Introduction to SQL Window Functions

Project overview

Welcome to this project-based course Introduction to SQL Window Functions. This is a hands-on project that introduces SQL users to the world of window functions.

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(), LEAD(), LAG(), and FIRST_VALUE() to manipulate data in the project-db database.

These window functions will be used together with the OVER() clause to query this database.

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

If you are not familiar with writing queries in SQL and want to learn these concepts, start with my previous guided projects titled “Querying Databases using SQL SELECT statement," and “Performing Data Aggregation using SQL Aggregate Functions.”

I taught these guided projects using PostgreSQL. So, taking these projects will give the needed requisite to complete this SQL window functions 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 when to use them
- Understand how the ORDER BY and PARTITION BY clauses work with the OVER() clause
- Manipulate data in tables using SQL SELECT statements together with window functions

- **Project Overview**

Welcome to Your Guided Project!

Welcome to **Introduction to SQL Window Functions**. This is an intermediate project-based course that should take approximately 2 hours 30 minutes to finish. Before diving into the project, please take a look at the course objectives and structure:

Guided Project Objectives

In this course, we are going to focus on **three** learning objectives:

1. Understand how SQL window functions work and when to use them.
2. Understand how the ORDER BY and PARTITION BY clauses work with the OVER() clause.
3. Manipulate data in tables using SQL SELECT statements together with window functions.

By the end of this course, you will be able to use different SQL window functions to perform data aggregation, reporting, manipulation, and querying for database insights using the projectdb database.

Guided Project Structure

This course is divided into **six (6)** parts:

1. Course Overview: This introductory reading material.
2. Reading: SQL files and data sets for the project
3. Additional Reading: PostgreSQL Window Functions Documentation

4. **Introduction to SQL Window Functions:** This is the hands-on project that we will work on together in Rhyme
5. Graded Quiz: This is the final assignment that you need to pass in order to finish the project successfully.
6. **Course End (Learner) Survey:** Tell us what you thought about this guided project!

Project Structure

The hands-on project on **Introduction to SQL Window Functions** 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: ROW_NUMBER() - Part One

- Assign numbers to each row of the departments table.
- **Exercise 2.1:** Assign numbers to each row of the department for the Entertainment division.

Task 3: ROW_NUMBER() - Part Two

- Retrieve all the data from the employees table.
- Retrieve a list of employee_id, first_name, hire_date, and department of all employees in the sports department ordered by the hire date.
- Order by multiple columns.
- Ordering in- and outside the OVER() clause.

Task 4: PARTITION BY

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

- **Exercise 4.1:** Order by the hire_date

Task 5: PARTITION BY WITH CTE

- Retrieve all data from the sales and customers tables.
- Create a common table expression to retrieve the customer_id, customer_name, segment, and how many times the customer has purchased from the mall.
- Number each customer by how many purchases they've made.
- **Exercise 5.1:** Number each customer by their customer segment and by how many purchases they've made in descending order.

Task 6: Fetching: LEAD() & LAG()

- Retrieve all employees first name, department, salary, and the salary after that employee.
- Retrieve all employees first name, department, salary, and the salary before that employee.
- Retrieve all employees first name, department, salary, and the salary after that employee in order of their salaries.
- **Exercise 6.1:** Retrieve all employees first name, department, salary, and the salary before that employee in order of their salaries. Call the new column closest_higher_salary.
- **Exercise 6.2:** Retrieve all employees first name, department, salary, and the salary after that employee for each department in order of their salaries. Call the new column closest_lowest_salary.

Task 7: FIRST_VALUE() - Part One

- Retrieve the first_name, last_name, department, and hire_date of all employees. Add a new column called first_emp_date that returns the hire date of the first hired employee.
- Find the difference between the hire date of the first employee hired and every other employee.
- **Exercise 7.1:** Partition by department.
- **Exercise 7.2:** Find the difference between the hire date of the first employee hired and every other employee partitioned by department.

Task 8: FIRST_VALUE() - Part Two

- **Exercise 8.1:** Return the first salary for different departments, Order by the salary in descending order.

- **Exercise 8.2:** Return the first salary for different departments. Order by the first_name in ascending order.
- Return the fifth salary for different departments. Order by the first_name in ascending order.
- 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 **Introduction to SQL Window Functions** 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.

- **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 **Introduction to SQL Window Functions** 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!

- **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.

- **Introduction to SQL Window Functions**

About Guided Project

Now we will query the project-db database **using SQL SELECT statement together with SQL window functions**. 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 - Part One
- Use the ROW_NUMBER() function - Part Two
- Use PARTITION BY in the OVER() clause
- Use PARTITION BY with CTE
- Use SQL Window Fetching functions: LEAD() & LAG()
- Use the FIRST_VALUE function - Part One
- Use the FIRST_VALUE function - Part Two

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, Introduction to SQL Window Functions, to enhance your learning experience. The app will reference basic information like your name, email, and Coursera ID.

- **Guided Project: Introduction to SQL Window Functions**

Task 1: Getting Started

In this task, we will get started with the project by retrieving all the data in the project-db database

1.1: Retrieve all the data in the project-db database

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

Task 2: ROW_NUMBER() - Part One

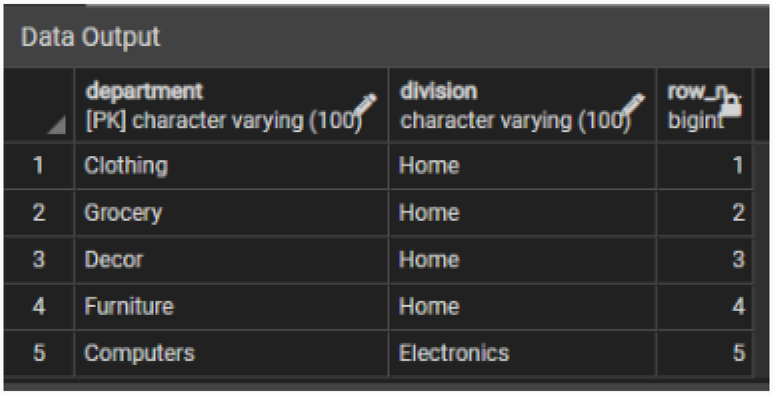
In this task, we will learn the ROW_NUMBER() and OVER() to assign numbers to each row

ROW_NUMBER() window function syntax:

```
ROW_NUMBER() OVER(
   [PARTITION BY column_1, column_2,…]
   [ORDER BY column_3,column_4,…]
)
```

2.1: Assign numbers to each row of the departments table

```
SELECT *, ROW_NUMBER() OVER() AS Row_N
FROM departments
ORDER BY Row_N;
```



	department [PK] character varying (100)	division character varying (100)	row_n bigint
1	Clothing	Home	1
2	Grocery	Home	2
3	Decor	Home	3
4	Furniture	Home	4
5	Computers	Electronics	5

Task 3: ROW_NUMBER() - Part Two

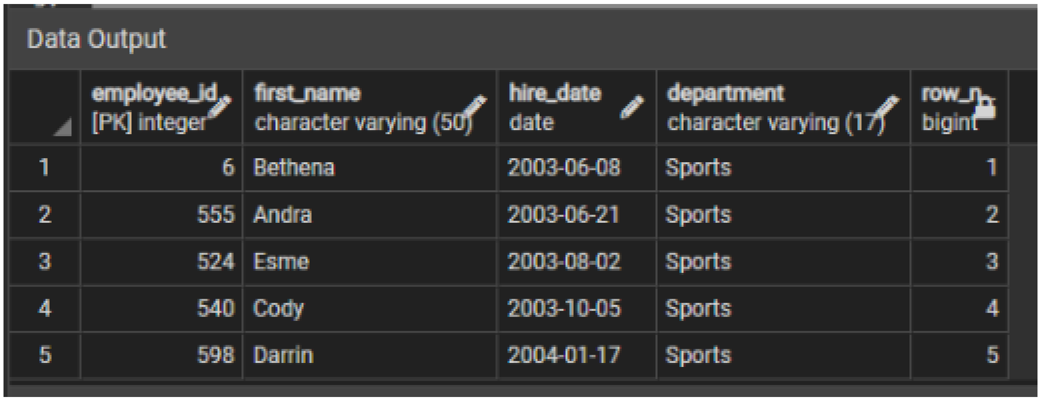
In this task, we will continue to learn how to assign numbers to each row using ROW_NUMBER() and OVER()

3.1: Retrieve all the data from the employees table:

```
SELECT * FROM employees;
```

3.2: Retrieve a list of employee_id, first_name, hire_date, and department of all employees in the sports department ordered by the hire date.

