Relational Databases Modeling and Design, part 2 : PL/SQL

Bart GEORGE
Rachid CHELOUAH
EISTI

References

- Oracle's PL / SQL User Guide and Reference
  - http://docs.oracle.com/cd/B14117_01/appdev.101/b10807.pdf
- PL / SQL Tutorial
  - http://www.tutorialspoint.com/plsql/plsql_tutorial.pdf

What next ?

- We'll spend 21 hours together (7 * 3 hours)
- Courses + exercises, and then an evaluation
- What we'll see (not necessarily in this order)
  - Variables, constants, conditionals, loops
  - Exceptions
  - Procedures and functions
  - Cursors and parametrized cursors
  - Triggers
  - Packages
  - Autonomous transactions

PL/SQL

Lecture 1
Introduction to PL/SQL

Introduction to PL/SQL

- Plan
  - PL/SQL : Why ?
  - PL/SQL : What ?
  - PL/SQL : How ?
  - PL/SQL : Basic concepts

PL/SQL : Why ? (1/2)

- Sometimes, you need to get a result out of complex treatments
  - Link several requests between them
  - Repeat a group of instructions several times
  - Display "this" result if a condition is satisfied, or "that" result otherwise
- The problem is that SQL has many limits
  - It's non-procedural, which means
    - No loops, nor conditionals
    - No functions, nor variables
  - SQL alone becomes insufficient

PL/SQL : Why ? (2/2)

- Example : you want to display the number of tennis rackets from an inventory
  - In SQL : "SELECT quantity FROM Inventory WHERE product = 'Tennis racket' "
- Now, you want to purchase a tennis racket
  - If there are still some...
    - Table "Inventory" is updated
    - The purchase record is inserted into a table *ad hoc*
  - Otherwise, if there are none...
    - Table "Inventory" remains unchanged

PL/SQL : What ?

- The "PL" part means "Procedural Language"
- Extension of SQL that adds all the facilities of procedural programming
- Appropriate for
  - Linking several requests and data manipulations
  - Checking data's integrity (triggers)
  - Storing frequently used operations into the database (procedures, functions, packages)
  - Minimizing the impact of updates
- SQL *manipulates* data, PL/SQL *processes* data

PL / SQL : How ?

- Provides all the facilities of a procedural (and sometimes object-oriented) language
- Completely portable and OS-independent
- Syntax : similar to Ada or Pascal
- Works with
  - Oracle (SQL*Plus)
  - PostGreSQL (PGAdmin)
- Does not work with
  - MySql (PHPMyAdmin)

PL/SQL : Basic Concepts

- Block structure
- Comments
- Displaying results
- Variables declaration
- Variables assignment
- Control structures

Block structure (1/3)

- PL/SQL, like all other procedural languages, is "block-structured"
- That is :
  - A program contains several blocks
  - These blocks can contain several sub-blocks
  - Each one is a problem or subproblem to solve
- A PL/SQL block has 3 parts
  - A declarative part (variables, optional)
  - An executable part (instructions, mandatory)
  - An exception-handling part (exceptions, optional)

Block structure (2/3)

- Basic structure of a PL/SQL block

```
DECLARE
    -- declarations section
BEGIN
    -- executable command(s)
EXCEPTION
    -- exception handling
END;
```

Block structure (3/3)

- The "Hello World" example (what else ?)

```
SET SERVEROUTPUT ON
DECLARE
    message varchar2(20):= 'Hello, World!';
BEGIN
    dbms_output.put_line(message);
END;
/
```

- Don't forget the semicolon (";") nor the "/"
- Result of the above code on SQL prompt

```
Hello World
PL/SQL procedure successfully completed.
```

Comments

- Won't be taken into account by the program
- ... will be taken into account by the reader !!
- On a single line

```
-- This is a comment
```

- On multiple lines

```
/*
 * These are some comments
 * Don't hesitate to insert as many as necessary
 */
```

Displaying results (1/2)

- As we'll see later, Oracle offers
  - Predefined PL/SQL code blocks, called "packages"
  - And sub-blocks called "procedures"
- In particular, if you want to display results on SQL prompt, you can use
  - Package DBMS_OUTPUT
  - In this package, procedures put and put_line
  - As a parameter, the value you want to display
- Syntax

```
DBMS_OUTPUT.put_line(message)
```

Displaying results (2/2)

- Beware !!
  - By default, prompt is disabled
  - Which means you cannot display anything
- If you want to activate prompt, you must write SQL*Plus command set serveroutput on

Variables declaration (1/8)

- Variable (and constant)
  - Oracle identifiers
  - At most 30 characters
  - Start by a letter
  - Can contain other letters, digits, _, $ and #
  - "constant" means the initial value will never change
- Case-Insensitive
- Are declared in the DECLARE part
- Must be declared *before* being used

Variables declaration (2/8)

- General syntax

  identifier [CONSTANT] type [:= initial_value];

- Examples

  age integer;

  name varchar2(30);

  birthDate date;

  ok boolean := true;

  pi CONSTANT number := 3.141592654;

- Multiple declarations are forbidden

  i, j integer -- NO !!

