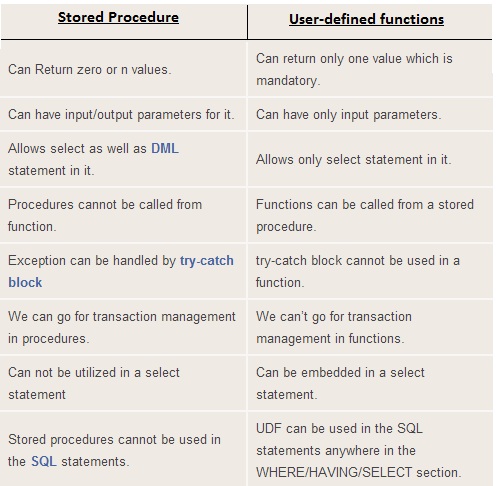
Below is the difference between Sp and UDF 

User defined functions have three main categories:

**Scalar-valued function** - returns a scalar value such as an integer or a timestamp. Can be used as column name in queries.

**Inline function** - can have a single SELECT statement.

**Table-valued function** - can contain any number of statements that populate the table variable to be returned. They are useful when a set of rows need to be returned.

Oracle 9i and Oracle 10g....there are more implementation differences. In terms of architecture, 9i is based on Internet technology while 10g is grid computing based one. Many DBA features like Automated Storage Management (ASM), Automatic Workload Repository (AWR), Automatic Database Diagnostic Monitor (ADDM) were introduced.  
  
For developers, 10g is more stable than 9i. All the earlier bugs reported were fixed in 10g release.  
  
Performance wise, it was more impressive. It increased the data chunk size during I/O between the SQL and PL/SQL engines. Use of collections was recommended.  
PLS\_INTEGER, a new data type was added which enhances performance. ANYDATA data type was introduced to hold a data of variant feature.  
FLASHBACK option was made more stronger like TABLE FLASHBACK was introduced. Regular expression function were introduced REGEXP\_SUBSTR, REGEXP\_INSTR, REGEXP\_REPLACE.   
Oracle Recyclebin was introduced. The dropped objects can be restored from the recyclebin until it is purged.

**Summary Of Differences Between oracle Database 9i and 10g.**

The below Points are completely New in oracle database 10g (Not there in oracle 9i)

* Automatic Storage Management(ASM).
* Automatic Shared Memory Management(ASMM).
* Automatic Database Diagnostic Monitor(ADDM).
* Automatic Workload Repository(AWR).
* Flashback Technologies.
* Data Pump replaces Traditional EXP/IMP.
* Automatic Checkpoint Tunning(FAST\_START\_MTTR\_TARGET).
* Automatic Undo Retention Tunning.
* Introduced Default Permanent Tablespace (USERS).
* Introduced SYSAUX tablespace.
* Streams Technology(STREAMS POOL).
* Introduced Big file Tablespace Option and Rename Tablespace Command.
* Automatic SQL Tunning.
* Temporary Tablespace Group and Default Temporary Tablespace.
* Recovery Manager Enhancements(RMAN).
* DBMS Scheduler Packages and DBMS File Transfer Packages.

**Detailed Explanation of the above summary of differences  between oracle database 10g and 9i.**

**Automatic Storage Management(ASM).**

ASM  means Automatic Storage Management to simplify the Storage of datafiles , controlfiles and  Redolog files.ASM  is the extension of OMF (oracle managed files).ASM functionality is controlled by ASM instance.It is not full database instance its just  a memory structures.

The main components of ASM are disk groups, each of which comprise of several physical disks that are controlled as a single unit. The physical disks are known as ASM disks, while the files that reside on the disks are know as ASM files.

ASM provides the following the functionalities

* Manages groups of disks, called disk groups.
* Manages disk redundancy within a disk group
* Supports large files.

**Initialization Parameters and ASM instance Creation**

* The initialization parameters that are of specific interest for an ASM instance are
* **INSTANCE\_TYPE** – Set to ASM or RDBMS depending on the instance type. The default is RDBMS.
* **DB\_UNIQUE\_NAME** – Specifies a globally unique name for the database. This defaults to +ASM but must be altered if you intend to run multiple ASM instances.
* **ASM\_POWER\_LIMIT** -The maximum power for a rebalancing operation on an ASM instance. The valid values range from 1 to 11, with 1 being the default. The higher the limit the more resources are allocated resulting in faster rebalancing operations. This value is also used as the default when the POWER clause is omitted from a rebalance operation.
* **ASM\_DISKGROUPS** - The list of disk groups that should be mounted by an ASM instance during instance startup, or by the ALTER DISKGROUP ALL MOUNT statement. ASM configuration changes are automatically reflected in this parameter.
* **ASM\_DISKSTRING** – Specifies a value that can be used to limit the disks considered for discovery. Altering the default value may improve the speed of disk group mount time and the speed of adding a disk to a disk group. Changing the parameter to a value which prevents the discovery of already mounted disks results in an error. The default value is NULL allowing all suitable disks to be considered.

**Automatic Shared Memory Management(ASMM).**

Oracle database 10g’s New features.By default sga\_target  is enabled.But  always sga\_max\_size is greater than or equal to sga\_target.So once sga\_target is enabled means, no need to set the auto tuned parameters like

1. db\_cache\_size
2. shared\_pool\_size
3. large\_pool\_size
4. java\_pool\_size

So ASMM automatically readjusts the sizes of the main pools.

**Automatic Database Diagnostic Monitor(ADDM).**

ADDM means automatic database diagnostic monitor and is a oracle database 10g’s new features. ADDM does analysis of the database, identifies problems and their potential causes, and comes up with recommendations for fixing the problems. It can call all other advisors also.

ADDM stores the snapshot in SYSAUX tablespace.

The main features of the ADDM are as follows

* ADDM runs automatically in the background process MMON whenever a snapshot of in-memory statistics is taken. ADDM does analysis of the statistics collected between two snapshots.
* ADDM analysis results are written back to the workload repository for further use.
* ADDM uses the new wait and time statistics model, where activities with high time consumption are analyzed on a priority basis. This is where the big impact lies.
* ADDM can also be invoked manually

The Automatic Database Diagnostic Monitor (ADDM) is a new diagnosis tool that runs automatically every hour, after the AWR takes a new snapshot. The ADDM uses the AWR performance snapshots to locate the root causes for poor performance and saves recommendations for improving performance in SYSAUX.

ADDM also gives the recommendations for

* Sql tuning advisor
* Sql access advisor
* Segment advisor
* Undo advisor
* Redo log file size advisor

**Automatic Workload Repository (AWR)**

AWR is the oracle database 10g’s new features.Oracle8i introduced the Statspack functionality which Oracle9i extended. In Oracle 10g statspack has evolved into the Automatic Workload Repository (AWR). In your database some repository tables are created automatically when database is created.Each and every seconds AWR collects issues and activities in the database.

The repository is a source of information for several other Oracle 10g features.

* Automatic Database Diagnostic Monitor
* SQL Tuning Advisor
* Undo Advisor
* Segment Advisor

To check the AWR status, Show parameter statistics\_level. If you want to enable the AWR report set statistics\_level=Typical.If you want to disable the AWR report statistics\_level=normal.

When you enable the AWR report the MMON (Manageability Monitor) background process will active.It will write issues in the repository tables. By default Every one hour AWR will generate a snapshot in SYSAUX tablespace and the default retention period is 7 days.If you want to take snapshot every 15 minutes and retention period 10 days means

begin

dbms\_workload\_repository.modify\_snapshot\_settings(

retention=>14400,       —–Minutes (=30 days). Current Value retained if null

interval=>15);              — Minutes .Current value retained if null.

End;

/

**Flashback Technologies**

Flashback technologies is the oracle database 10g’s new features.

**Types of Flashback Technologies**

There are six basic types of Flashback recovery, discussed below in detail:

* Flashback Query
* Flashback Version Query
* Flashback Transaction Query
* Flashback Table
* Flashback Drop (Recycle Bin)
* Flashback Database

How to Configure the flashback

In mount stage only we can enable the flashback technologies and  disable

SQL>alter database flashback on;         —–Enable

SQL>alter database flashback off;        ——–Disable

**Flashback Query**

You can perform a Flashback Query using a SELECT statement with an AS OF clause. You can use a Flashback Query to retrieve data as it existed at some time in the past. The query explicitly references a past time using a timestamp or SCN. It returns committed data that was current at that point in time.

his example uses a Flashback Query to examine the state of a table at a specified time in the past. Suppose, for instance, that a DBA discovers at 12:30 PM that data for employee JON has been deleted from the employee table, and the DBA knows that at 9:30AM the data for JON was correctly stored in the database. The DBA can use a Flashback Query to examine the contents of the table at 9:30, to find out what data has been lost. If appropriate, the DBA can then re-insert the lost data in the database.

The following query retrieves the state of the employee record for JOHN at 9:30AM, April 4, 2006:

2722440854

sql>select \* from  employee as of timestamp to\_timestamp(‘

2006-04-04 09:30:00′ , ‘YYYY-MM-DD HH:MI:SS ‘) where name=’JHON’;

this updates restores Jhon’s information to the employee table

sql> insert into employee (select \* from employee as of timestamp to\_timestamp(‘ 2006-04-04 09:30:00′ , ‘YYYY-MM-DD HH:MI:SS ‘) where name=’JHON’);

**Flashback Version Query**

Not only can the DBA run a manual analysis, but this is a powerful tool for the application’s developer as well. You can build customized applications for auditing purposes. Now everyone really is accountable for his or her actions. Various elements for this are shown below

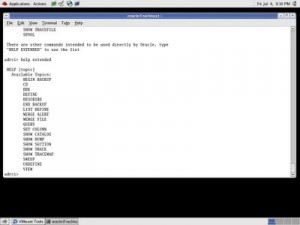
**Versions\_xid**-The transaction id that created this version of the row

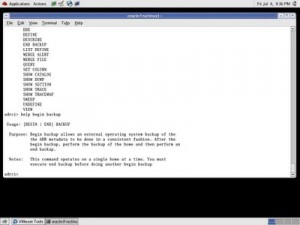
**Versions\_operations**-The action that created this version of the row (such as delete, insert, and update).

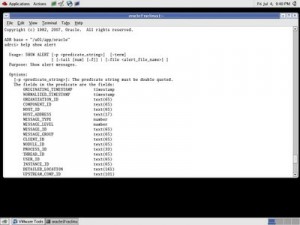
**Versions\_startscn**-The SCN in which this row version first occurred

**Versions\_Endscn**-The SCN in which this row version was changed

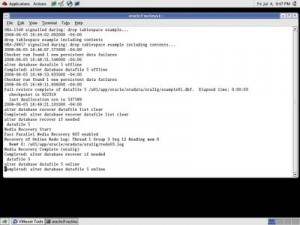
For example: we use the Dept table in Scott schema and update dept 10 to 11, then 12, and then 13 (with a commit after every update). Thus we have done three updates <!–[if gte mso 9]–>For extended help options we can use the *help extended* command from within the ADR command tool and shown below

[](http://dataqapps.com/wp-content/uploads/2011/03/33.jpg)We can get detailed explanations of an option such as the *BEGIN BACKUP* command which will present a series of screens much like that found in the old UNIX man pages.

[](http://dataqapps.com/wp-content/uploads/2011/03/42.jpg)What you should finally realize at this point is that Oracle 11g has taken monitoring to a whole new level over 10g release. In fact, think of it as monitoring on steroids. We will next view an example of how to look at the log files for database monitoring with Oracle 11g on Red Hat Linux

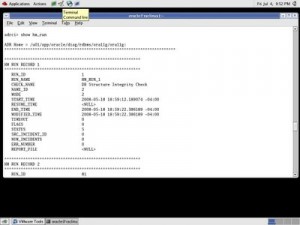
[](http://dataqapps.com/wp-content/uploads/2011/03/52.jpg)Lets scroll down and see more options for the show alert option with ADRCI

[](http://dataqapps.com/wp-content/uploads/2011/03/62.jpg)As you can see, there are now options to review the old style alert.log files as well as the ability to view the log files from the Oracle listener and host. We can also parse the log file to see just a recent activity which is useful for quick monitoring tasks. Lets take a peek at one of the recent log files in the next example.  
We choose option 1 and the alert log file is displayed below:  
By default the log files are stored under the Linux */tmp* directory structure.

[](http://dataqapps.com/wp-content/uploads/2011/03/74.jpg)To exit and return to the main ADR command screen we use the *:q* command within the Linux vi editor.

The ADR command tool also lets us view the new health monitor reports and functions.

For instance, if I wish to look at the most recent health monitor execution tasks then I can simply issue the *show hm\_run* command within the ADR command interface

[](http://dataqapps.com/wp-content/uploads/2011/03/82.jpg)

Also of note with Oracle 11g are several new V$ tables that correspond to the ADR and new health monitoring features:

V$HM\_CHECK  
V$HM\_CHECK\_PARAM  
V$HM\_FINDING  
V$HM\_INFO  
V$HM\_RECOMMENDATION  
V$HM\_RUN

**Online Patching**

Online patching introduced in oracle database 11g. This simplifies administration, because no downtime is needed, and also results in a much quicker turnaround time for installing or de-installing Online Patches.

A regular RDBMS patch can require many minutes to install, since it requires instance shutdown, a relink, and instance startup. On the other hand, you can install an online patch in just a few seconds

Online patches are only applicable for Oracle RDBMS and not any other products. Online patches are currently not supported in Windows, and only supported on the following UNIX platforms for version 11.1.0.7.0 and later:

* Linux x86
* Linux x86\_64
* HP-UX Itanium (HP-UX 11.31 and later)

Solaris SPARC 64-bit (Solaris 10 and later)

**Temporary Tablespace shrink**

**Oracle database 11g introduced the temporary tablesapce shrink feature.**

**SQL>**alter tablespace temp1 shrink space;

This deallocates all the unused segments from the tablespace and shrinks it. After the above operation, you can check the view DBA\_TEMP\_FREE\_SPACE to check how much the allocated space and free space currently is.

SQL> select \* from dba\_temp\_free\_space;

TABLESPACE\_NAME TABLESPACE\_SIZE ALLOCATED\_SPACE FREE\_SPACE

------------------------------ ---------------

TEMP            179306496       179306496      178257920

## Overview of Oracle Flashback Technology

Oracle Flashback Technology is a group of Oracle Database features that that let you view past states of database objects or to return database objects to a previous state without using point-in-time media recovery.

With flashback features, you can do the following:

* Perform queries that return past data
* Perform queries that return metadata that shows a detailed history of changes to the database
* Recover tables or rows to a previous point in time
* Automatically track and archive transactional data changes
* Roll back a transaction and its dependent transactions while the database remains online

Oracle Oracle Flashback features use the Automatic Undo Management (AUM) system to obtain metadata and historical data for transactions. They rely on undo data, which are records of the effects of individual transactions. For example, if a user executes an UPDATE statement to change a salary from 1000 to 1100, then Oracle Database stores the value 1000 in the undo data.

Undo data is persistent and survives a database shutdown. By using flashback features, you can use undo data to query past data or recover from logical damage. Besides using it in flashback features, Oracle Database uses undo data to perform the following actions:

* Roll back active transactions
* Recover terminated transactions by using database or process recovery
* Provide read consistency for SQL queries

Topics:

* [Application Development Features](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#g1025538)
* [Database Administration Features](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#g1025551)

For additional general information about flashback features, see [Oracle Database Concepts](http://docs.oracle.com/cd/B28359_01/server.111/b28318/backrec.htm#CNCPT1439)

### Application Development Features

In application development, you can use the following flashback features to report historical data or undo erroneous changes. (You can also use these features interactively as a database user or administrator.)

Oracle Flashback Query

Use this feature to retrieve data for a time in the past that you specify with the AS OF clause of the SELECT statement. For more information, see [Using Oracle Flashback Query (SELECT AS OF)](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#i1008579).

Oracle Flashback Version Query

Use this feature to retrieve metadata and historical data for a specific time interval (for example, to view all the rows of a table that ever existed during a given time interval). Metadata for each row version includes start and end time, type of change operation, and identity of the transaction that created the row version. To create a Oracle Flashback Version Query, use the VERSIONS BETWEEN clause of the SELECT statement. For more information, see [Using Oracle Flashback Version Query](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#i1019938).

Flashback Transaction Query

Use this feature to retrieve metadata and historical data for a given transaction or for all transactions in a given time interval. To perform a Oracle Flashback Transaction Query, select from the static data dictionary view FLASHBACK\_TRANSACTION\_QUERY. For more information, see [Using Oracle Flashback Transaction Query](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#i1007455).

Typically, you use Oracle Flashback Transaction Query in conjunction with a Oracle Flashback Version Query that provides the transaction IDs for the rows of interest (see [Using Oracle Flashback Transaction Query with Oracle Flashback Version Query](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#BJFBFGCC)).

DBMS\_FLASHBACK Package

Use this feature to set the internal Oracle Database clock to a time in the past so that you can examine data that was current at that time, or to roll back a transaction and its dependent transactions while the database remains online (see [Flashback Transaction](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#BJFIIEAA)). For more information, see [Using DBMS\_FLASHBACK Package](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#i1017341).

Flashback Transaction

Use Flashback Transaction to roll back a transaction and its dependent transactions while the database remains online. This recovery operation uses undo data to create and execute the corresponding compensating transactions that return the affected data to its original state. (Flashback Transaction is part of DBMS\_FLASHBACK package.) For more information, see [Using DBMS\_FLASHBACK Package](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#i1017341).

Flashback Data Archive (Oracle Total Recall)

Use Flashback Data Archive to automatically track and archive both regular queries and Oracle Flashback Query, ensuring SQL-level access to the versions of database objects without getting a snapshot-too-old error. For more information, see [Using Flashback Data Archive (Oracle Total Recall)](http://docs.oracle.com/cd/B28359_01/appdev.111/b28424/adfns_flashback.htm#BJFFDCEH).

|  |
| --- |
| [PDF](http://docs.oracle.com/cd/B19306_01/server.102/b14220.pdf) · [Mobi](http://docs.oracle.com/cd/B19306_01/server.102/B14220-02.mobi) · [ePub](http://docs.oracle.com/cd/B19306_01/server.102/B14220-02.epub) |

**22****Triggers**

This chapter discusses triggers, which are procedures stored in PL/SQL or Java that run (fire) implicitly whenever a table or view is modified or when some user actions or database system actions occur.

This chapter contains the following topics:

* [Introduction to Triggers](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i13313)
* [Parts of a Trigger](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i10234)
* [Types of Triggers](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i2013)
* [Trigger Execution](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i13362)

**Introduction to Triggers**

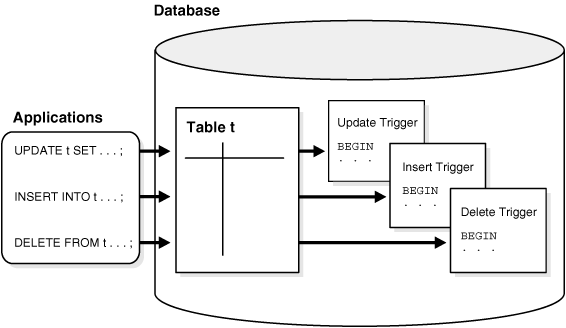
You can write triggers that fire whenever one of the following operations occurs:

1. DML statements (INSERT, UPDATE, DELETE) on a particular table or view, issued by any user
2. DDL statements (CREATE or ALTER primarily) issued either by a particular schema/user or by any schema/user in the database
3. Database events, such as logon/logoff, errors, or startup/shutdown, also issued either by a particular schema/user or by any schema/user in the database

Triggers are similar to stored procedures. A trigger stored in the database can include SQL and PL/SQL or Java statements to run as a unit and can invoke stored procedures. However, procedures and triggers differ in the way that they are invoked. A procedure is explicitly run by a user, application, or trigger. Triggers are implicitly fired by Oracle when a triggering event occurs, no matter which user is connected or which application is being used.

[Figure 22-1](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#BABJAIBB) shows a database application with some SQL statements that implicitly fire several triggers stored in the database. Notice that the database stores triggers separately from their associated tables.

***Figure 22-1 Triggers***

  
[Description of "Figure 22-1 Triggers"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/img_text/cncpt076.htm)

A trigger can also call out to a C procedure, which is useful for computationally intensive operations.

The events that fire a trigger include the following:

* DML statements that modify data in a table (INSERT, UPDATE, or DELETE)
* DDL statements
* System events such as startup, shutdown, and error messages
* User events such as logon and logoff

**Note:**

Oracle Forms can define, store, and run triggers of a different sort. However, do not confuse Oracle Forms triggers with the triggers discussed in this chapter.

**See Also:**

* [Chapter 24, "SQL, PL/SQL, and Java"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/sqlplsql.htm#g35564) for information on the similarities of triggers to stored procedures
* ["The Triggering Event or Statement"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i6555)

**How Triggers Are Used**

Triggers supplement the standard capabilities of Oracle to provide a highly customized database management system. For example, a trigger can restrict DML operations against a table to those issued during regular business hours. You can also use triggers to:

* Automatically generate derived column values
* Prevent invalid transactions
* Enforce complex security authorizations
* Enforce referential integrity across nodes in a distributed database
* Enforce complex business rules
* Provide transparent event logging
* Provide auditing
* Maintain synchronous table replicates
* Gather statistics on table access
* Modify table data when DML statements are issued against views
* Publish information about database events, user events, and SQL statements to subscribing applications

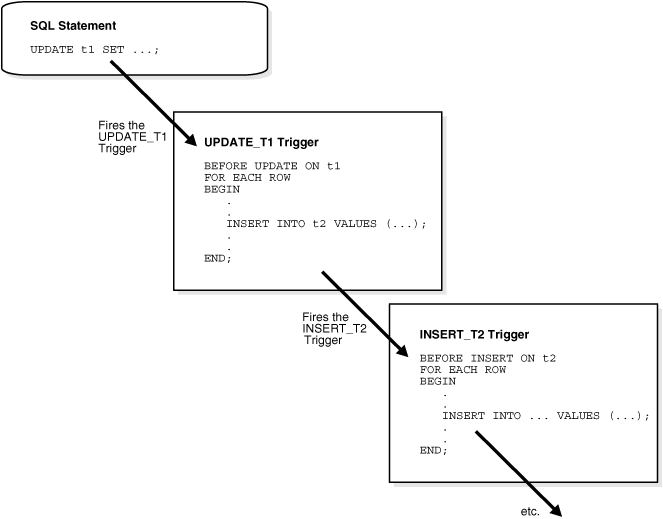
**See Also:**

[*Oracle Database Application Developer's Guide - Fundamentals*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/toc.htm) for examples of trigger uses

**Some Cautionary Notes about Triggers**

Although triggers are useful for customizing a database, use them only when necessary. Excessive use of triggers can result in complex interdependencies, which can be difficult to maintain in a large application. For example, when a trigger fires, a SQL statement within its trigger action potentially can fire other triggers, resulting in **cascading triggers**. This can produce unintended effects. [Figure 22-2](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#BABGDHHG) illustrates cascading triggers.

***Figure 22-2 Cascading Triggers***

  
[Description of "Figure 22-2 Cascading Triggers"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/img_text/cncpt077.htm)

**Triggers Compared with Declarative Integrity Constraints**

You can use both triggers and integrity constraints to define and enforce any type of integrity rule. However, Oracle strongly recommends that you use triggers to constrain data input only in the following situations:

* To enforce referential integrity when child and parent tables are on different nodes of a distributed database
* To enforce complex business rules not definable using integrity constraints
* When a required referential integrity rule cannot be enforced using the following integrity constraints:
  + NOT NULL, UNIQUE
  + PRIMARY KEY
  + FOREIGN KEY
  + CHECK
  + DELETE CASCADE
  + DELETE SET NULL

**See Also:**

["How Oracle Enforces Data Integrity"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/data_int.htm#i4773) for more information about integrity constraints

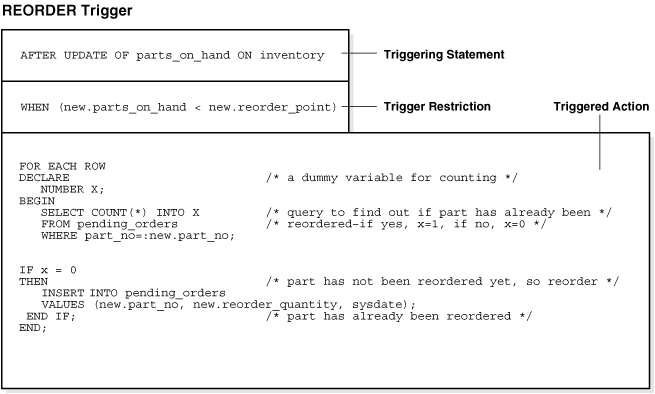
**Parts of a Trigger**

A trigger has three basic parts:

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

[Figure 22-3](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#BABIHCAJ) represents each of these parts of a trigger and is not meant to show exact syntax. The sections that follow explain each part of a trigger in greater detail.

***Figure 22-3 The REORDER Trigger***

  
[Description of "Figure 22-3 The REORDER Trigger"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/img_text/cncpt078.htm)

**The Triggering Event or Statement**

A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire. A triggering event can be one or more of the following:

* An INSERT, UPDATE, or DELETE statement on a specific table (or view, in some cases)
* A CREATE, ALTER, or DROP statement on any schema object
* A database startup or instance shutdown
* A specific error message or any error message
* A user logon or logoff

For example, in [Figure 22-3](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#BABIHCAJ), the triggering statement is:

... UPDATE OF parts\_on\_hand ON inventory ...

This statement means that when the parts\_on\_hand column of a row in the inventory table is updated, fire the trigger. When the triggering event is anUPDATE statement, you can include a column list to identify which columns must be updated to fire the trigger. You cannot specify a column list for INSERT andDELETE statements, because they affect entire rows of information.

A triggering event can specify multiple SQL statements:

... INSERT OR UPDATE OR DELETE OF inventory ...

This part means that when an INSERT, UPDATE, or DELETE statement is issued against the inventory table, fire the trigger. When multiple types of SQL statements can fire a trigger, you can use conditional predicates to detect the type of triggering statement. In this way, you can create a single trigger that runs different code based on the type of statement that fires the trigger.

**Trigger Restriction**

A trigger restriction specifies a Boolean expression that must be true for the trigger to fire. The trigger action is not run if the trigger restriction evaluates tofalse or unknown. In the example, the trigger restriction is:

new.parts\_on\_hand < new.reorder\_point

Consequently, the trigger does not fire unless the number of available parts is less than a present reorder amount.

**Trigger Action**

A trigger action is the procedure (PL/SQL block, Java program, or C callout) that contains the SQL statements and code to be run when the following events occur:

* A triggering statement is issued.
* The trigger restriction evaluates to true.

Like stored procedures, a trigger action can:

* Contain SQL, PL/SQL, or Java statements
* Define PL/SQL language constructs such as variables, constants, cursors, exceptions
* Define Java language constructs
* Call stored procedures

If the triggers are row triggers, the statements in a trigger action have access to column values of the row being processed by the trigger. Correlation names provide access to the old and new values for each column.

**Types of Triggers**

This section describes the different types of triggers:

* [Row Triggers and Statement Triggers](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm" \l "i6047)
* [BEFORE and AFTER Triggers](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i6052)
* [INSTEAD OF Triggers](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i10900)
* [Triggers on System Events and User Events](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i6061)

**Row Triggers and Statement Triggers**

When you define a trigger, you can specify the number of times the trigger action is to be run:

* Once for every row affected by the triggering statement, such as a trigger fired by an UPDATE statement that updates many rows
* Once for the triggering statement, no matter how many rows it affects

**Row Triggers**

A **row trigger** is fired each time the table is affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not run.

Row triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected. For example, [Figure 22-3](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#BABIHCAJ) illustrates a row trigger that uses the values of each row affected by the triggering statement.

**Statement Triggers**

A **statement trigger** is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even if no rows are affected. For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.

Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected. For example, use a statement trigger to:

* Make a complex security check on the current time or user
* Generate a single audit record

**BEFORE and AFTER Triggers**

When defining a trigger, you can specify the **trigger timing***—*whether the trigger action is to be run before or after the triggering statement. BEFORE andAFTER apply to both statement and row triggers.

BEFORE and AFTER triggers fired by DML statements can be defined only on tables, not on views. However, triggers on the base tables of a view are fired if anINSERT, UPDATE, or DELETE statement is issued against the view. BEFORE and AFTER triggers fired by DDL statements can be defined only on the database or a schema, not on particular tables.

**See Also:**

* ["INSTEAD OF Triggers"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i10900)
* ["Triggers on System Events and User Events"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i6061) for information about how BEFORE and AFTER triggers can be used to publish information about DML and DDL statements

**BEFORE Triggers**

BEFORE triggers run the trigger action before the triggering statement is run. This type of trigger is commonly used in the following situations:

* When the trigger action determines whether the triggering statement should be allowed to complete. Using a BEFORE trigger for this purpose, you can eliminate unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the trigger action.
* To derive specific column values before completing a triggering INSERT or UPDATE statement.

**AFTER Triggers**

AFTER triggers run the trigger action after the triggering statement is run.

**Trigger Type Combinations**

Using the options listed previously, you can create four types of row and statement triggers:

* **BEFORE** ***statement*** **trigger**

Before executing the triggering statement, the trigger action is run.

* **BEFORE** *row* **trigger**

Before modifying each row affected by the triggering statement and before checking appropriate integrity constraints, the trigger action is run, if the trigger restriction was not violated.

* **AFTER** ***statement*** **trigger**

After executing the triggering statement and applying any deferred integrity constraints, the trigger action is run.

* **AFTER** ***row*** **trigger**

After modifying each row affected by the triggering statement and possibly applying appropriate integrity constraints, the trigger action is run for the current row provided the trigger restriction was not violated. Unlike BEFORE *row* triggers, AFTER *row* triggers lock rows.

You can have multiple triggers of the same type for the same statement for any given table. For example, you can have two BEFORE *statement* triggers forUPDATE statements on the employees table. Multiple triggers of the same type permit modular installation of applications that have triggers on the same tables. Also, Oracle materialized view logs use AFTER *row* triggers, so you can design your own AFTER *row* trigger in addition to the Oracle-defined AFTER *row*trigger.

You can create as many triggers of the preceding different types as you need for each type of DML statement, (INSERT, UPDATE, or DELETE).

**See Also:**

[*Oracle Database Application Developer's Guide - Fundamentals*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/toc.htm) for examples of trigger applications

**INSTEAD OF Triggers**

INSTEAD OF triggers provide a transparent way of modifying views that cannot be modified directly through DML statements (INSERT, UPDATE, and DELETE). These triggers are called INSTEAD OF triggers because, unlike other types of triggers, Oracle fires the trigger instead of executing the triggering statement.

You can write normal INSERT, UPDATE, and DELETE statements against the view and the INSTEAD OF trigger is fired to update the underlying tables appropriately. INSTEAD OF triggers are activated for each row of the view that gets modified.

**Modify Views**

Modifying views can have ambiguous results:

* Deleting a row in a view could either mean deleting it from the base table or updating some values so that it is no longer selected by the view.
* Inserting a row in a view could either mean inserting a new row into the base table or updating an existing row so that it is projected by the view.
* Updating a column in a view that involves joins might change the semantics of other columns that are not projected by the view.

Object views present additional problems. For example, a key use of object views is to represent master/detail relationships. This operation inevitably involves joins, but modifying joins is inherently ambiguous.

As a result of these ambiguities, there are many restrictions on which views are modifiable. An INSTEAD OF trigger can be used on object views as well as relational views that are not otherwise modifiable.

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

INSTEAD OF triggers also enable you to modify object view instances on the client-side through OCI. To modify an object materialized by an object view in the client-side object cache and flush it back to the persistent store, you must specify INSTEAD OF triggers, unless the object view is inherently modifiable. However, it is not necessary to define these triggers for just pinning and reading the view object in the object cache.

**See Also:**

* [Chapter 27, "Object Datatypes and Object Views"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/objects.htm#g446482)
* [*Oracle Call Interface Programmer's Guide*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14250/toc.htm)
* [*Oracle Database Application Developer's Guide - Fundamentals*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/toc.htm) for an example of an INSTEAD OF trigger

**Views That Are Not Modifiable**

If the view query contains any of the following constructs, the view is not inherently modifiable and you therefore cannot perform inserts, updates, or deletes on the view:

* Set operators
* Aggregate functions
* GROUP BY, CONNECT BY, or START WITH clauses
* The DISTINCT operator
* Joins (however, some join views are updatable)

If a view contains pseudocolumns or expressions, you can only update the view with an UPDATE statement that does not refer to any of the pseudocolumns or expressions.

**See Also:**

["Updatable Join Views"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/schema.htm#i19228)

**INSTEAD OF Triggers on Nested Tables**

You cannot modify the elements of a nested table column in a view directly with the TABLE clause. However, you can do so by defining an INSTEAD OF trigger on the nested table column of the view. The triggers on the nested tables fire if a nested table element is updated, inserted, or deleted and handle the actual modifications to the underlying tables.

**See Also:**

* [*Oracle Database Application Developer's Guide - Fundamentals*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/toc.htm)
* [*Oracle Database SQL Reference*](http://docs.oracle.com/cd/B19306_01/server.102/b14200/toc.htm) for information on the CREATE TRIGGER statement

**Triggers on System Events and User Events**

You can use triggers to publish information about database events to subscribers. Applications can subscribe to database events just as they subscribe to messages from other applications. These database events can include:

* System events
  + Database startup and shutdown
  + Data Guard role transitions
  + Server error message events
* User events
  + User logon and logoff
  + DDL statements (CREATE, ALTER, and DROP)
  + DML statements (INSERT, DELETE, and UPDATE)

Triggers on system events can be defined at the database level or schema level. The DBMS\_AQ package is one example of using database triggers to perform certain actions. For example, a database shutdown trigger is defined at the database level:

CREATE TRIGGER register\_shutdown

ON DATABASE

SHUTDOWN

BEGIN

...

DBMS\_AQ.ENQUEUE(...);

...

END;

Triggers on DDL statements or logon/logoff events can also be defined at the database level or schema level. Triggers on DML statements can be defined on a table or view. A trigger defined at the database level fires for all users, and a trigger defined at the schema or table level fires only when the triggering event involves that schema or table.

**Event Publication**

Event publication uses the publish-subscribe mechanism of Oracle Streams Advanced Queuing. A **queue** serves as a message repository for subjects of interest to various subscribers. Triggers use the DBMS\_AQ package to enqueue a message when specific system or user events occur.

**See Also:**

* [*Oracle Streams Advanced Queuing User's Guide and Reference*](http://docs.oracle.com/cd/B19306_01/server.102/b14257/toc.htm)
* [*Oracle Database PL/SQL Packages and Types Reference*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/toc.htm)

**Event Attributes**

Each event allows the use of attributes within the trigger text. For example, the database startup and shutdown triggers have attributes for the instance number and the database name, and the logon and logoff triggers have attributes for the user name. You can specify a function with the same name as an attribute when you create a trigger if you want to publish that attribute when the event occurs. The attribute's value is then passed to the function or payload when the trigger fires. For triggers on DML statements, the :OLD column values pass the attribute's value to the :NEW column value.

**System Events**

System events that can fire triggers are related to instance startup and shutdown and error messages. Triggers created on startup and shutdown events have to be associated with the database. Triggers created on error events can be associated with the database or with a schema.

* STARTUP triggers fire when the database is opened by an instance. Their attributes include the system event, instance number, and database name.
* SHUTDOWN triggers fire just before the server starts shutting down an instance. You can use these triggers to make subscribing applications shut down completely when the database shuts down. For abnormal instance shutdown, these triggers cannot be fired. The attributes of SHUTDOWN triggers include the system event, instance number, and database name.
* SERVERERROR triggers fire when a specified error occurs, or when any error occurs if no error number is specified. Their attributes include the system event and error number.
* DB\_ROLE\_CHANGE triggers fire when a role transition (failover or switchover) occurs in a Data Guard configuration. The trigger notifies users when a role transition occurs, so that client connections can be processed on the new primary database and applications can continue to run.

**User Events**

User events that can fire triggers are related to user logon and logoff, DDL statements, and DML statements.

**Triggers on LOGON and LOGOFF Events**

LOGON and LOGOFF triggers can be associated with the database or with a schema. Their attributes include the system event and user name, and they can specify simple conditions on USERID and USERNAME.

* LOGON triggers fire after a successful logon of a user.
* LOGOFF triggers fire at the start of a user logoff.

**Triggers on DDL Statements**

DDL triggers can be associated with the database or with a schema. Their attributes include the system event, the type of schema object, and its name. They can specify simple conditions on the type and name of the schema object, as well as functions like USERID and USERNAME. DDL triggers include the following types of triggers:

* BEFORE CREATE and AFTER CREATE triggers fire when a schema object is created in the database or schema.
* BEFORE ALTER and AFTER ALTER triggers fire when a schema object is altered in the database or schema.
* BEFORE DROP and AFTER DROP triggers fire when a schema object is dropped from the database or schema.

**Triggers on DML Statements**

DML triggers for event publication are associated with a table. They can be either BEFORE or AFTER triggers that fire for each row on which the specified DML operation occurs. You cannot use INSTEAD OF triggers on views to publish events related to DML statements—instead, you can publish events using BEFORE orAFTER triggers for the DML operations on a view's underlying tables that are caused by INSTEAD OF triggers.

The attributes of DML triggers for event publication include the system event and the columns defined by the user in the SELECT list. They can specify simple conditions on the type and name of the schema object, as well as functions (such as UID, USER, USERENV, and SYSDATE), pseudocolumns, and columns. The columns can be prefixed by :OLD and :NEW for old and new values. Triggers on DML statements include the following triggers:

* BEFORE INSERT and AFTER INSERT triggers fire for each row inserted into the table.
* BEFORE UPDATE and AFTER UPDATE triggers fire for each row updated in the table.
* BEFORE DELETE and AFTER DELETE triggers fire for each row deleted from the table.

