Summary:

Welcome to this project-based course SQL Window Functions for Analytics. This is a hands-on project that will help SQL users use window functions extensively for database insights.
In this project, you will learn how to explore and query the project-db database extensively. We will start this hands-on project by retrieving the data in the table in the database. By the end of this 2-hour-and-a-half-long project, you will be able to use different window functions to retrieve the desired result from a database.

In this project, you will learn how to use SQL window functions like ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE(), and LAST_VALUE() to manipulate data in the project-db database.

Also, we will consider how to use aggregate window functions.
These window functions will be used together with the OVER() clause to query this database. By extension, we will use grouping functions like GROUPING SETS(), ROLLUP(), and CUBE() to retrieve sublevel and grand totals.

In this project, we will move systematically by first introducing the functions using a simple example. Then, we will write slightly complex queries using the window functions in real-life applications.

Also, for this hands-on project, we will use PostgreSQL as our preferred database management system (DBMS). Therefore, to complete this project, it is required that you have prior experience with using PostgreSQL.

Similarly, this project is an advanced SQL concept; so, a good foundation in writing SQL queries is vital to complete this project.

I recommend that you complete the project titled: “Introduction to SQL Window Functions” before you take this current project.

The introductory project to SQL Window Functions will provide every necessary foundation to complete this current project. However, if you are comfortable writing queries in PostgreSQL, please join me on this wonderful ride! Let’s get our hands dirty!

Learning Objectives

- Understand how SQL window functions work and how to use them
- Manipulate data in tables using SQL SELECT statements together with window functions
- Use different SQL windows to get database insights

1. Project Overview

Project Structure

The hands-on project on **SQL Window Functions for Analytics** is divided into the following tasks:

Task 1: Getting Started

- Overview of the project
- A brief introduction to the Rhyme platform
- What are window functions in SQL?
- Retrieve all the data in the projectdb database

Task 2: Window Functions - Refresher

- Retrieve a list of employee_id, first_name, hire_date, and department of all employees ordered by the hire date.
- Retrieve the employee_id, first_name, hire_date of employees for different departments.

Task 3: Ranking

- Recall the use of ROW_NUMBER()
- Let's use the RANK() function
- **Exercise 3.1:** Retrieve the hire_date. Return details of employees hired on or before 31st Dec, 2005 and are in First Aid, Movies and Computers departments

- Return the fifth-ranked salary for each department
- Create a common table expression to retrieve the customer_id, and how many times the customer has purchased from the mall

- Understand the difference between ROW_NUMBER(), RANK(), and DENSE_RANK()

Task 4: Paging: NTILE()

- Group the employees table into five groups based on the order of their salaries
- Group the employees table into five groups for each department based on the order of their salaries
- Create a CTE that returns details of an employee and groups the employees into five groups based on the order of their salaries
- Find the average salary for each group of employees

Task 5: Aggregate Window Functions - Part One

- Retrieve how many employees are in each department
- Retrieve the first names, departments, and number of employees working in that department
- Total Salary for all employees
- Total Salary for each department
- **Exercise 5.1:** Total Salary for each department and order by the hire date. Call the new column running_total

Task 6: Aggregate Window Functions - Part Two

- Retrieve the different region ids
- Retrieve the first names, departments, and number of employees working in that department and region
- **Exercise 6.1:** Retrieve the first names, departments, and number of employees working in that department and in region 2
- Create a common table expression to retrieve the customer_id, ship_mode, and how many times the customer has purchased from the mall
- **Exercise 6.2:** Calculate the cumulative sum of customers purchases for the different ship mode

Task 7: Window Frames - Part One

- Calculate the running total of salary
- Add the current row and previous row
- Find the running average

Task 8: Window Frames - Part Two

- Review of the FIRST_VALUE() function
- Retrieve the last department in the departments table

- Create a common table expression to retrieve the customer_id, ship_mode, and how many times the customer has purchased from the mall.

Task 9: GROUPING SETS(), ROLLUP(), & CUBE()

- Find the sum of the quantity for different ship modes
- Find the sum of the quantity for different categories
- Find the sum of the quantity for different subcategories
- Use the GROUPING SETS() clause
- Use the ROLLUP() clause
- Use the CUBE() clause
- Wrap up the project

About Rhyme

This project runs on Coursera's hands-on platform called Rhyme. On Rhyme, you do projects in a hands-on manner in your browser. You will get instant access to pre-configured cloud desktops that have all the software and data you will need. So, you can just focus on learning. For this project, this means instant access to a cloud desktop with PostgreSQL pre-installed. If you need help troubleshooting Rhyme, please refer to the Coursera Help Center for more information.

