Title

P16. Use record variables with SELECT, UPDATE, and INSERT statements.

Description

A record variable is a composite data type that allows you to store multiple values of different data types in a single variable.

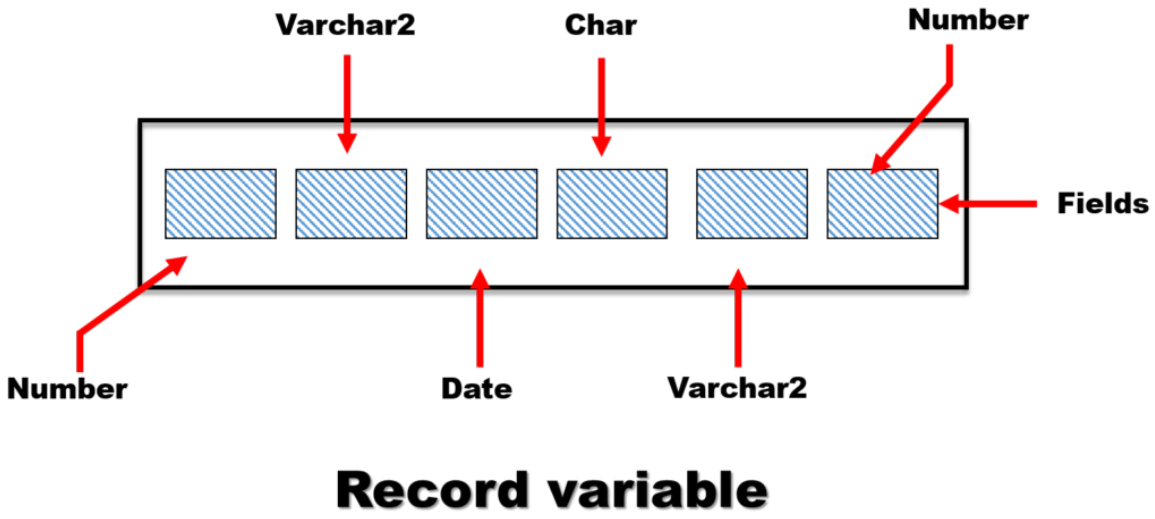

Image source: How to Initialize Table Based Record Type Variables In Oracle Database | Rebellion-Rider

A question may arise: when to use record variables in PL/SQL?

The best use cases are when you need to work with an entire row of a table. That is: 1. select all columns of a row into a record variable. 2. Update a row with the values in a record variable. 3. Insert a row with the values in a record variable.

Examples

Example 1: Select all columns of a row into a record variable

For example, you want to select all columns of an employee with employee ID 100 and store them in a local variable in PL/SQL for further processing.

You can declare a record variable with the same structure as the `employees` table and select the row into the record variable.

The `%rowtype` is used to define a record variable with the same structure as a table. You don't have to define the record type manually for the `employees` table.

```
declare
    l_emp_rec employees%rowtype;
begin
    select * into l_emp_rec from employees where employee_id = 100;
end;
/
```

Example 2: Update a row with the values in a record variable

Now, we have fetched a row from a table and stored it in a record variable.

Assume we increase the value of the salary field in the record by 10% and want to update the row back to the table.

In the UPDATE statement, you can use the `ROW` keyword to represent the target being updated and use the record variable to update the row.

The resultant block is as follows:

```
declare
    l_emp_rec employees%rowtype;
begin
    -- Fetch the row into the record variable
    select * into l_emp_rec from employees where employee_id = 100;

    -- Update the salary field in the record
    l_emp_rec.salary := l_emp_rec.salary * 1.1;
    -- Update the row back to the table
    update employees
        set ROW = l_emp_rec
        where employee_id = 100;
end;
/
```

In the above block, `set ROW = l_emp_rec` is used to update the entire row with the values in the record variable.

Oracle unpacks the ROW keyword and the record variable to columns automatically to assign values to columns. If the record variable does not have the same structure as the table, the block will raise an exception.

For example, the following block will raise an exception because we cannot assign a record variable to a column of salary data type (scalar data type).

```
declare
    l_emp_rec employees%rowtype;
begin
    -- Fetch the row into the record variable
    ...

    -- Update the salary field in the record
    ...
    -- Update the row back to the table
    update employees
        set salary = l_emp_rec
        where employee_id = 100;
end;
/
```

Example 3: Insert a row with the values in a record variable

Assume we have the following table:

```
create table emp (
    emp_id number,
    fname varchar2(50),
    salary number
);
```

We want to insert a row with the values 9000, 'Tom', and 8000 into the `emp` table.

You can declare a record variable with the same structure as the `emp` table. Then, use the SELECT INTO statement to populate values into the record variable. Finally, use the INSERT statement to insert the row into the `emp` table.

The block is as follows:

```
declare
    l_emp_rec emp%rowtype;
begin
    -- Populate values into the fields of the record variable
    select 9000, 'Tom', 8000 into l_emp_rec from dual;
    -- Insert the row into the table
    insert into emp values l_emp_rec;
end;
/
```

In the above block, the INSERT statement uses the record variable `l_emp_rec` to insert the row into the `emp` table. The `l_emp_rec` record variable is placed after the VALUES keyword in the INSERT

statement. Oracle unpacks the record variable to columns automatically for insertion.

References

1. Feuerstein, S., Working with records and pseudorecords in PL/SQL, Oracle Connect, https://blogs.oracle.com/connect/post/working-with-records