Orange Coast College
Business Division - CS/CIS Department
CIS 183 – Oracle™ SQL Programming
Instructor: Dr. Martha Malaty

Text & Original Presentations: Joan Casteel
Oracle 11g: SQL
Extra Chapter
Introduction to PL/SQL
Chapter Objectives

- Explain the benefits of using PL/SQL blocks
- Identify the sections of a PL/SQL block
- Identify the mandatory and optional sections
- Identify an anonymous block and its use
- Describe how to execute a PL/SQL block
- Explain the purpose of a variable
- Explain the difference between constants & variables
- List the valid data types for PL/SQL variables
- Dynamic data types for PL/SQL variables
- Initialize PL/SQL variables
- Use DML statements in a PL/SQL block
- Identify the clauses of an IF statement
- Identify the purpose and types of loops available in Oracle;
  - Basic loop, FOR loop, & WHILE loop
Chapter Objectives

- Study explicit cursors
- Declare, open, and close an explicit cursor
- Fetch data from an explicit cursor
- Identify attributes associated with a cursor
- Use a cursor FOR loop to retrieve data from a cursor
- Declare a cursor in the subquery of a cursor FOR loop
- Evaluate BOOLEAN conditions combined with logical operators
- Identify the purpose of the exception-handling section of a PL/SQL block
- Trap predefined & user-defined exceptions in a PL/SQL block
Each SQL statement requires **two trips** to the server

- From the user to server **sending** the query
- From the server to the user **returning result** of processing
- ➡ Too much traffic

Need some way to minimize traffic
PL/SQL

- Procedural Language SQL
- Advanced 4th generation (4GL) programming language
- Extends SQL capabilities
- Programming Languages generations
  - 1st Generation Programming Language: Machine code (0 & 1)
  - 2nd Generation Programming Language: Assembly
  - 3rd Generation Programming Language: General-purpose (C, C++, C#, Java, BASIC and Delphi)
  - 4th Generation Programming Language: Designed to reduce programming effort (SQL)
  - 5th Generation Programming Language: Solving problems using constraints given to the program, rather than using an algorithm written by a programmer (Prolog, LISP)
  - Domain-specific language: A programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique
Advantages of PL/SQL

- Can include
  - Error handling
  - Control structures
- Can be stored and used by various application programs or users
- Allows for tighter security by granting privileges for executing stored procedures rather than granting privileges directly on database object
Types of Blocks

- Named Blocks
  - Function
  - Procedure

- Anonymous block
Function

- Named block that is stored on the Oracle server
- Accepts zero or more input parameters
- Returns exactly one result value
- Can be called more than once
- Called from within an expression
Procedure

- Stored on the server
- Also called “Stored Procedures”
- Is a named block
- Can process several variables
- Returns no value in its name
- Interacts with application program using IN, OUT, or INOUT parameters
- Can be called more than once
- Called in a separate statement
Anonymous Block

- Not stored since it cannot be referenced by a name
- Usually embedded in an application program, stored in a script file, or manually entered when needed
- Can’t be called more than once
Basic Structure of a Block

- Three sections:
  - Declarative section
  - Executable section
  - Exception-handling section

- Syntax

  [DECLARE]
  Variables & constants declaration

  BEGIN
  ... SQL & other executable statements
  [EXCEPTION]
  Exception handling code

  END;

- Executable section is the only required section; the rest are optional
Declarative Section

- Identified by the **DECLARE** keyword
- Used to **define variables and constants** referenced in the block

**Variable**
- Reserve a **temporary storage area in memory**
- Manipulated without accessing a physical storage medium

**Constant**
- Its assigned **value doesn’t change** during execution

**Forward execution**
- Variable and constants must be **declared before they can be referenced**

- Both variables & constants can be **declared anywhere in the PL/SQL block**
Executable Section

- Identified by the **BEGIN** keyword
- Mandatory
- Can consist of several **SQL and/or PL/SQL statements**
- Used to **access & manipulate data** within the block
Exception-Handling Section

- Identified by the **EXCEPTION** keyword
- Used to **display messages** or identify other **actions** to be taken when an **error** occurs
- **Addresses errors** that occur during a statement’s execution
  - Examples of errors:
    - No rows returned
    - Divide by zero
END Keyword

