Session 6 – Manual

Window Functions

This manual covers advanced SQL operations, focusing on window functions and Combine query results. It provides detailed explanations and examples to help you understand and apply these concepts in real-world scenarios.

Objective:

By the end of this session, participants will be able to:

1. **Understand Window Functions:**
   - Define window functions and explain their use cases.
   - Describe the syntax of window functions, including the `OVER` clause, `PARTITION BY`, `ORDER BY`, and frame clauses.

2. **Apply Window Functions:**
   - Use various window functions such as `ROW_NUMBER`, `RANK`, and `DENSE_RANK` to perform complex calculations over a set of table rows.
   - Demonstrate practical examples of window functions to rank data and analyze partitions.

3. **Combine Query Results:**
   - Utilize the `UNION` operator to combine the results of multiple SELECT statements and understand how it removes duplicates.
   - Apply the `INTERSECT` operator to find common rows between multiple SELECT statements.
   - Use the `EXCEPT` operator to identify differences between two SELECT statements.

4. **Develop Advanced SQL Queries:**
   - Integrate window functions with other SQL clauses to create advanced and efficient queries.
   - Enhance data retrieval processes by combining results from different queries using set operations.

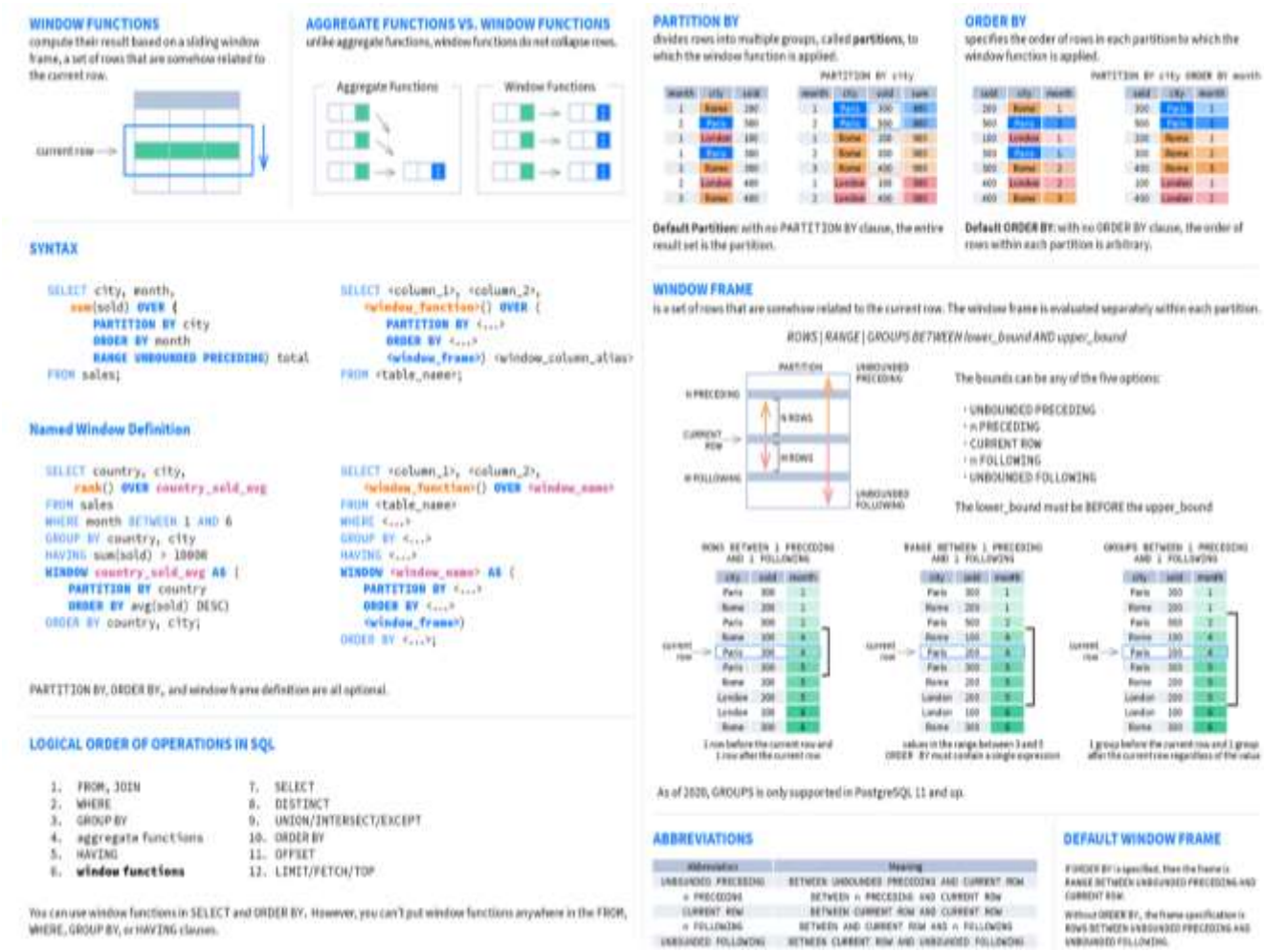
5. **Practical Application and Problem-Solving:**

- Solve real-world data problems by applying window functions and data modification techniques.
- Work on hands-on exercises and case studies to reinforce the concepts learned during the session.

Working with Window Functions

Introduction to Window Functions

Window functions perform calculations across a set of table rows related to the current row. Unlike aggregate functions, they do not collapse the rows into a single result, allowing detailed analytical operations over query results.



