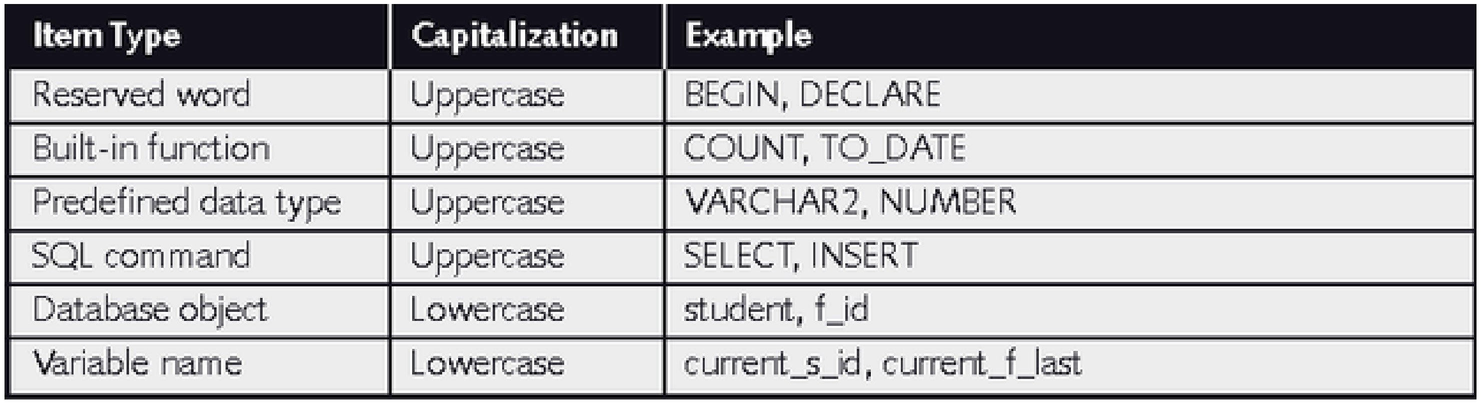
Introduction to PL/SQL

Objectives

- Learn the fundamentals of the PL/SQL programming language
- Write and execute PL/SQL programs in SQL*Plus
- Understand PL/SQL data type conversion functions
- Manipulate character strings in PL/SQL programs
- Learn how to debug PL/SQL programs

Fundamentals of PL/SQL

- Full-featured programming language
- An interpreted language
- Type in editor, execute in SQL*Plus



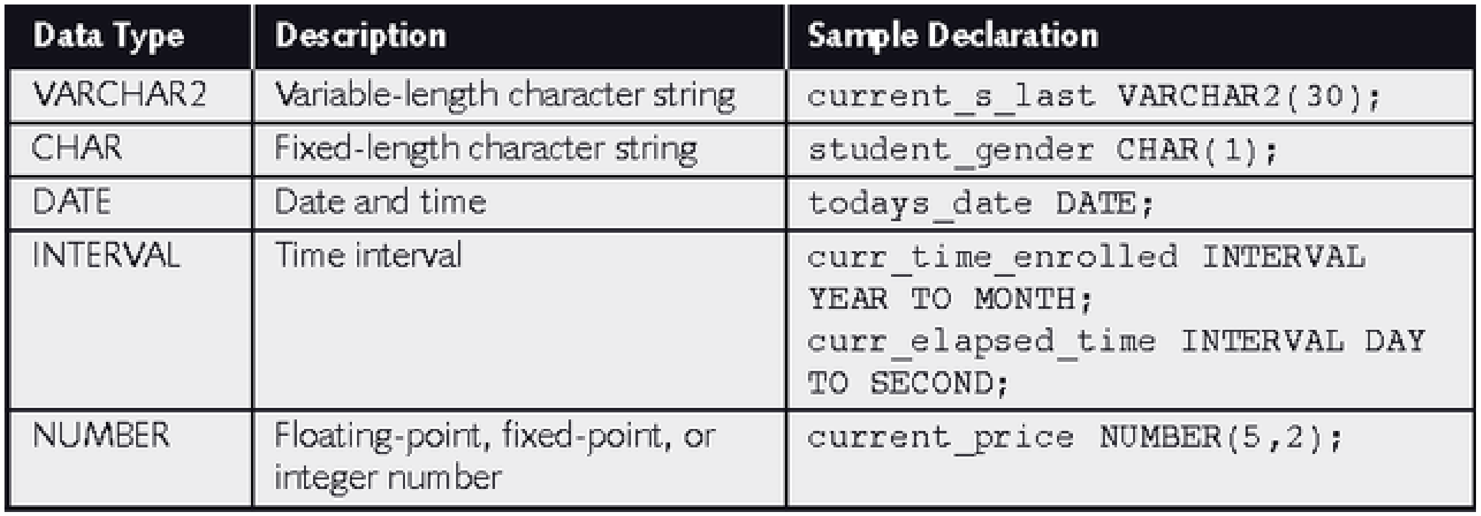
Item Type	Capitalization	Example
Reserved word	Uppercase	BEGIN, DECLARE
Built-in function	Uppercase	COUNT, TO_DATE
Predefined data type	Uppercase	VARCHAR2, NUMBER
SQL command	Uppercase	SELECT, INSERT
Database object	Lowercase	student, f_id
Variable name	Lowercase	current_s_id, current_f_last

Variables and Data Types

- Variables
  - Used to store numbers, character strings, dates, and other data values
  - Avoid using keywords, table names and column names as variable names
  - Must be declared with data type before use: *variable_name data_type_declaration*;

