Introduction to PL/SQL

Guide to Oracle 10*g*

Objectives

After completing this, 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

Introduction

- DB users use GUI applications to enter data into input boxes, and then click buttons to execute programs that perform DB functions.
- We will use the following utilities:
  - Form builder
  - Report builder
- To use these utilities effectively an understanding of PL/SQL is needed.
- PL/SQL is a procedural programming language that Oracle utilities use to manipulate DB data.
- Programming or scripting language concepts understanding is necessary.

Fundamentals of PL/SQL

- PL/SQL program combines SQL queries with procedural commands (if/then, loops).
- It is a full-featured programming language
- Executed using Oracle 10*g* utilities such as:
  - SQL*Plus or
  - Forms Builder
- It is an interpreted language
- Its commands are not a case sensitive. Inside single quotes are case sensitive.. ‘M’ != ‘m’
- Semicolon ends each command
- It contains reserved words and built in functions.

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

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**
- PL/SQL supports: scalar, composite, reference and LOB.

Scalar Variables

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

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;`

Composite Variables

- Data Structure is a data object made up of multiple individual data elements
- Composite variable is a data structure contains multiple scalar variables
- Types:
  - RECORD
  - TABLE
  - VARRAY

Reference Variables

- Reference variables directly reference specific database column or row
- Reference variables 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;

Reference Variables

- The (%TYPE) reference data type specifies a variable that references a single DB field.
  - current_f_last FACULTY.F_LAST%TYPE;

- The current_f_last variable assumes a data type of VARCHAR2(30), because this is the data type of the f_last column in the FACULTY table.

Reference Variables

- The (%ROWTYPE) reference data type creates composite variables that reference the entire data record.
  - `Faculty_row FACULTY%ROWTYPE;`

- The variable faculty_row references all of the columns in the FACULTY table, and each column has the same data type as its associated DB column.

LOB Data Types

- It declares variables that reference binary or character data objects up to 4 GB.

- LOB values in PL/SQL programs must be manipulated using special package called DBMS_LOB.

PL/SQL Program Blocks

- Declaration section (DECLARE)
  - Optional (variable declaration)
- Execution section (BEGIN)
  - Required (Program statements)
- Exception section (EXCEPTION)
  - Optional (Error handling statements)
- Comment statements
  - Enclosed within /* and */
  - Two hyphens (--) for a single line comment
- END;

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

Assignment Statements

- Assigns value to variable
- Operator
  - `:=`
- Syntax
  - *`variable_name := value;`*
- String literal must be enclosed in a single quotation marks.

Assignment Statements

- You can use the := in the declaration section to give initial values.
  - s_id NUMBER := 100;
- You can use DEFAULT instead:
  - s_id NUMBER DEFAULT 100;
- If NULL value is used in an assignment statement with arithmetic operation, the result is always NULL.

Executing a PL/SQL Program in SQL*Plus

- Create and execute PL/SQL program blocks within variety of Oracle 10*g* development environments:
  - SQL *Plus
  - Form Builder
  - Report Builder

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
  - Buffer size should be increased before output is displayed:
    - `SET SERVEROUTPUT ON SIZE` *`buffer_size`*
  - Default buffer size
    - 2000 bytes
  - If the output is more than *buffer_size*, error apperas.

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.
  - More than 255 characters gives an error.
- Example:
  - s_name := ‘Ahmed Mahmoud’;
  - `DBMS_OUTPUT.PUT_LINE(s_name);`

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 (/) (**to instruct the interpreter to execute the code**)
- 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

