Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Advanced PLSQL

Lesson 01 – Object-Types and Collect

Lesson Objectives

On completion of this lesson on Bulk-Operations, you will be able to:

- Create and use Object-types
- Understand collections and collection types
- Use of Object-types and Collection-types

Working with Object Types

- An Object, such as car, an order, a person, etc has specific attributes and behaviors
- An Object is a user-defined composite data type
- It encapsulates a data structure along with the functions and procedures to manipulate the data
- The variables that make up the data structures are called attributes
- The functions and procedures that characterize the behavior are called member methods in PL/SQL

Object-type Specification and Body

Object Type Structure:

- Like an Oracle database package, an object type has two parts:
  1. The object type specification:

     Is the interface to your applications. It declares the data structure (set of attributes) along with the operations (methods) to access, use, or manipulate the data. All declarations are public.
  2. The object type body:

     Implements the specification. It fully defines the methods.

- In object type specification, all attributes must be declared before any methods.
- You can not declare attributes in the object type body.

Syntax for Object-type Specification and Body

Syntax: Object Type Specification

```
CREATE [OR REPLACE] TYPE type_name IS/AS OBJECT
(           attribute1      datatype,
            attribute2      datatype,
 --------
            [MEMBER   procedure | function specification,
            MEMBER    procedure | function specification]
);
/
```

Syntax: Object Type Body

```
CREATE [OR REPLACE] TYPE BODY type_name IS/AS
            MEMBER  procedure_body | function_body;
            MEMBER  procedure_body | function_body;
END;
/
```

Example of Object-type Specification and Body

- Creating Object Type Specification:

```
CREATE OR REPLACE TYPE name_obj_type AS OBJECT
(  f_name VARCHAR2(20),
   l_name VARCHAR2(20),
   MEMBER FUNCTION full_name RETURN VARCHAR2
);
/
```

- Creating Object Type Body:

```
CREATE OR REPLACE TYPE BODY name_obj_type AS
   MEMBER FUNCTION full_name RETURN VARCHAR2
   IS
   BEGIN
            RETURN ( f_name || ' ' || l_name );
   END full_name;
   END;
/
```

Performance Difference created by Bulk-bind

- Performance difference in adding 10000 rows through a normal loop and through Bulk-bind :

```
Create table parts(partno number, partname varchar2(40));
 DECLARE
        TYPE NumTab IS TABLE OF NUMBER(6) INDEX BY BINARY_INTEGER;
        TYPE NameTab IS TABLE OF CHAR(25) INDEX BY BINARY_INTEGER;
        pnums NumTab;
        pnames NameTab;
        t1 NUMBER(9,6);
        t2 NUMBER(9,6);
        t3 NUMBER(9,6);
        PROCEDURE get_time (t OUT NUMBER) IS
        BEGIN
                select substr(systimestamp,17,9) into t from dual;
        END;
BEGIN
        FOR j IN 1..10000 LOOP -- load index-by tables
                pnums(j) := j;
                pnames(j) := 'Part No. ' || TO_CHAR(j);
        END LOOP;
        get_time(t1);
```

Performance Difference with Bulk-bind ….contd

```
  FOR i IN 1..10000 LOOP -- use FOR loop
          INSERT INTO parts VALUES (pnums(i), pnames(i));
  END LOOP;
  get_time(t2);
  FORALL i IN 1..10000 -- use FORALL statement INSERT INTO parts VALUES (pnums(i),
  pnames(i));
  get_time(t3);
  dbms_output.put_line('Execution Time (secs)');
  dbms_output.put_line('FOR loop: ' || TO_CHAR(t2 - t1));
  dbms_output.put_line('FORALL: ' || TO_CHAR(t3 - t2));
 END;
/
Execution Time (secs)
FOR loop: .194
FORALL: .007
```

FORALL command for Bulk-bind

- The keyword FORALL instructs the PL/SQL engine to bulk-bind input collections before sending them to the SQL engine with an INSERT/UPDATE/DELETE statement. Although the FORALL statement contains an iteration scheme, it is *not* a FOR loop. Its syntax is as follows:

```
FORALL index IN lower_bound..upper_bound sql_statement;
```

- The index can be referenced only within the FORALL statement and only as a collection subscript. The SQL statement must be an INSERT, UPDATE, or DELETE statement that references collection elements. And, the bounds must specify a valid range of consecutive index numbers. The SQL engine executes the SQL statement once for each index number in the range. But that happens in the SQL engine.

