大型数据库应用技术

PL/SQL

授课教师：王欢

内容提纲

- 复习
- 存储过程
- 函数
- 触发器
- 程序包

复习

□ 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

复习

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。

复习

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 ];
```

复习

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

复习

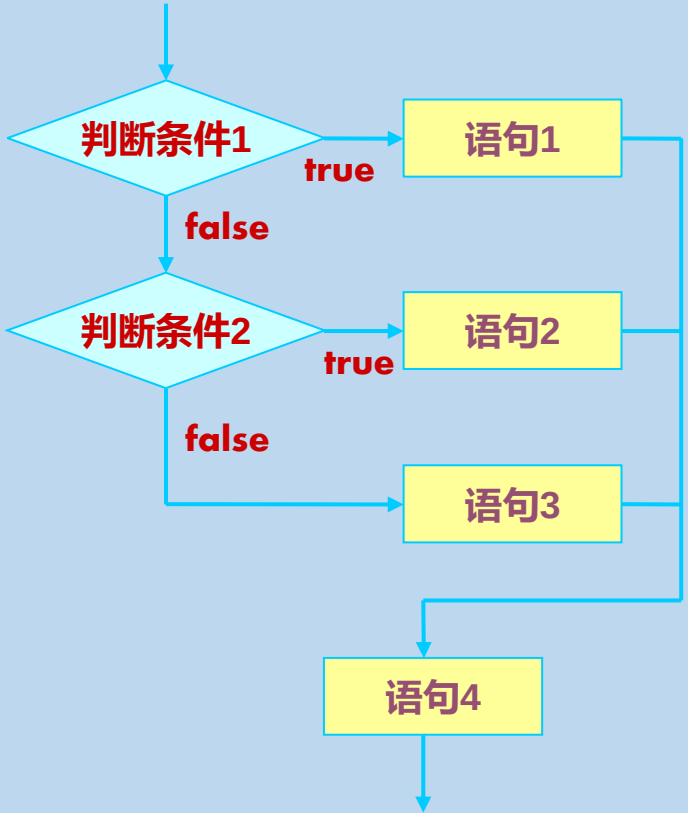
结构控制语句

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

复习

结构控制语句——if…then…elsif 语句

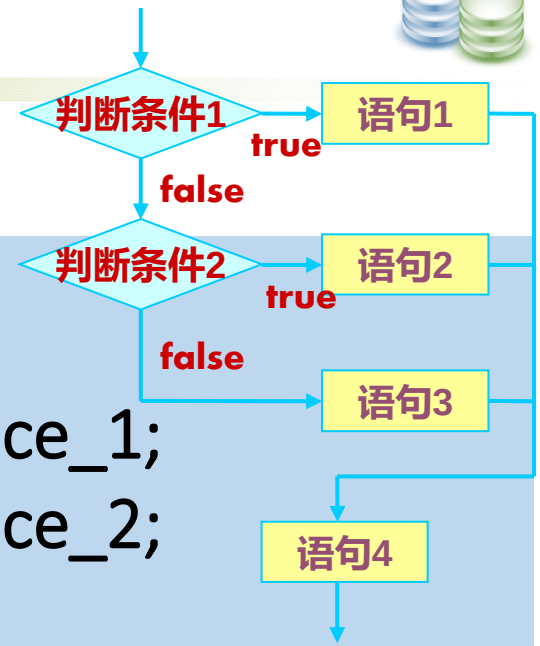
```
if < condition_expression1 > then
plsql_sentence_1;
elsif < condition_expression2 > then
plsql_sentence_2;
…
else
plsql_sentence_n;
end if;
```



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。

复习

结构控制语句——case 语句



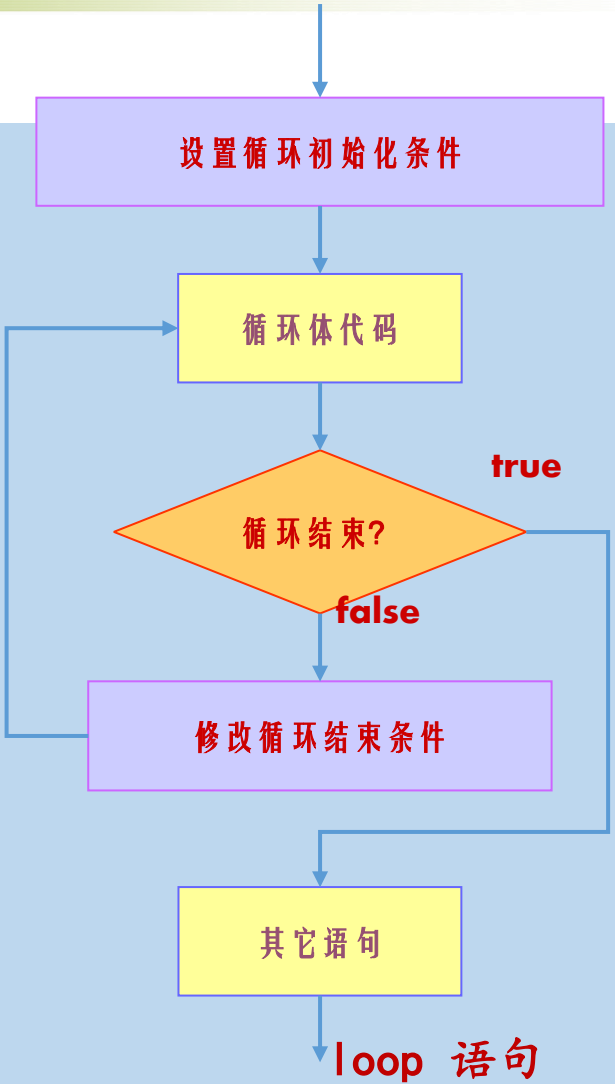

