Oracle PL/SQL Basic

Language & Standards

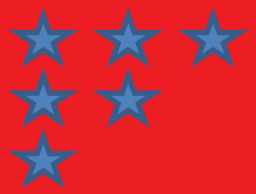
lingaro

The Value of Business Intelligence

Q & A

Course contains Lingaro PL/SQL Standard rules

- Rule is indicated by blue stars
- More stars means more important rule



Training Agenda

- PL/SQL Language Overview
- Code Description Standards
- Declarations
- Statements
- Expressions
- Predefined Exceptions Handling

Topic Agenda

PL/SQL Language Overview

- Language Properties
- Anonymous Block
- Named Subprogram Block
  - Procedure
  - Function
- Glossary of Terms
- Syntax
  - Engine Accepted
  - Lingaro Standards

Language Properties

- Oracle native closed solution
- Syntax similar to SQL
- Integrated with SQL
  - Execute SQL from PL/SQL code
  - Build SQL functions in PL/SQL
- More flexible then SQL
- Procedural - divided to subprograms, uses variables
- Modularized to units
  - anonymous blocks, subprograms, packages, triggers
- Used in data access and business logic layers
  - Can by used for WEB interface GUI layer
- Used in PL/SQL engine in
  - Oracle database
  - Old Oracle tools (Form, Reports, PLSQL Builder)

Anonymous Block

Structure

- PL/SQL code consist of PL/SQL blocks (named or anonymous)
- Anonymous block structure

```
DECLARE ← optional
  -- optional declarations section - can contain:
  -- types, subtypes, variables,
  -- exceptions, cursors, local subprograms
BEGIN
  -- statements of executable section (keep number of lines < 60 in all blocks)
EXCEPTION ← optional
  -- optional exception handling section
END;
```

- Simplest example – do nothing – SQL> EXECUTE NULL;

```
BEGIN
  NULL;
END;
```

Anonymous Block

Structure

- Example

```
DECLARE
  l_date DATE;                          <-- local variable
BEGIN
  -- get date from table                <-- comment
  SELECT start_date
    INTO l_date                         <-- SQL statement
    FROM prcss_date_prc
   WHERE prcss_code = 'ETL25';
  -- assign value to global variable
  g_date := l_date + 7;                 <-- PL/SQL statement
EXCEPTION
  -- handle exception
  WHEN NO_DATA_FOUND THEN
       g_date := SYSDATE + 7;
END;
```

Anonymous Block

Nesting and Declaration Scope

- You can put block inside block as statement
- Number of nesting levels is not limited
- Declarations are visible from levels down to up
- Use labels to reference same name declarations located on higher level

```
<<b_higher>>  ← label
DECLARE
  l_date DATE;
BEGIN
  ...
  <<b_lower>>  ← label
  DECLARE
    l_date DATE;
  BEGIN
    l_date := SYSDATE;
    b_higher.l_date := SYSDATE + 7;
  END;
  ...
END;
```

Anonymous Block

Purpose

- Statement wrapping one or more statements
  - Grouping statements
  - Can be send to and executed on database from any client
  - Many SQL statements with additional logic can be send to database in one network trip
  - Can't be created as standalone object in database

    Only can be nested in subprograms (procedures, functions) - named blocks even created in database
- Local declarations
  - Visible only by block wrapped statements
  - Not visible outside this block
- Local error handling
  - Local captured exceptions not visible outside this block

Named Subprogram Block - Procedure, Function Parameters

- Example

```
PROCEDURE pro_days_between (in_start_date    IN     DATE,
                            in_end_date      IN     DATE := NULL,
                            inout_gpa        IN OUT NUMBER,
                            out_days_between    OUT NUMBER)
```

- Declaration
  - Name (identifier) with mode prefix **in_ out_ inout_**
  - Mode
    - IN - to transfer value into block
      default mode but use explicit IN keyword
    - OUT - to transfer values from block
    - IN OUT - to transfer value into and from block
  - Data type (same as in variables)
  - For optional parameters assign default value
    - using := operator or DEFAULT keyword

