A1

Guide to Oracle 10*g*

Chapter 4: Introduction to PL/SQL

(Three Lessons: A, B, and C)

Edited by:

Dr. Emad Al-Shawakfa
CIS 360: Developing Database Applications Course
Department of Information Systems
Yarmouk University
2021

Lesson A

Slide 1

A1 Global to this PPT: Please note that most figure slides have figure caption as slide title, not heading - okay? (The individual slides are commented.)
Author, 8/1/2005

Lesson A Objectives

After completing this lesson, you should be able to:

- Describe the fundamentals of the PL/SQL programming language
- Write and execute PL/SQL programs in SQL*Plus
- Execute PL/SQL data type conversion functions
- Display output through PL/SQL programs
- Manipulate character strings in PL/SQL programs
- Debug PL/SQL programs

Fundamentals of PL/SQL

- Full-featured programming language
- Execute using Oracle 10*g* utilities
  - SQL*Plus
  - Forms Builder
- Interpreted language
- Semicolon ends each command
- Reserved words

A2

PL/SQL Command Capitalization Styles

Item Type	Capitalization	Example
Reserved word	Uppercase	BEGIN, DECLARE
Built-in function	Uppercase	COUNT, TO_DATE
Predefined data type	Uppercase	VARCHAR2, NUMBER
SQL command	Uppercase	SELECT, INSERT
Database object	Lowercase	student, f_id
Variable name	Lowercase	current_s_id, current_f_last

Table 4-1 PL/SQL command capitalization styles

PL/SQL Variables and Data Types

- Variable names must follow the Oracle naming standard
- Strongly typed language
  - Explicitly declare each variable including data type before using variable
- Variable declaration syntax:
  - *variable_name data_type_declaration;*
- Default value always NULL

Slide 5

A2 Okay that slide title is figure caption and not heading?

Author, 8/1/2005

Declaring PL/SQL Variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
     [:= | DEFAULT expr];
```

Examples

```
Declare
  v_hiredate    DATE;
  v_deptno      NUMBER(2) NOT NULL := 10;
  v_location    VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

7

Assigning Values to Variables

Syntax

- identifier := expr;

Examples

Set a predefined hiredate for new employees

```
v_hiredate := '31-DEC-98';
```

Set the employee name to Maduro.

```
v_ename := 'Maduro';
```

8

Scalar Variables

- Reference single value
- Data types correspond to Oracle 10*g* database data types
  - VARCHAR2
  - CHAR
  - DATE
  - NUMBER

Base Scalar Datatypes

- VARCHAR2 (maximum_length)
- NUMBER [(precision, scale)]
- DATE
- CHAR [(maximum_length)]
- LONG
- BOOLEAN
- BINARY_INTEGER

Scalar Variable Declarations

- Examples

```
v_job          VARCHAR2(9);
v_count        BINARY_INTEGER := 0;
v_total_sal      NUMBER(9,2) := 0;
v_orderdate      DATE := SYSDATE + 7;
c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;
v_valid        BOOLEAN NOT NULL := TRUE;
```

11

A3

General Scalar Data Types

Data Type	Description	Sample Declaration
Integer number subtypes (BINARY_INTEGER, INTEGER, INT, SMALLINT)	Integer	`counter BINARY_INTEGER;`
Decimal number subtypes (DEC, DECIMAL, DOUBLE PRECISION, NUMERIC, REAL)	Numeric value with varying precision and scale	`student_gpa REAL;`
BOOLEAN	True/False value	`order_flag BOOLEAN;`

Table 4-3 General scalar data types

Slide 12

A3 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

DATA TYPES

- PL/SQL has four data types (cont.):
  - **Composite**:
    - The composite data types are made up of elements or components.
    - PL/SQL supports three composite data types:
      - *records*
      - *tables*
      - *varrays*
  - **Reference**:
    - The reference data types deal with objects.
  - **LOB** (Large Object)

Composite Variables

- Data object made up of multiple individual data elements
- Data structure contains multiple scalar variables
- Types:
  - RECORD
  - TABLE
  - VARRAY

Reference Variables

- Directly reference specific database column or row
- Assume data type of associated column or row
- %TYPE data declaration syntax:
  - *variable_name tablename.fieldname*%TYPE;
- %ROWTYPE data declaration syntax:
  - *variable_name tablename*%ROWTYPE;

Declaring Variables with the %TYPE Attribute

- Examples

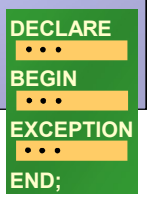
```
...
  v_ename              emp.ename%TYPE;
  v_balance            NUMBER(7,2);
  v_min_balance        v_balance%TYPE := 10;
...
```

