Chapter 1

Introduction to PL/SQL

What is PL/SQL?

- Stands for Procedural Language extension to SQL
- Oracle's standard data access language for relational databases
- Proprietary Oracle language
- Equivalents
  - IBM DB2 - SQL PL
  - Microsoft SQL Server - Transaction-SQL (T-SQL)

What is PL/SQL?

- Tightly integrated with SQL
- PL/SQL program units are compiled by the Oracle Database server and stored inside the database

Comparing SQL & PL/SQL

SQL	PL/SQL
A non-procedural language	A procedural language
Focus is on input and output of data from a database	Used to write the procedural logic to process data
Only SQL can access data from a database	Cannot access database without SQL
Works on sets of data – not row access	Row-based language allowing access to databases data one row at a time
Can be embedded within a PL/SQL program	Can be named, saved and executed whenever needed

PL/SQL and SQL

- PL/SQL provides the integration of procedural code with SQL
  - PL/SQL is used to write the procedural code
  - SQL is embedded within the PL/SQL code to access the database
- PL/SQL provides procedural constructs, such as:
  - Variables, constants, cursors
  - Control structures, such as conditional statements and loops
  - Reusable program units that are written once and executed many times
- "Just like other programming languages"

PL/SQL Blocks

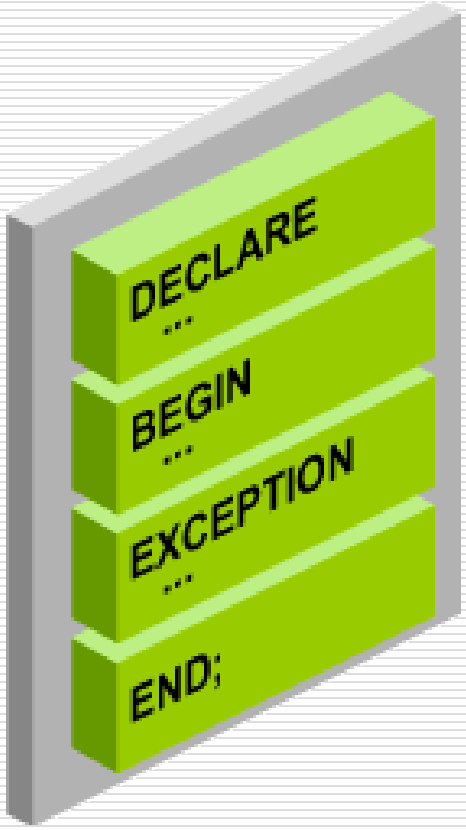
- The basic unit in a PL/SQL program is a block
- All PL/SQL programs consist of blocks
- Three types of PL/SQL blocks:
  - Anonymous blocks
  - Procedures
  - Functions

PL/SQL Anonymous Blocks

- Unnamed block
- Not stored in the database
- Compiled each time the application is executed
- Passed to the PL/SQL engine for execution at run time
- Cannot be invoked or called because it does not have a name and does not exist after it is executed

PL/SQL Anonymous Block Structure

- DECLARE (Declarative - optional)
  - Define variables, cursors, user-defined exceptions
- BEGIN (Executable - mandatory)
  - SQL, logic, loops, assignment statements
  - PL/SQL statements
- EXCEPTION (Exception Handling - optional)
  - Error handling
- END; (mandatory)
  - Close the block



DECLARE Section

- Begins with the keyword DECLARE
- Ends when the BEGIN (executable) section starts
- Not needed if no variables, cursors, or user-defined exceptions are required (Nearly all real-life blocks need a DECLARE section)

```
DECLARE
...    ← Variables, cursors, etc. are declared
BEGIN
```