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;
```

Output: Today's data is
18-OCT-07

PL/SQL procedure successfully completed

- If no output appears use: **SQL>** set serveroutput on

PL/SQL Data Conversion Functions

- **Implicit data conversions**
  - Interpreter automatically converts value from one data type to another (**the command DBMS_OUTPUT.PUT_LINE(todays_date), the interpreter automatically converted the todas_date DATE variable to a text string for output**).
  - If PL/SQL interpreter unable to implicitly convert value error occurs
- **Explicit data conversions**
  - Convert variables to different data types
  - Using built in data conversion functions

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');`

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
- Built in functions to concatenate and parse strings in SQL*Plus are as follows:

Concatenating Character Strings

- Double bar Operator
  - ||
- Syntax:
  - *new_string := string1 || string2;*
- Example:
  - Full_name := 'Qusai' || ' ' || 'Abuein'

- You must convert NUMBER and DATE data type variables (TO_CHAR)into a string before concatenating it.

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*);
- Space is included wherever it appears.
  - DBMS_OUTPUT.PUT_LINE(LENGTH('Today''s date is   '));
  - Gives 20.

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
  - If substring is found, the starting position of it within the original_string is returned as integer value. Otherwise, zero is returned.
  - Syntax:
    - *start_position* := INSTR(*original_string*, *substring*);