OTHER DATA TYPES

- **NLS**
  - **The National Language Support (NLS) data type is for character sets where multiple bytes are used for character representation.**
  - **NCHAR and NVARCHAR2 are examples of NLS data types.**
- **LOB**
  - **Like Oracle9i, PL/SQL also supports Large Object (LOB) data types to store large values of character, raw, or binary data.**
  - **They allow up to 4 gigabytes of data.**
    - **The *BLOB* type contains a pointer to the large binary object inside the database.**
    - **The *CLOB* type contains a pointer to a large block of single-byte character data of fixed width.**
    - **The *NCLOB* type contains a pointer to a large block of multibyte character data of fixed width.**
    - **The *BFILE* type contains a pointer to large binary objects in an external operating system file.**
      - **It would contain the directory name and the filename.**

PL/SQL Block Structure

- DECLARE – Optional
  Variables, cursors, user-defined exceptions
- BEGIN – Mandatory
  - SQL statements
  - PL/SQL statements
- EXCEPTION – Optional
  Actions to perform when errors occur
- END; – Mandatory



18

PL/SQL Program Blocks

- Declaration section
  - Optional
- Execution section
  - Required
- Exception section
  - Optional
- Comment statements
  - Enclosed within /* and */ or --

Comments

- Comments are used to document programs.
- They are written as part of a program, but are not executed.
- In fact, comments are ignored by the PL/SQL engine.
- It is a good programming practice to add comments to a program.
- It helps in readability and debugging of the program.
- There are two ways to write comments in PL/SQL.
  - To write a single-line comment, two dashes (- -) are entered at the beginning of a new line. For example,

```
- -This is a single-line comment.
```

  - To write a multi-line comment, comment text is placed between /* and */.
    - A multi-line can be written on a separate line by itself or can be used on a line of code also.

```
/* This is a
multiline comment
that ends here. */
```

A4

PL/SQL Arithmetic Operators in Describing Order of Precedence

Operator	Description	Example	Result
**	Exponentiation	2 ** 3	8
* /	Multiplication Division	2 * 3 9/2	6 4.5
+ -	Addition Subtraction	3 + 2 3 - 2	5 1
-	Negation	-5	-5

Table 4-5 PL/SQL arithmetic operators in describing order of precedence

Assignment Statements

- Assigns value to variable
- Operator
  - :=
- Syntax
  - *variable_name* := *value*;
- String literal

Slide 21

A4 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

Executing a PL/SQL Program in SQL*Plus

- Create and execute PL/SQL program blocks
  - Within variety of Oracle 10*g* development environments

Displaying PL/SQL Program Output in SQL*Plus

- PL/SQL output buffer
  - Memory area on database server
  - Stores program's output values before they are displayed to user
  - Should increase size
    - `SET SERVEROUTPUT ON SIZE` *`buffer_size`*
  - Default buffer size
    - 2000 bytes

Displaying PL/SQL Program Output in SQL*Plus (continued)

- Display program output
  - `DBMS_OUTPUT.PUT_LINE('display_text');`
  - Display maximum of 255 characters of text data

Writing a PL/SQL Program

- Write PL/SQL program in Notepad or another text editor
- Copy and paste program commands into SQL*Plus
- Press Enter after last program command
- Type front slash (/)
- Then press Enter again

Writing a PL/SQL Program (continued)

- Good programming practice
  - Place DECLARE, BEGIN, and END commands flush with left edge of text editor window
  - Indent commands within each section

A5

PL/SQL Program Commands

```
--PL/SQL program to display the current date
DECLARE
   todays_date DATE;
BEGIN
   todays_date := SYSDATE;
   DBMS_OUTPUT.PUT_LINE('Today''s date is ');
   DBMS_OUTPUT.PUT_LINE(todays_date);
END;
```

Figure 4-3 PL/SQL program commands

Slide 28

A5 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

PL/SQL Data Conversion Functions

- Implicit data conversions
  - Interpreter automatically converts value from one data type to another
  - If PL/SQL interpreter unable to implicitly convert value error occurs
- Explicit data conversions
  - Convert variables to different data types
  - Using data conversion functions

A6

PL/SQL Data Conversion Functions of PL/SQL

Data Conversion Function	Description	Example
TO_CHAR	Converts either a number or a date value to a string using a specific format model	`TO_CHAR(2.98, '$999.99');` `TO_CHAR(SYSDATE, 'MM/DD/YYYY');`
TO_DATE	Converts a string to a date using a specific format model	`TO_DATE('07/14/2003', 'MM/DD/YYYY');`
TO_NUMBER	Converts a string to a number	`TO_NUMBER('2');`

