CHAPTER 10

PL/SQL:

A Programming Language

Lesson A Objectives

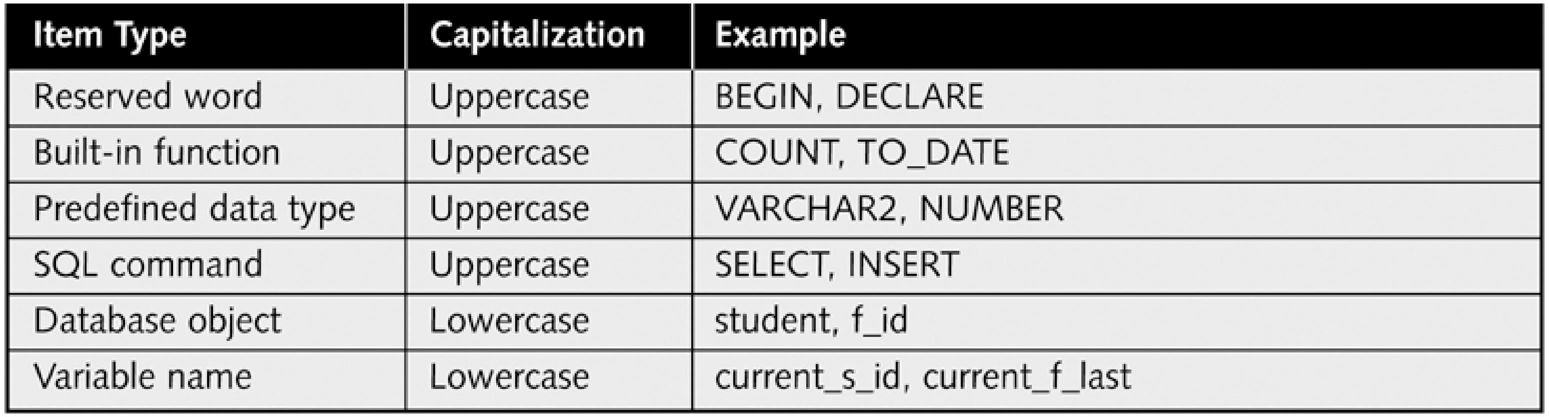
After completing this lesson, you should be able to:

