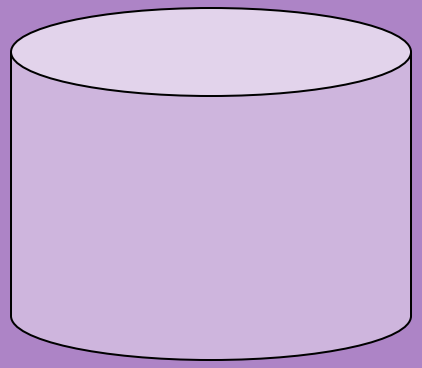
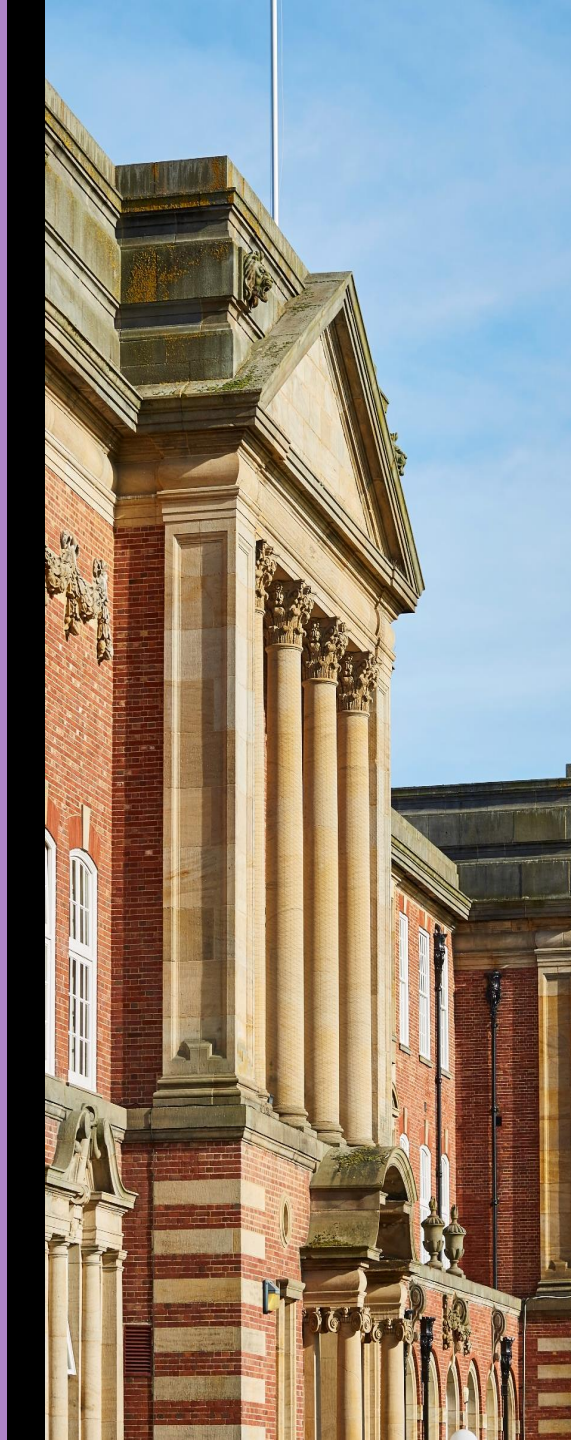


