**Introduction**

**What is data?**

Any meaning full thing which is present around us in the form of Numbers, character or Physical objects is known as data.

**What is information?**

A collection of data is known as information.

**What are databases?**

Databases are system allow and store data in organized manner,

**Types of databases**

1. File-system database model
2. Hierarchal database model
3. Network database model
4. Relational database model

**1. File-system database model**

This model was used during the year of 1950 and 1960 where all the data of information was stored in the form of numerous physical file.

**Disadvantages**

* There is no relationship among themselves.
* Data redundancy was present.
* Time taken to search information is more.

**Uses**

* This model is recommended only if data to be maintained at a minimum.

**2. Hierarchal database model**

Here data or information is stored in the form of tree like a structure there is partial relationship between table and files.

**Disadvantages**

* Data redundancy was reduced bit.
* Data searching becomes faster. **Uses**
* This model is effective when comparing with filesystem model.

**3. Network database model**

* It is same like Hierarchal model here data is stored in the form of tree like structure and where the files will be connected to each other over the network

**Disadvantages**

* Maintaining the network is a little complicated.
* If any single node is failed, then entire network is get shutdown **Uses**
* Where data redundancy is reduced maximum
* Searching becomes faster.

**4. Relational database model**

* This model was introduced in the year of 1970 by Edgar cod
* In this model data or information is stored in the form of numerous table and tables will be connected with each other using primary and foreign key
* Compared to previous model this model is effective and efficient model

**Uses of databases**

* Data integrity
* Can handle massive amounts of data.
* Quickly combine different datasets
* Automate steps for re-use.
* Can support data for website and applications.

**Users in databases**

1. Analysts
   * Marketing
   * Business
   * sales
2. Technical
   * Data scientist
   * Software engineer
   * Web developer

**Data base platform**

* PostgreSQL
* MySQL
* MariaSQL
* MS SQL
* SQLite

**PostgreSQL ->That stores data and read query and returns information.**

Note

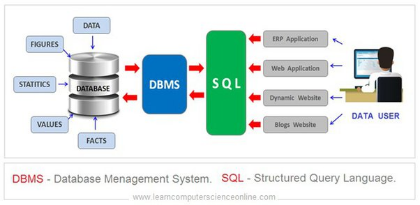
pgAdmin-> Graphical user interface for connecting with PostgreSQL

**SQL-> Structured query language or sequel query language**

**What is SQL?**

* Is a programming language used to communicate with the databases.

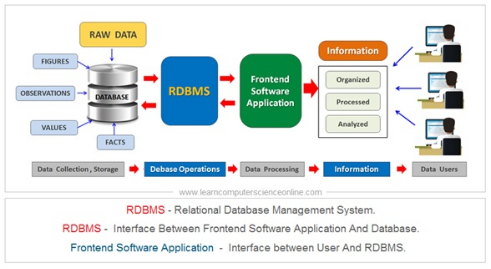
**DBMS**

[](https://github.com/vedantchimmad/SQL_Tutorial/blob/main/Image/DBMS.png)

* Data base management system
* Which is the software use to store and manage the data(modify, insertion, deletion)

Ex: foxpro,Foxbase, Dbase

**RDBMS**

[](https://github.com/vedantchimmad/SQL_Tutorial/blob/main/Image/RDBMS.png)

* Relational database management system
* It allow us to access the data more efficiently
* It manages and stores the data in the form of tables and enhanced security features, good performance and allow us to store huge volume of data

**SQL Basics**

**Data integrity**

* It is used to restrict invalid data entering the tables.

**Data integrity can be achieved in two ways.**

1. Data types
2. Constraints

**Data types**

* Data types are used to specify what type of data has to be stored in each column of the table

**The different type of datatypes**

1. Numeric
2. Non-numeric
3. Dates

**1. Numeric datatype**

**Number datatype**

* Which is used to store numeric values for the selected columns of the table

Syntax: Number(size/length)

**Number with precision datatype**

* Which is used to store numeric value along with the precision for the selected column of the table

Syntax: number(size, precision)

**2. Non-numeric datatype**

**Char data type**

* It is fixed in size which is used to store only the characters or alphanumeric values for the specific columns of the table

Syntax: char(size/length)

**Varchar/Varchar2 datatype**

* It is variable in size which is used to store only the character or alphanumeric values for the selected columns of the table
* Varchar data type can store up to 2000 values
* Varchar2 data type can store up to 4000 values

Syntax: varchar/varchar2(size/length)

**Date datatype**

* It is a type of datatype which is used to store data type of date into selected columns of the table
* The default format of the date datatype is presented by DD-MM-YYYY

**Constraints**

Constraints are used to restrict the invalid data entering into the table

1. Not null
2. Unique
3. Primary key
4. Foreign key
5. Check
6. Default

**Not null**

* It will ensure at least some value has to be present in the selected column of the table

Syntax:  
POSTGRESQL:  
-------------------------------  
CREATE TABLE Student

(

ID INT NOT NULL,

NAME VARCHAR(10) NOT NULL,

ADDRESS VARCHAR(20)

);

--------------------------------

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

ADDRESS varchar(20)

);

ALTER TABLE Emp modify Name Varchar(50) NOT NULL;

**Unique constraint**

* It will ensure that duplication of values are not allowed in the selected values of the column

Syntax:

CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20)

);

ALTER TABLE Instructor ADD UNIQUE(ID);

**Primary key**

* Primary key is a combination of unique and not null constraints
* Primary key is used to identify records uniquely
* Creating a primary key is not mandatory but it’s highly recommended to create
* only one primary key is allowed in the single table
* The columns which are eligible to become as primary key is called candidate key
* The column which are eligible to become as a primary key columns but not chosen as primary key Is called alternative key

CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20),

PRIMARY KEY(ID)

);

Alter Table Person add Primary Key(Id, Name);

ALTER table Person DROP PRIMARY KEY;

**Check**

* Whenever costumer asked for additional validation in the requirement then we make use of check constraints

Syntax:

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int NOT NULL CHECK (AGE >= 18)

);

**Foreign key**

* It Is also called as referential integrity constraint
* Used to create relationship between the tables
* To create relationship between the table the common column has to be present in both the tables
* More than one foreign key is allowed in single table
* Foreign key will accept duplication and null values

Syntax:

CREATE TABLE Orders

(

O\_ID int NOT NULL,

ORDER\_NO int NOT NULL,

C\_ID int,

PRIMARY KEY (O\_ID),

FOREIGN KEY (C\_ID) REFERENCES Customers(C\_ID)

)

**Default**

* This constraint is used to provide a default value for the fields.
* That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.

Syntax:

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int DEFAULT 18

);

ALTER TABLE tablename ALTER COLUMN columnname DROP DEFAULT;

**SQL COMMENTS**

**1. Single line comment**

* Comments starting and ending in a single line are considered single-line comments.

