# 第一篇 Concept

## 简介

SQL语言只是访问、操作数据库的语言，并不是一种具有流程控制的程序设计语言，而只有程序设计语言才能用于应用软件的开发。PL/SQL是一种高级数据库程序设计语言。PL/SQL同样具备高级语言的一些特性，比如模块化，过程控制，异常等。

**优势**

1能够使一组SQL语句的功能更具模块化程序特点；

2.采用了过程性语言控制程序的结构；

3.集成在数据库中，调用更快；

4.减少了网络的交互，有助于提高程序性能。

1) 多条SQL语句实现功能时，每条语句都需要在[客户端](http://baike.baidu.com/view/930.htm" \t "_blank)和[服务端](http://baike.baidu.com/view/1087294.htm" \t "_blank)传递，而且每条语句的执行结果也需要在网络中进行交互，占用了大量的网络带宽，消耗了大量网络传输时间，而在网络中传输的那些结果，往往都是中间结果，而不是我们所关心的。

2) PL/SQL程序代码存储在数据库中，程序的分析和执行完全在数据库内部进行，用户所需要做的就是在[客户端](http://baike.baidu.com/view/930.htm" \t "_blank)发出调用PL/SQL的执行命令，数据库接收到执行命令后，在数据库内部完成整个PL/SQL程序的执行，并将最终的执行结果返馈给用户。所以整体程序的执行性能会有明显的提高。

Print Switch:

set serveroutput on

dbms\_output.put\_line('cursor:')

### ROWID

ROWID它是一个伪列，它并不实际存在于表中。读取表中数据行时，根据每一行数据的物理地址编码而成的一个伪列。所以根据ROWID能找到数据的物理地址。从而快速地定位到数据行。

### ROWNUM

### TYPE&ROWTYPE

# 第二篇 Syntax

## 2.1 Date Type

自定义类型

[Reference](https://www.jianshu.com/p/0263aa8dc432)

ORACLE中自定义数据类型的方式一般是三种:RECORD、VARRAY、TABLE

### Record

1. TYPE RECORD\_NAME IS RECORD(
2. V1  DATA\_TYPE1 [NOT NULL][:=DEFAULT\_VALUE],
3. V2  DATA\_TYPE2 [NOT NULL][:=DEFAULT\_VALUE],
4. VN  DATA\_TYPEN [NOT NULL][:=DEFAULT\_VALUE]
5. );
6. **declare**
7. TYPE RECORD\_TEAM **IS** RECORD(
8. ID         TB\_TEAM.ID%TYPE,
9. **NAME**       TB\_TEAM.**NAME**%TYPE
10. );
11. v\_team RECORD\_TEAM;
13. **begin**
14. **select** ID,CHINESE\_NAME **into** v\_team **from** TB\_EMPLOYEE **where** id=:P11\_ID;
15. EXCEPTION
16. **WHEN** NO\_DATA\_FOUND **THEN**
17. dbms\_output.put\_line('error:');
18. **END**;
19. **end**;

直接赋值。RECORD1 :=RECORD2；

### Array

定义数组类型。

1. **DECLARE**
2. type T\_ARRAY **IS** VARRAY(5) **OF** VARCHAR2(10)[NOT NULL];
3. type T\_ARRAY2 **IS** VARRAY(1) **OF** TB\_EMPLOYEE.EMAIL%TYPE;
4. names T\_ARRAY ;
5. name\_list T\_ARRAY2 := T\_ARRAY2 ();
6. **CURSOR** cur1 **IS**
7. **SELECT** EMAIL **FROM** TB\_EMPLOYEE **where** id=284;
9. total **integer**;
10. v\_cnt number :=0;
11. **BEGIN**
12. --------------------------------------------------------------------
13. names := T\_ARRAY ('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
14. total := names.count;
15. **FOR** i in 1 .. total LOOP
16. dbms\_output.put\_line('Student: ' || names(i) );
17. **END** LOOP;
18. --------------------------------------------------------------------
19. **FOR** rec\_d IN cur1 LOOP
20. v\_cnt := v\_cnt + 1;
21. name\_list.extend;
22. name\_list(v\_cnt )  := rec\_d.EMAIL;
23. dbms\_output.put\_line(name\_list(v\_cnt));
24. **END** LOOP;

