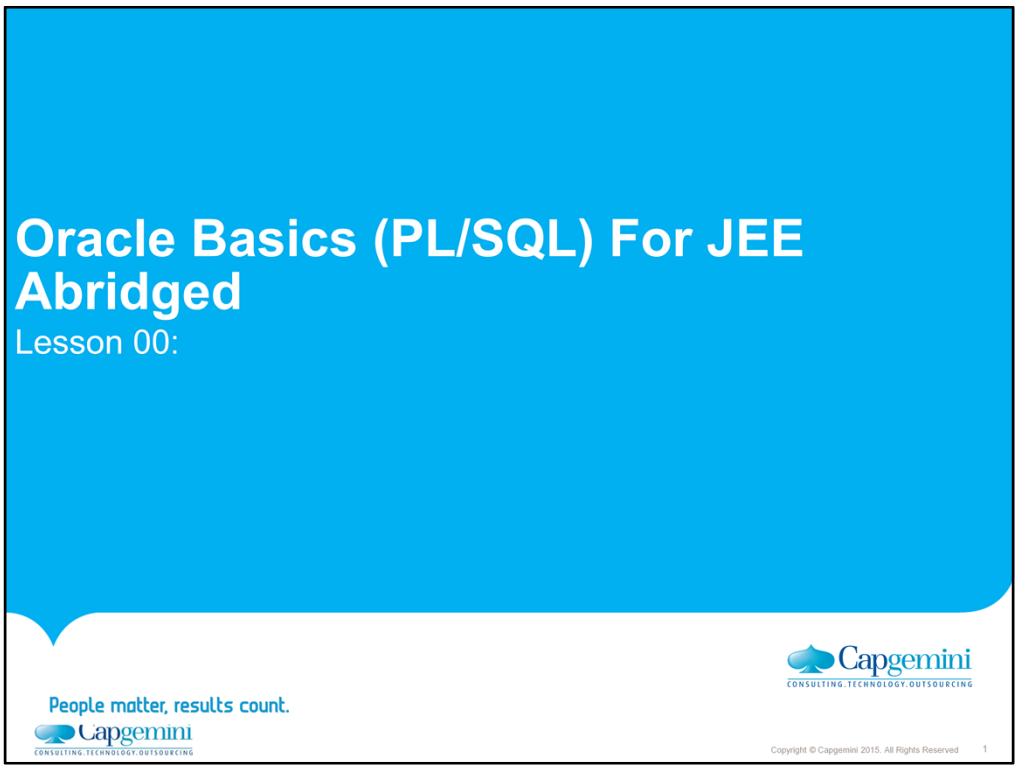


Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
13-Nov-2008	1.0	Oracle9i	Rajita Dhumal	Content Creation.
28-Nov-2008	1.1	Oracle9i	CLS team	Review.
14-Jan-2010	1.2	Oracle9i	Anu Mitra,	Review
14-Jan-2010	1.2	Oracle9i	Rajita Dhumal, CLS Team	Incorporating Review Comments
16-May-2011	2.0	Oracle 9i	Anu Mitra	Integration Refinements
17-May-2013	2.1	Oracle 9i	Hareshkumar Chandiramani	Courseware Refinements
7-July-2016	2.2	Oracle 9i	Kavita Arora	Integration Refinements

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

Course Goals and Non Goals

- Course Goals
  - To understand RDBMS Methodology.
  - To code PL/SQL Blocks for Implementing business rules.
  - To create Stored Subprograms using Procedures and Functions.
  - To implement Business Rule using Constraints.
- Course Non Goals
  - Object Oriented programming concepts (ORDBMS) are not covered as a part of this course.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Pre-requisites

- Require a fair proficiency level in Relational Database Concepts.
- Require good proficiency in DBMS SQL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Intended Audience

- Software Programmers
- Software Analysts

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

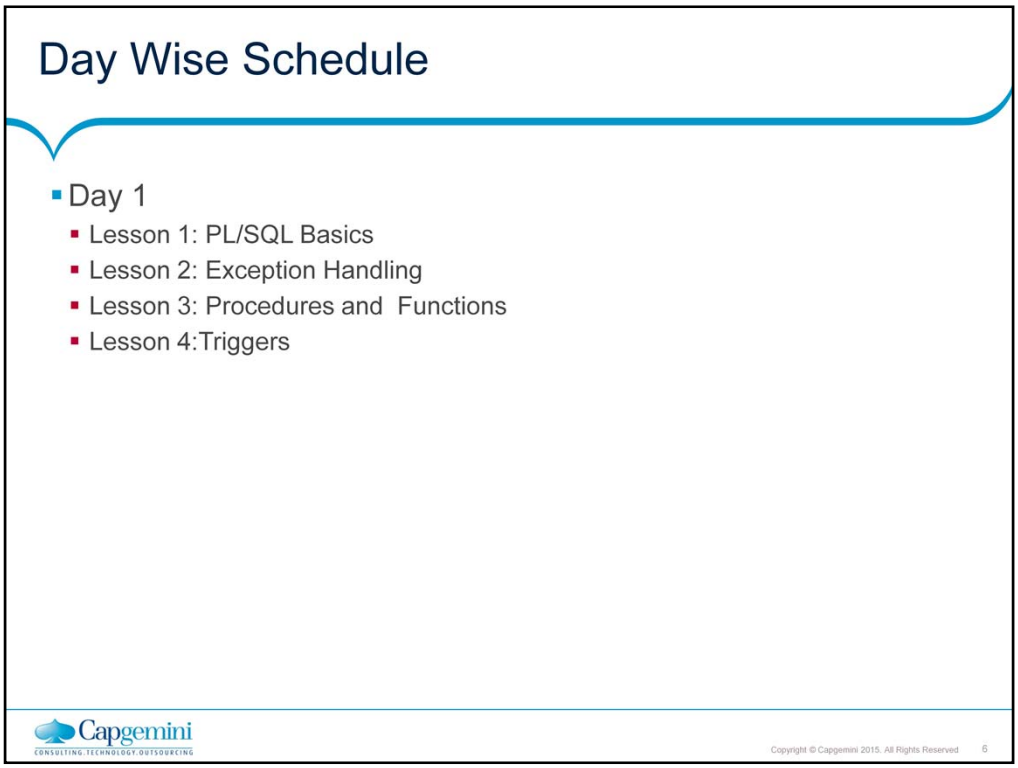

Day Wise Schedule
Day 1
Lesson 1: PL/SQL Basics
Lesson 2: Exception Handling
Lesson 3: Procedures and Functions
Lesson 4:Triggers
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
6

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

References

- Oracle PL/SQL Programming, Third Edition; by Steven Feuerstein
- Oracle 9i PL/SQL: A Developer's Guide
- Oracle9i PL/SQL Programming; by Scott Urman

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

Next Step Courses (if applicable)

- Data Warehousing Concepts
- Reporting / ETL tools
- Database Administration / Database Performance Tuning

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Other Parallel Technology Areas

- Microsoft SQL Server
- IBM DB2

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

Oracle Basics (PL/SQL)

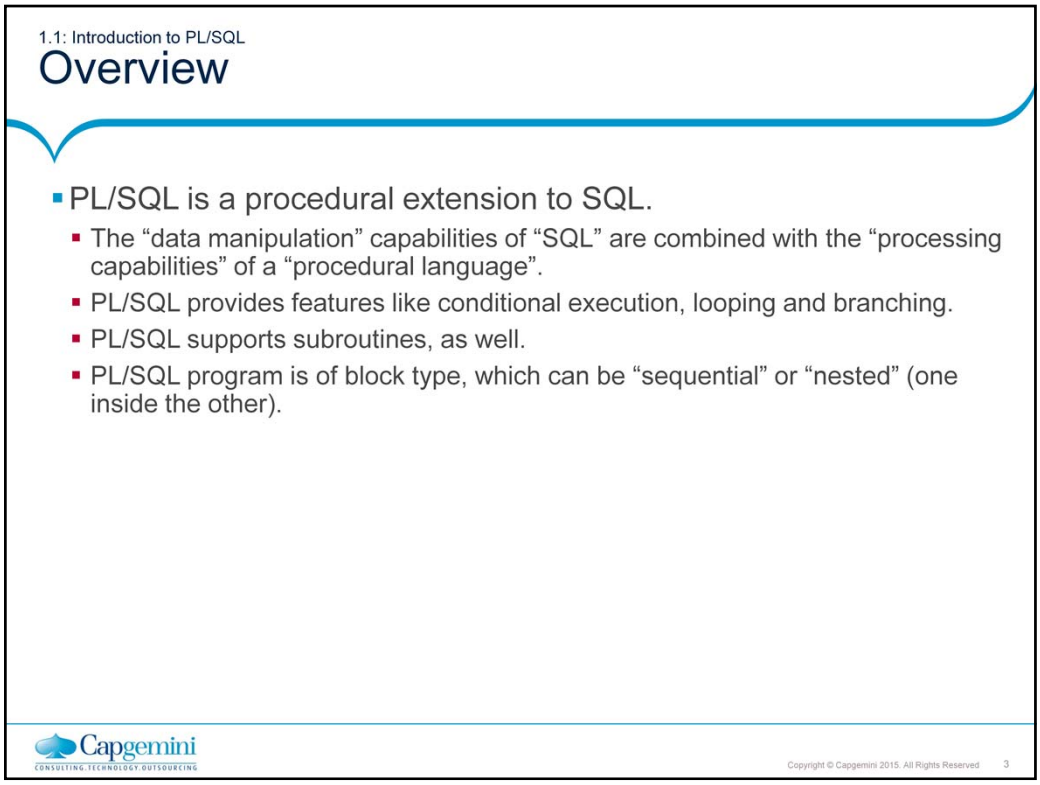
Lesson 01 Introduction to PL/SQL

Lesson Objectives

- To understand the following topics:
  - Introduction to PL/SQL
  - PL/SQL Block structure
  - Handling variables in PL/SQL
  - Variable scope and Visibility
  - SQL in PL/SQL
  - Programmatic Constructs

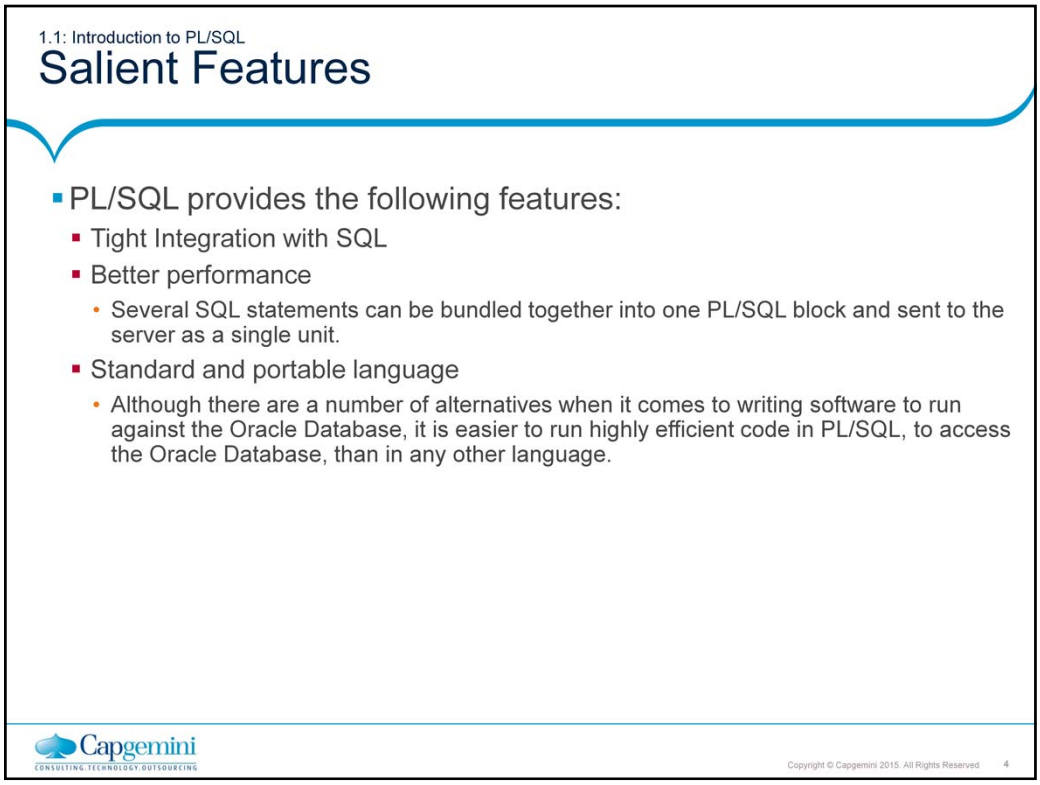
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



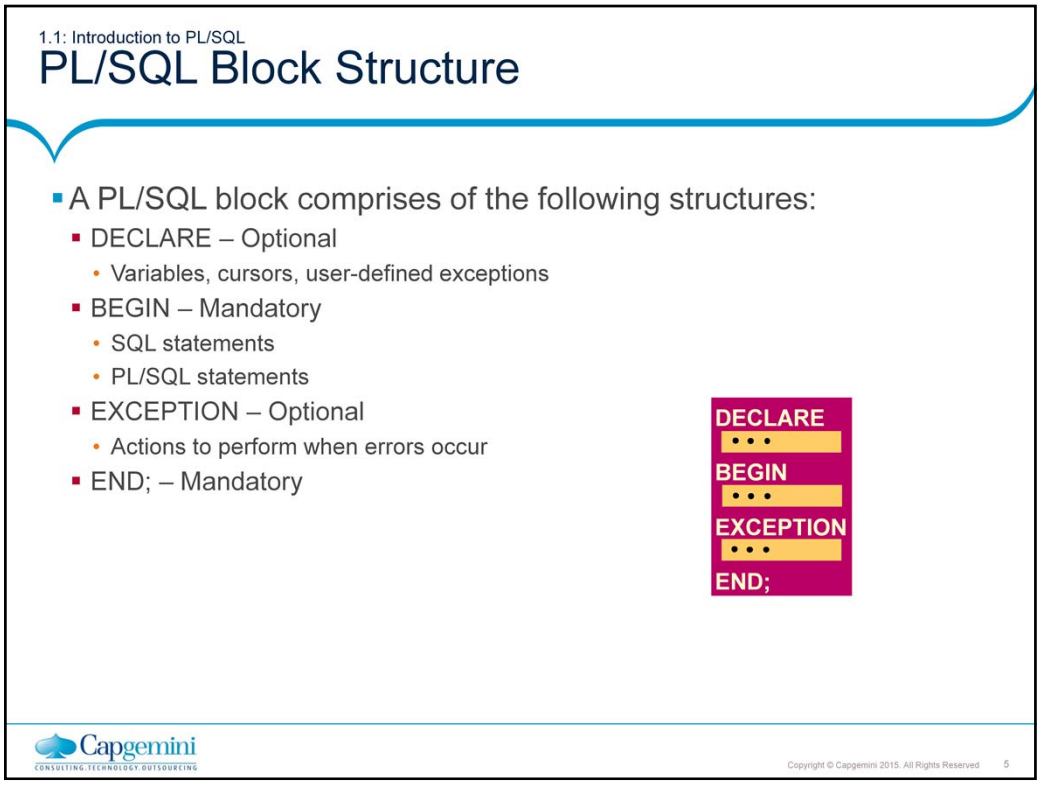
Introduction to PL/SQL:

- PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is “more powerful than SQL”.
  - With PL/SQL, you can use SQL statements to manipulate Oracle data and flow-of-control statements to process the data.
  - Moreover, you can declare constants and variables, define procedures and functions, and trap runtime errors.
  - Thus PL/SQL combines the “data manipulating power” of SQL with the “data processing power” of procedural languages.
- PL/SQL is an “embedded language”. It was not designed to be used as a “standalone” language but instead to be invoked from within a “host” environment.
  - You cannot create a PL/SQL “executable” that runs all by itself.
  - It can run from within the database through SQL*Plus interface or from within an Oracle Developer Form (called client-side PL/SQL).



Features of PL/SQL

- Tight Integration with SQL:
  - This integration saves both, your learning time as well as your processing time.
    - PL/SQL supports SQL data types, reducing the need to convert data passed between your application and database.
    - PL/SQL lets you use all the SQL data manipulation, cursor control, transaction control commands, as well as SQL functions, operators, and pseudo columns.
- Better Performance:
  - Several SQL statements can be bundled together into one PL/SQL block, and sent to the server as a single unit.
  - This results in less network traffic and a faster application. Even when the client and the server are both running on the same machine, the performance is increased. This is because packaging SQL statements results in a simpler program that makes fewer calls to the database.
- Portable:
  - PL/SQL is a standard and portable language.
  - A PL/SQL function or procedure written from within the Personal Oracle database on your laptop will run without any modification on your corporate network database. It is “Write once, run everywhere” with the only restriction being “everywhere” there is an Oracle Database.
- Efficient:
  - Although there are a number of alternatives when it comes to writing software to run against the Oracle Database, it is easier to run highly efficient code in PL/SQL, to access the Oracle Database, than in any other language.



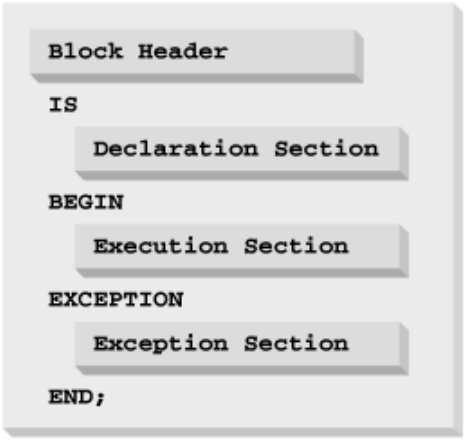
PL/SQL Block Structure:

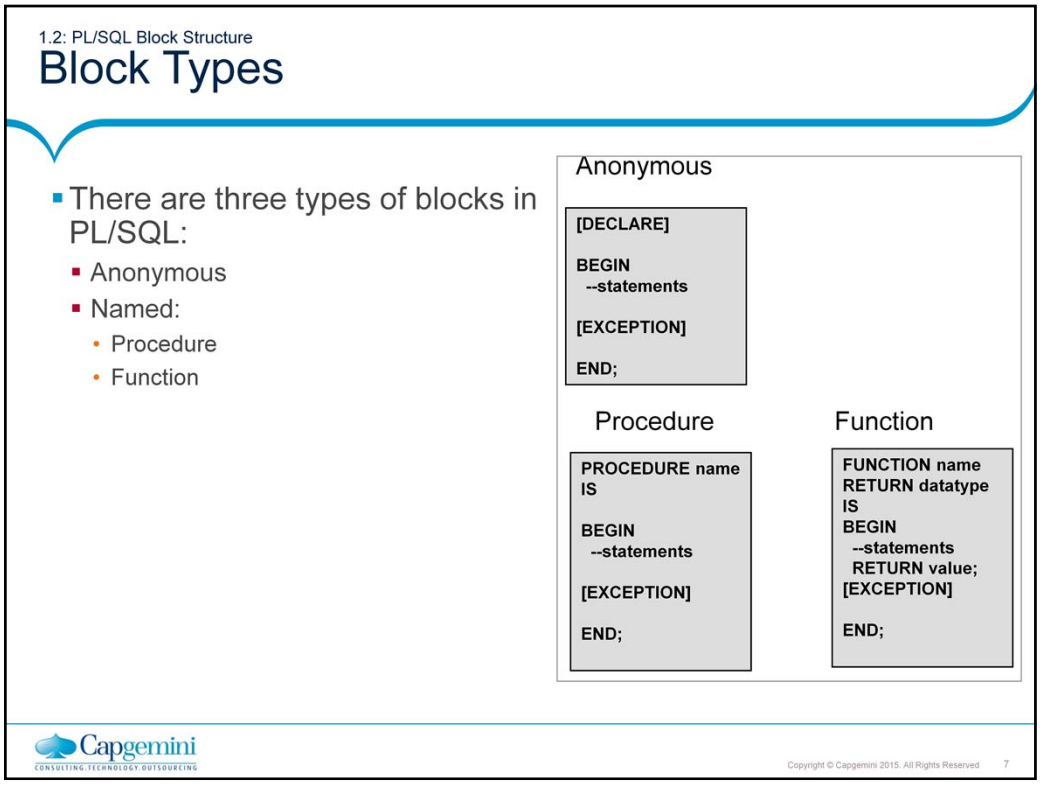
- PL/SQL is a block-structured language. Each basic programming unit that is written to build your application is (or should be) a “logical unit of work”. The PL/SQL block allows you to reflect that logical structure in the physical design of your programs.
- Each PL/SQL block has up to four different sections (some are optional under certain circumstances).

contd.

PL/SQL block structure (contd.):

- **Header**
  It is relevant for named blocks only. The header determines the way the named block or program must be called.
- **Declaration section**
  The part of the block that declares variables, cursors, and sub-blocks that are referenced in the Execution and Exception sections.
- **Execution section**
  It is the part of the PL/SQL blocks containing the executable statements; the code that is executed by the PL/SQL runtime engine.
- **Exception section**
  It is the section that handles exceptions for normal processing (warnings and error conditions).





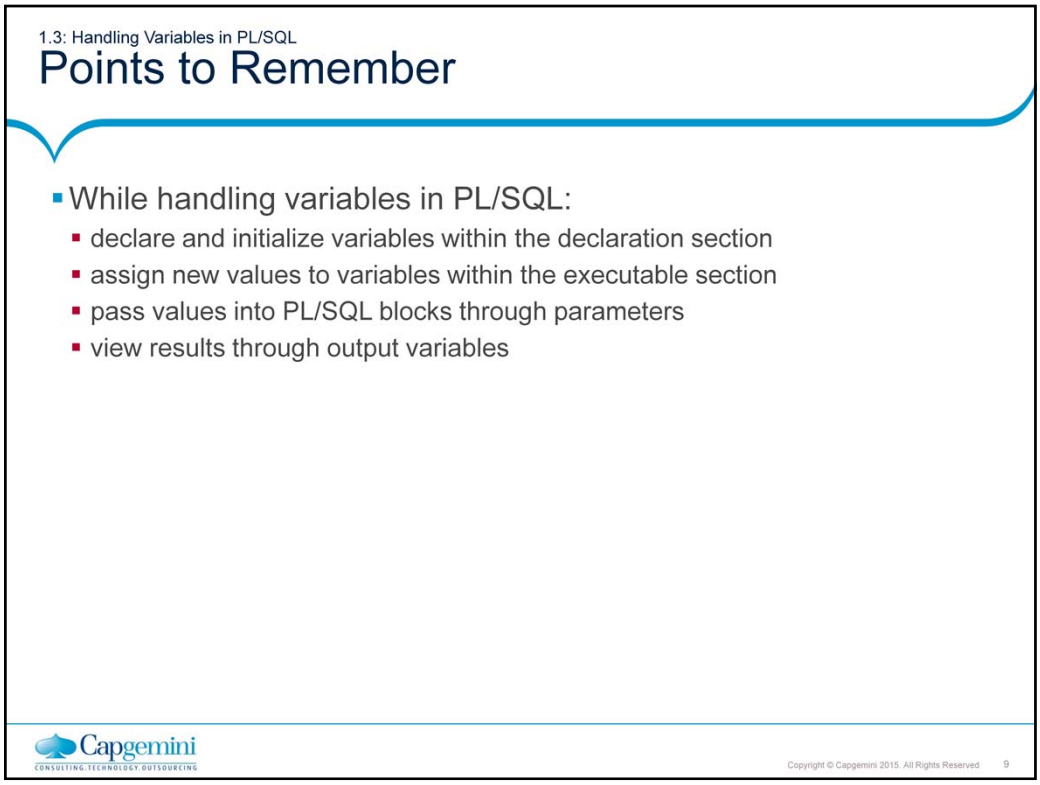
Block Types:

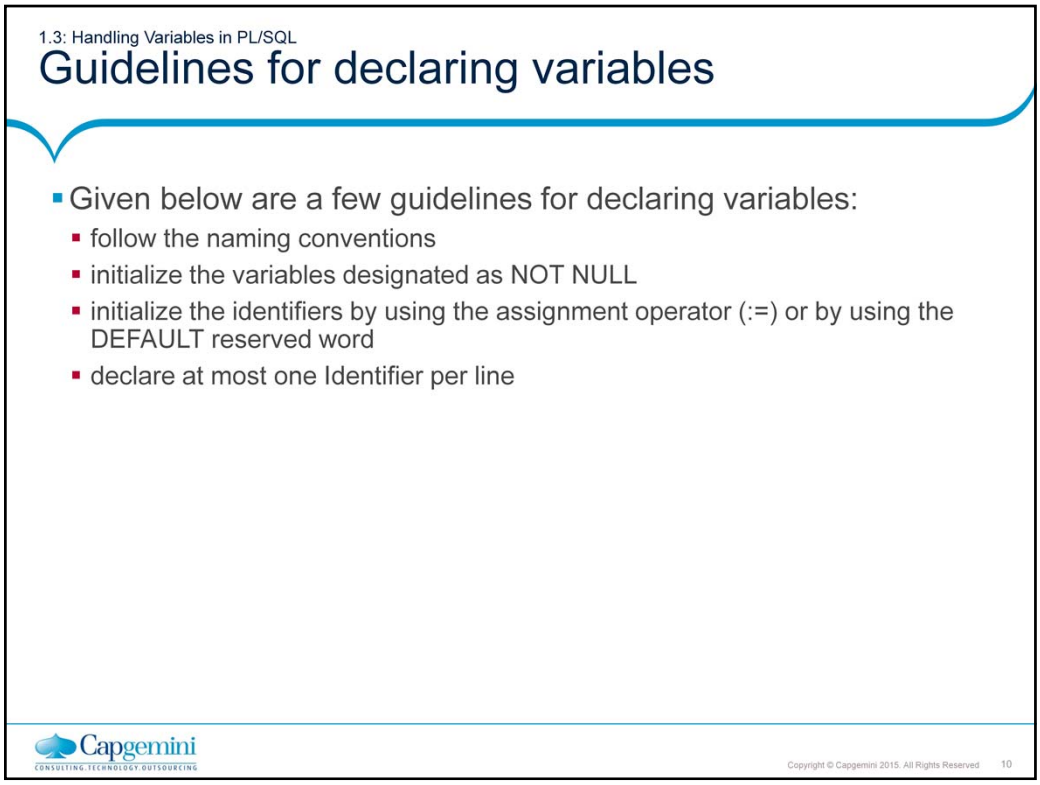
- The basic units (procedures and functions, also known as subprograms, and anonymous blocks) that make up a PL/SQL program are “logical blocks”, which can contain any number of nested sub-blocks.
- Therefore one block can represent a small part of another block, which in turn can be part of the whole unit of code.
  - **Anonymous Blocks**
    Anonymous blocks are unnamed blocks. They are declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at runtime.
  - **Named :**
    - **Subprograms**
      Subprograms are named PL/SQL blocks that can take parameters and can be invoked. You can declare them either as “procedures” or as “functions”.
      Generally, you use a “procedure” to perform an “action” and a “function” to compute a “value”.

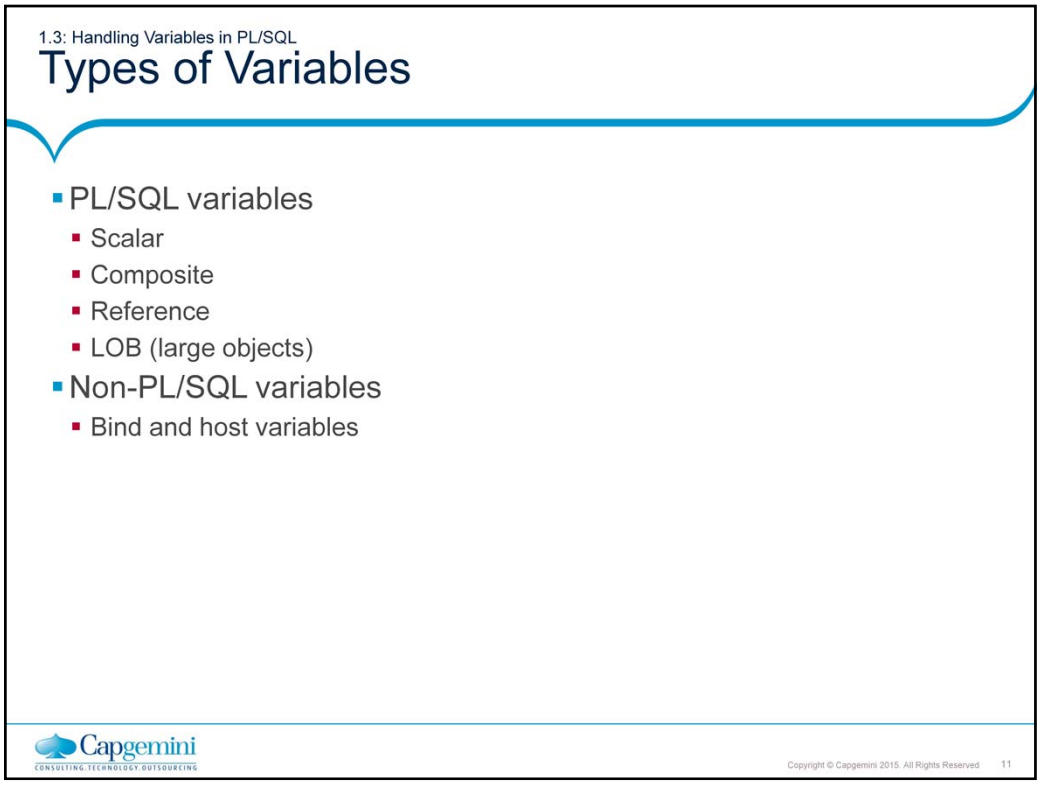
Representation of a PL/SQL block:

- The notations used in a PL/SQL block are given below:
  1. -- is a single line comment.
  2. /* */ is a multi-line comment.
  3. Every statement must be terminated by a semicolon (;).
  4. PL/SQL block is terminated by a slash (/) on a line by itself.
- A PL/SQL block must have an “Execution section”.
- It can optionally have a “Declaration section” and an Exception section, as well.

```
DECLARE                    -- Declaration Section
  V_Salary    NUMBER(7,2);
/* V_Salary is a variable declared in a PL/SQL block. This variable is
used to store JONES' salary. */
Low_Sal  EXCEPTION;                    -- an exception
BEGIN                          -- Execution Section
SELECT sal INTO V_Salary
FROM emp WHERE ename = 'JONES'
        IF V_Salary < 3000 THEN
                RAISE Low_Sal ;
        END IF;
EXCEPTION                      -- Exception Section
    WHEN Low_Sal     THEN
    UPDATE emp SET sal = sal + 500 WHERE ename = 'JONES' ;
END ;                          -- End of Block
/                              -- PL/SQL block terminator
Output
SQL> /
PL/SQL procedure successfully completed.
```

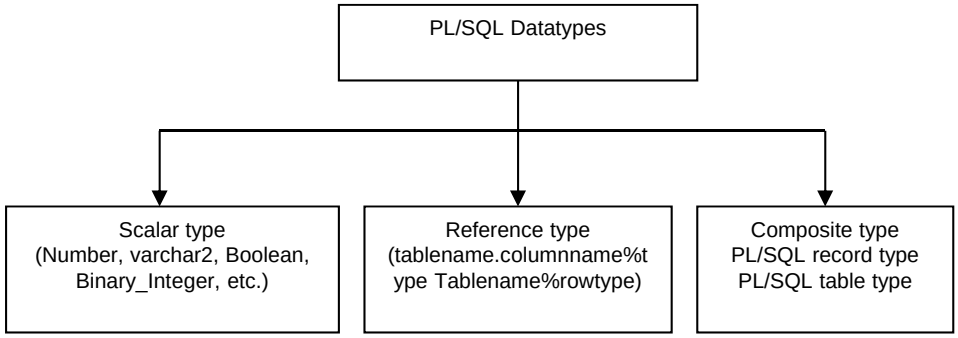

1.3: Handling Variables in PL/SQL
Points to Remember
While handling variables in PL/SQL:
declare and initialize variables within the declaration section
assign new values to variables within the executable section
pass values into PL/SQL blocks through parameters
view results through output variables
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
9

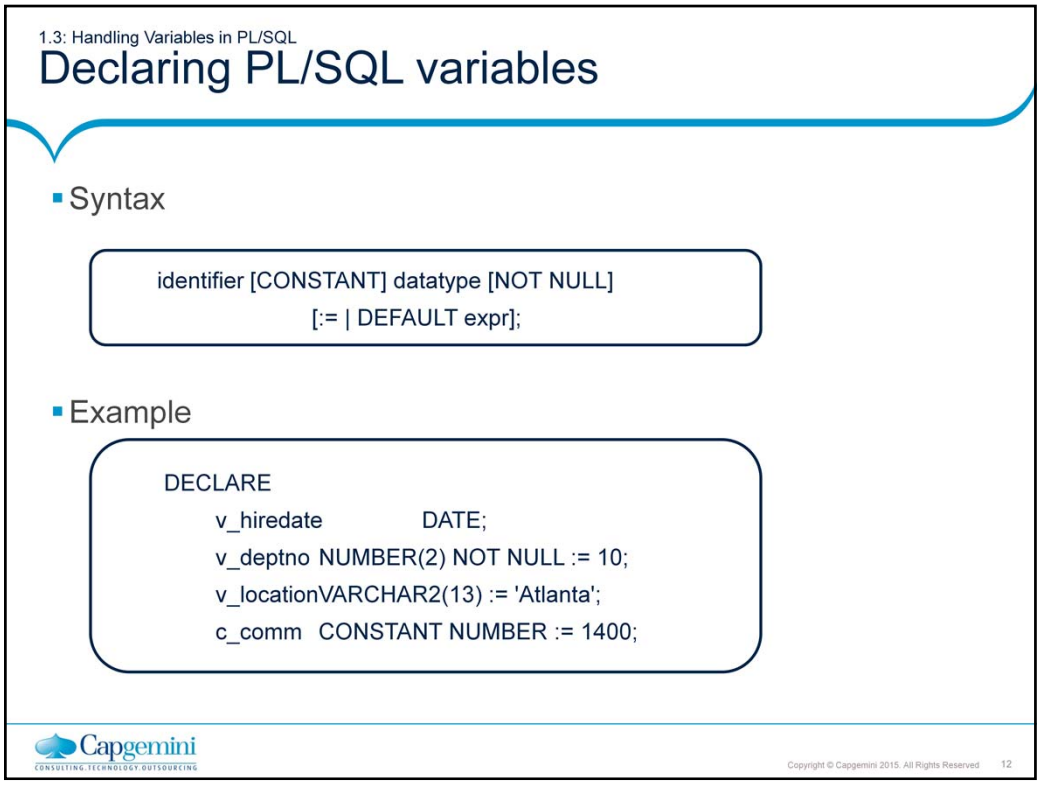

1.3: Handling Variables in PL/SQL
Guidelines for declaring variables
Given below are a few guidelines for declaring variables:
follow the naming conventions
initialize the variables designated as NOT NULL
initialize the identifiers by using the assignment operator (:=) or by using the DEFAULT reserved word
declare at most one Identifier per line
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
10



Types of Variables: PL/SQL Datatype:

- All PL/SQL datatypes are classified as scalar, reference and Composite type.
- Scalar datatypes do not have any components within it, while composite datatypes have other datatypes within them.
- A reference datatype is a pointer to another datatype.





