Experiment No: 8

Aim: To set up a PostgreSQL database, create relational tables, and perform basic **CRUD** operations (Create, Read, Update, Delete) using SQL queries.

Theory:

Introduction to PostgreSQL:

PostgreSQL is a robust, open-source object-relational database system known for its reliability, feature richness, and high performance. Originally developed at the University of California, Berkeley, it is now maintained by a global community. PostgreSQL supports advanced features like user-defined types, table inheritance, views, foreign keys, transactions, and multi-version concurrency control (MVCC).

Relational Databases:

A relational database organizes information into structured tables, where each table (relation) consists of rows (records) and columns (attributes). Every row in a table represents a unique entry, typically identified by a **primary key**. Tables can relate to one another using **foreign keys**, enabling data normalization and integrity across the database.

SQL - Structured Query Language:

SQL is the standard language for managing and interacting with relational databases. PostgreSQL enhances SQL by adding advanced features and constructs. The major categories of SQL commands include:

- **Data Definition Language (DDL):** Defines and modifies database schema objects using `CREATE`, `ALTER`, and `DROP`.

- **Data Manipulation Language (DML):** Performs data operations using `INSERT`, `UPDATE`, `DELETE`, and `SELECT`.

- **Data Control Language (DCL):** Manages permissions using `GRANT` and `REVOKE`.

- **Transaction Control Language (TCL):** Controls transactions using `BEGIN`, `COMMIT`, and `ROLLBACK`.

CRUD Operations:

CRUD stands for the four basic operations used in database interaction:

- **Create (INSERT):** Adds new records to a table.
  `INSERT INTO students (name, age, email) VALUES ('Manas Mungekar', 22, 'manas@example.com');`
- **Read (SELECT):** Retrieves data from a table.
  `SELECT * FROM students;`
- **Update (UPDATE):** Modifies existing records.
  `UPDATE students SET age = 23 WHERE name = 'Manas Mungekar';`
- **Delete (DELETE):** Removes records from a table.
  `DELETE FROM students WHERE name = 'Manas Mungekar';`

Creating a Table in PostgreSQL:

To perform CRUD operations, you need to first define a table structure using DDL commands.

- `CREATE TABLE students (`
- `  id SERIAL PRIMARY KEY,`
- `  name VARCHAR(100),`
- `  age INT,`
- `  email VARCHAR(100)`
- `);`

- **SERIAL:** Automatically generates a unique ID.

- **VARCHAR:** Used for variable-length character strings.

- **INT:** Stores integer values.

Relationships and Foreign Keys:

In larger applications, multiple tables are used and linked together. For example, a `students` table may be linked with a `courses` table through an `enrollments` table using **foreign keys** to maintain referential integrity.

Normalization:

Normalization is a process used to reduce data redundancy and improve consistency by organizing data into separate, related tables. Common forms include:

- **1NF:** Eliminates repeating groups.
- **2NF:** Removes partial dependencies.
- **3NF:** Eliminates transitive dependencies.

Indexes:

Indexes improve query performance by allowing faster data lookups. However, too many indexes can slow down insert/update operations and increase storage usage.

Constraints in PostgreSQL:

- **NOT NULL:** Prevents NULL values in a column.
- **UNIQUE:** Ensures that all values in a column are distinct.
- **CHECK:** Enforces specific conditions.
- **DEFAULT:** Assigns a default value to a column.

Transactions:

A transaction is a group of SQL operations executed as a single unit. PostgreSQL supports ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure data reliability.

- `BEGIN;`
- `UPDATE students SET age = 24 WHERE name = 'Manas Mungekar';`
- `COMMIT;`

If something goes wrong during the transaction, you can use `ROLLBACK` to undo the changes.

Advantages of PostgreSQL:

- Completely free and open-source.
- ACID-compliant for reliable transaction processing.
- Compatible with major programming languages and platforms.
- Extensible through user-defined types, plugins, and functions.
- Native support for JSON, XML, full-text search, and GIS data (via PostGIS).
- Actively maintained by a global developer community.

Common PostgreSQL Tools:

- **psql:** Command-line tool for PostgreSQL.
- **pgAdmin:** Graphical user interface for managing databases.
- **DBeaver / DataGrip:** Powerful third-party database IDEs.

