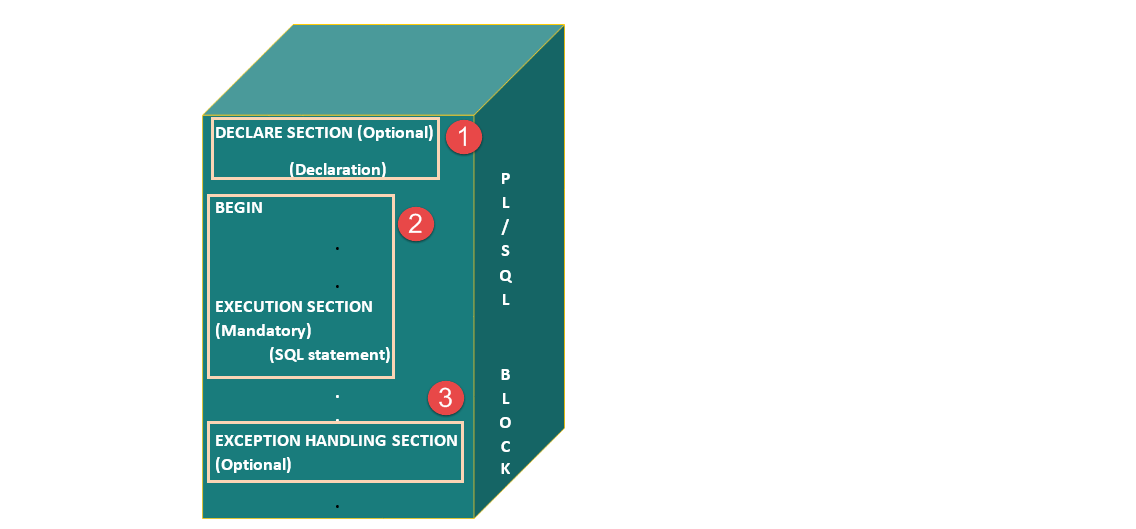
PL SQL basically stands for "Procedural Language extensions to SQL." This is the extension of Structured Query Language (SQL) that is used in Oracle.

PL/SQL blocks have a pre-defined structure in which the code is to be grouped. Below are different sections of PL/SQL blocks.

* Declaration section
* Execution section
* Exception-Handling section



**Declaration Section**

This is the first section of the PL/SQL blocks. This section is an optional part. This is the section in which the declaration of variables, cursors, exceptions, subprograms, pragma instructions and collections that are needed in the block will be declared. Below are few more characteristics of this part.

This particular section is optional and can be skipped if no declarations are needed.

This should be the first section in a PL/SQL block, if present.

This section starts with the keyword 'DECLARE' for triggers and anonymous block. For other subprograms, this keyword will not be present. Instead, the part after the subprogram name definition marks the declaration section.

This section should always be followed by execution section.

**Execution Section**

Execution part is the main and mandatory part which actually executes the code that is written inside it. Since the PL/SQL expects the executable statements from this block this cannot be an empty block, i.e., it should have at least one valid executable code line in it. Below are few more characteristics of this part.

This can contain both PL/SQL code and SQL code.

This can contain one or many blocks inside it as a nested block.

This section starts with the keyword 'BEGIN'.

This section should be followed either by 'END' or Exception-Handling section (if present)

**Exception-Handling Section:**

The exception is unavoidable in the program which occurs at run-time and to handle this Oracle has provided an Exception-handling section in blocks. This section can also contain PL/SQL statements. This is an optional section of the PL/SQL blocks.

This is the section where the exception raised in the execution block is handled.

This section is the last part of the PL/SQL block.

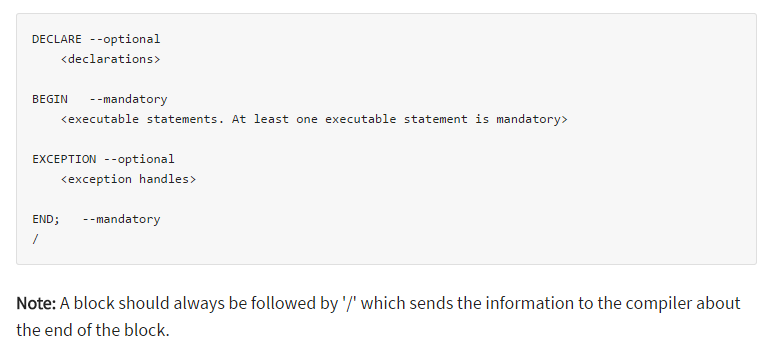
Control from this section can never return to the execution block.

This section starts with the keyword 'EXCEPTION'.

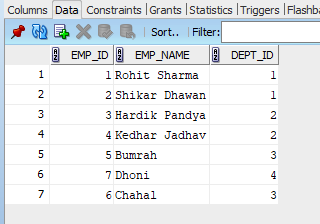
This section should always be followed by the keyword 'END'.

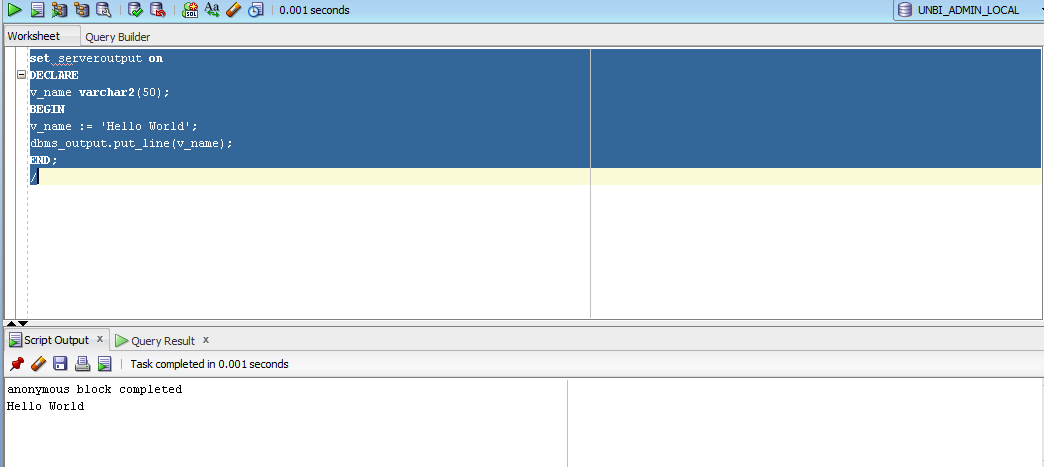
The Keyword 'END' marks the end of PL/SQL block.

**PL/SQL Block Syntax**



EMPLOYEE TABLE

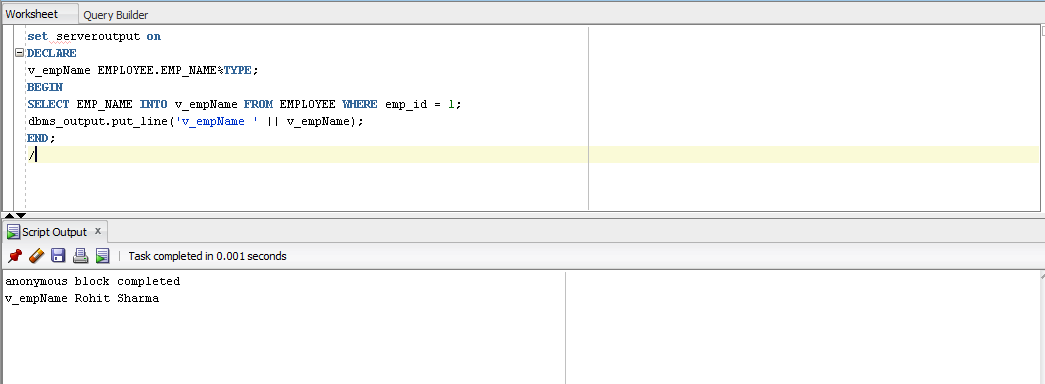




**%TYPE Attribute**

The %TYPE attribute lets you declare a constant, variable, field, or parameter to be of the same data type a previously declared variable, field, record, nested table, or database column. If the referenced item changes, your declaration is automatically updated.

An item declared with %TYPE (the referencing item) always inherits the data type of the referenced item. The referencing item inherits the constraints only if the referenced item is not a database column. The referencing item inherits the default value only if the referencing item is not a database column and does not have the NOT NULL constraint.



**Types of Loop in PL/SQL**

PL/SQL provides following three types of loops

* Basic loop statement
* For loop statement
* While loop statement

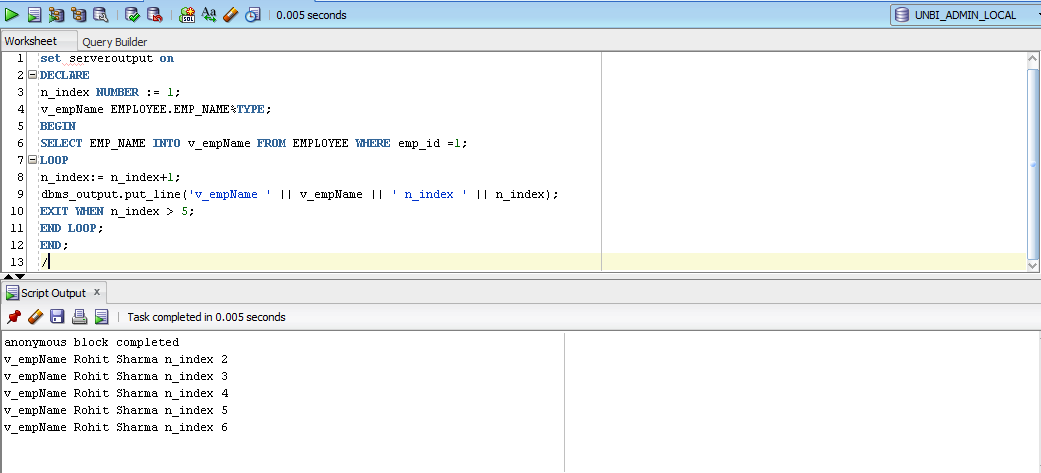
**Basic Loop Statement**

This loop statement is the simplest loop structure in PL/SQL. The execution block starts with keyword 'LOOP' and ends with the keyword 'END LOOP'.

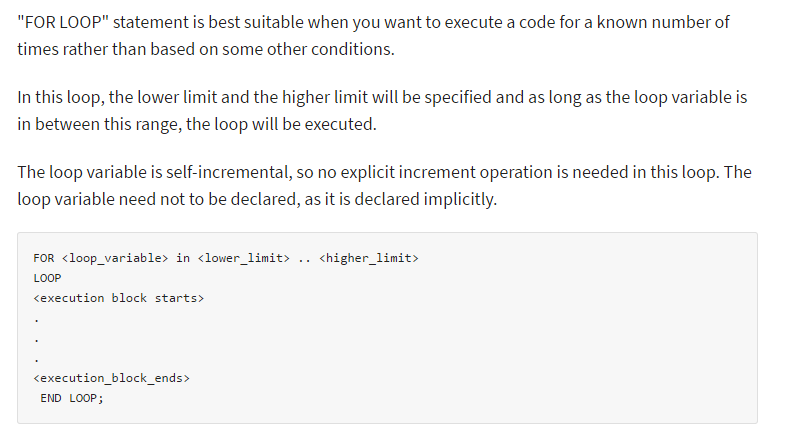
The exit condition should be given inside this execution block so that control exit from the loop.

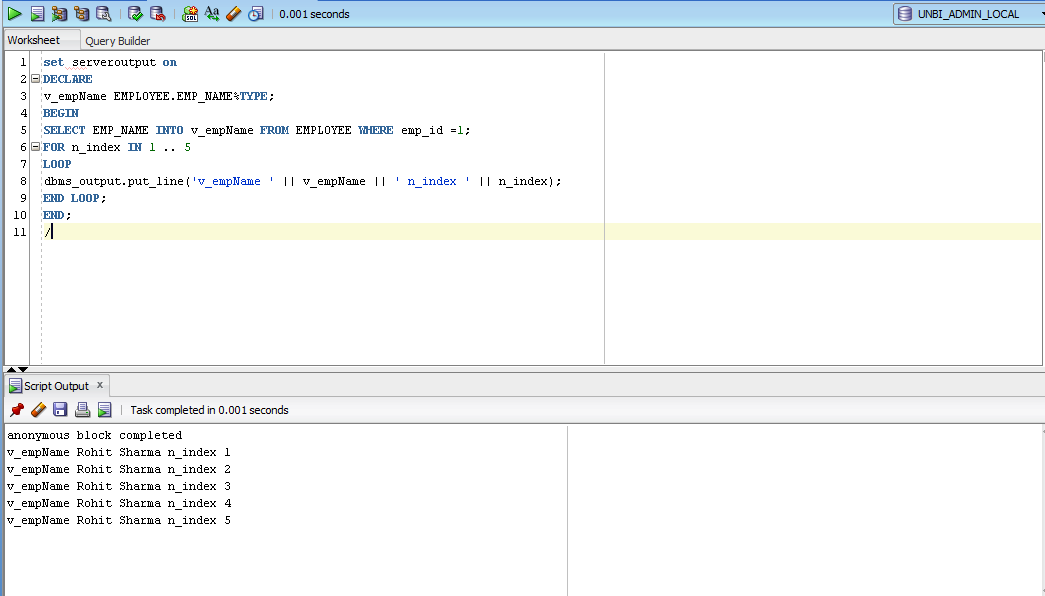
It needs EXIT keyword to be given explicitly in the execution part to exit from the loop.



