ORACLE Academy

Introduction to PL/SQL

Why Learn It?

Purpose

PL/SQL is Oracle Corporation's standard procedural language for relational databases.

To describe PL/SQL you learn its characteristics and get better prepared to describe the differences between PL/SQL and SQL.

Understanding the limitations of SQL will help you to understand why the PL/SQL language is needed.

You understand how important it is not to loose a key to your house when you are trying to get into the house without the key.

Tell Me/Show Me

PL/SQL Description

- Procedural Language extension to SQL
  - It allows basic program logic and control flow to be combined with SQL statements
- Oracle proprietary programming language
  - It can be used only with an Oracle database or tool
- Procedural language
  - It produces a result when a series of instructions are followed
- 3GL (third-generation programming language)
  - It is a “high-level” programming language

Tell Me/Show Me

Structured Query Language (SQL) Description

- Is the primary language used to access and modify data in a relational database
- Is a nonprocedural language
  - Also known as a "declarative language," it allows the programmer to focus on input and output rather than the program steps
- Is a 4GL (fourth-generation-programming language)
  - A language that is closer to natural language than a programming language; query languages are generally 4GL
- Is a common query language for many types of databases, including Oracle
- Has been standardized by the American National Standards Institute (ANSI)

Tell Me/Show Me

SQL Statement

```
SELECT class_id, stu_id,
  final_numeric_grade, final_letter_grade
FROM enrollments;
```

The SQL statement shown is simple and straightforward. However, if you want to alter any of the retrieved data in a conditional manner (if the data is xyz then do this to it), you come across the limitations of SQL.

For example, how would you write an SQL statement to update the `final_letter_grade` data with varying letter grades for students in different classes?

Tell Me/Show Me

Limitations of SQL

```
For class_id=1
If 66<final_numeric_grade<75 then final_letter_grade=A
If 56<final_numeric_grade<65 then final_letter_grade=B
If 46<final_numeric_grade<55 then final_letter_grade=C
If 36<final_numeric_grade<45 then final_letter_grade=D
Otherwise, final_letter_grade=F
```

Enrollments

CLASS_ID	STU_ID	FINAL_NUMERIC_GRADE	FINAL_LETTER_GRADE
1	101	75	
1	107	71	
1	131	65	
2	155	91	
2	114	93	

```
For class_id=2
If 91<numeric_grade<100 then  final_letter_grade=A
If 81<numeric_grade<90 then  final_letter_grade=B
If 71<numeric_grade<80 then  final_letter_grade=C
If 71<numeric_grade<80 then  final_letter_grade=D
Otherwise, final_letter_grade=F
```

Tell Me/Show Me

Limitations of SQL (continued)

One solution to updating the final letter grade data is shown.

```
UPDATE enrollments
SET final_letter_grade='A'
WHERE class_id=1 AND
Final_numeric_grade BETWEEN 66 and 75;

UPDATE enrollments
SET final_letter_grade='B'
WHERE class_id=1 AND
Final_numeric_grade between 56 and 65;

And so on…
```

How many SQL statements do you need to write for class_id=1? For class_id=2? What if there were 20 classes?

Tell Me/Show Me

Limitations of SQL (continued)

One solution is to write one SQL statement for each class_id plus number_grade combination. This results in five SQL statements for `class_id=1`:

```
UPDATE enrollments SET final_letter_grade='A'
    WHERE class_id=1
    AND final_numeric_grade BETWEEN 66 and 75;
UPDATE enrollments SET final_letter_grade='B'
    WHERE class_id=1
    AND final_numeric_grade BETWEEN 56 and 65;
UPDATE enrollments SET final_letter_grade='C'
    WHERE class_id=1
    AND final_numeric_grade BETWEEN 46 and 55;
UPDATE enrollments SET final_letter_grade='D'
    WHERE class_id=1
    AND final_numeric_grade BETWEEN 36 and 45;
UPDATE enrollments SET final_letter_grade='F'
    WHERE class_id=1
    AND final_numeric_grade <=35;
```

Tell Me/Show Me

Limitations of SQL (continued)

This is a lot of statements and it does not even include the statements for the other class IDs! Similarly, there would be five statements for `class_id=2`.

It would be easier to write a single statement to accomplish this task. The statement would require logic, otherwise known as conditional or procedural logic.

