Software Dependencies :

Psycopg is the most popular PostgreSQL database adapter for Python programming language to implement backend of any application. Its main features are thread safety, concurrency control and complete implementation of Python DB API 2.0.

It was designed for multithreaded application which perform CRUD operations using a cursor and perform vast number of insertions and updates.

Psycopg2 is both Unicode and Python3 friendly.

Simple user interface integrated with python backend :

We have made an application which performs CRUD operation on the database.

1. Insertion.py

```
import psycopg2

""" insertion operation """
try:
    connection = psycopg2.connect(host = "localhost",database="vehicleshowroom",user =
"postgres",password = "asdf")
    cursor = connection.cursor()
    postgres_insert_query = """ insert into vehicle values('car', 'v106', 'EcoBoost', 'In Stock',
'ta_501', 2084843, 'Mustang', 'Premuim Fastback');"""
    cursor.execute(postgres_insert_query)
    connection.commit()
    print("1 Record inserted successfully")
    cursor.execute("SELECT * from vehicle")
    record = cursor.fetchall()
    print("Result ", record)
    count = cursor.rowcount
    print(count, "Record inserted successfully into vehicle table")

except (Exception, psycopg2.Error) as error:
        print("Failed to insert record into vehicle table", error)

finally:
    if connection:
        cursor.close()
        connection.close()
        print("PostgreSQL connection is closed")
"""----------------------------------------------------------------------------------------"""
```

2. Read.py

```
import psycopg2

""" read operation """
try:
    connection = psycopg2.connect(host = "localhost",database="vehicleshowroom",user =
"postgres",password = "asdf")
    cursor = connection.cursor()
    print("connection succesfully established")



    #one table at a time
    print("ADMINISTRATION TABLE")
    print("")
    cursor.execute("SELECT * from administration")
    record = cursor.fetchall()
    for row in record:
        print(row)
    print("")
    print("")
    print("DEALER TABLE")
    print("")
    cursor.execute("SELECT * from dealer")
    record = cursor.fetchall()
    for row in record:
        print(row)
    print("")
    print("")
    print("VEHICLE TABLE")
    print("")
    cursor.execute("SELECT * from vehicle")
    record = cursor.fetchall()
    for row in record:
        print(row)
    print("")
    print("")
    print("IMG TABLE")
    print("")
    cursor.execute("SELECT * from img")
    record = cursor.fetchall()
    for row in record:
        print(row)
    print("")
    print("")
    print("SHOWROOM TABLE")
    cursor.execute("SELECT * from showroom")
    record = cursor.fetchall()
```

```
for row in record:
        print(row)
    print("")
    print("")
    print("CUSTOMER TABLE")
    cursor.execute("SELECT * from customer")
    record = cursor.fetchall()
    for row in record:
        print(row)
    print("")
    print("")
    print("SALES TABLE")
    cursor.execute("SELECT * from sales")
    record = cursor.fetchall()
    for row in record:
        print(row)

except (Exception, psycopg2.Error) as error:
        print("Failed to insert record into mobile table", error)

finally:
    if connection:
        cursor.close()
        connection.close()
        print("PostgreSQL connection is closed")
```

