

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 ")

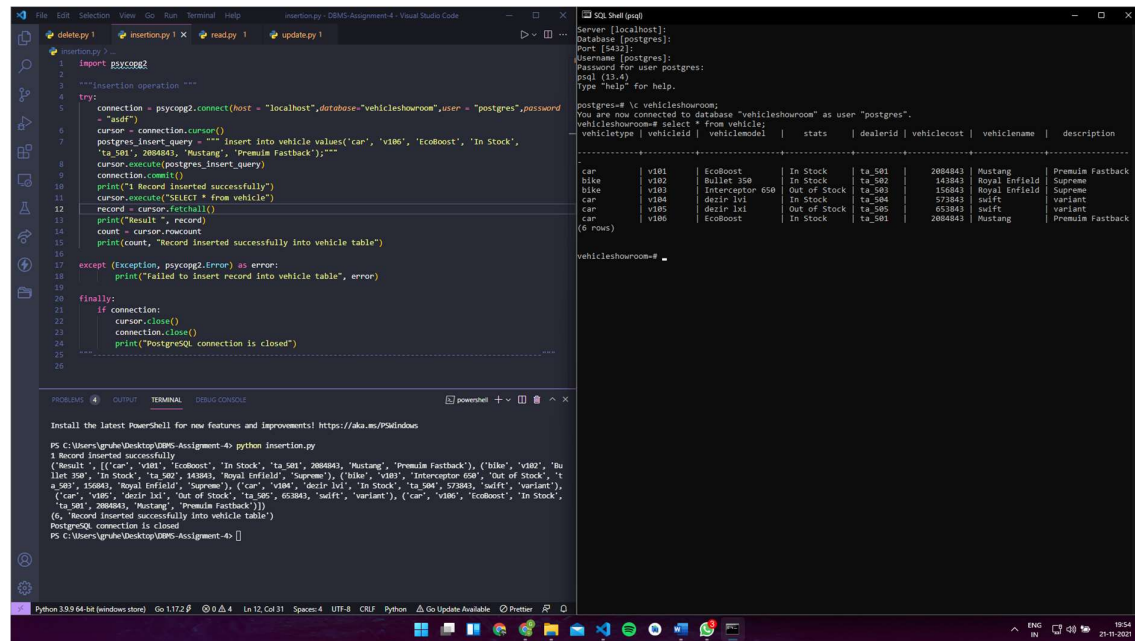
    except (Exception, psycopg2.Error) as error:
        print("Error in Delete operation", error)

    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



```
1 import psycopg2
2
3 """insertion operation"""
4
5 try:
6     connection = psycopg2.connect(host = "localhost", database = "vehicleshourroom", user = "postgres", password
7     cursor = connection.cursor()
8     postgres_insert_query = """ insert into vehicle values('car', 'v100', 'EcoBoost', 'In Stock',
9     ta_501, 2004843, 'Mustang', 'Premium Fastback');"""
10    cursor.execute(postgres_insert_query)
11    connection.commit()
12    print("1 Record inserted successfully")
13    cursor.execute("SELECT * from vehicle")
14    record = cursor.fetchall()
15    print("Result ", record)
16    count = cursor.rowcount
17    print(count, "Record inserted successfully into vehicle table")
18 except (Exception, psycopg2.Error) as error:
19     print("Failed to insert record into vehicle table", error)
20
21 finally:
22     if connection:
23         cursor.close()
24         connection.close()
25     print("PostgreSQL connection is closed")
```

Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (13.4)
Type "help" for help.