- Used to close a PL/SQL block
- Always followed by a semicolon
Example PL/SQL Block

```
DECLARE
    c_rateincrease    CONSTANT   NUMBER(3,2) :=1.2;
    v_title           VARCHAR2(30);
    v_retail          books.retail%TYPE;
    v_newretail       NUMBER(5,2);
BEGIN
    SELECT title, retail, retail*c_rateincrease
    INTO v_title, v_retail, v_newretail
    FROM books
    WHERE isbn = '1059831198';
    DBMS_OUTPUT.PUT_LINE
        ('The new price for ' || v_title || ' is $' || v_newretail);
END;
```
Declaring a Variable

- Reserves a temporary storage area in the computer’s memory

- Every variable must have:
  - A name
  - A data type
  - Optional initial value

- Variable name can be up to 30 characters

- Consist of letters, digits, or special symbols

- Variable name must begin with a letter
Variable Naming Convention

- Use the following prefixes for variables, constants, & global variables:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Variable Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_</td>
<td>Constants—variables whose values do not change during the execution of the block</td>
</tr>
<tr>
<td>g_</td>
<td>Global variables—variables that are referenced by the host or calling environment. These variables are commonly used with application programs.</td>
</tr>
<tr>
<td>v_</td>
<td>Variables—used in a PL/SQL block to denote values that might change during the execution of the block</td>
</tr>
</tbody>
</table>
Variables that have a value that does not change during the execution of the block

CONSTANT keyword can be used to designate a constant in the block’s declarative section
PL/SQL Data Types

- **Scalar** (or simple)
  - Holds a *single value*

- **Composite** (or structured)
  - Collection of *grouped data* treated as one unit

- **Reference**
  - Holds *pointers* to other program items

- **Large Object (LOB)**
  - Holds *location* of large objects
# PL/SQL Scalar Data Types

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>v_region CHAR(2);</td>
</tr>
<tr>
<td>NUMBER</td>
<td>v_retail NUMBER(5,2);</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>v_instock BOOLEAN;</td>
</tr>
<tr>
<td>DATE</td>
<td>v_pubdate DATE;</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>v_title VARCHAR2(30);</td>
</tr>
<tr>
<td>BINARY_INTEGER</td>
<td>v_onhand BINARY_INTEGER;</td>
</tr>
<tr>
<td>PLS_INTEGER</td>
<td>v_backordered PLS_INTEGER;</td>
</tr>
<tr>
<td>%TYPE</td>
<td>v_region customers.region%TYPE; v_newretail v_retail%TYPE;</td>
</tr>
</tbody>
</table>
PL/SQL Scalar Data Types

- **PLS_INTEGER**
  - More memory- & performance efficient
  - Range: From \(-2,147,483,647\) to \(+2,147,483,647\)
  - Used for local PL/SQL variables

- **BINARY_INTEGER**
  - Stores signed integer variables in binary format
  - Takes less space and generates faster calculations
  - Range: From \(-2,147,483,647\) to \(+2,147,483,647\)

- **%TYPE**
  - Assumes the same data type as another variable or a DB column
  - Frequently used in industry for consistency, even if column types change
  - Changes dynamically
PL/SQL Scalar Data Types

- Other types
  - TIMESTAMP
  - LONG
  - LONG ROW
  - ROWID

- Discussed in details in PL/SQL class
Variable Initialization

- Use **DEFAULT** keyword or (:=) assignment operator
- **Variable must be initialized** if it is assigned a **NOT NULL** constraint
- Non-numeric data types must be enclosed in single quotation marks
## Variable Initialization Examples

<table>
<thead>
<tr>
<th>Assignment Operator (:=)</th>
<th>DEFAULT keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v_adate DATE NOT NULL := '04-APR-03';</code></td>
<td><code>v_adate DATE NOT NULL DEFAULT '04-APR-03';</code></td>
</tr>
<tr>
<td><code>c_anumber NUMBER(5) := 25;</code></td>
<td><code>c_anumber NUMBER(5) DEFAULT 25;</code></td>
</tr>
<tr>
<td><code>c_acharacter VARCHAR2(12) := 'Howdy';</code></td>
<td><code>c_acharacter VARCHAR2(12) DEFAULT 'Howdy';</code></td>
</tr>
<tr>
<td><code>v_instock BOOLEAN := TRUE;</code></td>
<td><code>v_instock BOOLEAN DEFAULT TRUE;</code></td>
</tr>
<tr>
<td><code>c_bnumber BOOLEAN := FALSE;</code></td>
<td><code>c_bnumber BOOLEAN := DEFAULT FALSE;</code></td>
</tr>
</tbody>
</table>
SELECT Statement

