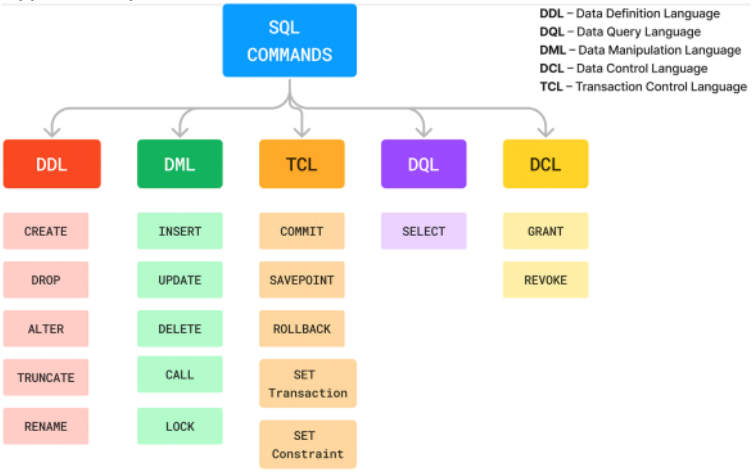
SQL

Saturday, January 4, 2025 11:36 AM

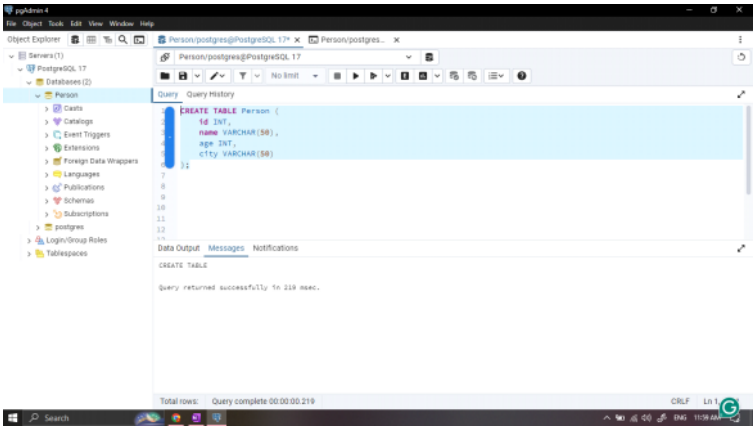
What is SQL?

- SQL stands for structured query language.
- Sql Is declarative language.
- SQL stands for Structured Query Language, a computer language that allows users to access and manipulate databases.

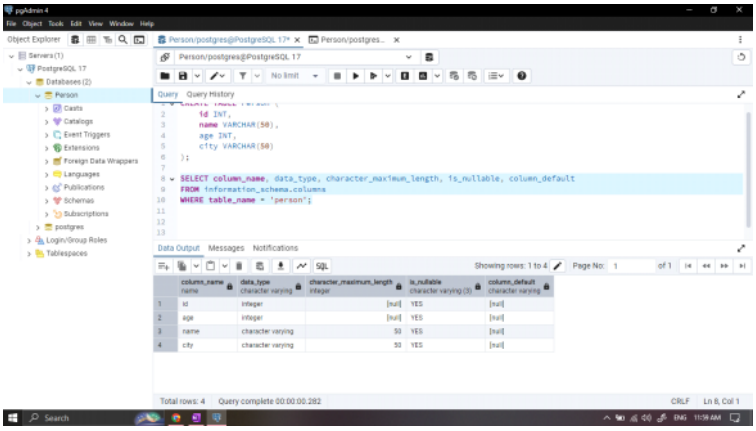
Types of sql command



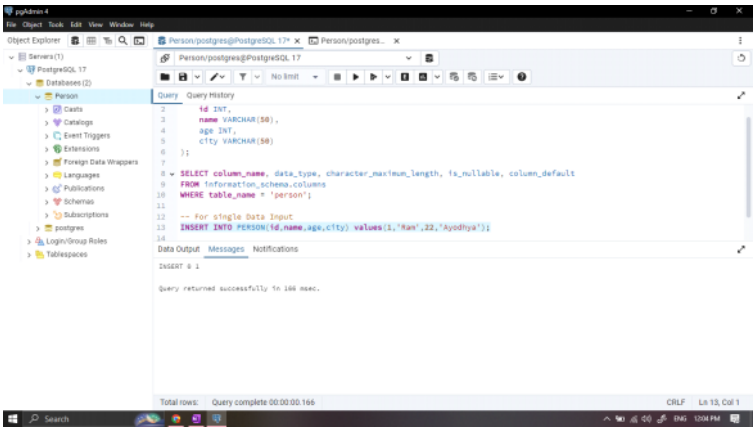
1] Create :

