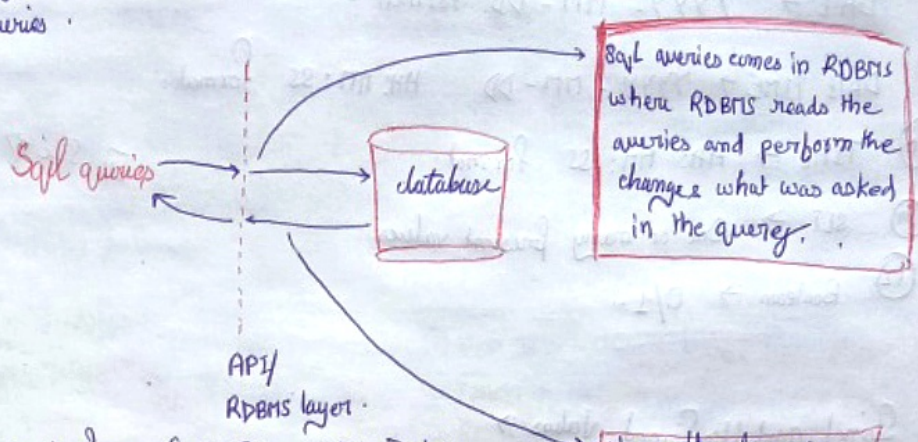
SQL

⊙ SQL => ⊡ SQL stands for structured Query Language.

⊡ It is a language that help us to Interact with the database using sql queries.



⊡ CRUD Operation => ⊡ CRUD stands from Create Read update Delete, which we do it to our data in database.

⊡ RDBMS => ⊡ Stands for Relational Database Management System.

⊡ Example of RDBMS software =>

(i) MySQL
(ii) MySQL Server.
(iii) Oracle.
(iv) IBM.

⊡

SQL	MySQL
(i) It is a Query Language.	(i) MySQL itself is a RDBMS.
(ii) It is a way to access data.	(ii) CRUD operation done here using SQL.

⊙ SQL datatypes =>

(1) CHAR => size (0-255). Size is fixed i.e. utilizes all space given even if not required.

(2) VARCHAR => size (0-255). Size is variable not fixed i.e. utilizes only that amount of space that is required.

(3) BLOB => used to store audio/video files in Bytes. Due to this the data of this files never gets corrupted.

(4) INT => 4 Bytes size.

⑤ FLOAT ⇒ can store decimal upto 23 digits precision.

⑥ DOUBLE ⇒ can store decimal with 24 to 53 digits precision.

⑦ DECIMAL ⇒ It is basically the double stored as strings.

⑧ DATE ⇒ YYYY-MM-DD format.

⑨ DATE TIME ⇒ YYYY-MM-DD HH:MM:SS format.

⑩ TIME ⇒ HH:MM:SS format.

⑪ SET ⇒ One or many present values.

⑫ Boolean ⇒ 0/1.

- Size ⇒ TINY < SMALL < MEDIUM INT < BIGINT.

① Signed and UnSigned Values ⇒

(i) Signed ⇒ -ve values present.

TINY INT ⇒ Size (-128 to 127)

- Using the unsigned ⇒

(ii) Unsigned ⇒ -ve values not present.
The space of negatives values are filled by positive values, hence increasing range of positive values.

```
create table <tablename> (
    col INT,
    col INT UNSIGNED,
);
```

UNSIGNED TINYINT = Size (0→255)

② Advance Datatypes ⇒

- JSON ⇒ (i) Stores values in key: value pairs.
(ii) Stands for Javascript Object Notation.
(iii) Example ⇒

Javascript Object Notation

```
create table <tablename>(
    col JSON,
);
```

① Types of Command in SQL →

① DDL (Data Definition Language)

defining the relation schema

- (i) CREATE → Creates table, DB and view.
- (ii) ALTER TABLE → modify table structure.
- (iii) DROP → delete, table, DB, view.
- (iv) TRUNCATE → remove all the tuples from table.
- (v) RENAME → rename DBNAME, table name, column name, etc.

② DRL/DQL (Data retrieval/Query Language)

retrieve data from tables

- (i) SELECT.

③ DML (Data Modification Language).

- (i) INSERT → insert data into relation.
- (ii) UPDATE → update relation data.
- (iii) DELETE → delete rows from relation.

to perform modification in DB.

