 SCOTT 模式中，对于某个查询的异常处理。

匿名块已完成
返回记录超过一行

deptno=10改为100

匿名块已完成
无数据记录

```
SET serveroutput ON
DECLARE
  var_empno NUMBER;          --定义变量，存储雇员编号
  var_ename VARCHAR2(50);    --定义变量，存储雇员名称
BEGIN
  /* 检索部门编号为10的雇员信息 */
  SELECT empno,ename INTO var_empno,var_ename FROM emp WHERE
deptno=10;
EXCEPTION                                   --捕获异常
WHEN too_many_rows THEN  --若SELECT INTO语句的返回记录超一行
  dbms_output.put_line('返回记录超过一行');
WHEN no_data_found THEN --若SELECT INTO语句的返回记录为0行
  dbms_output.put_line('无数据记录');
END;
```

异常—自定义异常

用户可以通过自定义异常来处理错误的发生，调用异常处理需要使用RAISE语句。

语法格式：

```
DECLARE                          --程序块声明部分
   exception_name EXCEPTION;              --异常声明
   [PRAGMA  EXCEPTION_INIT(exception_name, Oracle_error_number); ]
                                 --定义错误号，与标准Oracle错误联系起来
BEGIN                             --程序块执行部分
   IF condition THEN             --异常出现条件
      RAISE exception_name;
   END IF;
EXCEPTION                        --程序块异常处理部分
   WHEN  exception_name  THEN
   处理语句
END;
```

PL/SQL编程

【例】自定义异常示例

```
set serveroutput on
declare
        e_overnum EXCEPTION;        /*定义异常处理变量*/
        v_num NUMBER;
        max_num NUMBER:=5;          /*定义最大允许学生数变量*/
begin
        SELECT COUNT(*) INTO v_num
        FROM student;
        IF max_num<v_num THEN
           RAISE e_overnum;         /*使用RAISE语句抛出用户定义异常*/
        END IF;
exception                           /*异常处理部分*/
       WHEN e_overnum THEN
       DBMS_OUTPUT.PUT_LINE('现在学生数是：'||v_num||'而最大允许数是：'||max_num );
end;
```

自定义异常处理示例

【例】在 SCOTT 模式中，自定义一个异常变量，在向 dept 表插入数据时，若 loc 字段的值为空，则引发异常。

```
DECLARE
  null_exception EXCEPTION;   --声明一个exception类型的异常变量
  dept_row dept%rowtype;      --声明rowtype类型的变量dept_row
BEGIN
  dept_row.deptno := 66;                  --给部门编号变量赋值
  dept_row.dname  := '公关部';            --给部门名称变量赋值
  /* 向dept表中插入一条记录 */
  INSERT INTO dept VALUES(dept_row.deptno,dept_row.dname,dept_row.loc);
  IF dept_row.loc IS NULL THEN  raise null_exception;  --引发null异常，转入
exception部分
  END IF;
EXCEPTION
WHEN null_exception THEN   --当raise引发的异常是null_exception时
  dbms_output.put_line('loc字段的值不许为null'); --输出异常提示信息
  ROLLBACK;                                      --回滚插入的数据记录
END;
```

匿名块已完成
loc字段的值不许为null

空操作和空值

有时，特别是使用 IF 语句时，当条件为 TRUE 时，什么工作都不做；为 FALSE 时，则执行某些操作。例如：

```
IF n<0 THEN
    NULL                    --关键字NULL表示不执行操作
ELSE
    DBMS_OUTPUT.PUT_LINE('正常');
```

空操作和空值

注意：

- 如果用等号来判断两个NULL是否相等得到的结果一定是NULL。
- 对于建立了唯一约束的列，Oracle允许插入多个NULL值，因为Oracle不认为这些NULL是相等的。
- 但是有的时候，Oracle会认为NULL是相同的，比如在GROUP BY和DISTINCT操作中，Oracle会认为所有的NULL都是一类的。
- 聚合函数一般不会处理NULL值。不管是MAX、MIN、AVG还是SUM，这些聚集函数都不会处理NULL。这里说的不会处理NULL，是指聚集函数会直接忽略NULL值记录的存在。除非是聚集函数处理的列中包含的全部记录都是NULL。