- 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

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

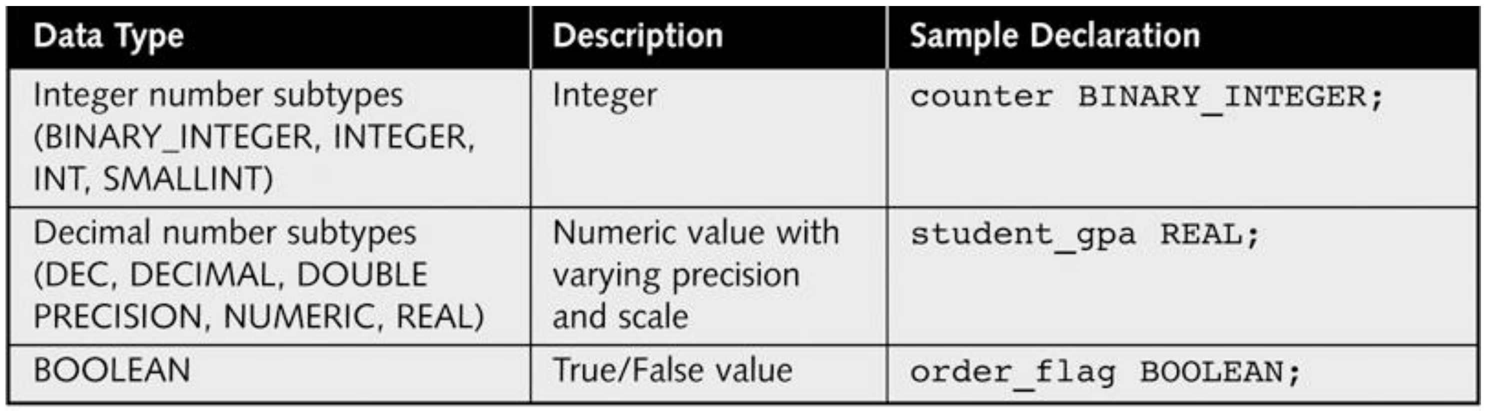
PL/SQL Variables and Data Types

- 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

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;`

Table 4-3 General scalar data types

Composite Variables

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

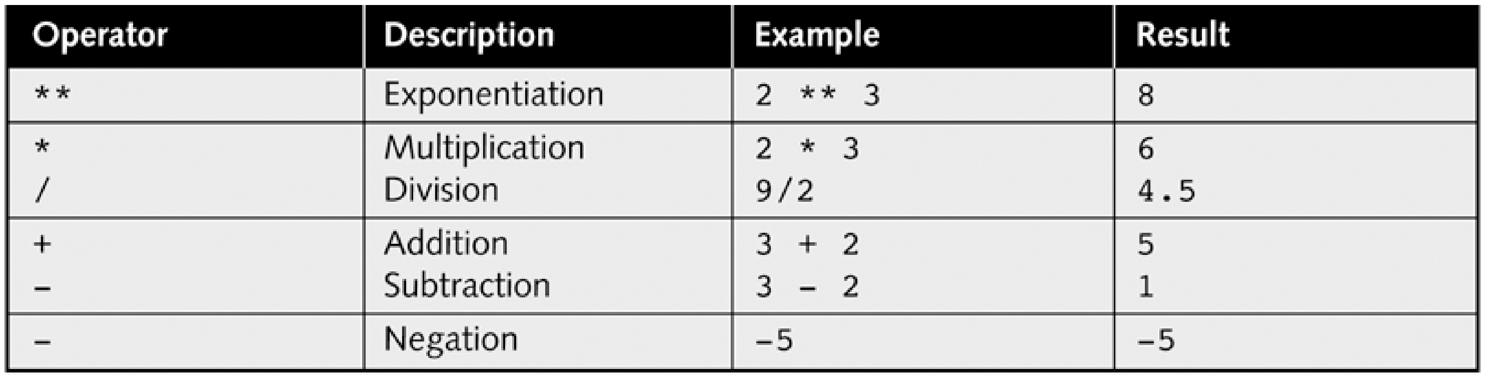
Reference Variables

