

# **Relational Databases and Datawarehousing – SQL**

Window Functions

# **WINDOW FUNCTIONS**

# Window functions: business case

- Often business managers want to compare current sales to previous sales
- Previous sales can be:
  - sales during previous month
  - average sales during last three months
  - last year's sales until current date (year-to-date)
- Window functions offer a solution to these kind of problems in a single, efficient SQL query
- Introduced in SQL: 2003

# OVER clause

- Results of a **SELECT** are partitioned
- Numbering, ordering and aggregate functions per partition
- The **OVER** clause creates partitions and ordering
- The partition behaves as a window that shifts over the data
- The **OVER** clause can be used with standard aggregate functions (sum, avg, ...) or specific window functions (rank, lag,...)

## Example: Running Total

- Make an overview of the UnitsInStock per Category and per Product

```
SELECT CategoryID, ProductID, UnitsInStock
FROM Products
order by CategoryID, ProductID
```

	CategoryID	ProductID	UnitsInStock
1	1	1	39
2	1	2	17
3	1	24	20
4	1	34	111
5	1	35	20
6	1	38	17
7	1	39	69
8	1	43	17
9	1	67	52
10	1	70	15
11	1	75	125
12	1	76	57
13	2	3	13
14	2	4	53
15	2	5	0
16	2	6	120
17	2	8	6
18	2	15	39
19	2	44	27
20	2	61	113
21	2	63	24
22	2	65	76
23	2	66	4

## Example: Running Total

- Add an extra column to calculate the running total of UnitsInStock per Category
- Solution 1 → correlated subquery

```
SELECT CategoryID, ProductID, UnitsInStock,
(SELECT SUM(UnitsInStock)
 FROM Products
 WHERE CategoryID = p.CategoryID
    and ProductID <= p.ProductID) TotalUnitsInStockPerCategory
FROM Products p
order by CategoryID, ProductID;
```

	CategoryID	ProductID	UnitsInStock	TotalUnitsInSt...
1	1	1	39	39
2	1	2	17	56
3	1	24	20	76
4	1	34	111	187
5	1	35	20	207
6	1	38	17	224
7	1	39	69	293
8	1	43	17	310
9	1	67	52	362
10	1	70	15	377
11	1	75	125	502
12	1	76	57	559
13	2	3	13	13
14	2	4	53	66
15	2	5	0	66
16	2	6	120	186
17	2	8	6	192
18	2	15	39	231
19	2	44	27	258
20	2	61	113	371
21	2	63	24	395
22	2	65	76	471
23	2	66	4	475

- Using a correlated subquery is very inefficient as for each line the complete sum is recalculated:  $O(n^2)$

## Example: Running Total

- Add an extra column to calculate the running total of UnitsInStock per Category
- Solution 2 → OVER clause
  - simpler + more efficient
  - The sum is calculated for each partition

```
SELECT CategoryID, ProductID, UnitsInStock,
SUM(UnitsInStock) OVER (PARTITION BY CategoryID ORDER BY ProductID)
TotalUnitsInStockPerCategory
FROM Products
```

	CategoryID	ProductID	UnitsInStock	TotalUnitsInSt...
1	1	1	39	39
2	1	2	17	56
3	1	24	20	76
4	1	34	111	187
5	1	35	20	207
6	1	38	17	224
7	1	39	69	293
8	1	43	17	310
9	1	67	52	362
10	1	70	15	377
11	1	75	125	502
12	1	76	57	559
13	2	3	13	13
14	2	4	53	66
15	2	5	0	66
16	2	6	120	186
17	2	8	6	192
18	2	15	39	231
19	2	44	27	258
20	2	61	113	371
21	2	63	24	395
22	2	65	76	471
23	2	66	4	475
	-	--	--	---

## Window functions – RANGE

- Real meaning of window functions: apply to a window that shifts over the result set
- The previous query works with the default window: start of resultset to current row

```
SELECT CategoryID, ProductID, UnitsInStock,  
SUM(UnitsInStock) OVER (PARTITION BY CategoryID ORDER BY ProductID) TotalUnitsInStockPerCategory  
FROM Products
```

-- The previous query is the shorter version of the following query. Exactly the same resultset!

```
SELECT CategoryID, ProductID, UnitsInStock,  
SUM(UnitsInStock) OVER (PARTITION BY CategoryID ORDER BY ProductID  
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) TotalUnitsInStockPerCategory  
FROM Products
```



# Window functions

- With RANGE you have three valid options:
  - RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  - RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
  - RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
- PARTITION is optional, ORDER BY is mandatory