内容提纲

- PL/SQL概述
- 数据类型
- 程序结构和语句
- 游标
- 存储过程、函数、触发器
- 程序包

游标

Cursor 基本概念

游标是从数据表中提取出来的数据，以临时表的形式存放到内存中，在游标中有一个数据指针， 在初始状态下指向的是首记录，利用 fetch 语句可以移动该指针，从而对游标中的数据进行各种操作，然后将操作结果写回到数据库中。

游标的作用：

- 查询数据库，获取记录集合（结果集）的指针，可以让开发者一次访问一行结果集， 在每条结果集上作操作。
- 用‘牺牲内存’来提升 SQL 执行效率。

游标

游标分类

静态游标：在使用前，游标的定义已经完成，不能再更改。静态游标在打开时会将数据集存储在tempdb中，因此显示的数据与游标打开时的数据集保持一致，在游标打开以后对数据库的更新不会显示在游标中。

- 显示游标：用户声明和操作的一种游标，通常用于操作查询结果集。
- 隐式游标：不用明确声明的游标，执行 DML 语句、SELECT…INTO 语句会使用隐式游标。

动态游标：声明时没有设定，在打开时可以对其进行修改。动态游标在打开后会反映对数据库的更改。可实现在程序间传递结果集的功能。

- 自定义类型 ref
- 系统类型 SYS_REFCURSOR

游标

游标属性

- 通过游标的属性可以获取 SQL 的执行结果以及游标的状态信息
- 游标属性只能用在PL/SQL的流程控制语句内，而不能用在SQL语句内

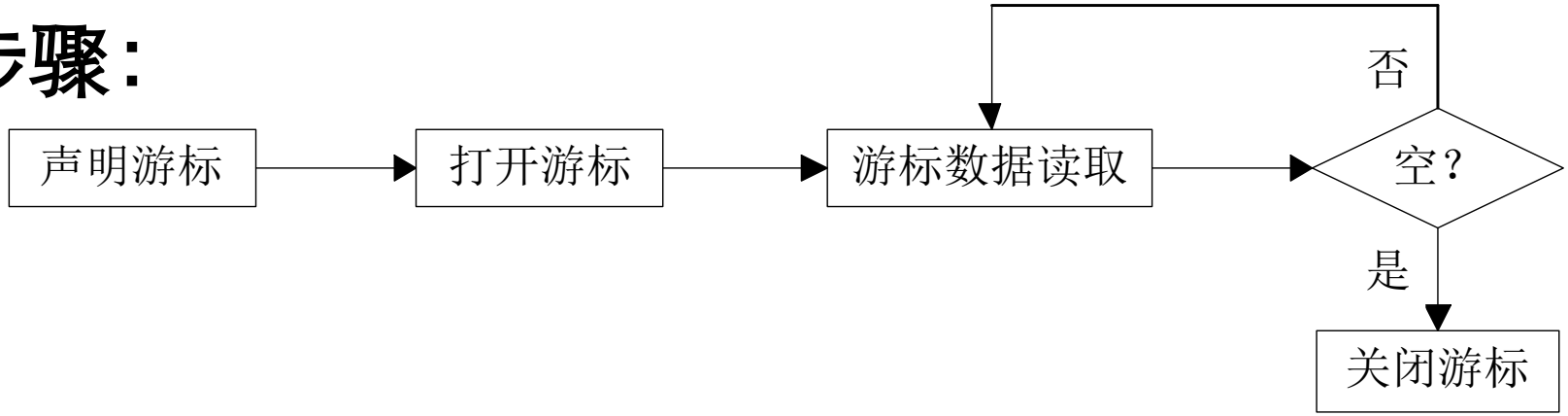
属性	返回值	描述
游标变量名%ISOPEN	布尔型	游标是否开启， true：开启，false：关闭。
游标变量名%FOUND	布尔型	游标发现数据，前一个 fetch 语句是否有值，true：有，false：没有。
游标变量名%NOTFOUND	布尔型	游标没有发现数据，常被用于退出循环，true：没有，false：有，null：空。
游标变量名%ROWCOUNT	布尔型	当前成功执行的数据行数（非总记录数）。

- 隐式游标的属性：*游标变量名* 部分为“SQL”，即四个属性分别为：SQL%ISOPEN、SQL%FOUND、SQL%NOTFOUND 和 SQL%ROWCOUNT。