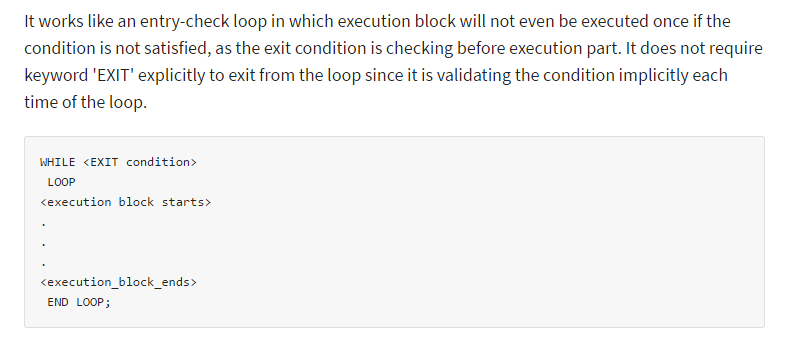


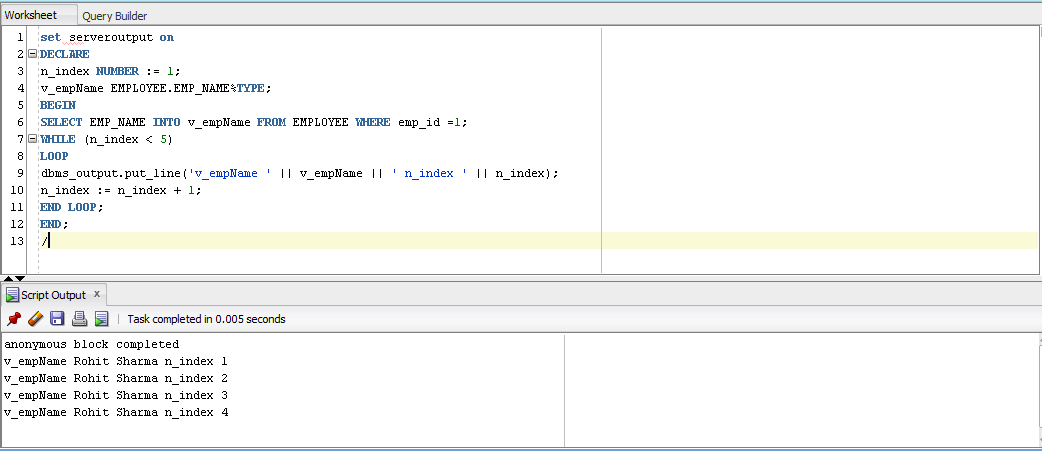
**FOR LOOP**





**WHILE LOOP**





**CURSOR**

A Cursor is a pointer to this context area. Oracle creates context area for processing an SQL statement which contains all information about the statement.

PL/SQL allows the programmer to control the context area through the cursor. A cursor holds the rows returned by the SQL statement. The set of rows the cursor holds is referred as active set. These cursors can also be named so that they can be referred from another place of the code.

The cursor is of two types.

* Implicit Cursor
* Explicit Cursor

**Implicit Cursor**

Whenever any DML operations occur in the database, an implicit cursor is created that holds the rows affected, in that particular operation. These cursors cannot be named and, hence they cannot be controlled or referred from another place of the code. We can refer only to the most recent cursor through the cursor attributes.

**Explicit Cursor**

Below are steps that involved in working with explicit cursors.

**Declaring the cursor**

Declaring the cursor simply means to create one named context area for the 'SELECT' statement that is defined in the declaration part. The name of this context area is same as the cursor name.

**Opening Cursor**

Opening the cursor will instruct the PL/SQL to allocate the memory for this cursor. It will make the cursor ready to fetch the records.

**Fetching Data from the Cursor**

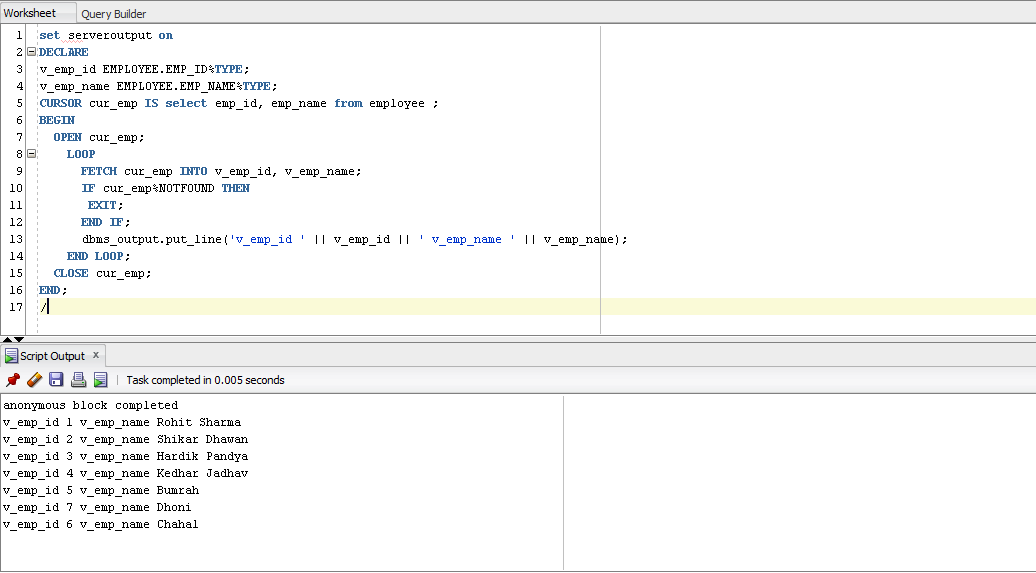
In this process, the 'SELECT' statement is executed and the rows fetched is stored in the allocated memory. These are now called as active sets. Fetching data from the cursor is a record-level activity that means we can access the data in a record-by-record way.

Each fetch statement will fetch one active set and holds the information of that particular record. This statement is same as 'SELECT' statement that fetches the record and assigns to the variable in the 'INTO' clause, but it will not throw any exceptions.

**Closing the Cursor**

Once all the record is fetched now, we need to close the cursor so that the memory allocated to this context area will be released.

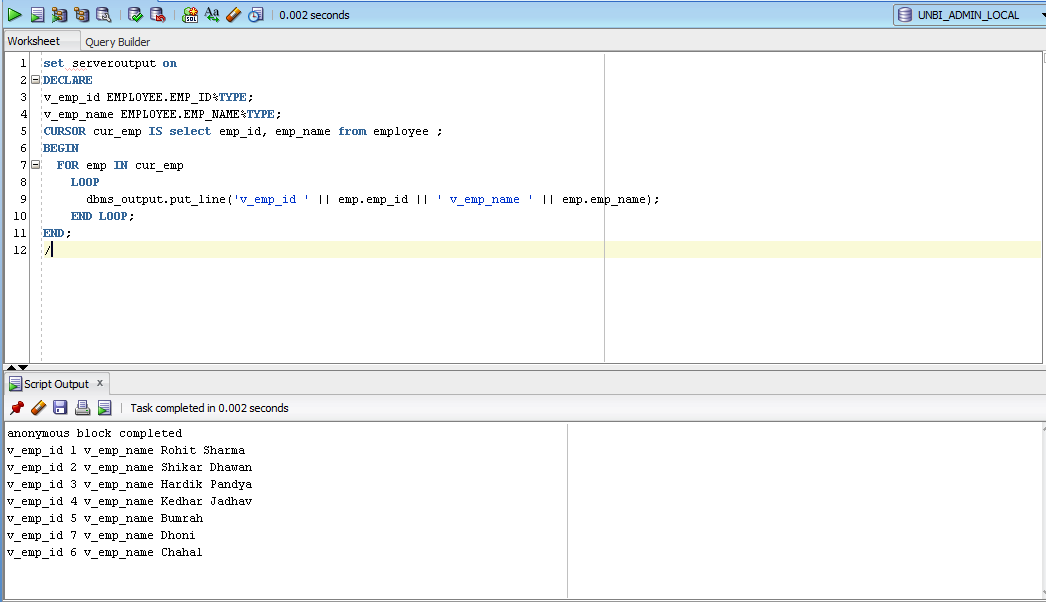




**CURSOR FOR LOOP**

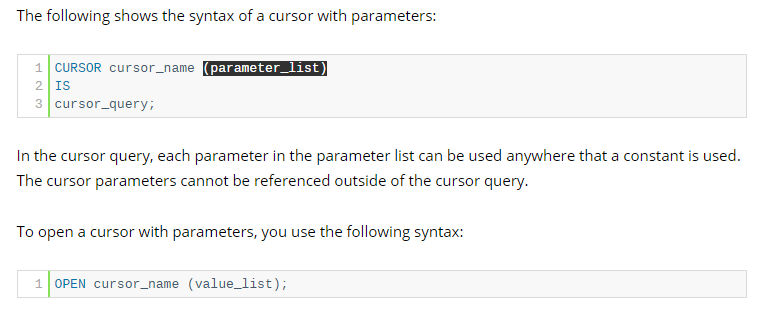
"FOR LOOP" statement can be used for working with cursors. We can give the cursor name instead of range limit in the FOR loop statement so that the loop will work from the first record of the cursor to the last record of the cursor. The cursor variable, opening of cursor, fetching and closing of the cursor will be done implicitly by the FOR loop.

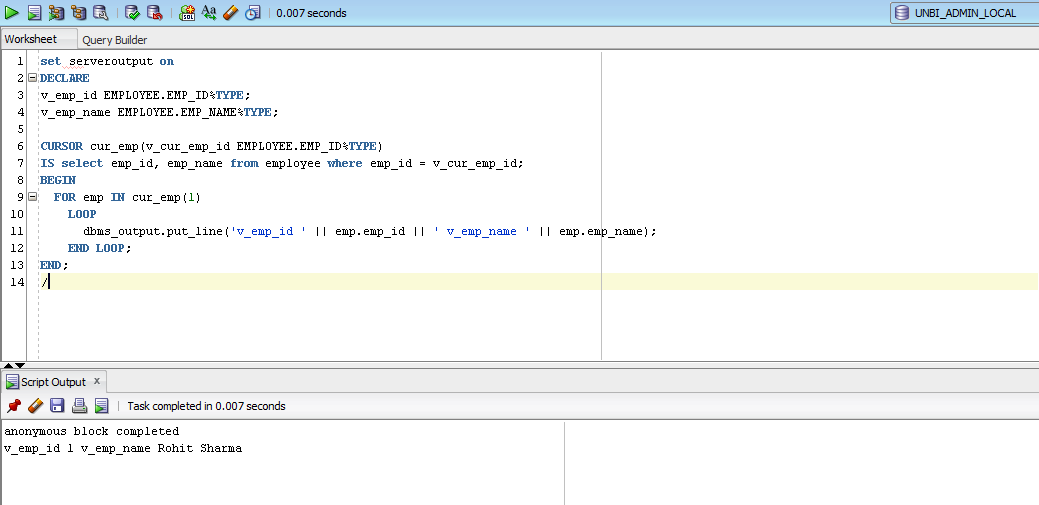




**CURSOR WITH PARAMETERS**

Syntax:





**RECORDS**

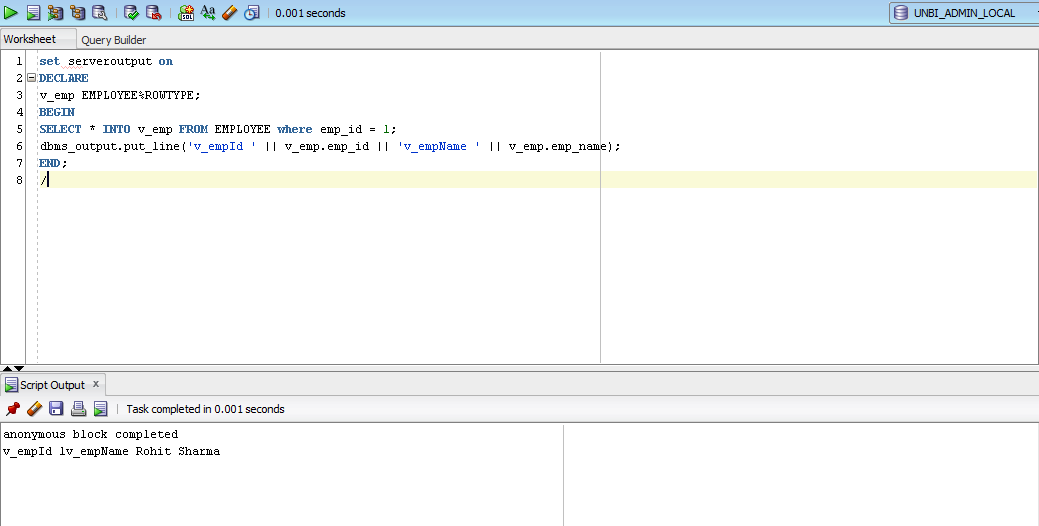
A record is a data structure that can hold data items of different kinds. Records consist of different fields, similar to a row of a database table.

