**Database (W3School)**

**Database:** A database is a structured collection of data that is organized and stored in a computer system. It typically consists of one or more tables, each containing rows and columns of data, and is designed to efficiently manage and retrieve information.  
  
**DBMS**: A DBMS is a software system that provides an interface for users and applications to interact with a database. It typically includes tools for defining the structure of the data (schema), storing and retrieving data, enforcing constraints, and managing transactions.  
  
 types of dbms: 1)RDBMS   
 2) Object-oriented Database Management System (OODBMS)

**RDBMS** (Relational Database Management System): An RDBMS is a type of DBMS that stores data in a tabular format, with rows and columns organized in tables. It enforces a relational model for representing and manipulating data, where relationships between tables are established using keys. RDBMSs support SQL as the query language for interacting with the database.   
 EX: Examples of RDBMSs include PostgreSQL, MySQL, Oracle Database, Microsoft SQL Server, and SQLite.   
   
 data stored in database objects called 🡪 tables  
 each tables make relationship by -🡪 keys  
 vertical parts of the table (columns) 🡪 fields  
 horizontal parts called (roes) -🡪 records

**SQL(STRUCTURED QUERY LANGUAGE)**

\*\*SQL lets you access and manipulate databases.

\*\*SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.  
   
 **SQL features**  
 1) SQL is **not case- sensitive** 1) keywords 🡪 SELECT =select=SELect  
 2) database ,table, column names 🡪 employee=EMPLOYEE  
 NOTE: but data is case sensitive   
 ex: select \* from emp where name=’ram’ 🡪 name =’Ram’ NOT SAME

**datatypes**1) integer   
2) vachar 🡪 in single (‘ ‘)  
3) date 🡪 in single   
4)json  
5)text   
   
date : SQL standard format (‘YYYY-MM-DD’)   
 auto conversion by postgres  
 YYYY/MM/DD🡪 YYYY-MM-DD = 2000/01/01🡪2000-01-01  
 DD-MM-YYYY🡪 YYYY-MM-DD = 1-1-2000🡪2000-01-01  
  
  
  
  
  
  
  
  
  
  
  
 **Types of SQL queries  
1) DDL (Data Definition Language) Queries**:

DDL queries are used to define, modify, and delete database objects such as tables, indexes, and constraints.

Examples:

**CREATE** TABLE: Creates a new table.  
**ALTER** TABLE**:** Modifies an existing table's structure. (modifies the table structure not table content).

**DROP TABLE:** Deletes a table and its data.  
  
 note: to define , modify , remove databases/table and it structured.

**2) DML (Data Manipulation Language) Queries:**

DML queries are used to manipulate data stored in the database tables.(table level)

Examples:

**INSERT INTO**: Inserts new records into a table.   
**UPDATE:** Modifies existing records in a table.

**DELETE FROM:** Deletes records from a table. (to delete the records in table not table/database)

**3)DQL (Data Query Language) Queries:**

DQL queries are used to retrieve data from the database tables.

Example:

**SELECT**: Retrieves data from one or more tables.

**4) TCL (Transaction Control Language) Queries:**

TCL queries are used to manage transactions within the database.

Examples:

**COMMIT**: Saves changes made during the current transaction.

**ROLLBACK:** Reverts changes made during the current transaction.   
**5)DCL (Data Control Language) Queries:**

DCL queries are used to control access permissions on database objects.

Examples:

**GRANT:** Provides specific privileges to database users.

Ex: GRANT SELECT, INSERT ON Employees TO user1;  
 **REVOKE:** Revokes previously granted privileges from users.  
 EX : REVOKE SELECT ON Employees FROM user1;   
  
NOTE: CREATE,DROP,ALTER,INSERT INTO,UPDATE,DELETE,SELECT….. called statements   
 WHERE,LIMIT,JOINS … clause

**COMMENTS :**  1) single line comment (--) 🡪 -- single comment   
 ex: SELECT \* FROM emp; --display all emp table  
 2) multiline comments(/\* ….\*/) 🡪 common in all lanuguage  **DDL (Data Definition Language) Queries**:

DATABASE:

1)Syntax: CREATE DATABASE database\_name; 🡪 creates a database   
 ex: CREATE DATABASE MoviesDB;   
 2) SHOW DTABASES 🡪 returns list of databases available   
 3) DROP DATABASE databasename 🡪 deletes the database (all data in db will be lost)  
 ex: DROP DATABASE MoviesDB  
 4) BACKUP DATABASE databasename;

TO DISK = 'filepath'   
  
**TABLE :**

5) CREATE TABLE tablename(column1 datatype, column2 datatype..) ; 🡪 creates a new table   
 ex: CREATE TABLE Actors (id int, name vachar);   
6)DROP TABLE tablename ; 🡪 removes the table form the data base.  
7) TRUNCATE TABLE tablename 🡪 removes the data in table not the table itself.   
8) DESCRIBE table\_name 🡪 returns the table structure .  
9) SHOW TABLES; 🡪 returns list of tables.

8**) ALTER:** The ALTER TABLE statement is used to add, delete, or modify columns in an existing table. The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

1) add column   
 syntax: ALTER TABLE table\_name ADD column\_name datatype;   
 2)removing column   
 syntax: ALTER TABLE table\_name DROP COLUMN column\_name;  
   
 3)renaming column   
 syntax: ALTER TABLE table\_name RENAME COLUMN old\_name TO new\_name;  
 4) modifying column (modifying datatype)   
 syntax: ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;  
   
 5) rename table   
 syntax: ALTER TABLE current\_table\_name RENAME TO new\_table\_name;

**DML (Data Manipulation Language) Queries:**  
 1) **INSERT INTO:** to add new records into the table.   
   
 1) inserting all column records.(no need to specify column names)  
 syntax: INSERT INTO table\_name VALUES (value1,value2….);

2) inserting records of specified columns.

Syntax: INSERT INTO table\_name (column1,column3, column5) VALUES (v1,v2,v3);  
 note: the rest of the column cells are filled with NULL by default.

3) inserting multiple records   
 syntax: INSERT INTO table\_name VALUES (value1,value2….),(value1,value2,)…. ;  
 NOTE: same syntax just separate records with coma(,).

2) **UPDATE** : to modify the existing records in table. (SET & WHERE are used in UPDATE )  
  
 syntax: UPDATE table\_name SET column\_name1=’update value’ , column\_name2=’update

value’ WHERE condition;   
 ex: UPDATE movies SET movie\_name=’gami’ , movie\_date=’1-1-2024’ WHERE

movie\_Id=1;  
 NOTE: should use WHERE clause otherwise all records in table are updated

3)**DELETE** : used to delete the records from the table.( WHERE used)   
  
 syntax: DELETE FROM table\_name WHERE condition;  
 ex: DELETE FROM table\_name WHERE movie\_name=’gani’;  
 🡪 deletes all the records with movie\_name gami.  
 NOTE: should use WHERE clause otherwise all records in table are deleted   
   
 deleting all records  
 1) DELETE FROM table\_name;   
 (OR)  
 2)TRUNCATE TABLE table\_name;

**DQL (Data Query Language) Queries:**

**SELECT** : used to select/retrieve data from the database( FROM, WHERE )  
  
 syntax: SELECT column1,column2.. FROM table\_name;  
 ex: SELECT mivie\_id,movie\_date FROM Movies;  
  
1) select all table data   
 syntax: SELECT \* FROM table\_name;  
 2) select specified columns data (filtering columns)  
 syntax: SELECT col1,col2.. FROM table\_name;  
3) select specified column data with condition.(filtering records )  
 syntax: SELECT col1,col2.. FROM table\_name WHERE col1=10;   
  
  
NOTE: WHERE clause used in 1)UPDATE 2) DELETE 3) SELECT queries  
 condition( OPERATERS USED)   
  
 \*\* where is condition so accepts only Boolean values ( true / false)   
  
 **ONLY USED IN SELECT** :  
 1) SELECT distinct   
 2) ORDERBY   
 3) AGGREGATE FUNCTIONS   
 4)LIMIT (mysql) clause   
 5) ALIASES

1)SELECT DISTINCT : returns unique columns values (eliminates duplicate column records)  
 syntax: SELECT DISTINCT column\_name FROM table\_name;  
 🡪 returns the unique values from of the given column.   
 🡪 by default gives in ascending order distinct results.