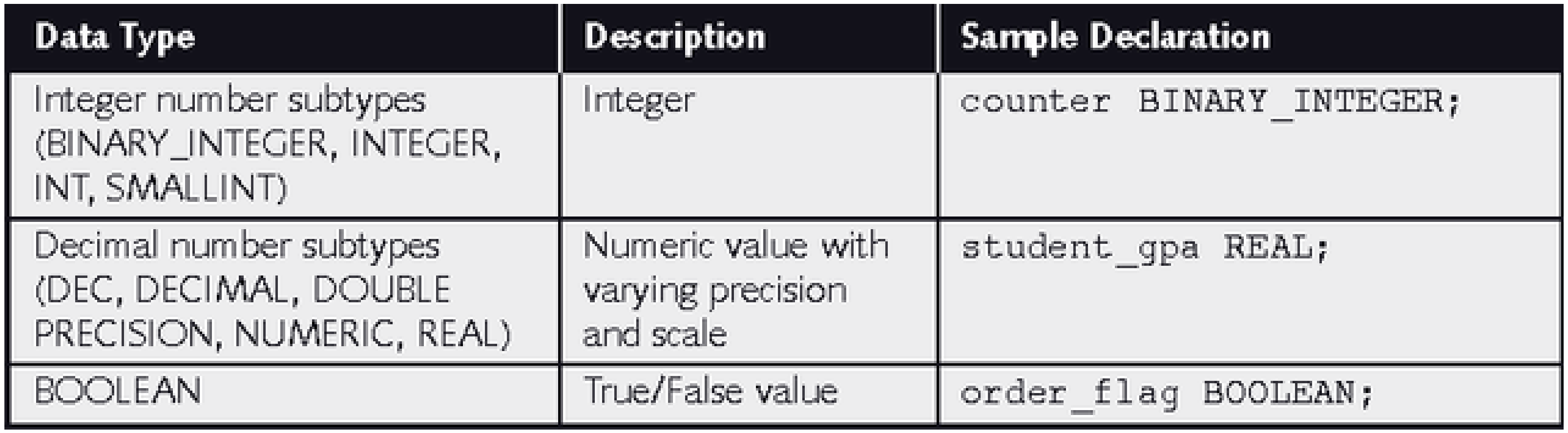
Scalar Data Types

- Represent a single value



Data Type	Description	Sample Declaration
VARCHAR2	Variable-length character string	`current_s_last VARCHAR2(30);`
CHAR	Fixed-length character string	`student_gender CHAR(1);`
DATE	Date and time	`todays_date DATE;`
INTERVAL	Time interval	`curr_time_enrolled INTERVAL YEAR TO MONTH;` `curr_elapsed_time INTERVAL DAY TO SECOND;`
NUMBER	Floating-point, fixed-point, or integer number	`current_price NUMBER(5,2);`

Scalar Data Types

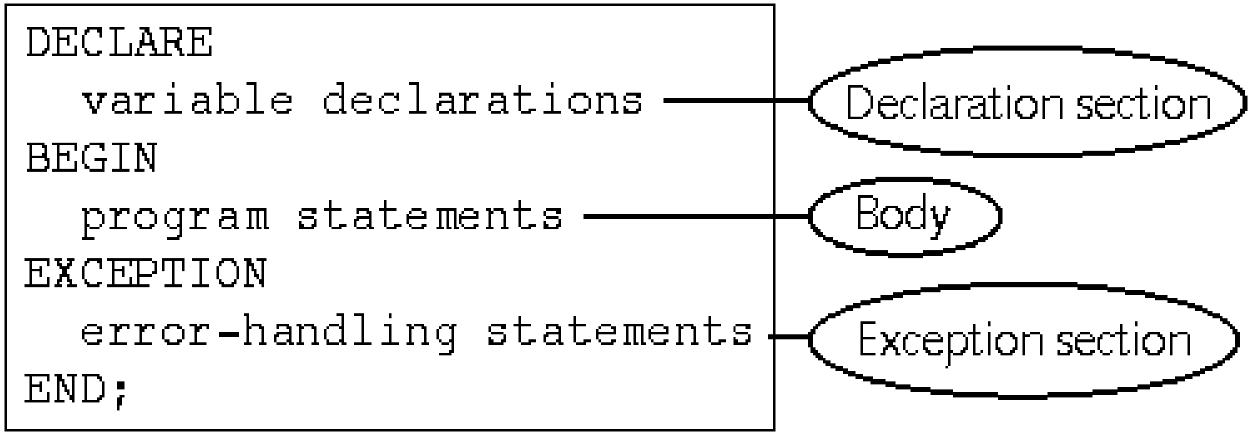


Data Type	Description	Sample Declaration
Integer number subtypes (BINARY_INTEGER, INTEGER, INT, SMALLINT)	Integer	`counter BINARY_INTEGER;`
Decimal number subtypes (DEC, DECIMAL, DOUBLE PRECISION, NUMERIC, REAL)	Numeric value with varying precision and scale	`student_gpa REAL;`
BOOLEAN	True/False value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

Composite and Reference Variables

- Composite variables
  - RECORD: contains multiple scalar values, similar to a table record
  - TABLE: tabular structure with multiple columns and rows
  - VARRAY: variable-sized array
- Reference variables
  - Directly reference a specific database field or record and assume the data type of the associated field or record
  - %TYPE: same data type as a database field
  - %ROWTYPE: same data type as a database record

