1 marks:

1) SGA is known as collection of background processes and complex memory architecture.
2) What is CKPT process? Ans: **CKPT** (Oracle **Checkpoint Process**) is an Oracle background **process** that timestamps all datafiles and control files to indicate that a **checkpoint** has occurred. The "DBWR checkpoints" statistic (v$sysstat) indicates the number of **checkpoint** requests completed.
3) Explain start command in sqlplus.

Ans: Runs the SQL*Plus statements in the specified script. The script can be called from a web server in *i*SQL*Plus, or from the local file system or a web server in SQL*Plus command-line. You can pass values to script variables in the usual way.

Terms

Refer to the following list for a description of each term or clause:

url

Specifies the Uniform Resource Locator of a script to run on the specified web server. SQL*Plus supports HTTP, FTP and gopher protocols.

file_name[.*ext*]

[The script you wish to execute. The file can contain any command that you can run interactively.

If you do not specify an extension, SQL*Plus assumes the default command-file extension (normally SQL). For information on changing this default extension, see the SUFFIX variable of the SET command in this chapter.

When you enter START *file_name.ext*, SQL*Plus searches for a file with the filename and extension you specify in the current default directory. If SQL*Plus does not find such a file, SQL*Plus will search a system-dependent path to find the file. Some operating systems may not support the path search. Consult the Oracle installation and user's manual(s) provided for your operating system for specific information related to your operating system environment.]

4) What is a VIEW in Oracle?

An Oracle VIEW, in essence, is a virtual table that does not physically exist. Rather, it is created by a query joining one or more tables.

Create VIEW

Syntax

The syntax for the CREATE VIEW Statement in Oracle/PLSQL is:

```
CREATE VIEW view_name AS
  SELECT columns
  FROM tables
  [WHERE conditions];
```

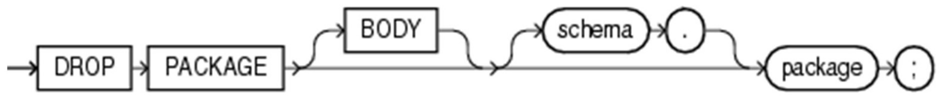
5) DROP PACKAGE Statement

The `DROP PACKAGE` statement drops a stored package from the database. This statement drops the body and specification of a package.

he package must be in your own schema or you must have the `DROP ANY PROCEDURE` system privilege.

Syntax

***drop_package*::=**



Dropping a Package: Example The following statement drops the specification and body of the emp_mgmt package, which was created in Creating a Package Body: Example, invalidating all objects that depend on the specification:

```
DROP PACKAGE emp_mgmt;
```

6) DB_CACHE_SIZE

Property	Description
Parameter type	Big integer
Syntax	`DB_CACHE_SIZE = integer [K	M	G]`

`DB_CACHE_SIZE` specifies the size of the `DEFAULT` buffer pool for buffers with the primary block size (the block size defined by the `DB_BLOCK_SIZE` initialization parameter).

The value must be at least `4M * number of cpus * granule size` (smaller values are automatically rounded up to this value). A user-specified value larger than this is rounded up to the nearest granule size. A value of zero is illegal because it is needed for the `DEFAULT` memory pool of the primary block size, which is the block size for the `SYSTEM` tablespace.

7) What is the difference between Shared Hosting and Dedicated server

Shared Hosting allows multiple users/websites/accounts to be hosted on a single web server. Dedicated hosting, in its turn, is a single server solely devoted to one user.
In a nutshell, the difference between a Shared and a Dedicated hosting server is like an apartment in a block of flats and a cottage house. Both are good for living in. Still, there are some advantages and disadvantages of choosing one or the other. Let's have a deeper look at each of the server type taking into consideration management, performance, resources, security and cost.

8) Object table stores object information.

9) To copy rows from one table to another

```
INSERT INTO TableNew
SELECT * FROM TableOld
WHERE [Conditions]
```

9) in out mode of parameter is default procedure and function true or false =true.

Triggers can be defined on the table, view, schema, or database with which the event is associated.

