Oracle Database 11g & PL/SQL

Tips and Tricks

Marcelo Vinícius Cysneiros Aragão
http://www.contactify.com/bf737

Topics

- Data Manipulation
  - DB Link
  - Insert as Select
  - Cursor
  - Associative Array
- Exception Handling
  - The usual way
  - SQLCODE and SQLERRM
  - Saving and Retrieving
- Hierarchical Queries
  - Keywords
  - Example
- Tips and Tricks
  - Modular Parallelization
  - Function within Procedure
  - NVL and NVL2
  - COALESCE and NULLIF
  - Regular Expression Substring

Data Manipulation

Data Manipulation: DB Link

- The following statement defines a shared public database link named `remote` that refers to the database specified by the service name `remote`:

```
CREATE PUBLIC DATABASE LINK remote
    USING 'remote';
```

- This database link allows user `hr` on the `local` database to update a table on the `remote` database (assuming `hr` has appropriate privileges):

```
UPDATE employees@remote
    SET salary=salary*1.1
    WHERE last_name = 'Baer';
```

Data Manipulation: Insert as Select

- With `INSERT ... SELECT`, you can quickly insert many rows into a table from one or many tables.

```
INSERT INTO suppliers(supplier_id, supplier_name)
    SELECT account_no, name
    FROM customers
    WHERE customer id > 5000;
```

- **Question:** How do I make sure that I do not enter the same client information again?

```
INSERT INTO clients(client_id, client_name, client_type)
    SELECT supplier_id, supplier_name, 'advertising'
    FROM suppliers
    WHERE NOT EXISTS (SELECT *
                      FROM clients
                      WHERE clients.client_id = suppliers.supplier_id);
```

Data Manipulation: Cursor

```
DECLARE

    l_total INTEGER := 10000;

    CURSOR employee_id_cur IS
        SELECT employee_id
        FROM plch_employees
        ORDER BY salary ASC;

    l_employee_id employee_id_cur%ROWTYPE;

--DECLARE-END
```

Data Manipulation: Cursor

```
BEGIN

    OPEN employee_id_cur;

    LOOP
    FETCH employee_id_cur INTO l_employee_id;
    EXIT WHEN employee_id_cur%NOTFOUND;

    assign_bonus (l_employee_id, l_total);
    EXIT WHEN l_total <= 0;
    END LOOP;

    CLOSE employees_cur;

END;
```

Data Manipulation: Associative Array

```
create or replace package associative_array as

    -- define an associative array type for each column in the jobs table

    type t_job_id     is table of jobs.job_id%type     index by pls_integer;
    type t_job_title  is table of jobs.job_title%type  index by pls_integer;
    type t_min_salary is table of jobs.min_salary%type index by pls_integer;
    type t_max_salary is table of jobs.max_salary%type index by pls_integer;

    -- define the procedure that will perform the array insert

    procedure array_insert (p_job_id     in t_job_id,
                            p_job_title  in t_job_title,
                            p_min_salary in t_min_salary,
                            p_max_salary in t_max_salary);

end associative_array;
```

Data Manipulation: Associative Array

```
create or replace package body associative_array as

    -- implement the procedure that will perform the array insert

    procedure array_insert (p_job_id     in t_job_id,
                            p_job_title  in t_job_title,
                            p_min_salary in t_min_salary,
                            p_max_salary in t_max_salary) is
    begin
        forall i in p_job_id.first..p_job_id.last
            insert into jobs (job_id,
                              job_title,
                              min_salary,
                              max_salary)
            values (p_job_id(i),
                    p_job_title(i),
                    p_min_salary(i),
                    p_max_salary(i));
    end array_insert;

end associative_array;
```

Exception Handling

Data Manipulation: The usual way

```
DECLARE

   pe_ratio NUMBER(3,1);

BEGIN

   SELECT price / earnings INTO pe_ratio FROM stocks
       WHERE symbol = 'XYZ';  -- might cause division-by-zero error
   INSERT INTO stats (symbol, ratio) VALUES ('XYZ', pe_ratio);
   COMMIT;

EXCEPTION  -- exception handlers begin

   WHEN ZERO_DIVIDE THEN  -- handles 'division by zero' error
      INSERT INTO stats (symbol, ratio) VALUES ('XYZ', NULL);
      COMMIT;
   ...
   WHEN OTHERS THEN  -- handles all other errors
      ROLLBACK;

END;  -- exception handlers and block end here
```