游标

显式游标

步骤：



- 定义游标：定义一个游标名，以及与其相对应的SELECT 语句。
- 打开游标：就是执行游标所对应的select 语句，将其查询结果放入工作区，游标指针指向工作区的首部，标识游标结果集合。
- 读取游标：就是检索结果集合中的数据行，放入指定的输出变量中。第一次使用FETCH语句时，游标指针指向第一条记录，因此操作的对象是第一条记录，使用后，游标指针指向下一条记录
- 关闭游标：以释放该游标所占用的系统资源

游标

显式游标

1. 声明（定义）游标（在程序块 DECLARE 部分）

```
declare cursor cur_name[(parameter[, parameter]...)] is
select_ sentence;
```

2. 打开游标（在程序块 BEGIN 部分）

```
open cur_name;
```

3. 读取游标（在程序块 BEGIN 部分）

```
fetch cur_name into {变量列表};
fetch cur_name into PL/SQL 记录;
```

4. 关闭游标（在程序块 BEGIN 部分）

```
close cur_name;
```

游标

显示游标使用示例

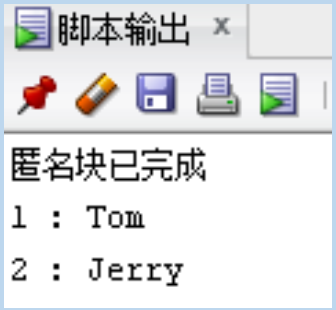
【例】在 SCOTT 模式中，创建一个 STU 表，演示游标的使用。

```
CREATE TABLE stu (
  s_id NUMBER(3),
  s_xm VARCHAR2(30)
);
ALTER TABLE stu ADD CONSTRAINT pk_stu_id PRIMARY KEY (s_id) ;
INSERT INTO stu(s_id, s_xm) VALUES (1, 'Tom');
INSERT INTO stu(s_id, s_xm) VALUES (2, 'Jerry');
```

	S_ID	S_XM
1	1	Tom
2	2	Jerry

游标 □ 显示游标使用示例

```
DECLARE
  CURSOR cur_stu IS SELECT * FROM stu;   -- 步骤1: 声明游标
  v_stu  cur_stu%ROWTYPE; -- 定义一个变量存放游标指示的内容
BEGIN
  OPEN cur_stu;                                  -- 步骤2: 打开游标
  LOOP
    FETCH cur_stu INTO v_stu;                    -- 步骤3: 提取数据
    EXIT WHEN cur_stu%NOTFOUND;
     dbms_output.put_line(v_stu.s_id ||' : '||v_stu.s_xm);
  END LOOP;
  CLOSE cur_stu;                                 -- 步骤4: 关闭游标
EXCEPTION
  WHEN OTHERS THEN
    dbms_output.put_line(SQLCODE || ' : ' || SQLERRM);
    dbms_output.put_line(dbms_utility.format_error_backtrace);
END;
```



游标 □ 参数游标使用示例

```
DECLARE
  v_name  dept.dname%TYPE;
  v_loc   dept.loc%TYPE;
  CURSOR c1(v_deptno NUMBER DEFAULT 10) IS
    SELECT dname, loc FROM dept
    WHERE deptno<= v_deptno;
 BEGIN
    OPEN c1;
    LOOP
       FETCH c1 INTO v_dname, v_loc;
       EXIT WHEN c1%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(v_dname||'---'||v_loc);
    END LOOP;
    CLOSE c1;
 END;
```

游标

隐式游标

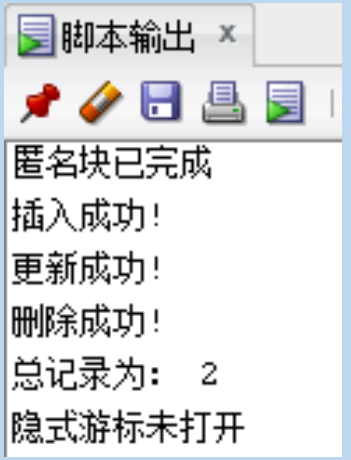
对于隐式游标的操作，如定义、打开、取值及关闭操作，都由 ORACLE系统自动地完成，无需用户进行处理。用户只能通过隐式游标的相关属性，来完成相应的操作。在隐式游标的工作区中，所存放的数据是与用户自定义的显示游标无关的、最新处理的一条 SQL 语句所包含的数据。