10) Benefits of Triggers

Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

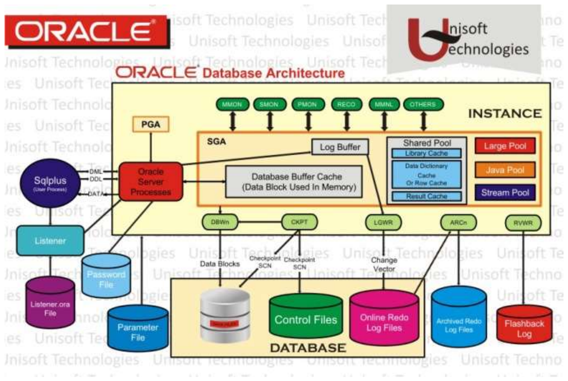
The syntax for creating a trigger is −

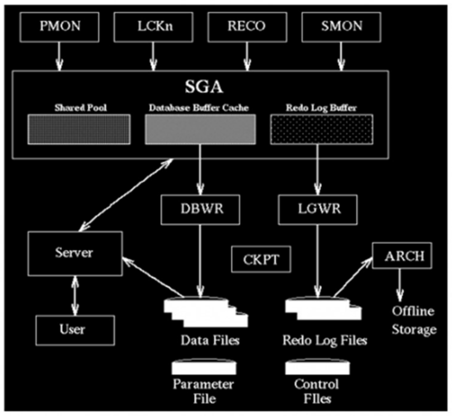
```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

11)

4 Marks:

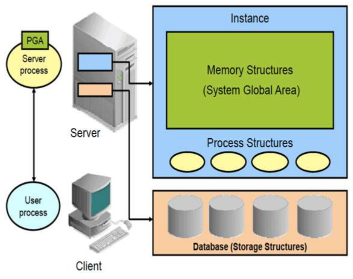
Explain Oracle Architecture :





Memory Structures

There are three major structures in Oracle Database server architecture: memory structures, process structures, and storage structures. A basic Oracle database system consists of an Oracle database and a database instance.



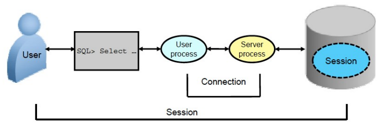
The database consists of both physical structures and logical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures.

The instance consists of memory structures and background processes associated with that instance. Every time an instance is started, a shared memory area called the System Global Area (SGA) is allocated and the background processes are started. Processes are jobs that work in the memory of computers. A process is defined as a "thread of control" or a mechanism in an operating system that can run a series of steps. After starting a database instance, the Oracle software associates the instance with a specific physical database. This is called mounting the database. The database is then ready to be opened, which makes it accessible to authorized users.

Note: Oracle Automatic Storage Management (ASM) uses the concept of an instance for the memory and process components, but is not associated with a specific database.

Connecting to the Database Instance

Connections and **sessions** are closely related to user processes but are very different in meaning.

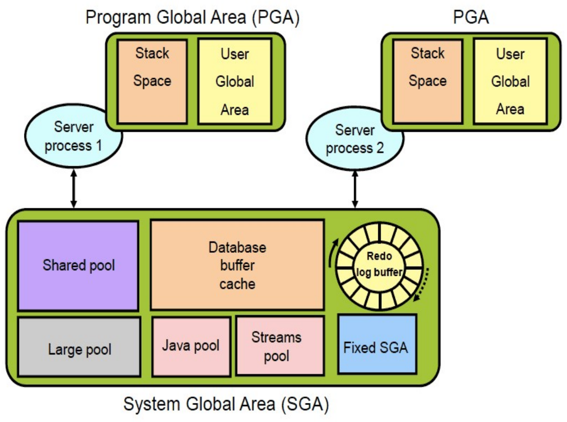


A **connection** is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established using available interprocess communication mechanisms (on a computer that runs both the user process and Oracle Database) or network software (when different computers run the database application and Oracle Database and communicate through a network).

A **session** represents the state of a current user login to the database instance. For example, when a user starts SQL*Plus, the user must provide a valid username and password, and then a session is established for that user. A session lasts from the time a user connects until the user disconnects or exits the database application. Multiple sessions can be created and exist concurrently for a single Oracle database user using the same username. For example, a user with the username/password of HR/HR can connect to the same Oracle Database instance several times.

Oracle Database Memory Structures

Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data that is shared among users, and private data areas for each connected user.



Two basic memory structures are associated with an instance:

- **System Global Area (SGA)**: Group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- **Program Global Areas (PGA)**: Memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

The SGA is the memory area that contains data and control information for the instance. The SGA includes the following data structures:

- **Shared pool**: Caches various constructs that can be shared among users
- **Database buffer cache**: Caches blocks of data retrieved from the database
- **Redo log buffer**: Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk
- **Large pool**: Optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and I/O server processes
- **Java pool**: Used for all session-specific Java code and data in the Java Virtual Machine (JVM)
- **Streams pool**: Used by Oracle Streams to store information required by capture and apply
- **Fixed SGA**: An internal housekeeping area containing general information about the state of the database and the instance, and information communicated between processes When you start the instance, the amount of memory allocated for the SGA is displayed.

A **Program Global Area (PGA)** is a memory region that contains data and control information for each server process. An Oracle server process services a client's requests. Each server process has its own private PGA that is allocated when the server process is started. Access to the PGA is exclusive to that server process, and the PGA is read and written only by the Oracle code acting on its behalf. The PGA is divided into two major areas: stack space and the user global area (UGA).

