Great! Here's a complete **PostgreSQL dataset** setup with multiple tables that cover nearly **all your practice questions**, along with **which tables to use for which topics**. You'll find:

✅ 1. Table Creation Scripts

```
-- CAR Table
CREATE TABLE car (
    id SERIAL PRIMARY KEY,
    carname VARCHAR(50),
    color VARCHAR(20),
    price NUMERIC(10,2),
    model_year INT
);

-- EMPLOYEE Table
CREATE TABLE employee (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    salary NUMERIC(10,2),
    date_of_joining DATE,
    department_id INT
);

-- DEPARTMENT Table
CREATE TABLE department (
    id SERIAL PRIMARY KEY,
    dept_name VARCHAR(50),
    manager_id INT
);

-- CUSTOMER Table
```

```
CREATE TABLE customer (
   id SERIAL PRIMARY KEY,
   full_name VARCHAR(100),
   total_spending NUMERIC(10,2)
);

-- PRODUCT Table
CREATE TABLE product (
   id SERIAL PRIMARY KEY,
   product_name VARCHAR(100),
   price NUMERIC(10,2)
);

-- ORDER Table
CREATE TABLE orders (
   id SERIAL PRIMARY KEY,
   customer_id INT REFERENCES customer(id),
   product_id INT REFERENCES product(id),
   order_amount NUMERIC(10,2),
   order_date DATE
);

-- STUDENT Table
CREATE TABLE student (
   id SERIAL PRIMARY KEY,
   full_name VARCHAR(100),
   age INT CHECK (age >= 18),
   department_name VARCHAR(50)
);
```

✅ **2. Insert Data**

```
-- CARS
INSERT INTO car (carname, color, price, model_year) VALUES
('Swift', 'Red', 500000, 2020),
('Baleno', 'Blue', 600000, 2021),
('i20', 'White', 650000, 2022),
('Verna', 'Black', 800000, 2023),
('Fortuner', 'White', 3500000, 2024),
('Zen', 'Green', 250000, 2018);

-- DEPARTMENTS
INSERT INTO department (dept_name, manager_id) VALUES
('Engineering', 1),
('HR', 2),
('Marketing', NULL),
('Sales', 3);

-- EMPLOYEES
INSERT INTO employee (first_name, last_name, salary, date_of_joining, department_id) VALUES
('David', 'Smith', 85000, '2023-01-10', 1),
('Daniel', 'Jones', 75000, '2022-06-15', 1),
('Amy', 'Brown', 65000, '2021-03-20', 2),
('Mark', 'Taylor', 70000, '2023-07-01', 2),
('Ethan', 'Clark', 95000, '2020-10-05', 1),
('Linda', 'Martin', 68000, '2024-01-12', 3),
('Sara', 'Johnson', 88000, '2021-09-25', NULL);

-- CUSTOMERS
INSERT INTO customer (full_name, total_spending) VALUES
('Ravi Kumar', 25000),
('Sneha Das', 50000),
('Kiran Rao', 0),
```

('Raj Mehta', 90000),

('Meena Kumari', 120000);

-- PRODUCTS

INSERT INTO product (product_name, price) VALUES

('Laptop', 70000),

('Phone', 30000),

('Headphones', 5000),

('Monitor', 12000),

('Mouse', 1500);

-- ORDERS

INSERT INTO orders (customer_id, product_id, order_amount, order_date) VALUES

(1, 1, 70000, '2024-02-01'),

(1, 5, 1500, '2024-02-01'),

(2, 2, 30000, '2024-03-10'),

(2, 3, 5000, '2024-03-15'),

(4, 1, 70000, '2024-05-01');

-- STUDENTS

INSERT INTO student (full_name, age, department_name) VALUES

('Rahul Sharma', 19, 'Engineering'),

('Priya Singh', 20, 'Engineering'),

('Karan Malhotra', 21, 'HR'),

('Sneha Kapoor', 22, 'Marketing'),

('Arjun Verma', 23, 'Engineering'),

('Rita Mehra', 19, 'Science');