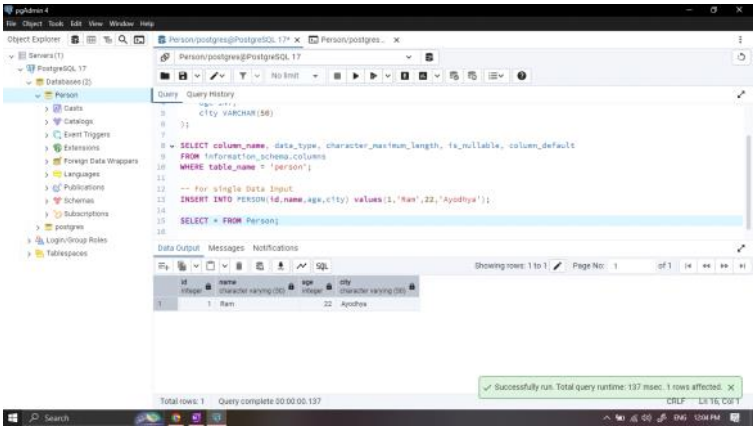


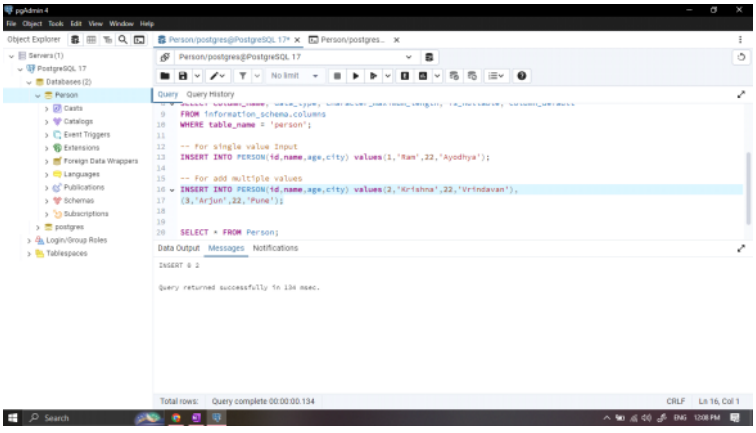
Describe Table :