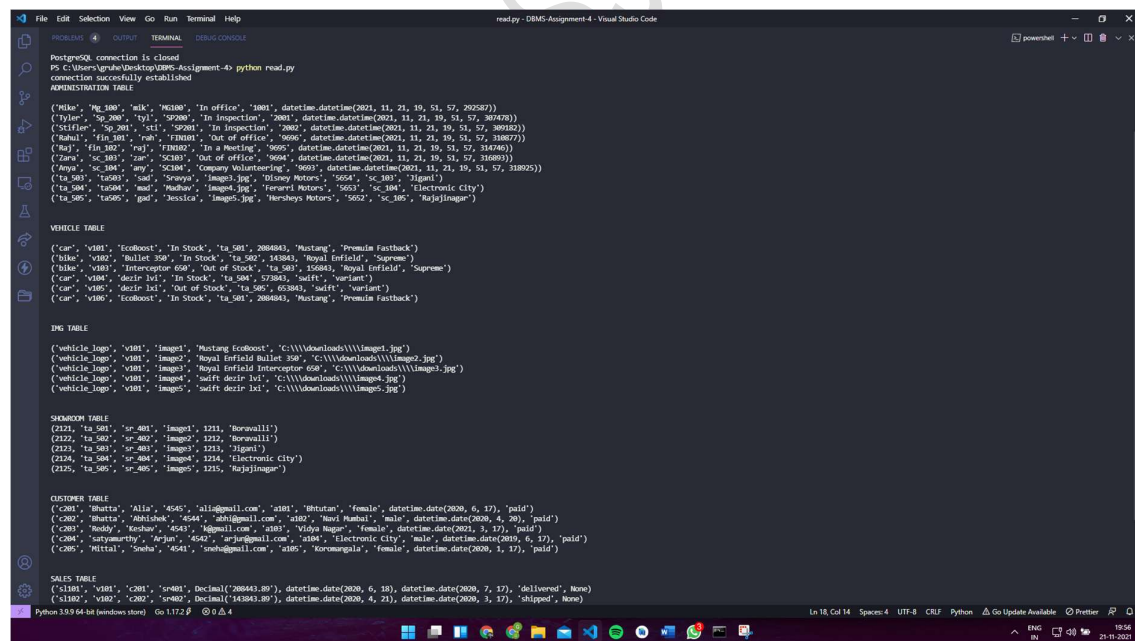
postgres=# \c vehicleshourroom;
You are now connected to database "vehicleshourroom" as user "postgres".
vehicleshourroom=# select * from vehicle;

vehicletype	vehicleid	vehiclename	stats	dealerid	vehiclecost	vehiclename	description
car	v101	EcoBoost	In Stock	ta_501	2004843	Mustang	Premium Fastback
bike	v102	Bullet 350	In Stock	ta_502	143843	Royal Enfield	Supreme
bike	v103	Interceptor 650	Out of Stock	ta_503	156843	Royal Enfield	Supreme
car	v104	dezir lxi	In Stock	ta_504	573843	swift	variant
car	v105	dezir lxi	Out of Stock	ta_505	653843	swift	variant
car	v106	EcoBoost	In Stock	ta_501	2004843	Mustang	Premium Fastback

vehicleshourroom=#

PS C:\Users\grube\Desktop\DBMS-Assignment-4> python Insertion.py
1 Record inserted successfully
(Result, (('car', 'v101', 'EcoBoost', 'In Stock', 'ta_501', 2004843, 'Mustang', 'Premium Fastback'), ('bike', 'v102', 'Bu
let 350', 'In Stock', 'ta_502', 143843, 'Royal Enfield', 'Supreme'), ('bike', 'v103', 'Interceptor 650', 'Out of Stock', 't
a_503', 156843, 'Royal Enfield', 'Supreme'), ('car', 'v104', 'dezir lxi', 'In Stock', 'ta_504', 573843, 'swift', 'variant'),
(('car', 'v105', 'dezir lxi', 'Out of Stock', 'ta_505', 653843, 'swift', 'variant'), ('car', 'v106', 'EcoBoost', 'In Stock',
'ta_501', 2004843, 'Mustang', 'Premium Fastback'))
(6, 'Record inserted successfully into vehicle table')
PostgreSQL connection is closed
PS C:\Users\grube\Desktop\DBMS-Assignment-4>

Read



```
1 """Read operation"""
2
3 try:
4     connection = psycopg2.connect(host = "localhost", database = "vehicleshourroom", user = "postgres", password
5     cursor = connection.cursor()
6     postgres_read_query = """ select * from vehicle """
7     cursor.execute(postgres_read_query)
8     records = cursor.fetchall()
9     print("Result ", records)
10 except (Exception, psycopg2.Error) as error:
11     print("Failed to read record from vehicle table", error)
12
13 finally:
14     if connection:
15         cursor.close()
16         connection.close()
17     print("PostgreSQL connection is closed")
```

PostgreSQL connection is closed
PS C:\Users\grube\Desktop\DBMS-Assignment-4> python Read.py
connection successfully established
ADMINISTRATION TABLE

```
('Mike', 'Mg_100', 'mike', 'M100', 'In office', '1001', datetime.datetime(2021, 11, 21, 19, 51, 57, 262547))
('Tyler', 'Sp_200', 'tyl', 'SP200', 'In inspection', '2001', datetime.datetime(2021, 11, 21, 19, 51, 57, 307478))
('Siddhar', 'Sp_201', 'scl', 'SP201', 'In inspection', '2002', datetime.datetime(2021, 11, 21, 19, 51, 57, 307892))
('Maha', 'fin_101', 'mah', 'FIN101', 'Out of office', '9999', datetime.datetime(2021, 11, 21, 19, 51, 57, 310777))
('Raj', 'fin_102', 'raj', 'FIN102', 'In a meeting', '9995', datetime.datetime(2021, 11, 21, 19, 51, 57, 314760))
('Zoya', 'sc_101', 'zay', 'SC101', 'Out of office', '9994', datetime.datetime(2021, 11, 21, 19, 51, 57, 316831))
('Anya', 'sc_104', 'any', 'SC104', 'Company Volunteering', '9993', datetime.datetime(2021, 11, 21, 19, 51, 57, 318025))
('ta_501', 'ta501', 'sad', 'Sroya', 'Images.jpg', 'Disney Motors', '5654', 'sc_101', 'ligant')
('ta_504', 'ta504', 'mad', 'Mashu', 'Images.jpg', 'Fernand Motors', '5652', 'sc_104', 'Electronic City')
('ta_505', 'ta505', 'gad', 'Jessica', 'Images.jpg', 'Mushneys Motors', '5652', 'sc_105', 'Rajajinagar')
```