LEEDS BECKETT UNIVERSITY

School of Built Environment, Engineering and Computing

PL/SQL: Procedural Language and SQL

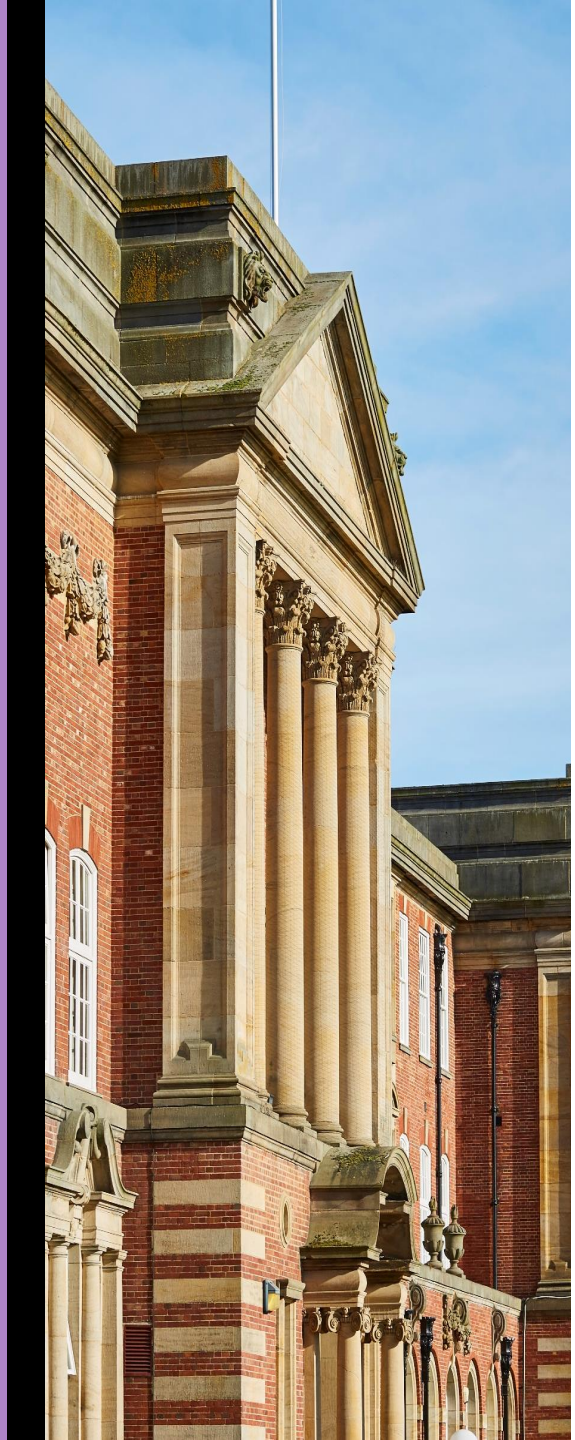
Sanela Lazarevski (Module Leader)



This sessions will cover

- PL/SQL introduction
- Procedures and Functions
- Triggers, types, when to use one, how to implement

LEEDS BECKETT UNIVERSITY



PL/SQL Introduction

- Why we need PL/SQL
  - Background
  - Structure
  - Syntax and constructs
- Server side triggers
- User defined exceptions
- How used
  - Packages, procedures and functions
  - Loops

LEEDS BECKETT UNIVERSITY

PL/SQL background

- **Databases** Module
  - used Oracle's nonprocedural language SQL
- **SQL (DDL and DML)**
  - **powerful**
  - **developers tool**
  - **command line driven**
- **SQL** is a great query language, but it has its limitations
  - **not user friendly, limited control**

LEEDS BECKETT UNIVERSITY

PL/SQL Introduction

- Combines **SQL** non procedural data manipulation commands with **P**rocedural **L**anguage constructs
- **PL** lets you use constants and variables, alter program flow and trap runtime errors
- Provides procedural language constructs.
- Similar to programming languages e.g. C

```
DECLARE
  Emp_number   INTEGER := 9999;
  Emp_name     VARCHAR2(10);
BEGIN
  SELECT Ename INTO Emp_name
   FROM Emp
   WHERE Empno = Emp_number;
  DBMS_OUTPUT.PUT_LINE('Employee
name is ' || Emp_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No such
employee: ' || Emp_number);
END;
```

PL/SQL background

- PL/SQL
  - embedding of SQL statements
  - and data manipulation in its sub programs
  - SQL statements are used to retrieve data
  - PL/SQL control statements are used to manipulate or process data in a PL/SQL program.
- The data can be inserted, deleted, or updated through a PL/SQL block, which makes it an efficient transaction processing language.

PL/SQL Introduction

- SQL – used to select and manipulate data (DML, DDL)

E.g. UPDATE emp SET comm = comm +10;

- PL – Procedural language, provides the ability to run code under conditions, alter program flow, trap errors

E.g. update the commission when an employee has been there a year, or if an employee made over 100 sales …

LEEDS BECKETT UNIVERSITY

Lets look at this example

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

SELECT  e.empno,  e.ename,  d.dname
  INTO  lv_empid,  lv_firstname,  lv_deptname
FROM  emp  e, dept  d
  WHERE  e.empno  = :g_empid
AND e.deptno  = d.deptno;

dbms_output.put_line  ('The  employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
  EXCEPTION
  WHEN no_data_found  THEN
     dbms_output.put_line('The  employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

Lets look at some

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

                    SELECT e.empno,  e.ename,  d.dname
                    INTO lv_empid,  lv_firstname,  lv_deptname
                    FROM emp1 e, dept d
                    WHERE e.empno = :g_empid
                    AND e.deptno = d.deptno;

dbms_output.put_line ('The employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
  EXCEPTION
  WHEN no_data_found  THEN
    dbms_output.put_line('The  employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