Named Subprogram Block

Procedure

- Overview
  - Do actions – contains statements
  - Used as PL/SQL statement
  - Can have parameters
  - Can return one or many values using parameters
  - Name prefix **pro_** 
- Block structure

```
PROCEDURE <procedure name> [( <parameter list> )]   <-- header
IS | AS
  -- optional declarations section
BEGIN                                               <-- body
  -- statements of executable section
EXCEPTION
  -- optional exception handling section
END;
```

Named Subprogram Block

Procedure

- Example

```
PROCEDURE pro_new_cust(in_cust_name IN VARCHAR2)
  ...
IS
  l_cust_skid NUMBER;
BEGIN
  l_cust_skid := cust_skid_seq.NEXTVAL;
  INSERT INTO cust_dim (cust_skid, cust_name)
    VALUES (l_cust_skid, in_cust_name);
  INSERT INTO cust_hist_prc (cust_skid, event_name, time_stamp)
    VALUES (l_cust_skid, 'Created', SYSDATE);
  COMMIT;
END;
```

Named Subprogram Block

Locally Declared

- Placed in anonymous or named block declarations section
- Can be executed only in this block or its nested blocks

```
DECLARE
  ...
  PROCEDURE pro_send_order
    ...
  END;
BEGIN
  ...
  pro_send_order;
  ...
  BEGIN
    ...
    pro_send_order;
    ...
  END;
 ...
END;
```

Named Subprogram Block

Create in Database Schema

- Use CREATE DDL statement

```
CREATE OR REPLACE PROCEDURE pro_new_cust(in_cust_name IN VARCHAR2)
  ...
END;
/
```

- Execute like statement in block executable section

```
  ...
BEGIN
  ...
  pro_new_cust(in_cust_name => 'Walmart');
  ...
END;
```

Named Subprogram Block

Function

- Overview
  - Compute value similar to expressions
  - Used in PL/SQL expressions
  - Used in SQL statements (with restrictions)
  - Can have parameters
  - Must return value
  - Name prefix **fn_** 
- Block structure

```
FUNCTION <function name> [( <parameter list> )]
  RETURN <data type>
IS | AS
  -- optional declarations section
BEGIN
  -- statements of executable section
  RETURN <expression>
EXCEPTION
  -- optional exception handling section
END;
```

Named Subprogram Block

Function Example

```
FUNCTION fn_work_day_ind(in_date IN DATE)
  RETURN BOOLEAN
IS
  l_dummy VARCHAR2(1);
BEGIN
  SELECT 'X'
    INTO l_dummy
    FROM no_work_day_prc
    WHERE in_date BETWEEN start_date AND end_date;
  RETURN FALSE;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
       RETURN (TO_CHAR(in_date, 'DAY') NOT IN ('SAT', 'SUN'));
  WHEN TOO_MANY_ROWS THEN
       RETURN FALSE;
END;
```

Named Subprogram Block

Function Usage

- Like procedures can be
  - Declared locally
  - Created in database schema
- Can be used in SQL if
  - Created in database schema (use PRAGMA UDF in 12c for better performance)
  - Has parameters only in IN mode
  - Parameters and returned data types are SQL data types
    otherwise use wrapper or Oracle version 12c
  - Not terminate transactions
  - Not execute DML statements
  - If executed queries don't read table modified by DML where function is used

```
CREATE OR REPLACE FUNCTION fn_max_date(in_cust_skid IN NUMBER)
  ...
END;

SELECT cust_name,
       fn_max_date(cust_skid) AS cust_max_date
  FROM cust_dim;
```

Glossary of Terms

- Keyword
  - Word reserved by SQL or PL/SQL engine
  - Keyword examples:

    PL/SQL Block: DECLARE, PROCEDURE, FUNCTION, BEGIN, END, EXCEPTION, …
    PL/SQL Operator: BETWEEN, IN, AND, OR, NOT …
    SQL keywords: SELECT, INSERT, WHERE, INTO, …
    PL/SQL Statements: EXECUTE, RAISE, RETURN, …