Arbitary collection-slices can be bound using FORALL

- You can use the bounds to bulk-bind arbitrary slices of a collection as follows :

```
DECLARE
        TYPE NumList IS VARRAY(15) OF NUMBER;
        depts NumList := NumList();
BEGIN
        Depts:=numlist(10,20,30,40,50,60);
        FORALL j IN 3..5 -- bulk-bind middle third of varray
        UPDATE emp SET sal = sal * 1.10 WHERE deptno = depts(j);
END;
/
```

How FORALL affects Rollbacks

- In a FORALL statement, if any execution of the SQL statement raises an unhandled exception, all database changes made during previous executions are rolleded back, if the exception goes unhandled. However, if a raised exception is caught and handled, changes are automatically rolled back to an implicit savepoint marked before each execution of the SQL statement. Thus, Changes made during previous executions are *not* rolled back.

```
CREATE TABLE zzz(c1 number, c2 varchar2(15));

INSERT INTO zzz vaLUES(10, 'Clerk');
INSERT INTO zzz vaLUES(10, 'Clerk');
INSERT INTO zzz vaLUES(20, 'Bookkeeper');
INSERT INTO zzz VALUES(30, 'Analyst');
INSERT INTO zzz VALUES(30, 'Analyst');
Commit;
```

How FORALL affects Rollbacks ……contd

➢Try to append the 7-character string ‘ (temp)’ to certain job titles using the following UPDATE statement:

```
DECLARE
        TYPE NumList IS TABLE OF NUMBER;
        depts NumList := NumList(10, 20, 30);
BEGIN
        FORALL j IN depts.FIRST..depts.LAST UPDATE zzz SET c2 = c2 || ' (temp)'  WHERE c1 =
        depts(j);
        -- raises a "value too large" exception
 EXCEPTION
        WHEN OTHERS THEN
                COMMIT;
 END;
 /
```

How FORALL affects Rollbacks ……contd

- Try to append the 7-character string ‘ (temp)’ to certain job titles using the following UPDATE statement:

```
DECLARE
        TYPE NumList IS TABLE OF NUMBER;
        depts NumList := NumList(10, 20, 30);
BEGIN
        FORALL j IN depts.FIRST..depts.LAST UPDATE zzz SET c2 = c2 || '        (temp)'  WHERE c1 =
  depts(j);
        -- raises a "value too large" exception
EXCEPTION
        WHEN OTHERS THEN
                COMMIT;
END;
/
select * from zzz;
```

How FORALL affects Rollbacks …..contd

- The SQL engine is supposed to execute the UPDATE statement three times, once for each index number in the specified range, that is, once for depts(10), once for depts(20), and once for depts(30). The first execution succeeds, but the second execution fails because the string value 'Bookkeeper (temp)' is too large for the job column. In this case, only the second execution is rolled back. When any execution of the SQL statement raises an exception, the FORALL statement halts. In our example, the second execution of the UPDATE statement raises an exception, so the third execution is never done.

Counting Rows Affected by FORALL Iterations

- To process SQL data manipulation statements, the SQL engine opens an implicit cursor named SQL. This cursor's scalar attributes, %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT, return useful information about the most recently executed SQL data manipulation statement.

- The SQL cursor has one composite attribute, %BULK_ROWCOUNT, designed for use with the FORALL statement. This attribute has the semantics of an index-by table. Its *i*th element stores the number of rows processed by the *i*th execution of an INSERT, UPDATE or DELETE statement. If the *i*th execution affects no rows, %BULK_ROWCOUNT(i) returns zero.