PL/SQL extends SQL with procedural logic.

Tell Me/Show Me

PL/SQL Extends SQL With Procedural Logic

```
DECLARE
    v_new_letter_grade varchar2(1);
    CURSOR c_enrollments IS
        SELECT stu_id, final_numeric_grade FROM enrollments WHERE class_id=1;
BEGIN
    FOR c1 in c_enrollments
    LOOP
      IF c1.final_numeric_grade BETWEEN 66 and 75 THEN v_new_letter_grade := 'A';
      ELSIF c1.final_numeric_grade BETWEEN 56 AND 65 THEN v_new_letter_grade :=
'B';
      ELSIF c1.final_numeric_grade BETWEEN 46 AND 55 THEN v_new_letter_grade :=
'C';
      ELSIF c1.final_numeric_grade BETWEEN 36 AND 45 THEN v_new_letter_grade :=
'D';
      ELSE
          v_new_letter_grade := 'F';
      END IF;
      UPDATE enrollments
        SET final_letter_grade=v_new_letter_grade WHERE class_id=1
        AND stu_id=c1.stu_id;
    END LOOP;
    COMMIT;
END;
```

Tell Me/Show Me

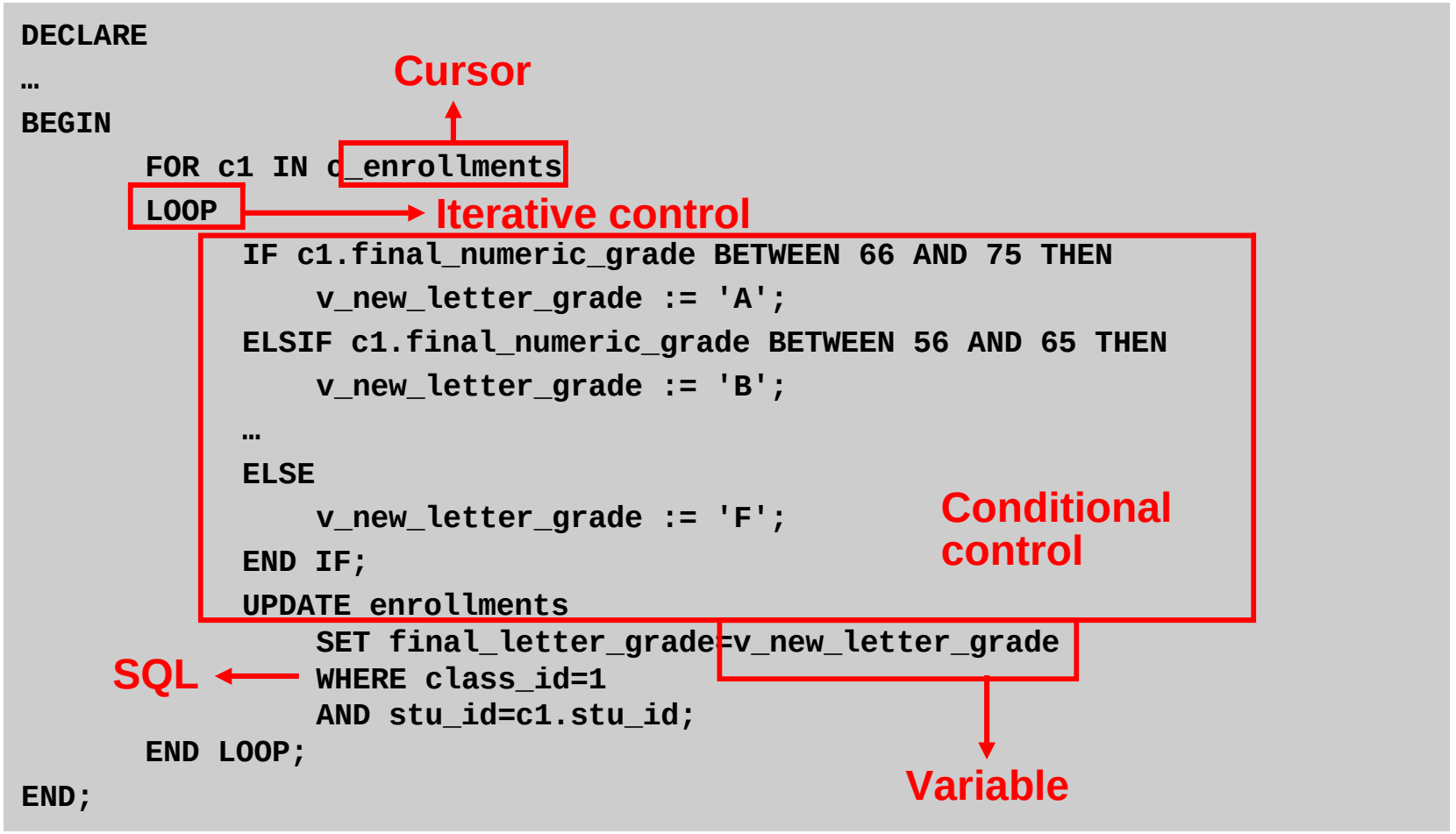
Procedural Constructs

You use PL/SQL to write the procedural code, and embed SQL data-accessing statements within the PL/SQL code.

- The PL/SQL code uses variables, cursors, and conditional logic.
- PL/SQL provides procedural constructs, such as:
  - Variables, constants, and types
  - Control structures, such as conditional statements and loops
