Oracle 存储过程总结

1、创建存储过程

```
create or replace procedure test(var_name_1 in type,var_name_2 out ty
pe) as
--声明变量(变量名 变量类型)
begin
--存储过程的执行体
end test;
```

打印出输入的时间信息

E.g:

```
create or replace procedure test(workDate in Date) is
begin
dbms_output.putline(The input date is:||to_date(workDate, yyyy-mm-d
d));
end test;
```

2、变量赋值

变量名 := 值;

E.g:

```
create or replace procedure test(workDate in Date) is
x number(4,2);
  begin
  x := 1;
end test;
```

3、判断语句:

if 比较式 then begin end; end if;

E.g

```
create or replace procedure test(x in number) is
begin
                if x >0 then
                  begin
                x := 0 - x;
```

```
                end;
        end if;
        if x = 0 then
              begin
                x: = 1;
        end;
        end if;
end test;
```

4、For 循环

```
For ... in ... LOOP
--执行语句
end LOOP;
```

(1)循环遍历游标

```
create or replace procedure test() as
Cursor cursor is select name from student;
name varchar(20);
begin
for name in cursor LOOP
begin
  dbms_output.putline(name);
end;
end LOOP;
end test;
```

(2)循环遍历数组

```
  create or replace procedure test(varArray in myPackage.TestArray) a
s
--(输入参数 varArray 是自定义的数组类型，定义方式见标题 6)
i number;
begin
i := 1;   --存储过程数组是起始位置是从 1 开始的，与 java、C、C++等语言不同。因为在 Oracle 中本是没有数组的概念的，数组其实就是一张
```

--表(Table),每个数组元素就是表中的一个记录,所以遍历数组时就相当于从表中的第一条记录开始遍历

```
for i in 1..varArray.count LOOP
dbms_output.putline(The No. || i ||record in varArray is: ||varArray(i));
  end LOOP;
end test;
```

5、While 循环

```
while 条件语句 LOOP
begin
end;
end LOOP;
```

E.g

```
create or replace procedure test(i in number) as
begin
while i < 10 LOOP
begin
  i:= i + 1;
end;
end LOOP;
  end test;
```

6、数组

首先明确一个概念：Oracle 中本是没有数组的概念的，数组其实就是一张表(Table),每个数组元素就是表中的一个记录。

使用数组时，用户可以使用 Oracle 已经定义好的数组类型，或可根据自己的需要定义数组类型。

(1)使用 Oracle 自带的数组类型

```
x array; --使用时需要需要进行初始化
```

e.g:

```
create or replace procedure test(y out array) is
  x array;
  begin
```

x := new array();

y := x;

end test;

(2)自定义的数组类型（自定义数据类型时，建议通过创建 Package 的方式实现，以便于管理）

E.g（自定义使用参见标题 4.2）create or replace package myPackage is

-- Public type declarations type info is record(name varchar(20), y number);

type TestArray is table of info index by binary_integer; --此处声明了一个 TestArray 的类型数据，其实其为一张存储 Info 数据类型的 Table 而已，及 TestArray 就是一张表，有两个字段，一个是

name，一个是 y。需要注意的是此处使用了 Index by binary_integer 编制该 Table 的索引项，也可以不写，直接写成：type TestArray is

table of info，如果不写的话使用数组时就需要进行初始化：varArray myPackage.TestArray; varArray := new myPackage.TestArray();

end TestArray;

7.游标的使用 Oracle 中 Cursor 是非常有用的，用于遍历临时表中的查询结果。其相关方法和属性也很多，现仅就常用的用法做一二介绍：

(1)Cursor 型游标(不能用于参数传递)

create or replace procedure test() is

cusor_1 Cursor is select std_name from student where ...; --Cursor 的使用方式 1 cursor_2 Cursor;

begin

select class_name into cursor_2 from class where ...; --Cursor 的使用方式 2

可使用 For x in cursor LOOP .... end LOOP; 来实现对 Cursor 的遍历

end test;

(2)SYS_REFCURSOR 型游标，该游标是 Oracle 以预先定义的游标，可作出参数进行传递

create or replace procedure test(rsCursor out SYS_REFCURSOR) is

cursor SYS_REFCURSOR; name varhcar(20);

begin

OPEN cursor FOR select name from student where ... --SYS_REFCURSOR 只能通过 OPEN 方法来打开和赋值

LOOP

fetch cursor into name --SYS_REFCURSOR 只能通过 fetch into 来打开和遍历 exit when cursor%NOTFOUND; --SYS_REFCURSOR 中可使用三个状态属性：

---%NOTFOUND(未找到记录信息) %FOUND(找到记录信息)

---%ROWCOUNT(然后当前游标所指向的行位置)

dbms_output.putline(name);

end LOOP;

rsCursor := cursor;

end test;

下面写一个简单的例子来对以上所说的存储过程的用法做一个应用：

现假设存在两张表，一张是学生成绩表(studnet)，字段为：stdId,math,article,language,music,sport,total,average,step

一张是学生课外成绩表(out_school)，字段为:stdId,parctice,comment

通过存储过程自动计算出每位学生的总成绩和平均成绩，同时，如果学生在课外课程中获得的评价为 A，就在总成绩上加 20 分。

```
create or replace procedure autocomputer(step in number) is
rsCursor SYS_REFCURSOR;
commentArray myPackage.myArray;
math number;
article number;
language number;
music number;
sport number;
total number;
average number;
stdId varchar(30);
record myPackage.stdInfo;
i number;
begin
i := 1;
```

```
get_comment(commentArray); --调用名为 get_comment()的存储过程获取学生课外评分信息
OPEN rsCursor for select stdId,math,article,language,music,sport from student t where t.step = step;
LOOP
fetch rsCursor into stdId,math,article,language,music,sport; exit when rsCursor%NOTFOUND;
total := math + article + language + music + sport;
for i in 1..commentArray.count LOOP
  record := commentArray(i);
if stdId = record.stdId then
  begin
  if record.comment = &apos;A&apos; then
    begin
  total := total + 20;
      go to next; --使用 go to 跳出 for 循环
   end;
end if;
end;
end if;
end LOOP;
<<continue>>   average := total / 5;
  update student t set t.total=total and t.average = average where t.stdId = stdId;
end LOOP;
end;
end autocomputer;
--取得学生评论信息的存储过程
create or replace procedure get_comment(commentArray out myPackage.myArray) is
rs SYS_REFCURSOR;
record myPackage.stdInfo;
stdId varchar(30);
```

```
comment varchar(1);
i number;
begin
open rs for select stdId,comment from out_school
i := 1;
LOOP
  fetch rs into stdId,comment; exit when rs%NOTFOUND;
record.stdId := stdId;
  record.comment := comment;
recommentArray(i) := record;
i:=i + 1;
end LOOP;
end get_comment;
--定义数组类型 myArray
create or replace package myPackage is begin
type stdInfo is record(stdId varchar(30),comment varchar(1));
type myArray is table of stdInfo index by binary_integer;
end myPackage;
```

