高级PL/SQL

-过程、函数、程序包及触发器

单世民

简介

- 用户可以命名自己编写的程序块，并将其存储起来，以便以后使用。这些命名的PL/SQL程序块称为存储过程和函数，他们的集合称为程序包。

过程与函数

- 过程
  从本质上来看，过程就是命名的PL/SQL程序块，它可以被赋予参数并存储在数据库中，然后由另外一个应用或PL/SQL例程调用。

- 函数
  函数与过程非常类似，也是数据库中存储的命名PL/SQL程序块。创建函数与创建过程要都遵循同样的规则。函数与过程的安全方式和参数传递也相同。函数的主要特征是它必须返回一个值。这个返回值既可以是number或varchar2这样简单的数据类型，也可以是PL/SQL数组或对象这样复杂的数据类型。

过程

- 基本语法
  - 创建

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN|OUT|IN OUT] type [,...])]
{IS|AS}
[variable type [,...]]
BEGIN
  procedure_body
END procedure_name;
```

  - 删除

```
DROP PROCEDURE procedure_name;
```

函数

- 基本语法
  - 创建

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN|OUT|IN OUT] type [,...])]
RETURN type
{IS|AS}
[variable type [,...]]
BEGIN
  function_body
END function_name;
```

  - 删除

```
DROP FUNCTION function_name;
```

过程和函数

- 调用过程
  可以使用EXECUTE指令调用过程，也可以在PL/SQL代码块中直接调用过程。
- 调用函数
  函数不能直接调用，只能作为表达式的一部分进行调用（PL/SQL中，函数总是要返回一个值，调用者必须接受这个值，否则就会出现错误）

在不带参数时，声明和调用过程或函数都不需要使用圆括弧

过程和函数

- 过程和函数可以统称为子程序，它们对参数的使用方式是一致的。
- 子程序的形式参数可有3种模式：

模式	说明
IN	默认模式，在过程内部，形参就像常量一样：只读，不可修改，
OUT	调用过程时，实参的值被忽略，在过程内部，形参就像未初始化的变量一样，会有一个NULL值，可读可写，过程执行完毕后，形参的值被赋给实参
IN OUT	IN 和OUT的复合模式

过程和函数

- 关于参数的进一步说明
  - 与OUT或IN OUT模式的形参相关联的实参必须是一个变量，不能是字面值或常量。必须有一个可以存储返回值的位置
  - PL/SQL编译器会在编译时检查IN模式的形参是否被修改过，如果发生修改，则产生编译警告。
  - 作为参数传递机制的一部分，调用子程序时也会传递变量上的约束。因此，**在子程序声明中，约束CHAR和VARCHAR2类型参数的长度或NUMBER类型参数的长度或精度都是不合法的，因为约束是从实际参数中带来的。**

过程和函数

- 关于参数的进一步说明
  - 可以使用%TYPE对形参进行一定的约束。如果形参是使用%TYPE声明的，而且底层的类型是受到约束的，那么这种约束就会作用到形参上，而不是作用到实参上。
  - 可以分别使用或混合使用位置表示法和名称表示法名称表示法示例：

```
BEGIN
  print_dept(did=>10);
END;
/
```

过程和函数

- 过程和函数的比较
  - 相同的功能特性
    - 都可以使用OUT参数返回多个值
    - 都可以由声明、执行和异常处理三个部分组成
    - 都可以接受默认值
    - 都可以使用位置表示法和名称表示法进行调用
  - 何时使用过程，何时使用函数？
    一般的原则是：如果返回多个值，使用过程。如果只有一个返回值，就使用函数。

过程和函数

- 练习：写一个过程用来打印指定编号的部门的部门名称，即输入参数是部门编号，打印输出部门名称。

如果变为NUMBER(38)，结果将会怎样？

```
CREATE OR REPLACE PROCEDURE
    print_deptname(p_did IN NUMBER)
IS
    v_dname varchar2(30);
BEGIN
    SELECT dname INTO v_dname FROM dept
       WHERE deptno=p_did;
    dbms_output.put_line(v_dname);
END print_deptname;
/
```

如果给p_did赋值，结果将会怎样

如果变为print_dept，结果将会怎样?去掉呢？

显示错误

- 查看出现的错误,可以使用SHOW ERRORS命令，这只适用于当前只编译了一个存储过程的情况。

```
SHOW ERRORS
```

- 在编译了多个存储过程或函数的情况下，可以把SHOW ERRORS更为细化。

```
SHOW ERRORS PROCEDURE proc_name;
```