Syntax:

-- single line comment

**2. Multiline comment**

* Comments starting in one line and ending in different lines are considered as multi-line comments.

Syntax:

/\* multi line comment

another comment \*/

**3. In-line comments**

* In-line comments are an extension of multi-line comments, comments can be stated in between the statements and are enclosed in between ‘/*’ and ‘*/’.

Syntax:

SELECT \* FROM /\* Customers; \*/

**Aliases**

* Aliases are the temporary names given to tables or columns for the purpose of a particular SQL query.

Syntax:

SELECT column as alias\_name FROM table\_name;

Example: Advantages of SQL Alias

1. It is useful when you use the function in the query.
2. It can also allow us to combine two or more columns.
3. It is also useful when the column names are big or not readable.
4. It is used to combine two or more columns.

**SQL common Commands**

* **SHOW USER** : this command is used to know which user has been logged in to the database
* **CL SCR /Clear Screen** : this command is used to clear the entire scree of the page
* **CONN/ Connect** : this command is used to switch from one user to another without existing from the page
* **SELECT \* FROM TAB**; : this command is used to know the list of tables present in the database
* **DESC TABLE\_NAME** : used to know the structure of the table

**Operator**

* Operators are used to perform specific kind of operations on the operands based on the certain-conditions

**Types of operations**

**1.Arithmetic operations**

* Operators are used to perform arithmetic operations on operands

SELECT SAL\*12 ANUL\_SALARY FROM EMP;

Ex : +,-,\*,%

**2.Logical operators**

* Logical operators are used to perform logical operations on operands

SELECT \* FROM EMP WHERE EMPNAME =”VEDANT” AND EMPNAME=”VIVEK”;

Ex : and, or, not,SOME/ANY,ALL,EXISTS

**3.ALL Operator**

* ALL operator is used to select all tuples of SELECT STATEMENT. It is also used to compare a value to every value in another value set or result from a subquery.
* The ALL operator returns TRUE if all of the subqueries values meet the condition. The ALL must be preceded by comparison operators and evaluates true if all of the subqueries values meet the condition.
* ALL is used with SELECT, WHERE, HAVING statement.

Syntax:

SELECT column\_name(s)

FROM table\_name

WHERE column\_name comparison\_operator ALL

(SELECT column\_name

FROM table\_name

WHERE condition(s));

Q) Find the name of the product if all the records in the OrderDetails has Quantity either equal to 6 or 2.

Ans)

SELECT ProductName

FROM Products

WHERE ProductID = ALL++++++++ (SELECT ProductId

FROM OrderDetails

WHERE Quantity = 6 OR Quantity = 2);

**4. SOME Operator**

* SOME operator evaluates the condition between the outer and inner tables and evaluates to true if the final result returns any one row. If not, then it evaluates to false.
* The SOME and ANY comparison conditions are similar to each other and are completely interchangeable.

Ans)

SELECT NAME

FROM INSTRUCTOR

WHERE SALARY > SOME(SELECT SALARY

FROM INSTRUCTOR

WHERE DEPT='COMPUTER SCIENCE');

**5.EXISTS Operator**

* The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.
* The result of EXISTS is a boolean value True or False.
* It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

Q) Using EXISTS condition with SELECT statement To fetch the first and last name of the customers who placed atleast one order.

Ans)

SELECT fname, lname

FROM Customers

WHERE EXISTS (SELECT \*

FROM Orders

WHERE Customers.customer\_id = Orders.c\_id);

**6.Relational operators**

* Are used to perform relation between operands

SELECT \* FROM EMP WHERE SAL = 2000;

Ex :<,>, <=,>=,=,!=

**7.Special operators/ Keywords operators**

Ex: IS, LIKE, IN, BETWEEN

**IS Operator or keyword**

* Is operator is used to compare with null or not null values

Q) Display all the employees who don’t have reporting manager

SELECT \* FROM EMP WHERE MGR IS NULL

**Like operator**

* It is used to perform pattern matching
* We can achieve pattern matching in two ways with help of wildcards is % or \_

1. % It matches with single to N characters

Q)Display all the employee whose name starts with later S

SELECT \* FROM EMP WHERE ENAME LIKE S%;

1. \_ It matches with single character or values

Q)Display all the employee whose name is having letter L as second character

SELECT \* FROM EMP WHERE ENAME LIKE \_L%;

Note: LIKE is case sensitive and IS case-insensitive.

**IN Operator**

* Whenever we want to make use of the OR operator more than one time instead of that we can make use of the IN operator

Q)Display all the employees whose job is clear or the salesman

SELECT \* FROM EMP WHERE JOB IN (“CLEAR”,”SALESMAN”);

**BETWEEN Operator**

* It is used to display the output for the range of values
* Whenever we use of the between operator it should be followed by and operator

Q)Display the employees who is earning salary between the range of 1000 and 4000

SELECT \* FROM EMP WHERE SAL BETWEEN 1000 AND 4000;

**8.Concatenation Operator**

* || or concatenation operator is use to link columns or character strings.

SELECT id, first\_name, last\_name, first\_name || last\_name,

salary, first\_name || salary FROM myTable

**9.Alternative Quote Operator**

* we want to use apostrophe in our literal value but we can’t use it directly.
* overcome the problem Oracle introduce an operator known as Alternative Quote Operator(q).

Query that uses Alternative Quote Operator(q)

SELECT id, first\_name, last\_name, salary,

first\_name||q'{ has salary's }'||salary

AS "new" FROM myTable

**SQL Statements**

**DQL( Data query language)**

* SELECT

**DDL(data definition language)🡪 schema**

* CREATE
* ALTER
* RENAME
* TRUNCATE
* DROP

**DML(data manipulation language)->tuple**

* INSERT
* UPDATE
* DELETE

**DTL(data transaction language)**

* COMMIT
* ROLBACK
* SAVEPOINT

**DCL(data control language)**

* REVOKE
* GRANT

**DQL(data query language)**

* DQL statements are used to fetch or retrieve the data based on the query we passed in database.

**SELECT**

Syntax

SELECT \* /DISTINCT COLUMN\_NAME OR DISTINCT(COLUMN\_NAME)/EXPRESSION(ALIAS\_NAME) FROM TABLE\_NAME

**DISTINCT KEYWORD**

* It is helpful when there is a need to avoid duplicate values present in any specific columns/table.

SELECT DISTINCT \* FROM Table\_name;

**DDL(data definition language)**

* Data definition language actually used to define the database schema and structure of the table
* DDL is a set of SQL commands used to create, modify, and delete database structures but not data.

**CREATE**

* This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).