Declaring PL/SQL Variables:

- You need to declare all PL/SQL identifiers within the "declaration section" before referencing them within the PL/SQL block.
- You have the option to assign an initial value.
  - You do not need to assign a value to a variable in order to declare it.
  - If you refer to other variables in a declaration, you must separately declare them in a previous statement.
  - Syntax:

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

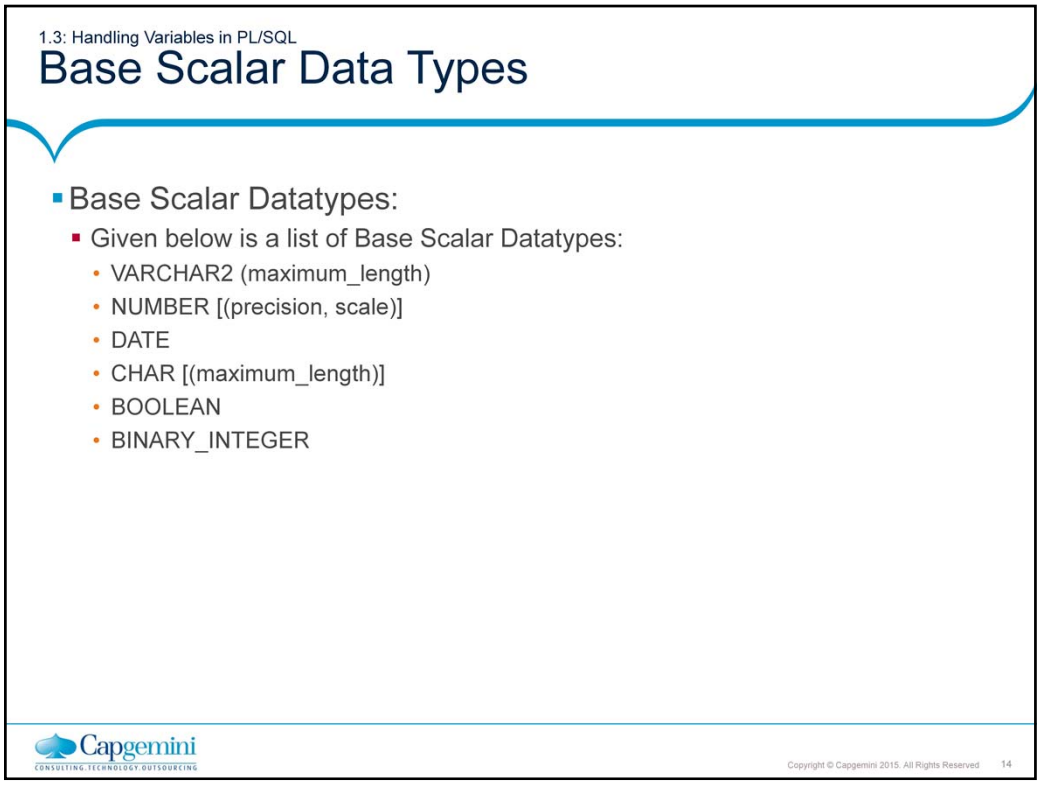
  - In the syntax given above:
    - **identifier** is the name of the variable.
    - **CONSTANT** constrains the variable so that its value cannot change. Constants must be initialized.
    - **datatype** is a scalar, composite, reference, or LOB datatype.
    - **NOT NULL** constrains the variable so that it must contain a value. NOT NULL variables must be initialized.
    - **expr** is any PL/SQL expression that can be a literal, another variable, or an expression involving operators and functions.

contd.

Declaring PL/SQL Variables (contd.):

For example:

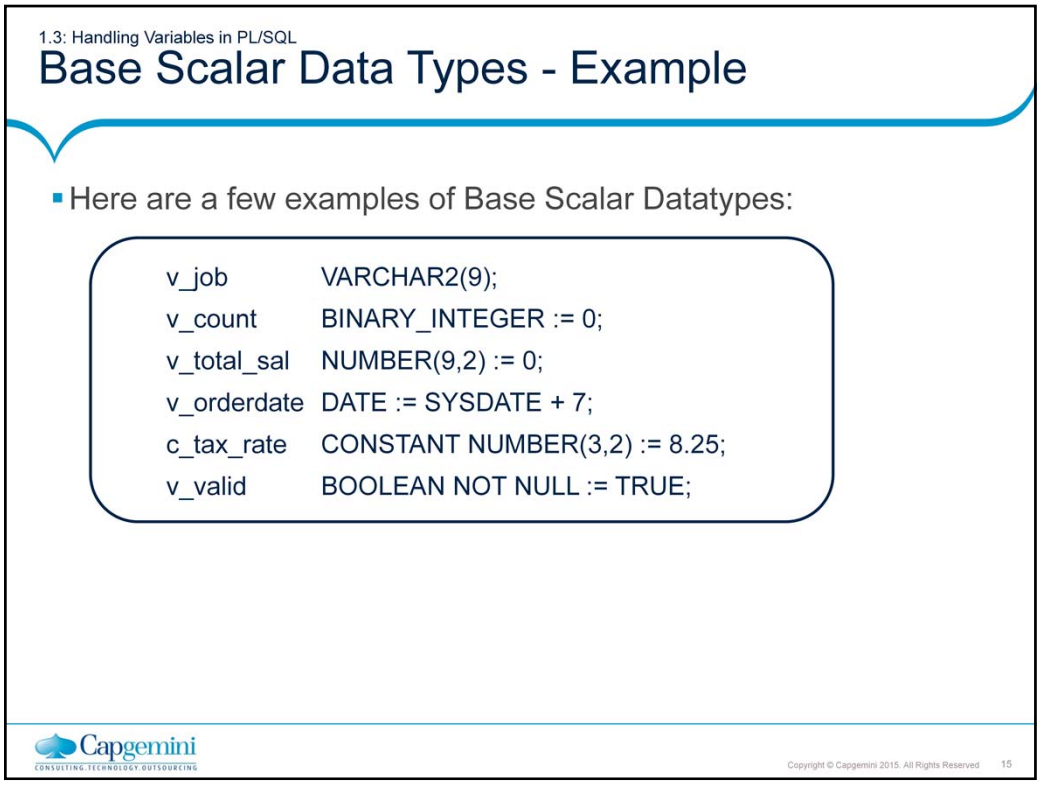
```
DECLARE
   v_description varchar2 (25);
   v_sal             number (5) not null: = 3000;
   v_compcode  varchar2 (20) constant: = 'abc
                        consultants';
   v_comm        not null default   0;
```



Base Scalar Datatypes:

1. **NUMBER**
   This can hold a numeric value, either integer or floating point. It is same as the number database type.
2. **BINARY_INTEGER**
   If a numeric value is not to be stored in the database, the BINARY_INTEGER datatype can be used. It can only store integers from -2147483647 to + 2147483647. It is mostly used for counter variables.
   V_Counter BINARY_INTEGER DEFAULT 0;
3. **VARCHAR2 (L)**
   L is necessary and is max length of the variable. This behaves like VARCHAR2 database type. The maximum length in PL/SQL is 32,767 bytes whereas VARCHAR2 database type can hold max 2000 bytes. If a VARCHAR2 PL/SQL column is more than 2000 bytes, it can only be inserted into a database column of type LONG.
4. **CHAR (L)**
   Here L is the maximum length. Specifying length is optional. If not specified, the length defaults to 1. The maximum length of CHAR PL/SQL variable is 32,767 bytes, whereas the maximum length of the database CHAR column is 255 bytes. Therefore a CHAR variable of more than 255 bytes can be inserted in the database column of VARCHAR2 or LONG type.

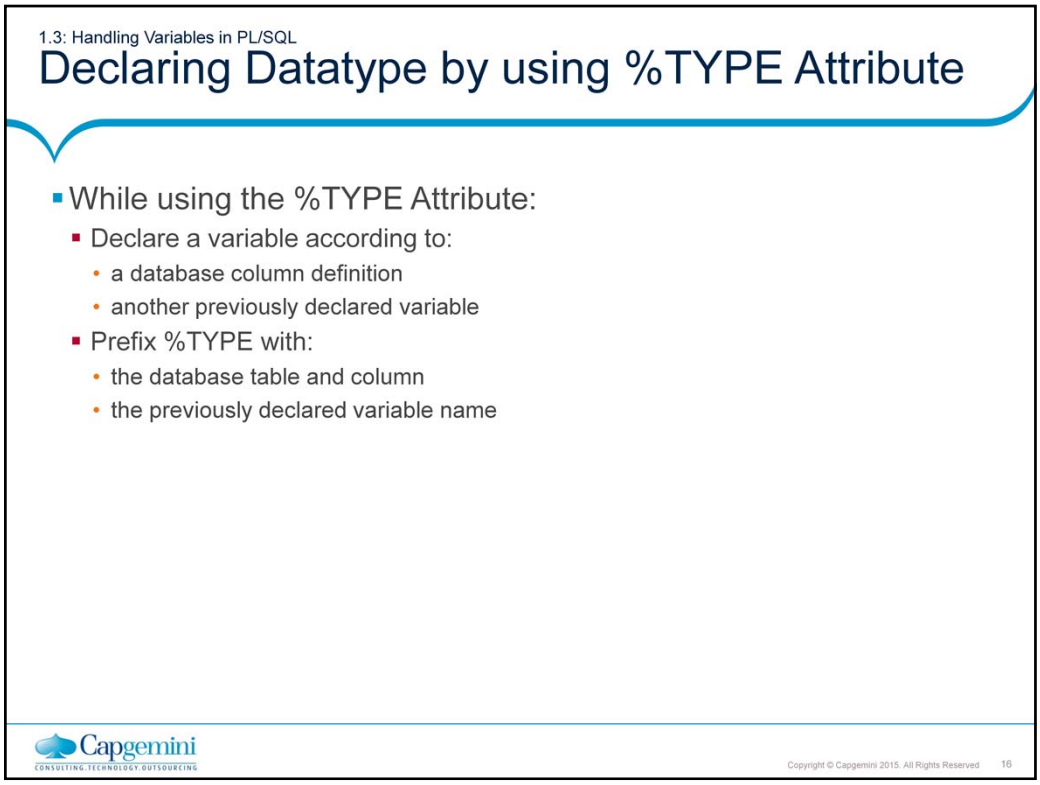
contd.



Base Scalar Datatypes (contd.):

5. **LONG**
   PL/SQL LONG type is just 32,767 bytes. It behaves similar to LONG DATABASE type.
6. **DATE**
   The DATE PL/SQL type behaves the same way as the date database type. The DATE type is used to store both date and time. A DATE variable is 7 bytes in PL/SQL.
7. **BOOLEAN**
   A Boolean type variable can only have one of the two values, i.e. either TRUE or FALSE. They are mostly used in control structures.
   V_Does_Dept_Exist BOOLEAN := TRUE;
   V_ Flag BOOLEAN := 0; -- illegal

```
declare
        pie constant number := 7.18;
        radius number := &radius;
begin
        dbms_output.put_line('Area:
'||pie*power(radius,2));
        dbms_output.put_line('Diameter: '||2*pie*radius);
end;
/
```



Reference types:

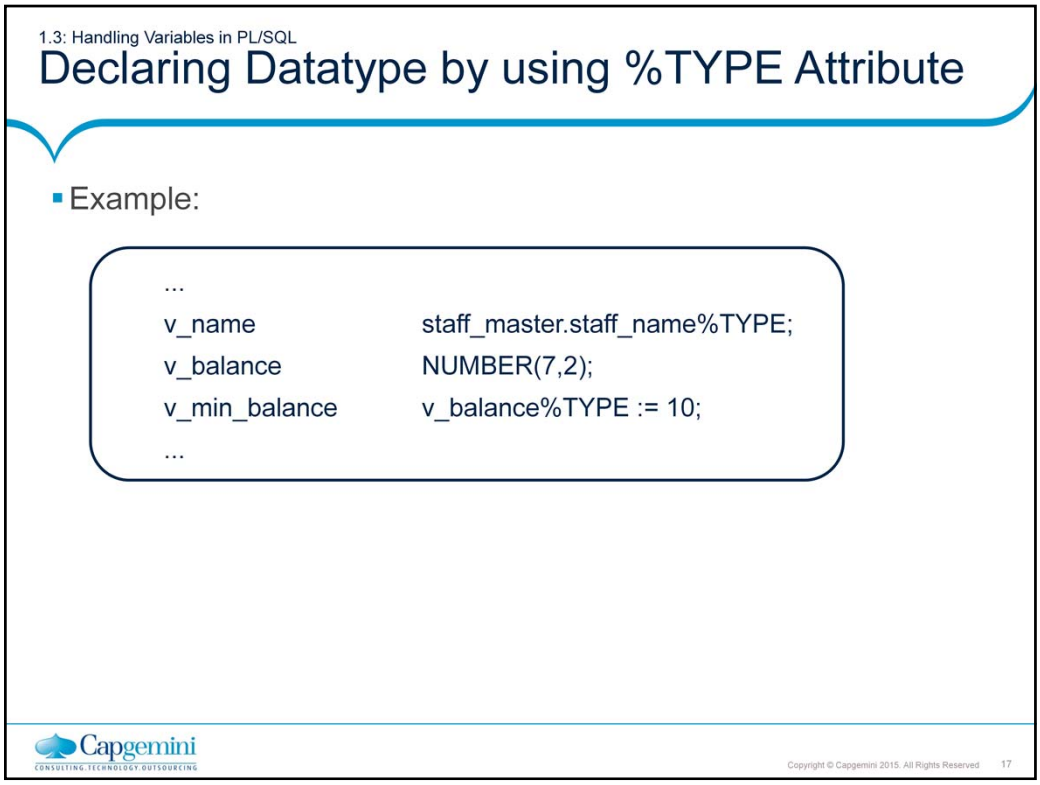
- A "reference type" in PL/SQL is the same as a "pointer" in C. A "reference type" variable can point to different storage locations over the life of the program.

Using %TYPE

- %TYPE is used to declare a variable with the same datatype as a column of a specific table. This datatype is particularly used when declaring variables that will hold database values.
- **Advantage:**
  - You need not know the exact datatype of a column in the table in the database.
  - If you change database definition of a column, it changes accordingly in the PL/SQL block at run time.
  - Syntax:

```
Var_Name        table_name.col_name%TYPE;
V_Empno         emp.empno%TYPE;
```

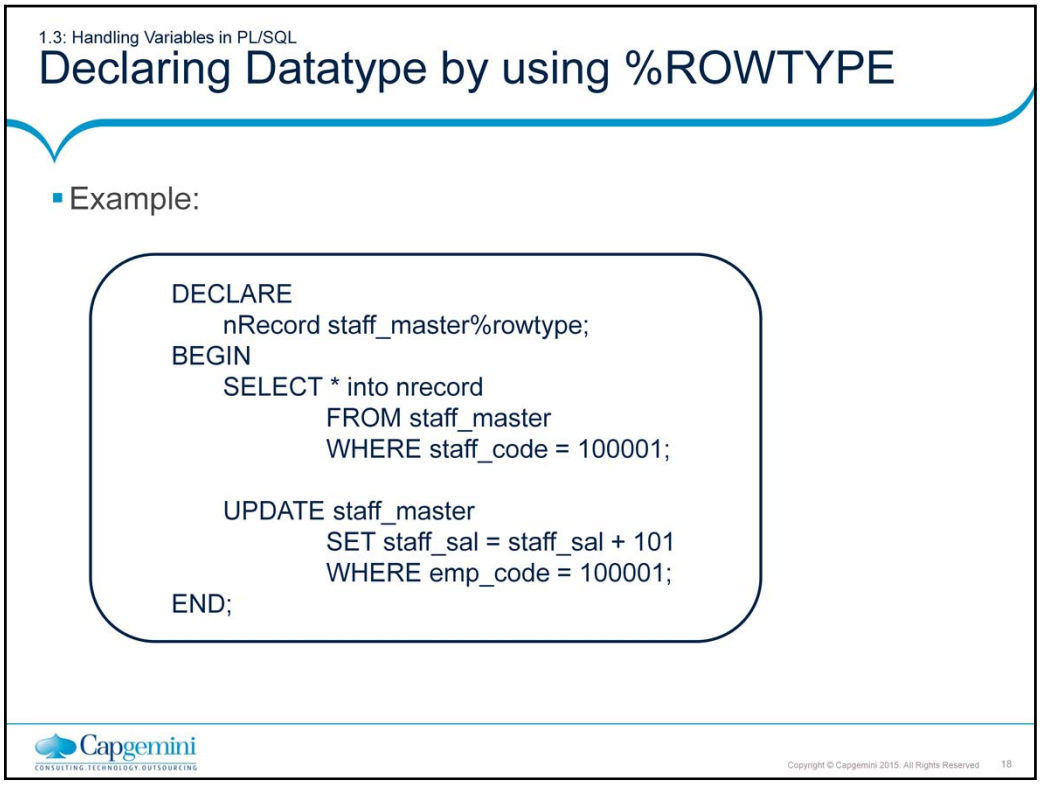
  - **Note:** Datatype of V_Empno is same as datatype of Empno column of the EMP table.



Using %TYPE (contd.)

- Example

```
declare
        nSalary employee.salary%type;
begin
      select salary into nsalary
        from employee
        where emp_code = 11;
        update employee set salary = salary + 101
where emp_code = 11;
end;
```



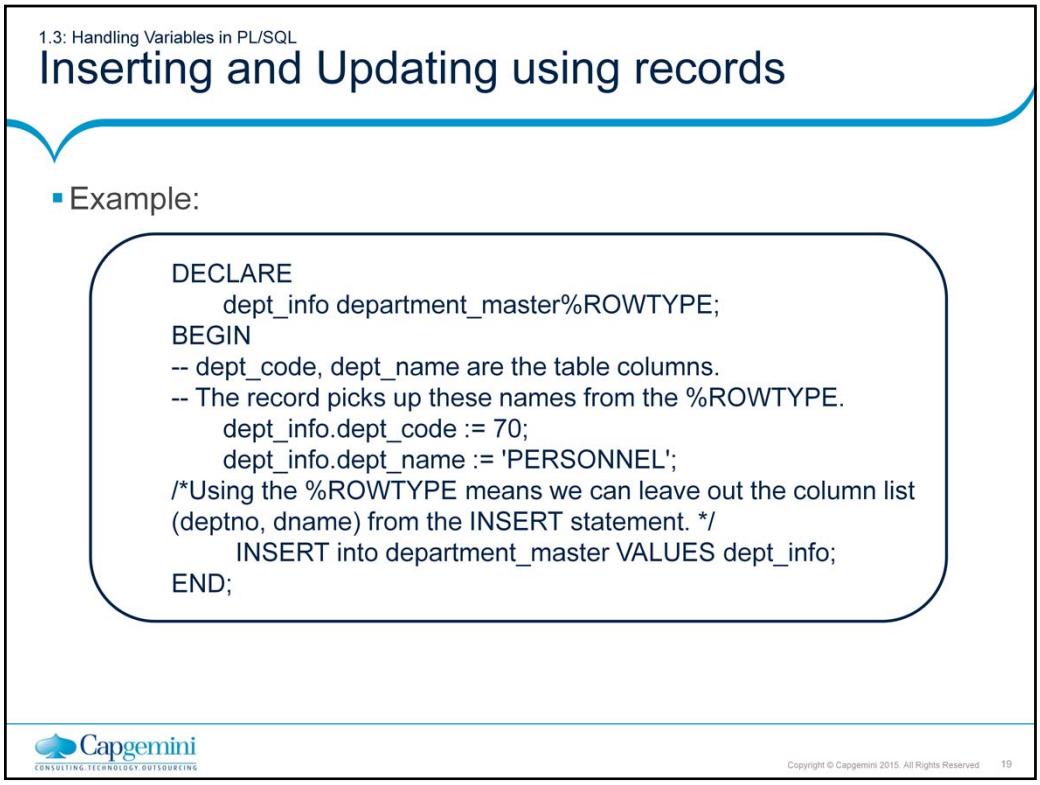
Using %ROWTYPE

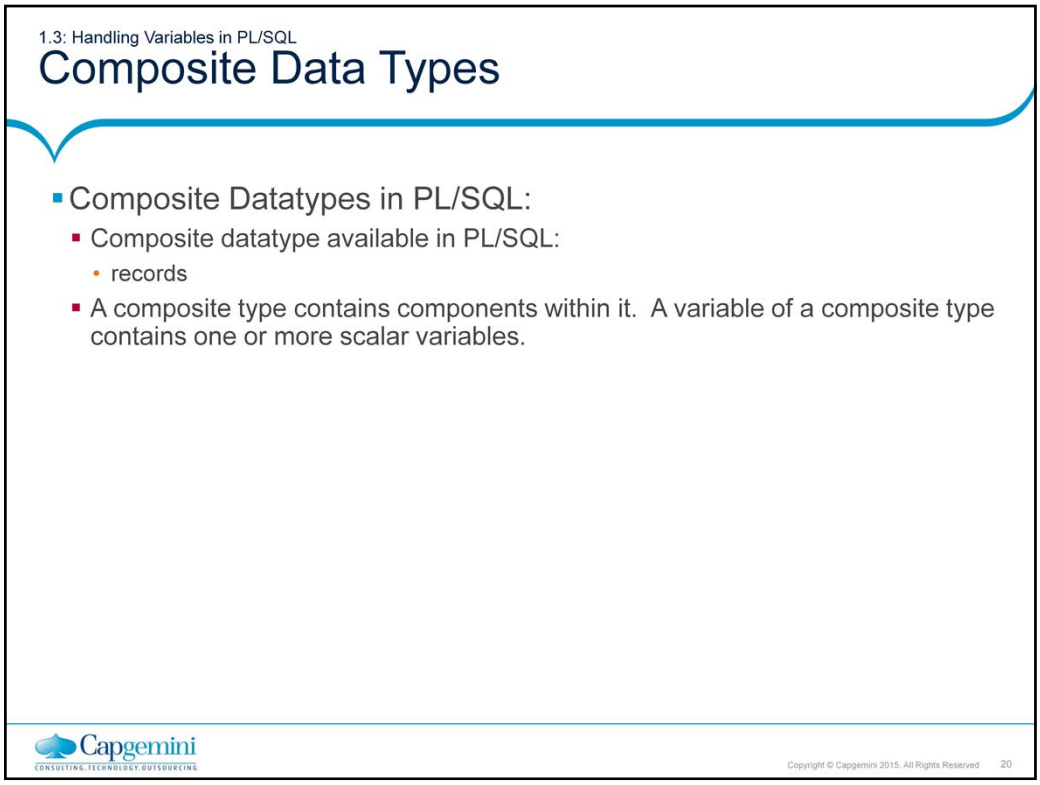
- %ROWTYPE is used to declare a compound variable, whose type is same as that of a row of a table.
- Columns in a row and corresponding fields in record should have same names and same datatypes. However, fields in a %ROWTYPE record do not inherit constraints, such as the NOT NULL, CHECK constraints, or default values.
- **Syntax:**
- V_Emprec emp%rowtype

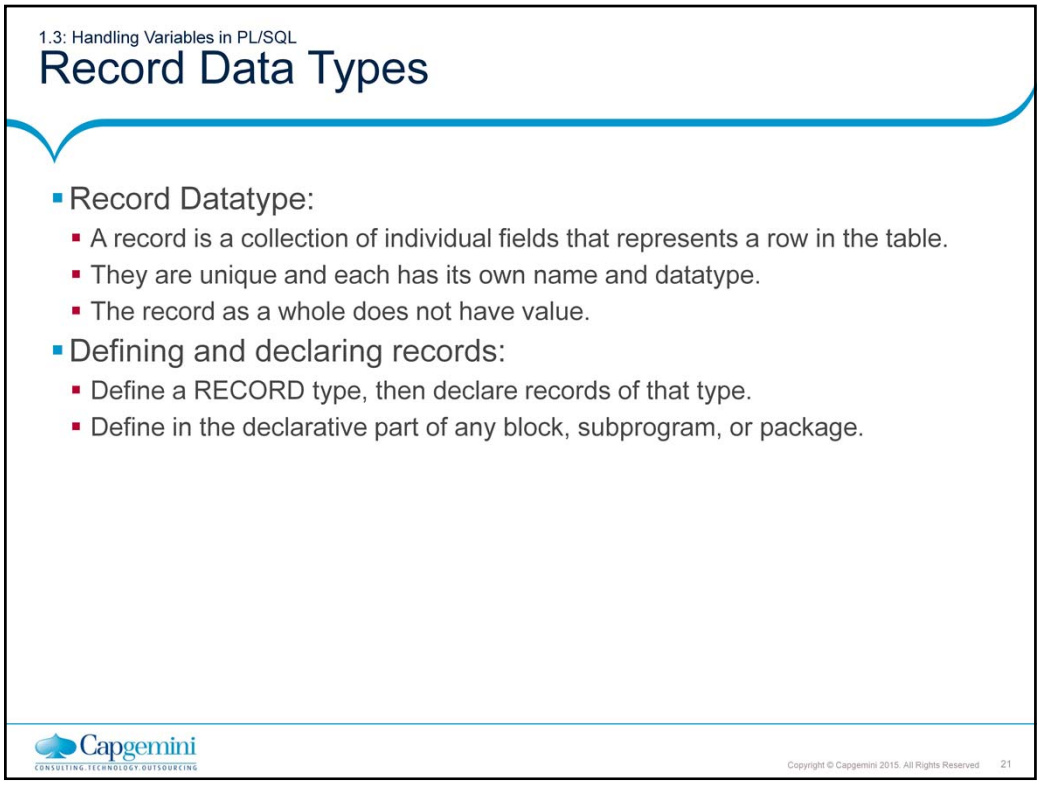
```
Var_Name        table_name%ROWTYPE;
V_Emprec        emp%ROWTYPE;
```

- where V_Emprec is a variable, which contains within itself as many variables, whose names and datatypes match those of the EMP table.
  - To access the Empno element of V_Emprec, use V_Emprec.empno;

```
DECLARE emprec emp%rowtype;
BEGIN
emprec.empno :=null;
emprec.deptno :=50;
dbms_output.put_line ('emprec.employee's
number'||emprec.empno);
END;
/
```

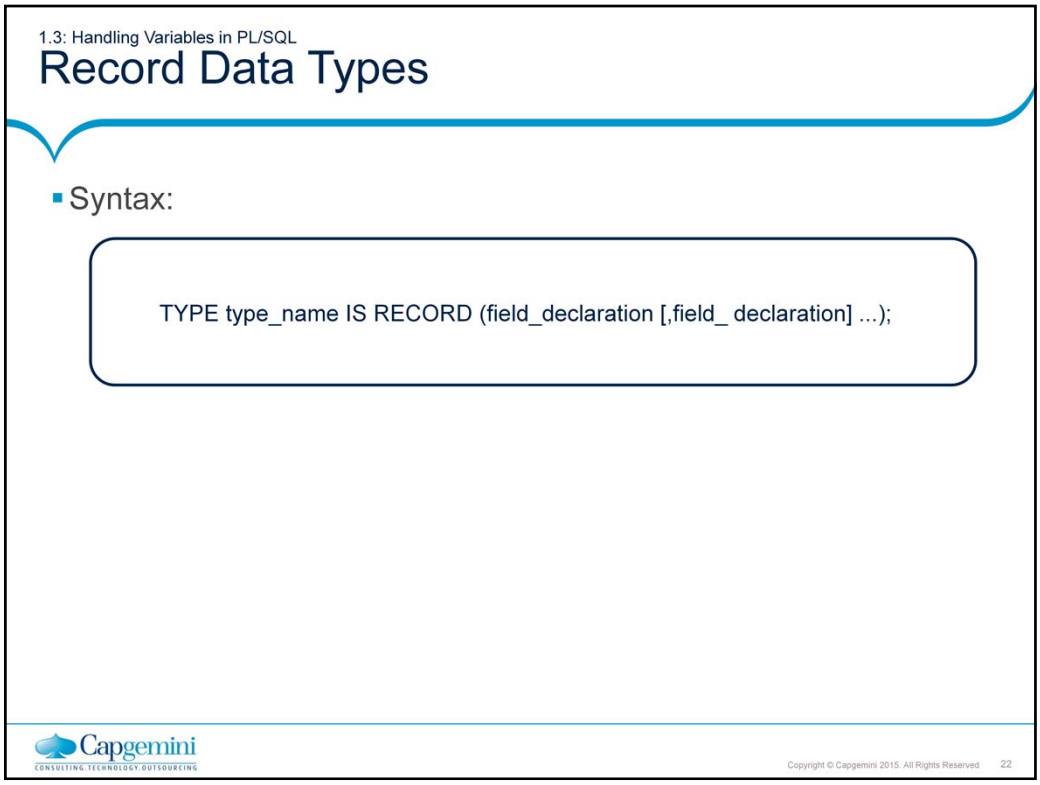

1.3: Handling Variables in PL/SQL
Inserting and Updating using records
Example:
DECLARE
dept_info department_master%ROWTYPE;
BEGIN
-- dept_code, dept_name are the table columns.
-- The record picks up these names from the %ROWTYPE.
dept_info.dept_code := 70;
dept_info.dept_name := 'PERSONNEL';
/*Using the %ROWTYPE means we can leave out the column list
(deptno, dname) from the INSERT statement. */
INSERT into department_master VALUES dept_info;
END;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
19


1.3: Handling Variables in PL/SQL
Composite Data Types
Composite Datatypes in PL/SQL:
Composite datatype available in PL/SQL:
records
A composite type contains components within it. A variable of a composite type contains one or more scalar variables.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 20



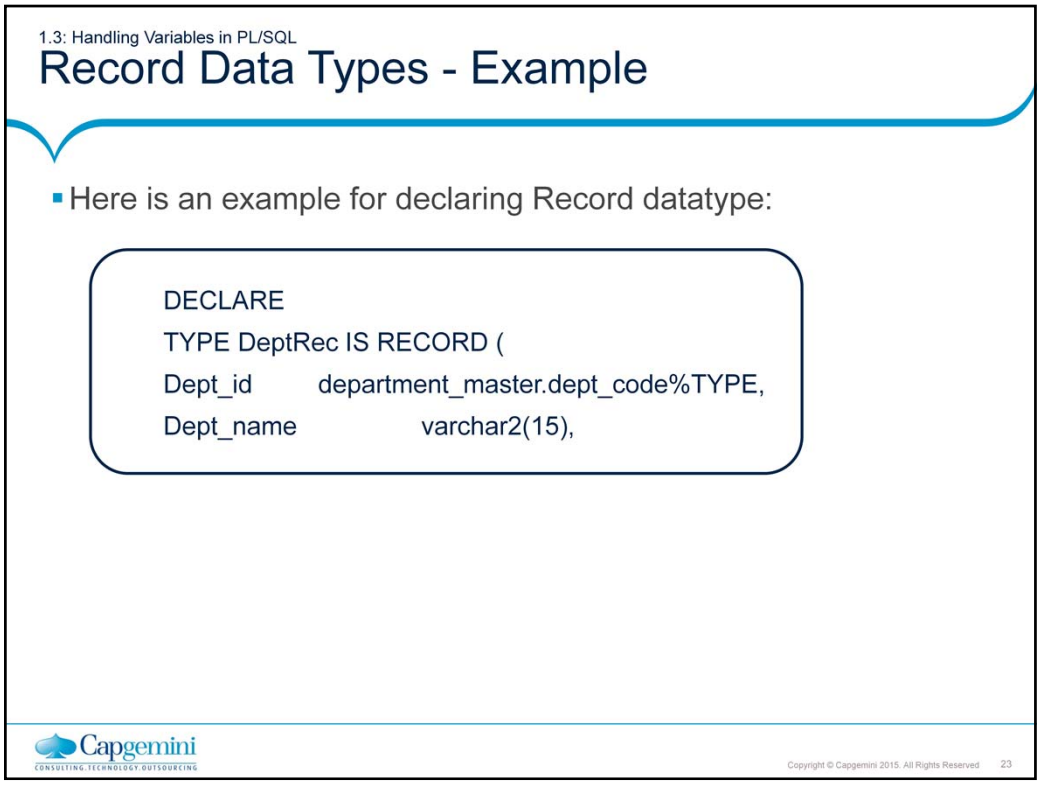
Record Datatype:

- A record is a collection of individual fields that represents a row in the table. They are unique and each has its own name and datatype. The record as a whole does not have value. By using records you can group the data into one structure and then manipulate this structure into one “entity” or “logical unit”. This helps to reduce coding and keeps the code easier to maintain and understand.