项目中有涉及存储过程对字符串的处理，所以就将在网上查找到的资料汇总，做一个信息拼接式的总结。

以下信息均来自互联网，贴出来一则自己保存以待以后使用，一则供大家分享。

字符函数——返回字符值

这些函数全都接收的是字符族类型的参数(CHR 除外)并且返回字符值.

除了特别说明的之外,这些函数大部分返回 VARCHAR2 类型的数值.

字符函数的返回类型所受的限制和基本数据库类型所受的限制是相同的。

字符型变量存储的最大值：

VARCHAR2 数值被限制为 2000 字符(ORACLE 8 中为 4000 字符)

CHAR 数值被限制为 255 字符(在 ORACLE8 中是 2000)

long 类型为 2GB

Clob 类型为 4GB

1、CHR

语法: chr(x)

功能：返回在数据库字符集中与 X 拥有等价数值的字符。CHR 和 ASCII 是一对反函数。经过 CHR 转换后的字符再经过 ASCII 转换又得到了原来的字

符。

使用位置：过程性语句和 SQL 语句。

2、CONCAT

语法： CONCAT（string1,string2）

功能：返回 string1，并且在后面连接 string2。

使用位置：过程性语句和 SQL 语句。

3、INITCAP

语法：INITCAP（string）

功能：返回字符串的每个单词的第一个字母大写而单词中的其他字母小写的 string。单词是用.空格或给字母数字字符进行分隔。不是字母的

字符不变动。

使用位置：过程性语句和 SQL 语句。

4、LTRIM

语法：LTRIM（string1,string2）

功能：返回删除从左边算起出现在 string2 中的字符的 string1。String2 被缺省设置为单个的空格。数据库将扫描 string1，从最左边开始。当

遇到不在 string2 中的第一个字符，结果就被返回了。LTRIM 的行为方式与 RTRIM 很相似。

使用位置：过程性语句和 SQL 语句。

5、NLS_INITCAP

语法：NLS_INITCAP（string[,nlsparams]）

功能：返回字符串每个单词第一个字母大写而单词中的其他字母小写的 string，nlsparams 指定了不同于该会话缺省值的不同排序序列。如果不指定参数，则功能和 INITCAP 相同。Nlsparams 可以使用的形式是：

‘NLS_SORT=sort’

这里 sort 制订了一个语言排序序列。

使用位置：过程性语句和 SQL 语句。

6、NLS_LOWER

语法：NLS_LOWER（string[,nlsparams]）

功能：返回字符串中的所有字母都是小写形式的 string。不是字母的字符不变。

Nlsparams 参数的形式与用途和 NLS_INITCAP 中的 nlsparams 参数是相同的。如果 nlsparams 没有被包含，那么 NLS_LOWER 所作的处理和

LOWER 相同。

使用位置；过程性语句和 SQL 语句。

7、NLS_UPPER

语法：nls_upper（string[,nlsparams]）

功能：返回字符串中的所有字母都是大写的形式的 string。不是字母的字符不变。nlsparams 参数的形式与用途和 NLS_INITCAP 中的相同。如果

没有设定参数，则 NLS_UPPER 功能和 UPPER 相同。

使用位置：过程性语句和 SQL 语句。

8、REPLACE

语法：REPLACE（string，search_str[,replace_str]）

功能：把 string 中的所有的子字符串 search_str 用可选的 replace_str 替换，如果没有指定 replace_str，所有的 string 中的子字符串

search_str 都将被删除。REPLACE 是 TRANSLATE 所提供的功能的一个子集。

使用位置：过程性语句和 SQL 语句。

9、RPAD

语法：RPAD（string1,x[,string2]）

功能：返回在 X 字符长度的位置上插入一个 string2 中的字符的 string1。如果 string2 的长度要比 X 字符少，就按照需要进行复制。如果 string2

多于 X 字符，则仅 string1 前面的 X 各字符被使用。如果没有指定 string2，那么使用空格进行填充。X 是使用显示长度可以比字符串的实际长度

要长。RPAD 的行为方式与 LPAD 很相似，除了它是在右边而不是在左边进行填充。

使用位置：过程性语句和 SQL 语句。

10、RTRIM

语法：　RTRIM（string1,[,string2]）

功能：　返回删除从右边算起出现在 string1 中出现的字符 string2. string2 被缺省设置为单个的空格.数据库将扫描 string1,从右边开始.当遇

到不在 string2 中的第一个字符,结果就被返回了 RTRIM 的行为方式与 LTRIM 很相似.

使用位置：过程性语句和 SQL 语句。

11、SOUNDEX

语法：　SOUNDEX（string）

功能:　返回 string 的声音表示形式.这对于比较两个拼写不同但是发音类似的单词而言很有帮助.

使用位置：过程性语句和 SQL 语句。

12、SUBSTR

语法:　SUBSTR（string,a[,b]）

功能:　返回从字母为值 a 开始 b 个字符长的 string 的一个子字符串.如果 a 是 0,那么它就被认为从第一个字符开始.如果是正数,返回字符是从左

边向右边进行计算的.如果 b 是负数,那么返回的字符是从 string 的末尾开始从右向左进行计算的.如果 b 不存在,那么它将缺省的设置为整个字符

串.如果 b 小于 1,那么将返回 NULL.如果 a 或 b 使用了浮点数,那么该数值将在处理进行以前首先被却为一个整数.

使用位置：过程性语句和 SQL 语句。

13、TRANSLATE

语法:　TRANSLATE(string,from_str,to_str)

功能:　返回将所出现的 from_str 中的每个字符替换为 to_str 中的相应字符以后的 string. TRANSLATE 是 REPLACE 所提供的功能的一个超集.