PL/SQL can handle the following types of records −

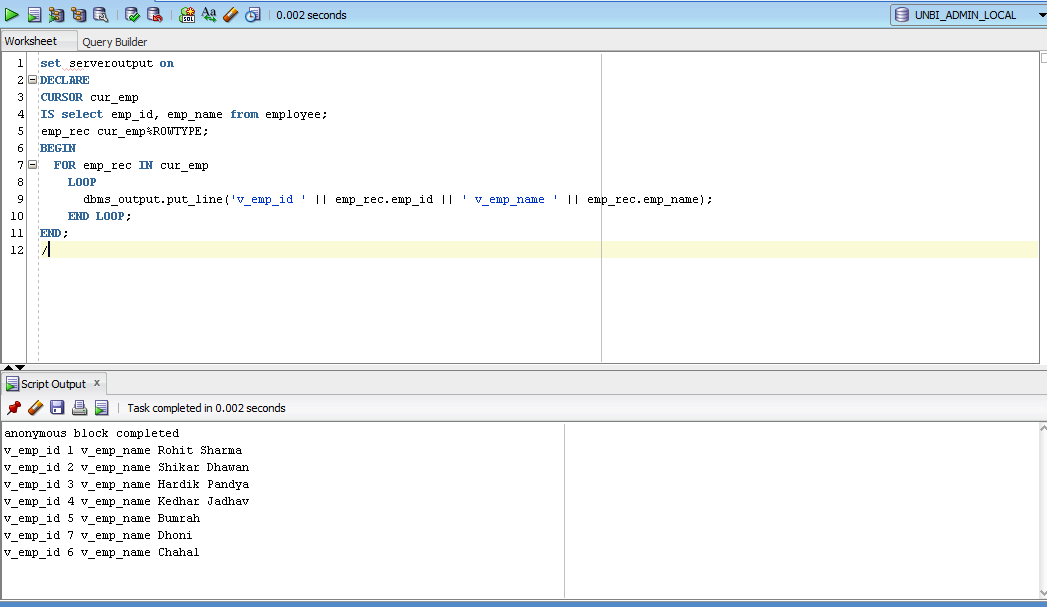
* Table-based
* Cursor-based records
* User-defined records

**Table-Based Records**





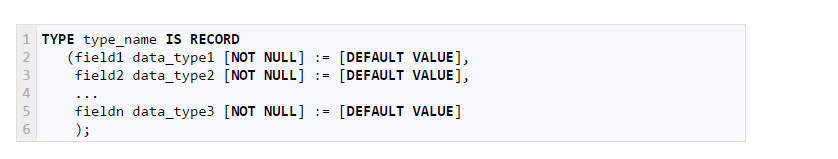
**Cursor based**

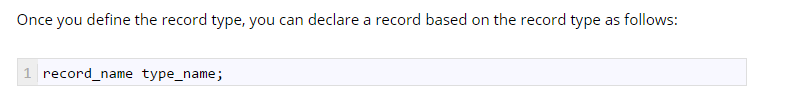


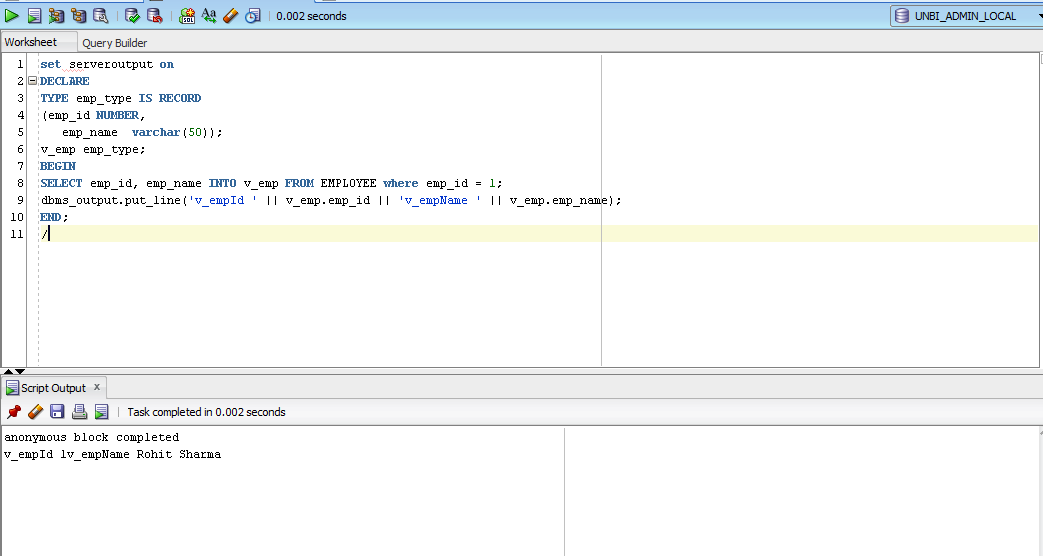
**User defined**

To declare programmer-defined record, first you have to define a record type by using TYPE statement with the fields of record explicitly. Then, you can declare a record based on record type that you’ve defined.

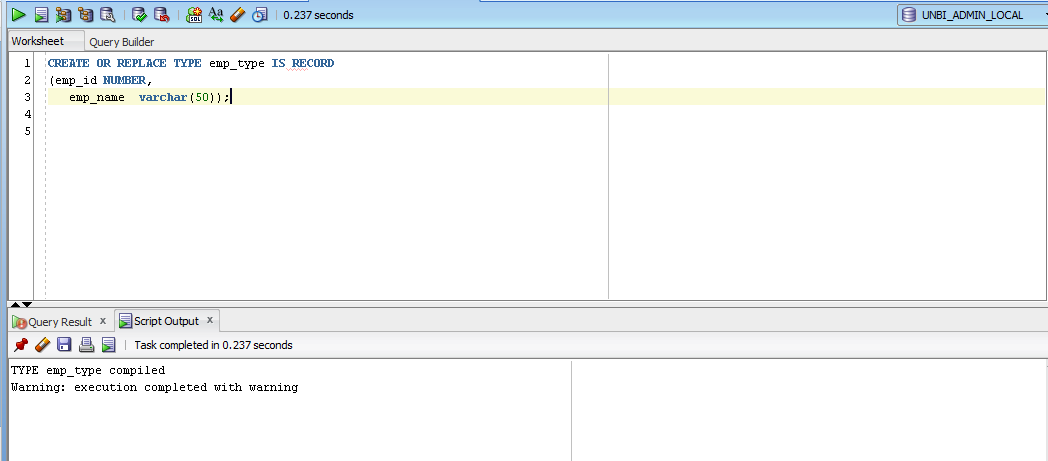
Syntax





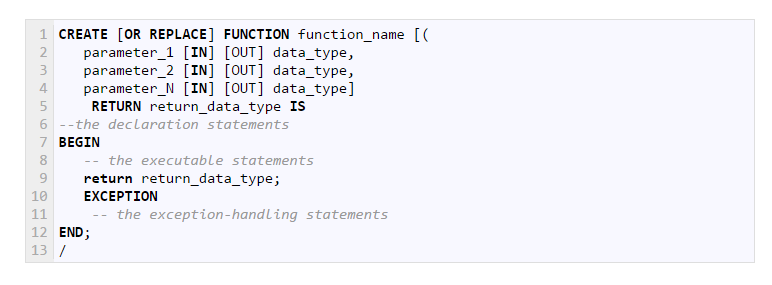


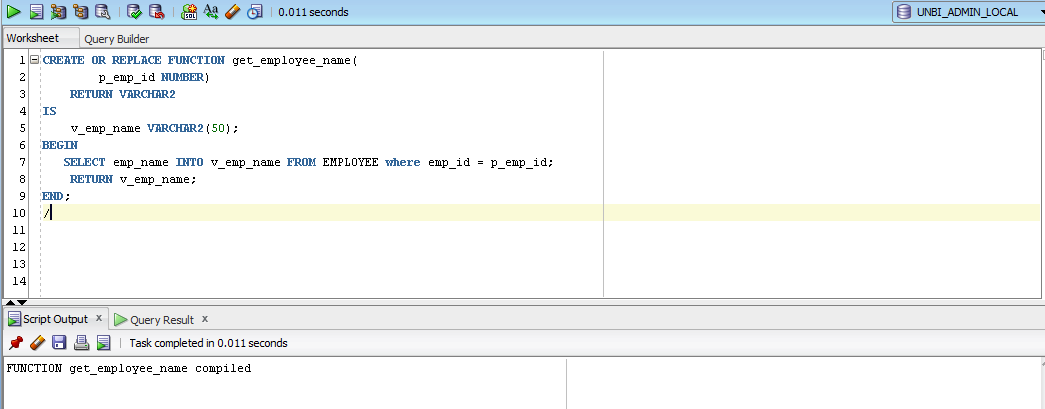
**Permanent type**



**FUNCTIONS**

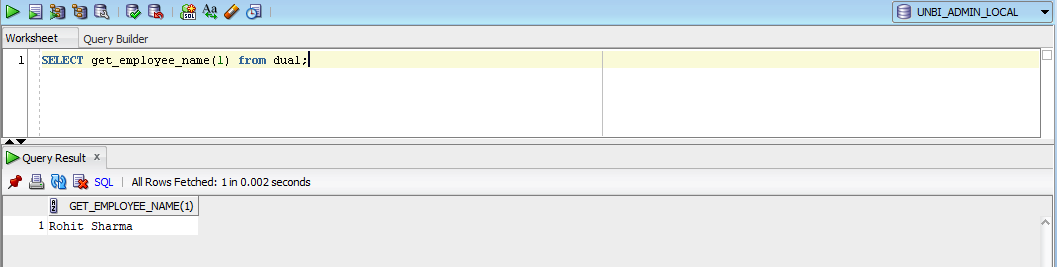
PL/SQL function is a named block that returns a value.



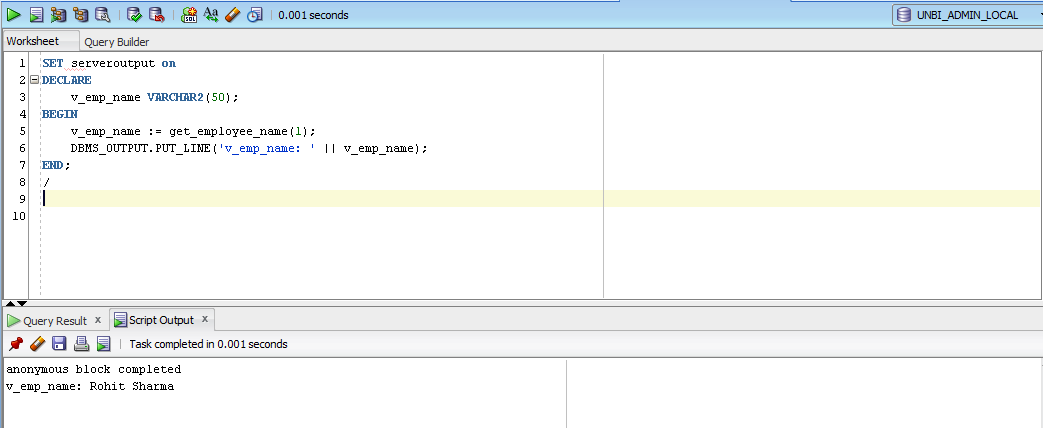


**Calling a PL/SQL function**

SQL statement

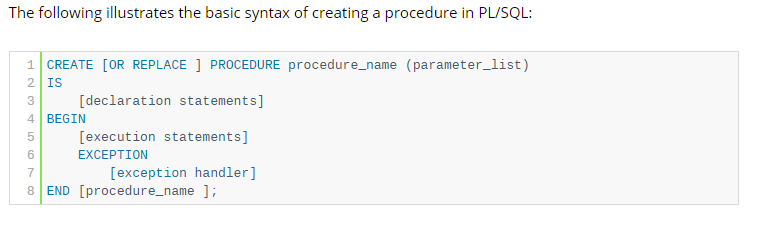


PL/SQL Block

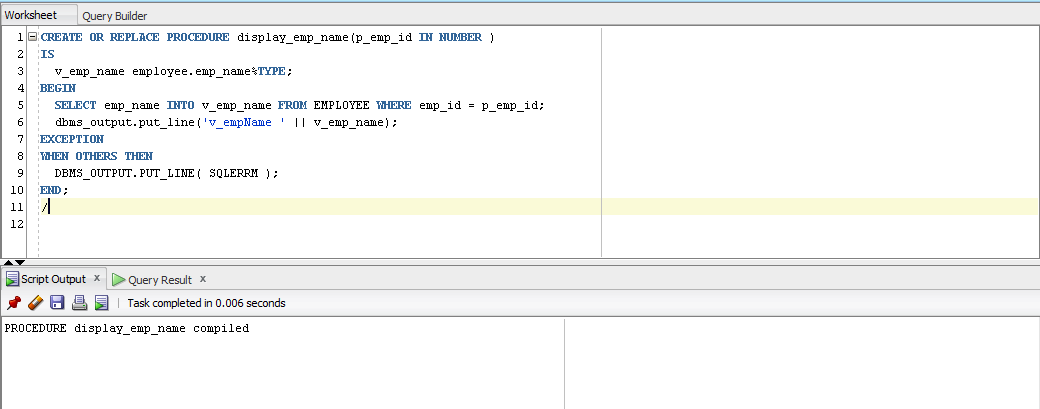


**STORED PROCEDURES**

A PL/SQL procedure is a reusable unit that encapsulates a specific business logic of the application. Technically speaking, a PL/SQL procedure is a named block stored as a schema object in the Oracle Database.

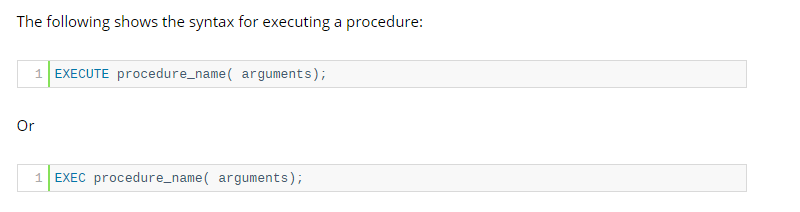


**WITH ONLY IN PARAMETER**



**EXECUTE**

**Using keyword**





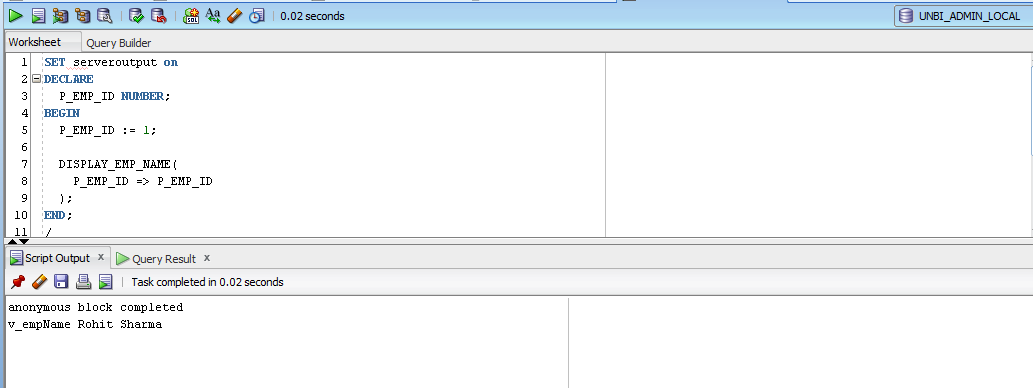
**PL/SQL BLOCK**

**POSITIONAL NOTATION**

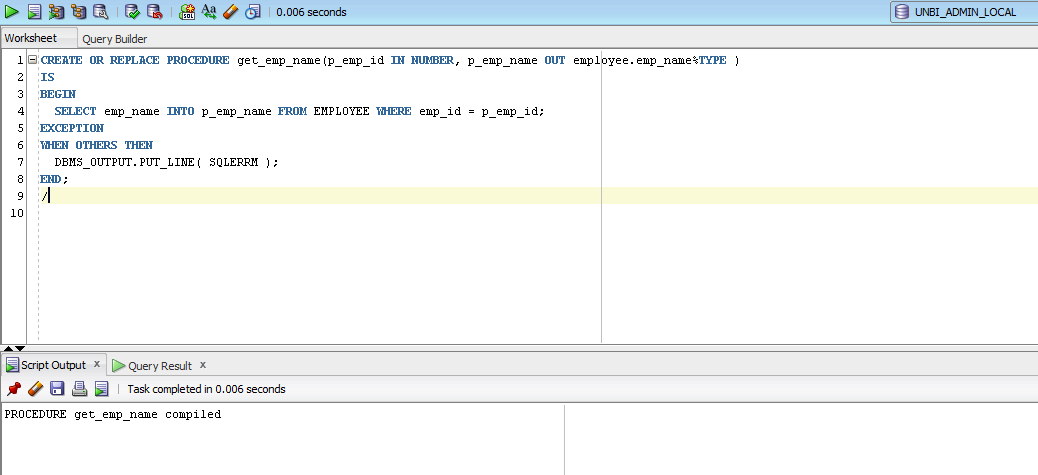
In positional notation, the first actual parameter is substituted for the first formal parameter; the second actual parameter is substituted for the second formal parameter, and so on