Security Features:

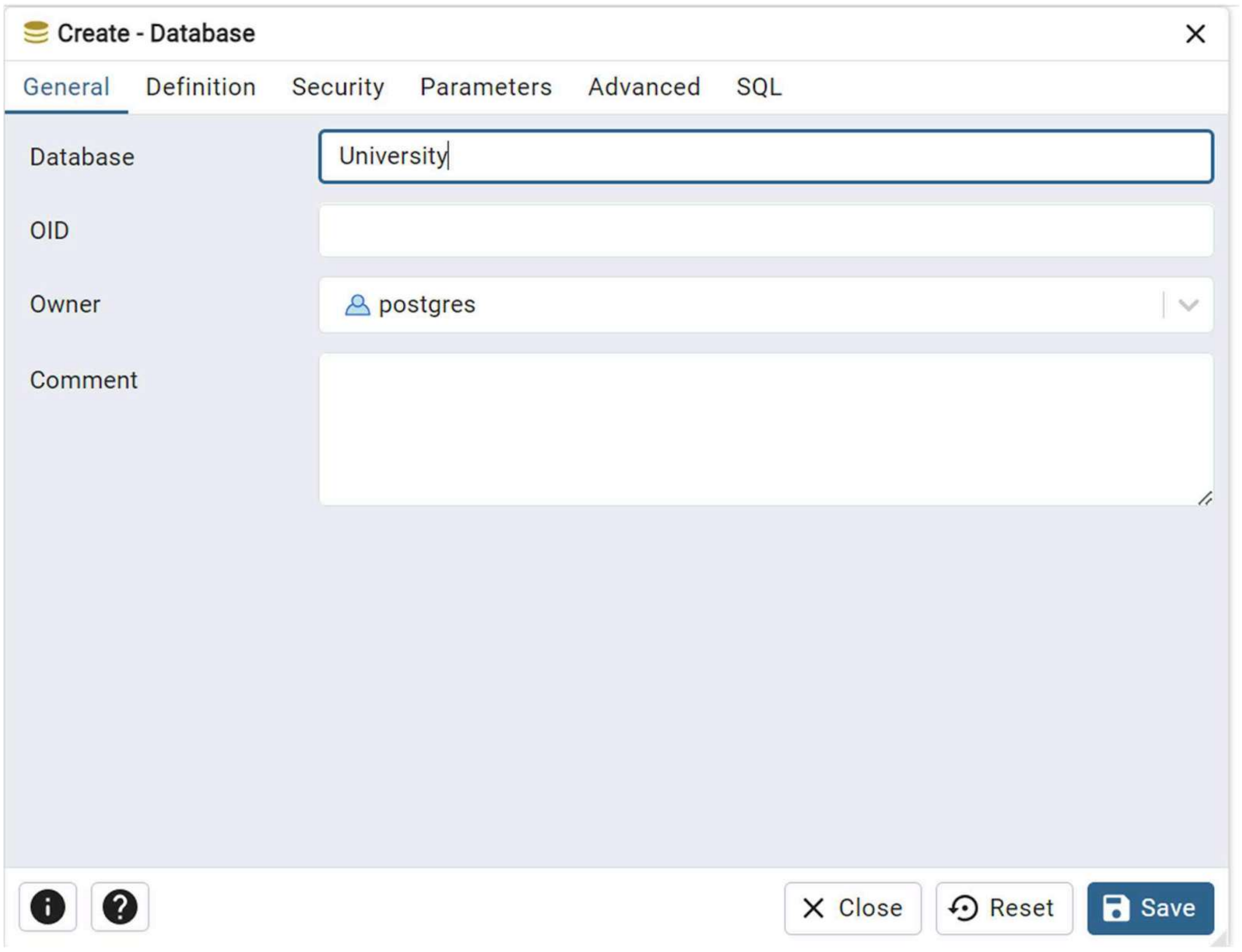
PostgreSQL includes strong access control systems using roles, passwords, and permissions. It also supports SSL for secure connections and row-level security for fine-grained access control.

