Introduction to SQL

SQL is a standard language for accessing and manipulating databases.

What is SQL?

SQL (Structured Query Language) is a standard interactive and programming language for getting information from and updating a database. Although SQL is both an ANSI and an ISO standard, many database products support SQL with proprietary extensions to the standard language. Queries take the form of a command language that lets you select, insert, update, find out the location of data, and so forth. There is also a programming interface.

Why SQL?

 Allows users to access data in relational database management systems.

 Allows users to describe the data.

 Allows users to define the data in database and manipulate that data.

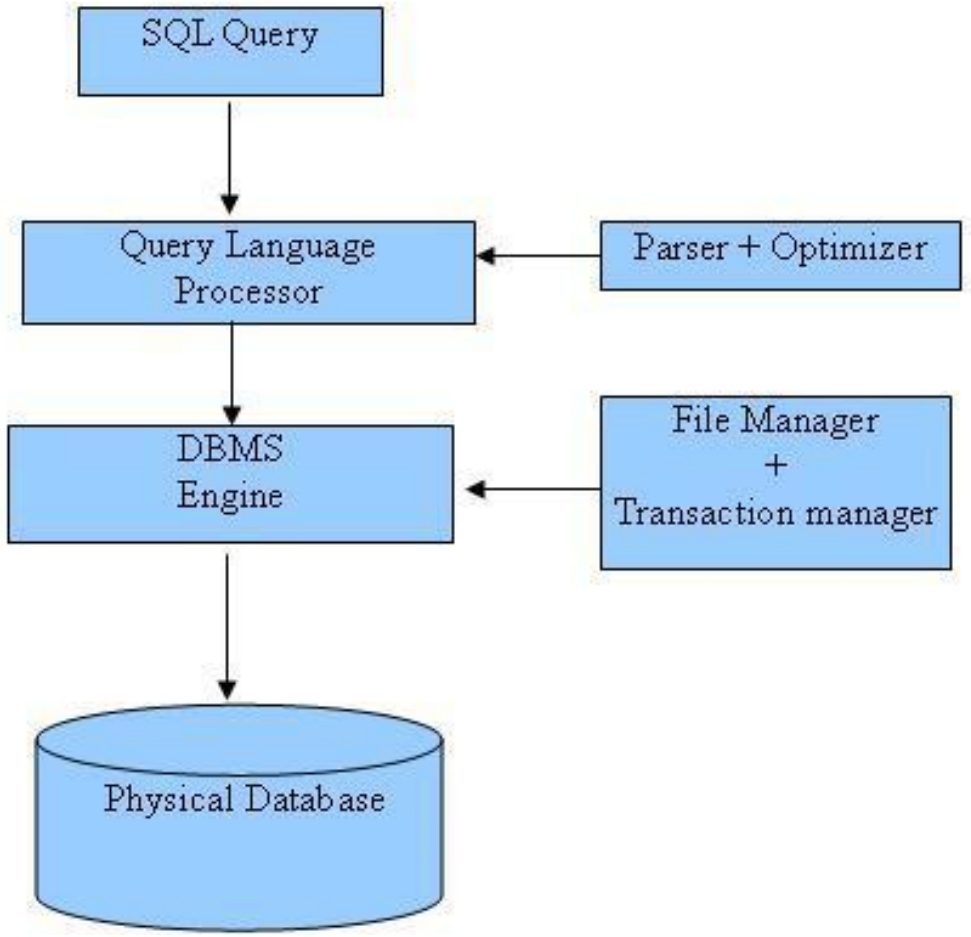
 Allows to embed within other languages using SQL modules, libraries & pre-compilers.

 Allows users to create and drop databases and tables.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database

- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views



Using SQL in Your Web Site

To build a web site that shows data from a database, you will need:

- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like PHP or ASP
- To use SQL to get the data you want
- To use HTML / CSS

Types of SQL Statements

The tables in the following sections provide a functional summary of SQL statements and are divided into these categories:

- Data Definition Language(DDL) Statement
- Data Manipulation Language(DML) Statement
- Transaction Control Statement
- Session Control Statement
- System Control Statement
- Embedded SQL Statement

Data Definition Language (DDL) Statements

Data definition language (DDL) statements let you to perform these tasks:

- Create, alter, and drop schema objects
- Grant and revoke privileges and roles
- Analyze information on a table, index, or cluster
- Establish auditing options
- Add comments to the data dictionary

The CREATE, ALTER, and DROP commands require exclusive access to the specified object. For example, an ALTER TABLE statement fails if another user has an open transaction on the specified table.