还可以有的选项是FUNCTION、PACKAGE、TRIGGER，分别对应函数，包，触发器

过程和函数

- 练习：参照前述的过程print_deptname，实现另一个过程getdeptname使其能够根据部门号将部门名称以变量的形式输出，而不是直接打印输出，如何实现？

```
CREATE OR REPLACE PROCEDURE
 getdeptname(p_did IN NUMBER, p_dname OUT VARCHAR2)
IS
BEGIN
    SELECT dname INTO p_dname FROM dept
       WHERE deptno=p_did;
END getdeptname;
/
```

过程和函数

- 调用getdeptname过程的方法如下：

```
DECLARE
    did number(38);
    dname varchar2(30);
BEGIN
    did:=10;
    dname:='haha';
    getDeptName(did,dname);
    dbms_output.put_line(dname);
END;
/
```

过程和函数

- 练习：实现一个过程getFullDeptName，只使用一个参数p_dname，以得到部门名称中包含指定字符串的部门名称全名（现假设仅有一个部门与输入的字符串匹配），如何实现？

```
CREATE OR REPLACE PROCEDURE
  getFullDeptName(p_dname IN OUT VARCHAR2)
IS
    v_oldname varchar2(30);
BEGIN
     IF p_dname IS NULL THEN
        dbms_output.put_line('p_dname is null');
        RETURN;
     END IF;
     v_oldname:=p_dname;
     SELECT dname INTO p_dname FROM dept
        WHERE dname like '%'||p_dname||'%';
EXCEPTION
     WHEN TOO_MANY_ROWS THEN
        dbms_output.put_line('不止一个部门包含字符'''||
       v_oldname||'''');
END getFullDeptName;
/
```

过程和函数

- 练习：将过程getFullDeptName“改造为”函数func_getFullDeptName，令其能够返回执行成功与否的信息，如何是实现？

```
CREATE OR REPLACE FUNCTION
     func_getFullDeptName(p_dname IN OUT VARCHAR2)
     RETURN BOOLEAN
IS
     v_oldname varchar2(30);
BEGIN
     IF p_dname IS NULL THEN
        dbms_output.put_line('p_dname is null');
        RETURN FALSE;
     END IF;
     v_oldname:=p_dname;
     SELECT dname INTO p_dname FROM dept
        WHERE dname LIKE '%'||p_dname||'%';
     RETURN TRUE;
EXCEPTION
     WHEN TOO_MANY_ROWS THEN
        dbms_output.put_line('不止一个部门包含字符'||v_oldname);
        RETURN FALSE;
END func_getFullDeptName;
/
```

过程和函数

- 调用func_getfulldeptname函数的方法如下：

```
DECLARE
    dname varchar2(30);
BEGIN
    dname:='SALE';
    IF Func_getFullDeptName(dname) THEN
       dbms_output.put_line('now,dname is:'||dname);
    ELSE
       dbms_output.put_line('有点不对劲！');
    END IF;
END;
/
```

函数

- 确定性
  可以使用DETERMINISTIC关键字将一个自定义的PL/SQL函数声明为是确定性函数。这意味着，Oracle数据库系统会相信：对于此函数而言，只要给定相同的输入，不论做多少次调用，它肯定能返回相同的值。
- 对于要在一个用户编写的函数上创建索引的情况，DETERMINISTIC关键字是必要的。

包（Package）

- 包可以将彼此相关的功能划分到一个自包含的单元中。通过使用包，可以将PL/SQL代码模块化，可以构建供其他编程人员重用的代码库。

- 包通常由两部分组成：
  - 规范（Specification）
  - 包体（body）

包

- 包规范包含有关包的信息，其中列出可用的过程和函数。所有的数据库用户对这些信息都具有潜在的访问能力，这些过程和函数称为公有（Public）对象。规范中通常不包括构成这些过程和函数的代码，包体中才包含实际的代码。
- 规范中列出的过程和函数可被外部访问，但是只在包体中包含的过程和函数只能被包自身访问，它们对于这个包体是私有（Private）对象

包-创建包规范

- 基本语法

```
CREATE [OR REPLACE] PACKAGE package_name
{IS|AS}
  package_specification
END package_name;
```