- Identifier
  - Name assigned to element defined in block declaration section
  - Must contain identifier type prefix to avoid name conflict with:

    Database object names
    SQL and PL/SQL engine keywords
    PL/SQL engine build in:
    procedures, functions

Syntax

PL/SQL compiler accepted syntax

- Statements and declarations:
  - written in one or many lines terminated by semicolon
  - number and kind of white spaces are not important
  - keywords, identifiers and object names are case unsensitive :
  - Single line comments starts from --
  - Multiline comments enclosed by /* and */
- Without code standards code is:
  - not: readable, extensible, reusable, manageable
  - make problems with maintenance and support
  - lack of standards can degrade performance or security

Syntax

Lingaro Standards ★★★

- Keywords in uppercase
- Object names and identifiers in lowercase
- Identifier
  - Could be composed of one or more words
  - Separate words by underscores
  - Use only lowercase letters
  - Use abbreviation standards from PDM
  - Use prefix(/suffix) to indicate identifier type
- Single space to separate
  - identifiers, operators and literals
- No whitespace
  - after(and before)
  - between two or more brackets (((… or)))…
  - Before semicolon

```
  l_pos_amt NUMBER;
BEGIN
  SELECT cust_name || ' ' ||
         cntry_code AS cust_cntry_code,
    FROM cust_dim
   WHERE used_ind = 'Y';

  l_amt := 100 - ((l_pos_amt - 5) / 2);
```

Syntax

Lingaro Standards ★★★

- Indenting
  - 2-space indent increment per nesting level
  - Start with the same indent if same nesting level
  - Do not use automatic format in editors or tools

```
BEGIN
  LOOP
    ...
    IF (l_date > SYSDATE) THEN
      l_date := SYSDATE;
    END IF;
    EXIT WHEN (l_step_cnt > l_step_max_cnt);
  END LOOP;
END;
```

- Multiline declaration, statement, expression
  - Place each element from list in separate line
  - Place each clause in separate line
  - Break line if no fit into 80 characters if make sense

Topic Agenda

Code Description Standards 

- Description Header Comments
- Inside Code Comments

Description Header Comments

Lingaro Standards ★★★

- Use only English and DD-MON-YYYY date format in
  - Headers and comments and all code if possible
- Place header into PL/SQL units – example for procedure

```
PROCEDURE pro_do_work(in_parm1    IN NUMBER,
                      in_parm2    IN VARCHAR)
--------------------------------------------------------------------------------
-- Procedure    : pro_do_something_useful                                      --
-- Author       : Jan Kowalski(jan.kowalski@lingaro.com)                       --
-- Date         : 01-JAN-2014                                                  --
-- Purpose      : Do useful work                                               --
-- Version      : 1.0.0                                                        --
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- Change History                                                             --
-- Date        Programmer        Description                                  --
-- ----------- ----------------- -------------------------------------------- --
-- 01-JAN-2014 Jan Kowalski      Initial Version                              --
-- 23-MAY-2014 Piotr Nowak       Version 1.0.1 - Corrected bug in SELECT stmt --
--------------------------------------------------------------------------------
...
```

Description Header Comments

Lingaro Standards ★★★

- Describe parameters and exit values if used

```
...
-------------------------------------------------------------------------------
-- Parameters                                                                 --
-------------------------------------------------------------------------------
-- No  Name                   Description / Example                   Default  --
-- --  ---------------------- --------------------------------------- -------- --
--  1  in_sfct_tbl_name       Name of the sfct table                           --
--  2  in_fct_tbl_name        Name of the fct table                            --
--                                                                             --
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Exit Codes   : Code  Description                                           --
--                -----  ------------------------------------------------- --
--               -20001 Input in_sfct_tbl_name is NULL and can't be NULL     --
--               -20002 Input in_sfct_tbl_name does not exist                --
--               -20030 Other Oracle error                                   --
-------------------------------------------------------------------------------
IS
...
```