**NAMED NOTATION**

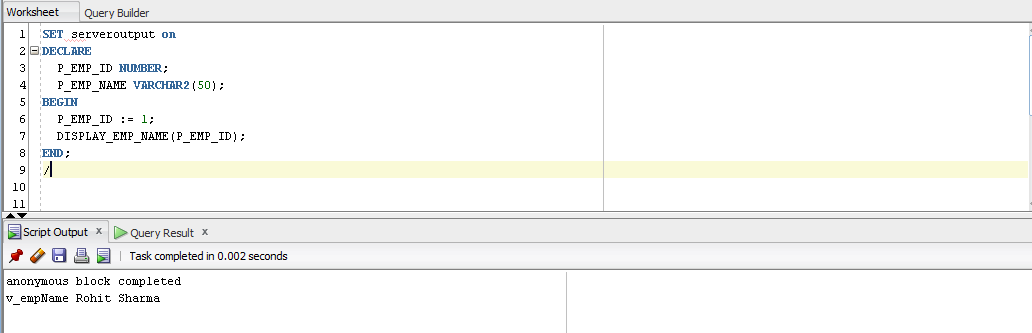
In named notation, the actual parameter is associated with the formal parameter using the **arrow symbol ( => )**



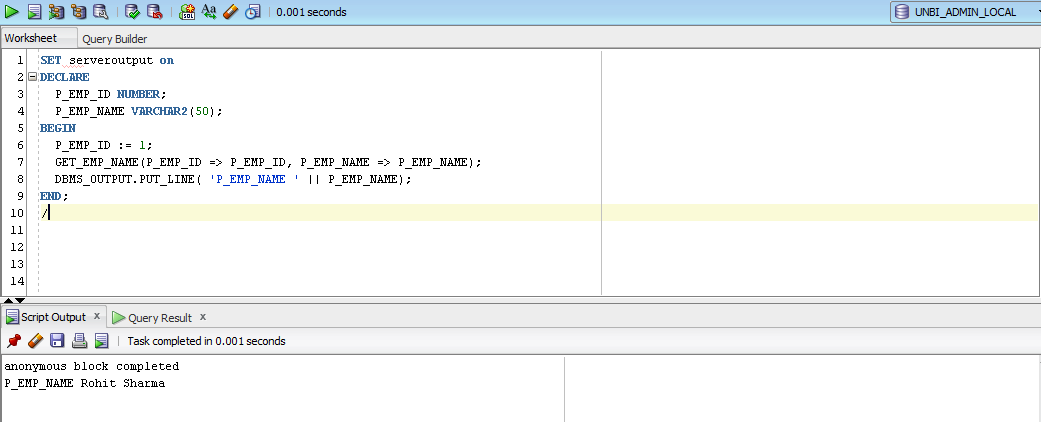
**OUT PARAMETER**



**POSITION NOTATION**



**NAMED NOTATION**



**PACKAGES**

In PL/SQL, a package is a schema object that contains definitions for a group of related functionalities. A package includes variables, constants, cursors, exceptions, procedures, functions, and subprograms. It is compiled and stored in the Oracle Database.

Typically, a package has a specification and a body. A package specification is mandatory while the package body can be required or optional, depending on the package specification.

**Package specification**

The package specification declares the public objects that are accessible from outside the package.

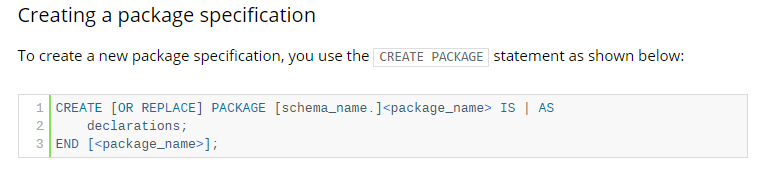
If a package specification whose public objects include cursors and subprograms, then it must have a body which defines queries for the cursors and code for the subprograms.

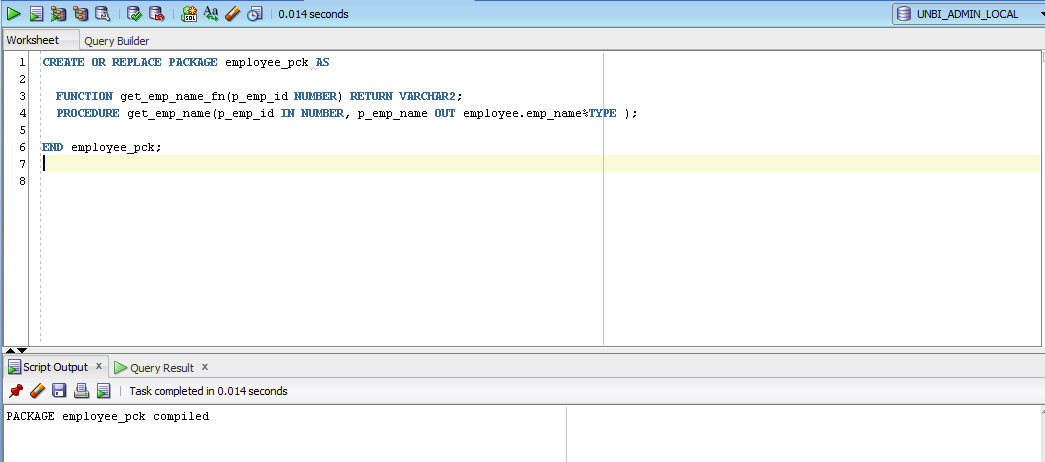
**Package body**

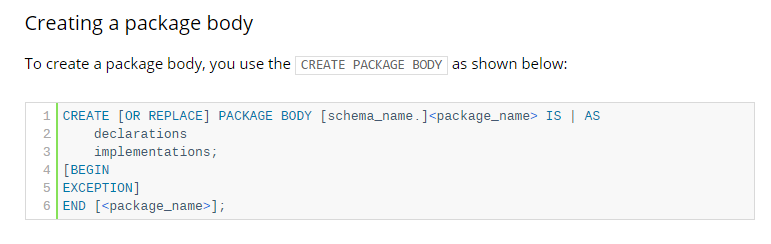
A package body contains the implementation of the cursors or subprograms declared in the package specification. In the package body, you can declare or define private variables, cursors, etc., used only by package body itself.

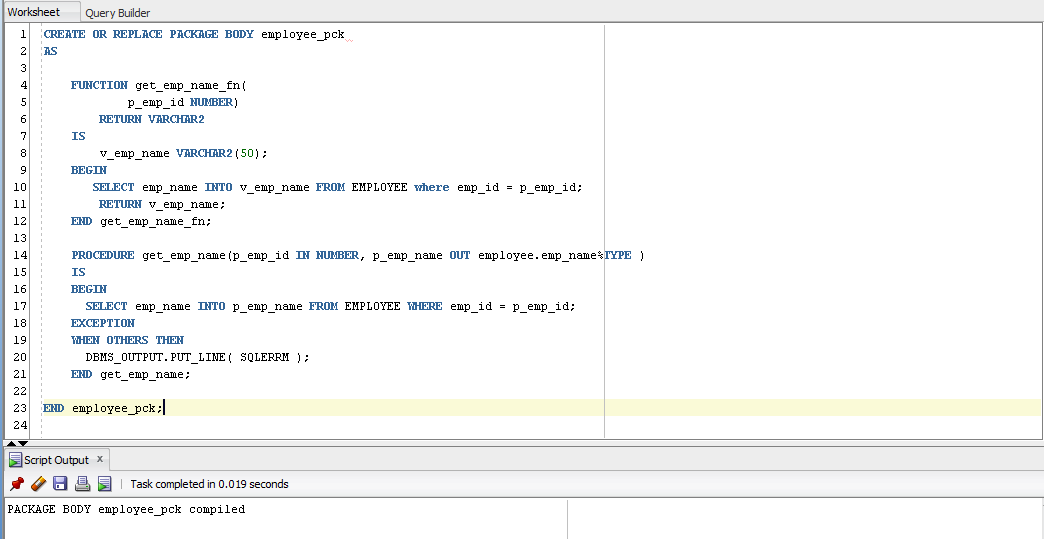
A package body can have an initialization part whose statements initialize variables or perform other one-time setups for the whole package.

A package body can also have an exception-handling part used to handle exceptions.



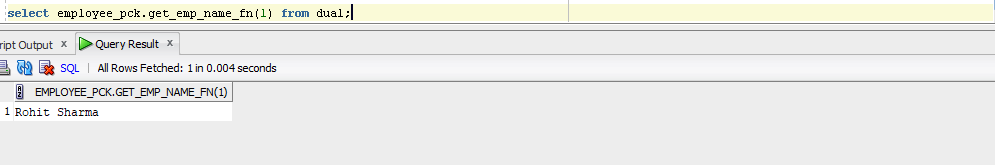






**CALLING PACKAGE FUNCTIONS/PROCEDURES**

SAME AS NORMAL FUNCTION/PROCUDURE – prefix package name

****

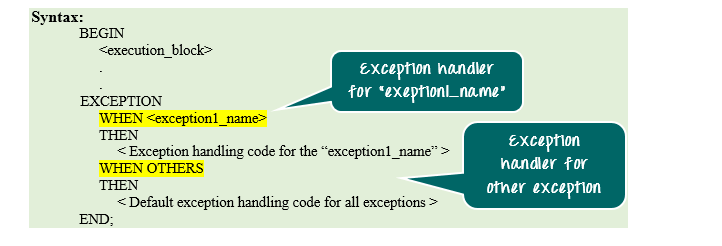
**EXCEPTIONS**

An exception occurs when the PL/SQL engine encounters an instruction which it cannot execute due to an error that occurs at run-time. These errors will not be captured at the time of compilation and hence these needed to handle only at the run-time.

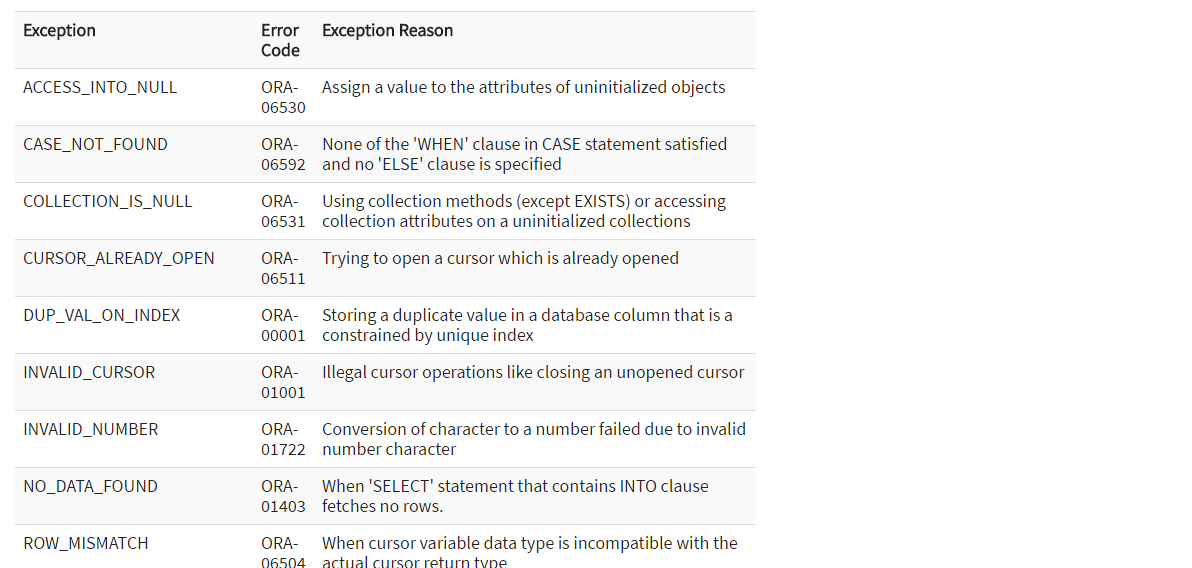
For example, if PL/SQL engine receives an instruction to divide any number by '0', then the PL/SQL engine will throw it as an exception. The exception is only raised at the run-time by the PL/SQL engine.