```
case < selector>
    when <expression_1> then plsql_sentence_1;
    when <expression_2> then plsql_sentence_2;
    …
    when <expression_n> then plsql_sentence_n;
    [else plsql_sentence;]
end case;
```

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

复习

□ 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

复习

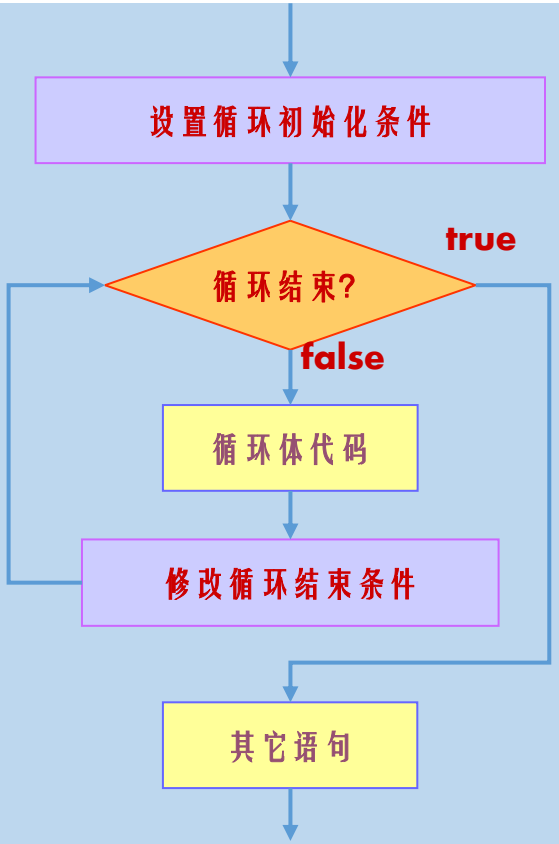
结构控制语句——while…loop 语句

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

复习

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。

复习

□ 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。

复习

异常

程序执行过程中产生的错误情况称为异常。

- **预定义异常**：Oracle 能够检测出并确定类型的异常，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。

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。

复习

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

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;
```