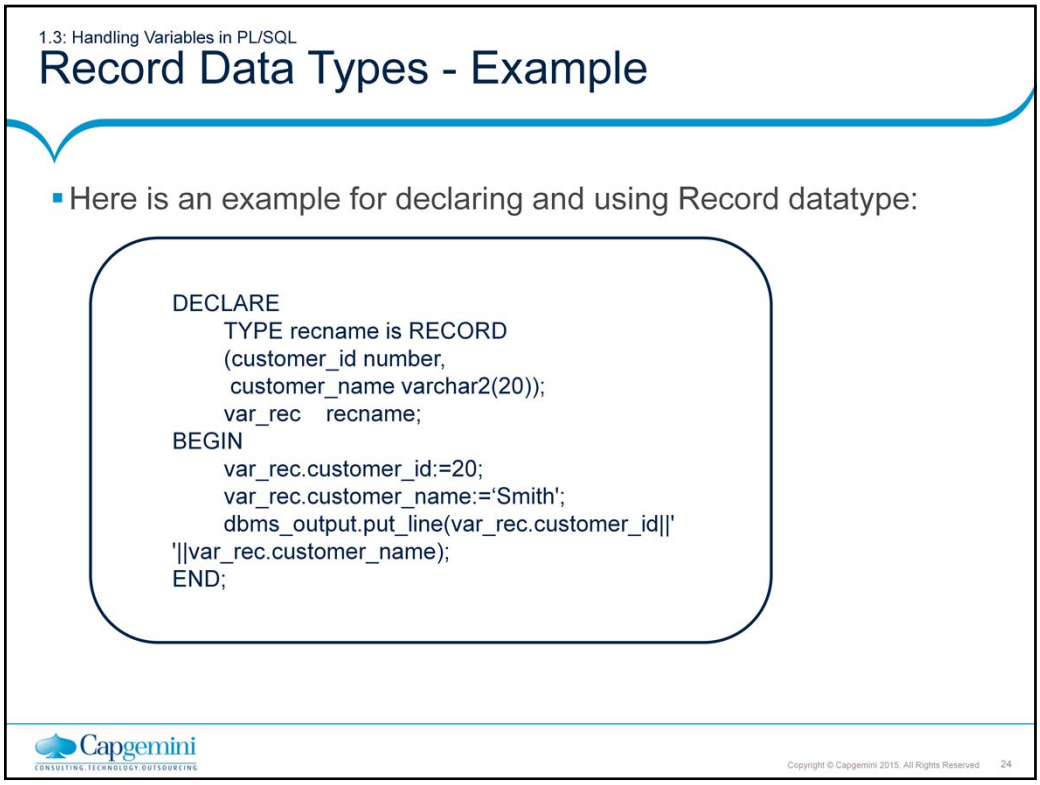
Defining and Declaring Records

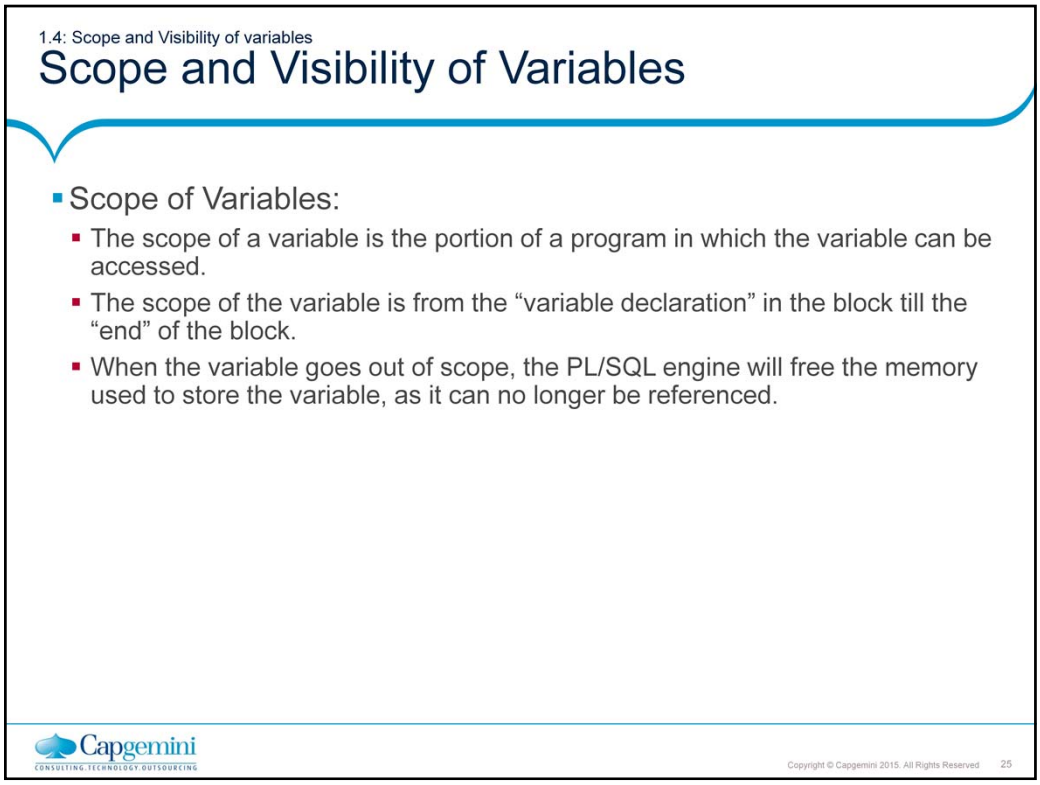
- To create records, you define a RECORD type, then declare records of that type. You can define RECORD types in the declarative part of any PL/SQL block, subprogram, or package by using the syntax.
- where field_declaration stands for:
  - •field_name field_type [[NOT NULL] {:= | DEFAULT} expression]
  - •type_name is a type specifier used later to declare records. You can use %TYPE and %ROWTYPE to specify field types.



Record Datatype (contd.):

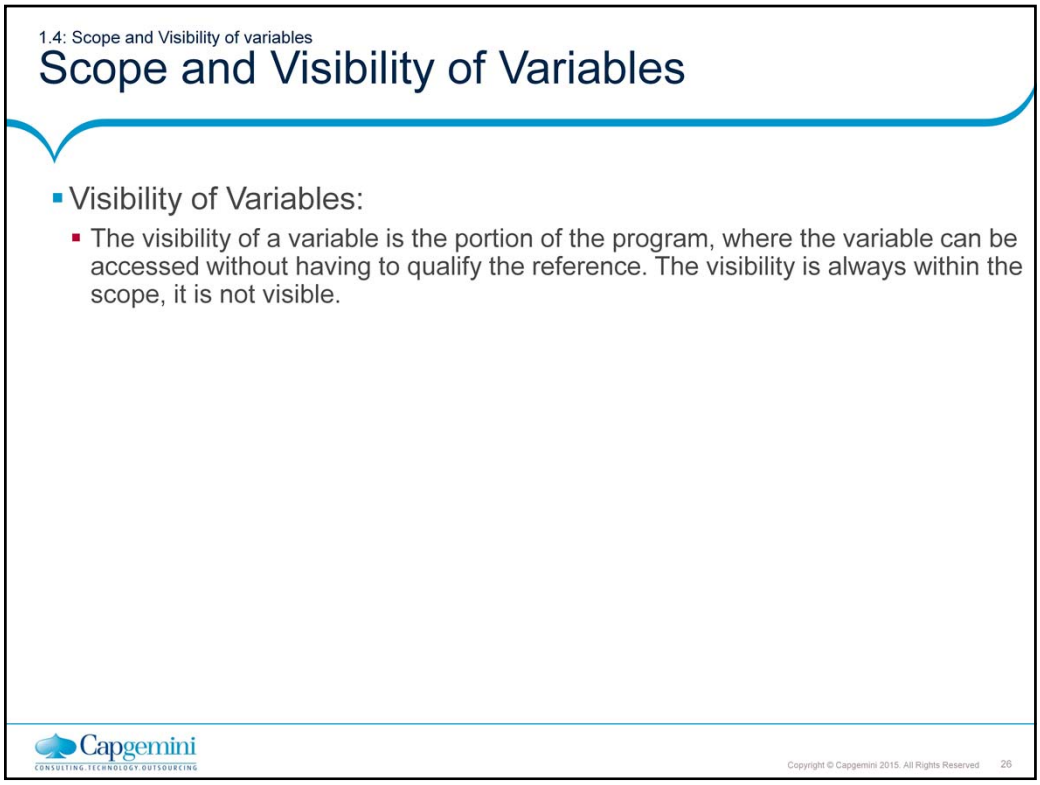
- **Field declarations** are like variable declarations.
- Each field has a unique name and specific datatype.
- Record members can be accessed by using "." (Dot) notation.
- The value of a record is actually a collection of values, each of which is of some simpler type. The attribute %ROWTYPE lets you declare a record that represents a row in a database table.
- After a record is declared, you can reference the record members directly by using the "." (Dot) notation. You can reference the fields in a record by indicating both the record and field names.
  For example: To reference an individual field, you use the dot notation DeptRec.deptno;
- You can assign expressions to a record.
  For example: DeptRec.deptno := 50;
- You can also pass a record type variable to a procedure as shown below: get_dept(DeptRec);


1.3: Handling Variables in PL/SQL
Record Data Types - Example
Here is an example for declaring and using Record datatype:
DECLARE
TYPE recname is RECORD
(customer_id number,
customer_name varchar2(20));
var_rec recname;
BEGIN
var_rec.customer_id:=20;
var_rec.customer_name:='Smith';
dbms_output.put_line(var_rec.customer_id||'
'||var_rec.customer_name);
END;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
24



Scope and Visibility of Variables:

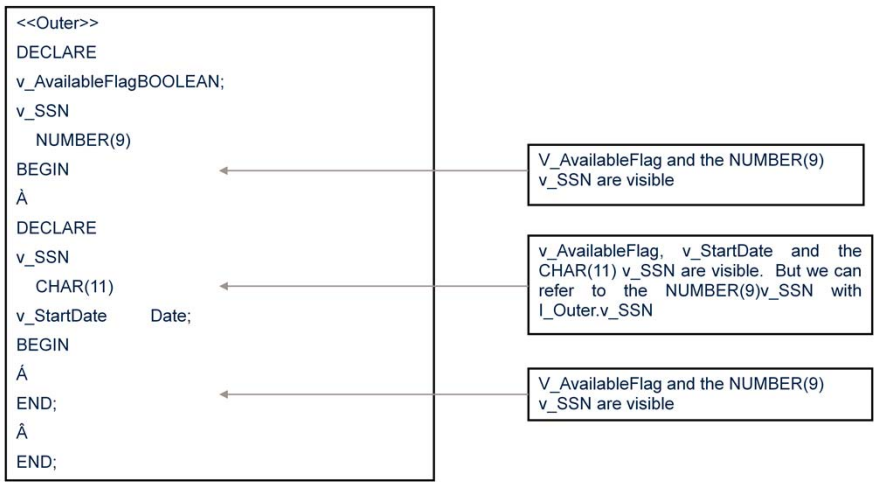
- References to an identifier are resolved according to its scope and visibility.
  - The scope of an identifier is that region of a program unit (block, subprogram, or package) from which you can reference the identifier.
  - An identifier is visible only in the regions from which you can reference the identifier using an unqualified name.
- Identifiers declared in a PL/SQL block are considered "local" to that "block" and "global" to all its "sub-blocks".
  - If a global identifier is re-declared in a sub-block, both identifiers remain in scope. However, the local identifier is visible within the sub-block only because you must use a qualified name to reference the global identifier.
- Although you cannot declare an identifier twice in the same block, you can declare the same identifier in two different blocks.
  - The two items represented by the identifier are "distinct", and any change in one does not affect the other. However, a block cannot reference identifiers declared in other blocks at the same level because those identifiers are neither local nor global to the block.


1.4: Scope and Visibility of variables
Scope and Visibility of Variables
Visibility of Variables:
The visibility of a variable is the portion of the program, where the variable can be accessed without having to qualify the reference. The visibility is always within the scope, it is not visible.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 26

1.4: Scope and Visibility of variables

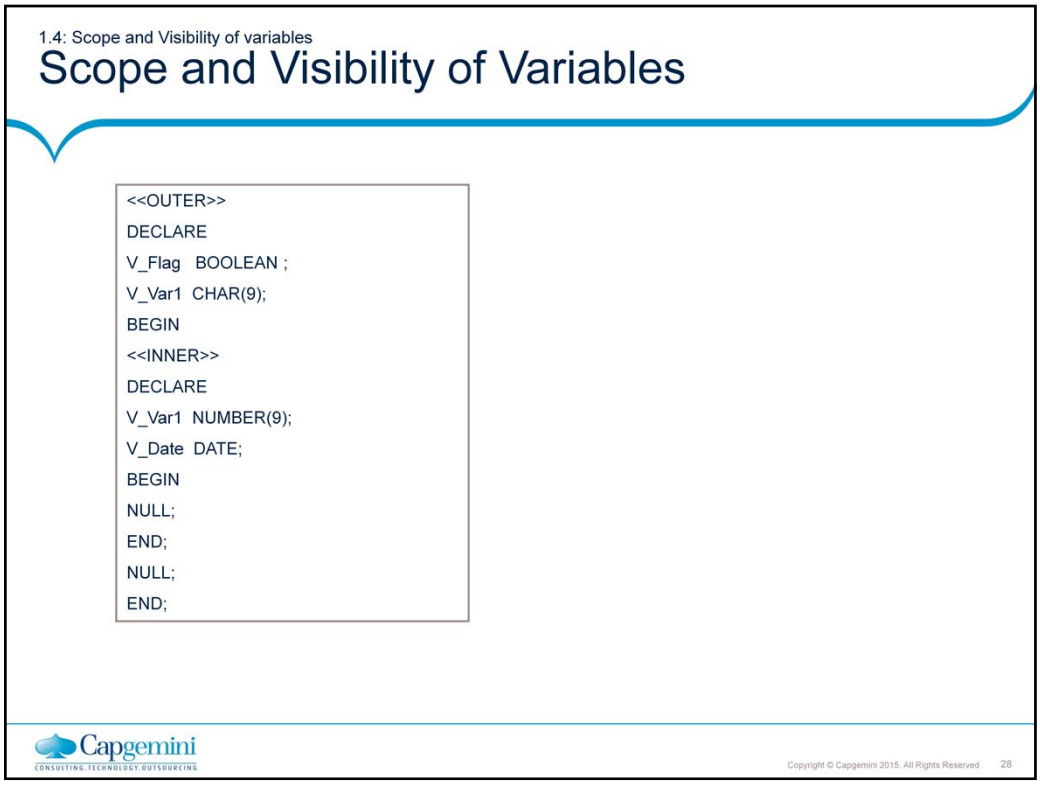
Scope and Visibility of Variables

- Pictorial representation of visibility of a variable:



Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 27


1.4: Scope and Visibility of variables
Scope and Visibility of Variables
<<OUTER>>
DECLARE
V_Flag BOOLEAN ;
V_Var1 CHAR(9);
BEGIN
<<INNER>>
DECLARE
V_Var1 NUMBER(9);
V_Date DATE;
BEGIN
NULL;
END;
NULL;
END;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
28

1.5: SQL in PL/SQL

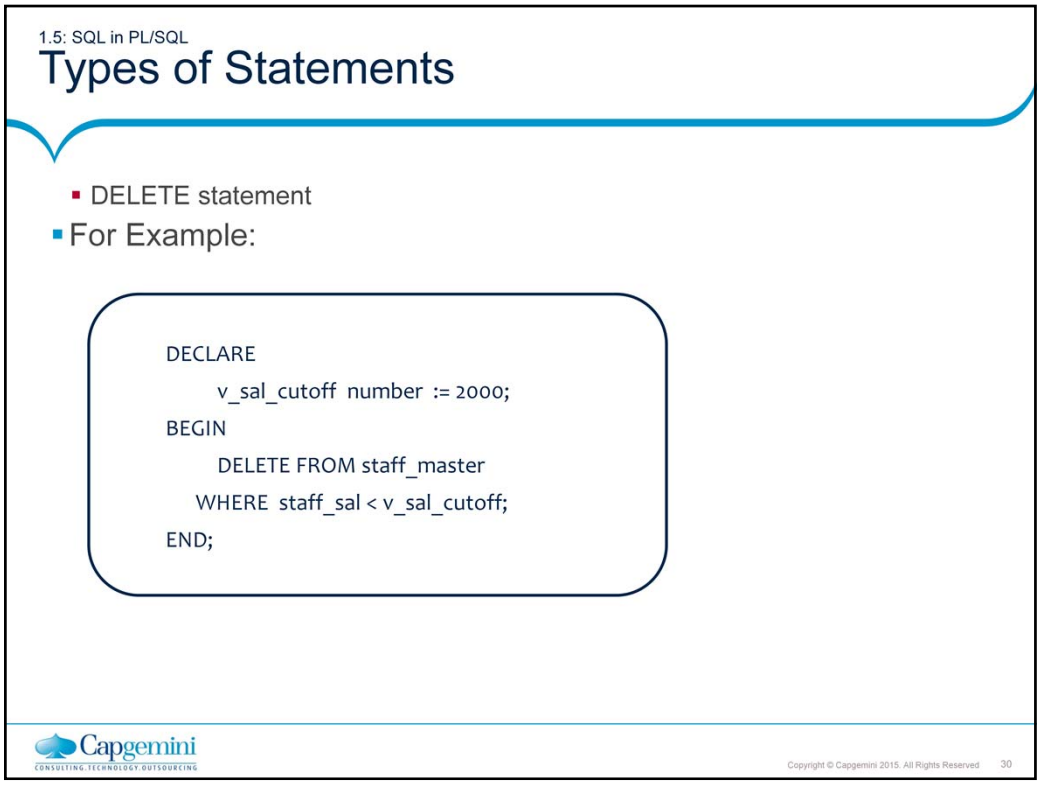
Types of Statements

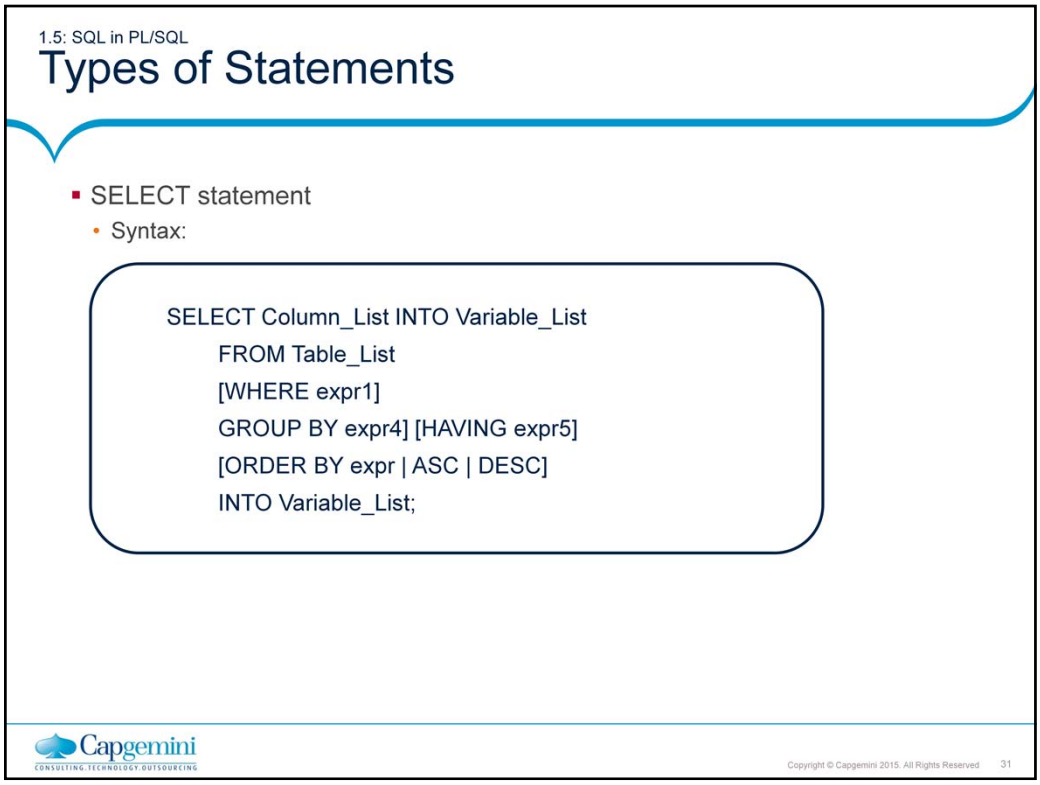
- Given below are some of the SQL statements that are used in PL/SQL:
  - INSERT statement
    - The syntax for the INSERT statement remains the same as in SQL-INSERT.
    - For example:

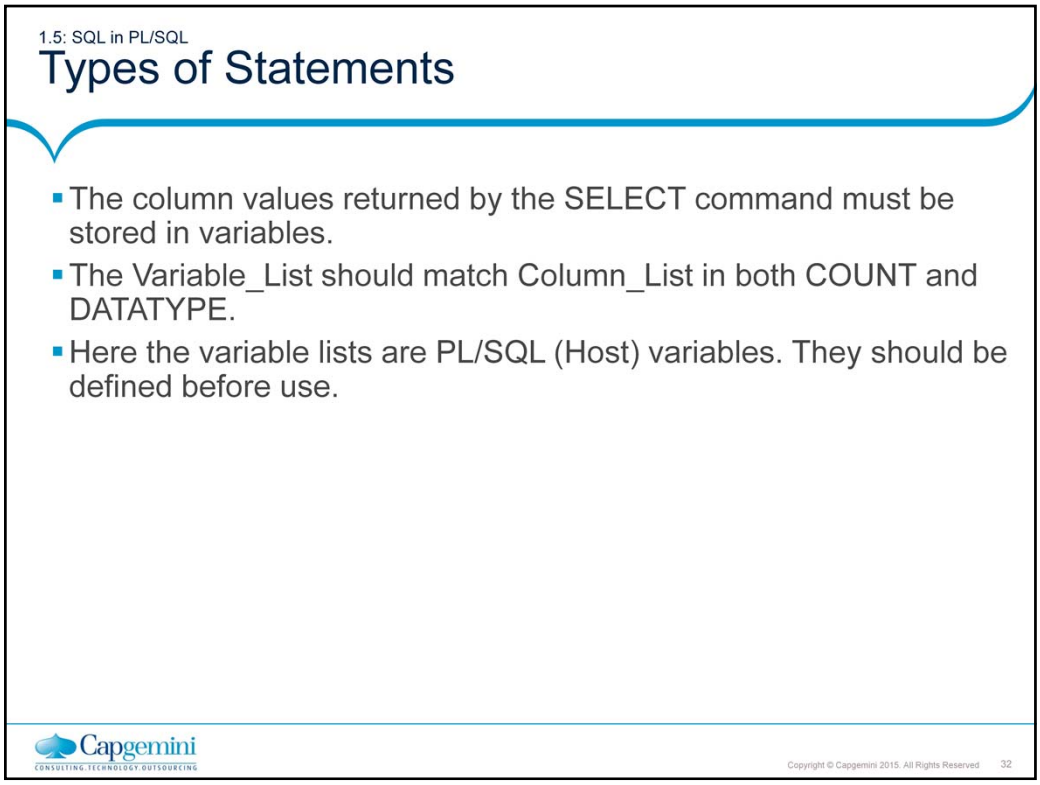
```
DECLARE
      v_dname varchar2(15) := 'Accounts';
BEGIN
       INSERT into department_master
      VALUES (50, v_dname);
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29


1.5: SQL in PL/SQL
Types of Statements
DELETE statement
For Example:
DECLARE
v_sal_cutoff number := 2000;
BEGIN
DELETE FROM staff_master
WHERE staff_sal < v_sal_cutoff;
END;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
30

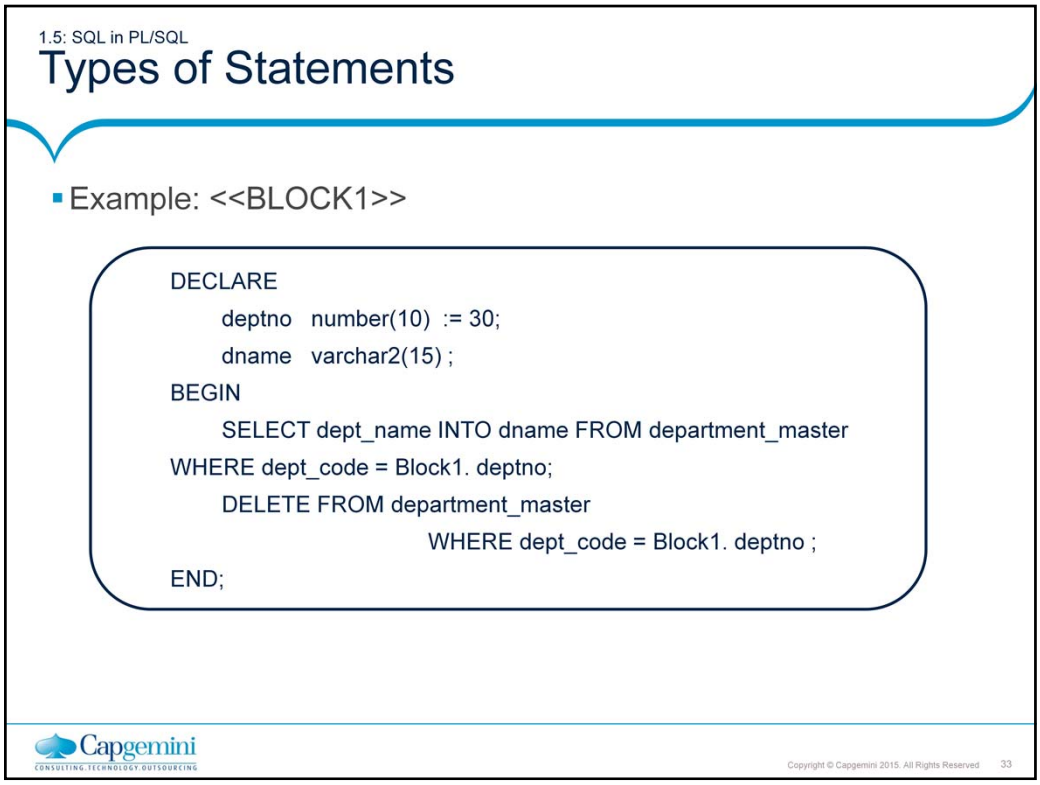

1.5: SQL in PL/SQL
Types of Statements
SELECT statement
Syntax:
SELECT Column_List INTO Variable_List
FROM Table_List
[WHERE expr1]
GROUP BY expr4] [HAVING expr5]
[ORDER BY expr | ASC | DESC]
INTO Variable_List;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
31



SELECT Statement:

Note:

- The SELECT clause is used if the selected row must be modified through a DELETE or UPDATE command.



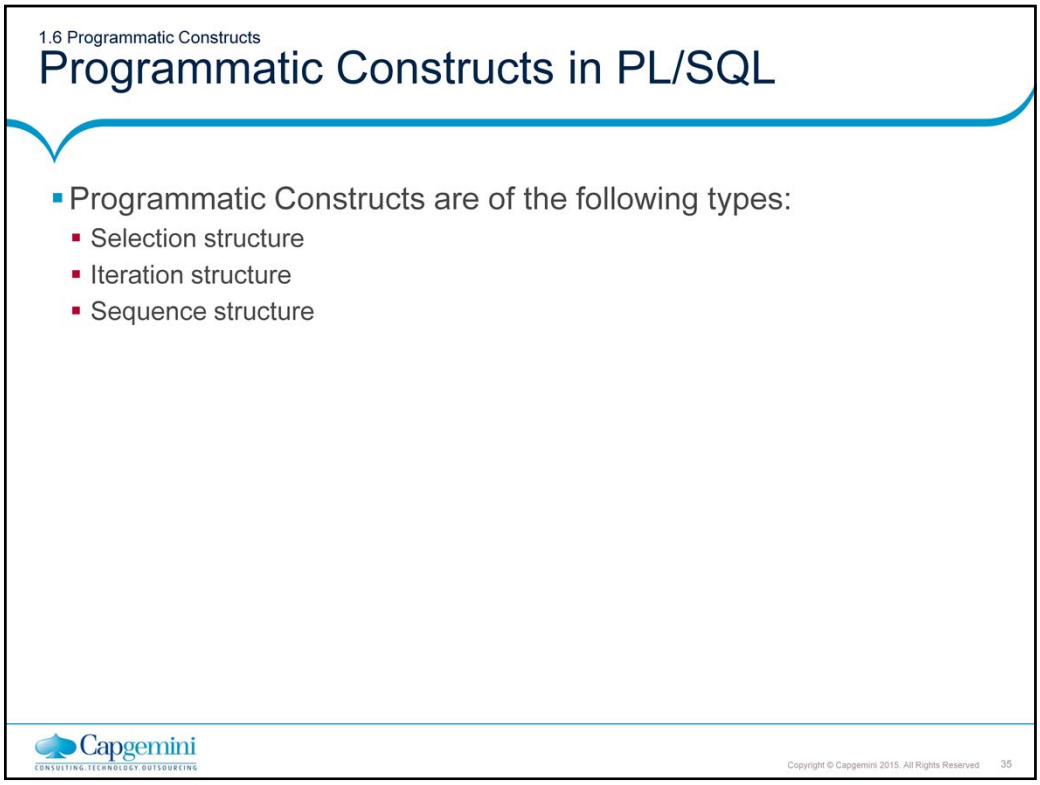
SELECT statement (contd.):

SELECT statement (contd.):

More examples

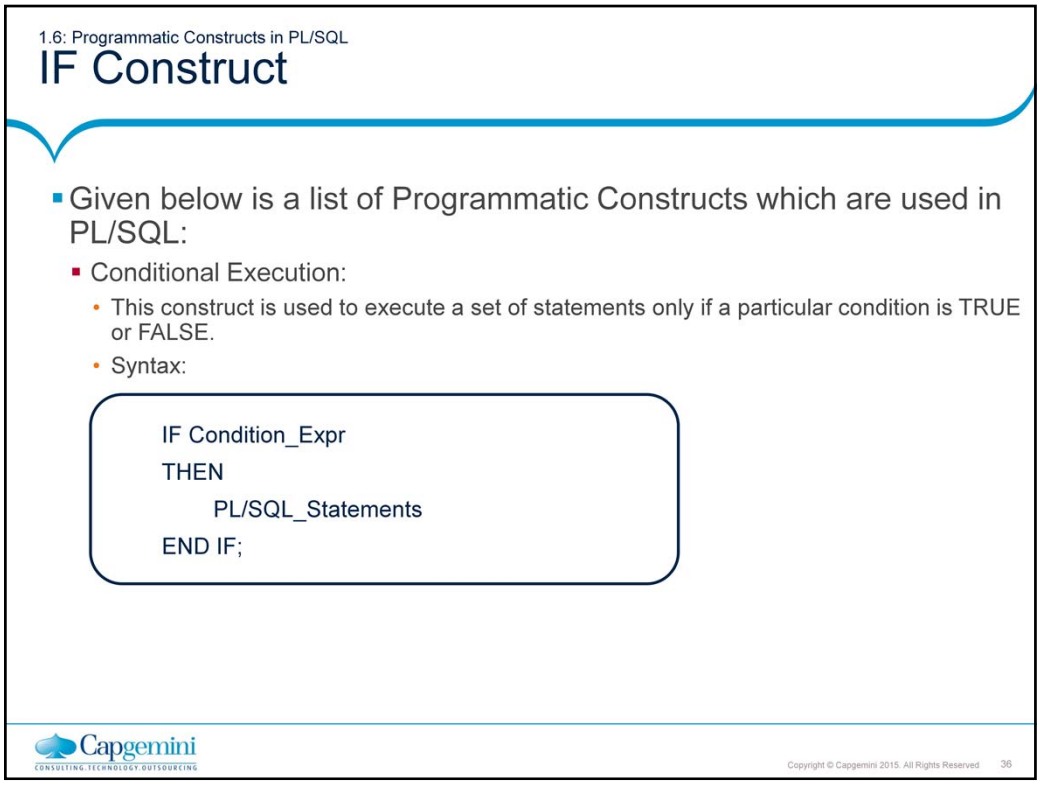
- Here the SELECT statement will select names of all the departments and not only deptno. 30.
- The DELETE statement will delete all the employees.
  - This happens because when the PL/SQL engine sees a condition expr1 = expr2, the expr1 and expr2 are first checked to see whether they match the database columns first and then the PL/SQL variables.
  - So in the above example, where you see deptno = deptno, both are treated as database columns, and the condition will become TRUE for every row of the table.
- If a block has a label, then variables with same names as database columns can be used by using <<blockname>>. Variable_Name notation.
- It is not a good programming practice to use same names for PL/SQL variables and database columns.