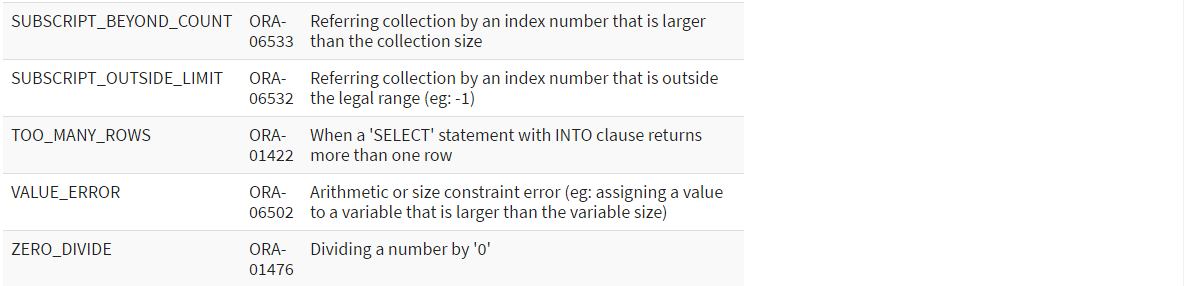
Exceptions will stop the program from executing further, so to avoid such condition, they need to be captured and handled separately. This process is called as Exception-Handling, in which the programmer handles the exception that can occur at the run time.

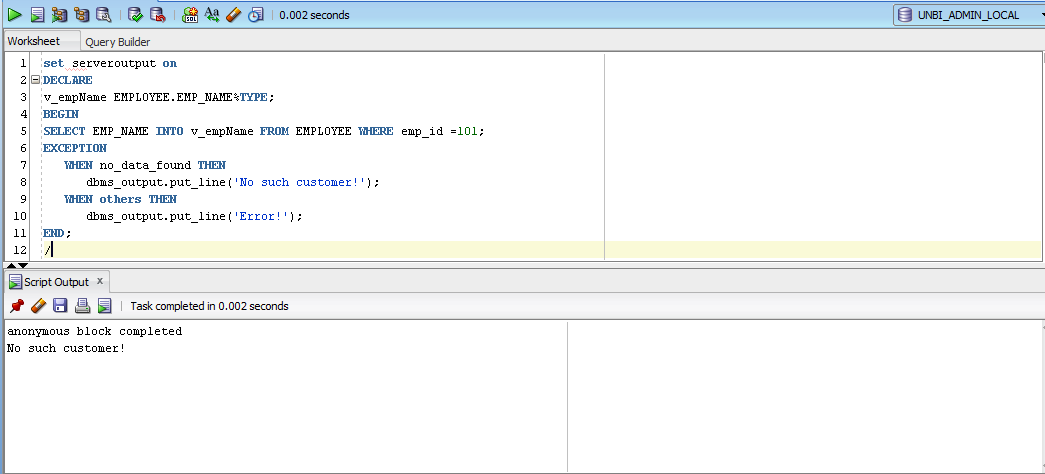
Exceptions are handled at the block, level, i.e., once if any exception occurs in any block then the control will come out of execution part of that block. The exception will then be handled at the exception handling part of that block. After handling the exception, it is not possible to resend control back to the execution section of that block.



**PREDEFINED EXCEPTIONS**



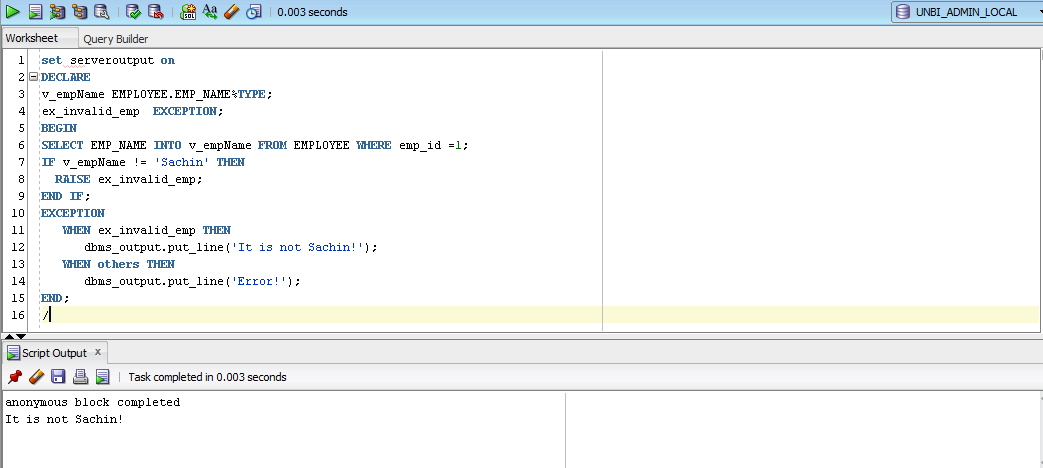




**USER DEFINED EXCEPTIONS**

In Oracle, other than the above-predefined exceptions, the programmer can create their own exception and handle them. They can be created at a subprogram level in the declaration part. These exceptions are visible only in that subprogram. The exception that is defined in the package specification is public exception, and it is visible wherever the package is accessible



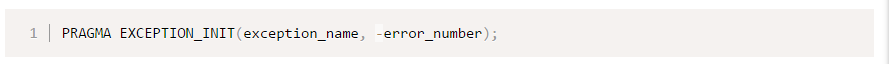


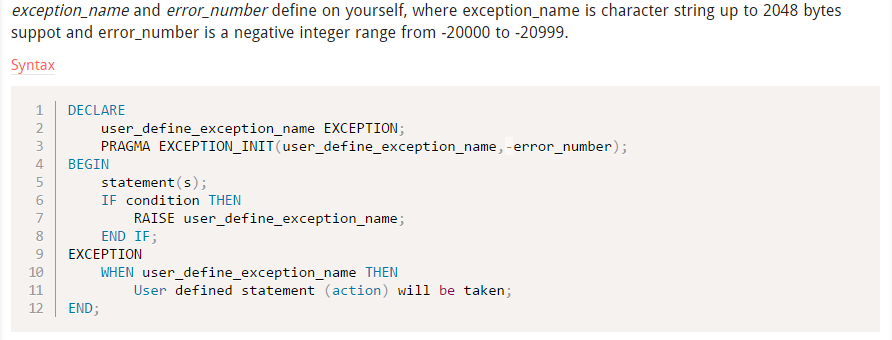
**PRAGMA EXCEPTION\_INIT**

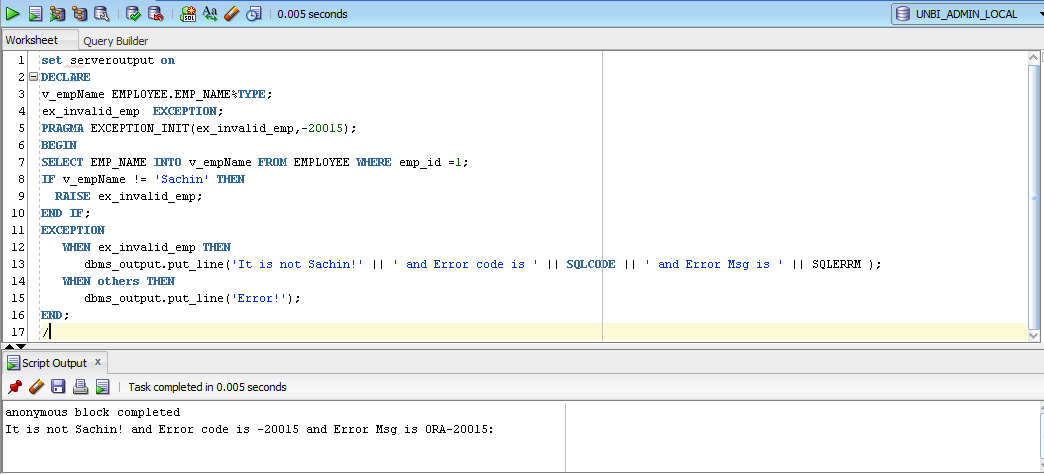
Pragma is a keyword directive to execute proceed at compile time. pragma EXCEPTION\_INIT function take this two argument,

exception\_name

error\_number



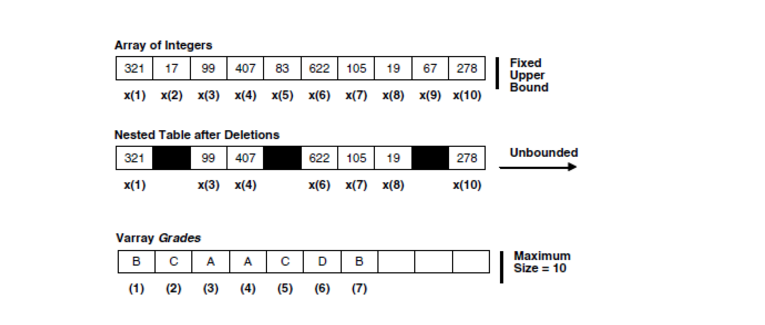


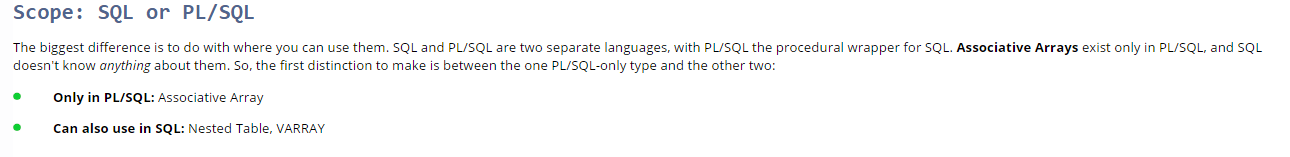


**COLLECTIONS**

A collection is an ordered group of elements, all of the same type. It is a general concept that encompasses lists, arrays, and other datatypes used in classic programming algorithms. Each element is addressed by a unique subscript.

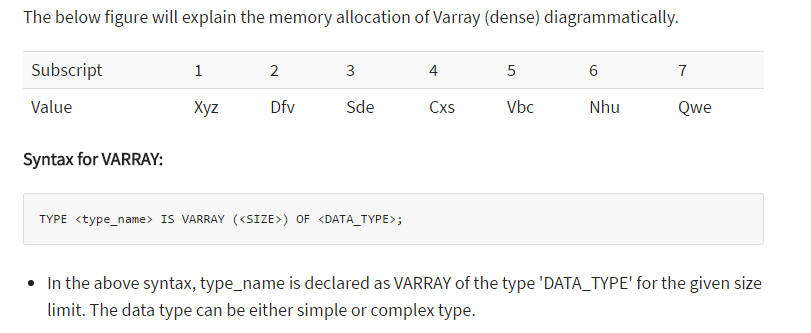
* Index-by tables or Associative array
* Nested table
* Variable-size array or Varray

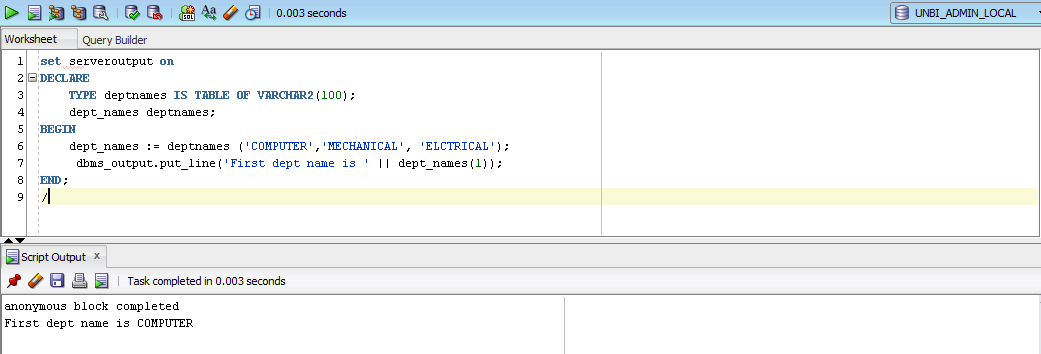




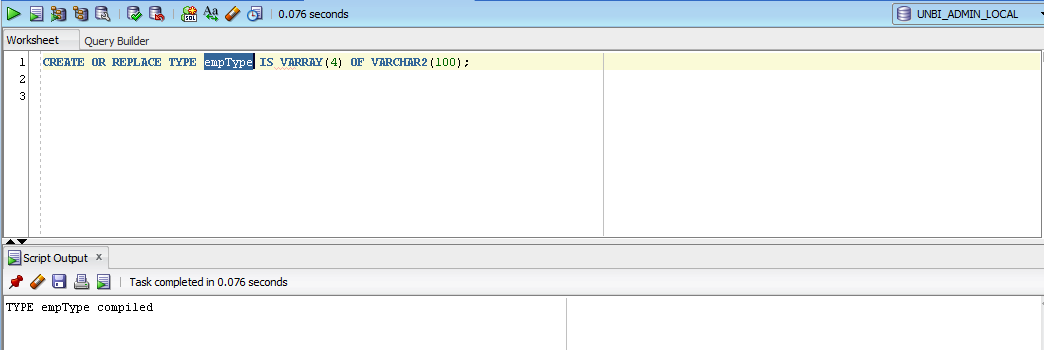
**VARRAYS**

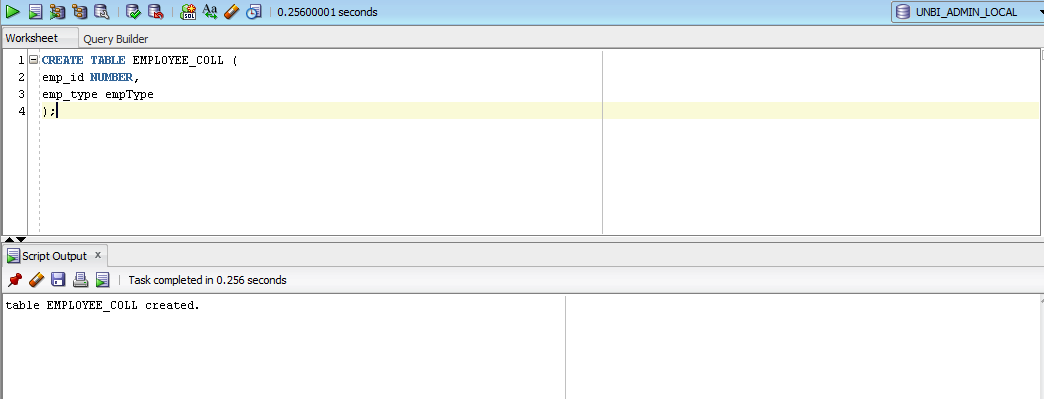
* Upper limit size is fixed
* Populated sequentially starting with the subscript '1'
* This collection type is always dense, i.e. we cannot delete any array elements. Varray can be deleted as a whole, or it can be trimmed from the end.
* Since it always is dense in nature, it has very less flexibility.
* It is more appropriate to use when the array size is known and to perform similar activities on all the array elements.
* The subscript and sequence always remain stable, i.e. the subscript and count of the collection is always same.
* They need to be initialized before using them in programs. Any operation (except EXISTS operation) on an uninitialized collection will throw an error.
* It can be created as a database object, which is visible throughout the database or inside the subprogram, which can be used only in that subprogram.