- 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;`

PL/SQL Program Blocks

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

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

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

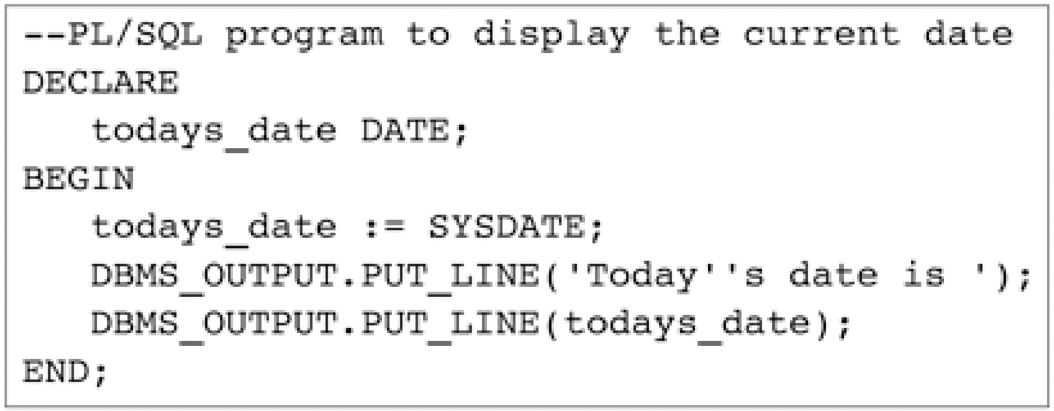
Writing a PL/SQL Program

- 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

PL/SQL Program Commands



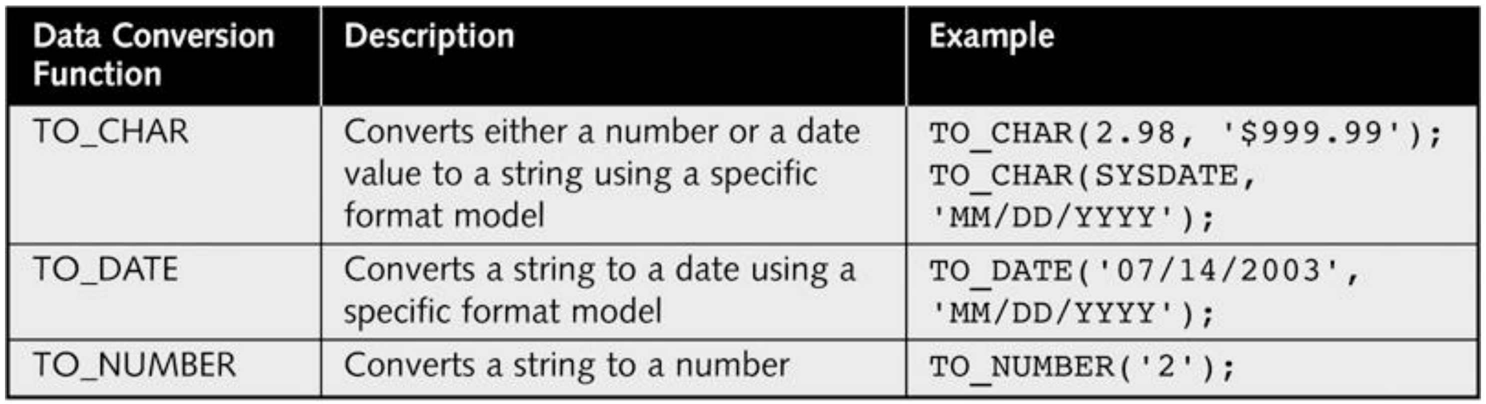
```