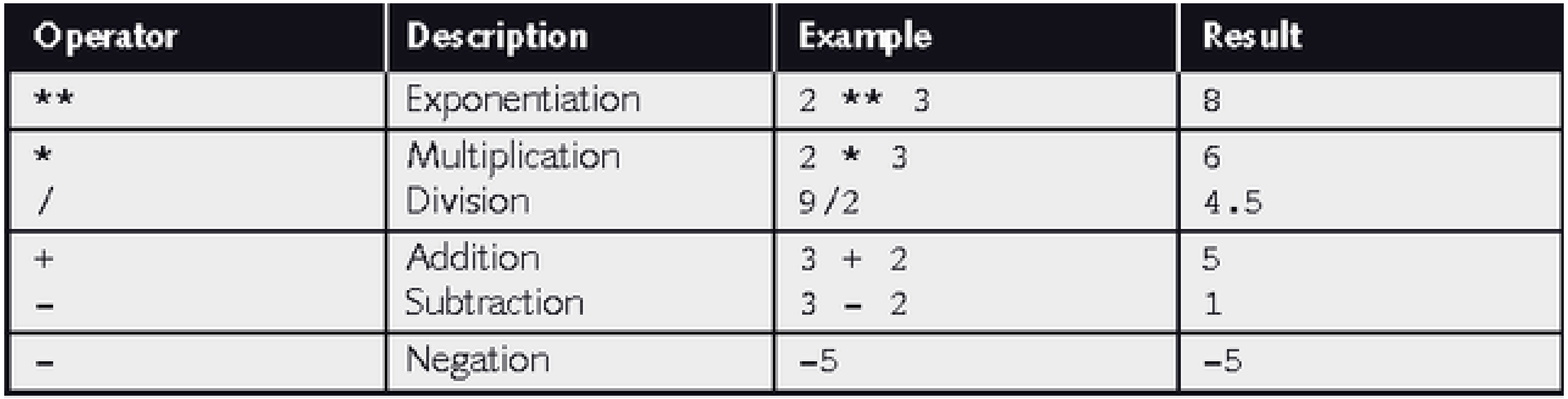
PL/SQL Program Blocks



```
DECLARE
  variable declarations
BEGIN
  program statements
EXCEPTION
  error-handling statements
END;
```

- Comments:
  - Not executed by interpreter
  - Enclosed between /* and */
  - On one line beginning with --

Arithmetic Operators



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

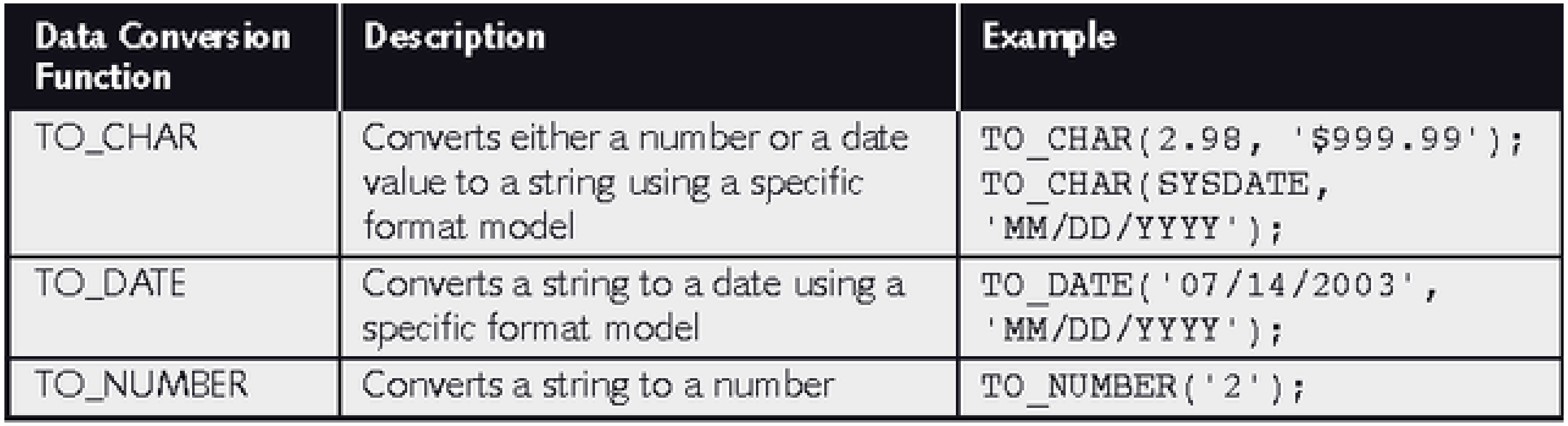
- Assigns a value to a variable
- *variable_name* := *value*;
- Value can be a literal:
  - current_s_first_name := 'John';
- Value can be another variable:
  - current_s_first_name := s_first_name;

Executing a PL/SQL Program in SQL*Plus

```
--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;
```

- Create program in text editor
- Paste into SQL*Plus window
- Press Enter, type / then enter to execute

PL/SQL Data Conversion Functions



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

- To concatenate two strings in PL/SQL, you use the double bar (||) operator:
  - *new_string* := *string1* || *string2*;
- To remove blank leading spaces use the LTRIM function:
  - *string := LTRIM(string_variable_name);*
- To remove blank trailing spaces use the RTRIM function:
  - *string := RTRIM(string_variable_name);*
- To find the number of characters in a character string use the LENGTH function:
  - *string_length := LENGTH(string_variable_name);*

Manipulating Character Strings with PL/SQL

- To change case, use UPPER, LOWER, INITCAP
- INSTR function searches a string for a specific substring:
  - *start_position := INSTR(original_string, substring);*
- SUBSTR function extracts a specific number of characters from a character string, starting at a given point:
  - *extracted_string := SUBSTR(string_variable, starting_point, number_of_characters);*

Debugging PL/SQL Programs

- Syntax error:
  - Command does not follow the guidelines of the programming language
  - Generates compiler or interpreter error messages
- Logic error:
  - Program runs but results in an incorrect result
  - Caused by mistake in program