Earn a Certificate

After you have completed the **SQL Window Functions for Analytics** hands-on project, you will be able to assess your knowledge using an ungraded assignment. Once you are comfortable with the concepts, take the final quiz, score higher than 80% to earn your certificate.

2. SQL files and data sets for the project

Note that your cloud desktop is already set up with all you need to complete this hands-on project. However, I have uploaded the CSV files and the SQL file we used for the project to help you practice on your own.
Download the **SQL Window Functions for Analytics** file here
Download the **project-db database** file here
Download the **Customers.csv** file here
Download the **Sales.csv** file here
I strongly recommend that you should install PostgreSQL on your local machine. Then, use these files to practice what you will learn in this project.

Good luck with practicing!

3. Links to PostgreSQL Window Functions Documentation

You can read and explore the **PostgreSQL documentation on window functions** below:

- Read about window functions here and here
- More details about window functions can be found in Section 4.2.8, Section 9.19, Section 7.2.4, and the SELECT reference page.

4. SQL Window Functions for Analytics

About Guided Project

Now we will query the project-db database **using SQL SELECT together with SQL window functions to get database insights**. We will accomplish this by completing each task in the project:

- Retrieve data from tables in the project-db database
- Use the ROW_NUMBER() function
- Work with ranking functions
- Use the NTILE() function to group result sets
- Work extensively with aggregate window functions
- Use aggregate functions with window frames
- Work with grouping functions

While you are watching me work on each step, you will get a cloud desktop with all the required software pre-installed. This will allow you to follow along with the instructions to complete the above-mentioned tasks. **If you notice that I am too fast, you can pause my video so as to follow along.** After all, we learn best with active, hands-on learning
Please note that you will have 3 opportunities to use the Rhyme platform for this project. So, if you couldn't finish the project on the first go, you can try again. Please refer to the Coursera Help Center for more information.

Ready to get started?

Check the **I agree to use this tool responsibly** box and click the Open Tool button to access your cloud workspace.

This course uses a third-party app, SQL Window Functions for Analytics, to enhance your learning experience. The app will reference basic information like your name, email, and Coursera ID.

5. Guided Project

Task 1: Getting Started:

- Overview of the project:

  This project uses the superstore dataset. There are 5 tables.
  Customers
  Departments
  Employees
  Regions
  Sales

  We are using PostgreSQL as our RDBMS.

- What are window functions in SQL?

  A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.

- Retrieve all the data in the projectdb database

```
SELECT * FROM employees;
SELECT * FROM departments;
SELECT * FROM regions;
SELECT * FROM customers;
SELECT * FROM sales;
```

Task 2: Window Functions - Refresher

- Retrieve a list of employee_id, first_name, hire_date, and department of all employees ordered by the hire date.

```
SELECT employee_id, first_name, department, hire_date,

ROW_NUMBER() OVER (ORDER BY hire_date) AS Row_N
```

```
FROM employees;
```

Data Output | Explain | Messages | Notifications

	employee_id [PK] integer	first_name character varying (50)	department character varying (17)	hire_date date	row_n bigint
1	271	Norbie	First Aid	2003-01-01	1
2	300	Cassandra	Beauty	2003-01-01	2
3	79	Rora	Children Clothing	2003-01-12	3
4	5	Feliks	Computers	2003-01-14	4
5	739	Cecilius	Vitamins	2003-01-20	5
6	488	Eugenius	Toys	2003-01-26	6
7	75	Fiorenze	Phones & Tablets	2003-02-17	7

- Retrieve the employee_id, first_name, hire_date of employees for different departments.

```
SELECT employee_id, first_name, department, hire_date,

ROW_NUMBER() OVER (PARTITION BY department ORDER BY hire_date) AS
Row_N

FROM employees;
```