- Requires use of **INTO** clause to identify variable assigned to each data element
- To get output displayed, you need to set the **SET SERVEROUTPUT ON**

```
SET SERVEROUTPUT ON;
```

- Syntax:

```
SELECT columnname [, columnname,...] INTO variablename [, variablename,...] FROM tablename WHERE condition;
```
SELECT Statement Example

```
SQL> DECLARE
  2    c_rateincrease CONSTANT NUMBER(3,2) := 1.2;
  3    v_title     VARCHAR2(30);
  4    v_retail    books.retail%TYPE;
  5    v_newretail NUMBER(5,2);
  6    BEGIN
  7        SELECT title, retail, retail * c_rateincrease
  8            INTO v_title, v_retail, v_newretail
  9            FROM books
 10            WHERE isbn = '1059831198';
 11        DBMS_OUTPUT.PUT_LINE
 12           ('The new price for ' || v_title || ' is $' || v_newretail);
 13    END;
 14 /
The new price for BODYBUILD IN 10 MINUTES A DAY is $37.14
PL/SQL procedure successfully completed.
SQL>
```
Execution Control

- **Decision:**
  - **IF statement** – executes statements based on a condition

- **Loops:**
  - **Basic loop**
    - Executes statements until condition in `EXIT` clause is **TRUE**
  - **FOR loop**
    - Uses counter
  - **WHILE loop**
    - Executes statements until condition is **FALSE**
IF Statement Syntax & Example

IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;

```sql
DECLARE
    v_gift VARCHAR2(20);
    c_retailprice NUMBER(5, 2) := 29.95;
BEGIN
    IF c_retailprice > 56 THEN
        v_gift := 'FREE SHIPPING';
    ELSIF c_retailprice > 25 THEN
        v_gift := 'BOOKCOVER';
    ELSIF c_retailprice > 12 THEN
        v_gift := 'BOX OF BOOK LABELS';
    ELSE
        v_gift := 'BOOKMARKER';
    END IF;
    DBMS_OUTPUT.PUT_LINE ('The gift for a book costing ' || c_retailprice || ' is a ' || v_gift);
END;
/
```

The gift for a book costing 29.95 is a BOOKCOVER

PL/SQL procedure successfully completed.
Basic Loop Syntax & Example

```sql
LOOP
    statements;
    EXIT [WHEN condition];
END LOOP;
```

```sql
SQL> DECLARE
2      v_counter     NUMBER(1) := 0;
3  BEGIN
4      LOOP
5          v_counter := v_counter + 1;
6          DBMS_OUTPUT.PUT_LINE ('The current value of the counter is ' || v_counter);
7          EXIT WHEN v_counter = 4;
8      END LOOP;
9  END;
10 /
```

The current value of the counter is 1
The current value of the counter is 2
The current value of the counter is 3
The current value of the counter is 4
PL/SQL procedure successfully completed.

```sql
SQL>
```
FOR Loop Syntax & Example

FOR counter IN [REVERSE] lower_limit .. upper_limit LOOP
  statements;
END LOOP;

```
SQL> BEGIN
  2    FOR i IN 1 .. 10 LOOP
  3        DBMS_OUTPUT.PUT_LINE ('The current value of the counter is ' || i);
  4    END LOOP;
  5 END;
  6 /
The current value of the counter is 1
The current value of the counter is 2
The current value of the counter is 3
The current value of the counter is 4
The current value of the counter is 5
The current value of the counter is 6
The current value of the counter is 7
The current value of the counter is 8
The current value of the counter is 9
The current value of the counter is 10
PL/SQL procedure successfully completed.
SQL>
```
WHILE Loop Syntax & Example

WHILE condition LOOP
    statements;
END LOOP;

```
DECLARE
    v_counter NUMBER(2) := 0;
BEGIN
    WHILE v_counter < 15 LOOP
        DBMS_OUTPUT.PUT_LINE ('The current value of the counter is ' || v_counter);
        v_counter := v_counter + 1;
    END LOOP;
END;
/
```

The current value of the counter is 0
The current value of the counter is 1
The current value of the counter is 2
The current value of the counter is 3
The current value of the counter is 4
The current value of the counter is 5
The current value of the counter is 6
The current value of the counter is 7
The current value of the counter is 8
The current value of the counter is 9
The current value of the counter is 10
The current value of the counter is 11
The current value of the counter is 12
The current value of the counter is 13
The current value of the counter is 14

PL/SQL procedure successfully completed.