With the dynamic SGA infrastructure, the sizes of the database buffer cache, the shared pool, the large pool, the Java pool, and the Streams pool can change without shutting down the instance.

The Oracle Database server uses initialization parameters to create and manage memory structures. The simplest way to manage memory is to allow the database to automatically manage and tune it for you. To do so (on most platforms), you only have to set a target memory size initialization parameter (**MEMORY_TARGET**) and a maximum memory size initialization parameter (**MEMORY_MAX_TARGET**).

Process Architecture

The processes in an Oracle Database system can be divided into three major groups:

- **User processes** that run the application or Oracle tool code
- **Oracle Database processes** that run the Oracle Database server code (including server processes and background processes)
- **Oracle daemons and application processes** not specific to a single database

When a user runs an application program or an Oracle tool such as SQL*Plus, the term user process is used to refer to the user's application. The user process may or may not be on the database server machine. Oracle Database also creates a server process to execute the commands issued by the user process. In addition, the Oracle server also has a set of background processes for an instance that interact with each other and with the operating system to manage the memory structures, asynchronously perform I/O to write data to disk, and perform other required tasks. The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options. The code for connected users can be configured as a dedicated server or a shared server.

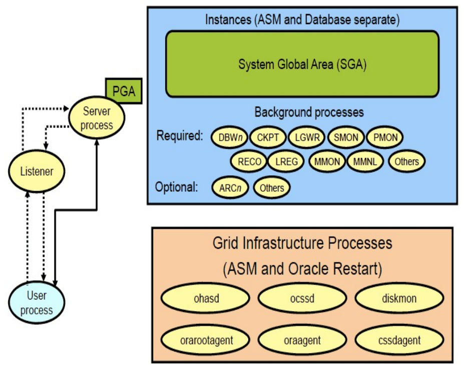
Dedicated server

For each session, the database application is run by a user process that is served by a dedicated server process that executes Oracle database server code.

Shared server

Eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. A shared server process serves any client request.

Process Structures



Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to the instance. The user process represents the application or tool that connects to the Oracle database. It may be on the same machine as the Oracle database, or it may exist on a remote client and use a network to reach the Oracle database. The user process first communicates with a listener process that creates a server process in a dedicated environment.

Server processes created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application.
- Read necessary data blocks from data files on disk into the shared database buffers of the SGA (if the blocks are not already present in the SGA).
- Return results in such a way that the application can process the information.

Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called background processes. An Oracle Database instance can have many background processes.

The background processes commonly seen in non-RAC, non-ASM environments can include the following:

- Database Writer process (DBWn)
- Log Writer process (LGWR)
- Checkpoint process (CKPT)
- System monitor process (SMON)
- Process monitor process (PMON)
- Recoverer process (RECO)

Database Objects in DBMS

A **database object** is any defined object in a database that is used to store or reference data.Anything which we make from **create command** is known as Database Object.It can be used to hold and manipulate the data.Some of the examples of database objects are : view, sequence, indexes, etc.

- **Table –** Basic unit of storage; composed rows and columns
- **View –** Logically represents subsets of data from one or more tables
- **Sequence –** Generates primary key values
- **Index –** Improves the performance of some queries
- **Synonym –** Alternative name for an object

Different database Objects :

1. **Table –** This database object is used to create a table in database.
   Syntax :

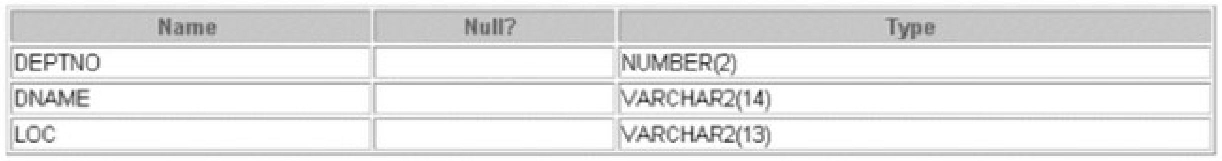
```
CREATE TABLE [schema.]table
               (column datatype [DEFAULT expr][, ...]);
```

   Example :