# Window functions

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

```
SELECT CategoryID, ProductID, UnitsInStock,  
SUM(UnitsInStock) OVER (PARTITION BY CategoryID ORDER BY ProductID  
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) TotalUnitsInStockPerCategory  
FROM Products
```

	CategoryID	ProductID	UnitsInStock	TotalUnitsInSt...
1	1	1	39	39
2	1	2	17	56
3	1	24	20	76
4	1	34	111	187
5	1	35	20	207
6	1	38	17	224
7	1	39	69	293
8	1	43	17	310
9	1	67	52	362
10	1	70	15	377
11	1	75	125	502
12	1	76	57	559
13	2	3	13	13
14	2	4	53	66
15	2	5	0	66
16	2	6	120	186
17	2	8	6	192
18	2	15	39	231
19	2	44	27	258
20	2	61	113	371
21	2	63	24	395
22	2	65	76	471
23	2	66	4	475

# Window functions

- RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

```
SELECT CategoryID, ProductID, UnitsInStock,  
SUM(UnitsInStock) OVER (PARTITION BY CategoryID ORDER BY ProductID  
RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) TotalUnitsInStockPerCategory  
FROM Products
```

	CategoryID	ProductID	UnitsInStock	TotalUnitsInStockPerCategory
1	1	1	39	559
2	1	2	17	520
3	1	24	20	503
4	1	34	111	483
5	1	35	20	372
6	1	38	17	352
7	1	39	69	335
8	1	43	17	266
9	1	67	52	249
10	1	70	15	197
11	1	75	125	182
		76	57	57
		3	13	507
		4	53	494
15	2	5	0	441
16	2	6	120	441
17	2	8	6	321
18	2	15	39	315
19	2	44	27	276
20	2	61	113	249
21	2	63	24	136
22	2	65	76	112
23	2	66	4	36

# Window functions

- RANGE BETWEEN  
UNBOUNDED PRECEDING AND  
UNBOUNDED FOLLOWING

```
SELECT CategoryID, ProductID, UnitsInStock,  
SUM(UnitsInStock) OVER (PARTITION BY CategoryID ORDER BY ProductID  
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)  
TotalUnitsInStockPerCategory  
FROM Products
```

	CategoryID	ProductID	UnitsInStock	TotalUnits...
1	1	1	39	559
2	1	2	17	559
3	1	24	20	559
4	1	34	111	559
5	1	35	20	559
6	1	38	17	559
7	1	39	69	559
8	1	43	17	559
9	1	67	52	559
10	1	70	15	559
11	1	75	125	559
12	1	76	57	559
13	2	3	13	507
14	2	4	53	507
15	2	5	0	507
16	2	6	120	507
17	2	8	6	507
18	2	15	39	507
19	2	44	27	507
20	2	61	113	507
21	2	63	24	507
22	2	65	76	507
23	2	66	4	507

## Window functions - ROWS

- When you use RANGE, the current row is compared to other rows and grouped based on the ORDER BY predicate.
- This is not always desirable. You might actually want a physical offset.
- In this scenario, you would specify ROWS instead of RANGE. This gives you three options in addition to the three options enumerated previously:
  - ROWS BETWEEN N PRECEDING AND CURRENT ROW
  - ROWS BETWEEN CURRENT ROW AND N FOLLOWING
  - ROWS BETWEEN N PRECEDING AND N FOLLOWING

# Example

- Make an overview of the salary per employee and the average salary of this employee and the 2 employees preceding him

```
SELECT EmployeeID, FirstName + ' ' + LastName As FullName, Salary,  
AVG(Salary) OVER (ORDER BY Salary DESC ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) As AvgSalary2Preceding  
FROM Employees
```

	EmployeeID	FullName	Salary	AvgSalary2Preceding
1	2	Andrew Fuller	90000.00	90000.000000
2	5	Steven Buchanan	55000.00	72500.000000
3	8	Laura Callahan	51000.00	65333.333333
4	1	Nancy Davolio	48000.00	51333.333333
5	7	Robert King	42000.00	47000.000000
6	4	Margaret Peacock	40000.00	43333.333333
7	9	Anne Dodsworth	40000.00	40666.666666
8	3	Janet Leverling	36000.00	38666.666666
9	6	Michael Suyama	35000.00	37000.000000

# Example

- Make an overview of the salary per employee and the average salary of this employee and the 2 employees following him

```
SELECT EmployeeID, FirstName + ' ' + LastName As FullName, Salary,  
AVG(Salary) OVER (ORDER BY Salary DESC ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING) As AvgSalary2Following  
FROM Employees
```

