PLSQL

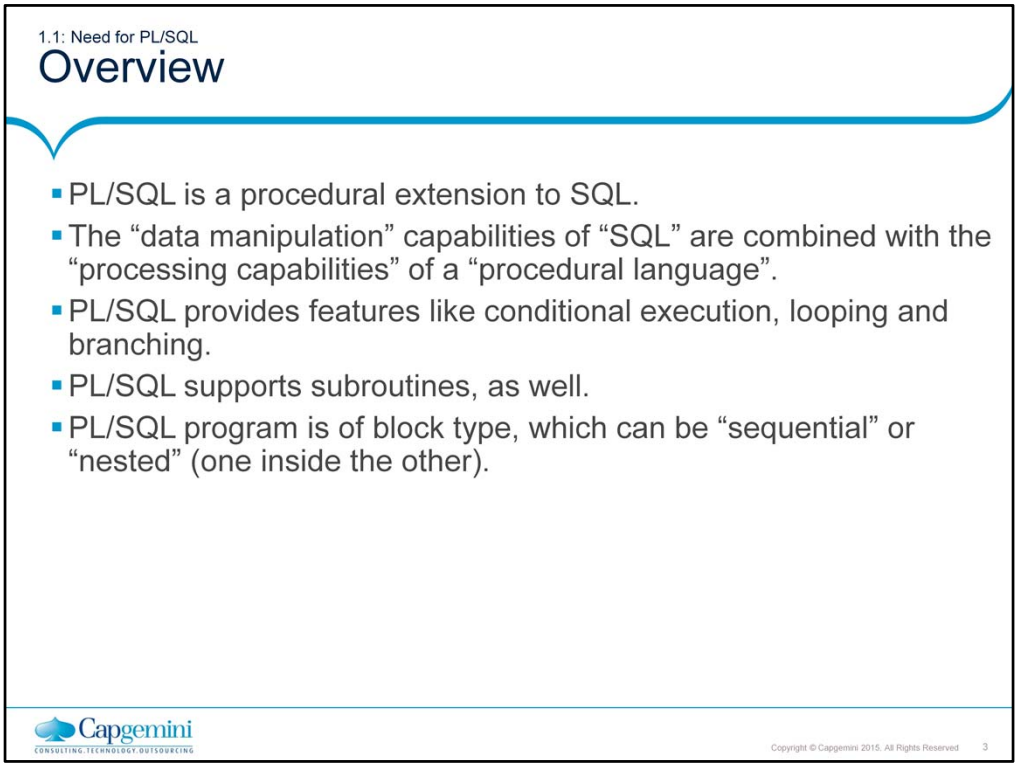
Lesson 01: PLSQL Basics, Datatypes

Lesson Objectives

- To understand the following topics:
  - Introduction and Need for PLSQL
  - Datatypes
  - Scalar
  - Composite Variables

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

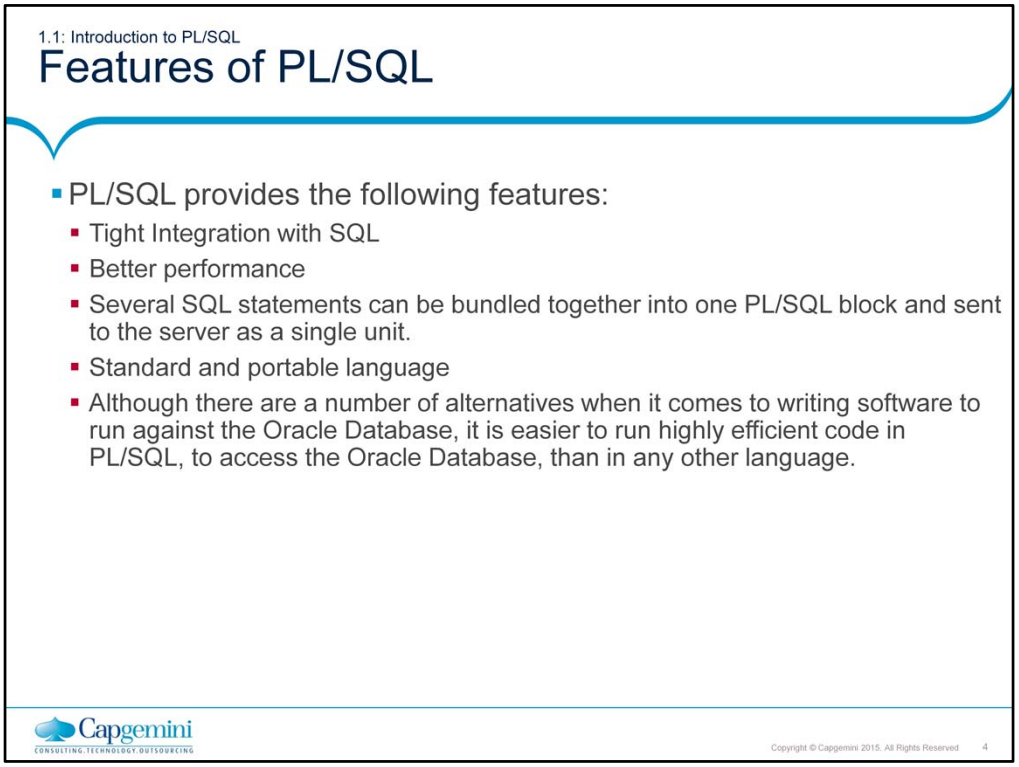


Introduction to PL/SQL:
PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is “more powerful than SQL”.

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.

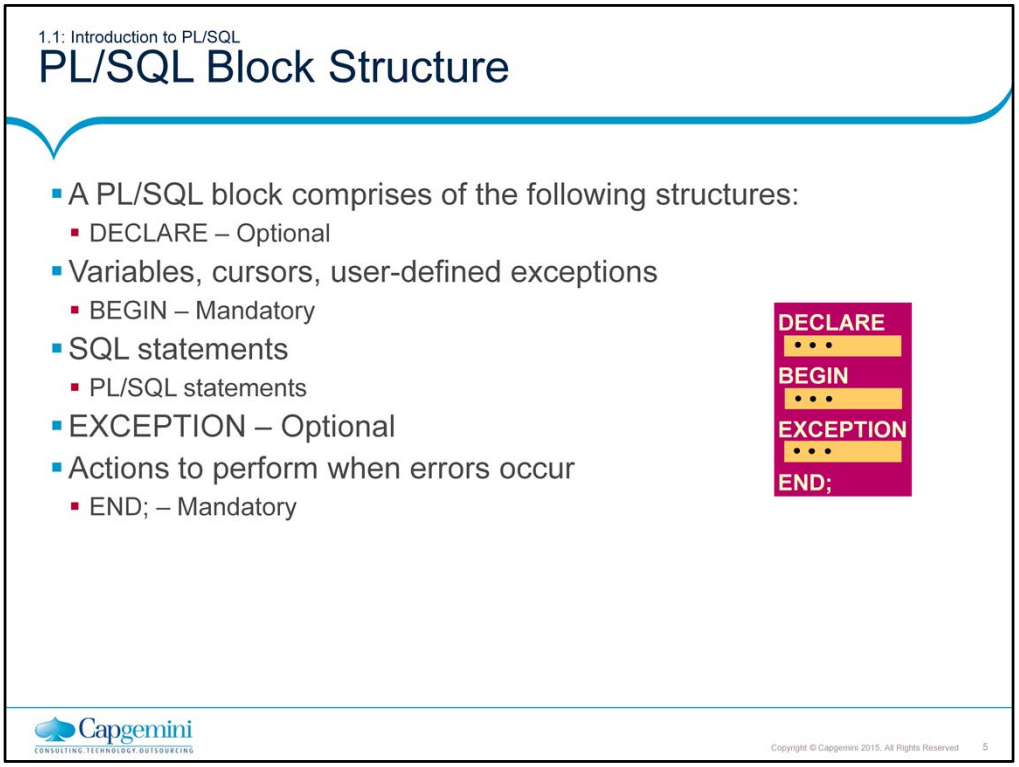
Portable:

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.