Real-World Use Cases of PostgreSQL:

- Web application backends (e.g., Django, Express, Rails)
- Data warehousing and analytics
- Financial and banking systems

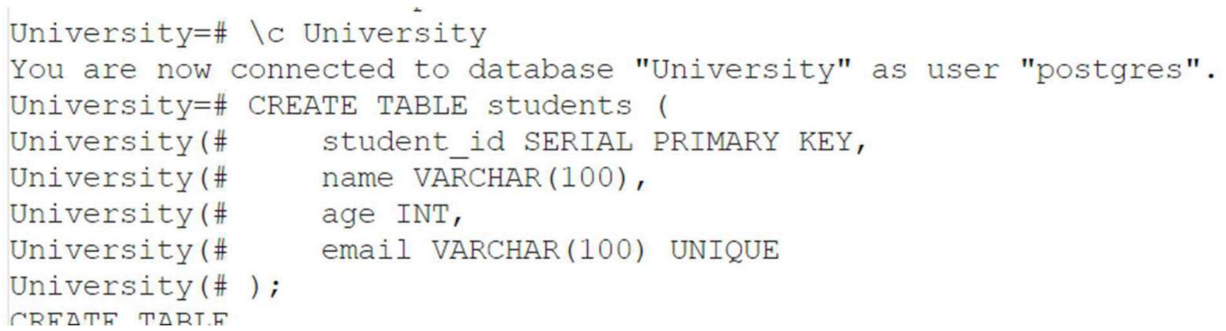
- Geographic Information Systems (PostGIS)
- ERP and CRM systems
- Internet of Things (IoT) platforms

CREATING A DATABASE



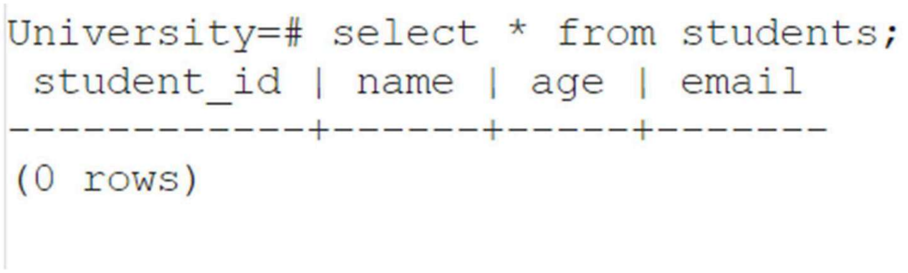
Database	University
OID	
Owner	postgres
Comment	

CREATING TABLE



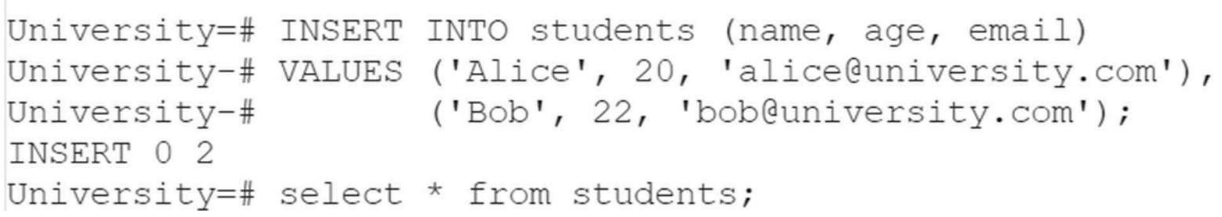
```
University=# \c University
You are now connected to database "University" as user "postgres".
University=# CREATE TABLE students (
University(#     student_id SERIAL PRIMARY KEY,
University(#     name VARCHAR(100),
University(#     age INT,
University(#     email VARCHAR(100) UNIQUE
University(# );
CREATE TABLE
```

Table:



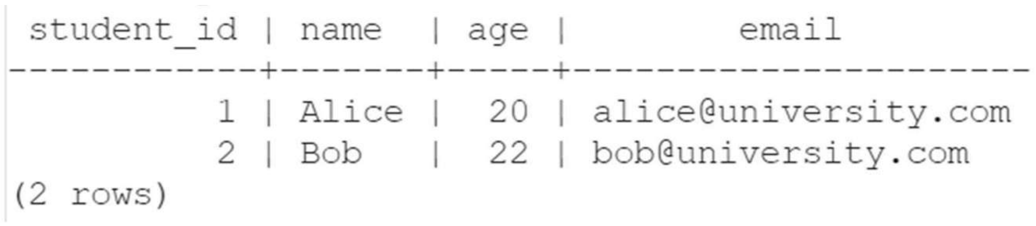
```
University=# select * from students;
 student_id | name | age | email
------------+------+-----+-------
(0 rows)
```