Finding and Fixing Syntax Errors

- Interpreter flags the line number and character location of syntax errors
- If error message appears and the flagged line appears correct, the error usually occurs on program lines *preceding* the flagged line
- Comment out program lines to look for hidden errors
- One error (such as missing semicolon) may cause more – fix one error at a time

Finding and Fixing Logic Errors

- Locate logic errors by viewing variable values during program execution
- There is no SQL*Plus debugger
- Use DBMS_OUTPUT statements to print variable values

Lesson B Objectives

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

PL/SQL Decision Control Structures

- Use IF/THEN structure to execute code if condition is true
  - IF *condition* THEN

      commands that execute if condition is TRUE;

    END IF;
- If condition evaluates to NULL it is considered false
- Use IF/THEN/ELSE to execute code if condition is true or false
  - IF *condition* THEN

      commands that execute if condition is TRUE;

    ELSE

      commands that execute if condition is FALSE;

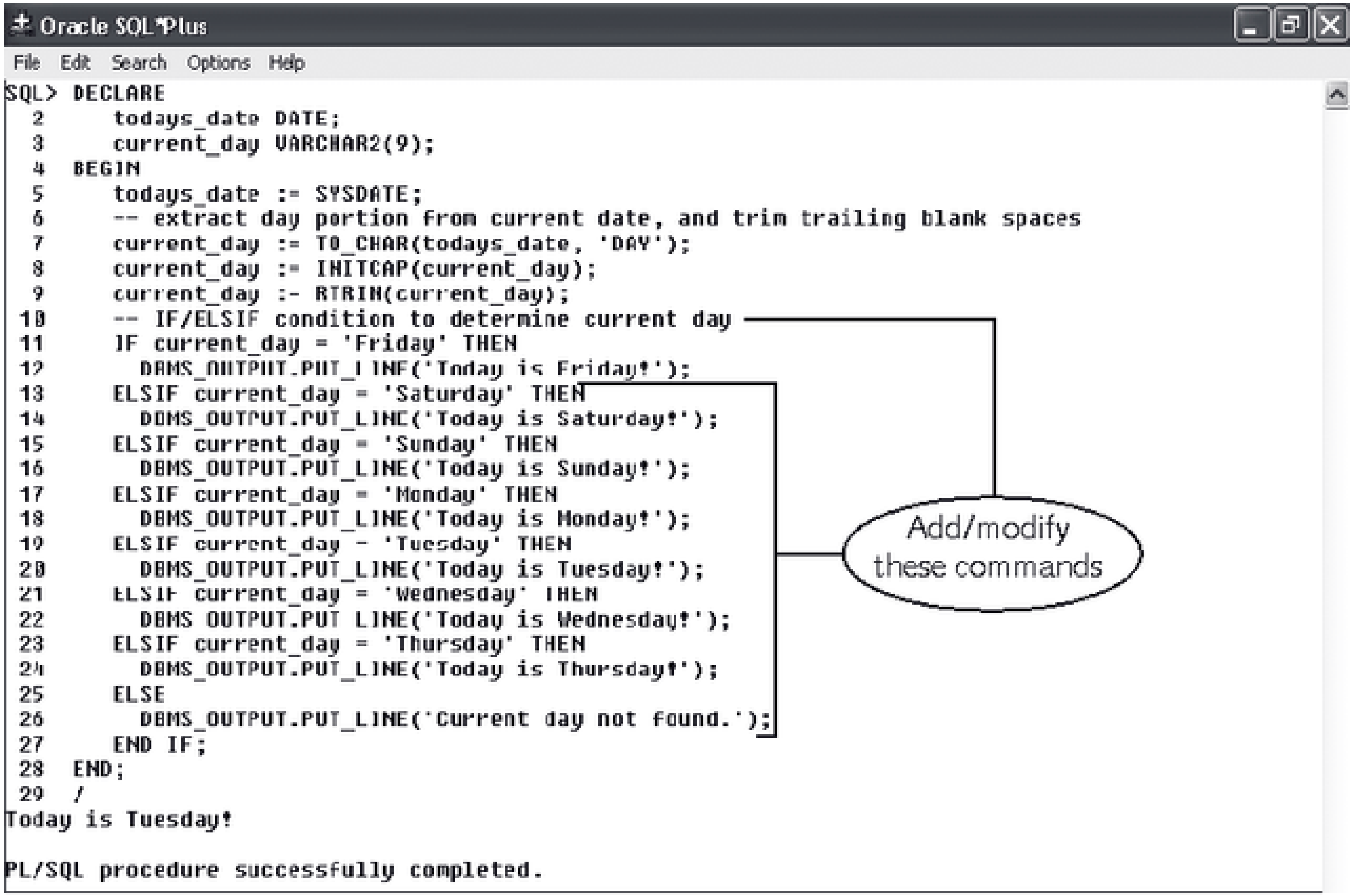
    END IF;
- Can be nested – be sure to end nested statements

PL/SQL Decision Control Structures

- Use IF/ELSIF to evaluate many conditions:
  - IF *condition1* THEN

```
    commands that execute if condition1 is TRUE;
ELSIF condition2 THEN
    commands that execute if condition2 is TRUE;
ELSIF condition3 THEN
    commands that execute if condition3 is TRUE;
...
ELSE
    commands that execute if none of the
    conditions are TRUE;
END IF;
```