包-创建包体

- 基本语法

```
CREATE [OR REPLACE] PACKAGE BODY package_name
{IS|AS}
  package_body
END package_name;
```

关于包的一些规则

- 调用包中的函数和过程时，应该调用中使用包名。
- 除非包规范成功通过编译，否则包主体是不能成功通过编译的。
- 在包中，过程和函数是可以重载的，即可以让多个过程或函数公用同一个名称，但是带有不同的参数。不过有一些限制：
  - 如果两个子程序仅在参数名称和参数模式上不同，则不能重载这两个子程序
  - 如果两个函数只在返回类型上存在不同，不能进行重载
  - 重载函数的参数必须分别属于不同的类型系列，不能在同一类型系列的参数上使用重载
  - 在Oracle10gR1中，如果两个子程序仅在数字的数据类型上存在不同，可以进行重载

包

- 将Func_getFullDeptName放入包ss_pkg中，以便对代码进行管理和调用，如何实现？

- 分为两步：

  - 包规范的书写（类似于C语言中的.h文件）

  - 包体的书写（类似于C语言中的.c文件）

包

- 包规范

```
CREATE OR REPLACE PACKAGE ss_pkg
AS
    FUNCTION  func_getFullDeptName(
           p_dname IN OUT VARCHAR2)
       RETURN BOOLEAN;
END ss_pkg;
/
```

包

- 包体

```
CREATE OR REPLACE PACKAGE BODY ss_pkg AS
 FUNCTION  func_getFullDeptName(p_dname IN OUT VARCHAR2)RETURN BOOLEAN
 IS
   v_oldname varchar2(30);
 BEGIN
   IF p_dname IS NULL THEN
     dbms_output.put_line('p_dname is null');
     RETURN FALSE;
   END IF;
   v_oldname:=p_dname;
   SELECT dname INTO p_dname FROM dept
     WHERE dname LIKE '%'||p_dname||'%';
   RETURN TRUE;
 EXCEPTION
   WHEN TOO_MANY_ROWS THEN
     dbms_output.put_line('不止一个部门包含字符'||v_oldname);
     RETURN FALSE;
   END func_getFullDeptName;
 END ss_pkg;
 /
```

查看过程，函数和包的有关信息

```
SELECT * FROM user_procedures WHERE object_name='my_proc';
```

```
SELECT * FROM all_procedures WHERE object_name='my_func';
```

显示代码

- 查看过程或函数代码

```
SELECT text FROM user_source WHERE name='MY_PROC';
```

- 查看包代码

```
SELECT text FROM user_source
WHERE name='MY_PKG' AND type='PACKAGE';
```

- 查看触发器代码

```
SELECT text FROM user_source
WHERE name='MY_TRIG';
```

查看错误

- SHOW ERRORS

```
SHOW ERRORS PROCEDURE my_proc
```

```
SHOW ERRORS FUNCTION my_func
```

```
SHOW ERRORS PACKAGE my_pack
```

显示代码

- 查看过程或函数代码

```
SELECT text FROM user_source WHERE name='MY_PROC';
```

- 查看包代码

```
SELECT text FROM user_source
WHERE name='MY_PKG' AND type='PACKAGE';
```

- 查看触发器代码

```
SELECT text FROM user_source
WHERE name='MY_TRIG';
```

过程和函数（补充）

- 调用ss_pkg.func_getfulldeptname函数的方法如下：

```
DECLARE
    dname varchar2(30);
BEGIN
    dname:='SALE';
    IF ss_pkg.Func_getFullDeptName(dname) THEN
       dbms_output.put_line('now,dname is:'||dname);
    ELSE
       dbms_output.put_line('有点不对劲！');
    END IF;
END;
/
```

过程和函数（补充）

- 可以利用游标来解决输出结果多于一行的情况

```
CREATE OR REPLACE PROCEDURE getFullDeptName
    (p_dname IN varchar2,p_fullname OUT sys_refcursor)
IS
BEGIN
    IF p_dname IS NULL THEN
            dbms_output.put_line('p_dname is null');
            RETURN;
    END IF;
    OPEN p_fullname FOR
       SELECT dname FROM dept
       WHERE dname like '%'||p_dname||'%';
END getFullDeptName;
/
```