Data Output | Explain | Messages | Notifications

	employee_id [PK] integer	first_name character varying (50)	department character varying (17)	hire_date date	row_n bigint
1	927	Maryellen	Automotive	2003-04-19	1
2	840	Archibold	Automotive	2003-04-26	2
3	988	Tabb	Automotive	2003-05-02	3
4	648	Abbott	Automotive	2003-06-05	4
5	305	Ladonna	Automotive	2003-08-10	5
6	126	Roslyn	Automotive	2003-08-11	6
7	274	Lorelle	Automotive	2004-01-27	7

Task 3: Ranking

- Recall the use of ROW_NUMBER()

```
SELECT first_name, email, department, salary,
```

```
        ROW_NUMBER() OVER(PARTITION BY department
        ORDER BY salary DESC)
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	row_number bigint
1	Mill	Automotive	162522	1
2	Irita	Automotive	160783	2
3	Tammie	Automotive	160039	3
4	Roslyn	Automotive	157260	4
5	Betsey	Automotive	152141	5
6	Cherianne	Automotive	150821	6
7	Chrissy	Automotive	146522	7

- Let's use the RANK() function

```
SELECT first_name, department, salary,
    RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS
    Rank_N
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	rank_n bigint
1	Mill	Automotive	162522	1
2	Irita	Automotive	160783	2
3	Tammie	Automotive	160039	3
4	Roslyn	Automotive	157260	4
5	Betsey	Automotive	152141	5
6	Cherianne	Automotive	150821	6
7	Chrissy	Automotive	146522	7

Note: Rows having the same value are assigned the same rank. For the next rank after two same rank values, one rank value will be skipped.

- **Exercise 3.1:** Retrieve the hire_date. Return details of employees hired on or before 31st Dec, 2005 and are in First Aid, Movies and Computers departments

```
SELECT first_name,department, salary, hire_date,
        RANK() OVER(PARTITION BY department
                ORDER BY salary DESC)
FROM employees
WHERE hire_date <= '2005-12-31' AND
      department IN('First Aid', 'Movies','Computers');
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	hire_date date	rank bigint
1	Parnell	Computers	155000	2004-03-29	1
2	Webb	Computers	152504	2005-10-13	2
3	Javier	Computers	87199	2003-11-12	3
4	Retha	Computers	74771	2004-06-29	4
5	Feliks	Computers	55307	2003-01-14	5
6	Kimberley	Computers	49788	2004-01-08	6
7	Levi	Computers	35076	2003-10-21	7

- Return the fifth-ranked salary for each department

```
SELECT * FROM (
                SELECT first_name, department, salary,
                ROW_NUMBER() OVER(PARTITION BY department
                ORDER BY salary DESC) AS Rank_N
                FROM employees) a
WHERE Rank_N=5;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	rank bigint
1	Betsey	Automotive	152141	5
2	Garald	Beauty	145225	5
3	Christine	Books	149864	5
4	Kevin	Camping	154856	5
5	Nicolis	Children Clothing	137567	5
6	Hester	Clothing	150887	5
7	Laryssa	Computers	152831	5

- Create a common table expression to retrieve the customer_id, and how many times the customer has purchased from the mall

```
WITH purchase_count AS (
SELECT customer_id, COUNT(sales) AS purchase
FROM sales
GROUP BY customer_id
ORDER BY purchase DESC
)
SELECT *
FROM purchase_count;
```

- Understand the difference between ROW_NUMBER(), RANK(), and DENSE_RANK()

Ranking functions are, RANK(), DENSE_RANK(), ROW_NUMBER()

RANK() –

As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having the same value are assigned the same rank. For the next rank after two same rank values, one rank value will be skipped.

DENSE_RANK() –

It assigns rank to each row within the partition. Just like the rank function, the first row is assigned rank 1 and rows having the same value have the same rank. The difference between RANK() and DENSE_RANK() is that in DENSE_RANK(), for the next rank after two of the same rank, consecutive integers are used, no rank is skipped.

ROW_NUMBER() –

It assigns consecutive integers to all the rows within the partition. Within a partition, no two rows can have the same row number.

Note –

ORDER BY() should be specified compulsorily while using rank window functions.