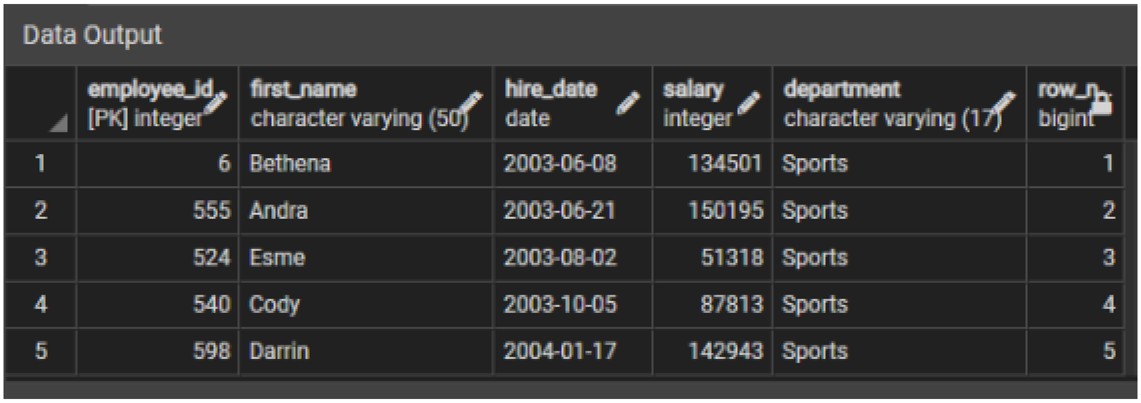
```
SELECT employee_id, first_name, hire_date, department,ROW_NUMBER()
OVER(ORDER BY hire_date ASC) AS Row_N
FROM employees
WHERE department='Sports'
ORDER BY Row_N ASC;
```



	employee_id [PK] integer	first_name character varying (50)	hire_date date	department character varying (17)	row_n bigint
1	6	Bethena	2003-06-08	Sports	1
2	555	Andra	2003-06-21	Sports	2
3	524	Esme	2003-08-02	Sports	3
4	540	Cody	2003-10-05	Sports	4
5	598	Darrin	2004-01-17	Sports	5

3.3: Retrieve a list of employee_id, first_name, hire_date, and department of all employees in the sports department ordered by the hire date ASC and salary DESC

```
SELECT employee_id, first_name, hire_date, salary, department,
ROW_NUMBER() OVER(ORDER BY hire_date ASC, salary DESC) AS Row_N
FROM employees
WHERE department='Sports'
ORDER BY Row_N
```



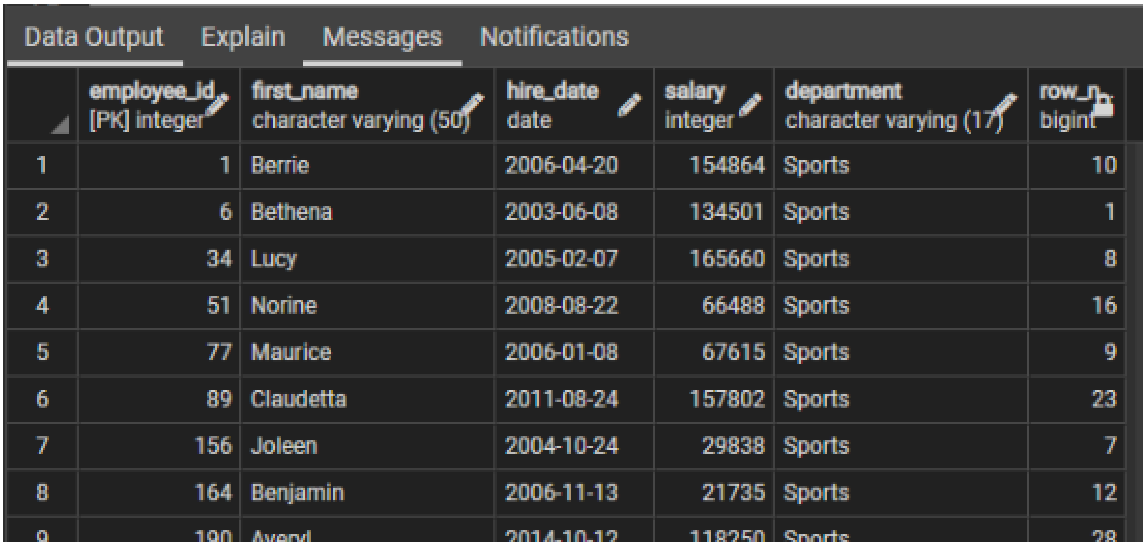
Data Output

	employee_id [PK] integer	first_name character varying (50)	hire_date date	salary integer	department character varying (17)	row_n bigint
1	6	Bethena	2003-06-08	134501	Sports	1
2	555	Andra	2003-06-21	150195	Sports	2
3	524	Esme	2003-08-02	51318	Sports	3
4	540	Cody	2003-10-05	87813	Sports	4
5	598	Darrin	2004-01-17	142943	Sports	5

In this result, we notice that the salary is not ordered correctly because the OVER() clause takes in consideration the first order by then the second.
The outside ORDER BY clause runs first then any ORDER BY clause inside the OVER() clause.

3.4: Ordering in- and outside the OVER() clause

```
SELECT employee_id, first_name, hire_date, salary, department,
ROW_NUMBER() OVER(ORDER BY hire_date ASC, salary DESC) AS Row_N
FROM employees
WHERE department = 'Sports'
ORDER BY employee_id;
```



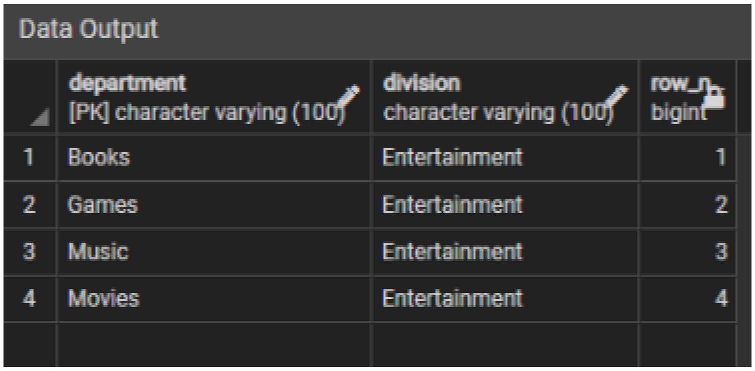
	employee_id [PK] integer	first_name character varying (50)	hire_date date	salary integer	department character varying (17)	row_n bigint
1	1	Berrie	2006-04-20	154864	Sports	10
2	6	Bethena	2003-06-08	134501	Sports	1
3	34	Lucy	2005-02-07	165660	Sports	8
4	51	Norine	2008-08-22	66488	Sports	16
5	77	Maurice	2006-01-08	67615	Sports	9
6	89	Claudetta	2011-08-24	157802	Sports	23
7	156	Joleen	2004-10-24	29838	Sports	7
8	164	Benjamin	2006-11-13	21735	Sports	12
9	190	Averyl	2014-10-12	118250	Sports	28

Here also the ORDER BY priority goes to the outside ORDER BY clause which is emploee_id.

-- Exercise 2.1:

Assign numbers to each row of the department for the Entertainment division:

```
SELECT *,
ROW_NUMBER() OVER() AS Row_N
FROM departments
WHERE division ='Entertainment'
ORDER BY Row_N ASC;
```



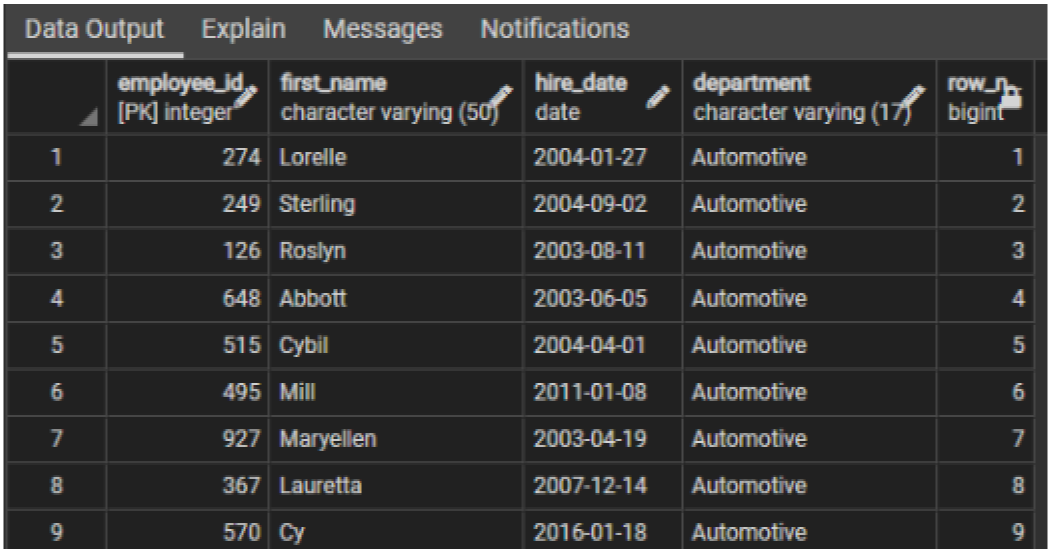
	department [PK] character varying (100)	division character varying (100)	row_n bigint
1	Books	Entertainment	1
2	Games	Entertainment	2
3	Music	Entertainment	3
4	Movies	Entertainment	4

Task 4: PARTITION BY

In this task, we will learn how to use the PARTITION BY clause inside OVER()