```
CREATE TABLE dept
           (deptno NUMBER(2),
            dname VARCHAR2(14),
            loc VARCHAR2(13));
```

   Output :

```
DESCRIBE dept;
```



Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

2. **View –** This database object is used to create a view in database.A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.
   Syntax :

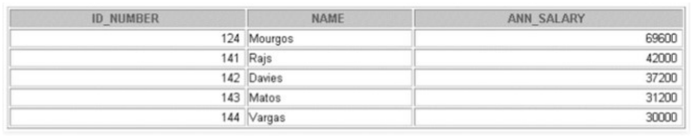
```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
                       [(alias[, alias]...)]
                       AS subquery
                       [WITH CHECK OPTION [CONSTRAINT constraint]]
                       [WITH READ ONLY [CONSTRAINT constraint]];
```

   Example :

```
CREATE VIEW salvu50
               AS SELECT employee_id ID_NUMBER, last_name NAME,
               salary*12 ANN_SALARY
               FROM employees
               WHERE department_id = 50;
```

   Output :

```
SELECT *
FROM salvu50;
```



ID_NUMBER	NAME	ANN_SALARY
124	Mourgos	69600
141	Rajs	42000
142	Davies	37200
143	Matos	31200
144	Vargas	30000

3. **Sequence –** This database object is used to create a sequence in database.A sequence is a user created database object that can be shared by multiple users to generate unique integers. A typical usage for sequences is to create a primary key value, which must be unique for each row.The sequence is generated and incremented (or decremented) by an internal Oracle routine.
   Syntax :
```
CREATE SEQUENCE sequence
                    [INCREMENT BY n]
                    [START WITH n]
                    [{MAXVALUE n | NOMAXVALUE}]
                    [{MINVALUE n | NOMINVALUE}]
                    [{CYCLE | NOCYCLE}]
                    [{CACHE n | NOCACHE}];
```
   Example :
```
CREATE SEQUENCE dept_deptid_seq
                        INCREMENT BY 10
                        START WITH 120
                        MAXVALUE 9999
                        NOCACHE
                        NOCYCLE;
```
   Check if sequence is created by :
```
SELECT sequence_name, min_value, max_value,
                       increment_by, last_number
                       FROM   user_sequences;
```
4. **Index –** This database object is used to create a indexes in database.An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer.Indexes can be created explicitly or automatically. If you do not have an index on the column, then a full table scan occurs.
   An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle server. Once an index is created, no direct activity is required by the user.Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.
   Syntax :
```
CREATE INDEX index
            ON table (column[, column]...);
```
   Example :
```
CREATE INDEX emp_last_name_idx
                ON  employees(last_name);
```
5. **Synonym –** This database object is used to create a indexes in database.It simplify access to objects by creating a synonym(another name for an object). With synonyms, you can Ease referring to a table owned by another user and shorten lengthy object names.To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence,procedure, or other objects. This method can be especially useful with lengthy object names, such as views.
   In the syntax:
   PUBLIC : creates a synonym accessible to all users
   synonym : is the name of the synonym to be created
   object : identifies the object for which the synonym is created
   Syntax :
```
CREATE [PUBLIC] SYNONYM synonym FOR  object;
```

Example :
```
CREATE SYNONYM d_sum FOR dept_sum_vu;
```

Oracle Background Processes :

Database writer (DBW*n*)

The database writer writes modified blocks from the database buffer cache to the datafiles. Oracle Database allows a maximum of 20 database writer processes (DBW0-DBW9 and DBWa-DBWj). The `DB_WRITER_PROCESSES` initialization parameter specifies the number of DBW*n* processes. The database selects an appropriate default setting for this initialization parameter or adjusts a user-specified setting based on the number of CPUs and the number of processor groups.

For more information about setting the `DB_WRITER_PROCESSES` initialization parameter, see the *Oracle Database Performance Tuning Guide*.

Log writer (LGWR)

The log writer process writes redo log entries to disk. Redo log entries are generated in the redo log buffer of the system global area (SGA). LGWR writes the redo log entries sequentially into a redo log file. If the database has a multiplexed redo log, then LGWR writes the redo log entries to a group of redo log files. See Chapter 10, "Managing the Redo Log" for information about the log writer process.

Checkpoint (CKPT)

At specific times, all modified database buffers in the system global area are written to the datafiles by DBW*n*. This event is called a checkpoint. The checkpoint process is responsible for signalling DBW*n* at checkpoints and updating all the datafiles and control files of the database to indicate the most recent checkpoint.