2) ORDER BY -🡪 to select the sorted table. (assending (ASC)or deassending(DESC) by column)

A)syntax: SELECT \* FROM table\_name ORDERBY column\_name;  
 🡪 gives the table data in assending order by colum\_name.

B) syntax: SELECT \* FROM table\_name ORDERBY column\_name1, column\_name2;  
  
 🡪 sorting by column\_name1(priority) then on by column\_name2   
 ex:

|  |  |  |
| --- | --- | --- |
| Column\_name1(1st sort) | column\_name2(2nd sort) | column\_name3 |
| 1 | A . |  |
| 1 | B . |  |
| 2 | S |  |

C) syntax: SELECT \* FROM table\_name ORDERBY column\_name1 DESC;   
 🡪 gives the sorted table data by descending order by the colum\_name1.  
  
 D) syntax: SELECT \* FROM table\_name ORDERBY col1 ASC , col2 DESC;

Ex: SELECT \* FROM Customers ORDER BY Country ASC, CustomerName DESC;  
 🡪 using multiple column and different sortings.  
  
3) AGGREGATE functions

**SQL comparison operators**

Operator Description

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal. Note: In some versions of SQL this operator may be written as !=

**SQL LOGICAL OPERATOES**  
1) AND   
2)OR  
3)NOT   
4) LIKE Search for a pattern

5) BETWEEN Between a certain range

6) IN To specify multiple possible values for a column

**AGGREGATE FUNCTIONS   
 🡪** returns a single value

1) MIN() -- returns minimum value from column

2)MAX() -- returns maximum value from column

3)SUM() --applicable on number columns

4)AVG() --applicable on number columns  
5)COUNT() -- returns the count of the records( NULL also considered as record count)  
 ex: 1,2,3,NULL ,4,NULL count =6 (not 4)  
  
 1)MIN();  
  
 1) syntax : SELECT MIN(column\_name) FROM table\_name ;  
 ex: SELECT MIN(salary) FROM emp\_table;  
  
 🡪 returns the single value(minimum value of the salary column).  
 2) syntax: SELECT MIN(salary) FROM table\_name WHERE dept=sales;  
 3) syntax: SELECT MIN(salary) ,MAX(age) FROM table\_name;  
   
 RESULT:

|  |  |
| --- | --- |
| 1200 | 22 |

note: a single function takes only single argument and returns single value. BUT u can use multiple functions in single SELECT query.

NOTE: same as rest aggregate functions like MAX(),SUM(),AVG(),COUNT().

**LIMIT clause**

**LIMIT clause** : used to specify the number or records to be returned in select query.  
 syntax: SEECT column\_name(s) FROM table\_name LIMIT=number;  
 ex: SELECT salary FROM emp\_table LIMIT =3; 🡪 returns top 3 records only   
  
 **ALIASES**   
\*\* aliases are used to give a table / a column in a table a temporary name.

\*\*alias names exists for the duration of the query.(no effet the original table)  
\*\* used with the AS keyword (optional in some databases).  
  
 syntax: SELECT \* FROM table\_name AS aliases\_name; 🡪 table name aliases  
 ex: SELECT \* FROM employee AS emp;

Syntax: SELECT col\_1 AS alias\_name , col\_2 AS alias\_2 FROM table\_name;  
 ex: SELECT salary AS s , dateofbirth AS dob FROM emp;

If aliases name is group of words then enclose in “ “ or in [ ];   
 syntax: SELECT \* FROM employee AS [emp data table current] ;   
  
  
  
 JOINS

JOINS clause is used to combine rows from two or more tables based on a related column.  
🡪 two table must have the common column. (i.e ID in table1 and ID in table2).

