Multiple GROUP BY using GROUPING SETS in Single SQL Query

In this post, I am sharing one demonstration of PostgreSQL GROUPING SETS.
This is very useful for PostgreSQL Database Developers who require to perform multiple GROUP BY in one single query.

Now take one example, you want to find the count of Employee based on two columns: Employee Department , Employee Joining Year.
For this query, you have to write different two queries and If you want to combine results of both the query, you should use UNION clause.

But using GROUPING SETS, we can prepare this result into one single query.

Below is a full demonstration of this:

Create a table with Sample data:

```
CREATE TABLE tbl_Employees
(
EmpID INT
,EmpName CHARACTER VARYING
,EmpDepartment CHARACTER VARYING
,JoiningDate TIMESTAMP WITHOUT TIME ZONE
);

INSERT INTO tbl_Employees
VALUES
(1,'Anvesh','Database','2012-09-06')
,(2,'Roy','Animation','2012-10-08')
,(3,'Martin','JAVA','2013-12-25')
,(4,'Eric','JAVA','2014-01-26')
,(5,'Jenny','JAVA','2014-05-20')
,(6,'Kavita','Database','2014-12-12')
,(7,'Marlin','SQL','2015-08-08')
,(8,'Mahesh','PHP','2016-06-16');
```

Count of Employee, Group By Department column:

```
SELECT
EmpDepartment
,COUNT(1) AS EmployeeCount
FROM tbl_Employees
GROUP BY EmpDepartment;
```

Count of Employee, Group By Joining Year:

```
SELECT
EXTRACT(YEAR FROM JoiningDate) AS JoiningYear
,COUNT(1) AS EmployeeCount
FROM tbl_Employees
GROUP BY EXTRACT(YEAR FROM JoiningDate);
```

GROUP BY above two queries using GROUPING SETS:

```
SELECT
EmpDepartment
,EXTRACT(YEAR FROM JoiningDate) AS JoiningYear
,COUNT(1) AS EmployeeCount
FROM tbl_Employees
GROUP BY GROUPING SETS (EmpDepartment,EXTRACT(YEAR FROM JoiningDate));
```

```
SELECT
EmpDepartment, null ,COUNT(1) AS EmployeeCount
FROM tbl_Employees
GROUP BY EmpDepartment
union all
SELECT null,EXTRACT(YEAR FROM JoiningDate) AS JoiningYear,COUNT(1) AS EmployeeCount
FROM tbl_Employees
GROUP BY EXTRACT(YEAR FROM JoiningDate);
```

```
select e.department_id ,e.job_id ,count(*)
from employees e
group by GROUPING SETS (department_id,job_id);
```

Explain Group by ROLLUP with an example

In this post, I am sharing a demonstration of PostgreSQL GROUP BY ROLLUP which we are using for data analytics purpose.

The data analytics is a huge area and there are hundreds of tools available in the market. OLTP System like PostgreSQL is not made for Data Analytics but supports medium level of data analytics on real-time data.

ROLLUP is generally used for analysis over hierarchical data like generating sub-total and grand-total.

Check the below example:

Create a table with sample data:

```
CREATE TABLE tbl_ProductSales (ID INT, Product_Category CHARACTER VARYING,
Product_Name CHARACTER VARYING, TotalSales INT);

INSERT INTO tbl_ProductSales
VALUES
(1,'Game','Mobo Game',200),(2,'Game','PKO Game',400)
,(3,'Fashion','Shirt',500),(4,'Fashion','Shorts',100);
```

Check the ROLLUP result:

```
SELECT
   Product_Category
   ,Product_Name
   ,SUM(TotalSales) AS TotalSales
FROM tbl_ProductSales
GROUP BY ROLLUP (Product_Category, Product_Name)
ORDER BY Product_Category, Product_Name
```

```
select d.department_name,e.first_name,sum(e.salary)
from departments d join employees e
```

```
on d.department_id =e.department_id
group by rollup(d.department_name,e.first_name)
order by  d.department_name,e.first_name;


select d.department_name,sum(e.salary)
from departments d join employees e
on d.department_id =e.department_id
group by d.department_name
order by  d.department_name;


--Grand Total =684400.00
select sum(e.salary)
from departments d join employees e
on d.department_id =e.department_id ;
```

STRING_AGG () to Concatenate String Per Each Group (Like SQL Server STUFF ())

Most of the Database Developers require to perform String Aggregation based on different group of records.

Since PostgreSQL 9.0, STRING_AGG(expression, delimiter) function is available to perform String Aggregation operation.
Using STRING_AGG(), We can concatenate strings using different type of delimiter symbols.