Syntax

CREATE DATABASE DATABSE\_NAME;

CREATE VIEW VIEW\_NAME;

CREATE TABLE TABLE\_NAME(COLUMN1 datatype(size) constraints, COLUMN2 datatype(size) constraints);

Q) create the tables Products and Orders Products Productid(pk) ProductName(not null) qty(check) Description Orders Productid(fk from product)OrderID(pk) Qty\_sold(check>0) price order\_DT

CREATE TABLE PRODUCTS(ProductID Number(4) Primary key,ProductName varchar(10) Not Null,QTY Number(3) check(Qty>0),Description varchar(20));

CREATE TABLE Orders(ProductID Number(4) reference products(productID),OrderID number(4) Primary key,QTY\_sold Number(5) check(Qty\_sold>0),price Number(5),ORDERDT date);

Q)create table from another table

CREATE TABELE DEMO AS SELECT \* FROM EMP

**ALTER**

* This is used to alter the structure of the database and there objects.
* This command is used to change the structure of the table I.e adding column, Removing column and renaming colmn

Syntax : To add column

ALTER TABLE TABLE\_NAME ADD COLUMN\_NAME datatype(size);

Syntax: To rename a column

ALTER TABLE TABLE\_NAME RENAME COLUMN COLUMN\_NAME TO NEW\_COLUMN\_NAME;

Syntax: to drop a column

ALTER TABLE TABLE\_NAME DROP COLUMN COLUMN\_NAME;  
 ALTER TABLE TABLE\_NAME DROP COLUMN (COLUMN1,COLUMN2);

**RENAME**

* This is used to rename an object existing in the database.

Syntax

RENAME TABLE\_NAME TO NEW\_TABLE\_NAME;

**TRUNCATE**

* This is used to remove all records from a table, including all spaces allocated for the records are removed.
* Used to remove all the records permanently but structure of the table is remains same

Syntax

TRUNCATE TABLE TABLE\_NAME;

**DROP**

* This command is used to delete objects from the database.
* This statement is used to remove both structure and data from the table.

Syntax

DROP TABLE TABLE\_NAME

Note

Oracle using recycle bin even though table is dropped we can restore back using flashback and drop permanently using purge command

FLASHBACK TABLE TABLE\_NAME TO BEFORE\_DROP;

PURGE TABLE TABLE\_NAME;

**DML(Data Manipulation language)**

* It is the component of the SQL statement that controls access to data and to the database.
* DML commands are not auto committed it means changes made are not permanent to database it can be rolled back

**Insert**

* This command is used to insert the data into a table

Synatx :

INSERT INTO TABLE\_NAME VALUES(V1,V2…); OR INSERT INTO TABLE\_NAME (COLUMN1,COLUMN2..) VALUES(V1,V2,….);

**Delete**

* Delete statement is used to delete the data from table
* We can also use to delete the particular row based on where clause
* Even after deleting the table the structure of the table is there
* The table can be restored after deletion

Syntax:

DELETE FROM TABLE\_NAME WHERE COLUMN\_NAME=VALUE; DELETE FROM TABLE\_NAME;

**Update**

* Update command is used to update one or more records from the table

Syntax

UPDATE TABLE\_NAME SET COLUMN\_NAME=VALUE WHERE CONDITION

| **DELETE COMMAND** |  | **TRUNCATE COMMAND** |
| --- | --- | --- |
| IS DML command |  | Is a DDL command |
| We can use where clause in delete |  | We can’t use where clause |
| We can use to delete particular row and all the row from table |  | Truncate statement is used to remove all the records permanently |
| Delete statement is slower than Truncate command |  | Truncate is faster than delete command |
| We can restore the data back |  | We can’t restore the data back |

**TCL(Transaction control language) OR DTL(data transaction language)**

* These commands are used to control the DML changes
* DML change on the table is not permanent one we need to save in order to make it as permanent
* We can undo(ignore) the same DML changes on a table

**ROLLBACK**

* It will undo the DML changes performed on the table

Synatax:

ROLLBACK; COMMIT

It will save the DML changes permanently

Syntax:

COMMIT; SAVEPOINT

From the point of save point we can Undo the changes of DML instead of whole changes

Syntax:

SAVEPOINT SAVEPOINT\_NAME;

ROLLBACK TO SAVEPOINT SAVE\_POINT\_NAME;

**DCL(Data control language)**

* Commands or statements are used to control the access and permission of the table

**GRANT**

* The statement is used to provide the access permission to the different user.

Syntax:

GRANT SELECT/UPDATE ON TABLE\_NAME TO USER\_NAME,….;

**REVOKE**

* This command is used to take back the given permission

Syntax:

REVOKE SELECT/UPDATE ON TABLE\_NAME FROM USER\_NAME,…;

**Merge statement**

* The MERGE command in SQL is actually a combination of three SQL statements: INSERT, UPDATE and DELETE. Syntax:

//.....syntax of MERGE statement....//

//you can use any other name in place of target

MERGE target\_table\_name AS TARGET

//you can use any other name in place of source

USING source\_table\_name AS SOURCE

ON condition (for matching source and target table)

WHEN MATCHED (another condition for updation)

//now use update statement syntax accordingly

THEN UPDATE

WHEN NOT MATCHED BY TARGET

//now use insert statement syntax accordingly

THEN INSERT

WHEN NOT MATCHED BY SOURCE

THEN DELETE;

Example:

/\* Selecting the Target and the Source \*/

MERGE PRODUCT\_LIST AS TARGET

USING UPDATE\_LIST AS SOURCE

/\* 1. Performing the UPDATE operation \*/

/\* If the P\_ID is same,

check for change in P\_NAME or P\_PRICE \*/

ON (TARGET.P\_ID = SOURCE.P\_ID)

WHEN MATCHED

AND TARGET.P\_NAME <> SOURCE.P\_NAME

OR TARGET.P\_PRICE <> SOURCE.P\_PRICE

/\* Update the records in TARGET \*/

THEN UPDATE

SET TARGET.P\_NAME = SOURCE.P\_NAME,

TARGET.P\_PRICE = SOURCE.P\_PRICE

/\* 2. Performing the INSERT operation \*/

/\* When no records are matched with TARGET table

Then insert the records in the target table \*/

WHEN NOT MATCHED BY TARGET

THEN INSERT (P\_ID, P\_NAME, P\_PRICE)

VALUES (SOURCE.P\_ID, SOURCE.P\_NAME, SOURCE.P\_PRICE)

/\* 3. Performing the DELETE operation \*/

/\* When no records are matched with SOURCE table

Then delete the records from the target table \*/

WHEN NOT MATCHED BY SOURCE

THEN DELETE

/\* END OF MERGE \*/

**Case statement or control statement**

* Are uses to query filtering and query optimization by carefully selecting tuples that match our requirements. Syntax:

CASE case\_value

WHEN when\_value THEN statement\_list

[WHEN when\_value THEN statement\_list] …

[ELSE statement\_list]

END CASE

Example:

SELECT CustomerName, Age,

CASE

WHEN Age> 22 THEN 'The Age is greater than 22'

WHEN Age = 21 THEN 'The Age is 21'

ELSE 'The Age is over 30'

END AS QuantityText

FROM Customer;

**View**

* Views in SQL are kind of virtual tables.
* We can create a view by selecting fields from one or more tables present in the database.  Syntax:

CREATE OR REPLACE VIEW view\_name AS

SELECT column1,column2,..