④ DCL (Data Control Language)

- (i) GRANT → access priveleges to DB.
- (ii) REVOKE → revoke user access priveleges.

grants or revokes authorities from user.

⑤ TCL (Transaction Control LANguage).

- (i) START TRANSACTION → being a transaction.
- (ii) COMMIT → apply all changes and end transaction.
- (iii) ROLLBACK → discard changes and end transaction.
- (iv) SAVEPOINT → checkout within the group of transactions in which to rollback

manage transactions done in DB.

② DDL (Data Definition Language) →

① Creating a database → Create database if not exists <databasename>;

② Change current database → Use <databasename>;

③ Creating Tables → Create Table <tablename>(

```
    id INT Primary key,
    name varchar(255),
    roll INT,
);
```

} attributes/columns name.

④ Inserting values in table → Insert into <tablename> values (1, 'Rohit', 27); → Please give the values orderwise according to columns.

⑤ To display all rows of table → Select * from <tablename>;

⑥ Deleting database → Drop database if exists <dbname>;

⑦ Show list of all databases → Show databases;

⑧ Show all list of table present in database → show tables;

⊙ DRL (Data Retrieval Language) ⇒

(9) To see a particular column from a table ⇒ Select col1, col2, ... from <tablename>;
(Order of execution Right to Left.)

(10) Dual table concept (it enables us to use select without using from keyword with it.) ⇒
- (i) To perform calculation ⇒ Select 50+20;
- (ii) To get time of your server ⇒ Select now();
- (iii) To do all characters in lowercase ⇒ select lcase ('RohiT');
- (iv) To do all characters in uppercase ⇒ Select ucase ('Pandey');

(11) where clause ⇒ ⊡ provides a condition with Select in order to select rows which satisfy that particular condition.

⊡ Select * from <tablename> where <condition>;

⊡ Example ⇒ (i) Select * from customer where age > 18;
(ii) Select * from customer where gender = 'Male';

(12) Between clause ⇒ ⊡ Helps in selecting the rows from a table based on condition having some sort of range.

⊡ Example ⇒ Select * from customer where Age between 0 and 100;

(13) IN clause ⇒ ⊡ Helps in reducing OR condition.

⊡ Example ⇒ Select * from worker where dept = 'HR' OR dept = 'SDE' OR dept = 'MD';
⇓
Select * from worker where dept in ('HR', 'SDE', 'MD');

(14) AND, OR, NOT ⇒ ⊡ And and OR is seen above.

⊡ NOT Example ⇒ Select * from worker where dept not in ('HR', 'SDE');
↓
dept having 'HR' and 'SDE' are ignored here.

(15) Is NULL ⇒ ⊡ Used to view the row having null values in there column.

⊡ Example ⇒ Select * from Customers where pincode is null;

(16) Pattern Searching / wild card ⇒ ⊡ (i) % ⇒ Any number between 0 → n. space.
(ii) _ ⇒ only one space occupied.

⊡ patterns → (a) '% pa %' → ⊡ Rohit pa Rohit
⊡ Rohit pa.
⊡ pa Rohit

(b) '_ pa_' → ⊡ A pa B

⊡ Example →
Select * from worker where name like '_i%';
It is used to perform specific filtering.

(17) Sorting ⇒ ⊡ If you want to display the rows, in sorting order of their column values we use this techniques.

⊡ Example ⇒ (i) Select * from worker order by salary ASC; (extra required keyword.) Ascending order
(ii) Select * from worker order by salary DESC; descending order.

(18) DISTINCT ⇒ ⊡ To have rows which have distinct column/field values.
⊡ Example ⇒ Select distinct department from worker;

(19) Data Grouping ⇒ ⊡ If we donot use the Aggregation function by default Group by act as distinct.

Example ⇒ Select department from worker group by department;
(always same columns will be used both sides.)

⊡ Aggregation function + Group by ⇒

(i) Count() ⇒ ⊡ Shows distinct departments with their counts present.
⊡ Example ⇒ Select department, count(department) from worker group by department;

(ii) Avg() ⇒ ⊡ Shows distinct departments with their average salary.
⊡ Example ⇒ Select department, Avg(salary) from worker group by department;

(iii) MIN() ⇒ ⊡ distinct department ke jitna bhi same wala honge usme se min value leka, departments ke sath show karega.