Example of STRING_AGG():

Create a sample Students table:

```
	CREATE TABLE tbl_Students
(
	StudID INT
	,StudName CHARACTER VARYING
```

```
	,StudGrades CHAR(1)
);
```

Insert few sample records:

```
INSERT INTO tbl_Students
VALUES
(1,'Anvesh','A'),(2,'Kimly','B')
,(3,'Jenny','C'),(4,'Ali','B')
,(5,'Mukesh','D'),(6,'Sofia','A')
,(7,'Roy','C'),(8,'Martin','C');
```

Concatenate Students Name per each Student Grade and arrange by Grade wise row (Using STRING_AGG()):

```
SELECT
	StudGrades
	,STRING_AGG(StudName,', ') AS StudPerGrade
FROM tbl_Students
GROUP BY StudGrades
ORDER BY 1 ;
```

```
SELECT
	d.department_name
	,STRING_AGG(e.first_name,', ') AS emp_per_department_name
FROM departments d join employees e
on d.department_id =e.department_id
GROUP BY d.department_name
ORDER BY 1 ;
```

Allow single NULL for UNIQUE Constraint Column

Once you define the UNIQUE constraint, you can insert N number of NULL values for that column which is the principal rule of UNIQUE Constraint.

What if I need an only single NULL record for UNIQUE Column, we should check the below demonstration.

Create a sample table with UNIQUE Constraint:

```
CREATE TABLE tbl_testunique (ID INTEGER UNIQUE);
```

Insert sample multiple NULLs:

You can insert all NULLs.

```
INSERT INTO tbl_testunique
VALUES (1),(NULL),(NULL),(NULL);
```

Now, Truncate the table:

```
truncate table tbl_testunique;
```

Create a UNIQUE INDEX with IS NULL:

```
CREATE UNIQUE INDEX idx_id ON tbl_testunique ((ID IS NULL))
WHERE ID IS NULL;
```

Now, Try to execute same INSERT statement:

```
INSERT INTO tbl_testunique
VALUES (1),(NULL),(NULL),(NULL);
```

How to convert Table Data into JSON formatted Data?

PostgreSQL 9.4 introduced very powerful data type called JSON data type. It also introduced a variety of new operators and functions related to JSON data type. In this post, I am also going share one of the important query to convert PostgreSQL tabular data into JSON formatted data.

Generally, we are storing JSON formatted data into PostgreSQL and access base on the different filters.

Here, I am sharing one type of utility script to convert PostgreSQL table data into JSON formatted data. Sometimes it requires to populate JSON formatted data for a web service purpose.

I am doing this using, json_agg and row_to_json function.

json_agg(expression) : aggregates values as a JSON array.
row_to_json(record [, pretty_bool]) : Returns the row as JSON.

Below is a full demonstration of this:

First, create table with sample data:

```
CREATE TABLE tbl_EmployeeDetails
(
	EmpID INTEGER
	,DepartmenetName VARCHAR(50)
	,EmpFirstName VARCHAR(50)
	,EmpLastName VARCHAR(50)
);

INSERT INTO tbl_EmployeeDetails VALUES
(1,'Sales','Anvesh','Patel')
,(2,'Sales','Neevan','Patel')
,(3,'Order','Roy','Loother')
,(4,'Marketing','Martin','Farook')
,(5,'Marketing','Jenny','Pandya')
```

```
,(6,'Marketing','Mahi','Patel');
```

Script to convert data into JSON format without column name for key:

```
SELECT
	DepartmenetName
	,json_agg(row_to_json((EmpFirstName, EmpLastName))) AS JsonData
FROM tbl_EmployeeDetails
GROUP BY DepartmenetName;
```

Script to convert data into JSON format with column name for key:

```
SELECT
	DepartmenetName
	,json_agg(row_to_json
	(
	  (SELECT ColumnName FROM (SELECT EmpFirstName, EmpLastName) AS ColumnName
(EmpFirstName, EmpLastName))
	)) AS JsonData
FROM tbl_EmployeeDetails
GROUP BY DepartmenetName;
```

Using FILTER CLAUSE, multiple COUNT (*) in one SELECT Query for Different Groups

PostgreSQL 9.4 has introduced one of the very good FILTER CLAUSE which is used to apply filters in aggregate functions.

Using FILTER, you can use different types of aggregate functions without applying any GROUP BY CLAUSE.

Now Imagine, that I have one Student table and I want total number of Students based different grades.
What happened without FILTER CLAUSE, We have to perform this calculation in the individual SELECT query.