FROM table\_name

WHERE condition;

[!Note]:

we can perform all the operation as comparing to table but for update we need to follow some rules

**Roles**

* A role is created to ease the setup and maintenance of the security model.
* First DBA(data base administrate) create the role

Syntax:

CREATE ROLE ROLE\_NAME;

* Provide privileged access to the Created role

Syntax:

GRANT create table, create view TO manager;

* Provide manger access to the users

Syntax:

GRANT manager TO SAM, STARK;

* Revoke access from role

Syntax:

REVOKE create table FROM manager;

* Drop role

Syntax:

DROP ROLE manager;

**Clause**

**1.Where clause**

* WHERE keyword is used for fetching filtered data in a result set.

Syntax:

SELECT column1,column2 FROM table\_name WHERE column\_name operator value;

Q) fetch records of Employee with ages equal to 24.

SELECT \* FROM Emp1 WHERE Age=24;

**2.Using clause**

* USING Clause is used to match only one column when more than one column matches.
* NATURAL JOIN and USING Clause are mutually exclusive.
* NATURAL JOIN uses all the columns with matching names and datatypes to join the tables. The USING Clause can be used to specify only those columns that should be used for an EQUIJOIN.

Q) Write SQL query to find the working location of the employees. Also give their respective employee\_id and last\_name?

SELECT e.EMPLOYEE\_ID, e.LAST\_NAME, d.LOCATION\_ID

FROM Employees e JOIN Departments d

USING(DEPARTMENT\_ID);

**3.Intersect clause**

* As the name suggests, the intersect clause is used to provide the result of the intersection of two select statements.

Syntax:

SELECT column-1, column-2 ……

FROM table 1

WHERE…..

INTERSECT

SELECT column-1, column-2 ……

FROM table 2

WHERE…..

Example:

SELECT ID, Name, Bonus

FROM

table1

LEFT JOIN

table2

ON table1.ID = table2.Employee\_ID

INTERSECT

SELECT ID, Name, Bonus

FROM

table1

RIGHT JOIN

table2

ON table1.ID = table2.Employee\_ID;

**4.Except clause**

* contains all the rows that are returned by the first SELECT operation, and not returned by the second SELECT operation.

Syntax:

SELECT column-1, column-2 ……

FROM table 1

WHERE…..

EXCEPT

SELECT column-1, column-2 ……

FROM table 2

WHERE…..

Example:

SELECT ID, Name, Bonus

FROM

table1

LEFT JOIN

table2

ON table1.ID = table2.Employee\_ID

EXCEPT

SELECT ID, Name, Bonus

FROM

table1

RIGHT JOIN

table2

ON table1.ID = table2.Employee\_ID;

**5.Union clause**

* The Union Clause is used to combine two separate select statements and produce the result set as a union of both select statements.

[!NOTE:]

The fields to be used in both the select statements must be in the same order, same number, and same data type.

The Union clause produces distinct values in the result set, to fetch the duplicate values too UNION ALL must be used instead of just UNION.

Syntax:

SELECT column\_name(s) FROM table1

UNION

SELECT column\_name(s) FROM table2;

Example:

SELECT ROLL\_NO FROM Students UNION

SELECT ROLL\_NO FROM Student\_Details;

Syntax for UNION ALL:

* The resultant set consists of duplicatet values.

SELECT column\_name(s) FROM table1

UNION ALL

SELECT column\_name(s) FROM table2;

Example:

SELECT ROLL\_NO FROM Students

UNION ALL

SELECT ROLL\_NO FROM Student\_Details;

**6.Limit clause**

* The LIMIT clause is used to set an upper limit on the number of tuples returned by SQL.

Example 1:

SELECT \* FROM student LIMIT 3;

Example 2:

SELECT \*FROM Student ORDER BY ROLLNO LIMIT 5 OFFSET 2; offset means skipping values.

or

SELECT \*

FROM Student

ORDER BY ROLLNO LIMIT 2,5; # it skips the

first 2 values and then return the next 5 entries

Note

offset will skip first 2 rows

It can’t be used directly means we can use only in order by clause

**7.With clause**

* The clause is used for defining a temporary relation such that the output of this temporary relation is available and is used by the query that is associated with the WITH clause.
* The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.

Syntax:

WITH temporaryTable (averageValue) as

(SELECT avg(Attr1)

FROM Table)

SELECT Attr1

FROM Table, temporaryTable

WHERE Table.Attr1 > temporaryTable.averageValue;

Example:

Q) Find all the employee whose salary is more than the average salary of all employees.

WITH temporaryTable(averageValue) as

(SELECT avg(Salary) from Employee) SELECT EmployeeID,Name, Salary FROM Employee, temporaryTable WHERE Employee.Salary > temporaryTable.averageValue;

**8.OFFSET-FETCH Clause**

**Offset**

* The OFFSET argument is used to identify the starting point to return rows from a result set. Basically, it exclude the first set of records.

[!Note:]

OFFSET can only be used with ORDER BY clause. It cannot be used on its own.

OFFSET value must be greater than or equal to zero. It cannot be negative, else return error.

Syntax:

SELECT column\_name(s)

FROM table\_name

WHERE condition

ORDER BY column\_name

OFFSET rows\_to\_skip ROWS;

Example:

SELECT Fname, Lname

FROM Employee

ORDER BY Salary

OFFSET 1 ROWS;

**Fetch**

* The FETCH argument is used to return a set of number of rows.
* FETCH can’t be used itself, it is used in conjunction with OFFSET. Syntax:

SELECT column\_name(s)

FROM table\_name

ORDER BY column\_name

OFFSET rows\_to\_skip

FETCH NEXT number\_of\_rows ROWS ONLY;

Example:

SELECT Fname, Lname

FROM Employee

ORDER BY Salary

OFFSET 2 ROWS

FETCH NEXT 4 ROWS ONLY;

**9.With ties clause**

* With ties clause will return ties records also

Example:

SELECT \* from myTable

order by salary desc

fetch first 3 rows With Ties;

**10.Top clause**

* It will help to fetch top no of records

Syntax:

SELECT TOP value column1,column2 FROM table\_name;

value: number of rows to return from top

Example

SELECT TOP 1 \* FROM Customers

WHERE Country=’Spain’;

**Functions**

SQL functions are used to perform calculations on data and to modify the individual data items

**SQL functions are divided into 2 types**

1. Single Row functions
2. Multi row functions

**Single row functions**

* The no of input rows are equal to the no of outputs

**Numeric functions**

* which can be used to perform mathematical calculations.

**1. SQRT()**

* It takes any numeric values and returns the square root value of that number. Syntax:

SELECT SQRT(..value..)