复习

□ 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。

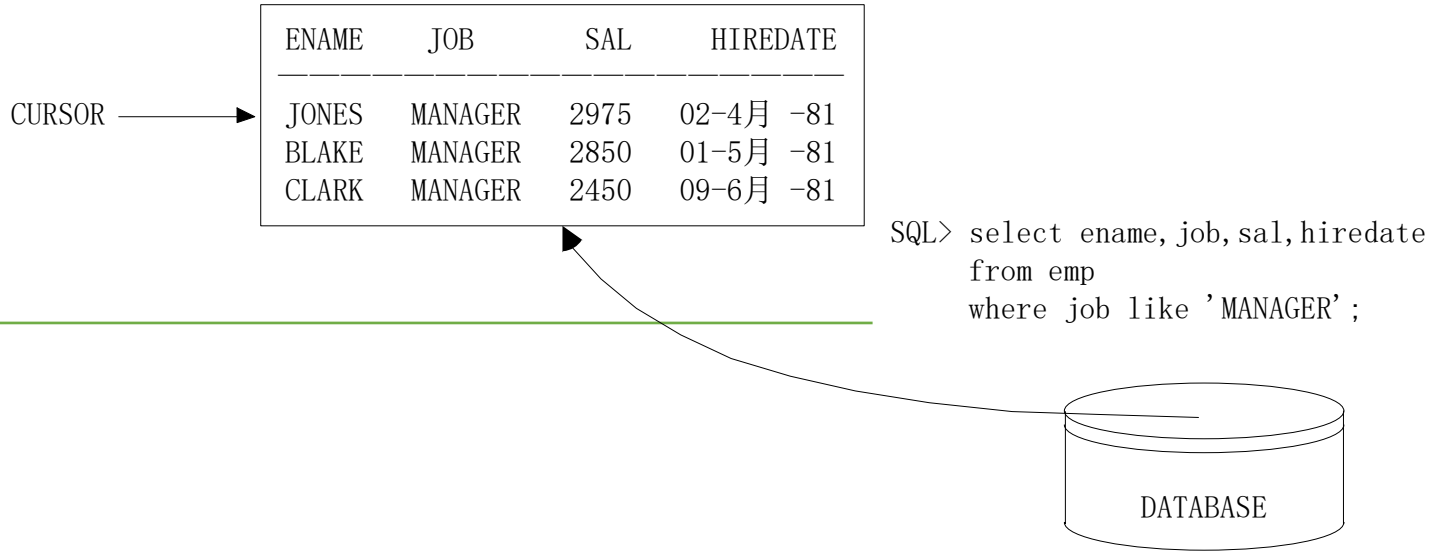
语法格式：

```
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;
```

游标 □ Cursor 基本概念

在使用SQL的SELECT语句查询数据时，往往会返回一组记录的集合，为了依次处理记录集合中的每条记录，Oracle使用游标来完成遍历每个记录的功能。其实，游标可以看作指向记录集合的指针，它可以在集合记录中移动，以访问每条记录。

例如用户选择表EMP中的JOB为MANAGER的记录，而该查询结果有三条记录，通过使用LOOP循环语句，结合使用游标就可以遍历整个查询记录集合。



游标

游标定义

游标——是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。游标分为显式游标和隐式游标两种：

- **隐式游标**：每次执行一个SQL DML语句或者SELECT INTO语句，后者直接把数据库中的一行返回到一个PL/SQL数据结构中，PL/SQL都会声明和使用一个隐式游标。这种游标之所以称为“隐式”是因为绝大部分游标相关操作都是由数据库自动做的，比如分配一个游标、打开游标、取出记录，甚至包括关闭游标。
- **显式游标**：这种游标是在应用程序中明确的把一个SELECT语句声明成一个游标，还要明确地执行各种游标操作（打开、提取、关闭等），当要使用静态SQL语句从数据源提取多行数据时，一般都会使用显式游标。

