

在 Oracle之被迫学习

在 Oracle之被迫学习

Oracle用户权限分配

系统权限管理

实体权限管理

SQL约束

序列sequence

子查询

PL/SQL

PL/SQL条件控制

PL/SQL循环

PL/SQL字符串

PL/SQL游标

显式游标

隐式游标

PL/SQL异常

PL/SQL触发器

PL/SQL存储过程

ORACLE®

🔍 查看用户下所有的表

```
select * from user_tables;
```

🔍 查看当前用户的角色

```
select * from user_role_privs;
```

NOLLE--CONNECT, RESOURCE

TAMMIE--CONNECT, RESOURCE

SYSTEM--AQ_ADMINISTRATOR_ROLE, DBA

🔍 查看所有的表空间: system

```
select * from dba_tablespaces;
```

🔍 查看当前用户的缺省表空间

```
select username,default_tablespace from user_users;
```

system--SYSTEM

TAMMIE--SYSTEM

NOLLE--QST_201807_LZ_TS

🔍 查看当前用户对象表

```
select object_name,object_type from user_objects;
```

Oracle用户权限分配

系统权限：系统规定用户使用数据库的权限。(系统权限是对用户而言)。

实体权限：某种权限用户对其它用户的表或视图的存取权限。(是针对表或视图而言的)。

系统权限管理

DBA：拥有全部特权，是系统最高权限，只有 **DBA** 才可以创建数据库结构。

RESOURCE：拥有 **Resource** 权限的用户只可以创建实体，不可以创建数据库结构。

CONNECT：拥有 **Connect** 权限的用户只可以登录 **Oracle**，不可以创建实体，不可以创建数据库结构。

对于普通用户：授予connect，resource权限。

对于DBA管理用户：授予connect，resource，dba权限。

系统权限只能由DBA用户授出和回收：sys，system

普通用户通过授权可以具有与system相同的用户权限，但永远不能达到与sys用户相同的权限，system用户的权限也可以被回收。

A授予B权限，B授予C权限，如果A收回B的权限，C的权限不受影响；系统权限可以跨用户回收，即A可以直接收回C用户的权限。

创建用户：create user 用户 identified by 密码；

用户授权：grant connect，resource，dba to 用户

删除用户：drop user 用户 cascade

回收权限：revoke connect，resource from 用户；

实体权限管理

实体权限分类：**select**，**update**，**insert**，**alter**，**index**，**delete**，**all** //all包括所有权限

用户授权，使某个用户拥有查询当前用户某个表的权限：

grant select，update，insert on 表 to 用户1；

select * from 用户0.表

实体权限回收

revoke select，update on 表 from 用户；

SQL约束

****NOT NULL 约束****

```
CREATE TABLE Persons
(
  P_Id int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
)
```

****UNIQUE 约束****

*/*未指定约束名*/*

```
CREATE TABLE Persons
(
  P_Id int NOT NULL UNIQUE,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
```

```

City varchar(255)
)
/*指定约束名*/
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
)
/*修改表时添加约束*/
ALTER TABLE Persons
ADD UNIQUE (P_Id)
/*撤销约束*/
ALTER TABLE Persons
DROP CONSTRAINT uc_PersonID
**PRIMARY KEY约束**
/*未指定约束名*/
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
/*指定约束名*/
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
/*修改表时添加约束*/
ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
/*撤销约束*/
ALTER TABLE Persons
DROP CONSTRAINT pk_PersonID
**FOREIGN KEY约束**
/*指定约束名*/
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)

```

```

REFERENCES Persons(P_Id)
)
/*修改表时添加约束*/
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
/*撤销约束*/
ALTER TABLE Orders
DROP CONSTRAINT fk_PerOrders
**CHECK约束**
/*未指定约束名*/
CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
/*修改表时添加约束*/
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
)
/*修改表时添加约束*/
ALTER TABLE Persons
ADD CHECK (P_Id>0)
/*撤销约束*/
ALTER TABLE Persons
DROP CONSTRAINT chk_Person

```

序列sequence

创建序列: create sequence 序列名 increment by 步数 start with 起始值;
 使用序列: insert into 表名(id,..) values(序列名.nextval,..);
 删除序列: drop sequence 序列名;

子查询

<https://www.cnblogs.com/cthon/p/9075774.html> join/ins

<https://blog.csdn.net/ahqianjin/article/details/5993829> exists

```

🔒--用户
CREATE TABLE USER
(
USERID INTEGER NOT NULL,  ---用户ID