--PL/SQL program to display the current date
DECLARE
   todays_date DATE;
BEGIN
   todays_date := SYSDATE;
   DBMS_OUTPUT.PUT_LINE('Today''s date is ');
   DBMS_OUTPUT.PUT_LINE(todays_date);
END;
```

Figure 4-3 PL/SQL program commands

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

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

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

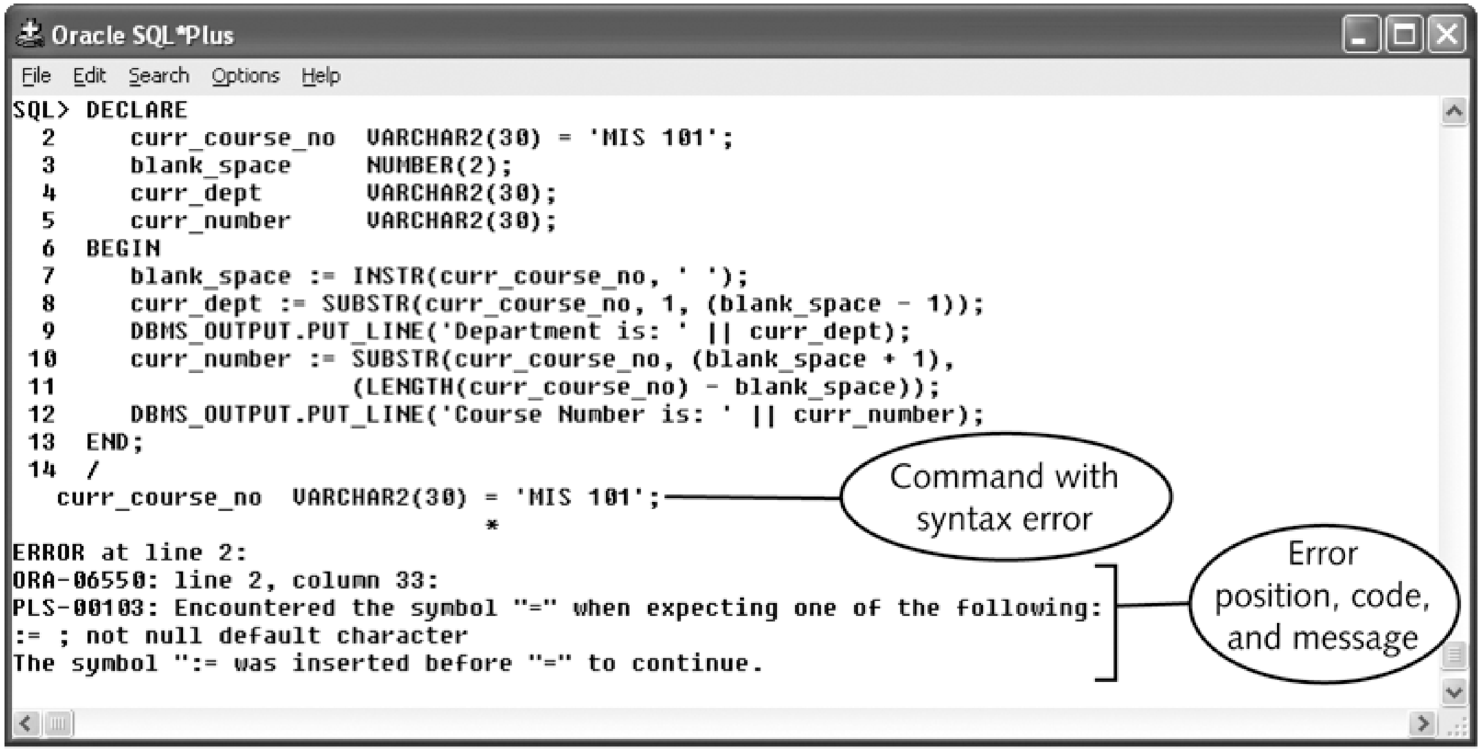
Parsing Character Strings (continued)