Syntax of Window Functions

A window function has the following syntax:

```
function_name([arguments]) OVER (
```

```
    [PARTITION BY partition_expression]
    [ORDER BY sort_expression]
    [frame_clause]
)
```

Components:

- **function_name**: The name of the window function (e.g., `RANK`, `DENSE_RANK`, `ROW_NUMBER`).
- **arguments**: Function-specific arguments.
- **OVER**: Specifies the window.
- **PARTITION BY**: Divides the result set into partitions.
- **ORDER BY**: Orders the rows within each partition.
- **frame_clause**: Defines a subset of rows within the partition.

Examples of Window Functions

1. **ROW_NUMBER**: Assigns a unique number to each row based on the specified order.

```
SELECT
    column1,
    ROW_NUMBER() OVER (PARTITION BY column2 ORDER BY column3) AS
row_num
FROM
    table_name;
```

2. **RANK**: Assigns a rank to each row within a partition, with gaps for ties.

```
SELECT
    column1,
    RANK() OVER (PARTITION BY column2 ORDER BY column3) AS rank
FROM
    table_name;
```

3. **DENSE_RANK**: Similar to RANK, but without gaps between ranks.

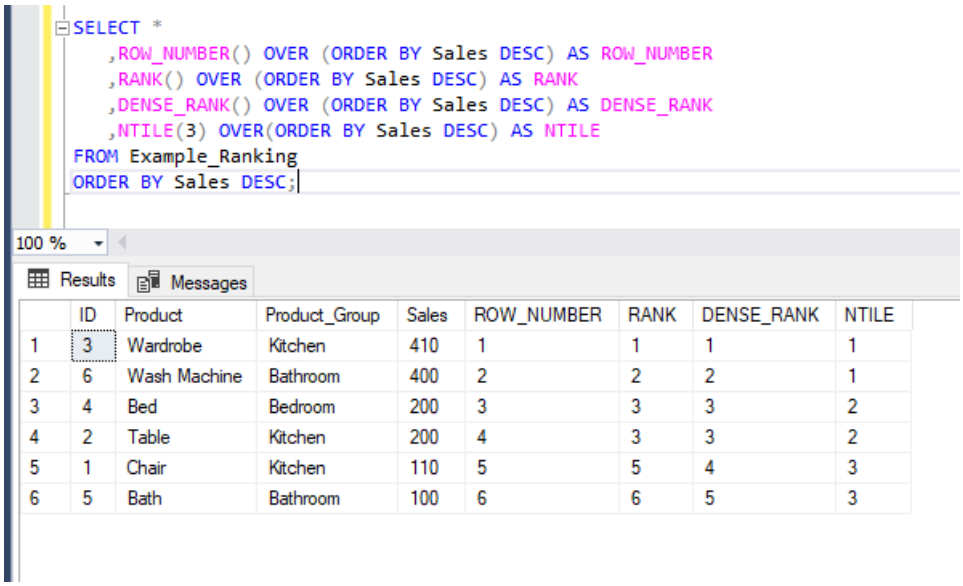
```
SELECT
    column1,
    DENSE_RANK() OVER (PARTITION BY column2 ORDER BY column3) AS
dense_rank
```

```
FROM
    table_name;
```

Ranking Data

Using RANK

General Example of Rank



```
SELECT *
    ,ROW_NUMBER() OVER (ORDER BY Sales DESC) AS ROW_NUMBER
    ,RANK() OVER (ORDER BY Sales DESC) AS RANK
    ,DENSE_RANK() OVER (ORDER BY Sales DESC) AS DENSE_RANK
    ,NTILE(3) OVER(ORDER BY Sales DESC) AS NTILE
FROM Example_Ranking
ORDER BY Sales DESC;
```

100 %

Results | Messages

	ID	Product	Product_Group	Sales	ROW_NUMBER	RANK	DENSE_RANK	NTILE
1	3	Wardrobe	Kitchen	410	1	1	1	1
2	6	Wash Machine	Bathroom	400	2	2	2	1
3	4	Bed	Bedroom	200	3	3	3	2
4	2	Table	Kitchen	200	4	3	3	2
5	1	Chair	Kitchen	110	5	5	4	3
6	5	Bath	Bathroom	100	6	6	5	3

The `RANK` function provides a rank for each row within a partition of a result set, with the same rank assigned to rows with identical values. This can be useful for identifying top performers in groups or similar use cases.

```
SELECT
    employee_id,
    department_id,
    salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rank
FROM
    employees;
```

Using DENSE_RANK

`DENSE_RANK` works similarly to `RANK`, but it does not leave gaps in the ranking sequence. Consecutive rows with identical values receive the same rank, and the next rank is incremented by 1.

```
SELECT
    employee_id,
    department_id,
    salary,
    DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS
dense_rank
FROM   employees;
```

Combining Query Results

Using UNION

The `UNION` operator combines the results of two or more SELECT statements, removing duplicate rows.

```
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

Using INTERSECT

The `INTERSECT` operator returns the intersection of two or more SELECT statements, i.e., rows that are present in all queries.

```
SELECT column1, column2 FROM table1
INTERSECT
SELECT column1, column2 FROM table2;
```

Using EXCEPT

The `EXCEPT` operator returns the difference between two SELECT statements, i.e., rows from the first query that are not in the second query.

```
SELECT column1, column2 FROM table1
EXCEPT
SELECT column1, column2 FROM table2;
```

This manual provides a comprehensive guide to working with window functions in SQL. By mastering these techniques, you can perform sophisticated data analysis and efficiently manage database records.