SQL>
Nested Loops

- Any type of loop can be nested inside another loop
- Execution of the inner loop must be completed before control is returned to the outer loop
- Example:

```
SQL> DECLARE
2     v_counter NUMBER(2) := 0;
3     BEGIN
4       WHILE v_counter < 3 LOOP
5           FOR i IN 1..2 LOOP
6               DBMS_OUTPUT.PUT_LINE ('The current value of the FOR LOOP counter is ' || i);
7           END LOOP;
8       DBMS_OUTPUT.PUT_LINE ('The current value of the WHILE counter is ' || v_counter);
9       v_counter := v_counter + 1;
10     END LOOP;
11   END;
12 /
The current value of the FOR LOOP counter is 1
The current value of the FOR LOOP counter is 2
The current value of the WHILE counter is 0
The current value of the FOR LOOP counter is 1
The current value of the FOR LOOP counter is 2
The current value of the WHILE counter is 1
The current value of the FOR LOOP counter is 1
The current value of the FOR LOOP counter is 2
The current value of the WHILE counter is 2
The current value of the FOR LOOP counter is 2
PL/SQL procedure successfully completed.
SQL>
```
Cursors

- Memory areas that allows accessing information retrieved from a SQL statement

- Example:
  - Use a cursor operator on all rows for customers with a specific condition

- Syntax:

```sql
CURSOR cursor_name IS SELECT query;
```
Cursor Types

- **Implicit cursor**
  - Created for DML operations or a `SELECT` statement that retrieves only one row of results

- **Explicit cursor**
  - Required for `SELECT` statements retrieving more than one row of results
Opening & Closing an Explicit Cursor

- When opened
  - Memory is allocated
  - The **SELECT** statement is executed
  - Necessary data is loaded into the cursor

- When closed
  - Memory is released
Fetching Data from the Cursor

- Data is retrieved from an explicit cursor using the **FETCH** command
- Assigns the values to previously declared variables

```
FETCH cursor_name INTO variablename [, ... variablename];
```
# Cursor Attributes

<table>
<thead>
<tr>
<th>Cursor Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%ROWCOUNT</code></td>
<td>Identifies the number of rows processed</td>
</tr>
<tr>
<td><code>%FOUND</code></td>
<td>Contains the value of TRUE if one or more rows are processed—FALSE if no rows are processed</td>
</tr>
<tr>
<td><code>%NOTFOUND</code></td>
<td>Contains the value of TRUE if no rows are processed—FALSE if one or more rows are processed</td>
</tr>
<tr>
<td><code>%ISOPEN</code></td>
<td>Contains the value TRUE if a cursor is not closed after processing—FALSE if the cursor is closed. The value will always be FALSE after processing when an implicit cursor occurs because its closure is automatic.</td>
</tr>
</tbody>
</table>
Cursor Example

```
SQL> DECLARE
2    v_title  books.title%TYPE;
3    v_retail books.retail%TYPE;
4  CURSOR books_cursor IS
5      SELECT title, retail
6         FROM books NATURAL JOIN orderitems
7       WHERE order# = 1012;
8 BEGIN
9    OPEN books_cursor;
10   LOOP
11      FETCH books_cursor INTO v_title, v_retail;
12    EXIT WHEN books_cursor%NOTFOUND;
13      DBMS_OUTPUT.PUT_LINE ('Book title: ' || v_title || ' Retail price: ' || v_retail);
14    END LOOP;
15  CLOSE books_cursor;
16  END;
/
Book title: BIG BEAR AND LITTLE DOVE Retail price: 8.95
Book title: HANDCRANKED COMPUTERS Retail price: 25
Book title: PAINLESS CHILD-REARING Retail price: 89.95
Book title: REVENGE OF MICKEY Retail price: 22
PL/SQL procedure successfully completed.
SQL>
```
Cursor FOR Loop Syntax & Example

- Automatically opens and closes a cursor, and fetches data from a cursor

```
FOR record_name IN cursor_name LOOP
    statement;
    [statement;...]
END LOOP;
```

```
BEGIN
FOR r_books IN (SELECT title, retail
    FROM books NATURAL JOIN orderitems
    WHERE order# = 1012) LOOP
    DBMS_OUTPUT.PUT_LINE ('Book title: ' || r_books.title || ' Retail price: ' || r_books.retail);
END LOOP;
/
```

