SQUADSTACK

Problem Statement Analysis :-

The problem states that a Parking ticketing system should be implemented.

The system can allow ‘n’ cars at a time.

There are ‘n’ slots. Slot number starts from 1 to ‘n’.



When a car enters,

we note down :-   
 -> driver age

-> Vehicle number

We assign a slot (if available)

-> to car which is closest to the entry

When car leaves,  
 we get command “leave <x> “

-> if car exists in that slot then we mark it as vacant

To be able to give results for :-

-> given age, find vehicle reg nos. of cars parked (currently)

-> slot no of a car, given reg. no

-> given age, find slot nos. of cars parked (currently)

Edge cases :-

1. Cars > n come, then deny parking

“Parking full”

1. Given age, no cars then -> null (if asked for slots or vehicle no)
2. Vacate a slot but that is not occupied, just print ->

“Slot <x> vacated”

1. Extra space between elements in command

ASSUMPTIONS :-

1. Commands have no spell error, if they have spell error, then I print “Incorrect command and read the next command”
2. Car reg. no format is fixed
3. Only 1 create\_parking\_lot command at the start
4. Cars have distinct registration no
5. If commands have case different, then I ignore case
6. File provided is of format .txt and readable
7. Output displayed in terminal
8. N < 1000 cars
9. When we get vehicle nos. of all cars whose drivers are of Age = ‘x’, we return a Comma separated list of vehicle nos.
10. History is not saved
11. Considering the user as a good user and not a hacker who tries to inject sql. But will add some basic protection against SQL Injection attack.

TECH STACK :-

Python, SQLAlchemy (because FAQ specified that we can use DB Connector libraries), Postgresql

Reason for using this tech stack -

**SQLAlchemy** provides ORM with bare python (no frameworks which was specified in the question). ORM is powerful any day compared to using bare connecting library and running SQL queries. Could obviously use simple database connectors, but it will risk SQL Injection attacks. SQLAlchemy is a library that facilitates the communication between Python programs and databases. So, this must satisfy the condition specified in the FAQ.

**Postgresql** is one of the best relational databases widely trusted and used in the market. It is scalable and versatile.

**Python** because I’m comfortable with the language.

STEPS TO RUN :-

1. Install Postgresql server and create database of name - parking. (can have oyur custom name)

Download it from <https://www.postgresql.org/download/>

To install on Mac,

<https://www.enterprisedb.com/postgresql-tutorial-resources-training?cid=438>

sudo -u postgres psql

​

# Create database

create database parking;

# Create database user

create user myuser with encrypted password 'supersecurepassword123#';

# set permissions

grant all privileges on database parking to myuser;

1. Install Pgadmin4

This is a database visualization tool, where we can also execute queries.

<https://www.pgadmin.org/download/>

Must be straightforward installation

MYSQL :-

/usr/local/mysql/bin/mysql -u root -p

mysql> CREATE DATABASE parking;

Query OK, 1 row affected (0.00 sec)

mysql> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| parking |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.00 sec)

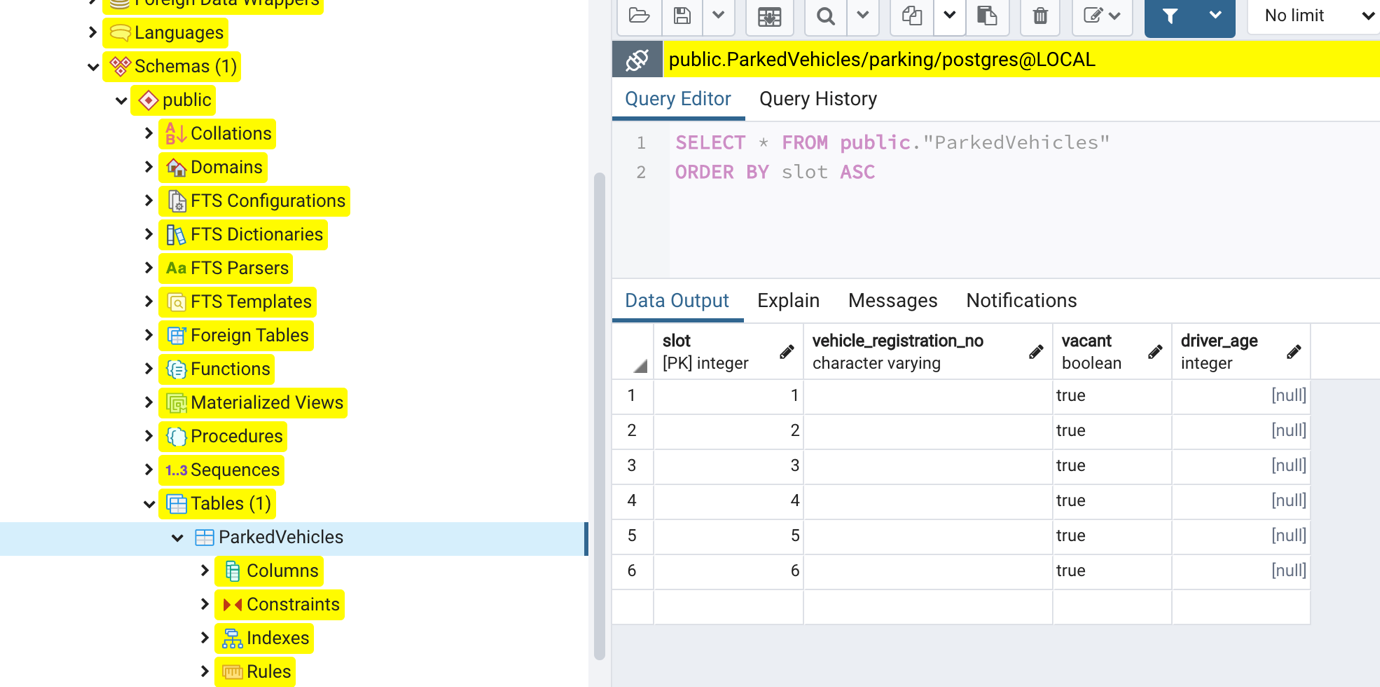
mysql> CREATE USER 'rego' IDENTIFIED BY 'zappy123\_';

Query OK, 0 rows affected (0.02 sec)

mysql> GRANT ALL PRIVILEGES ON \* . \* TO 'rego';

Query OK, 0 rows affected (0.00 sec)

Initially Database :- once it’s created without running any commands :-



Output :-

Graphical user interface, text

Description automatically generated

Database output after running the input.txt :-Graphical user interface, application

Description automatically generated