Oracle 九阴真经

本帮助的示例主要是针对SCOTT/tiger下的表
由于本人初学Oracle还有很多不明白的地方,
可能出现错误和不全的地方,希望各位大侠给予指出!
联系方式:e_mail flyer_uo@163.com
QQ 6744428
本人在这给予衷心的感谢!

制作人：谭凯

创建表

CREATE TABLE <table_name>(
column1 DATATYPE [NOT NULL] [PRIMARY KEY],
column2 DATATYPE [NOT NULL],
...
[constraint <约束名> 约束类型 (要约束的字段)
...] ）
说明:
DATATYPE --是Oracle的数据类型,可以查看附录。
NUT NULL --可不可以允许资料有空的（尚未有资料填入）。
PRIMARY KEY --是本表的主键。
constraint --是对表里的字段添加约束.(约束类型有
Check,Unique,Primary key,not null,Foreign key)。

示例:
```
create table stu(
s_id number(8) PRIMARY KEY,
s_name varchar2(20) not null,
s_sex varchar2(8),
clsid number(8),
constraint u_1 unique(s_name),
constraint c_1 check (s_sex in ('MALE','FEMALE'))
);
```

复制表

CREATE TABLE <table_name> as <SELECT 语句>

(需注意的是复制表不能复制表的约束);

示例:
```
create table test as select * from emp;
```

如果只复制表的结构不复制表的数据则:
```
create table test as select * from emp where 1=2;
```

创建索引

CREATE [UNIQUE] INDEX <index_name> ON <table_name>(字段 [ASC|DESC]);

UNIQUE --确保所有的索引列中的值都是可以区分的。
[ASC|DESC] --在列上按指定排序创建索引。

(创建索引的准则:
1.如果表里有几百行记录则可以对其创建索引(表里的记录行数越多索引的效果就越明显)。
2.不要试图对表创建两个或三个以上的索引。
3.为频繁使用的行创建索引。
)

示例
create index i_1 on emp(empno asc);

创建同义词

CREATE SYNONYM <synonym_name> for <tablename/viewname>

同义词即是给表或视图取一个别名。

示例:
create synonym mm for emp;

修改表

1.向表中添加新字段
ALTER TABLE <table_name> ADD (字段1 类型 [NOT NULL],
字段2 类型 [NOT NULL]
....);

2.修改表中字段
ALTER TABLE <table_name> modify(字段1 类型,
字段2 类型
....);

3 .删除表中字段
ALTER TABLE <table_name> drop(字段1,
字段2
....);

4 .修改表的名称
RENAME <table_name> to <new table_name>;

5 .对已经存在的表添加约束
ALTER TABLE <table_name> ADD CONSTRAINT <constraint_name> 约束类型 (针对的字段名);
示例:
Alter table emp add constraint S_F Foreign key (deptno) references dept(deptno);

6 .对表里的约束禁用;
ALTER TABLE <table_name> DISABLE CONSTRAINT <constraint_name>;

7 .对表里的约束重新启用;
ALTER TABLE <table_name> ENABLE CONSTRAINT <constraint_name>;

8 .删除表中约束
ALTER TABLE <table_name> DROP CONSTRAINT <constraint_name>;
示例:
ALTER TABLE emp drop CONSTRAINT <Primary key>;

删除表

DROP TABLE <table_name>;

示例
drop table emp;

删除索引

DROP INDEX <index_name>;

示例
drop index i_1;

删除同义词

DROP SYNONYM <synonym_name>;

示例
drop synonym mm;

插入记录

INSERT INTO table_name (column1,column2,...)
values (value1,value2, ...);

示例
insert into emp (empno,ename) values(9500,'AA');

把 一个表中的数据插入另一个表中

INSERT INTO <table_name> <SELECT 语句>
示例
create table a as select * from emp where 1=2;
insert into a select * from emp where sal>2000;

查询记录

一般查询
SELECT [DISTINCT] <column1 [as new name] ,columns2,...>
FROM <table1>
[WHERE <条件>]
[GROUP BY <column_list>]
[HAVING <条件>]
[ORDER BY <column_list> [ASC|DESC]]

DISTINCT --表示隐藏重复的行
WHERE --按照一定的条件查找记录
GROUP BY --分组查找(需要汇总时使用)
HAVING --分组的条件
ORDER BY --对查询结果排序

要显示全部的列可以用*表示
示例:
select * from emp;

WHERE 语句的运算符
where <条件 1 >**AND**<条件 2 > --两个条件都满足
示例:
select * from emp where deptno=10 and sal>1000;

where <条件 1 >**OR**<条件 2 > --两个条件中有一个满足即可
示例:
select * from emp where deptno=10 OR sal>2000;

where **NOT** <条件> --不满足条件的
示例:
select * from emp where not deptno=10;

where **IN**(条件列表) --所有满足在条件列表中的记录
示例:
select * from emp where empno in(7788,7369,7499);

where **BETWEEN** .. **AND** .. --按范围查找
示例:

select * from emp where sal between 1000 and 3000;

where 字段 **LIKE** --主要用与字符类型的字段
示例1:
select * from emp where ename like '_C%'; --查询姓名中第二个字母是'C'的人
'-' 表示任意字符;
'%' 表示多字符的序列;

where 字段 **IS [NOT] NULL** --查找该字段是[不是]空的记录

汇总数据是用的函数
SUM --求和
示例:
select deptno,sum(sal) as sumsal from emp GROUP BY deptno;