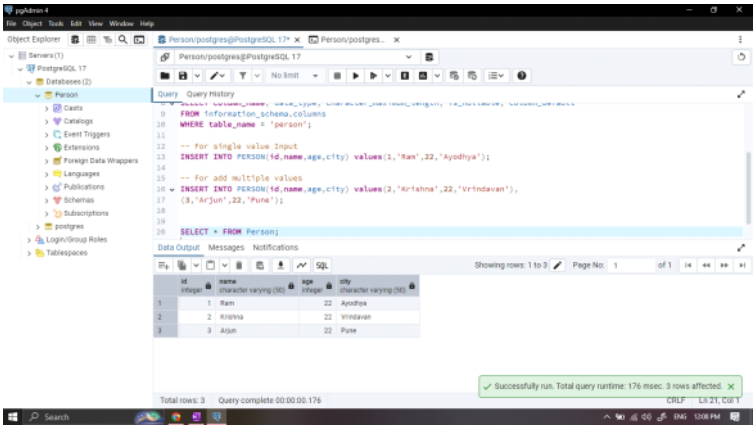


2] Insert : To add data in table

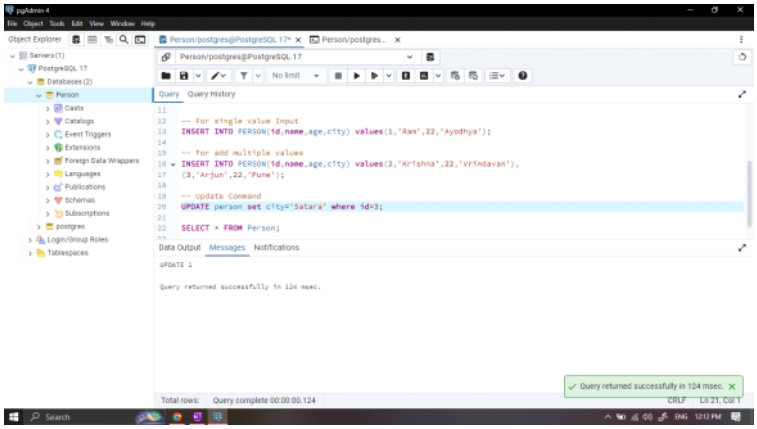


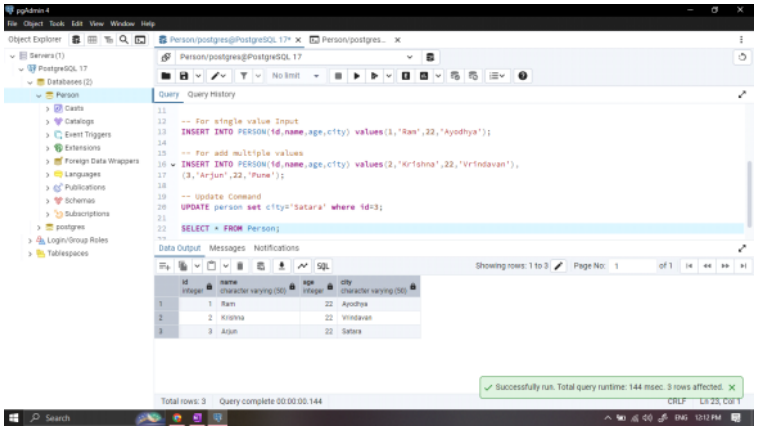




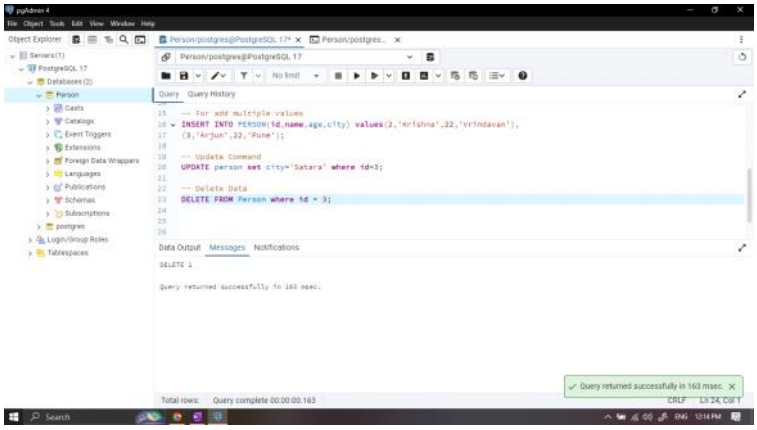


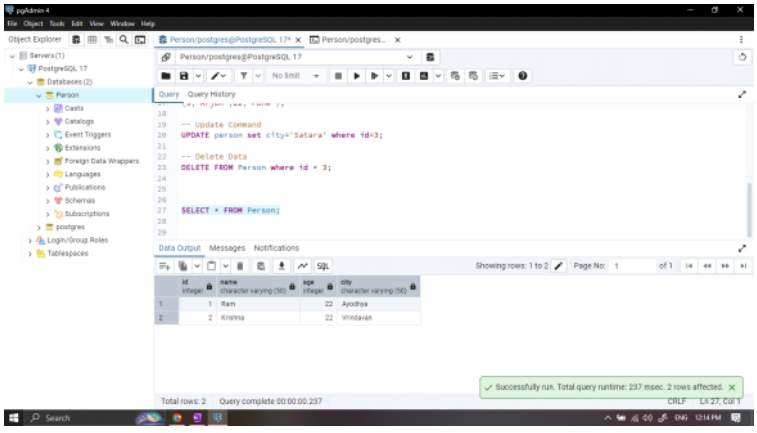
3] Update : Used to changes in available data



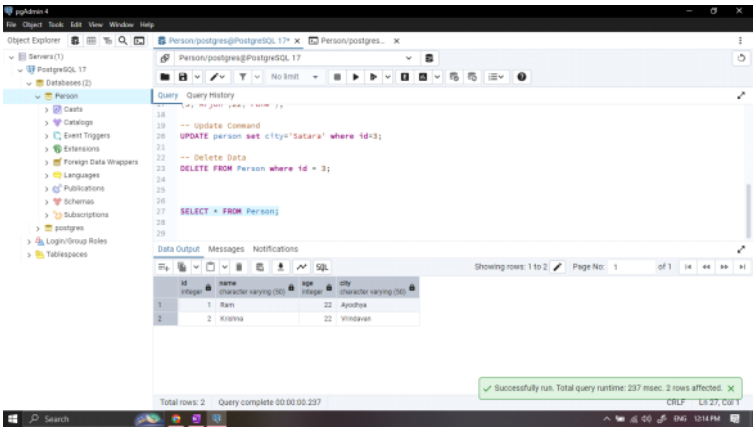


4] Delete command





5] Read :(select)



6] Alter : Alter is used to changes in schema
For e.g delete table,add column,remove column,add constraints,modify datatype/column_name,remove constraints,..