如果 from_str 比 to_str 长,那么在 from_str 中而不在 to_str 中而外的字符将从 string 中被删除,因为它们没有相应的替换字符. to_str 不能为空

.Oracle 把空字符串认为是 NULL,并且如果 TRANSLATE 中的任何参数为 NULL,那么结果也是 NULL.

使用位置：过程性语句和 SQL 语句。

14、UPPER

语法: UPPER（string）

功能: 返回大写的 string.不是字母的字符不变.如果 string 是 CHAR 数据类型的,那么结果也是 CHAR 类型的.如果 string 是 VARCHAR2 类型的,那么

结果也是 VARCHAR2 类型的.

使用位置: 过程性语句和 SQL 语句。

字符函数——返回数字

这些函数接受字符参数回数字结果.参数可以是 CHAR 或者是 VARCHAR2 类型的.尽管实际下许多结果都是整数值,但是返回结果都是简单的 NUMBER

类型的,没有定义任何的精度或刻度范围.

16、ASCII

语法: ASCII（string）

功能：数据库字符集返回 string 的第一个字节的十进制表示.请注意该函数仍然称作为 ASCII.尽管许多字符集不是 7 位 ASCII.CHR 和 ASCII 是互为

相反的函数.CHR 得到给定字符编码的响应字符. ASCII 得到给定字符的字符编码.

使用位置：过程性语句和 SQL 语句。

17、INSTR

语法: INSTR（string1, string2[a,b]）

功能: 得到在 string1 中包含 string2 的位置. string1 时从左边开始检查的,开始的位置为 a,如果 a 是一个负数,那么 string1 是从右边开始进行

扫描的.第 b 次出现的位置将被返回. a 和 b 都缺省设置为 1,这将会返回在 string1 中第一次出现 string2 的位置.如果 string2 在 a 和 b 的规定下没有

找到,那么返回 0.位置的计算是相对于 string1 的开始位置的,不管 a 和 b 的取值是多少.

使用位置：过程性语句和 SQL 语句。

18、INSTRB

语法: INSTRB（string1, string2[a,[b]]）

功能: 和 INSTR 相同,只是操作的对参数字符使用的位置的是字节.

使用位置：过程性语句和 SQL 语句。

19、LENGTH

语法: LENGTH（string）

功能: 返回 string 的字节单位的长度.CHAR 数值是填充空格类型的,如果 string 由数据类型 CHAR,它的结尾的空格都被计算到字符串长度中间.

如果 string 是 NULL,返回结果是 NULL,而不是 0.

使用位置：过程性语句和 SQL 语句。

20、LENGTHB

语法: LENGTHB（string）

功能: 返回以字节为单位的 string 的长度.对于单字节字符集 LENGTHB 和 LENGTH 是一样的.

使用位置：过程性语句和 SQL 语句。

21、NLSSORT

语法：NLSSORT（string[,nlsparams]）

功能：得到用于排序 string 的字符串字节.所有的数值都被转换为字节字符串,这样在不同数据库之间就保持了一致性. Nlsparams 的作用和

NLS_INITCAP 中的相同.如果忽略参数,会话使用缺省排序.

使用位置：过程性语句和 SQL 语句。

oracle 存储过程的基本语法

1.基本结构

```
CREATE OR REPLACE PROCEDURE 存储过程名字
(
   参数 1 IN NUMBER,
   参数 2 IN NUMBER
) IS
变量 1 INTEGER :=0;
变量 2 DATE;
BEGIN

END 存储过程名字
```

2.SELECT INTO STATEMENT

将 select 查询的结果存入到变量中，可以同时将多个列存储多个变量中，必须有一条记录，否则抛出异常(如果没有记录抛出 NO_DATA_FOUND)

例子：

```
BEGIN
SELECT col1,col2 into 变量 1,变量 2 FROM typestruct where xxx;
EXCEPTION
WHEN NO_DATA_FOUND THEN
   xxxx;
END;
...
```

3.IF 判断

```
IF V_TEST=1 THEN
  BEGIN
    do something
  END;
END IF;
```

4.while 循环

```
WHILE V_TEST=1 LOOP
BEGIN
XXXX
END;
END LOOP;
```

5.变量赋值

```
V_TEST := 123;
```

6.用 for in 使用 cursor

```
...
IS
CURSOR cur IS SELECT * FROM xxx;
BEGIN
FOR cur_result in cur LOOP
BEGIN
 V_SUM :=cur_result.列名 1+cur_result.列名 2
END;
END LOOP;
END;
```

7.带参数的 cursor

```
CURSOR C_USER(C_ID NUMBER) IS SELECT NAME FROM USER WHERE
TYPEID=C_ID;
OPEN C_USER(变量值);
LOOP
FETCH C_USER INTO V_NAME;
EXIT FETCH C_USER%NOTFOUND;
  do something
END LOOP;
CLOSE C_USER;
```

8.用 pl/sql developer debug

连接数据库后建立一个 Test WINDOW

在窗口输入调用 SP 的代码,F9 开始 debug,CTRL+N 单步调试

关于 oracle 存储过程的若干问题备忘

1.在 oracle 中，数据表别名不能加 as，如：

```
select a.appname from appinfo a;-- 正确
select a.appname from appinfo as a;-- 错误
```

也许，是怕和 oracle 中的存储过程中的关键字 as 冲突的问题吧

2.在存储过程中，select 某一字段时，后面必须紧跟 into，如果 select 整个记录，利用游标的话就另当别论了。

```
  select af.keynode into kn from APPFOUNDATION af where af.appid=aid and af.foundationid=fid;-- 有 into，正确编译
  select af.keynode from APPFOUNDATION af where af.appid=aid and af.foundationid=fid;-- 没有 into，编译报错，提示：Compilation
  Error: PLS-00428: an INTO clause is expected in this SELECT statement
```

3.在利用 select...into...语法时，必须先确保数据库中有该条记录，否则会报出"no data found"异常。

可以在该语法之前，先利用 select count(*) from 查看数据库中是否存在该记录，如果存在，再利用 select...into...