  - start_positioin should be integer data type.

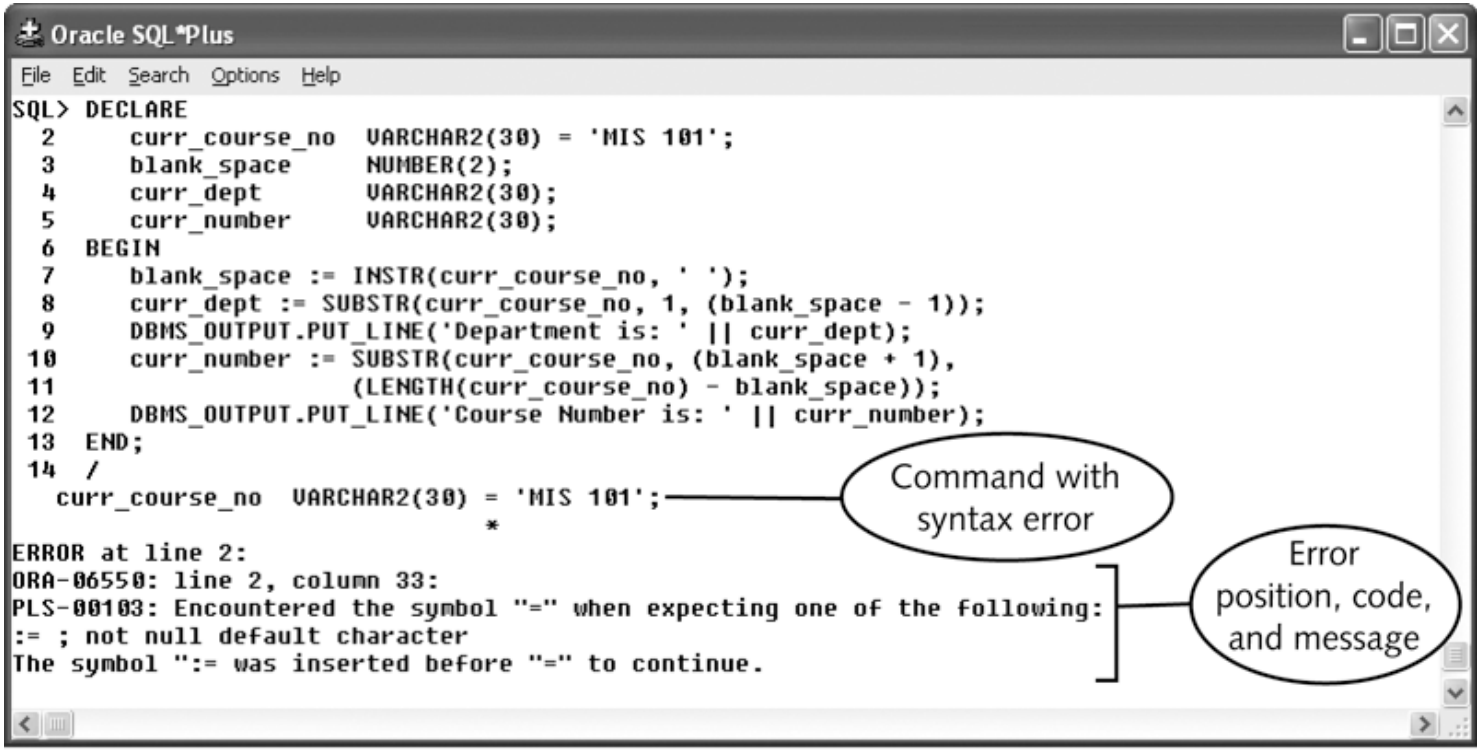
Parsing Character Strings

- SUBSTR function
  - Extracts specific number of characters from character string starting at given point
  - Syntax:
    - `extracted_string := SUBSTR(string_variable, starting_point, number_of_characters);`
- To parse a string, usually INSTR is used to find delimiter and then use the SUBSTR.
  - Example page 206, 207.

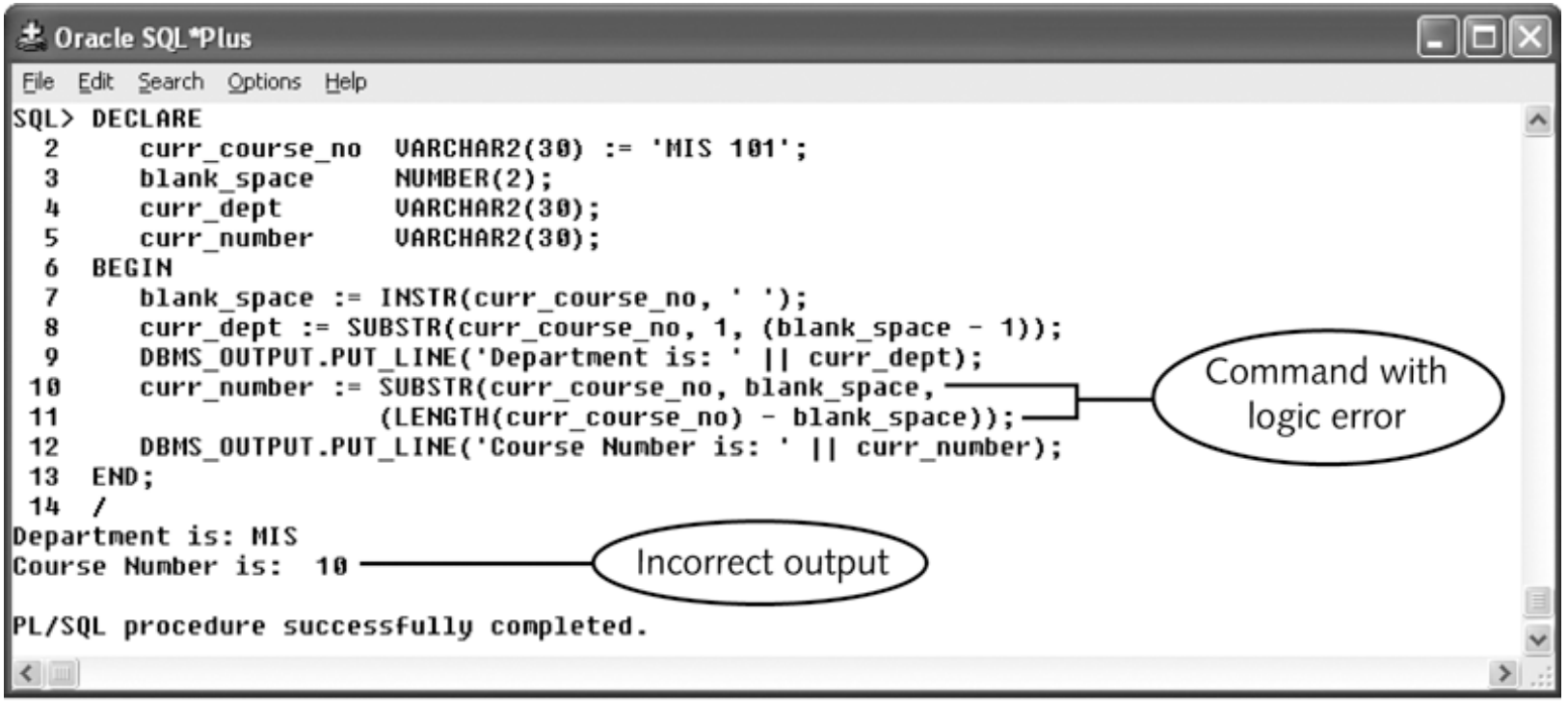
Debugging PL/SQL Programs

- 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

Program with a Syntax Error



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

Finding Syntax Errors (continued)

- Comment out program lines
  - To find error
- Cascading errors are called so when
  - One syntax error can generate many more errors
  - Fix the first error and rerun the program. Do not fix next errors before rerunning the program.

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

Summary

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

Objectives

After completing, 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

Objectives (continued)

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

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
  - See Examples pages 221,222

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

Do not get confused with the equal (=) sign and the assignment (:=) operator

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

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;
```