Table 4-6 PL/SQL data conversion functions of PL/SQL

Slide 30

A6 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

Manipulating Character Strings with PL/SQL

- String
  - Character data value
  - Consists of one or more characters
- Concatenating
  - Joining two separate strings
- Parse
  - Separate single string consisting of two data items separated by commas or spaces

Concatenating Character Strings

- Operator
  - ||
- Syntax:
  - `new_string := string1 || string2;`

Removing Blank Leading and Trailing Spaces from Strings

- LTRIM function
  - Remove blank leading spaces
  - *string* :=
    LTRIM(*string_variable_name*);
- RTRIM function
  - Remove blank trailing spaces
  - *string* :=
    RTRIM(*string_variable_name*);

Finding the Length of Character Strings

- LENGTH function syntax
  - *string_length* :=
    LENGTH(*string_variable_name*);

Character String Case Functions

- Modify case of character strings
- Functions and syntax:
  - *string* := UPPER(*string_variable_name*);
  - *string* := LOWER(*string_variable_name*);
  - *string* := INITCAP(*string_variable_name*);

Parsing Character Strings

- INSTR function
  - Searches string for specific substring
  - Syntax:
    - *start_position* := INSTR(*original_string*, *substring*);
- SUBSTR function
  - Extracts specific number of characters from character string
  - Starting at given point

Parsing Character Strings (continued)

- SUBSTR function (continued)
  - Syntax:
    - *extracted_string* := SUBSTR(*string_variable*, *starting_point*, *number_of_characters*);
- Use INSTR to find delimiter

Commenting Code

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

```
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_sal := v_monthly_sal * 12;
END;     -- This is the end of the block
```

Debugging PL/SQL Programs

- Syntax error
  - Occurs when command does not follow guidelines of programming language
  - Generate compiler or interpreter error messages
- Logic error
  - Does not stop program from running
  - Results in incorrect result

A7

Program with a Syntax Error

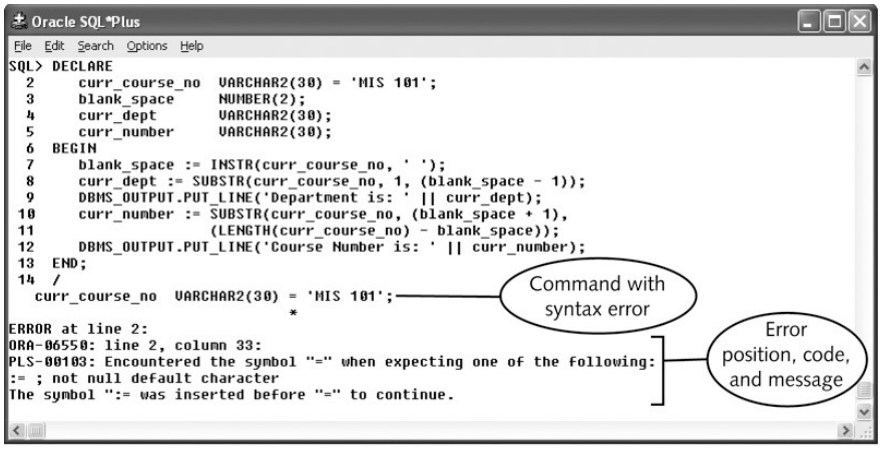


Figure 4-8 Program with a syntax error

Slide 40

A7 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

A8

Program with a Logic Error

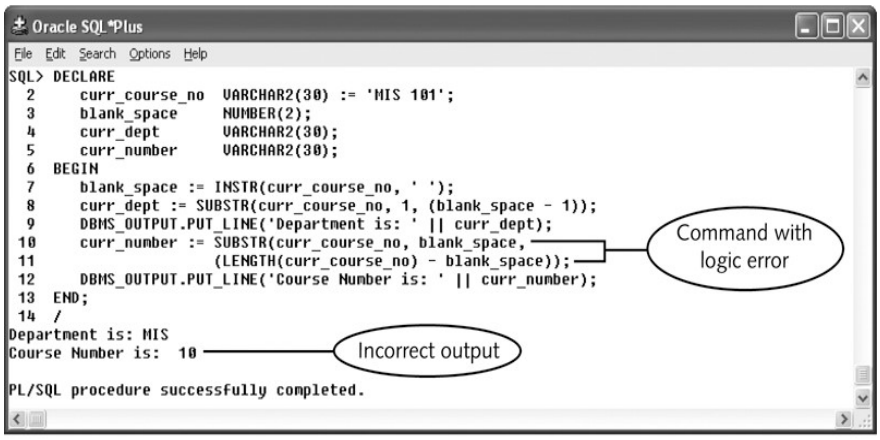


Figure 4-9 Program with a logic error

Finding Syntax Errors

- Often involve:
  - Misspelling reserved word
  - Omitting required character in command
  - Using built-in function improperly
- Interpreter
  - Flags line number and character location of syntax errors
    - May actually be on preceding line
  - Displays error code and message

