SQL: Recursive Queries and OLAP Queries

Chapter content:

- SQL Recursive queries
- SQL (OLAP) GROUP BY Extensions
- GROUPING functions
- Analytical functions

Table des matières

Database instance for our hierarchical queries:

Employees

EMPLOYEE_ID	LASTNAME	REPORTS_TO	TITLE
1	Davolio	2	Sales Representative
2	Fuller		Vice President, Sales
3	Leverling	2	Sales Representative
4	Peacock	2	Sales Representative
5	Buchanan	2	Sales Manager
6	Suyama	5	Sales Representative
7	King	5	Sales Representative
8	Callahan	2	Inside Sales Coordinator
9	Dodsworth	5	Sales Representative

WITH clause/CTE

CTE=Common Table Expressions

basic syntax

```
WITH subquery_name AS
(<subquery_expressions>)
   <query using subquery_name>
;
```

Supported in Oracle, SQL Server, IBM, MariaDB, PostgresQL.

Query expression defined once can be used multiple times:

- avoid reevaluation overhead
- may avoid incoherent results derived from multiple executions returning different answers
- enable recursive queries

WITH clause (Oracle)

more general syntax

WITH *subquery_name*$_1$ AS (*expressions*$_1$) *search_clause*$_1$ *cycle_clause*$_1$,
...
subquery_name$_n$ AS (*expressions*$_n$) *search_clause*$_n$ *cycle_clause*$_n$,
query_expression
;

Column aliases can be introduced after *subquery_name*.

```
WITH
  reports_to_2 (eid, mgr_id, reportLevel) AS
  (
    SELECT employee_id, reports_to, 0 reportLevel
    FROM employees
    WHERE employee_id = 2
  UNION ALL
    SELECT e.employee_id, e.reports_to, reportLevel+1
    FROM employees e, reports_to_2 r
    WHERE r.eid = e.reports_to
  )
SELECT eid, mgr_id, reportLevel
FROM reports_to_2
ORDER BY reportLevel, eid;
```

recursive query

WITH clause (Oracle) (2)

Result of the query:

EID	MGR_ID	REPORTLEVEL
2		0
1	2	1
3	2	1
4	2	1
5	2	1
8	2	1
6	5	2
7	5	2
9	5	2

Table des matières

Querying the DW in the relational model

- Analyst formulates OLAP queries on OLAP client.

- The server must return the answer to queries efficiently.
  $\hookrightarrow$ in the case of ROLAP: simulate dimensional operations.

$\Longrightarrow$ query language for relational data: SQL.

Querying the DW in the relational model

- Analyst formulates OLAP queries on OLAP client.

- The server must return the answer to queries efficiently.
  $\hookrightarrow$ in the case of ROLAP: simulate dimensional operations.

$\Longrightarrow$ query language for relational data: SQL.

No specific query language for OLAP but SQL-99 extends SQL-92 with (among others) aggregation functions for OLAP.

OLAP with SQL-92

Why an extension?

Typical OLAP operations:

- DRILL-DOWN: join facts and dimension, GROUP BY
- ROLL-UP: join, GROUP BY
- SLICE, DICE: WHERE clause
- DRILL ACROSS: JOIN, GROUP BY

In theory can be expressed in SQL-92.

OLAP with SQL-92

pivot table for Sales=
all cube aggregations

	Beverages	Produce	Condiments	Total by Quarters
Q1	21	10	18	49
Q2	27	14	11	52
Q3	26	12	35	73
Q4	14	20	47	81
Total by Product	88	56	111	255

Sales

Quarter Key	Category Key	Amount
Q1	B	21
Q1	P	10
Q1	C	18
Q2	B	27
Q2	P	14
Q2	C	11
Q3	B	26
Q3	P	12
Q3	C	35
Q4	B	14
Q4	P	20
Q4	C	47

OLAP with SQL-92

pivot table for Sales=
all cube aggregations

	Beverages	Produce	Condiments	Total by Quarters
Q1	21	10	18	49
Q2	27	14	11	52
Q3	26	12	35	73
Q4	14	20	47	81
Total by Product	88	56	111	255

SQL query to compute all aggregations from Sales?