3. Update.py

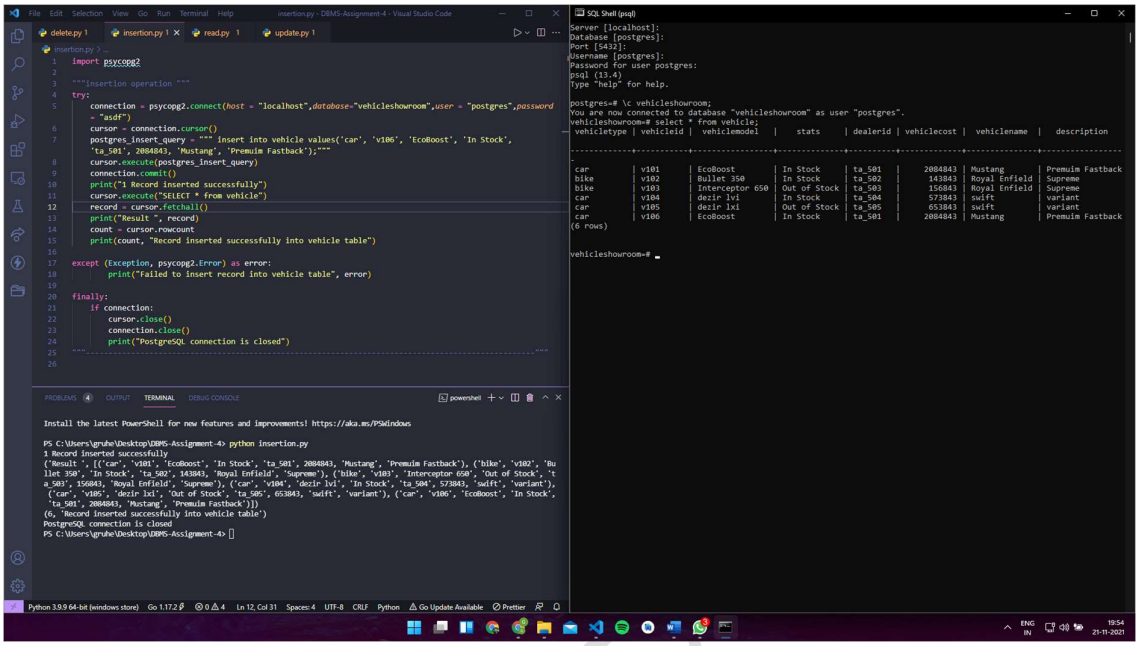
```
import psycopg2

'''update operation'''
def updateTable(vehicleID, vehcost):
    try:
        connection = psycopg2.connect(host = "localhost",database="vehicleshowroom",user =
"postgres",password = "asdf")
        cursor = connection.cursor()
        print("Table Before updating record ")
        sql_select_query = """select * from vehicle where vehicleID = %s"""
        cursor.execute(sql_select_query, (vehicleID,))
        record = cursor.fetchone()
        print(record)

        # Update single record now
        sql_update_query = """Update vehicle set vehiclecost = %s where vehicleID = %s"""
        cursor.execute(sql_update_query, (vehcost, vehicleID))
        connection.commit()
        count = cursor.rowcount
        print(count, "Record Updated successfully ")

        print("Table After updating record")
        sql_select_query = """select * from vehicle where vehicleID = %s"""
        cursor.execute(sql_select_query, (vehicleID,))
        record = cursor.fetchone()
        print(record)

    except (Exception, psycopg2.Error) as error:
        print("Error in update operation", error)

    finally:
        # closing database connection.
        if connection:
            cursor.close()
            connection.close()
            print("PostgreSQL connection is closed")

id = 'v101'
price = 3084843
updateTable(id, price)
```