AVG --求平均值
MAX --求最大值
MIN --求最小值
COUNT --求个数

子查询
SELECT <字段列表> from <table_name> where 字段 运算符(<SELECT 语句>);

示例:
select * from emp where sal=(select max(sal) from emp);

运算符
Any
示例:
select * from emp where sal>ANY(select sal from emp where deptno=30) and deptno<>30;
--找出比deptno=30的员工最低工资高的其他部门的员工

ALL
select * from emp where sal>ALL(select sal from emp where deptno=30) and deptno<>30;
--找出比deptno=30的员工最高工资高的其他部门的员工

连接查询
SELECT <字段列表> from <table1,table2> WHERE table1.字段[(+)]=table2.字

段[(+)]

示例
select empno,ename,dname from emp,dept where emp.deptno=dept.deptno;

查询指定行数的数据
SELECT <字段列表> from <table_name> WHERE ROWNUM<行数;
示例:
select * from emp where rownum<=10;--查询前10行记录
注意ROWNUM只能为1 因此不能写 select * from emp where rownum between 20 and 30;

要查第几行的数据可以使用以下方法:
select * from emp where rownum<=3 and empno not in (select empno from emp where rownum<=3);
结果可以返回整个数据的3-6行;
不过这种方法的性能不高；如果有别的好方法请告诉我。

更新数据

UPDATE table_name set column1=new value,column2=new value,...
WHERE <条件>

示例
update emp set sal=1000,empno=8888 where ename='SCOTT'

更新数据

DELETE FROM <table_name>
WHERE <条件>

示例
delete from emp where empno='7788'

数据控制语言

1.授权
GRANT <权限列表> to <user_name>;

2.收回权限
REVOKE <权限列表> from <user_name>

Oracle 的权限列表
connect 连接
resource 资源
unlimited tablespace 无限表空间
dba 管理员
session 会话

数据控制语言

1.COMMIT 提交;

2.ROLLBACK [TO savepoint]　回滚;

3.SAVEPOINT <savepoint> 保存位置。

创建视图

CREATE [OR REPLACE] VIEW <view_name>
AS
<SELECT 语句>;

OR REPLACE --表示替换以有的视图

删除视图

DROP VIEW <view_name>

创建序列

CREATE SEQUENCE <sequencen_name>
INCREMENT BY n
START WITH n
[MAXVALUE n][MINVALUE n]
[CYCLE|NOCYCLE]
[CACHE n|NOCACHE];

INCREMENT BY n --表示序列每次增长的幅度;默认值为1.
START WITH n --表示序列开始时的序列号。默认值为1.
MAXVALUE n --表示序列可以生成的最大值(升序).
MINVALUE n --表示序列可以生成的最小值(降序).
CYCLE --表示序列到达最大值后，在重新开始生成序列.默认值为NOCYCLE。
CACHE --允许更快的生成序列.

示例:
create sequence se_1
increment by 1
start with 100
maxvalue 999999
cycle;

修改序列

ALTER SEQUENCE <sequencen_name>
INCREMENT BY n
START WITH n
[MAXVALUE n][MINVALUE n]
[CYCLE|NOCYCLE]
[CACHE n|NOCACHE];

删除序列

DROP SEQUENCE <sequence_name>

使用序列

1.CURRVAL
返回序列的当前值.
注意在刚建立序列后,序列的CURRVAL值为NULL，所以不能直接使用。
可以先初始化序列:
方法:select <sequence_name>.nextval from dual;
示例:select se_1.nextval from dual;
之后就可以使用CURRVAL属性了

2.NEXTVAL
返回序列下一个值;
示例:

```
begin
for i in 1..5
loop
insert into emp(empno) values(se_1.nextval);
end loop;
end;
```

查看序列的当前值
select <sequence_name>.currval from dual;

示例:select se_1.currval from dual;

创建用户

CREATE USER <user_name> [profile "DEFAULT"]
identified by "<password>" [default tablespace "USERS"]

删除用户

DROP USER <user_name> CASCADE

创建角色

CREATE ROLE <role_name>
identified by "<password>"

删除角色

DROP ROLE <role_name>

PL/SQL 结构

DECLARE　　--声明部分
　　声明语句
BEGIN　　--执行部分
　　执行语句

EXCEPTION　　--异常处理部分
　　执行语句

END;

变量声明
<变量名> 类型[:=初始值];
特殊类型 字段%type
示例: name emp.ename%type --表示name的类型和emp.ename的类型相同
表 %rowtype
示例: test emp%rowtype --表示test的类型为emp表的行类型;也有 .empno; .ename; .sal ;等属性

常量声明
<变量名> CONSTANT 类型:=初始值;
示例: pi constant number(5,3):=3.14;

全局变量声明
VARIABLE <变量名> 类型;
示例: VARIABLE num number;