```

```

COMPANYID INTEGER, ---公司ID
TELNO VARCHAR(12) ---用户电话
);
⊗--公司
CREATE TABLE COMPANY
(
COMPANYID INTEGER NOT NULL, ---公司ID
TELNO VARCHAR(12) ---公司电话
);
⊗查询一下公司电话是 88888888 的用户有哪些, 我们可以使用如下语句:
⊗--非相关子查询 (Uncorrelated Sub-Query)
SELECT * FROM USER WHERE COMPANYID IN
(
SELECT COMPANYID FROM COMPANY WHERE TELNO='88888888'
);
⊗--相关子查询 (Correlated Sub-Query)
SELECT * FROM USER AS U WHERE EXISTS
(
SELECT * FROM COMPANY AS C WHERE TELNO='88888888' AND U.COMPANYID=C.COMPANYID
);

```

PL/SQL

PL/SQL条件控制

```

**IF-THEN**    ✓
DECLARE
    a number(2) := 10;
BEGIN
    -- check the boolean condition using if statement
    IF( a < 20 ) THEN
        -- if condition is true then print the following
        dbms_output.put_line('a is less than 20 ' );
    END IF;
    dbms_output.put_line('value of a is : ' || a);
END;

**IF-ELSE**    ✓
DECLARE
    a number(3) := 100;
BEGIN
    -- check the boolean condition using if statement
    IF( a < 20 ) THEN
        -- if condition is true then print the following
        dbms_output.put_line('a is less than 20 ' );
    ELSE
        dbms_output.put_line('a is not less than 20 ' );
    END IF;
    dbms_output.put_line('value of a is : ' || a);
END;

**IF-ELSIF-ELSE**    ✓
DECLARE
    a number(3) := 100;
BEGIN

```

```

IF ( a = 10 ) THEN
    dbms_output.put_line('Value of a is 10' );
ELSIF ( a = 20 ) THEN
    dbms_output.put_line('Value of a is 20' );
ELSIF ( a = 30 ) THEN
    dbms_output.put_line('Value of a is 30' );
ELSE
    dbms_output.put_line('None of the values is matching');
END IF;
dbms_output.put_line('Exact value of a is: ' || a );
END;

**CASE语句**    ✓
DECLARE
    grade char(1) := 'A';
BEGIN
    CASE grade
        when 'A' then dbms_output.put_line('Excellent');
        when 'B' then dbms_output.put_line('Very good');
        when 'C' then dbms_output.put_line('Well done');
        when 'D' then dbms_output.put_line('You passed');
        when 'F' then dbms_output.put_line('Better try again');
        else dbms_output.put_line('No such grade');
    END CASE;
END;

**搜索CASE语句**
DECLARE
    grade char(1) := 'B';
BEGIN
    case
        when grade = 'A' then dbms_output.put_line('Excellent');
        when grade = 'B' then dbms_output.put_line('Very good');
        when grade = 'C' then dbms_output.put_line('Well done');
        when grade = 'D' then dbms_output.put_line('You passed');
        when grade = 'F' then dbms_output.put_line('Better try again');
        else dbms_output.put_line('No such grade');
    end case;
END;

**嵌套IF语句**
DECLARE
    a number(3) := 100;
    b number(3) := 200;
BEGIN
    -- check the boolean condition
    IF( a = 100 ) THEN
        -- if condition is true then check the following
        IF( b = 200 ) THEN
            -- if condition is true then print the following
            dbms_output.put_line('Value of a is 100 and b is 200' );
        END IF;
    END IF;
    dbms_output.put_line('Exact value of a is : ' || a );
    dbms_output.put_line('Exact value of b is : ' || b );
END;

```

PL/SQL循环

****基本循环****

```
DECLARE
    x number := 10;
BEGIN
    LOOP
        dbms_output.put_line(x);
        x := x + 10;
        IF x > 50 THEN
            exit;
        END IF;
    END LOOP;
    -- after exit, control resumes here
    dbms_output.put_line('After Exit x is: ' || x);
END;
```

****FOR循环****

```
DECLARE
    i number(1);
    j number(1);
BEGIN
    << outer_loop >>
    FOR i IN 1..3 LOOP
        << inner_loop >>
        FOR j IN 1..3 LOOP
            dbms_output.put_line('i is: ' || i || ' and j is: ' || j);
        END loop inner_loop;
    END loop outer_loop;
END;
```

****反转FOR循环****

```
DECLARE
    a number(2) ;
BEGIN
    FOR a IN REVERSE 10 .. 20 LOOP
        dbms_output.put_line('value of a: ' || a);
    END LOOP;
END;
```