Slide 41

A8 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

Finding Syntax Errors (continued)

- Comment out program lines
  - To find error
- Cascading errors
  - One syntax error can generate many more errors

Finding Logic Errors

- Caused by:
  - Not using proper order of operations in arithmetic functions
  - Passing incorrect parameter values to built-in functions
  - Creating loops that do not terminate properly
  - Using data values that are out of range or not of right data type

Finding Logic Errors (continued)

- Debugger
  - Program that enables software developers to pause program execution and examine current variable values
  - Best way to find logic errors
  - SQL*Plus environment does not provide PL/SQL debugger
  - Use DBMS_OUTPUT to print variable values

Lesson A Summary

- PL/SQL data types:
  - Scalar
  - Composite
  - Reference
  - LOB
- Program block
  - Declaration
  - Execution
  - Exception

Lesson B

Lesson B Objectives

After completing this lesson, you should be able to:

- Create PL/SQL decision control structures
- Use SQL queries in PL/SQL programs
- Create loops in PL/SQL programs
- Create PL/SQL tables and tables of records
- Use cursors to retrieve database data into PL/SQL programs

Lesson B Objectives (continued)

- Use the exception section to handle errors in PL/SQL programs

PL/SQL BLOCK STRUCTURE

- PL/SQL is a block-structured language.
- A program can be divided into logical blocks.
- The block structure gives modularity to a PL/SQL program, and each object within a block has "scope."
- Blocks are of two types:
  - **An anonymous block** is a block of code without a name.
    - It can be used anywhere in a program and is sent to the server engine for execution at runtime.
  - **A named block** is a block of code that is named.
    - A subprogram is a named block that can be called and can take arguments.
    - A procedure is a subprogram that can perform an action, whereas a function is a subprogram that returns a value.
    - A package is formed from a group of procedures and functions.
    - A trigger is a block that is called implicitly by a DML statement.

Nested Blocks and Variable Scope

Example

```
...
  x  BINARY_INTEGER;
BEGIN
  ...
  DECLARE
    y  NUMBER;
  BEGIN
    ...
  END;
  ...
END;
```

Scope of *x*

Scope of *y*

51

Nested Blocks Example

```
SQL> DECLARE
  v_m VARCHAR2(50) :='Oracle';
  BEGIN
  DECLARE
       v_z VARCHAR2(20):='Second';
  BEGIN
       v_m:=v_m || v_z;
  END;
       v_m:=v_m || ' Exam';
   DBMS_OUTPUT.PUT_LINE(v_m);
 END;
```

Output: Oracle Second Exam

52

PL/SQL Decision Control Structures

- Sequential processing
  - Processes statements one after another
- Decision control structures
  - Alter order in which statements execute
  - Based on values of certain variables

IF/THEN

- Syntax:

```
IF condition THEN
  commands that execute if condition
   is TRUE;
END IF;
```

- Condition
  - Expression evaluates to TRUE or FALSE
  - If TRUE commands execute

A9

PL/SQL Comparison Operators

Operator	Description	Example
=	Equal to	count = 5
<>	Not equal to	count <> 5
!=	Not equal to	count != 5
>	Greater than	count > 5
<	Less than	count < 5
>=	Greater than or equal to	count >= 5
<=	Less than or equal to	count <= 5