✅ **3. Suggested Practice Mapping**

Topic	Relevant Tables	Example Questions to Practice
Table Creation/Modification	car, employee, department, student	- Create foreign key - Add/rename columns - CHECK, DEFAULT, AUTO_INCREMENT
Insertions	All	- Batch inserts, insert with default, stored procedures
Basic Selects	All	- Display records, filtering, sorting
Aggregations	employee, car, product, orders	- AVG, COUNT, MAX, SUM - Group by age, department
Subqueries	car, employee, customer, orders	- Second highest price/salary - Products above avg price
Joins	employee + department, orders + customer/product	- Join + filter, department with highest avg salary
Views	car, employee	- car_price view - Employees starting with 'D'
CTEs & Window Functions	employee, car, student	- Rank employees by salary - Dense_rank to get top 3
Triggers	Use orders or employee	- On INSERT/UPDATE to log table
Transactions	orders, employee	- COMMIT, ROLLBACK for salary update
Update/Delete	employee, car, student	- Delete by condition - Update salary of 'D'
Indexes	product, employee	- Create index on salary or price
Security	customer, orders	- GRANT SELECT - Prevent injection
Complex Joins	All	- Manager with most employees - Departments without employees

---

🔁 **Practice Order Suggestion:**

1. Start with car – simple structure: use for filters, aggregations, views.
2. Move to employee + department – test joins, grouping, updates, CTEs.
3. Then practice customer, product, orders – good for subqueries, joins, transactions.
4. student table – good for age-based filters, CHECK constraints, grouping by category.

Here are the **PostgreSQL practice questions** only, categorized by difficulty and table:

🚀 Beginner Level (🔰)

[car]

1. Show all cars.
2. List all cars that cost more than ₹600000.
3. Show cars of color "White".
4. Count cars of each color.

[employee]

5. Show employee names and salaries.
6. Find employees with salary > ₹80,000.
7. List employees who joined in the last 2 years.

🧩 Intermediate Level (🟡)

[employee + department]

8. Show employees and their department names.
9. Count how many employees each department has.
10. Find departments without employees.

[orders + customer]

11. Show each customer's total order amount.
12. Find customers who haven’t placed any orders.
13. Find the number of orders placed per product.

[student]

14. Group students by age and count how many share the same age.
15. List students not in 'HR' or 'Marketing'.

🧠 Advanced Level (🔴)

[employee]

16. Find the second highest salary.
17. Find employees with duplicate salaries.

18. Use window functions to rank employees by salary.
19. Show the latest 3 employees with department names.

[orders + customer + product]

20. Find total products purchased by each customer.
21. Find products not purchased by anyone.

🛠️ **Challenge Level (🔥)**

22. Find the model year of the car that appears most frequently.
23. Find the department with the highest average salary.
24. Find the manager with the highest number of employees.
25. Find the full name of the employee with the longest name.

Absolutely! Here's a **PostgreSQL Practice Query Workout Set** 🔄 categorized by **difficulty levels**, tied to the **tables you've created** (car, employee, department, student, product, customer, orders).

🔄 **PostgreSQL Query Workouts**

🚀 **Beginner Level (🔰)**

[car]

1. 🔰 Show all cars.
2. SELECT * FROM car;
3. 🔰 List all cars that cost more than ₹600000.
4. SELECT carname, price FROM car WHERE price > 600000;
5. 🔰 Show cars of color "White".
6. SELECT carname FROM car WHERE color = 'White';
7. 🔰 Count cars of each color.
8. SELECT color, COUNT(*) FROM car GROUP BY color;

[employee]

5. 🔰 Show employee names and salaries.
6. SELECT first_name || ' ' || last_name AS full_name, salary FROM employee;
7. 🔰 Find employees with salary > ₹80,000.
8. SELECT * FROM employee WHERE salary > 80000;
9. 🔰 List employees joined in last 2 years.
10. SELECT * FROM employee WHERE date_of_joining >= CURRENT_DATE - INTERVAL '2 years';