- Note there is no E fire IF in ELSIF, and no spaces.

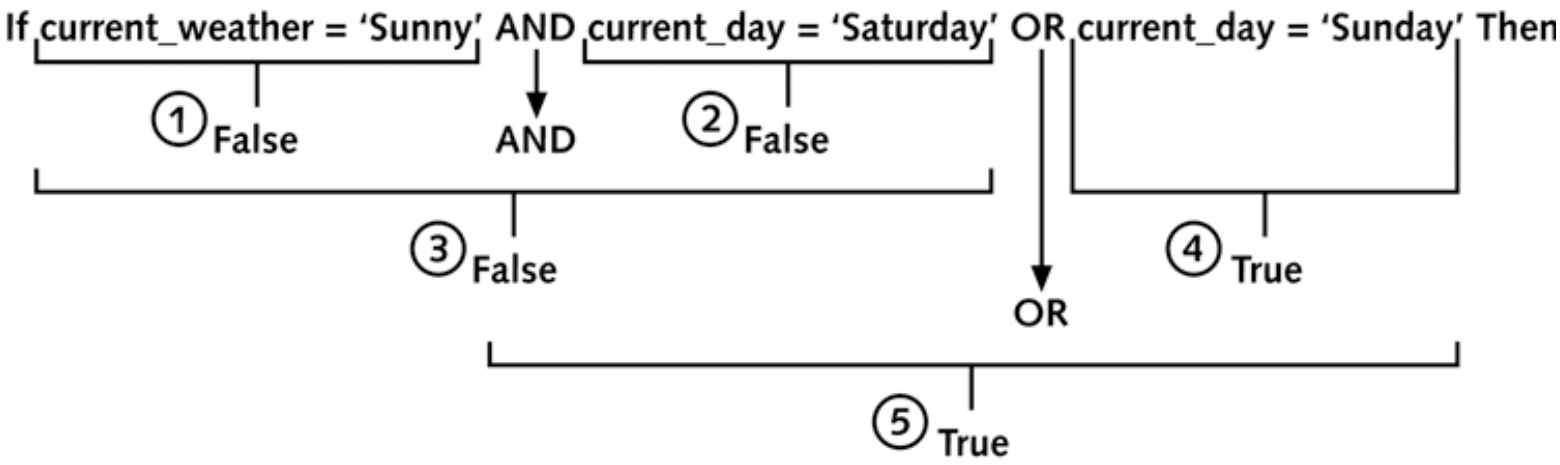
Logical Operators AND, OR, and NOT

- 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 (evaluated first)
  - AND
  - OR

Evaluating AND and OR in an Expression



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
    - Ex: curr_f_name := “Ahmad’
    - We need to insert ‘Ahmad’ into a column S_FIRST
    - INSERT INTO student (s_first)
    - VALUES (curr_f_name);
  - Ex: WHERE s_first = curr_f_name;

PL/SQL Command

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

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
- In PL/SQL there are FIVE loop structures.

The LOOP...EXIT Loop

- This kind of loop can be either Pretest or posttest
- Syntax:

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

The LOOP...EXIT Loop

- LOOP keyword signals the beginning of the loop.
-
- If the IF/THEN decision structure is the first code in the loop, it is a pretest loop.

- If the IF/THEN decision structure is the last code in the loop, it is a posttest loop.

The LOOP...EXIT WHEN Loop

- Can be either a Pretest or posttest
- Syntax:

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

The WHILE...LOOP

- It is a Pretest loop that test the condition before it execute any program statements.
- Syntax:

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

The Numeric FOR Loop

- In the previous LOOP structures we had to increment the counter variable.
- The numeric FOR does not require explicit counter increment
  - Automatically increments counter
- Syntax:

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

- Start_value and end_value must be integer. The increment is always done by 1.

Cursors FOR Loop

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

Implicit Cursors

- Context area
  - Is a memory location on the DB server contains information about query, such as the number of rows the query processed.
  - Created by INSERT, UPDATE, DELETE, or SELECT
- Active set
  - Set of data rows that query retrieves using SELECT
- Implicit cursor
  - Is a Pointer to the context area.
  - Called so because you do not need explicit command to create it.

Implicit Cursors (continued)

- Can be used to assign output of SELECT query to PL/SQL program variables:
  - When query will return only one record
  - If the query returns more than one record or zero records, an error occurs.

Implicit Cursors (continued)

- Syntax:

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

- The value of field1 is assigned to variable1, field2 to variable2 and so on.

- Data types of field1, field2 and variable1, variable2 must be the same.