Table 4-7 PL/SQL comparison operators

Simple IF Statements

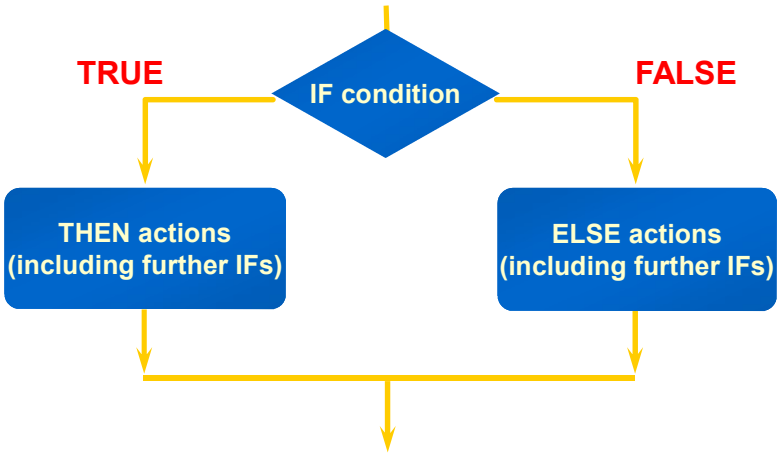
- Set the job title to Salesman, the department number to 35, and the commission to 20% of the current salary if the last name is Miller.
- **Example**

```
. . .
IF v_ename       = 'MILLER' THEN
   v_job        := 'SALESMAN';
   v_deptno     := 35;
   v_new_comm   := sal * 0.20;
END IF;
. . .
```

Slide 55

A9 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

IF-THEN-ELSE Statement Execution Flow



57

IF-THEN-ELSE Statements

- Set a flag for orders where there are fewer than five days between order date and ship date.
- **Example**

```
...
IF v_shipdate - v_orderdate < 5 THEN
   v_ship_flag := 'Acceptable';
ELSE
   v_ship_flag := 'Unacceptable';
END IF;
...
```

58

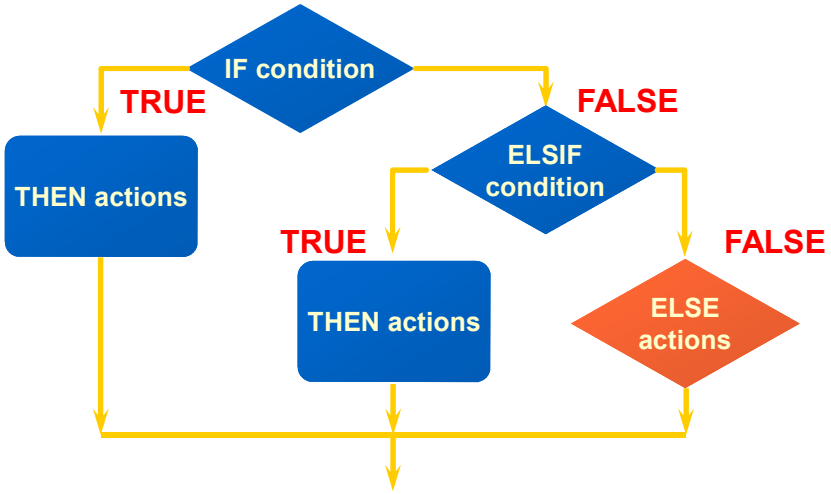
IF/THEN/ELSE

- Syntax:

```
IF condition THEN
  commands that execute if condition
   is TRUE;
ELSE
  commands that execute if condition
   is FALSE;
END IF;
```

- Evaluates ELSE command if condition FALSE

IF-THEN-ELSIF Statement Execution Flow



IF-THEN-ELSIF Statements

- For a given value, calculate a percentage of that value based on a condition.
- Example

```
. . .
IF    v_start > 100 THEN
      v_start := 2 * v_start;
ELSIF v_start >= 50 THEN
      v_start := .5 * v_start;
ELSE
      v_start := .1 * v_start;
END IF;
. . .
```

61

Nested IF/THEN/ELSE

- Placing one or more IF/THEN/ELSE statements within program statements that execute after IF or ELSE command
- Important to properly indent program lines

IF/ELSIF

- Syntax:

```
IF condition1 THEN
     commands that execute if condition1 is
  TRUE;
ELSIF condition2 THEN
     commands that execute if condition2 is
  TRUE;
...
ELSE
     commands that execute if no conditions
  TRUE;
END IF;
```


Logical Operators AND, OR, and NOT

- Create complex expressions for decision control structure condition
- AND
  - Expressions on both sides of operator must be true for combined expression to be TRUE
- OR
  - Expressions on either side of operator must be true for combined expression to be TRUE

Logical Operators AND, OR, and NOT (continued)