The GRANT, REVOKE, ANALYZE, AUDIT, and COMMENT commands do not require exclusive access to the specified object. For example, you can analyze a table while other users are updating the table.

Oracle Database implicitly commits the current transaction before and after every DDL statement.

Many DDL statements may cause Oracle Database to recompile or reauthorize schema objects. For information on how Oracle Database recompiles and reauthorizes

schema objects and the circumstances under which a DDL statement would cause this, see *Oracle Database Concepts*.

The DDL statements are:

ALTER ... (All statements beginning with ALTER)
ANALYZE
ASSOCIATE STATISTICS
AUDIT
COMMENT
CREATE ... (All statements beginning with CREATE)
DISASSOCIATE STATISTICS

DROP ... (All statements beginning with DROP)
FLASHBACK ... (All statements beginning with FLASHBACK)
GRANT
NOAUDIT
PURGE
RENAME
REVOKE
TRUNCATE
UNDROP

Data Manipulation Language (DML) Statements

Data manipulation language (DML) statements access and manipulate data in existing schema objects. These statements do not implicitly commit the current transaction. The data manipulation language statements are:

CALL
DELETE
EXPLAIN PLAN
INSERT
LOCK TABLE
MERGE
SELECT
UPDATE

The SELECT statement is a limited form of DML statement in that it can only access data in the database. It cannot manipulate data in the database, although it can operate on the accessed data before returning the results of the query.

The CALL and EXPLAIN PLAN statements are supported in PL/SQL only when executed dynamically. All other DML statements are fully supported in PL/SQL.

Transaction Control Statements

Transaction control statements manage changes made by DML statements. The transaction control statements are:

COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

All transaction control statements, except certain forms of the COMMIT and ROLLBACK commands, are supported in PL/SQL. For information on the restrictions, see COMMIT and ROLLBACK .

Session Control Statements

Session control statements dynamically manage the properties of a user session. These statements do not implicitly commit the current transaction.

PL/SQL does not support session control statements. The session control statements are:

ALTER SESSION
SET ROL

System Control Statement

The single system control statement, ALTER SYSTEM, dynamically manages the properties of an Oracle Database instance. This statement does not implicitly commit the current transaction and is not supported in PL/SQL.

Embedded SQL Statements

Embedded SQL statements place DDL, DML, and transaction control statements within a procedural language program. Embedded SQL is supported by the Oracle precompilers and is documented in the following books:

SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

DDL -Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML -Data Manipulation Language:

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL -Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

DQL -Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

Sql function

I .CHARACTER FUNCTIONS

Oracle provide a variety functions for working with strings in SQL. These functions accept literal string of characters(‘cake’, ‘book’,etc) or name of character columns and return both character and numeric values. Following are some character functions:

1.ASCII(string): It returns the decimal representation in the database character set of the first character of the string.

- select ascii('a'),ascii('A'),ascii('roshini') as r_in_roshini ,ascii('Roshini')from dual

ASCII('A')	ASCII('A')	R_IN_ROSHINI	ASCII('ROSHINI')
97	65	114	82

2. CHR(n):

It returns the character having the binary equivalent to **n** in either the database character set or national character set.

- select chr(35),chr(40),chr(50),chr(60),chr(70),chr(80),chr(90),chr(100)from dual
-

CHR(35)	CHR(40)	CHR(50)	CHR(60)	CHR(70)	CHR(80)	CHR(90)	CHR(100)
#	(	2	<	F	P	Z	d

3. CONCAT(m,n):

It concatenate **m** with **n** where **m** or **n** can be either column name or literals. It is equivalent to concatenation operator (||).

- select concat('Shreya',' Sharma') as name from dual

NAME
Shreya Sharma

- select concat(rollnum,name) name from student

NAME
1265dimpy
1262suprina
1261aman
1271smily

4.INSTR:

It is used to determine the numeric position of a particular search string within a character string or iterating through a document looking for multiple instances of a character string.

- select instr('kabir','i') from dual

INSTR('KABIR','I')
4

- select instr(name,'i'),name from student;

INSTR(NAME,'I')	NAME
2	dimpy
5	suprina
0	aman
3	smily

5.LENGTH(n):-

It returns the number of characters in the string/column name **n**.

- select length('jalandhar'),length('ludhiana')city from dual

LENGTH('JALANDHAR')	CITY
9	8

- select length(name),name from student

LENGTH(NAME)	NAME
5	dimpy
7	suprina
4	aman
5	smily

6.LPAD:

(but in 10g its trimming)

It makes a string of certain length by adding certain set of characters to the left of string.