```
DECLARE
dept_code          number(10)  := 30;
v_dname            varchar2(15) ;
BEGIN
SELECT dept_name INTO v_dname FROM
department_master  WHERE dept_code=dept_code
DELETE FROM department_master
        WHERE dept_code = dept_code ;
END;
```



Programming Constructs:

- The **selection structure** tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is TRUE or FALSE.
  - A condition is any variable or expression that returns a Boolean value (TRUE or FALSE).
- The **iteration structure** executes a sequence of statements repeatedly as long as a condition holds true.
- The **sequence structure** simply executes a sequence of statements in the order in which they occur.

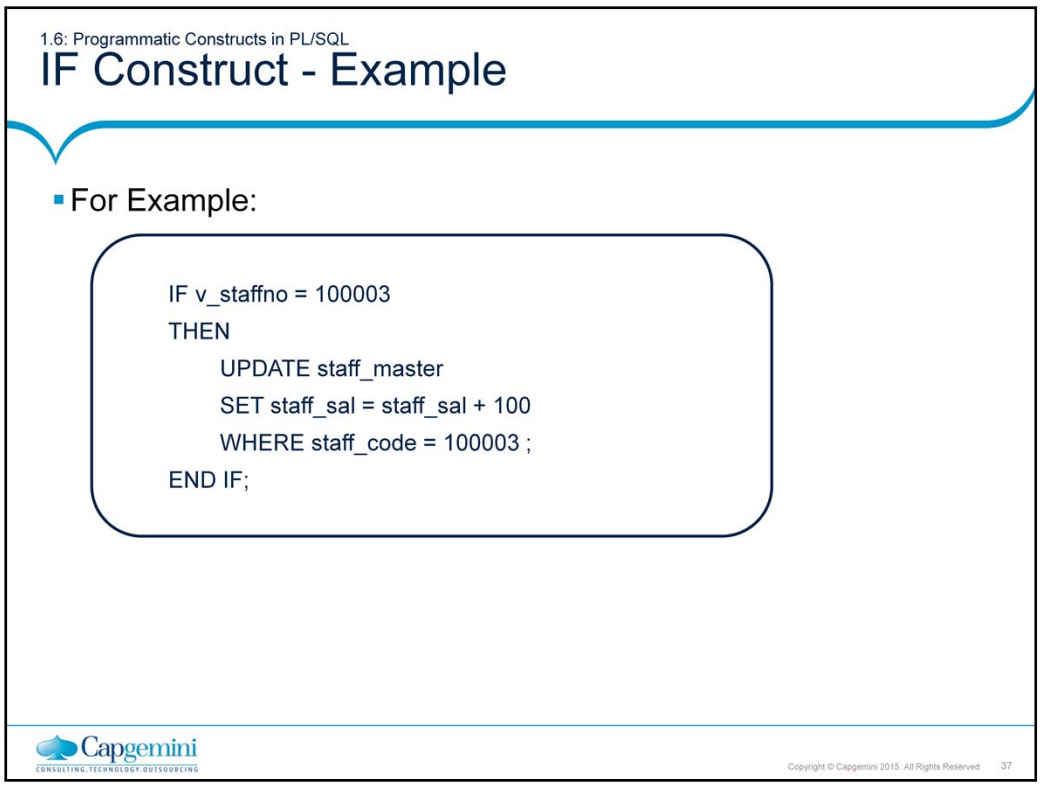


Programmatic Constructs (contd.)

Conditional Execution:

- Conditional execution is of the following type:
  - IF-THEN-END IF
  - IF-THEN-ELSE-END IF
  - IF-THEN-ELSIF-END IF
- Conditional Execution construct is used to execute a set of statements only if a particular condition is TRUE or FALSE.

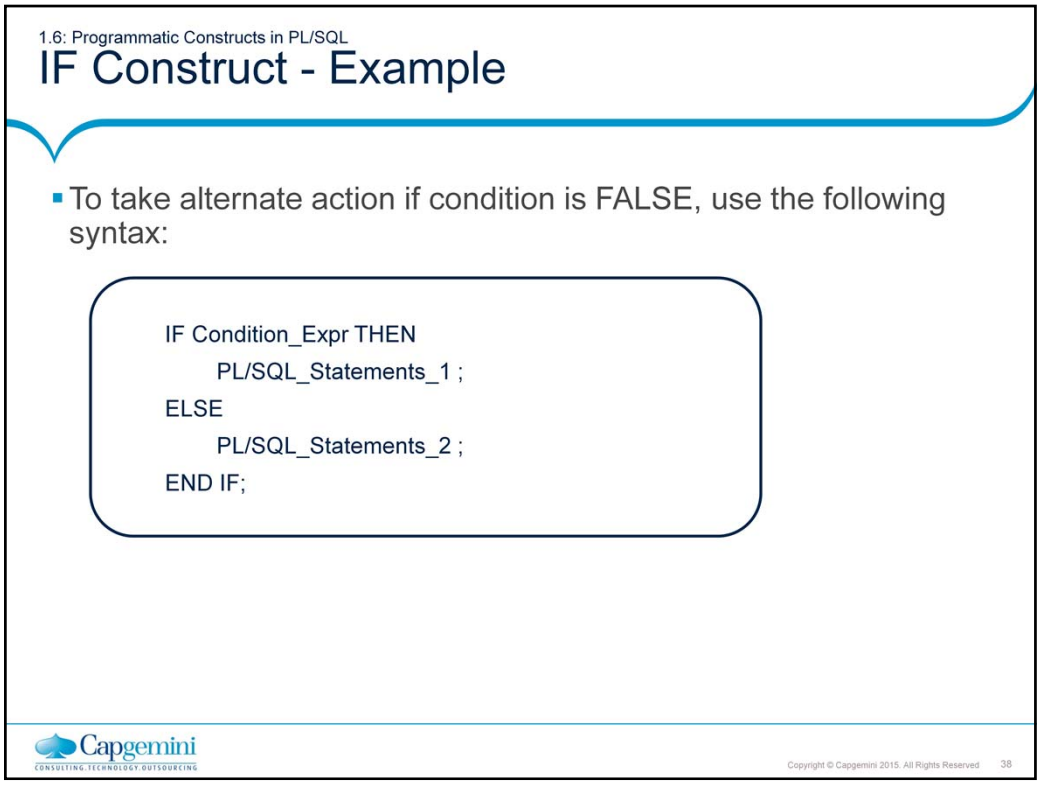
contd.



Programmatic Constructs (contd.)

Conditional Execution (contd.):

- As shown in the example in the slide, when the condition evaluates to TRUE, the PL/SQL statements are executed, otherwise the statement following END IF is executed.
- UPDATE statement is executed only if value of v_staffno variable equals 100003.
- PL/SQL allows many variations for the IF – END IF construct.

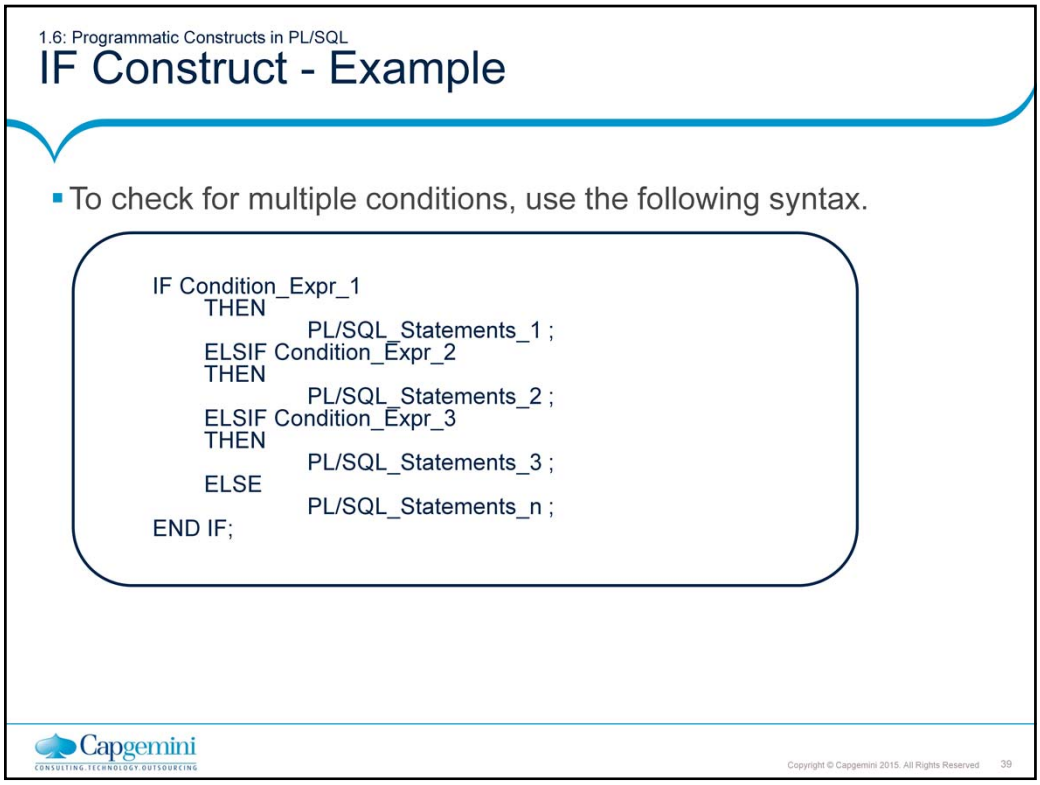


Programmatic Constructs (contd.)

Conditional Execution (contd.):

Note:

- When the condition evaluates to TRUE, the PL/SQL_Statements_1 is executed, otherwise PL/SQL_Statements_2 is executed.
- The above syntax checks **only one** condition, namely Condition_Expr.



Programmatic Constructs (contd.)
Conditional Execution (contd.):

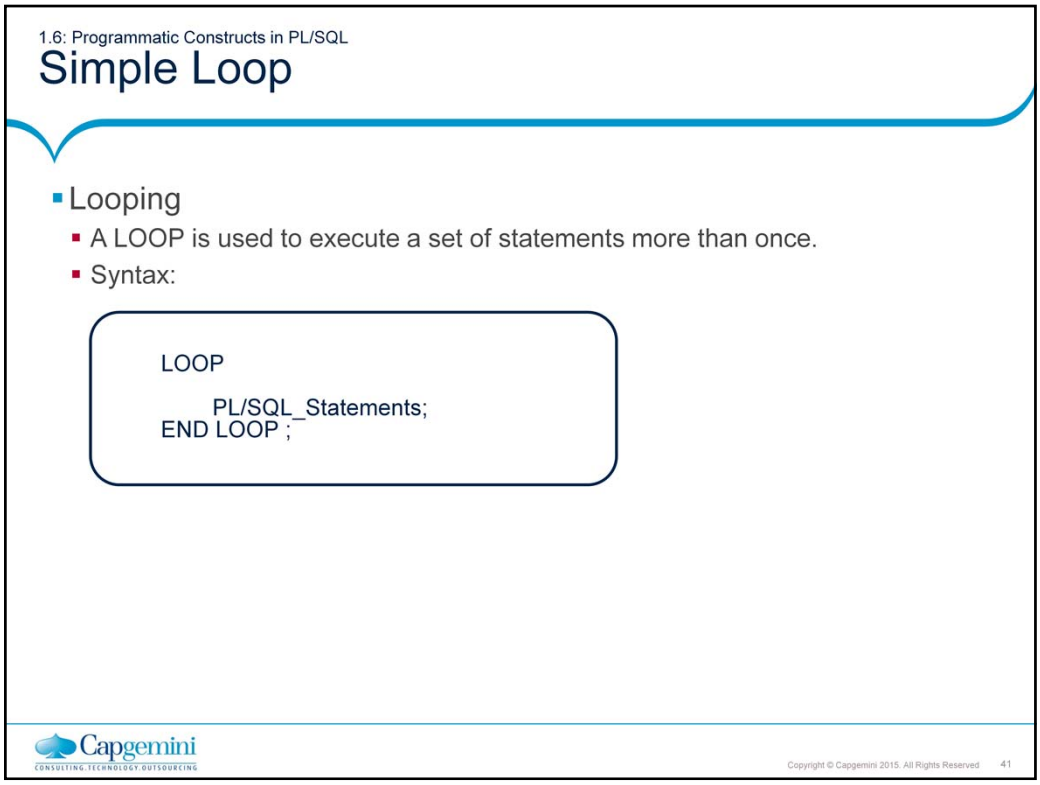
```
DECLARE
    D VARCHAR2(3): = TO_CHAR(SYSDATE, 'DY')
BEGIN
    IF   D= 'SAT'    THEN
        DBMS_OUTPUT.PUT_LINE('ENJOY YOUR
WEEKEND');
    ELSIF  D= 'SUN'    THEN
        DBMS_OUTPUT.PUT_LINE('ENJOY YOUR
WEEKEND');
    ELSE
        DBMS_OUTPUT.PUT_LINE('HAVE A NICE
DAY');
    END IF;
END;
```

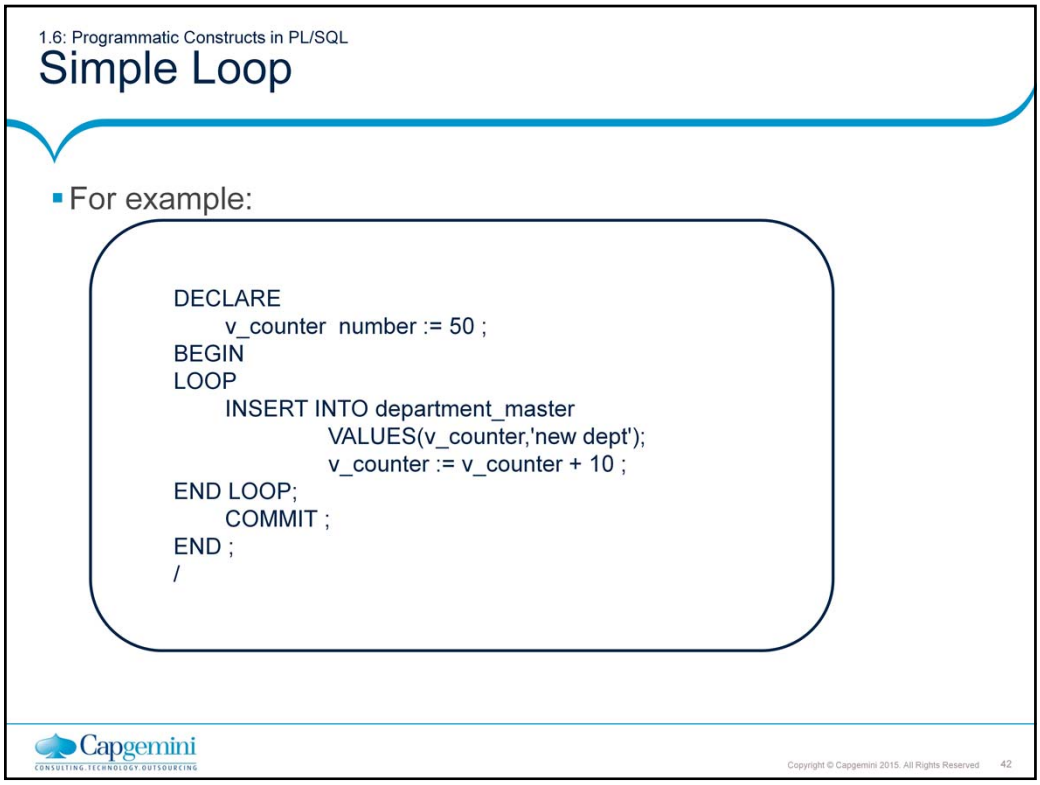
Programmatic Constructs (contd.)

Conditional Execution (contd.):

- As every condition must have at least one statement, NULL statement can be used as filler.
- NULL command does nothing.
- Sometimes NULL is used in a condition merely to indicate that such a condition has been taken into consideration, as well.
- Conditions for NULL are checked through IS NULL and IS NOT NULL predicates.

```
IF Condition_Expr_1 THEN

PL/SQL_Statements_1 ;
ELSIF Condition_Expr_2 THEN

PL/SQL_Statements_2 ;
ELSIF Condition_Expr_3 THEN
        Null;
END IF;
```


1.6: Programmatic Constructs in PL/SQL
Simple Loop
Looping
A LOOP is used to execute a set of statements more than once.
Syntax:
LOOP
PL/SQL_Statements;
END LOOP ;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
41



Programmatic Constructs (contd.)

Looping

- The example shown in the slide is an endless loop.
- When LOOP ENDLOOP is used in the above format, then an exit path must necessarily be provided. This is discussed in the following slide.

1.6: Programmatic Constructs in PL/SQL

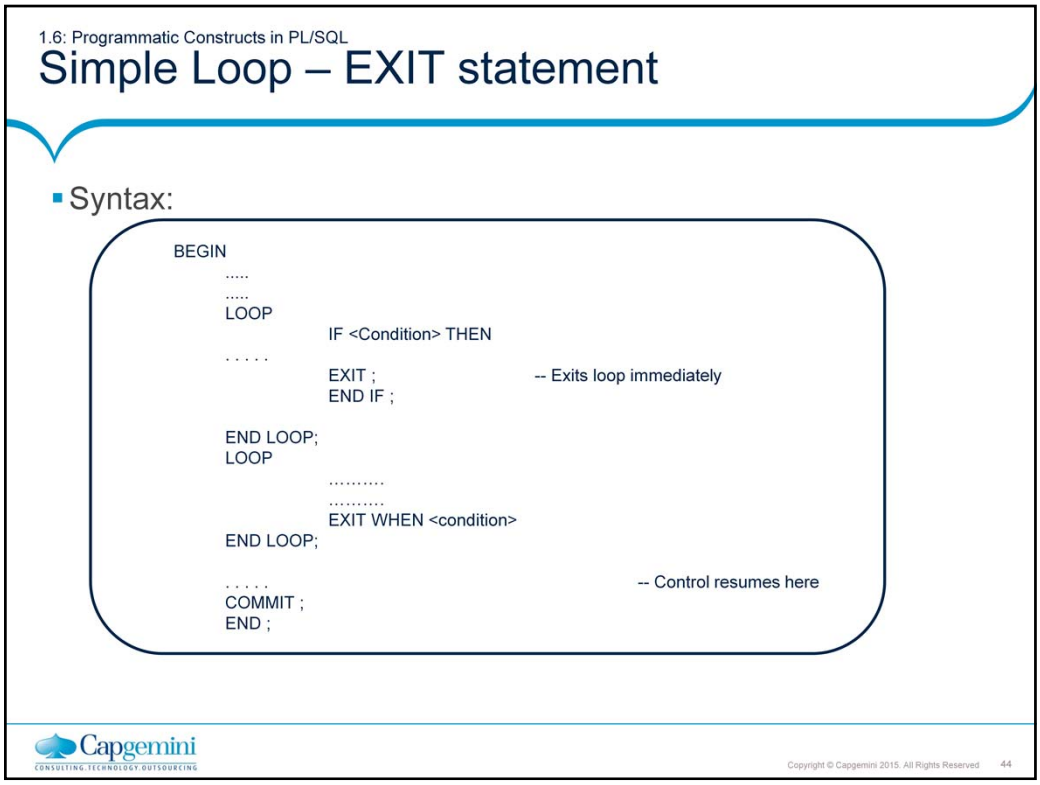
Simple Loop – EXIT statement

- EXIT
  - Exit path is provided by using EXIT or EXIT WHEN commands.
  - EXIT is an unconditional exit. Control is transferred to the statement following END LOOP, when the execution flow reaches the EXIT statement.

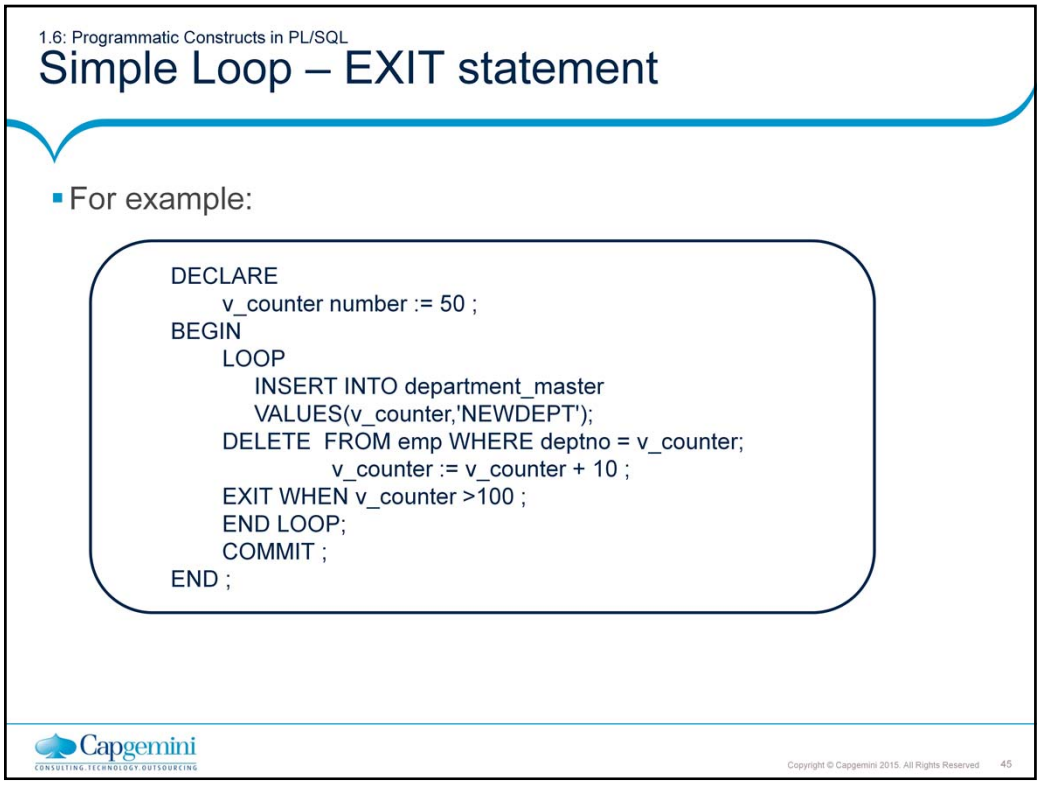
contd.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

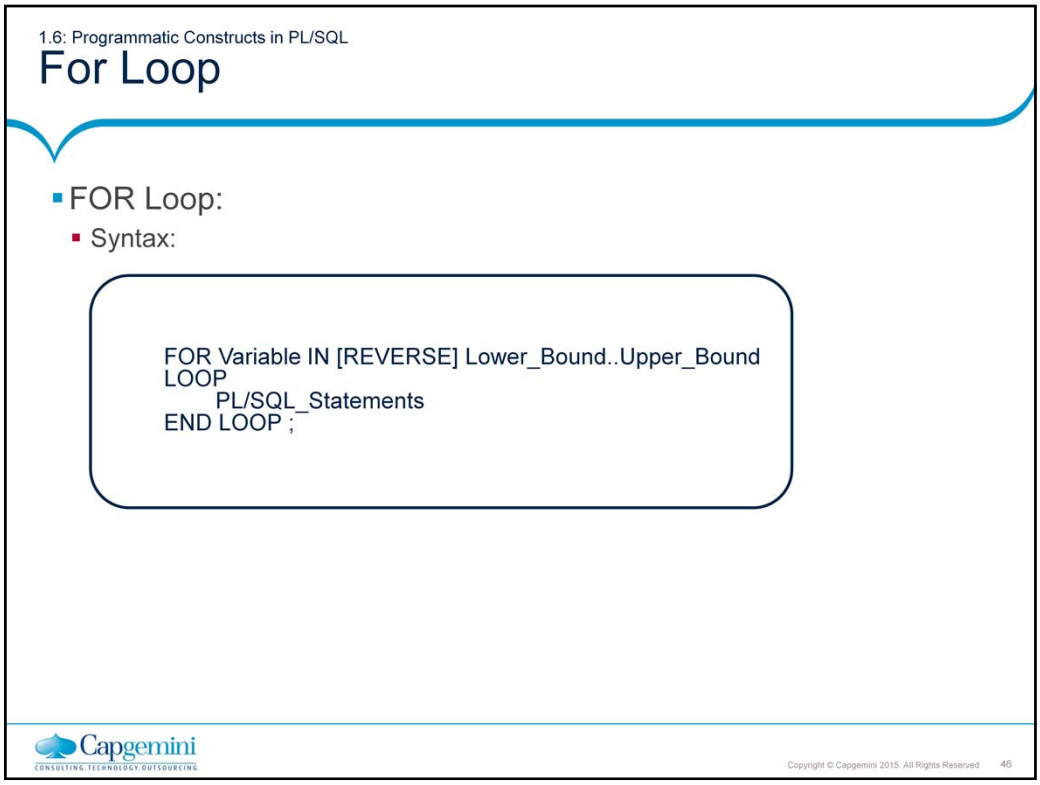
 43



Note:
EXIT WHEN is used for conditional exit out of the loop.



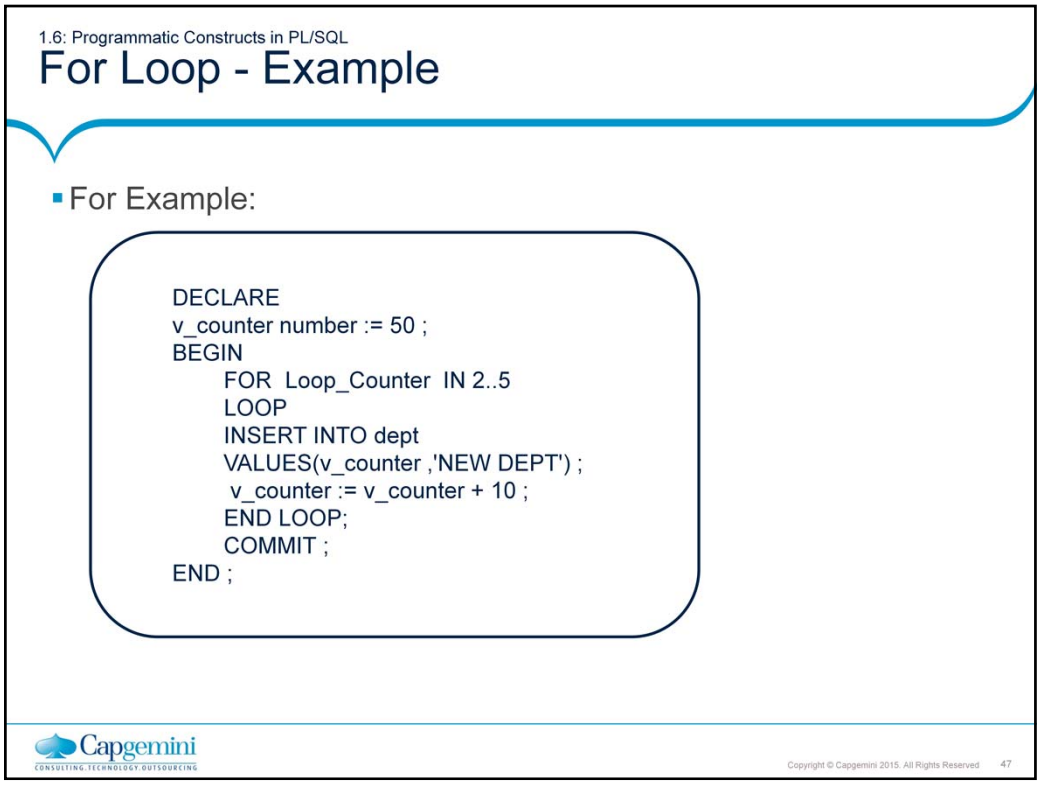
Note:
LOOP.. END LOOP can be used in conjunction with FOR and WHILE for better control on looping.



Programmatic Constructs (contd.)

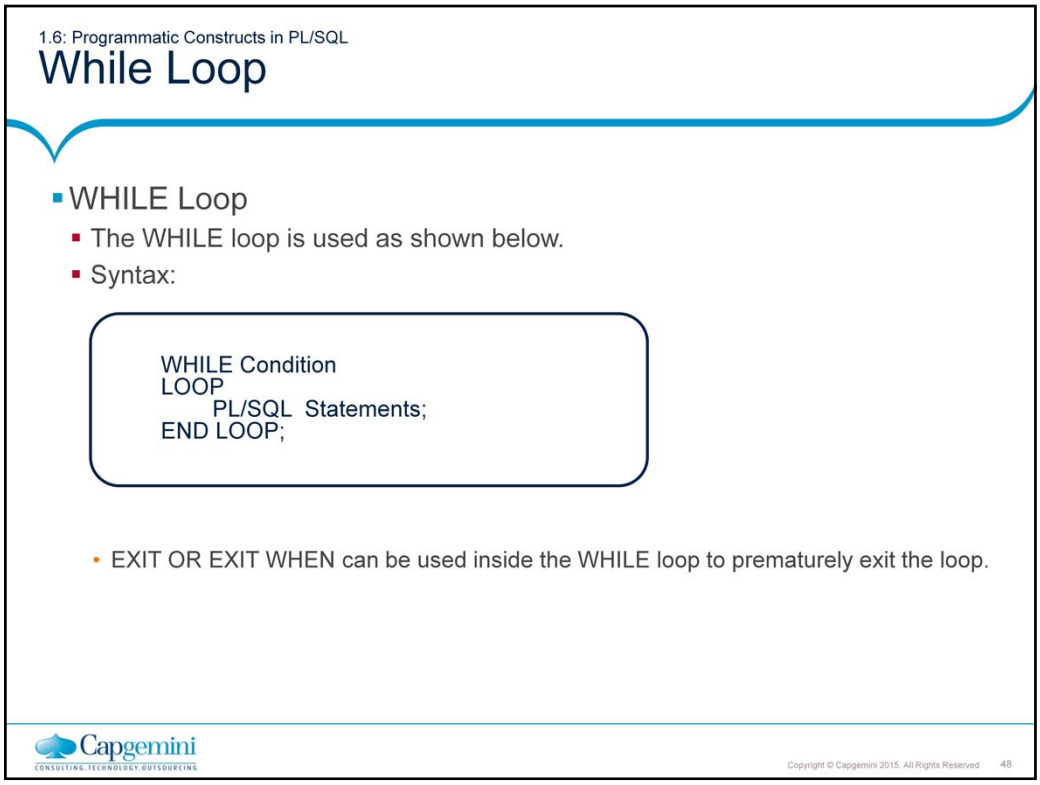
FOR Loop:

- FOR loop is used for executing the loop a fixed number of times. The number of times the loop will execute equals the following:
  - Upper_Bound - Lower_Bound + 1.
- Upper_Bound and Lower_Bound must be integers.
- Upper_Bound must be equal to or greater than Lower_Bound.
- Variables in FOR loop need not be explicitly declared.
  - Variables take values starting at a Lower_Bound and ending at a Upper_Bound.
  - The variable value is incremented by 1, every time the loop reaches the bottom.
  - When the variable value becomes equal to the Upper_Bound, then the loop executes and exits.
- When REVERSE is used, then the variable takes values starting at Upper_Bound and ending at Lower_Bound.
- Value of the variable is decremented each time the loop reaches the bottom.



Programmatic Constructs (contd.)

- In the example in the above slide, the loop will be executed (5 - 2 + 1) = 4 times.
- A Loop_Counter variable can also be used inside the loop, if required.
- Lower_Bound and/or Upper_Bound can be integer expressions, as well.

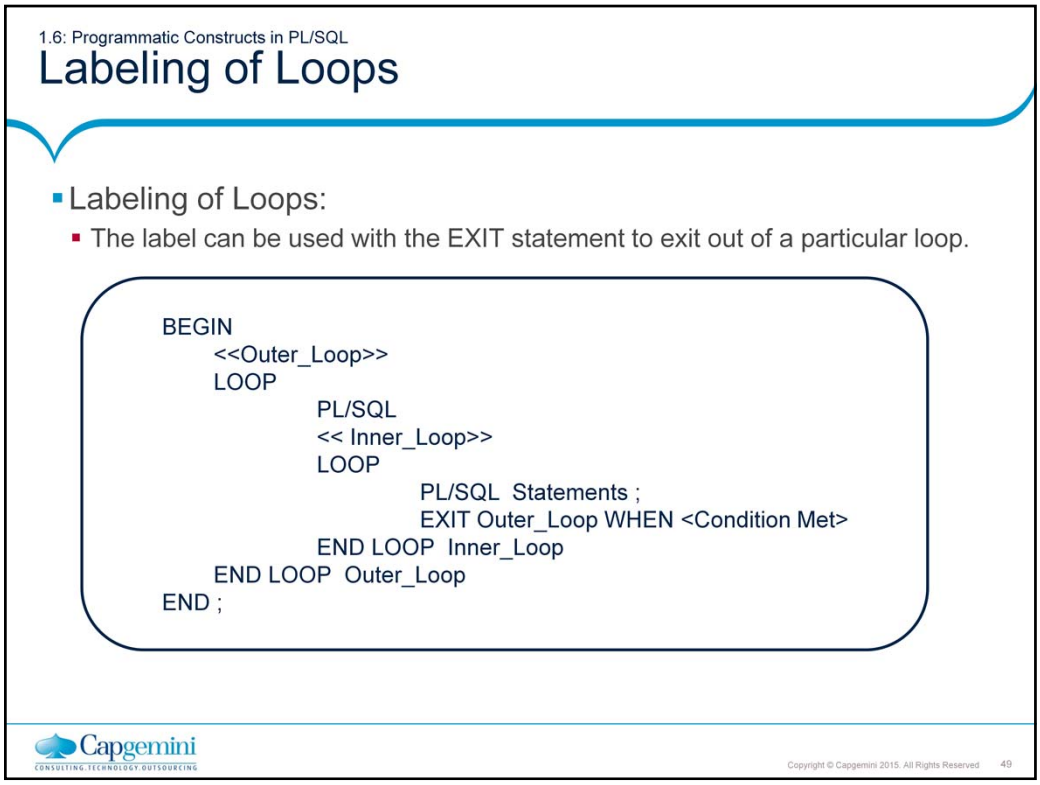


Programmatic Constructs (contd.)

WHILE Loop:

Example:

```
DECLARE
     ctr number := 1;
BEGIN
     WHILE ctr <= 10
     LOOP
     dbms_output.put_line(ctr);
          ctr := ctr+1;
     END LOOP;
END;
/
```



Programmatic Constructs (contd.)

Labeling of Loops:

- Loops themselves can be labeled as in the case of blocks.
- The label can be used with the EXIT statement to exit out of a particular loop.

Summary

- In this lesson, you have learnt:
  - PL/SQL is a procedural extension to SQL.
  - PL/SQL exhibits a block structure, different block types being: Anonymous, Procedure, and Function.
  - While declaring variables in PL/SQL:
    - declare and initialize variables within the declaration section
    - assign new values to variables within the executable section

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 50

Summary

In this lesson, you have learnt:

- PL/SQL is a procedural extension to SQL.
- PL/SQL exhibits a block structure, different block types being: Anonymous, Procedure, and Function.
- While declaring variables in PL/SQL:
  - declare and initialize variables within the declaration section
  - assign new values to variables within the executable section

Summary

- Different types of PL/SQL Variables are: Scalar, Composite, Reference, LOB
- Scope of a variable: It is the portion of a program in which the variable can be accessed.
- Visibility of a variable: It is the portion of the program, where the variable can be accessed without having to qualify the reference.
- Different programmatic constructs in PL/SQL are Selection structure, Iteration structure, Sequence structure

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 52

Review & Question

- Question 1: A record is a collection of individual fields that represents a row in the table.
  - True/ False
- Question 2: %ROWTYPE is used to declare a variable with the same datatype as a column of a specific table.
  - True / False
- Question 3: While using FOR loop, Upper_Bound, and Lower_Bound must be integers.
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 53

Oracle Basics (PL/SQL)

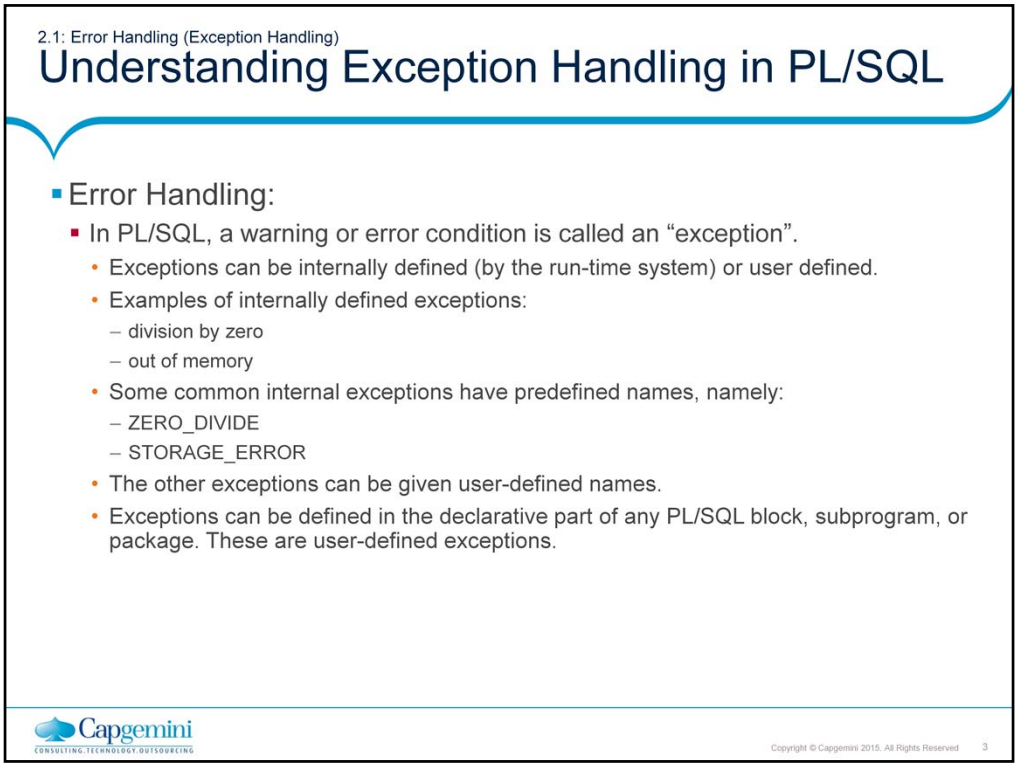
Lesson 02 Exception Handling

Lesson Objectives

- To understand the following topics:
  - Error Handling
  - Declaring Exceptions
    - Predefined Exceptions
    - User Defined Exceptions
      - Raising Exceptions
  - OTHERS exception handler

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

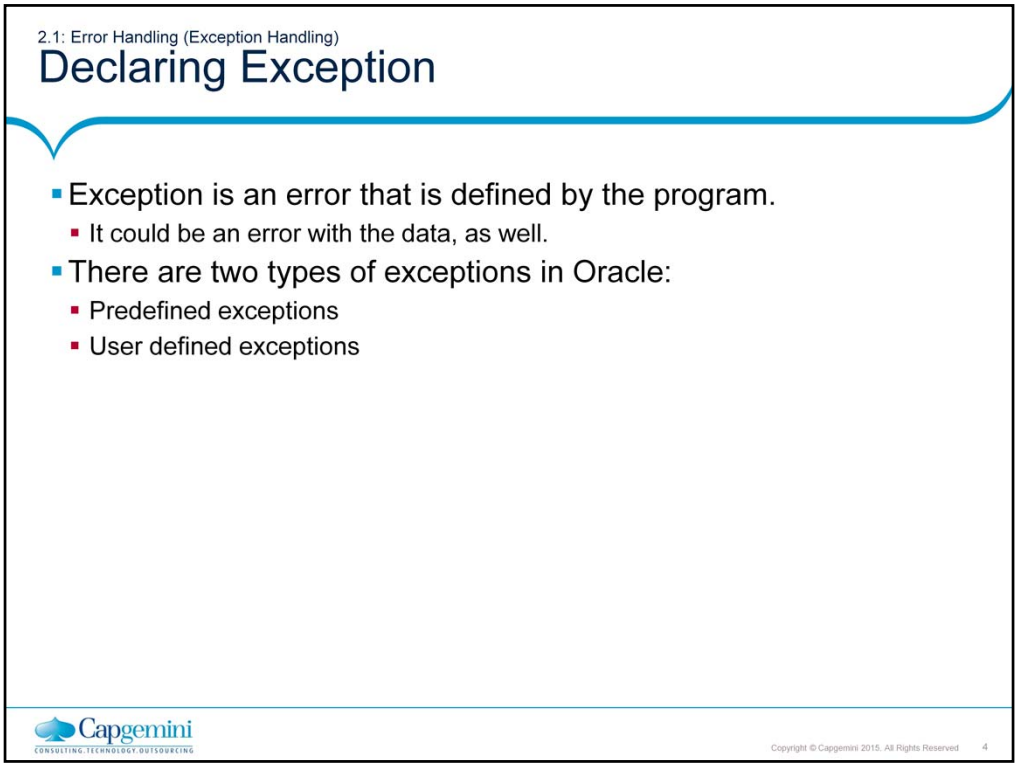


Error Handling:

- A good programming language should provide capabilities of handling errors and recovering from them if possible.
- PL/SQL implements Error Handling via “exceptions” and “exception handlers”.

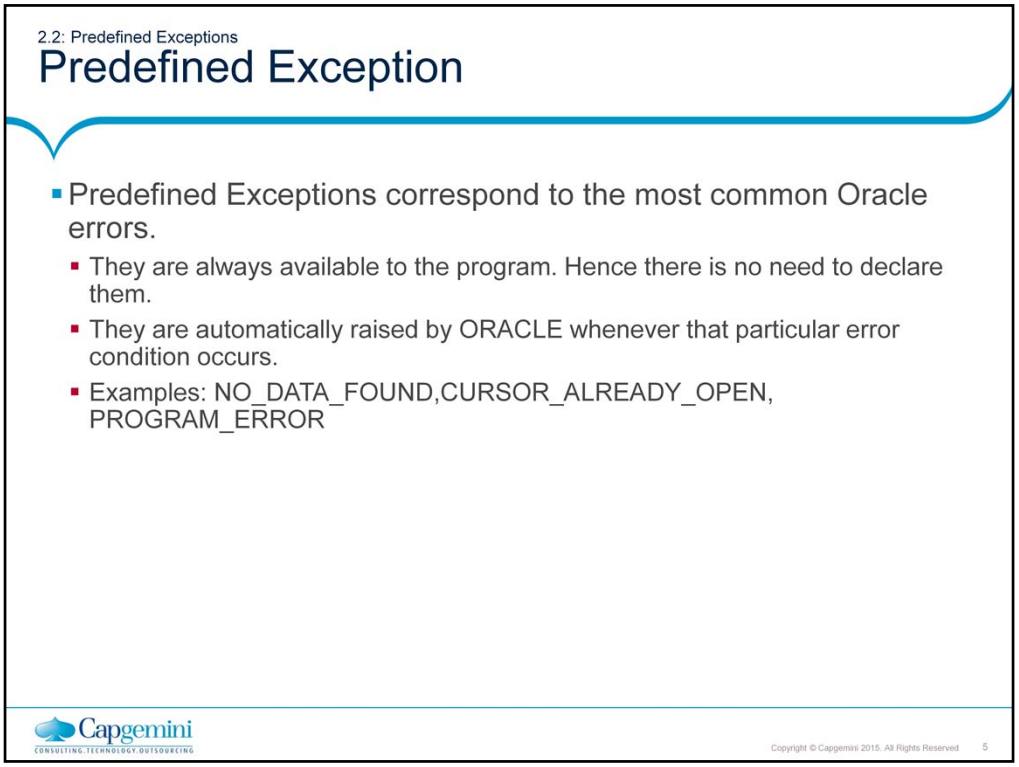
Types of Errors in PL/SQL

- **Compile Time errors:** They are reported by the PL/SQL compiler, and you have to correct them before recompiling.
- **Run Time errors:** They are reported by the run-time engine. They are handled programmatically by raising an exception, and catching it in the Exception section.



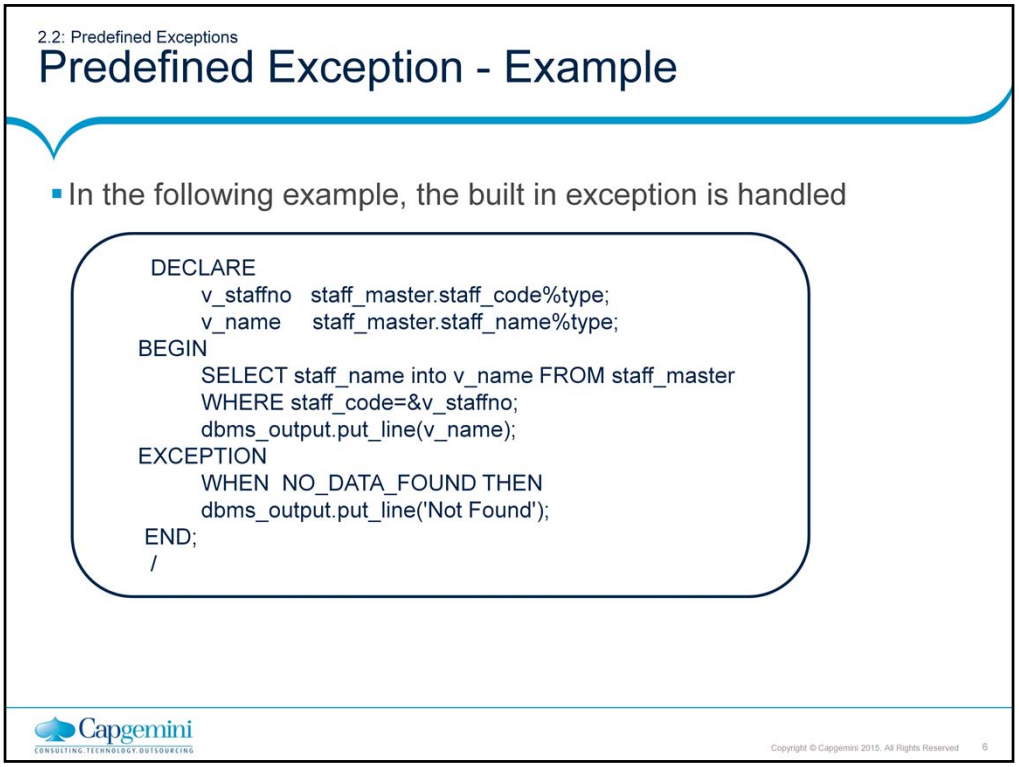
Declaring Exceptions:

- Exceptions are declared in the Declaration section, raised in the Executable section, and handled in the Exception section.

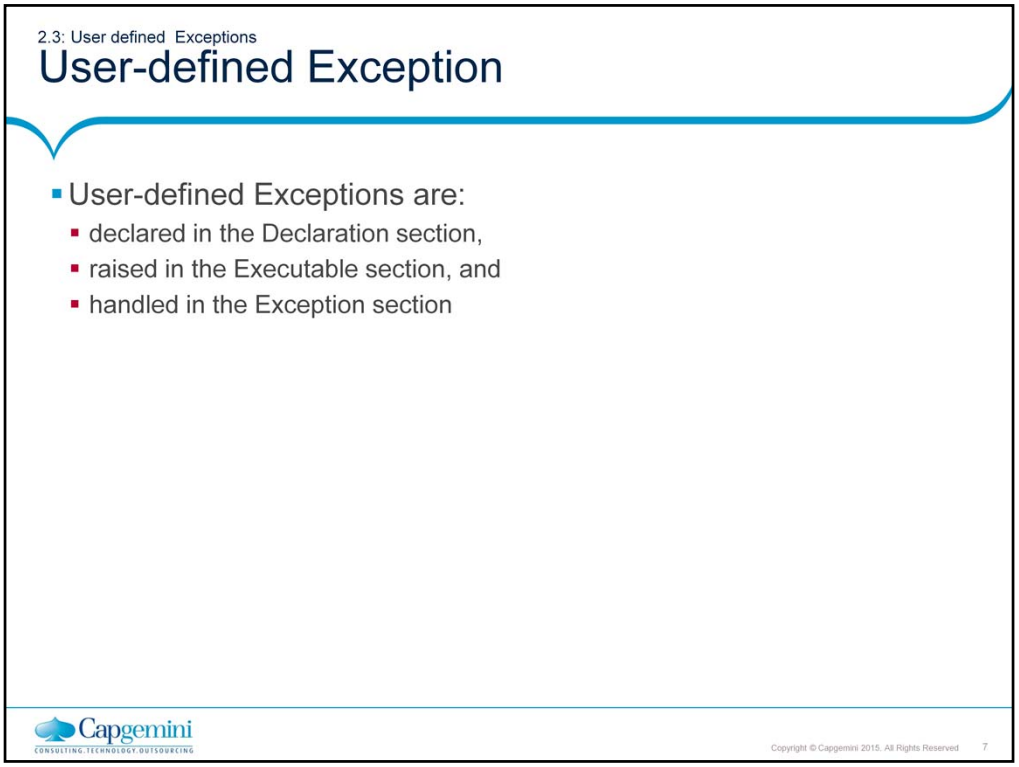


Predefined Exceptions:

- An internal exception is raised implicitly whenever your PL/SQL program violates an Oracle rule or exceeds a system-dependent limit. Every Oracle error has a number, but exceptions must be handled by name. So, PL/SQL predefines some common Oracle errors as exceptions. For example, PL/SQL raises the predefined exception NO_DATA_FOUND if a SELECT INTO statement returns no rows.
- Given below are some Predefined Exceptions:
  - NO_DATA_FOUND
    - This exception is raised when SELECT INTO …. statement does not return any rows.
  - TOO_MANY_ROWS
    - This exception is raised when SELECT INTO …. statement returns more than one row.
  - INVALID_CURSOR
    - This exception is raised when an illegal cursor operation is performed such as closing an already closed cursor.
  - VALUE_ERROR
    - This exception is raised when an arithmetic, conversion, truncation, or constraint error occurs in a procedural statement.
  - DUP_VAL_ON_INDEX
    - This exception is raised when the UNIQUE CONSTRAINT is violated.



Predefined Exceptions:
In the example shown on the slide, the NO_DATA_FOUND built in exception is handled. It is automatically raised if the SELECT statement does not fetch any value and populate the variable.



User-Defined Exceptions:

- These exception are entirely user defined based on the application. The programmer is responsible for declaring, raising and handling them.

2.3: User defined Exceptions

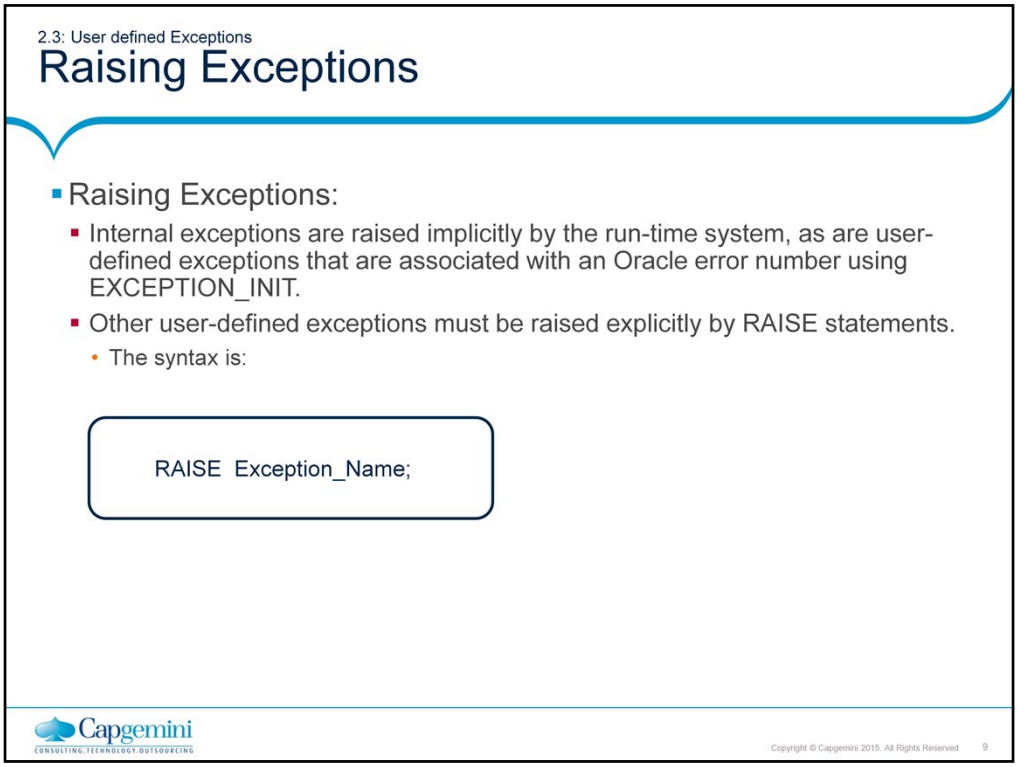
User-defined Exception - Example

- Here is an example of User Defined Exception:

```
DECLARE
      E_Balance_Not_Sufficient EXCEPTION;
      E_Comm_Too_Large EXCEPTION;
      …
BEGIN
      NULL;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8



Raising Exceptions:
When the error associated with an exception occurs, the exception is raised. This is done through the RAISE command.

2.3: User defined Exceptions

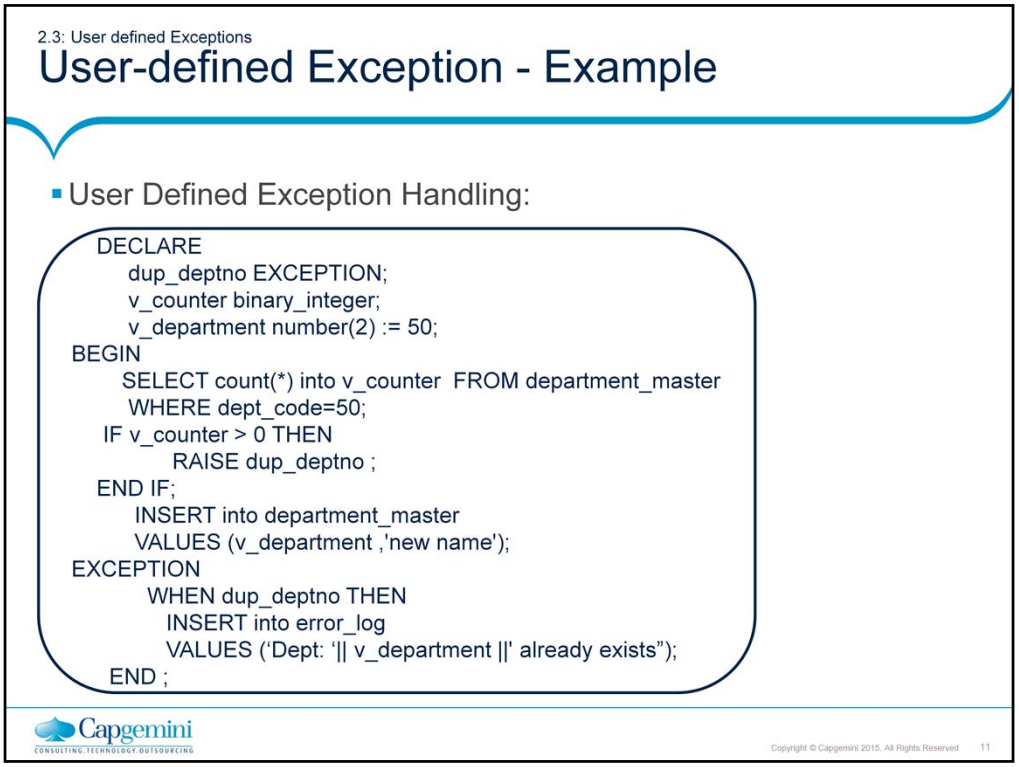
Raising Exceptions - Example

- An exception is defined and raised as shown below:

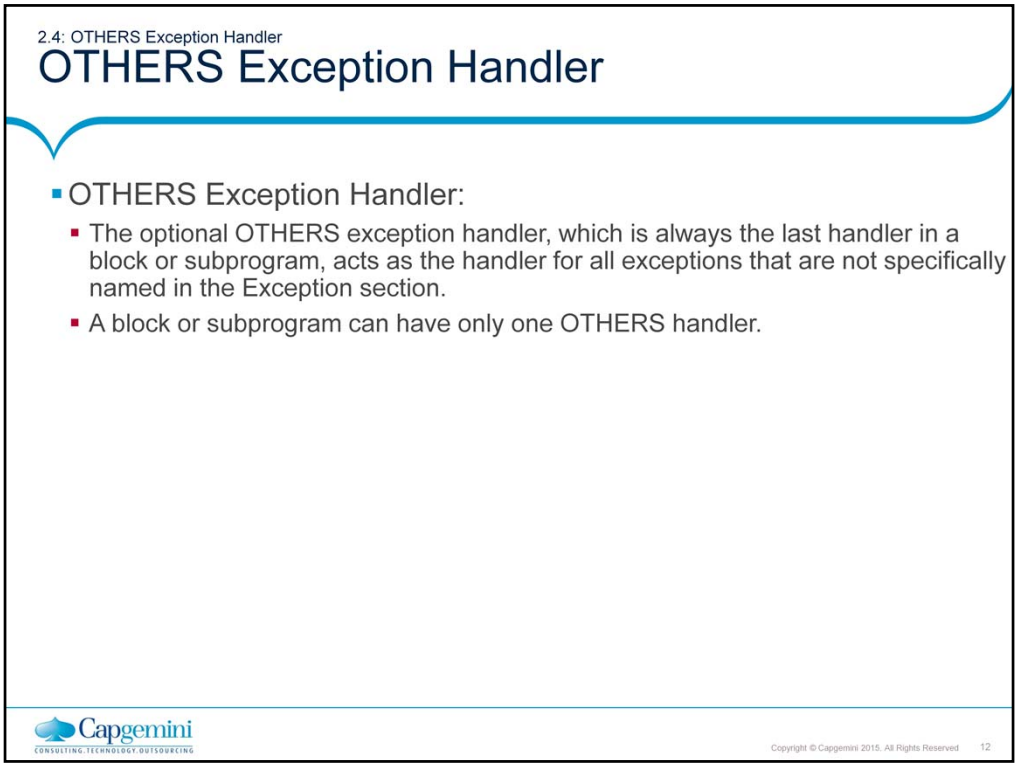
```
DECLARE
      ...
      retired_emp EXCEPTION ;
BEGIN
      pl/sql_statements ;
      if error_condition then
                  RAISE retired_emp ;
                              pl/sql_statements ;
EXCEPTION
      WHEN retired_emp THEN
      pl/sql_statements ;
END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10



The example on the slide demonstrates user-defined exceptions. It checks for department no value to be inserted in the table. If the value is duplicated it will raise an exception.


2.4: OTHERS Exception Handler
OTHERS Exception Handler
OTHERS Exception Handler:
The optional OTHERS exception handler, which is always the last handler in a block or subprogram, acts as the handler for all exceptions that are not specifically named in the Exception section.
A block or subprogram can have only one OTHERS handler.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
12

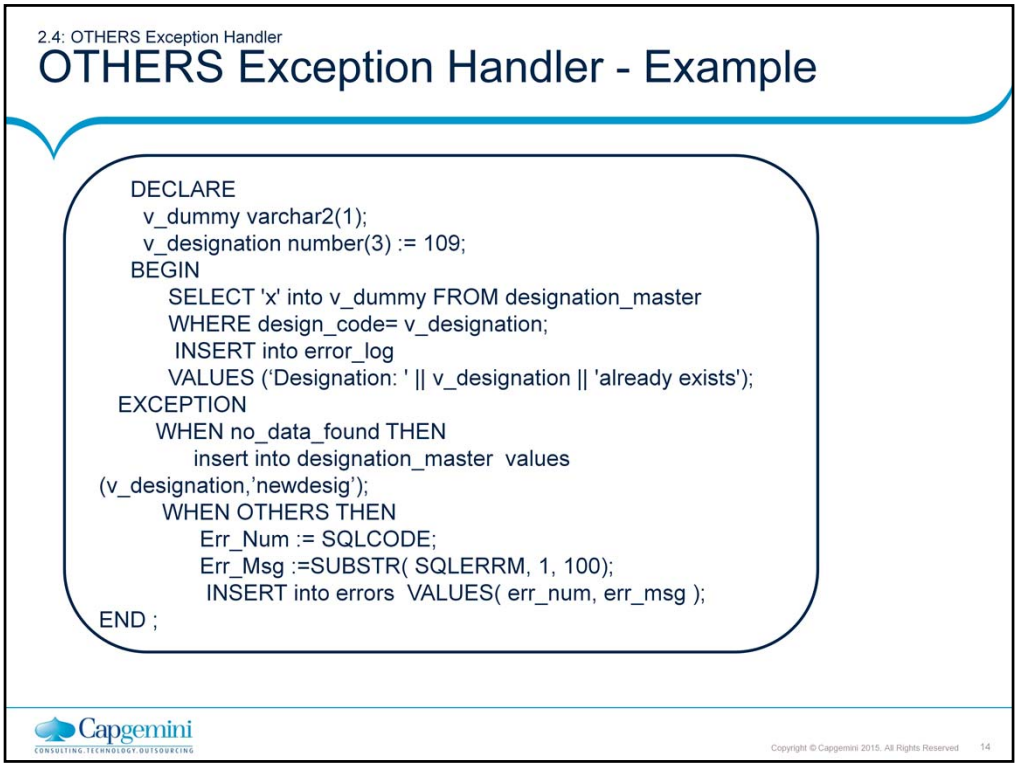
2.4: OTHERS Exception Handler

OTHERS Exception Handler (contd..)

- To handle a specific case within the OTHERS handler, predefined functions SQLCODE and SQLERRM are used.
  - SQLCODE returns the current error code. And SQLERRM returns the current error message text.
  - The values of SQLCODE and SQLERRM should be assigned to local variables before using it within a SQL statement.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

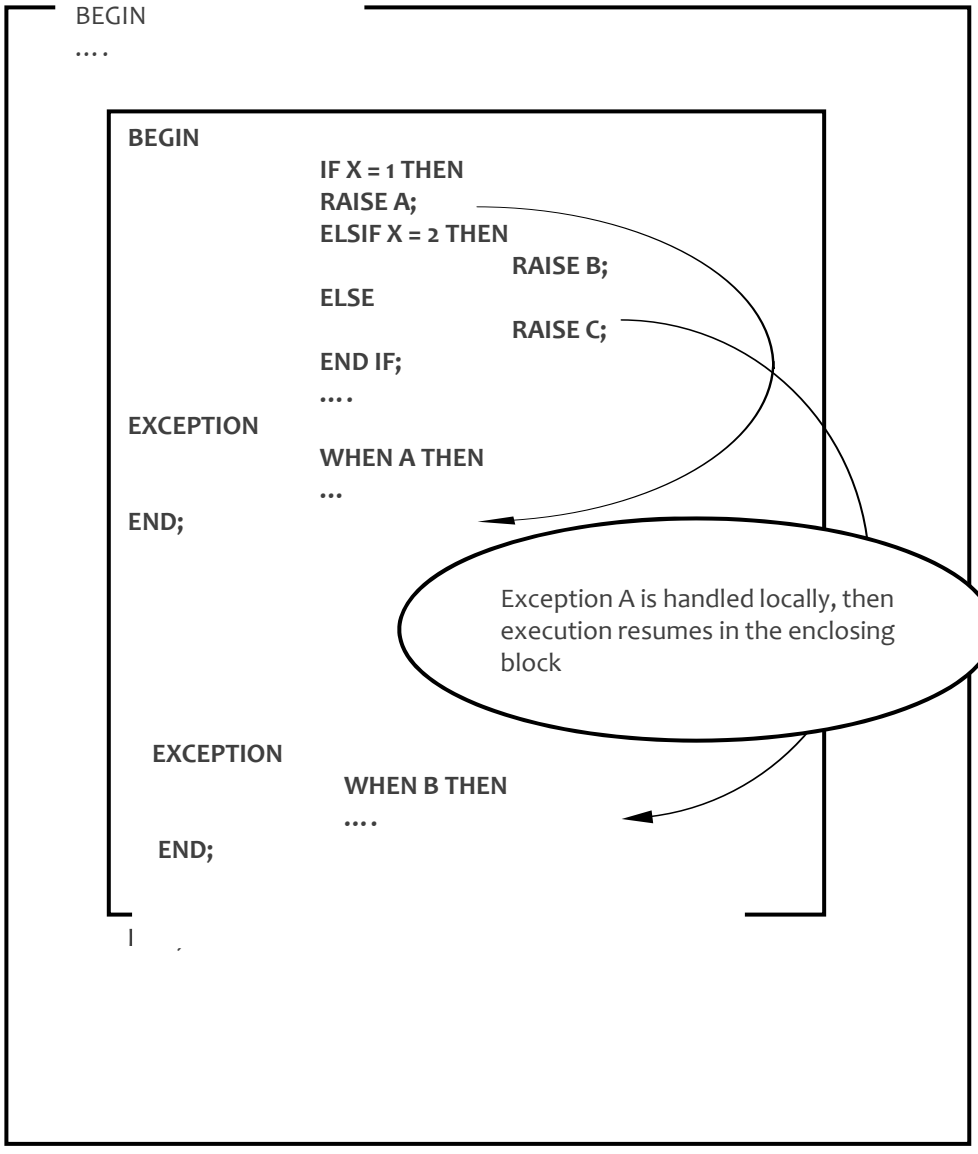


The example on the slide uses OTHERS Exception handler. If the exception that is raised by the code is not NO_DATA_FOUND, then it will go to the OTHERS exception handler since it will notice that there is no appropriate exception handler defined.

Also observe that the values of SQLCODE and SQLERRM are assigned to variables defined in the block.

Propagation of Exceptions:

- When an exception is raised, if PL/SQL cannot find a handler for it in the current block or subprogram, then the exception propagates. That is, the exception reproduces itself in successive enclosing blocks until a handler is found or there are no more blocks to search.



Masking Location of an Error:

- Since the same Exception section is examined for the entire block, it can be difficult to determine, which SQL statement caused the error.

```
SELECT
SELECT
SELECT
EXCEPTION
WHEN  NO_DATA_FOUND THEN
--You Don't Know  which caused the NO_DATA_FOUND
END ;
```

```
DECLARE
V_Counter  NUMBER:= 1;
BEGIN
SELECT …….
V_Counter := 2;
SELECT  …..
V_Counter :=3;
SELECT  …
WHEN NO_DATA_FOUND THEN
 -- Check values of V_Counter to  find out which SELECT
statement
 -- caused the exception NO_DATA_FOUND
END ;
```

```
BEGIN
-- PL/SQL Statements
BEGIN
SELECT ….
EXCEPTION
WHEN NO_DATA_FOUND THEN
        --
END;
BEGIN
SELECT ….
EXCEPTION
WHEN NO_DATA_FOUND THEN
        --
END;
BEGIN
SELECT ….
EXCEPTION
WHEN NO_DATA_FOUND THEN
        --
END;
END ;
```

Masking Location of an Error (contd.):

```
BEGIN
--------
./* PL/SQL statements */
BEGIN
SELECT ……..
WHEN  NO_DATA_FOUND  THEN
-- Process  the error for NO_DATA_FOUND
END;

/* Some more PL/SQL statements
This will execute irrespective of when
NO_DATA_FOUND */
 END;
```

Summary

- In this lesson, you have learnt about:
  - Exception Handling
    - Predefined Exceptions
    - User-defined Exceptions
  - OTHERS exception handler

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

Review – Questions

- Question 1: ___ returns the current error message text.
- Question 2: ___ returns the current error code.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

Oracle Basics (PL/SQL)

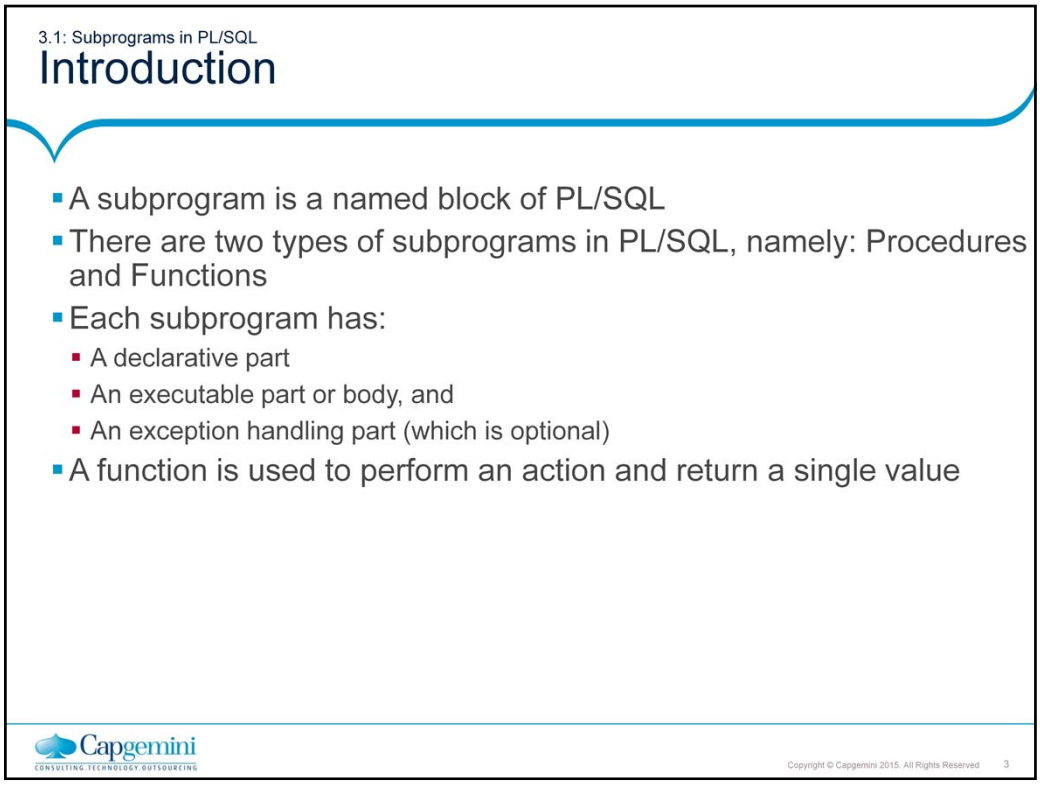
Lesson 03 Procedures and Functions

Lesson Objectives

- To understand the following topics:
  - Subprograms in PL/SQL
  - Anonymous blocks versus Stored Subprograms
  - Procedure
    - Subprogram Parameter modes
  - Functions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Subprograms in PL/SQL:

- The subprograms are compiled and stored in the Oracle database as "stored programs", and can be invoked whenever required. As the subprograms are stored in a compiled form, when called they only need to be executed. Hence this arrangement saves time needed for compilation.
- When a client executes a procedure or function, the processing is done in the server. This reduces the network traffic.
- Subprograms provide the following advantages:
  - They allow you to write a PL/SQL program that meets our need.
  - They allow you to break the program into manageable modules.
  - They provide reusability and maintainability for the code.

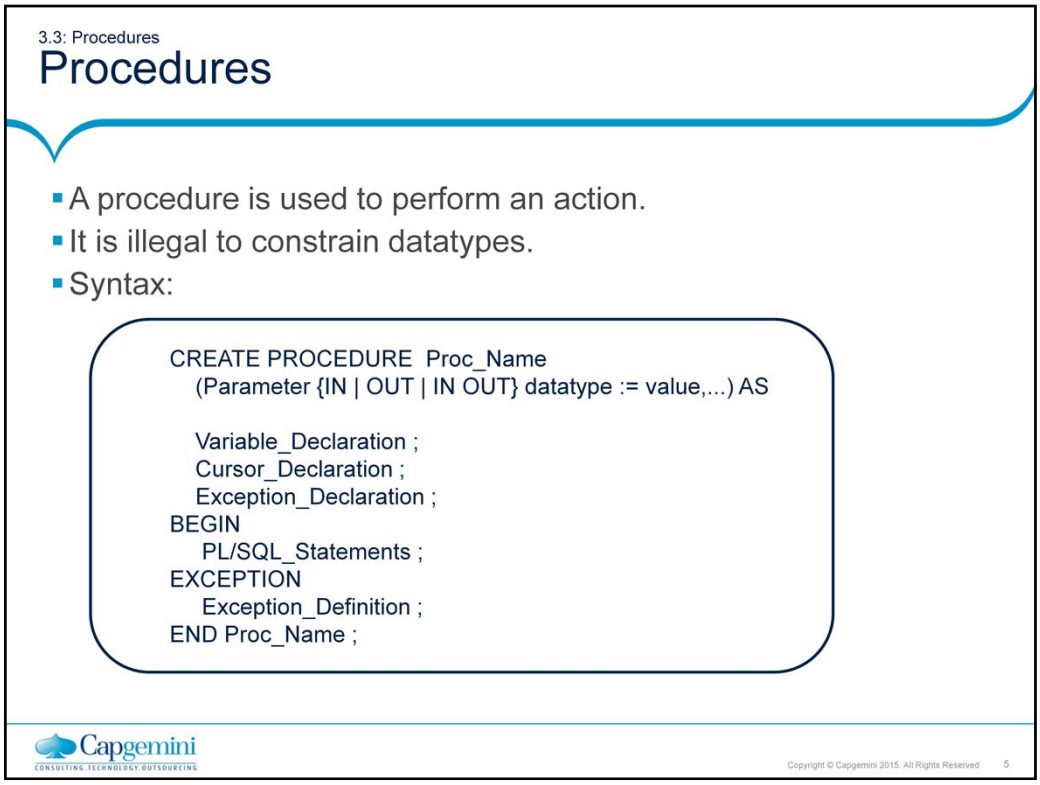
3.2: Anonymous Blocks & Stored Subprograms

Anonymous Blocks & Stored Subprograms Comparison

Anonymous Blocks	Stored Subprograms/Named Blocks
1. Anonymous Blocks do not have names.	1. Stored subprograms are named PL/SQL blocks.
2. They are interactively executed. The block needs to be compiled every time it is run.	2. They are compiled at the time of creation and stored in the database itself. Source code is also stored in the database.
3. Only the user who created the block can use the block.	3. Necessary privileges are required to execute the block.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

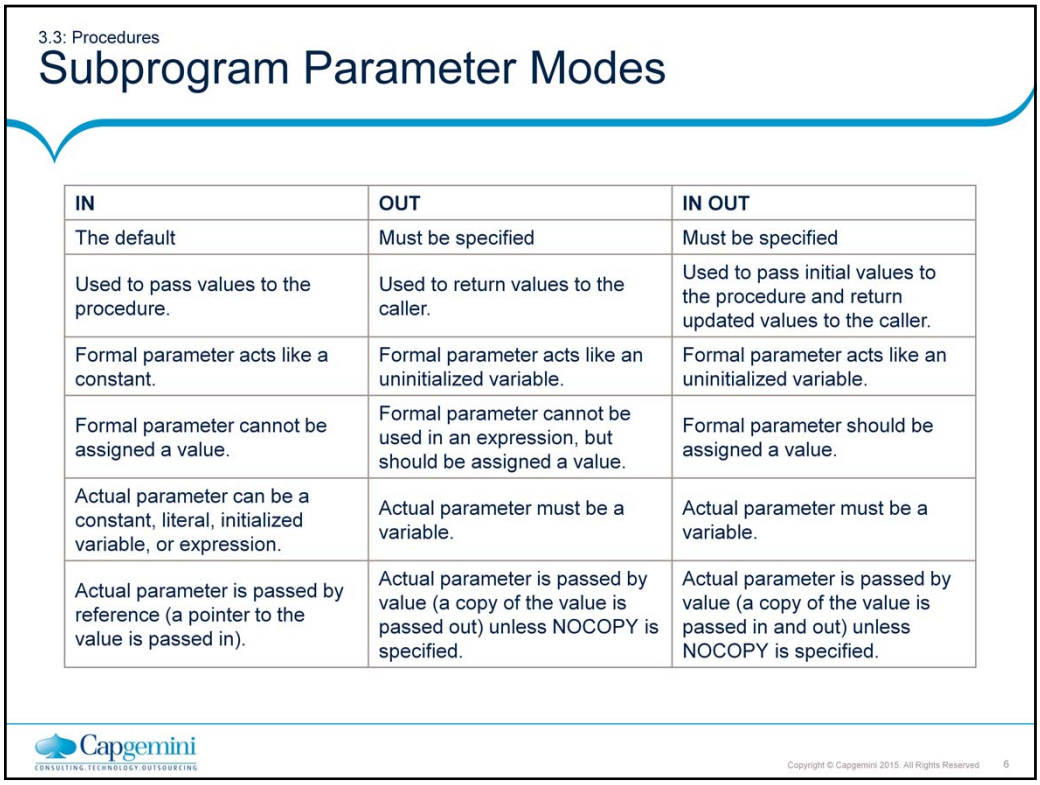
 4



Procedures:

- A procedure is a subprogram used to perform a specific action.
- A procedure contains two parts:
  - the specification, and
  - the body
- The procedure specification begins with CREATE and ends with procedure name or parameters list. Procedures that do not take parameters are written without a parenthesis.
- The procedure body starts after the keyword IS or AS and ends with keyword END.

contd.



Subprogram Parameter Modes:

- You use “parameter modes” to define the behavior of “formal parameters”. The three parameter modes are IN (the default), OUT, and INOUT. The characteristics of the three modes are shown in the slide.
- Any parameter mode can be used with any subprogram.
- Avoid using the OUT and INOUT modes with functions.
- To have a function return multiple values is a poor programming practice. Besides functions should be free from side effects, which change the values of variables that are not local to the subprogram.
- Example1:

```
CREATE PROCEDURE split_name
(
  phrase IN VARCHAR2, first OUT VARCHAR2, last
OUT VARCHAR2
)
IS
  first := SUBSTR(phrase, 1, INSTR(phrase, ' ')-1);
  last := SUBSTR(phrase, INSTR(phrase, ' ')+1);
  IF first = 'John' THEN
    DBMS_OUTPUT.PUT_LINE('That is a common first
name.');
  END IF;
END;
```

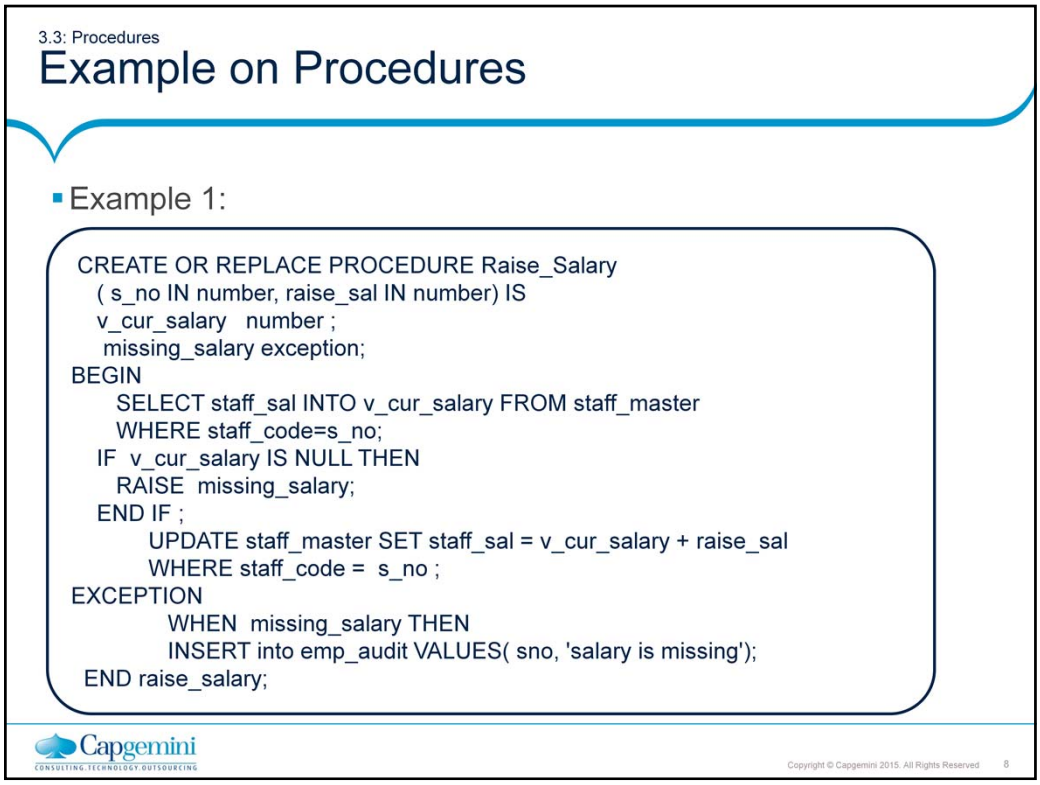
Subprogram Parameter Modes (contd.):
Examples:

Example 2:

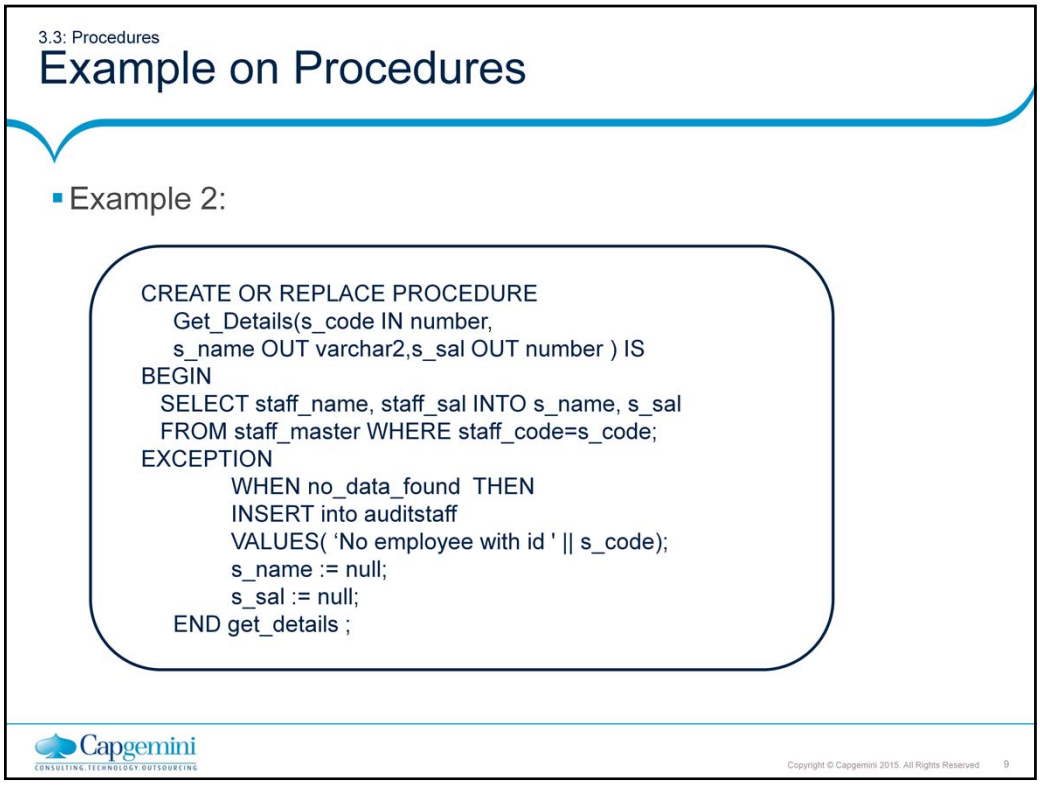
```
SQL > SET SERVEROUTPUT ON
SQL > CREATE OR REPLACE PROCEDURE PROC1 AS
    2   BEGIN
    3       DBMS_OUTPUT.PUT_LINE('Hello from procedure …');
    4   END;
    5   /
Procedure created.
SQL > EXECUTE PROC1
Hello from procedure …
PL/SQL procedure successfully created.
```

Example 3:

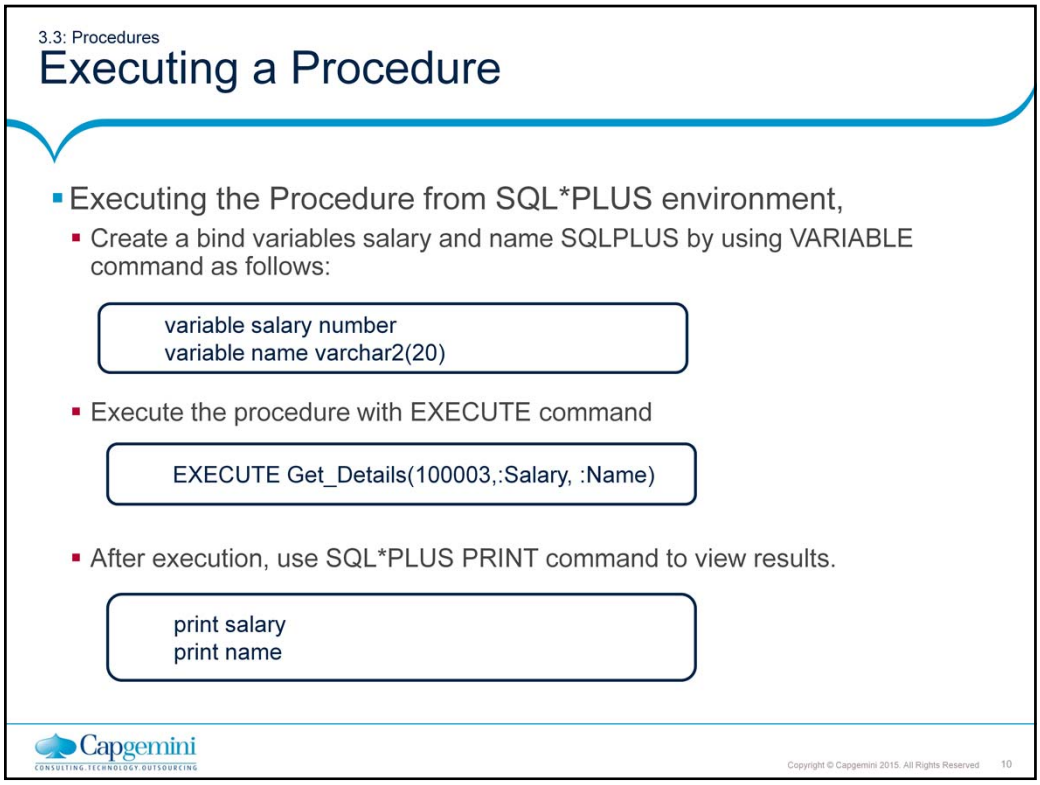
```
SQL > CREATE OR REPLACE       PROCEDURE PROC2
    2       (N1 IN NUMBER, N2 IN NUMBER, TOT OUT NUMBER) IS
    3   BEGIN
    4       TOT := N1 + N2;
    5   END;
    6   /

Procedure created.

SQL > VARIABLE T NUMBER
SQL > EXEC PROC2(33, 66, :T)

PL/SQL procedure successfully completed.

SQL > PRINT T

             T
-- - - - - - - - - -
            99
```



The procedure example on the slide modifies the salary of staff member. It also handles exceptions appropriately. In addition to the above shown exception you can also handle “NO_DATA_FOUND” exception. The procedure accepts two parameters which is the staff_code and amount that has to be given as raise to the staff member.



The procedure on the slide accept three parameters, one is IN mode and other two are OUT mode. The procedure retrieves the name and salary of the staff member based on the staff_code passed to the procedure. The S_NAME and S_SAL are the OUT parameters that will return the values to the calling environment

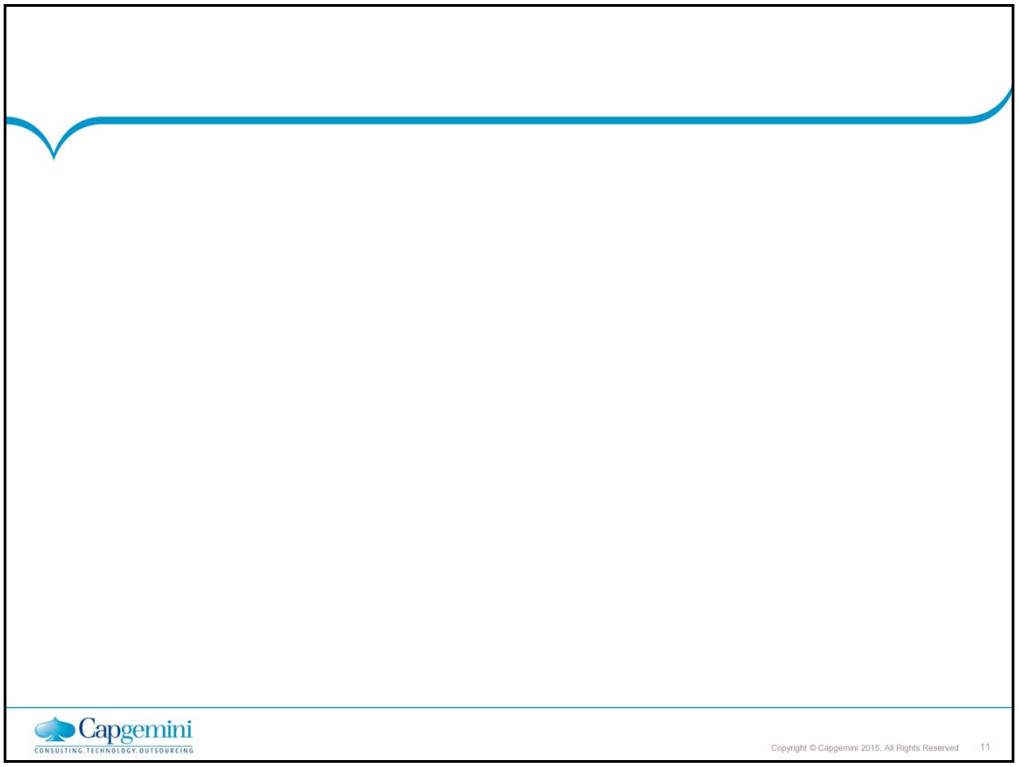


Procedures can be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.

On the slide the first snippet declares two variables viz. salary and name. The second snippet calls the procedure and passes the actual parameters. The first is a literal string and the next two parameters are empty variables which will be assigned with values within the procedure.

Calling the procedure from an anonymous PL/SQL block

```
DECLARE
      s_no  number(10):=&sno;
      sname varchar2(10);
      sal  number(10,2);
BEGIN
      Get_Details(s_no,sname,sal);
      dbms_output.put_line('Name:'||sname||'Salary'||sal);
END;
```



A bind variable is a variable that you declare in a host environment and then use to pass runtime values.
These values can be character or numeric. You can pass these values either in or out of one or more PL/SQL programs, such as packages, procedures, or functions.
To declare a bind variable in the SQL*Plus environment, you use the command VARIABLE.
For example,
VARIABLE salary NUMBER

Upon declaration, the **bind variables** now become **host** to that environment, and you can now use these variables within your PL/SQL programs, such as packages, procedures, or functions.
To reference host variables, you must add a prefix to the reference with a colon (:) to distinguish the host variables from declared PL/SQL variables.
example
EXECUTE Get_Details(100003,:salary, :name)

Parameter default values:

- Like variable declarations, the formal parameters to a procedure or function can have default values.
- If a parameter has default values, it does not have to be passed from the calling environment.
  - If it is passed, actual parameter will be used instead of default.
- Only IN parameters can have default values.

