

PostgreSQL 9.5 queries and PostgreSQL shell

please write the postgresql 9.5 query with example and query on postgresql shell?

Sure! Here are some examples of common PostgreSQL 9.5 queries along with explanations and how you would run them on the PostgreSQL shell.

1. Connecting to the PostgreSQL Shell

First, you need to connect to your PostgreSQL database using the **psql** command-line tool:

```
psql -h hostname -U username -d database_name
```

Replace **hostname** with your database host, **username** with your PostgreSQL username, and **database_name** with the name of the database you want to connect to. After running this command, you will be prompted to enter your password.

2. Creating a Table

Here's an example of creating a table called **employees**:

```
sql
```

Copy code

```
CREATE TABLE employees ( id SERIAL PRIMARY KEY, first_name VARCHAR(50), last_name VARCHAR(50), email VARCHAR(100), hire_date DATE );
```

To execute this on the PostgreSQL shell, simply paste the query and press Enter.

3. Inserting Data

Now, let's insert some data into the **employees** table:

```
sql
```

Copy code

```
INSERT INTO employees (first_name, last_name, email, hire_date) VALUES ('John', 'Doe', 'john.doe@example.com', '2023-01-15'), ('Jane', 'Smith', 'jane.smith@example.com', '2023-03-22');
```

4. Querying Data

To retrieve data from the **employees** table, you can use a SELECT statement:

sql

Copy code

```
SELECT * FROM employees;
```

This will display all rows in the **employees** table.

5. Updating Data

If you want to update a specific row, for example, changing John Doe's email:

sql

Copy code

```
UPDATE employees SET email = 'john.newemail@example.com' WHERE  
first_name = 'John' AND last_name = 'Doe';
```

6. Deleting Data

To delete a row from the **employees** table, for instance, removing Jane Smith:

sql

Copy code

```
DELETE FROM employees WHERE first_name = 'Jane' AND last_name = 'Smith';
```

7. Adding a Column

If you need to add a new column to the **employees** table:

sql

Copy code

```
ALTER TABLE employees ADD COLUMN department VARCHAR(50);
```

8. Dropping a Table

To drop the **employees** table:

sql

Copy code

```
DROP TABLE employees;
```

Example Session

Here is an example session combining all these commands:

sh

Copy code

```
psql -h localhost -U myuser -d mydb
```

sql

Copy code

```
-- Create the table CREATE TABLE employees ( id SERIAL PRIMARY KEY,  
first_name VARCHAR(50), last_name VARCHAR(50), email VARCHAR(100),  
hire_date DATE ); -- Insert data INSERT INTO employees (first_name,  
last_name, email, hire_date) VALUES ('John', 'Doe', 'john.doe@example.com',  
'2023-01-15'), ('Jane', 'Smith', 'jane.smith@example.com', '2023-03-22'); --  
Query data SELECT * FROM employees; -- Update data UPDATE employees SET  
email = 'john.newemail@example.com' WHERE first_name = 'John' AND  
last_name = 'Doe'; -- Delete data DELETE FROM employees WHERE first_name  
= 'Jane' AND last_name = 'Smith'; -- Add a column ALTER TABLE employees  
ADD COLUMN department VARCHAR(50); -- Drop the table DROP TABLE  
employees;
```

Running these queries in sequence will demonstrate the creation, manipulation, and deletion of data in a PostgreSQL database.

please write the postgresql 9.5 query maximum 100 query with example and query on postgresql shell ?

ChatGPT

Sure! Here are 100 example queries for PostgreSQL 9.5, ranging from basic operations to more advanced queries. These queries cover creating tables, inserting data, querying data, updating data, deleting data, altering tables, using functions, and more.