	EmployeeID	FullName	Salary	AvgSalary2Following
1	2	Andrew Fuller	90000.00	65333.333333
2	5	Steven Buchanan	55000.00	51333.333333
3	8	Laura Callahan	51000.00	47000.000000
4	1	Nancy Davolio	48000.00	43333.333333
5	7	Robert King	42000.00	40666.666666
6	4	Margaret Peacock	40000.00	38666.666666
7	9	Anne Dodsworth	40000.00	37000.000000
8	3	Janet Leverling	36000.00	35500.000000
9	6	Michael Suyama	35000.00	35000.000000

# Example

- Make an overview of the salary per employee and the average salary of this employee and the employee preceding and following him

```
SELECT EmployeeID, FirstName + ' ' + LastName As FullName, Salary,  
AVG(Salary) (ORDER BY Salary DESC ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) As AvgSalary1Preceding1Following  
FROM Employees
```

	EmployeeID	FullName	Salary	AvgSalary1Preceding1Following
1	2	Andrew Fuller	90000.00	72500.000000
2	5	Steven Buchanan	55000.00	65333.333333
3	8	Laura Callahan	51000.00	51333.333333
4	1	Nancy Davolio	48000.00	47000.000000
5	7	Robert King	42000.00	43333.333333
6	4	Margaret Peacock	40000.00	40666.666666
7	9	Anne Dodsworth	40000.00	38666.666666
8	3	Janet Leverling	36000.00	37000.000000
9	6	Michael Suyama	35000.00	35500.000000



# Window functions

- `ROW_NUMBER()` numbers the output of a result set. More specifically, returns the sequential number of a row within a partition of a result set, starting at 1 for the first row in each partition.
- `RANK()` returns the rank of each row within the partition of a result set. The rank of a row is one plus the number of ranks that come before the row in question.

# Window functions

- ROW\_NUMBER and RANK are similar. ROW\_NUMBER numbers all rows sequentially (for example 1, 2, 3, 4, 5). RANK provides the same numeric value for ties (for example 1, 2, 2, 4, 5).
- DENSE\_RANK() returns the rank of each row within the partition of a result set, with no gaps in the ranking values (for example 1, 2, 2, 3, 4).
- PCT\_RANK() shows the ranking on a scale from 0 - 1

# Window functions

- Example: Give ROW\_NUMBER / RANK / DENSE\_RANK / PERCENT\_RANK for each employee based on his salary

```
SELECT EmployeeID, FirstName + ' ' + LastName As 'Full Name', Title, Salary,
ROW_NUMBER() OVER (ORDER BY Salary DESC) As 'ROW_NUMBER',
RANK() OVER (ORDER BY Salary DESC) AS 'RANK',
DENSE_RANK() OVER (ORDER BY Salary DESC) AS 'DENSE_RANK',
PERCENT_RANK() OVER (ORDER BY Salary DESC) AS 'PERCENT_RANK'
FROM Employees
```

	EmployeeID	Full Name	Title	Salary	ROW_NUMBER	RANK	DENSE_RANK	PERCENT_RANK
1	2	Andrew Fuller	Vice President, Sales	90000.00	1	1	1	0
2	5	Steven Buchanan	Sales Manager	55000.00	2	2	2	0,125
3	8	Laura Callahan	Inside Sales Coordinator	51000.00	3	3	3	0,25
4	1	Nancy Davolio	Sales Representative	48000.00	4	4	4	0,375
5	7	Robert King	Sales Representative	42000.00	5	5	5	0,5
6	4	Margaret Peacock	Sales Representative	40000.00	6	6	6	0,625
7	9	Anne Dodsworth	Sales Representative	40000.00	7	6	6	0,625
8	3	Janet Leverling	Sales Representative	36000.00	8	8	7	0,875
9	6	Michael Suyama	Sales Representative	35000.00	9	9	8	1

# Window functions

- Example: Give ROW\_NUMBER / RANK / DENSE\_RANK / PERCENT\_RANK per title for each employee based on his salary

```
SELECT EmployeeID, FirstName + ' ' + LastName As 'Full Name', Title, Salary,
ROW_NUMBER() OVER (PARTITION BY Title ORDER BY Salary DESC) As 'ROW_NUMBER',
RANK() OVER (PARTITION BY Title ORDER BY Salary DESC) AS 'RANK',
DENSE_RANK() OVER (PARTITION BY Title ORDER BY Salary DESC) AS 'DENSE_RANK',
PERCENT_RANK() OVER (PARTITION BY Title ORDER BY Salary DESC) AS 'PERCENT_RANK'
FROM Employees
```