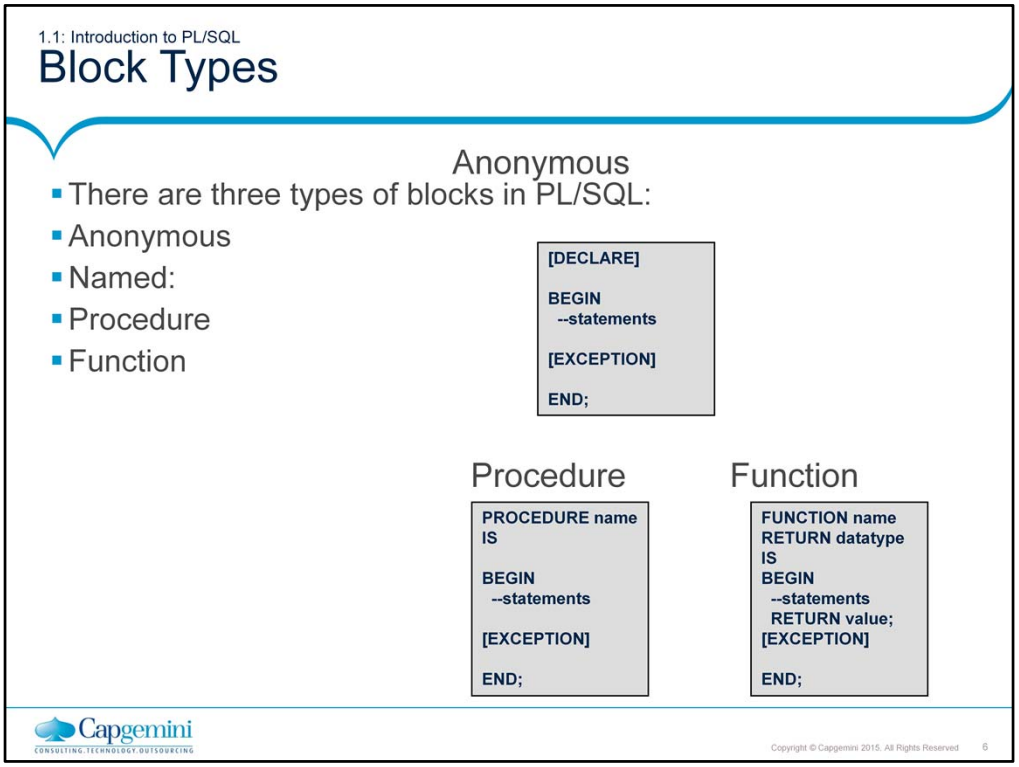
Efficient:

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.



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

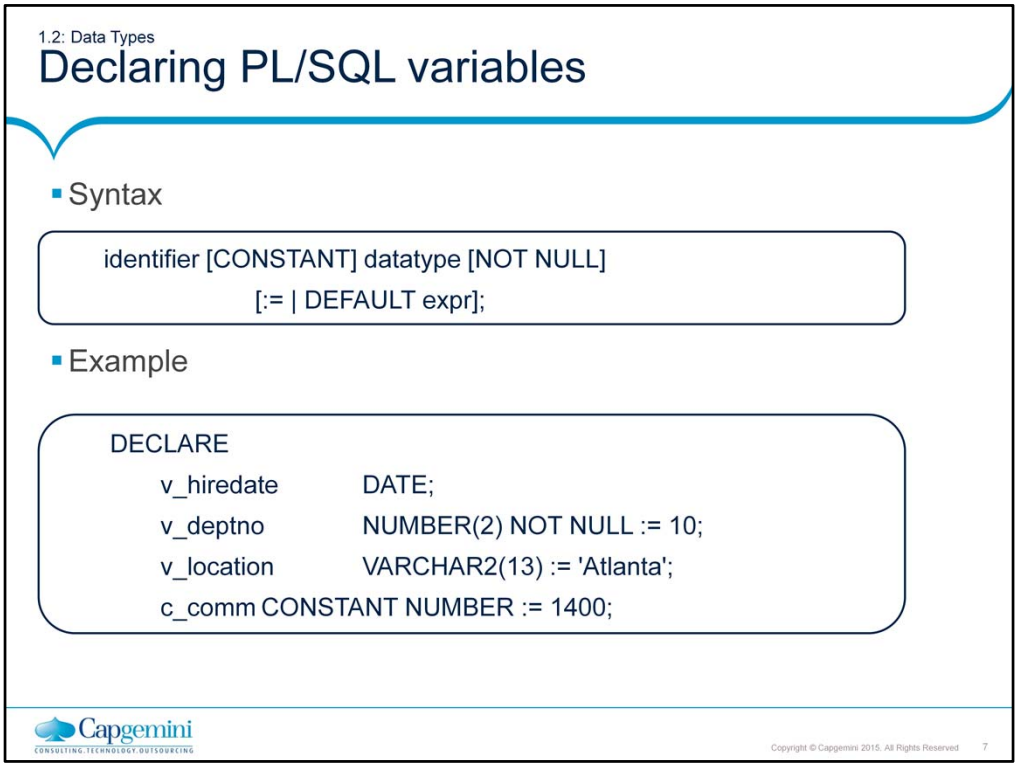
Anonymous blocks are unnamed blocks. They are declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at runtime.

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”.



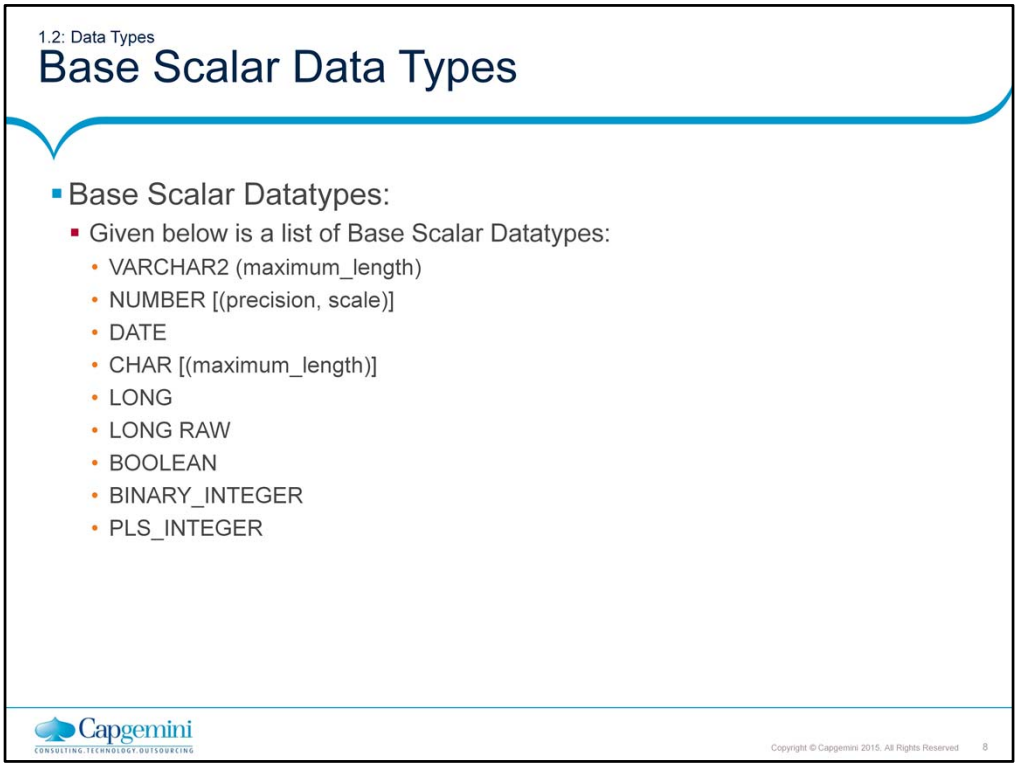
Declaring PL/SQL Variables:
You need to declare all PL/SQL identifiers within the “declaration section” before referencing them within the PL/SQL block.
You have the option to assign an initial value.