  - Reusable program units that are written once and executed many times

Tell Me/Show Me

Procedural Constructs Highlighted



ORACLE Academy

Benefits of PL/SQL

Tell Me/Show Me

Benefits of PL/SQL

There are many benefits to using the PL/SQL programming language with an Oracle database.

1. Integration of procedural constructs with SQL
2. Modularized program development
3. Improved performance
4. Integration with Oracle tools
5. Portability
6. Exception handling

Tell Me/Show Me

Benefit 1: Integration of Procedural Constructs With SQL

The first and foremost advantage of PL/SQL is the integration of procedural constructs with SQL.

- SQL is a nonprocedural language. When you issue an SQL command, your command tells the database server what to do. However, you cannot specify how to do it or how often to do it.
- PL/SQL integrates control statements and conditional statements with SQL. This gives you better control of your SQL statements and their execution.

Tell Me/Show Me

Benefit 2: Modularized Program Development

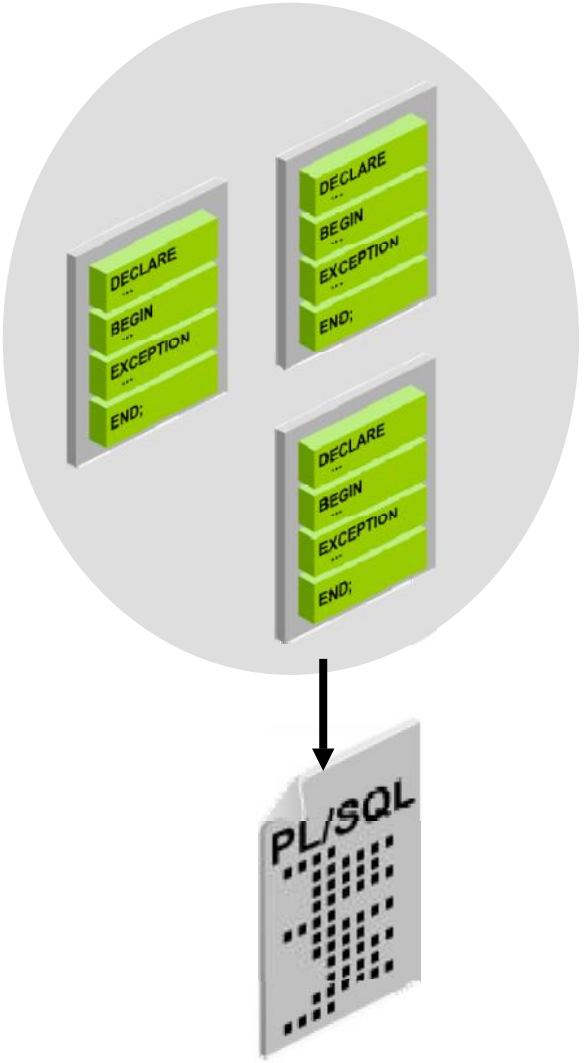
The basic unit in a PL/SQL program is a block. All PL/SQL programs consist of blocks. You can think of these blocks as modules and you can “modularize” these blocks in a sequence or nest them in other blocks.