4. Delete.py

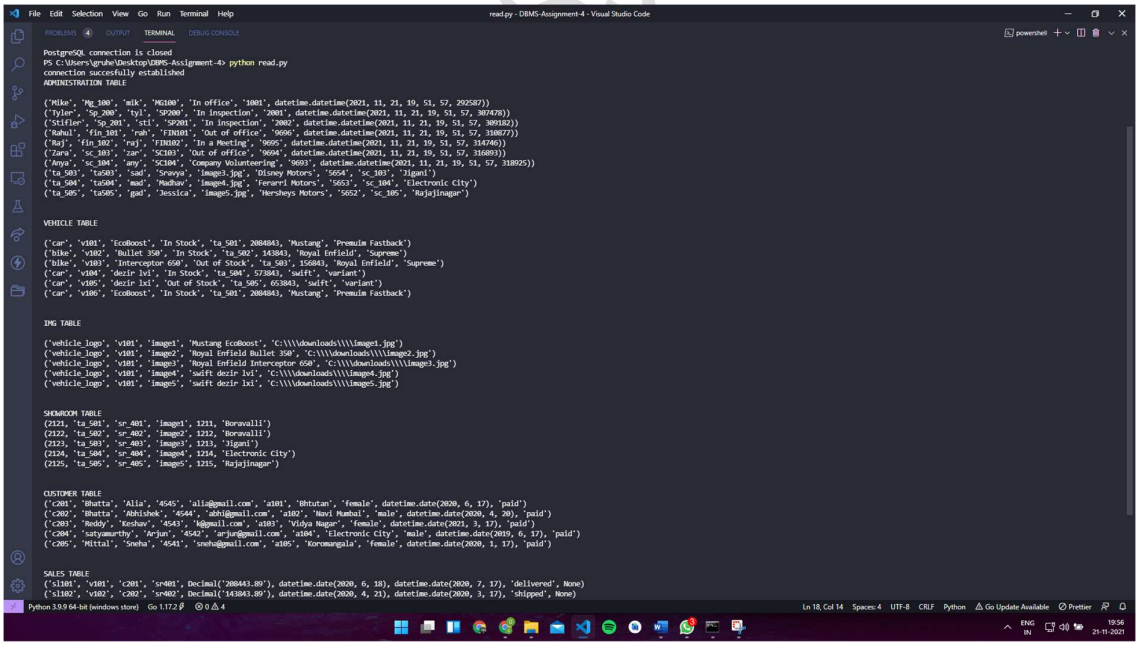
```
import psycopg2


def deleteData(vehicleID):
    try:
        connection = psycopg2.connect(host = "localhost",database="vehicleshowroom",user =
"postgres",password = "asdf")
        cursor = connection.cursor()
        # Update single record now
        print("Table Before updating record ")
        sql_select_query = """select * from vehicle where vehicleID = %s"""
        cursor.execute(sql_select_query, (vehicleID,))
        record = cursor.fetchone()
        print(record)
        sql_delete_query = """Delete from vehicle where vehicleID = %s"""
        cursor.execute(sql_delete_query, (vehicleID,))
        connection.commit()
        print("Table After updating record")
        sql_select_query = """select * from vehicle where vehicleID = %s"""
        cursor.execute(sql_select_query, (vehicleID,))
        record = cursor.fetchone()
        print(record)
        count = cursor.rowcount
        print(count, "Record deleted successfully ")

    except (Exception, psycopg2.Error) as error:
        print("Error in Delete operation", error)

    finally:
        # closing database connection.
        if connection:
            cursor.close()
            connection.close()
            print("PostgreSQL connection is closed")

id4 = 'v102'
id5 = 'v102'
deleteData(id4)
deleteData(id5)
```

Outputs :