🡪 selecting the columns is our choice in query but rows are the results of the join query.  
🡪 use Dot(.) operator to call the columns from the table.

syntax; table\_name.con\_name

ex: SELECT A.Age,B.Dept

TABLE:A TABLE:B

|  |  |  |
| --- | --- | --- |
| ID | Name | Age |
| 1 | sam | 20 |
| 2 | Joy | 22 |
| 3 | ayan | 34 |
| 4 | hamja | 28 |

|  |  |  |
| --- | --- | --- |
| Role | ID | Dept |
| jr | 1 | sales |
| sr | 2 | mark |
| sr | 4 | prod |
| jr | 5 | test |

Types of joins   
 1) INNER JOIN  
 2) LEFT JOIN ( left outer join)  
 3) RIGHT JOIN(right outer join)  
 4) FULL JOIN(full outer join)

INNER JOIN: the inner join returns only the matching records for the two tables.   
  
Syntax: SELECT table1.comun(s),table2.column(s) FROM table1 INNER JOIN table2

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

  
   
 ex: SELECT A.ID,A.Name,B.Role FROM A INNER JOIN B ON A.ID=B.ID;

|  |  |  |
| --- | --- | --- |
| ID | Name | Role |
| 1 | sam | sales |
| 2 | Joy | mark |
| 4 | hamja | prod |

Note: here only the matching records i.e 1,2,4 only returned rest ignoded(3,5).  
   
LEFT JOIN: returns the all records from the left table(table1) and only the matching records from the right table(tabke2)   
 

syntax: SELECT table1.comun(s),table2.column(s) FROM table1 LEFT JOIN table2

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

ex: SELECT A.ID,A.Name,B.Role FROM A LEFT JOIN B ON A.ID=B.ID;

|  |  |  |
| --- | --- | --- |
| ID | Name | Role |
| 1 | sam | sales |
| 2 | Joy | mark |
| 3 | ayan |  |
| 4 | hamja | prod |

Here : all the records from the left table(A) of selected columns and only matching records from the right table(B) are selected.   
 NOTE: non matching records from table B are empty   
   
RIGHT JOIN: returns the all records from the RIGHT table(table2) and only the matching records from the left table(tabke1).  
 

syntax: : SELECT table2.comun(s),table1.column(s) FROM table2 RIGHT JOIN table1

ON table2.common\_column= table1.common\_column;  
   
 ex: SELECT B.ID,B.Role,A.name FROM B RIGHT JOIN A ON B.ID=A.ID;

|  |  |  |
| --- | --- | --- |
| ID | Name | Role |
| 1 | sam | jr |
| 2 | joy | sr |
| 4 | hamja | sr |
| 5 |  | jr |

Here : all the records from the RIGHT table(B) of selected columns and only matching records from the LEFTtable(A) are selected.   
 NOTE: non matching records from table A are empty

FULL JOIN: returns the records from both the tables. Unmatched records are empty.  
 syntax: SELECT table1.comun(s),table2.column(s) FROM table1 FULL JOIN table2

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

|  |  |  |
| --- | --- | --- |
| ID | Name | Role |
| 1 | sam | sales |
| 2 | Joy | mark |
| 3 | ayan |  |
| 4 | hamja | prod |
| 5 |  | jr |

eg: SELECT A.ID,A.Name,B.Role FROM A FULL JOIN B ON A.ID=B.ID;   
   
 

note: here all the records from the both dabbles are covered. (id,1234 and 1245).  
 unmatched ID 3 and ID 5 also.  
  