Tell Me/Show Me

Benefit 2: Modularized Program development (continued)

Modularized program development has the following advantages:

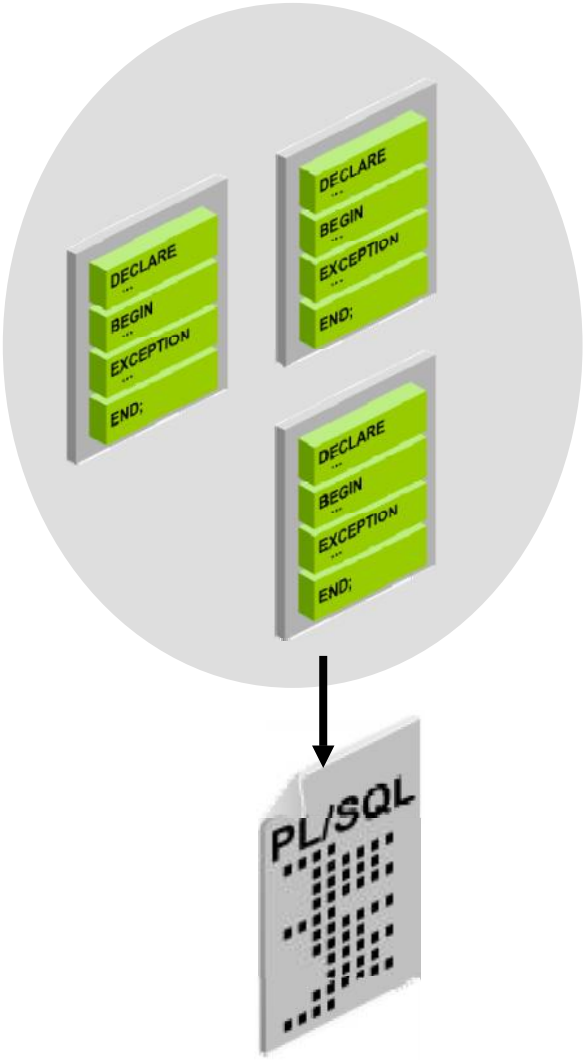
- You can group logically related statements within blocks.
- You can nest blocks inside other blocks to build powerful programs.
- You can share blocks with other programmers to speed up development time.



Tell Me/Show Me

Benefit 2: Modularized Program Development (continued)

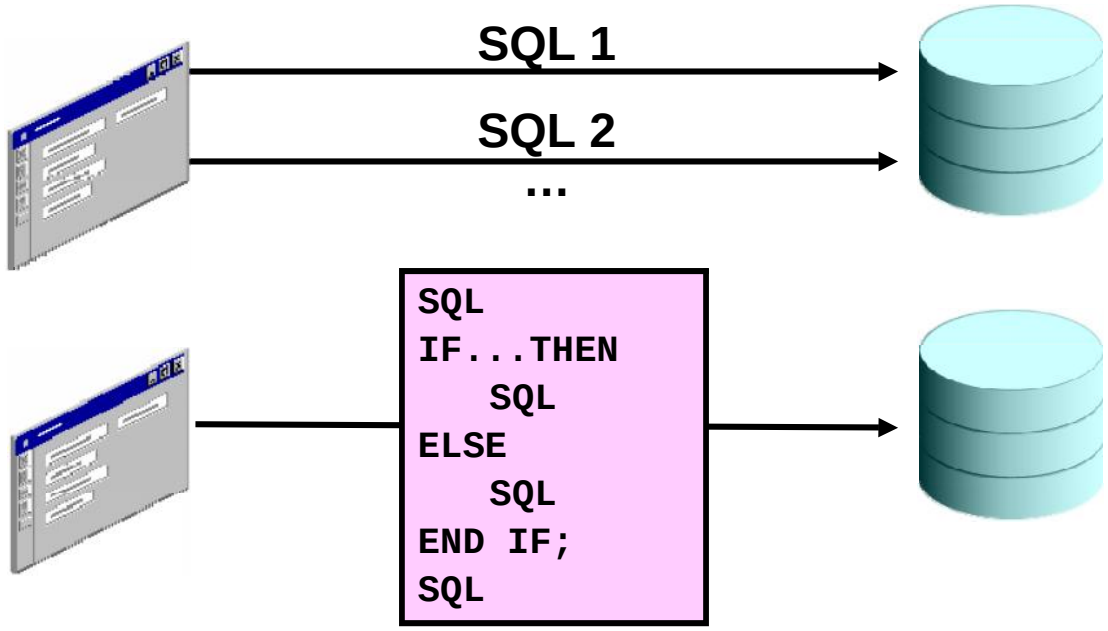
- You can break your application into smaller modules. If you are designing a complex application, PL/SQL allows you to break down the application into smaller, manageable, and logically related modules.
- You can easily read, maintain, and debug the programming statements.