Connecting to the PostgreSQL Shell

sh

Copy code

```
psql -h localhost -U username -d database_name
```

1. Create a Database

sql

Copy code

```
CREATE DATABASE company;
```

2. Connect to the Database

sh

Copy code

```
\c company
```

3. Create a Table

sql

Copy code

```
CREATE TABLE employees ( id SERIAL PRIMARY KEY, first_name VARCHAR(50),  
last_name VARCHAR(50), email VARCHAR(100), hire_date DATE );
```

4. Insert Data

sql

Copy code

```
INSERT INTO employees (first_name, last_name, email, hire_date) VALUES  
( 'John', 'Doe', 'john.doe@example.com', '2023-01-15'), ('Jane', 'Smith',  
'jane.smith@example.com', '2023-03-22');
```

5. Select All Data

sql

Copy code

```
SELECT * FROM employees;
```

6. Select Specific Columns

sql

Copy code

```
SELECT first_name, last_name FROM employees;
```

7. Select with WHERE Clause

sql

Copy code

```
SELECT * FROM employees WHERE last_name = 'Doe';
```

8. Update Data

sql

Copy code

```
UPDATE employees SET email = 'john.newemail@example.com' WHERE  
first_name = 'John' AND last_name = 'Doe';
```

9. Delete Data

sql

Copy code

```
DELETE FROM employees WHERE first_name = 'Jane' AND last_name = 'Smith';
```

10. Add a Column

sql

Copy code

```
ALTER TABLE employees ADD COLUMN department VARCHAR(50);
```

11. Drop a Column

sql

Copy code

```
ALTER TABLE employees DROP COLUMN department;
```

12. Rename a Column

sql

Copy code

```
ALTER TABLE employees RENAME COLUMN email TO contact_email;
```

13. Rename a Table

sql

Copy code

```
ALTER TABLE employees RENAME TO staff;
```

14. Create an Index

sql

Copy code

```
CREATE INDEX idx_last_name ON employees(last_name);
```

15. Drop an Index

sql

Copy code

```
DROP INDEX idx_last_name;
```

16. Count Rows

sql

Copy code

```
SELECT COUNT(*) FROM employees;
```

17. Average Function

sql

Copy code

```
SELECT AVG(salary) FROM employees;
```

18. Sum Function

sql

Copy code

```
SELECT SUM(salary) FROM employees;
```

19. Max Function

sql

Copy code

```
SELECT MAX(salary) FROM employees;
```

20. Min Function

sql

Copy code

```
SELECT MIN(salary) FROM employees;
```

21. Group By Clause

sql

Copy code

```
SELECT department, COUNT(*) FROM employees GROUP BY department;
```

22. Having Clause

sql

Copy code

```
SELECT department, COUNT(*) FROM employees GROUP BY department  
HAVING COUNT(*) > 5;
```

23. Order By Clause

sql

Copy code

```
SELECT * FROM employees ORDER BY last_name;
```

24. Limit Clause

sql

Copy code

```
SELECT * FROM employees LIMIT 5;
```

25. Offset Clause

sql

Copy code

```
SELECT * FROM employees LIMIT 5 OFFSET 10;
```

26. Create a View

sql

Copy code

```
CREATE VIEW v_employees AS SELECT first_name, last_name, email FROM employees;
```

27. Select from a View

sql

Copy code

```
SELECT * FROM v_employees;
```

28. Drop a View

sql

Copy code

```
DROP VIEW v_employees;
```

29. Create a Sequence

sql

Copy code

```
CREATE SEQUENCE emp_seq START 1;
```

30. Use Sequence in Insert

sql

Copy code

```
INSERT INTO employees (id, first_name, last_name, email, hire_date) VALUES (nextval('emp_seq'), 'Alice', 'Johnson', 'alice.johnson@example.com', '2023-04-10');
```

31. Create a Function

sql

Copy code

```
CREATE FUNCTION get_full_name(emp_id INT) RETURNS TEXT AS $$ DECLARE full_name TEXT; BEGIN SELECT first_name || ' ' || last_name INTO full_name FROM employees WHERE id = emp_id; RETURN full_name; END; $$ LANGUAGE plpgsql;
```