Example ⇒ select department, MIN(salary) from worker group by department;

(iv) MAX() ⇒ ⊡ Same as above bass max value hoga.

Example ⇒ select department, MAX(salary) from worker group by department;

(v) Sum() ⇒ ⊡ Yaha sara distinct department display hoga, with sum of their same wala departments ka salary.

Example ⇒ select department, Sum(salary) from worker group by department;

(20) ⊙ Having ⇒ ⊡ It can be only used with groupby used for filtering inside groupby.

⊡ Example

select department, count(department) from worker groupby department having count(department)>1;

⇓

Here distinct departments whose count is >1 are displayed.

⊙ WHERE Vs Having

(i) filter row from table based on specific condition	(i) filter rows from groups based on specific condition.
(ii) used before groupby.	(ii) used after groupby.
(iii) Groupby not necessary.	(iii) Groupby is necessary.

⊙ Note

⊡ where can be used with select, update and delete.

while, groupby is only used with delete.

⊙ DDL

⊙ Primary key →

(21)

- (i) Should not be NULL.
- (ii) should be unique.
- (iii) only 1 primary key per table.
- (iv) Good method => Always keep primary keys as Integer.

▣ Example →

```
Create table Customer(
        id INT Primary key,
        name varchar(255)},
);
```

(22) Foreign key →

- ▣ It refers to primary key of the other table.
- Example =>

```
Create table customer(
        id int Primary key,
        name varchar(255),
);
Create table order_details(
        order_id int Primary key,
        delivery date,
        cust_id int,
        Foreign key(cust_id) references customer(cid),
);
```

(23) Unique

- ▣ Makes sure that a particular column in our tables must have unique values present.
- ▣ Example =>

```
Create table <table name>(
        name varchar(255) unique,
);
```

(24) Check => ▣ while creation of tables sets a particular condition, which should be followed by while inserting values in a table.

▣ Example =>

```
Create table <table name>(
        id int primary key,
        balance int,
        constraints balance_check check(balance > 1000),
);
```

(25) Default → ⊡ Sets a default value to the column if its value is not inserted.

⊡ Usecase ⇒ In Amazon prime if we enter as a new user it shows the prime status of new user as not prime.

⊡ Note ⇒ An attribute can be a primary and foreign key both in a table.

⊡ Example ⇒

```
create table <tablename>(
    id int primary key,
    name varchar(255) unique,
    balance INT NOT NULL Default 0,
);
```

→ default value set to 0.

(26) Alter operations →

(i) Add new column ⇒

```
Alter table <tablename> Add <colname> <col datatype> Not NULL
default 0;
```

(ii) Modify the table →

```
Alter table <tablename> Modify <colname> <col datatype> NOT NULL
default 0;
```

(new)

Note ⇒ To describe a table

```
Desc tablename;
```

(iii) Change column ⇒ (Used for renaming a column)

```
Alter table <tablename> change column <old colname> <new colname>
<new coldatatype> Not NULL default 0;
```

of new col

(iv) Drop column ⇒

```
Alter table <tablename> DROP column <column name>;
```

(v) Rename the table ⇒

```
Alter table <table name> rename to <newtable name>;
```

(27) Insert ⇒ 3ways ⇒

(i)
```
Insert into <tablename> (col1, col2, ....)
values (val1, val2, ....),
       (val1', val2', ....);
```

(ii) Insert into <tablename> values (val1, val2, ...);

(iii) Insert into <tablename> (col1, col2)
values (val1, val2);

→ column ke order mai values daliyo.

→ If all column are not to be assigned few values in a row, then we assign few values to the column using this, the rest other columns contains NULL values.

(28) Update →

(i) Updating 1 row →

Example ⇒ Update <tablename> set Address = 'Mumbai', gender = 'M' where id = 121;

updated values will be applied where id = 121

(ii) Updating multiple rows →

Example ⇒ Set SQL_Safe_updates = 0;
update <tablename> set <attribute> = value;

MySQL not allows updating multiple rows together, but this statement enables to do so

Ex ⇒ (i) update customer set pincode = 11000;
(ii) update customer set pincode = pincode+1;

(29) Delete →

- Delete all entries in a table → Set SQL_Safe_updates = 0;
  Delet from <tablename>;
- Deleting a particular row from a table → Delete from <tablename> where id = 121;
  (id = 121 ↓ condition)

Note →

Referential Constraints

(1) Insert constraints ⇒ we cannot insert a value into child table until value lie in parent table.