-- Add column
ALTER TABLE Person add column gender varchar(10);
-- Remove column
ALTER TABLE Person drop column gender;
-- Modify data type
ALTER TABLE Person Alter column age::INT;
-- Modify column name
Alter Table Person rename column id to pid;
-- rename table name
Alter Table Person rename to p;
-- Add constraints
Alter Table p add primary key(pid);

7] Drop : Used to delete table
Drop table table_name

- Difference between Alter and Update:

SR.NO	ALTER Command	UPDATE Command
1	ALTER command is Data Definition Language (DDL).	UPDATE Command is a Data Manipulation Language (DML).
2	Alter command will perform the action on structure level and not on the data level.	Update command will perform on the data level.
3	ALTER Command is used to add, delete, modify the attributes of the relations (tables) in the database.	UPDATE Command is used to update existing records in a database.
4	ALTER Command by default initializes values of all the tuple as NULL.	UPDATE Command sets specified values in the command to the tuples.
5	This command make changes with table structure.	This command makes changes with data inside the table.
6	It works on the attributes of a relation.	It works on the attribute of a particular tuple in a table.
7	Example : Table structure, Table Name, SP, functions etc.	Example : Change data in the table in rows or in column etc.

- Difference between Drop, Delete, Truncate

Delete	Truncate
The DELETE command is used to delete specified rows(one or more).	While this command is used to delete all the rows from a table.
It is a DML(Data Manipulation Language) command.	While it is a DDL(Data Definition Language) command.
There may be a WHERE clause in the DELETE command in order to filter the records.	While there may not be WHERE clause in the TRUNCATE command.
In the DELETE command, a tuple is locked before removing it.	While in this command, the data page is locked before removing the table data.
The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row.	TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.
DELETE command is slower than TRUNCATE command.	While the TRUNCATE command is faster than the DELETE command.
To use Delete you need DELETE permission on the table.	To use Truncate on a table we need at least ALTER permission on the table.
The identity of the fewer column retains the identity after using DELETE Statement on the table.	Identity the column is reset to its seed value if the table contains an identity column.
The delete can be used with indexed views.	Truncate cannot be used with indexed views.
This command can also active trigger.	This command does not active trigger.
DELETE statement occupies more transaction spaces than Truncate.	Truncate statement occupies less transaction spaces than DELETE.
Delete operations can be ROLLED back.	TRUNCATE cannot be Rolled back as it causes an implicit commit.
Delete doesn't DROP the whole table. It acquires a lock on table and starts deleting the rows.	TRUNCATE first drops the table & then re-create it, which is faster than deleting individual rows.

Constraint	**Purpose**
NOT NULL	Prevents NULL values in a column.
UNIQUE	Ensures unique values in a column.
PRIMARY KEY	Unique identifier for each row.
FOREIGN KEY	Ensures referential integrity between tables.
CHECK	Validates data based on a condition.
DEFAULT	Sets a default value for a column.
INDEX	Optimizes query performance.
AUTO_INCREMENT	Automatically generates sequential values.
COMPOSITE KEY	Combines multiple columns as a primary key.
EXCLUSION (PGSQL)	Prevents overlapping values based on conditions.

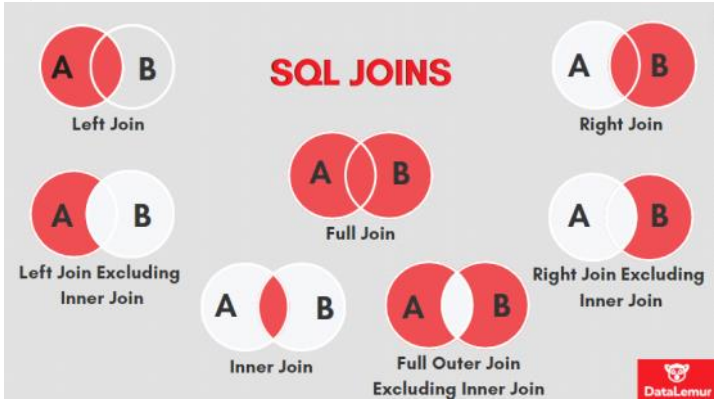
Joins

Saturday, January 4, 2025 9:55 AM

What is Join?