System monitor (SMON)

The system monitor performs recovery when a failed instance starts up again. In an Oracle Real Application Clusters database, the SMON process of one instance can perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use and recovers dead transactions skipped during system failure and instance recovery because of file-read or offline errors. These transactions are eventually recovered by SMON when the tablespace or file is brought back online.

Process monitor (PMON)

The process monitor performs process recovery when a user process fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using. PMON also checks on the dispatcher processes (described later in this table) and server processes and restarts them if they have failed.

Archiver (ARC*n*)

One or more archiver processes copy the redo log files to archival storage when they are full or a log switch occurs. Archiver processes are the subject of Chapter 11, "Managing Archived Redo Logs".

Recoverer (RECO)

The recoverer process is used to resolve distributed transactions that are pending because of a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions. For information about this process and how to start it, see Chapter 33, "Managing Distributed Transactions".

Dispatcher (D*nnn*)

Dispatchers are optional background processes, present only when the shared server configuration is used. Shared server was discussed previously in "Configuring Oracle Database for Shared Server".

Global Cache Service (LMS)

In an Oracle Real Application Clusters environment, this process manages resources and provides inter-instance resource control.

Optimal Flexible Architecture (OFA)

Oracle Corporation recommends the Optimal Flexible Architecture (OFA) standard for Oracle8. The OFA standard is a set of configuration guidelines for fast, reliable Oracle databases that require little maintenance.

OFA is designed to:

- organize large amounts of complicated software and data on disk to avoid device bottlenecks and poor performance

- facilitate routine administrative tasks like software and data backup functions, which are often vulnerable to data corruption
- alleviate switching among multiple Oracle databases
- adequately manage and administer database growth
- help eliminate fragmentation of free space in the data dictionary, isolate other fragmentation, and minimize resource contention

Characteristics of OFA-Compliant Database

An OFA-compliant database provides the following benefits:

File System Organization

The file system is organized to allow easy administration and accommodate scalability for:

- adding data into existing databases
- adding users
- creating databases
- adding hardware

Distributed I/O Loads

I/O loads are distributed across enough disk drives to prevent performance bottlenecks.

Hardware Support

Hardware costs are minimized only when it does not conflict with operational considerations.

Safeguards Against Drive Failures

By spreading applications across more than one drive, drive failures impact as few applications as possible.

Distribution of Home Directories

The following items can be distributed across more than one disk drive:

- the collection of home directories
- the contents of an individual home directory

Integrity of Login Home Directories

It is possible to add, move, or delete login home directories without having to revise programs that refer to them.

Independence of UNIX Directory Subtrees

Categories of files are separated into independent UNIX directory subtrees so that files in one category are minimally affected by operations on files in other categories.

Supports Concurrent Execution of Application Software

It is possible to execute multiple versions of applications software simultaneously, allowing the user to test and use a new release of an application before abandoning the previous version. Transferring to a new version after an upgrade is simple for the administrator and transparent for the user.

Distinguishes Administrative Information for each Database

The ability to separate administrative information about one database from that of another, ensures a reasonable structure for the organization and storage of administrative data.

Uses Consistent Database File Naming

Database files are named so that:

- database files are easily distinguishable from all other files
- files of one database are easily distinguishable from files of another database
- control files, redo log files, and data files are identifiable as such
- the association of data file to tablespace is clearly indicated

Separation of Tablespace Contents

Tablespace contents are separated to:

- minimize tablespace free space fragmentation
- minimize I/O request contention
- maximize administrative flexibility

