dataford.io

PostgreSQL Mastery Cheat Sheet

Dataford

1. Basic SQL Queries

1. Select All Data

Query to retrieve all columns from the `users` table.

```sql
SELECT *
FROM users;
```

2. Select Specific Column

Retrieve only the `first_name` column from the `customers` table.

```sql
SELECT first_name
FROM customers;
```

3. Filter Records by Condition

Fetch all users where the first name is "john" from the `user_details` table.

```sql
SELECT *
FROM user_details
WHERE first_name = 'john';
```

4. Sort Records

Get all product names sorted by price in descending order from the products table.

```sql
SELECT product_name
FROM products
ORDER BY price DESC;
```

5. Filter Data by Range

Retrieve all employees whose age is between 25 and 40 from the `employees` table.

```sql
SELECT *
FROM employees
WHERE age BETWEEN 25 AND 40;
```

6. Find Null Values

Select users from the `customers` table who don't have an email address.

```sql
SELECT *
FROM customers
WHERE email IS NULL;
```

7. Remove Duplicate Entries

Fetch unique `departments` from the departments table.

```sql
SELECT DISTINCT department
FROM departments;
```

2. Aggregation Functions

8. Count Records

Count the total number of orders from the `orders` table.

```sql
SELECT COUNT(*)
FROM orders;
```

9. Sum Column Values

Calculate the total sales from the `sales` table.

```sql
SELECT SUM(sale_amount)
FROM sales;
```

10. Find Maximum Value

Retrieve the highest salary from the `salaries` table.

```sql
SELECT MAX(salary)
FROM salaries;
```

11. Find Minimum Value

Retrieve the lowest score from the `exam_scores` table.

```sql
SELECT MIN(score)
FROM exam_scores;
```

12. Calculate Average

Get the average order amount from the `orders` table.

```sql
SELECT AVG(order_amount)
FROM orders;
```

3. Grouping

13. Group by and Count

Count the number of users per country from the `users` table.

```sql
SELECT country, COUNT(*)
FROM users
GROUP BY country;
```

14. Group by and Sum

Calculate the total sales per product category from the `product_sales` table.

```sql
SELECT category, SUM(sales)
FROM product_sales
GROUP BY category;
```

15. Group by and Filter (HAVING)

Get departments where the total salary is greater than $1,000,000.

```sql
SELECT department, SUM(salary)
FROM salaries
GROUP BY department
HAVING SUM(salary) > 1000000;
```

4. Joining Tables

16. Inner Join

Retrieve employee names along with their department names.

```sql
SELECT e.first_name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;
```

17. Left Join

Get all employees and their assigned projects, even if some employees don't have projects.

```sql
SELECT e.first_name, p.project_name
FROM employees e
LEFT JOIN projects p
ON e.employee_id = p.employee_id;
```

18. Right Join

List all projects and the employees assigned to them, including projects without employees.

```sql
SELECT p.project_name, e.first_name
FROM projects p
RIGHT JOIN employees e
ON p.project_id = e.project_id;
```

19. Full Outer Join

Retrieve all employees and all projects, including those with no matching records.

```sql
SELECT e.first_name, p.project_name
FROM employees e
FULL OUTER JOIN projects p
ON e.employee_id = p.employee_id;
```

6. Window Functions

23. Rank Over Partition

Rank products based on sales within each category.

```sql
SELECT product_id, category_id, sales_amount,
       RANK() OVER (PARTITION BY category_id ORDER BY
sales_amount DESC) AS rank
FROM products;
```

24. Running Total

Calculate the cumulative sales for each product over time.

```sql
SELECT product_id, sale_date, sales_amount,
       SUM(sales_amount) OVER (ORDER BY sale_date) AS
running_total
FROM sales;
```

25. Lag Function

Find the previous order amount for each user.

```sql
SELECT user_id, order_date, order_amount,
       LAG(order_amount, 1) OVER (PARTITION BY user_id
ORDER BY order_date) AS previous_order
FROM orders;
```

5. Subqueries

20. Subquery in WHERE

Fetch all products with a price greater than the average price.

```sql
SELECT *
FROM products
WHERE price > (SELECT AVG(price) FROM products);
```

21. Subquery in SELECT

Retrieve employee names and their corresponding department names using a subquery.

```sql
SELECT first_name,
      (SELECT department_name
       FROM departments
       WHERE department_id = employees.department_id) AS
department
FROM employees;
```

22. Subquery in FROM

Find the total sales per product using a subquery.

```sql
SELECT department, SUM(salary)
FROM salaries
GROUP BY department
HAVING SUM(salary) > 1000000;
```

7. String Functions

26. Concatenate Strings

Combine first name and last name to display full name.

```sql
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM users;
```

27. String Length

Get the length of the product name.

```sql
SELECT LENGTH(product_name)
FROM products;
```

28. Substring

Extract the first 3 characters of the product name.

```sql
SELECT SUBSTRING(product_name, 1, 3)
FROM products;
```

29. Upper and Lower Case

Display customer names in uppercase.

```sql
SELECT UPPER(customer_name)
FROM customers;
```

8. Date and Time Functions

30. Current Date

Retrieve all users who joined today.

```sql
SELECT *
FROM users
WHERE join_date = CURRENT_DATE;
```

31. Extract Year

Get the year from the `join_date` column.

```sql
SELECT EXTRACT(YEAR FROM join_date)
FROM users;
```

32. Date Difference

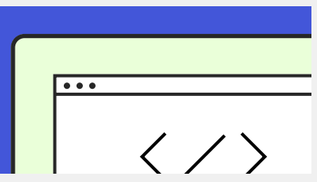
Calculate the number of days between `order_date` and today.

```sql
SELECT age(CURRENT_DATE, order_date)
FROM orders;
```

33. Add Date Interval

Find all users who signed up in the last 30 days.

```sql
SELECT *
FROM users
WHERE join_date > CURRENT_DATE - INTERVAL '30 days';
```

9. Advanced SQL

34. Case Statements

Display gender as 'M' or 'F' based on the `gender` column.

```sql
SELECT first_name, last_name,
       CASE
         WHEN gender = 'male' THEN 'M'
         WHEN gender = 'female' THEN 'F'
       END AS gender_abbr
FROM users;
```

35. Coalesce Function

Replace null values in the `address` column with 'unknown'.

```sql
SELECT COALESCE(address, 'unknown')
FROM customers;
```

36. Casting and Conversion

Convert `order_amount` to a decimal value with 2 decimal places.

```sql
SELECT CAST(order_amount AS DECIMAL(10, 2))
FROM orders;
```

37. Recursive Query

Generate a sequence of numbers from 1 to 10.

```sql
WITH numbers AS (
    SELECT 1 AS number
    UNION ALL
    SELECT number + 1
    FROM numbers
    WHERE number < 10
)
SELECT number
FROM numbers;
```

38. Cross Join Example

Retrieve a combination of every employee with every project.

```sql
SELECT e.first_name, p.project_name
FROM employees e
CROSS JOIN projects p;
```

39. Cross Join with Filter

Perform a cross join between employees and projects, but only show combinations where the employee's department matches the project's department.

```sql
SELECT e.first_name, p.project_name
FROM employees e
CROSS JOIN projects p
WHERE e.department_id = p.department_id;
```