Implicit Cursors (continued)

- Useful to use %TYPE reference data type
  - To declare variables used with implicit cursors
  - Syntax: var_name tablename.fieldname%TYPE
  - Ex: curr_f_last faculty.f_last%TYPE
  - Ex: Page 239 (fig. 4-26)

- Error “ORA-01422: exact fetch returns more than requested number of rows” . Ex. Page 239 (fig. 4-27).
  - Implicit cursor query tried to retrieve multiple records
- Error “ORA-014032: no data found. Ex. Page 240 (fig.4-28).
- If one of these errors occur, you should modify the query to retrieve only one record, or use Explicit cursor.

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: by declaring an explicit cursor, you create a memory location on the DB server that processes a query and stores the retrieved records .
  - Syntax:
  - `CURSOR cursor_name IS select_query;`
    - *cursor_name*: Any valid PL/SQL name.
    - *select_query*: The query that retrieves the data values.
- Open explicit cursor syntax: by doing so the interpreter checks for syntax error, upon correct it translate the query into machine-language format. The system then creates the memory location that stores the active set (contains the retrieved data using SELECT) (see slide 61) .
  - Syntax
  - `OPEN cursor_name;`

Explicit Cursors (continued)

- Fetch command is used to retrieve the data from the DB into the active set, one row at a time.
- Because a query might return several rows, you execute the FETCH command within a (LOOP…EXIT WHEN) loop:
- Syntax:

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

- Cursor_name is name of the cursor defined previously.
- Variable_name(s) represents either a single variable or a list of variables that receives data from the cursor's SELECT query. (every retrieved column from the DB table is associated with a program variable variable, %TYPE or %ROWTYPE)

Explicit Cursors (continued)

- Active set pointer is created when processing explicit cursor that:
  - Indicates memory location of the next record retrieved from database
  - After the last record of the query is executed, that pointer points to an empty record.
  - The condition (`EXIT WHEN cursor_name%NOTFOUND`) guarantees the exit when the pointer points to an empty record.

Explicit Cursors (continued)

- Close cursor syntax:
  - `CLOSE cursor_name;`

- You should close a cursor after processing all records so as the memory and other resources are available to the system for other tasks.

- The system automatically closes the cursor when the program ends, if you forget to do so.

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 (See example Figure 4-29 page 243)
- Use %ROWTYPE variable to display explicit cursor values (See example Figure 4-30 page 244)