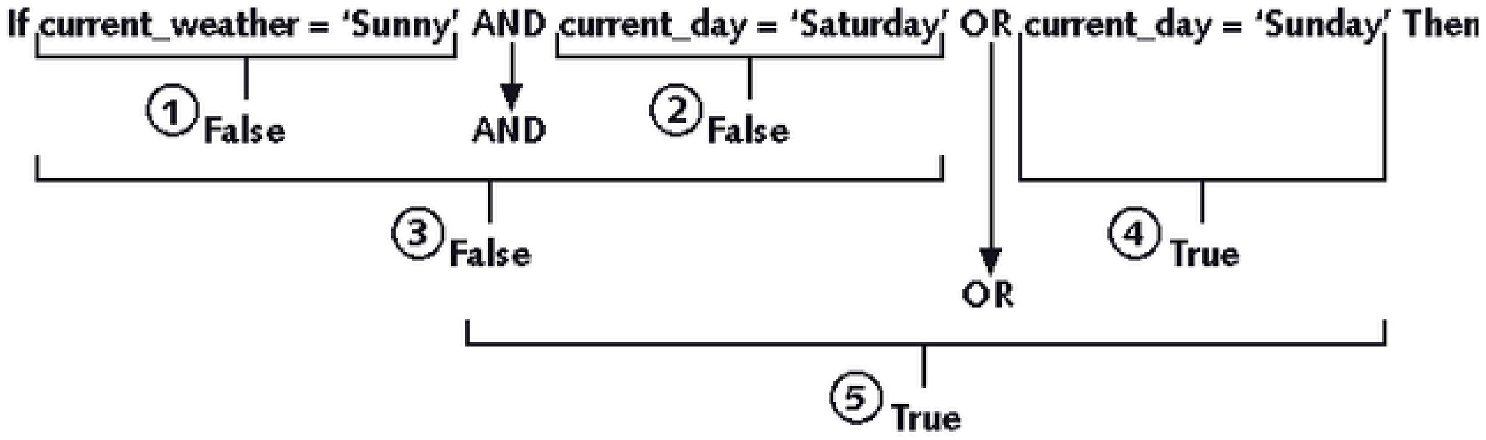
IF/ELSIF Example



```
SQL> DECLARE
      todays_date DATE;
      current_day VARCHAR2(9);
    BEGIN
      todays_date := SYSDATE;
      -- extract day portion from current date, and trim trailing blank spaces
      current_day := TO_CHAR(todays_date, 'DAY');
      current_day := INITCAP(current_day);
      current_day := RTRIM(current_day);
      -- IF/ELSIF condition to determine current day
      IF current_day = 'Friday' THEN
        DBMS_OUTPUT.PUT_LINE('Today is Friday!');
      ELSIF current_day = 'Saturday' THEN
        DBMS_OUTPUT.PUT_LINE('Today is Saturday!');
      ELSIF current_day = 'Sunday' THEN
        DBMS_OUTPUT.PUT_LINE('Today is Sunday!');
      ELSIF current_day = 'Monday' THEN
        DBMS_OUTPUT.PUT_LINE('Today is Monday!');
      ELSIF current_day = 'Tuesday' THEN
        DBMS_OUTPUT.PUT_LINE('Today is Tuesday!');
      ELSIF current_day = 'Wednesday' THEN
        DBMS_OUTPUT.PUT_LINE('Today is Wednesday!');
      ELSIF current_day = 'Thursday' THEN
        DBMS_OUTPUT.PUT_LINE('Today is Thursday!');
      ELSE
        DBMS_OUTPUT.PUT_LINE('Current day not found.');
      END IF;
    END;
    /
Today is Tuesday!

PL/SQL procedure successfully completed.
```

Complex Conditions

- Created with logical operators AND, OR and NOT
- AND is evaluated before OR
- Use () to set precedence


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

Using SQL Queries in PL/SQL Programs

- Action queries can be used as in SQL*Plus
- May use variables in action queries
- DDL commands may not be used in PL/SQL

Loops

- Program structure that executes a series of program statements, and periodically evaluates an exit condition to determine if the loop should repeat or exit
- Pretest loop: evaluates the exit condition before any program commands execute
- Posttest loop: executes one or more program commands before the loop evaluates the exit condition for the first time
- PL/SQL has 5 loop structures

