INDEX

1. SQL Commands
2. Constraints
3. Operators
4. Functions
5. Joins
6. Sub Queries and Correlated Sub query
7. View
8. Procedure and Function
9. Cursor
10. Trigger

**INTRODUCTION**

[**SQL**](https://en.wikipedia.org/wiki/SQL), which is an abbreviation for **Structured Query Language**, is a language to request data from a database, to add, update, or remove data within a database, or to manipulate the metadata of the database.

Generally procedural query languages requires the details of the operations to be specified, such as opening and closing tables, loading and searching indexes and writing data to file systems.

SQL is a nonprocedural query language in which the expected results is given without the specific details about how to accomplish the task.

**SQL COMMANDS**

SQL contains different types of commands

**DDL** (Data Definition Language) Commands

**DML**(Data Manipulation Language) Commands

**TCL**(Transaction Control Language) Commands

**DCL**(Data Control Language) Commands

The Data Definition Languages (**DDL**) Commands are used to create a table structure, modify the table structure and delete the table structure.

**I DDL Commands:**

1. **CREATE TABLE**

The CREATE TABLE command is used to create a new table

**Syntax: Create table tablename (column\_name1 data\_ type (size) constraint, column\_name2 data\_ type (size) constraints, …);**

**Example:**

create table student (sname varchar2(20) , rollno number(10) ,dob date );

create table student (sname varchar2(20) not null, rollno number(10) not null,dob date );

1. **The ALTER TABLE Statement**

The ALTER TABLE statement is used to modify structure of an existing table. There are three different syntaxes

1. **ALTER TABLE ADD**

This command is used to add a new column in to an existing table syntax:

**ALTER TABLE table\_name ADD( column\_name data\_type(size) , column\_name data\_type,…) ;**

**alter table student add(address varchar2(20),phone\_no number(10));**

1. **ALTER TABLE MODIFY**

This command is used to change the data type and size of an existing column in to an existing table

syntax:

**ALTER TABLE table\_name MODIFY (old\_column\_name new\_data\_type(new\_size), old\_column\_name new\_data\_type(new\_size),….);**

**alter table student modify (address varchar2(40));**

1. **ALTER TABLE DROP**

To delete a column in a table, use the following syntax

**ALTER TABLE table\_name DROP COLUMN column\_name1,… ;**

**alter table student drop phone\_no;**

1. **Drop:**

This command is used to delete the structure of the table as well as records in the relation.

**Syntax : drop table table\_name;**

Ex: drop table student;

**4.Create AS select:**

This is used to create a new relation or table from an existing relation or table

Syntax:

**create table new\_table\_name (column\_1, column\_2,…..column\_n) as select column\_1, column\_2,……column\_n from old\_table\_name;**

Ex: create table student2(sno,sname,saddress) AS select sno, name, address from student;

**5.RENAME**

**RENAME** command is used to rename the table.

SQL> Rename student to student1;

**6.TRUNCATING TABLE:**

Truncate command will delete all the records permanently in a specified table but structure of the table will not be deleted.

* Syntax
  + **TRUNCATE TABLE TableName;**

**Eg:**

**TRUNCATE TABLE student;**

**II Commands in DML:**

1. **The SQL INSERT INTO Statement**

The INSERT INTO statement is used to insert new records in a table. There are different forms of insert commands are used to insert a records into a table.

**Syntax**

The first form of insert is used to insert values for all columns in a table.

**INSERT INTO *table\_name* VALUES (*value1*,*value2*,*value3*,...);**

The second form of insert is used to insert values for some columns of a record in a table.

**INSERT INTO *table\_name* (*column1*,*column2*,*column3*,...)  
VALUES (*value1*,*value2*,*value3*,...);**

1. **Select:**

The select command is used to retrieve records from the tables.

This type is used to display the set of fields or all the fields of the selected or all records in a table.

1. **Syntax: Select \* from <*table\_name*> [WHERE condition];**

Ex: select \* from student;

1. **Syntax: select set of fields from table\_name [where condition];**

Ex: **select** sno,sname,address **from** student **where** address=’BVRM’;

1. **Update:**

The update command is used to update the content of a table.