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

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

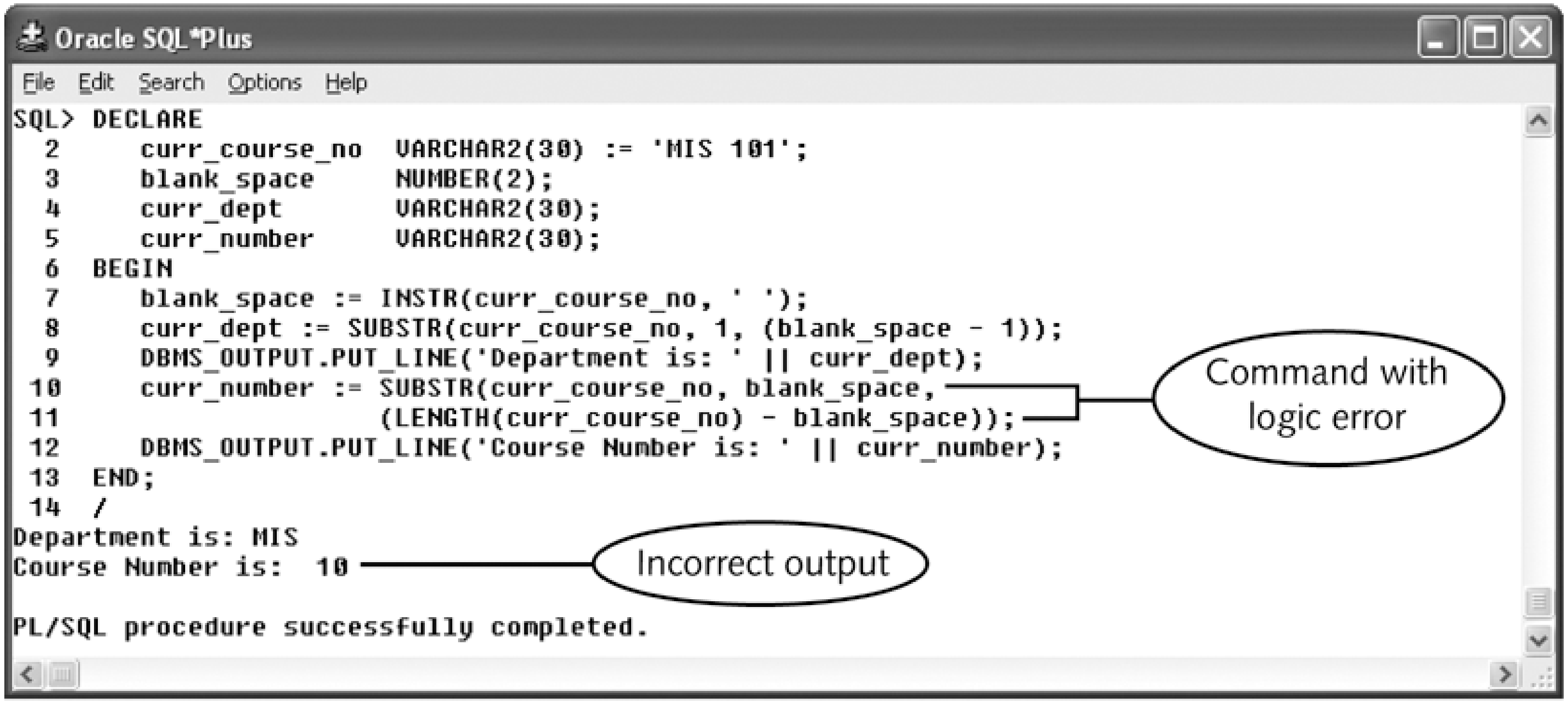
Program with a Syntax Error


Oracle SQL*Plus
File Edit Search Options Help
Command with syntax error
Error position, code, and message

```
SQL> DECLARE
  2      curr_course_no   VARCHAR2(30) = 'MIS 101';
  3      blank_space      NUMBER(2);
  4      curr_dept        VARCHAR2(30);
  5      curr_number      VARCHAR2(30);
  6   BEGIN
  7      blank_space := INSTR(curr_course_no, ' ');
  8      curr_dept := SUBSTR(curr_course_no, 1, (blank_space - 1));
  9      DBMS_OUTPUT.PUT_LINE('Department is: ' || curr_dept);
 10      curr_number := SUBSTR(curr_course_no, (blank_space + 1),
 11                     (LENGTH(curr_course_no) - blank_space));
 12      DBMS_OUTPUT.PUT_LINE('Course Number is: ' || curr_number);
 13   END;
 14   /
    curr_course_no   VARCHAR2(30) = 'MIS 101';
                                  *
ERROR at line 2:
ORA-06550: line 2, column 33:
PLS-00103: Encountered the symbol "=" when expecting one of the following:
:= ; not null default character
The symbol ":= was inserted before "=" to continue.
```

Figure 4-8 Program with a syntax error

Program with a Logic Error



```
SQL> DECLARE
  2     curr_course_no  VARCHAR2(30) := 'MIS 101';
  3     blank_space     NUMBER(2);
  4     curr_dept       VARCHAR2(30);
  5     curr_number     VARCHAR2(30);
  6  BEGIN
  7     blank_space := INSTR(curr_course_no, ' ');
  8     curr_dept := SUBSTR(curr_course_no, 1, (blank_space - 1));
  9     DBMS_OUTPUT.PUT_LINE('Department is: ' || curr_dept);
 10     curr_number := SUBSTR(curr_course_no, blank_space,
 11                    (LENGTH(curr_course_no) - blank_space));
 12     DBMS_OUTPUT.PUT_LINE('Course Number is: ' || curr_number);
 13  END;
 14  /
Department is: MIS
Course Number is:  10

PL/SQL procedure successfully completed.
```

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

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 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 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