The LOOP...EXIT Loop

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

The LOOP...EXIT WHEN Loop

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

The WHILE...LOOP

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

The Numeric FOR Loop

FOR *counter_variable* IN *start_value* .. *end_value*

LOOP

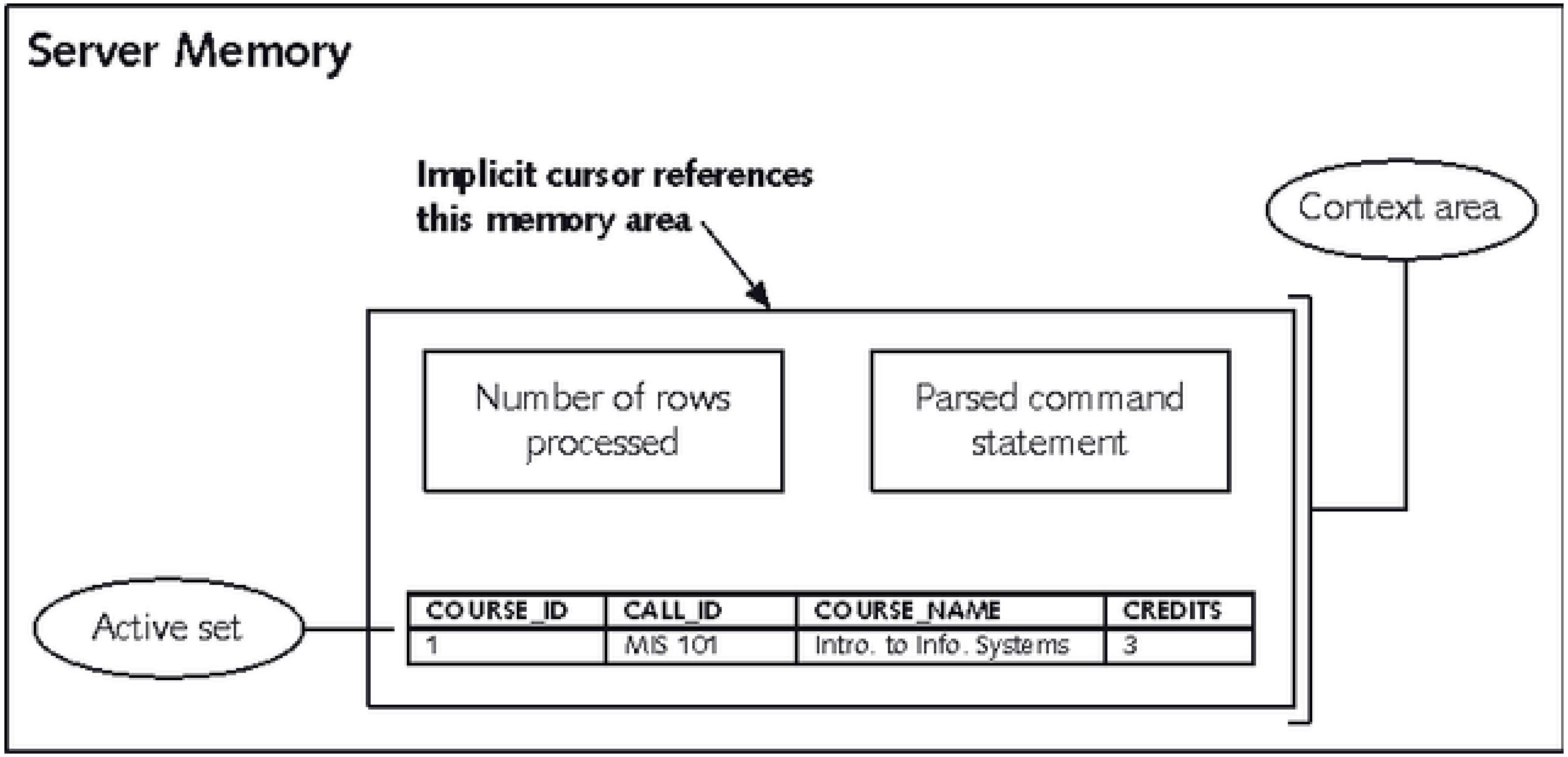
    program statements

END LOOP;

Cursors

- Pointer to a memory location that the DBMS uses to process a SQL query
- Use to retrieve and manipulate database data

Implicit Cursor



COURSE_ID	CALL_ID	COURSE_NAME	CREDITS
1	MIS 101	Intro. to Info. Systems	3

Using an Implicit Cursor

- Executing a SELECT query creates an implicit cursor
- To retrieve it into a variable use INTO:
  - SELECT *field1*, *field2*, ...

    INTO *variable1*, *variable2*, ...

    FROM *table1*, *table2*, ...

    WHERE *join_ conditions*

    AND *search_condition_to_retrieve_1_record*;
- Can only be used with queries that return exactly one record

Explicit Cursor

- Use for queries that return multiple records or no records
- Must be explicitly declared and used