	EmployeeID	Full Name	Title	Salary	ROW_NUMBER	RANK	DENSE_RANK	PERCENT_RANK
1	8	Laura Callahan	Inside Sales Coordinator	51000.00	1	1	1	0
2	5	Steven Buchanan	Sales Manager	55000.00	1	1	1	0
3	1	Nancy Davolio	Sales Representative	48000.00	1	1	1	0
4	7	Robert King	Sales Representative	42000.00	2	2	2	0.2
5	4	Margaret Peacock	Sales Representative	40000.00	3	3	3	0.4
6	9	Anne Dodsworth	Sales Representative	40000.00	4	3	3	0.4
7	3	Janet Leverling	Sales Representative	36000.00	5	5	4	0.8
8	6	Michael Suyama	Sales Representative	35000.00	6	6	5	1
9	2	Andrew Fuller	Vice President, Sales	90000.00	1	1	1	0

## LAG and LEAD

- LAG refers to the previous line. This is short for LAG(..., 1)
- LAG(..., 2) refers to the line before the previous line, ...
- LEAD refers to the next line. This is short for LEAD(..., 1)
- LEAD(..., 2) refers to the line after the next line, ...

# LAG

- Example: Calculate for each employee the percentage difference between this employee and the employee preceding him

```
SELECT EmployeeID, FirstName + ' ' + LastName, Salary,  
FORMAT((Salary - LAG(Salary) OVER (ORDER BY Salary DESC)) / Salary, 'P') As EarnsLessThanPreceding  
FROM Employees
```

	EmployeeID	(No column name)	Salary	EarnsLessThanPreceding
1	2	Andrew Fuller	90000.00	NULL
2	5	Steven Buchanan	55000.00	-63.64%
3	8	Laura Callahan	51000.00	-7.84%
4	1	Nancy Davolio	48000.00	-6.25%
5	7	Robert King	42000.00	-14.29%
6	4	Margaret Peacock	40000.00	-5.00%
7	9	Anne Dodsworth	40000.00	0.00%
8	3	Janet Leverling	36000.00	-11.11%
9	6	Michael Suyama	35000.00	-2.86%

# LEAD

- Example: Calculate for each employee the percentage difference between this employee and the employee following him

```
SELECT EmployeeID, FirstName + ' ' + LastName, Salary,  
FORMAT((Salary - LEAD(Salary) OVER (ORDER BY Salary DESC)) / Salary, 'P') As EarnsMoreThanFollowing  
FROM Employees
```

	EmployeeID	(No column name)	Salary	EarnsMoreThanFollowing
1	2	Andrew Fuller	90000.00	38.89%
2	5	Steven Buchanan	55000.00	7.27%
3	8	Laura Callahan	51000.00	5.88%
4	1	Nancy Davolio	48000.00	12.50%
5	7	Robert King	42000.00	4.76%
6	4	Margaret Peacock	40000.00	0.00%
7	9	Anne Dodsworth	40000.00	10.00%
8	3	Janet Leverling	36000.00	2.78%
9	6	Michael Suyama	35000.00	NULL

# Exercises

```
-- Exercise 1
-- Create the following overview in which each customer gets a sequential number.
-- The companynames are sorted alphabetically
-- The number is reset when the country changes
/*
country      rownum      CompanyName
Argentina    1          Cactus Comidas para llevar
Argentina    2          Océano Atlántico Ltda.
Argentina    3          Rancho grande
Austria      1          Ernst Handel
Austria      2          Piccolo und mehr
Belgium      1          Maison Dewey
Belgium      2          Suprêmes délices
Brazil       1          Comércio Mineiro
Brazil       2          Família Arquibaldo
Brazil       3          Gourmet Lanchonetes
Brazil       4          Hanari Carnes
...
*/
```



# Exercises

```
-- Exercise 2
-- First create an overview that shows for each productid the amount sold per year

-- Now create an overview that shows for each productid the amount sold per year and for the previous year.
/*
1      2016   125   NULL
1      2017   304   125
1      2018   399   304
2      2016   226   NULL
2      2017   435   226
2      2018   396   435
3      2016    30   NULL
3      2017   190    30
3      2018   108   190
...
*/
```

# Exercises

```
-- Use a CTE and the previous SQL Query to calculate the year over year performance for each productid.  
-- If the amountPreviousYear is NULL, then the year over year performance becomes N/A.
```

```
/*  
1      2016   125    NULL   N/A  
1      2017   304    125    143.20%  
1      2018   399    304    31.25%  
2      2016   226    NULL   N/A  
2      2017   435    226    92.48%  
2      2018   396    435    -8.97%  
3      2016   30     NULL   N/A  
3      2017   190    30     533.33%  
3      2018   108    190    -43.16%  
...  
*/
```