**Syntax:**

Syntax 1

* + UPDATE TableName SET ColumnName1=Expression1,

ColumnName2=Expression2,….;

Syntax 2

* + UPDATE TableName SET ColumnName1=Expression1, ColumnName2=Expression2,… WHERE Condition;

Ex: **update** student **SET** total = sub1+sub2+sub3;

**UPDATE** student **SET** total = sub1+sub2+sub3 **where** sno = 10;

1. **Delete:**

The delete command is used to delete all the records or selected records in a table. But the structure of the table remains.

**Syntax:**

Syntax 1

* + DELETE FROM TableName;
  + **Delete** from student;

Syntax 2

* + DELETE FROM TableName WHERE Condition;
  + **Delete** from student **where** address=’BVRM’;

**DCL Commands:**

Two types of DCL commands are

1. Grant
2. Revoke
3. **Grant**

SQL Grant command is used to grant a privileges on the database objects to the users.

The syntax for the GRANT command is:

**GRANT privilege\_name ON object\_name TO user\_name ;**

Here, privilege\_name: is the access right or privilege granted to the user.

object\_name: is the name of the database object like table, view etc.,

user\_name: is the name of the user

(a) **GRANT SELECT ON employee TO user1 ;**

This command grants a SELECT permission on employee table to user1.

|  |
| --- |
| **(b) GRANT SELECT, INSERT, UPDATE, DELETE ON employee TO SCOTT;** |

The above statement will GRANT SELECT, INSERT, UPDATE and DELETE privileges to SCOTT user on employee table.

1. **REVOKE**

The revoke command removes user privileges to the database objects.

•  The syntax for the REVOKE command is:

**REVOKE privilege\_name ON object\_name FROM User\_name ;**

For Example:

**REVOKE SELECT ON employee FROM user1;**

This command will revoke a SELECT privilege on employee table from user1.

Suppose we want to revoke SELECT, INSERT, UPDATE and DELETE privileges from ‘SCOTT’ user on employee table, then we have to execute the following statement;

**REVOKE SELECT,INSERT,UPDATE,DELETE ON employee FROM SCOTT;**

**TCL Commands**

Transaction Control (TCL) statements are used to manage the changes made by DML statements.

* COMMIT - To make the changes permanent in the database
* SAVEPOINT - saving  point in a transaction to which you can later roll back
* ROLLBACK - Restore database to original since the last COMMIT

**Constraint**

The constraint is a mechanism used by oracle to prevent invalid data into the table.

1. **Not null:**

we know that by default all the columns in a table allow null values. When a **not null** constraint is enforced through either on a column or a set of columns in a table, it will not allow null values.

**Adding a constraint at the time of table creation:**

create table stu1(sno number(5) constraint nn1 not null, name char(10) not null, address varchar2(10) not null);

Dropping a Constraint

**Alter table** stu1 **drop** **constraint** nn1;

1. **Unique:**

The unique key constraint is used to prevent the duplication of values with in the rows of a specified column or a set of columns in a table. Columns defined with this constraint can also allow null values.

create table stu2(sno number(5) constraint un1 unique, name char(20) not null, address varchar2(10));

alter table stu2 add constraint un2 unique(address);

alter table stu2 drop constraint un2;

1. **Primary key:**

The primary key constraint avoids duplication of rows and does not allow null values.

A table can have only one primary key. If a primary key constraint is assigned to a combination of columns. It is said to be a composite primary key which contains a maximum of 16 columns.

**To set a constraint:**

create table stu3(sno number(5) constraint pk1 primary key, name char(10));

**To drop a constraint:**

alter table stu3 drop constraint PK1;

**To add a constraint:**

alter table stu3 add constraint PK1 primary key(name);

1. **Foreign key constraint:**

To establish a parent child relationship between two tables having a same column. To implement this we should define the column in the parent table as a primary key and the same column in the child table as a foreign key referencing to the corresponding parent entity.

**Create table dept(dept\_no number(5) constraint PK1 primary key, dept\_name char(20),dept\_loc varchar2(20));**

**Create table emp(eno number(5) constraint PK2 primary key, name char(10), sal number(8,2),designation char(10),dept\_no number(5) constraint fk1 references dept(dept\_no));**

Create table emp1(eno number(5), name char(10), sal number(8,2),designation char(10),dept\_no number(5));

Alter table emp1 add constraint fk2 foreign key (dept\_no) references dept(dept\_no);

Alter table emp1 drop constraint fk2;

1. **Check:**

The check constraint specifies condition that each row must satisfy. These rules are formed by logical expressions or Boolean expression.