4.在存储过程中，别名不能和字段名称相同，否则虽然编译可以通过，但在运行阶段会报错

```
 select keynode into kn from APPFOUNDATION where appid=aid and foundationid=fid;-- 正确运行
 select af.keynode into kn from APPFOUNDATION af where af.appid=appid and af.foundationid=foundationid;-- 运行阶段报错，提示
 ORA-01422:exact fetch returns more than requested number of rows
```

5.在存储过程中，关于出现 null 的问题

假设有一个表 A，定义如下：

```
create table A(
id varchar2(50) primary key not null,
vcount number(8) not null,
bid varchar2(50) not null -- 外键
);
```

如果在存储过程中，使用如下语句：

```
select sum(vcount) into fcount from A where bid='xxxxxx';
```

如果 A 表中不存在 bid="xxxxxx"的记录，则 fcount=null(即使 fcount 定义时设置了默认值，如：fcount number(8):=0 依然无效，fcount 还是会变成 null)，这样以后使用 fcount 时就可能有问题，所以在这里最好先判断一下：

```
if fcount is null then
   fcount:=0;
end if;
```

这样就一切 ok 了。

6.Hibernate 调用 oracle 存储过程

```
        this.pnumberManager.getHibernateTemplate().execute(
                new HibernateCallback() {
                  public Object doInHibernate(Session session)
                          throws HibernateException, SQLException {
                      CallableStatement cs = session
                              .connection()
                              .prepareCall("{call modifyapppnumber_remain(?)}");
                      cs.setString(1, foundationid);
                      cs.execute();
                      return null;
                  }
              });
```

oracle 存储过程语法总结及练习

```
--1. 存储过程之 if
clear;
```

```
create or replace procedure mydel(
in_a in integer)
as
begin
if in_a<100 then
dbms_output.put_line('小于100.');
elsif in_a<200 then
dbms_output.put_line('大于100小于200.');
else
dbms_output.put_line('大于200.');
end if;
end;
/

set serveroutput on;
begin
mydel(1102);
end;
/
```

```
--2.存储过程之case1
clear;
create or replace procedure mydel(
in_a in integer)
as
begin
case in_a
when 1 then
dbms_output.put_line('小于100.');
when 2 then
dbms_output.put_line('大于100小于200.');
else
dbms_output.put_line('大于200.');
end case;
end;
/

set serveroutput on;
begin
mydel(2);
end;
/
```

```
--1.存储过程之 loop1
clear;
create or replace procedure mydel(
in_a in integer)
as
a integer;
begin
a:=0;
loop
dbms_output.put_line(a);
a:=a+1;
exit when
a>301;
end loop;
end;
/


set serveroutput on;
begin
mydel(2);
end;
/
-----------------------------------------------
--1.存储过程之 loop2
clear;
create or replace procedure mydel(
in_a in integer)
as
a integer;
begin
a:=0;
while a<300 loop
dbms_output.put_line(a);
a:=a+1;
end loop;
end;
/


set serveroutput on;
begin
mydel(2);
```

```
end;
-----------------------------------------------
--1.存储过程之 loop3
clear;
create or replace procedure mydel(
in_a in integer)
as
a integer;
begin
for a in 0..300
loop
dbms_output.put_line(a);
end loop;
end;
/


set serveroutput on;
begin
mydel(2);
end;
/
clear;
select ename,cc:=(case
when comm=null then sal*12;
else (sal+comm)*12;
end case from emp order by salpersal;

-------------------------------------------------
clear;
create or replace procedure getstudentcomments(
i_studentid in int,o_comments out varchar)
as
exams_sat int;
avg_mark int;
tmp_comments varchar(100);
begin
select count(examid) into exams_sat from studentexam
where studentid=i_studentid;
if exams_sat=0 then
tmp_comments:='n/a-this student did not attend the exam!';
else
select avg(mark) into avg_mark from studentexam
where studentid=i_studentid;
```

```
case
when avg_mark<50 then tmp_comments:='very bad';
when avg_mark<60 then tmp_comments:='bad';
when avg_mark<70 then tmp_comments:='good';
end case;
end if;
o_comments:=tmp_comments;
end;
/


set serveroutput on;
declare
pp studentexam.comments%type;
begin
getstudentcomments(8,pp);
dbms_output.put_line(pp);
end;
/
------------------------------------------------------
```

```
delete from emp where empno<6000;
clear;
create or replace procedure insertdata(
in_num in integer)
as
myNum int default 0;
emp_no emp.empno%type:=1000;
begin
while myNum<in_num loop
insert into emp
values(emp_no,'hui'||myNum,'coder',7555,current_date,8000,6258,30);
emp_no:=emp_no+1;
myNum:=myNum+1;
end loop;
end;
/
```

```
set serveroutput on;
begin
insertdata(10);
end;
/
select * from emp;

------------------------------------------------------------

clear;
select studentname,averageMark,case
when averageMark<60 then '不及格'
when averageMark<70 then '考得好'
when averageMark<80 then '考得很好'
end case
from (select (
select b.name from student b where b.studentid=a.studentid) as studentname,
round(avg(mark),2) as averageMark from studentexam a group by a.studentid);
```

ORACLE 查询练习

emp 员工表(empno 员工号/ename 员工姓名/job 工作/mgr 上级编号/hiredate 受雇日期/sal 薪金/comm 佣金/deptno 部门编号)

dept 部门表(deptno 部门编号/dname 部门名称/loc 地点)

工资 ＝ 薪金 ＋ 佣金

1．列出至少有一个员工的所有部门。

2．列出薪金比“SMITH”多的所有员工。

3．列出所有员工的姓名及其直接上级的姓名。

4．列出受雇日期早于其直接上级的所有员工。

5．列出部门名称和这些部门的员工信息，同时列出那些没有员工的部门

6．列出所有“CLERK”（办事员）的姓名及其部门名称。

7．列出最低薪金大于 1500 的各种工作。

8．列出在部门“SALES”（销售部）工作的员工的姓名，假定不知道销售部的部门编号。

9．列出薪金高于公司平均薪金的所有员工。

10．列出与“SCOTT”从事相同工作的所有员工。

11．列出薪金等于部门 30 中员工的薪金的所有员工的姓名和薪金。

12．列出薪金高于在部门 30 工作的所有员工的薪金的员工姓名和薪金。

13．列出在每个部门工作的员工数量、平均工资和平均服务期限。

14．列出所有员工的姓名、部门名称和工资。

15．列出所有部门的详细信息和部门人数。

16．列出各种工作的最低工资。

17．列出各个部门的 MANAGER（经理）的最低薪金。

18．列出所有员工的年工资,按年薪从低到高排序。

--------1----------
```
select dname from dept where deptno in(
select deptno from emp);
```