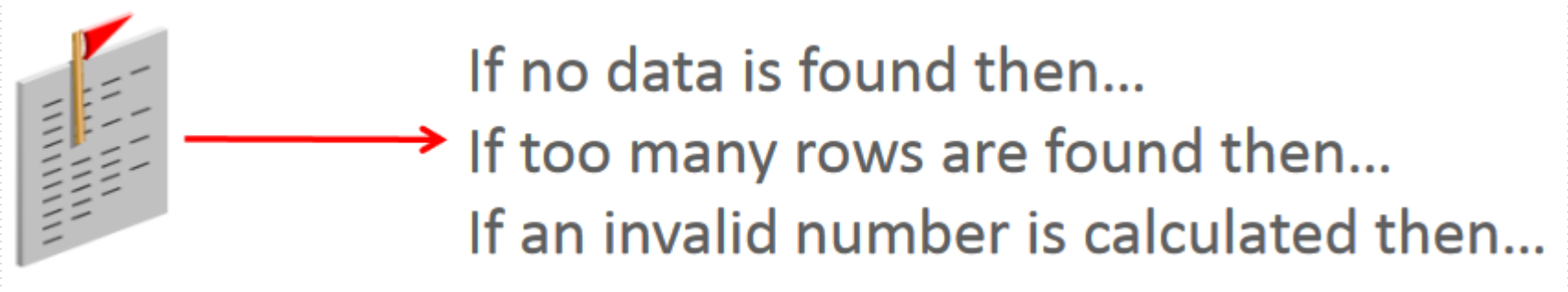
BEGIN or Executable Section

- Begins with the keyword BEGIN
- Ends with END; (must have a semicolon)
- Main executable section of the program

```
DECLARE
   ...
BEGIN
   ...  ← main executable routines
END;
```

EXCEPTION Section

- Exception is an error that occurs during processing in the BEGIN or EXCEPTION section
- Nested at the bottom of the BEGIN section



- Application can handle the error, communicating the problem to the user, without causing a system crash

EXCEPTION Section

- Starts with EXCEPTION
- Terminates when the BEGIN section terminates with END;

```
DECLARE
   ...
BEGIN
   ...
EXCEPTION
   ...      ← exception handling routines
END;
```

DBMS_OUTPUT.PUT_LINE

- DBMS_OUTPUT is a predefined PL/SQL package
- PUT_LINE is a procedure in the DBMS_OUTPUT package that displays its argument ('Hello PL/SQL World') on the screen
- Can be used to check that the block is working or output information

- DBMS_OUTPUT.PUT_LINE('Hello PL/SQL World');

Examples of Anonymous Blocks

- Executable section only (minimum required)

```
BEGIN
   DBMS_OUTPUT.PUT_LINE('PL/SQL is easy!');
END;
```

- Declarative and executable sections

```
DECLARE
   v_date      DATE := SYSDATE;
BEGIN
   DBMS_OUTPUT.PUT_LINE(v_date);
END;
```

PL/SQL Compiler

- Every PL/SQL program must be checked and translated into binary code before it can execute
- Compiler
  - Automatically executes when needed
  - Checks syntax
  - Checks that referenced database objects exist
  - User has the necessary privilege to access database objects

PL/SQL Variable

- The requirements when declaring a variable are:
  - Variable name
  - Data type and Length
  - Initial value (optional)

Data Types

- Variables must be declared with a data type before they can be references in the program
- Only hold a single value

Data type	Description
VARCHAR2(*size*) VARCHAR(*size*)	Stores alphanumeric string of variable length *size* in parentheses represents the maximum length Maximum length of a variable is 32,767 characters. If no length is specified, the default length of 1 is used
CHARACTER(*size*) CHAR(*size*)	Fixed-length character string (*size*) represents the fixed-length
NUMBER(n)	A numeric value with no decimal point
NUMBER(p,s) NUMERIC(p,s) DECIMAL(p,s)	Stores a numeric value Precision (*p*) is the total number of digits Scale (*s*) is the number of digits to the right of the decimal point, and can range from zero (0) to the value specified for precision If scale is not specified, it defaults to zero
PLS_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
INTEGER	A numeric value without a decimal point Stored in binary format Stored as a length of 4-bytes binary with a precision of 10 digits
SMALLINT	Same as **INTEGER** except holds a smaller range of values Stored as a length of 2-bytes binary with a precision of 5 digits
BIGINT	Same as **INTEGER** except holds a larger range of values Stored as a length of 8-bytes binary with a precision of 19 digits
DATE	Stores dates
Boolean	Provide storage for dates and times