27. **END**;
28. **declare**
29. --定义一个最多保存5个VARCHAR(25)数据类型成员的VARRAY数据类型
30. type my\_array\_type **is**
31. varray ( 5 ) **of** **varchar**(25);
32. v\_varray my\_array\_type;
33. **begin**
34. dbms\_output.put\_line('---------------- VARRAY ---------------------------------------');
35. v\_varray := my\_array\_type(
36. 'a','b','c','d','e'
37. );
38. dbms\_output.put\_line(
39. 'ENTRIES :' ||v\_varray.count() ||',CAPACITY :' ||v\_varray.limit()||'FIRST :' ||v\_varray.**FIRST**||'LAST :' ||v\_varray.**LAST**
40. );
42. **for** i in 1..v\_varray.count () loop
43. dbms\_output.put\_line(
44. 'V(' ||i ||')=' ||v\_varray(i)
45. );
46. **end** loop;
48. v\_varray(5) := 'chi';
49. dbms\_output.put\_line('v5：' ||v\_varray(5) );
51. **FOR** i IN v\_varray.**FIRST**..v\_varray.**LAST** LOOP
52. DBMS\_OUTPUT.PUT\_LINE ( 'FOR-v\_varray :'||v\_varray(i));
53. **END** LOOP;
55. **end**;

### Table

定义记录表。 相对于Record，可以处理多行记录，类似于C语言中的二维数组，使得可以在PL/SQL中模仿数据库中的表。

语法：

1. TYPE type\_name **IS** **TABLE** **OF** element\_type [NOT NULL] **INDEX** **BY** subscript\_type; table\_name type\_name;   //声明

一维数组

1. **DECLARE**
2. TYPE T\_ARRAY **IS** **TABLE** **OF** **varchar**(50) **INDEX** **BY** BINARY\_INTEGER;
3. strings T\_ARRAY;
4. **CURSOR** cur1 **IS**
5. **SELECT** EMAIL **FROM** TB\_EMPLOYEE;
6. v\_cnt number:=0;
8. **BEGIN**
9. **FOR** rec\_d IN cur1 LOOP
10. v\_cnt :=v\_cnt +1;
11. strings(v\_cnt) :=rec\_d.EMAIL;
12. dbms\_output.put\_line(strings(v\_cnt));
13. **END** LOOP;
14. **END**;

INDEX BY BINARY\_INTEGER 下标自增长

插入元素时，不需要初始化，不需要每次EXTEND增加一个空间。而如果没有这句话“INDEX BY BINARY\_INTEGER”，那就得要显示对初始化，且每插入一个元素到NUMBERS类型的TABLE中时，都需要先EXTEND。

结合ROWTYPE

采用bulk collect可以将查询结果一次性加载到collections中。不用通过cursor一条一条地处理

1. declare
2. type t\_type is table of TB\_TEAM%rowtype;
3. v\_type t\_type;
4. begin
5. select id,name, MANAGER\_ID bulk collect into v\_type from TB\_TEAM
6. where id <= 300;
8. dbms\_output.put\_line('id : ' || v\_type(2).id ||' name : ' || v\_type(2).name);
10. **for** i in v\_type.first .. v\_type.last -- 这里留意一下first  last的用法
11. loop
12. dbms\_output.put\_line('id : ' || v\_type(i).id ||' name : ' || v\_type(i).name);
14. end loop;
16. end;

结合RECORD

1. declare
2. type v\_test is record(
3. test\_id testkd.id%type,
4. test\_name testkd.name%type,
5. test\_flag testkd.flag%type
6. );
7. --定义一个table类型(即表结构)，它的列来自于v\_test
8. type t\_type is table of v\_test;
9. -- 声明t\_type类型的变量
10. v\_type t\_type;
11. begin
12. select id,name,flag bulk collect into v\_type from testkd
13. where testkd.id <= 5;
15. **for** i in v\_type.first .. v\_type.last loop
16. dbms\_output.put\_line('id : ' || v\_type(i).test\_id);
17. end loop;
18. end;

遍历

[Reference](https://blog.csdn.net/kanlianhui/article/details/7192589)

## 2.2 Exception

1. EXCEPTION
2. **WHEN** e\_name1 [OR e\_name2 ... ] **THEN**
3. statements;
4. ...
5. **WHEN** OTHERS **THEN**
6. statements;