--------2----------
```
select * from emp where sal>(
select sal from emp where ename='SMITH');
```

--------3----------
```
select a.ename,(
select ename from emp b where b.empno=a.mgr) as bossname from emp a;
```

--------4----------
```
select a.ename from emp a where a.hiredate<(
select hiredate from emp b where b.empno=a.mgr);
```

--------5----------
```
select a.dname,b.empno,b.ename,b.job,b.mgr,b.hiredate,b.sal,b.comm,b.deptno
from dept a left join emp b on a.deptno=b.deptno;
```

--------6----------

```
select a.ename,b.dname from emp a join dept b
on a.deptno=b.deptno and a.job='CLERK';
```

--------7----------

```
select distinct job as HighSalJob from emp group by job having min(sal)>1500;
```

--------8----------

```
select ename from emp where deptno=(
select deptno from dept where dname='SALES');
```

--------9----------

```
select ename from emp where sal>(
select avg(sal) from emp);
```

--------10---------

```
select ename from emp where job=(
select job from emp where ename='SCOTT');
```

--------11---------

```
select a.ename,a.sal from emp a where a.sal in (
select b.sal from emp b where b.deptno=30) and a.deptno<>30;
```

--------12---------

```
select ename,sal from emp where sal>(
select max(sal) from emp where deptno=30);
```

--------13---------

```
select
(select b.dname from dept b where a.deptno=b.deptno) as deptname ,
count(deptno) as deptcount,
avg(sal) as deptavgsal
from emp a group by deptno;
```

--------14---------

```
select
a.ename,
(select b.dname from dept b where b.deptno=a.deptno) as deptname,
sal
from emp a;
```

--------15---------
select
a.deptno,
a.dname,
a.loc,
(select count(deptno) from emp b where b.deptno=a.deptno group by b.deptno) as deptcount
from dept a;

--------16---------
select job,avg(sal) from emp group by job;

--------17---------
select deptno,min(sal) from emp where job='MANAGER' group by deptno;

--------18---------
select ename,(sal+nvl(comm,0))*12 as salpersal from emp order by salpersal;

ORACLE 子句查询,分组等

A.同表子查询作为条件
a. 给出人口多于 Russia(俄国)的国家名称 SELECT name FROM bbc
WHERE population>
(SELECT population FROM bbc
WHERE name='Russia')

b.给出'India'(印度), 'Iran'(伊朗)所在地区的所有国家的所有信息 SELECT * FROM bbc
WHERE region IN
(SELECT region FROM bbc
WHERE name IN ('India','Iran'))

c.给出人均 GDP 超过'United Kingdom'(英国)的欧洲国家. SELECT name FROM bbc
WHERE region='Europe' AND gdp/population >
(SELECT gdp/population FROM bbc
WHERE name='United Kingdom')

d.这个查询实际上等同于以下这个:

select e1.ename from emp e1,(select empno from emp where ename = 'KING') e2 whe

re e1.mgr = e2.empno;
你可以用 EXISTS 写同样的查询，你只要把外部查询一栏移到一个像下面这样的子查询环境中就可以了：
select ename from emp e
where exists (select 0 from emp where e.mgr = empno and ename = 'KING');
当你在一个 WHERE 子句中写 EXISTS 时，又等于向最优化传达了这样一条信息，即你想让外部查询先运行，使用每一个值来从内部查询（假定：EXISTS＝由外而内）中得到一个值。
B.异表子查询作为条件
a.select * from studentExam where studentid=(select studentid from student where name='吴丽丽');

b.select * from studentexam where studentid in (
select studentid from student) order by studentid;

c.select * from student where studentid in (select studentid from studentexam where mark>80);

3.select studentexam.mark,studentexam.studentid as seid, student.studentid,student.name from studentexam,student where student.studentid=studentexam.studentid;

过滤分组:
顺序为先分组,再过滤,最后进行统计(实际值).
select studentid,count(*) as highpasses from studentexamwhere mark>70group by studentid;
假使我们不想通过数据表中的实际值,而是通过聚合函数的结果来过过滤查询的结果.
select studentid,avg(mark) as averagemarkfrom studentexamwhere avg(mark)<50 or avg(mark)>70group by studentid;(此句错误,where 句子是不能用聚合函数作条件的)此时要用 having.
select studentid,avg(mark) from studentexam group by studentid having avg(mark)>70 or avg(mark)<50;

select studentid,avg(mark) from studentexam where studentid in(1,7,9,5)group by studentid having avg(mark)>70;(先分组,再过滤,再 having 聚合,最后再统计).

select studentid ,avg(mark) as averagemarkfrom studentexamwhere examid in(5,8,1

1)group by studentidhaving avg(mark)<50 or avg(mark)>70;

返回限定行数查询:

select name from student where rownum<=10;

oracle 中使用 rownum 关键字指定,但该关键字必须在 where 子句中与一个比较运算符一起指定,而不能与 order by 一起配合使用,因为 rownum 维护的是原始行号.如果需要用 group by\order by 就用子句查询作表使用的方法:

select studentid,averagemark from(select studentid,avg(mark) as averagemarkfrom studentexamgroup by studentid order by averagemark desc)where rownum<=10;

oracle 存储过程语法:Oracle 存储过程入门学习基本语法

1.基本结构

```
create OR REPLACE PROCEDURE 存储过程名字
(
参数 1 IN NUMBER,
参数 2 IN NUMBER
) IS
变量 1 INTEGER :=0;
变量 2 DATE;
BEGIN
END 存储过程名字
```

2.select INTO STATEMENT

将 select 查询的结果存入到变量中，可以同时将多个列存储多个变量中，必须有一条记录，否则抛出异常(如果没有记录抛出 NO_DATA_FOUND)

例子:

```
BEGIN
  select col1,col2 into 变量 1,变量 2 FROM typestruct where xxx;
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
      xxxx;
  END;
  ...
```