Example:

```
WITH purchase_count AS (
                        SELECT customer_id, COUNT(sales) AS purchase
                        FROM sales
                        GROUP BY customer_id
                        ORDER BY purchase DESC)

SELECT customer_id, purchase,
            ROW_NUMBER() OVER (ORDER BY purchase DESC) AS Row_N,
            RANK() OVER (ORDER BY purchase DESC) AS Rank_N,
            DENSE_RANK() OVER (ORDER BY purchase DESC) AS
            Dense_Rank_N
FROM purchase_count
ORDER BY purchase DESC;
```

Data Output | Explain | Messages | Notifications

	customer_id character (8)	purchase bigint	row_n bigint	rank_n bigint	dense_rank_n bigint
1	WB-21850	37	1	1	1
2	PP-18955	34	2	2	2
3	JL-15835	34	3	2	2
4	MA-17560	34	4	2	2
5	EH-13765	32	5	5	3
6	JD-15895	32	6	5	3
7	CK-12205	32	7	5	3

ROW_: NUMBER: Give a sequential number without skipping any number.

RANK: Sequential number but will assign same number for duplicate and skip the count for the next one.

DENSE_RANK: same as RANK but without skipping numbers for the next.

Task 4: Paging: NTILE() :Creates evenly **tiles** or **groups.**

NTILE() function is a window function that distributes rows of an ordered partition into a pre-defined number of roughly equal groups.
It assigns each group a number_expression ranging from 1. NTILE() function assigns a number_expression for every row in a group, to which the row belongs.

Syntax :

```
NTILE(number_expression) OVER (
  [PARTITION BY partition_expression ]
  ORDER BY sort_expression [ASC | DESC]
)
```

Parameters of syntax in detail :

Number_expression: The number_expression is the integer into which the rows are divided.
PARTITION BY clause: The PARTITION BY is optional, it differs the rows of a result set into partitions where the NTILE() function is used.
ORDER BY clause: The ORDER BY clause defines the order of rows in each partition where the NTILE() is used.

Note:
When a number of rows aren't divisible by the number_expression, the **NTILE()** function results in groups of two sizes with a difference by one.
The larger groups always come ahead of the smaller group within the order specified by the ORDER BY within the OVER() clause.
Also, when all of the rows are divisible by the number_expression, the function divides evenly the rows among number_expression.

- Group the employees table into five groups based on the order of their salaries

```
SELECT first_name, department, salary,
        NTILE(5)OVER(ORDER BY salary ASC )
FROM employees;
```

Group 1 …. Group 5.

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	ntile integer
1	Renault	Pharmacy	20542	1
2	Addia	Grocery	20613	1
3	Roger	Device Repair	20664	1
4	Marylin	Music	20776	1
5	Roarke	Device Repair	21023	1
6	Lenka	Decor	21024	1
7	Lucius	Grocery	21120	1

- - - - - - - - -

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	ntile integer
994	Ev	Grocery	164582	5
995	Barby	Clothing	164588	5
996	Lucy	Sports	165660	5
997	Lauren	Pharmacy	166016	5
998	Riley	Camping	166569	5
999	Carissa	Music	166765	5
1000	Jacklyn	Clothing	166976	5

- Group the employees table into five groups for each department based on the order of their salaries

```
SELECT first_name,department, salary,
    NTILE(5) OVER(PARTITION BY department ORDER BY salary DESC)
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	ntile integer
1	Mill	Automotive	162522	1
2	Irita	Automotive	160783	1
3	Tammie	Automotive	160039	1
4	Roslyn	Automotive	157260	1
5	Betsey	Automotive	152141	1
6	Cherianne	Automotive	150821	1
7	Chrissy	Automotive	146522	1

- - - - - - - - -

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	ntile integer
12	Charis	Automotive	130995	2
13	Stormy	Automotive	126983	2
14	Noelyn	Automotive	125305	2
15	Cybil	Automotive	123828	3
16	Lorelle	Automotive	119959	3
17	Maryellen	Automotive	115973	3
18	Ladonna	Automotive	111775	3

Here, each department is divided into 5 groups and ordered by salary ASC.

- Create a CTE that returns details of an employee and groups the employees into five groups based on the order of their salaries:

```
WITH salary_ranks AS (
```

```
                SELECT first_name,department, salary,
                NTILE(5) OVER(ORDER BY salary DESC) AS rank_of_salary
FROM employees)
SELECT * FROM salary_ranks;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	rank_of_salary integer
1	Jacklyn	Clothing	166976	1
2	Carissa	Music	166765	1
3	Riley	Camping	166569	1
4	Lauren	Pharmacy	166016	1
5	Lucy	Sports	165660	1
6	Barby	Clothing	164588	1
7	Ev	Grocery	164582	1