DML操作和单行SELECT语句会使用隐式游标：

* 插入操作： INSERT
* 更新操作： UPDATE
* 删除操作： DELETE
* 单行查询操作： SELECT ... INTO

□ 隐式游标使用示例

```
DECLARE
  v_count NUMBER;
BEGIN
  INSERT INTO stu(s_id, s_xm) VALUES(3, 'Micky');
  IF SQL%FOUND THEN dbms_output.put_line('插入成功!');  END IF;
  UPDATE stu SET stu.s_xm = 'Donald' WHERE stu.s_id = 3;
  IF SQL%FOUND THEN dbms_output.put_line('更新成功!');  END IF;
  DELETE FROM stu t WHERE t.s_id = 3;
  IF SQL%FOUND THEN dbms_output.put_line('删除成功!');  END IF;
  SELECT COUNT(*) INTO v_count FROM stu t;
  IF SQL%FOUND THEN dbms_output.put_line('总记录为：  '||v_count);
END IF;
  IF SQL%ISOPEN THEN    --对于隐式游标而言永远为FALSE
    dbms_output.put_line('隐式游标已打开');
  ELSE
    dbms_output.put_line('隐式游标未打开');
  END IF;
EXCEPTION   略
END;
```



游标

动态游标

- 如同常量和变量的区别一样，前面所讲的游标都是与一个 SQL 语句相关联，并且在编译该块的时候此语句已经是可知的，是静态的，而游标变量可以在运行时与不同的语句关联，是动态的。游标变量被用于处理多行的查询结果集。在同一个PL\SQL块中，游标变量不同于特定的查询绑定，而是在打开游标时才确定所对应的查询。因此，游标变量可以一次对应多个查询。
- 使用游标变量之前，必须先声明，然后在运行时必须为其分配存储空间，游标变量是 REF 类型的变量。

游标

动态游标

1. 声明（定义）游标变量

```
TYPE <游标变量> IS REF CURSOR
RETURN <返回类型>;
```

2. 打开游标变量

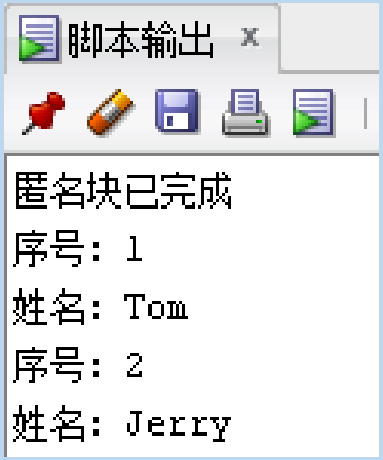
```
OPEN <游标变量> FOR <SELECT语句>;
```

3. 关闭游标（在程序块 BEGIN 部分）

```
CLOSE <游标变量>;
```

游标 □ 自定义类型 ref 游标使用示例

```
SET serveroutput ON
DECLARE
  v_stu stu%ROWTYPE;
  TYPE cur_stu_type IS REF CURSOR RETURN stu%ROWTYPE;
  cur_stu cur_stu_type;
BEGIN
  OPEN cur_stu FOR SELECT s_id, s_xm FROM stu;
  LOOP
    FETCH cur_stu
      INTO v_stu; --v_id, v_xm;
    EXIT WHEN cur_stu%NOTFOUND;
    dbms_output.put_line('序号： ' || v_stu.s_id || chr(10) || '姓名： ' ||
v_stu.s_xm);
  END LOOP;
  CLOSE cur_stu;
EXCEPTION
  WHEN OTHERS THEN
    dbms_output.put_line(SQLCODE || ' : ' || SQLERRM);
    dbms_output.put_line(dbms_utility.format_error_backtrace);
END;
```



内容提纲

- PL/SQL概述
- 数据类型
- 程序结构和语句
- 游标
- 存储过程、函数、触发器
- 程序包

存储过程、函数、触发器

存储过程

存储过程是一种命名的 PL/SQL 程序块，用于完成特定的数据库功能，该程序块经过编译后存储在数据库系统中。在使用的时候，用户通过指定已经定义的存储过程名称并给出相应的存储过程参数来调用并执行它，从而完成数据库操作。

