PostgreSQL Querying Guide

Section 1. Querying Data
Select – show you how to query data from a single table.
Column aliases – learn how to assign temporary names to columns or expressions within a query.
Order By – guide you on how to sort the result set returned by a query.
Select Distinct – show you how to remove duplicate rows from the result set.

```
SELECT
   select_list
FROM
   table_name;
```

PostgreSQL evaluates the FROM clause before the SELECT clause in the SELECT statement:

```
SELECT column_name AS alias_name
FROM table_name;
```

```
SELECT
  select_list
FROM
  table_name
ORDER BY
  sort_expression1 [ASC | DESC],
  sort_expression2 [ASC | DESC],
  ...;
```

PostgreSQL evaluates the clauses in the SELECT statement in the following order: FROM, SELECT, and ORDER BY:

```
SELECT
  DISTINCT column1
FROM
  table_name;
```

Make Document for this

PostgreSQL: Querying Data

1. SELECT Statement

The `SELECT` statement is used to retrieve data from one or more tables.

Syntax:

```sql
SELECT select_list
FROM table_name;
```

- `select_list`: Specifies the columns or expressions to be retrieved.
- `table_name`: Specifies the table to query from.

Execution Order of SELECT Statement:
PostgreSQL evaluates the clauses in the following order:

1. **FROM** clause (to determine the data source).
2. **SELECT** clause (to choose the columns).
3. **ORDER BY** clause (to sort the result set).

---

2. Column Aliases

Column aliases are used to assign a temporary name to a column or an expression in the result set. This can make the output more readable.

Syntax:

```sql
SELECT column_name AS alias_name
FROM table_name;
```

Example:
Suppose you have a table `employees` with columns `first_name` and `last_name`.

```sql
SELECT first_name AS "First Name", last_name AS "Last Name"
FROM employees;
```

Output:

First Name	Last Name
John	Doe
Jane	Smith

---

3. ORDER BY Clause

The `ORDER BY` clause is used to sort the result set based on one or more columns or expressions. You can specify ascending (`ASC`, default) or descending (`DESC`) order.

Syntax:

```sql
SELECT select_list
FROM table_name
ORDER BY sort_expression1 [ASC | DESC], sort_expression2 [ASC | DESC], ...;
```

Example:
Sort employees by `last_name` in ascending order and `first_name` in descending order.

```sql
SELECT first_name, last_name
FROM employees
ORDER BY last_name ASC, first_name DESC;
```

Output:

First Name	Last Name
Jane	Doe
John	Doe
Alan	Smith

---

4. SELECT DISTINCT

The `SELECT DISTINCT` statement is used to remove duplicate rows from the result set.

Syntax:

```sql
SELECT DISTINCT column1
FROM table_name;
```

Example:
Retrieve distinct department names from the `employees` table.

```sql
SELECT DISTINCT department
FROM employees;
```

Output:

Department
HR
IT
Sales

Summary of Clauses Evaluation Order in PostgreSQL

1. **FROM**: Specifies the table(s) to query.
2. **SELECT**: Defines the columns or expressions to return.
3. **ORDER BY**: Sorts the result set based on specified criteria.

Document Example Query

```sql


-- Query to retrieve distinct department names, sort by name in ascending order:
SELECT DISTINCT department AS "Department Name"
FROM employees
ORDER BY "Department Name" ASC;
```

Output:

Department Name
HR
IT
Sales

Section 4. Grouping Data
Group By – divide rows of a result set into groups and optionally apply an aggregate function to each group.
Having – apply conditions to groups, which allow you to filter groups.

SynTX
SELECT
  column_1,
  column_2,
  ...,
  aggregate_function(column_3)
FROM
  table_name
GROUP BY
  column_1,
  column_2,
  ...;

PostgreSQL evaluates the GROUP BY clause after the FROM and WHERE clauses and before the HAVING SELECT, DISTINCT, ORDER BY and LIMIT clauses.

The GROUP BY clause divides the rows returned from the SELECT statement into groups.

For each group, you can apply an aggregate function such as SUM() to calculate the sum of items or COUNT() to get the number of items in the groups.

PostgreSQL evaluates the HAVING clause after the FROM, WHERE, GROUP BY, and before the DISTINCT, SELECT, ORDER BY and LIMIT clauses:

HAVING vs. WHERE
The WHERE clause filters the rows based on a specified condition whereas the HAVING clause filter groups of rows according to a specified condition.

In other words, you apply the condition in the WHERE clause to the rows while you apply the condition in the HAVING clause to the groups of rows.

Make Document

PostgreSQL: Grouping Data

1. GROUP BY Clause

The `GROUP BY` clause is used to divide rows into groups based on one or more columns. You can then apply aggregate functions (e.g., `SUM()`, `COUNT()`, `AVG()`, `MAX()`, `MIN()`) to these groups for calculations.

Syntax:

```sql


  SELECT
     column_1,
     column_2,
     ...,
     aggregate_function(column_3)
  FROM
     table_name
  GROUP BY
     column_1,
     column_2,
     ...;
```

Key Points:

- PostgreSQL evaluates the `GROUP BY` clause after the `FROM` and `WHERE` clauses.
- It is evaluated **before** the `HAVING`, `SELECT`, `DISTINCT`, `ORDER BY`, and `LIMIT` clauses.

Example:
Suppose you have a table `sales` with the following columns:

- `region` (sales region),
- `product` (product name),
- `revenue` (sales revenue).

To calculate the total revenue per region:

```sql
SELECT
   region,
   SUM(revenue) AS total_revenue
FROM
   sales
GROUP BY
   region;
```

Output:

Region	Total Revenue
North	50000
South	75000
East	30000

---

2. HAVING Clause

The `HAVING` clause is used to filter groups created by the `GROUP BY` clause based on a specified condition.

Syntax:

```sql
SELECT
   column_1,
   aggregate_function(column_2)
FROM
   table_name
GROUP BY
   column_1
HAVING
   condition;
```

Key Points:

- PostgreSQL evaluates the `HAVING` clause **after** the `FROM`, `WHERE`, and `GROUP BY` clauses.
- It is evaluated **before** the `DISTINCT`, `SELECT`, `ORDER BY`, and `LIMIT` clauses.

Difference Between WHERE and HAVING

WHERE Clause	HAVING Clause
Filters rows before grouping.	Filters groups after grouping.
Cannot use aggregate functions.	Can use aggregate functions.

Example:
Using the same `sales` table, filter regions where total revenue exceeds 50,000:

```sql
SELECT
   region,
   SUM(revenue) AS total_revenue
FROM
   sales
GROUP BY
   region
HAVING
   SUM(revenue) > 50000;
```

Output:

Region	Total Revenue
South	75000

Summary of Evaluation Order in PostgreSQL for GROUPING DATA

1. **FROM**: Specifies the table(s) to query.
2. **WHERE**: Filters rows before grouping.
3. **GROUP BY**: Groups rows based on specified columns.
4. **HAVING**: Filters groups based on conditions.
5. **SELECT**: Retrieves columns and aggregate calculations.
6. **ORDER BY**: Sorts the result set.
7. **LIMIT**: Limits the number of rows returned.

Document Example Query

Scenario:
Suppose you want to know the total revenue of each product in the `East` region, and only include products where revenue exceeds 20,000.

```sql
SELECT
   product,
   SUM(revenue) AS total_revenue
FROM
   sales
WHERE
   region = 'East'
GROUP BY
   product
HAVING
   SUM(revenue) > 20000
ORDER BY
   total_revenue DESC;
```

Output:

Product	Total Revenue
Product A	25000