

OLAP

OLAP functions

- OLAP functions are similar to aggregate functions.
 - Aggregate functions will return only one value.
 - OLAP function will provide the individual rows in addition to the aggregates.
- Aggregation functions can be:
- SUM , COUNT , AVG , MIN , MAX , MSUM , MAVG , MDIFF , CSUM
- RANK , DENSE_RANK , ROW_NUMBER , LAG , LEAD , FIRST_VALUE , LAST_VALUE

Syntax

```
analytical_function_name([column_name])  
OVER (  
  [PARTITION BY COLUMN1] [ORDER BY COLUMN2][DESC/ASC]  
  [ROWS BETWEEN n FOLLOWING|PRECEDING(start window)  
  AND m FOLLOWING|PRECEDING|CURRENT ROW)(end window)]  
)
```

Window features (optional)

- **PARTITION BY:** Perform analysis within sub categories.
 - **Example:** Calculate salary per department.
- **ORDER BY:** In which order should columns be processed.
- **ROWS BETWEEN:** Check within a window of rows. Must specify start/end (or `UNBOUNDED`).

Example

Consider the following Salary table.

EmployeeNo	Gross	Deduction	NetPay
101	40,000	4,000	36,000
102	80,000	6,000	74,000
103	90,000	7,000	83,000
104	75,000	5,000	70,000

Example (cont.)

- Calculate running total of NetPay

```
SELECT  
EmployeeNo, NetPay,  
SUM(Netpay)  
OVER(ORDER BY EmployeeNo  
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)  
as RunningSalary  
FROM Salary;
```

RANK

- **RANK** function orders the records based on the column provided.
- **RANK** function can also filter the number of records returned based on the rank.

```
RANK() OVER  
([PARTITION BY columnlist]  
[ORDER BY columnlist][DESC|ASC])
```

Example

Consider the following Employee table.

EmployeeNo	JoinedDate	DepartmentID	BirthDate
101	3/27/2005	1	1/5/1980
102	4/25/2007	2	3/5/1983
103	3/21/2007	2	4/1/1983
104	2/1/2008	2	11/6/1984
105	1/4/2008	3	12/1/1984

Example (cont.)

The following query orders the records of the employee table by `Joined Date` and assigns the ranking on `JoinedDate` .

```
SELECT EmployeeNo, JoinedDate, RANK(  
OVER(ORDER BY JoinedDate) as Seniority  
FROM Employee;
```

Example (cont.)

Running the above query we get:

EmployeeNo	JoinedDate	Seniority
101	2005-03-27	1
103	2007-03-21	2
102	2007-04-25	3
105	2008-01-04	4
104	2008-02-01	5

PARTITION BY

- **PARTITION BY** clause groups the data by the prescribed columns and performs the OLAP function within each group.
- Following is an example of the query that uses **PARTITION BY** clause.

```
SELECT EmployeeNo, JoinedDate, RANK()  
OVER(PARTITION BY DepartmentNo ORDER BY JoinedDate)  
as Seniority  
FROM Employee;
```

Result

EmployeeNo	DepartmentNo	JoinedDate	Seniority
------------	--------------	------------	-----------

101	1	2005-03-27	1
-----	---	------------	---

103	2	2007-03-21	1
-----	---	------------	---

102	2	2007-04-25	2
-----	---	------------	---

104	2	2008-02-01	3
-----	---	------------	---

105	3	2008-01-04	1
-----	---	------------	---

Filtering: **QUALIFY**

- Similar to **WHERE** or **HAVING**

```
SELECT EmployeeNo, JoinedDate  
, RANK() OVER(ORDER BY JoinedDate) as Seniority  
FROM Employee  
QUALIFY (  
    RANK() OVER(ORDER BY JoinedDate)  
) < 3;
```

Your turn!

Exercise - Warm up

- Use the script `04_0LAP.sql` to create a sample table.
- Write queries for:
 - i. Total sum of salary within department against each of employee of that department
 - ii. Cumulative salary within each department
 - iii. Total count of employees within department against each of employee of that department.
 - iv. Find employees whose total department salary is greater than 9000.

Exercise - RFM summary

- Load the `orders.csv` file.
- Create a macro that will return, for each customer and date:
 - The total money spent (up to that date).
 - The number of *repeat* purchases it has made (total purchases minus 1).