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.



Base Scalar Datatypes:
NUMBER

This can hold a numeric value, either integer or floating point. It is same as the number database type.

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;

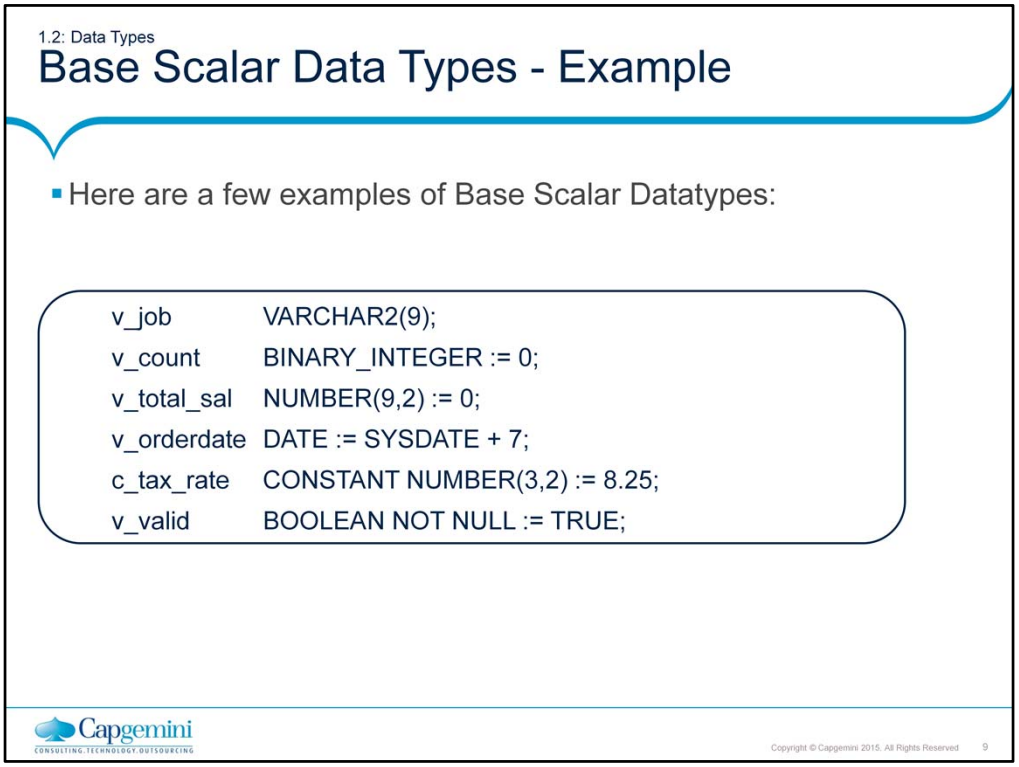
VARCHAR2 (L)

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.

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.):

LONG

PL/SQL LONG type is just 32,767 bytes. It behaves similar to LONG DATABASE type.

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.

BOOLEAN

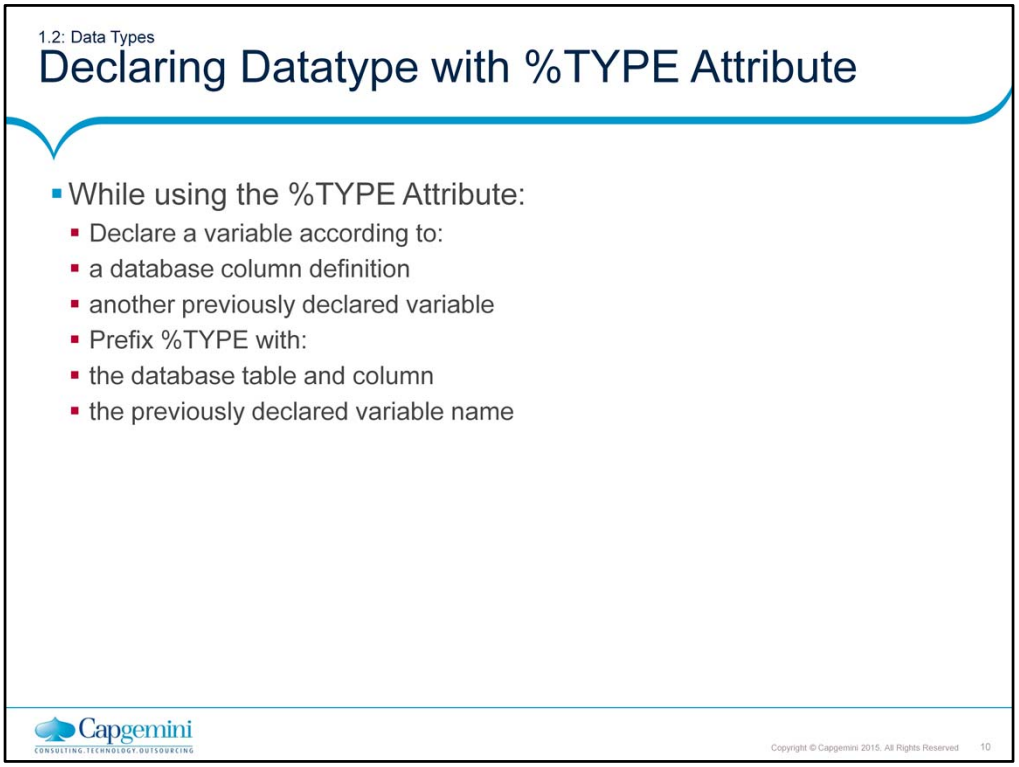
A Boolean type variable can only have one of the two values, i.e. either TRUE or FALSE. They are mostly used in control structures.