Inside Code Comments

Lingaro Standards ★★★

- Separate each logical step in block executable section with comments
- Use STEP010 for parameters validation and increment next step by 10

```
-----------------------------------------------------------------------------
-- STEP010 Validate input parameters                                       --
-----------------------------------------------------------------------------
-- If any of these parameters are NULL then processing will fail
IF (in_tbl_name IS NULL) THEN
  RAISE_APPLICATION_ERROR(-20001, 'in_tbl_name is NULL and can''t be NULL');
END IF;
-----------------------------------------------------------------------------
-- STEP020 Task 2 info                                                     --
-----------------------------------------------------------------------------
...
-----------------------------------------------------------------------------
-- STEP030 Task 3 info                                                     --
-----------------------------------------------------------------------------
...
```

Inside Code Comments

Lingaro Standards ★★★

- Every code change should by described
  - in header
  - In comment one line before modified line
- Place single line comments before statements not inline
- In multiline comments
  - place closing symbols */ on its own line
  - place first words in all lines at the same position

```
-------------------------------------------------------------------------------
-- Change History                                                            --
-- Date        Programmer        Description                                 --
-- ----------- ----------------- ------------------------------------------- --
-- 01-JAN-2014 Jan Kowalski      Initial Version                             --
-- 23-MAY-2014 Piotr Nowak       Version 1.0.1 - Corrected bug in SELECT stmt --
-------------------------------------------------------------------------------
...
-- Version 1.0.1 - Corrected bug in SELECT stmt
SELECT cust_code || ' ' ||
       cntry_code AS cust_cntry_code,
  FROM cust_dim
 WHERE used_ind = 'Y';

-- This is single line comment
/* This is
   Multi-line comment
*/
```

Topic Agenda

Declarations

- Variable
  - %TYPE Attribute
  - Constant & Hardcoding
- Data types
  - PL/SQL Build in Types and Subtypes
  - User Defined Types and Subtypes

Variables

- Name prefix l_ 
- Value can be assigned in
  - Declaration (start - default value using := operator or DEFAULT keyword)
  - By := operator statement in executable block section
- If not assigned on declaration then start value is NULL
- If NOT NULL constraint used than must be assigned in declaration
- Before 12c maximum type size can be different than in SQL
  - e.g. VARCHAR2: PL/SQL - 32,767 bytes, SQL - 4,000 bytes
- Syntax

```
<indentifier> <type> [NOT NULL] [:= <expression>];
```

- Examples

```
IS
  l_sql           VARCHAR2(32767);
  l_date          DATE := SYSDATE;
  l_sfct_tbl_name VARCHAR2(30) NOT NULL := UPPER(in_sfct_tbl_name);
BEGIN
  l_sql := 'SELECT 1 FROM DUAL';
```

Variables

%TYPE attribute

- Used to define variable or parameter with the same type as
  - table column
  - another variable
- Variable type is changed automatically when base type is changed
- Use it whenever applicable 
- Very useful when variable or parameter corresponds to column
  - Use same variable type as corresponding column avoid implicit conversion
  - After column size change variable size is changed automatically without code change (fit longer data)

- Syntax

```
<variable2> <variable1>%TYPE;
<variable3> <tablename.columnname>%TYPE;
```

- Examples

```
IS
  l_prcss_id         prcss_tbl_prc.prcss_id%TYPE;
BEGIN
```

Variables

Constants & Hardcoding

- Constant
  - Is like read only variable
  - Has CONSTANT keyword in definition
  - Name prefix **const_** 
  - Assigning value in definition is mandatory
  - Is kind of hardcoding values in code
- Constant example

```
DECLARE
  const_km_per_mile CONSTANT NUMBER := 1.609344;
```

- Hardcoding
  - Literals used in SQL and PL/SQL statements are also hardcoded
  - Use hardcoding only if you are 100% sure that value should be always the same
  - Hardcoding is generally not recommended 
  - To avoid hardcoding - change literal to parameter - put value in file, table or environment variable

```
DECLARE
  const_schema_name CONSTANT VARCHAR(30) := fn_get_parm('SCHEMA');
  l_role_name                VARCHAR(30) := fn_get_parm('ROLE');
```