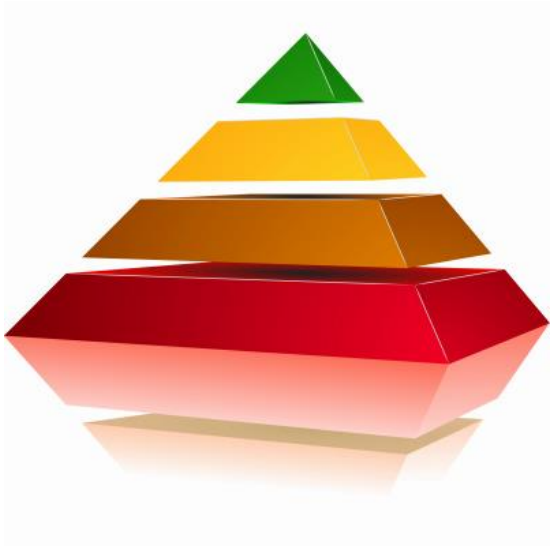
Data Manipulation: SQLCODE and SQLERRM

```
DECLARE

   name employees.last_name%TYPE;
   v_code NUMBER;
   v_errm VARCHAR2(64);

BEGIN

   SELECT last_name INTO name FROM employees WHERE employee_id = 1000;

   EXCEPTION

      WHEN OTHERS THEN
         v_code := SQLCODE;
         v_errm := SUBSTR(SQLERRM, 1 , 64);
         DBMS_OUTPUT.PUT_LINE('The error code is ' || v_code || '- ' || v_errm);

END;
```

Data Manipulation: Saving and Retrieving

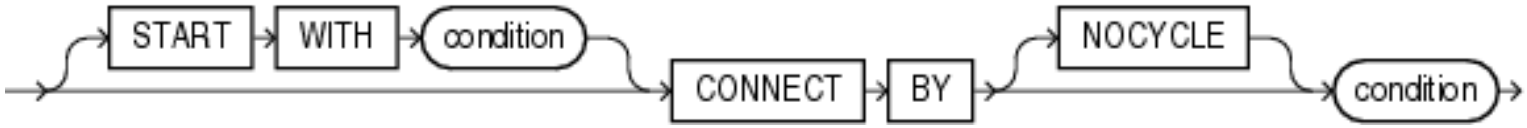
```
-- Perform a bulk operation.

BEGIN

    FORALL i IN l_tab.first .. l_tab.last SAVE EXCEPTIONS
        INSERT INTO exception_test
        VALUES l_tab(i);

EXCEPTION

    WHEN OTHERS THEN

        l_error_count := SQL%BULK_EXCEPTIONS.count;
        DBMS_OUTPUT.put_line('Number of failures: ' || l_error_count);

        FOR i IN 1 .. l_error_count LOOP
            DBMS_OUTPUT.put_line('Error: ' || i ||
                ' Array Index: ' || SQL%BULK_EXCEPTIONS(i).error_index ||
                ' Message: ' || SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
        END LOOP;

END;
```

Hierarchical Queries



Hierarquical Queries: Keywords

- If a table contains hierarchical data, then you can select rows in a hierarchical order using the hierarchical query clause:



- **START WITH** specifies the root row(s) of the hierarchy.
- **CONNECT BY** specifies the relationship between parent rows and child rows of the hierarchy.
- In a hierarchical query, one expression in *condition* must be qualified with the **PRIOR** operator to refer to the parent row.
- The **LEVEL** pseudocolumn is used to show parent and child rows
- The **SIBLINGS** keyword is used to preserve ordering within the hierarchy.

Hierarquical Queries: Example

```
SELECT last_name, employee_id, manager_id, LEVEL
    FROM employees
    START WITH employee_id = 100
    CONNECT BY PRIOR employee_id = manager_id
    ORDER SIBLINGS BY last_name;
```

LAST_NAME	EMPLOYEE_ID	MANAGER_ID	LEVEL
King	100		1
Cambrault	148	100	2
Bates	172	148	3
Bloom	169	148	3
Fox	170	148	3
De Haan	102	100	2
Hunold	103	102	3
Austin	105	103	4
Ernst	104	103	4
Errazuriz	147	100	2
Ande	166	147	3

Tips and Tricks

Tips and Tricks: Modular Parallelization

```
CREATE OR REPLACE PROCEDURE MOD_PROCESS_EMPLOYEES
(
    max_value    IN  NUMBER,
    remainder    IN  NUMBER
) IS

    BEGIN

        SELECT * FROM EMPLOYEE
        WHERE MOD(EMPLOYEE.ID, max_value) = remainder;

END MOD_PROCESS_EMPLOYEES;
```

- If we call MOD_PROCESS_EMPLOYEES(2, 0), only the records with even IDs will be processed.
- If we call MOD_PROCESS_EMPLOYEES(2, 1), only the records with odd IDs will be processed.