- select lpad('hello12',4,' '),lpad('12345',4,'*') from dual

LPAD('HELLO12',4,'')	LPAD('12345',4,'*')
hell	1234

- select lpad(name,4,' ')from student

LPAD(NAME,4,'')
dimp
supr
aman
smil

7.LTRIM:

It returns leading spaces or a designated character on the left side of the string.

- selectltrim(name,'m'),ltrim(name) as orignal from student

LTRIM(NAME,'M')	ORIGNAL
dimpy	dimpy
suprina	suprina
aman	aman
smily	smily

8. RTRIM:

It returns leading spaces or a designated character on the right side of the string.

- select rtrim(name,'a'),rtrim(name) as orignal from student

RTRIM(NAME,'A')	ORIGNAL
dimpy	dimpy
suprin	suprina
aman	aman
smily	smily

- select ltrim(rtrim(name,'a')),rtrim(name) as orignal from student

LTRIM(RTRIM(NAME,'A'))	ORIGNAL
dimpy	dimpy
suprin	suprina
aman	aman
smily	smily

7. SUBSTR:

It is used to extract a piece of character string.

- select substr(name,2,4) ,name from student

SUBSTR(NAME,2,4)	NAME
impy	dimpy
upri	suprina
man	aman
mily	smily

- select substr(name,-4) ,name from student

SUBSTR(NAME,-4)	NAME
impy	dimpy
rina	suprina
aman	aman
mily	smily

8. REPLACE:

This function is used to replace character or group of characters in a string with zero or more characters.

- Select replace('hello world am here!','world','HMV') "repalceIt"from dual

Repalce It
hello HMV am here!

- Select name as orignal ,replace(name,'anita','anjali') "repalce It" from student

ORIGNAL	Repalce It
dimpy	dimpy
suprina	suprina
aman	aman
smily	smily
anita	anjali

CASE Conversion funcs

1.LOWER(string):

This function converts every letter in a string to lowercase.

- Select lower('KAREENA') from dual

LOWER('KAREENA')
kareena

- **Select** name as orignal ,lower(name) from student

ORIGNAL	LOWER(NAME)
dimpy	dimpy
suprina	suprina
aman	aman
smily	smily
anita	anita

2. UPPER(string):

This function is used to convert all the characters in the string to uppercase. **Example:-**

- Select upper('kareena')from dual

UPPER('KAREENA')
KAREENA

- Select name as orignal ,upper(name) from student

ORIGNAL	UPPER(NAME)
dimpy	DIMPY
suprina	SUPRINA
aman	AMAN
smily	SMILY
anita	ANITA

3. INITCAP(string)

- Select initcap('anamika') from dual

INITCAP('ANAMIKA')
Anamika

- Select name as orignal ,initcap(name) from student

ORIGNAL	INITCAP(NAME)
dimpy	Dimpy
suprina	Suprina
aman	Aman
smily	Smily
anita	Anita

4. TRANSLATE

- select translate('samsung','s',8) from dual

TRANSLATE('SAMSUNG','S',8)
8am8ung

- select name as orignal,translate(name,'a','x') from student

ORIGNAL	TRANSLATE(NAME,'A','X')
dimpy	dimpy
suprina	suprinx
aman	xmxn
smily	smily
anita	xnitx

II .NUMBER FUNCTIONS

1. ABS(n):to get the absolute value

- select abs(-50) as absolute_function from dual

ABSOLUTE_FUNCTION
50

- select abs(-500) "absolute" from dual

Absolute
500

- select abs(-100), abs(25), abs(-0) from dual

ABS(-100)	ABS(25)	ABS(-0)
100	25	0

2. CEIL(n): returns the next smallest integer value greater than or equal to

- select ceil(12.23),ceil(6.7),ceil(0) from dual

CEIL(12.23)	CEIL(6.7)	CEIL(0)
13	7	0

3. EXP(n): returns e raised to nth power(e=2.7182…..).

- Select exp(0) "exponent" from dual

Exponent
1

4. COS:

- select cos(30) from dual

COS(30)
.154251449887584050718662146614210196 73

5. FLOOR(n):

returns the largest integer value less than or equal to the parameter **n**.

- Select floor(28.3), floor(-28.3),floor(28.89),floor(28) from dual

FLOOR(28.3)	FLOOR(-28.3)	FLOOR(28.89)	FLOOR(28)
28	-29	28	28

6. MOD(m,n):

returns reminder of **m** divided by **n**. Returns m if n=0.