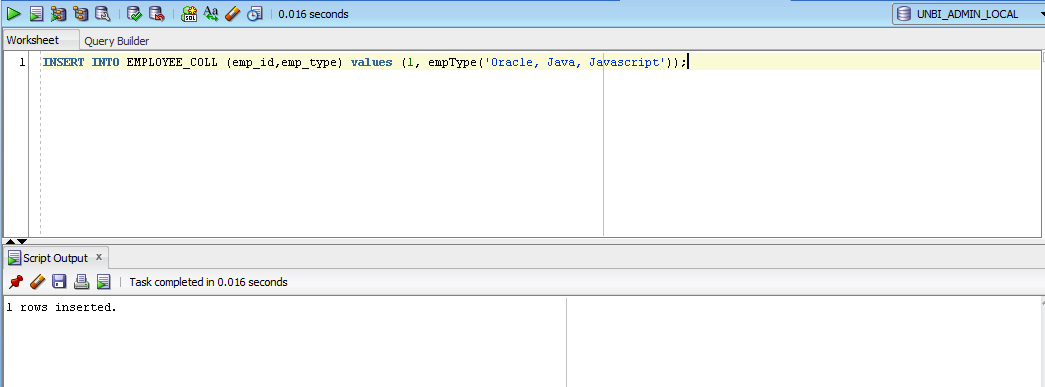


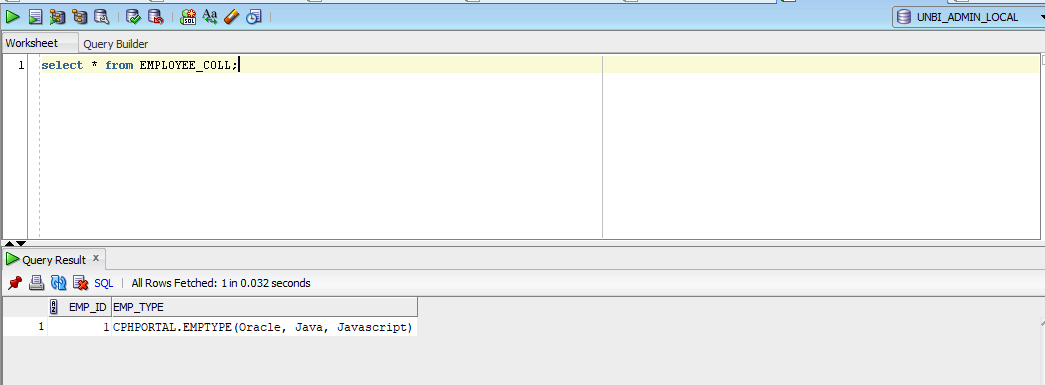


**VARRAY AS DATABASE OBJECTS**



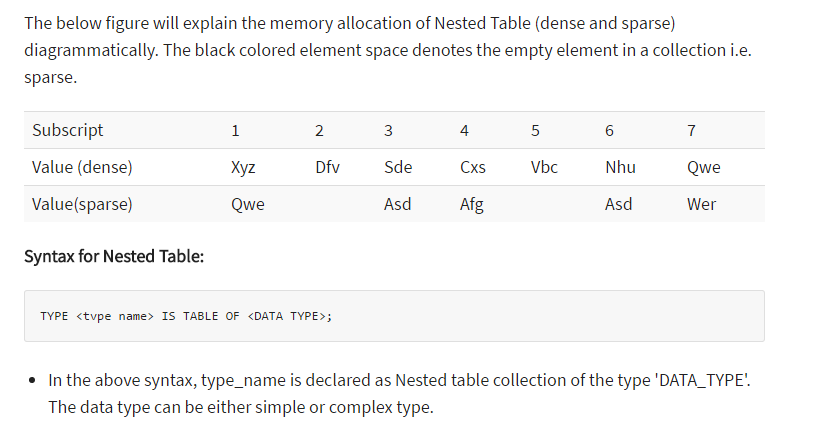


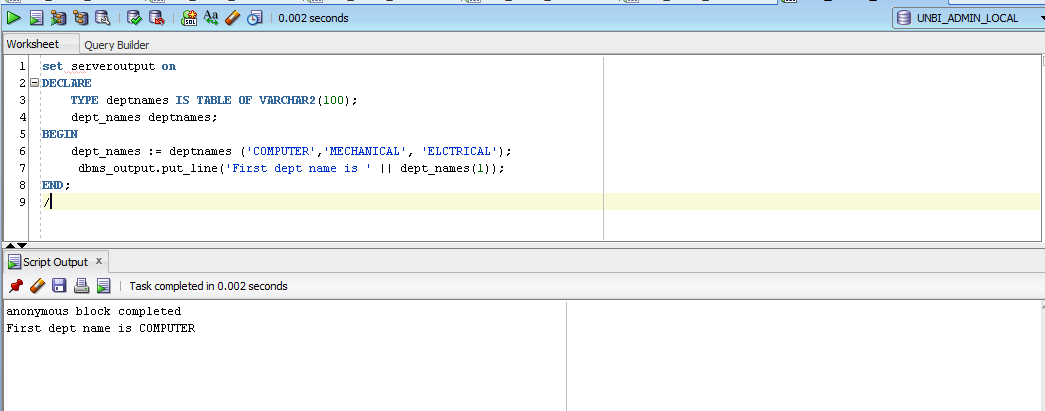




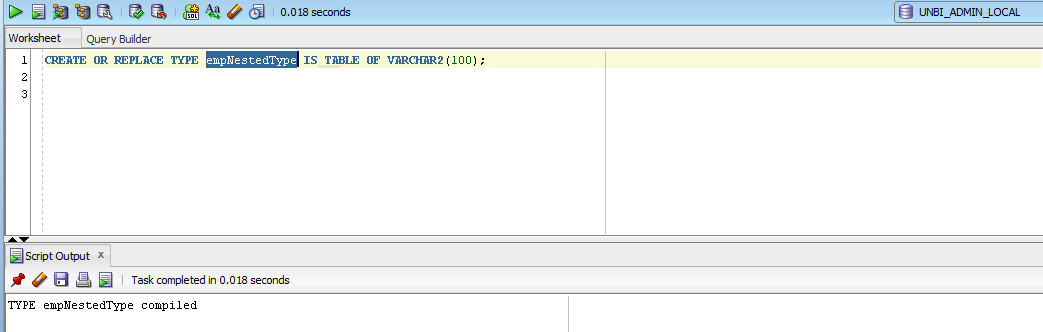
**NESTED TABLES**

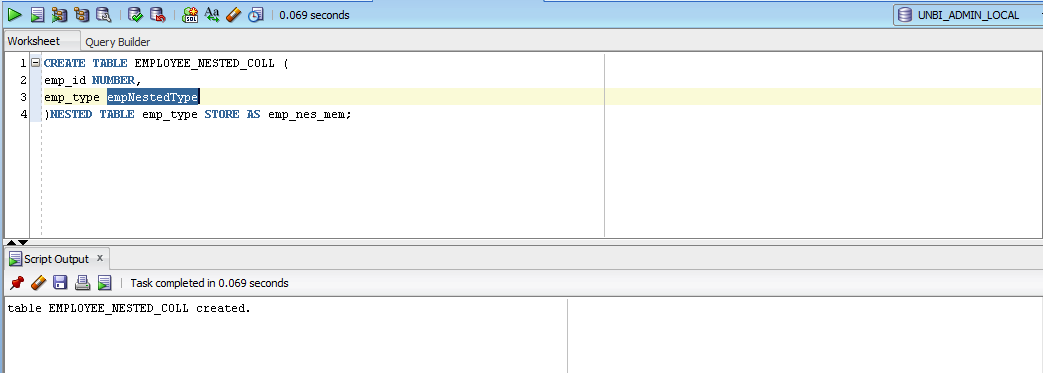
* The Nested table has no upper size limit.
* Since the upper size limit is not fixed, the collection, memory needs to be extended each time before we use it. We can extend the collection using 'EXTEND' keyword.
* Populated sequentially starting with the subscript '1'.
* This collection type can be of both dense and sparse, i.e. we can create the collection as a dense, and we can also delete the individual array element randomly, which make it as sparse.
* It gives more flexibility regarding deleting the array element.
* It is stored in the system generated database table and can be used in the select query to fetch the values.
* The subscript and sequence are not stable, i.e. the subscript and the count of the array element can vary.
* They need to be initialized before using them in programs. Any operation (except EXISTS operation) on the uninitialized collection will throw an error.
* It can be created as a database object, which is visible throughout the database or inside the subprogram, which can be used only in that subprogram.

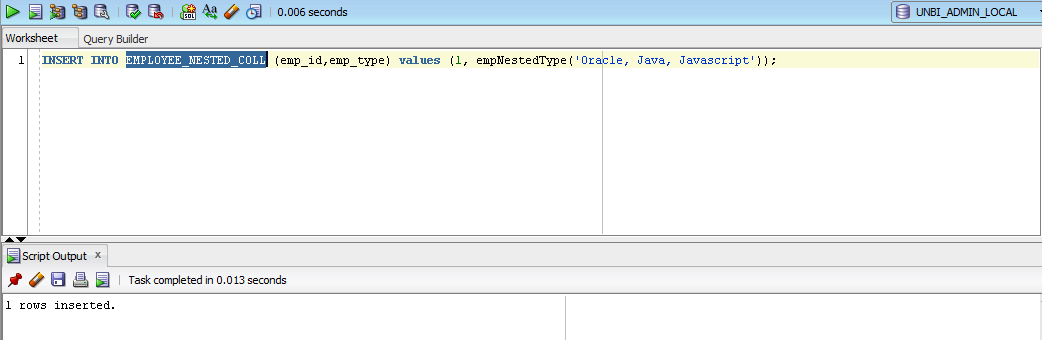


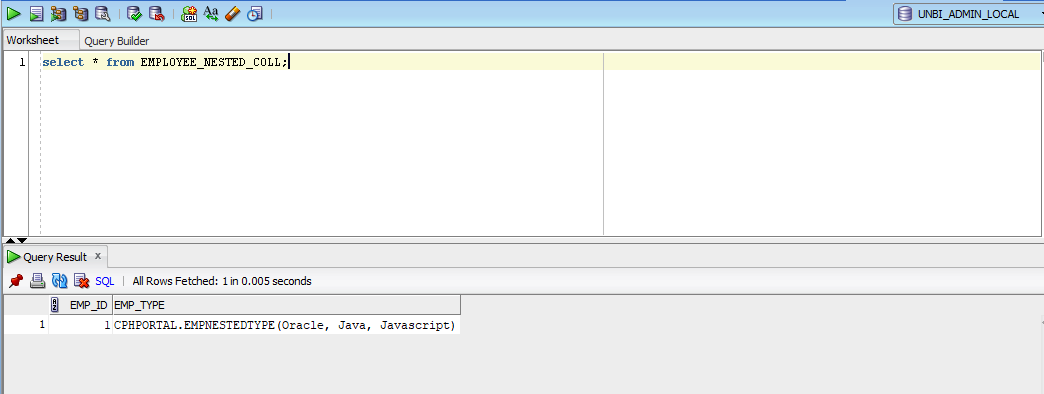


**NESTED TABLE AS DATABASE OBJECT**







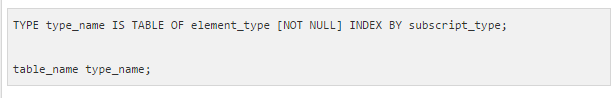


**ASSOCIATIVE ARRAY (INDEX BY TABLE)**

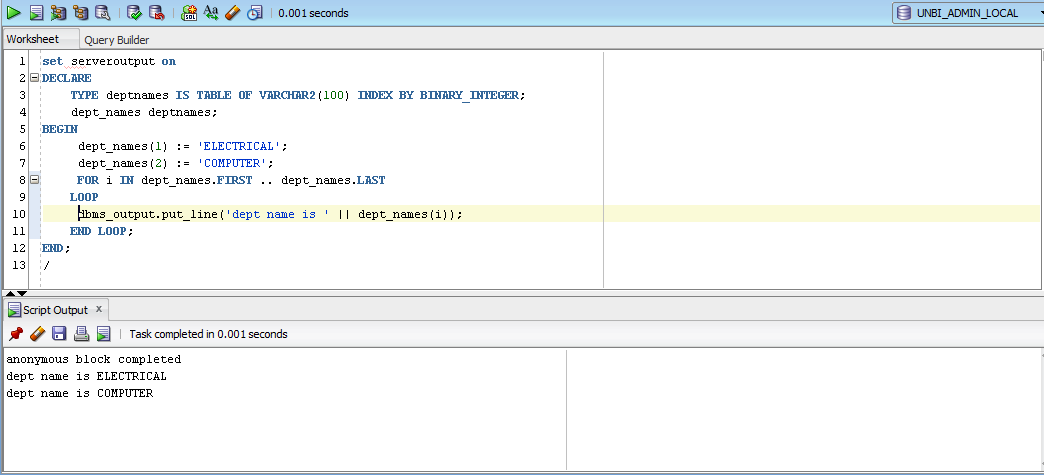
An index-by table (also called an associative array) is a set of key-value pairs. Each key is unique and is used to locate the corresponding value. The key can be either an integer or a string.