Example:

SELECT SQRT(100)

**2.PI()**

* Using pi() function, value of PI can be used anywhere in the query.

Syntax:

SELECT PI()

Example:

SELECT PI()

**3.SQUARE()**

* SQUARE() function is used to find the square of any number.

Syntax:

SELECT SQUARE(..value..)

Example:

SELECT SQUARE(12)

**4.ROUND()**

* ROUND() function is used to round a value to the nearest specified decimal place.

Syntax:

SELECT ROUND(..value.., number\_of\_decimal\_places)

Example:

SELECT ROUND(125.315,2)

**5.CEILING()**

* CEILING() function is used to find the next highest value (integer).

Syntax:

SELECT CEILING(..value..)

Example:

SELECT CEILING(45.56)

**6.FLOOR()**

* FLOOR() function returns the next lowest value (integer).

Syntax:

SELECT FLOOR(..value..)

Example:

SELECT FLOOR(45.56)

**Character function**

**1.Case Manipulation**

**1.UPPER**

* Function is used to convert all the character in a string into Uppercase

Syntax:

UPPER(literal/Column\_name)

Example:

SELECT UPPER(“sql”) from dual;

**2.LOWER**

* Function is used to convert all the character in a string into lowecase

Syntax:

LOWER(literal/Column\_name)

Example:

SELECT LOwer(“SQL”) from dual;

**3.INITCAP**

* This function converts alpha character values to uppercase for the first letter of each word and all others in lowercase.

Syntax:

INITCAP(literal/Column\_name)

Example:

SELECT INITCAP('geeksforgeeks is a computer science portal for geeks') FROM DUAL;

**2.Character Manipulation**

**1.REPLACE**

* This function is used to replace substring from a given string

Syntax:

REPLACE(string, Substring to be replaced, New string)

Example:

SELECT REPLACE(“JAVA”, “J”, ”L”) from dual;

**2.SUBSTR**

* This function is used to extract the substring from a given string
* The extraction of a substring happens from LHS to RHS

Syntax:

SUBSTR(string, starting position, length(optional))

Example:

SELECT SUBSTR(“SPIDER” ,2,3) FROM DUAL;

**3.CONCAT**

* This function is used to merge the 2 literals or columns
* Concat function always accepts 2 parameter

Syntax:

CONCAT(STRING1,STRING2)

Example:

SELECT CONCAT(CONCAT(FNAME,SAL)JOB) FROM EMP

**4.LENGTH**

* This function is used to find the length of a given string

Syntax:

LENGTH(literals/Column\_name)

Example:

select length(ename) from emp

**5.TRIM**

* This function is used to remove the specified character from given string
* The deferent types of trim functions are

1. LTRIM
2. RTRIM
3. TRIM

**1. LTRIM**

* This function is used is used to remove all specified characters from the LHS of a given string

Syntax:

LTRIM(string, trim-string)

Example:

select LTRIM(“vedant”, “ev”) from dual;

**2. RTRIM**

* This function is used to remove the specified characters from the RHS of a given string

Syntax:

RTRIM(string, trim-string)

Example:

select RTRIM(“QSPIDER”, “QSP”) FROM DUAL;

**6.INSTR function**

* This function is used to find the position of a substring from a given string based on no of occurrence

Syntax:

INSTR(String, substring, starting position, no of occurrence)

Example:

SELECT( “QSPIDERS”, ”S”, 1,1) from dual

**Date function**

* The format of the data in the table must be matched with the input data to insert.
* DATE format YYYY-MM-DD
* DATETIME format: YYYY-MM-DD HH:MI: SS
* TIMESTAMP format: YYYY-MM-DD HH:MI: SS
* YEAR format YYYY or YY

**1.NOW()**

* Returns the current date and time

Syntax:

SELECT NOW()

**2.CURDATE() or CUR\_DATE()**

* Returns the current date

Syntax:

SELECT CURDATE()

**3.CURTIME() or CUR\_TIME()**

* Returns the current time

Syntax:

SELECT CURTIME()

**4.DATE()**

* Extracts the date part of a date or date/time expression. Example: For the below table named ‘Test’

Syntax:

SELECT Name, DATE(BirthTime)

AS BirthDate FROM Test;

**5.EXTRACT()**

* Returns a single part of a date/time.
* Several units can be considered but only some are used such as MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc. And ‘date’ is a valid date expression.

Syntax:

EXTRACT(unit FROM date);

Example:

SELECT Name, Extract(DAY FROM

BirthTime) AS BirthDay FROM Test;

**6.DATE\_ADD()**

* Adds a specified time interval to date

Syntax:

DATE\_ADD(date, INTERVAL expr type);

Example:

SELECT Name, DATE\_ADD(BirthTime, INTERVAL

1 YEAR) AS BirthTimeModified FROM Test;

**7.DATE\_SUB()**

* Subtracts a specified time interval from a date.

Syntax:

DATE\_SUB(date, INTERVAL expr type);

Example:

SELECT Name, DATE\_SUB(BirthTime, INTERVAL

1 YEAR) AS BirthTimeModified FROM Test;

**8.DATEDIFF()**

* Returns the number of days between two dates

Syntax:

DATEDIFF(date1, date2);

Example:

SELECT DATEDIFF('2017-01-13','2017-01-03') AS DateDiff;

**9.DAY()**

* It returns the day portion of a date value.

Syntax:

SELECT DAY("2018-07-16");

**10.DAYNAME()**

* It returns the weekday name for a date.

Syntax:

SELECT DAYNAME('2008-05-15');

**11.MONTHNAME()**

* It returns the full month name for a date.

Syntax:

SELECT MONTHNAME("2018/07/18");

**12.TO\_DAYS()**

* It converts a date into numeric days.

Syntax:

SELECT TO\_DAYS("2018-07-18");

**Null function**

* This functions are basically used to perform on null values
* IS NULL and IS NOT NULL Keywords are used to check null values

**1.ISNULL()**

* In SQL Server, ISNULL() function is used to replace NULL values.

Syntax:

SELECT column(s), ISNULL(column\_name, value\_to\_replace) FROM table\_name;

**2.IFNULL()**

* If the first argument is not NULL, the function returns the first argument. Otherwise, the second argument is returned.

Syntax:

SELECT column(s), IFNULL(column\_name, value\_to\_replace)

FROM table\_name;

**3.COALESCE()**

* COALESCE function in SQL returns the first non-NULL expression among its arguments.

Syntax:

SELECT column(s), CAOLESCE(expression\_1,….,expression\_n)

FROM table\_name;

Example:

SELECT Name, COALESCE(Phone1, Phone2) AS Contact

FROM Contact\_info;

**4.NULLIF()**

* The NULLIF function takes two arguments. If the two arguments are equal, then NULL is returned. Otherwise, the first argument is returned.