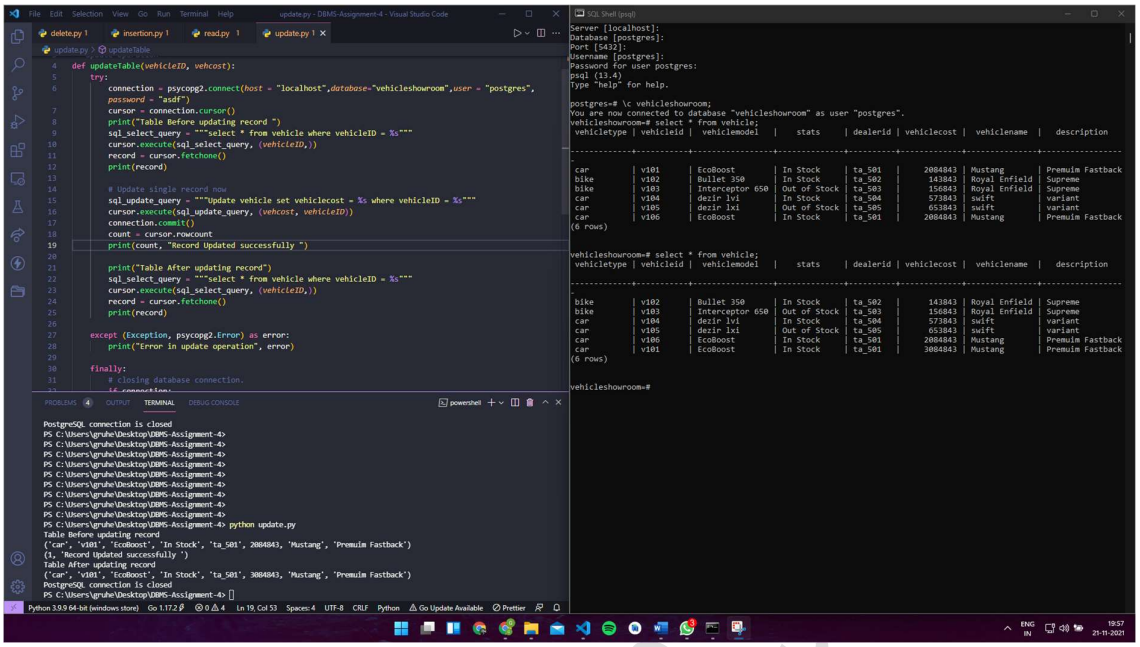
Insertion



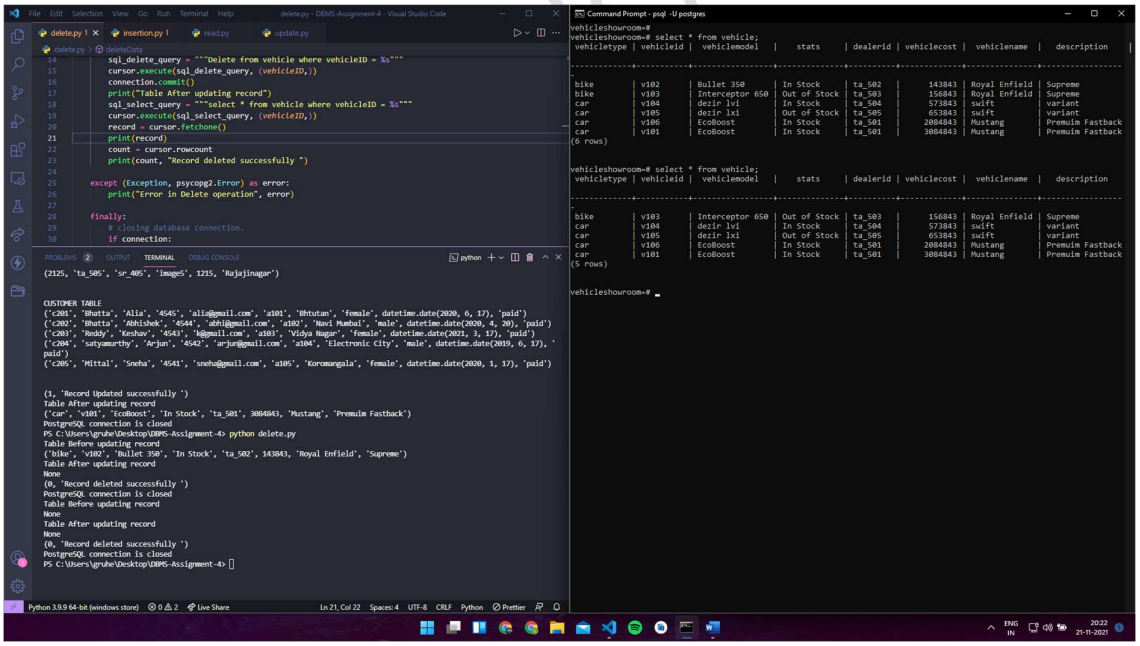
Read



Update



Deletion



Additional queries :

1.

```
create table salesperson(
    salesperson_name varchar(15),
    salesperson_id varchar(10),
    showroomid int,
    passwrd varchar(7),
    username varchar(7),
    status varchar(20),
    contactno varchar(10),
    lastlogin TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    primary key (salesperson_id, showroomid),
    foreign key (showroomid) references showroom(showroomid) on update cascade
);
```

```
vehicleshowroom=# create table salesperson(
vehicleshowroom(# salesperson_name varchar(15),
vehicleshowroom(#     salesperson_id varchar(10),
vehicleshowroom(#     showroomid int,
vehicleshowroom(# passwrd varchar(7),
vehicleshowroom(# username varchar(7),
vehicleshowroom(# status varchar(20),
vehicleshowroom(# contactno varchar(10),
vehicleshowroom(#     lastlogin TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
vehicleshowroom(#     primary key (salesperson_id, showroomid),
vehicleshowroom(#     foreign key (showroomid) references showroom(showroomid) on update cascade
vehicleshowroom(# );
CREATE TABLE
vehicleshowroom=#
```