3.IF 判断

```
IF V_TEST=1 THEN
  BEGIN
     do something
  END;
END IF;
```

```
    4.while 循环
    WHILE V_TEST=1 LOOP
    BEGIN
  XXXX
    END;
   END LOOP;
    5.变量赋值
    V_TEST := 123;
    6.用 for in 使用 cursor
   ...
   IS
    CURSOR cur IS select * FROM xxx;
   BEGIN
  FOR cur_result in cur LOOP
   BEGIN
     V_SUM :=cur_result.列名 1+cur_result.列名 2
   END;
  END LOOP;
    END;
    7.带参数的 cursor
   CURSOR C_USER(C_ID NUMBER) IS select NAME FROM USER where TYPEID=C_ID;
   OPEN C_USER(变量值);
    LOOP
    FETCH C_USER INTO V_NAME;
    EXIT FETCH C_USER%NOTFOUND;
      do something
   END LOOP;
    CLOSE C_USER;
    8.用 pl/sql developer debug
    连接数据库后建立一个 Test WINDOW
    在窗口输入调用 SP 的代码,F9 开始 debug,CTRL+N 单步调试
```

oracle 语法:Oracle 触发器语法及实例基础知识（一）

一 Oracle 触发器语法

触发器是特定事件出现的时候，自动执行的代码块　　类似于存储过程，触发器和存储过程的区别在于:存储过程是由用户或应用程序显式调用的,而触发器是不能被直接调用的

功能:

1、 允许/限制对表的修改

2、 自动生成派生列，比如自增字段

3、 强制数据一致性

4、 提供审计和日志记录

5、 防止无效的事务处理

6、 启用复杂的业务逻辑

触发器触发时间有两种:after 和 before

1、触发器的语法:

CREATE [OR REPLACE] TIGGER 触发器名 触发时间 触发事件

ON 表名

[FOR EACH ROW]

BEGIN

pl/sql 语句

END

其中:

触发器名:触发器对象的名称

由于触发器是数据库自动执行的，因此该名称只是一个名称，没有实质的用途

触发时间:指明触发器何时执行，该值可取:

before---表示在数据库动作的前触发器执行;

after---表示在数据库动作的后出发器执行

触发事件:指明哪些数据库动作会触发此触发器:

insert:数据库插入会触发此触发器;

update:数据库修改会触发此触发器;

delete:数据库删除会触发此触发器

表 名:数据库触发器所在的表

for each row:对表的每一行触发器执行一次　　如果没有这一选项，则只对整个表执行一次

2、举例:

下面的触发器在更新表 auths 的前触发，目的是不允许在周末修改表:

```
create triggerauth_secure before insert or update or delete //对整表更新前触发
    on auths
    begin
    if(to_char(sysdate,'DY')='SUN'
    RAISE_APPLICATION_ERROR(-20600,'不能在周末修改表 auths');
    end if;
    end
```

例子:

```
  CREATE OR REPLACE TRIGGER CRM.T_SUB_USERINFO_AUR_NAME AFTER UPDATE OF
STAFF_NAME
    ON CRM.T_SUB_USERINFO
    REFERENCING OLD AS OLD NEW AS NEW
    FOR EACH ROW
    declare
    begin
    if :NEW.STAFF_NAME!=:OLD.STAFF_NAME then
    begin
```

客户投诉

```
        update T_COMPLAINT_MANAGE set SERVE_NAME=:NEW.STAFF_NAME where
SERVE_SEED=:OLD.SEED;
```

客户关怀

```
update T_CUSTOMER_CARE set EXECUTOR_NAME=:NEW.STAFF_NAME
    where EXECUTOR_SEED=:OLD.SEED;
```

客户服务

```
update T_CUSTOMER_SERVICE set EXECUTOR_NAME=:NEW.STAFF_NAME
    where EXECUTOR_SEED=:OLD.SEED;
    end;
    end if;
    end T_sub_userinfo_aur_name;
    /
```

2 Oracle 触发器详解

开始:

```
create triggerbiufer_employees_department_id
    beforeinsertorupdateofdepartment_idonemployees
    referencingoldasold_value new as new_value
    for each row
    when (new_value.department_id<>80 )
    begin
    :new_value.commission_pct :=0;
    end;
    /
```

1、触发器的组成部分:

1、 触发器名称

2、 触发语句

3、 触发器限制

4、 触发操作

1.1、触发器名称

```
create trigger biufer_employees_department_id
```

命名习惯:

biufer(before insert update for each row)

employees 表名

department_id 列名

1.2、触发语句

比如:

表或视图上的 DML 语句

DDL 语句

数据库关闭或启动,startup shutdown 等等

```
before insert or update
    of department_id
    on employees
    referencing old as old_value
    new as new_value
    for each row
```

介绍说明:

1、 无论是否规定了 department_id，对 employees 表进行 insert 的时候

2、 对 employees 表的 department_id 列进行 update 的时候

1.3、触发器限制

when (new_value.department_id<>80)

限制不是必须的　　此例表示如果列 department_id 不等于 80 的时候，触发器就会执行

其中的 new_value 是代表更新的后的值

1.4、触发操作

是触发器的主体

```
begin
    :new_value.commission_pct :=0;
    end;
```

主体很简单，就是将更新后的 commission_pct 列置为 0

触发:

```
insert into employees(employee_id,last_name,first_name,hire_date,job_id,email,
    department_id,salary,commission_pct )
    values( 12345,'Chen','Donny', sysdate, 12, 'donny@hotmail.com',60,10000,.25);
    select commission_pct from employees where employee_id=12345;
```

2、触发器的类型有:

触发器类型:

1、 语句触发器

2、 行触发器

3、INSTEAD OF 触发

4、 系统条件触发器

5、 用户事件触发器

2.1、语句级触发器.(语句级触发器对每个 DML 语句执行一次)

是在表上或者某些情况下的视图上执行的特定语句或者语句组上的触发器 能够和 INSERT、UPDATE、DELETE 或者组合上进行关联 但是无论使用什么样的组合，各个语句触发器都只会针对指定语句激活一次 比如，无论 update 多少行，也只会调用一次 update 语句触发器

例子:

```
create or replace trigger tri_test
    after insert or update or delete _disibledevent=>
```

测试,插入几条记录

```
insert into test values(0,'ff');
    insert into test values(0,'ff');
    insert into test values(0,'tt');
```

例子 2:

创建一个触发器,无论用户插入新记录,还是修改 emp 表的 job 列,都将用户指定的 job 列的值转换成大写.

```
 create or replace trigger trig_job
    before insert or update of job
    on emp
    for each row
    begin
    if inserting then
    :new.job:=upper(:new.job);
    else
    : new.job:=upper(:new.job);
    end if;
    end;
```

2.3、instead of 触发器.

(此触发器是在视图上而不是在表上定义的触发器,它是用来替换所使用实际语句的触发器.)

