

Oracle 常用命令举例

- 基本语法
 - [% type 用法](#)
 - [%rowtype 用法](#)
 - [TYPE 用法](#)
 - [游标的使用](#)
 - [for 循环](#)
 - [loop 循环](#)
 - [while 循环](#)
 - [if / else 的用法](#)
 - [case 的用法](#)
- 错误定义
 - [error 的设置](#)
 - [exception 用法](#)
- 存储过程及函数
 - [procedure 的建立和调用](#)
 - [function 的建立和调用](#)
- [参数的调用 \(in 模式为按址调用, out / in out 模式为按值调用。NOCOPY 强行转换成按址调用\)](#)
- 软件包及封装
 - [软件包 \(PACKAGE\) 的建立和调用](#)
 - [软件包的全局结构](#)
 - [封装函数的纯度](#)
- 查看源代码及建立用户、用户的权限
 - [源代码的查看](#)
 - [建立用户及登陆](#)
 - [授予权限和权限收回](#)
- 依赖
 - [直接依赖](#)
 - [查看依赖](#)
 - [包之间调用](#)
- 触发器
 - [建立简单的触发器](#)
 - [触发器分类](#)
 - [稍复杂的触发器](#)
 - [条件谓词](#)
 - [触发器中不可使用 Commit](#)
 - [系统触发器举例 \(LOGON\)](#)
 - [Instead of 触发器](#)

1. % type 用法，提取% type 所在字段的类型

```
declare
    myid dept.id % type;
    myname dept.name % type;
begin
    select id,name into myid,myname from dept;
    dbms_output.put_line(myid);
    dbms_output.put_line(myname);
end;
/
```

2. %rowtype 用法，提取%rowtype 所在的字段的类型

```
declare
    type type_dept is table of dept % rowtype
    index by binary_integer;
    tb type_dept;
begin
    tb(1).id:='001';
    tb(2).id:='001';
    dbms_output.put_line(tb.COUNT);
end;
/
```

3. TYPE 用法，相当于结构体

```
declare
    lv_order_date DATE:=sysdate;
    lv_last_txt varchar2(5) default '001';
    lv_last varchar2(10) not null:='us';
    TYPE type_test is record(
        myid dept.id % type,
        myname dept.name % type);
    rec type_test;
begin
    lv_order_date:=sysdate;
    dbms_output.put_line(lv_last);
    select id,name into rec from dept;

    dbms_output.put_line(rec.myid);
    dbms_output.put_line(rec.myname);
end;
```

/

4. 游标的使用

```
declare
```

```
g_id char(10):='002';
```

```
find_not char(1):='N';
```

```
cursor cur is
```

```
select * from dept;
```

cur 指向表

```
TYPE type_dept is record(
```

```
myid dept.id % type,
```

```
myname dept.name % type,
```

```
myaddr dept.addr % type);
```

```
rect type_dept;
```

```
begin
```

```
open cur;
```

```
loop
```

```
fetch cur into rect;
```

提取 cur 指向的记录到 rect 结构中

```
exit when cur% NOTFOUND;
```

```
if rect.myid=g_id then
```

```
find_not:='Y';
```

```
dbms_output.put_line('Find it!!');
```

```
dbms_output.put_line('DEPT ID:' || rect.myid);
```

```
dbms_output.put_line('NAME:' || rect.myname);
```

```
dbms_output.put_line('ADDR:' || rect.myaddr);
```

```
end if;
```

```
end loop;
```

```
close cur;
```

```
if find_not='N' then
```

```
dbms_output.put_line('no record');
```

```
end if;
```

```
end;
```

```
/
```

5. for 循环

```
begin
```

```
for i in 1..5 loop
```

```
dbms_output.put_line(i);
```

```
end loop;
```

```
end;
```

```
/
```

6. loop 循环

```
declare
  v number:=1;
begin
  loop
    dbms_output.put_line(v);
    exit when v>5;
    v:=v+1;
  end loop;
end;
/
```