使用全局变量
:<变量名>
示例:
:num:=100;
i=:num;

查看全局变量的值
print <变量名>
示例: print num;

赋值运算符: **:=**
示例: num := 100;

使用SELECT <列名> INTO <变量名> FROM <表名> WHERE <条件>
注意select into 语句的返回结果只能为一行;
示例: test emp%rowtype;
select * into test from emp where empno=7788;

用户交互输入
<变量>:='&变量'

示例:
num:=#

注意oracle的用户交互输入是先接受用户输入的所有值后在执行语句;
所以不能使用循环进行用户交互输入;

条件控制语句

```
IF <条件1> THEN
   语句
[ELSIF <条件2> THEN
  语句
        .
        .
        .
ELSIF <条件n> THEN
  语句]
[ELSE
   语句]
END IF;
```

循环控制语句

1.LOOP

```
LOOP
  语句;
  EXIT WHEN <条件>
END LOOP;
```

2.WHILE LOOP

```
WHILE <条件>
LOOP
  语句;
END LOOP;
```

3.**FOR**

```
FOR <循环变量> IN 下限..上限
LOOP
  语句;
END LOOP;
```

NULL 语句

null;
表示没有操作;

注释使用
单行注释： --
多行注释： /* .....

.....*/

异常处理

```
EXCEPTION
  WHEN <异常类型> THEN
        语句;
  WHEN OTHERS THEN
        语句;
END;
```

关于异常类型请查看附录.

显示游标

定义:CURSOR <游标名> IS <SELECT 语句> [FOR UPDATE | FOR UPDATE OF 字段];

[FOR UPDATE | FOR UPDATE OF 字段] --给游标加锁,既是在程序中
有"UPDATE","INSERT","DELETE"语句对数据库操作时。
游标自动给指定的表或者字段加锁，防止同时有别的程序对指定的表或字段进
行"UPDATE","INSERT","DELETE"操作.
在使用"DELETE","UPDATE"后还可以在程序中使用CURRENT OF <游标名> 子句引用当前行.

操作:OPEN <游标名> --打开游标
FETCH <游标名> INTO 变量1,变量2,变量3,....变量n,;
或者
FETCH <游标名> INTO 行对象; --取出游标当前位置的值
CLOSE <游标名> --关闭游标
属性: %NOTFOUND --如果FETCH语句失败，则该属性为"TRUE"，否则为"FALSE";
%FOUND --如果FETCH语句成果，则该属性为"TRUE"，否则为"FALSE";
%ROWCOUNT --返回游标当前行的行数;
%ISOPEN --如果游标是开的则返回"TRUE"，否则为"FALSE";

使用:
LOOP循环
示例:

```
DECLARE
    cursor c_1 is select * from emp;    --定义游标
    r c_1%rowtype;     --定义一个行对象，用于获得游标的值
BEGIN
    if c_1%isopen then
        CLOSE c_1;
    end if;
    OPEN c_1;         --判断游标是否打开.如果开了将其关闭，然后在打开
    dbms_output.put_line('行号 姓名 薪水');
    LOOP
    FETCH c_1 INTO r;    --取值
    EXIT WHEN c_1%NOTFOUND;    --如果游标没有取到值，退出循环.
    dbms_output.put_line(c_1%rowcount||' '||r.ename||' '||r.sal);   --输出结果,需要 set serverout on 才
能显示.
    END LOOP;
END;
```

FOR循环
示例:

```
DECLARE
    cursor c_1 is select ename,sal from emp;    --定义游标
BEGIN
    dbms_output.put_line('行号 姓名 薪水');
```

```
    FOR i IN c_1        --for循环中的循环变量i为c_1%rowtype类型;
    LOOP
    dbms_output.put_line(c_1%rowcount||' '||i.ename||' '||i.sal);    --输出结果,需要 set serverout on 才能显示.
    END LOOP;
END;
```

for循环使用游标是在循环开始前自动打开游标，并且自动取值到循环结束后，自动关闭游标.

游标加锁示例:

```
DECLARE
    cursor c_1 is select ename,sal from emp for update of sal;    --定义游标对emp表的sal字段加锁.

BEGIN
    dbms_output.put_line('行号 姓名 薪水');
    FOR i IN c_1        --for循环中的循环变量i为c_1%rowtype类型;
    LOOP
    UPDATE EMP set sal=sal+100 WHERE CURRENT OF c_1; --表示对当前行的sal进行跟新.
    END LOOP;
    FOR i IN c_1
    LOOP
    dbms_output.put_line(c_1%rowcount||' '||i.ename||' '||i.sal);    --输出结果,需要 set serverout on 才能显示.
    END LOOP;
END;
```

代参数的游标

定义:CURSOR <游标名>(参数列表) IS <SELECT 语句> [FOR UPDATE | FOR UPDATE OF 字段];

示例:

```
DECLARE
    cursor c_1(name emp.ename%type) is select ename,sal from emp where ename=name;    --定义游标
BEGIN
    dbms_output.put_line('行号 姓名 薪水');
    FOR i IN c_1('&name')        --for循环中的循环变量i为c_1%rowtype类型;
    LOOP
    dbms_output.put_line(c_1%rowcount||' '||i.ename||' '||i.sal);    --输出结果,需要 set serverout on 才能显示.
    END LOOP;
END;
```

隐试游标

隐试游标游标是系统自动生成的。每执行一个DML语句就会产生一个隐试游标，起名字为SQL;

隐试游标不能进行"OPEN" ,"CLOSE","FETCH"这些操作;

属性:
%NOTFOUND --如果DML语句没有影响到任何一行时，则该属性为"TRUE"，否则为"FALSE";
%FOUND --如果DML语句影响到一行或一行以上时，则该属性为"TRUE"，否则为"FALSE";
%ROWCOUNT --返回游标当最后一行的行数;

个人认为隐试游标的作用是判断一个DML语句;
示例:

```
BEGIN
   DELETE FROM EMP WHERE empno=&a;
   IF SQL%NOTFOUND THEN
      dbms_output.put_line('empno不存在');
   END IF;
  IF SQL%ROWCOUNT>0 THEN
      dbms_output.put_line('删除成功');
   END IF;
END;
```

PL/SQL表

pl/sql表只有两列,其中第一列为序号列为INTEGER类型,第二列为用户自定义列.

定义:TYPE <类型名> IS TABLE OF <列的类型> [NOT NULL] INDEX BY BINARY_INTEGER;
<列的类型>可以为Oracle的数据类行以及用户自定义类型;

属性方法:
.count --返回pl/sql表的总行数
.delect --删除pl/sql表的所有内容
.delect(行数) --删除pl/sql表的指定的行
.delct(开始行，结束行) --删除pl/sql表的多行
.first --返回表的第一个INDEX;
.next(行数) --这个行数的下一条的INDEX;
.last --返回表的最后一个INDEX;

使用
示例:

```
DECLARE
    TYPE mytable IS TABLE OF VARCHAR2(20) index by binary_integer; --定义一个名为mytable
的PL/sql表类型;
    cursor c_1 is select ename from emp;
    n number:=1;
    tab_1 mytable; --为mytable类型实例化一个tab_1对象;
BEGIN
    for i in c_1
    loop
        tab_1(n):=i.ename; --将得到的值输入pl/sql表
        n:=n+1;
   end loop;
    n:=1;
    tab_1.delete(&要删除的行数); --删除pl/sql表的指定行
    for i in tab_1.first..tab_1.count
    loop
        dbms_output.put_line(n||' '||tab_1(n)); --打印pl/sql表的内容
        n:=tab_1.next(n);
    end loop;
EXCEPTION
    WHEN NO_DATA_FOUND THEN                --由于删除了一行，会发生异常，下面语句可
以接着删除的行后显示
        for i in n..tab_1.count+1
    loop
        dbms_output.put_line(n||' '||tab_1(n));
        n:=tab_1.next(n);
    end loop;
END;
```

PL/SQL记录

pl/sql表只有一行,但是有多列。

定义:TYPE <类型名> IS RECORD <列名1 类型1,列名2 类型2,...列名n 类型n,> [NOT NULL]
<列的类型>可以为Oracle的数据类行以及用户自定义类型;可以是记录类型的嵌套

使用
示例:

```
DECLARE
    TYPE myrecord IS RECORD(id emp.empno%type,
    name emp.ename%type,sal emp.sal%type);     --定义一个名为myrecoed的PL/sql记录类型;
    rec_1 myrecord; --为myrecord类型实例化一个rec_1对象;
BEGIN
        select empno,ename,sal into rec_1.id,rec_1.name,rec_1.sal
        from emp where empno=7788;        --将得到的值输入pl/sql记录
        dbms_output.put_line(rec_1.id||' '||rec_1.name||' '||rec_1.sal); --打印pl/sql记录的内容
END;
```

结合使用PL/SQL表和PL/SQL记录
示例:

```
DECLARE
    CURSOR c_1 is select empno,ename,job,sal from emp;
    TYPE myrecord IS RECORD(empno emp.empno%type,ename emp.ename%type,
job emp.job%type,sal emp.sal%type);     --定义一个名为myrecoed的PL/sql记录类型;
    TYPE mytable IS TABLE OF myrecord index by binary_integer;
                                            --定义一个名为mytable的PL/sql表类型;字段类型为
PL/sql记录类型;

    n number:=1;
    tab_1 mytable; --为mytable类型实例化一个tab_1对象;
BEGIN
        --赋值
        for i in c_1
        loop
            tab_1(n).empno:=i.empno;
            tab_1(n).ename:=i.ename;
            tab_1(n).job:=i.job;
            tab_1(n).sal:=i.sal;
            n:=n+1;
        end loop;
        n:=1;
        --输出
        for i in n..tab_1.count
        loop
```

```
            dbms_output.put_line(il|| ' ||tab_1(i).empno
            ||' ||tab_1(i).ename||' ||tab_1(i).job||' ||tab_1(i).sal);
        end loop;
END;
```

强型REF游标

定义:TYPE <游标名> IS REF CURSOR RETURN<返回类型>;