Ex: create table emp(eno number(5),sal number(8,2) constraint ch1 check(sal>500));

1. **Default:**

When a row is inserted, it is not necessary that value for every column should not be inserted. SQL offers the provision to specify default values for columns.

Ex: create table emp(eno number(5), sal number(8,2) default 5000);

**Operators :**

1. **Null values:**

NULL values represent missing unknown data. By default, a table column can hold NULL values.

**Syntax:**

SELECT LastName,FirstName,Address FROM Persons WHERE Address is null;

1. **Distinct:**

The SELECT DISTINCT statement is used to return only distinct (different) values.

**Syntax**

SELECT DISTINCT *column\_name*,*column\_name* FROM *table\_name*;

**select distinct (sno) from emp;**

1. **NVL function:**

The NVL is a built in function in oracle. NVL returns another value of a Null. NVL accepts two parameters.

1. A column name
2. The value to be returned if the first parameter is null.

Ex: **select eno, ename,NVL(grade,’unknown grade’) from emp;**

**4.ALL:**

The ALL operator compares one value with all values in a list

Ex: **select \* from emp where eno> all(1,2,3);**

**5.Any:**

The Any operator compares one value with any values in a list

Ex: **select \* from emp where eno> any(1,2,3);**

**6.LIKE:**

The LIKE operator is used to search for a specified pattern in a column.

Underscore( \_ ) 🡪 matches one character in a specified position

Percent(%): 🡪 matches any no. of characters beginning at the specified position.

**Syntax**

SELECT *column\_name(s)* FROM *table\_name* WHERE *column\_name* LIKE *pattern*;

Ex: **select \* from emp where ename like’\_a%’;**

**7.IN operator:**

The ‘in’ operator in a where clause to retrieve the rows whose column values is in a list.

SELECT *column\_name(s)* FROM *table\_name* WHERE *column\_name* IN (*value1*,*value2*,...);

Ex: select \* from emp where eno in (1,2,3);

**8.Between:**

The BETWEEN operator is used to select values within a range.

**Syntax:**

SELECT *column\_name(s)* FROM *table\_name* WHERE *column\_name*BETWEEN *value1* AND *value2;*

Ex: Select \* from emp where eno between 1 and 3;

**9.SQL Logical Operators**

There are three Logical Operators namely, AND, OR, and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output. When retrieving data using a SELECT statement, you can use logical operators in the WHERE clause, which allows you to combine more than one condition.

**"OR" Logical Operator:**

If you want to select rows that satisfy at least one of the given conditions, you can use the logical operator, OR.

**For example:** if you want to find the names of students who are studying either Maths or Science, the query would be like,

SELECT first\_name, last\_name, subject  FROM student\_details   
WHERE subject = 'Maths' OR subject = 'Science'

**"AND" Logical Operator:**

If you want to select rows that must satisfy all the given conditions, you can use the logical operator, AND.

**For Example:** To find the names of the students between the age 10 to 15 years, the query would be like:

SELECT first\_name, last\_name, age  FROM student\_details   
WHERE age >= 10 AND age <= 15;

**"NOT" Logical Operator:**

If you want to find rows that do not satisfy a condition, you can use the logical operator, NOT. NOT results in the reverse of a condition. That is, if a condition is satisfied, then the row is not returned.

**For example:** If you want to find out the names of the students who do not play football, the query would be like:

SELECT first\_name, last\_name, games  FROM student\_details   
WHERE NOT games = 'Football'

**ORDER BY Clause**

The order by clause to sort the rows retrieved by a query. The ORDER BY clause may specify one or more columns on which to sort the data

select \* from customers order by last\_name;

The above query sort the rows retrieved from the customers table by the last name.

select \* from customers order by first\_name ASC , last\_name DESC;

The above query sort the rows retrieved from the customers table by ascending first\_name and descending last\_name.

**SQL Functions**

The Oracle implementation of SQL provides a number of functions that can be used in SELECT statements. Functions are typically grouped into the following:

* **Single row functions** - Operate on column values for each row returned by a query.
* **Group functions** - Operate on a collection (group) of rows.

**SINGLE ROW FUNCTIONS**

**Character functions:**

Characters functions accept character input, which may come from a column in a table or from any expression. This input is processed and a result returned.