(2) Delete constraints ⇒ Value cannot be deleted from parent if the corresponding value is present in child value.

Two ways to remove this constraints - - - -

→ (i) On Delete Cascade ⇒ If we delete a row in parent table, then its corresponding row present in child table will also be deleted.

→ (ii) On Delete Set NULL ⇒ If we delete a row in parent table, then the foreign key of that corresponding row present in child table will be set to NULL.

Note → Q. Can a foreign key can have null value?
Ans. Yes

▣ Example ⇒

```
Create table <tablename>(
    order_id INT Primary key,
    cust_id INT,
    Foreign key (cust_id) references Customers (id) ON DELETE Cascade
);
```

⇓

ON DELETE SET NULL

(30) REPLACE ⇒

→ ▣ If data already present then replaces that data with new data.

→ ▣ If data not present, acts as Insert.

Example ⇒

(i) Replace into <tablename> (id, col1, col2) values (val1, val2, val3)

id dena necessary hai always as it helps to identify the row.

id ⇒ colo

OR,

(ii) Replace into <tablename> set id = val1, col1 = val2, col2 = val3;

⇓

colo

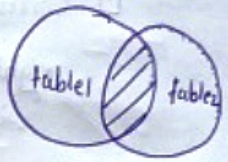
▣ Note ⇒ Replace vs. Update

If a row is not present, replace will add a new row, while update does nothing.

(31) Joins → ⊡ using Foreign keys we establish relation.
Now, using this relation how we fetch the data is done by joins in SQL.

(i) INNER JOIN →

⊡ To apply inner join, there must be common attributes/ among both parent and child table.
column



⊡ Syntax →

As → Aliasing:
Now we refer table1.col as T_1.col

```
Select T1.collist1, T2.collist2, .... from table1 as T1 innerjoin table2 as T2
on condition;
```

⊡ Example →

~~Select StudentCourse.courseId, student.name,~~

```
Select S1.courseId, S2.name, S2.age from student as S2 inner join
studentCourse as S1 on student.id = studentCourse.id;
```

(ii) Outer Join

(1) Left Join → ⊡ returns a resulting table that contains all data's from left table and matching data from right table.

⊡ Corresponding to non-matching data of left table right tables entries/fields/columns will have null values.

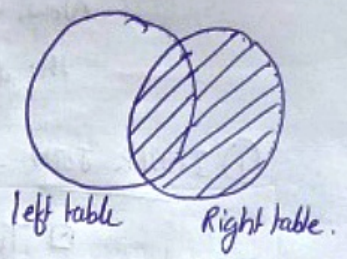
⊡ Example →

```
Select C.id, C.name, O.id, O.names from table1 as C Left join table2 as O
On C.id = O.cust_id;
```

different

② Right Join →

- Returns a resulting table that contains all data from right table and matching data from left table.
- Corresponding to non-matching data of right table, left tables columns/entries will contain NULL values.



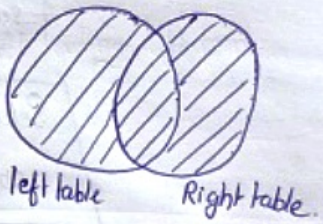
- Example →

```
Select C.id, C.name, O.id, O.names from table1 as C right join table2
as O on C.id = O.cust_id;
```

↓ different

③ Full Join →

- Returns a resulting table consisting of all datas from left as well as right table.
- In MySQL, there is no Full Join keyword so, we write query for Left and Right join and do its union.



- Left join query (Union) Right Join Query.

↓ extra used.

④ Cross Join →

- It is a cartesian Product.
- Example: Table1 (5 rows) × Table2 (10 rows) ⇒ 50 rows (Resultant table has.)
- Has no Industrial use.
- Example →

```
Select E.name, E.id, P.name from Employee as E
cross join Product as P;
```

↓ Here no condition is required.

5. Self Join →

- There is no keyword as Self Join. It is implemented by INNER Join with alias (AS).
- It is used to get output from a particular table when the same table is joined to itself.

- Example →

```
Select E1.id, E2.id, E2.name from employee as E1 inner join
employee as E2 on E1.id = E2.id;
```

- Note →

Q. Can we use join without using Join keyword?

ans. (Yes)

- Syntax →

```
Select e.id, e.name, p.id, p.name from employee as e,
project as p where e.id = p.id;
```

using , and where we have removed inner join.