过程和函数（补充）

- 调用示例：

```
variable x refcursor
variable name varchar2(20)
exec :name:='A';
exec getfulldeptname(:name,:x)
print x
```

- 这种方法显然难以满足我们的需要，这时可以结合包的使用来满足这样的需要。

```
CREATE OR REPLACE PACKAGE ss_pkg
AS
     TYPE CURTYPE_DEPTNAME IS ref cursor;
     FUNCTION Func_getFullDeptName(p_dname IN OUT VARCHAR2)
         RETURN BOOLEAN;
     FUNCTION Func_getFullDeptName(p_dname varchar2,
             p_fullname OUT CURTYPE_DEPTNAME)
         RETURN BOOLEAN;
END ss_pkg;
/
```

包（补充）

- 包体

```
CREATE OR REPLACE PACKAGE BODY ss_pkg AS
    ......(原先的Func_getFullDeptName实现部分)
    FUNCTION Func_getFullDeptName(p_dname IN varchar2,
             p_fullname OUT CURTYPE_DEPTNAME)
       RETURN boolean
    IS
    BEGIN
       IF p_dname IS NULL THEN
           dbms_output.put_line('p_dname is null');
           RETURN false;
       END IF;
       OPEN p_fullname FOR SELECT dname FROM dept
           WHERE dname LIKE '%'||p_dname||'%';
       RETURN true;
    END Func_getFullDeptName;
END ss_pkg;
/
```

包（补充）

- 调用示例：

```
DECLARE
     result ss_pkg.CURTYPE_DEPTNAME;
     name dept.dname%TYPE;
     ret boolean;
BEGIN
     name:='A';
     ret:=ss_pkg.func_getfulldeptname(name,result);
     IF ret THEN
         dbms_output.PUT_line('包含有'''||name||'''字符的部门有:');
     FETCH result INTO name;
     WHILE result%FOUND LOOP
            dbms_output.put_line(name);
            FETCH result INTO name;
     END LOOP;
     END IF;
END;
/
```

不能使用 for 的那种遍历游标的方式，否则会出现错误
因为系统并不知道动态游标内部的情况

触发器

- 触发器是一种特殊的过程。当特定的事件发生时，触发器被自动执行。

- 触发器具有以下优势：
  - 能够进行复杂的有效性检验。
  - 能够用于审计。
  - 能够根据对一个表中的操作去自动修改其他表的内容。

触发器

- 触发器的作用
  - 防止非法的数据库操纵、维护数据库安全
  - 对数据库的操作进行审计，存储历史数据
  - 完成数据库初始化处理
  - 控制数据库的数据完整性
  - 进行相关数据的修改
  - 完成数据复制
  - 自动完成数据库统计计算
  - 限制数据库操作的时间、权限等，控制实体的安全性。

触发器

- 触发器与存储过程
  - 数据库触发器是在进行数据操纵时自动触发的，存储过程要通过程序调用。
  - 在数据库触发器中可以调用存储过程、函数。
  - 在触发器中禁止使用COMMIT 、ROLLBACK语句，存储过程中可以使用PL/SQL中可以使用的全部SQL语句。
  - 在触发器中不得间接调用含有COMMIT、ROLLBACK语句的存储过程。

触发器

- 触发器的分类（1）
  - DML触发器
  - Instead-of触发器
  - 系统触发器
- 触发器的分类（2）
  - 语句级触发器
  - 行级触发器
  - instead of触发器
  - 系统事件触发器
  - 用户事件触发器