7. while 循环

```
declare
  v number:=1;
begin
  while v<5 loop
    dbms_output.put_line(v);
    v:=v+1;
  end loop;
end;
/
```

8. error 的设置

```
declare
  v1 number:=90;
begin
  if v1=10 then dbms_output.put_line('v1 is 10');
  elsif v1=20 then dbms_output.put_line('v2 is 20');
  else goto err;
    dbms_output.put_line('normal end');
  <<err>>
    dbms_output.put_line('error found');
  end if;
end;
/
```

9. exception 用法

```
declare
    ex Exception;
begin
    Update dept set name='Edison'
        where id='100';
    if SQL%NOTFOUND Then
        Raise ex;
    end if;
    Exception
        When ex then
            dbms_output.put_line('update failed.');
```

```
end;
/

declare
    type rc_dept is record (
        myid dept.id%type,
        myname dept.name%type,
        myaddr dept.addr%type
    );
    tb rc_dept;
begin
    select id,name,addr into tb from dept where id=:gb_id;
    dbms_output.put_line('id:' || tb.myid);
    dbms_output.put_line('name:' || tb.myname);
    dbms_output.put_line('addr:' || tb.myaddr);
exception
    when NO_DATA_FOUND then
        dbms_output.put_line('no record is found');
    when TOO_MANY_ROWS then
        dbms_output.put_line('too many rows are selected');
    when OTHERS then
        dbms_output.put_line('undefine error');
        dbms_output.put_line('error coede: ' || SQLCODE);
        dbms_output.put_line('error message:' || SQLERRM);
end;
/
```

```

declare
type rc_dept is record (
    myid dept.id%type,
    myname dept.name%type,
    myaddr dept.addr%type
);
tb rc_dept;
begin
    begin
        select id,name,addr into tb from dept where id=:gb_id;
        dbms_output.put_line('id:' || tb.myid);
        dbms_output.put_line('name:' || tb.myname);
        dbms_output.put_line('addr:' || tb.myaddr);
    exception
        when NO_DATA_FOUND then
            dbms_output.put_line('no record is found, occur in inner.');
```

内层错误捕捉其始点，在此之前发生的错误由外层进行捕捉。

```

    end;
exception
    when TOO_MANY_ROWS then
        dbms_output.put_line('too many rows are selected, occur in outer.');
```

内层的错误捕捉到后，外层的错误就不捕捉了。否则由外层捕获错误。

```

    when OTHERS then
        dbms_output.put_line('undefine error');
        dbms_output.put_line('error coede: ' || SQLCODE);
        dbms_output.put_line('error message:' || SQLERRM);
end;
/
```

10. if / else 的用法

```

declare
v1 number:=90;
begin
    if v1=10 then dbms_output.put_line('v1 is 10');
    elsif v1=20 then dbms_output.put_line('v2 is 20');
    else dbms_output.put_line('v2 is others');
end if;
end;
/
```

11. case 的用法

```

declare
v number:=10;
```

```
begin
case :v
when 10 then dbms_output.put_line('v is 10');
when 20 then dbms_output.put_line('v is 20');
else dbms_output.put_line('v is not 10 and 20');
end case;
end;
/
```

12. procedure 的建立和调用

```
create or replace procedure test_sp
(test in number, outtest out number)
is
begin
if test > 10 then
printsomthing ('test is over 10!!');
else
begin
outtest := test;
printsomthing (outtest);
end;
end if;
end;
/
```

参数的声明不要对它的大小进行定义。
IN 表示传入的参数，不能修改，OUT 表示传出的参数。

```
create or replace procedure printsomthing
(print in number)
is
begin
dbms_output.put_line(print);
end;
/
```

```
create or replace procedure printsomthing
(print in char)
is
begin
dbms_output.put_line(print);
end;
/
```

过程调用过程的参数调用格式注意，不加 “:”

exec test_sp(:test,:outtest); 外部执行的时候注意参数调用方式要加 “:”