Book title: BIG BEAR AND LITTLE DOVE Retail price: 8.95
Book title: HANDCRANKED COMPUTERS Retail price: 25
Book title: PAINLESS CHILD-REARING Retail price: 89.95
Book title: REVENGE OF MICKEY Retail price: 22

PL/SQL procedure successfully completed.
```
%ROWTYPE Attribute

- Specifies that a record will have the same structure as the row being retrieved

```sql
SQL> DECLARE
2  CURSOR books_cursor IS
3    SELECT title, retail
4    FROM books NATURAL JOIN orderitems
5    WHERE order# = 1012;
6  r_books books%ROWTYPE;
7 BEGIN
8  FOR r_books IN books_cursor LOOP
9    DBMS_OUTPUT.PUT_LINE ('Book title: ' || r_books.title || ' Retail price: ' || r_books.retail);
10 END LOOP;
11 END;
12 /

PL/SQL procedure successfully completed.
```
Logic (Truth) Table

- Determines how a statement will be evaluated when conditions are joined with logical operators

<table>
<thead>
<tr>
<th>AND</th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
<th>OR</th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>FALSE</td>
<td>NULL</td>
<td>NULL</td>
<td>TRUE</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Exception Handling

- **Exception**
  - Signal that an **error** has occurred
  - Does not include **syntax** errors

- **Common examples:**
  - **NO_DATA_FOUND**
  - **TOO_MANY_ROWS**
  - **ZERO_DIVIDE**

- **Syntax:**

```
EXCEPTION
WHEN exception_type THEN
  statement;
  [WHEN exception_type THEN
   statement;...]
  [WHEN OTHERS THEN
   statement;...]
```
Exception-Handling Section Example

```
SQL> DECLARE
2       v_title  books.title%TYPE;
3       v_retail books.retail%TYPE;
4   BEGIN
5       SELECT title, retail
6           INTO v_title, v_retail
7           FROM books
8           WHERE retail > 100;
9   EXCEPTION
10      WHEN NO_DATA_FOUND THEN
11          DBMS_OUTPUT.PUT_LINE ('No rows were retrieved from the table');
12   END;
13 /
No rows were retrieved from the table
PL/SQL procedure successfully completed.
SQL>
```
PRAGMA

- A compiler directive or "hint"
- Used to provide an instruction to the compiler
- Does not affect the meaning of a program
- Simply conveys information to the compiler
- Processed at compile time
- 5 types of **Pragma directives** available in Oracle:
  - PRAGMA AUTONOMOUS_TRANSACTION
  - PRAGMA SERIALLY_REUSABLE
  - PRAGMA RESTRICT_REFERENCES
  - PRAGMA INLINE
  - PRAGMA EXCEPTION_INIT
Associates an exception name with an Oracle error number

Binds a user defined exception to a particular error number

Syntax

PRAGMA EXCEPTION_INIT (exception_name, error_number);

- **Error_number**
  - Any valid Oracle error number
  - Same error numbers returned by the function **SQLCODE** that returns the number code of the most recent exception

- **Exception_name**
  - Identifies a user–defined exception previously declared within the current scope
User–Defined Exception Handling

- Name of exception must be declared with data type EXCEPTION
- Declared exception must be associated with Oracle server error number using PRAGMA EXCEPTION_INIT statement
- Must be included in the block’s exception–handling section
- You can use EXCEPTION_INIT in the declarative part of any PL/SQL block, subprogram, or package
- The pragma must appear in the same declarative part as its associated exception, somewhere after the exception declaration.
PRAGMA EXCEPTION_INIT Example

```sql
SQL> DECLARE
  2  id_already_in_use EXCEPTION;
  3  PRAGMA EXCEPTION_INIT(id_already_in_use, -00001);
  4  BEGIN
  5  INSERT INTO publisher
  6  VALUES ('1', 'A NEW PUBLISHER', 'GUY SMART', '800-555-2211');
  7  EXCEPTION
  8  WHEN id_already_in_use THEN
  9    DBMS_OUTPUT.PUT_LINE ('Please choose another publisher ID');
 10  END;
 11 /
Please choose another publisher ID
PL/SQL procedure successfully completed.
SQL>
```