参数[IN|OUT|IN OUT] 参数类型（默认为in）

```
create [ or replace ] procedure pro_name [ ( parameter1
[ ,parameter2 ]... ) ] is|as [inner_variable]
begin
   plsql_sentences;
[exception]
   [dowith _ sentences;]
end [pro_name];
```

内部变量

存储过程执行特定操作，不需要返回值

说明：

（1）OR REPLACE：如果指定的过程已存在，则覆盖同名的存储过程。

（2）参数名：存储过程的参数名必须符合有关标识符的规则，存储过程中的参数称为形式参数(简称形参)，可以声明一个或多个形参，调用带参数的存储过程则应提供相应的实际参数(简称实参)。

（3）参数类型：存储过程的参数类型有IN、OUT和IN OUT 三种模式，默认的模式是IN模式。

- IN：向存储过程传递参数，只能将实参的值传递给形参，在存储过程内部只能读不能写，对应IN模式的实参可以是常量或变量。
- OUT：从存储过程输出参数，存储过程结束时形参的值会赋给实参，在存储过程内部可以读或写，对应OUT模式的实参必须是变量。
- IN OUT：具有前面两种模式的特性，调用时，实参的值传递给形参，结束时，形参的值传递给实参，对应IN OUT模式的实参必须是变量。

（4）DEFAULT：指定IN参数的默认值，默认值必须是常量。

存储过程、函数、触发器

存储过程

- 调用过程：调用过程一般使用 EXECUTE 语句，但在 PL/SQL 块中可以直接使用过程的名称来调用。

```
[exec|execute]  pro_name[parameter_list]
```

- 修改过程：使用 CREATE OR REPLACE PROCEDURE 命令，修改已有存储过程的本质就是重新创建一个新的过程，使用原来的名字。
- 删除过程

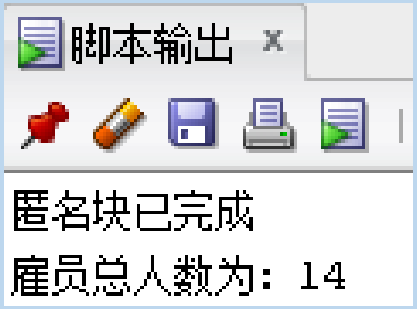
```
drop procedure pro_name
```

存储过程、函数、触发器

存储过程示例

【例】在 SCOTT 模式中，创建一个显示雇员总人数的存储过程。

```
--创建过程
CREATE OR REPLACE PROCEDURE EMP_COUNT AS
  V_TOTAL NUMBER(10);
BEGIN
  SELECT COUNT(*) INTO V_TOTAL FROM EMP;
  DBMS_OUTPUT.PUT_LINE('雇员总人数为：'||V_TOTAL);
END;

--执行过程
SET serveroutput ON
EXECUTE EMP_COUNT;
```



存储过程、函数、触发器

函数

函数一般用于计算和返回一个值，函数与存储过程在创建形式上有些相似，也是编译后放在内存供用户使用，但是函数的调用是表达式的一部分，函数必须要有返回值。Oracle 自带函数包括如下几类：

- 字符类函数：ASCII、CHR、CONCAT、INSTR、LENGTH 等；
- 数学类函数：绝对值、三角函数、幂函数等；
- 日期类函数：ADD_MONTH、NEW_TIME、SYSDATE等；
- 转换类函数：用于数据类型转换；
- 聚集类函数：求和、均值、最值、方差等。

存储过程、函数、触发器

函数

- 创建函数：函数必须有返回值。

```
create [or replace] function fun_name [ (parameter1
[ ,parameter2]… ) return data_type is [inner_variable]
begin
   plsql_ sentence;
[exception]
   [dowith _ sentences;]
end [fun_name];
```