ASCII()

The function ASCII(x) to get the ASCII value of for the character x.

select ascii('a') from dual;

CHR()

The function CHR(x) to get the character with ASCII value of x.

select CHR(95) from dual;

CONCAT()

The function CONCAT(x,y)to append y to x and then return the new string.

select concat(‘bvraju’,’college’) from dual;

INITCAP()

The function INITCAP(x) to convert the initial letter of each word in x to uppercase.

select initcap(‘a description of modern science’) from dual;

INSTR()

The function INSTR(x,find\_string[,start] [,occurrence]) to search for find\_string in x. This functrion returns the position at which find\_string occurs. We can supply optional start position to begin the search.

select instr(‘modern science’, ‘science’) from dual;

LENGTH()

The function LENGTH(x) to get the number of characters in x.

select length(‘rama’) from dual;

LOWER()

The function LOWER(x) to convert the letters in x to lower case.

select lower(‘RAMA’)from dual;

UPPER()

The function UPPER(x) to convert the letters in x to upper case.

select upper(‘rama’) from dual;

LPAD()

The function LPAD(x,width[,pad\_string]) to pad x with spaces to left to bring the total length of the string up to width characters. We can supply optional pad\_string.

select lpad(10000,8,’\*’) from dual;

RPAD()

The function RPAD(x,width[,pad\_string]) to pad x with spaces to right to bring the total length of the string up to width characters. We can supply optional pad\_string.

select rpad(‘rama’,10,’.’) from dual;

LTRIM()

The function LTRIM(x[,trim\_string]) to trim characters from the left to x. we can specify optional trim\_string , which specifies the characters to trim. If no trim\_string is supplied spaces are trimmed by default.

select ltrim(‘ Hello’) from dual;

RTRIM()

The function RTRIM(x[,trim\_string]) to trim characters from the right to x. we can specify optional trim\_string , which specifies the characters to trim. If no trim\_string is supplied spaces are trimmed by default.

select rtrim(‘hello‘,’o’) from dual;

TRIM()

The function TRIM() to trim characters from the left and right of x.

select trim(‘0’ from ‘000Hey Rama 00000’ )from dual;

REPLACE()

The function REPLACE(x,search\_string,replace\_string)to search x for search\_string and replace it with replace\_string.

Select replace (‘modern science’,’science’,’physics’) from dual;

SUBSTR()

The function SUBSTR(x,start[,length]) to return a substring of x that begins at the position specified by start. We can also provide an optional length for the substring.

select substr(‘bvraju college’,2,7) from dual;

**NUMERIC FUCTIONS**

These functions accept an input number , which may come from a numeric column or any expression that evaluates to a number. A calculation is then performed and a number is returned.

ABS()

The function ABS() to get the absolute value of x. The absolute value of a number is that number without any positive or negative sign.

SELECT ABS(10),ABS(-10) FROM dual;

MOD()

The function MOD(x,y) to get the remainder when x is divided by y.

select mod(8,3) from dual;

POWER()

The function POWER(x,y) to get the result of x raised to the power y.

select power(2,3) from dual;

ROUND()

The function ROUND(x,[y]) to get the result of rounding x to an optional y decimal places. If y is omitted x is rounded to zero decimal places.

select round(5.75) ,round(5.75,1) from dual;

SIGN()

The function SIGN(x) to get the sing of the x. SIGN() returns -1 if x is negative , 1 if x is positive , or 0 if x is zero.

select sign(-5) , sign(5) , sign(0) from dual;

SQRT()

The function SQRT(x) to get the square root of x.

select sqrt(9 ) from dual;

TRUNC()

The function TRUNC(x[,y]) to get the result of truncating the number x to an optional y decimal places.. if y is omitted ,x is truncated to zero decimal places.

select trunc(5.75),turnc(5.75,1) from dual;

**CONVERSION FUNCTONS**

These functions are used to convert a value from one data type to another.

TO\_CHAR()

The function

TO\_CHAR(x[,format])

to convert x to a string. The format is optional.

SELECT TO\_CHAR(12345.67) FROM DUAL;

SELECT TO\_CHAR(12345.67,’99,999.99’)FROM DUAL;

TO\_NUMBER()

The function TO\_NUMBER(x[,format]) to convert x to a number.

SELECT TO\_NUMBER(‘970.13’) FROM DUAL;

**DATETIME FUNCTIONS**

These functions are used to get or process datetimes and timestamps.