语法如下:

```
  create or replace triggertrig_test
    instead ofinsert or update _disibledevent=>
```

2.5、数据库级触发器.

可以创建在数据库事件上的触发器,包括关闭,启动,服务器错误,登录等.这些事件都是例子范围的,不和特定的表或视图关联.

例子:

```
create or replace trigger trig_name
    after startup _disibledevent=>
```

4、 测试

```
  update employees_copy set salary= salary*1.1;
    select *from employess_log;
```

5、 确定是哪个语句起作用?

即是 INSERT/UPDATE/DELETE 中的哪一个触发了触发器?

可以在触发器中使用 INSERTING / UPDATING / DELETING 条件谓词，作判断:

```
begin
    if inserting then
    -----
    elsif updating then
    -----
    elsif deleting then
    ------
    end if;
    end;
    if updating('COL1') or updating('COL2') then
    ------
    end if;
```

2.8、[试验]

1、 修改日志表

```
  alter table employees_log
    add (action varchar2(20));
```

2、 修改触发器，以便记录语句类型

```
then
    l_action:='Delete';
    else
    raise_application_error(-20001,'You should never ever get this error.');
    Insert into employees_log(Who,action,when)
    Values( user, l_action,sysdate);
    End;Create or replace trigger biud_employee_copy
    Before insert or update or delete
    On employees_copy
    Declare
    L_action employees_log.action%type;
    Begin
    if inserting then
    l_action:='Insert';
    elsif updating then
    l_action:='Update';
    elsif deleting
    /
```

3、 测试

```
 insert into employees_copy( employee_id, last_name, email, hire_date, job_id)
values(12345,'Chen','Donny@hotmail',sysdate,12);
select *from employees_log
```

oracle 基本语法备忘

2008 年 11 月 13 日 星期四 09:58

1.基本结构

```
CREATE OR REPLACE PROCEDURE 存储过程名字
(
   参数 1 IN NUMBER,
   参数 2 IN NUMBER
) IS
变量 1 INTEGER :=0;
变量 2 DATE;
BEGIN
END 存储过程名字
```

2.SELECT INTO STATEMENT

将 select 查询的结果存入到变量中，可以同时将多个列存储多个变量中，必须有一条记录，否则抛出异常(如果没有记录抛出 NO_DATA_FOUND)

例子：

```
BEGIN
```

```
SELECT col1,col2 into 变量1,变量2 FROM typestruct where xxx;
EXCEPTION
WHEN NO_DATA_FOUND THEN
         xxxx;
END;
...
```

3.IF 判断

```
IF V_TEST=1 THEN
       BEGIN
             do something
       END;
END IF;
```

4.while 循环

```
WHILE V_TEST=1 LOOP
BEGIN
XXXX
END;
END LOOP;
```

5.变量赋值

```
V_TEST := 123;
```

6.用 for in 使用 cursor

```
...
IS
CURSOR cur IS SELECT * FROM xxx;
BEGIN
FOR cur_result in cur LOOP
     BEGIN
       V_SUM :=cur_result.列名1+cur_result.列名2
     END;
END LOOP;
END;
```

7.带参数的 cursor

```
CURSOR C_USER(C_ID NUMBER) IS SELECT NAME FROM USER WHERE TYPEID=C_ID;
OPEN C_USER(变量值);
LOOP
FETCH C_USER INTO V_NAME;
EXIT FETCH C_USER%NOTFOUND;
       do something
END LOOP;
```

```
CLOSE C_USER;
```

8.用 pl/sql developer debug
连接数据库后建立一个 Test WINDOW
在窗口输入调用 SP 的代码,F9 开始 debug,CTRL+N 单步调试

关于 oracle 存储过程的若干问题备忘

1.在 oracle 中，数据表别名不能加 as，如：

```
select a.appname from appinfo a;-- 正确
select a.appname from appinfo as a;-- 错误
```

也许，是怕和 oracle 中的存储过程中的关键字 as 冲突的问题吧

2.在存储过程中，select 某一字段时，后面必须紧跟 into，如果 select 整个记录，利用游标的话就另当别论了。

```
select af.keynode into kn from APPFOUNDATION af where af.appid=aid and af.foundationid=fid;-- 有 into，正确编译
select af.keynode from APPFOUNDATION af where af.appid=aid and af.foundationid=fid;-- 没有 into，编译报错，提示：Compilation
    Error: PLS-00428: an INTO clause is expected in this SELECT statement
```

3.在利用 select...into...语法时，必须先确保数据库中有该条记录，否则会报出"no data found"异常。
可以在该语法之前，先利用 select count(*) from 查看数据库中是否存在该记录，如果存在，再利用 select...into...

4.在存储过程中，别名不能和字段名称相同，否则虽然编译可以通过，但在运行阶段会报错

```
select keynode into kn from APPFOUNDATION where appid=aid and foundationid=fid;-- 正确运行
select af.keynode into kn from APPFOUNDATION af where af.appid=appid and af.foundationid=foundationid;-- 运行阶段报错，提示
ORA-01422:exact fetch returns more than requested number of rows
```

5.在存储过程中，关于出现 null 的问题
假设有一个表 A，定义如下：

```
create table A(
id varchar2(50) primary key not null,
vcount number(8) not null,
bid varchar2(50) not null -- 外键
);
```

如果在存储过程中，使用如下语句：

```
select sum(vcount) into fcount from A where bid='xxxxxx';
```

如果 A 表中不存在 bid="xxxxxx"的记录，则 fcount=null(即使 fcount 定义时设置了默认值，如：fcount number(8):=0 依然无效，fcount 还是会变成 null)，这样以后使用 fcount 时就可能有问题，所以在这里最好先判断一下：

```
if fcount is null then
    fcount:=0;
end if;
```

这样就一切 ok 了。

1、创建存储过程

```
create or replace procedure test(var_name_1 in type,var_name_2 out type) as
```

--声明变量(变量名 变量类型)

begin

--存储过程的执行体

end test;

打印出输入的时间信息

E.g:

```
create or replace procedure test(workDate in Date) is

begin

dbms_output.putline(&apos;The input date
is:&apos;||to_date(workDate,&apos;yyyy-mm-dd&apos;));

end test;
```

2、变量赋值

变量名 := 值;

E.g:

```
create or replace procedure test(workDate in Date) is

x number(4,2);

begin

x := 1;

end test;
```

3、判断语句:

if 比较式 then begin end; end if;