Syntax:

SELECT column(s), NULLIF(expression1, expression2)

FROM table\_name;

**Conversion function**

**I.Implicit data type conversion**

* In this type of conversion, the data is converted from one type to another implicitly (by itself/automatically).

Example:

SELECT employee\_id,first\_name,salary

FROM employees

WHERE salary > '15000';

**II.Explicit data type conversion**

* In this type of conversion, the data is converted from one type to another explicitly (by the user).

**1.TO\_CHAR()**

* TO\_CHAR function is used to typecast a numeric or date input to a character type with a format model (optional).

Example:

TO\_CHAR(number1, [format], [nls\_parameter])

**2.TO\_CHAR() with Dates**

* Can include any valid date format element in the query

Syntax:

TO\_CHAR(date, ’format\_model’)

Example:

SELECT employee\_id, TO\_CHAR(hire\_date, ’MM/YY’) Month\_Hired

FROM employees

WHERE last\_name = ’Higgins’;

**3.TO\_NUMBER()**

* Convert a character string to a number format

Syntax:

TO\_NUMBER(char[, ’format\_model’])

**4.TO\_DATE()**

* Convert a character string to a date format

Syntax:

TO\_DATE(char[, ’format\_model’])

**Aggregate functions**

* In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

**Various Aggregate Functions**

**1.Count()**

* It will return non null values

Syntax:

count(salary)

**2.Sum()**

* Returns sum of non null values

Syntax:

SUM(salary)

**3.Avg()**

* Return avg of non null values

Syntax:

Avg(salary)=Sum(salary) / count(salary)

**4.Min()**

* Minimum value in the salary column except NULL

Syntax:

Min(salary)

**5) Max()**

* Maximum value in the salary

Syntax: Max(salary)

**6)First()**

* The FIRST() function returns the first value of the selected column.

Syntax:

SELECT FIRST(column\_name) FROM table\_name;

**7)Last()**

* The LAST() function returns the last value of the selected column.

Syntax:

SELECT LAST(column\_name) FROM table\_name;

In SQL a Subquery can be simply defined as a query within another query. **Key points**

* A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.
* Subquery must be enclosed in parentheses.
* ORDER BY command cannot be used in a Subquery. GROUPBY command can be used to perform same function as ORDER BY command.

Syntax:

SELECT column\_name

FROM table\_name

WHERE column\_name expression operator

( SELECT COLUMN\_NAME from TABLE\_NAME WHERE ... );

Example 1:

Select NAME, LOCATION, PHONE\_NUMBER from DATABASE

WHERE ROLL\_NO IN

(SELECT ROLL\_NO from STUDENT where SECTION=’A’);

Example 2:

INSERT INTO Student1 SELECT \* FROM Student2

Example 3:

DELETE FROM Student2

WHERE ROLL\_NO IN ( SELECT ROLL\_NO

FROM Student1

WHERE LOCATION = ’chennai’);

Example 4:

UPDATE Student2

SET NAME=’geeks’

WHERE LOCATION IN ( SELECT LOCATION

FROM Student1

WHERE NAME IN (‘Raju’,’Ravi’));

**Co-related subquery**

* Each subquery is executed once for every row of the outer query.
* Correlated subqueries are used for row-by-row processing.

Example:

SELECT last\_name, salary, department\_id

FROM employees outer

WHERE salary >

(SELECT AVG(salary)

FROM employees

WHERE department\_id =

outer.department\_id group by department\_id);

**Grouping data**

**Group By**

* used to arrange identical data into groups with the help of some functions.
* Group by clause is used in the select statement to collect the records across multiple records group the result by one or more

**Features**

* The column which is present in the select statement should also be present in the group by clause.
* Where clause is used to restrict the non-grouped data and it should be present before the group by clause.
* Having clause is used to restrict the grouped data and it should be present in after the group by clause
* Where clause and having clause are optional and it is used only on a certain condition
* We can’t use group by function in where clause.
* Order by clause should be present always at the end of the query

Syntax:

SELECT column1, function\_name(column2)

FROM table\_name

WHERE condition

GROUP BY column1, column2

HAVING condition

ORDER BY column1, column2;

Example:

SELECT NAME, SUM(sal) FROM Emp

GROUP BY name

HAVING SUM(sal)>3000;

**Order by**

* used to sort the fetched data in either ascending or descending according to one or more columns.

**Rule**

* By default ORDER BY sorts the data in ascending order.
* We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

Syntax:

SELECT \* FROM table\_name ORDER BY column\_name ASC | DESC

OR

SELECT \* FROM table\_name ORDER BY column1 ASC|DESC , column2 ASC|DESC

Example:

SELECT \* FROM students ORDER BY Age ASC , ROLL\_NO DESC;

**Having clause**

* HAVING clause is used to apply a filter on the result of GROUP BY based on the specified condition.

Syntax:

SELECT col\_1, function\_name(col\_2)

FROM tablename

WHERE condition

GROUP BY column1, column2

HAVING Condition

ORDER BY column1, column2;

| **Having** | **Where** |
| --- | --- |
| In the HAVING clause it will check the condition in group of a row. | In the WHERE condition it will check or execute at each row individual. |
| HAVING clause can only be used with aggregate function. | The WHERE Clause cannot be used with aggregate function like Having |
| Priority Wise HAVING Clause is executed after Group By. | Priority Wise WHERE is executed before Group By. |

**Joins**

* SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

**Types of Join**

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN
5. NATURAL JOIN

**1.Inner join**

* The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied.
* It will fetch the common records from both the table

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

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

Example:

SELECT StudentCourse.COURSE\_ID, Student.NAME, Student.AGE FROM Student

INNER JOIN StudentCourse

ON Student.ROLL\_NO = StudentCourse.ROLL\_NO;

**2.Left join**

* his join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join.
* LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

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

Example:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**3.Right join**

* This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.
* RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

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

Example:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**4.Full join**

* FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN.
* The result-set will contain all the rows from both tables.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

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

Example:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**5.Self join**

* a self join is a regular join that is used to join a table with itself.

Syntax:

SELECT columns

FROM table AS alias1

JOIN table AS alias2 ON alias1.column = alias2.column;

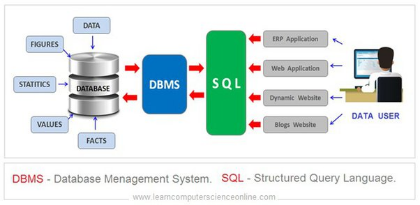
Example:

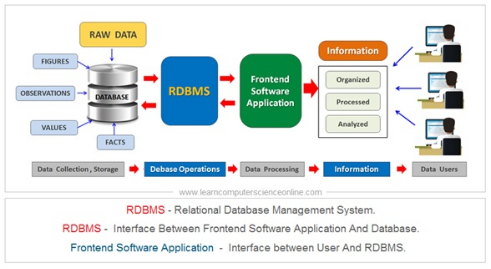
SELECT e.employee\_name AS employee,

m.employee\_name AS manager FROM

GFGemployees AS e JOIN GFGemployees

AS m ON e.manager\_id = m.employee\_id;



  
  
  
  
  
  
##########################################################################  
  
 **Few important topic:**

* 1. **What is protocol and how many types are there ?**

A **protocol** is a set of rules or standards that define how data is transmitted and received over a network or between devices. Protocols are crucial because they ensure that different systems can communicate with each other effectively, even if they are built by different vendors or run different software.