Rules for Declaring Variable Names

- Must start with a letter
- Can include letters and numbers
- Can include special characters ($ (dollar sign), _ (underscore), and # (pound sign/hash sign)
- Limit of 30 characters in length
- Cannot contain spaces
- Case insensitive
- Must not be reserved words

Declaring Variables

- Variables must be declared in the DECLARE section
- Optionally, variables can be initialized in the DECLARE section

```
DECLARE
  v_name VARCHAR2(20) := 'Peter';
BEGIN
  ...
END;
```

- PL/SQL assignment operator is represented by a colon and equal sign together (:=)
- Used and assigned new values in the BEGIN or EXCEPTION sections

Initializing Variables

- Variables can be assigned an initial value (optional)
- Initialize variables with the assignment operator (:=) or DEFAULT keyword:

```
Myname VARCHAR2(20):='John';
```

```
Myname VARCHAR2(20) DEFAULT 'John';
```

- Default initial value is NULL

Initializing Variables

- Variables can be modified in the executable section
- Assignment operator (:=)

```
DECLARE
    v_counter   INTEGER := 0;
BEGIN
    v_counter   := v_counter + 1;
    DBMS_OUTPUT.PUT_LINE(v_counter);
END;
```

Reserved Words

- Reserved words are words that have special meaning to the Oracle database
- Reserved words cannot be used as identifiers in a PL/SQL program
- The following is a partial list of PL/SQL reserved words:

ALL	CREATE	FROM	MODIFY	SELECT
ALTER	DATE	GROUP	NOT	SYNONYM
AND	DEFAULT	HAVING	NULL	SYSDATE
ANY	DELETE	IN	NUMBER	TABLE
AS	DESC	INDEX	OR	THEN
ASC	DISTINCT	INSERT	ORDER	UPDATE
BETWEEN	DROP	INTEGER	RENAME	VALUES
CHAR	ELSE	INTO	ROW	VARCHAR2
COLUMN	EXISTS	IS	ROWID	VIEW
COMMENT	FOR	LIKE	ROWNUM	WHERE

Using Reserved Words

- What happens when you try to use a reserved word as a variable in a PL/SQL program?

```
DECLARE
     date DATE;
BEGIN
     SELECT ADD_MONTHS(SYSDATE,3) INTO date
     FROM dual;
END;
```

```
ORA-06550:  line 4, column 37:
PL/SQL:  ORA-00936:  missing expression
ORA-06550:  line 4, column 3:
PL/SQL:  SQL Statement ignored
2.          date DATE;
3.  BEGIN
4.          SELECT ADD_MONTHS(SYSDATE,3) INTO date
5.          FROM DUAL;
6.  END;
```

Character Literals

- Character literals are **case sensitive** and, therefore, 'PL/SQL' is not equivalent to 'pl/sql'
- Character literals have the **data type CHAR** and must be **enclosed in single quotation marks**

```
v_first_name := 'John';
v_classroom  := '12C';
v_date_today := '20-MAY-2005';
```

Numeric Literals

- Represents a number
- Not enclosed in quotation marks

```
v_elevation       := 428;
v_order_subtotal  := 1025.69;
v_growth_rate     := .56;
v_distance_sun_to_centauri := 4.3E13;
```

Boolean Literals

- Values that are assigned to Boolean variables
- TRUE, FALSE, and NULL

```
v_new_customer              := FALSE;
v_paid_in_full              := TRUE;
v_authorization_approved    := FALSE;
v_high_school_diploma       := NULL;
v_island                    := FALSE;
```

Passing Variables as Parameters to PL/SQL Subprograms

- Parameters are values passed to a subprogram by the user or by another program
- Parameters are values required by the subprogram to complete its operations
- Variables can be passed as parameters to procedures and functions