Variables declaration (3/8)

- Data types
  - Scalar types
  - Reference types
  - Composite types
- Scalar types
  - Single values, without internal components
  - Numeric values : INTEGER, FLOAT, REAL...
  - Character values : CHAR, VARCHAR2, LONG...
  - BOOLEAN values
  - DATE values (01-OCT-12)

Variables declaration (4/8)

- User-defined subtypes
  - You can define your own subtype out of an existing type, and then declare variables of this subtype
  - Syntax : SUBTYPE my_subtype IS existing_type
- Example

```
DECLARE
    SUBTYPE name IS char(20);
    SUBTYPE message IS varchar2(100);
    salutation name;
    greetings message;
BEGIN
    salutation := 'Reader ';
    greetings := 'Welcome to the World of PL/SQL';
    dbms_output.put_line('Hello ' || salutation || greetings);
END;
```

Variables declaration (5/8)

- Reference types
  - %TYPE : a variable has the same type as a table column
  - %ROWTYPE : a variable contains all the columns of a table row
- Syntax
  - variable_name table_name.column_name%TYPE ;
  - variable_name table_name%ROWTYPE ;

Variables declaration (6/8)

- Composite types
  - Structures or Record types
  - Table types
  - Varray types
- Structures
  - A type containing several other (different) types
  - Once defined, you can use it as any other type
  - Syntax

```
TYPE type_name IS RECORD (
    Field1 type1,
    Field2 type 2, ...
);
```

Variables declaration (7/8)

- Structures : example

```
DECLARE
    TYPE point IS RECORD
    (
        absciss NUMBER,
        ordinate NUMBER
    );
    p point ;
BEGIN
    p.absciss := 1 ;
    p.ordinate := 3 ;
    DBMS_OUTPUT . PUT_LINE ( 'p.absciss = ' || p.absciss ||
    ' and p.ordinate = ' || p.ordinate ) ;
END;
/
```

Variables declaration (8/8)

- Table types
  - Modeled as (but not same as) database tables
  - Syntax

```
TYPE table_type_name IS TABLE OF datatype [NOT NULL]
INDEX BY BINARY_INTEGER;
```

- Array types
  - Varray : "Variable Array"
  - Syntax

```
TYPE type_name IS VARRAY size OF type_elements;
```

  - Example

```
TYPE numtab IS VARRAY 10 OF number;
numtab t;
```

Variables assignment (1/5)

- The most basic instruction is to assign values to variables
- Assignments (like any other instructions) are made into "BEGIN ... END" blocks
- Two major ways to assign values
  - Assignment operator : :=
  - Database fetching (SELECT ... INTO)

Variables assignment (2/5)

- Assignment operator
  - Syntax

    variable_name := value;
  - Examples

    x:=5;

    valid_id := FALSE;

    bonus := current_salary * 0.10;

    tax := price * tax_rate;

    dateNaissance := ’10/10/2004’;

Variables assignment (3/5)

- Using assignment operator with array types
  - A varray is initialized empty

    Syntax : varray_name := varray_type();

    Example : t := numtab();
  - Once initialized, you can access to each entry

    t(10) := 5; -- the 10th entry of t takes the value 5

    k := T(5) -- k takes the value of T's 5th entry
  - You can also modify the size of a varray

    Syntax : varray_name.EXTEND(new_size);

    Example : t.EXTEND(10);

Variables assignment (4/5)

- Fetching database values
  - Syntax

```
SELECT field_1, field_2, ... , field_n
INTO variable_1, variable_2, ... , variable_n
FROM ...
```

  - Assigns to variable_1, variable_2, ... , variable_n the values returned by the request and store into field_1, field_2, ... , field_n

Variables assignment (5/5)

- Fetching database values : example

```
DECLARE
    num NUMBER
BEGIN
    SELECT numprod INTO num
    FROM Product
    WHERE nameprod = 'Barbie doll';
    DBMS_OUTPUT.put_line('The number corresponding to Barbie
    doll article is ' || num);
END;
/
```

Control structures (1/6)

- Conditionals
  - IF ... THEN ... ELSE
  - CASE ... WHEN
- Loops
  - FOR
  - WHILE

Control structures (2/6)

- IF conditional (first form)

```
IF boolean_condition
THEN instructions1
ELSE instructions 2
END IF;
```

- IF conditional (second form)

```
IF boolean_condition1
THEN instructions1
ELSIF boolean_condition2
THEN instructions2
ELSE instructions3
END IF;
```

Control structures (3/6)

- CASE conditional

```
CASE variable_name
WHEN value_1 THEN instructions_1
WHEN value_2 THEN instructions_2
...
WHEN value_n THEN instructions_n
ELSE default_instructions
END CASE;
```

- Beware ! A set of several instructions must be put into a "BEGIN ... END;" sub-block
  - IF condition THEN BEGIN instr1; instr2; END; ELSE ...
  - CASE variable WHEN value1 THEN BEGIN ... END; ...

Control structures (4/6)

- Loops : used to repeat a set of instructions several times

```
LOOP
    instructions
END LOOP;
```

- EXIT command : allows to leave a loop anytime

```
LOOP
    instructions
    EXIT WHEN boolean_condition
END LOOP;
```

- WHILE loop : repeat while a condition is OK

```
WHILE boolean_condition LOOP
    instructions
END LOOP;
```

Control structures (5/6)

- FOR loop : repeat a certain amount of times

```
FOR count IN lowest_value .. highest_value LOOP
    Instructions
END LOOP;
```

- You can reverse the order of a FOR loop

```
FOR count IN REVERSE lowest_value .. highest_value LOOP
    Instructions
END LOOP;
```

Control structures (6/6)

- Loops : examples

```
LOOP
    total := total + salary;
    EXIT WHEN total > 25000;
END LOOP;

FOR num IN 1..500 LOOP
    DBMS_OUTPUT.put_line(num);
END LOOP;

WHILE salary <= 2500 LOOP
    salary := salary +1;
END LOOP;
```