Tuning of I/O Loads across all Drives

I/O loads are tuned across all drives, including drives storing Oracle data in raw devices.

Additional Benefits of OFA for Parallel Server

For Oracle Parallel Server Installations:

- administrative data is stored in a central place, accessible to all database administrators
- administrative data for an instance is associated with the instance by the file name

What is a 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.

In either the package specification or package body, you can map a package subprogram to an external Java or C subprogram by using a **call specification**, which maps the external subprogram name, parameter types, and return type to their SQL counterparts.

The `AUTHID` **clause** of the package specification determines whether the subprograms and cursors in the package run with the privileges of their definer (the default) or invoker, and whether their unqualified references to schema objects are resolved in the schema of the definer or invoker.

The `ACCESSIBLE BY` **clause** of the package specification lets you specify a white list of PL/SQL units that can access the package. You use this clause in situations like these:

- You implement a PL/SQL application as several packages—one package that provides the application programming interface (API) and helper packages to do the work. You want clients to have access to the API, but not to the helper packages. Therefore, you omit the `ACCESSIBLE BY` clause from the API package specification and include it in each helper package specification, where you specify that only the API package can access the helper package.
- You create a utility package to provide services to some, but not all, PL/SQL units in the same schema. To restrict use of the package to the intended units, you list them in the `ACCESSIBLE BY` clause in the package specification.

Reasons to Use Packages

Packages support the development and maintenance of reliable, reusable code with the following features:

- **Modularity**

  Packages let you encapsulate logically related types, variables, constants, subprograms, cursors, and exceptions in named PL/SQL modules. You can make each package easy to understand, and make the interfaces between packages simple, clear, and well defined. This practice aids application development.

- **Easier Application Design**

When designing an application, all you need initially is the interface information in the package specifications. You can code and compile specifications without their bodies. Next, you can compile standalone subprograms that reference the packages. You need not fully define the package bodies until you are ready to complete the application.

- **Hidden Implementation Details**

  Packages let you share your interface information in the package specification, and hide the implementation details in the package body. Hiding the implementation details in the body has these advantages:

  - You can change the implementation details without affecting the application interface.
  - Application users cannot develop code that depends on implementation details that you might want to change.
- **Added Functionality**

  Package public variables and cursors can persist for the life of a session. They can be shared by all subprograms that run in the environment. They let you maintain data across transactions without storing it in the database. (For the situations in which package public variables and cursors do not persist for the life of a session, see "Package State".)

- **Better Performance**

  The first time you invoke a package subprogram, Oracle Database loads the whole package into memory. Subsequent invocations of other subprograms in same the package require no disk I/O.

  Packages prevent cascading dependencies and unnecessary recompiling. For example, if you change the body of a package function, Oracle Database does not recompile other subprograms that invoke the function, because these subprograms depend only on the parameters and return value that are declared in the specification.

- **Easier to Grant Roles**

  You can grant roles on the package, instead of granting roles on each object in the package.

Note:

You cannot reference host variables from inside a package.

Package Specification

A **package specification** declares **public items**. The scope of a public item is the schema of the package. A public item is visible everywhere in the schema. To reference a public item that is in scope but not visible, qualify it with the package name. (For information about scope, visibility, and qualification, see "Scope and Visibility of Identifiers".)

Each public item declaration has all information needed to use the item. For example, suppose that a package specification declares the function `factorial` this way:

```
FUNCTION factorial (n INTEGER) RETURN INTEGER; -- returns n!
```

The declaration shows that `factorial` needs one argument of type `INTEGER` and returns a value of type `INTEGER`, which is invokers must know to invoke `factorial`. Invokers need not know how `factorial` is implemented (for example, whether it is iterative or recursive).

DML Triggers in SQL Server

DML triggers in SQL Server are fired when a DML event occurs. i.e. when data is inserted/ updated/deleted in the table by a user.

Creating triggers for a DML event

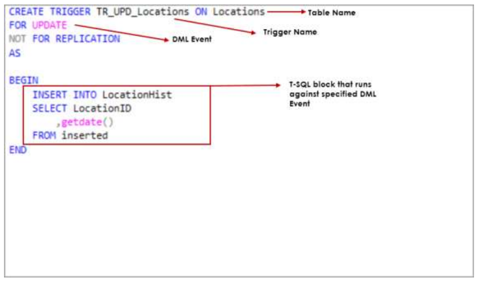
Let us create some sample tables and triggers in SQL Server.

```
CREATE TABLE Locations (LocationID int, LocName varchar(100))

CREATE TABLE LocationHist (LocationID int, ModifiedDate DATETIME)
```

We can create a DML trigger for a specific event or multiple events. The triggers in SQL Server(DML) fire on events irrespective to the number of rows affected.

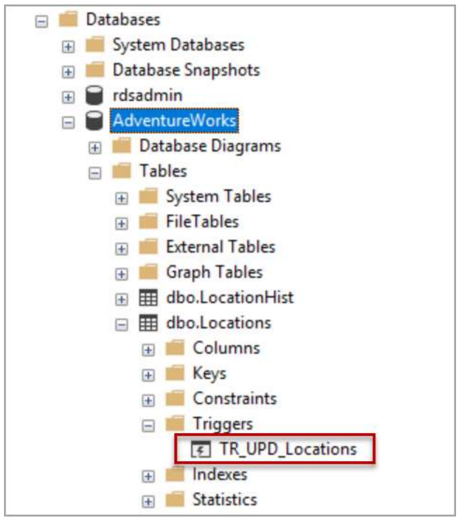
Below is the sample syntax for creating a DML trigger for update event.

```
CREATE TRIGGER TR_UPD_Locations ON Locations

FOR UPDATE

NOT FOR REPLICATION

AS



BEGIN

 INSERT INTO LocationHist

 SELECT LocationID

  ,getdate()

 FROM inserted

END
```



```
CREATE TRIGGER TR_UPD_Locations ON Locations
FOR UPDATE
NOT FOR REPLICATION
AS

BEGIN
    INSERT INTO LocationHist
    SELECT LocationID
        ,getdate()
    FROM inserted
END
```

These triggers are created at the table level. Upon successful creation of trigger, we can see the triggers by navigating to **Triggers** folder at table level. Please refer to the below image.



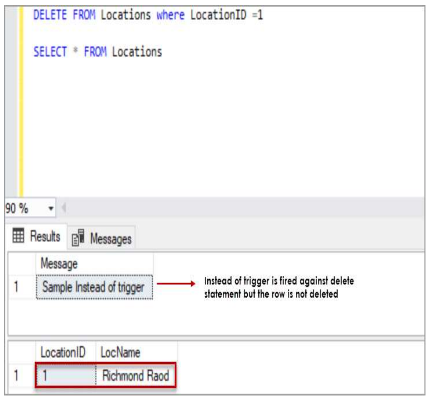
Instead of triggers in SQL Server

These triggers are fired before the DML event and the actual data is not modified in the table.

For example, if we specify an instead of trigger for delete on a table, when delete statement is issued against the table, the instead of trigger is fired and the T-SQL block inside the triggers in SQL Server is executed but the actual delete does not happen.

T-SQL Syntax for creating an instead of trigger

```
CREATE TRIGGER TR_DEL_Locations ON Locations

INSTEAD OF DELETE

AS

BEGIN

 Select 'Sample Instead of trigger' as [Message]

END
```



```
DELETE FROM Locations where LocationID =1

SELECT * FROM Locations
```

```
   Message
1  Sample Instead of trigger
```

Instead of trigger is fired against delete statement but the row is not deleted

```
   LocationID  LocName
1  1           Richmond Raod
```

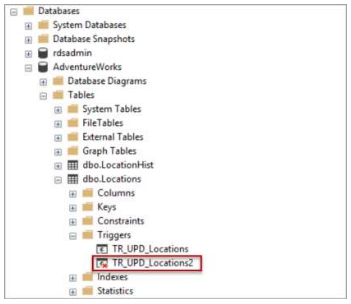
- If there are multiple triggers along with instead of trigger on the table, the instead of trigger is fired first in the order
- INSTEAD of triggers can be created on views
- we can define only one instead of trigger per INSERT, UPDATE, or DELETE statement on a table or view

Enabling and disabling DML triggers on a table

Navigate to triggers folder at the table level, select the trigger, Right click on trigger and Click on **Enable**/**Disable** to Enable or disable the trigger using **SSMS**.

Disabling specific SQL Server trigger on a table using T-SQL.

```
DISABLE TRIGGER TR_UPD_Locations2 on Locations
```



Enabling specific trigger on the table using T-SQL.