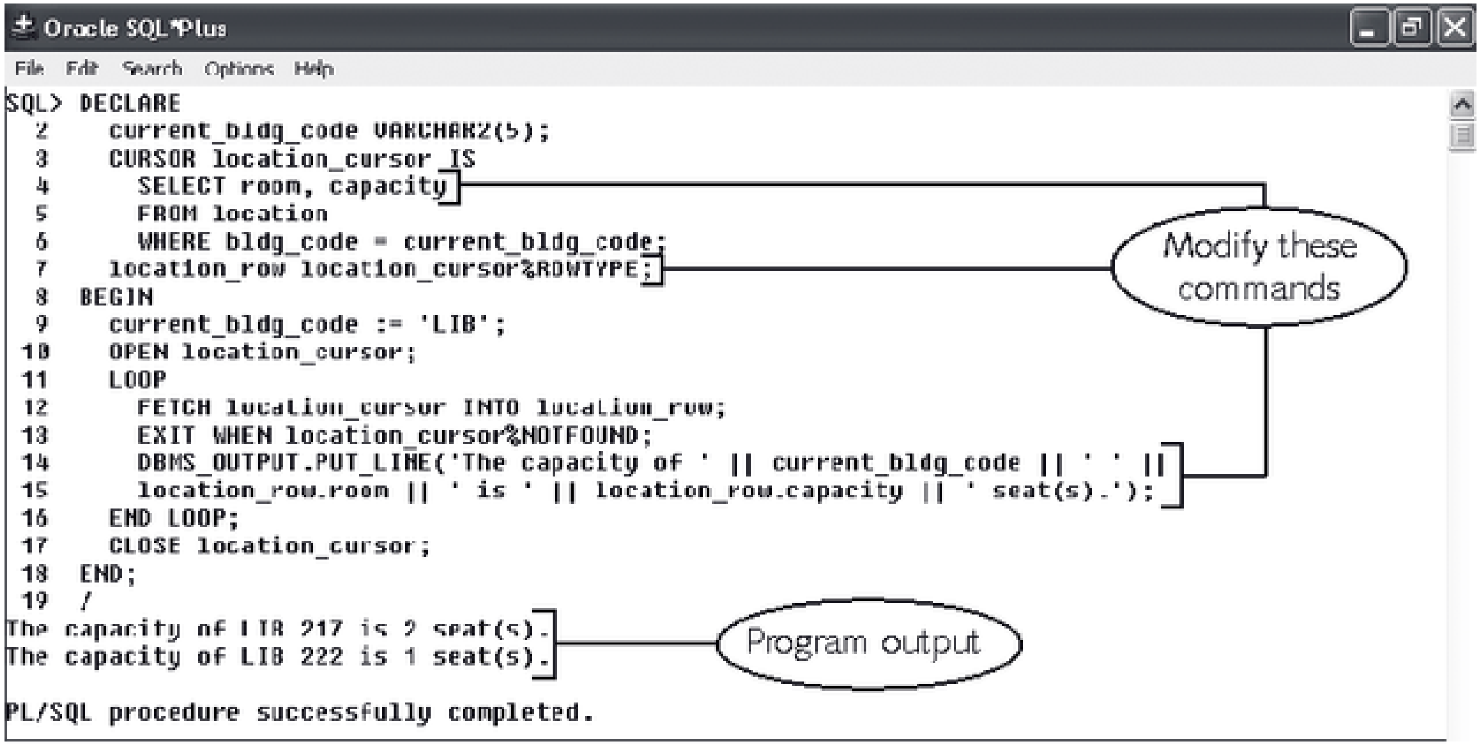
Using an Explicit Cursor

- Declare the cursor
  - *CURSOR cursor_name IS select_query;*
- Open the cursor
  - *OPEN cursor_name;*
- Fetch the data rows
  - *LOOP*

    FETCH cursor_name INTO variable_name(s);

    EXIT WHEN cursor_name%NOTFOUND;
- Close the cursor
  - *CLOSE cursor_name;*

Explicit Cursor with %ROWTYPE



```
SQL> DECLARE
       current_bldg_code VARCHAR2(5);
       CURSOR location_cursor IS
         SELECT room, capacity
         FROM location
         WHERE bldg_code = current_bldg_code;
       location_row location_cursor%ROWTYPE;
     BEGIN
       current_bldg_code := 'LIB';
       OPEN location_cursor;
       LOOP
         FETCH location_cursor INTO location_row;
         EXIT WHEN location_cursor%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('The capacity of ' || current_bldg_code || ' ' ||
         location_row.room || ' is ' || location_row.capacity || ' seat(s).');
       END LOOP;
       CLOSE location_cursor;
     END;
     /
The capacity of LIB 217 is 2 seat(s).
The capacity of LIB 222 is 1 seat(s).

PL/SQL procedure successfully completed.
```

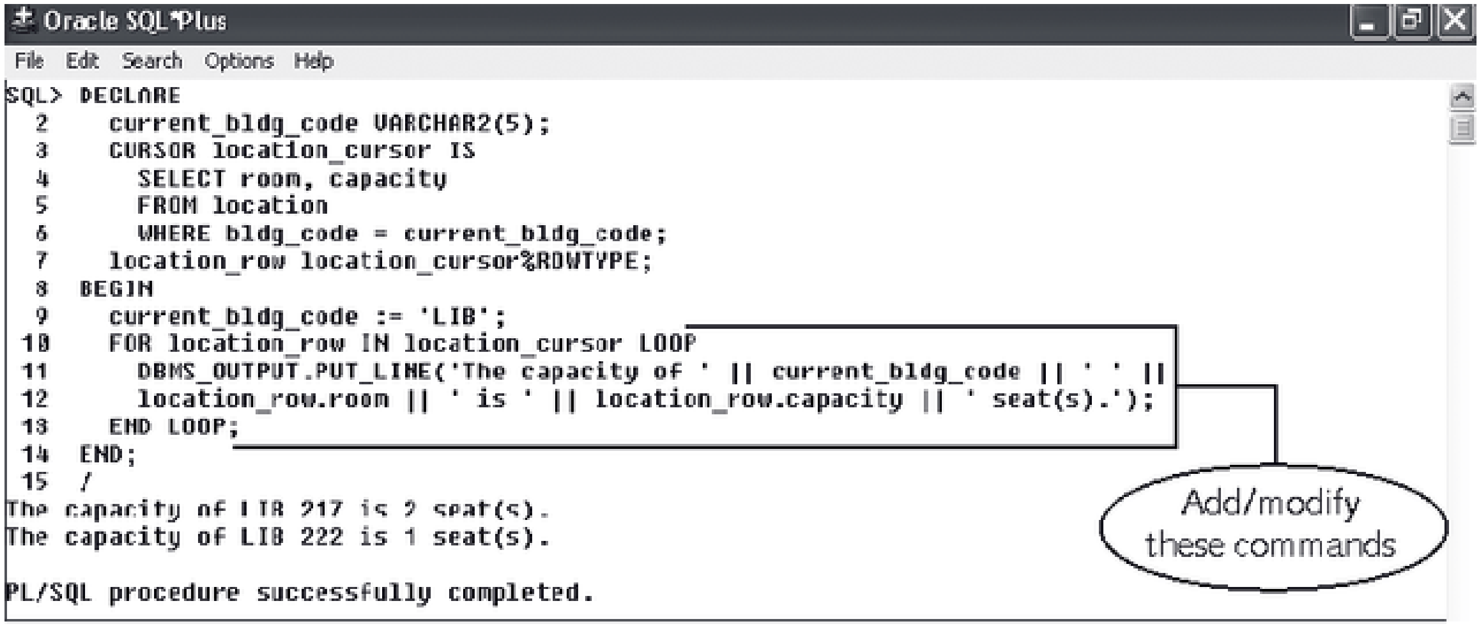
Cursor FOR Loop

- Automatically opens the cursor, fetches the records, then closes the cursor
- *FOR variable_name(s) IN cursor_name LOOP*