ADD\_MONTHS()

The function ADD\_MONTHS(x,y) returns the result of adding y months to x.

SELECT ADD\_MONTHS(’01-JAN-2007’ , 13) from dual;

LAST\_DAY()

The function LAST\_DAY(x) returns the date of the last day of the month of x.

SELECT LAST\_DAY (’01-JAN-2008’) FROM DUAL;

MONTHS\_BETWEEN ()

The function MONTHS\_BETWEEN(x,y) returns the number of months between x and y.

SELECT MONTHS\_BETWEEN (’25-may-2008’,’15-jan-2008’) FROM DUAL;

NEXT\_DAY ()

The function NEXT\_DAY(x,day) returns the date of the next day following x.

SELECT NEXT\_DAY(’01-JAN-2008’,’SATURDAY’) FROM DUAL;

ROUND()

The function ROUND(x[,unit]) rounds x, by default, to the beginning of the nearest day. If you supply an optional unit string , x is rounded to that unit.

SELECT ROUND(TO\_DATE(’25-OCT-2008’),’YYYY’) FROM DUAL;

SELECT ROUND(TO\_DATE(’25-MAY-2008’),’MM’)FROM DUAL;

SYSDATE

The SYSDATE return the current datetime set in the database server’s operating system.

SELECT SYSDATE FROM DUAL;

TRUNC()

The function TRUNC(X[,unit]) truncates x. By default, x is truncated to the beginning of the day. If you supply the optional unit string, x is truncated to that unit.

SELECT TRUNC(TO\_DATE(’25-OCT-2008’),’YYYY’) FROM DUAL;

SELECT TRUNC(TO\_DATE(’25-MAY-2008’),’MM’)FROM DUAL;

**Aggregative functions:**

These functions takes more than one row as input and returns one row as output.

1. COUNT ([Distinct] **A**) : The number of (unique) values in the A column.

Syntax: **select count(column\_name) from table name ;**

Eg : select count(emp\_no) from emp;

1. SUM (A) : The sum of all values in the A column.

Syntax:  **select sum(column\_name) from table name ;**

Eg. **select sum(sal) from emp;**

1. AVG (A) : The average of all values in the A column.

Syntax:  **select avg(column\_name) from table name ;**

Eg. **select avg(sal) from emp;**

1. MAX (**A**) : The maximum values in the A column.

Syntax:  **select max(column\_name) from table name ;**

Eg. **select max(sal) from emp;**

1. MIN (**A**) : The minimum value in the A column.

Syntax:  **select min(column\_name) from table name ;**

Eg. **select min(sal) from emp;**

**GROUP BY and HAVING Clause:**

The GROUP BY clause is used to group the rows base on some column , normally group by clause return only one row . Group by clause is used with **group functions** only.

**SELECT column\_name (or) aggregate\_function(column\_name) ,…**

**FROM table\_name**

**WHERE condition**

**GROUP BY column\_name**

**ORDER BY column\_name ;**

**Select job , sum(sal)**

**From emp**

**Where address=’bvrm’**

**Group by job**

**Order by job;**

**The HAVING Clause**

Having clause is used to filters the rows return by **group by clause.**

**SQL HAVING Syntax**

**SELECT column\_name (or) aggregate\_function(column\_name) ,…**

**FROM table\_name**

**WHERE condition**

**GROUP BY column\_name**

**HAVING condition;**

**Eg. Select job , sum(sal)**

**From emp**

**Where address=’bvrm’**

**Group by job**

**HAVING job=’cleark’ ;**

**JOINS**

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

There are different types of joins available in SQL:

**INNER JOIN:**

 returns rows when there is a match in both tables.

**LEFT JOIN:**

returns all rows from the left table, even if there are no matches in the right table.

**RIGHT JOIN:**

returns all rows from the right table, even if there are no matches in the left table.

**FULL JOIN:**

returns rows when there is a match in one of the tables.

**SELF JOIN:**

is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

**CARTESIAN JOIN:**

returns the Cartesian product of the sets of records from the two or more joined tables.

The most frequently used and important of the joins is the **INNER JOIN**. They are also referred to as an EQUIJOIN.

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

The basic syntax of **INNER JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

INNER JOIN table2

ON table1.common\_field = table2.common\_field;

**(or)**

SELECT table1.column1, table2.column2...