Counting Rows Affected by FORALL Iterations

- To show the number of rows affected by each execution of the DML :

```
DECLARE
        TYPE NumList IS TABLE OF NUMBER;
        depts NumList := NumList(10, 20, 30,70);
BEGIN
        FORALL j IN depts.FIRST..depts.LAST UPDATE emp SET sal = sal * 1.10
        WHERE deptno = depts(j);
        For i in 1.. depts.count loop
                Dbms_output.put_line(SQL%BULK_ROWCOUNT(i));
        End loop;
END;
/
```

Counting Rows Affected by FORALL Iterations

- %BULK_ROWCOUNT is usually equal to 1 for inserts, because a typical insert operation affects only a single row. But for the INSERT ... SELECT construct, %BULK_ROWCOUNT might be greater than 1.

```
Create table emp_by_dept(empno number, deptno number);

DECLARE
        TYPE num_tab IS TABLE OF NUMBER;
        deptnums num_tab;
BEGIN
        SELECT deptno BULK COLLECT INTO deptnums FROM DEPT;
        FORALL i IN 1..deptnums.COUNT INSERT INTO emp_by_dept SELECT empno, deptno FROM emp WHERE
deptno = deptnums(i);

        FOR i IN 1..deptnums.COUNT LOOP
        -- Count how many rows were inserted for each department
                dbms_output.put_line('Dept '||deptnums(i)||': inserted '|| SQL%BULK_ROWCOUNT(i)||' records');
        END LOOP;
        dbms_output.put_line('Total records inserted =' || SQL%ROWCOUNT);
END;
/
```

Counting Rows Affected by FORALL Iterations

You can also use the scalar attributes %FOUND, %NOTFOUND, and %ROWCOUNT with bulk binds. For example, %ROWCOUNT returns the total number of rows processed by all executions of the SQL statement. %FOUND and %NOTFOUND refer only to the last execution of the SQL statement. However, you can use %BULK_ROWCOUNT to infer their values for individual executions. For example, when %BULK_ROWCOUNT(i) is zero, %FOUND and %NOTFOUND are FALSE and TRUE, respectively.

Handling FORALL Exceptions with the %BULK_EXCEPTIONS Attribute

- PL/SQL provides a mechanism to handle exceptions raised during the execution of a FORALL statement. This mechanism enables a bulk-bind operation to save information about exceptions and continue processing. To have a bulk bind complete despite errors, add the keywords SAVE EXCEPTIONS to your FORALL statement as follows :

FORALL index IN lower_bound..upper_bound SAVE EXCEPTIONS {insert_stmt | update_stmt | delete_stmt}

- All exceptions raised during the execution are saved in the new cursor attribute %BULK_EXCEPTIONS, which stores a collection of records. Each record has two fields. The first field, %BULK_EXCEPTIONS(i).ERROR_INDEX, holds the "iteration" of the FORALL statement during which the exception was raised. The second field, %BULK_EXCEPTIONS(i).ERROR_CODE, holds the corresponding Oracle error code.

- The values stored by %BULK_EXCEPTIONS always refer to the most recently executed FORALL statement. The number of exceptions is saved in the count attribute of %BULK_EXCEPTIONS, that is, %BULK_EXCEPTIONS.COUNT. Its subscripts range from 1 to COUNT.

Handling FORALL Exceptions with the %BULK_EXCEPTIONS Attribute

If you omit the keywords SAVE EXCEPTIONS, execution of the FORALL statement stops when an exception is raised. In that case, SQL%BULK_EXCEPTIONS.COUNT returns 1, and SQL%BULK_EXCEPTIONS contains just one record. If no exception is raised during execution, SQL%BULK_EXCEPTIONS.COUNT returns 0.

```
declare
        TYPE NumList IS TABLE OF NUMBER;
        num_tab NumList := NumList(10,0,11,12,30,0,20,199,2,0,9,1);
        errnum NUMBER;
begin
        FORALL i IN num_tab.FIRST..num_tab.LAST SAVE EXCEPTIONS  DELETE FROM emp WHERE sal>500000/num_tab(i);
exception
        when others then
                errnum :=SQL%BULK_EXCEPTIONS.COUNT;
                dbms_output.put_line('Number of errors is ' || errnum);
                FOR i IN 1..errnum LOOP
                        dbms_output.put_line('Error ' || i || ' occurred during '|| 'execution ' ||  SQL%BULK_EXCEPTIONS(i).ERROR_INDEX);
                        dbms_output.put_line('Oracle error code is ' || SQL%BULK_EXCEPTIONS(i).ERROR_CODE);
                        dbms_output.put_line('Oracle error message is ' || sqlerrm(-(SQL%BULK_EXCEPTIONS(i).ERROR_CODE)));
                END LOOP;
END;
/
```