  processing commands

  END LOOP;
- Cursor variables cannot be used outside loop

Using Cursor FOR Loop



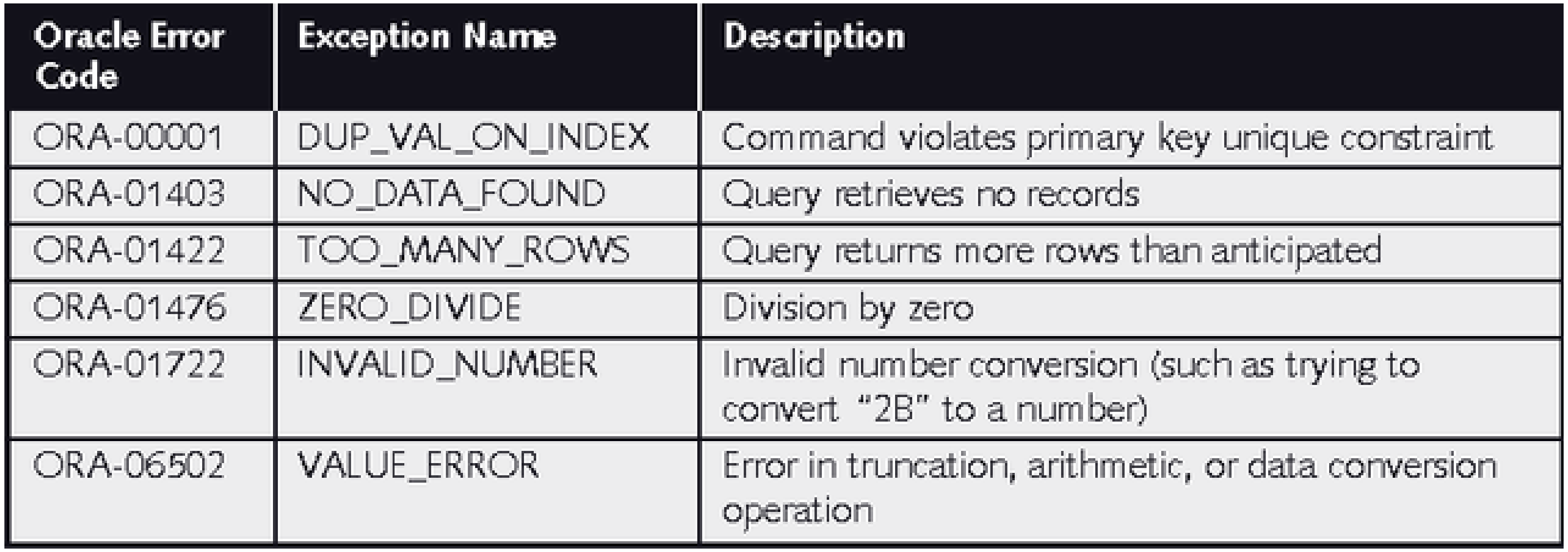
```
SQL> DECLARE
    current_bldg_code VARCHAR2(5);
    CURSOR location_cursor IS
      SELECT room, capacity
      FROM location
      WHERE bldg_code = current_bldg_code;
    location_row location_cursor%ROWTYPE;
  BEGIN
    current_bldg_code := 'LIB';
    FOR location_row IN location_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('The capacity of ' || current_bldg_code || ' ' ||
      location_row.room || ' is ' || location_row.capacity || ' seat(s).');
    END LOOP;
  END;
  /
The capacity of LIB 217 is 2 seat(s).
The capacity of LIB 222 is 1 seat(s).

PL/SQL procedure successfully completed.
```

Handling Runtime Errors in PL/SQL Programs

- Runtime errors cause exceptions
- Exception handlers exist to deal with different error situations
- Exceptions cause program control to fall to exception section where exception is handled

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

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

Undefined Exceptions

- Less common errors
- Do not have predefined names
- Must declare your own name for the exception code in the declaration section
  - *DECLARE*

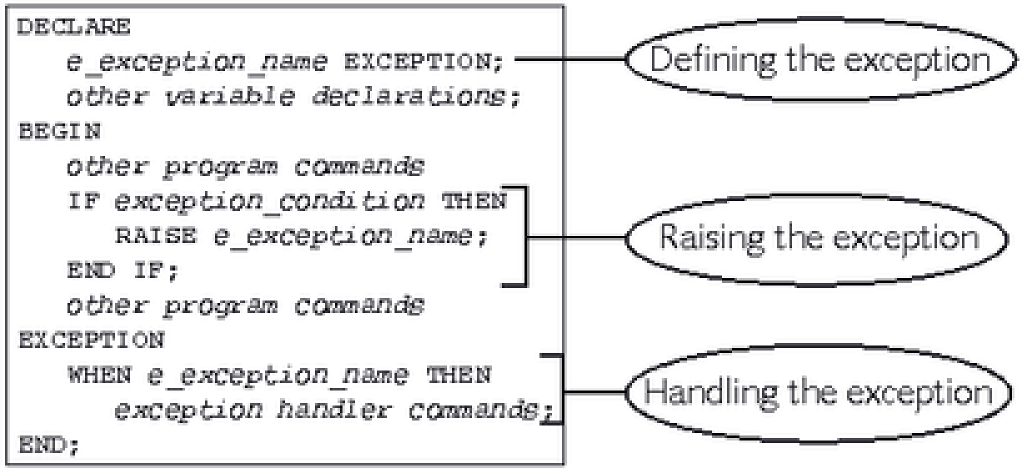
    e_exception_name EXCEPTION;

    PRAGMA EXCEPTION_INIT(e_exception_name,

    -Oracle_error_code);

User-Defined Exceptions

- Not a real Oracle error
- Use to enforce business rules



```
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;
```

Summary

- PL/SQL is a programming language for working with an Oracle database
- Scalar, composite and reference variables can be used
- The IF/THEN/ELSE decision control structure allows branching logic
- Five loop constructs allow repeating code
- Cursors are returned from queries and can be explicitly iterated over
- Exception handling is performed in the exception section. User defined exceptions help to enforce business logic