V_Does_Dept_Exist BOOLEAN := TRUE;

V_ Flag BOOLEAN := 0; -- illegal

One more example

```
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;
/
```

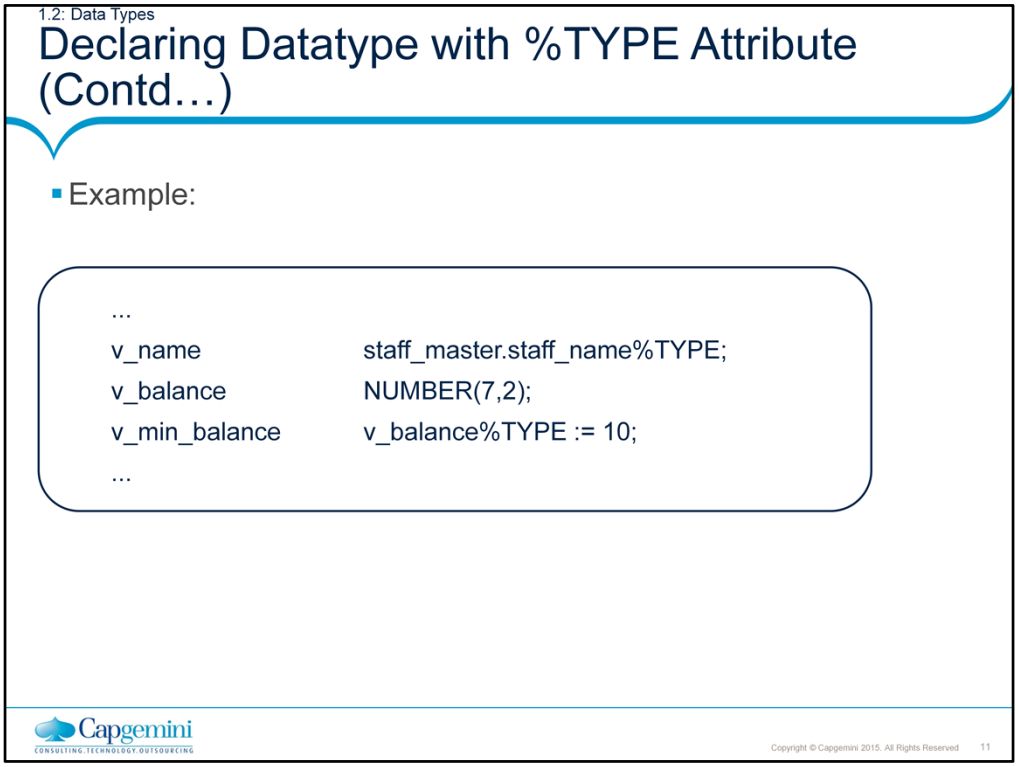


Reference types:
A “reference type” in PL/SQL is the same as a “pointer” in C. A “reference type” variable can point to different storage locations over the life of the program.
Using %TYPE
%TYPE is used to declare a variable with the same datatype as a column of a specific table. This datatype is particularly 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:

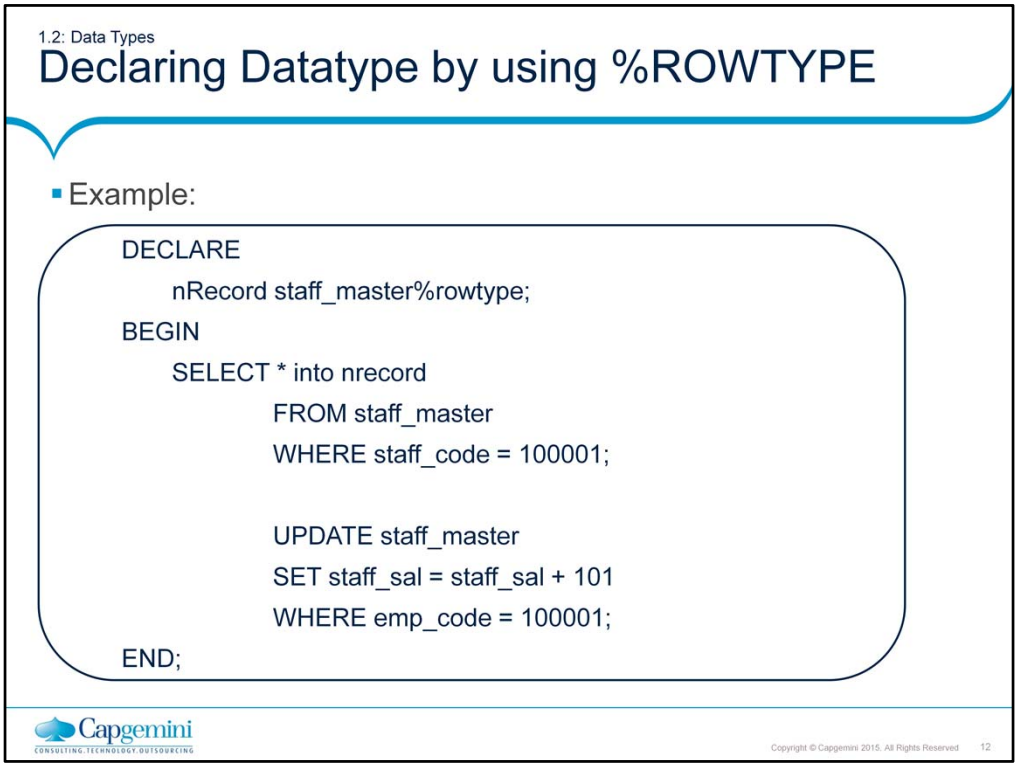
Note: Datatype of V_Empno is same as datatype of Empno column of the EMP table.

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



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:

```
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;

For example:

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

1.2: Data Types

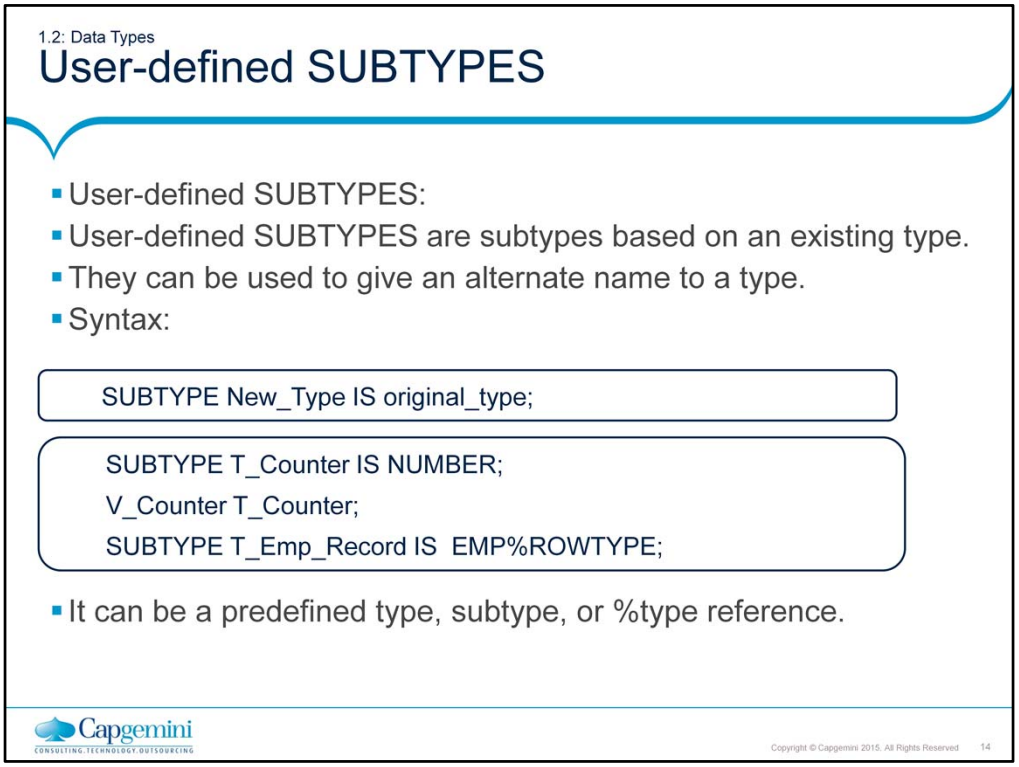
Inserting and Updating using records

- 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

 13



User-defined SUBTYPES:
A SUBTYPE is a PL/SQL type based on an existing type. A subtype can be used to give an alternate name to a type to indicate its purpose.
A new sub_type base type can be a predefined type, subtype, or %type reference. You can declare a dummy variable of the desired type with the constraint and use %TYPE in the SUBTYPE definition.