An index-by table is created using the following syntax. Here, we are creating an index-by table named table\_name, the keys of which will be of the subscript\_type and associated values will be of the element\_type

* These collections are not stored sequentially.
* They are always sparse in nature.
* The array size is not fixed.
* They cannot be stored in the database column. They shall be created and used in any program in that particular session.
* They give more flexibility in terms of maintaining subscript.
* The subscripts can be of negative subscript sequence also.
* They are more appropriate to use for relatively smaller collective values in which the collection can be initialized and used within the same subprograms.
* They need not be initialized before start using them.
* It cannot be created as a database object. It can only be created inside the subprogram, which can be used only in that subprogram.
* BULK COLLECT cannot be used in this collection type as the subscript should be given explicitly for each record in the collection.

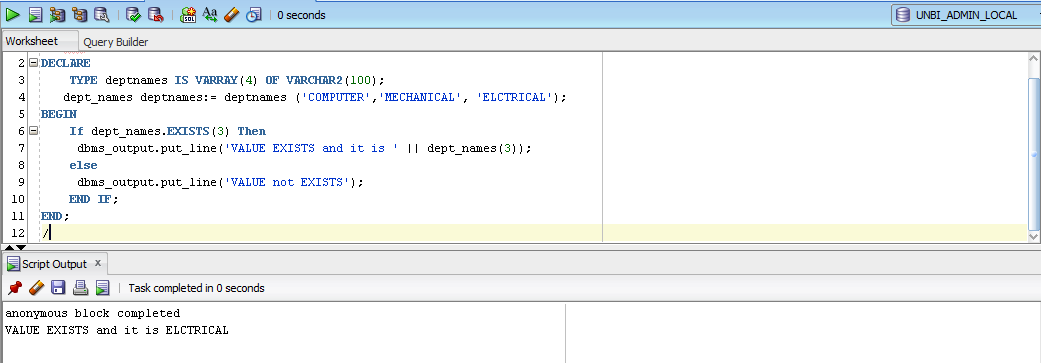


ASSOCIATIVE ARRAYS can be indexed by BINARY\_INTEGER or a string type (VARCHAR2)



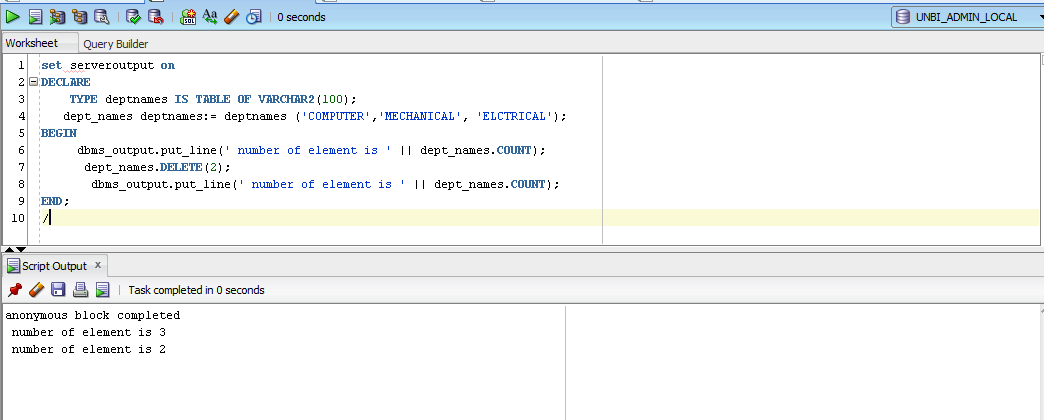
**EXISTS**

EXISTS(n) returns TRUE if the nth element in a collection exists. Otherwise, EXISTS(n) returns FALSE.



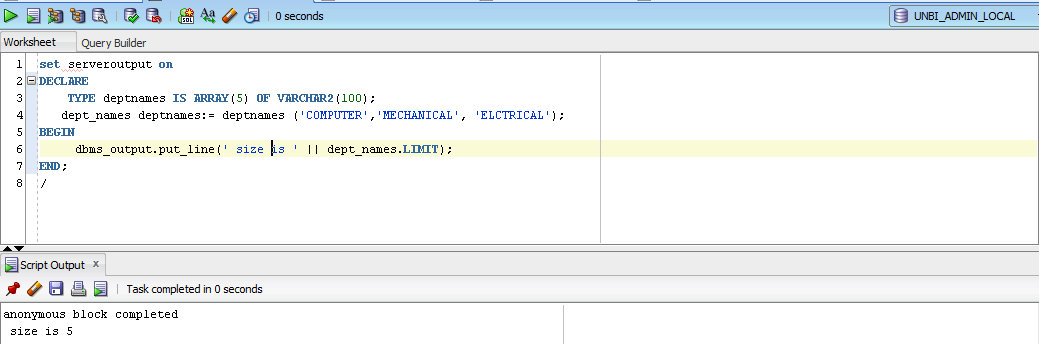
**COUNT**

COUNT returns the number of elements that a collection currently contains



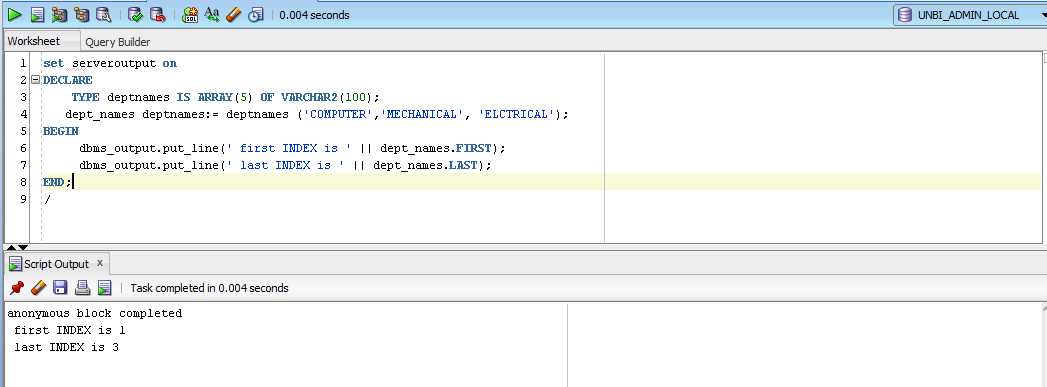
**LIMIT**

For nested tables and associative arrays, which have no declared size, LIMIT returns NULL. For varrays, LIMIT returns the maximum number of elements that a varray can contain



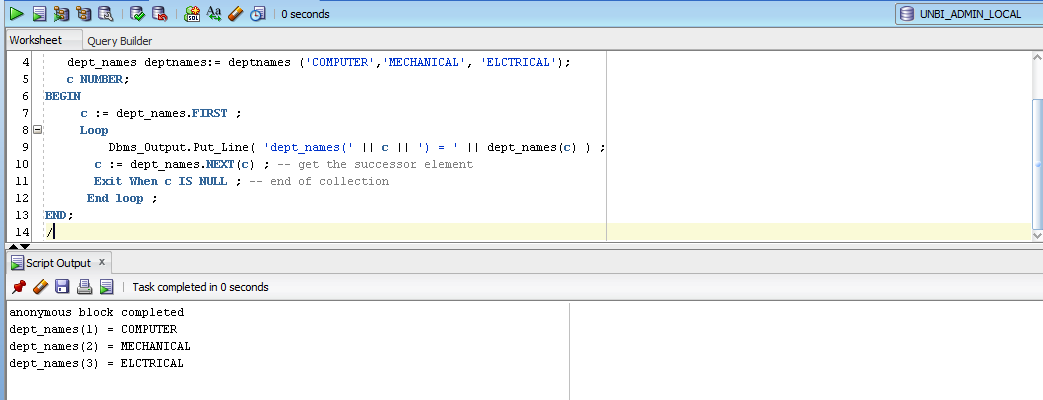
**FIRST AND LAST**

FIRST and LAST return the first and last (smallest and largest) index numbers in a collection that uses integer subscripts.



**PRIOR AND NEXT**

PRIOR(n) returns the index number that precedes index n in a collection. NEXT(n)returns the index number that succeeds index n. If n has no predecessor, PRIOR(n)returns NULL. If n has no successor, NEXT(n)returns NULL.



**EXTEND**

To increase the size of a nested table or varray, use EXTEND.

This procedure has three forms:

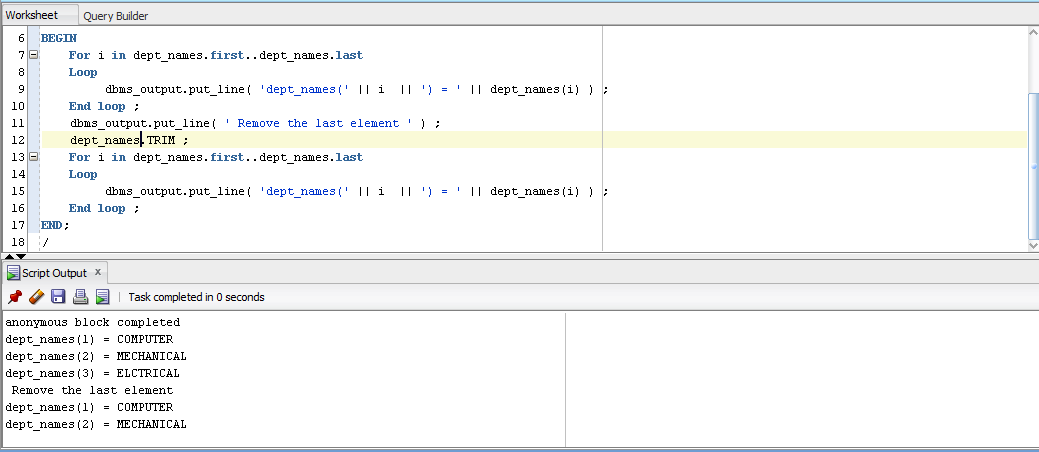
* EXTEND appends one null element to a collection.
* EXTEND(n) appends n null elements to a collection.
* EXTEND(n,i) appends n copies of the ith element to a collection.

**TRIM**

This procedure has two forms:

* TRIM removes one element from the end of a collection.
* TRIM(n) removes n elements from the end of a collection.

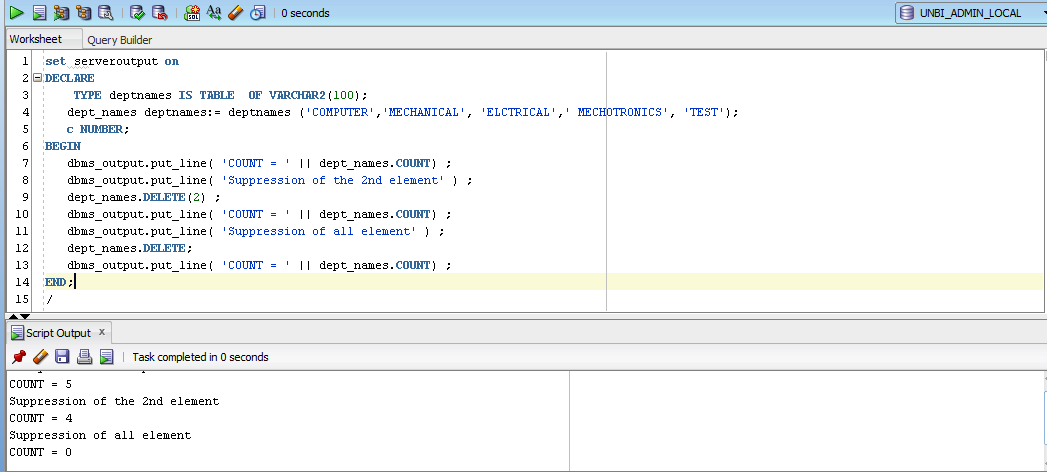
If you want to remove all elements, use DELETE without parameters.



**DELETE**

This procedure has various forms:

* DELETE with no parameters removes all elements from a collection, setting COUNT to 0.
* DELETE(n) removes the nth element from an associative array with a numeric key or a nested table. If the associative array has a string key, the element corresponding to the key value is deleted. If n is null, DELETE(n) does nothing.
* DELETE(m,n) removes all elements in the range m..n from an associative array or nested table. If m is larger than n or if m or n is null, DELETE(m,n)does

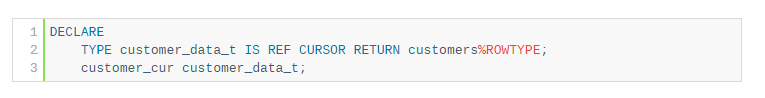


**REF CURSOR**

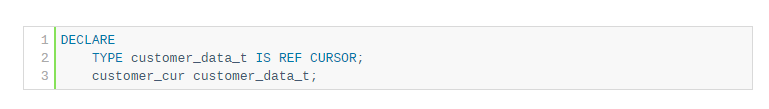
The most important benefit of a cursor variable is that it enables passing the result of a query between PL/SQL programs. Without a cursor variable, you have to fetch all data from a cursor, store it in a variable e.g., a collection, and pass this variable as an argument. With a cursor variable, you simply pass the reference to that cursor.

To declare a cursor variable, you use the REF CURSOR is the data type. PL/SQL has two forms of REF CURSORs: strong and weak.