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

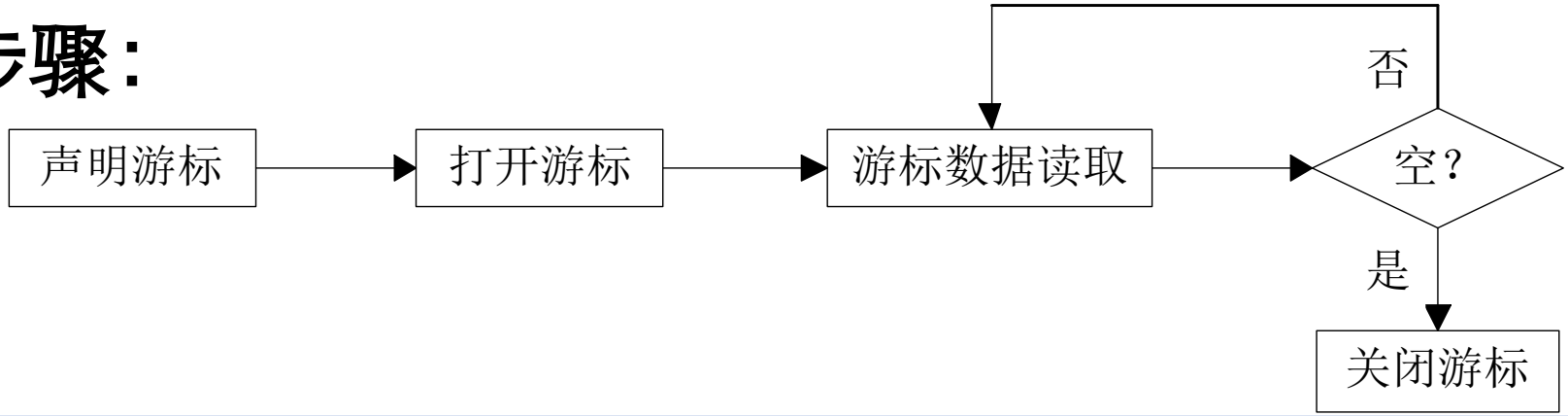
属性	返回值	描述
游标变量名%ISOPEN	布尔型	游标是否开启， true：开启，false：关闭。
游标变量名%FOUND	布尔型	游标发现数据，前一个 fetch 语句是否有值，true：有，false：没有。
游标变量名%NOTFOUND	布尔型	游标没有发现数据，常被用于退出循环，true：没有，false：有，null：空。
游标变量名%ROWCOUNT	布尔型	当前成功执行的数据行数（非总记录数）。

- 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）检索结果集合中的数据行，放入指定的输出变量中

```
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

游标定义

使用显式游标时，需注意以下事项：

- 在使用游标过程中，每次都要用%FOUND和%NOTFOUND属性检查是否返回成功，即是否还有要操作的行;
- 在游标中的行存放到变量组中时，对应变量个数和数据类型必须完全一致;
- 使用完游标必须将其关闭，以释放相应内存资源。

内容提纲

- 复习
- 存储过程
- 函数
- 触发器
- 程序包

存储过程

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。

说明：

（4）DEFAULT：指定IN参数的默认值，默认值必须是常量。

（5）关键字IS和AS本身没有区别，选择其中一个即可。

（6）IS后面是一个完整的PL/SQL程序块的三个部分，可以定义局部变量、游标等，但不能以DECLARE开始。

存储过程

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
```

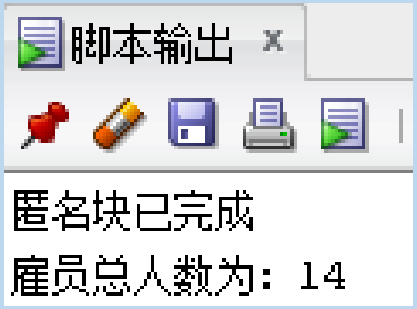
存储过程

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;
```



函数

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有一系列的内置函数，主要分为以下几类：

- 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；
- 聚合类函数：求和、均值、最值、方差等。

函数

函数-自定义函数

- 创建函数： 函数必须有返回值。

[参数[IN] 参数类型]

```
create [or replace] function fun_name [ (parameter1
[ ,parameter2]... ) return data_type is|as [inner_variable]
begin
    plsql_ sentence;          --其中必须要有一个RETURN子句
[exception]
    [dowith _ sentences;]
end [fun_name];
```

返回值类型

说明：

（1）OR REPLACE：如果指定的函数已存在，则覆盖同名的函数。

（2）与存储过程相似，创建函数时，可以定义零个或多个形式参数，并且都为IN模式，IN可以省略不写。

（3）函数是靠RETURN语句返回结果，并且只能返回一个结果。

（4）在函数定义的头部，参数列表之后，必须包含一个RETURN语句来指明函数返回值的类型，但不能约束返回值的长度、精度等。

（5）在函数体的定义中，必须至少包含一个RETURN语句，来指明函数的返回值。

函数

函数-自定义函数

- 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。调用函数时可以用全局变量接收其返回值，可以在程序块中调用它，也可以在SQL语句中调用函数。
- 删除函数：

```
drop function fun_name
```

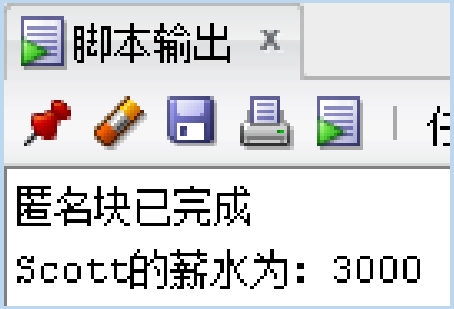
函数

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;
```