## 2.3 Features

### Procedure

### Function

### Cursor

1. procedure  SYN\_EMPLOYEE
2. AS
3. CURSOR cur1 IS
4. SELECT EMAIL FROM TB\_EMPLOYEE\_CN\_VIEW WHERE where deleted='NO');
5. Cursor cur2(p\_emp\_id number) is
6. select name,email from TB\_EMPLOYEE where id=p\_emp\_id
7. begin
8. FOR rec\_d IN cur1 LOOP
9. insert into TB\_EMPLOYEE values(null,NULL,rec\_d.EMAIL);
10. END LOOP;
12. end;

### Trigger

1. **create** or replace **trigger** HCM\_ESS\_JOBS\_INVENTORY\_ID
2. before **insert** **on** FUSION.HCM\_ESS\_JOBS\_INVENTORY
3. **for** each row
4. **begin**
5. if inserting **then**
6. if :NEW.id **is** null **then**
7. **select** HCM\_ESS\_JOBS\_INVENTORY\_ID\_SEQ.nextval **into** :NEW.id **from** dual;
8. **end** if;
9. **end** if;
10. **end**;

### View

# 第三篇 面向对象

OBJECT VS Type table

OBJECT虽然能提供数组一样使用，但必须CREATE。实际使用中用系统自带的Type table

## 3.1 对象类型

Object Address

1. **CREATE** OR REPLACE TYPE address **AS** OBJECT (
2. house\_no varchar2(10),
3. street varchar2(30),
4. city varchar2(20),
5. state varchar2(10),
6. pincode varchar2(10)
7. );

Object Customer

1. **CREATE** OR REPLACE TYPE customer **AS** OBJECT(
2. code number(5),
3. **name** varchar2(30),
4. contact\_no varchar2(12),
5. addr address,//复合类型
6. member **procedure** display
7. );

## 3.2 实例化

1. **DECLARE**
2. residence address;
3. **BEGIN**
4. residence := address('103A', 'M.G.Road', 'Jaipur', 'Rajasthan','201301');   dbms\_output.put\_line('House No: '|| residence.house\_no);
5. **END**;

## 3.3 成员方法

构造函数

每个对象都有一个系统定义的构造方法

1. residence := address('103A', 'M.G.Road', 'Jaipur', 'Rajasthan','201301');

定义方法

1. **CREATE** OR REPLACE TYPE BODY rectangle **AS**
2. MEMBER **FUNCTION** enlarge(inc number) **return** rectangle **IS**
3. **BEGIN**
4. **return** rectangle(self.length + inc, self.width + inc);
5. **END** enlarge;
7. MEMBER **PROCEDURE** display **IS**
8. **BEGIN**
9. dbms\_output.put\_line('Length: '|| length);
10. dbms\_output.put\_line('Width: '|| width);
11. **END** display;
12. **END**;

## 3.4 继承

声明

基本对象应被声明为NOT FINAL。默认值是FINAL。

1. **CREATE** OR REPLACE TYPE rectangle **AS** OBJECT(
2. length number,
3. width number,
4. member **function** enlarge( inc number) **return** rectangle,
5. NOT FINAL member **procedure** display
6. ) NOT FINAL

create body—略

子对象

1. **CREATE** OR REPLACE TYPE tabletop UNDER rectangle(
2. material varchar2(20);
3. OVERRIDING member **procedure** display
4. )

creat table top body—略

## 3.5 抽象

1. **CREATE** OR REPLACE TYPE rectangle **AS** OBJECT(
2. length number,
3. width number,
4. NOT INSTANTIABLE NOT FINAL MEMBER **PROCEDURE** display
5. ) NOT INSTANTIABLE NOT FINAL

创建一个子类型或子类型使用其功能。

## 3.6 Package

**Specification**

1. **create** or replace package  HCM\_ACTIVITY\_PKG **as**

4. **function** get\_enroll\_activity\_status(
5. p\_employee\_id in number)
6. **return** VARCHAR2;
8. **procedure** SYN\_EMPLOYEE ;
9. **procedure** UPDATE\_ENROLL\_STATUS(id in number);
10. **end**;