Data Types

Build in Types

- Used do define user types, subtypes, variables, parameters
- Build in scalar types
  - All scalar SQL data types like

    `VARCHAR2, NUMBER, BINARY_INTEGER, DATE, TIMESTAMP, INTERVAL, ROWID, ...`
  - PL/SQL only scalar data types

    `BOOLEAN, PLS_INTEGER, SIMPLE_INTEGER`
- Build in scalar subtypes
  - BINARY_INTEGER PL/SQL subtypes

    `NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE`
  - NUMBER PL/SQL subtypes

    `DECDECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NUMERIC, REAL, SMALLINT`
- None scalar SQL types

  `REF CURSOR, REF OBJECT`
- Composite SQL types

  `TABLE, VARRAY, OBJECT`
- Composite PL/SQL only types

  `RECORD, TABLE … INDEX OF`

Data Types

User Defined

- Name prefix **t_** 
- Declared in PL/SQL block
  - Scalar Subtypes (no scalar declared types)
  - REF CURSOR (described in intermediate training)
  - REF OBJECT (described in advanced training)
  - RECORD type (described in intermediate training)
  - Associated arrays (described in intermediate training)

```
DECLARE
  SUBTYPE t_sub_vc2_nn_100 IS VARCHAR2(100) NOT NULL;
  l_cntry_name t_sub_vc2_nn_100;
```

- Created in database schema
  - TABLE (described in intermediate training)
  - VARRAY (described in intermediate training)
  - OBJECT (described in intermediate training)

```
CREATE OR REPLACE TYPE t_name_tbl
  IS TABLE OF VARCHAR2(100);

DECLARE
  l_variable t_name_tbl;
...
```

Topic Agenda

Statements Execution

- Assign Value to Variable
- PLSQL Subprogram Execution
- Actual Parameters Specification
- SQL Statement Execution
- Loops
- Conditional Statements

Assign Value to Variable

- Syntax

```
<variable> := <expression>;
```

- Place := operators in the same column ★★
  - if many assign statements included
  - to make more readable code
- Example

```
BEGIN
  l_cust_name := 'Walmart';
  l_prod_skid := in_prod_skid;
  l_date      := NEXT_DAY(SYSDATE, 'SUNDAY') - 7 + in_day_of_week;
  ...
```

PLSQL Subprogram Execution

- Declared in block or created in database schema

```
BEGIN
  ...
  pro_load_tbl(in_tbl_name => 'cust_dim');
  ...
```

- Created inside schema package

```
BEGIN
  ...
  pkg_etl_utils.pro_load_tbl(in_tbl_name => 'prod_dim');
  ...
```

- Placed in Oracle provided package
  - Use lowercase in Oracle provided packages reference 
  - Send diagnostic information using dbms_output.put_line 

```
BEGIN
  ...
  dbms_output.put_line('prod_dim');
  ...
```

Actual Parameters Specification

- Formal parameters example

```
pro_delet_old_log(in_log_name   IN VARCHAR2 := 'ETL_PLC',
                  in_rtnsn_days IN NUMBER   := 30,
                  in_arch_ind   IN BOOLEAN  := FALSE)
```

- By parameter position

```
pro_delet_old_log('ETL_LOG_PLC', 90, TRUE);
```

- By parameter name - recommended 

```
pro_delet_old_log(in_rtnsn_days => 90, in_arch_ind => TRUE);
```

- Mixed method

```
pro_delet_old_log('ETL_LOG_PLC', in_arch_ind => TRUE);
```

SQL Statement Execution

- DML and TCL
  - Simple example

```
BEGIN
  ...
  UPDATE appl_parm_prc
     SET parm_val  = '5'
   WHERE parm_name = 'load_retry_max';
  COMMIT;
END
```

  - Using PL/SQL variables

```
BEGIN
  UPDATE appl_parm_prc
     SET parm_val  = l_parm_val
   WHERE parm_name = l_parm_name;
  ...
  ROLLBACK;
  ...
END
```