```
ENABLE TRIGGER TR_UPD_Locations2 on Locations
```

To enable all triggers on a table, use below syntax.

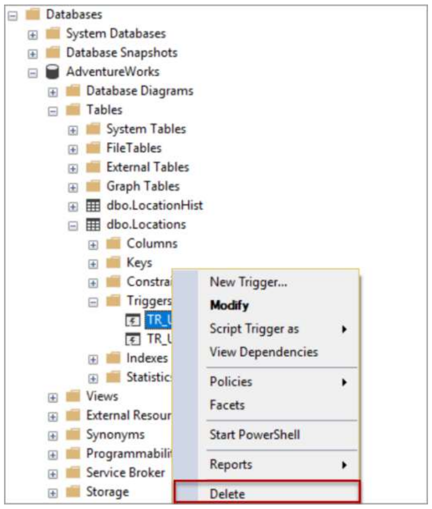
```
ENABLE TRIGGER ALL ON Locations
```

To disable all triggers on a table, use below syntax. This statement is not supported if the table is part of merge replication.

```
DISABLE TRIGGER ALL ON Locations
```

Dropping a trigger on a table.

To drop a DML trigger on the table using SQL Server management studio, navigate to the **Triggers** folder under the table. Select the table you want to drop, Right click on the trigger and click on **Delete**. Click **Ok**.



T-SQL to drop a trigger on the table.

```
DROP TRIGGER TRL_UPD_Locations2
```

Dropping a table will drop all the SQL Server triggers on the table along with the table.

Syntax for Exception Handling

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using ***WHEN others THEN*** −

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling goes here >
   WHEN exception1 THEN
      exception1-handling-statements
   WHEN exception2  THEN
      exception2-handling-statements
   WHEN exception3 THEN
      exception3-handling-statements
   ........
   WHEN others THEN
      exception3-handling-statements
END;
```

Example

Let us write a code to illustrate the concept. We will be using the CUSTOMERS table we had created and used in the previous chapters −

```
DECLARE
   c_id customers.id%type := 8;
   c_name customerS.Name%type;
   c_addr customers.address%type;
BEGIN
   SELECT  name, address INTO  c_name, c_addr
   FROM customers
   WHERE id = c_id;
   DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
   DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);

EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
No such customer!

PL/SQL procedure successfully completed.
```

The above program displays the name and address of a customer whose ID is given. Since there is no customer with ID value 8 in our database, the program raises the run-time exception **NO_DATA_FOUND**, which is captured in the **EXCEPTION block**.

Raising Exceptions

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**. Following is the simple syntax for raising an exception −

```
DECLARE
   exception_name EXCEPTION;
BEGIN
   IF condition THEN
      RAISE exception_name;
   END IF;
EXCEPTION
   WHEN exception_name THEN
   statement;
END;
```

You can use the above syntax in raising the Oracle standard exception or any user-defined exception. In the next section, we will give you an example on raising a user-defined exception. You can raise the Oracle standard exceptions in a similar way.

User-defined Exceptions

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure **DBMS_STANDARD.RAISE_APPLICATION_ERROR**.

The syntax for declaring an exception is −

```
DECLARE
```

```
   my-exception EXCEPTION;
```

Example

The following example illustrates the concept. This program asks for a customer ID, when the user enters an invalid ID, the exception **invalid_id** is raised.

```
DECLARE
   c_id customers.id%type := &cc_id;
   c_name customerS.Name%type;
   c_addr customers.address%type;
   -- user defined exception
   ex_invalid_id  EXCEPTION;
BEGIN
   IF c_id <= 0 THEN
      RAISE ex_invalid_id;
   ELSE
      SELECT  name, address INTO  c_name, c_addr
      FROM customers
      WHERE id = c_id;
      DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
      DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
   END IF;

EXCEPTION
   WHEN ex_invalid_id THEN
      dbms_output.put_line('ID must be greater than zero!');
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Enter value for cc_id: -6 (let's enter a value -6)
old  2: c_id customers.id%type := &cc_id;
new  2: c_id customers.id%type := -6;
ID must be greater than zero!

PL/SQL procedure successfully completed.
```

Pre-defined Exceptions

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO_DATA_FOUND is raised when a SELECT INTO statement returns no rows. The following table lists few of the important pre-defined exceptions −

Exception	Oracle Error	SQLCODE	Description
ACCESS_INTO_NULL	06530	-6530	It is raised when a null object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the WHEN clause of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.

INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when a program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero