The basic form of an SQL query is:

SELECT [DISTINCT] {*| column_name (, column_name,…)} FROM table_name [alias] (, table_name,…) [WHERE condition] [GROUP BY column_list] [HAVING condition] [ORDER BY column_list].

- SELECT specifies which columns are to appear in the output DISTINCT eliminates duplicate
- FROM specifies the tables to be used
- WHERE filters the rows according to the condition The where condition is a boolean combination (using AND, OR, and NOT) of conditions of the form expression op expression where op is one of the comparison operators (<=, =, <>, >=, >)
- GROUP BY forms groups of rows with the same column value
- HAVING filters the group
- ORDER BY sorts the order of the output

Set Operations:

- Union,
- Except(minus)
- Intersect

student1

sname	Account
Aijay	A1
Vijay	A2
Ram	A3

student2

sname	loan
Vishal	L1
Ram	L2

Union(U)**:-** it is the binary operation between the two relations r and s. denoted by r U s . It is the union of set of tuples of the two relations. Duplicate tuples are automatically removed from the result. A tuple will appear in r Us if it exists in r or in s or both

For U to be possible, r and s must be compatible

a) r and s must be of same degree i.e. they must have same no of attributes

b) For all i, the domain of i th attribute of r must be same as the domain of the i th attribute of s.

Query:-. Get the names of those students who have either account or loan or both at the bank

SQL: select sname from student1 union select sname from student2;

Result:

Sname
Aijay
Vijay
Ram
Vishal

Except(minus)(-):. The set difference operation (r -s) between two relations r and s produced a relation with tuples which are in r but not there in s. To possible r-s, r and s must be compatible

Cardinality of r-s = cardinality (r) – cardinality (r∩ s)

Query:. Get the names of those students who have account in the bank but do not have loan

SQL: select sname from student1 minus select sname from student2;

Result:

Sname
Aijay
Vijay

Intersect(∩): This operation r∩ s between the relations r and s produced a relation with tuples which are there in r as well as s. For is to be possible, relations r and s must be compatible

Query :get the names of those students who have account as well as loan

SQL: select sname from student1 intersect select sname from student2;

Result:

Sname
Ram

Data types:-

- Each value in oracle is maintained by a data type.
- The value of one data type if different from other data type.
- The data type defines the domain of values that each column can caontain

Character data types:-

These store character data

Different character data types are

1. char
2. varchar2

1. **char data type:-** it specifies fixed length character string. Size should be specified. If the data is less then original specific size, blank spaces are applied. The default length is 1byte and maximum length is 200 bytes.
   Ex:- char(10);
2. **varchar2 data types:-** it specifies the variable length character string. It occupies only that space for which the data is supplied. The maximum size is 1byte and the maximum size is 400 bytes.
   Ex:- varchar2(10);

Number data types:-

1. number(p,s)
   p→ precision, range is 1 to 38
   s→ scale, range is -84 to 127
   ex:- number(8,3);
2. **float:-** it is used to specify floating point values. It specifies decimal precision 38.
3. **Long data types:-** these are used to store very large text strings. A single table can have only one long column.
4. **Data and time data type:-**
   1. **Date:-** it is used to store date information. The default date format in oracle is DD-MM-YYY
   2. **Time:-** this is used to store time information. It has atleast 8 positions embedded in single quotes. 'HH:MM:SS'
      Ex:- :- 11:07:05
   3. **Time stamp:-** it includes both time and date along with minimum 6digits representing decimal fraction of seconds.
      The format is 'DD-MM-YYYY HH:MM:SS'
      Ex:- '31-05-1950 01:02:05 123456'
5. **Large object data types:-** these can store large and un structural data like text, image, video and special data.
   the max size is upto 4 GB

the types are
1. BLOB(binary large object)
2. CLOB(character large object)
   Maximum size is 4 GB

Note:- in place of along data type which is deprecated, now a days we are using LOB data datatype.

6. **Raw and long raw data types:-** these are used to store binary data or byte strings. These are variable length data types. They are mostly used to store graphics, sound documents etc.

Types of SQL commands:-

1. **Data definition language(DDL):-** it is used to define the database schema. The commands used under this languages are:-
   1. create
   2. after
   3. drop

syntaxes and examples:-

syntax:-

1. create table <table-name>(col1 datatype[size] constraints list, col2 datatype[size] constraints list,----------);
   ex:- create table student (sid number(4)primary key, sname varchar2(10)not null);
2. **after:-** used to alter the table definition

a) **alter with add option:-**
   syntax:-
   alter table <table-name> add <col-name> datatype[size]
   ex:- alter table dept add loc1 varchar2(10);

b) **alter with drop option:-**
   syntax:- alter table <table-name>drop column <col-name>;
   ex:- alter table dept drop column loc1;

c) **after with modify option:-**
   syntax:- alter table <table-name>modify <col-name>datatype[size];
   ex:- alter table dept modify loc varchar2(10);

d) **alter with rename option:-**
   syntax:- alter table<table-name> rename column<old col name> to <new column name>
   ex:- alter table rename column loc to location

3. **drop:-** used to drop a database table permanently.
   Syntax:- drop table<table-name>
   Ex:- drop table dept;
2. **Data manipulation language(DML):-** these are used to manipulate the data in the databases. The commands used in the languages are

1. Insert
2. Update
3. Delete

Syntaxes and examples:-

1. **Insert:-** used to insert rows into a table
   Syntax:- insert into <table-name>(col1,col2,---,coln)values(val1,val2,----,valn);
   Ex:- insert into dept(deptno,dname,loc)values(50,'xyz','hyd');
2. **Update:-** used to update rows of table
   Syntax:-update <table-name>set <column-name>=<value>where <col-name>=<value>
   Ex:- update dept set dname='pqr' where deptno=50;
3. **Delete:-** used to delete rows from a table
   Syntax:- delete from <table-name> where <col-name>=<value>;
   Ex:- delete from dept where deptno=50;

3. Data query language(DQL):- it is used to extract data from database tables. The command comes under the language is

1. Select

Syntax:-

Select <col-list>,<group functions>from <table-name> where <condition>groupby<column>having<group condition>orderby<column-name>

Ex:-

Select deptno, sum(sal), max(sal), min(sal), avg(sal) from emp

Where job='clerk' group by deptno having avg(sal)>1000 order by deptno;

4. **Data control languages:-** these commands control the user access to the database. The commands comes under these languages are

1. Grant
2. Revoke

Grant:-used to grant the permissions to the user on the db tables.

Syntax:- grant <priviliges-name>ON <object name>to<user-name>

Ex:- grant select, insert, delete on emp to operators;

Revoke:- used to take back the permissions from the user.

Syntax:- revoke<priviliges-name>ON <object name>from<user-name>

Ex:- revoke insert, delete on emp from operators;

5. **Data administrative language(DAL):-**these commands are used for audit, the commands are

1. Start audit;
2. Sleep audit;

6. **Transaction control language(TCL):-** these commands are used to control the transactions
   1. Commit
   2. Rollback
   3. Savepoint

Syntaxes:-

1. **commit;**
2. **rollback;**
3. **rollback to<save point name>;**

Relational set operators:-

1. **union:-** merges the output of two or more queries into a single set of rows and columns.
   Ex:-select job from emp where deptno=10 union select job from emp where deptno=30;
2. **union all:-** union suppresses the duplicates where as union all will also display duplicates.
   Ex:- select empno, ename from emp where deptno=10 union all select empno, ename from emp where deptno=30;
3. **intersect:-** this operator returns the common rows that are common between two queries.
   Ex:- select job from emp where deptno=20 intersect select job from emp where deptno=30;
4. **minus:-** this returns the rows unique to the first query.
   Ex:- select job from emp where deptno=20 minus select job from emp where deptno=10;

Sub queries/Nested queries/ sub select/inner select:-

- It is the concept of placing one query inside the other query
- Inner or sub query returns a value which is used by the outer query.

Types of subqueries:-

1. Single row sub query
2. Multiple row subquery
3. Multiple column subquery
4. Inline subquery
5. Correlated subquery

1) **Single row subquery:-** these return sonly one row from inner select statement. It uses only single row operator. (>,=,<,<=,>=)

Ex:- select ename, sal, job from emp where sal>(select sal from emp where empno=7566);

2) **Multiple row subquery:-**the subqueries return more than one row are called multiple row sub queries. In this case multiple row operators are used.
   a) **IN→**equal to any number of list.
   b) **ANY→**compares value to each returned by subquery.
      I. **<any→**less than the max value
      II. **>any→**greater than min value
   c) **ALL→**compares value the each value returned by subquery
      <any→less than the max value
      >any→greater than min value
      Ex:- select empno, ename, job, from emp where sal<any(select sal from emp whee job='clerk');
3) **Multiple column subquery:-** in this the subquery return multiple columns.
   Ex:- select ename, deptno from emp where(empno.deptno)in(select empno, deptno from emp where sal>1200);
4) **Inline subquery:-**in this the subquery may be applied in select list and inform clause.
   Ex:-
   Select ename, sal, deptno from(select ename, sal, deptno, mgr, hiredate from emp);
5) **correlatedsubquery:-** in this the information of outer select participate as a condition in inner select.
   Ex:- select deptno, ename, sal, from emp x where sal>(select avg(sal)from emp where x.deptno=deptno)orer by deptno;
   - here first outer query is executed and it pass the value of deptno to the inner query then the inner query executed and give the result to the outer query.

Aggregate functions:- these are used to display the aggregated data from group of values.

1. **max():-** used to get max value from the list of values

ex:- select max(sal) from emp;

output:-

MAX(sal)

10000

2. **min():-** used to get min value from the group of values.

Ex: select min(sal) from emp;

Output:-

MIN(sal)

800

3. **sum():-** used to get the total sum of vaues

ex:- select sum(sal)from emp;

output:-

SUM(sal)

37525

4. **avg():-** used to get the average value of the given values.

Ex:- select avg(sal) from emp;

Output:-

AVG(sal)

2680.35714

5. **count():-** used to count the list of values

ex:- selsct count(sal)from emp;

output:-

COUNT(sal)

14

Order by clause:- it is used sort the values of column in ascending or descending oreder.

Ex:- select ename from emp order by ename;

Output:-

ENAME

Adems

Allen

Blake

Clerk

Ford

James

Jhons

King

Martin

Miller

Scort

Smith

Turner

Ward

14 rows are selected

By default order by clause sort the values in ascending order

Ex:- select sal from emp order by sal desc;

Output:-

SAL

10000

5000

3000

3000

2975

2850

2450

1600

1300

1250

1250

1100

950

800

14 rows selected

Group by clause:-this is used to display the group wise data i.e. department, job wise etc….

Ex:-

Select deptno, count(*) from emp group by deptno;

output:-

DEPTNO	COUNT(*)
30	6
20	5
10	3

Select deptno, min(sal)from emp group by deptno;

Output:-

DEPTNO	MIN(sal)
30	950
20	800

101300

Having clause:- it is used to define conditions on a grouping column. Where clause defines conditions on the selected columns where has the having clause places conditions on groups created by the group by clause.

Ex:- select deptno, min(sal) from emp group by deptno having min(sal)>800;

Output:-

DEPTNO	MIN(sal)
30	950
10	1300

Ex:- select job, min(sal)from emp group by job having min(sal)>800;

Output:-

JOB	MIN(SAL)
Salesman	1250
President	5000
Manager	2450
analyst	3000

ex:- select job, sum(sal), avg(sal), min(sal), max(sal) from emp where deptno=20 group by job having avg(sal)>1000 order by job;

output:-

JOB	SUM(SAL)	AVG(SAL)	MIN(SAL)	MAX(SAL)
Analyst	6000	3000	3000	3000
Manager	2976	2975	2975	2975

- order of statements executing(placing) in an sql query

select, from, where, group by, having, order by

Importance of null values:-

- A 'NULL' is a term used to represent a missing value.
- Null is undefined, unknown, unavailable and it is not equal to zero or a space.
- The regular operators like +, -, *, %, =, <, >, <=, >=, will be fail with null values.

Why should we avoid placing of null values into DB:-

- All arithmetic and comparison operators will fail with null values i.e. if we add a column to the null value column then the result will become null only.
- A null will occupy large space in a databases.
- We use two operators with the null values.
    1. is null
    2. is not null

Ex:- select ename from emp where comm is null

We have the following functions to handle with the null values.

1. nvl()
2. nvl2()
3. coalesce()

nvl(expr/column, default value):-takes two arguments

this function returns first argument value if the first argument is not null, if it is null then it return the 2nd argument value.

Ex:- select nvl(comm,0) from emp;

In the output of above query if comm is null then the default value (2nd argument) i.e. 0 will be displayed, if comm is not null then that value is displayed as it is.

Output:-

nvl(comm,0)

0

300

500

0

The actual comm column from emp is

Comm

300

500

1400

-

-

-

null values

1400

0

0

0

0

0

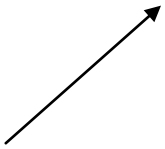
0

0

0

0

14 rows selected.



nvl2(expr1/column1, expr2/column2, expr3/column2):-

if the first argument value is not null then this function returns the value of 2nd argument, if the first argument value is null thoutpien this function returns the value of 3rd argument

ex:- select nvl2(comm, sal+com,sal) from emp;

output:-

nvl2(comm, sal+com, sal)

800

1900

1750

2975

2650

2850

2450

3000

5000

10000

1100

950

3000

1300

14 rows selected.

Coalesce(expr1/column1, expr2/column2,--------- expr n/column n):-

It takes 'n' arguments.

This function accepts two or more arguments and returns the first not null value in the list. If all the arguments contain null values then this function returns a null value.

Ex:- select coalesce(20,30,null) from dual;

Output:-

20→first not null value.

select coalesce(null,null,30) from dual;

Output:-

30→ first not null value in the argument list.

Joins:- joins is a query that combines rows from two or more tables or views

- if some column name appears more than one table, the name must be prefixed with table name.
- To join n tables together, we need a minimum of n-1 conditions.

Join types:-

1. Simple join/equi join/inner join
2. Non equi join
3. Self join
4. Cartesian product
5. Natural join
6. Outer join

1. **Simple join:-** in this the join condition containing equality operator.

Ex:- select E.empno, e.ename, D.deptno, D.dname, from emp E, dept D where E.deptno=D.deptno;

Join condition

2. **Non equi join:-** in this no column of one table will not corresponds to any column of other table means the domain of no column in a table is not same as the domain of other table.
   - in this type no equal operator based on common columns in the join condition.

Ex:- select E.ename, E.sal, S.grade from emp E, salgrade S where E.sal between S.losal and S.hisal;

3. **self join:-** it is a join of table itself.
   Ex:- select E1.ename "employee name",
   E2.anme "managers name",
   From emp E1, emp E2 where
   E1.mgr=E2.empno;
4. **Cartesian product:-** the Cartesian product is a join without a join condition consider the following relations

student1

sname	account
Ajay	A1
Vijay	A2
ram	A3

Student2

Sname	loan
Vishal	L1
ram	L2

Student1*stuednt2

Student1.sname	account	Student2.sname	loan
Ajay	A1	Vishal	L1
Ajay	A1	Ram	L2
Vijay	A2	Vishal	L1
Vijay	A2	Ram	L2
Ram	A3	Vishal	L1
ram	A3	ram	L2

SQL ex:- select * from student1,student2;

5. **Natural join:-** natural join is equal to the following sequence of operators
   a) Cartesian product of two relations
   b) Select the tuples based on the common column(attributes) of the two relations.
   c) Removing the duplicate attributes from the resultant relation

natural join of student1*student2

student1*student2

sname	account	Loan
ram	A3	L2

SQL ex:- select * from student1 natural join student2;

6. **Outer join:-** outer join extends the result of natural join. natural join will give the tuples only based on the common attributes of the two relations. The information of the other tuples will not be given by the natural join. it is possible to get such tuples information by using outer join.

There are three types of outer joins:-

1. Left outer join(L.O.J)
2. Right outer join(R.O.J)
3. Full outer join(F.O.J)

L.O.J:- it gives the full information of left side table (1st table)along with the natural join.

L.O.J

sname	**account**	**sname**	**loan**
Ram	A3	Ram	L2
Ajay	A1	Ajay	Null
vijay	A2	ajay	Null

R.O.J:- it gives the full information about right side table (2nd) along with the natural join output

ROJ

sname	account	sname	loan
Ram	A3	Ram	L2
vishal	Null	vishal	L1

Full outer join:- it will give the full information about lest and right side tables along with natural join.

FOJ

sname	account	sname	loan
Ram	A3	Ram	L2
Ajay	A1	Ajay	Null
Vijay	A2	Vijay	Null
vishal	null	vishal	L1

SQL queries:-

- Select * from student1 left outer join students on stydent1.sname=student2.sname;
- Select * from student1 right outer join students on stydent1.sname=student2.sname;

- Select * from student1 full outer join students on stydent1.sname=student2.sname;

Complex integrity constraints:

We have discussed the integrity constraints in the unit-II but we can make them more complex by defining a table with two or more foreign keys in a table by referring primary keys of different tables as shown below

SQL> create table sailors(sid number(2)primary key,sname varchar2(10),rating num
ber(2),age float);

Table created.

SQL> desc sailors;

Name	Null?	Type
SID	NOT NULL	NUMBER(2)
SNAME		VARCHAR2(10)
RATING		NUMBER(2)
AGE		FLOAT(126)

SQL> create table boats(bid number(3)primary key,bname varchar2(10),color varcha
r2(10));

Table created.

SQL> desc boats;

Name	Null?	Type
BID	NOT NULL	NUMBER(3)
BNAME		VARCHAR2(10)
COLOR		VARCHAR2(10)

SQL> create table reserves(sid number(2) references sailors(sid),bid number(3)re
ferences boats(bid),day date);

Table created.

SQL> desc reserves;

Name	Null?	Type

SID	NUMBER(2)
BID	NUMBER(3)
DAY	DATE

Sid and bid in the above table are foreign keys which are referring from the tables sailors and boats.

PL/SQL

Basic Syntax of PL/SQL which is a **block-structured** language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts –

S.No	Sections & Description
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a **NULL command** to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION**. This optional section contains **exception(s)** that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**. Following is the basic structure of a PL/SQL block −

```
DECLARE
  <declarations section>
```

```
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

The 'Hello World' Example

```
DECLARE
  message  varchar2(20):= 'Hello, World!';
BEGIN
  dbms_output.put_line(message);
END;
/
```

The **end;** line signals the end of the PL/SQL block. To run the code from the SQL command line, you may need to type / at the beginning of the first blank line after the last line of the code. When the above code is executed at the SQL prompt, it produces the following result −

```
Hello World

PL/SQL procedure successfully completed.
```

The PL/SQL Identifiers

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

By default, **identifiers are not case-sensitive**. So you can use **integer** or **INTEGER** to represent a numeric value. You cannot use a reserved keyword as an identifier.

The PL/SQL Delimiters

A delimiter is a symbol with a special meaning. Following is the list of delimiters in PL/SQL −

Delimiter	Description
+, -, *, /	Addition, subtraction/negation, multiplication, division

%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator
;	Statement terminator
:=	Assignment operator
=>	Association operator
||	Concatenation operator
**	Exponentiation operator
<<, >>	Label delimiter (begin and end)

/*, */	Multi-line comment delimiter (begin and end)
--	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operators
<>, '=, ~=, ^=	Different versions of NOT EQUAL

The PL/SQL Comments

Program comments are explanatory statements that can be included in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow some form of comments.

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler. The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.

```
DECLARE
  -- variable declaration
  message  varchar2(20):= 'Hello, World!';
BEGIN
  /*
  *  PL/SQL executable statement(s)
  */
  dbms_output.put_line(message);
END;
/
```

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

```
Hello World


PL/SQL procedure successfully completed.
```

PL/SQL Program Units

A PL/SQL unit is any one of the following −

- PL/SQL block
- Function
- Package
- Package body
- Procedure
- Trigger
- Type
- Type body

Triggers:

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Syntax:

CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER | INSTEAD OF }

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

```
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Here,

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

Types of Triggers in Oracle

Triggers can be classified based on the following parameters.

- Classification based on the **timing**
  - **BEFORE** Trigger: It fires before the specified event has occurred.
  - **AFTER** Trigger: It fires after the specified event has occurred.
  - **INSTEAD OF** Trigger: "INSTEAD OF trigger" is the special type of trigger. It is used only in DML triggers. It is used when any DML event is going to occur on the complex view.
- Classification based on the **level**
  - **STATEMENT level** Trigger: It fires one time for the specified event statement.
  - **ROW level** Trigger: It fires for each record that got affected in the specified event. (only for DML)
- Classification based on the **Event**
  - **DML** Trigger: It fires when the DML event is specified (INSERT/UPDATE/DELETE)
  - **DDL** Trigger: It fires when the DDL event is specified (CREATE/ALTER)
  - **DATABASE** Trigger: It fires when the database event is specified (LOGON/LOGOFF/STARTUP/SHUTDOWN)

:NEW and :OLD Clause

In a row level trigger, the trigger fires for each related row. And sometimes it is required to know the value before and after the DML statement.

Oracle has provided two clauses in the RECORD-level trigger to hold these values. We can use these clauses to refer to the old and new values inside the trigger body.

- :NEW – It holds a new value for the columns of the base table/view during the trigger execution
- :OLD – It holds old value of the columns of the base table/view during the trigger execution

This clause should be used based on the DML event. Below table will specify which clause is valid for which DML statement (INSERT/UPDATE/DELETE).

	INSERT	UPDATE	DELETE
:NEW	VALID	VALID	INVALID. There is new value in delete case.
:OLD	INVALID. There is no old value in insert case	VALID	VALID

Examples on Triggers:

client_master

```
 CLIENTNO NAME  BAL_DUE ADDRESS    CITY
---------- ---------- ---------- ---------- ----------
         1 abc              300 sacet      vetapalem
         2 xyz              500 saec       chirala
         3 pqr              700 sacet      vetapalem
```

audit_client

SQL> create table audit_client(clientno number,name varchar2(10),bal_due number(10,2),operation varchar2(10),userid varchar2(10),odate date);

Creating BEFORE UPDATE Trigger:

```
create or replace trigger audit_trail before update on client_master
  for each row
  declare
  oper varchar2(10);
  clientno      client_master.clientno%type;
  name          client_master.name%type;
  bal_due       client_master.bal_due%type;
  begin
  if updating then
 oper:='update';
 end if;
 if deleting then
```

```
oper:='delete';
end if;
clientno:=:old.clientno;
name:=:old.name;
bal_due:=:old.bal_due;
insert into audit_client values(clientno,name,bal_due,oper,user,sysdate);
end;
/
```

Trigger created.

```
SQL> select * from client_master;

 CLIENTNO NAME  BAL_DUE ADDRESS    CITY
---------- ---------- ---------- ---------- ----------
         1 abc              300 sacet      vetapalem
         2 xyz              500 saec       chirala
         3 pqr              700 sacet      vetapalem

SQL> update client_master set bal_due=bal_due+100
  2  where clientno=2;

1 row updated.


SQL> select *from client_master;

 CLIENTNO NAME        BAL_DUE ADDRESS    CITY
---------- ---------- ---------- ---------- ----------
       1 abc          300 sacet     vetapalem
       2 xyz          600 sace      chirala
       3 pqr          700 sacet     vetapalem


SQL> select *from audit_client;

 CLIENTNO NAME        BAL_DUE OPERATION  USERID     ODATE
---------- ---------- ---------- ---------- ---------- ---------
       2 xyz          500 update     SCOTT      29-JUL-18
```

Creating AFTER DELETE Trigger:

```
SQL> create or replace trigger audit_trail after delete on client_master
```

```
for each row
declare
oper        varchar2(10);
clientno    client_master.clientno%type;
name        client_master.name%type;
bal_due     client_master.bal_due%type;
begin
if updating then
oper:='update';
end if;
if deleting then
oper:='delete';
end if;
clientno:=:old.clientno;
name:=:old.name;
bal_due:=:old.bal_due;
insert into audit_client values(clientno,name,bal_due,oper,user,sysdate);
end;
/
```

Trigger created.

SQL> delete from client_master where clientno=3;

1 row deleted.

SQL> select *from client_master;

CLIENTNO	NAME	BAL_DUE	ADDRESS	CITY
1	abc	300	sacet	vetapalem
2	xyz	600	sace	chirala

SQL> select *from audit_client;

CLIENTNO	NAME	BAL_DUE	OPERATION	USERID	ODATE
2	xyz	500	update	SCOTT	29-JUL-18
3	pqr	700	delete	SCOTT	29-JUL-18

Creating INSTEAD OF Trigger:

```
create or replace trigger instead_of_view
```

```
instead of update on client_master_view
for each row
begin
update audit_client set name=:new.name where clientno=:old.clientno;
end;
/
```

Trigger created.

SQL> select *from client_master_view;

```
 CLIENTNO NAME        BAL_DUE ADDRESS    CITY
---------- ---------- ---------- ---------- ----------
       1 yuga           300 sacet     vetapalem
       2 xyz            600 sace      chirala
```

SQL> select *from audit_client;

```
 CLIENTNO NAME        BAL_DUE OPERATION  USERID     ODATE
---------- ---------- ---------- ---------- ---------- ---------
       2 xyz            500 update    SCOTT     29-JUL-18
       3 pqr            700 delete    SCOTT     29-JUL-18
```

SQL> update client_master_view set name='ffff' where clientno=2;

1 row updated.

SQL> select *from client_master_view;

```
 CLIENTNO NAME        BAL_DUE ADDRESS    CITY
---------- ---------- ---------- ---------- ----------
       1 yuga           300 sacet     vetapalem
       2 xyz            600 sace      chirala
```

SQL> select *from audit_client;

```
 CLIENTNO NAME        BAL_DUE OPERATION  USERID     ODATE
---------- ---------- ---------- ---------- ---------- ---------
       2 ffff           500 update    SCOTT     29-JUL-18
       3 pqr            700 delete    SCOTT     29-JUL-18
```

Try to create a trigger using FOR EACH STATEMENT (not in oracle)

```
create or replace trigger for_each_statement
after insert or  update or delete on client_master
   for each statement
   begin
   delete from aa;
 end;
/
```

It will give the following error:

```
   for each statement
          *
ERROR at line 3:
ORA-01912: ROW keyword expected
```

If we use ROW in place of STATEMENT then

```
 1  create or replace trigger for_each_statement
 2  after insert or  update or delete on client_master
 3      for each row
 4      begin
 5      delete from aa;
 6*     end;
SQL> /

Trigger created.

SQL> select *from aa;

         X Y
---------- ----------
        12 jjj
           joj
           joj
           joj

SQL> select *from client_master;

 CLIENTNO NAME        BAL_DUE ADDRESS    CITY
---------- ---------- ---------- ---------- ----------
         1 yuga            300 sacet      vetapalem
         2 xyz             600 sace       chirala
```

SQL> update client_master set name='hhhh' where clientno=1;

1 row updated.

SQL> select *from client_master;

CLIENTNO	NAME	BAL_DUE	ADDRESS	CITY
1	hhhh	300	sacet	vetapalem
2	xyz	600	sace	chirala

SQL> select *from aa;

no rows selected

Active databases:

An active database is a database that includes an event-driven architecture (often in the form of ECA rules) which can respond to conditions both inside and outside the database. Possible uses include security monitoring, alerting, statistics gathering and authorization