2.

```
alter table administration
    add constraint contactno_admin check(char_length(contactno)=4);
```

```
vehicleshowroom=# alter table administration
vehicleshowroom-#       add constraint contactno_admin check(char_length(contactno)=4);
ALTER TABLE
vehicleshowroom=#
```

3.

```
alter table dealer
    add constraint contactno_dealer check(char_length(contactno)=4);
```

```
vehicleshowroom=# alter table dealer
vehicleshowroom-#     add constraint contactno_dealer check(char_length(contactno)=4);
ALTER TABLE
vehicleshowroom=#
```

4.

```
alter table salesperson
    add constraint contactno_salesperson check(char_length(contactno)=4);
```

```
vehicleshowroom=# alter table salesperson
vehicleshowroom-#     add constraint contactno_salesperson check(char_length(contactno)=4);
ALTER TABLE
vehicleshowroom=#
```

5.

```
alter table sales
    add column soldby varchar(10);
```

```
vehicleshowroom=# alter table sales
vehicleshowroom-#     add column soldby varchar(10);
ALTER TABLE
vehicleshowroom=#
```

6.

```
update sales set soldby='sp001' where salesid='sl101';
update sales set soldby='sp003' where salesid='sl102';
update sales set soldby='sp002' where salesid='sl103';
update sales set soldby='sp004' where salesid='sl104';
update sales set soldby='sp005' where salesid='sl105';
```

```
vehicleshowroom=# update sales set soldby='sp001' where salesid='sl101';
UPDATE 1
vehicleshowroom=# update sales set soldby='sp003' where salesid='sl102';
UPDATE 1
vehicleshowroom=# update sales set soldby='sp002' where salesid='sl103';
UPDATE 1
vehicleshowroom=# update sales set soldby='sp004' where salesid='sl104';
UPDATE 1
vehicleshowroom=# update sales set soldby='sp005' where salesid='sl105';
UPDATE 1
vehicleshowroom=#
```

Database Migration :

PostGRES is excellent for data persistence and monolithic backend but with the rising use of microservices and data structure but with the advent of rise of big data with a lot of schema less data from multiple sources as the database grows in size and complexity and in case we start to collect data from the users which is very widely different from the current data we register into the schema based database

the issues in Postgres that compel us to move away to new database technologies to adapt to growing needs

- Inefficient architecture for writes
- Inefficient data replication (for resilient storage)
- Issues with table corruption
- Poor replica MVCC support(Multi version concurrency control)
- Difficulty upgrading to newer releases

In addition to explaining some of Postgres's limitations, we also explain why MySQL is an important tool for newer Uber Engineering storage projects, such as Schema less. In many cases, we found MySQL more favourable for our uses. To understand the differences, we examine MySQL's architecture and how it contrasts with that of Postgres. We specifically analyse how MySQL works with the InnoDB storage engine. Not only do we use InnoDB, it's perhaps is the most popular MySQL storage engine.

InnoDB supports advanced features like MVCC and mutable data.

Like other open-source databases, PostgreSQL is easy to run in both containers and virtual machines and is highly portable. Nowadays, many companies have support for PostgreSQL in cloud hosting environments, including all the major cloud providers.

Google Cloud offers integration of PostgreSQL on its cloud servers. It uses PgBouncer as a connection pooler to minimize application downtime and helps set up tools for monitoring the results. This article is designed for PostgreSQL administrators and sysadmins working in a Linux environment.

To perform the migration, you shut down the current master and then promote the subordinate Google Cloud replica to master. PgBouncer reroutes traffic to the new master node on Google Cloud.

To perform the migration, you shut down the current master and then promote the subordinate Google Cloud replica to master. PgBouncer reroutes traffic to the new master node on Google Cloud

Migrating to cloud offers the following benefits:

1. high availability
2. replication of databases
3. hot standby mode

Migrating to cloud gets rid of handling hardware why can get risky of not managed properly. Also, there is tons of storage available, where you can create replicas and have one in hot standby mode in case of any kind of failure.

A load balancer can be used to reduce load on any one server and the whole system can be decentralised.