32. Call a Function

sql

Copy code

```
SELECT get_full_name(1);
```

33. Create a Trigger

sql

Copy code

```
CREATE TRIGGER update_timestamp BEFORE UPDATE ON employees FOR  
EACH ROW EXECUTE PROCEDURE update_timestamp();
```

34. Create Trigger Function

sql

Copy code

```
CREATE FUNCTION update_timestamp() RETURNS TRIGGER AS $$ BEGIN  
NEW.updated_at = NOW(); RETURN NEW; END; $$ LANGUAGE plpgsql;
```

35. Add Foreign Key

sql

Copy code

```
ALTER TABLE employees ADD CONSTRAINT fk_department FOREIGN KEY  
(department_id) REFERENCES departments(id);
```

36. Drop Foreign Key

sql

Copy code

```
ALTER TABLE employees DROP CONSTRAINT fk_department;
```

37. Create a Table with Foreign Key

sql

Copy code

```
CREATE TABLE departments ( id SERIAL PRIMARY KEY, name VARCHAR(100) );  
CREATE TABLE employees ( id SERIAL PRIMARY KEY, first_name VARCHAR(50),  
last_name VARCHAR(50), email VARCHAR(100), hire_date DATE,  
department_id INT, CONSTRAINT fk_department FOREIGN KEY(department_id)  
REFERENCES departments(id) );
```

38. Insert Data with Foreign Key

sql

Copy code

```
INSERT INTO departments (name) VALUES ('HR'), ('IT'), ('Finance'); INSERT INTO  
employees (first_name, last_name, email, hire_date, department_id) VALUES  
('John', 'Doe', 'john.doe@example.com', '2023-01-15', 1);
```

39. Select with Join

sql

Copy code

```
SELECT e.first_name, e.last_name, d.name AS department_name FROM  
employees e JOIN departments d ON e.department_id = d.id;
```

40. Left Join

sql

Copy code

```
SELECT e.first_name, e.last_name, d.name AS department_name FROM  
employees e LEFT JOIN departments d ON e.department_id = d.id;
```

41. Right Join

sql

Copy code

```
SELECT e.first_name, e.last_name, d.name AS department_name FROM  
employees e RIGHT JOIN departments d ON e.department_id = d.id;
```

42. Full Join

sql

Copy code

```
SELECT e.first_name, e.last_name, d.name AS department_name FROM  
employees e FULL JOIN departments d ON e.department_id = d.id;
```

43. Cross Join

sql

Copy code

```
SELECT e.first_name, d.name AS department_name FROM employees e CROSS  
JOIN departments d;
```

44. Create a Composite Primary Key

sql

Copy code

```
CREATE TABLE project_assignments ( employee_id INT, project_id INT,  
assigned_date DATE, PRIMARY KEY (employee_id, project_id) );
```

45. Insert Data with Composite Key

sql

Copy code

```
INSERT INTO project_assignments (employee_id, project_id, assigned_date)  
VALUES (1, 101, '2023-05-01');
```

46. Create a Table with Unique Constraint

sql

Copy code

```
CREATE TABLE users ( id SERIAL PRIMARY KEY, username VARCHAR(50)  
UNIQUE, email VARCHAR(100) UNIQUE );
```

47. Insert Data with Unique Constraint

sql

Copy code

```
INSERT INTO users (username, email) VALUES ('johndoe',  
'john.doe@example.com');
```

48. Create a Table with Check Constraint

sql

Copy code

```
CREATE TABLE products ( id SERIAL PRIMARY KEY, name VARCHAR(100), price  
NUMERIC CHECK (price > 0) );
```

49. Insert Data with Check Constraint

sql

Copy code

```
INSERT INTO products (name, price) VALUES ('Laptop', 999.99);
```

50. Create a Table with Default Value

sql

Copy code

```
CREATE TABLE orders ( id SERIAL PRIMARY KEY, order_date DATE DEFAULT  
CURRENT_DATE );
```