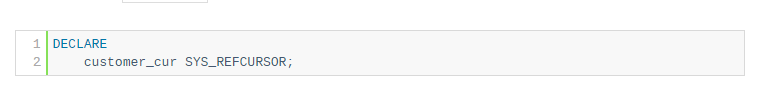
**STRONG REF CURSOR**

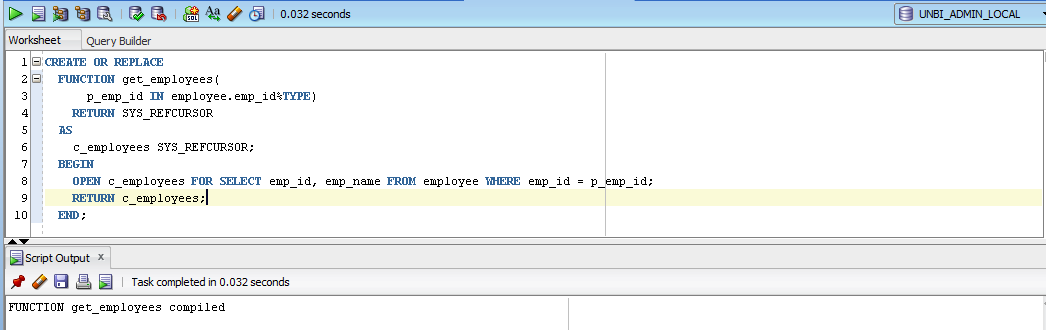


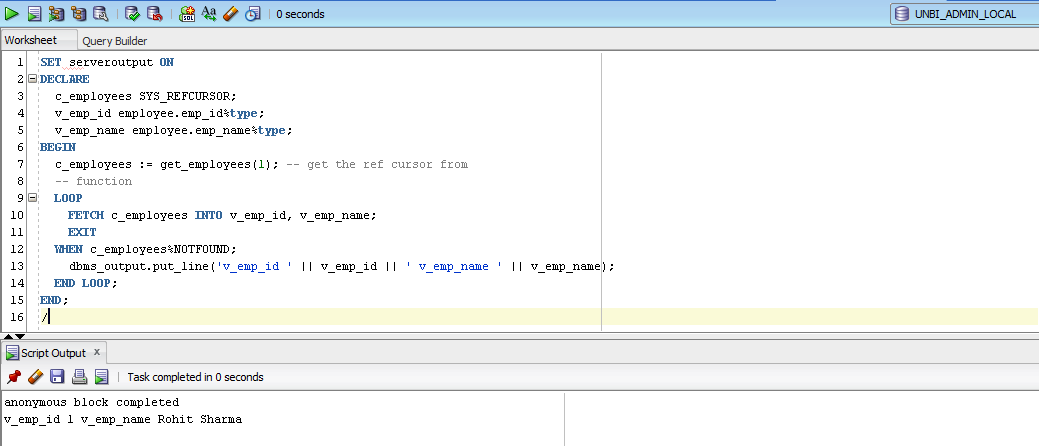
**WEAK REF CURSOR**



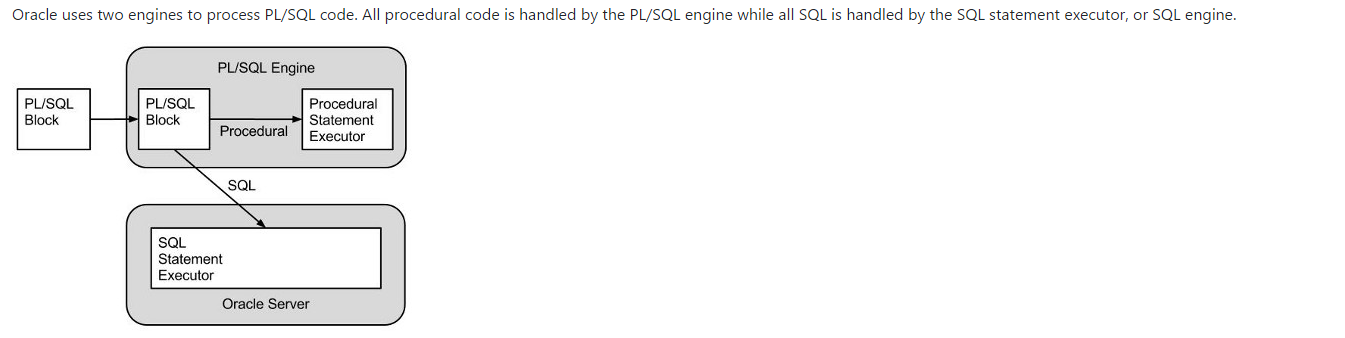
**SYSREF CURSOR**







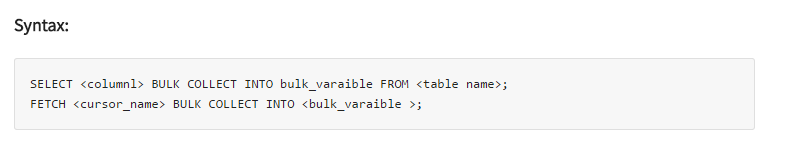
**BULK COLLECT**

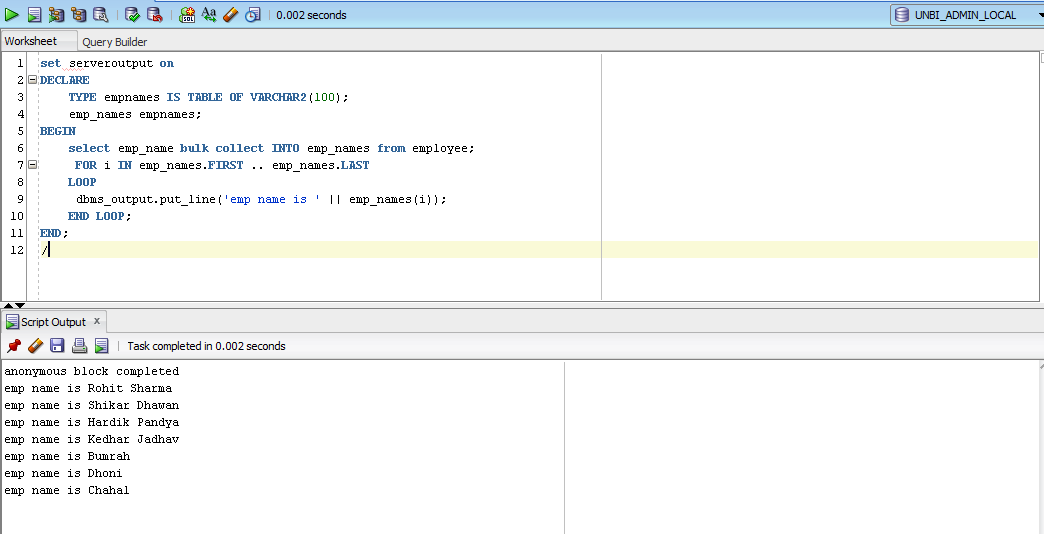


There is an overhead associated with each context switch between the two engines. If PL/SQL code loops through a collection performing the same DML operation for each item in the collection it is possible to reduce context switches by bulk binding the whole collection to the DML statement in one operation.

In Oracle8i a collection must be defined for every column bound to the DML which can make the code rather long winded. Oracle9i allows us to use Record structures during bulk operations so long as we don't reference individual columns of the collection. This restriction means that updates and deletes which have to reference inividual columns of the collection in the where clause are still restricted to the collection-per-column approach used in Oracle8i.

Bulk binds can improve the performance when loading collections from a queries. The BULK COLLECT INTO construct binds the output of the query to the collection.



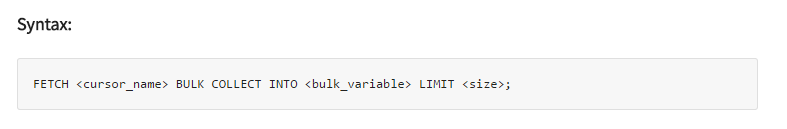


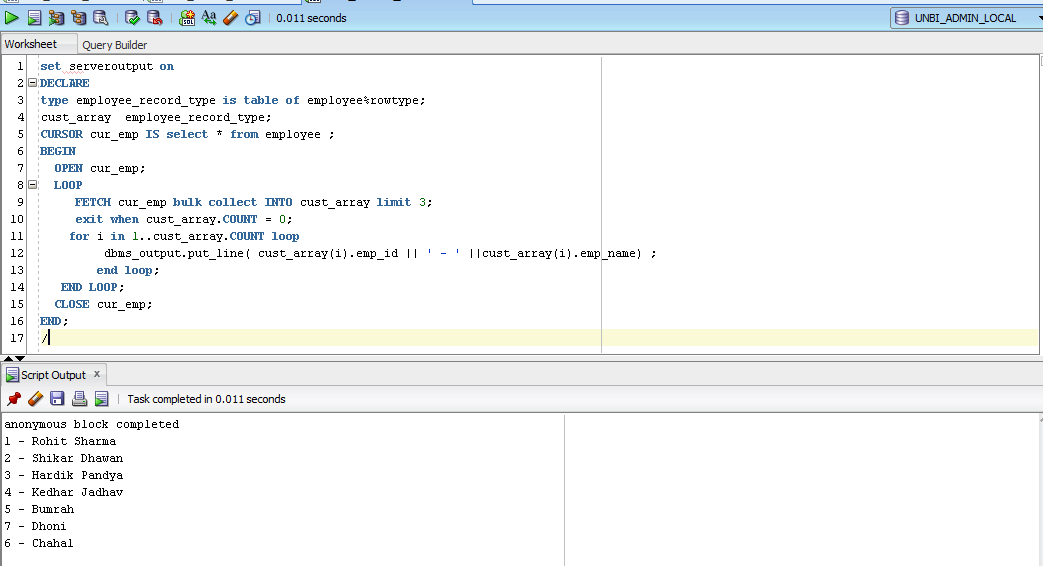
**LIMIT CLAUSE**

The bulk collect concept loads the entire data into the target collection variable as a bulk i.e. the whole data will be populated into the collection variable in a single-go. But this is not advisable when the total record that needs to be loaded is very large, because when PL/SQL tries to load the entire data it consumes more session memory. Hence, it is always good to limit the size of this bulk collect operation.

However, this size limit can be easily achieved by introducing the ROWNUM condition in the 'SELECT' statement, whereas in the case of cursor this is not possible.

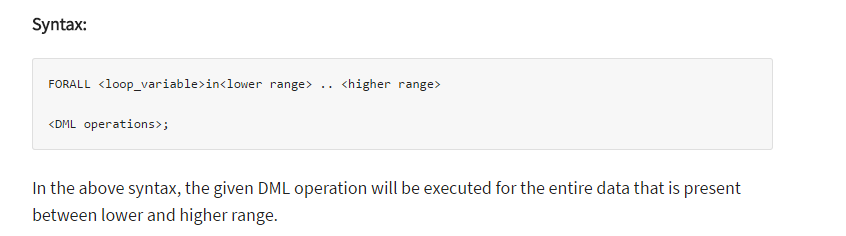
To overcome this Oracle has provided 'LIMIT' clause that defines the number of records that needs to be included in the bulk.

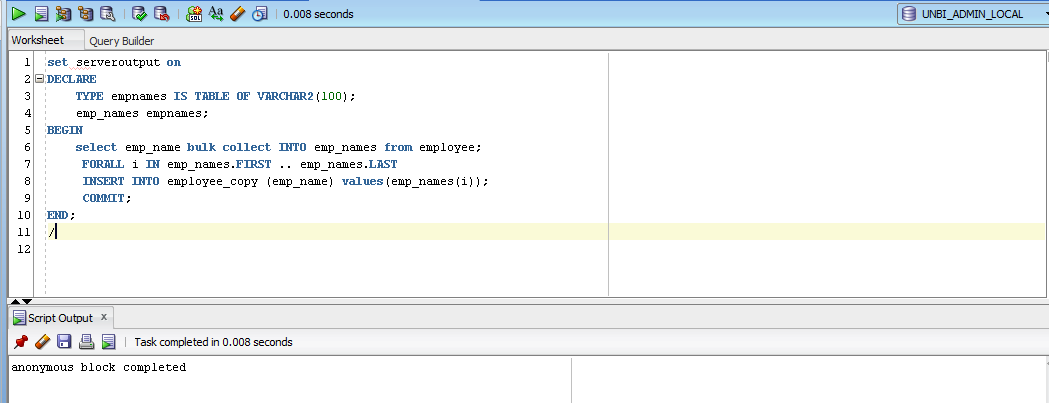


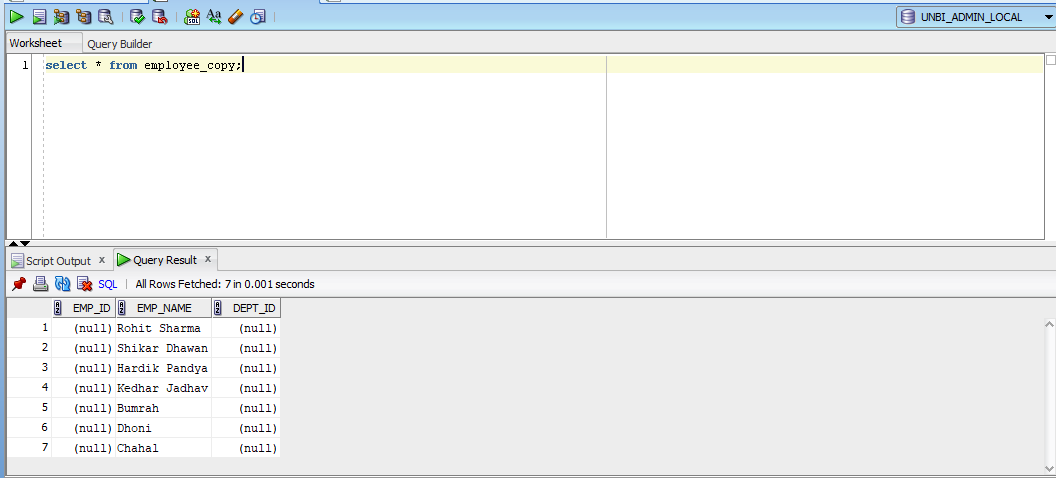


**FOR ALL**

The FORALL allows to perform the DML operations on data in bulk. It is similar to that of FOR loop statement except in FOR loop things happen at the record-level whereas in FORALL there is no LOOP concept. Instead the entire data present in the given range is processed at the same time.







**DYNAMIC NATIVE – EXECUTE IMMEDIATE**

Dynamic SQL is a programming methodology for generating and running statements at run-time. It is mainly used to write the general-purpose and flexible programs where the SQL statements will be created and executed at run-time based on the requirement.

Native Dynamic SQL is the easier way to write dynamic SQL. It uses the 'EXECUTE IMMEDIATE' command to create and execute the SQL at run-time. But to use this way, the datatype and number of variable that to be used at a run time need to be known before.