```
PROCEDURE Create_Dept( New_Deptno IN NUMBER,
   New_Dname IN  VARCHAR2  DEFAULT 'TEMP') IS
BEGIN
  INSERT INTO  department_master
        VALUES ( New_Deptno, New_Dname, New_Loc) ;
END ;
```

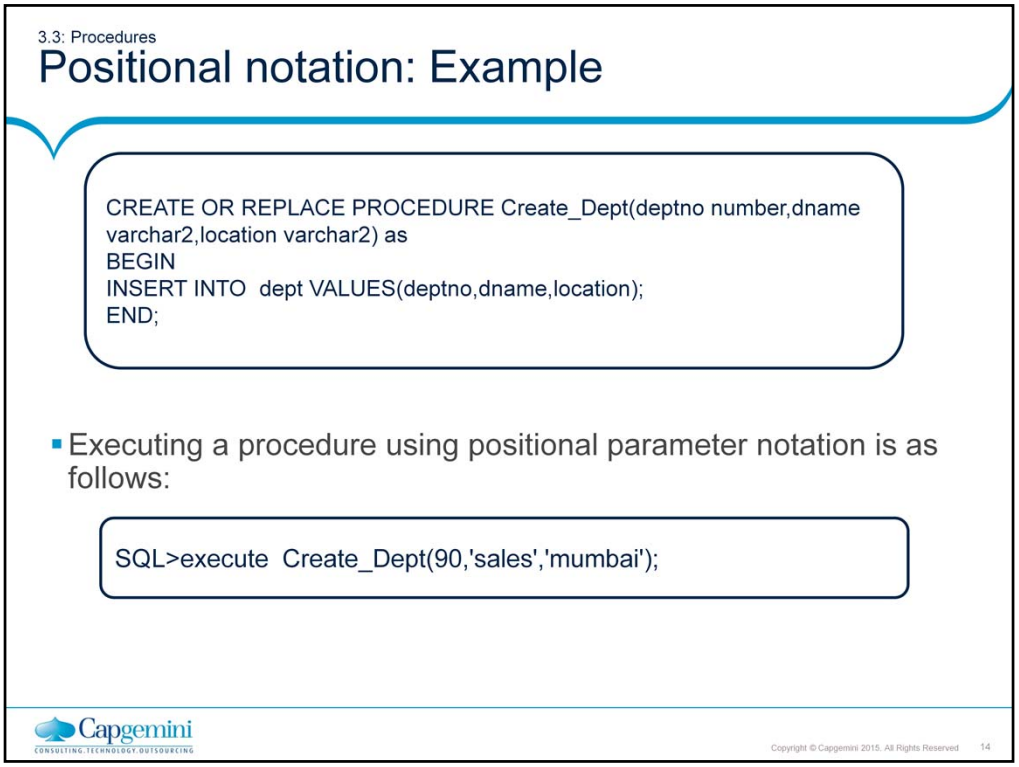
```
BEGIN
Create_Dept( 50);
--  Actual  call will be  Create_Dept ( 50, 'TEMP',
'TEMP')

Create_Dept ( 50, 'FINANCE');
-- Actual call will be  Create_Dept ( 50, 'FINANCE'
,'TEMP')

Create_Dept( 50, 'FINANCE', 'BOMBAY') ;
-- Actual  call will be  Create_Dept(50, 'FINANCE',
'BOMBAY' )

END;
```

Procedures (contd.):

Using Positional, Named, or Mixed Notation for Subprogram Parameters:

- When calling a subprogram, you can write the actual parameters by using either Positional notation, Named notation, or Mixed notation.
  - **Positional notation:** You specify the same parameters in the same order as they are declared in the procedure. This notation is compact, but if you specify the parameters (especially literals) in the wrong order, the bug can be hard to detect. You must change your code if the procedure's parameter list changes.
  - **Named notation:** You specify the name of each parameter along with its value. An arrow (=>) serves as the "association operator". The order of the parameters is not significant.
  - **Mixed notation:** You specify the first parameters with "Positional notation", and then switch to "Named notation" for the last parameters. You can use this notation to call procedures that have some "required parameters", followed by some "optional parameters".
- We have already seen a few examples of calling procedures with Positional notation.

```
Create_Dept (New_Deptno=> 50, New_Dname=>'FINANCE');
```

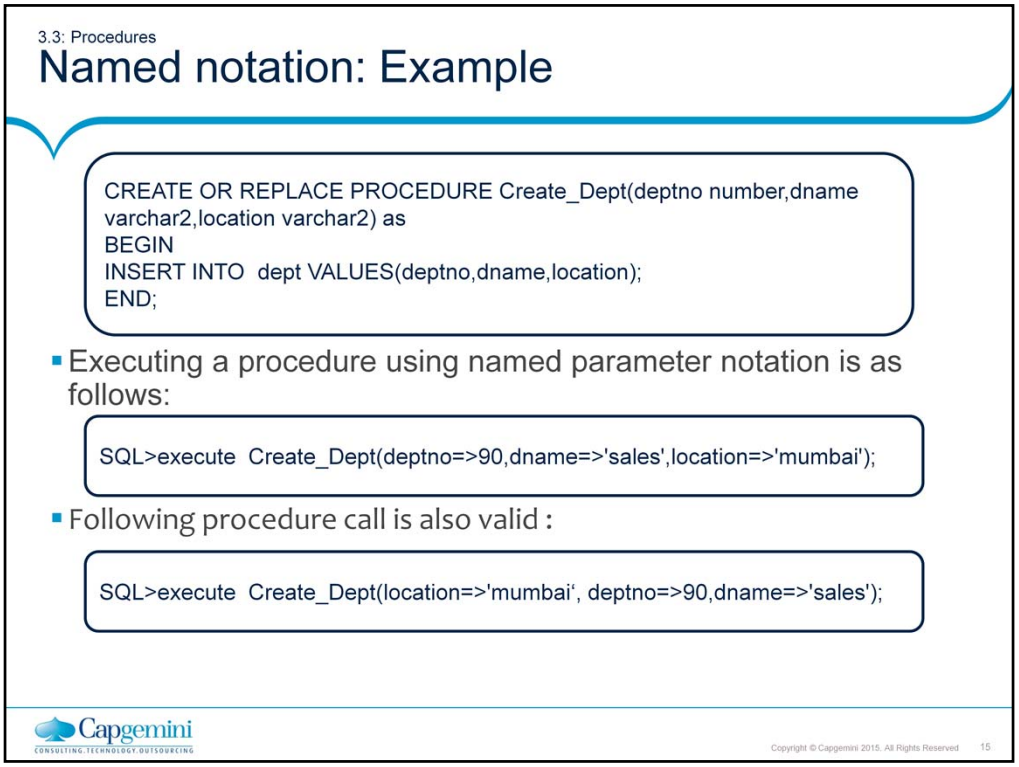


Positional Notation :

You specify the parameters in the same order as they are declared in the procedure.

This notation is compact, but if you specify the parameters (especially literals) in the wrong order, the bug can be hard to detect.

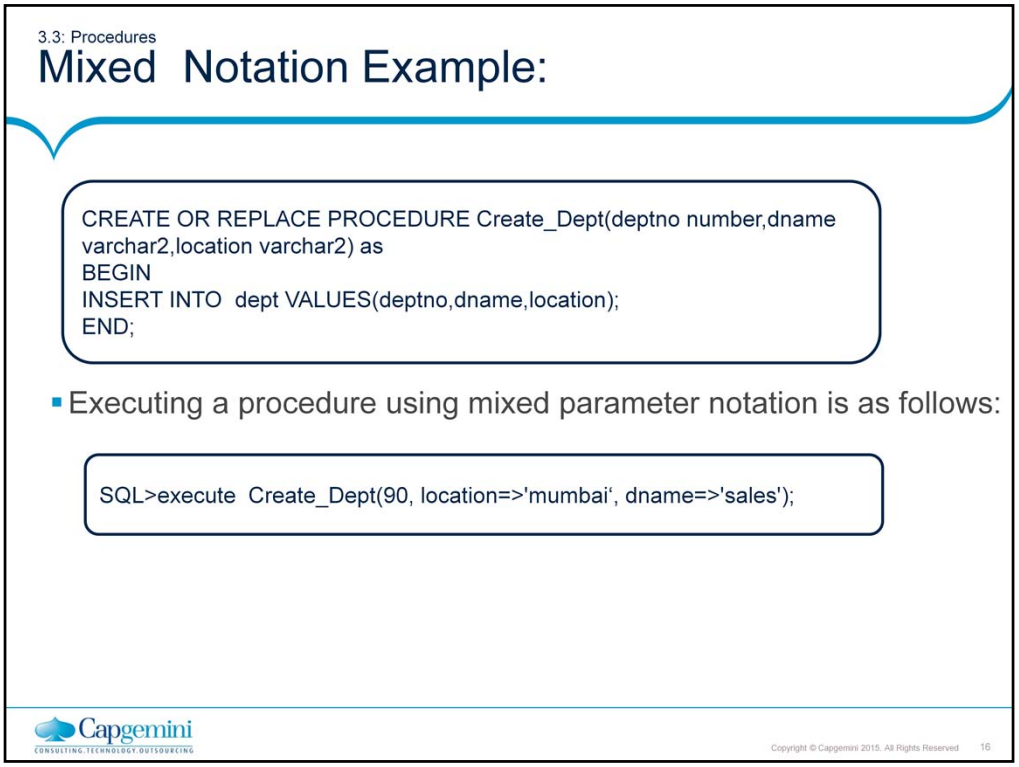
You must change your code if the procedure's parameter list changes



Named notation:

You specify the name of each parameter along with its value. An arrow (=>) serves as the “association operator”. The order of the parameters is not significant.

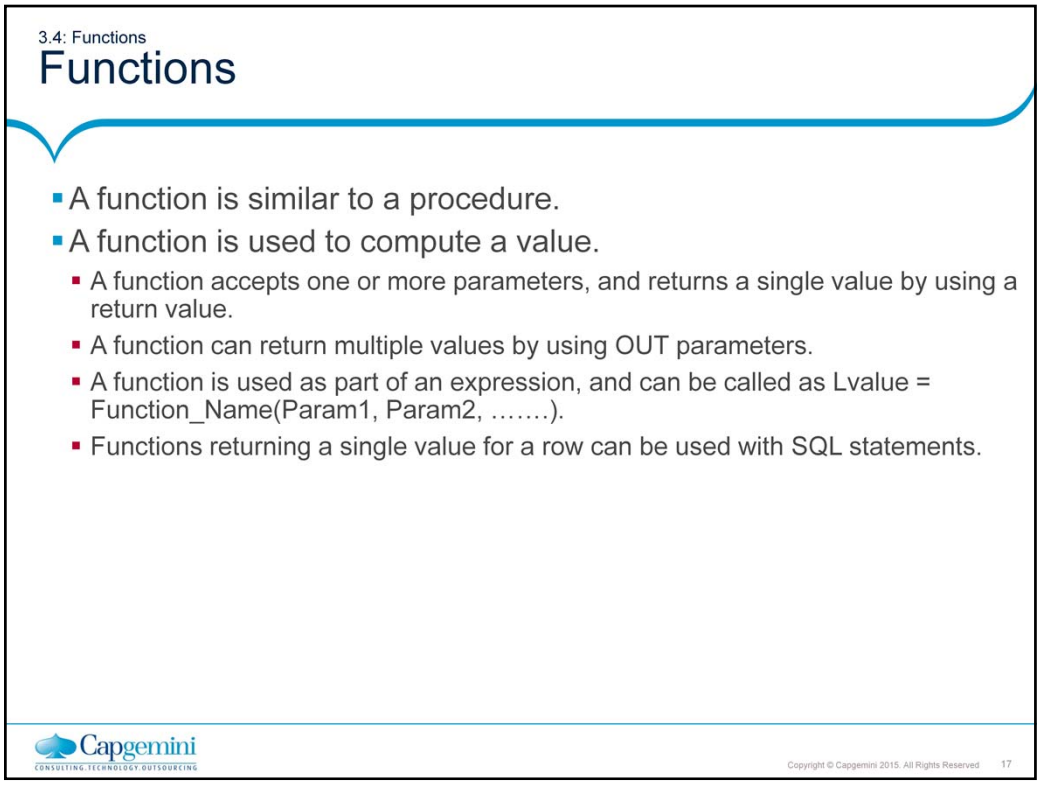
While executing the procedure, the names of the parameters must be the same as those in the procedure declaration.

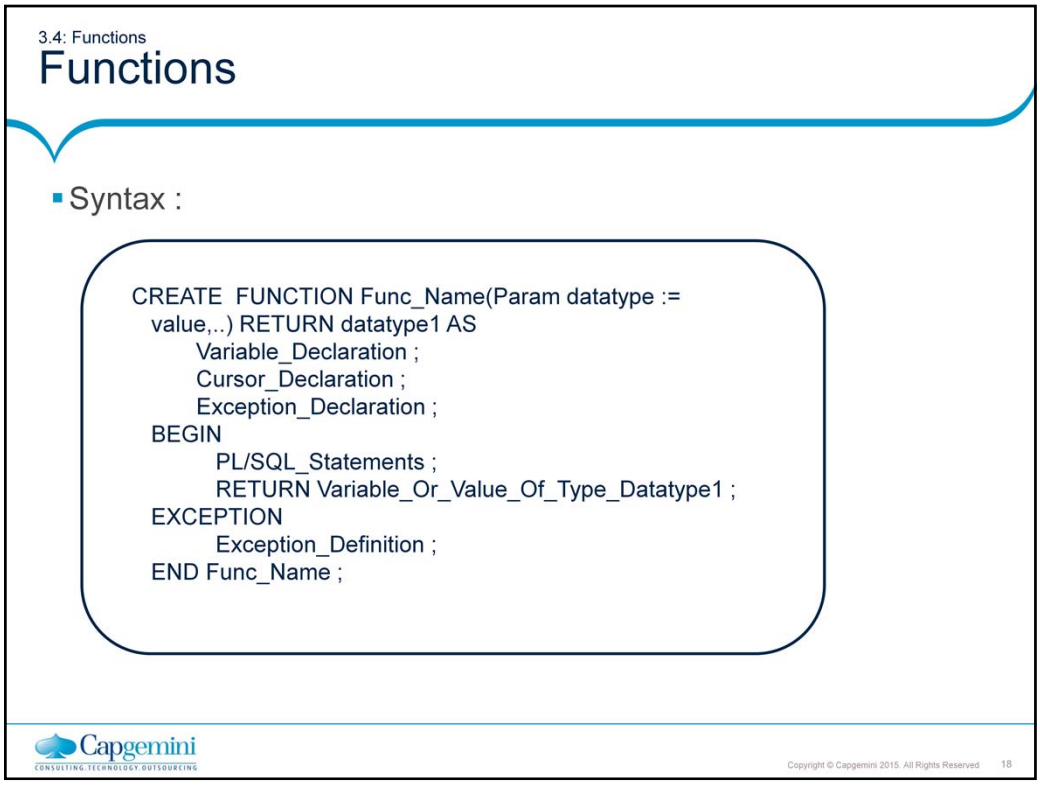


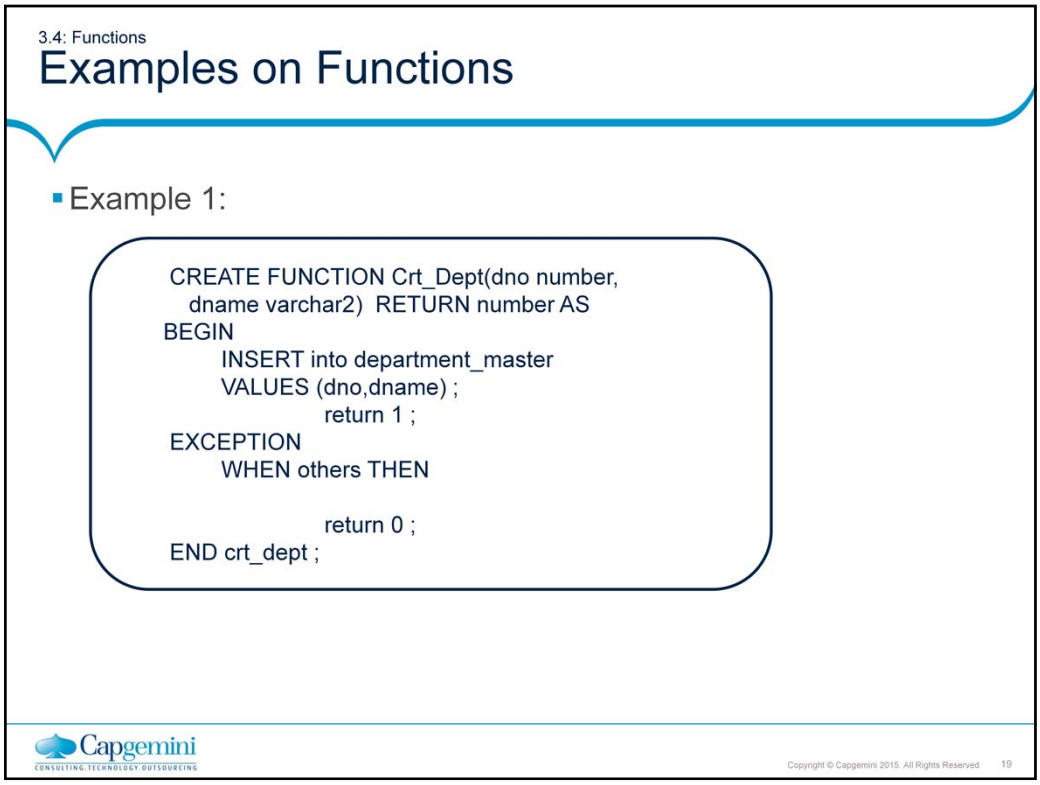
Mixed notation:

You specify the first parameters with “Positional notation”, and then switch to “Named notation” for the last parameters.

You can use this notation to call procedures that have some “required parameters”, followed by some “optional parameters”.


3.4: Functions
Functions
A function is similar to a procedure.
A function is used to compute a value.
A function accepts one or more parameters, and returns a single value by using a return value.
A function can return multiple values by using OUT parameters.
A function is used as part of an expression, and can be called as Lvalue = Function_Name(Param1, Param2, …….).
Functions returning a single value for a row can be used with SQL statements.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
17

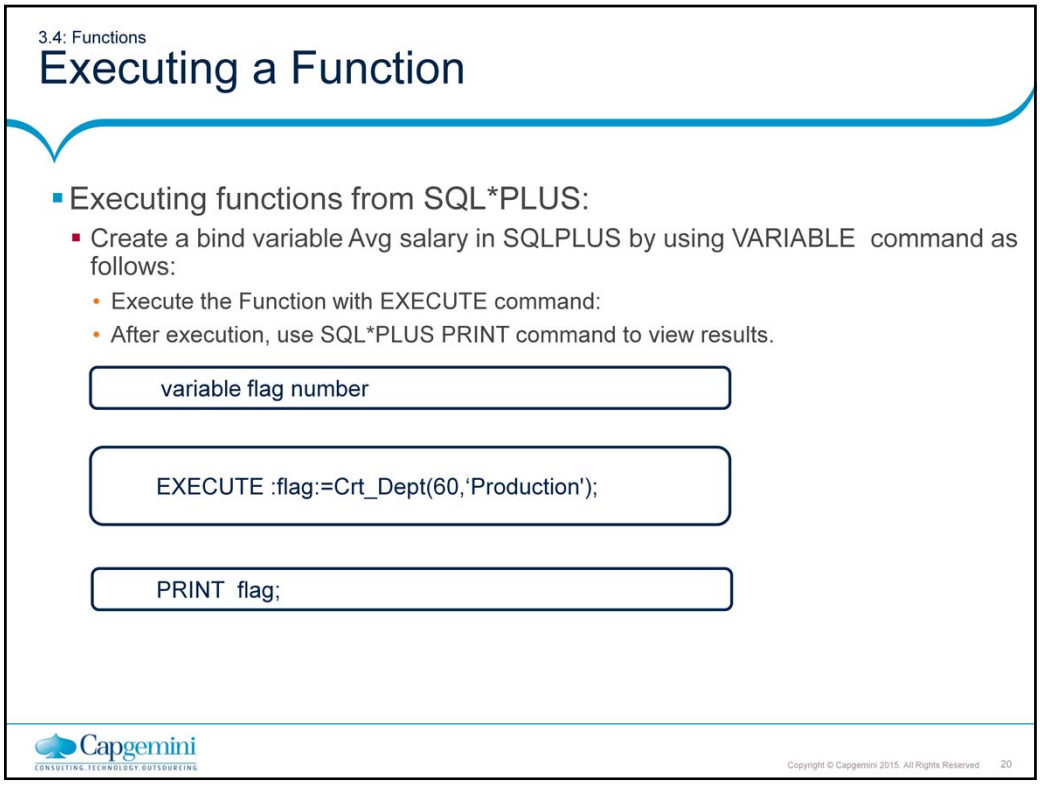

3.4: Functions
Functions
Syntax :
CREATE FUNCTION Func_Name(Param datatype :=
value,..) RETURN datatype1 AS
Variable_Declaration ;
Cursor_Declaration ;
Exception_Declaration ;
BEGIN
PL/SQL_Statements ;
RETURN Variable_Or_Value_Of_Type_Datatype1 ;
EXCEPTION
Exception_Definition ;
END Func_Name ;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
18



Example 2:

- Function to calculate average salary of a department:
  - Function returns average salary of the department
  - Function returns -1, in case no employees are there in the department.
  - Function returns -2, in case of any other error.

```
CREATE OR REPLACE FUNCTION Get_Avg_Sal(p_deptno
in number) RETURN number AS
   V_Sal number;
BEGIN
        SELECT Trunc(Avg(staff_sal)) INTO V_Sal
        FROM staff_master
        WHERE deptno=P_Deptno;
        IF v_sal is null THEN
                v_sal := -1 ;
         END IF;
                return v_sal;
EXCEPTION
       WHEN others THEN
              return -2;  --signifies any other errors
END get_avg_sal;
```



Functions can also be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.
The second snippet calls the function and passes the actual parameters. The variable declared earlier is used for collecting the return value from the function
Calling the function from an anonymous PL/SQL block

```
DECLARE
  avgsalary  number;
BEGIN
     avgsalary:=Get_Avg_Sal(20);
     dbms_output.put_line('The average salary of Dept 20 is'||
     avgsalary);
END;
```

Calling function using a Select statement

```
SELECT  Get_Avg_Sal(30) FROM staff_master;
```

Summary

- In this lesson, you have learnt:
  - Subprograms in PL/SQL are named PL/SQL blocks.
  - There are two types of subprograms, namely: Procedures and Functions
  - Procedure is used to perform an action
    - Procedures have three subprogram parameter modes, namely: IN, OUT, and INOUT
  - Functions are used to compute a value
    - A function accepts one or more parameters, and returns a single value by using a return value
    - A function can return multiple values by using OUT parameters

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21

Review – Questions

- Question 1: Anonymous Blocks do not have names.
  - True / False
- Question 2: A function can return multiple values by using OUT parameters
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

Review – Questions

- Question 3: An ___ parameter returns a value to the caller of a subprogram.
- Question 4: A procedure contains two parts: _____ and _____.
- Question 5: In ___ notation, the order of the parameters is not significant.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23

Oracle for Developers (PL/SQL)

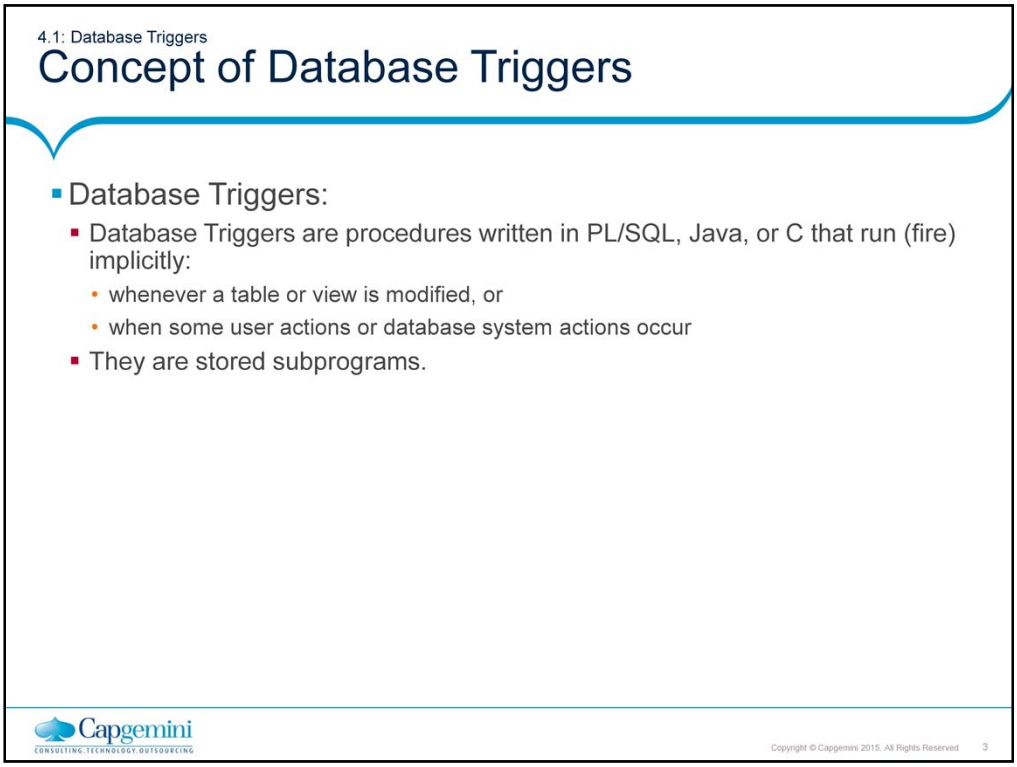
Database Triggers

Lesson Objectives

- To understand the following topics:
  - Concept of Database Triggers
  - Types of Triggers
  - Disabling and Dropping Triggers
  - Restriction on Triggers
  - Order of Trigger firing
  - Using :Old and :New values, WHEN clause, Trigger predicates

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Database Triggers:

A Trigger defines an action the database should take when some database related event occurs.

Anonymous blocks as well as stored subprograms need to be explicitly invoked by the user.

Database Triggers are PL/SQL blocks, which are implicitly fired by ORACLE RDBMS whenever an event such as INSERT, UPDATE, or DELETE takes place on a table.

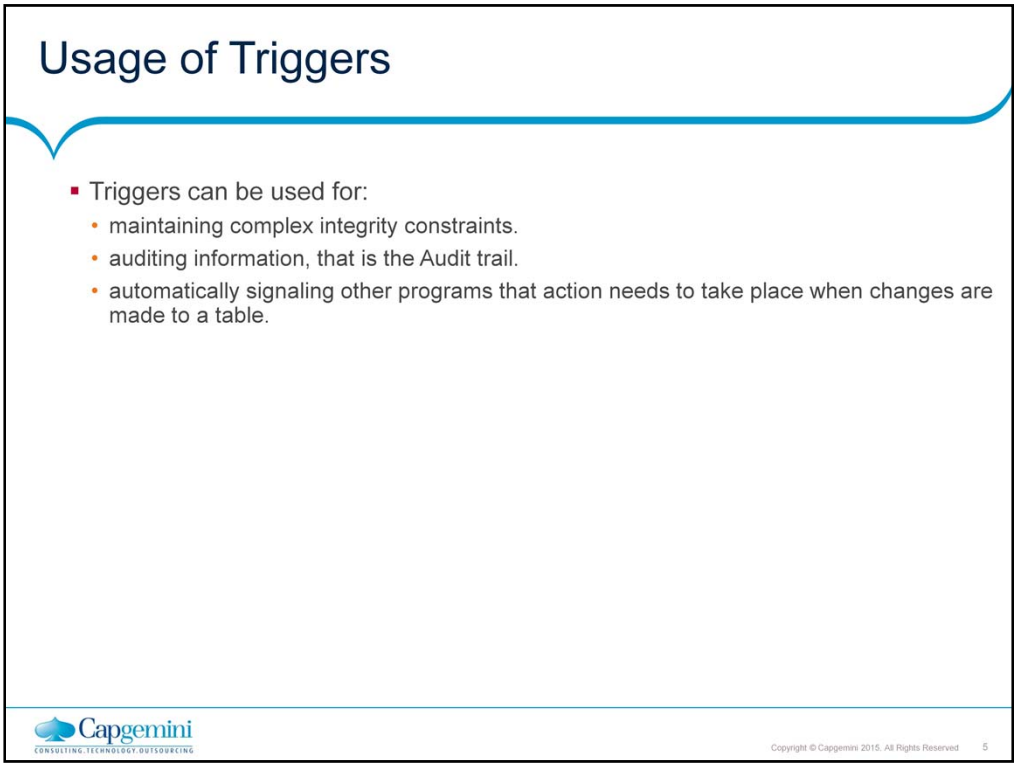
Database triggers are also stored subprograms.

Concept of Database Triggers

- You can write triggers that fire whenever one of the following operations occur:
  - User events:
    - DML statements on a particular schema object
    - DDL statements issued within a schema or database
    - user logon or logoff events
  - System events:
    - server errors
    - database startup
    - instance shutdown

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4

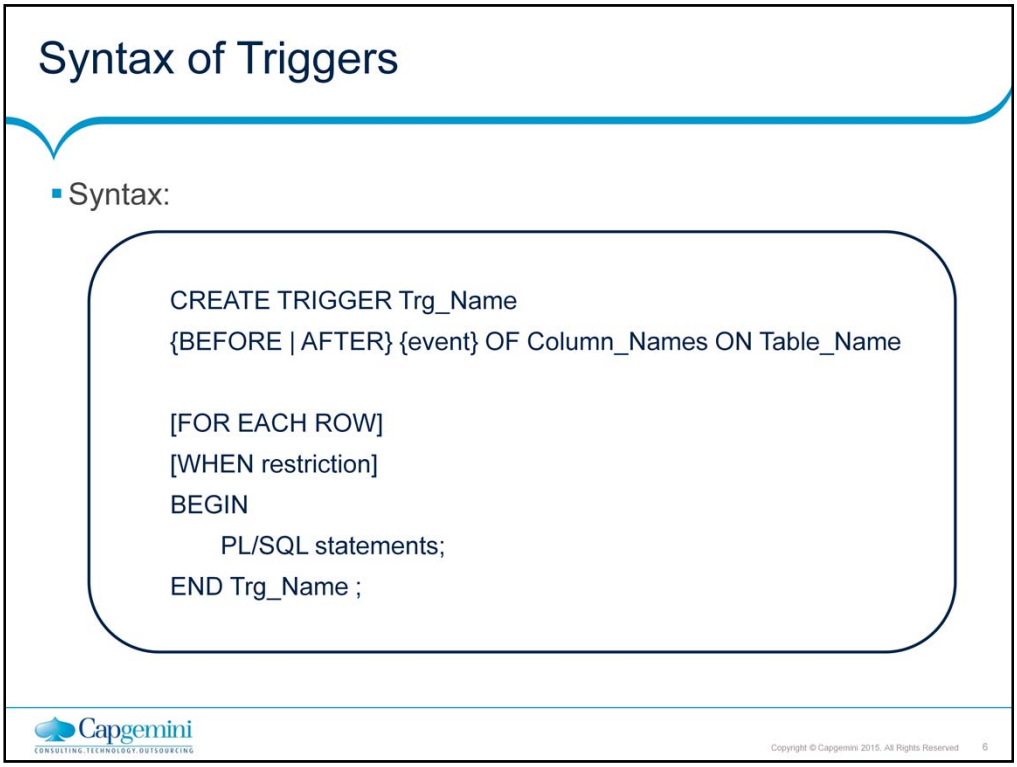


Database Triggers (contd.):

Triggers can be used for:

- Maintaining complex integrity constraints. This is not possible through declarative constraints that are enabled at table creation.
- Auditing information in a table by recording the changes and the identify of the person who made them. This is called as an audit trail.
- Automatically signaling other programs that action needs to take place when changes are made to a table.

contd.



Database Triggers (contd.):
To create a trigger on a table you must be able to alter that table. The slide shows the syntax for creating a table.

contd.

Database Triggers (contd.):

Parts of a Trigger

A trigger has three basic parts:

- A triggering event or statement
- A trigger restriction
- A trigger action

Triggering Event or Statement

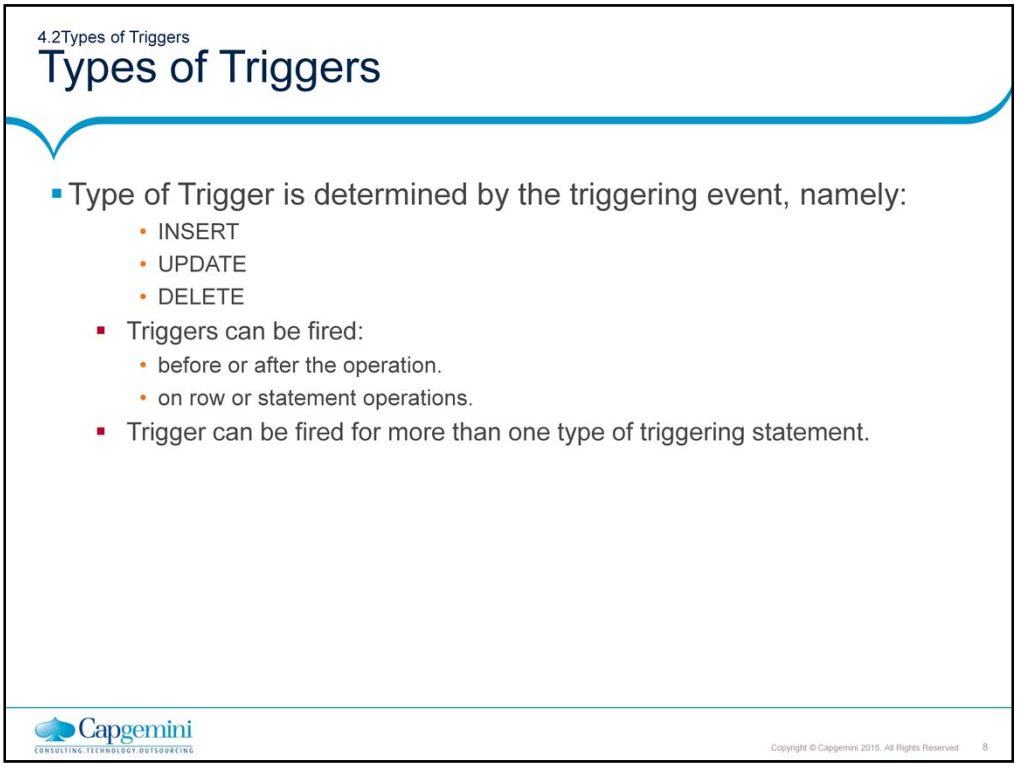
- A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire. A triggering event can be one or more of the following:
    - An INSERT, UPDATE, or DELETE statement on a specific table (or view, in some cases)
    - A CREATE, ALTER, or DROP statement on any schema object
    - A database startup or instance shutdown
    - A specific error message or any error message
    - A user logon or logoff

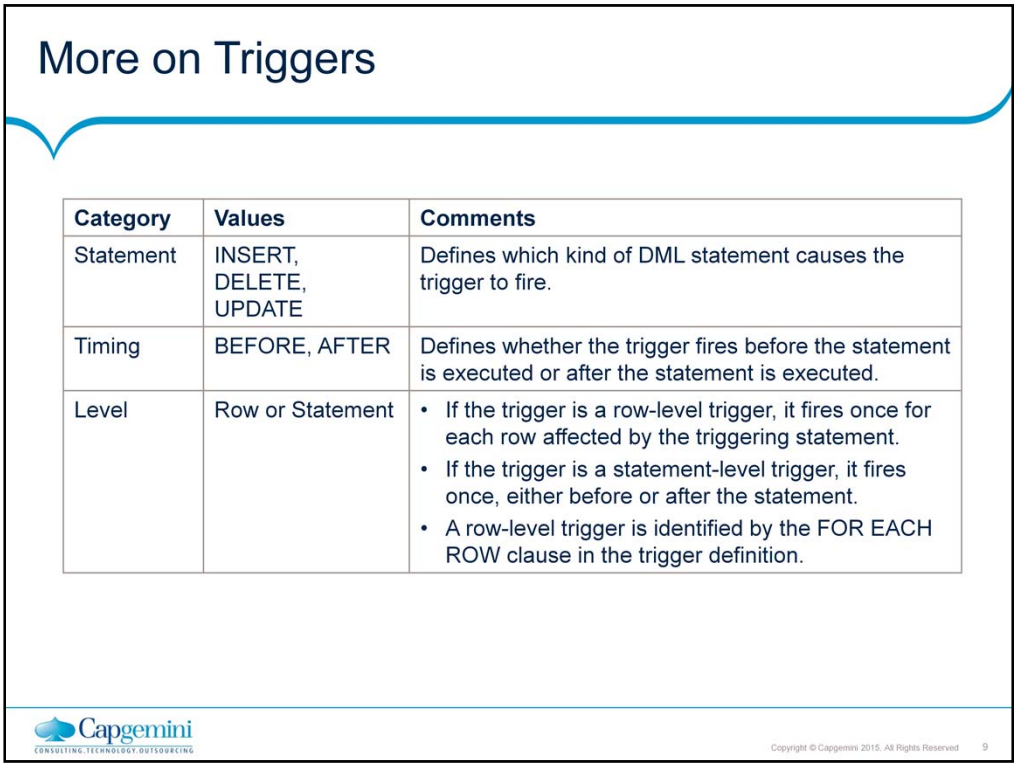
Trigger Restriction

- A trigger restriction specifies a Boolean expression that must be true for the trigger to fire. The trigger action is not run if the trigger restriction evaluates to false or unknown.

Trigger Action

- A trigger action is the procedure (PL/SQL block, Java program, or C callout) that contains the SQL statements and code to be run when the following events occur:
    - a triggering statement is issued
    - the trigger restriction evaluates to true
- Like stored procedures, a trigger action can:
    - contain SQL, PL/SQL, or Java statements
    - define PL/SQL language constructs such as variables, constants, cursors, exceptions
    - define Java language constructs
    - call stored procedures


4.2Types of Triggers
Types of Triggers
Type of Trigger is determined by the triggering event, namely:
INSERT
UPDATE
DELETE
Triggers can be fired:
before or after the operation.
on row or statement operations.
Trigger can be fired for more than one type of triggering statement.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
8



Category	Values	Comments
Statement	INSERT, DELETE, UPDATE	Defines which kind of DML statement causes the trigger to fire.
Timing	BEFORE, AFTER	Defines whether the trigger fires before the statement is executed or after the statement is executed.
Level	Row or Statement	• If the trigger is a row-level trigger, it fires once for each row affected by the triggering statement. • If the trigger is a statement-level trigger, it fires once, either before or after the statement. • A row-level trigger is identified by the FOR EACH ROW clause in the trigger definition.

Note:

- A trigger can be fired for more than one type of triggering statement. In case of multiple events, OR separates the events.
- From ORACLE 7.1 there is no limit on number of triggers you can write for a table.
    - Earlier you could write maximum of 12 triggers per table, one of each type.

Types of Triggers (contd.)

Row Triggers and Statement Triggers

- When you define a trigger, you can specify the number of times the trigger action has to be run:
  - Once for every row affected by the triggering statement, such as a trigger fired by an UPDATE statement that updates many rows.
  - Once for the triggering statement, no matter how many rows it affects.

Row Triggers

- A Row Trigger is fired each time the table is affected by the triggering statement. For example: If an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement.
- If a triggering statement affects no rows, a row trigger is not run.
- Row triggers are useful if the code in the trigger action depends on data provided by the triggering.

INSTEAD OF Triggers

- INSTEAD OF triggers provide a transparent way of modifying views that cannot be directly modified through DML statements (INSERT, UPDATE, and DELETE). These triggers are called INSTEAD OF triggers because, unlike other types of triggers, Oracle fires the trigger instead of executing the triggering statement.
- You can write normal INSERT, UPDATE, and DELETE statements against the view and the INSTEAD OF trigger is fired to update the underlying tables appropriately. INSTEAD OF triggers are activated for each row of the view that gets modified.

Modify Views

- Modifying views can have ambiguous results:
  - Deleting a row in a view could either mean deleting it from the base table or updating some values so that it is no longer selected by the view.
  - Inserting a row in a view could either mean inserting a new row into the base table or updating an existing row so that it is projected by the view.
  - Updating a column in a view that involves joins might change the semantics of other columns that are not projected by the view.
- Object views present additional problems. For example, a key use of object views is to represent master/detail relationships. This operation inevitably involves joins, but modifying joins is inherently ambiguous.
- As a result of these ambiguities, there are many restrictions on which views are modifiable. An INSTEAD OF trigger can be used on object views as well as relational views that are not otherwise modifiable.

contd.

Types of Triggers (contd.)

Modify Views (contd.)

- A view is inherently modifiable if data can be inserted, updated, or deleted without using INSTEAD OF triggers and if it conforms to the restrictions. Even if the view is inherently modifiable, you might want to perform validations on the values being inserted, updated or deleted. INSTEAD OF triggers can also be used in this case. Here the trigger code performs the validation on the rows being modified and if valid, propagates the changes to the underlying tables, statement or rows that are affected.

Statement Triggers

- A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected.
  For example: If a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.
- Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected.
- For example: Use a statement trigger to:
  - make a complex security check on the current time or user
  - generate a single audit record

BEFORE and AFTER Triggers

- When defining a trigger, you can specify the trigger timing — whether the trigger action has to be run before or after the triggering statement.
- BEFORE and AFTER apply to both statement and row triggers.
- BEFORE and AFTER triggers fired by DML statements can be defined only on tables, not on views. However, triggers on the base tables of a view are fired if an INSERT, UPDATE, or DELETE statement is issued against the view.
  BEFORE and AFTER triggers fired by DDL statements can be defined only on the database or a schema, and not on particular tables.

Examples on Trigger

Example 1