Protocols define:

* **How data is formatted** for transmission.
* **How data is compressed** or encrypted.
* **Error handling** mechanisms during transmission.
* The **start and stop points** of communication.

**Types of Protocols:**

Protocols can be categorized based on different layers of the **OSI (Open Systems Interconnection) model** or the **TCP/IP model**. Below are some of the most common types of protocols, categorized by function and layer.

**1. Network Layer Protocols (OSI Layer 3)**

These protocols handle the **routing of data** across the network.

* **IP (Internet Protocol)**: The foundational protocol for the internet, used to deliver packets of data from source to destination using IP addresses.
  + **IPv4**: The most widely used version, which uses 32-bit addresses.
  + **IPv6**: A newer version with 128-bit addresses, developed to overcome the limitations of IPv4.
* **ICMP (Internet Control Message Protocol)**: Used for error messages and operational queries, such as the ping command to test connectivity.
* **ARP (Address Resolution Protocol)**: Maps an IP address to a MAC (Media Access Control) address in a local network.

**2. Transport Layer Protocols (OSI Layer 4)**

These protocols ensure **end-to-end communication** between devices, including reliable data transfer, error checking, and flow control.

* **TCP (Transmission Control Protocol)**: A connection-oriented protocol that ensures reliable, ordered, and error-checked data delivery between applications. It is widely used for services like HTTP, FTP, and email.
* **UDP (User Datagram Protocol)**: A connectionless protocol that sends data without ensuring its delivery. It is used for real-time applications like video streaming or online gaming where speed is more critical than reliability.

**3. Application Layer Protocols (OSI Layer 7)**

These protocols define **high-level services** that applications use to communicate over the network.

* **HTTP (HyperText Transfer Protocol)**: Used for transmitting web pages and content over the internet. The secure version is **HTTPS** (HTTP Secure), which uses SSL/TLS encryption.
* **FTP (File Transfer Protocol)**: Used for transferring files between systems on a network. Its secure versions include **FTPS** (FTP Secure) and **SFTP** (SSH File Transfer Protocol).
* **SMTP (Simple Mail Transfer Protocol)**: The protocol used to send emails between mail servers.
* **IMAP (Internet Message Access Protocol)** and **POP3 (Post Office Protocol)**: Used for retrieving emails from a mail server. IMAP allows email syncing across devices, whereas POP3 downloads emails to a single device.
* **DNS (Domain Name System)**: Translates human-readable domain names (e.g., [www.example.com](http://www.example.com)) into IP addresses that computers use to identify each other.
* **SSH (Secure Shell)**: Provides secure command-line access to remote systems.

**4. Data Link Layer Protocols (OSI Layer 2)**

These protocols deal with **data transfer between adjacent network nodes**, like between a computer and a switch or router.

* **Ethernet**: The most commonly used LAN (Local Area Network) protocol that defines how devices in a network communicate over wired connections.
* **Wi-Fi (IEEE 802.11)**: A protocol for wireless networks, enabling devices to communicate without physical connections.
* **PPP (Point-to-Point Protocol)**: Used for establishing a direct connection between two nodes, typically used in older dial-up networks.

**5. Session Layer Protocols (OSI Layer 5)**

These protocols are responsible for establishing, managing, and terminating **communication sessions** between applications.

* **RPC (Remote Procedure Call)**: Allows a program to execute a procedure (function) on a remote server as if it were a local call.
* **PPTP (Point-to-Point Tunneling Protocol)**: A protocol used to implement VPN (Virtual Private Networks).

**6. Presentation Layer Protocols (OSI Layer 6)**

These protocols handle **data translation and encoding**, making sure that the data sent from one system is in a format that the receiving system can understand.

* **TLS (Transport Layer Security)** and **SSL (Secure Sockets Layer)**: Protocols for establishing encrypted and secure communication channels over the internet, used in HTTPS.
* **JPEG, PNG, GIF**: Image encoding protocols that define how images are compressed and transferred.

**Common Protocol Categories:**

1. **Communication Protocols**: Enable the exchange of data between devices.
   * **TCP/IP**: Foundational protocol suite for the internet.
   * **HTTP/HTTPS**: Web communication.
   * **SMTP/IMAP/POP3**: Email communication.
2. **Routing Protocols**: Help route data between networks.
   * **OSPF (Open Shortest Path First)**: Used in large enterprise networks for routing.
   * **BGP (Border Gateway Protocol)**: Manages how packets are routed across the internet.
3. **Network Management Protocols**: Help manage and monitor network devices.
   * **SNMP (Simple Network Management Protocol)**: Used to monitor and configure network devices.
4. **Security Protocols**: Provide secure data transmission.
   * **IPsec (Internet Protocol Security)**: Ensures secure IP communication by authenticating and encrypting each IP packet.
   * **SSL/TLS**: Secure internet communication.
   * **Kerberos**: Network authentication protocol for secure authentication.

########################################################################  
  
**2. How to connect to the DB using** **through a REST API?**

When interacting with a database through a REST API, you typically perform various **CRUD operations** (Create, Read, Update, Delete) using **HTTP methods** like GET, POST, PUT, PATCH, and DELETE. These HTTP methods map to database operations as follows:

**1. POST (Create)**

The POST method is used to **create a new resource** in the database. When you send a POST request to a REST API, you are essentially asking the server to add a new record to the database. The data you want to insert is usually sent in the request body in JSON format.

* **Use Case**: Inserting a new record (e.g., creating a new user, adding a new product).
* **Example**:
  + URL: POST /users
  + Request body:

json

Copy code

{

"name": "John Doe",

"email": "john.doe@example.com"

}

* + Database Operation: Insert a new row into the users table.

**2. GET (Read)**

The GET method is used to **retrieve data** from the database. This can be used to get a single record, a list of records, or filtered data based on query parameters.

* **Use Case**: Fetching data from the database (e.g., getting a user’s details, retrieving a list of products).
* **Example**:
  + URL: GET /users/1
  + Response:

json

Copy code

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com"

}

* + Database Operation: SELECT \* FROM users WHERE id = 1;

**3. PUT (Update/Replace)**

The PUT method is used to **update an existing resource** in the database. It typically replaces the entire resource with the new data provided in the request body. If the resource does not exist, some APIs will create it, but this behavior varies.