  - Specify column list after INSERT INTO table 

```
INSERT INTO appl_parm_plc (parm_name, parm_value)
  VALUES('inst', '2');
```

SQL Statement Execution

- Query
  - Additional INTO clause need to store results into variables
  - Predicate must get exactly one row
  - If no rows found then NO_DATA_FOUND exception is raised
  - If more then one row found then TOO_MANY_ROWS exception is raised
  - Example

```
BEGIN
  SELECT parm_val, parm_class
    INTO l_parm_val, l_parm_class
    FROM appl_parm_prc
   WHERE parm_name = l_parm_name;
END
```

SQL Statement Execution

- DDL and DCL
  - Can't be placed directly into PL/SQL block
  - Dynamic SQL execution wrapper needed - EXECUTE IMMEDIATE
  - Example

```
BEGIN
  EXECUTE IMMEDIATE 'ALTER SESSION SET ENABLE PARALLEL DML';
  l_sql := 'GRANT SELECT ON appl_parm_prc to ' || l_user_name;
  EXECUTE IMMEDIATE l_sql;
END
```

Loops

Loop Types

- Basic - conditional exit explicitly declared

```
l_next_task_name := 'First Task';
LOOP
  l_next_task_name := fn_run_task(in_task_name => l_next_task_name);
  EXIT WHEN (l_next_task_name IS NULL);
END LOOP;
```

- WHILE - conditional exit always only before loop step

```
WHILE (l_next_task_name IS NOT NULL) LOOP
  l_next_task_name := fn_run_task(in_task_name => l_next_task_name);
END LOOP;
```

- FOR - steps controlled by integer index variable values range

```
FOR l_num IN 1..5 LOOP
  pro_run_task(in_task_name => 'Task' || TO_CHAR(l_num));
END LOOP;
```

Loops

Loop Capabilities

- EXIT statement can be used to exit from loop
  - From any loop point
  - Is mandatory in basic loop
  - Use EXIT only in last basic loop included statements if possible ★ ★
  - Use EXIT only with its own condition ★ ★ ★
  - In nested loops EXIT is don only from current loop
  - Label can be used to exit to higher level

```
<<external>>
FOR l_step_num IN 1..l_step_max LOOP
  FOR l_sub_step_num IN 1..l_sub_step_max LOOP
    pro_run_task(in_task_name => 'Task' || TO_CHAR(l_num),
                 out_end_ind  => l_end_ind);
    l_end_loop := (...);
    EXIT external WHEN l_end_loop;
  END LOOP;
END LOOP;
```

Loops

Loop Capabilities

- Use CONTINUE statement to bypass current loop step

```
CONTINUE [<label>] [WHEN (<condition>)];
```

- Be careful. Infinite loop = hanged session !
  - IN basic loop EXIT messing or with always FALSE or NULL condition
  - WHILE with always TRUE condition
- FOR loop is save and easy to use
  - Number of steps is determined on loop start
  - Number of steps can't be changed inside loop
  - Index variable is automatically declared and incremented
  - Index range can be set using variables

```
FOR I in l_start_num..l_end_num LOOP
```

Loops

Choosing Loop Type

- Choose FOR statement always when ★
  - Number of steps is determined before loop
    - Statements inside loop do not modify exit condition variables
  - One step is needed for each value from known values set
- Choose WHILE when
  - Exit condition is checked before loop step
- Choose basic loop when
  - Exit condition is checked after loop step
  - There are many exit conditions in different places inside loop

Conditional Statements

IF

- Example

```
BEGIN
   ...
   IF (l_sales > 50000) THEN
      l_bonus := 1500;
   ELSIF (l_sales > 35000) THEN
      l_bonus := 800;
   ELSIF (l_sales > 12000) THEN
      l_bonus := 300;
   ELSE
      l_bonus := 100;
   END IF;
   ...
END;
```