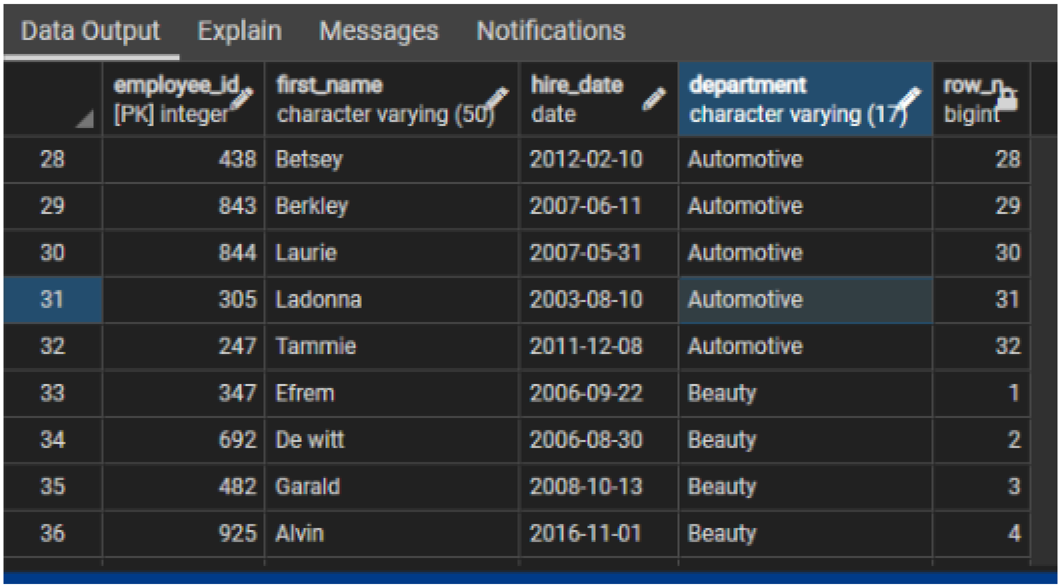
4.1: Retrieve the employee_id, first_name, hire_date of employees for different departments

```
SELECT employee_id, first_name, hire_date,department,
ROW_NUMBER() OVER( PARTITION BY department ) AS Row_N
FROM employees
ORDER BY department ASC;
```



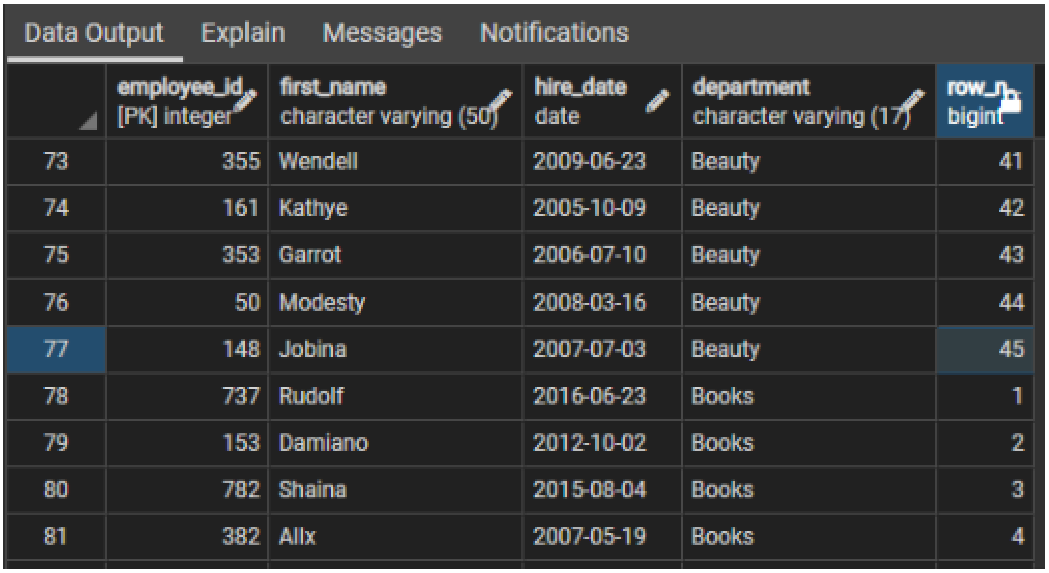
	employee_id [PK] integer	first_name character varying (50)	hire_date date	department character varying (17)	row_n bigint
1	274	Lorelle	2004-01-27	Automotive	1
2	249	Sterling	2004-09-02	Automotive	2
3	126	Roslyn	2003-08-11	Automotive	3
4	648	Abbott	2003-06-05	Automotive	4
5	515	Cybil	2004-04-01	Automotive	5
6	495	Mill	2011-01-08	Automotive	6
7	927	Maryellen	2003-04-19	Automotive	7
8	367	Lauretta	2007-12-14	Automotive	8
9	570	Cy	2016-01-18	Automotive	9

- - - - - - - - - - -



	employee_id [PK] integer	first_name character varying (50)	hire_date date	department character varying (17)	row_n bigint
28	438	Betsey	2012-02-10	Automotive	28
29	843	Berkley	2007-06-11	Automotive	29
30	844	Laurie	2007-05-31	Automotive	30
31	305	Ladonna	2003-08-10	Automotive	31
32	247	Tammie	2011-12-08	Automotive	32
33	347	Efrem	2006-09-22	Beauty	1
34	692	De witt	2006-08-30	Beauty	2
35	482	Garald	2008-10-13	Beauty	3
36	925	Alvin	2016-11-01	Beauty	4

- - - - - - - - - - -

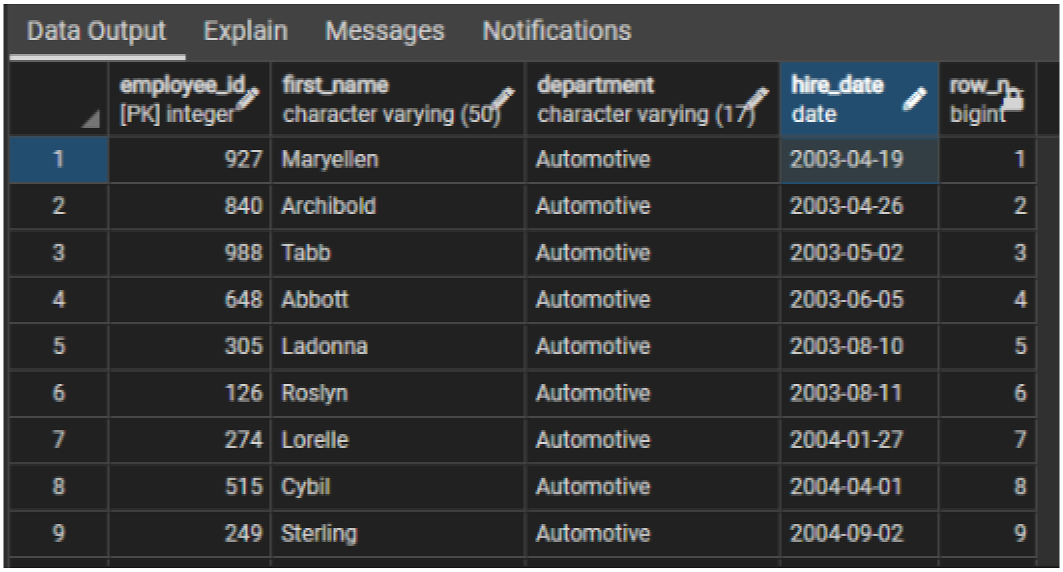


	employee_id [PK] integer	first_name character varying (50)	hire_date date	department character varying (17)	row_n bigint
73	355	Wendell	2009-06-23	Beauty	41
74	161	Kathye	2005-10-09	Beauty	42
75	353	Garrot	2006-07-10	Beauty	43
76	50	Modesty	2008-03-16	Beauty	44
77	148	Jobina	2007-07-03	Beauty	45
78	737	Rudolf	2016-06-23	Books	1
79	153	Damiano	2012-10-02	Books	2
80	782	Shaina	2015-08-04	Books	3
81	382	Allx	2007-05-19	Books	4

From these results, you will notice that the Row_N starts at 1 and increases by 1 until it reaches the last row for that department name and anytime the department name changes the Row_N starts at 1 again. The Row_N is in scope by the department name.

Exercise 4.1:
Order by the hire_date.

```
SELECT employee_id, first_name, department, hire_date,ROW_NUMBER()
        OVER(PARTITION BY department ORDER BY hire_date) AS Row_N
FROM employees
ORDER BY department ASC;
```



	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
8	515	Cybil	Automotive	2004-04-01	8
9	249	Sterling	Automotive	2004-09-02	9

The result is ordered by the department name first and then ordered by the hire_date inside the OVER() clause for each department name.

Task 5: PARTITION BY WITH CTE

A common table expression, or CTE, is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement.
Each SQL CTE is like a named query, whose result is stored in a virtual table (a CTE) to be referenced later in the main query.