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。

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。

- 触发器的缺点是增加决策和维护的复杂程度。

触发器

触发器分类

- **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（INSTEAD OF）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。

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。

触发器

创建、启用和删除触发器

- 创建触发器语法：

```
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

触发级别

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

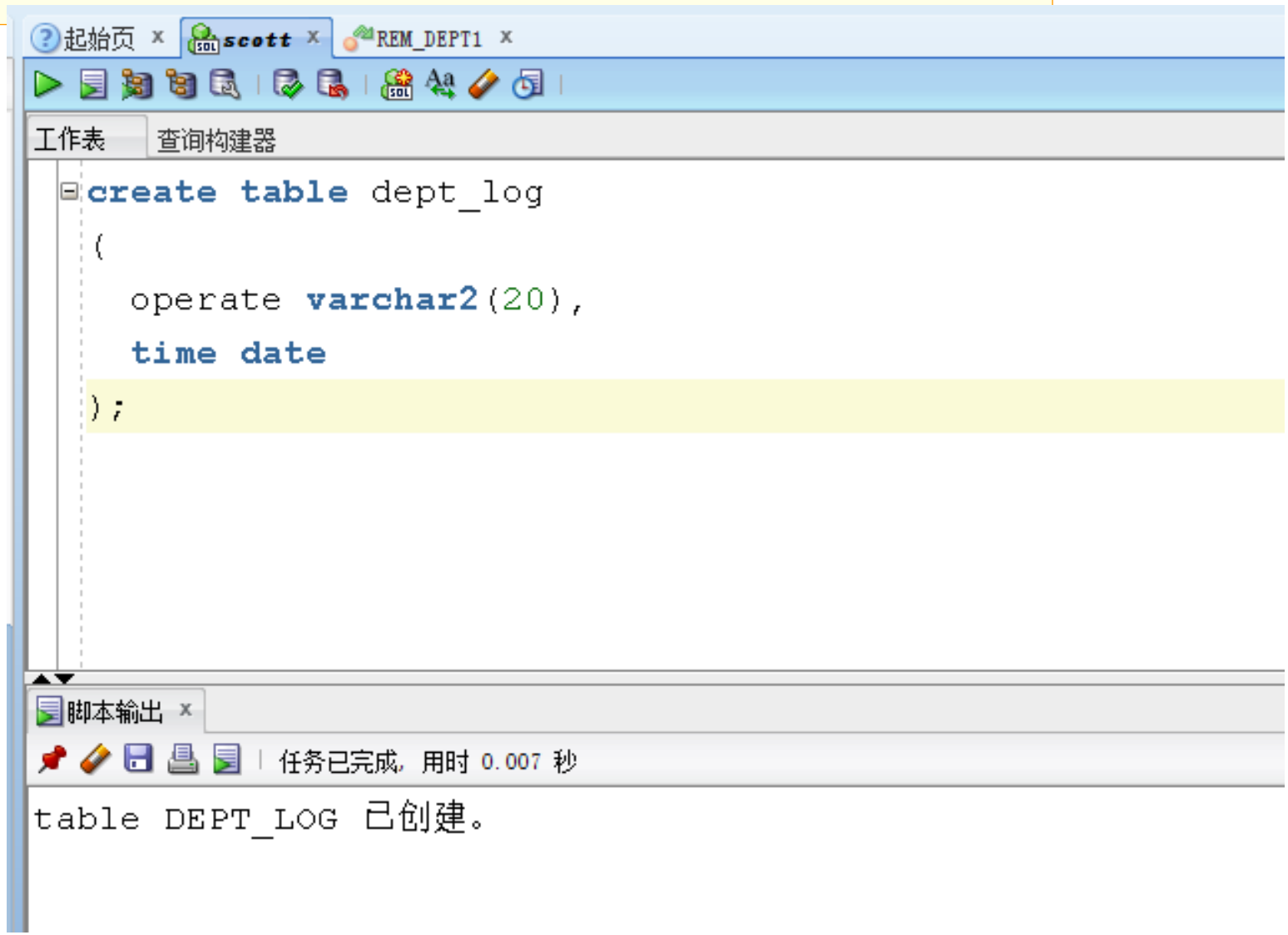
触发器示例

【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