Passing Variables as Parameters to PL/SQL Subprograms

The parameter v_date is passed to the procedure PUT_LINE, which is part of the package DBMS_OUTPUT

```
DECLARE
  v_date DATE := SYSDATE;  -- SYSDATE = 18-Jan-2020
BEGIN
  DBMS_OUTPUT.PUT_LINE(v_date);
END;

18-Jan-2020
Statement processed.
```

Numeric Data Types

- NUMBER
- PLS_INTEGER
- INTEGER
- CONSTANT

```
DECLARE
  v_dept_total_sal   NUMBER(9,2) := 0;
  v_count_loop       INTEGER := 0;
  c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
```

Numeric Data Types

- NUMBER(p,s)
  - Numeric data
  - The p indicates precision, the total number of digits to the left and right of the decimal position
  - The s, or scale, indicates the number of positions to the right of the decimal
  - salary NUMBER(9,2) - can store a numeric value up to 9999999.99
- ANSI Standard is DECIMAL
  - If DECIMAL is used, Oracle converts it to NUMBER internally

Declaring Boolean Variables

- Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable
- Conditional expressions use the logical operators AND and OR and the operator NOT to check the variable values
- Arithmetic, character, and date expressions can be used to return a Boolean value

```
DECLARE
  v_valid          BOOLEAN NOT NULL := TRUE;
  v_is_found       BOOLEAN := FALSE;
  v_underage       BOOLEAN;
```

Variable Declaration Options

- Begin variables with v_
- Constants should be all upper-case or begin with c_
- NOT NULL – the variable must always contain a value, even at initialization
- CONSTANT
  - Must have an initial value
  - Cannot be changed in the block
- DEFAULT – Assigns a default value to the variable

Variable Declaration Options

```
DECLARE
  v_ship_country VARCHAR2(15) NOT NULL := 'US';
  v_tax_rate     CONSTANT     NUMBER(2,2) := .06;
  v_order_date   DATE         DEFAULT SYSDATE;
  v_ship_flag    BOOLEAN := TRUE;
  v_ship_status  VARCHAR2(10);
BEGIN
  IF v_ship_flag  THEN
    v_ship_status := 'Shipped';
  END IF;
  DBMS_OUTPUT.PUT_LINE('The country is : ' || v_ship_country);
  DBMS_OUTPUT.PUT_LINE('The tax rate is : ' || v_tax_rate);
  DBMS_OUTPUT.PUT_LINE('The order_date is : ' || v_order_date);
  DBMS_OUTPUT.PUT_LINE('The ship status is : ' || v_ship_status);
END;
```

```
The country is : US
The tax rate is : .06
The order_date is : 16-JAN-13
The ship status is : Shipped
```

Assigning Values in the Execution Section

```
DECLARE
  v_myname VARCHAR2(20):= 'John';
BEGIN
  v_myname := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myname);
END;
```

```
My name is: Steven
```

NOT NULL

```
DECLARE
    v_myvar1  BOOLEAN      NOT NULL := TRUE;
    v_myvar2  VARCHAR2(15) NOT NULL := 'Initial value';
    v_myvar3  VARCHAR2(15)          := 'Initial value';
--  v_myvar4  VARCHAR2(15) NOT NULL;  -- a variable declared NOT NULL must have an initialization assignment
BEGIN
    IF v_myvar1 THEN
      DBMS_OUTPUT.PUT_LINE('The value of v_myvar1 is TRUE');
    END IF;

    DBMS_OUTPUT.PUT_LINE('v_myvar2 = ' || v_myvar2);

--  v_myvar2 := NULL;  -- expression is of wrong type

    DBMS_OUTPUT.PUT_LINE('v_myvar3 = ' || v_myvar3);
    v_myvar3 := NULL;
    IF v_myvar3 IS NULL THEN
      DBMS_OUTPUT.PUT_LINE('The value of v_myvar3 is NULL');
    END IF;
END;
```

```
The value of v_myvar1 is TRUE
v_myvar2 = Initial value
v_myvar3 = Initial value
The value of v_myvar3 is NULL
```

NULL Default Value

- A variable is initialized to NULL by default
- Any operation calculated with a NULL returns a NULL

```
DECLARE
   v_counter INTEGER;
BEGIN
   v_counter := v_counter + 1;
   DBMS_OUTPUT.PUT_LINE(v_counter);
END;
```

```
Statement processed.
```

NULL Default Value (cont.)