Syntax of a common table expression:

```
WITH my_cte AS ( SELECT a,b,c  FROM T1)
SELECT a,c
FROM my_cte
WHERE ....
```

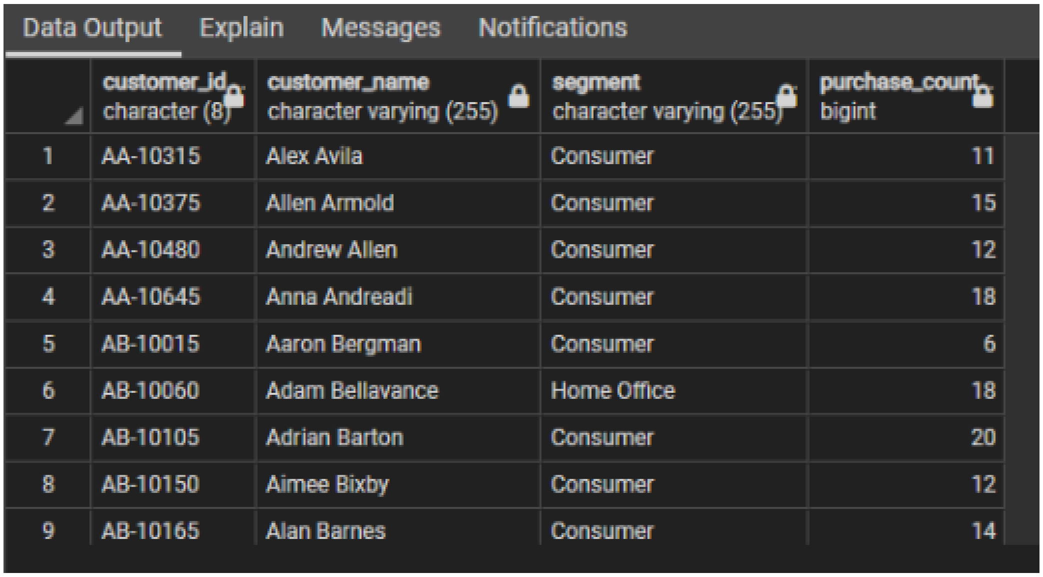
5.1: Retrieve all data from the sales and customers tables

```
SELECT * FROM sales;
SELECT * FROM customers;
```

5.2: Create a common table expression to retrieve the customer_id, customer_name, segment and how many times the customer has purchased from the mall.

```
WITH customer_purchase AS (
        SELECT s.customer_id, c.customer_name, c.segment,
        COUNT(*) AS purchase_count
        FROM sales s
        JOIN customers c
        ON s.customer_id = c.customer_id
        GROUP BY s.customer_id, c.customer_name, c.segment
        ORDER BY customer_id)
```

When I run the SQL statement inside the CTE, I will get the following results:

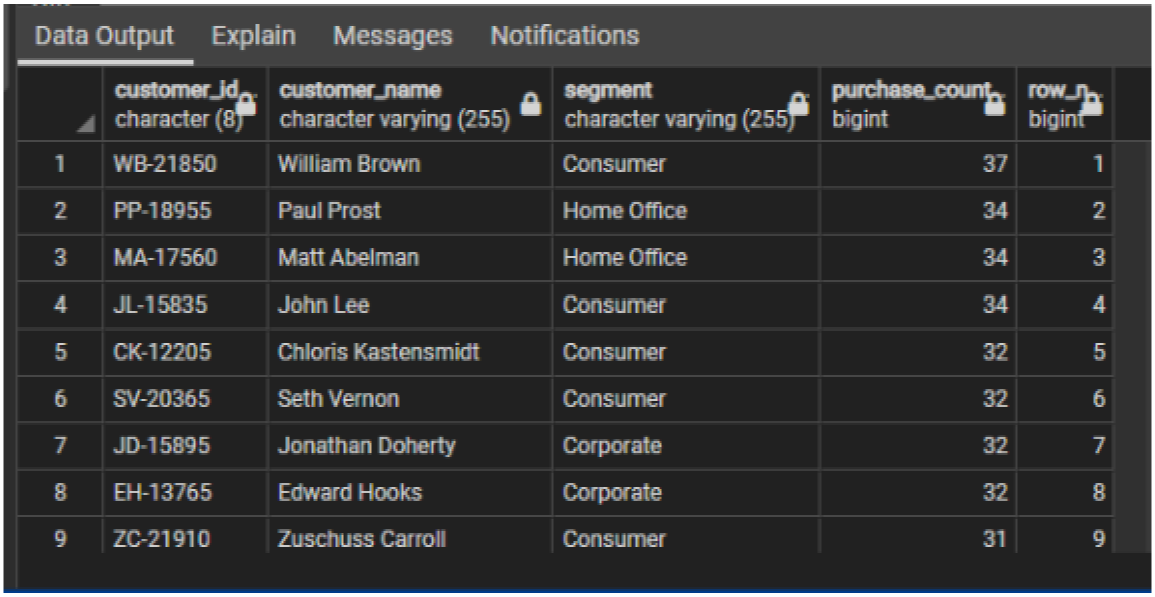


	customer_id character (8)	customer_name character varying (255)	segment character varying (255)	purchase_count bigint
1	AA-10315	Alex Avila	Consumer	11
2	AA-10375	Allen Armold	Consumer	15
3	AA-10480	Andrew Allen	Consumer	12
4	AA-10645	Anna Andreadi	Consumer	18
5	AB-10015	Aaron Bergman	Consumer	6
6	AB-10060	Adam Bellavance	Home Office	18
7	AB-10105	Adrian Barton	Consumer	20
8	AB-10150	Aimee Bixby	Consumer	12
9	AB-10165	Alan Barnes	Consumer	14

5.3: Number each customer by how many purchases they've made

```
WITH customer_purchase AS(
        SELECT s.customer_id, c.customer_name, c.segment,
        COUNT(*) AS purchase_count
        FROM sales s
        JOIN customers c
        ON s.customer_id = c.customer_id
        GROUP BY s.customer_id, c.customer_name, c.segment
        ORDER BY customer_id)
```

```
SELECT customer_id, customer_name, segment, purchase_count,
        ROW_NUMBER() OVER(ORDER BY purchase_count DESC ) AS Row_N
FROM customer_purchase
ORDER BY Row_N, purchase_count DESC
```

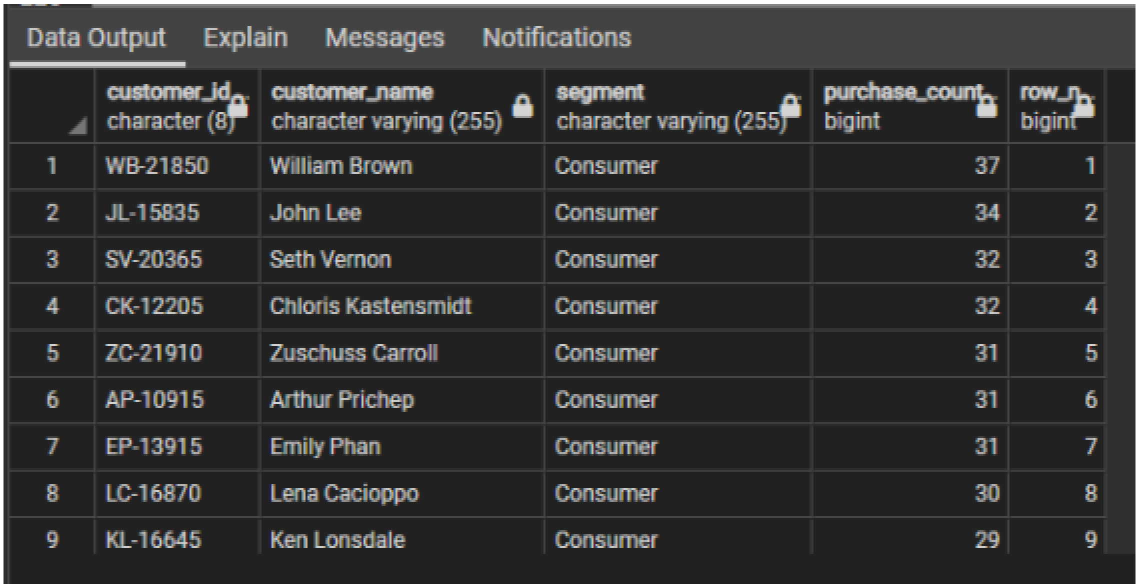


	customer_id character (8)	customer_name character varying (255)	segment character varying (255)	purchase_count bigint	row_n bigint
1	WB-21850	William Brown	Consumer	37	1
2	PP-18955	Paul Prost	Home Office	34	2
3	MA-17560	Matt Abelman	Home Office	34	3
4	JL-15835	John Lee	Consumer	34	4
5	CK-12205	Chloris Kastensmidt	Consumer	32	5
6	SV-20365	Seth Vernon	Consumer	32	6
7	JD-15895	Jonathan Doherty	Corporate	32	7
8	EH-13765	Edward Hooks	Corporate	32	8
9	ZC-21910	Zuschuss Carroll	Consumer	31	9

Exercise 5.1:
Number each customer by their customer segment and by how many purchases they've made in descending order.

```
WITH customer_purchase AS (
        SELECT s.customer_id, c.customer_name, c.segment,
        COUNT(*) AS purchase_count
        FROM sales s
        JOIN customers c
        ON s.customer_id = c.customer_id
        GROUP BY s.customer_id, c.customer_name, c.segment
        ORDER BY customer_id)

SELECT customer_id, customer_name, segment, purchase_count,
ROW_NUMBER() OVER (PARTITION BY segment
                        ORDER BY purchase_count DESC) AS Row_N
FROM customer_purchase
ORDER BY segment, purchase_count DESC;
```



	customer_id character (8)	customer_name character varying (255)	segment character varying (255)	purchase_count bigint	row_n bigint
1	WB-21850	William Brown	Consumer	37	1
2	JL-15835	John Lee	Consumer	34	2
3	SV-20365	Seth Vernon	Consumer	32	3
4	CK-12205	Chloris Kastensmidt	Consumer	32	4
5	ZC-21910	Zuschuss Carroll	Consumer	31	5
6	AP-10915	Arthur Prichep	Consumer	31	6
7	EP-13915	Emily Phan	Consumer	31	7
8	LC-16870	Lena Cacioppo	Consumer	30	8
9	KL-16645	Ken Lonsdale	Consumer	29	9

Task 6: Fetching: LEAD() & LAG()

The **LAG()** function is used to get a value from a row that **precedes** the current row.
The **LEAD()** function is used to get value from a row that **succeeds** the current row.