51. Insert Data with Default Value

sql

Copy code

```
INSERT INTO orders DEFAULT VALUES;
```

52. Create a Table with Not Null Constraint

sql

Copy code

```
CREATE TABLE categories ( id SERIAL PRIMARY KEY, name VARCHAR(100) NOT  
NULL );
```

53. Insert Data with Not Null Constraint

sql

Copy code

```
INSERT INTO categories (name) VALUES ('Electronics');
```

54. Create a Table with Enum

sql

Copy code

```
CREATE TYPE order_status AS ENUM ('pending', 'shipped', 'delivered', 'canceled'); CREATE TABLE orders ( id SERIAL PRIMARY KEY, status order_status );
```

55. Insert Data with Enum

sql

Copy code

```
INSERT INTO orders (status) VALUES ('pending');
```

56. Update Enum Value

sql

Copy code

```
UPDATE orders SET status = 'shipped' WHERE id = 1;
```

57. Create a JSON Column

sql

Copy code

```
CREATE TABLE customers ( id SERIAL PRIMARY KEY, info JSON );
```

58. Insert JSON Data

sql

Copy code

```
INSERT INTO customers (info) VALUES ('{"name": "Alice", "age": 30}');
```

59. Query JSON Data

sql

Copy code

```
SELECT info->>'name' AS name FROM customers;
```

60. Create a JSONB Column

sql

Copy code

```
CREATE TABLE products ( id SERIAL PRIMARY KEY, details JSONB );
```

61. Insert JSONB Data

sql

Copy code

```
INSERT INTO products (details) VALUES ('{"name": "Laptop", "price": 999.99}');
```

62. Query JSONB Data

sql

Copy code

```
SELECT details->>'name' AS name FROM products;
```

63. Create a Table with Array

sql

Copy code

```
CREATE TABLE meetings ( id SERIAL PRIMARY KEY, attendees TEXT[] );
```

64. Insert Array Data

sql

Copy code

```
INSERT INTO meetings (attendees) VALUES (ARRAY['Alice', 'Bob']);
```

65. Query Array Data

sql

Copy code

```
SELECT * FROM meetings WHERE 'Alice' = ANY(attendees);
```

66. Create a Table with Hstore

sql

Copy code

```
CREATE EXTENSION IF NOT EXISTS hstore; CREATE TABLE settings ( id SERIAL PRIMARY KEY, configuration HSTORE );
```

67. Insert Hstore Data

sql

Copy code

```
INSERT INTO settings (configuration) VALUES ('email => john.doe@example.com, phone => 123-456-7890');
```

68. Query Hstore Data

sql

Copy code

```
SELECT configuration->'email' AS email FROM settings;
```

69. Create a Table with XML

sql

Copy code

```
CREATE TABLE books ( id SERIAL PRIMARY KEY, info XML );
```

70. Insert XML Data

sql

Copy code

```
INSERT INTO books (info) VALUES ('<book><title>PostgreSQL</title><author>Author Name</author></book>');
```

71. Query XML Data

sql

Copy code

```
SELECT xpath('//title/text()', info) FROM books;
```

72. Create a Temporary Table

sql

Copy code

```
CREATE TEMPORARY TABLE temp_employees ( id SERIAL PRIMARY KEY, first_name VARCHAR(50), last_name VARCHAR(50) );
```

73. Insert Data into Temporary Table

sql

Copy code

```
INSERT INTO temp_employees (first_name, last_name) VALUES ('Temporary', 'Employee');
```

74. Query Temporary Table

sql

Copy code

```
SELECT * FROM temp_employees;
```

75. Create a Table with Range

sql

Copy code

```
CREATE TABLE events ( id SERIAL PRIMARY KEY, event_time TSRANGE );
```