**See Also:**

* ["Row Triggers"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i8815)
* ["BEFORE and AFTER Triggers"](http://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm#i6052)
* [*Oracle Database Application Developer's Guide - Fundamentals*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/toc.htm) for more information about event publication using triggers on system events and user events

**Trigger Execution**

A trigger is in either of two distinct modes:

| **Trigger Mode** | **Definition** |
| --- | --- |
| Enabled | An enabled trigger runs its trigger action if a triggering statement is issued and the trigger restriction (if any) evaluates to true. |
| Disabled | A disabled trigger does not run its trigger action, even if a triggering statement is issued and the trigger restriction (if any) would evaluate totrue. |

For enabled triggers, Oracle automatically performs the following actions:

* Oracle runs triggers of each type in a planned firing sequence when more than one trigger is fired by a single SQL statement. First, statement level triggers are fired, and then row level triggers are fired.
* Oracle performs integrity constraint checking at a set point in time with respect to the different types of triggers and guarantees that triggers cannot compromise integrity constraints.
* Oracle provides read-consistent views for queries and constraints.
* Oracle manages the dependencies among triggers and schema objects referenced in the code of the trigger action
* Oracle uses two-phase commit if a trigger updates remote tables in a distributed database.
* Oracle fires multiple triggers in an unspecified, random order, if more than one trigger of the same type exists for a given statement; that is, triggers of the same type for the same statement are not guaranteed to fire in any specific order.

**The Execution Model for Triggers and Integrity Constraint Checking**

A single SQL statement can potentially fire up to four types of triggers:

* BEFORE *row* triggers
* BEFORE *statement* triggers
* AFTER *row* triggers
* AFTER *statement* triggers

A triggering statement or a statement within a trigger can cause one or more integrity constraints to be checked. Also, triggers can contain statements that cause other triggers to fire (cascading triggers).

Oracle uses the following execution model to maintain the proper firing sequence of multiple triggers and constraint checking:

1. Run all BEFORE *statement* triggers that apply to the statement.
2. Loop for each row affected by the SQL statement.
   1. Run all BEFORE *row* triggers that apply to the statement.
   2. Lock and change row, and perform integrity constraint checking. (The lock is not released until the transaction is committed.)
   3. Run all AFTER *row* triggers that apply to the statement.
3. Complete deferred integrity constraint checking.
4. Run all AFTER *statement* triggers that apply to the statement.

The definition of the execution model is recursive. For example, a given SQL statement can cause a BEFORE *row* trigger to be fired and an integrity constraint to be checked. That BEFORE *row* trigger, in turn, might perform an update that causes an integrity constraint to be checked and an AFTER *statement* trigger to be fired. The AFTER *statement* trigger causes an integrity constraint to be checked. In this case, the execution model runs the steps recursively, as follows:

Original SQL statement issued.

1. BEFORE *row* triggers fired.
   1. AFTER *statement* triggers fired by UPDATE in BEFORE *row* trigger.

i. Statements of AFTER *statement* triggers run.

ii. Integrity constraint checked on tables changed by AFTER *statement* triggers.

* 1. Statements of BEFORE *row* triggers run.
  2. Integrity constraint checked on tables changed by BEFORE *row* triggers.

1. SQL statement run.
2. Integrity constraint from SQL statement checked.

There are two exceptions to this recursion:

* When a triggering statement modifies one table in a referential constraint (either the primary key or foreign key table), and a triggered statement modifies the other, only the triggering statement will check the integrity constraint. This allows row triggers to enhance referential integrity.
* Statement triggers fired due to DELETE CASCADE and DELETE SET NULL are fired before and after the user DELETE statement, not before and after the individual enforcement statements. This prevents those statement triggers from encountering mutating errors.

An important property of the execution model is that all actions and checks done as a result of a SQL statement must succeed. If an exception is raised within a trigger, and the exception is not explicitly handled, all actions performed as a result of the original SQL statement, including the actions performed by fired triggers, are rolled back. Thus, integrity constraints cannot be compromised by triggers. The execution model takes into account integrity constraints and disallows triggers that violate declarative integrity constraints.

For example, in the previously outlined scenario, suppose that the integrity constraint is violated. As a result of this violation, all changes made by the SQL statement, the fired BEFORE *row* trigger, and the fired AFTER *statement* trigger are rolled back.

**Note:**

Although triggers of different types are fired in a specific order, triggers of the same type for the same statement are not guaranteed to fire in any specific order. For example, all BEFORE *row* triggers for a single UPDATE statement may not always fire in the same order. Design your applications so they do not rely on the firing order of multiple triggers of the same type.

**Data Access for Triggers**

When a trigger is fired, the tables referenced in the trigger action might be currently undergoing changes by SQL statements in other users' transactions. In all cases, the SQL statements run within triggers follow the common rules used for standalone SQL statements. In particular, if an uncommitted transaction has modified values that a trigger being fired either needs to read (query) or write (update), then the SQL statements in the body of the trigger being fired use the following guidelines:

* Queries see the current read-consistent materialized view of referenced tables and any data changed within the same transaction.
* Updates wait for existing data locks to be released before proceeding.

**Storage of PL/SQL Triggers**

Oracle stores PL/SQL triggers in compiled form, just like stored procedures. When a CREATE TRIGGER statement commits, the compiled PL/SQL code, called P code (for pseudocode), is stored in the database and the source code of the trigger is flushed from the shared pool.

**See Also:**

[*Oracle Database PL/SQL User's Guide and Reference*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/toc.htm) for more information about compiling and storing PL/SQL code

**Execution of Triggers**

Oracle runs a trigger internally using the same steps used for procedure execution. The only subtle difference is that a user has the right to fire a trigger if he or she has the privilege to run the triggering statement. Other than this, triggers are validated and run the same way as stored procedures.

**See Also:**

[*Oracle Database PL/SQL User's Guide and Reference*](http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/toc.htm) for more information about stored procedures

**Dependency Maintenance for Triggers**

Like procedures, triggers depend on referenced objects. Oracle automatically manages the dependencies of a trigger on the schema objects referenced in its trigger action. The dependency issues for triggers are the same as those for stored procedures. Triggers are treated like stored procedures. They are inserted into the data dictionary.