操作:OPEN <游标名> For <select 语句> --打开游标
FETCH <游标名> INTO 变量1,变量2,变量3,....变量n,;
或者
FETCH <游标名> INTO 行对象; --取出游标当前位置的值
CLOSE <游标名> --关闭游标
属性: %NOTFOUND --如果FETCH语句失败，则该属性为"TRUE"，否则为"FALSE";
%FOUND --如果FETCH语句成果，则该属性为"TRUE"，否则为"FALSE";
%ROWCOUNT --返回游标当前行的行数;
%ISOPEN --如果游标是开的则返回"TRUE"，否则为"FALSE";

使用:
示例:

```
DECLARE
    type c_type is ref cursor return emp%rowtype;    --定义游标
    c_1 c_type;    --实例化这个游标类型
    r emp%rowtype;
BEGIN
    dbms_output.put_line('行号 姓名 薪水');
    open c_1 for select * from emp;
    loop
    fetch c_1 into r;
    exit when c_1%notfound;
    dbms_output.put_line(c_1%rowcount||' '||r.ename||' '||r.sal);   --输出结果,需要 set serverout on 才能显示.
    END LOOP;
close c_1;
END;
```

弱型REF游标

定义:TYPE <游标名> IS REF CURSOR;

操作:OPEN <游标名> For <select 语句> --打开游标
FETCH <游标名> INTO 变量1,变量2,变量3,....变量n,;
或者
FETCH <游标名> INTO 行对象; --取出游标当前位置的值
CLOSE <游标名> --关闭游标
属性: %NOTFOUND --如果FETCH语句失败，则该属性为"TRUE"，否则为"FALSE";
%FOUND --如果FETCH语句成果，则该属性为"TRUE"，否则为"FALSE";

%ROWCOUNT --返回游标当前行的行数;

%ISOPEN --如果游标是开的则返回"TRUE"，否则为"FALSE";

示例:

```
set autoprint on;
var c_1 refcursor;
DECLARE
  n number;
BEGIN
  n:=&请输入;
  if n=1 then
       open :c_1 for select * from emp;
  else
       open :c_1 for select * from dept;
  end if;
END;
```

过程

定义:CREATE [OR REPLACE] PROCEDURE <过程名>[(参数列表)] IS
[局部变量声明]
BEGIN
可执行语句
EXCEPTION
异常处理语句
END [<过程名>];

变量的类型:in 为默认类型,表示输入; out 表示只输出;in out 表示即输入又输出;

操作以有的过程:在PL/SQL块中直接使用过程名;在程序外使用execute <过程名>[(参数列表)]

使用:
示例:
创建过程:

```
create or replace procedure p_1(n in out number) is
   r emp%rowtype;
BEGIN
    dbms_output.put_line('姓名 薪水');
    select * into r from emp where empno=n;
    dbms_output.put_line(r.ename|| ' '||r.sal);    --输出结果,需要 set serverout on 才能显示.
   n:=r.sal;
END;
```

使用过程:

```
declare
   n number;
begin
   n:=&请输入员工号;
   p_1(n);
   dbms_output.put_line('n的值为 '||n);
end;
```

删除过程:
DROP PROCEDURE <过程名>;

函数

定义:CREATE [OR REPLACE] FUNCTION <过程名>[(参数列表)] RETURN 数据类型 IS
[局部变量声明]
BEGIN
可执行语句
EXCEPTION
异常处理语句
END [<过程名>];

变量的类型:in 为默认类型,表示输入; out 表示只输出;in out 表示即输入又输出;

使用:

示例:

创建函数:

```
create or replace function f_1(n number) return number is
   r emp%rowtype;
BEGIN
    dbms_output.put_line('姓名 薪水');
    select * into r from emp where empno=n;
    dbms_output.put_line(r.ename||' '||r.sal);    --输出结果,需要 set serverout on 才能显示.
    return r.sal;
END;
```

使用函数:

```
declare
   n number;
    m number;
begin
   n:=&请输入员工号;
   m:=f_1(n);
   dbms_output.put_line('m的值为 '||m);
end;
```

删除函数:

DROP FUNCTION <函数名>;

数据包

定义：

定义包的规范

```
CREATE [OR REPLACE] PACKAGE <数据包名> AS
        --公共类型和对象声明
        --子程序说明
END;
```

定义包的主体

```
CREATE [OR REPLACE] PACKAGE BODY <数据包名> AS
        --公共类型和对象声明
        --子程序主体
BEGIN
     -初始化语句
END;
```

使用：

示例:

创建数据包规范:

```
create or replace package pack_1 as
    n number;
    procedure p_1;
    FUNCTION f_1 RETURN number;
end;
```

创建数据包主体:

```
create or replace package body pack_1 as
  procedure p_1 is
     r emp%rowtype;
  begin
     select * into r from emp where empno=7788;
     dbms_output.put_line(r.empnoll' 'llr.enamell' 'llr.sal);
   end;

   FUNCTION f_1 RETURN number is
      r emp%rowtype;
   begin
      select * into r from emp where empno=7788;
     return r.sal;
   end;
end;
```

使用包:

```
declare
    n number;
begin
    n:=&请输入员工号;
    pack_1.n:=n;
    pack_1.p_1;
```

```
    n:=pack_1.f_1;
    dbms_output.put_line('薪水为 '||n);
end;
```

在包中使用REF游标

示例:

创建数据包规范:

```
create or replace package pack_2 as
   TYPE c_type is REF CURSOR; --建立一个ref游标类型
   PROCEDURE p_1(c1 in out c_type); --过程的参数为ref游标类型;
end;
```

创建数据包主体:

```
create or replace package body pack_2 as
  PROCEDURE p_1(c1 in out c_type) is
  begin
     open c1 for select * from emp;
   end;
end;
```

使用包:

```
var c_1 refcursor;
set autoprint on;
execute pack_2.p_1(:c_1);
```

删除包:

DROP PACKAGE <包名>;

触发器

创建触发器:

```
CREATE [OR REPLACE] TRIGGER <触发器名>
BEFORE|AFTER
INSERT|DELETE|UPDATE [OF <列名>] ON <表名>
[FOR EACH ROW]
 WHEN (<条件>)
 <pl/sql块>
```

关键字"BEFORE"在操作完成前触发;"AFTER"则是在操作完成后触发;
关键字"FOR EACH ROW"指定触发器每行触发一次.
关键字"OF <列名>" 不写表示对整个表的所有列.
WHEN (<条件>)表达式的值必须为"TRUE".

特殊变量:

:new --为一个引用最新的列值;
:old --为一个引用以前的列值;

这些变量只有在使用了关键字 "FOR EACH ROW"时才存在.且update语句两个都有,而insert只有:new ,delect 只有:old;

使用RAISE_APPLICATION_ERROR

语法:RAISE_APPLICATION_ERROR(错误号(-20000到-20999),消息[,{truelfalse}]);
抛出用户自定义错误.
如果参数为'TRUE',则错误放在先前的堆栈上.

INSTEAD OF 触发器

INSTEAD OF 触发器主要针对视图（VIEW)将触发的dml语句替换成为触发器中的执行语句，而不执行dml语句.

禁用某个触发器

```
ALTER TRIGGER <触发器名> DISABLE
```

重新启用触发器

```
ALTER TRIGGER <触发器名> ENABLE
```

禁用所有触发器

```
ALTER TRIGGER <触发器名> DISABLE ALL TRIGGERS
```

启用所有触发器

```
ALTER TRIGGER <触发器名> ENABLE ALL TRIGGERS
```

删除触发器

```
DROP TRIGGER <触发器名>
```

自定义对象

创建对象:

CREATE [OR REPLACE] TYPE <对象名> AS OBJECT(
属性1 类型
属性2 类型
.
.
方法1的规范(MEMBER PROCEDURE <过程名>
方法2的规范 (MEMBER FUNCTION <函数名> RETURN 类型)
.
.
PRAGMA RESTRIC_REFERENCES(<方法名>,WNDS/RNDS/WNPS/RNPS);
关键字"PRAGMA RESTRIC_REFERENCES"通知ORACLE函数按以下模式之一操作;
WNDS-不能写入数据库状态;
RNDS-不能读出数据库状态;
WNPS-不能写入包状态;
RNDS-不能读出包状态;

创建对象主体:

CREATE [OR REPLACE] TYPE body <对象名> AS
方法1的规范(MEMBER PROCEDURE <过程名> is <PL/SQL块>
方法2的规范 (MEMBER FUNCTION <函数名> RETURN 类型 is <PL/SQL块>
END;

使用MAP方法和ORDER方法

用于对自定义类型排序。每个类型只有一个MAP或ORDER方法。
格式:MAP MEMBER FUNCTION <函数名> RETURN 类型
ORDER MEMBER FUNCTION <函数名> RETURN NUMBER

创建对象表

CREATE TABLE <表名> OF <对象类型>

示例:

1. 创建name 类型

```
create or replace type name_type as object(
    f_name varchar2(20),
    l_name varchar2(20),
    map member function name_map return varchar2);

create or replace type body name_type as
    map member function name_map return varchar2 is --对f_name和l_name排序
    begin
        return f_name||l_name;
    end;
    end;
```

2 创建address 类型

```
create or replace type address_type as object
 ( city varchar2(20),
```

```
    street varchar2(20),
    zip number,
order member function address_order(other address_type) return number);

create or replace type body address_type as
order member function address_order(other address_type) return number is --对zip排序
begin
    return self.zip-other.zip;
end;
end;
```

3 创建stu对象

```
 create or replace type stu_type as object (
 stu_id number(5),
 stu_name name_type,
 stu_addr address_type,
 age number(3),
 birth date,
 map member function stu_map return number,
member procedure update_age);

create or replace type body stu_type as
 map member function stu_map return number is --对stu_id排序
 begin
      return stu_id;
 end;
member procedure update_age is --求年龄用现在时间-birth
 begin
      update student set age=to_char(sysdate,'yyyy')-to_char(birth,'yyyy') where stu_id=self.stu_id;
 end;
 end;
```

4. 创建对象表

```
create table student of stu_type(primary key(stu_id));
```

5.向对象表插值

```
 insert into student values(1,name_type('关','羽'),address_type('武汉','成都路',43000), null,sysdate-365*20);
```

6.使用对象的方法

```
delcare
    aa stu_type;
begin
    select value(s) into aa from student s where stu_id=1; --value()将对象表的每一行转成行对象 括号中必须为表的别名
    aa.update_age();
end;
```

7.select stu_id,s.stu_name.f_name,s.stu_name.l_name from student s; --查看类型的值

8.select ref(s) from student s ; --ref()求出行对象的OID,括号中必须为表的别名;deref()将oid变成行队像;

命令	描述
DESC 表名	查看表的信息.
SET SERVEROUT [ON|OFF]	设置系统输出的状态.
SET PAGESIZE <大小>	设置浏览中没页的大小
SET LINESIZE <大小>	设置浏览中每行的长度
SET AUTOPRINT [ON|OFF]	设置是否自动打印全局变量的值
SELECT SYSDATE FROM DUAL	查看当前系统时间
ALTER SESSION SET nls_date_format='格式'	设置当前会话的日期格式 示例:ALTER SESSION SET nls_date_format='dd-mon-yy hh24:mi:ss'
SELECT * FROM TAB	查看当前用户下的所有表
SHOW USER	显示当前用户
HELP TOPIC	显示有那些命令
SAVE <file_name>	将buf中的内容保存成一个文件
RUN <file_name>	执行已经保存的文件:也可以写成@<file_name>
GET <file_name>	显示文件中的内容
LIST	显示buf中的内容
ED	用记事本打开buf,可以进行修改
DEL 行数	删除buf中的单行
DEL 开始行 结束行	删除buf中的多行
INPUT 字符串	向buf中插入一行
APPEND 字符串	将字符串追加到当前行
C/以前的字符串/替换的字符串	修改buf中当前行的内容
CONNECT	连接
DISCONNECT	断开连接
QUIT	退出sql*plus
EXP	导出数据库（可以在DOS键入exp help=y 可以看到详细说明) 示例: exp scott/tiger full=y file=e:\a.dmp; --导出scott下的所有东西 exp scott/tiger tables=(emp,dept) file=e:\emp.dmp --导出scott下的 emp,dept表
IMP	导入数据库（可以在DOS键入imp help=y 可以看到详细说明) imp scott/tiger tables=(emp,dept) file=e:\emp.dmp

可以通过help <命令>获得命令的帮助

异常	描述
CURSOR_ALREADY_OPEN	试图"OPEN"一个已经打开的游标
DUP_VAL_ON_INDEX	试图向有"UNIQUE"中插入重复的值
INVALID_CURSOR	试图对以关闭的游标进行操作
INVALID_NUMBER	在SQL语句中将字符转换成数字失败
LOGIN_DENIED	使用无效用户登陆
NO_DATA_FOUND	没有找到数据时
NOT_LOGIN_ON	没有登陆Oracle就发出命令时
PROGRAM_ERROR	PL/SQL存在诸如某个函数没有"RETURN"语句等内部问题
STORAGE_ERROR	PL/SQL耗尽内存或内存严重不足
TIMEOUT_ON_RESOURCE	Oracle等待资源期间发生超时
TOO_MANY_ROWS	"SELECT INTO"返回多行时
VALUE_ERROR	当出现赋值错误
ZERO_DIVIDE	除数为零

字符函数

名称	描述
CONCAT(字符串1,字符串2)	将字符串1和字符串2连接成一个新的字符串 示例: select CONCAT(job,ename) from emp
LPAD(字段,总的大小,添充字符)	左填充即向右对齐 示例: select empno,lpad(sal,10,'*') from emp
RPAD(字段,总的大小,添充字符)	右填充即向左对齐 示例: select empno,rpad(sal,10) from emp
LOWER(字符串)	将字符串全部变成小写;
UPPER(字符串)	将字符串全部变成大写;
INITCAP(字符串)	将字符串变成第一个字母大写,其余都变成小写;
LENGTH(字符串)	求出字符串的长度;
SUBSTR(字符串,开始位置,长度)	从字符串中取子串; 示例: select substr(ename,2,3) from emp;--从ename的第2位开始取3位
INSTR(字符串,字符)	查看字符是否在字符串中存在;不存在返回0;存在则返回字符所在的的位置; 如果有两个以上的字符则返回第一个的位置. 示例:select instr(ename,'S') from emp;
TRIM(字符 FROM 字符串)	去掉字符串首尾的字符; 示例: select trim('S' from ename) from emp;
TO_CHAR()	将不是其他类型转成字符类型; 对于日期型可以控制其格式:TO_CHAR(日期,'格式'); 其中格式有: 'YYYY' --以4为显示年; 'YEAR' --以标准格式显示年; 'MM' ; 'MON' ; 'DD' ; 'DAY'; 'HH' ; 'MI' ;'SS'
REPLACE(字符串,字符串1,字符串2)	将字符串中的字符1替换成字符2; 示例: select replace(ename,'SC','SS') from emp;
TRANSLATE(字符串,字符串1,字符串2)	替换多的字符; 示例: select translate(ename,'SH','AB') from emp; --表示将ename中的'S'换成'A','H'换成'B';
ASCII(char)	求字符的ascii码
NLSSORT(字符串)	对字符串排序.

