Database and Information Systems

Michal Krátý & Radim Bača

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VŠB – Technical University of Ostrava

2020/2021

Content

Packages

- Packages have a similar function as the libraries (or namespaces) in other programming languages.
- They put together PL/SQL objects (functions, procedures, variable, exceptions, and so on).
- Packages has several advantages:
  - Allows us to declare variables and exceptions which are available even outside of a procedure or a function.
  - Setting the private and public procedures and functions.
  - Setting a namespace without a conflict of names (of objects).
  - More simple orientation in PL/SQL code, good for bigger projects.

Packages

- Packages are created in two parts: *package specification* and *package body*.
- Package specification represents the public part. It contains the definition of public variables, exceptions and the headers of the procedures and functions.
- Package body then contains the private part and the procedure declarations with their body.

Packages

Package specification:

```
CREATE [OR REPLACE ] PACKAGE package_name IS|AS
   definition of variables, exceptions and
   the headers of the procedures.
END [package_name];
```

Package body:

```
CREATE [OR REPLACE ] PACKAGE BODY package_name IS|AS
   private part of the package
END [package_name];
```

Bulk operations in PL/SQL

- It is typical that applications inserts their records one-by-one. It is rather problematic when we insert a high number of records.
- Therefore RDBMS often offers so called bulk operations.
- If we insert the records using the bulk operation then we decrease log write and data structures overhead.
- The result is faster insertion of records.
- In Oracle we use the `BULK COLLECT` and `FORALL` operations.

BULK COLLECT

- `BULK COLLECT` is a bulk operation which write the data into a collection.
- Syntax is the following:
  `... BULK COLLECT INTO collection_name[, collection_name] ...`
- `BULK COLLECT` is used mainly with the `SELECT INTO` command.

FORALL

- It is not a cycle.
- FORALL is used for bulk operations which read from the collection.
- Syntax:

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

- Where:
  - `index` is array index.
  - `sql_statement` is INSERT, UPDATE or DELETE.
  - `lower_bound..upper_bound` specify the collection range which will be used.

Example

```
DECLARE
   TYPE NumTab IS TABLE OF employees.employee_id%TYPE;
   TYPE NameTab IS TABLE OF employees.last_name%TYPE;
   enums NumTab;
   names NameTab
BEGIN
   SELECT employee_id, last_name
      BULK COLLECT INTO enums, names
      FROM employees WHERE employee_id > 1000;
   ...
   FORALL i IN enums.FIRST..enums.LAST
      UPDATE myemp SET name = names(i)
                    WHERE employee=enums(i);
```

Example 1/2 - Insertion of 100 000 records

```
DECLARE
  TYPE UserArray IS VARRAY(10000) OF Usertab%ROWTYPE;
  v_userArray UserArray;
  v_counter NUMBER := 0;
  v_start NUMBER DEFAULT DBMS_UTILITY.GET_TIME;
BEGIN
  v_userArray := UserArray(); -- inicialization
  v_userArray.EXTEND(10000); -- resize

  -- you must run it 10x beacuse 100 000 items
  -- must be inserted
  FOR i IN 1 .. 10
  LOOP
    FOR j IN 1 .. 10000          -- prepare array
```

Example 2/2 - Insertion of 100 000 records

```
    LOOP
      v_counter := v_counter + 1;
      v_userArray(j).id := v_counter;
      v_userArray(j).fname := 'fname' || v_counter;
      v_userArray(j).lname := 'lname' || v_counter;
    END LOOP;

    -- bulk insert
    FORALL k IN v_userArray.FIRST..v_userArray.LAST
      INSERT INTO Usertab VALUES v_userArray(k);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE(round((
   DBMS_UTILITY.GET_TIME-v_start)/100, 2) || ' s' );
END;
```

Comparison of a bulk insert[1]

Example:

- We insert 100 00 of records into a table with a primary key.
- We compare inserting of records one-by-one with a bulk insert per 10 000 records.

	Inserting of records one-by-one	**Bulk insert**	**Difference**
Total time [s]	7	0,9	6.1

Improvement of the bulk insert is 87% ($7.7\times$)!