- Set operations →
  1. Union
  2. Intersect
  3. Minus.

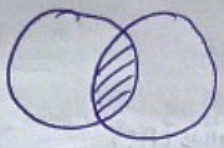
1. Union →

- It combines 2 or more select statements.
- Example →

```
Select * from table1 Union select * from table2;
```

- Number and order of column should be same for table1 and table2.

② Intersect ⇒

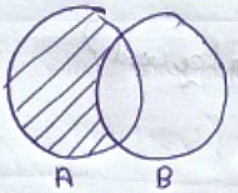


- Has no keyword / attribute · Implemented using Inner Join.
- Example ⇒

```
Select table1.* from table1 inner Join table2 using(id);
```

- `table1.*` → To avoid Increasing column we use this
- `using(id)` → as id will be present as same in both tables

③ Minus ⇒ (A-B)



- Has no keyword / attribute to use. Implemented using left join.
- Example ⇒

```
Select table1.* from table1 Left join table2 using(id)
where table2.id is null;
```

→ Same elements to removed.

④ Join Vs Set Operation ⇒

Join	Set Operation
① Columnwise combination	① Rowwise combination.
② No. of column ↑es	② No. of Row ↑es.
③ Can generate distinct and duplicate rows.	③ Generate distinct row.
④ Datatypes of 2 tables can be different.	④ Datatypes of corresponding columns of each table should be same.
⑤ Combines multible table based on matching condition.	⑤ Combination is resulting set of 2 or more select statements.

⑥ Example

~~$T_1 \cup T_2$~~

T_1 and T_2 (Full Join)

col1	col2	col1	col2
A	1	A	1
B	1	B	2
C	2	Null	Null
Null	Null	D	3

⑥ Example.

$T_1 \cup T_2$

col1	col2
A	1
B	1
C	2
B	2
D	3

T_1

col1	col2
A	1
B	1
C	2

Table1

T_2

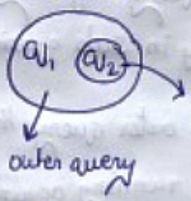
col1	col2
A	1
B	2
D	3

Table2

⊙ Sub Queries →

→ ⊡ alternative to joins.

→ ⊡ Sub query means nested queries.



⊡ Generally, outer query depends on inner query

⊡ Subqueries with where clause ⇒

Ex → employees with age > 30.

```
select * from employee where age in (select * from employee
                                     where age>30) ;
```

Subqueries single valued ⇒

- Example ⇒ Employee details with age > average (age).

```
Select * from employee where age > (select avg(age) from employe);
```

- returns a single value that is average age. So, called single valued subquery.

Subqueries with from clause ⇒

- Example ⇒ Select max age person who first name contains 'a'.

```
Select max(age) from (select * from employee where name like '%a') As temp ;
```

- used to refer always the table returned
- After from there should be a table always returned.

Correlated Subqueries ⇒

- It is inner query referring to outer query in sub queries.
- For 1 value of outer query the inner query is running again and again.

Example ⇒ find 3rd. oldest employee.

```
Select * from employee e1 where 3 = (select count (e2.age) from Employee e2
where e2.age >= e1.age);
```

dry run ⇒

age
32
44
22
31
41

e1.age (i) 32 ⇒ (2)
e1.age (ii) 44 ⇒ (1)
e1.age (iii) 22 ⇒ (4)
e1.age (iv) 31 ⇒ (3) ✓ Yeh age wala pura row will get return.

① Joins Vs. Subqueries ⇒

Joins	Subqueries
① Faster	① slower.
② maximise calculation burden on DBMS.	② Keep responsibility of calculation on user.
③ choosing optimal join for optimal use case is difficult.	③ Easy.
④ Difficult to understand and implement.	④ Easy to understand and implement.

⊡ SQL views ⇒

- ⊡ Views in SQL are virtual tables created by selecting fields/columns from one or more tables in database.
- ⊡ Creating view ⇒ `Create view <viewname> As select col1, col2, ... from <tablename>;`
- ⊡ Viewing View ⇒ `Select * from view_name;`
- ⊡ Altering view ⇒ `Alter view <viewname> As select col1, col2, col3 from <tablename>;`
- ⊡ Dropping the view ⇒ `Drop view if exists <view name>;`