In this example, PL/SQL raised the predefined exception ZERO_DIVIDE when i equaled 2, 6, 10. After the bulk-bind completed, SQL%BULK_EXCEPTIONS.COUNT returned 3, and the contents of SQL%BULK_EXCEPTIONS were (2,1476), (6,1476), and (10,1476). To get the Oracle error message (which includes the code), we passed SQL%BULK_EXCEPTIONS(i).ERROR_CODE to the error-reporting function SQLERRM

Restriction on FORALL

- You can use FOR ALL clause only in tools which include PL SQL engine. Otherwise, you get the error *this feature is not supported in client-side programs*.
- The INSERT, UPDATE, or DELETE statement must reference at least one collection.

```
CREATE TABLE pairs (n NUMBER, m NUMBER);
DECLARE
        TYPE NumTab IS TABLE OF NUMBER;
        nums NumTab := NumTab(1, 2, 3);
BEGIN
        FORALL i IN nums.FIRST..nums.LAST  INSERT INTO pairs VALUES(nums(i), 10); -- works
        FORALL i IN 1..3 INSERT INTO pairs VALUES(5, 10); -- causes an error
END;
/
INSERT INTO pairs VALUES(5, 10); -- causes an error
*
ERROR at line 8:
ORA-06550: line 8, column 1:
PLS-00435: DML statement without BULK In-BIND cannot be used inside FORALL
```

Restriction on FORALL ……contd

- All collection elements in the specified range must exist. If an element is missing or was deleted, you get an error.

```
declare
        TYPE NumList IS TABLE OF NUMBER;
        depts NumList := NumList(10, 20, 30, 40);
 begin
        depts.DELETE(3); -- delete third element
        for i in depts.first..depts.last loop
                dbms_output.put_line(depts(i));
        end loop;
exception
        when others then
                dbms_output.put_line(sqlerrm);
end;
/
```

10
20
ORA-01403: no data found
the above normal loop returns the error :no_data_found

Restriction on FORALL ……contd

- All collection elements in the specified range must exist. If an element is missing or was deleted, you get an error.

```
declare
        TYPE NumList IS TABLE OF NUMBER;
        depts NumList := NumList(10, 20, 30, 40);
begin
        depts.DELETE(3); -- delete third element
        FORALL i IN depts.FIRST..depts.LAST  DELETE FROM emp WHERE deptno = depts(i); -- causes an error
exception
        when others then
                dbms_output.put_line(sqlerrm);
END;
/
```

ORA-22160: element at index [3] does not exist

Restriction on FORALL ……contd

- Collection subscripts cannot be expressions, as the following example shows or else there would be a compilation error :

```
declare
        TYPE NumList IS TABLE OF NUMBER;
        depts NumList := NumList(10, 20, 30, 40);
begin
        FORALL i IN depts.FIRST..depts.LAST  DELETE FROM emp WHERE deptno = depts(i+10);
exception
        when others then
                dbms_output.put_line('error');
                dbms_output.put_line(sqlerrm);
END;
/
```

PLS-00430: FORALL iteration variable I is not allowed in this context

Restriction on FORALL ……contd

- You cannot use the SELECT ... BULK COLLECT statement in a FORALL statement. There has to be a INSERT/UPDATE/DELETE with a FOR ALL. Otherwise, you get the error *implementation restriction: "cannot use* FORALL *and* BULK COLLECT INTO *together in* SELECT *statements*.":

```
declare
        TYPE dNumList IS TABLE OF dept.deptno%TYPE;
        dnums dNumList:= dNumList(10,20,30,40);
        TYPE eNumList IS TABLE OF emp.empno%TYPE;
        enums eNumList;
        a number;
        ctr number :=1;
begin
        forall i in dnums.first..dnums.last select empno bulk collect into enums from emp where deptno=dnums(i);
        a:=enums.count;
        While ctr<=a loop
                Dbms_output.put_line(enums(ctr));
                ctr:=ctr+1;
        End loop;
End;
/
```