- - - - - - - - - -

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	rank_of_salary integer
292	Rhianna	Beauty	120753	2
293	Jaymee	Movies	120395	2
294	Johnnie	Beauty	120362	2
295	Roma	Tools	120070	2
296	Lorelle	Automotive	119959	2
297	Terry	Beauty	119848	2
298	Brooks	Cosmetics	119774	2

- Find the average salary for each group of employees:

```
WITH salary_rank AS (
                SELECT first_name, salary,department,
                NTILE(5) OVER(ORDER BY salary DESC)AS rank_salary
FROM employees
```

```
)
SELECT rank_salary, ROUND(AVG(salary))
FROM salary_rank
GROUP BY rank_salary
ORDER BY rank_salary;
```

Data Output | Explain | M

	rank_salary integer	round numeric
1	1	151648
2	2	119197
3	3	90136
4	4	62086
5	5	34791

Task 5: Aggregate Window Functions - Part One

- Retrieve how many employees are in each department

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

Data Output | Explain | Messages | N

	department character varying (17)	dept_count bigint
1	Automotive	32
2	Beauty	45
3	Books	37
4	Camping	36
5	Children Clothing	47
6	Clothing	49
7	Computers	47

- Retrieve the first names, departments, and number of employees working in that department

```
SELECT first_name, department,
COUNT(*)
OVER(PARTITION BY department ORDER BY department ASC)
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	count bigint
1	Lorelle	Automotive	32
2	Sterling	Automotive	32
3	Roslyn	Automotive	32
4	Abbott	Automotive	32
5	Cybil	Automotive	32
6	Mill	Automotive	32
7	Maryellen	Automotive	32

- - - - - - - -

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	count bigint
994	Gran	Vitamins	37
995	Daren	Vitamins	37
996	Norri	Vitamins	37
997	Mendie	Vitamins	37
998	Odessa	Vitamins	37
999	Elvera	Vitamins	37
1000	Buckie	Vitamins	37

- Total Salary for all employees

```
SELECT first_name, department, salary,
SUM(salary) OVER(ORDER BY department) AS total_salary
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	total_salary bigint
1	Lorelle	Automotive	119959	3553477
2	Sterling	Automotive	56095	3553477
3	Roslyn	Automotive	157260	3553477
4	Abbott	Automotive	106517	3553477
5	Cybil	Automotive	123828	3553477
6	Mill	Automotive	162522	3553477
7	Maryellen	Automotive	115973	3553477

- - - - - - - - - -

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	total_salary bigint
994	Gran	Vitamins	43666	91571594
995	Daren	Vitamins	67549	91571594
996	Norri	Vitamins	158847	91571594
997	Mendie	Vitamins	141707	91571594
998	Odessa	Vitamins	123268	91571594
999	Elvera	Vitamins	163933	91571594
1000	Buckie	Vitamins	30289	91571594

- Total Salary for each department

```
SELECT first_name, department, salary,
SUM(salary) OVER(PARTITION  BY department) AS total_salary
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	total_salary bigint
1	Lorelle	Automotive	119959	3553477
2	Sterling	Automotive	56095	3553477
3	Roslyn	Automotive	157260	3553477
4	Abbott	Automotive	106517	3553477
5	Cybil	Automotive	123828	3553477
6	Mill	Automotive	162522	3553477
7	Maryellen	Automotive	115973	3553477

- - - - - - - - - -

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	salary integer	total_salary bigint
994	Gran	Vitamins	43666	3343246
995	Daren	Vitamins	67549	3343246
996	Norri	Vitamins	158847	3343246
997	Mendie	Vitamins	141707	3343246
998	Odessa	Vitamins	123268	3343246
999	Elvera	Vitamins	163933	3343246
1000	Buckie	Vitamins	30289	3343246

- **Exercise 5.1:** Total Salary for each department and order by the hire date. Call the new column running_total

```
SELECT first_name, hire_date, department, salary,
        SUM(salary) OVER(PARTITION BY  department
        ORDER BY hire_date) AS running_total
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	hire_date date	department character varying (17)	salary integer	running_total bigint
1	Maryellen	2003-04-19	Automotive	115973	115973
2	Archibold	2003-04-26	Automotive	69379	185352
3	Tabb	2003-05-02	Automotive	47591	232943
4	Abbott	2003-06-05	Automotive	106517	339460
5	Ladonna	2003-08-10	Automotive	111775	451235
6	Roslyn	2003-08-11	Automotive	157260	608495
7	Lorelle	2004-01-27	Automotive	119959	728454

This will return the running total by each department.