- Select mod(12,4),mod(4,2), mod(20,3) from dual

MOD(12,4)	MOD(4,2)	MOD(20,3)
0	0	2

7. POWER(m,n):

This function returns **m** raised to **n**th power. The base **m** and exponent **n** can be any numbers, but if **m** is negative, **n** must be integer.

- Select power(2,3)as cube,power(3,2),power(3,-2) as neg_square,power(10,-2) as neg_square from dual

CUBE	POWER(3,2)	NEG_SQUARE	NEG_SQUARE
8	9	.11111111111111111111111111111111111111	.01

8. ROUND(m,n):

This function rounds a number **m** either to the left or right of the decimal point depending upon the sign of **n**.

- Select round(24.65,1),round(24.65,2),round(24.6526,3),round(24.65),round(24.65,-1),round(25.65,-1) from dual

ROUND(24.65,1)	ROUND(24.65,2)	ROUND(24.6526,3)	ROUND(24.65)	ROUND(24.65,-1)	ROUND(25.65,-1)
24.7	24.65	24.653	25	20	30

9.SIGN(n):

It returns sign of value **n**. Returns -1, 0 or 1 depending on weather n<0, n=0 or n>0 respectively.

- Select sign(-8), sign(8), sign(-0) from dual

SIGN(-8)	SIGN(8)	SIGN(-0)
-1	1	0

10. SQRT(n):

This function returns the square root of **n**. When **n** is greater than or equal to 0.

- Select sqrt(25),sqrt(49) as sqaure_root from dual

SQRT(25)	SQAURE_ROOT
5	7

11.TRUNC(m,n):

This function drops all digits of number **m** either to left or right of the decimal point depending upon the size of **n**, where **n** must be an integer.

- Select trunc(25.35,1),trunc(25.35,-1) ,trunc(25.35) from dual

TRUNC(25.35,1)	TRUNC(25.35,-1)	TRUNC(25.35)
25.3	20	25

III DATE FUNCTIONS

In order to process and manipulate dates, Oracle provides a number of functions that operate on various dates related datatypes in Oracle. The default date format in Oracle is DD-Mon-YY HH:MI:SS. Following are some date functions:-

1. **SYSDATE:**

This function returns the current date and time in a date

values based on the database time zone.

- **select sysdate from dual**

SYSDATE
06-APR-16

2.ADD_MONTHS: add_months(date,<no of months u wanna add>)

This function adds or subtracts a number of months to/from a date.

- select add_months('15-mar-16',12)as future,add_months('15-mar-16',-12) as past from dual

FUTURE	PAST
15-MAR-17	15-MAR-15

3. LAST_DAY(date):

This function returns the date of the last day or the month containing the **date** parameter.

- select last_day('15-mar-16') "current",last_day('15-feb-16') "feb16",last_day('15-feb-15') "feb15" from dual

Current	Feb16	Feb15
31-MAR-16	29-FEB-16	28-FEB-15

4. MONTHS_BETWEEN:

Note→If the first date is earlier in time than the second date, value returned is a negative number.

- select months_between('15-nov-16','15-mar-16') from dual

MONTHS_BETWEEN('15-NOV-16','15-MAR-16')
8

5. NEXT_DAY(date,day_week):

It returns the value of next named **day** of the week after the given **date**.

6. ROUND(date[,format]):

This function round off the date/time value formatted to the next highest part of date.

Select

7. TRUNC(date[,format]):

This function when used with the date return a date value truncated to the values optionally specified in the format parameter.

Example:-

8. ARITHMETIC OPERATIONS ON DATE:

The database stores dates as numbers, this allows users to perform various arithmetic operation such as addition and substraction. Some of operations which are performed are:-

OPERATION	RESULT	PURPOSE
Date+Number	Date	Adds number of days to date
Date-Number	Date	Subtracts number of days from date
Date1-Date2	Number of days	Subtracts one date from another date

Note →v cannot add a date to another date.

- **Select sysdate+30,sysdate-30,sysdate-sysdate from dual**

SYSDATE+30	SYSDATE-30	SYSDATE-SYSDATE
06-MAY-16	07-MAR-16	0

AGGREGATE FUNCTIONS

The aggregate functions act on a group of rows to give a result per group or ows rather on single row. This is the reason why they are known as group functions.

1. **COUNT:**

The COUNT function returns the number of NON-NULL values in a set of records. Null values are not counted unless COUNT(*) is used, which counts the total number of rows in table.