VEHICLE TABLE

```
('car', 'v101', 'EcoBoost', 'In Stock', 'ta_501', 2004843, 'Mustang', 'Premium Fastback')
('bike', 'v102', 'Bullet 350', 'In Stock', 'ta_502', 143843, 'Royal Enfield', 'Supreme')
('bike', 'v103', 'Interceptor 650', 'Out of Stock', 'ta_503', 156843, 'Royal Enfield', 'Supreme')
('car', 'v104', 'dezir lxi', 'In Stock', 'ta_504', 573843, 'swift', 'variant')
('car', 'v105', 'dezir lxi', 'Out of Stock', 'ta_505', 653843, 'swift', 'variant')
('car', 'v106', 'EcoBoost', 'In Stock', 'ta_501', 2004843, 'Mustang', 'Premium Fastback')
```

IMG TABLE

```
('vehicle_img', 'v101', 'image1', 'Mustang EcoBoost', 'C:\\\\downloads\\\\image2.jpg')
('vehicle_img', 'v101', 'image2', 'Royal Enfield Bullet 350', 'C:\\\\downloads\\\\image2.jpg')
('vehicle_img', 'v101', 'image3', 'Royal Enfield Interceptor 650', 'C:\\\\downloads\\\\image3.jpg')
('vehicle_img', 'v103', 'image4', 'swift dezir lxi', 'C:\\\\downloads\\\\image4.jpg')
('vehicle_img', 'v103', 'image5', 'swift dezir lxi', 'C:\\\\downloads\\\\image5.jpg')
```

SHOWROOM TABLE

```
(2121, 'ta_501', 'sr_401', 'image1', 1211, 'Boravalli')
(2122, 'ta_502', 'sr_402', 'image2', 1212, 'Boravalli')
(2123, 'ta_503', 'sr_403', 'image3', 1213, 'Jigaji')
(2124, 'ta_504', 'sr_404', 'image4', 1214, 'Electronic City')
(2125, 'ta_505', 'sr_405', 'image5', 1215, 'Rajajinagar')
```

CUSTOMER TABLE

```
('C201', 'Shruti', 'Alia', '4545', 'alia@gmail.com', 'a101', 'Shruti', 'female', datetime.date(2020, 6, 17), 'paid')
('C202', 'Shruti', 'Abhishek', '4544', 'abhi@gmail.com', 'a102', 'Navi Mumbai', 'male', datetime.date(2020, 4, 20), 'paid')
('C203', 'Raddy', 'Keshav', '4543', 'kesh@gmail.com', 'a103', 'Vidya Nagar', 'female', datetime.date(2021, 3, 17), 'paid')
('C204', 'Sanyamrthy', 'Arjun', '4542', 'arjun@gmail.com', 'a104', 'Electronic City', 'male', datetime.date(2019, 6, 17), 'paid')
('C205', 'Mittal', 'Sneha', '4541', 'sneh@gmail.com', 'a105', 'Koramangala', 'female', datetime.date(2020, 1, 17), 'paid')
```

SALES TABLE

```
('s1101', 'v101', 'C201', 'sr401', Decimal('2004843.00'), datetime.date(2020, 6, 10), datetime.date(2020, 7, 17), 'delivered', None)
('s1102', 'v102', 'C202', 'sr402', Decimal('143843.00'), datetime.date(2020, 4, 21), datetime.date(2020, 3, 17), 'shipped', None)
```

Update

```
def updateTable(vehicleID, vehcost):
    try:
        connection = psycopg2.connect(host="localhost", database="vehicleshourroom", 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 vehcost = %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:
```

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (13.4)
Type "help" for help.