- Order of evaluation:
  - NOT
  - AND
  - OR

A10

Evaluating AND and OR in an Expression

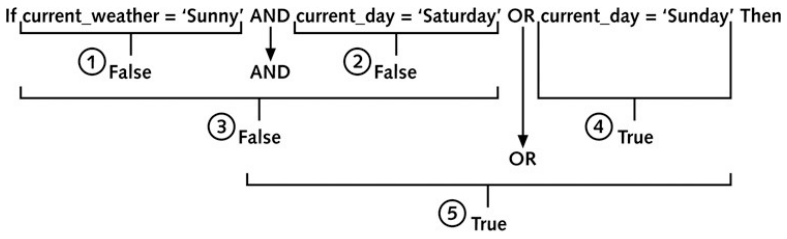


Figure 4-18 Evaluating AND and OR in an expression

Slide 66

A10 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

Using SQL Queries in PL/SQL Programs

- Use SQL action query
  - Put query or command in PL/SQL program
  - Use same syntax as execute query or command in SQL*Plus
  - Can use variables instead of literal values
    - To specify data values

A11

Using SQL Commands in PL/SQL Programs

Category	Purpose	Examples	Can Be Used in PL/SQL Programs
DDL	Creates and modifies database objects	CREATE, ALTER, DROP	No
DML	Manipulates data values in tables	SELECT, INSERT, UPDATE, DELETE	Yes
Transaction Control	Organizes DML commands into logical transactions	COMMIT, ROLLBACK, SAVEPOINT	Yes

Table 4-8 Using SQL commands in PL/SQL programs

Slide 68

A11 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

Comparing SQL and PL/SQL Statement Types

- A PL/SQL block is not a transaction unit. Commits, savepoints, and rollbacks are independent of blocks, but you can issue these commands within a block.
- PL/SQL does not support data definition language (DDL), such as CREATE TABLE, ALTER TABLE, or DROP TABLE.
- PL/SQL does not support data control language (DCL), such as GRANT or REVOKE.

SQL Statements in PL/SQL

- Extract a row of data from the database by using the SELECT command. Only a single set of values can be returned.
- Make changes to rows in the database by using DML commands.
- Control a transaction with the COMMIT, ROLLBACK, or SAVEPOINT command.
- Determine DML outcome with implicit cursors.

SELECT Statements in PL/SQL

- Retrieve data from the database with SELECT.
- Syntax

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
WHERE   condition;
```

71

Loops

- Systematically executes program statements
- Periodically evaluates exit condition to determine if loop should repeat or exit
- Pretest loop
  - Evaluates exit condition before any program commands execute
- Posttest loop
  - Executes program commands before loop evaluates exit condition for first time

Basic Loop

- Syntax

```
LOOP                                    -- delimiter
  statement1;                           -- statements
  . . .
  EXIT [WHEN condition];                -- EXIT statement
END LOOP;                               -- delimiter
```

```
where:  condition        is a Boolean variable or
                         expression (TRUE, FALSE,
                         or NULL);
```

•A basic loop can contain multiple EXIT statements.

73

Basic Loop

- Example

```
DECLARE
  v_ordid   item.ordid%TYPE := 601;
  v_counter       NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO item(ordid, itemid)
    VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
END;
```

74

The LOOP...EXIT Loop

- Pretest or posttest
- Syntax:

```
LOOP
    [program statements]
  IF condition THEN
    EXIT;
  END IF;
  [additional program statements]
END LOOP;
```


The LOOP...EXIT WHEN Loop

- Pretest or posttest
- Syntax:

```
LOOP
      program statements
      EXIT WHEN condition;
END LOOP;
```

The WHILE...LOOP

- Pretest
- Syntax:

```
WHILE condition LOOP
      program statements
END LOOP;
```


WHILE Loop

- Syntax

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Condition is evaluated at the beginning of each iteration.

-

WHILE Loop

- Example

```
SQL>ACCEPT v_dept_name PROMPT 'Enter the dept. name: '
SQL>ACCEPT  num_depts PROMPT 'Enter number of depts: '
SQL>DECLARE
v_count     NUMBER(2) := 1;
BEGIN
  WHILE v_count <= &num_depts LOOP
    INSERT INTO dept(deptno,dname)
    VALUES (v_count, &v_dept_name);
    v_count := v_count + 1;
  END LOOP;
  COMMIT;
END;
/
```

79

The Numeric FOR Loop

- Does not require explicit counter increment
  - Automatically increments counter
- Syntax:

```
FOR counter_variable IN start_value .. end_value
LOOP
     program statements
END LOOP;
```

Cursors

Cursors

- Pointer to memory location on database server
  - DBMS uses to process a SQL query
- Use to:
  - Retrieve and manipulate database data in PL/SQL programs
- Types:
  - Implicit
  - Explicit

Implicit Cursors

- Context area
  - Contains information about query
  - Created by INSERT, UPDATE, DELETE, or SELECT
- Active set
  - Set of data rows that query retrieves
- Implicit cursor
  - Pointer to context area

Implicit Cursors (continued)

- Use to assign output of SELECT query to PL/SQL program variables
  - When query will return only one record

Implicit Cursors (continued)

- Syntax:

```
SELECT field1, field2, ...
INTO variable1, variable2, ...
FROM table1, table2, ...
WHERE join_conditions
AND search_condition_to_retrieve_1_record;
```


Implicit Cursors (continued)

- Useful to use %TYPE reference data type
  - To declare variables used with implicit cursors
- Error "ORA-01422: exact fetch returns more than requested number of rows"
  - Implicit cursor query tried to retrieve multiple records

Explicit Cursors

- Retrieve and display data in PL/SQL programs for query that might
  - Retrieve multiple records
  - Return no records at all
- Steps for creating and using explicit cursor
  - Declare cursor
  - Open cursor
  - Fetch data rows
  - Close cursor

Explicit Cursors (continued)

- Declare explicit cursor syntax:
  - `CURSOR cursor_name IS select_query;`
- Open explicit cursor syntax:
  - `OPEN cursor_name;`

Explicit Cursors (continued)

- Fetch values using LOOP…EXIT WHEN loop:

```
LOOP
    FETCH cursor_name INTO
    variable_name(s);
EXIT WHEN cursor_name%NOTFOUND;
```

- Active set pointer
  - Indicates memory location of next record retrieved from database

Explicit Cursors (continued)

- Close cursor syntax:
  - `CLOSE cursor_name;`

Processing Explicit Cursors Using a LOOP...EXIT WHEN Loop

- Often used to process explicit cursors that retrieve and display database records
- Use %TYPE variable to display explicit cursor values
- Use %ROWTYPE variable to display explicit cursor values

Processing Explicit Cursors Using a Cursor FOR Loop

- Make it easier to process explicit cursors
- Automatically:
  - Opens cursor
  - Fetches records
  - Closes cursor

Processing Explicit Cursors Using a Cursor FOR Loop (continued)

- Syntax:

```
FOR variable_name(s) IN cursor_name
  LOOP
   processing commands
END LOOP;
```


Exception Handling

Handling Runtime Errors in PL/SQL Programs

- Exception handling
  - Programmers place commands for displaying error messages
  - Give users options for fixing errors in program's exception section
- Runtime errors
  - Cause program to fail during execution
- Exception
  - Unwanted event

Handling Runtime Errors in PL/SQL Programs (continued)

- Handle exception options
  - Correct error without notifying user of problem
  - Inform user of error without taking corrective action
- After exception handler executes
  - Program ends

Predefined Exceptions

- Most common errors that occur in programs
- PL/SQL language:
  - Assigns exception name
  - Provides built-in exception handler for each predefined exception
- System automatically displays error message informing user of nature of problem
- Can create exception handlers to display alternate error messages

A12

Common PL/SQL Predefined Exceptions

Oracle Error Code	Exception Name	Description
ORA-00001	DUP_VAL_ON_INDEX	Command violates primary key unique constraint
ORA-01403	NO_DATA_FOUND	Query retrieves no records
ORA-01422	TOO_MANY_ROWS	Query returns more rows than anticipated
ORA-01476	ZERO_DIVIDE	Division by zero
ORA-01722	INVALID_NUMBER	Invalid number conversion (such as trying to convert "2B" to a number)
ORA-06502	VALUE_ERROR	Error in truncation, arithmetic, or data conversion operation

Table 4-10 Common PL/SQL predefined exceptions

Slide 98

A12 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

A14

Exception Handler Syntax