存储过程可以重载，符合 C++ 的重载规则。

13. function 的建立和调用

```
create or replace function test(t in number) return number
```

```
is
```

```
begin
```

```
    if t>10 then
```

```
        dbms_output.put_line(t);
```

```
    elsif t<10 then
```

```
        dbms_output.put_line(t);
```

```
    end if;
```

```
    return t;
```

```
end;
```

```
/
```

Function 的建立，需要返回值，
但不需要说明大小。

注意：调用的方法，不能以 procedure 那样独立进行调用。函数是表达式的一部分（有返回值）。

exec test(1); 错误

exec :tt:=test(2); 正确

Tips:建议使用 return 模式，而不是使用 out 模式。

procedure 中也能用 return，这里的 return 只表示当前 procedure 的中断。

参数如同 procedure 一样，不能修改 in 的参数

```
create or replace function test(t in number) return number
```

```
is
```

```
begin
```

```
    if t>10 then
```

```
        t:=t+10;
```

```
        dbms_output.put_line(t);
```

```
    elsif t<10 then
```

```
        dbms_output.put_line(t);
```

```
    end if;
```

```
    return t;
```

```
end;
```

```
/
```

LINE/COL ERROR

5/3 PLS-00363: 表达式 'T' 不能用作赋值目标

5/3 PL/SQL: Statement ignored

多路 return

```
create or replace function test(t in number) return number
```

```
is
```

```
begin
```

```
    if t<10 then
```

```
        return 1;
```

```
    elsif t>=10 then
```

```
        return 2;
```

```
    end if;
```



```
end;  
/
```

function 中调用 procedure

```
create or replace function test(t in number) return number
```

```
is  
begin  
    printnumber(t);  
    return t;  
end;  
/
```

```
create or replace procedure printnumber  
    (print in number)  
is  
begin  
    dbms_output.put_line(print);  
end;  
/
```

14. 参数的调用 (in 模式为按址调用 , out / in out 模式为按值调用。NOCOPY 强行转换成按址调用。

```
create or replace procedure test_nocopy_sp(p_in in number, p_out in out nocopy number)
```

```
is  
begin  
    p_out:=5;  
    if p_in=1 then  
        raise no_data_found;  
    end if;  
end;  
/
```

强行抛出一个异常，以显示参数的结果。

```
create or replace procedure run_nocopy_sp
```

```
is  
lv_test_num number;  
begin  
    lv_test_num:=1;  
    test_nocopy_sp(1,lv_test_num);  
exception  
    when others then  
        dbms_output.put_line('error happened' || lv_test_num);  
end;  
/
```

因为 test_nocopy_sp 这个过程的第二个参数是 nocopy 的，也就是传址的，所以修改了 lv_test_num，为 5。
如果 test_nocopy_sp 这个过程的第二个参数不是 nocopy，那么就是传值，lv_test_num 不被修改，仍然为 1。

```
error happened 5  
error happened 1
```

15. 软件包 (package) 的建立 (包含了函数的重载)

软件包声明

```
create or replace package test_package
```

```
is
```

```
procedure test_sp  
    (test in number, outtest out number);
```

只声明过程、函数的原型。

```
procedure printsomthing  
    (print in number);
```

函数 printsomthing 的重载

```
procedure printsomthing  
    (print in char);
```

```
function test  
    (t in number) return number;
```

```
end;
```

```
/
```

软件包体的建立

```
create or replace package body test_package
```

```
is
```

无 begin

```
procedure test_sp  
    (test in number, outtest out number)
```

```
is
```

```
begin
```

```
    if test > 10 then
```

```
        printsomthing ('test is over 10!!');
```

```
    else
```

```
        begin
```

```
            outtest := test;
```

```
            printsomthing (test);
```

```
        end;
```

```
    end if;
```

```
end test_sp;
```

end 的注意

```
procedure printsomthing
```

```
    (print in number)
```

```
is
```

```
begin
```

```
    dbms_output.put_line(print);
```

```
end printsomthing;
```

```
procedure printsomthing
```

```
(print in char)
is
begin
    dbms_output.put_line(print);
end printsomething;

function test(t in number) return number
is
begin
    if t>10 then
        dbms_output.put_line(t);
    elsif t<10 then
        dbms_output.put_line(t);
    end if;
    return t;
end test;
end;
/
```

具体写体的时候和以前一样写，只不过不用写 create or replace，在 end 最后还要紧跟过程或函数名。

执行结果

SQL> var test1 number;

SQL> **exec test_package.test_sp(20,:test1);** ←

test is over 10!!