```
V_Dummy      NUMBER(4);
SUBTYPE  T_Counter  IS  V_Dummy%TYPE;

V_Counter          T_Counter ;
SUBTYPE  T_Numeric  IS NUMBER;

V_Counter  IS  T_Numeric(5);
```

1.2: Data Types

User-defined SUBTYPES (Contd…)

- It is illegal to constrain a subtype.

```
SUBTYPE  T_Counter  IS NUMBER(4)    -- Illegal
```

- Possible solutions:

```
V_Dummy   NUMBER(4);
SUBTYPE T_Counter   IS       V_Dummy%TYPE;
V_Counter    T_Counter ;
SUBTYPE    T_Numeric  IS     NUMBER;
V_Counter  IS  T_Numeric(5);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

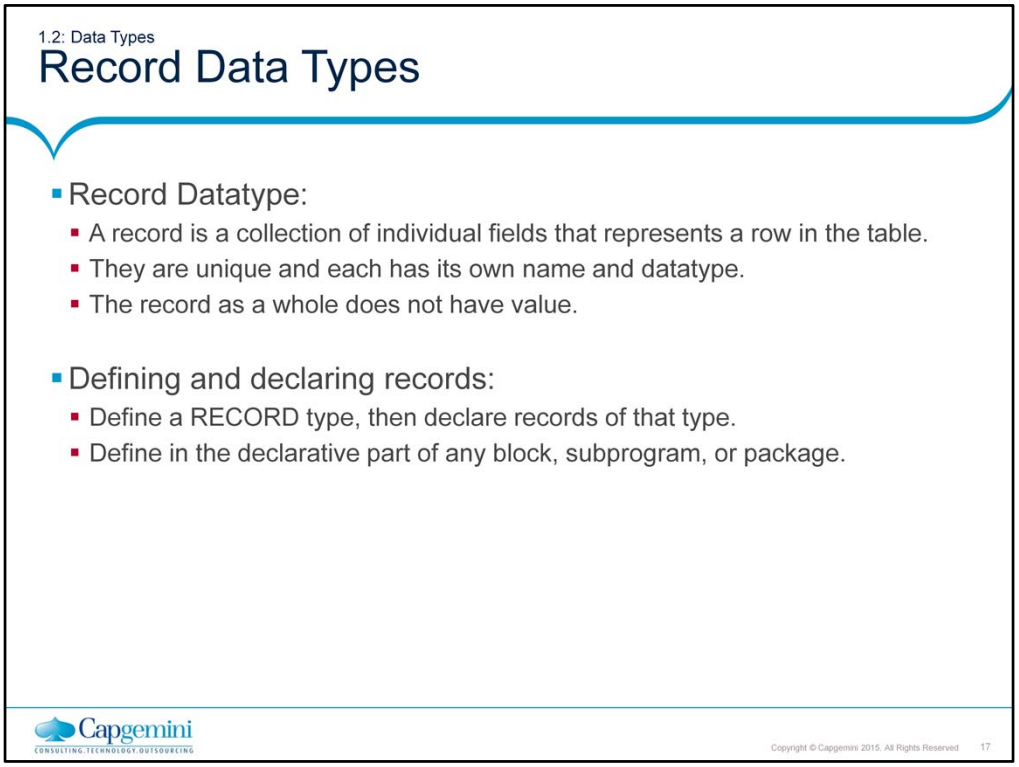
1.2: Data Types

Composite Data Types

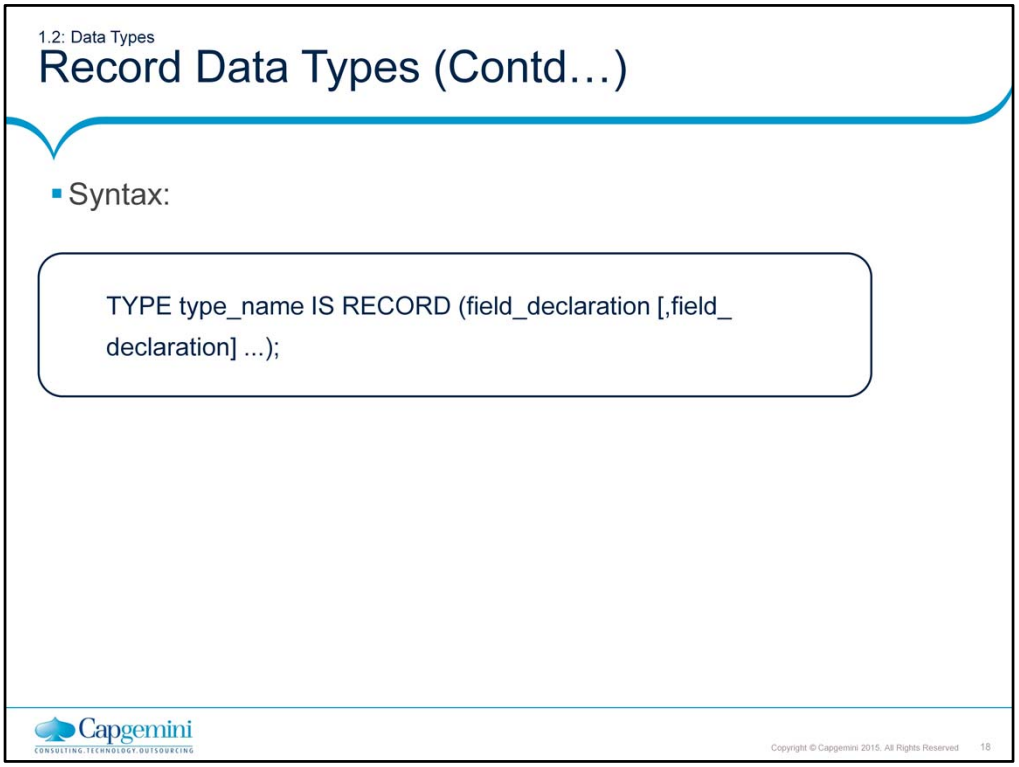
- Composite Datatypes in PL/SQL:
  - Two composite datatypes are available in PL/SQL:
  - records
  - tables
- A composite type contains components within it. A variable of a composite type contains one or more scalar variables.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16



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.