  
 **GROUP BY**   
  
the group by statement groups the ROWS that have the same values in specified column.   
 \*\* all same valued rows are considered as a group.  
 ex: emp table

|  |  |  |
| --- | --- | --- |
| Id | Name | Gender |
| 1 | A | M |
| 2 | B | M |
| 3 | C | F |
| 4 | D | M |

Are one group

(if grouped by gender)  
  
 other group.  
  
  
 syntax: SELECT colm(s) FROM table-name GROUP BY column(s).  
 ex : SELECT gender FROM emp GROUP BY gender.   
 result : M  
 F   
can be used with aggregate functions and WHERE clause .   
  
with WHERE clause   
 syntax: SELECT gender FROM emp WHERE ID>3 GROUP BY gender;  
 result: M   
**with aggregate functions;** syntax: SELECT gender ,COUNT(names) GROUP BY gender.  
 result: F 1 (F first as returns in ascending order by default)  
 M 3

**🡪** in similar use MAX(),MIN(),AVG(),SUM() to get for each group.

Note: first the group by statement groups the rows with same gender(M-group,F-group)   
 then the count the names in each group. F-1, M-3,   
  
\*\* select a single and a aggregation function(). 🡪 not more than one column.

\*\* give the same column in the group by statement. (for easy).  
  
  
 **HAVING clause**   
 The having clause is introduced in SQL because of the WHERE clause can’t be used with aggregate function.  
\*\* you not allowed to use aggregate functions in WHERE clause. (we required to use when group by is used ). So we use HAVING clause.   
 used instead of WHERE clause.

Constraints :  
1) primary key   
2) foreign key

3)not null

4) unique  
  
  
**Primary key:** used to uniquely identify the record in the table.  
 \*\* a table can have more than one primary key.

\*\* it should be UNIQUE

\*\* it should be NOT NULL  
Syntax: CREATE TABLE emp( emp\_id SERIAL PRIMARY KEY , emp\_name VACHAR(20));

\*\* SERIAL data type is of integer and autoincrements

**Foreign key :** it is the reference key column to the other table.  
 it is used to establish the relationships between the tables.  
 \*\* a table can have more that 1 foreign keys.  
 \*\* REFERENCE key word is used   
  
  
Syntax: CREATE TABLE emp(emp\_id SERIAL PRIMARY KEY, emp\_name REFERENCE   
   
 reference\_table\_name( referencing column\_name );

Ex: CREATE TABLE emp(emp\_id SERIAL PRIMARY KEY, emp\_name REFERENCE   
 dept(emp\_name));

Not null: that column should not contain null values.  
  
 Syntax: CREATE TABLE table\_name( id int NOT NULL , name Vachar(20));  
  
unique: the column should not contain duplicate values.

Syntax: CREATE TABLE table\_name( id int UNIQUE , name Vachar(20));

**SQL with data type JSON**

* Create a column of JSON datatype   
  syntax: create table patient\_json( id serial primary key ,patient\_resource json);
* Inserting json data

Syntax: insert into patinet\_json (patient\_resource) values('{"name":"ram","age":20,"gender":"male"}');

insert into patinet\_json (patient\_resource) values('{"name":"sam","age":30,"gender":"male"}');

* Retrieving the json column data

Syntax; select \* from patinet\_json; -- all data

select patient\_resource->'name' from patinet\_json; --selecting only the specified key associated value from the json column of all rows.

ex: "ram"

"sam"

* Returns the json columns data into json array

Syntax: select json\_agg(patient\_resource) from patient\_json  
 ex: [{"name":"ram","age":20,"gender":"male"}, {"name":"sam","age":30,"gender":"male"}]

* Select record based on the json value.  
  syntax: select \* from patient\_json where patient\_resource->>'gender'='female';  
   ex: 4 {""name"":""navi"",""age"":22,""gender"":""female""} \

**PG ADMIN tool**

\*\* can run multiple queries at a time.( useful when using same kind of queries like insert)  
ex:  
  
note: 1 query (current) --- executed   
 2 query( error) --- not executed if you run 3 at a time .  
 3 query(correct) ---not executed (same as try block )

DATABASE LEVEL

\*\* using the PG admin you can import the query file(.sql extension). And we can upload the current query file(.sql file in our local system).  
  
\*\* we can import the result(output) as the excel sheet in local system.  
  
\*\* we can directly add / remove records(rows) from a table.

.