```
CREATE TABLE Account_log
(
deleteInfo VARCHAR2(20),
logging_date DATE
)
```

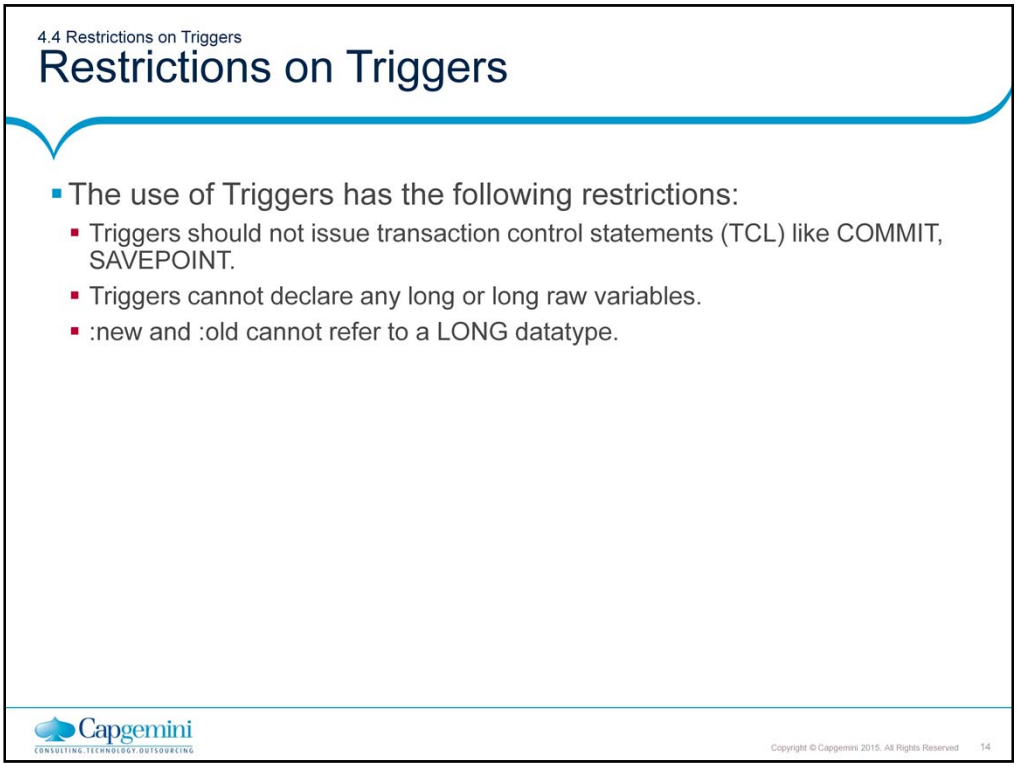
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Trigger creation code

```
CREATE or REPLACE TRIGGER

After_Delete_Row_product

AFTER   delete On Account_masters
FOR EACH ROW
BEGIN
INSERT INTO  Account_log
Values('After delete, Row level',sysdate);
END;
```

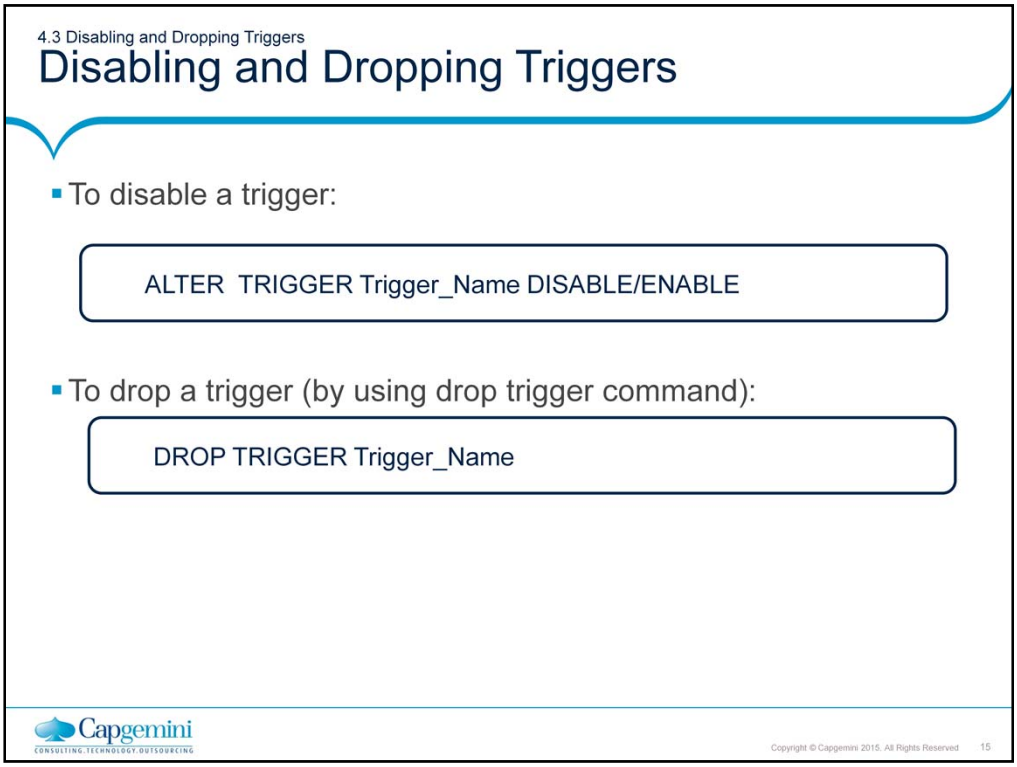
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13



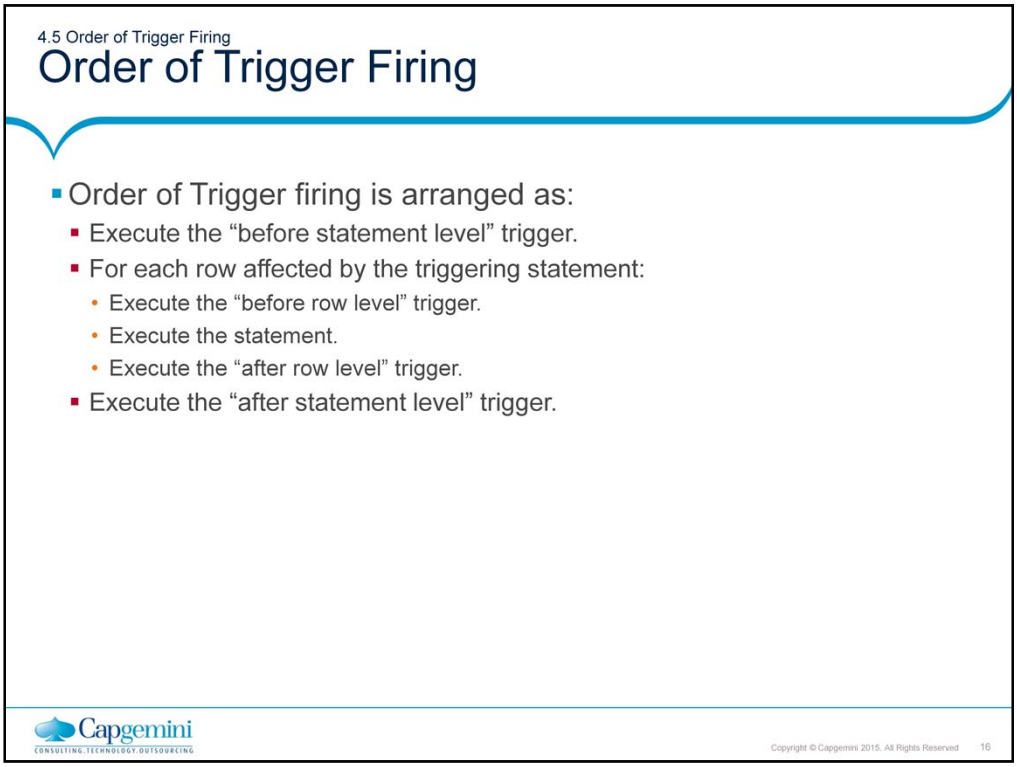
Restrictions on Triggers:

- A Trigger may not issue any transaction control statements (TCL) like COMMIT, SAVEPOINT.
  - Since the triggering statement and the trigger are part of the same transaction, whenever triggering statement is committed or rolled back the work done in the trigger gets committed or rolled back, as well.
- Any procedures or functions that are called by the Trigger cannot have any transaction control statements.
- Trigger body cannot declare any long or long raw variables. Also :new and :old cannot refer to a LONG datatype.
- There are restrictions on the tables that a trigger body can access depending on type of “triggers” and “constraints” on the table.



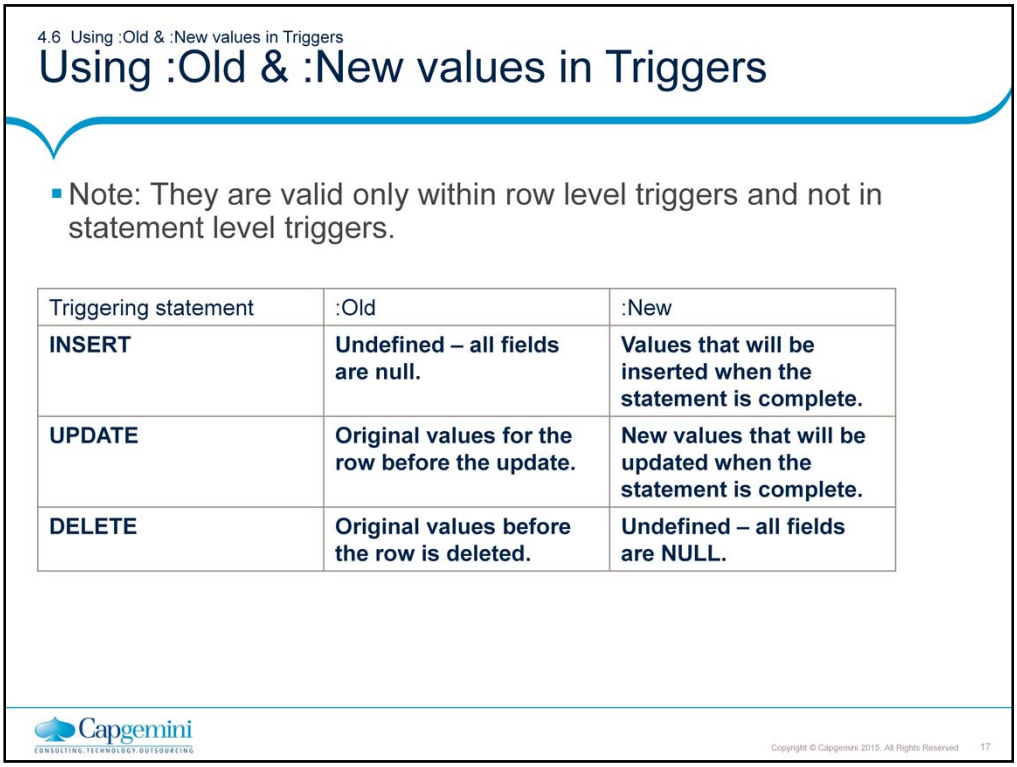
Note:

An UPDATE or DELETE statement affects multiple rows of a table.
If the trigger is fired once for each affected row, then FOR EACH ROW clause is required. Such a trigger is called a ROW trigger.
If the FOR EACH ROW clause is absent, then the trigger is fired only once for the entire statement. Such triggers are called STATEMENT triggers



Note:

- As each trigger is fired, it will see the changes made by the earlier triggers, as well as the database changes made by the statement.
- The order in which triggers of same type are fired is not defined.
- If the order is important, combine all the operations into one trigger.

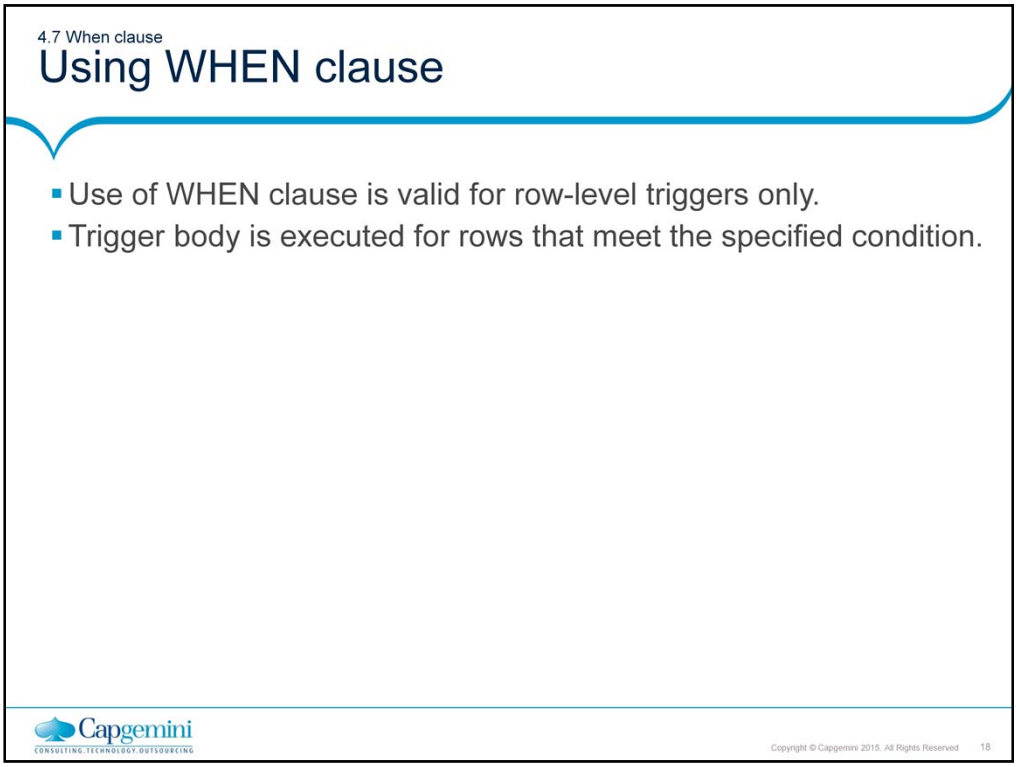


Using :old and :new values in Row Level Triggers:

- Row level trigger fires once per row that is being processed by the triggering statement.
- Inside the trigger, you can access the row that is currently being processed. This is done through keywords :new and :old (they are called as pseudo records). The meaning of the terms is as shown in the slide.

Note:

- The pseudo records are valid only within row level triggers and not in statement level triggers.
  - :old values are not available if the triggering statement is INSERT.
  - :new values are not available if the triggering statement is DELETE.
- Each column is referenced by using the notation :old.Column_Name or :new.Column_Name.
- If a column is not updated by the triggering update statement, then :old and :new values remain the same.



Using WHEN Clause:

- The WHEN clause is valid for row-level triggers only. If present, the trigger body will be executed only for those rows that meet the condition specified by the WHEN clause.
- The :new and :old records can be used here without colon.

contd.

4.8 Examples on Trigger

Example 2

```
CREATE TABLE Account_masters
(
account_no NUMBER(6) PRIMARY KEY,
cust_id NUMBER(6) ,
account_type CHAR(3) CONSTRAINT chk_acc_type
CHECK(account_type IN ('SAV','SAL')) ,
Ledger_balance NUMBER(10)
)
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

Trigger creation code

```
CREATE OR REPLACE TRIGGER trg_acc_master_ledger
   before INSERT OR UPDATE OF Ledger_balance ,account_type
ON Account_masters
   FOR EACH ROW

   WHEN (NEW.account_type ='SAV')
   DECLARE

    v_led_bal NUMBER(10);
   BEGIN
    v_led_bal:= :NEW.Ledger_balance;

    IF v_led_bal < 5000 THEN
      :NEW.Ledger_balance := 5000;
    END if;
   END trg_acc_master_ledger;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Summary

- Database Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly: Database Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly:
  - whenever a table or view is modified, or
  - when some user actions or database system actions occur
- There are three types of triggers:
  - Statement based triggers
  - Timing based triggers
  - Level based triggers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21

Summary

- Disabling and Dropping triggers can be done instead of actually removing the triggers
- Order of trigger firing is decided depending on the type of triggers used in the sequence

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Review – Questions

- Question 1: Triggers should not issue Transaction Control Statements (TCL).
  - True / False

- Question 2: The :new and :old records must be used in WHEN clause with a colon.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23

Review – Questions

- Question 3: A ___ is a table that is currently being modified by a DML statement.
- Question 4: A ___ is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects.

Oracle PL/SQL

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
05-Feb-2009	0.1D	Rajita Dhumal	Content Creation
09-Feb-2009		CLS team	Review
02-Jun-2011	2.0	Anu Mitra	Integration Refinements
30-Nov-2012	3.0	HareshkumarChandiramani	Revamp of Assignments and Conversion to iGATE format.
22-Apr--2015	4.0	Kavita Arora	Rearranging the lab questions
9-May-2016	5.0	Kavita Arora	Integration Refinements

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning Oracle 9i. It comprises 'To Do' assignments. Follow the steps provided and work out the 'To Do' assignments.

Setup Checklist for Oracle 9i

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP,7.
- Memory: 32MB of RAM (64MB or more recommended)

Please ensure that the following is done:

- Oracle Client is installed on every machine
- Connectivity to Oracle Server

Instructions

- For all coding standards refer Appendix A. All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory Oracle 9i_assgn. For each lab exercise create a directory as lab <lab number>.

Learning More (Bibliography if applicable)

- Oracle10g - SQL - Student Guide - Volume 1 by Oracle Press
- Oracle10g - SQL - Student Guide - Volume 2 by Oracle Press
- Oracle10g database administration fundamentals volume 1 by Oracle Press
- Oracle10g Complete Reference by Oracle Press
- Oracle10g SQL with an Introduction to PL/SQL by Lannes L. Morris-Murphy

Lab 1.Introduction to PL/SQL and Cursors

Goals	• The following set of exercises are designed to implement the following • PL/SQL variables and data types • Create, Compile and Run anonymous PL/SQL blocks • Usage of Cursors
Time	1hr

1.1 Identify the problems(if any) in the below declarations:

```
DECLARE
V_Sample1 NUMBER(2);
V_Sample2 CONSTANT NUMBER(2) ;
V_Sample3 NUMBER(2) NOT NULL ;
V_Sample4 NUMBER(2) := 50;
V_Sample5 NUMBER(2) DEFAULT 25;
```

Example 1: Declaration Block

1.2 The following PL/SQL block is incomplete.

Modify the block to achieve requirements as stated in the comments in the block.

```
DECLARE --outer block
var_num1 NUMBER := 5;
BEGIN
DECLARE --inner block
var_num1 NUMBER := 10;
BEGIN
DBMS_OUTPUT.PUT_LINE('Value for var_num1:' ||var_num1);
--Can outer block variable (var_num1) be printed here.IfYes,Print the same.
END;
--Can inner block variable(var_num1)  be printed here.IfYes,Print the same.
END;
```

Example 2: PL/SQL block

1.3 Write a PL/SQL program

Write a PL/SQL program to display the details of the employee number 7369.

1.4 Write a PL/SQL program

Write a PL/SQL program to accept the Employee Name and display the details of that Employee including the Department Name.

1.5.Write a PL/SQL block to increase the salary of employees

Write a PL/SQL block to increase the salary of employees either by 30 % or 5000 whichever is minimum for a given Department_Code.

Find out 30% of salary, if it is more than 5000, increase by 5000. If it is less than 5000, increase by 30% of salary

Lab 2.Lab 2.Exception Handling

Goals	Implementing Exception Handling ,Analyzing and Debugging
Time	30 mins

2.1 The following PL/SQL block attempts to calculate bonus of staff.

The following PL/SQL block attempts to calculate bonus of staff for a given MGR_CODE. Bonus is to be considered as twice of salary. Though Exception Handling has been implemented but block is unable to handle the same.

Debug and verify the current behavior to trace the problem.

```
DECLARE
V_BONUS V_SAL%TYPE;
V_SAL STAFF_MASTER.STAFF_SAL%TYPE;

BEGIN
SELECT STAFF_SAL INTO V_SAL
FROM STAFF_MASTER
WHERE MGR_CODE=100006;

V_BONUS:=2*V_SAL;
DBMS_OUTPUT.PUT_LINE('STAFF SALARY IS ' || V_SAL);
DBMS_OUTPUT.PUT_LINE('STAFF BONUS IS ' || V_BONUS);

EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('GIVEN CODE IS NOT VALID.ENTER VALID CODE');
END;
```

Example 3: PL/SQL block

2.2 Rewrite the above block.

Rewrite the above block to achieve the requirement.

2.3: Write a PL/SQL program

Write a PL/SQL program to check for the commission for an employee no 7369. If no commission exists, then display the error message. Use Exceptions.

Lab 3.Database Programming

Goals	• The following set of exercises are designed to implement the following • Implement business logic using Database Programming like Procedures and Functions • Implement validations in Procedures and Functions
Time	2 Hrs

Note: Procedures and functions should handle validations, pre-defined oracle server and user defined exceptions wherever applicable. Also use cursors wherever applicable.

3.1. Write a function to compute age.

The function should accept a date and return age in years.

3.2 Write a procedure to find the manager of a staff.

Procedure should return the following – Staff_Code, Staff_Name, Dept_Code and Manager Name.

3.3. Write a function to compute the following.

Function should take Staff_Code and return the cost to company.

DA = 15% Salary, HRA= 20% of Salary, TA= 8% of Salary.

Special Allowance will be decided based on the service in the company.

< 1 Year	Nil
>=1 Year< 2 Year	10% of Salary
>=2 Year< 4 Year	20% of Salary
>4 Year	30% of Salary

3.4. Write a procedure that accept Staff_Code

Write a procedure that accept Staff_Code and update the salary and store the old salary details in Staff_Master_Back (Staff_Master_Back has the same structure without any constraint) table.

Exp< 2 then no Update

Exp> 2 and < 5 then 20% of salary

Exp> 5 then 25% of salary

3.5. Write a procedure to insert details into Book_Transaction table.

Procedure should accept the book code and staff/student code. Date of issue is current date and the expected return date should be 10 days from the current date. If the expected return date falls on Saturday or Sunday, then it should be the next working day

Appendices

Appendix A: Oracle Standards

Key points to keep in mind:

1. Write comments in your stored Procedures, Functions and SQL batches generously, whenever something is not very obvious. This helps other programmers to clearly understand your code. Do not worry about the length of the comments, as it will not impact the performance.
2. Prefix the table names with owner names, as this improves readability, and avoids any unnecessary confusion.

Some more Oracle standards:

To be shared by Faculty in class

Appendix B: Coding Best Practices

1. Perform all your referential integrity checks and data validations by using constraints (foreign key and check constraints). These constraints are faster than triggers. So use triggers only for auditing, custom tasks, and validations that cannot be performed by using these constraints.
2. Do not call functions repeatedly within your stored procedures, triggers, functions, and batches. For example: You might need the length of a string variable in many places of your procedure. However do not call the LENGTH function whenever it is needed. Instead call the LENGTH function once, and store the result in a variable, for later use.

Appendix C: Table of Examples