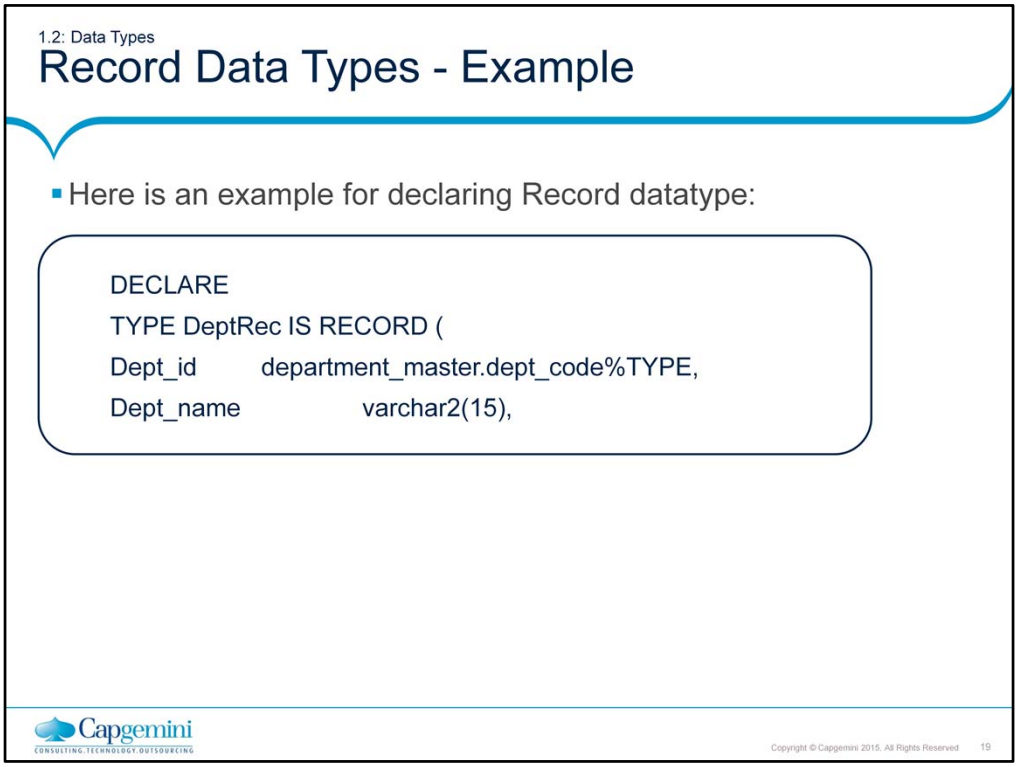
Defining and Declaring Records

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);

1.2: Data Types

Record Data Types - Example (Contd…)

- 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

 20

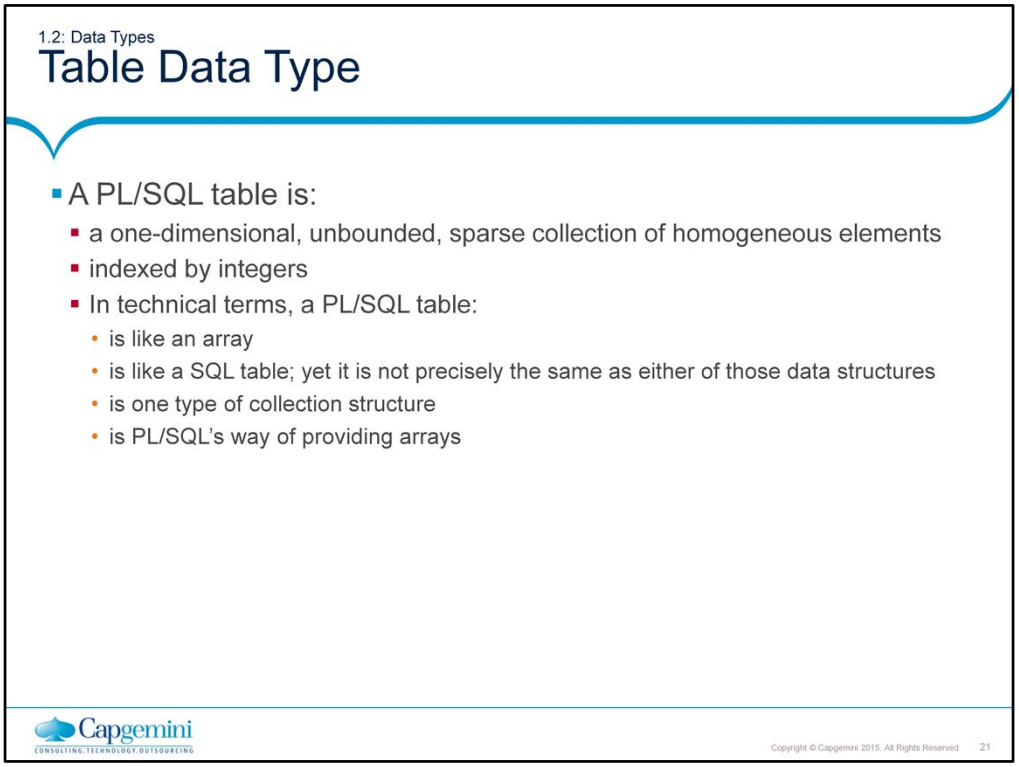


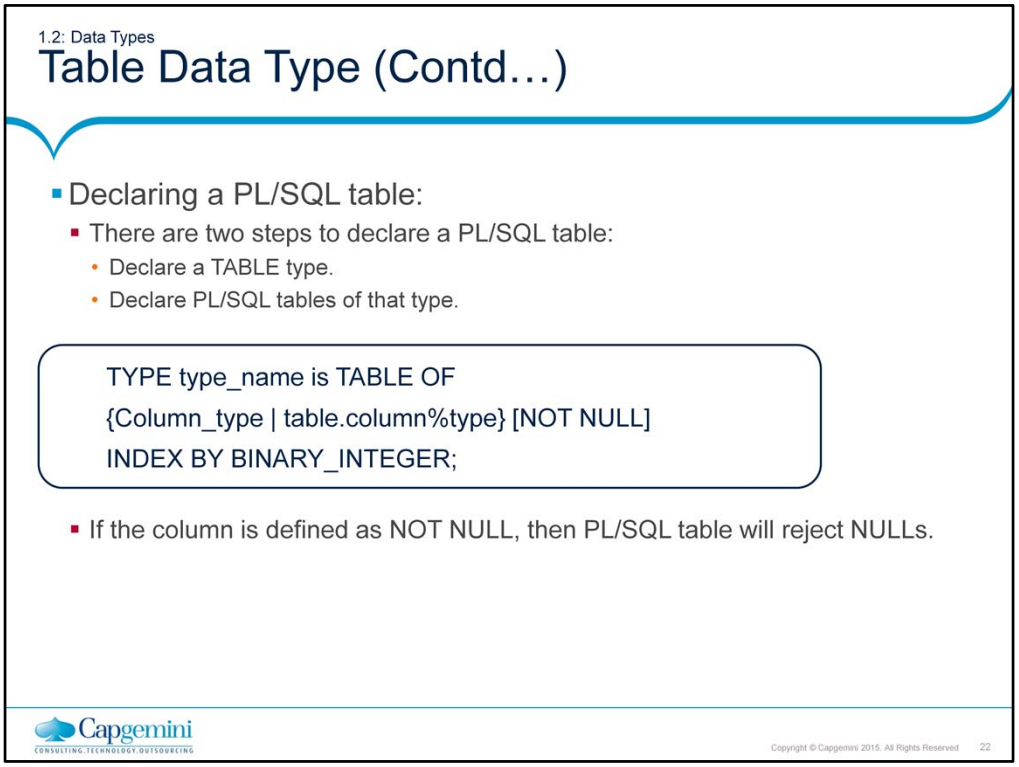
Table Datatype

Like PL/SQL records, the table is another composite datatype. PL/SQL tables are objects of type TABLE, and look similar to database tables but with slight difference. PL/SQL tables use a primary key to give you array-like access to rows.

Like the size of the database table, the size of a PL/SQL table is unconstrained. That is, the number of rows in a PL/SQL table can dynamically increase. So your PL/QSL table grows as new rows are added. PL/SQL table can have one column and a primary key, neither of which can be named.

The column can have any datatype, but the primary key must be of the type BINARY_INTEGER.

Arrays are like temporary tables in memory. Thus they are processed very quickly. Like the size of the database table, the size of a PL/SQL table is unconstrained. The “column” can have any datatype. However, the “primary key” must be of the type BINARY_INTEGER.