**Body**

1. create or replace package body HCM\_ACTIVITY\_PKG is
3. procedure  SYN\_EMPLOYEE
4. AS
5. CURSOR cur1 IS
6. SELECT EMAIL FROM TB\_EMPLOYEE\_CN\_VIEW WHERE where deleted='NO');
7. begin
8. FOR rec\_d IN cur1 LOOP
9. insert into TB\_EMPLOYEE values(null,NULL,rec\_d.EMAIL);
10. END LOOP;
12. end;
13. -- ----------------------------------------------------------------------- --  get\_enroll\_activity\_status
14. -- -------------------------------------------------------------------------
15. function get\_enroll\_activity\_status( p\_employee\_id in number)
16. **return** VARCHAR2
17. as
18. v\_activity\_status varchar(50);
19. begin
20. null
21. end;
23. end;

# 附件 Function

## Date

StringToDate

to\_date('20-Jun-2018 14:10', 'dd-mm-yyyy hh24:mi')

DateToString

to\_char(sysdate,'yyyy-mm-dd HH24:MI:SS')

select TO\_CHAR(sysdate,'YYYY-MM-DD HH24:MI:SS') as now FROM dual;

当前时间

select TO\_CHAR(sysdate,'YYYY-MM-DD HH24:MI:SS') as now FROM dual;

select to\_date(sysdate, 'dd-mm-yyyy hh24:mi') from dual;

## Instr

instr(‘abc’,’a’) 返回1

if instr(‘abc’,’a’)>0 then //

null;

end if;

## substr

1. declare
2. v\_str varchar;
3. begin
4. v\_str=’abc d’ ;
5. select substr(v\_str,1,instr(v\_str,' ')) from dual;
6. end;

## Decode

1. **SELECT** supplier\_name,
2. DECODE(supplier\_id, 10000, 'IBM',
3. 10001, 'Microsoft',
4. 10002, 'Hewlett Packard',
5. 'Gateway') result
6. **FROM** suppliers;

Equal Code

1. if(supplier\_id = 1000) **then**
2. **return** ‘IBM’;
3. **else** if(supplier\_id = 1001) **then**
4. **return** ‘Microsoft’;
5. **else** if(supplier\_id = 1002) **then**
6. **return** ‘Hewlett Packard';
7. **else**
8. **return** ‘Gateway’;
9. **end** if;

## NVL

NVL (expr1, expr2)

两者的类型要一致

1. select nvl(expr1,expr2)  from dual
3. Equal Code
4. **if** expr1is not null then
5. **return** expr1;
6. **else**
7. **return** expr2;

NVL2 (expr1, expr2, expr3)

1. select NVL2 (expr1,expr2,expr3)  from dual
3. Equal Code
4. **if** expr1 not null then
5. **return** expr2;
6. **else**
7. **return** expr3;

## Scheduler

Create

1. **BEGIN**
2. DBMS\_SCHEDULER.create\_job (
3. job\_name        => 'SYN\_EMPLOYEE\_JOB',
4. job\_type        => 'PLSQL\_BLOCK',
5. job\_action      => 'BEGIN CADB.HCM\_ACTIVITY\_PKG.SYN\_EMPLOYEE; END;',
6. start\_date      => SYSTIMESTAMP+19/24,
7. repeat\_interval => 'FREQ=DAILY;INTERVAL=1;',
8. enabled         => **TRUE**);
9. **END**;

start\_date

SYSTIMESTAMP + 60/86400 86400是24小时的秒数,这里加60秒

SYSTIMESTAMP+1/24 当前时间+一小时

repeat\_interval

每天：FREQ=DAILY;INTERVAL=1;

每个月一号： FREQ=MONTHLY; BYMONTHDAY=1;

Drop

1. **begin**
2. DBMS\_SCHEDULER.drop\_job (job\_name => 'RESET\_ENROLL\_STATUS\_JOB');
3. **end**;

Query

1. **select** JOB\_NAME,START\_DATE,LAST\_START\_DATE, NEXT\_RUN\_DATE,REPEAT\_INTERVAL,JOB\_ACTION
2. **from** user\_scheduler\_jobs
3. **where** JOB\_NAME='SYN\_EMPLOYEE\_JOB' ;

查看执行结果

1. **select** \* **from** all\_scheduler\_job\_log t
2. **where**  t.job\_name = 'TEST\_SCHEDULER\_JOBS2'
3. **order** **by** t.LOG\_DATE **desc**;
5. **select** \* **from** all\_scheduler\_job\_run\_details  t
6. **where**  t.job\_name = 'TEST\_SCHEDULER\_JOBS2'
7. **order** **by** t.LOG\_DATE **desc**;