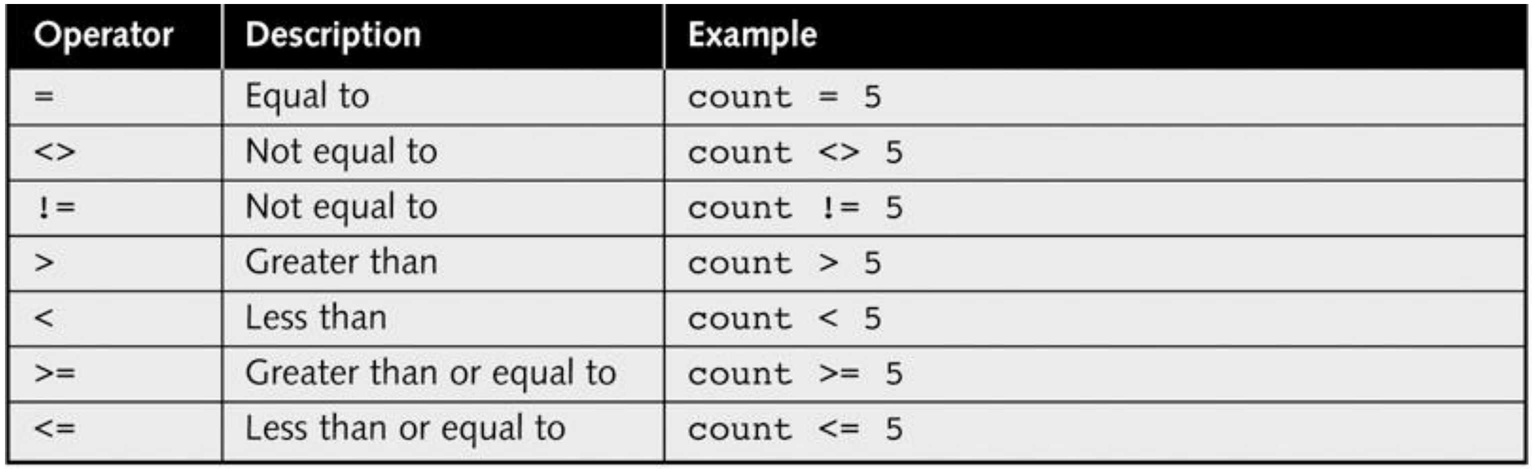
IF/THEN

- Syntax:

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

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

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

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;
```

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

Evaluating AND and OR in an Expression

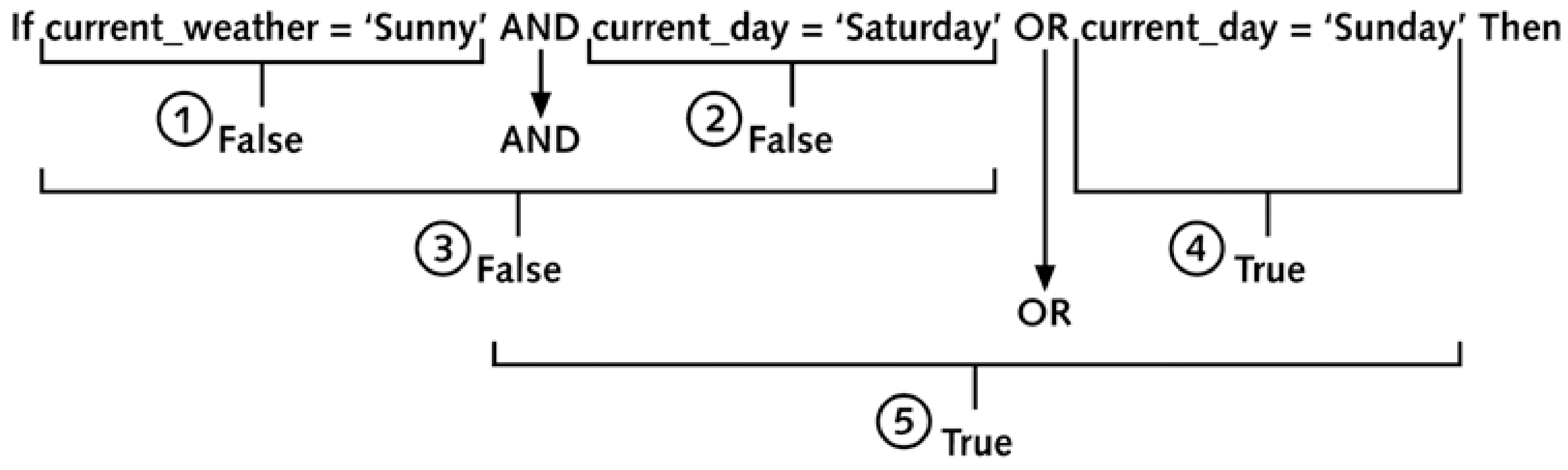