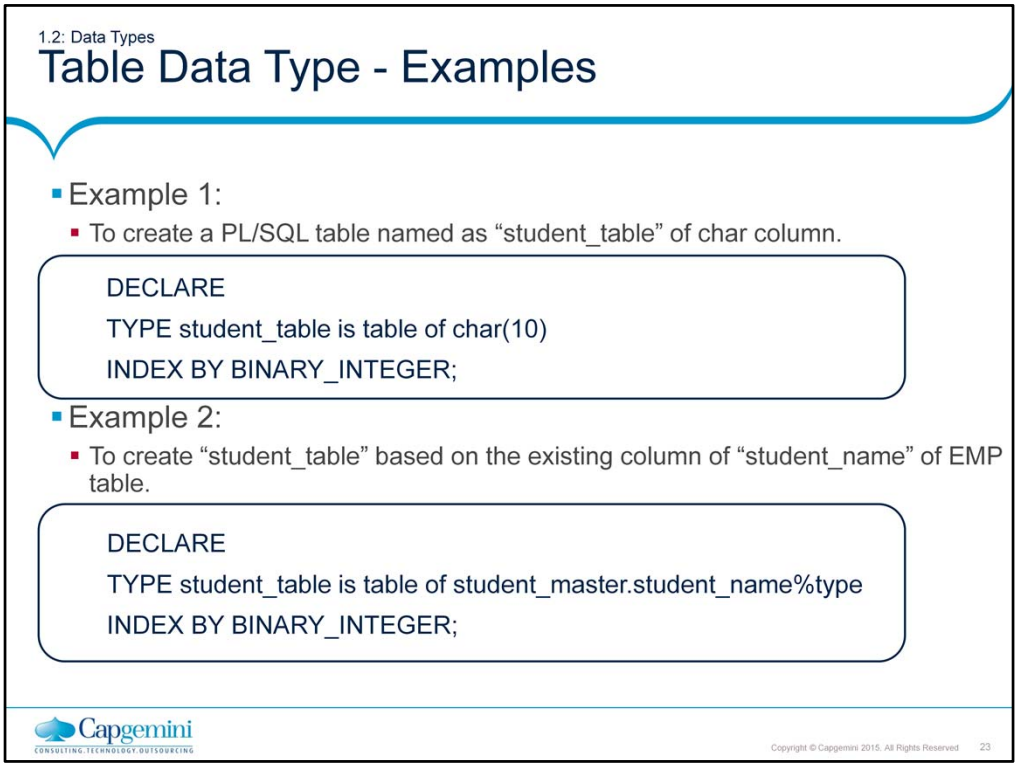
Declaring a PL/SQL table

PL/SQL tables must be declared in two steps. First you declare a TABLE type, then declare PL/SQL tables of that type. You can declare TABLE type in the declarative part of any block, subprogram or package.

In the syntax on the above slide:

Type_name is type specifier used in subsequent declarations to define PL/SQL tables and column_name is any datatype.

You can use %TYPE attribute to specify a column datatype. If the column to which table.column refers is defined as NOT NULL, the PL/SQL table will reject NULLs.

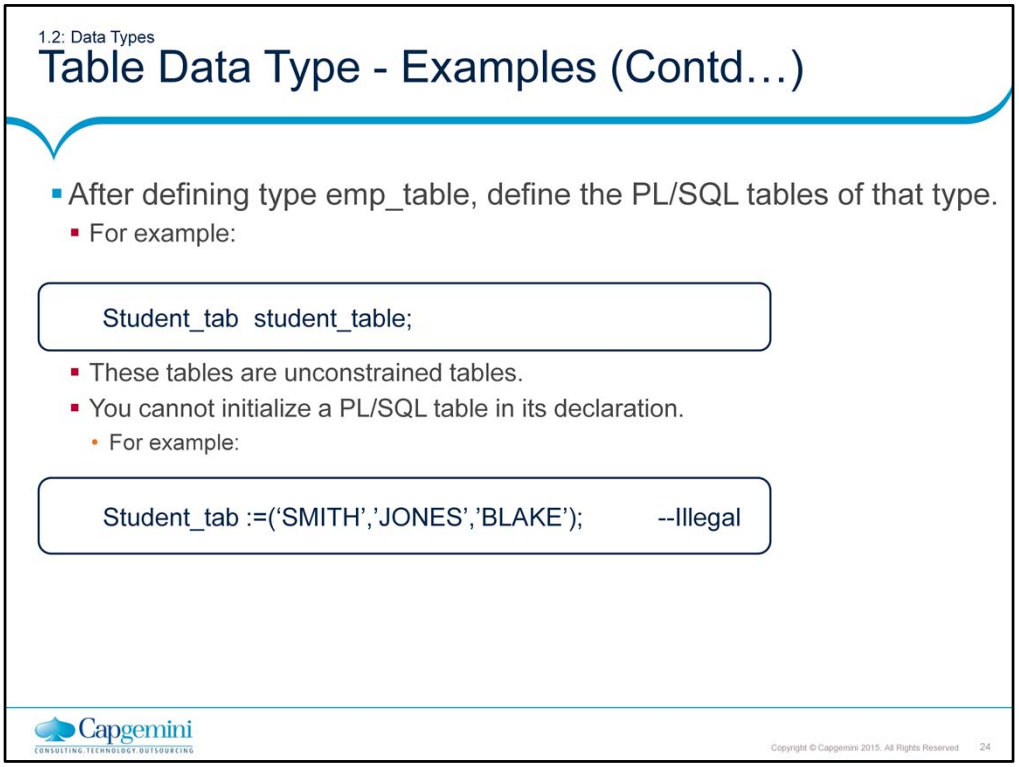


Declaring a PL/SQL table (contd.):
Example 3:

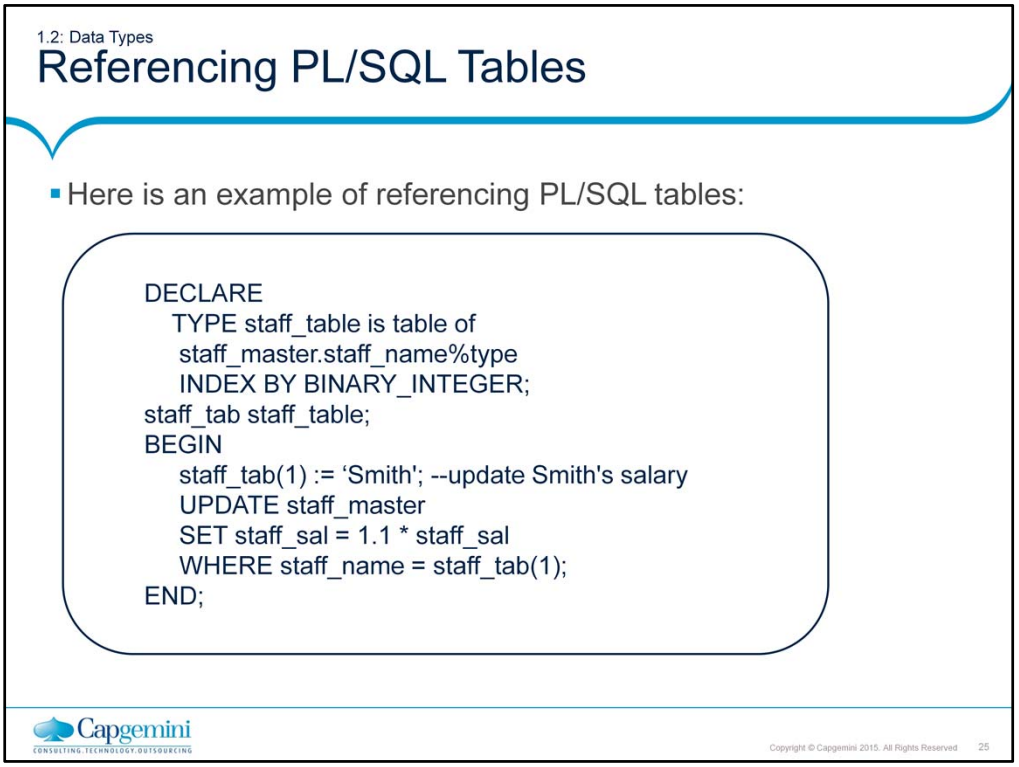
To declare a NOT NULL constraint

Note: INDEX BY BINARY INTERGER is a mandatory feature of the PL/SQL table declaration.

```
DECLARE
TYPE student_table is table of
student_master.student_name%TYPE NOT NULL
INDEX BY BINARY_INTEGER;
```