Conditional Statements

CASE

- Without selector – similar example

```
BEGIN
  ...
  CASE
    WHEN (l_sales > 50000) THEN
      l_bonus := 1500;
    WHEN (l_sales > 35000) THEN
     l_bonus := 800;
    WHEN (l_sales > 12000) THEN
     l_bonus := 300;
    ELSE
     l_bonus := 100;
  END CASE;
  ...
END;
```

Conditional Statements

CASE

- With selector

```
BEGIN
  ...
  CASE l_bonus_code
    WHEN 'HIGH' THEN
      l_bonus_val := 1500;
    WHEN 'MEDIUM' THEN
      l_bonus_val := 800;
    WHEN 'LOW' THEN
      l_bonus_val := 300;
    ELSE
      l_bonus := 0;
  END CASE;
  ...
END;
```

Topic Agenda

Expressions

- Conditional Expression – CASE
- Expression Overview
- String Expressions
- Numeric Expressions
- Date and Time Expressions
- Logical Expressions
- Polymorphous Functions in Expressions

Conditional Expression

- CASE without selector

```
BEGIN
   ...
   l_bonus := CASE
                WHEN (l_sales > 50000) THEN 1500
                WHEN (l_sales > 35000) THEN 800
                WHEN (l_sales > 12000) THEN 300
                ELSE 100
              END + l_extra_bonus;
END;
```

Conditional Expression

- CASE with selector

```
BEGIN
  ...
  l_bonus_val := CASE l_bonus_code
                   WHEN 'HIGH'   THEN 1500
                   WHEN 'MEDIUM' THEN 800
                   WHEN 'LOW'    THEN 300
                   ELSE 0
                 END + l_extra_bonus;
  ...
END;
```

- First accepted “WHEN” returns value
- If no accepted “WHEN” and no “ELSE” returns NULL

Expression Overview

- Used to create not ready value
- Can contain
  - Literals e.g. ‘text’ 935 ‘10-MAY-2014’ TRUE
  - Variables e.g. l_start_date l_cust_name l_step_cnt
  - Operators e.g. + - / * || AND IN BETWEEN
  - Functions provided by Oracle
    Included in “STANDARD” package – are the same as SQL functions
    Grouping functions like SUM MIN MAX AVG COUNT are not included
  - Functions declared in PL/SQL block
  - Functions created in database schema
  - Functions included in package created in database
  - Whitespaces
- Use general Lingaro PL/SQL Standard rules ★★★
  - like in statements and declarations
- Can use expression instead of literal

String Expressions

- Operators

  `||` - concatenation

- "STANDARD" package functions

  `UPPER, LOWER, INITCAP, SUBSTR, INSTR, RPAD, LPAD, TRIM, LENGTH, TO_DATE, TO_CHAR, REPLACE, ASCII, CHR, REGEXP_SUBSTR ...`

- Example

```
l_address := INITCAP(l_first_name) || ' ' || INITCAP(l_last_name) || CHR(10)
             SUBSTR(l_location,24) || ' ' || l_nr || CHR(10)
             l_zip || ' ' || UPPER(l_city);
```

Date and Time Expressions

- Operators

  <date> + *<number>* *<date>* - *<number>* *<date>* - *<date>*

- "STANDARD" package functions

  SYSDATE, SYSTIMESTAMP, CURRENT_DATE, CURRENT_TIMESTAMP,
  LAST_DAY, MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, ROUND, TRUNC,
  TO_DATE, TO_TIMESTAMP, TO_TIMESTAMP_TZ, TO_YMINTERVAL,
  LOCALTIMESTAMP, SESSIONTIMEZONE, DBTIMEZONE, FROM_TZ, TZ_OFFSET

- Example

```
DECLARE
  l_end_date DATE   := ADD_MONTHS(in_start_date + 7, 12);
  l_day_cnt  NUMBER := SYSDATE - l_end_date;
  ...
```

Numeric Expressions

- Operators

```
+  -  *  /     ( )     ** - power of
```

- "STANDARD" package functions

```
EXTRACT, TO_NUMBER, ROUND, TRUNC, ABS, EXP, CEIL, FLOOR
GREATEST, LEAST, LN, LOG, MOD, SQRT, SIN, COS, TAN, ...
```