DML触发器

- 基本语法

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER} trigger_event
ON table_name
[REFERENCING [OLD AS old_name][NEW AS new_name]]
[WHEN trigger_condition]
[FOR EACH ROW]
[DECLARE]
BEGIN
  trigger_body
END trigger_name;
```

DML触发器

触发器执行顺序：

1. 执行before语句级触发器
2. 对于受该语句影响的每一行：
    1. 执行before行级触发器
    2. 执行DML语句
    3. 执行after行级触发器
3. 执行after语句级触发器

DML触发器

- **:old和:new是两个很特殊的关联标识符**。
- 关联标识符是PL/SQL的一种特殊的绑定变量。标识符前面的冒号，既说明两者都是绑定变量，也说明他们不是一般的PL/SQL变量。PL/SQL编译器会把它们看作是定义为触发表的%ROWTYPE类型的记录。

DML触发器

- :old和:new的含义

触发语句	:old	:new
insert	未定义-所有字段均为null	触发语句完成时，要插入的值
update	更新以前相应记录行的原始值	触发语句完成时，要更新的值
delete	更新以前相应记录行的原始值	未定义-所有字段均为null

DML触发器

- 不能在after行级触发器中更改:new的值，因为该语句已经处理过了。
- 不能更改:old的值（不论何种情况）。
- 一般情况下，只会在before行级触发器中更改:new的值，并且永远不修改:old的值，只会从:old标识符中取值。

DML触发器

- 只能在行级触发器中使用:new和:old记录。如果在语句级触发器中引用它们，就会产生一个编译错误。为什么？

- 因为:new和:old是记录类型变量

DML触发器

- 虽然在语法上可将:new和:old看作triggering_table%ROWTYPE类型的记录，但实际上它们并不是记录：某些在记录上能够正常执行的操作并不适用于:new和:old。比如，不能将它们作为一个整体赋值，只能对其中的各个字段分别赋值；不能将:old和:new传递给参数类型为triggering_table%ROWTYPE的过程或函数。

- 所以，:new和:old也被称为伪记录。

DML触发器

- REFERENCING子句中，关联标识符都不带冒号。

- DML触发器中的WHEN子句只能在行级触发器中使用。触发器主体只对满足WHEN所定义条件的那些记录行执行。
  可以在WHEN子句的条件中使用:new和:old记录，但使用时不需要冒号。

- 关联标识符仅在触发器主体中才需要使用冒号

触发器

```
CREATE TABLE testlog(
        curr_user varchar2(100),
        target varchar2(100),
        curr_date date,
        act varchar2(1));
```

```
 CREATE OR REPLACE TRIGGER TRIG_DMLLOG
BEFORE insert or delete or update on DEPT
BEGIN
      dbms_output.put_line('DML Trigger fired!');
      IF inserting then
        insert into testlog values(user,'scott.DEPT',sysdate,'I');
      ELSIF deleting then
        insert into testlog values(user,'scott.DEPT',sysdate,'D');
      ELSE
        insert into testlog values(user,'scott.DEPT',sysdate,'U');
      END IF;
END;
/
```

触发器

```
CREATE TABLE salhis(
  id number(38) primary key, time date, empno number(38),newsal number
);
```

```
CREATE SEQUENCE SEQ_salHis START WITH 100 INCREMENT BY 2;
```

```
CREATE OR REPLACE TRIGGER trig_EMPSal_DML
AFTER update or insert or delete on Emp
FOR EACH ROW
BEGIN
  IF updating THEN
        IF :new.sal!=:old.sal THEN
                insert into salhis values(seq_salhis.nextval,
                        sysdate,:new.empno,:old.sal);
        END IF;
  ELSIF inserting THEN
                insert into salhis values(seq_salhis.nextval,
                        sysdate,:new.empno,0);
  ELSE
        delete from salhis where empno=:old.empno;
  END IF;
END trig_EMPSal_DML;
/
```

DML触发器

- 在同一个表上可以定义触发器的数目没有限制，定义不同类型的DML触发器的数目也没有限制。同一种类的所有触发器会相继激活。

- 每一个触发器激活的时候，它都会查看之前的触发器对数据执行的更改，以及迄今为止，该触发语句对数据库执行的所有更改。（举例）

- 如果触发器的激活顺序非常重要，怎么办？
- 将所有这些操作联合成一个触发器

触发器

- 利用触发器和序列来实现整型数据的自增

```
CREATE OR REPLACE TRIGGER trig_seqSalHis
BEFORE insert ON SalHis
FOR EACH ROW
BEGIN
  select seq_salhis.nextval into :new.id from DUAL;
END;
/
```

如果BEFORE换成AFTER会怎样？

触发器

- salhis中的序号不是递增2,而是递增了4,也就是说seq运行了两次nextval求值过程，为什么？

- 因为刚才在对emp的触发器中还是采用seq.nexval的形式，查看一下原先的定义。

```
select text from user_source where name='TRIG_EMPSAL_DML';
```

触发器

- 将前述触发器trig_EMPSal_DML变为：