Syntax:

LEAD(expr, N, default) **OVER** (Window_specification | Window_name)
LAG(expr, N, default) **OVER** (Window_specification | Window_name)
The N and default argument in the function is optional.

Parameters used:

expr: It can be a column or any built-in function.

N: It is a positive value which determines the number of rows preceding/succeeding the current row. If it is omitted in the query then its default value is 1.

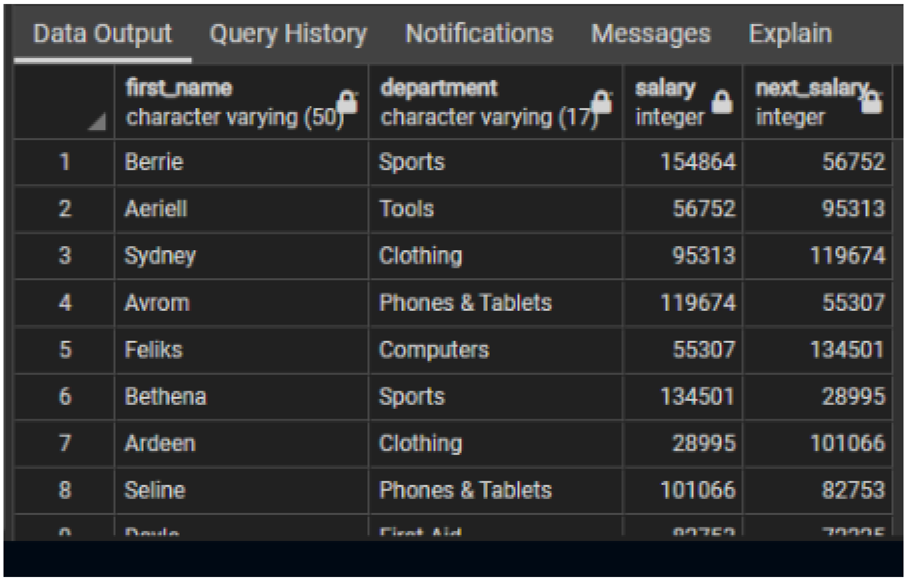
default: It is the default value return by function in-case no row precedes/succeeds the current row by N rows. If it is missing then it is by default NULL.

OVER(): It defines how rows are partitioned into groups. If OVER() is empty then the function computes the result using all rows.

Window_specification: It consists of a query partition clause which determines how the query rows are partitioned and ordered.
Window_name: If window is specified elsewhere in the query then it is referenced using this Window_name.

6.1 Retrieve all employees first name, department, salary, and the salary after that employee.

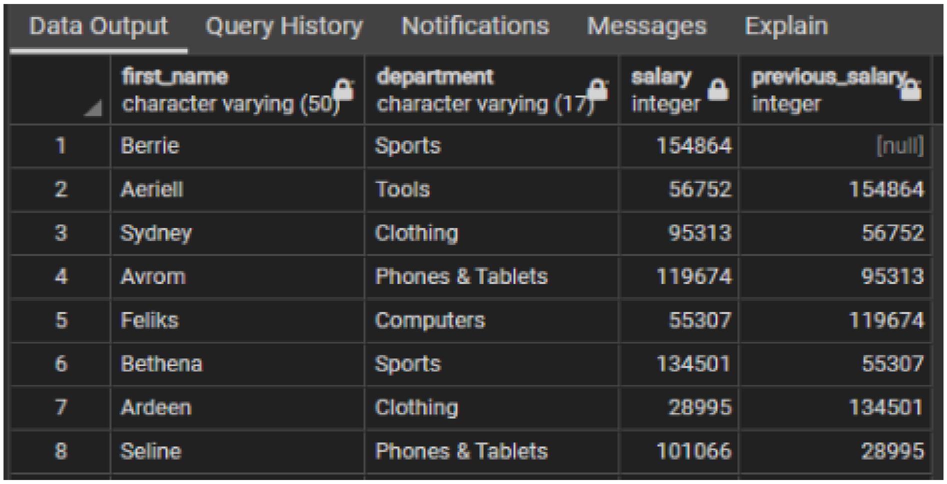
```
SELECT first_name, department, salary, LEAD(salary,1) OVER()AS Next_Salary
FROM employees;
```



	first_name character varying (50)	department character varying (17)	salary integer	next_salary integer
1	Berrie	Sports	154864	56752
2	Aeriell	Tools	56752	95313
3	Sydney	Clothing	95313	119674
4	Avrom	Phones & Tablets	119674	55307
5	Feliks	Computers	55307	134501
6	Bethena	Sports	134501	28995
7	Ardeen	Clothing	28995	101066
8	Seline	Phones & Tablets	101066	82753
9	Doyle	First Aid	82753	[illegible]

6.2 Retrieve all employees first name, department, salary, and the salary before that employee.

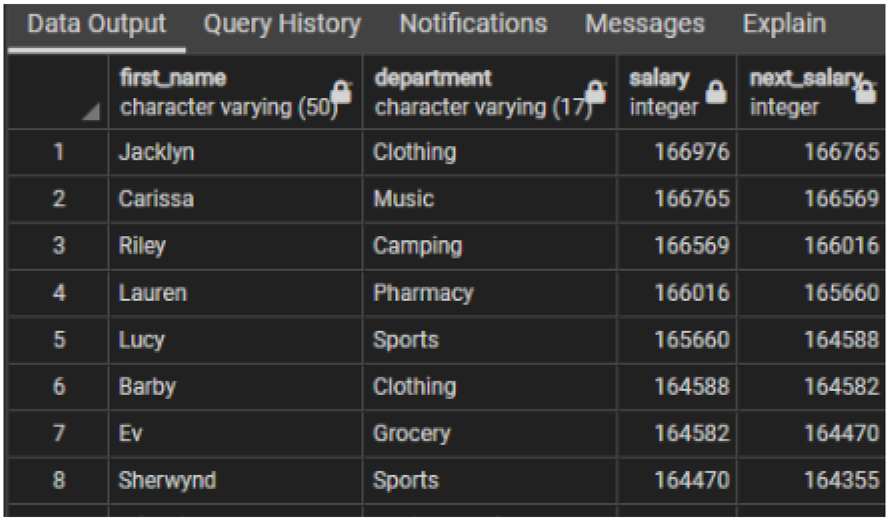
```
SELECT first_name, department, salary, LAG(salary,1) OVER()AS Previous_Salary
FROM employees;
```



	first_name character varying (50)	department character varying (17)	salary integer	previous_salary integer
1	Berrie	Sports	154864	[null]
2	Aeriell	Tools	56752	154864
3	Sydney	Clothing	95313	56752
4	Avrom	Phones & Tablets	119674	95313
5	Feliks	Computers	55307	119674
6	Bethena	Sports	134501	55307
7	Ardeen	Clothing	28995	134501
8	Seline	Phones & Tablets	101066	28995

6.3 Retrieve all employees first name, department, salary, and the salary after that employee in order of their salaries.