- Variable is initialized to NULL
- Value of variable is changed during execution

```
DECLARE
   v_counter INTEGER;
BEGIN
   v_counter := 0;
   v_counter := v_counter + 1;
   DBMS_OUTPUT.PUT_LINE(v_counter);
END;
```

```
1

Statement processed.
```

Calculations with Variables

```
DECLARE
  v_taxrate_num CONSTANT NUMBER(2,2) := .06;
  v_total_num NUMBER(6,2) := 50;
  v_taxamt_num NUMBER(4,2);
BEGIN
  v_taxamt_num := v_total_num * v_taxrate_num;
  DBMS_OUTPUT.PUT_LINE(v_taxamt_num);
END;
```

```
3

Statement processed.
```

Delimiters

- Delimiters are symbols that have special meaning to the Oracle database

Simple delimiters

Symbol	Meaning
+	Addition operator
–	Subtraction/negation operator
*	Multiplication operator
/	Division operator
=	Equality operator
'	Character string delimiter
;	Statement terminator

Compound delimiters

Symbol	Meaning
<>	Inequality operator
!=	Inequality operator
||	Concatenation operator
--	Single-line comment indicator
/*	Beginning comment delimiter
*/	Ending comment delimiter
:=	Assignment operator

%TYPE Attribute

- The %TYPE attribute:
  - Declares a variable with the same data type and size as:
    - A database column definition
    - Another declared variable
  - Is prefixed with either of the following:
    - The database table and column
    - The name of the other declared variable

Example of %TYPE Attribute

```
CREATE TABLE myemps (
  emp_name          VARCHAR2(6),
  emp_salary        NUMBER(6,2));

DECLARE
  v_emp_salary    myemps.emp_salary%TYPE;
BEGIN
  SELECT emp_salary INTO v_emp_salary
  FROM myemps WHERE emp_name = 'Smith';
END;
```

Comments

Commenting a single line:

- Two dashes -- are used for commenting a single line

Commenting multiple lines:

- /* text here */ is used for commenting multiple lines

- Ignored by PL/SQL

Example of Comments

```
DECLARE
...
  v_annual_sal NUMBER (9,2);
BEGIN     -- Begin the executable section

/* Compute the annual salary based on the
   monthly salary input from the user */
  v_annual_sal := v_monthly_sal * 12;
END;      -- This is the end of the block
```

GOOD PROGRAMMING PRACTICES

Good Programming Practices

- Conversions:
  - Do not rely on implicit data type conversions because they can be slower, and the rules can change in later software releases

- Declaring and initializing PL/SQL variables:
  - Use meaningful names
  - Declare one identifier per line for better readability and code maintenance
  - Use the NOT NULL constraint when the variable must hold a value
  - Avoid using column names as identifiers
  - Use the %TYPE attribute to declare a variable according to another previously declared variable or database column

Naming Conventions

Commonly-used convention is to name:

- Variables starting with v_
- Constants should be all upper-case or start with c_
- Parameters (passed to procedures and functions) starting with p_

Case Standard

Category	Case Convention	Examples
SQL keywords	Uppercase	SELECT, INSERT
PL/SQL keywords	Uppercase	DECLARE, BEGIN, IF
Data types	Uppercase	VARCHAR2, BOOLEAN
Identifiers and parameters	Lowercase	v_sal, emp_cursor, g_sal, p_empno
Database tables and columns	Lowercase	employees, employee_id, department_id

Commenting Code

- Prefix single-line comments with two dashes
  - (--).
- Place multiple-line comments between the symbols /* and */

Indenting Code

```
DECLARE
  v_deptno        NUMBER(4);
  v_location_id   NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
    INTO  v_deptno,
          v_location_id
    FROM  departments
    WHERE department_name = 'Sales';
...
END;
```