```
CREATE OR REPLACE TRIGGER trig_EMPSal_DML
AFTER update or insert or delete on Emp
FOR EACH ROW
BEGIN
  IF updating THEN
        IF :new.sal!=:old.sal THEN
          insert into salhis values(1,sysdate,:new.empno,:old.sal);
        END IF;
  ELSIF inserting THEN
          insert into salhis values(1,sysdate,:new.empno,0);
  ELSE
        delete from salhis where empno=:old.empno;
  END IF;
END trig_EMPSal_DML;
/
```

Instead of 触发器

- 基本语法

```
CREATE [OR REPLACE] TRIGGER trigger_name
INSTEAD OF trigger_event
ON view_name
[REFERENCING [OLD AS old_name][NEW AS new_name]]
[WHEN trigger_condition]
[FOR EACH ROW]
[DECLARE]
BEGIN
  trigger_body
END trigger_name;
```

instead of 触发器

- Instead of 触发器只能定义在视图上
- instead of 触发器必须是行级触发器
- 不能创建instead of 系统触发器

Instead of 触发器中的 for each row 子句是可选的。不管有没有定义此子句，所有的instead of 触发器都是行级触发器

Instead of 触发器

- Instead of 触发器示例

```
CREATE OR REPLACE VIEW vw_DeptSal
AS
select dept.deptno,dname,max(sal) maxsal,min(sal) minsal
from emp,dept
where emp.deptno=dept.deptno
group by dept.deptno,dname;
```

Instead of 触发器

- Instead of 触发器示例

```
CREATE OR REPLACE TRIGGER trig_upd_vw_deptsal
INSTEAD OF update ON vw_deptsal
FOR EACH ROW
BEGIN
       IF :new.maxsal!=:old.maxsal THEN
              update emp
              set sal=:new.maxsal
              where deptno=:new.deptno and sal=:old.maxsal;
       END IF;
       IF :new.minsal!=:old.minsal THEN
              update emp
              set sal=:new.minsal
              where deptno=:new.deptno and sal=:old.minsal;
       END IF;
END;
```

系统触发器

- 基本语法

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER}
{ddl_event_list|database_event_list}
ON {DATABASE|[schema.]SCHEMA}
[WHEN trigger_condition]
[FOR EACH ROW]
[DECLARE]
BEGIN
  trigger_body
END trigger_name;
```

系统触发器

- 系统触发器的激活基于两种不同的事件：
  - ¤ DDL事件
    - CREATE
    - ALTER
    - DROP
    - （TRUNCATE，…）
  - ¤ 数据库事件
    - 数据库服务器的启动/关闭事件
    - 用户的登录/断开事件
    - 服务器错误

系统触发器

- DATABASE和SCHEMA关键字决定了某个给定系统触发器的级别（数据库级/模式级）。
- 只要发生了激活事件，数据库级触发器就会激活。而只有激活事件发生在某个具体的模式中时，相应的模式级触发器才会激活。
- 如果使用SCHEMA关键字时未指定具体模式的名称，那么默认设置为拥有这个触发器的模式。

系统触发器

- 系统触发器也可以使用WHEN子句，指定触发器激活的条件。但在可以对每种触发器指定条件的类型上，存在一些限制：
  - STARTUP和SHUTDOWN触发器不能带有任何条件
  - SERVERERROR触发器可以使用ERRNO检测，只检查某一特定的错误
  - LOGON和LOGOFF触发器可以使用USERID或USERNAME检测，检查用户的ID或名称。
  - DDL触发器既可以检查正被更新对象的类型和名称，还可以检查用户的ID或名称。

触发器

- 触发器的禁用、启用与删除

```
ALTER TRIGGER my_trig {DISABLE|ENABLE};
```

```
ALTER TABLE trig_table {DISABLE|ENABLE} ALL TRIGGERS;
```

```
DROP TRIGGER my_trig;
```

触发器

- 查看触发器信息

```
SELECT trigger_name,table_name FROM user_triggers;
```

小结

- 过程
- 函数
- 程序包
- 触发器

尝试

- 写出PL/SQL程序以完成如下功能：
  - 记录每个数据库用户登录数据库系统和退出数据库系统的时间；
  - 记录每个用户删除表的操作时间与操作对象（提示：可通过DBA_OBJECTS获得用户及其拥有的数据库对象的名称；可通过数据库级的drop事件获得删除表的触发时机）；
  - 查询某一给定用户（通过输入参数指定）在某一天（通过输入参数指定）的数据库登录和退出次数统计显示。

The End