- Example

```
pro_save_bonus(in_person_skid => l_person_skid,
               in_bonus => ROUND((l_sale * 0.2 + l_cmmsn)/2, 2));
```

Boolean Expressions

- Operators

```
=       <>  ~=  !=      >   <   >=   <=
LIKE    BETWEEN     IN      IS NULL
NOT     AND     OR
```

- "STANDARD" package functions

```
REGEXP_LIKE, EXISTS
```

- Put logical expressions into brackets ★★★
- Example

```
IF ((l_pos_sales > 100000) AND (l_date - SYSDETE < 5)) THEN
...
l_exit_ind := (l_step_cnt >= l_step_max);
...
l_state_ind := REGEXP_LIKE(l_addres, ' TX | MX | CA ');
```

Polymorphous Functions in Expressions

- "STANDARD" package functions

  `NVL, NVL2, COALESCE, DECODE, NULLIF, ...`

- Example

```
l_grade := DECODE(l_grade_code,
                  1, 'Low',
                  2, 'High',
                  'Unknown');

l_address := COALESCE(l_address1, l_address2, l_address3);
```

Topic Agenda

Exceptions

- Predefined Exceptions
- OTHERS exceptions
- RAISE_APPLICATION_ERROR procedure

Predefined Exceptions

- Names

```
DUP_VAL_ON_INDEX,
INVALID_CURSOR,
NO_DATA_FOUND
TOO_MANY_ROWS,
ZERO_DIVIDE
```

full list from documentation

- Are assigned to server error code
- When assigned error happen – exception is raised
- Next statements in block section are not executed
- If exception is not handled then propagate outside block
- Indent statements below WHEN using 5 characters 
- Handling Example

```
EXCEPTION
  WHEN NO_DATA_FOUND OR TO_MANY_ROWS THEN
       pro_error_message('Query must return one row');
  WHEN ZERO_DIVIDE THEN
       pro_error_message('division by zero');
END;
```

OTHERS Keyword

- You can handle all errors by using OTHERS keyword
- RAISE statement must be used to propagate other errors ★★★
- Do not hide other errors using NULL statement ★★★
- Rollback transaction ★★★
- Use SQLERRM and FORMAT_ERROR_BACKTRACE functions ★★★
  - To get standard error message for this unknown error

- Example

```
EXCEPTION
  WHEN NO_DATA_FOUND OR TO_MANY_ROWS THEN
       pro_errmsg('Query must return one row');
  WHEN OTHERS THEN
       l_sql_errmsg := SQLERRM;
       l_err_stack  := dbms_utility.format_error_backtrace;
       ROLLBACK;
       pro_errmsg(l_sql_errmsg || l_err_stack);
       RAISE;
END;
```

RAISE_APPLICATION_ERROR Procedure

- This procedure is used to generate custom application level server error
- For this kind of errors server reserved codes between -20000 and -20999
- Syntax

```
RAISE_APPLICATION_ERROR( -<numeric code>, 'error message');
```

- Can be used in exception handling instead of RAISE statement

```
EXCEPTION
 ...
  WHEN OTHERS THEN
       RAISE_APPLICATION_ERROR(-20154, 'Process ' ||
         l_prcss_name || ' has unexpected error: ' ||
         SQLERRM || dbms_utility.format_error_backtrace);
END;
```

- Can be used in block executable section on application error

```
IF ((l_month_num < 1) OR (l_month_num >12)) THEN
  RAISE_APPLICATION_ERROR(-20154, 'Wrong month number');
END IF;
```

Q & A

PL/SQL Resources

- Oracle Database Documentation Library

  http://docs.oracle.com/cd/E11882_01/index.htm

  - PL/SQL Language Reference

    http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm

  - PL/SQL Packages and Types Reference

    http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/toc.htm

- OTN

  http://www.oracle.com/technetwork/database/application-development/index-101230.html

- O'Reilly

  - **Oracle PL/SQL Programming**

    ISBN:978-0-596-51446-48