But with the use of FILTER CLAUSE we can perform aggregation based on different FILTER values in a single SQL Query

Below is a small demonstration of this:

Create one Student table with sample data:

```
CREATE TABLE tbl_Students
(
	StudID INT PRIMARY KEY
	,StudentName CHARACTER VARYING
	,Marks INTEGER
);

INSERT INTO tbl_Students
VALUES (1,'Anvesh',88),(2,'Jenny',55),(3,'Tushar',85)
,(4,'Kavita',75),(5,'Manas',42),(6,'Martin',69)
,(7,'Roy',95),(8,'Benny',92),(9,'Neevan',82)
,(10,'Lee',43),(11,'Loother',65),(12,'Eric',58);
```

Apply FILTER clause to count number of Students based on Marks:

```
SELECT
	COUNT(1) AS TotalStudents
	,COUNT(1) FILTER (WHERE Marks BETWEEN 40 AND 60) AS TotalGrade_C
	,COUNT(1) FILTER (WHERE Marks BETWEEN 60 AND 80) AS TotalGrade_B
	,COUNT(1) FILTER (WHERE Marks BETWEEN 80 AND 100) AS TotalGrade_A
FROM tbl_Students;
```

CREATE DOMAIN to Abstract Data Type and Enforce Business Rules

PostgreSQL has one of the very good data type to store all the optional constraints and business rule of the application.

The DOMAIN is a type of DATA TYPE to abstract the common constraint and other business rule of Database system.

For example, we have several tables with Email column and we require to apply CHECK CONSTRAINT for validation of those Email columns.

For this kind of requirement, we can create one DOMAIN DATA TYPE, Instead of creating different CHECK CONSTRAINT for all the columns. The DOMAIN DATA TYPE is also very easy to manage at one single place, without modifying each individual CHECK CONSTRAINT.

Below is a small demonstration of this:

CREATE one DOMAIN data type to validate simple email expression: Doesn't allow numbers in the domain name and doesn't allow for top level domains that are less than 2 or more than 3 letters.

```
CREATE DOMAIN domain_email AS TEXT
CHECK(
   VALUE ~ '^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$'
```

```
);
```

CREATE a Sample table using domain_email data type:

```
CREATE TABLE tbl_TestEmail
(
	ID INT
	,Email domain_email
);
```

Try to insert invalid email and test the CHECK CONSTRAINT of DOMAIN DATA TYPE:

```
INSERT INTO tbl_TestEmail VALUES (4,'anvesh@dbrnd.co.in');
INSERT INTO tbl_TestEmail VALUES (5,'anvesh_08@08dbrnd.org');
INSERT INTO tbl_TestEmail VALUES (6,'dba@aol.info');
```

Optimized way to get first Record per each GROUP (using DISTINCT ON, LATERAL)

Dont you think, this is a very common requirement for most of the Database Developer.

In PostgreSQL, We can get a first record for each GROUP using different options like:

- Using DISTINCT ON
- Using LATERAL
- CTE with ROW_NUMBER()
- CTE with LATERAL
- Subquery with ROW_NUMBER()
- Using array_agg()

In this demonstration, I am going to give two optimized solutions to get one record per each GROUP.

Create a table with Sample records:

```
CREATE TABLE tbl_Employees
(
	EmpID INT
	,EmpName CHARACTER VARYING
	,EmpDepartment CHARACTER VARYING
	,EmpSalary INT
);

INSERT INTO tbl_Employees
VALUES
(1,'Anvesh','Database',90000)
,(2,'Jenny','JAVA',65000)
,(3,'Martin','PHP',85000)
,(4,'Roy','PHP',94000)
,(5,'Eric','PHP',70000)
,(6,'Rajesh','Animation',50000)
,(7,'Mahi','Database',40000)
,(8,'Sofia','JAVA',60000);
```

First Solution using DISTINCT ON:

As per the Postgres official document,

The DISTINCT ON expression(s) must match the leftmost ORDER BY expression(s).
The ORDER BY clause will normally contain additional expression(s) that determine the desired precedence of rows within each DISTINCT ON group.

```
SELECT DISTINCT ON (EmpDepartment) *
FROM   tbl_Employees
ORDER  BY EmpDepartment
	,EmpSalary DESC;
```

Second Solution using LATERAL:
A LATERAL allows sub-queries to reference columns provided by preceding FROM items.

```
SELECT DISTINCT T.*
FROM   tbl_Employees e
,LATERAL
(
   SELECT *
   FROM   tbl_Employees
   WHERE  EmpDepartment = e.EmpDepartment
   ORDER  BY EmpSalary DESC
   LIMIT  1
) AS T;
```

CREATE PIVOT TABLE to arrange Rows into Columns form

In this post, I am going to demonstrate arrangement of the row data to columns which is called a something like Pivot table in PostgreSQL.

What is a Pivot Table?

Pivot table is one kind of summary and representation of data, like Microsoft Spreadsheets.

Pivot table arranges some of row categories into column and also create count and average of that data for better representation.

Like Microsoft SQL Server, PostgreSQL doesn't provide a feature like Pivot Table, We can achieve using SQL Query.

