**EXPERIMENT NO-13**

**AIM-** How to bundle related PL/SQL code and data into a package. The package might include a set of procedures that forms an API, or a pool of type definitions and variable declarations. The package is compiled and stored in the database, where its contents can be shared by many applications.

**PACKAGE-**

A package is a schema object that groups logically related PL/SQL types, variables, constants, subprograms, cursors, and exceptions. A package is compiled and stored in the database, where many applications can share its contents. A package always has a specification, which declares the public items that can be referenced from outside the package.

If the public items include cursors or subprograms, then the package must also have a body. The body must define queries for public cursors and code for public subprograms. The body can also declare and define private items that cannot be referenced from outside the package, but are necessary for the internal workings of the package. Finally, the body can have an initialization part, whose statements initialize variables and do other one-time setup steps, and an exception-handling part. You can change the body without changing the specification or the references to the public items; therefore, you can think of the package body as a black box.

**PACKAGE SPECIFICATIONS:**

The package specification contains:

* Name of the package
* Names of the data types of any arguments
* This declaration is local to the database and global to the package

This means that procedures, functions, variables, constants, cursors and exceptions and other objects, declared in a package are accessible from anywhere in the package.  
Therefore, all the information a package needs, to execute a stored subprogram, is contained in the package specifications itself.

**SYNTAX:**

*Create [or replace] package package\_name*

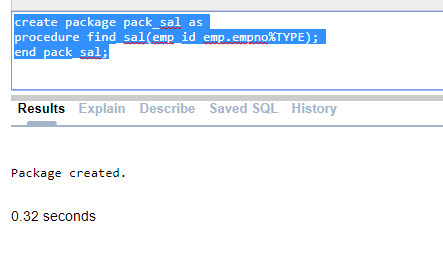
*{is | as}*

*Package\_specification*

*End package\_name;*

**EXAMPLE:**

Create package named as pack\_sal that continues procedure to find sal of particular employee by specifying its empno.



**PACKAGE BODY:**

The body of a package contains the definition of public objects that are declared in the specification.The body can also contain other object declarations that are private to the package.  
The objects declared privately in the package body are not accessible to other objects outside the package.  
Unlike package specification, the package body can contain subprogram bodies.  
After the package is written, debugged, compiled and stored in the database applications can reference the package's types, call its subprograms, use its cursors, or raise its exceptions.

**SYNTAX:**

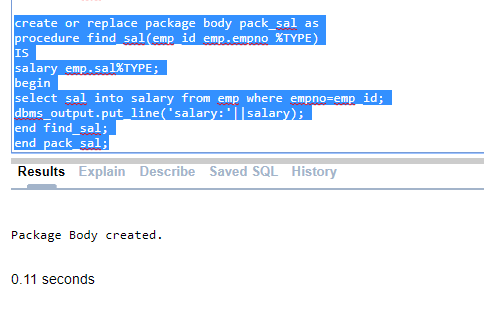
*Create [or replace] package body package\_name*

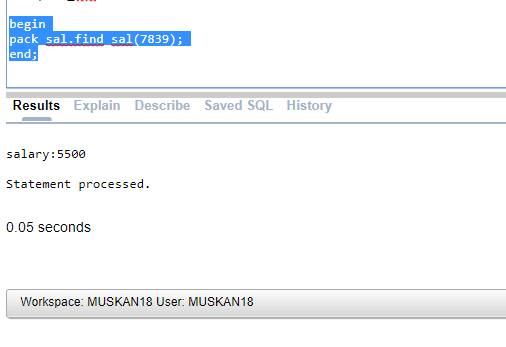
*{is | as}*

*Package\_body*

*End package\_name;*

**EXAMPLE:**

****

****

**ADVANTAGES:**

* Packages enable the organization of commercial applications into efficient modules. Each package is easily understood and the interfaces between packages are simple, clear and well defined.
* Packages allow granting of privileges efficiently .
* A package's public variables and cursors persist for the duration of the session. Therefore all cursors and procedures that execute in this environment can share them .

**EXPERIMENT NO-14,15**

**AIM-** To create a pl/sql block structure which is fired when DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed

**TRIGGER:**

 A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

**PURPOSE:**

It helps in auditing, imposing security checks, preventing invalid transactions, enforcing refrential integrity constraint and also storing some information on table axis.

**SYNTAX:**

*CREATE [OR REPLACE ] TRIGGER trigger\_name*

*{BEFORE | AFTER | INSTEAD OF}*

*{INSERT [OR] | UPDATE [OR] | DELETE}*

*[OF col\_name]*

*ON table\_name*

*[REFERENCING OLD AS o NEW AS n]*

*[FOR EACH ROW]*

*WHEN (condition)*

*DECLARE*

*Declaration-statements*

*BEGIN*

*Executable-statements*

*EXCEPTION*

*Exception-handling-statements*

*END;*

Where,

* CREATE [OR REPLACE] TRIGGER trigger\_name − Creates or replaces an existing trigger with the *trigger\_name*.
* {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
* [OF col\_name] − This specifies the column name that will be updated.
* [ON table\_name] − This specifies the name of the table associated with the trigger.
* [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
* [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

**EXAMPLE:**

Consider a table employee having id, name, job, salary and department. create a row level trigger that is executed for insertion, deletion and updation before that trigger is made to show the difference of salary between old and new salary.

