Oracle PL/SQL –
Parameters, Variables, and Views

Using the Single ampersand characters to input column names, table names, and conditions

SQL> select &col1, &col2, &col3
    2  from &table_name
    3  where &condition;
Enter value for col1: last_name
Enter value for col2: first_name
Enter value for col3: wages
old 1: select &col1, &col2, &col3
new 1: select last_name, first_name, wages
Enter value for table_name: employee
old 2: from &table_name
new 2: from employee
Enter value for condition: fk_department = 'POL'
old 3: where &condition
new 3: where fk_department = 'POL'

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
<th>WAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WILSON</td>
<td>WOODROW</td>
<td>9000</td>
</tr>
<tr>
<td>DWORCZAK</td>
<td>ALICE</td>
<td>9800</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>LYNDON</td>
<td>12000</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>ANDREW</td>
<td>7500</td>
</tr>
<tr>
<td>CLINTON</td>
<td>WILLIAM</td>
<td>15000</td>
</tr>
<tr>
<td>NIXON</td>
<td>RICHARD</td>
<td>12500</td>
</tr>
<tr>
<td>KENNEDY</td>
<td>JOHN</td>
<td>11500</td>
</tr>
<tr>
<td>ROOSEVELT</td>
<td>FRANKLIN</td>
<td>10400</td>
</tr>
</tbody>
</table>

8 rows selected.

SQL>

End listing
Using the Double ampersand variable

In the previous examples in this section, each of the variables had a name that was unique to the statement. If you have Single ampersand variables with the same name, Oracle will still prompt you to enter values for each of the variables (unless a permanent variable is created). This can inconvenience a user when the same value must be used several times in a Select statement. Users may not understand why they must enter the same value several times.

This problem is solved by the Double ampersand variable. The Double ampersand variable will cause Oracle to prompt the user for a value and will create a permanent (to the current session) variable. The permanent variable will use the name supplied by the Double ampersand variable. Subsequent Single ampersand variables using the variable name will use the Double ampersand variable value rather than prompt the user for a value.

Using the Double ampersand variable to create a variable

```
SQL> select fk_department, last_name, first_name, tool_name
2  from employee, emp_tools
3  where payroll_number = fk_payroll_number
4    and fk_department = &&department_code
5  union
6  select fk_department, last_name, first_name, tool_name
7  from employee, emp_tools
8  where payroll_number = fk_payroll_number
9    and fk_department = &department_code;
```

Enter value for department_code: 'POL'

```
FK_D LAST_NAME       FIRST_NAME      TOOL_NAME
---- --------------- --------------- --------------------
POL  JOHNSON         ANDREW          Fountain Pen
POL  JOHNSON         ANDREW          Greenberg Optical
POL  JOHNSON         ANDREW          Shovel
POL  JOHNSON         LYNDON          Peralman Optical
POL  NIXON           RICHARD         Downtown Optical
POL  NIXON           RICHARD         Hack Saw
POL  NIXON           RICHARD         Pliers
POL ROOSEVELT        FRANKLIN        CIGARETTE HOLDE
POL ROOSEVELT        FRANKLIN        HYDE PARK OPTICAL
POL ROOSEVELT        FRANKLIN        STAPLER
POL WILSON           WOODROW         DUST PAN
POL WILSON           WOODROW         STERLING OPTICAL
POL WILSON           WOODROW         VACUUM
POL WILSON           WOODROW         VISE GRIPS
```

14 rows selected.

SQL>
Defining user variables

Variable definition commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define variable_name = value</td>
<td>Creates a user variable with a data type of Char. It also assigns a value to the variable.</td>
</tr>
<tr>
<td>Define variable_name</td>
<td>Displays the variable, its datatype, and its value.</td>
</tr>
<tr>
<td>Define</td>
<td>Displays all existing variables. This also includes their values and data type.</td>
</tr>
<tr>
<td>Accept variable_name</td>
<td>Sets up a variable that displays a custom prompt when it is called.</td>
</tr>
</tbody>
</table>

Illustrating the following Define command uses:

- Using the Define command on a variable that does not exist.
- Using the Define command to create and populate a variable
- Using the Define command to view the variable attribute information.
- Using the variable in a Select statement
- Using the Define command to view all variables
Using the Define command

SQL> define user_var
SP2-0135: symbol user_var is UNDEFINED
SQL> define user_var = 35
SQL> define user_var
DEFINE USER_VAR = "35" (CHAR)
SQL> select last_name, first_name
2 from employee
3 where payroll_number = &user_var;
old 3: where payroll_number = &user_var
new 3: where payroll_number = 35

LAST_NAME       FIRST_NAME
--------------- ---------------
REAGAN          RONALD

SQL> define
DEFINE _SQLPLUS_RELEASE = "801050000" (CHAR)
DEFINE _EDITOR = "Notepad" (CHAR)
DEFINE _O_VERSION = "Oracle8i Personal Edition Release 8.1.5.0.0 - Production"
With the Java option
PL/SQL Release 8.1.5.0.0 - Production" (CHAR)
DEFINE _O_RELEASE = "801050000" (CHAR)
DEFINE USER_VAR = "35" (CHAR)
SQL>

End Listing

Administering Database Views

Create [or replace] [force|noforce] view viewname as select statement
[with check option [constraint constraint]]
[with read only];
Using a view to mask the SQL complexity

```
SQL> create or replace view employee_data as
2  select department_name, last_name||', '||first_name
3     name,
4     wages, birth_date,
5     trunc(months_between(sysdate, birth_date)/12,0)
6     age,
7     employment_date,
8     trunc(months_between(sysdate, birth_date)/12,0)
9     seniority_yrs
10    from department, employee
11   where department = fk_department;
View created.
```

```
SQL> select department_name, name, seniority_yrs
2  from employee_data
3  order by seniority_yrs desc;
```

```
DEPARTMENT_NAME NAME
--------------- -------------------------------- -----------
POLITICAL SCIEN WILSON, WOODROW 144
WELFARE BUREAU TAFT, WILLIAM 143
INTERIOR DESIGN ROOSEVELT, THEODORE 142
INTERIOR DESIGN COOLIDGE, CALVIN 129
WELFARE BUREAU HOOVER, HERBERT 127
POLITICAL SCIEN ROOSEVELT, FRANKLIN 119
INTERIOR DESIGN TRUMAN, HAROLD 117
WELFARE BUREAU ROOSEVELT, ELEANOR 116
INTERIOR DESIGN EISENHOWER, DWIGHT 110
```

```
SQL> desc employee_data
```

```
Name                                             Null?    Type
---------------------------------------------- -------- --------------
DEPARTMENT_NAME                                          VARCHAR2(15)
NAME                                                     VARCHAR2(32)
WAGES                                                    NUMBER(8,2)
BIRTH_DATE                                               DATE
AGE                                                      NUMBER
EMPLOYMENT_DATE                                          DATE
SENIORITY_YRS                                            NUMBER
```

```
SQL>
```
Create view command options

<table>
<thead>
<tr>
<th>View Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or Replace</td>
<td>Replaces an existing view with the new one. If this option is missing, Oracle will not allow you to overwrite an existing view. It must be dropped first.</td>
</tr>
<tr>
<td>Force</td>
<td>Allows the view to be created even if the tables do not exist or if the view references any invalid components.</td>
</tr>
<tr>
<td>Noforce</td>
<td>Allows the view to be created only if the tables exist and the components are valid. This is the default.</td>
</tr>
<tr>
<td>With Check Option</td>
<td>Restricts DML operations to only the rows that are accessible to the view.</td>
</tr>
<tr>
<td>With Read Only</td>
<td>Ensures that no DML operations can be performed using the view.</td>
</tr>
</tbody>
</table>

DML and views

View type properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Simple View</th>
<th>Complex view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tables</td>
<td>One</td>
<td>One or more</td>
</tr>
<tr>
<td>Contains functions</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Grouped Values</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DML Allowed</td>
<td>Yes</td>
<td>Not Always</td>
</tr>
</tbody>
</table>
Simple views can be used in DML statements. This is because the records displayed in a simple view relate to one row of a table. Complex view records are a virtual record and do not generally relate to one row of a table. The Delete command cannot be used against a complex view if it has any of the following features:

- View contains a group function.
- View contains a Group By clause.
- View contains the Distinct keyword.
- View contains the pseudocolumn Rownum.

A view cannot be updated if the view contains any of the above properties. It also cannot be updated if the view contains the following property:

- View contains a column that is defined by expressions. These are columns that are concatenated or modified with functions such as Substr or Add_months.
A view cannot be used to add records to the database if it contains any of the above properties. It also cannot be used to insert records if it contains the following property:

? The base tables contain Not Null constrained columns that do not exist in the view.

The With Check Option
Examples of creating a view with the With Check Option and executing Update commands against the view

```
SQL> create view employee_names as
      2  select fk_department, last_name, first_name
      3  from employee
      4  where fk_department = 'INT'
      5  with check option;
View created.

SQL> update employee_names set fk_department = 'WEL'
      2  where first_name = 'DWIGHT';
update employee_names set fk_department = 'WEL'

ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation

SQL> update employee_names set first_name = 'IKE'
      2  where first_name = 'DWIGHT';
1 row updated.

SQL> update employee set fk_department = 'WEL'
      2  where first_name = 'IKE';
1 row updated.
```

With Check Option prevents the modification of the row
Update statement successfully updates the table through the view.
With Check Option does not affect normal DML operations
The Read Only Option

Using the Read Only option

```sql
SQL> create view employee_names as
2    select fk_department, last_name, first_name
3    from employee
4    where fk_department = 'INT'
5    with read only;

View created.

SQL> update employee_names set first_name = 'IKE'
2    where first_name = 'DWIGHT';
where first_name = 'DWIGHT'
*
ERROR at line 2:
ORA-01733: virtual column not allowed here

SQL>
End Listing
```

How to update records using a view that can't be updated

Oracle has developed a tool that allows a developer to update views that are not normally updateable. This tool is called an Instead Of trigger. This trigger can only be used if the target is a view. It executes the trigger’s PL/SQL rather than the DML transaction that launched the trigger.
create or replace view wages_difference as

    select employee.fk_department, payroll_number, first_name, last_name, wages, employee_purchases/(dept_purchases/employee_amt) percent_of_Department_Avg
    from employee, (select fk_payroll_number, sum(cost) employee_purchases
                  from (select fk_payroll_number, nvl(tool_cost,0) cost
                         from emp_tools
                         union all
                         select fk_payroll_number, nvl(cost,0)
                         from glasses)
                  group by fk_payroll_number) e,
    (select fk_department, sum(cost) dept_purchases
     from (select fk_department, nvl(tool_cost,0) cost
           from employee, emp_tools
           where payroll_number = fk_payroll_number(+)
           union all
           select fk_department, nvl(cost,0)
           from employee, glasses
           where Payroll_number = fk_payroll_number(+))
     group by fk_department) f,
    (select fk_department, count(*) employee_amt
     from employee
     group by fk_department) c
    where employee.payroll_number = e.fk_payroll_number(+)
    and employee.fk_department = f.fk_department(+)
    and employee.fk_department = c.fk_department;

View created.