76. Insert Range Data

sql

Copy code

```
INSERT INTO events (event_time) VALUES ('[2023-01-01 10:00, 2023-01-01 12:00)');
```

77. Query Range Data

sql

Copy code

```
SELECT * FROM events WHERE event_time @> '2023-01-01 11:00';
```

78. Create a Table with CIDR

sql

Copy code

```
CREATE TABLE networks ( id SERIAL PRIMARY KEY, network CIDR );
```

79. Insert CIDR Data

sql

Copy code

```
INSERT INTO networks (network) VALUES ('192.168.1.0/24');
```

80. Query CIDR Data

sql

Copy code

```
SELECT * FROM networks WHERE network >= '192.168.1.100';
```

81. Create a Table with INET

sql

Copy code

```
CREATE TABLE devices ( id SERIAL PRIMARY KEY, ip INET );
```

82. Insert INET Data

sql

Copy code

```
INSERT INTO devices (ip) VALUES ('192.168.1.100');
```

83. Query INET Data

sql

Copy code

```
SELECT * FROM devices WHERE ip << '192.168.1.0/24';
```

84. Create a Table with UUID

sql

Copy code

```
CREATE TABLE sessions ( id UUID PRIMARY KEY, user_id INT, login_time  
TIMESTAMP );
```

85. Insert UUID Data

sql

Copy code

```
INSERT INTO sessions (id, user_id, login_time) VALUES (uuid_generate_v4(), 1, NOW());
```

86. Query UUID Data

sql

Copy code

```
SELECT * FROM sessions;
```

87. Create a Table with Geometric Types

sql

Copy code

```
CREATE TABLE shapes ( id SERIAL PRIMARY KEY, circle CIRCLE );
```

88. Insert Geometric Data

sql

Copy code

```
INSERT INTO shapes (circle) VALUES ('<(0,0),10>');
```

89. Query Geometric Data

sql

Copy code

```
SELECT * FROM shapes WHERE circle @> '(5,5)';
```

90. Create a Table with Point

sql

Copy code

```
CREATE TABLE locations ( id SERIAL PRIMARY KEY, position POINT );
```

91. Insert Point Data

sql

Copy code

```
INSERT INTO locations (position) VALUES ('(10, 20)');
```

92. Query Point Data

sql

Copy code

```
SELECT * FROM locations WHERE position <@ BOX '((0,0),(15,25))';
```

93. Create a Table with LSEG

sql

Copy code

```
CREATE TABLE lines ( id SERIAL PRIMARY KEY, segment LSEG );
```

94. Insert LSEG Data

sql

Copy code

```
INSERT INTO lines (segment) VALUES ('[(0,0),(1,1)]');
```

95. Query LSEG Data

sql

Copy code

```
SELECT * FROM lines WHERE segment ?# '[(0,0),(2,2)]';
```

96. Create a Table with Path

sql

Copy code

```
CREATE TABLE paths ( id SERIAL PRIMARY KEY, path PATH );
```

97. Insert Path Data

sql

Copy code

```
INSERT INTO paths (path) VALUES ('((0,0),(1,1),(2,2))');
```

98. Query Path Data

sql

Copy code

```
SELECT * FROM paths WHERE path @> '((1,1),(2,2))';
```

99. Create a Table with Polygon

sql

Copy code

```
CREATE TABLE polygons ( id SERIAL PRIMARY KEY, poly POLYGON );
```

100. Insert Polygon Data

sql

Copy code

```
INSERT INTO polygons (poly) VALUES ('((0,0),(0,1),(1,1),(1,0))');
```

101. Query Polygon Data

sql

Copy code

```
SELECT * FROM polygons WHERE poly @> '(0.5,0.5)';
```

To execute these queries in the PostgreSQL shell, you can simply copy and paste them one by one after connecting to your database with the **psql** command. Adjust the data and structure as per your specific requirements.