返回值类型

- 调用函数：由于函数有返回值，所以在调用函数时，必须使用一个变量来保存函数的返回值，这样函数和这个变量就组成了一个赋值表达式。
- 删除函数：

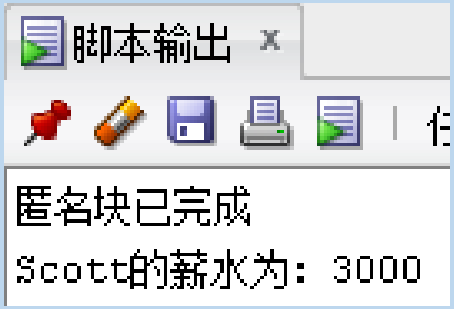
```
drop function fun_name
```

存储过程、函数、触发器

函数示例

【例】在 SCOTT 模式中，创建一个获取雇员工资的函数。

```
--创建函数
CREATE OR REPLACE FUNCTION get_sal(empname IN VARCHAR2)
RETURN NUMBER IS
   Result NUMBER;
BEGIN
 SELECT sal INTO Result FROM emp WHERE ename=empname;
 RETURN(Result);
END get_sal;

--调用函数
BEGIN
DBMS_OUTPUT.PUT_LINE('Scott的薪水为：'||get_sal('SCOTT'));
END;
```



存储过程、函数、触发器

触发器

触发器是存储在服务器中的程序单元，当数据库中某些事件发生时，数据库自动启动触发器，执行触发器中的相应操作。触发器不能接受参数，不能被显式调用，常用于加强数据的完整性约束和业务规则等。

触发器是一种特殊的存储过程，与表的关系密切，其特殊性主要体现在不需要用户调用，而是在对特定表（或列）进行特定类型的数据修改时激发。

存储过程、函数、触发器

触发器

触发器与存储过程的差别：

- 触发器是自动执行，而存储过程需要显式调用才能执行。
- 触发器是建立在表或视图之上的，而存储过程是建立在数据库之上的。

触发器用于实现数据库的完整性，触发器具有以下优点：

- 可以提供比CHECK 约束、FOREIGN KEY约束更灵活、更复杂、更强大的约束。
- 可对数据库中的相关表实现级联更改。
- 可以评估数据修改前后表的状态，并根据该差异采取措施。
- 强制表的修改要合乎业务规则。

触发器的缺点是增加决策和维护的复杂程度。

存储过程、函数、触发器

触发器

Oracle的触发器有三类：DML 触发器、INSTEAD OF触发器和系统触发器。

- **DML 触发器**：在 DML 操作前或操作后进行触发，并且可以在每个行或语句操作上进行触发。DML 触发器可分为INSERT触发器、UPDATE 触发器和 DELETE触发器三类。
- **替代触发器**：在 ORACLE 里，不能直接对由两个以上的表建立的视图进行操作。所以给出了替代触发器，为进行视图操作提供处理方法。
- **系统触发器**：系统触发器由数据定义语言(DDL)事件(如CREATE语句、ALTER语句、DROP语句)、数据库系统事件(如系统启动或退出、异常操作)、用户事件 (如用户登录或退出数据库)触发。

存储过程、函数、触发器

触发器

- 触发事件：引起触发器被触发的事件。 例如DML语句、DDL语句、数据库系统事件、用户事件等。
- 触发时间：触发事件和触发器的操作顺序，BEFORE 或 AFTER。
- 触发操作：触发器本身要做的事情。
- 触发对象：包括表、视图、模式、数据库。只有在这些对象上发生了符合触发条件的触发事件，才会执行触发操作。
- 触发条件：由 WHEN 子句指定一个逻辑表达式。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。
- 触发频率：说明触发器内定义的动作被执行的次数。即语句级触发器和行级触发器。

存储过程、函数、触发器

创建、启用和删除触发器

- 创建触发器

```
create  [or replace]  trigger  tri_name
[ before | after | instead of ] tri_event
 on table_name | view_name | user_name |db_name
    [ for each row [ when tri_condition ]]
begin
    plsql_sentences;
end  tri_name;
```

触发时间

触发事件
DML、DDL等

触发对象

触发条件

说明：

- 触发器名：指定触发器名称。
- BEFORE：执行DML操作之前触发。
- AFTER：执行DML操作之后触发。
- INSTEAD OF：替代触发器，触发时触发器指定的事件不执行，而执行触发器本身的操作。
- DELETE、INSERT、UPDATE：指定一个或多个触发事件，多个触发事件之间用OR连接。
- FOR EACH ROW：由于DML语句可能作用于多行，因此触发器的PL/SQL语句可能为作用的每一行运行一次，这样的触发器称为行级触发器(row-level trigger)；也可能为所有行只运行一次，这样的触发器称为语句级触发器(statement-level trigger)。如果未使用FOR EACH ROW子句，指定为语句级触发器，触发器激活后只执行一次。如果使用FOR EACH ROW子句，指定为行级触发器，触发器将针对每一行执行一次。WHEN子句用于指定触发条件。