INSERTING VALUES IN TABLE



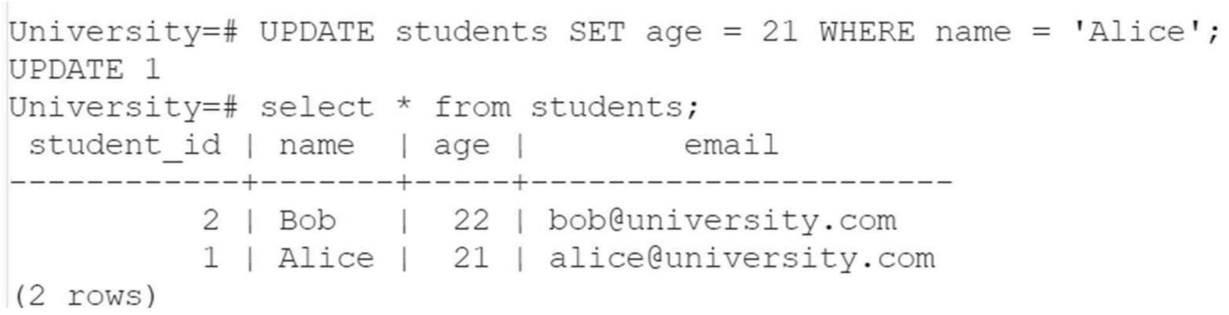
```
University=# INSERT INTO students (name, age, email)
University-# VALUES ('Alice', 20, 'alice@university.com'),
University-#        ('Bob', 22, 'bob@university.com');
INSERT 0 2
University=# select * from students;
```

Tables:



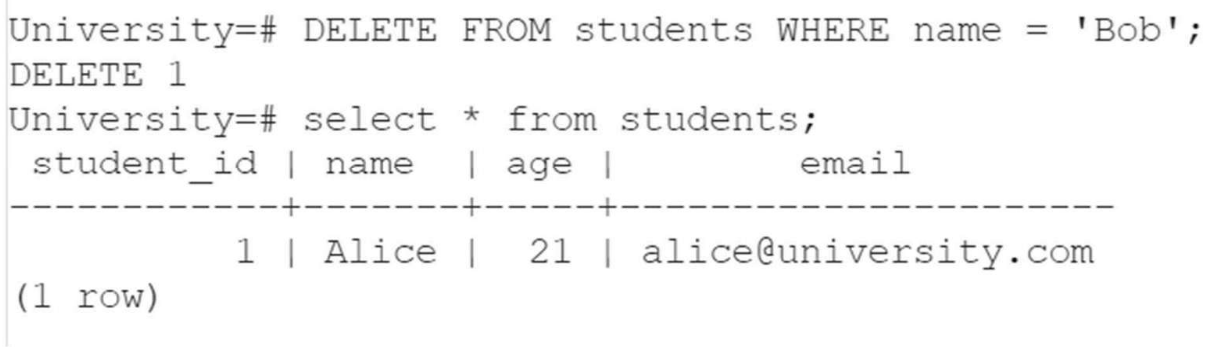
```
 student_id | name  | age |        email
------------+-------+-----+----------------------
          1 | Alice |  20 | alice@university.com
          2 | Bob   |  22 | bob@university.com
(2 rows)
```

UPDATING VALUES IN TABLE



```
University=# UPDATE students SET age = 21 WHERE name = 'Alice';
UPDATE 1
University=# select * from students;
 student_id | name  | age |        email
------------+-------+-----+----------------------
          2 | Bob   |  22 | bob@university.com
          1 | Alice |  21 | alice@university.com
(2 rows)
```

DELETING VALUES IN A TABLE



```
University=# DELETE FROM students WHERE name = 'Bob';
DELETE 1
University=# select * from students;
 student_id | name  | age |        email
------------+-------+-----+---------------------
          1 | Alice |  21 | alice@university.com
(1 row)
```

Conclusion:

In this experiment, we successfully learned the core concepts of setting up and interacting with a PostgreSQL database. We created relational tables to organize structured data efficiently and applied the four primary CRUD operations using SQL queries. These operations form the backbone of any application that requires data storage and management. Understanding CRUD in PostgreSQL lays the foundation for more advanced database concepts such as indexing, joins, transactions, triggers, and stored procedures.

By mastering PostgreSQL and SQL operations, students and developers are equipped with essential tools for building scalable and data-driven applications in real-world environments. This hands-on practice also strengthens the ability to design efficient schemas, enforce data integrity, and perform robust data manipulation tasks in production-level software systems.