Retrieving Query Results with the BULK COLLECT Clause

- The keywords BULK COLLECT tell the SQL engine to bulk-bind output collections before returning them to the PL/SQL engine. The SQL engine bulk-binds all collections referenced in the INTO list.

```
declare
        TYPE NumTab IS TABLE OF emp.empno%TYPE;
        TYPE NameTab IS TABLE OF emp.ename%TYPE;
        enums NumTab; -- no need to initialize
        names NameTab;
        a number;
        ctr number:=1;
 begin
        SELECT empno, ename BULK COLLECT INTO enums, names FROM emp;
        a:=enums.count;
        While ctr<=a loop
                Dbms_output.put_line(enums(ctr)||names(ctr));
                Ctr:=ctr+1;
        End loop;
 End;
/
```

Retrieving Query Results from Object-columns

- In the following example, SQL engine loads all the values from an object column into a nested table before returning to the PL/SQL engine.

```
CREATE TYPE Coords AS OBJECT (x NUMBER, y NUMBER);
/
CREATE TABLE grid (num NUMBER, loc Coords);
INSERT INTO grid VALUES(10, Coords(1,2));  INSERT INTO grid VALUES(20, Coords(3,4));  INSERT INTO grid VALUES(30, Coords(5,6));   INSERT INTO grid VALUES(40, Coords(7,8));

DECLARE
        TYPE CoordsTab IS TABLE OF Coords;
        pairs CoordsTab;
        a number;
        ctr number :=1;
 BEGIN
        SELECT loc BULK COLLECT INTO pairs FROM grid;
        a:=pairs.count;
        While ctr<=a loop
                Dbms_output.put_line(pairs(ctr).x||' '||pairs(ctr).y);
                Ctr:=ctr+1;
        End loop;
End;
/
```

Retrieving DML Results into a Collection with the RETURNING INTO Clause

- You can use the BULK COLLECT clause in the RETURNING INTO clause of an INSERT, UPDATE, or DELETE statement, as the following example shows.

```
DECLARE
        TYPE NumList IS TABLE OF emp.empno%TYPE;
        enums NumList;
        a number;
        ctr number :=1;
BEGIN
        DELETE FROM emp WHERE deptno = 30 RETURNING empno BULK COLLECT INTO enums;
        a:=enums.count;
        While ctr<=a loop
                Dbms_output.put_line(enums(ctr));
                ctr:=ctr+1;
        End loop;
END;
/
```

Restriction on RETURNING…..BULK COLLECT INTO

- You can use the BULK COLLECT clause only in tools which include PL SQL engine Otherwise, you get the error *this feature is not supported in client-side programs*.

- All targets in a BULK COLLECT INTO clause must be collections, as the following example shows names is a collection, but not salary, hence error.

```
DECLARE
        TYPE NameList IS TABLE OF emp.ename%TYPE;
        names NameList;
        salary emp.sal%TYPE;
BEGIN
        SELECT ename, sal BULK COLLECT INTO names, salary from emp;
END;
/
```

PLS-00497: cannot mix between single row and multi-row (BULK) in INTO list

Using FORALL and BULK COLLECT Together

- You can combine the BULK COLLECT clause with a FORALL statement, in which case, the SQL engine bulk-binds column values incrementally. In the following example, if collection dnums has 3 elements, each of which causes 2 rows to be deleted, then collection enums has 6 elements when the statement completes:

```
Declare
        TYPE dNumList IS TABLE OF dept.deptno%TYPE;
        dnums dNumList;
        TYPE eNumList IS TABLE OF emp.empno%TYPE;
        enums eNumList;
        a number;
        ctr number :=1;
begin
        select deptno bulk collect into dnums from dept where deptno<=30;
        FORALL j IN dnums.FIRST..dnums.LAST DELETE FROM emp WHERE deptno = dnums(j) RETURNING empno BULK COLLECT INTO enums;
        a:=enums.count;
        While ctr<=a loop
                Dbms_output.put_line(enums(ctr));
                ctr:=ctr+1;
        End loop;
END;
/
```

The column values returned by each execution of DELETE are added to the values of enum returned previously. This is possible with FOR ALL. With a FOR loop, the previous values would have been overwritten.