Task 6: Aggregate Window Functions - Part Two

- Retrieve the different region ids

```
SELECT DISTINCT region_id
FROM employees;
```

Data Output

	region_id integer
1	7
2	1
3	5
4	2
5	4
6	6
7	3

- Retrieve the first names, departments, and number of employees working in that department and region.

```
SELECT first_name, department,
        COUNT(*) OVER(PARTITION BY department) AS dep_count,
        region_id,COUNT(*) OVER(PARTITION BY region_id)AS region_count
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	dep_count bigint	region_id integer	region_count bigint
1	Ladonna	Automotive	32	2	141
2	Clementina	Automotive	32	1	152
3	Cybil	Automotive	32	1	152
4	Merlina	Automotive	32	1	152
5	Jessalyn	Automotive	32	7	152
6	Abbott	Automotive	32	7	152
7	Irita	Automotive	32	1	152

- **Exercise 6.1:** Retrieve the first names, departments, and number of employees working in that department and in region 2.

```
SELECT first_name, department,
COUNT(*) OVER(PARTITION BY department) AS dept_count
FROM employees
WHERE region_id = 2;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	department character varying (17)	dept_count bigint
1	Archibold	Automotive	4
2	Betsey	Automotive	4
3	Ladonna	Automotive	4
4	Poppy	Automotive	4
5	Lauree	Beauty	12
6	Mill	Beauty	12
7	Claribel	Beauty	12

- Create a common table expression to retrieve the customer_id, ship_mode, and how many times the customer has purchased from the mall.

```
WITH purchase_count AS (
SELECT customer_id, ship_mode, COUNT(sales) AS purchase
FROM sales
GROUP BY customer_id, ship_mode
ORDER BY purchase DESC
)
SELECT * FROM purchase_count;
```

Data Output | Explain | Messages | Notifications

	customer_id character (8)	ship_mode character varying (255)	purchase bigint
1	ZC-21910	Standard Class	26
2	EH-13765	Standard Class	25
3	EP-13915	Standard Class	24
4	SH-19975	Standard Class	23
5	XP-21865	Standard Class	23
6	GT-14710	Standard Class	22
7	SP-20860	Standard Class	22

- **Exercise 6.2:** Calculate the cumulative sum of customers purchases for the different ship mode

```
WITH purchase_count AS (
SELECT customer_id, ship_mode, COUNT(sales) AS purchase
FROM sales
GROUP BY customer_id, ship_mode
ORDER BY purchase DESC
)
SELECT customer_id, ship_mode, purchase,
SUM(purchase) OVER(PARTITION BY ship_mode
                   ORDER BY customer_id ASC) AS sum_of_sales
FROM purchase_count;
```

Data Output | Explain | Messages | Notifications

	customer_id character (8)	ship_mode character varying (255)	purchase bigint	sum_of_sales numeric
1	AA-10315	First Class	1	1
2	AA-10375	First Class	4	5
3	AA-10645	First Class	7	12
4	AB-10015	First Class	5	17
5	AB-10060	First Class	8	25
6	AB-10105	First Class	4	29
7	AB-10165	First Class	3	32

Task 7: Window Frames - Part One

- Calculate the running total of salary

```
SELECT first_name,hire_date, salary,
SUM(salary) OVER(ORDER BY hire_date) AS running_total
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	hire_date date	salary integer	running_total bigint
1	Norbie	2003-01-01	82215	189151
2	Cassandra	2003-01-01	106936	189151
3	Rora	2003-01-12	153489	342640
4	Feliks	2003-01-14	55307	397947
5	Cecilius	2003-01-20	98882	496829
6	Eugenius	2003-01-26	152118	648947
7	Fiorenze	2003-02-17	51266	700213

- Add the current row and previous row