Tips and Tricks: Function within Procedure

```
CREATE OR REPLACE PROCEDURE TEST_SCOPE (proc_p1 IN NUMBER, proc_p2 IN NUMBER) IS

    procedure_scoped_var BOOLEAN;

    FUNCTION SUMMATION(func_p1 IN NUMBER, func_p2 IN NUMBER) RETURN NUMBER IS
        function_scoped_var BOOLEAN;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('INNER FUNCTION');
        RETURN func_p1 + func_p2;
    END soma;

BEGIN
    DBMS_OUTPUT.PUT_LINE('OUTER PROCEDURE');
    DBMS_OUTPUT.PUT_LINE('SUMMATION = ' || SUMMATION(proc_p1,proc_p2));
    EXCEPTION
      WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Exception raised');
        RAISE;

END TEST_SCOPE;
```

Tips and Tricks: NVL and NVL2

- The NVL function allows you to replace null values with a default value.
  - If the value in the first parameter is null, the function returns the value in the second parameter.
  - If the first parameter is any value other than null, it is returned unchanged.

```
SELECT id, NVL(col1, 'ZERO') AS output FROM null_test_tab ORDER BY id;
```

- The NVL2 function accepts three parameters.
  - If the first parameter value is not null it returns the value in the second parameter.
  - If the first parameter value is null, it returns the third parameter.

```
SELECT id, NVL2(col1, col2, col3) AS output FROM null_test_tab ORDER BY id;
```

Tips and Tricks: COALESCE and NULLIF

- The COALESCE accepts two or more parameters and returns the first non-null value in a list. If all parameters contain null values, it returns null.

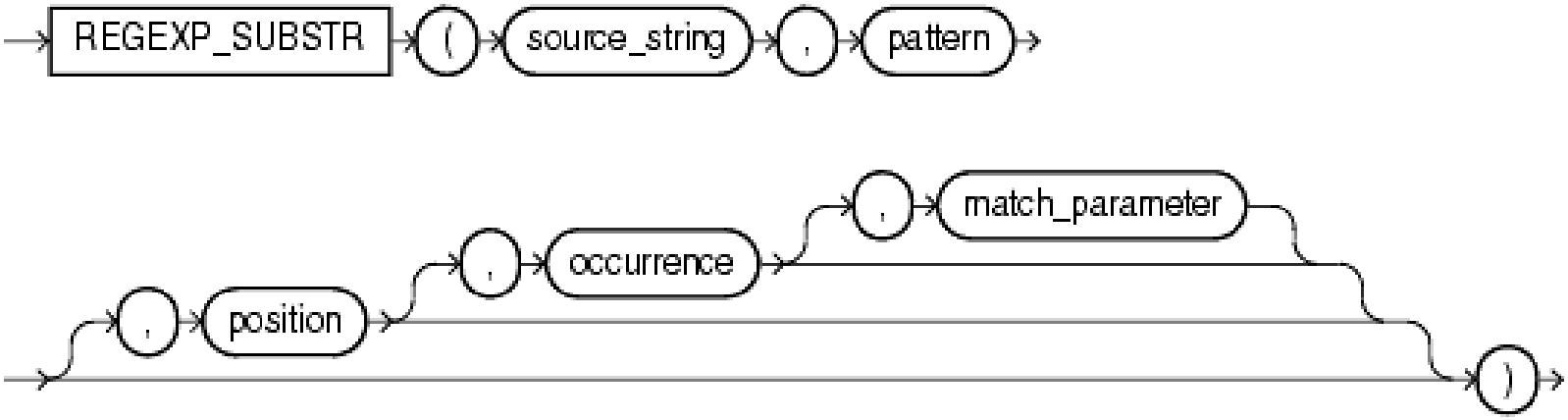
```
SELECT id, COALESCE(col1, col2, col3) AS output FROM null_test_tab ORDER BY id;
```

- The NULLIF accepts two parameters and returns null if both parameters are equal. If they are not equal, the first parameter value is returned.

```
SELECT id, NULLIF(col3, col4) AS output FROM null_test_tab ORDER BY id;
```

Tips and Tricks: Regular Expression Substring

- The function REGEXP_SUBSTR is useful if you need the contents of a match string but not its position in the source string.



- The function returns the string as VARCHAR2 or CLOB data in the same character set as *source_string*.

Tips and Tricks: Regular Expression Substring

```
SET SERVEROUTPUT ON;

DECLARE

    source_string VARCHAR2(32) := '1one1|2two2|3three3';

BEGIN

    dbms_output.put_line(REGEXP_SUBSTR( source_string, '[^|]+', 1, 1 ));
    dbms_output.put_line(REGEXP_SUBSTR( source_string, '[^|]+', 1, 2 ));
    dbms_output.put_line(REGEXP_SUBSTR( source_string, '[^|]+', 1, 3 ));

END;
```

Output:

```
1one1
2two2
3three3
```