FROM table1 , table2

table1.common\_field = table2.common\_field;

CUSTOMER table

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

(b) Another table is ORDERS as follows:

+-----+---------------------+-------------+--------+

| OID | DATE | ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables using INNER JOIN as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

INNER JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

**(or)**

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS , ORDERS

CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+----+----------+--------+---------------------+

The left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax:

The basic syntax of **LEFT JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

LEFT JOIN table2

ON table1.common\_field = table2.common\_field;

(or)

SELECT table1.column1, table2.column2...

FROM table1 , table2

table1.common\_field = table2.common\_field(+);

Now, let us join above two tables using LEFT JOIN as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

**(or)**

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS , ORDERS

CUSTOMERS.ID = ORDERS.CUSTOMER\_ID(+);

The right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax:

The basic syntax of **RIGHT JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

RIGHT JOIN table2

ON table1.common\_field = table2.common\_field;

(or)

(or)

SELECT table1.column1, table2.column2...

FROM table1 , table2

table1.common\_field (+)= table2.common\_field;

Now, let us join above two tables using LEFT JOIN as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

RIGHT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

**(or)**

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS , ORDERS

CUSTOMERS.ID(+) = ORDERS.CUSTOMER\_ID;

The SQL **FULL JOIN** combines the results of both left and right outer joins.

The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Syntax:

The basic syntax of **FULL JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

FULL JOIN table2

ON table1.common\_field = table2.common\_field;

Example:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

FULL JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

The SQL **SELF JOIN** is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax:

The basic syntax of **SELF JOIN** is as follows:

SELECT a.column\_name, b.column\_name...

FROM table1 a, table1 b

WHERE a.common\_field = b.common\_field;

The **CARTESIAN JOIN** or **CROSS JOIN** returns the Cartesian product of the sets of records from the two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is absent from the statement.

Syntax:

The basic syntax of **CARTESIAN JOIN OR CROSS JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1, table2 [, table3 ] ;

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS, ORDERS;

**SUB QUERY**

A Sub query or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

**Types of Sub queries:**  
  
There are two types of sub queries in oracle:  
  
**Single Row Sub queries**: The sub query returns only one row. Use single row comparison operators like =, > etc while doing comparisons.

**Multiple Row Sub queries:** The sub query returns more than one row. Use multiple row comparison operators like IN, ANY, ALL in the comparisons.

Sub queries are most frequently used with the SELECT statement. The basic syntax is as follows:

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE])

**Example of Singel Row Sub Query** :

SQL> SELECT \*

FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS

WHERE SALARY > 4500) ;

 Write a query to find the employees whose salary is equal to the salary of at least one employee in department of id 300?

**Example of Multi Row Sub Query**

SELECT EMPLOYEE\_ID, SALARY

FROM EMPLOYEES

WHERE SALARY IN

(

SELECT SALARY

FROM EMPLOYEES

WHERE DEPARTMENT\_ID = 300

);

### **CORRELATED SUB QUERY**

Correlated sub query is used for row by row processing. The sub query is executed for each row of the main query.

Write a query to list the department names which have at lease one employee?

SELECT DEPARTMENT\_ID,

DEPARTMENT\_NAME

FROM DEPARTMENTS D

WHERE EXISTS

(

SELECT 1

FROM EMPLOYEES E

WHERE E.DEPARTMENT\_ID = D.DEPARTMENT\_ID)

Write a query to find the departments which do not have employees at all?

SELECT DEPARTMENT\_ID,

DEPARTMENT\_NAME

FROM DEPARTMENTS D

WHERE NOT EXISTS

(

SELECT 1

FROM EMPLOYEES E

WHERE E.DEPARTMENT\_ID = D.DEPARTMENT\_ID)

Write a query to find the departments which do not have employees at all?

SELECT DEPARTMENT\_ID,

DEPARTMENT\_NAME

FROM DEPARTMENTS D

WHERE NOT EXISTS

(

SELECT 1

FROM EMPLOYEES E

WHERE E.DEPARTMENT\_ID = D.DEPARTMENT\_ID)

**View:**

A view is a pre defined query on one or more tables. Retrieving information from a view is done in the same manner as retrieving from a table. We simply include the view in the from clause of a query with some views we can also perform DML operation on the base table.

**Creating and using a view:**

We create the view using **CREATE VIEW** statement which has th following syntax:

Syntax:

**Create [or REPLACE] VIEW view\_name AS Sub\_Query [with { check option |read only} constraint constraint-name];**

REPLACE means the view replace an existing view.