```
SELECT first_name,hire_date, salary,
                SUM(salary) OVER(ORDER BY hire_date
                ROWS BETWEEN 1 PRECEDING
                AND CURRENT ROW) AS running_total
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	hire_date date	salary integer	running_total bigint
1	Norbie	2003-01-01	82215	82215
2	Cassandra	2003-01-01	106936	189151
3	Rora	2003-01-12	153489	260425
4	Feliks	2003-01-14	55307	208796
5	Cecilius	2003-01-20	98882	154189
6	Eugenius	2003-01-26	152118	251000
7	Fiorenze	2003-02-17	51266	203384

- Find the running average

```
SELECT first_name,hire_date, salary,
AVG(salary) OVER(ORDER BY hire_date
                 ROWS BETWEEN 1 PRECEDING
                 AND CURRENT ROW) AS running_total
FROM employees;
```

Data Output | Explain | Messages | Notifications

	first_name character varying (50)	hire_date date	salary integer	running_total numeric
1	Norbie	2003-01-01	82215	82215.000000000000
2	Cassandra	2003-01-01	106936	94575.500000000000
3	Rora	2003-01-12	153489	130212.500000000000
4	Feliks	2003-01-14	55307	104398.000000000000
5	Cecilius	2003-01-20	98882	77094.500000000000
6	Eugenius	2003-01-26	152118	125500.000000000000
7	Fiorenze	2003-02-17	51266	101692.000000000000

Task 8: Window Frames - Part Two

- Review of the FIRST_VALUE() function

```
SELECT department, division,
FIRST_VALUE(department) OVER(ORDER BY department ASC)
first_department
FROM departments;
```

Data Output | Explain | Messages | Notifications

	department [PK] character varying (100)	division character varying (100)	first_department character varying
1	Automotive	Hardware	Automotive
2	Beauty	Fashion	Automotive
3	Books	Entertainment	Automotive
4	Camping & Fishing	Outdoors	Automotive
5	Children Clothing	Kids	Automotive
6	Clothing	Home	Automotive
7	Computers	Electronics	Automotive

- Retrieve the first and last department in the departments table.

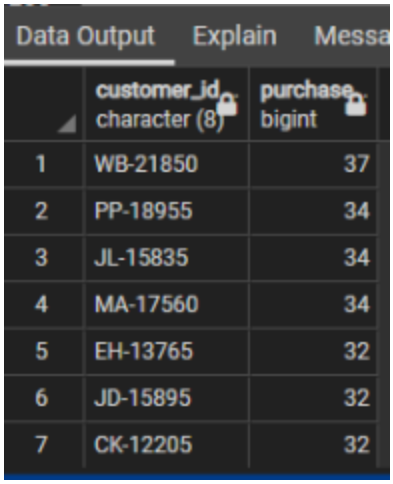
```
SELECT department, division,
    FIRST_VALUE(department) OVER(ORDER BY department ASC) first_department,
    LAST_VALUE(department) OVER(ORDER BY department ASC
    RANGE BETWEEN UNBOUNDED PRECEDING
    AND UNBOUNDED FOLLOWING) last_department
FROM departments;
```

Data Output | Explain | Messages | Notifications

	department [PK] character varying (100)	division character varying (100)	first_department character varying	last_department character varying
1	Automotive	Hardware	Automotive	Vitamins
2	Beauty	Fashion	Automotive	Vitamins
3	Books	Entertainment	Automotive	Vitamins
4	Camping & Fishing	Outdoors	Automotive	Vitamins
5	Children Clothing	Kids	Automotive	Vitamins
6	Clothing	Home	Automotive	Vitamins
7	Computers	Electronics	Automotive	Vitamins

- Create a common table expression to retrieve the customer_id, ship_mode, and how many times the customer has purchased from the mall.

```
WITH purchase_count AS (
SELECT customer_id, COUNT(sales) AS purchase
FROM sales
GROUP BY customer_id
ORDER BY purchase DESC
)
SELECT * FROM purchase_count;
```

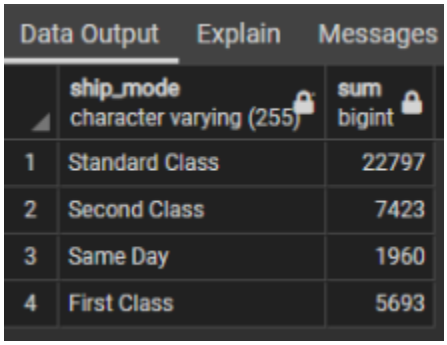


	customer_id character (8)	purchase bigint
1	WB-21850	37
2	PP-18955	34
3	JL-15835	34
4	MA-17560	34
5	EH-13765	32
6	JD-15895	32
7	CK-12205	32