调用方法就是在前面加个包名，其余注意点和过程或函数相同。

PL/SQL 过程已成功完成。

SQL> exec test_package.test_sp(1,:test1);

1

PL/SQL 过程已成功完成。

SQL> **exec :test1:=test_package.test(20);** ←

20

调用方法就是在前面加个包名，其余注意点和过程或函数相同。

PL/SQL 过程已成功完成。

SQL> print test1;

TEST1

16. 软件包全局结构

```
create or replace package test_global
is
    global_v number(3):=0;
    procedure setValue(p1 in number);
end;
/
```

包的全局变量

```
create or replace package body test_global
is
    procedure setValue(p1 in number)
    is
        begin
            global_v:=p1;
            dbms_output.put_line(global_v);
        end setValue;
end;
/
```

修改全局变量并输出

建立 2 个会话：

```
exec test_global.setValue(20);
```

```
exec test_global.setValue(10);
```

2 个会话分别维护自己的全局变量。互不影响。

17. 封装函数的纯度

```
create or replace package test_global
is
    global_v number(3):=0;
    function setValue(p1 in number) return number;
    pragma restrict_references(setValue,WNPS);
end;
/
```

指定纯度。

WNDS Writes No Database State

函数不修改任何数据库表

RNDS Reads No Database State

函数不读取任何表

WNPS Writes No Package State

函数不修改任何封装变量

RNPS Reads No Package State

函数不读取任何封装变量

```
create or replace package body test_global
is
    function setValue(p1 in number) return number
    is
        begin
            global_v:=p1;

```

违反了 WNPS 的约束。

PACKAGE BODY TEST_GLOBAL 出现错误:

LINE/COL ERROR

PLS-00452: 子程序 'SETVALUE' 违反了它的相关注记

```

        dbms_output.put_line(global_v);
        return global_v;
    end setValue;
end;
/
    
```

18. 源代码的查看

SQL> desc user_source

名称	是否为空? 类型	

NAME	VARCHAR2(30)	
TYPE	VARCHAR2(12)	
LINE	NUMBER	
TEXT	VARCHAR2(4000)	存放源代码的字段

SQL> **select text from user_source where name='TEST';** ← 注意：name 一定要大写。

TEXT

```

function test(t in number) return number
is
begin
    if t>10 then
        dbms_output.put_line(t);
    elsif t<10 then
        dbms_output.put_line(t);
    end if;
    return t;
end;
    
```

已选择 10 行。 显示行号

SQL> select **rownum**,text from user_source where name='TEST';

19. 建立用户及登陆

SQL> **create user MascotZhuang IDENTIFIED BY MascotZhuang;**

用户已创建



SQL> **grant create session to MascotZhuang;** 赋予能够连上数据库的权力

授权成功。

如果没有这一句会发生以下错误：

ERROR：

ORA-01045: user MASCOTZHUANG lacks CREATE SESSION privilege; logon denied

C:\>**sqlplus "MascotZhuang/MascotZhuang"**

连接到:

Oracle9i Enterprise Edition Release 9.0.1.1.1 - Production

With the Partitioning option

JServer Release 9.0.1.1.1 - Production

SQL>

注意：

如果以 MascotZhuang/MascotZhuang as sysdba

登陆，相当于 / as sysdba 登陆。

通过 show user 可以看到 user 还是 sys，而不是

MascotZhuang

20. 授权和收回权限

授予全部权限

SQL> **grant all on test_package to MascotZhuang;**

授权成功。

授予特定权限

SQL> **grant execute on test_package to MascotZhuang;**

授权成功。

收回权限

SQL> **revoke all on test_package from MascotZhuang;**

撤销成功。

创建的用户使用包

SQL> var test1 number;