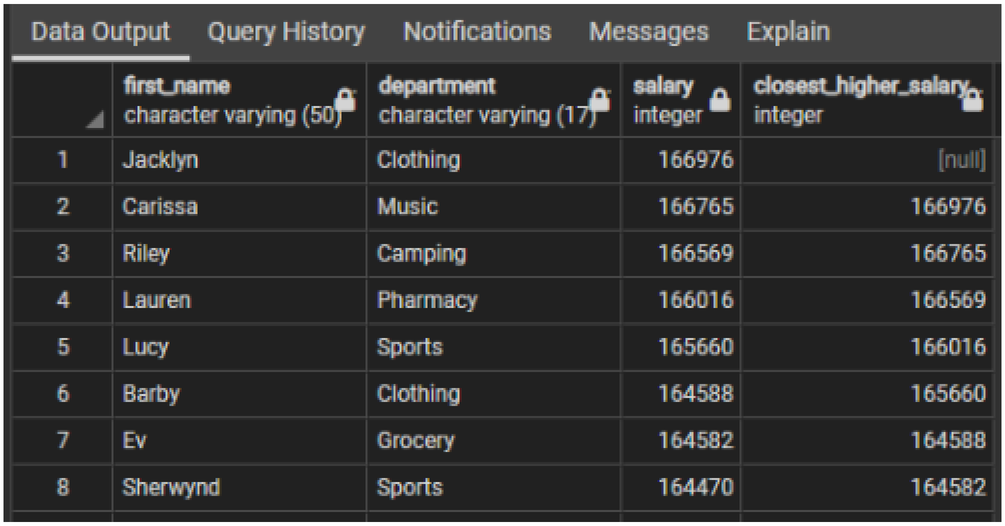
```
SELECT first_name, department, salary,
LEAD(salary,1) OVER(ORDER BY salary DESC)AS Next_Salary
FROM employees
```



	first_name character varying (50)	department character varying (17)	salary integer	next_salary integer
1	Jacklyn	Clothing	166976	166765
2	Carissa	Music	166765	166569
3	Riley	Camping	166569	166016
4	Lauren	Pharmacy	166016	165660
5	Lucy	Sports	165660	164588
6	Barby	Clothing	164588	164582
7	Ev	Grocery	164582	164470
8	Sherwynd	Sports	164470	164355

Exercise 6.1: Retrieve all employees first name, department, salary, and the salary before that employee in order of their salaries. Call the new column closest_higher_salary.

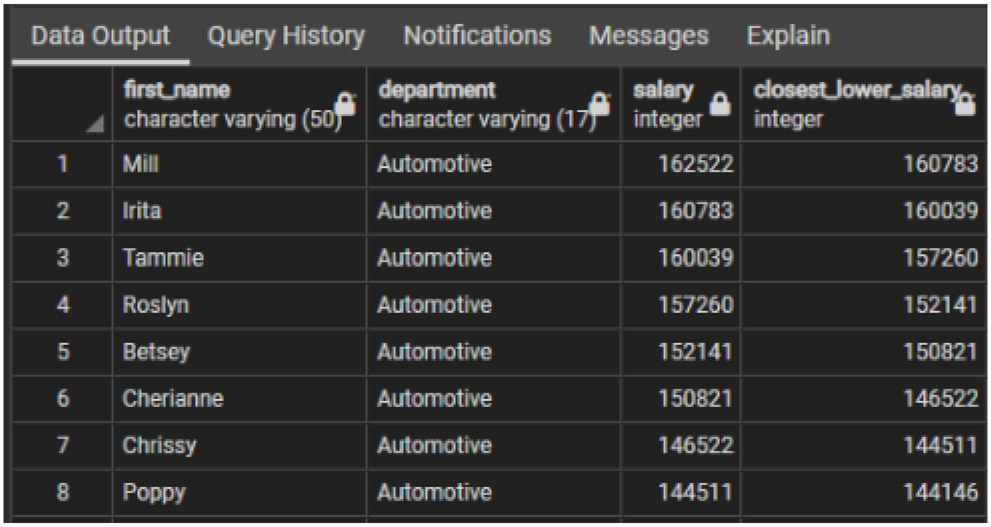
```
SELECT first_name, department, salary,
LAG(salary) OVER (ORDER BY salary DESC) AS closest_higher_salary
FROM employees;
```



	first_name character varying (50)	department character varying (17)	salary integer	closest_higher_salary integer
1	Jacklyn	Clothing	166976	[null]
2	Carissa	Music	166765	166976
3	Riley	Camping	166569	166765
4	Lauren	Pharmacy	166016	166569
5	Lucy	Sports	165660	166016
6	Barby	Clothing	164588	165660
7	Ev	Grocery	164582	164588
8	Sherwynd	Sports	164470	164582

Exercise 6.2: Retrieve all employees first name, department, salary, and the salary after that employee for each department in order of their salaries. Call the new column closest_lowest_salary.

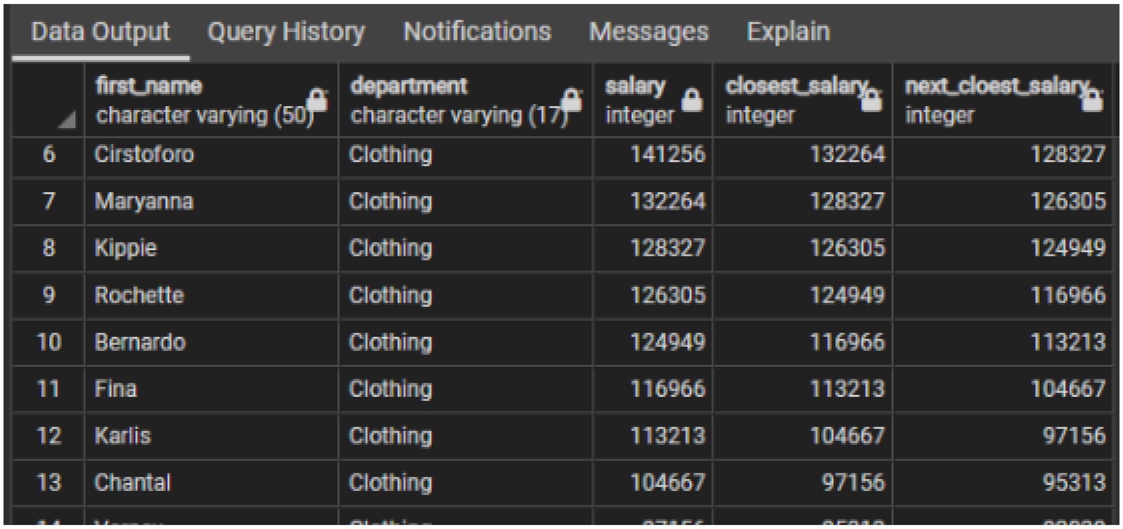
```
SELECT first_name, department, salary,
LEAD(salary) OVER (PARTITION BY department ORDER BY salary DESC
                        ) AS closest_lower_salary
FROM employees;
```



	first_name character varying (50)	department character varying (17)	salary integer	closest_lower_salary integer
1	Mill	Automotive	162522	160783
2	Irita	Automotive	160783	160039
3	Tammie	Automotive	160039	157260
4	Roslyn	Automotive	157260	152141
5	Betsey	Automotive	152141	150821
6	Cherianne	Automotive	150821	146522
7	Chrissy	Automotive	146522	144511
8	Poppy	Automotive	144511	144146

What do you think this query will return?

```
SELECT first_name, department, salary,
LEAD(salary, 1) OVER (ORDER BY salary DESC) closest_salary,
LEAD(salary, 2) OVER (ORDER BY salary DESC) next_cloest_salary
FROM employees
WHERE department = 'Clothing';
```



	first_name character varying (50)	department character varying (17)	salary integer	closest_salary integer	next_cloest_salary integer
6	Cirstoforo	Clothing	141256	132264	128327
7	Maryanna	Clothing	132264	128327	126305
8	Kippie	Clothing	128327	126305	124949
9	Rochette	Clothing	126305	124949	116966
10	Bernardo	Clothing	124949	116966	113213
11	Fina	Clothing	116966	113213	104667
12	Karlis	Clothing	113213	104667	97156
13	Chantal	Clothing	104667	97156	95313

This query returns the first and the second next salary in the clothing department ordered by the salary descending.

Task 7: FIRST_VALUE() - Part One

Overview of FIRST_VALUE() function

The FIRST_VALUE() is a window function that returns the first value in an ordered set of values.

Syntax:

```
FIRST_VALUE(expression) OVER (
                                        partition_clause
                                        order_clause
                                        Frame_clause
                                              )
```

Expression:The return value of the expression from the first row in a partition or result set.

The OVER clause consists of three clauses: partition_clause, order_clause, and frame_clause.

Partition_clause:The partition_clause clause has the following syntax:

PARTITION BY expr1, expr2, ...

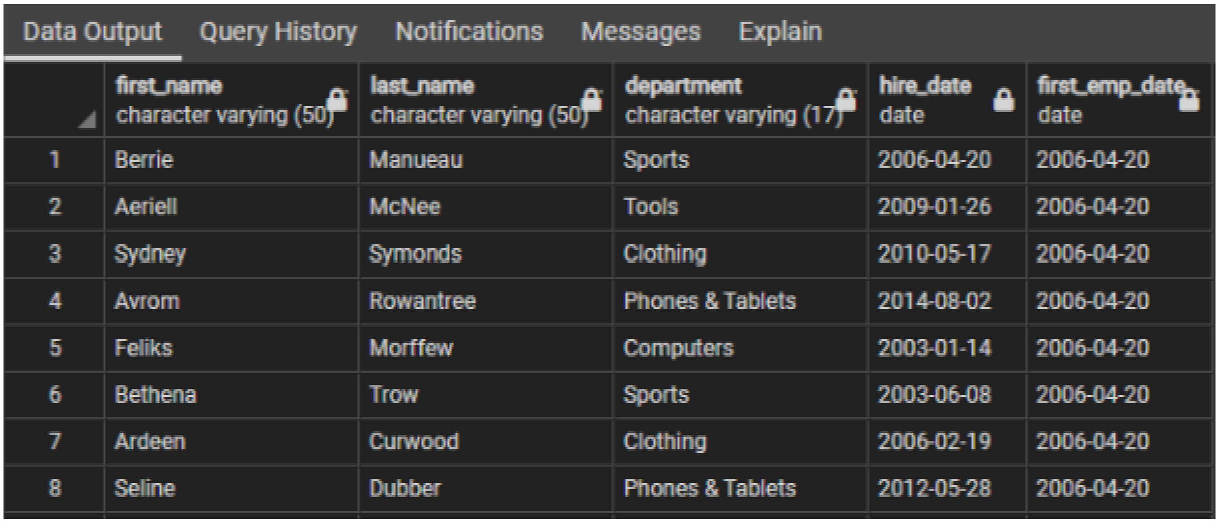
The PARTITION BY clause divides the rows of the result sets into partitions to which the FIRST_VALUE() function applies. If you skip the PARTITION BY clause, the function treats the whole result set as a single partition.