View\_name is the name of the view.

Sub query is the sub query that retrives from the base table.

WITH CHECK OPTION means that only the rows that would be retrieved by the sub query can be inserted updated or deleted.

Constraint\_name is the name of the WITH CHECK OPTION or WITH READ ONLY CONSTRAINT.

WITH READ ONLY means that the rows may only read from the base table.

There are two types of views

1. Simple views which cointains a subquery that retrives from one base table
2. Complex view which cointains a subquery that retrives from multiple base tables.

**create table emp(eno number(5), ename char(10),address varchar2(20));**

**Examples of View:**

1. **create view emp\_view as select \* from emp;**

In this example we can insert , retrieve ,update and delete records from the view name emp\_view.

1. **create view emp\_view2 as select \* from emp where eno<10 with check option constraint cn1;**

In this example we can insert , retrieve ,update and delete records from the view name emp\_view those reocrds having eno less 10.

1. **create view emp\_view3 as select \* from emp where eno<10 with read only constraint cn2;**

In this example we can retrieve records from the view name emp\_view those reocrds having eno less 10 , we cannot insert, update and delete records from this view.

**Dropping a View :** Dropping A VIEW is used to delete the view.

**Syntax : DROP VIEW view\_name;**

Eg : DROP VIEW emp\_view;

**PROCEDURE**

A procedure is a module performing one or more actions: it does not need to return any value. The syntax for creating a procedure is

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(PARAMETER 1 {IN,OUT,INOUT} DATATYPE(SIZE),

PARAMETER 2 {IN,OUT,INOUT} DATATYPE(SIZE),….

PARAMETER N {IN,OUT,INOUT} DATATYPE(SIZE))]

AS

[local declaration]

BEGIN

Executable statements

END [name];

Where [OR REPLACE] means the procedure is to replace an existing procedure

Procedure\_name : is the name of procedure

Parameter\_name is the name of the parameter i.e., passed to the procedure

IN|OUT|IN OUT : is the mode of parameter

IN – which is the default mode , An IN operator must be set to a value , when the procedure run . The value of IN parameter cannot be changed in the procedure body

OUT – which means the parameter is set to a value in the procedure body.

IN OUT – which means the parameter can have a value when the procedure is run and value can be changed the body.

Type : is the type of the parameter

Procedure\_body : contains the actual code for the procedure.

**Calling a Procedure :**

We run a procedure using the call statement . The call statement followed by a procedure\_name.

CALL procedure\_name( );

**Dropping a Procedure :**

We can drop a procedure using DROP PROCEDURE statement

DROP PROCEDURE procedure\_name ;

Example

Create a procedure to add two number and call the block with a PL/SQL block?

SQL> **create or replace procedure sum(a in number, b in number)**

2 is

3 c number := 1;

4 begin

5 c := a+b;

6 dbms\_output.put\_line('c value '||c);

7 end;

8 /

Procedure created.

SQL> declare

2 a number := &a;

3 b number := &b;

4 begin

5 sum(a,b);

6 end;

7 /

Enter value for a: 10

old 2: a number := &a;

new 2: a number := 10;

Enter value for b: 20

old 3: b number := &b;

new 3: b number := 20;

c value 30

PL/SQL procedure successfully completed.

Create a procedure to add two number and return the value to a PL/SQL block?

SQL> create or replace procedure sum1(a in number,b in number,d out number)

2 is

3 c number := 1;

4 begin

5 c := a+b;

6 d:=c;

7 end;

8 /

Procedure created.

SQL> declare

2 a number := &a;

3 b number := &b;

4 d number;

5 begin

6 sum1(a,b,d);

7 dbms\_output.put\_line('d value '||d);

8 end;

9 /

Enter value for a: 10

old 2: a number := &a;

new 2: a number := 10;

Enter value for b: 30

old 3: b number := &b;

new 3: b number := 30;

d value 40

PL/SQL procedure successfully completed.

The syntax for creating a function is as follows:

CREATE OR REPLACE FUNCTION function\_name

[(PARAMETER 1 {IN,OUT,INOUT} DATATYPE(SIZE),

PARAMETER 2 {IN,OUT,INOUT} DATATYPE(SIZE),….