Tell Me/Show Me

Benefit 3: Improved Performance

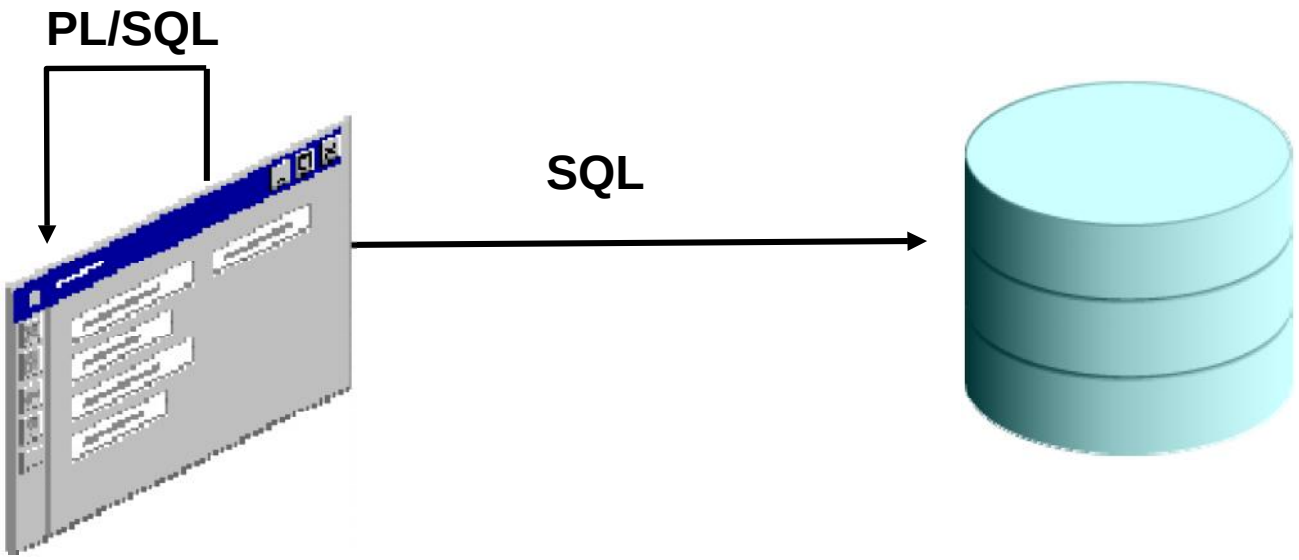
PL/SQL allows you to logically combine multiple SQL statements as one unit or block. The application can send the entire block to the database instead of sending the SQL statements one at a time. This significantly reduces the number of database calls (consider a database with several million records).



Tell Me/Show Me

Benefit 4: Integration With Oracle Tools

PL/SQL is integrated in Oracle tools, such as Oracle Forms Developer, Oracle Report Builder, and Application Express.



Tell Me/Show Me

Benefit 5: Portability

PL/SQL programs can run anywhere an Oracle server runs, regardless of the operating system and the platform. PL/SQL programs do not need to be tailored for different operating systems and platforms.

You can write portable program packages and create libraries that can be reused on Oracle databases in different environments. You can even anticipate those differences and establish instructions to run a specific way given a specific environment.

Linux

HP Tru64

IBM z/OS

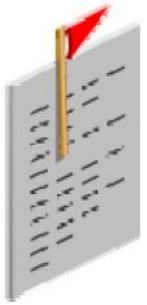
Solaris

Tell Me/Show Me

Benefit 6: Exception Handling

An exception is an error that occurs in the database or in a user's program during runtime. Examples of errors include: hardware or network failures, application logic errors, data integrity errors, and so on. You can prepare for errors by writing exception handling code. Exception handling code tells your program what to do in the event of an exception.