Order_clause: The order_clause clause sorts the rows in partitions to which the FIRST_VALUE() function applies. The ORDER BY clause has the following syntax:
ORDER BY expr1 [ASC | DESC], expr2, ...

Frame_clause:The frame_clause defines the subset (or frame) of the current partition.

7.1: Retrieve the first_name, last_name, department, and hire_date of all employees. Add a new column called first_emp_date that returns the hire date of the first hired employee.

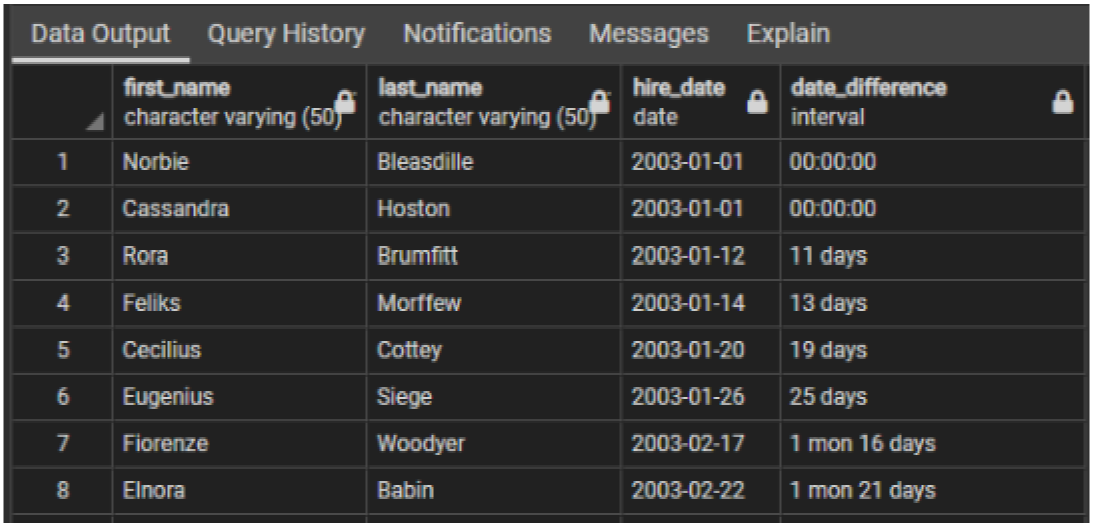
```
SELECT first_name, last_name, department, hire_date,
FIRST_VALUE(hire_date) OVER() AS first_emp_date
FROM employees;
```



	first_name character varying (50)	last_name character varying (50)	department character varying (17)	hire_date date	first_emp_date date
1	Berrie	Manueau	Sports	2006-04-20	2006-04-20
2	Aeriell	McNee	Tools	2009-01-26	2006-04-20
3	Sydney	Symonds	Clothing	2010-05-17	2006-04-20
4	Avrom	Rowantree	Phones & Tablets	2014-08-02	2006-04-20
5	Feliks	Morffew	Computers	2003-01-14	2006-04-20
6	Bethena	Trow	Sports	2003-06-08	2006-04-20
7	Ardeen	Curwood	Clothing	2006-02-19	2006-04-20
8	Seline	Dubber	Phones & Tablets	2012-05-28	2006-04-20

7.2: Find the difference between the hire date of the first employee hired and every other employee.

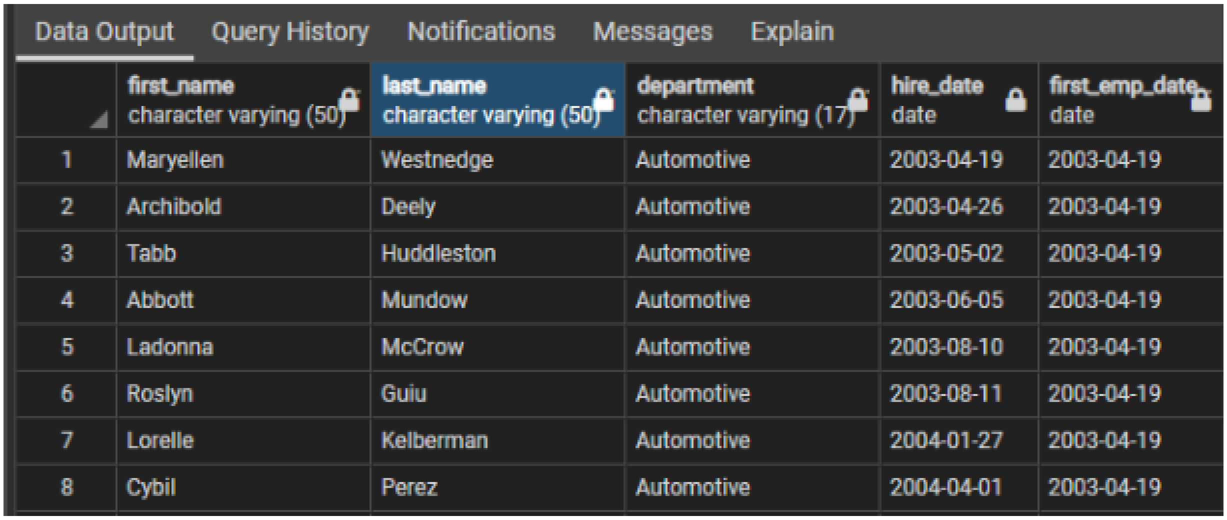
```
SELECT first_name, last_name, hire_date,
AGE(hire_date , FIRST_VALUE(hire_date) over(ORDER BY hire_date)) AS
date_difference
FROM employees
ORDER BY hire_date;
```



	first_name character varying (50)	last_name character varying (50)	hire_date date	date_difference interval
1	Norbie	Bleasdille	2003-01-01	00:00:00
2	Cassandra	Hoston	2003-01-01	00:00:00
3	Rora	Brumfitt	2003-01-12	11 days
4	Feliks	Morffew	2003-01-14	13 days
5	Cecilius	Cottey	2003-01-20	19 days
6	Eugenius	Siege	2003-01-26	25 days
7	Fiorenze	Woodyer	2003-02-17	1 mon 16 days
8	Elnora	Babin	2003-02-22	1 mon 21 days

Exercise 7.1: Partition by department.

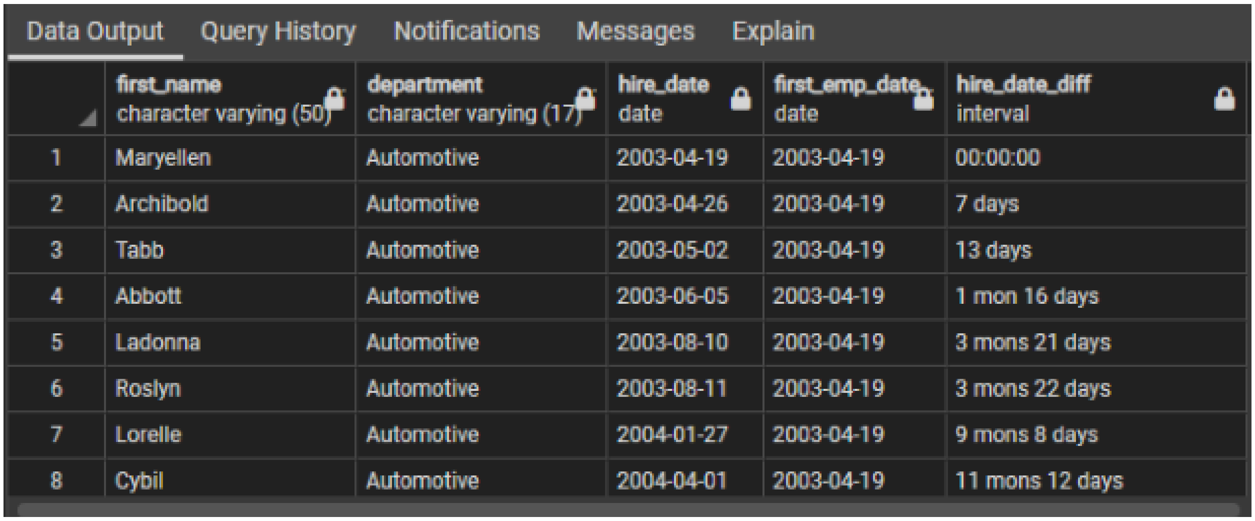
```
SELECT first_name, last_name, department, hire_date,
        FIRST_VALUE(hire_date) OVER (PARTITION BY department
         ORDER BY hire_date) AS first_emp_date
FROM employees;
```



	first_name character varying (50)	last_name character varying (50)	department character varying (17)	hire_date date	first_emp_date date
1	Maryellen	Westnedge	Automotive	2003-04-19	2003-04-19
2	Archibold	Deely	Automotive	2003-04-26	2003-04-19
3	Tabb	Huddleston	Automotive	2003-05-02	2003-04-19
4	Abbott	Mundow	Automotive	2003-06-05	2003-04-19
5	Ladonna	McCrow	Automotive	2003-08-10	2003-04-19
6	Roslyn	Guiu	Automotive	2003-08-11	2003-04-19
7	Lorelle	Kelberman	Automotive	2004-01-27	2003-04-19
8	Cybil	Perez	Automotive	2004-04-01	2003-04-19