A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

Types of join :

- 

Note :

Join :- cross product + condition statement(Select statement)
Common attributes are necessary

1. Natural Join :

- **Common Column(s) with the Same Name**:
  - Both tables must have at least one column with the same name.
  - Example:
    - **Table1**: emp_id, dept_id
    - **Table2**: dept_id, dept_name
    - dept_id is the common column
- **Same Data Type for Common Columns**:
  - The columns with the same name must have compatible data types (e.g., both INT, both VARCHAR, etc.).
  - Mismatched data types will cause an error.
- **Primary Key-Foreign Key Relationship (Recommended)**:
  - Typically, one table's common column is a primary key, and the other table's common column is a foreign key.
  - This ensures that rows in one table match rows in the other.
- **No Extra Columns with Unintended Matches**:
  - Avoid using **natural join** if multiple columns have the same name but unrelated meanings. This can cause incorrect joins.

```
SELECT *
FROM table1
NATURAL JOIN table2
WHERE condition;
```

2. Self Join :

Conditions for a Self Join:

1. **Table Must Have Related Data**:
   - The table must contain data where some rows can logically be related to others.
2. **Use of Aliases**:
   - Aliases are mandatory to differentiate between the two instances of the same table in the query.
3. **Valid Relationship**:
   - A column (or a combination of columns) must exist to define the relationship between rows in the table.
4. **Join Condition**:
   - You need a meaningful condition in the ON clause to specify how rows from one instance of the table relate to

rows in the other instance

```
SELECT a.column_name, b.column_name
FROM table_name a
JOIN table_name b
ON a.common_column = b.common_column;
```

3. Left join :A **LEFT JOIN** (or **LEFT OUTER JOIN**) retrieves all rows from the left table and the matching rows from the right table. If no match is found, NULL values are returned for columns from the right table.

   Conditions for a LEFT JOIN:

   1. **Two Tables with a Logical Relationship**:
      - The two tables should have a relationship that allows meaningful comparisons, typically involving a foreign key-primary key relationship.
   2. **Join Condition**:
      - A valid condition in the ON clause is required to define how rows from the left table relate to rows in the right table.
      - Example: left_table.common_column = right_table.common_column.
   3. **No Restrictions on Matching Rows**:
      - A LEFT JOIN always includes all rows from the left table, regardless of whether a match exists in the right table.
   4. **Nullable Columns from the Right Table**:
      - Columns from the right table may contain NULL values for rows that do not have a matching entry.
   5. **Filtering (Optional)**:
      - You can add a WHERE clause to filter rows further, but be cautious to avoid unintentionally converting the LEFT JOIN into an INNER JOIN (e.g., avoid filtering on NULLs directly).

```
SELECT Employees.emp_id, Employees.emp_name, Departments.dept_name
FROM Employees
LEFT JOIN Departments
ON Employees.dept_id = Departments.dept_id;
```

4] Right Join:A **RIGHT JOIN** (or **RIGHT OUTER JOIN**) retrieves all rows from the **right table** and the matching rows from the **left table**. If no match is found, NULL values are returned for columns from the left table.

Conditions for a RIGHT JOIN:

1. **Two Tables with a Logical Relationship**:
   - The two tables should have a meaningful relationship, often based on a primary key and foreign key.
2. **Join Condition**:
   - A valid condition in the ON clause is required to define how rows from the right table relate to rows in the left table.
   - Example: left_table.common_column = right_table.common_column.
3. **Include All Rows from the Right Table**:
   - All rows from the right table will be included in the result, regardless of whether they have matching rows in the left table.
4. **Nullable Columns from the Left Table**:
   - Columns from the left table may contain NULL values for rows that do not have a matching entry in the right table.
5. **Filtering (Optional)**:
   - A WHERE clause can be used to further filter the results but should not unintentionally filter out NULLs from the left table if they are needed.

```
SELECT Employees.emp_id, Employees.emp_name, Departments.dept_name
FROM Employees
RIGHT JOIN Departments
ON Employees.dept_id = Departments.dept_id;
```