Lets look at some

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

SELECT e.empno,  e.ename,  d.dname
                                INTO lv_empid,  lv_firstname,  lv_deptname
FROM emp1 e, dept d
  WHERE e.empno = :g_empid  (replace this with an empno  eg 7499)
AND e.deptno = d.deptno;

dbms_output.put_line  ('The employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
  EXCEPTION
  WHEN no_data_found  THEN
     dbms_output.put_line('The  employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

Lets look at some

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

SELECT e.empno, e.ename, d.dname
                          INTO lv_empid, lv_firstname, lv_deptname
FROM emp1 e, dept d
  WHERE e.empno = 7499
AND e.deptno = d.deptno;

dbms_output.put_line ('The employee: ' || lv_firstname || ' is the department: ' || lv_deptname || ' ');
  EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('The employee: ' || :g_empid || ' is not in this Company.');
END;
```

Lets look at some

SELECT e.empno, e.ename, d.dname

FROM emp1 e, dept d

WHERE e.empno = *7499*

AND e.deptno = d.deptno;

This is SQL selecting the department for an employee (7499).

But we need to re-write this SQL for each employee we cant to find the department for.

Lets look at some

```
SELECT e.empno, e.ename, d.dname
FROM emp1 e, dept d
   WHERE e.empno = :g_empid
AND e.deptno = d.deptno;
```

This code, when run in SQL commands prompts us to enter a value for g_empid, and uses it.

* It can also pick up the field from an apex application and use that.

Back to this, PL/SQL

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

SELECT  e.empno,  e.ename,  d.dname
  INTO  lv_empid,  lv_firstname,  lv_deptname
FROM  emp1  e, dept d
  WHERE  e.empno  = :g_empid
AND e.deptno  = d.deptno;

dbms_output.put_line  ('The  employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
  EXCEPTION
  WHEN no_data_found  THEN
    dbms_output.put_line('The  employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

Back to this, PL/SQL

```
DECLARE
   lv_empid  emp.empno%type;
   lv_firstname  emp.ename%type;
   lv_deptname  dept.dname%type;
BEGIN

SELECT  e.empno,  e.ename,  d.dname
   INTO  lv_empid,  lv_firstname,  lv_deptname
FROM  emp1  e, dept d
   WHERE  e.empno  = :g_empid
AND e.deptno  = d.deptno;

dbms_output.put_line  ('The employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
   EXCEPTION
   WHEN no_data_found  THEN
      dbms_output.put_line('The  employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

Declare/Begin/End – structure of PL/SQL

Back to this, PL/SQL

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

SELECT e.empno, e.ename, d.dname
  INTO lv_empid,  lv_firstname,  lv_deptname
FROM  emp1  e, dept d
  WHERE  e.empno  = :g_empid
AND e.deptno  = d.deptno;

dbms_output.put_line  ('The employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
  EXCEPTION
  WHEN no_data_found  THEN
    dbms_output.put_line('The  employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

These are variables.
Local variables
lv_

Back to this, PL/SQL

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

SELECT  e.empno,  e.ename,  d.dname
  INTO  lv_empid,  lv_firstname,  lv_deptname
FROM  emp1  e, dept d
  WHERE  e.empno  = :g_empid
AND e.deptno  = d.deptno;

dbms_output.put_line  ('The employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
  EXCEPTION
  WHEN no_data_found  THEN
    dbms_output.put_line('The employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

Exception is error handling. no_data_found is an oracle error constant. There are others – google!

Back to this, PL/SQL

```
DECLARE
  lv_empid  emp.empno%type;
  lv_firstname  emp.ename%type;
  lv_deptname  dept.dname%type;
BEGIN

SELECT  e.empno,  e.ename,  d.dname
  INTO  lv_empid,  lv_firstname,  lv_deptname
FROM  emp1  e, dept d
  WHERE  e.empno  = :g_empid
AND e.deptno  = d.deptno;

dbms_output.put_line ('The employee:  ' || lv_firstname  || ' is the department:  ' || lv_deptname  || ' ');
  EXCEPTION
  WHEN no_data_found  THEN
    dbms_output.put_line('The  employee:  ' || :g_empid  || ' is not in this Company.');
END;
```

This is a recognised oracle command (use in SQL commands) to output,
The pipes || are used to output the value of a variable

SQL vs. PL/SQL

1. SQL is a data oriented language for selecting and manipulating sets of data. PL/SQL is a procedural language to create applications.
2. SQL is used to code queries, DML (Data Manipulation language) and DDL (Data definition Language) statements. PL/SQL is used to code program blocks, triggers, functions, procedures and packages.
3. We can embed SQL in a PL/SQL program, but we cannot embed PL/SQL within a SQL statement.

This example I have just shown you is a program block.