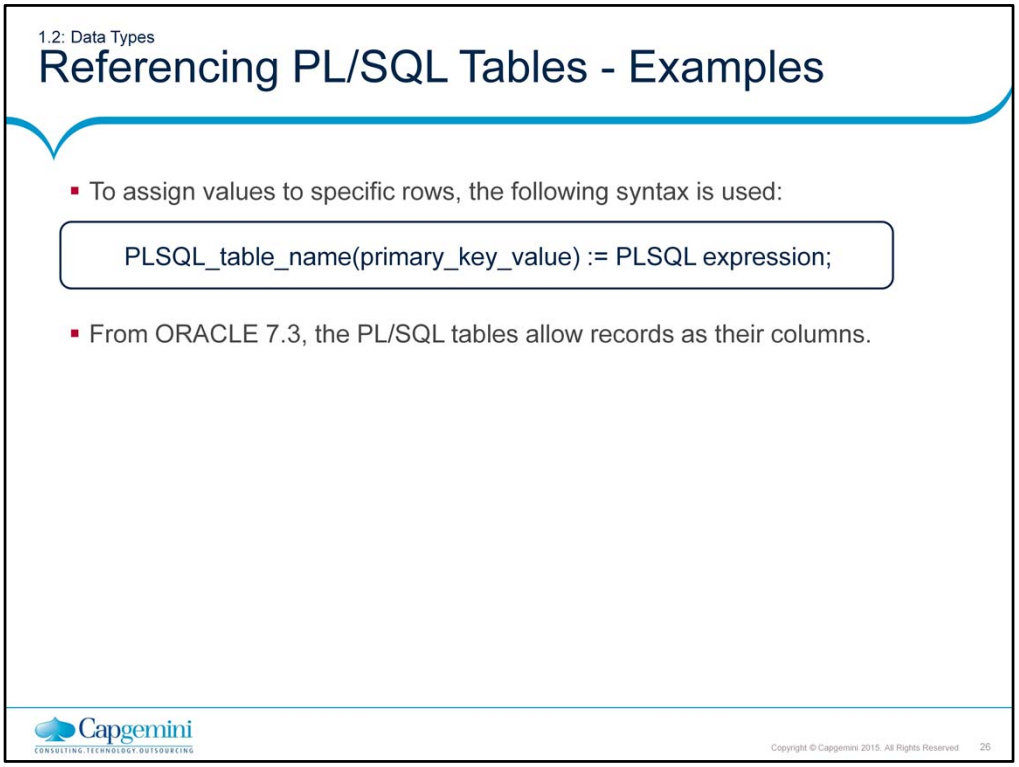
Note:
The PL/SQL tables are unconstrained tables, because its primary key can assume any value in the range of values defined by BINARY_INTEGER.
You cannot initialize a PL/SQL table in its declaration.



Referencing PL/SQL tables:
To reference rows in a PL/SQL table, you specify the PRIMARY KEY value using the array-like syntax as shown below:

When primary key value belongs to type BINARY_INTEGER you can reference the first row in PL/SQL table named emp_tab as shown in the slide.

PL/SQL table_name (primary key value)



Referencing PL/SQL tables:
Examples:

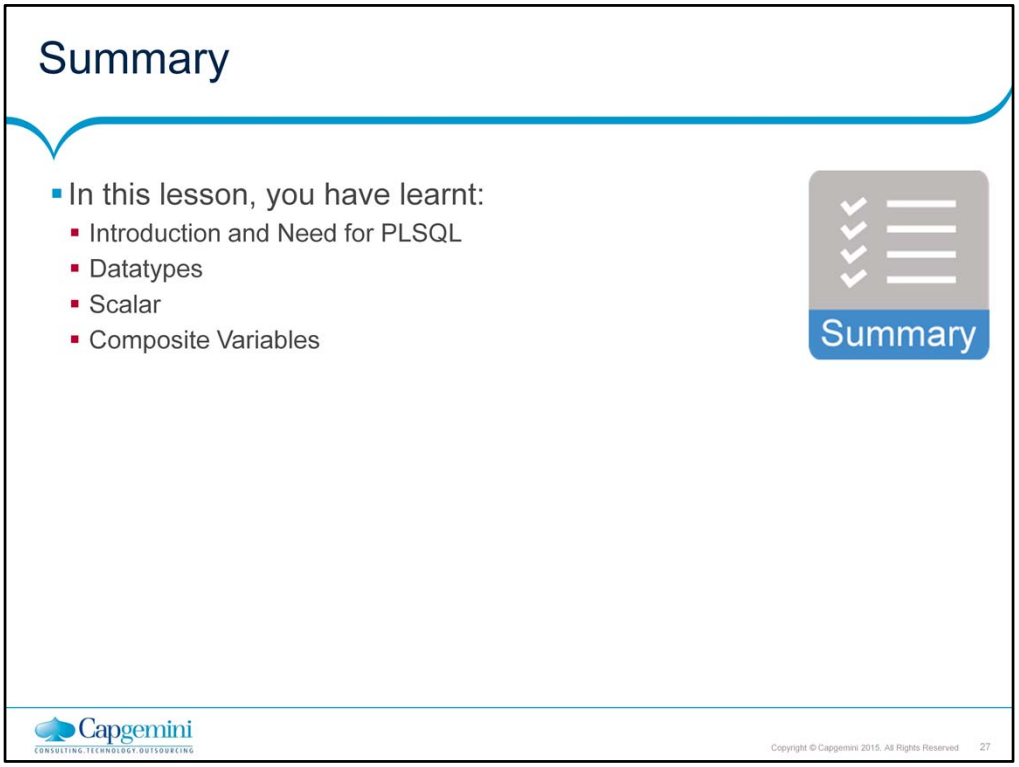
```
type staff_rectype is record (
staff_id integer,
staff_sname varchar2(60)) ;

type staff_table is table of staff_rectype
index by binary_integer;

staff_tab       staff_table;
```

Referencing fields of record elements in PL SQL tables:

```
staff_tab(375).staff_sname := 'SMITH';
```



Add the notes here.

Review Question

- Question 1: User-defined SUBTYPES are subtypes based on an existing type.
  - True / False
- Question 2: A record is a collection of individual fields that represents a row in the table.
  - True/ False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 28

Add the notes here.

Review Question

- Question 3: %ROWTYPE is used to declare a variable with the same datatype as a column of a specific table.
  - True / False
- Question 4: PL/SQL tables use a primary key to give you array-like access to rows.
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

Add the notes here.