```
EXCEPTION
  WHEN exception1_name THEN
     exception1 handler commands;
  WHEN exception2_name THEN
     exception2 handler commands;
  …
  WHEN OTHERS THEN
     other handler commands;
END;
```

Figure 4-33 Exception handler syntax

Undefined Exceptions

- Less common errors
- Do not have predefined names
- Must explicitly declare exception in program's declaration section
- Associate new exception with specific Oracle error code
- Create exception handler in exception section
  - Using same syntax as for predefined exceptions

Slide 99

A14 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

User-defined Exceptions

- Do not raise Oracle runtime error
- Require exception handling to
  - Enforce business rules
  - Ensure integrity of database

A15

General Syntax for Declaring, Raising, and Handling a User-defined Exception

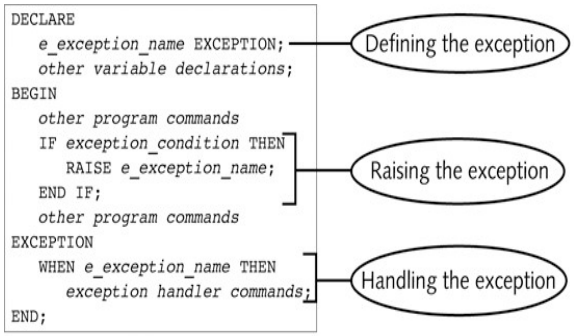


Figure 4-39 General syntax for declaring, raising, and handling a user-defined exception

Slide 102

A15 Okay that slide title is figure caption and not heading?
Author, 8/1/2005

BIND VARIABLES

- When a numeric variable is declared with VARIABLE command, precision and scale values are not used.
- If a VARCHAR2 type variable is declared, length is not used.
- A host variable's value can be printed in the SQL*Plus environment by using the PRINT command.
- The use of host variables should be minimized in a program block, because they affect performance. Every time a host variable is accessed within the block, the PL/SQL engine has to stop to request the host environment for the value of the host variable.
  - The variable's value can be assigned to a local variable in order to minimize calls to the host.

BIND VARIABLES

- An example:

```
SQL> VARIABLE g_double NUMBER
SQL> DECLARE
  2    v_num NUMBER(2);
  3  BEGIN
  4    v_num:=5;
  5    :g_double:=v_num*2;
  6  END;
  7  /

PL/SQL procedure successfully completed.

SQL> PRINT g_double

        G_DOUBLE
        ----------
                10
```

SUBSTITUTION VARIABLES

- PL/SQL does not have any input capabilities in terms of having an input statement.
- There are no explicit I/O statements, but substitution variables of SQL are also available in PL/SQL.
- Substitution variables have limitations, which becomes apparent in a loop.
- The value of bind/host variable is printed with PRINT command.
- The value of the local variable v_num cannot be printed with PRINT command, because the scope of a local variable is within the block where it is declared.
- When substitution variable is used in a program, the output contains lines showing how substitution was done for it.
- You can suppress those lines by using *SET VERIFY OFF* command before running the program.

SUBSTITUTION VARIABLES

- An example:

```
SQL> VARIABLE g_double NUMBER
SQL> DECLARE
  2    v_num NUMBER(2);
  3  BEGIN
  4    v_num:=&p_num;
  5    :g_double:=v_num*2;
  6  END;
  7  /
Enter value for p_num: 10

PL/SQL procedure successfully completed.

SQL> PRINT g_double
      G_DOUBLE
      ----------
              20
```

PRINTING IN PL/SQL

- Another procedure DBMS_OUTPUT.PUT also performs same task of displaying information from buffer, but it does not put a new-line marker at the end.
  - If there is another DBMS_OUTPUT.PUT or DBMS_OUTPUT.PUT_LINE statement following that statement, its output will be displayed on the same line.

```
SQL> VARIABLE num NUMBER
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2    double NUMBER;
  3  BEGIN
  4    :num:=5;
  5    double:=:num*2;
  6      DBMS_OUTPUT.PUT_LINE('Double of' ||
  7        TO_CHAR(:num) || 'is' || TO_CHAR(double));
  8  END;
  9  /
Double of 5 is 10
```

Practice Exercises

- **Create a program script that uses a PL/SQL anonymous block to perform the following:**
  - **Use a host variable AREA to store the result.**
  - **Declare a local variable RADIUS with numeric data type.**
  - **Declare a constant PI with value 3.14.**
  - **Assign a value to the variable RADIUS by using a substitution variable.**
  - **Calculate area of a circle by using formula**

    AREA = PI * RADIUS * RADIUS
  - **Then print result in SQL*Plus.**

Practice Exercises

- **Write a PL/SQL block to find the square, cube, and double of a number inputted with a substitution variable, and print results using built-in package DBMS_OUTPUT.**
- **Write a PL/SQL block to swap the values of two variables.**
  - **Print variables before and after swapping.**
- **Write a PL/SQL program to input hours and rate.**
  - **Find gross pay and net pay.**
  - **The tax rate is 28%.**
  - **Print your results.**
- **Write a PL/SQL program with two variables for the first name and the last name.**
  - **Print the full name with last name and first name separated by comma and a space.**

Lesson B Summary

- Decision control structures:
  - IF/THEN
  - IF/THEN/ELSE
  - IF/ELSIF
- Loop
  - Repeats action multiple times until it reaches exit condition
  - Five types of loops

Lesson B Summary (continued)

- Cursor
  - Pointer to memory location that DBMS uses to process SQL query
  - Types:
    - Implicit
    - Explicit
- Exception handling
  - Three exception types