Another example.

docs.oracle.com

```
DECLARE
   acct_balance NUMBER(11,2);
   acct CONSTANT NUMBER(4) := 3;
   debit_amt CONSTANT NUMBER(5,2) := 500.00;
BEGIN
      SELECT bal INTO acct_balance FROM accounts
                  WHERE account_id = acct
                   FOR UPDATE OF bal;
       IF acct_balance >= debit_amt  THEN
          UPDATE accounts SET bal = bal - debit_amt
          WHERE account_id = acct;
       ELSE
           INSERT INTO temp VALUES
           (acct, acct_balance, 'Insufficient funds');
            -- insert account, current balance, and message
       END IF;
      COMMIT;
END;
```

So far, PL/SQL

- SQL statements are sent one at the time to the server for execution.
- SQL statements within a PL/SQL block are sent in a single call to the server. This way
  - reduces the overhead
  - and improves performance.

- Provides procedural functionality
- Declare variables and constants
- Use processing loops
- Conditional and sequential control
- Error handling
- Query, insert, update & delete database data

LEEDS BECKETT UNIVERSITY

PL/SQL block structure

- PL/SQL is a block-structured language
- Divided into logical blocks
- Two types of blocks:
  - **An anonymous block**
    - Used anywhere in a program
    - Sent to the server engine for execution at runtime
  - **Named blocks**
    - **A package** is formed from a group of procedures and functions
    - **Trigger** is associated with a database table, executed when automatically fired by a DML statement.

- **A procedure** is a subprogram can be called and can take arguments
- **A function** is a subprogram that returns a calculated value.

LEEDS BECKETT UNIVERSITY

Fundamentals of PL/SQL

- A PL/SQL program consists of statements
- Not case sensitive, except for the character string values enclosed in single quotes.
- PL/SQL consists of reserved words, user-defined words, punctuation marks, and literal values

- Reserved Words:
  - Provided by the language, and they have specific use in the language.
  - Examples:
  - BEGIN
  - END
  - IF
  - WHILE
  - EXCEPTION
  - DECLARE

PL/SQL block structure

- BASIC BLOCK STRUCTURE
  - HEADER
    - Relevant for named blocks only
  - DECLARE
    - Declarations of constants, variables, cursors, and exceptions
  - BEGIN
    - PL/SQL and SQL statements
  - EXCEPTION
    - Actions for Error conditions
  - END

- These are some of the reserved words

Fundamentals of PL/SQL

- **User-defined names:**
  - Used to name variables, constants, procedures,functions,cursors,tables ,records and exceptions.
  - Examples of valid user-defined names:
    - Rate_of_pay, Num, A1234, Dollars$_and_etc, SS#
  - Examples of invalid user-defined names:
    - 2Number, employee-name, END, dept no, taxrate%, largest_yearly_salary_paid_to_employees
- **Literal Values:**
  - Values not represented by user-defined names.
  - Three types:
    - Number: 100, 3.14, -55, 5.25E7, or NULL
    - Character string: 'A', 'this is a string', '0001', '25-MAY-00', ' ', or NULL
    - Boolean: TRUE, FALSE, or NULL

LEEDS BECKETT UNIVERSITY

PL/SQL

- **Data Types VARIABLES including:**
  - CHAR(n)
  - VARCHAR2(n)
  - NUMBER[(n [,d])]
  - DATE
  - BOOLEAN

- **Variable DECLARATION**

DECLARE

IdentifierName[Constant] DataType[NOT NULL] [:=/DEFAULT expression];

DECLARE

weight **CONSTANT NUMBER** (4,3) :=2.345;

title **VARCHAR2(15);**

xdate **CHAR(8) :=** TO_CHAR (SYSDATE, 'DD MM YY');

- **COLUMN & RECORD VARIABLES**
  - VariableName TypeAttribute%TYPE[value assignment];

    DECLARE

    v_sal1 NUMBER (3);

    v_sal2 v_sal1%TYPE;
  - VariableName TableName.ColumnName%TYPE

    DECLARE

    v_sal1 employee.Salary%TYPE;

- **Assignment Operation**

VariableName := *Literal/VariableName/Expression*;

For example:

v_sal1 :=100; *Literal value*

v_sal2 := v_sal1; *Variable value*

v_sum:=v_sal1 + v_num2; *Expression value*

LEEDS BECKETT UNIVERSITY

PL/SQL - Summary

- Why we need PL/SQL
- Structure
- Syntax and constructs
- Conditions and Loops
- For more on PL/SQL read chapters 10-14 of

  Database Systems Using Oracle A simplified guide to SQL and PL/SQL 2nd Edition

  by Nilesh Shah

Thank you

Any questions?