****WHILE循环****

```
DECLARE
    a number(2) := 10;
BEGIN
    WHILE a < 20 LOOP
        dbms_output.put_line('value of a: ' || a);
        a := a + 1;
    END LOOP;
END;
```

****EXIT****

```
DECLARE
    a number(2) := 10;
BEGIN
    -- while loop execution
    WHILE a < 20 LOOP
        dbms_output.put_line ('value of a: ' || a);
```

```


        a := a + 1;
    IF a > 15 THEN
        -- terminate the loop using the exit statement
        EXIT;
    END IF;
END LOOP;
END;


**EXIT WHEN**
DECLARE
    a number(2) := 10;
BEGIN
    -- while loop execution
    WHILE a < 20 LOOP
        dbms_output.put_line ('value of a: ' || a);
        a := a + 1;
        -- terminate the loop using the exit when statement
        EXIT WHEN a > 15;
    END LOOP;
END;


**CONTINUE**
DECLARE
    a number(2) := 10;
BEGIN
    -- while loop execution
    WHILE a < 20 LOOP
        dbms_output.put_line ('value of a: ' || a);
        a := a + 1;
        IF a = 15 THEN
            -- skip the loop using the CONTINUE statement
            a := a + 1;
            CONTINUE;
        END IF;
    END LOOP;
END;


```


PL/SQL字符串


 ASCII(x); 返回字符 x 的 ASCII 值
 dbms_output.put_line(ASCII('0'));

 CONCAT(x, y); 连接字符串x和y
 dbms_output.put_line(CONCAT('x', 'y'));

 INITCAP(x); 首字符大写
 dbms_output.put_line(INITCAP('hello'));

 LENGTH(x); 返回x中的字符数
 dbms_output.put_line(LENGTH('hello'));

 LPAD(x, width[, pad_string]); 左填充空格(或指定字符)达到指定宽度
 dbms_output.put_line(LPAD('hello', 10, '0'));

 LTRIM(x [, trim_string]); 修剪最左侧的空字符(或指定字符)
 dbms_output.put_line(LTRIM('hello', 'h'));

PL/SQL游标

在PL/SQL块中执行CRUD操作时，ORACLE会在内存中为其分配上下文区。用数据库语言来描述游标就是：映射在上下文区结果集中一行数据上的位置实体。

用户可以使用游标访问结果集中的任意一行数据，将游标指向某行后，即可对该行数据进行操作。游标为应用提供了一种对具有多行数据查询结果集中的每一行数据分别进行单独处理的方法，是设计嵌入式SQL语句的应用程序的常用编程方式。

Oracle中的游标有两种：

显式游标：

用CURSOR...IS 命令定义的游标，它可以对查询语句(SELECT)返回的多条记录进行处理。

隐式游标：

是在执行插入(INSERT)、删除(DELETE)、修改(UPDATE)和返回单条记录的查询(SELECT)语句时有PL/SQL自动定义的。

显式游标

显式游标使用主要有四个步骤：

--声明/定义游标

--打开游标，执行游标所对应的SELECT语句，将其查询结果放入工作区，并且指针指向工作区的首部

--读取数据，检索结果集中的数据行，放入指定的输出变量中

--关闭游标

声明/定义游标

```
CURSOR cursor_name
  [(parameter_dec [, parameter_dec ]...)]
  [RETURN datatype]
  IS
    select_statement;
```

示例：

*/*带有return的游标*/*

```
CURSOR c1 RETURN departments%ROWTYPE IS
  SELECT * FROM departments
  WHERE department_id = 110;
```

*/*带有传入参数的游标*/*

```
CURSOR c4(sal number) IS
  SELECT employee_id, job_id, salary FROM employees
  WHERE salary > sal;
```

打开游标

```
OPEN cursor_name [ ( cursor_parameter [ [,] actual_cursor_parameter ]... ) ]
```

示例：

*/*打开游标并指定入参*/*

```
OPEN c4 (1300);
```

读取数据

```
FETCH { cursor | cursor_variable | :host_cursor_variable }
  { into_clause | bulk_collect_into_clause [ LIMIT numeric_expression ] } ;
```

示例：

```
fetch c4 into eid, jid, sal;
```

关闭游标

```
CLOSE cursor_name;
```

完整示例

```
DECLARE
  -- 定义游标
  CURSOR c_cursor IS
```

```

SELECT first_name || last_name, Salary FROM EMPLOYEES WHERE rownum<11;
-- 声明变量
v_ename EMPLOYEES.first_name%TYPE;
v_sal EMPLOYEES.Salary%TYPE;
BEGIN
-- 打开游标
OPEN c_cursor;
-- 获取数据
FETCH c_cursor INTO v_ename, v_sal;
-- 处理数据
WHILE c_cursor%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE(v_ename||'---'||to_char(v_sal) );
    FETCH c_cursor INTO v_ename, v_sal;
END LOOP;
-- 关闭游标
CLOSE c_cursor;
END;
```

****显式游标属性****

游标的状态（如是否打开，获取了多少行数据等）可以使用游标属性来获取。

游标属性以“%属性名”的形式加在游标名之后。

属性名	说明
%FOUND	如果记录成功获取，返回TRUE，否则返回FALSE
%NOTFOUND	如果记录获取失败，返回TRUE，否则返回FALSE
%ROWCOUNT	返回已经从游标中获取的记录数
%ISOPEN	如果游标是打开的，返回TRUE，否则返回FALSE

示例：

```

DECLARE
    v_empno EMPLOYEES.EMPLOYEE_ID%TYPE;
    v_sal EMPLOYEES.Salary%TYPE;
-- 定义游标
CURSOR c_cursor IS SELECT EMPLOYEE_ID, Salary FROM EMPLOYEES;
BEGIN
-- 打开游标
OPEN c_cursor;
LOOP
-- 获取数据
FETCH c_cursor INTO v_empno, v_sal;
EXIT WHEN c_cursor%NOTFOUND; -- 未读取到记录，则退出循环
IF v_sal<=1200 THEN
    UPDATE EMPLOYEES SET Salary=v_sal+50 WHERE EMPLOYEE_ID=v_empno;
    DBMS_OUTPUT.PUT_LINE('编码为'||v_empno||'工资已更新!');
END IF;
    DBMS_OUTPUT.PUT_LINE('记录数:' || c_cursor %ROWCOUNT);
END LOOP;
-- 关闭游标
CLOSE c_cursor;
END;
```

****基于游标定义记录变量****

使用%ROWTYPE属性不仅可以基于表和视图定义记录变量，也可以基于游标定义记录变量。

为了简化显式游标的数据处理，建议使用基于游标的记录变量存放游标数据。📄

示例：

```

DECLARE
    -- 定义游标
    CURSOR emp_cursor IS SELECT ename,sal FROM emp;
    emp_reocrd emp_cursor%ROWTYPE; -- 游标变量
BEGIN
    -- 打开游标
    OPEN emp_cursor;
    LOOP
        -- 获取记录
        FETCH emp_cursor INTO emp_record;
        EXIT WHEN emp_record%NOTFOUND;
        dbms_output.put_line('雇员名:' || emp_record.ename || ',雇员工资:' || emp_record.sal);
    END LOOP;
    -- 关闭游标
    CLOSE emp_cursor;
END;

```

****游标FOR循环****

游标FOR循环是显示游标的一种快捷使用方式，当FOR循环开始时，游标自动打开（不需要OPEN），每循环一次系统自动读取游标当前行的数据（不需要FETCH），当退出FOR循环时，游标被自动关闭（不需要使用CLOSE）使用游标FOR循环的时候不能使用OPEN语句，FETCH语句和CLOSE语句，否则会产生错误。🔗

示例：

```

DECLARE
    CURSOR emp_cur(vartype number) IS
        SELECT emp_no,emp_zc FROM cus_emp_basic WHERE com_no=vartype;
BEGIN
    FOR person IN emp_cur(123) LOOP
        DBMS_OUTPUT.PUT_LINE('编号:' || person.emp_no || ',地址:' || person.emp_zc);
    END LOOP;
END;

```

隐式游标

如果在PL/SQL块中使用了SELECT语句进行操作，PL/SQL会隐含处理游标定义，而对于非查询语句，如修改、删除操作，则由ORACLE系统自动地为这些操作设置游标并创建其工作区。由系统隐含创建的游标称为隐式游标，隐式游标的名字为SQL。

对于隐式游标的操作，如定义、打开、取值及关闭操作，都由ORACLE系统自动地完成，无需用户进行处理。用户只能通过隐式游标的相关属性，来完成相应的操作。

****隐式游标的属性****

属性名	说明
SQL%FOUND	如果记录成功获取，返回TRUE，否则返回FALSE
SQL%NOTFOUND	如果记录获取失败，返回TRUE，否则返回FALSE
SQL%ROWCOUNT	返回已经从游标中获取的记录数
SQL%ISOPEN	如果游标是打开的，返回TRUE，否则返回FALSE

示例：

```

DECLARE
    v_rows NUMBER;
BEGIN
    -- 更新表数据
    UPDATE employees SET salary = 5000 WHERE department_id = 90 AND job_id = 'AD_VP';
    -- 获取受影响行数
    v_rows := SQL%ROWCOUNT;

```

```
DBMS_OUTPUT.PUT_LINE('更新了'||v_rows||'个员工的工资');  
END;
```

PL/SQL异常

异常分为预定义异常和用户自定义异常。预定义异常是由系统定义的异常。由于他们已在STANDARD包中预定义了，因此，这些预定义异常可以直接在程序中使用，而不用在预定义部分声明。而用户自定义异常则需要在定义部分声明后才能在可执行部分使用。

常见预定义异常

dup_val_on_index	违反唯一性约束	√
timeout_on_resource	等待资源超时	
transaction_backed_out	由于死锁使事务回退	
invalid_cursor	执行了非法的游标操作，例如关闭了一个已经关闭的游标	
not_logged_on	没有连接到oracle	
login_denied	错误的用户名或口令	
no_data_found	select..into..没有返回任何一行，或者试图使用一个未被赋值的pl/sql变量	√
too_many_rows	一条select..into语句查到多行	√
zero_divide	被0除	√
invalid_number	未能将sql中的字符串转换成数字	
storage_error	pl/sql运行时内存溢出	
program_error	内部pl/sql错误，属于系统异常	
value_error	在过程性语句中出现转换，截断，算术错误	
rowtype_mismatch	宿主游标变量与pl/sql变量有不兼容行类型	
cursor_already_open	试图打开一个已打开的游标	
access_into_null	试图给一个null对象的属性赋值	
collection_is_null	试图对一个null的表或变长数组执行除exists以外的操作	
subscript_outside_limit	引用的嵌套表或变长数组索引超出了其声明范围pl/sql2.0以上	
subscript_beyond_count	引用的嵌套表或变长数组索引大于表中元素个数pl/sql8.0	

EXCEPTION

```
WHEN first_exception THEN <code to handle first exception >  
WHEN second_exception THEN <code to handle second exception >  
WHEN OTHERS THEN <code to handle others exception >
```

END;

示例:

DECLARE

```
c_id customers.id%type := 8;  
c_name customers.name%type;  
c_addr customers.address%type;
```

BEGIN

```
SELECT name, address INTO c_name, c_addr  
FROM customers  
WHERE id = c_id;  
DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);  
DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
```

EXCEPTION

```
WHEN no_data_found THEN  
    dbms_output.put_line('No such customer!');  
WHEN others THEN  
    dbms_output.put_line('Error!');
```

END;

PL/SQL触发器

PL/SQL存储过程

<https://www.cnblogs.com/enjoyjava/p/9131169.html>

```
/*存储过程定义*/
create or replace procedure 过程名称 [(参数列表)] as/is
begin
...
end 过程名称;
```

```
/*存储过程调用*/
begin
    过程名称;
end;
```

```
/*带输入参数的存储过程实例*/ ☆
create or replace procedure p_query(i_empno IN emp.empno%TYPE) as
--声明变量
    v_ename emp.ename%TYPE;
    v_sal    emp.sal%TYPE;
begin
    SELECT ename,sal INTO v_ename,v_sal FROM emp WHERE empno = i_empno;
    --打印变量
    DBMS_OUTPUT.PUT_LINE('姓名: ' || V_ENAME || '薪水: ' || V_SAL);
end p_query;
```

```
/*存储过程调用*/ ☆
begin
    p_query(7839);
end;
```

```
/*带输入输出的存储过程*/ 🤖
create or replace procedure p_shuchu(i_empno IN emp.empno%TYPE,o_sal OUT emp.sal%TYPE) as
--声明变量
    v_ename emp.ename%TYPE;
    v_sal    emp.sal%TYPE;
begin
    SELECT sal INTO o_sal FROM emp WHERE empno = i_empno;
END;
```

```
/*调用存储过程*/ 🤖
DECLARE
--声明变量
    v_sal emp.sal%TYPE;
begin
    p_shuchu(7839,v_sal);
    dbms_output.put_line('薪水: ' || v_sal);
end;
```

```
doFilter(ServletRequest request, ServletResponse response, FilterChain chain){  
    输出 filterone pre  
    chain.doFilter(request, reponse);  
    输出 filterone after  
}
```