Q. count the number of employees in student table

Select count from student

Q. list the number of dept. in table student_hmv

Select count(distinct dept) from student_hmv

2.SUM:

The SUM function returns the sum of all th values for a column, ignoring nul values. It can apply to only columns with NUMBER data type.

- **Select sum(marks) from student**

SUM(MARKS)
2250

- **Select sum(distinct marks) from student**

SUM(DISTINCTMARKS)
2250

3.AVG:

The AVG function returns the average of all the NON-NULL values for a column. It can apply to only columns with NUMBER data type.

- **Select avg(marks) from student**
-

AVG(MARKS)
450

4.MAX:-

The MAX function returns the maximum value for a specified column which may be of any data type.

- **Select max(marks) from student**

MAX(MARKS)
790

5.MIN:-

The MIN function returns the minimum value for a specified column which may be of any data type.

Select min(name),min(marks) from student

MIN(NAME)	MIN(MARKS)
aman	290

operators...

logical operators

ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to **search for values that are within a set of values, given the minimum value and the maximum value.**
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN (to check value within a set)	The IN operator is used to compare

	a value to a list of literal values that have been specified.
LIKE (pattrn matching)	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL> SELECT * FROM STUDENT WHERE AGE >= 20 AND MARKS >= 650;

NAME	ROLNO	AGE	MARKS
anita	1260	25	790

SQL> SELECT * FROM STUDENT WHERE AGE >= 21 OR MARKS >= 650;

NAME	ROLNO	AGE	MARKS
suprina	1262	21	390
aman	1261	22	340
anita	1260	25	790

SQL> SELECT * FROM STUDENT WHERE AGE IS **NOTNULL**;

NAME	ROLNO	AGE	MARKS
anita	1260	-	790

SQL> SELECT * FROM STUDENT WHERE AGE **IN** (20, 19);

NAME	ROLNO	AGE	MARKS
dimpy	1265	20	290
smily	1271	19	440

SQL>SELECT * FROM STUDENT WHERE AGE **BETWEEN** 20 AND 17;

NAME	ROLNO	AGE	MARKS
dimpy	1265	20	290
smily	1271	19	440
kaju	1260	17	790

SQL> SELECT AGE FROM STUDENT

AGE
20
21
22
19
25
-
17

WHERE EXISTS (SELECT AGE FROM STUDENT WHERE MARKS> 6500);

AGE
25
-
17

SQL> SELECT * FROM STUDENT

NAME	ROLNO	AGE	MARKS
dimpy	1265	20	290
suprina	1262	21	390
aman	1261	22	340
smily	1271	19	440
anita	1260	25	790
anita	1260	-	790
kaju	1260	17	790

WHERE AGE > ALL (SELECT AGE FROM STUDENT WHERE MARKS> 6500);

AGE
21
22
19
25
-
17

SQL> SELECT * FROM CUSTOMERS

AGE	SALARY
20	2000
24	6000
23	4000
25	5000

WHERE AGE > ALL (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);

AGE
20
24
23
25

SQL> SELECT * FROM CUSTOMERS

AGE	SALARY
20	2000
24	6000
23	4000
25	5000

4 rows returned in 0

WHERE AGE > ANY (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);

AGE
20
24
23
25

Distinct (special operator-elimination of duplicate records)

select distinct class from student;

CLASS
b.voc
ba
bmm

select distinct class,marks from student;

CLASS	MARKS
b.voc	440
ba	290
bmm	340

---

select**sysdate** from dual

SYSDATE
07-APR-16

Order by clause

Select name,marks from student order by marks desc;

NAME	MARKS
smily	440
aman	340
dimpy	290

Q. list the name and marks of students who are in bvoc ,sorting them in decending order of there marks

Select name,marks from student where class=’bvoc’ order by marks desc;

NAME	MARKS
annu	524

Q. list the name and marks of students who are in bvoc ,sorting them in alphabetically

Select name, marks from student where class='bvoc' order by name asc;

NAME	MARKS
annu	524

// sorting on base of first colum

Select name, marks from student where class='bvoc' order by 1asc;

NAME	MARKS
tannu	224

// sorting on base of second column aka marks here

Select name, marks from student where class='bvoc' order by 2 asc;

NAME	MARKS
smily	440

Select name,class,marks from student order by class desc,marksdesc;