* **Use Case**: Updating an entire record (e.g., updating user information).
* **Example**:
  + URL: PUT /users/1
  + Request body:

json

Copy code

{

"name": "Jane Doe",

"email": "jane.doe@example.com"

}

* + Database Operation: UPDATE users SET name = 'Jane Doe', email = 'jane.doe@example.com' WHERE id = 1;

**4. PATCH (Partial Update)**

The PATCH method is used to **partially update a resource** in the database. Unlike PUT, which requires the entire object to be replaced, PATCH only changes the fields you specify in the request body.

* **Use Case**: Updating a specific field of a record (e.g., changing only the user’s email).
* **Example**:
  + URL: PATCH /users/1
  + Request body:

json

Copy code

{

"email": "jane.doe@example.com"

}

* + Database Operation: UPDATE users SET email = 'jane.doe@example.com' WHERE id = 1;

**5. DELETE (Delete)**

The DELETE method is used to **delete a resource** from the database. When a DELETE request is made, it typically removes the specified record from the database.

* **Use Case**: Deleting a record from the database (e.g., removing a user or product).
* **Example**:
  + URL: DELETE /users/1
  + Database Operation: DELETE FROM users WHERE id = 1;

**Example REST API Requests with Database Mapping:**

1. **POST** Request - Create a New Record:
   * HTTP Request:

http

Copy code

POST /api/products

Content-Type: application/json

{

"name": "Laptop",

"price": 1200,

"category": "Electronics"

}

* + **Database Query**: INSERT INTO products (name, price, category) VALUES ('Laptop', 1200, 'Electronics');

1. **GET** Request - Retrieve a Record:
   * HTTP Request:

http

Copy code

GET /api/products/101

* + **Database Query**: SELECT \* FROM products WHERE id = 101;

1. **PUT** Request - Update a Record (Replace Entire Record):
   * HTTP Request:

http

Copy code

PUT /api/products/101

Content-Type: application/json

{

"name": "Gaming Laptop",

"price": 1500,

"category": "Electronics"

}

* + **Database Query**: UPDATE products SET name = 'Gaming Laptop', price = 1500, category = 'Electronics' WHERE id = 101;

1. **PATCH** Request - Partial Update (Modify Certain Fields):
   * HTTP Request:

http

Copy code

PATCH /api/products/101

Content-Type: application/json

{

"price": 1400

}

* + **Database Query**: UPDATE products SET price = 1400 WHERE id = 101;

1. **DELETE** Request - Remove a Record:
   * HTTP Request:

http

Copy code

DELETE /api/products/101

* + **Database Query**: DELETE FROM products WHERE id = 101;

**REST API and Database Interaction:**

* When building REST APIs, the backend interacts with the database to perform these CRUD operations.
* A REST API is often used as a layer between the client (front-end or other systems) and the database. The API handles requests from clients, processes them (with business logic if necessary), and communicates with the database to either read, write, update, or delete data.

###########################################################################

* 1. **Dango with CRUD operation?**

### 4. ****Creating Views for CRUD Operations****

Django’s views handle the logic behind rendering templates and managing data. Let's create views for the CRUD operations in myapp/views.py.

#### a. ****Create a New Record (Create)****

To create a new book record, we will handle a **GET** request to show the form and a **POST** request to submit the form.

from django.shortcuts import render, redirect

from .models import Book

from .forms import BookForm

def create\_book(request):

if request.method == 'POST':

form = BookForm(request.POST)

if form.is\_valid():

form.save()

return redirect('book\_list')

else:

form = BookForm()

return render(request, 'create\_book.html', {'form': form})

#### b. ****Read Records (List)****

This view will retrieve and display all books from the database.

def book\_list(request):

books = Book.objects.all()

return render(request, 'book\_list.html', {'books': books})

#### c. ****Update a Record (Update)****

To update a record, we need to fetch the existing data and update it through a form submission.

def update\_book(request, pk):

book = Book.objects.get(id=pk)

if request.method == 'POST':

form = BookForm(request.POST, instance=book)

if form.is\_valid():

form.save()

return redirect('book\_list')

else:

form = BookForm(instance=book)

return render(request, 'update\_book.html', {'form': form})

#### d. ****Delete a Record (Delete)****

To delete a book, we will confirm the deletion and then remove it from the database.

def delete\_book(request, pk):

book = Book.objects.get(id=pk)

if request.method == 'POST':

book.delete()

return redirect('book\_list')

return render(request, 'delete\_book.html', {'book': book})

### 5. ****Creating Forms****

In Django, forms are used to handle input from users. Let's create a form for the Book model in myapp/forms.py:

from django import forms

from .models import Book

class BookForm(forms.ModelForm):

class Meta:

model = Book

fields = ['title', 'author', 'published\_date']

### 6. ****Creating Templates****

Create templates for displaying forms and lists. You'll store these HTML templates in a directory called templates/myapp.

#### a. ****Template for Listing Books (****book\_list.html****)****

html

Copy code

<!DOCTYPE html>

<html>

<head>

<title>Books List</title>

</head>

<body>

<h1>Books</h1>

<a href="{% url 'create\_book' %}">Add New Book</a>

<ul>

{% for book in books %}

<li>

{{ book.title }} by {{ book.author }}

<a href="{% url 'update\_book' book.id %}">Edit</a> |

<a href="{% url 'delete\_book' book.id %}">Delete</a>

</li>

{% endfor %}

</ul>

</body>

</html>

#### b. ****Template for Creating/Updating a Book (****create\_book.html****,**** update\_book.html****)****

html

Copy code

<!DOCTYPE html>

<html>

<head>

<title>Add Book</title>

</head>

<body>

<h1>Add New Book</h1>

<form method="POST">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Save</button>

</form>

</body>

</html>

#### c. ****Template for Deleting a Book (****delete\_book.html****)****

html

Copy code

<!DOCTYPE html>

<html>

<head>

<title>Delete Book</title>

</head>

<body>

<h1>Delete Book: {{ book.title }}</h1>

<p>Are you sure you want to delete this book?</p>

<form method="POST">

{% csrf\_token %}

<button type="submit">Yes, delete</button>

</form>

<a href="{% url 'book\_list' %}">Cancel</a>

</body>

</html>

### 7. ****Configuring URLs****

Now, wire up your views to the URLs. In myapp/urls.py, create the routes for each CRUD operation:

from django.urls import path

from . import views

urlpatterns = [

path('', views.book\_list, name='book\_list'),

path('create/', views.create\_book, name='create\_book'),

path('update/<int:pk>/', views.update\_book, name='update\_book'),

path('delete/<int:pk>/', views.delete\_book, name='delete\_book'),

]

Include these myapp/urls.py in the main project’s urls.py file:

python

Copy code

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('myapp.urls')),

]

### 8. ****Running the Project****

Finally, start the Django development server:

python manage.py runserver

Navigate to http://127.0.0.1:8000/ in your browser to access the CRUD app.