SQL> set serveroutput on

SQL> exec :test1:=**sys**.test_package.test(20);

20

PL/SQL 过程已成功完成。

SQL>

21. 直接依赖性

create or replace procedure test_dependency

(p_print char)

is

begin

printsomething(p_print);

end;

```
/
```

当创建这个 procedure 的时候，会发现一下错误。

```
LINE/COL ERROR
```

```
5/3      PLS-00201: 必须说明标识符 'PRINTSOMETHING'
5/3      PL/SQL: Statement ignored
```

因为这个时候 ,printsomething 这个 procedure 还没有创建。换句话说 ,test_dependency 依赖于 printsomething。所以在 printsomething 创建之前，test_dependency 是无效的。

```
SQL> select status from user_objects where object_name='TEST_DEPENDENCY';
```

STATUS

INVALID

因此，必须创建 printsomething 这个过程。

```
create or replace procedure printsomething
  (p_print char)
is
  begin
    dbms_output.put_line(p_print);
  end;
```

```
/
```

这时候，test_dependency 的 status 还是 invalid。需要进行重现编译，才能使得 status 为 valid。

```
SQL> alter procedure test_dependency compile;
```

```
SQL> select status from user_objects where object_name='TEST_DEPENDENCY';
```

STATUS

VALID

22. 查看依赖性

```
SQL> select referenced_name,referenced_type from user_dependencies where name='TEST_DEPENDENCY';
```

REFERENCED_NAME	REFERENCED_T	
STANDARD	PACKAGE	← 依赖系统的 PACKAGE
SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE	←
PRINTSOMETHING	PROCEDURE	← TEST_DEPENDENCY 所依赖的 procedure

```
SQL> select referenced_name,referenced_type from user_dependencies where name='PRINTSOMETHING';
```

REFERENCED_NAME	REFERENCED_T

STANDARD	PACKAGE
SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
DBMS_OUTPUT	PACKAGE

依赖系统的 PACKAGE

23. 包之间的调用

```
create or replace package test_global
```

```
is
```

```
  global_v number(3):=0;
```

```
  procedure setValue(p1 in number);
```

```
  procedure print(p1 in number);
```

```
end;
```

```
/
```

都有一个包的全局变量

```
create or replace package body test_global
```

```
is
```

```
  procedure setValue(p1 in number)
```

```
  is
```

```
    begin
```

```
      global_v:=p1;
```

```
      dbms_output.put_line('this is test_global,global_v is ' || global_v);
```

```
      test1_global.setValue(3);
```

```
    end setValue;
```

```
  procedure print(p1 in number)
```

```
  is
```

```
    begin
```

```
      dbms_output.put_line('test_global, global_v is ' || global_v);
```

```
    end print;
```

```
end;
```

```
/
```

调用 test1_global 这个包中的 setValue 这个过程。只需加上这个包名即可。

```
create or replace package test1_global
```

```
is
```

```
  global_v number(3):=0;
```

```
  procedure setValue(p1 in number);
```

```
  procedure print(p1 in number);
```

```
end;
```

```
/
```

都有一个包的全局变量


```
create or replace package body test1_global
is
  procedure setValue(p1 in number)
  is
    begin
      global_v:=p1;
      dbms_output.put_line('this is test1_global,global_v is ' || global_v);
    end setValue;
  procedure print(p1 in number)
  is
    begin
      dbms_output.put_line('test1_global, global_v is ' || global_v);
    end print;
end;
/
```

setValue 过程对全局变量赋值。但是各自的包调用各自的过程，修改各自的全局变量。

SQL> exec test_global.setValue(200);

this is test_global,global_v is 200

this is test1_global,global_v is 3

PL/SQL 过程已成功完成。

SQL> exec test_global.print(10);

test_global, global_v is 200

PL/SQL 过程已成功完成。

SQL> exec test1_global.print(10);

test1_global, global_v is 3

PL/SQL 过程已成功完成。

各自的过程修改的自己的全局变量。

各自的过程修改的自己的全局变量。

24. 建立触发器（仅说明触发器的工作原理）

建立一个简单的表：

```
create table table_test_trigger(id char(10));
```

建立简单的触发器：

```
create or replace trigger myTrigger
```

```
after insert on table_test_trigger
```