postgres=# \c vehicleshourroom;
You are now connected to database "vehicleshourroom" as user "postgres".
vehicleshourroom=# select * from vehicle;
 vehicletype | vehicleid | vehiclemodel | stats | dealerid | vehiclecost | vehiclename | description
-----
car          | v101      | EcoBoost      | In Stock | ta_501   | 2084843    | Mustang     | Premium Fastback
bike         | v102      | Bullet 350    | In Stock | ta_502   | 143843    | Royal Enfield | Supreme
bike         | v103      | Interceptor 650 | Out of Stock | ta_503   | 156843    | Royal Enfield | Supreme
car          | v104      | dezir lvi     | In Stock | ta_504   | 573843    | swift         | variant
car          | v105      | dezir lxi     | Out of Stock | ta_505   | 653843    | swift         | variant
car          | v106      | EcoBoost      | In Stock | ta_501   | 2084843    | Mustang     | Premium Fastback
(6 rows)

vehicleshourroom=# select * from vehicle;
 vehicletype | vehicleid | vehiclemodel | stats | dealerid | vehiclecost | vehiclename | description
-----
bike         | v102      | Bullet 350    | In Stock | ta_502   | 143843    | Royal Enfield | Supreme
bike         | v103      | Interceptor 650 | Out of Stock | ta_503   | 156843    | Royal Enfield | Supreme
car          | v104      | dezir lvi     | In Stock | ta_504   | 573843    | swift         | variant
car          | v105      | dezir lxi     | Out of Stock | ta_505   | 653843    | swift         | variant
car          | v106      | EcoBoost      | In Stock | ta_501   | 2084843    | Mustang     | Premium Fastback
(6 rows)

vehicleshourroom=#
```

Deletion

```
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:
```

```
vehicleshourroom=# select * from vehicle;
 vehicletype | vehicleid | vehiclemodel | stats | dealerid | vehiclecost | vehiclename | description
-----
bike         | v102      | Bullet 350    | In Stock | ta_502   | 143843    | Royal Enfield | Supreme
bike         | v103      | Interceptor 650 | Out of Stock | ta_503   | 156843    | Royal Enfield | Supreme
car          | v104      | dezir lvi     | In Stock | ta_504   | 573843    | swift         | variant
car          | v105      | dezir lxi     | Out of Stock | ta_505   | 653843    | swift         | variant
car          | v106      | EcoBoost      | In Stock | ta_501   | 2084843    | Mustang     | Premium Fastback
(6 rows)

vehicleshourroom=# select * from vehicle;
 vehicletype | vehicleid | vehiclemodel | stats | dealerid | vehiclecost | vehiclename | description
-----
bike         | v103      | Interceptor 650 | Out of Stock | ta_503   | 156843    | Royal Enfield | Supreme
car          | v104      | dezir lvi     | In Stock | ta_504   | 573843    | swift         | variant
car          | v105      | dezir lxi     | Out of Stock | ta_505   | 653843    | swift         | variant
car          | v106      | EcoBoost      | In Stock | ta_501   | 2084843    | Mustang     | Premium Fastback
(5 rows)

vehicleshourroom=#
```

```
CREATE TABLE
("C001", "Bhutta", "A11a", "4545", "ali@gmail.com", "a101", "Bhutan", "female", datetime.date(2020, 6, 17), "paid")
("C002", "Bhutta", "A11b", "4544", "ali@gmail.com", "a102", "Naxi Mumbai", "male", datetime.date(2020, 4, 20), "paid")
("C003", "Rohdy", "Arjun", "4543", "arjun@gmail.com", "a103", "Vidya Nagar", "female", datetime.date(2021, 3, 17), "paid")
("C004", "satyamurthy", "Arjun", "4542", "arjun@gmail.com", "a104", "Electronic City", "male", datetime.date(2019, 6, 17), "paid")
("C005", "Mittal", "Sneha", "4541", "sneha@gmail.com", "a105", "Koramangala", "female", datetime.date(2020, 1, 17), "paid")

(1, "Record updated successfully")
Table After updating record
('car', 'v101', 'EcoBoost', 'In Stock', 'ta_501', 2084843, 'Mustang', 'Premium Fastback')
PostgreSQL connection is closed
Table Before updating record
('bike', 'v102', 'Bullet 350', 'In Stock', 'ta_502', 143843, 'Royal Enfield', 'Supreme')
Table After updating record
None
(0, "Record deleted successfully")
PostgreSQL connection is closed
Table Before updating record
None
Table After updating record
None
(0, "Record deleted successfully")
PostgreSQL connection is closed
```

Additional queries :

1.

```
create table salesperson(  
    salesperson_name varchar(15),  
    salesperson_id varchar(10),  
    showroomid int,  
    passwd 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=# passwd 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);
```

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

5.

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

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

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';
```

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

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 which can get risky if 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.