If you can, use the bulk operations!

[1](c) David Krch, Oracle CR: PL/SQL

Access rights of stored procedures

- A procedure implicitly runs with access rights of the procedure owner.
- This mode is called `AUTHID DEFINER`.
- Actual schema = schema of the procedure owner.
- The second possibility is that code will run with access rights of an actual user who invoked the procedure.
- That is `AUTHID CURRENT_USER` mode (from Oracle 8i).
- Actual schema = schema of an actual user.

Example

```
CREATE TABLE stud (
   id     INT primary key NOT NULL,
   lname VARCHAR2(50) NOT NULL);

CREATE OR REPLACE PROCEDURE myproc (a IN OUT VARCHAR)
AUTHID CURRENT_USER AS
BEGIN
  SELECT lname into a FROM stud WHERE id=1;
END;

-- run
DECLARE
   a VARCHAR(50);
BEGIN
   myproc(a);
END;
```

COMMIT

- Oracle supports the following types of COMMIT:
  `COMMIT [WRITE [WAIT | NOWAIT] [IMMEDIATE | BATCH]]`
- If the data being committed are important then we use WAIT:
  - `COMMIT` waits until the log write is processed.
  - $\Rightarrow$ correct update is guaranteed.

COMMIT

- Oracle supports the following types of COMMIT:
  `COMMIT [WRITE [WAIT | NOWAIT] [IMMEDIATE | BATCH]]`
- There can be a lost of data in the case of NOWAIT option:
  - COMMIT requires REDO log write but it does not wait until data are written.
  - Data are made accessible to other transactions.
  - The log write can happen with a small delay - there is a possibility of a transaction lost during a short time.
  - This can make a sense when maximal throughput is important (a number of transactions per second), but the price of one transaction is low.

SQL Injection

- Let us have a text `p_name` retrieved from a user.
- Our PL/SQL code then creates an SQL query using the code like this:

```
OPEN c FOR
   'SELECT fname, lname, salary '
     || 'FROM employees '
     || 'WHERE lname=''' || p_lname
     ||'''';
```

SQL Injection

- If a user put the following value into the `p_name`: `X' or 1=1 --`
- then we get the query:

```
SELECT fname, lname, salary
  FROM employees
  WHERE lname='X' or 1=1 --'
```

- Using the combo boxes in the web page form is not a solution since a user can write it directly into a URL.

```
http://server/page?input=X' or 1=1 --
```

How to avoid SQL Injection in PL/SQL?

- Use static queries if possible.
- Use bind variables in the dynamic SQL commands if possible.
- Try to restrict access for users according to their role in the system. You can minimize the consequences of the attack like that.

How to avoid SQL Injection in PL/SQL?

- If it is not possible to use the bind variables, do not use the value directly:
    - For example, convert the string from a user into a number.
    - Or check the string using the DBMS_ASSERT or own checking procedure.

How to avoid SQL Injection in PL/SQL?

Use the bind variables.

- *Do not use:*

```
'SELECT * FROM employees
  WHERE fname = ''' || p_fname ||
          ''' AND lname = ''' || p_lname || '''';
```

- *Use:*

```
'SELECT * FROM employees
   WHERE fname = :1 AND lname = :2';
```

You avoid the SQL injection like that.

PL/SQL / T-SQL Discussion

- Does it make a sense to write a logic in PL/SQL / T-SQL[2]?
- Absolutely yes, if you are interested in the application performance.
- PL/SQL is nearest to the database and we can significantly influence the performance using it.
- Rules:
  - What you can write in SQL, write in SQL.
  - What you cannot write in SQL, write in static PL/SQL.
  - What you cannot write in static PL/SQL, write in dynamic PL/SQL.
  - What you cannot write in PL/SQL, write in the client application.

[2]Or in another procedural extension of SQL.

References

- **Oracle Portal**: `https://docs.oracle.com/en/database/oracle/oracle-database/21/books.html`:
  - PL/SQL Language Reference
  - PL/SQL Packages and Types Reference