Exercise 7.2: Find the difference between the hire date of the first employee hired and every other employee partitioned by department.

```
SELECT *,AGE(hire_date,first_emp_date)AS hire_date_diff
FROM (
    SELECT first_name, department, hire_date,
    FIRST_VALUE(hire_date) OVER (PARTITION BY department
    ORDER BY hire_date) AS first_emp_date
    FROM employees) a;
```

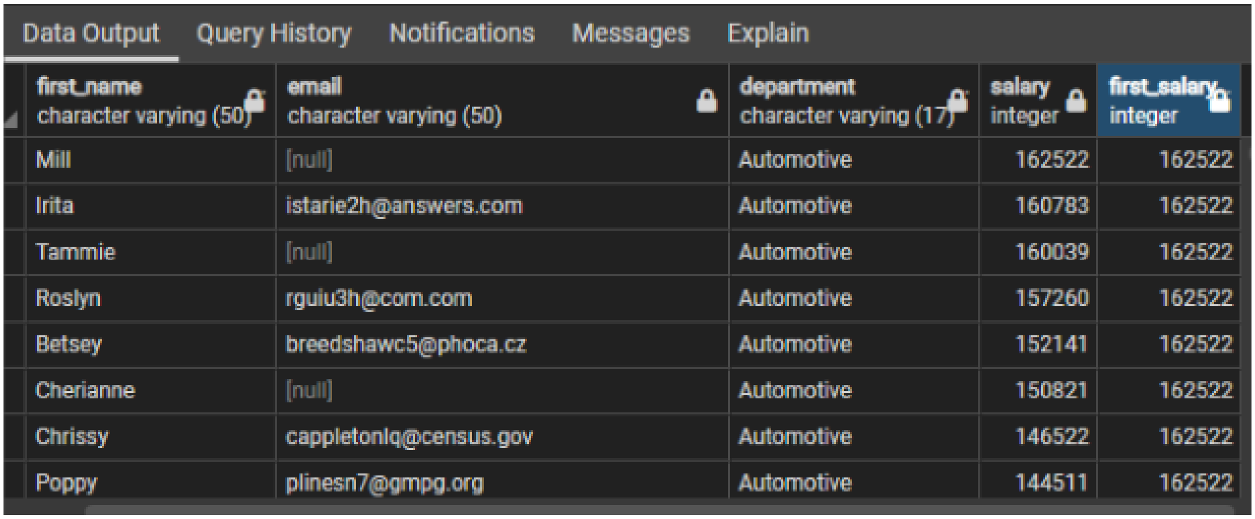


	first_name character varying (50)	department character varying (17)	hire_date date	first_emp_date date	hire_date_diff interval
1	Maryellen	Automotive	2003-04-19	2003-04-19	00:00:00
2	Archibold	Automotive	2003-04-26	2003-04-19	7 days
3	Tabb	Automotive	2003-05-02	2003-04-19	13 days
4	Abbott	Automotive	2003-06-05	2003-04-19	1 mon 16 days
5	Ladonna	Automotive	2003-08-10	2003-04-19	3 mons 21 days
6	Roslyn	Automotive	2003-08-11	2003-04-19	3 mons 22 days
7	Lorelle	Automotive	2004-01-27	2003-04-19	9 mons 8 days
8	Cybil	Automotive	2004-04-01	2003-04-19	11 mons 12 days

Task 8: FIRST_VALUE() - Part Two

Exercise 8.1: Return the first salary for different departments, Order by the salary in descending order.

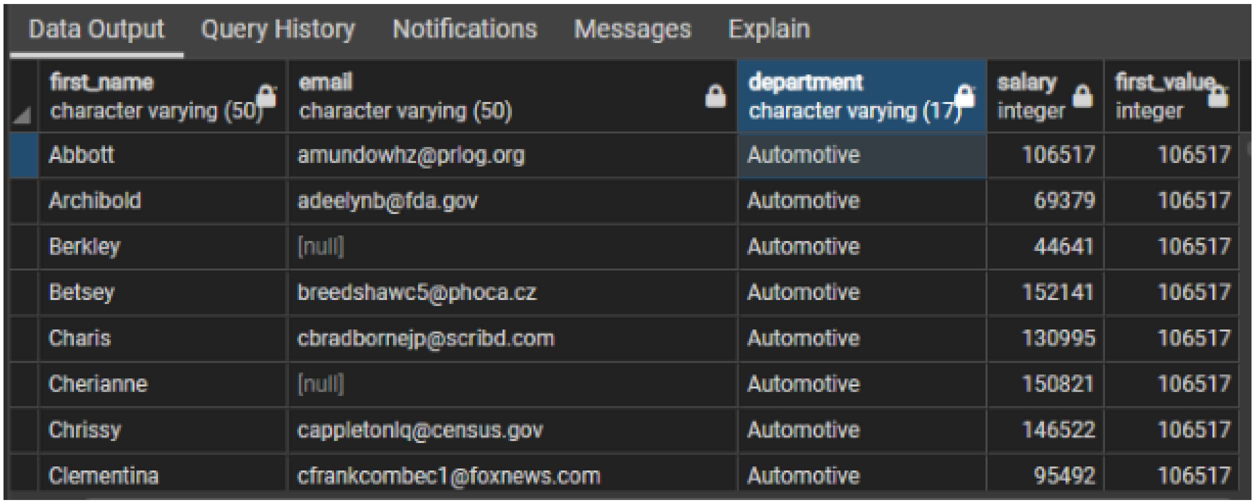
```
SELECT first_name, email, department, salary,
        FIRST_VALUE(salary) OVER(PARTITION BY department
         ORDER BY salary DESC) first_salary
FROM employees;
```



first_name character varying (50)	email character varying (50)	department character varying (17)	salary integer	first_salary integer
Mill	[null]	Automotive	162522	162522
Irita	istarie2h@answers.com	Automotive	160783	162522
Tammie	[null]	Automotive	160039	162522
Roslyn	rguiu3h@com.com	Automotive	157260	162522
Betsey	breedshawc5@phoca.cz	Automotive	152141	162522
Cherianne	[null]	Automotive	150821	162522
Chrissy	cappletonlq@census.gov	Automotive	146522	162522
Poppy	plinesn7@gmpg.org	Automotive	144511	162522

Exercise 8.2: Return the first salary for different departments. Order by the first_name in ascending order.

```
SELECT first_name, email, department, salary,
        FIRST_VALUE(salary) OVER(PARTITION BY department
        ORDER BY first_name ASC)
FROM employees;
```



first_name character varying (50)	email character varying (50)	department character varying (17)	salary integer	first_value integer
Abbott	amundowhz@prlog.org	Automotive	106517	106517
Archibold	adeelynb@fda.gov	Automotive	69379	106517
Berkley	[null]	Automotive	44641	106517
Betsey	breedshawc5@phoca.cz	Automotive	152141	106517
Charis	cbradbornejp@scribd.com	Automotive	130995	106517
Cherianne	[null]	Automotive	150821	106517
Chrissy	cappletonlq@census.gov	Automotive	146522	106517
Clementina	cfrankcombec1@foxnews.com	Automotive	95492	106517

8.3: NTH_VALUE() function:

The **NTH_VALUE()** function is used to get a value from the nth row in a result set.
Syntax:
NTH_VALUE(expression, offset) **OVER** ([**PARTITION BY** partition_expression]
[**ORDER BY** sort_expression [ASC | DESC]
frame_clause]
)

- **The expression** is the column from which the query is to be made or an expression on which the NTH_VALUE() function operates on.
- **The offset** is a positive integer that sets the row number according to the first row in the window against the value that the expression evaluates.
- **The PARTITION BY** clause distributes rows of the result set into partitions to which the NTH_VALUE() function applies.
- **The ORDER BY** clause is used for sorting the result of the query.
- **The frame_clause** is used to define the subset (or the frame) of the partition being used.

- Return the fifth salary for different departments. Order by the first_name in ascending order.

```
SELECT first_name, department, salary,
NTH_VALUE(salary,5)OVER(PARTITION BY department ORDER BY
first_name) AS fifth_dep_salary
FROM employees;
```

8.4: Wrap up the project.

- **Introduction to SQL Window Functions**
- **Quiz**