NAME	CLASS	MARKS
aman	bmm	340
dimpy	ba	290
smily	b.voc	440

Pattern matching

//Q. list the name of students having ly in end of the name
SELECT * FROM STUDENT WHERE NAME **LIKE** '%ly';

NAME	ROLNO	AGE	MARKS	CLASS
smily	1271	19	440	b.voc

//Q. list the name of students having 5 character length
SELECT * FROM STUDENT WHERE NAME **LIKE** '- - - - -';

//Q. list the name of students where 2nd character is n
SELECT * FROM STUDENT WHERE NAME **LIKE** '_n'

//Q. list the name of students where A letter occurs two times
SELECT * FROM STUDENT WHERE NAME **LIKE** '%a%a%';

NAME	ROLNO	AGE	MARKS	CLASS
aman	1261	22	340	bmm

==========

set operator

(used to combine information of similar type from 1 or more than 1 table)

union

merges the o/p of two or more queries in a single set rows or columns. eliminates duplicate values

select class from student where dept="multimedia"

CLASS
ba
bmm

union

select class from student_hmv where dept="masscomm"

CLASS
bmm

Intersect

retrieves common records from two or more tables

select name from student_hmv where dept="multimedia"

NAME
tannu
kaju

intersect

select name from student_hmv where dept="masscomm"

NAME
annu

Minus operator

(retrieves the rows which are unique to first query,)

(a-b !=b-a)

select class from student where class="bvoc"

CLASS
b.voc

minus

select class from student where class="bmm"

CLASS
bmm
bmm

note:> union and intersect follow commutative property but minus operator doesn't follow

---

changing column heading without changing column aliases

select name "student_name"from student;

Student_Name
tannu
annu
kaju

```
CREATE TABLE student_clg
( rollno number(3),
name varchar2(20),
father_name varchar2(20),
class varchar2(20),
city varchar2(20),
hobby varchar2(20),
foreign KEY (rollno)references student_bvoc(rollno)
)
```

SQL Join Types:

- INNER JOIN/equi join :

  >>In which condition contains equality operator { Operator : = }
  >>combine rows having equivalent values
  >>returns all the rows when there is a match in both tables.

- CARTESIAN JOIN:**returns the Cartesian product of the sets of records from the two or more joined tables**
  >>in this join each row of one table is**joined to every row of another table**
  >>result is Cartesian product when **join condition is omitted from join query**

- Outer join:
  >> extends the result of inner join.
  >>if one table have certain records and corresponding table have no same records then those rows will not be selected in inner join but outer join can obtain even those records forcefully.
  >>result will be all d rows fulfilling d join condition ,having rows from one table datdont have corresponding rows in other tables.
  >>uses PLUS OPERATOR (+) ,placed on d side of join dat is incomplete of information

- 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**.

INNER JOIN / EQUIJOIN.

```
SELECT field1 ,field2...
FROM table1 ,table2
Where table1.pk = table2.fk;
List the name of the students with their dept
Eg :
        SELECT rollno,name,father_name,computer_marks
        FROM student_bvoc,student_clg
        Where student_bvoc.rollno= student_clg.rollno;
```

Using AND OR NOT

```
SELECT rollno,name,computer_marks,city
FROM student_bvoc,student_clg
Where student_bvoc.rollno= student_clg.rollno
AND city='jalandhar';
```

Using natural join

```
SELECT rollno,name,computer_marks,city
FROM student_bvoc,student_clg
Natural join rollno;
```

CARTESIAN JOIN / CROSS JOIN

Syntax:
SELECT columname1, columname2, ... FROM tablename1, tablename2;

Eg:
```
SELECT rollno,name,english_marks,hobby
FROM student_bvoc,student_clg
Order by hobby;
```

Outer join:

Syntax :
SELECT table1.column, table2.column,..... FROM table1, table2
WHERE table1.column(+) = table2.column;

Eg:

```
SELECT rollno,name,father's_name,computer_marks
FROM student_bvoc,student_clg
Where student_bvoc.rollno (+)= student_clg.rollno;
```

SELF JOIN :

The self-join can be seen as a join of two copies of the same table.The table is not actually copied,but SQL performs the command as though it were.

Eg:

```
SELECT BVOC1.name "name" , BVOC2.name "grp_head"
FROM student_bvoc BVOC1,student_bvoc BVOC2
Where BVOC1.rollno= BVOC2.group_head;
```

Views

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

```
CREATE VIEW view_name AS
```

```
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| 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|
+----+----------+-----+-----------+----------+
```

Sub Queries

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

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement:

Subqueries 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:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1|Ramesh|35|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|
```