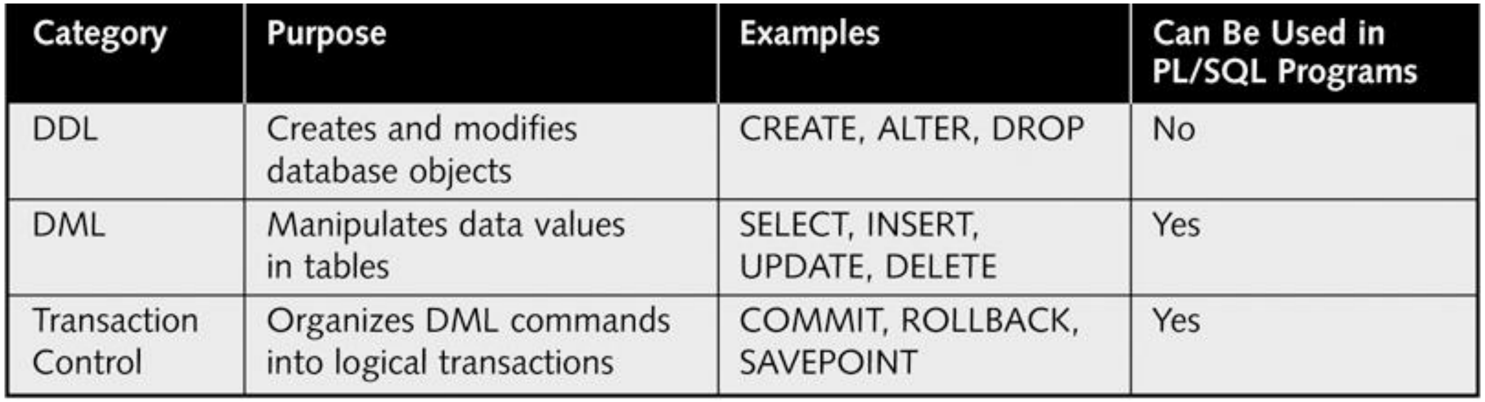

If current_weather = 'Sunny' AND current_day = 'Saturday' OR current_day = 'Sunday' Then
1 False
AND
2 False
3 False
OR
4 True
5 True

Figure 4-18 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

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

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

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;
```