E.g

```
create or replace procedure test(x in number) is

begin
                    if x >0 then
                      begin
                    x := 0 - x;
                    end;
          end if;
          if x = 0 then
```

```
            begin
              x: = 1;
        end;
        end if;
end test;
```

4、For 循环

```
For ... in ... LOOP
--执行语句
end LOOP;
```

(1)循环遍历游标

```
create or replace procedure test() as
Cursor cursor is select name from student; name varchar(20);
begin
for name in cursor LOOP
begin
dbms_output.putline(name);
end;
end LOOP;
end test;
```

(2)循环遍历数组

```
create or replace procedure test(varArray in myPackage.TestArray) as
--(输入参数 varArray 是自定义的数组类型，定义方式见标题 6)
i number;
begin
i := 1; --存储过程数组是起始位置是从 1 开始的，与 java、C、C++等语言不同。因为在 Oracle 中本是没有数组的概念的，数组其实就是一张
--表(Table),每个数组元素就是表中的一个记录，所以遍历数组时就相当于从表中的第一条记录开始遍历
for i in 1..varArray.count LOOP
dbms_output.putline(&apos;The No.&apos;|| i || &apos;record in varArray
```

```
is:&apos;||varArray(i));
end LOOP;
end test;
```

5、While 循环

```
while 条件语句 LOOP
begin
end;
end LOOP;
```

E.g

```
create or replace procedure test(i in number) as
begin
while i < 10 LOOP
begin
i:= i + 1;
end;
end LOOP;
end test;
```

6、数组

首先明确一个概念：Oracle 中本是没有数组的概念的，数组其实就是一张表(Table),每个数组元素就是表中的一个记录。

使用数组时，用户可以使用 Oracle 已经定义好的数组类型，或可根据自己的需要定义数组类型。

(1)使用 Oracle 自带的数组类型

```
x array; --使用时需要需要进行初始化
```

e.g:

```
create or replace procedure test(y out array) is
x array;
begin
x := new array();
y := x;
end test;
```

(2)自定义的数组类型（自定义数据类型时，建议通过创建 Package 的方式实现，以便于管理）

E.g（自定义使用参见标题 4.2）create or replace package myPackage is

-- Public type declarations type info is record(name varchar(20), y number);

type TestArray is table of info index by binary_integer; --此处声明了一个 TestArray 的类型数据，其实其为一张存储 Info 数据类型的 Table 而已，及 TestArray 就是一张表，有两个字段，一个是 name，一个是 y。需要注意的是此处使用了 Index by binary_integer 编制该 Table 的索引项，也可以不写，直接写成：type TestArray is

table of info，如果不写的话使用数组时就需要进行初始化：varArray myPackage.TestArray; varArray := new myPackage.TestArray();

end TestArray;

7.游标的使用 Oracle 中 Cursor 是非常有用的，用于遍历临时表中的查询结果。其相关方法和属性也很多，现仅就常用的用法做一二介绍：

(1)Cursor 型游标(不能用于参数传递)

create or replace procedure test() is

cusor_1 Cursor is select std_name from student where ...; --Cursor 的使用方式 1 cursor_2 Cursor;

begin

select class_name into cursor_2 from class where ...; --Cursor 的使用方式 2

可使用 For x in cursor LOOP .... end LOOP; 来实现对 Cursor 的遍历

end test;

(2)SYS_REFCURSOR 型游标，该游标是 Oracle 以预先定义的游标，可作出参数进行传递

create or replace procedure test(rsCursor out SYS_REFCURSOR) is

cursor SYS_REFCURSOR; name varhcar(20);

begin

OPEN cursor FOR select name from student where ... --SYS_REFCURSOR 只能通过 OPEN 方法来打开和赋值

LOOP

fetch cursor into name --SYS_REFCURSOR 只能通过 fetch into 来打开和遍历 exit when cursor%NOTFOUND; --SYS_REFCURSOR 中可使用三个状态属性：---%NOTFOUND(未找到记录信息) %FOUND(找到记录信息) ---%ROWCOUNT(然后当前游标所指向的行位置)

dbms_output.putline(name);

```
end LOOP;

rsCursor := cursor;

end test;
```

下面写一个简单的例子来对以上所说的存储过程的用法做一个应用：

现假设存在两张表，一张是学生成绩表(studnet)，字段为：
stdId,math,article,language,music,sport,total,average,step
一张是学生课外成绩表(out_school),字段为:stdId,parctice,comment

通过存储过程自动计算出每位学生的总成绩和平均成绩，同时，如果学生在课外课程中获得的评价为 A，就在总成绩上加 20 分。

```
create or replace procedure autocomputer(step in number) is

rsCursor SYS_REFCURSOR;

commentArray myPackage.myArray;

math number;

article number;

language number;

music number;

sport number;

total number;

average number;

stdId varchar(30);

record myPackage.stdInfo;

i number;

begin

i := 1;

get_comment(commentArray); --调用名为 get_comment()的存储过程获取学生课外评分信息

OPEN rsCursor for select stdId,math,article,language,music,sport from student t where t.step = step;

LOOP

fetch rsCursor into stdId,math,article,language,music,sport; exit when rsCursor%NOTFOUND;

total := math + article + language + music + sport;

for i in 1..commentArray.count LOOP
```

```
record := commentArray(i);
if stdId = record.stdId then
begin
if record.comment = &apos;A&apos; then
     begin
total := total + 20;
       go to next; --使用 go to 跳出 for 循环
end;
end if;
end;
end if;
end LOOP;
<<continue>> average := total / 5;
update student t set t.total=total and t.average = average where t.stdId = stdId;
end LOOP;
end;
end autocomputer;
--取得学生评论信息的存储过程
create or replace procedure get_comment(commentArray out myPackage.myArray) is
rs SYS_REFCURSOR;
record myPackage.stdInfo;
stdId varchar(30);
comment varchar(1);
i number;
begin
open rs for select stdId,comment from out_school
i := 1;
LOOP
fetch rs into stdId,comment; exit when rs%NOTFOUND;
record.stdId := stdId;
```

```
record.comment := comment;
recommentArray(i) := record;
i:=i + 1;
end LOOP;
end get_comment;
--定义数组类型 myArray
create or replace package myPackage is begin
type stdInfo is record(stdId varchar(30),comment varchar(1));
type myArray is table of stdInfo index by binary_integer;
end myPackage;
```