PARAMETER N {IN,OUT,INOUT} DATATYPE(SIZE))]

RETURN datatype

IS

[local declaration]

BEGIN

Executable statements

END [name];

The function does not necessarily have any parameters, but it must have a RETURN value declared in the header, and it must return values for all the varying possible execution streams.

**Calling a Function :**

We run a procedure using the call statement . The call statement followed by a procedure\_name.

Variable = CALL function\_name( );

**Dropping a Function :**

We can drop a function using DROP FUNCTION statement

DROP FUNCTION function\_name ;

Create a function to add two number and return the value to a PL/SQL block?

SQL> **create or replace function f(a in number)**

2 return number

3 as

4 b number;

5 begin

6 b:=10;

7 b:=a+b;

8 return b;

9 end;

10 /

Function created.

SQL> declare

2 a number:=10;

3 c number;

4 begin

5 c := f(a);

6 dbms\_output.put\_line('c value'||c);

7 end;

8 /

c value20

PL/SQL procedure successfully completed.

**CURSOR**

1. Declaring the cursor defines the name of the cursor and associatedit with a SELECT statements **CURSOR cursor\_name IS SELECT statement**
2. Opening the cursor is to process the select statement and set a active pointer to the first row **OPEN cursor\_name;**
3. Fetching cursor into PL/SQL local variables, these variables are declared as ROWTYPE in declare section  **FETCH cursor\_name INTO PL/SQL variables**
4. Closing a cursor by **CLOSE cursor\_name**;

Write a PL/SQL program to find the total salary of employees

declare

v\_emp\_no employee.emp\_no%type;

v\_first\_name employee.first\_name%type;

v\_salary employee.sal%type;

total\_salary number(10,2);

cursor c\_emp is select emp\_id,first\_name,sal from emp where sal<10000;

begin

open c\_emp;

total\_salary:= 0;

loop

fetch c\_emp into v\_emp\_no,v\_first\_name,v\_salary;

dbms\_output.put\_line(v\_emp\_no||v\_first\_name||v\_salary);

total\_salary:=total\_salary + v\_salary;

exit when c\_emp%notfound;

end loop;

close c\_emp;

dbms\_output.put\_line(‘Total Salary :’|| total\_salary);

end;

**TRIGGER**

A trigger is a procedure i.e., run automatically by the database when a specified DML statement (INSERT, UPDATE, DELETE) is run against a certain database .

A trigger may fire before or after DML statements run.

A row level trigger runs each and every row updated in the table

For example , if we update a 10 rows in the table the row level trigger runs trigger body 10 times.

A statement level trigger runs only once when we update a table.

For example , when we update a 10 rows in the table the statement level trigger runs only once trigger body

A row level trigger has access to the old the new column values when the trigger fires. As a result of an update statement on that column.

**CREATE [OR REPLACE] TRIGGER trigger\_name**

**{BEFORE | AFTER } trigger\_event**

**ON table\_name**

**[FOR EACH ROW]**

**[WHEN trigger\_condition]**

**BEGIN**

**Trigger\_body ;**

**END trigger\_name ;**

Where OR REPLACE means the trigger is to replace an existing trigger , if present

**trigger\_name** is the name of the trigger

**BEFORE** means the trigger fires before the trigger\_event is performed

**AFTER** means the trigger fires after the trigger\_event is performed

**trigger\_event** is the event that causes the trigger to fire

**table\_name** is the table name that the trigger references

**FOR EACH ROW** : means the trigger is a row level trigger , if we vomit , FOR EACH ROW the trigger is a statement level trigger .

**trigger\_condiditon**: is a Boolean condition

**trigger\_body** : contains the code for the trigger

Eg.

CREATE OR REPLACE trigger before\_product\_update

BEFORE update of price on product for each row

when (new.price < old.price \* 0.75)

begin

dbms\_output.put\_line(‘Product\_id’||:old.prod\_id);

dbms\_output.put\_line(‘Old\_Price’||:old.price);

dbms\_output.put\_line(‘New\_Price’||:new.price);

dbms\_output.put\_line(‘The price reduction is more than 25%’);

Insert into product\_audit(prod\_id,old\_price,new\_price)

values(:old.prod\_id,:old.price,:new.price)

End before\_produt\_update;

**Firing a Trigger**

To fire a trigger we must perform the DML operations on the table.

Eg : Update product set price = price \* 0.70 where prod\_id in (10,15);