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

- 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;
```

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

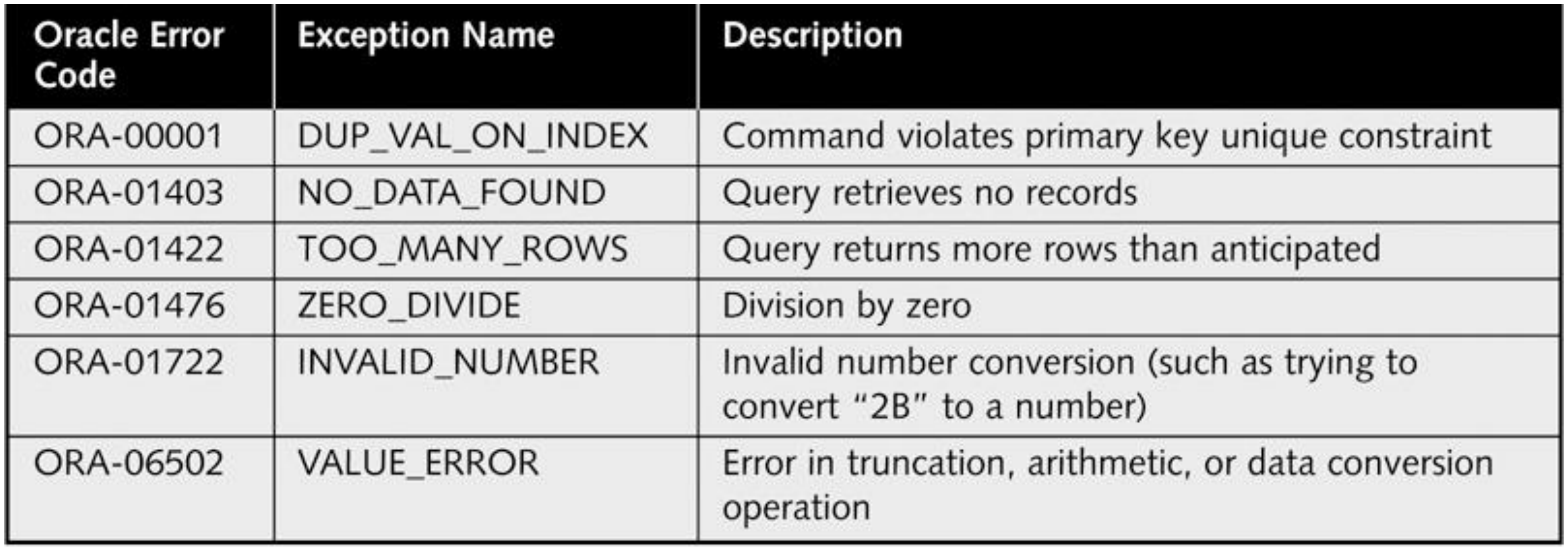
Handling Runtime Errors in PL/SQL Programs (continued)

- 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

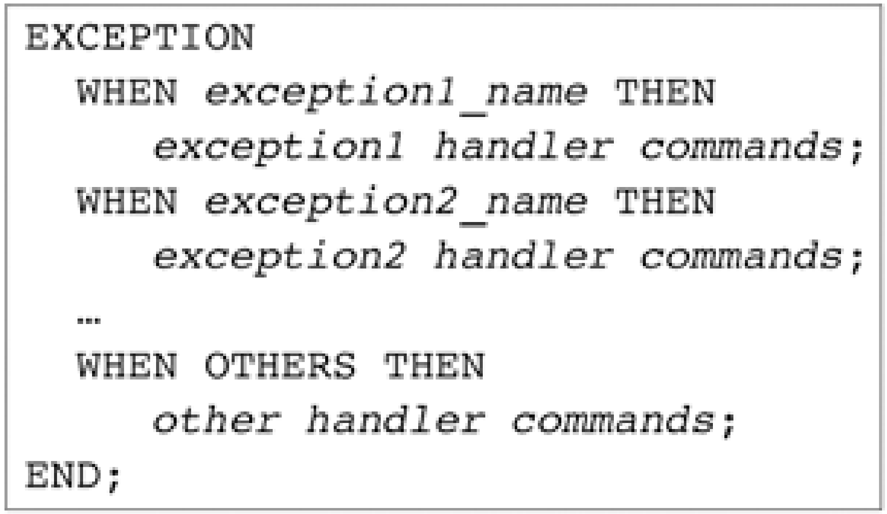
Common PL/SQL Predefined Exceptions



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

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

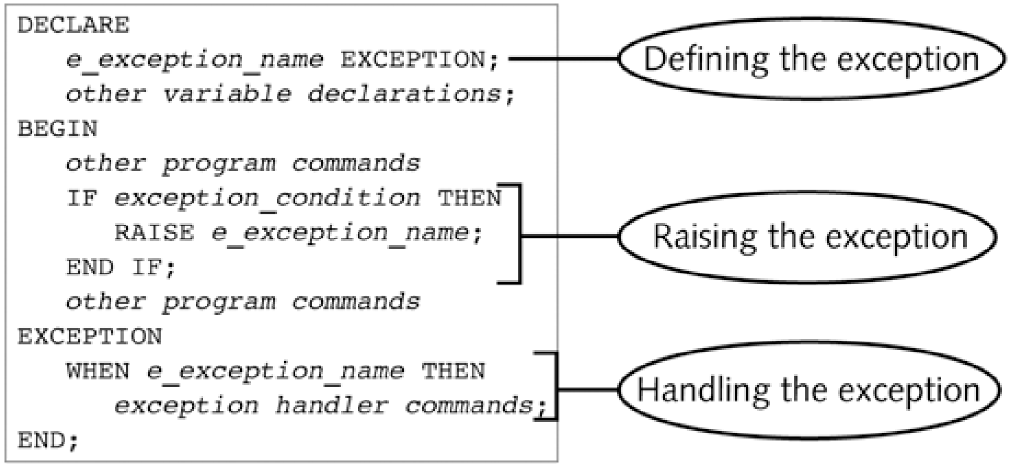
Undefined Exceptions

- 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

User-defined Exceptions

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

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



```
DECLARE
   e_exception_name EXCEPTION;
   other variable declarations;
BEGIN
   other program commands
   IF exception_condition THEN
      RAISE e_exception_name;
   END IF;
   other program commands
EXCEPTION
   WHEN e_exception_name THEN
      exception handler commands;
END;
```

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

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