触发器触发的时机，有记录插入到 table_test_trigger 后就触发。

```
begin
```

```
    dbms_output.put_line('something is inserted!!');
```

触发器体，触发器处理的东西

```
end;
```

```
/
```

```
SQL> insert into table_test_trigger Values('01');
```

插入一条记录来触发触发器。

```
something is inserted!!
```

触发器被触发。

已创建 1 行。

25. 触发器的分类

分为：行级触发器和语句级触发器

行级触发器：对于 DML 语句影响的每一行都触发触发器代码。只适合于 UPDATE 和 DELETE 事件。

语句级触发器：对该事件触发一次触发器。INSERT 事件

26. 稍复杂的触发器

第一个触发器：

```
create table table_test_trigger(id char(10),name char(10));
```

```
insert into table_test_trigger values('01','a');
```

```
insert into table_test_trigger values('02','b');
```

```
insert into table_test_trigger values('03','c');
```

```
insert into table_test_trigger values('04','d');
```

```
insert into table_test_trigger values('05','e');
```

```
create or replace trigger myTrigger
```

```
after update of name on table_test_trigger
```

这里比 24 建立触发器中多了 of name，表示只有当 UPDATE NAME 这个字段后，才触发触发器。

```
begin
```

```
    dbms_output.put_line('something is inserted!!');
```

```
end;
```

```
/
```

```
SQL> update table_test_trigger set name='zz' where id='01';
```

```
something is inserted!!
```

触发了触发器。

已更新 1 行

```
SQL> update table_test_trigger set id='99' where name='b';
```

已更新 1 行。

没有触发触发器。因为没对 name 进行 update

第二个触发器：

```
create or replace trigger myTrigger
after update of name on table_test_trigger
for each row
when(OLD.name='a')
begin
    dbms_output.put_line('something is inserted!');
end;
/
```

For each row 表示每行都要触发，但要满足 for each row 下面 when 的条件。OLD.name 表示未更新前的 name 指，相对的有个 NEW.name 表示更新后的值。

```
SQL> update table_test_trigger set name='zb';
something is inserted!
已更新 5 行。
```

只触发了一次，因为只有一条记录满足条件。

第三个触发器：

```
create or replace trigger myTrigger
after update of name on table_test_trigger
for each row
begin
    dbms_output.put_line('something is inserted!');
end;
/
```

无条件，对于每行都触发，这里的每行是指外部 update 语句影响到的每行。

```
SQL> update table_test_trigger set name='zb';
something is inserted!
something is inserted!
something is inserted!
something is inserted!
something is inserted!
已更新 5 行。
```

这里是无条件的 UPDATE，因此，表中有几条记录，就应该触发几次。

27. 条件谓词

```
create or replace trigger myTrigger
after update of name on table_test_trigger
for each row
begin
    if UPDATING THEN
        dbms_output.put_line('UPDATING!');
    end if;
```

If DELETING THEN

```
    dbms_output.put_line('DELETING!');  
end if;
```

If INSERTING THEN

```
    dbms_output.put_line('INSERTING!');  
end if;  
end;  
/
```

SQL> update table_test_trigger set name='zb';

触发器触发

UPDATING!

UPDATING!

UPDATING!

UPDATING!

UPDATING!

已更新 5 行。

```
create or replace trigger myTrigger  
after update of name on table_test_trigger  
for each row  
begin  
    If UPDATING('name') THEN  
        dbms_output.put_line('UPDATING!');  
    end if;  
end;  
/
```

当 name 字段 update 的时候才触发触发器。

SQL> update table_test_trigger set **name**='zb';

触发触发器

UPDATING!