存储过程、函数、触发器

创建、启用和删除触发器

- 禁用/启用触发器

```
ALTER  TRIGGER  [schema.] tri_name  DISABLE | ENABLE;
```

- 删除触发器

```
DROP TRIGGER tri_name;
```

存储过程、函数、触发器

触发器示例

【例】在 SCOTT 模式中，每次向 DEPT 表插入数据后，给出提示。

```
--创建触发器
CREATE OR REPLACE TRIGGER REM_DEPT
 AFTER INSERT ON DEPT
 FOR EACH ROW    --对表的每一行触发器执行一次
BEGIN
  DBMS_OUTPUT.PUT_LINE('您向DEPT表中插入了一条新数据！ ');
END;

--测试触发器
INSERT INTO DEPT (DEPTNO,DNAME,LOC) VALUES(05,'HR','BEIJING');
INSERT INTO DEPT (DEPTNO,DNAME,LOC)
VALUES(06,'MARKET','SHANGHAI');
INSERT INTO DEPT (DEPTNO,DNAME,LOC)
VALUES(07,'COMPANY','LONDON');
```

```
1 行已插入。
您向DEPT表中插入了一条新数据!

1 行已插入。
您向DEPT表中插入了一条新数据!

1 行已插入。
您向DEPT表中插入了一条新数据!
```