PL/SQL Common Errors

- Use = rather than :=
- Not declaring a variable
- Misspelling a variable name
- Not terminating a statement with ;
- No data returned from a SELECT statement

FIRST PL/SQL PROGRAM

Bind/Host Variables

- Used to prompt the application environment for variables that are returned to the PL/SQL block
- APEX is an application environment

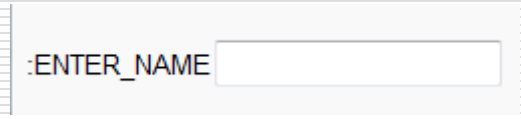
```
DECLARE
  v_my_name VARCHAR2(20) := :enter_name;
BEGIN
  DBMS_OUTPUT.PUT_LINE(v_my_name);
END;
```

Bind/Host Variables

- Not declared as a variable in the PL/SQL block
- Reference host variables with a preceding colon in PL/SQL
- Begins with colon (:)
  - :Enter_customer_number
- Launches an input box

Bind/Host Variable Examples

```
DECLARE
  v_my_name VARCHAR2(20) := :Enter_name;
BEGIN
  DBMS_OUTPUT.PUT_LINE(v_my_name);
END;
```




```
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is ' || :Enter_name);
END;
```

First PL/SQL Program

- Requirements:
  - Enter customer name using Bind variable
  - Enter sale amount using Bind variable

Submit

	Value
:ENTER_CUSTOMER_NAME	Sally Smith
:ENTER_SALE_AMOUNT	300

First PL/SQL Program

- Requirements:
  - Calculate tax
  - Calculate total sale
  - Output details

```
Customer Sales for Thursday, January 30, 2020
_______________

Customer:        Sally Smith
Sale:            $300.00
Tax:              $39.00
Total sale:      $339.00
```

```oracle_plsql
DECLARE
  v_customer_name VARCHAR2(30)          := :Enter_customer_name;
  v_sale          NUMBER(7,2)           := :Enter_sale_amount;
  v_tax           v_sale%TYPE           := 0;
  v_total_sale    v_sale%TYPE           := 0;
  v_sale_date     DATE;
  v_ship_status   BOOLEAN               := FALSE; -- Cannot print Boolean value
  C_TAX_RATE      CONSTANT NUMBER(3,2) := 0.13;
  C_TAB           CONSTANT VARCHAR2(1) := CHR(9);
BEGIN
  v_sale_date := CURRENT_DATE;
  v_tax := v_sale * C_TAX_RATE;
  v_total_sale := v_sale + v_tax;
  DBMS_OUTPUT.PUT_LINE('Customer Sales for ' || TO_CHAR(v_sale_date, 'FMDay, Month DD, YYYY'));
  DBMS_OUTPUT.PUT_LINE('______________');
  DBMS_OUTPUT.NEW_LINE;
  DBMS_OUTPUT.PUT_LINE('Customer:' || C_TAB || v_customer_name);
  DBMS_OUTPUT.PUT_LINE('Sale:' || LPAD(TO_CHAR(v_sale, '$99,999.99'), 18));
  DBMS_OUTPUT.PUT_LINE('Tax:'  || LPAD(TO_CHAR(v_tax, '$99,999.99'), 19));
  DBMS_OUTPUT.PUT_LINE('Total sale:' || LPAD(TO_CHAR(v_total_sale, '$99,999.99'), 12));
END;
```

Using Host/Bind Variables You Try

```
DECLARE
  v_state_code VARCHAR2(2)  := :enter_state_code;
  v_sale_amount NUMBER(7,2) := :enter_sale_amount;
  v_tax_amount NUMBER(4,2);
BEGIN
  IF v_state_code = 'VA' THEN
     v_tax_amount := v_sale_amount * .06;
  ELSIF v_state_code = 'CA' THEN
    v_tax_amount := v_sale_amount * .08;
  ELSE
    v_tax_amount := v_sale_amount * .04;
  END IF;
  DBMS_OUTPUT.PUT_LINE('The tax on $' || v_sale_amount || ' in ' ||
    v_state_code || ' is $' || v_tax_amount);
END;
```


Q & A
Questions &
Answers