Create a table with sample data:

```
CREATE TABLE tbl_EmployeePivotTest
(
    EmpName VARCHAR(255)
    ,EmpDeptName VARCHAR(255)
    ,EmpAvgWorkingHours INTEGER
);

INSERT INTO tbl_EmployeePivotTest VALUES
('Anvesh','Computer-IT',226)
,('Anvesh','Computer-IT',100)
,('Anvesh','Account',142)
,('Anvesh','Marketing',110)
,('Anvesh','Finance',236)
,('Anvesh','Account',120)
,('Jeeny','Computer-IT',120)
,('Jeeny','Finance',852)
,('Jeeny','Account',326)
,('Jeeny','Marketing',50)
,('Jeeny','Finance',140);
```

Examine this data where two employees with working hour in a different department.
In this data, employee worked more than one time in some of the

department.
Now requirement is, to populate different column for different department with total of working hours.

SQL Query to PIVOT Table (Using GroupBy Clause):

```
SELECT
     EmpName
    ,SUM(Computer_IT) AS Total_IT
    ,SUM(Account) AS Total_Account
    ,SUM(Marketing) AS Total_Marketing
    ,SUM(Finance) AS Total_Finance
FROM
(
    SELECT
       EmpName
       ,CASE WHEN EmpDeptName = 'Computer-IT'
              THEN EmpAvgWorkingHours END AS Computer_IT
       ,CASE WHEN EmpDeptName = 'Account'
              THEN EmpAvgWorkingHours END AS Account
       ,CASE WHEN EmpDeptName = 'Marketing'
              THEN EmpAvgWorkingHours END AS Marketing
       ,CASE WHEN EmpDeptName = 'Finance'
              THEN EmpAvgWorkingHours END AS Finance
    FROM tbl_EmployeePivotTest
) AS T
GROUP BY EmpName;
```

The Result:

```
1 empname | total_it | total_account | total_marketing | total_finance
2 ---------+----------+---------------+-----------------+---------------
3  Anvesh  |     326 |         262 |           110 |         236
4  Jeeny   |     120 |         326 |            50 |         992
```

In the above result,

You can see the result of two employees with total working hour by each department.

The first step is inner query; in which we have selected employee working hour base on department category.

The Second step is to select this data in the outer query and apply group by clause for SUM ().

```
SELECT
     count(Shipping) AS Total_Shipping
    ,count(Sales) AS Total_Sales
    ,count(Purchasing) AS Total_Purchasing
    ,count(Finance) AS Total_Finance
FROM
(
 SELECT
       e.first_name
       ,CASE WHEN department_id = 50
             THEN first_name END AS Shipping
       ,CASE WHEN department_id = 80
             THEN first_name END AS Sales
       ,CASE WHEN department_id = 30
             THEN first_name END AS Purchasing
       ,CASE WHEN department_id = 100
             THEN first_name END AS Finance
    FROM employees e
   ) AS T;
```

Input Table : Maxrows

Name	Amount1	Amount2	Amount3
Vishal	5000	6800	4300
Rahul	3500	1000	2200
Simran	9800	9999	9990
Sukarn	5600	7757	8897
Vijay	6647	9898	10000

Output:

Name	MaxAmt
Vishal	6800
Rahul	3500
Simran	9999
Sukarn	8897
Vijay	10000

```
create table employee2(id serial, name varchar, amount1 dec(8, 2), amount2 dec(8,2),
amount3 dec(8,2));

insert into employee2(name, amount1, amount2, amount3) values
('Vishal', 5000, 6800, 6300),
('Rahul', 3500, 1000, 2200),
```

('Simran', 9800, 9999, 9990),
('Sukarn', 5600, 7757, 8897),
('Vijay', 6647, 9898, 10000);

select name, Greatest(amount1, amount2, amount3) as Maxamount
from employee2;

Input Table: Employee

Empid	Name	Gender	Department
1	Alexa	Female	IT
2	Naman	Male	Finance
3	Rita	Female	Finance
4	Priya	Female	HR
5	Shivangi	Female	HR
6	Rahul	Male	IT
7	Tanya	Female	Finance
8	Arun	Male	HR
9	Lakshya	Male	IT
10	Deepika	Female	HR

Output :

Department	TotalMale	TotalFemale
Finance	1	2
HR	1	3
IT	2	1

```
select department,
count(gender) filter (where gender='male') as TotalMale ,
count(gender) filter (where gender='Female') as TotalFemale
from employee3
group by department;
```

```
select t.department, count(t.male)as TotalMale,count(t.female)
from (
select department,case when gender='male' then 1 end as male,
case when gender='Female' then 1 end as female
from employee3) t
group by t.department;
```

or

```
select department, count( case when gender='male' then 5 end) as malecount
,count(case when gender='Female' then 2 end) as femalecount
from employee3
```