Task 9: GROUPING SETS(), ROLLUP(), & CUBE()

- Find the sum of the quantity for different ship modes

```
SELECT ship_mode, SUM(quantity)
FROM sales
GROUP BY ship_mode;
```



	ship_mode character varying (255)	sum bigint
1	Standard Class	22797
2	Second Class	7423
3	Same Day	1960
4	First Class	5693

- Find the sum of the quantity for different categories

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

Data Output | Explain | Messages

	category character varying (255)	sum bigint
1	Furniture	8028
2	Office Supplies	22906
3	Technology	6939

- Find the sum of the quantity for different subcategories

```
SELECT sub_category, SUM(quantity)
FROM sales
GROUP BY sub_category;
```

Data Output | Explain | Messages

	sub_category character varying (255)	sum bigint
1	Tables	1241
2	Art	3000
3	Storage	3158
4	Bookcases	868
5	Fasteners	914
6	Envelopes	906
7	Appliances	1729

- Use the GROUPING SETS() clause instead of GROUP BY.

```
SELECT ship_mode, category, sub_category,SUM(quantity)
FROM sales
GROUP BY GROUPING SETS(ship_mode,category,sub_category);
```

Data Output | Explain | Messages | Notifications

	ship_mode character varying (255)	category character varying (255)	sub_category character varying (255)	sum bigint
1	Standard Class	[null]	[null]	22797
2	Second Class	[null]	[null]	7423
3	Same Day	[null]	[null]	1960
4	First Class	[null]	[null]	5693
5	[null]	Furniture	[null]	8028
6	[null]	Office Supplies	[null]	22906
7	[null]	Technology	[null]	6939

- Aggregation using GROUPING SETS with Grand Total:

  SELECT ship_mode, category, sub_category,SUM(quantity)

  FROM sales

  GROUP BY GROUPING SETS(ship_mode,category,sub_category,**()**);

Data Output | Explain | Messages | Notifications

	ship_mode character varying (255)	category character varying (255)	sub_category character varying (255)	sum bigint
1	[null]	[null]	[null]	37873
2	Standard Class	[null]	[null]	22797
3	Second Class	[null]	[null]	7423
4	Same Day	[null]	[null]	1960
5	First Class	[null]	[null]	5693
6	[null]	Furniture	[null]	8028
7	[null]	Office Supplies	[null]	22906

- Use the ROLLUP() clause:

  ROLLUP enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions. It also calculates a grand total.

```
SELECT ship_mode, category, sub_category,SUM(quantity)
FROM sales
GROUP BY ROLLUP (ship_mode,category,sub_category);
```

Data Output | Explain | Messages | Notifications

	ship_mode character varying (255)	category character varying (255)	sub_category character varying (255)	sum bigint
1	[null]	[null]	[null]	37873
2	Standard Class	Furniture	Chairs	1372
3	Second Class	Office Supplies	Appliances	324
4	Standard Class	Office Supplies	Labels	830
5	Same Day	Office Supplies	Envelopes	58
6	First Class	Technology	Copiers	45
7	Standard Class	Furniture	Furnishings	2186

- Use the CUBE() clause:

  The CUBE operations let you group by different subsets of a specified set of columns in a SELECT clause. CUBE(column_list) groups by all subsets of the columns specified in column_list . For example, CUBE(a, b) groups by (a) , (b) , (a, b) and () (which represents the entire input set of rows).

```
SELECT ship_mode, category, sub_category,SUM(quantity)
FROM sales
GROUP BY CUBE (ship_mode,category,sub_category);
```

Data Output | Explain | Messages | Notifications

	ship_mode character varying (255)	category character varying (255)	sub_category character varying (255)	sum bigint
1	[null]	[null]	[null]	37873
2	Standard Class	Furniture	Chairs	1372
3	Second Class	Office Supplies	Appliances	324
4	Standard Class	Office Supplies	Labels	830
5	Same Day	Office Supplies	Envelopes	58
6	First Class	Technology	Copiers	45
7	Standard Class	Furniture	Furnishings	2186

- Wrap up the project

6. SQL Window Functions for Analytics
7. Quiz