🧩 **Intermediate Level (🟡)**

[employee + department]

8. 🟡 Show employees and their department names.
9. SELECT e.first_name, d.dept_name
10. FROM employee e
11. LEFT JOIN department d ON e.department_id = d.id;

12. 🟡 Count how many employees each department has.
13. SELECT d.dept_name, COUNT(e.id) AS employee_count
14. FROM department d
15. LEFT JOIN employee e ON e.department_id = d.id
16. GROUP BY d.dept_name;
17. 🟡 Find departments without employees.

```
SELECT d.*
FROM department d
LEFT JOIN employee e ON e.department_id = d.id
WHERE e.id IS NULL;
```

[orders + customer]

11. 🟡 Show each customer's total order amount.

```
SELECT c.full_name, SUM(o.order_amount) AS total_spent
FROM customer c
JOIN orders o ON c.id = o.customer_id
GROUP BY c.full_name;
```

12. 🟡 Find customers who haven’t placed any orders.

```
SELECT c.*
FROM customer c
LEFT JOIN orders o ON c.id = o.customer_id
WHERE o.id IS NULL;
```

13. 🟡 Find the number of orders placed per product.

```
SELECT p.product_name, COUNT(o.id) AS order_count
FROM product p
LEFT JOIN orders o ON p.id = o.product_id
GROUP BY p.product_name;
```

[student]

14. 🟡 Group students by age and count how many share the same age.

```
SELECT age, COUNT(*) FROM student GROUP BY age;
```

15. 🟡 List students not in 'HR' or 'Marketing'.

```
SELECT * FROM student WHERE department_name NOT IN ('HR', 'Marketing');
```

🧠 **Advanced Level (🔴)**

[employee]

16. 🔴 Find the second highest salary.

```
SELECT MAX(salary) AS second_highest
FROM employee
WHERE salary < (SELECT MAX(salary) FROM employee);
```

17. 🔴 Find employees with duplicate salaries.

```
SELECT salary, COUNT(*)
FROM employee
GROUP BY salary
HAVING COUNT(*) > 1;
```

18. 🔴 Use window functions to rank employees by salary.

```
SELECT *, RANK() OVER (ORDER BY salary DESC) AS sal_rank FROM employee;
```

19. 🔴 Show latest 3 employees with department names.

```
SELECT e.*, d.dept_name
FROM employee e
JOIN department d ON e.department_id = d.id
ORDER BY date_of_joining DESC
LIMIT 3;
```

[orders + customer + product]

20. 🔴 Find total products purchased by each customer.

```
SELECT c.full_name, COUNT(o.product_id) AS products_bought
FROM customer c
JOIN orders o ON c.id = o.customer_id
GROUP BY c.full_name;
```

21. 🔴 Find products not purchased by anyone.

```
SELECT * FROM product
WHERE id NOT IN (SELECT DISTINCT product_id FROM orders);
```

🛠️ Challenge Level (🔥 Pro Skills)

22. 🔥 Find the model of the car that appears most frequently.

```
SELECT model_year, COUNT(*)
FROM car
GROUP BY model_year
ORDER BY COUNT(*) DESC
LIMIT 1;
```

23. 🔥 Find the department with the highest average salary.

```
SELECT d.dept_name, AVG(e.salary) AS avg_salary
FROM department d
JOIN employee e ON d.id = e.department_id
GROUP BY d.dept_name
ORDER BY avg_salary DESC
LIMIT 1;
```

24. 🔥 Find manager with most employees.

```
SELECT d.manager_id, COUNT(e.id) AS emp_count
FROM department d
JOIN employee e ON d.id = e.department_id
GROUP BY d.manager_id
ORDER BY emp_count DESC
LIMIT 1;
```

25. 🔥 Find full name of employee with longest full name.

```
SELECT first_name || ' ' || last_name AS full_name
FROM employee
ORDER BY LENGTH(first_name || ' ' || last_name) DESC
LIMIT 1;
```