UPDATING!

UPDATING!

UPDATING!

UPDATING!

已更新 5 行。

SQL> update table_test_trigger set **id**='99';

未触发触发器

已更新 5 行。

28. Trigger 中不能使用 Commit

```
create table test(id char(10));  
create table test1(id char(10),logdate date);
```

```
create or replace trigger myTrigger
```

```
after insert on test
```

```
begin
```

```
insert into test1 values('001',sysdate);
```

没有 commit 的触发器

```
end;
```

```
/
```

```
SQL> insert into test values('001');
```

已创建 1 行。

```
SQL> select * from test1;
```

ID	LOGDATE
001	14-10月-04

```
SQL> rollback;
```

回滚后发现，两个表的操作均被撤销

回退已完成。

```
SQL> select * from test1;
```

未选定行

```
SQL> select * from test;
```

未选定行

```
create or replace trigger myTrigger
```

```
after insert on test
```

```
begin
```

```
insert into test1 values('001',sysdate);
```

```
commit;
```

添加了 commit

```
end;
```

```
/
```

```
SQL> insert into test values('009');
```

```
insert into test values('009')
```

*

ERROR 位于第 1 行:

ORA-04092: COMMIT 不能在触发器中

ORA-06512: 在"SCOTT.MYTRIGGER", line 3

ORA-04088: 触发器 'SCOTT.MYTRIGGER' 执行过程中出错

29. 系统触发器举例

```
create table test1(id char(10),logdate date);
```

```
create or replace trigger logon_trg
```

```
after logon on schema
```

Logon 是系统触发器。系统触发器详见书本。

```
begin
```

```
insert into test1 values(user,sysdate);
```

```
end;
```

```
/
```

再用 scott/tiger 重新在新连接中登陆，会发现触发了系统触发器。注：用 sys/sas as dba 登陆无效。

```
SQL> select * from test1;
```

```
ID          LOGDATE
```

```
-----
```

```
SCOTT      14-10 月-04
```

30. instead of 触发器

```
create table test1(id char(10),name char(10));
```

```
insert into test1 values('01','ab');
```

```
create or replace view test_view
```

建立视图

```
as select name from test1;
```

```
create or replace trigger test_trigger
```

用触发器来代替 UPDATE ,这就是为什么叫 instead of 触发器。

```
instead of update on test_view
```

```
for each row
```

```
begin
```

```
update test1 set name='zz';
```

```
end;
```

```
/
```

```
SQL> update test_view set name='aa';
```

已更新 1 行。

```
SQL> select * from test1;
```

```
ID          NAME
```

```
-----
```

```
01          zz
```

31. 创建主键

```
CREATE TABLE test
```

```
( id char(10) CONSTRAINT id_pk PRIMARY KEY
```

主键设置

```
, name varchar2(20)
```

```
);
```

```
CREATE TABLE test
```

```
( id char(10) CONSTRAINT id_pk NOT NULL
```

设置不为空的设置

```
, name varchar2(20)
```

```
);
```

```
insert into test values('b03011117','zb');
```

32. 创建外键

```
CREATE TABLE test
```

```
( id char(10) CONSTRAINT id_pk PRIMARY KEY
```

```
, name varchar2(20)
```

```
);
```

```
CREATE TABLE test1
```

```
( studentID varchar2(20) CONSTRAINT studentID_pk PRIMARY KEY
```

```
, id CONSTRAINT fk_id REFERENCES test(id)
```

设置外键，也就是说，这里的 id 的值受到 test 表中 id 值的限制。

```
);
```

```
insert into test values('b03011117','zb');
```

```
insert into test1 values('b03011117','zb');
```

ERROR 位于第 1 行:

违反了约束条件，出错。

ORA-02291: 违反完整约束条件 (SCOTT.FK_ID) - 未找到父项关键字