存储过程、函数、触发器

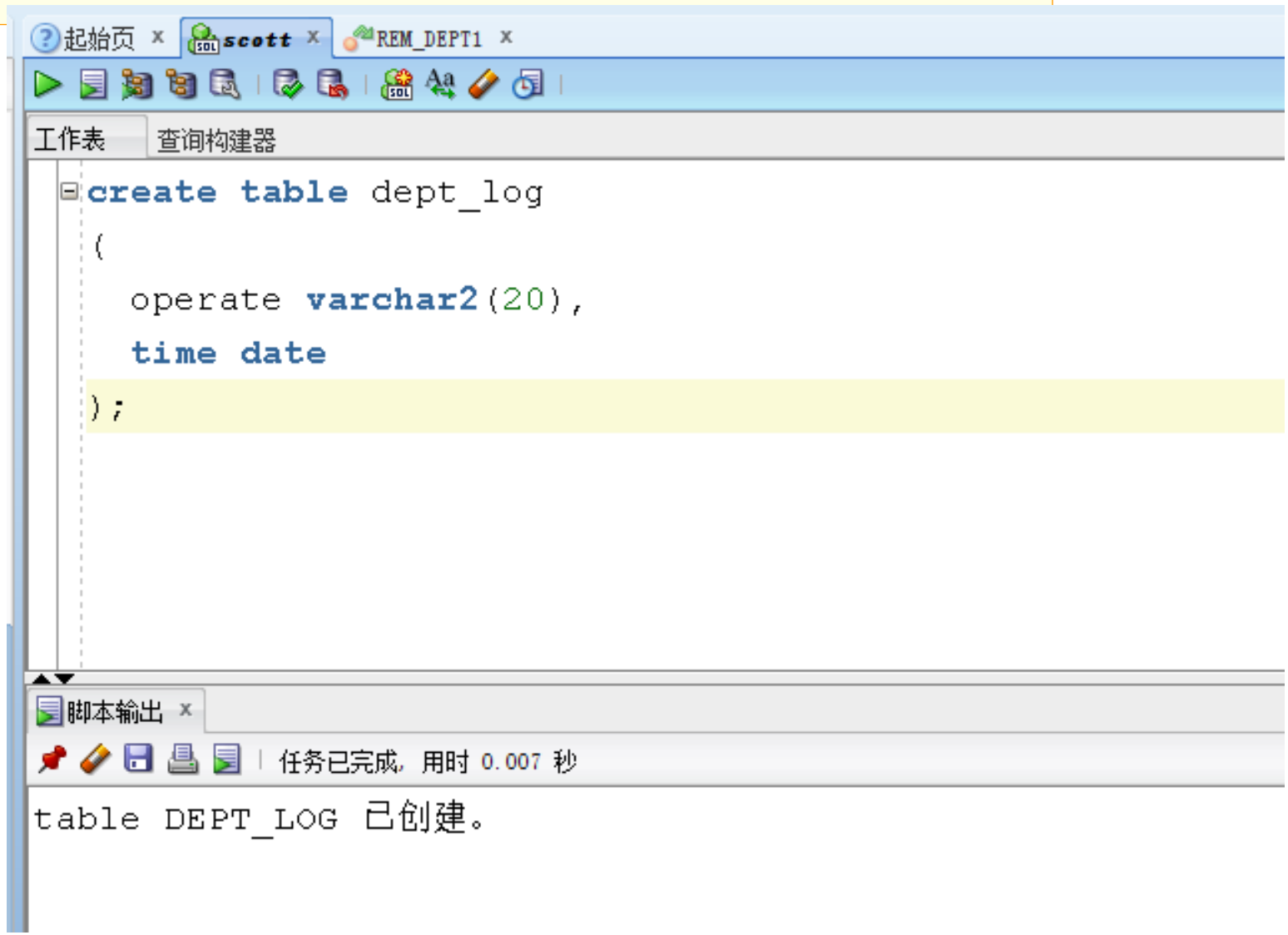
触发器示例

【例】为emp表创建一个触发器，增加只能上班时间进行DML操作的限制

```
CREATE[OR REPLACE] TRIGGER my_trigger
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 18) THEN
  RAISE_APPLICATION_ERROR(-20001,'不是上班时间,不能修改emp
表');
 END IF;
END;
```

试一试

创建一个触发器tri_dept，该触发器在insert、update和delete事件下都可以被触发，并且操作的数据对象是dept表。然后要求在触发器执行时输出对dept表所做的具体操作。



试一试

起始页 | scott | REM_DEPT1

0.031 秒

工作表 | 查询构建器

```
create trigger tri_dept before insert or delete or update on dept
declare
  var_tag varchar2(20);
begin
  if inserting then var_tag:='插入';
  elsif updating then var_tag:='修改';
  elsif deleting then var_tag:='删除';
  end if;
  insert into dept_log values (var_tag,sysdate);
end tri_dept;
```

脚本输出

任务已完成，用时 0.031 秒

```
table DEPT_LOG 已创建。
TRIGGER TRI_DEPT 已编译
```

试一试

起始页 × scott ×

0.031 秒

工作表 查询构建器

```
insert into dept values(0,'HR','shi');
update dept set loc='beijing' where deptno=0;
delete from dept where deptno=0;
select * from dept_log;
```

脚本输出 ×

任务已完成，用时 0.031 秒

```
1 行已插入。
1 行已更新。
1 行已删除。
OPERATE                TIME
---------------------- ---------
插入                   19-11月-19
修改                   19-11月-19
删除                   19-11月-19
```

内容提纲

- 数据类型
- 程序结构和语句
- 游标
- 存储过程、函数、触发器
- 程序包

程序包

程序包

包是一组相关过程、函数、变量、常量和游标等 PL/SQL 程序设计元素的组合，它具有面向对象程序设计语言的特点，是对这些 PL/SQL 程序设计元素的封装。包类似于 C++ 和 JAVA 语言中的类。

- 说明部分：包与应用程序之间的接口， 只是过程、函数、游标等的名称或首部，其中过程和函数只包括原型信息，不包含任何子程序代码。
- 包体部分：过程、函数、游标等的具体实现。包体中还可以包括在规范中没有声明的变量、游标、类型、异常、过程和函数，但是它们是私有元素，只能由同一包体中其他过程和函数使用。

程序包

创建包

说明部分：

```
CREATE [OR REPLACE] PACKAGE package_name
IS
   变量、常量及数据类型定义;
   游标声明;
   函数、过程声明
END [package_name];
```

包体部分：

```
CREATE [OR REPLACE] PACKAGE BODY package_name
AS
   游标、函数、过程的具体定义
END [package_name];
```

程序包

调用和删除包

- 调用包

```
package_name.变量（常量）名
package_name.游标名
package_name.函数（过程名）
```

- 删除包

```
DROP PACKAGE package_name;
```

课后作业