Processing Explicit Cursors Using a Cursor FOR Loop

- Make it easier than LOOP...EXIT WHEN to process explicit cursors.
- For Loop 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;
```

Explicit Cursors Using a Cursor FOR Loop, LOOP...EXIT WHEN

- The cursor variable can be referenced outside the Loop … Exit When. While can not be referenced outside the For Loop.

Handling Runtime Errors in PL/SQL Programs

- Pl/SQL supports Exception handling, that is:
  - Programmers place commands for displaying error messages **and**
  - Give users options for fixing errors in program's exception section (Optional section in a PL/SQL program block. Page 195)
- Runtime errors
  - Cause program to fail during execution
- Exception section is used to handle runtime error.
  - Unwanted event.
  - See Example Figure 4-32. Page 247 (runtime error).

Handling Runtime Errors in PL/SQL Programs (cont.)

- An exception or unwanted event is raised when a runtime error occurs.
- The control is transferred to the exception section, where the exception handler has the following options:
  - Correct error without notifying user of problem
  - Inform user of error without taking corrective action
  - Correct and inform
  - Inform the user and allow him to decide what action to take.
- After exception handler executes
  - Program ends

Handling Runtime Errors in PL/SQL Programs (cont.)

- There are three kinds of exceptions:
  - Predefined
  - Undefined
  - User defined

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 the nature of the problem
- One can create exception handlers to display alternate error messages for predefined exception

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

Exception Handler Syntax

The syntax to write an alternate error messages for predefined handler.

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

Exception Handler Syntax

- *exception_name1*, *exception_name2*,… refer to the name of the predefined exception.
- *Exception handler commands* represent commands that display the error message.
- WHEN OTHERS THEN clause is a handler presents a general message to describe errors that do not have specific exception handlers.

Exception Handler Syntax

- When using alternate error message, such as in the example, which handles only the NO_DATA_FOUND. An unanticipated error might occur. That way no handler exists to handle such unanticipated errors.
- For that reason it is helpful to use the WHEN OTHERS exception handler too, so as to display messages for that situation.
- To do this, use SQLERRM built-in function (SQL ERRor Message).
- That function returns a string that contains the most recent error code and its description.
- First define a varchar2 variable of length 512 to assign it the error message: error_message := SQLERRM;
- .

Undefined Exceptions

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

Syntax of Undefined Exceptions

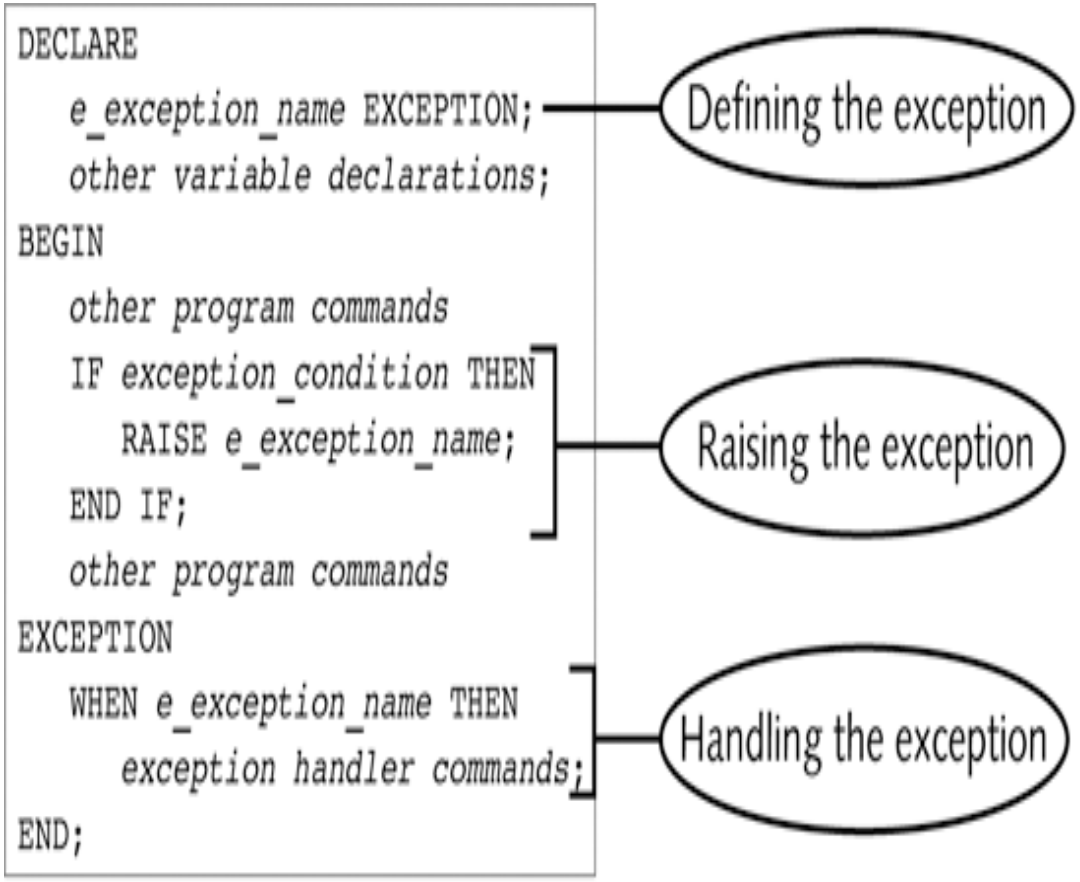
```
DECLARE
    e-exception_name EXCEPTION;
    PRAGMA EXCEPTION INIT(e-exception_name, -Oracle_error_code);
```

- e-exception_name represents name you assign to the exception.
- PRAGMA EXCEPTION INIT: a command associates your exception name with a specific Oracle error code.
- The hyphen (-) must precede the Oracle_error_code.

User-defined Exceptions

- User-defined exceptions do not raise Oracle runtime error. BUT
- Require exception handling to:
  - Enforce business rules (special rules of a company, example page 253. Rule: one can only delete grades with values NULL. Other grades can not be deleted. If a user tries to delete a grade that its value is not NULL, the program should raise an exception and display a message) or
  - Ensure the integrity of database

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



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

- To raise the exception, you must use the IF/THEN, which evaluates the exception condition (grade IS NOT NULL).
- If the condition is TRUE, the RAISE command is executed and raises the exception (transfer execution to the exception section).
- The exception handler executes and displays the error-handling messages.

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

Summary (continued)

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