# Exercises

```
-- Exercise 3
-- Which is the most popular shipper
-- Step 1: Use a CTE and add DENSE_RANK
-- Step 2: FILTER on DENSE_RANK = 1

-- ShipperID CompanyName ShipVia      NumberOfOrders    DENSE_RANK
-- 2 United Package 2 326 1
```

# Exercises

```
-- Exercise 4
-- Which is the TOP 3 of countries in which most customers live?
-- Step 1: Use a CTE and add DENSE_RANK
-- Step 2: FILTER on DENSE_RANK
```

--Country	NumberOfCustomers	DENSE_RANK
--USA 13	1	
--France 11	2	
--Germany 11	2	
--Brazil 9	3	

# Exercises

```
-- Exercise 5
-- Imagine there is a bonussystem for all the employees: the best employee gets 10 000EUR bonus, the second
one 5000 EUR, the third one 2500 EUR, ...
-- Let's calculate the bonus for each employee, based on the revenue per year per employee
-- First create an overview of the revenue (unitprice * quantity) per year per employeeid
/*
1      2016   38789,00
1      2017   97533,58
1      2018   65821,13
2      2016   22834,70
2      2017   74958,60
2      2018   79955,96
3      2016   19231,80
3      2017   111788,61
3      2018   82030,89
4      2016   53114,80
4      2017   139477,70
4      2018   57594,95
...
*/
```

# Exercises

```
-- Now add a ranking per year per employeeid
/*
4      2016   53114,80      1
1      2016   38789,00      2
8      2016   23161,40      3
2      2016   22834,70      4
5      2016   21965,20      5
3      2016   19231,80      6
7      2016   18104,80      7
6      2016   17731,10      8
9      2016   11365,70      9
...
*/
```

# Exercises

```
-- Imagine there is a bonussystem for all the employees: the best employee gets 10 000EUR bonus, the second one 5000 EUR, the third one 2500 EUR, ...
```

```
/*  
4      2016  53114,80      10000  
1      2016  38789,00      5000  
8      2016  23161,40      3333  
2      2016  22834,70      2500  
5      2016  21965,20      2000  
3      2016  19231,80      1666  
7      2016  18104,80      1428  
6      2016  17731,10      1250  
9      2016  11365,70      1111  
...  
*/
```

# Exercises

```
-- Exercise 6: Calculate for each month the percentage difference between the revenue for this month and the
previous month
/*
2016  7      30192,10      NULL  NULL
2016  8      26609,40      30192,10    -11.86%
2016  9      27636,00      26609,40     3.85%
2016 10      41203,60      27636,00    49.09%
2016 11      49704,00      41203,60    20.63%
2016 12      50953,40      49704,00     2.51%
2017  1      66692,80      50953,40    30.88%
2017  2      41207,20      66692,80   -38.21%
...
*/
-- Step 1: calculate the revenue per year and per month
-- Step 2: Add an extra column for each row with the revenue of the previous month
-- Step 3: Calculate the percentage difference between this month and the previous month
```



# Exercises

```
-- Exercise 3
-- Which is the most popular shipper
-- Step 1: Use a CTE and add DENSE_RANK
-- Step 2: FILTER on DENSE_RANK = 1

-- ShipperID CompanyName ShipVia      NumberOfOrders    DENSE_RANK
-- 2 United Package 2 326 1

WITH cte AS
(SELECT ShipVia, COUNT(OrderID) As NumberOfOrders, DENSE_RANK() OVER (ORDER BY COUNT(OrderID) DESC) AS
'DENSE_RANK'
FROM Orders
GROUP BY ShipVia)

SELECT s.ShipperID, s.CompanyName, c.*
FROM Shippers s JOIN cte c ON s.ShipperID = c.ShipVia
WHERE DENSE_RANK = 1
```

# Exercises

```
-- Exercise 4
-- Which is the TOP 3 of countries in which most customers live?
-- Step 1: Use a CTE and add DENSE_RANK
-- Step 2: FILTER on DENSE_RANK
```

--Country	NumberOfCustomers	DENSE_RANK
--USA 13	1	
--France 11	2	
--Germany 11	2	
--Brazil 9	3	

```
WITH cte
AS (
SELECT Country, COUNT(customerID) As NumberOfCustomers, DENSE_RANK() OVER (ORDER BY COUNT(customerID) DESC) AS
'DENSE_RANK'
FROM customers
GROUP BY Country)

SELECT * FROM cte WHERE DENSE_RANK < 4
```