PL/SQL allows you to handle database and program exceptions efficiently. You can define separate blocks for dealing with exceptions.



If there is no data found then …
If too many rows are found then…
If an invalid number is calculated then …

Tell Me/Show Me

PL/SQL Compared to Other Languages

	PL/SQL	C	Java
Requires Oracle database or tool	Yes	No	No
Object-oriented	Some features	No	Yes
Performance against an Oracle database	Very efficient	Less efficient	Less efficient
Portable to different operating systems	Yes	Somewhat	Yes
Ease of learning	Relatively easy	More difficult	More difficult

ORACLE Academy

Creating PL/SQL Blocks

Tell Me/Show Me

PL/SQL Block Structure

A PL/SQL block consists of three sections:

Declarative (optional): The declarative section begins with the keyword `DECLARE` and ends when your executable section starts.

Executable (mandatory): The executable section begins with the keyword `BEGIN` and ends with `END`. Observe that `END` is terminated with a semicolon. The executable section of a PL/SQL block can include any number of nested PL/SQL blocks.

Exception handling (optional): The exception section is nested within the executable section. This section begins with the keyword `EXCEPTION`.

Tell Me/Show Me

PL/SQL Block Structure Sections

Section	Description	Inclusion
Declarative (`DECLARE`)	Contains declarations of all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and exception sections.	Optional
Executable (`BEGIN ... END;`)	Contains SQL statements to retrieve data from the database and PL/SQL statements to manipulate data in the block. Must contain at least one statement.	Mandatory
Exception (`EXCEPTION`)	Specifies the actions to perform when errors and abnormal conditions arise in the executable section.	Optional

Tell Me/Show Me

Anonymous Block

- Unnamed block
- Not stored in the database
- Declared inline at the point in an application where they are executed
- Compiled each time the application is executed
- Passed to the PL/SQL engine for execution at run time
- Cannot be invoked or called because it does not have a name and does not exist after it is executed

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

Tell Me/Show Me

Examples of Anonymous Blocks

1. No declaration or exception sections, execution only

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('PL/SQL is easy!');
END;
```

2. Declaration and execution sections, but no exception section

```
DECLARE
  v_date DATE := SYSDATE;
BEGIN
  DBMS_OUTPUT.PUT_LINE(v_date);
END;
```

Tell Me/Show Me

Examples of Anonymous Blocks (continued)

3. Declaration and exception sections

```
DECLARE
  v_country_name VARCHAR2(40);
  v_region_id      NUMBER;
BEGIN
  SELECT country_name, region_id
    INTO v_country_name, v_region_id
    FROM countries WHERE country_id='CA';
  DBMS_OUTPUT.PUT_LINE ('The country name is: '
           ||v_country_name||' and is located in '
           ||v_region_id||'.') ;
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE ('Your select statement retrieved
      multiple rows. Consider using a cursor.');
END;
```

Tell Me/Show Me

Subprograms

- Are named PL/SQL blocks
- Are stored in the database
- Can be invoked whenever you want depending on your application
- Can be declared as procedures or as functions
  - Procedure: Performs an action
  - Function: Computes and returns a value

```
PROCEDURE name
IS
  --variable
declaration(s)
BEGIN
  --statements

[EXCEPTION]

END;
```

```
FUNCTION name
RETURN datatype
  --variable
declaration(s)
IS
BEGIN
  --statements
  RETURN value;

[EXCEPTION]

END;
```

Tell Me/Show Me

Examples of Subprograms

1. Procedure to print the current date

```
CREATE PROCEDURE print_date IS
  v_date VARCHAR2(30);
BEGIN
  SELECT TO_CHAR(SYSDATE,'Mon DD, YYYY')
    INTO v_date
    FROM DUAL;
  DBMS_OUTPUT.PUT_LINE(v_date);
END;
```

2. Function to return the number of characters in a string

```
CREATE FUNCTION num_characters (p_string IN VARCHAR2)
  RETURN INTEGER IS
  v_num_characters INTEGER;
BEGIN
  SELECT LENGTH(p_string) INTO v_num_characters
    FROM DUAL;
    RETURN v_num_characters;
END;
```