数学函数

名称	描述
ABS(数字)	一个数的绝对值
CEIL(数字)	向上取整;不论小数后的书为多少都要向前进位; CEIL(123.01)=124; CEIL(-123.99)=-123;

FLOOR(数字)	向下取整;不论小数后的书为多少都删除;l
	floor(123.99)=123;
	floor(-123.01)=-124;

MOD(被除数,除数)	取余数; MOD(20,3)=2
ROUND(数字,从第几为开始取)	四舍五入; ROUND(123.5,0)=124; ROUND(-123.5,0)=-124; ROUND(123.5,-2)=100; ROUND(-123.5,-2)=-100;
SIGN(数字)	判断是正数还是负数;正数返回1，负数返回-1，0返回0;
SQRT(数字)	对数字开方;
POWER(m,n)	求m的n次方;
TRUNC(数字,从第几位开始)	切数字; TRUNC(123.99,1)=123.9 TRUNC(-123.99,1)=-123.9 TRUNC(123.99,-1)=120 TRUNC(-123.99,-1)=-120 TRUNC(123.99)=123
GREATEST(数字列表)	找出数字列表中最大的数; 示例: select greatest(100,200,-100) from dual; --结果为200
LEAST(数字列表)	找出数字列表中最小的数;
SIN(n)	求n的正旋
COS(n)	求n的余旋
TAN(n)	求n的正切
ACos(n)	求n的反正切
ATAN(n)	求n的反正切
exp(n)	求n的指数
LN(n)	求n的自然对数,n必须大于0
LOG(m,n)	求n以m为底的对数,m和n为正数,且m不能为0

日期函数

名称	描述
ADD_MONTHS(日期,数字)	在以有的日期上加一定的月份; 示例: select add_months(hiredate,20),hiredate from emp;
LAST_DAY(日期)	求出该日期的最后一天.
MONTHS_BETWEEN(日期1，日期2)	求出两个月之间的天树（注意返回的天数为小数); 示例: select months_between(sysdate,hiredate) from emp;
NEW_TIME(时间,时区,'gmt')	按照时区设定时间.
NEXT_DAY(d,char)	返回d指定的日期之后并满足char指定条件的第一个日期

其他函数

名称	描述
VSIZE(类型)	求出数据类型的大小;
NVL(字符串,替换字符)	如果字符串为空则替换，否则不替换

其他

1.在PL/SQL中使用DDL

将sql语句赋给一个varchar2变量,在用execute immediate 这个varchar2变量即可;

示例:

```
declare
    str varchar2(200);
begin
    str:='create table test(id number,name varchar2(20))'; --创建表
    execute immediate str;
    str:='insert into test values(3,"c")'; --向表里插数据
    execute immediate str;
end;
```

但是要队这个表插入数据也必须使用execute immediate 字符变量

2.判断表是否存在;

示例:

```
declare
    n tab.tname%type;
begin
    select tname into n from tab where tname='&请输入表名';
    dbms_output.put_line('此表以存在');
exception
    when no_data_found then
    dbms_output.put_line('还没有此表');
end;
```

2.查看以有的过程;

示例:

```
select object_name,object_type,status from user_objects where object_type='PROCEDURE';
```

数据类型	描述
VARCHAR2(size)	可变长度的字符串,其最大长度为size个字节;size的最大值是4000,而最小值是1;你必须指定一个VARCHAR2的size;
NVARCHAR2(size)	可变长度的字符串,依据所选的国家字符集,其最大长度为size个字符或字节;size的最大值取决于储存每个字符所需的字节数,其上限为4000;你必须指定一个NVARCHAR2的size;
NUMBER(p,s)	精度为p并且数值范围为s的数值;精度p的范围从1到38;数值范围s的范围是从-84到127; 例如:NUMBER(5,2) 表示整数部分最大3位，小数部分为2位； NUMBER(5,-2) 表示数的整数部分最大为7其中对整数的倒数2位为0,前面的取整。 NUMBER 表示使用默认值,即等同于NUMBER(5);
LONG	可变长度的字符数据,其长度可达2G个字节;
DATE	有效日期范围从公元前4712年1月1日到公元后4712年12月31日
RAW(size)	长度为size字节的原始二进制数据,size的最大值为2000字节；你必须为RAW指定一个size;
LONG RAW	可变长度的原始二进制数据，其最长可达2G字节;
CHAR(size)	固定长度的字符数据,其长度为size个字节;size的最大值是2000字节,而最小值和默认值是1;
NCHAR(size)	也是固定长度。根据Unicode标准定义
CLOB	一个字符大型对象,可容纳单字节的字符;不支持宽度不等的字符集;最大为4G字节
NCLOB	一个字符大型对象,可容纳单字节的字符;不支持宽度不等的字符集;最大为4G字节;储存国家字符集
BLOB	一个二进制大型对象;最大4G字节
BFILE	包含一个大型二进制文件的定位器,其储存在数据库的外面；使得可以以字符流I/O访问存在数据库服务器上的外部LOB;最大大小为4G字节.