Sales aggregations

Quarter Key	Category Key	Amount
Q1	B	21
Q1	P	10
Q1	C	18
Q1	NULL	49
Q2	B	27
Q2	P	14
Q2	C	11
Q2	NULL	52
Q3	B	26
Q3	P	12
Q3	C	35
Q3	NULL	73
Q4	B	14
Q4	P	20
Q4	C	47
Q4	NULL	81
NULL	B	81
NULL	P	56
NULL	C	111
NULL	NULL	255

Shortcomings of SQL-92 for OLAP

Multiple aggregations using UNIONs of GROUP BY:

- painful query formulation
- inefficient

How many granularities for n dimensions (detailed$\rightarrow$ Top)?

Also hard to express:

- compare the aggregation values from one year to the other
- moving averages
- cumulative aggregations, etc

SQL 99 extensions

Multiple aggregations groupings

Aim: make OLAP queries easier and faster.

Additional aggregation commands from SQL 99:

- CUBE: all grouping combinations
- ROLLUP: combinations along a hierarchy
- GROUPING SETS: specify explicitly a list of grouping combinations requested

Supported in Oracle, SQL Server, IBM DB2, PostgresQL, MariaDB, *but not MYSQL*.

Solution for multiple aggregations in OLAP. For the other issues: additional aggregate functions...

CUBE

```
SELECT city,month, sum(quantity) AS Quantity
FROM Facts F, Time T, Product P, Location L
WHERE P.PID = F.PID
  AND T.TID = F.TID
  AND L.LID = F.LID
  AND P.category = 'DVD'
  AND T.quarter = 'Q1 2010'
GROUP BY CUBE (month, city)
;
```

CITY	MONTH	QUANTITY
		226
Lyon		32
Paris		85
Berlin		67
Stuttgart		42
	fev10	77
Lyon	fev10	12
Paris	fev10	25
Berlin	fev10	25
Stuttgart	fev10	15
	jan10	70
Lyon	jan10	10
Paris	jan10	30
Berlin	jan10	20
Stuttgart	jan10	10
	mar10	79
Lyon	mar10	10
Paris	mar10	30
Berlin	mar10	22
Stuttgart	mar10	17

ROLLUP

```
SELECT quarter, month, SUM(sales) AS sales
FROM Facts F, Time T, Product P
WHERE P.PID = F.PID
  AND T.TID = F.TID
GROUP BY ROLLUP(quarter,month)
;
```

QUARTER	MONTH	SALES
Q1 2010	fev10	422
Q1 2010	jan10	675
Q1 2010	mar10	400
Q1 2010		1497
Q4 2010	dec10	253
Q4 2010		253
		1750

GROUPING SETS

Syntax (basic)

$$\cdots \text{GROUP BY GROUPING SETS } \Big(\big(a_{1,1}, a_{1,2}, \ldots, a_{1,n_1} \big), \big(a_{2,1}, a_{2,2}, \ldots, a_{2,n_2} \big), \vdots \big(a_{k,1}, a_{k,2}, \ldots, a_{k,n_k} \big), \Big);$$

Specifies each requested grouping with a sequence $(a_{i,1}, a_{i,2}, \ldots, a_{i,n_i})$.

Equivalent to UNION ALL of GROUP BY queries with NULLs.

GROUPING SET

GROUPING SET

```
SELECT quarter, type, city,
  SUM(sales) AS sales
FROM Facts F, Time T,
    Product P, Location L
WHERE P.PID = F.PID
  AND T.TID = F.TID
  AND L.LID = F.LID
  AND T.year = '2010'
GROUP BY GROUPING SETS (
  (quarter, type),
  (quarter, city)
);
```

QUARTER	TYPE	CITY	SALES
Q1 2010	Media		1200
Q1 2010	Livres		297
Q4 2010	Media		253
Q1 2010		Berlin	385
Q1 2010		Paris	385
Q1 2010		Stuttgart	372
Q1 2010		Lyon	355
Q4 2010		Lyon	103
Q4 2010		Berlin	150

GROUPING SETS vs ROLLUP/CUBE

GROUPING SETS can express ROLLUP, CUBE, GROUP BY(a,b).

```
SELECT T.year_id, T.month_id, T.day_id, SUM(amount)
FROM sales S, Time T
WHERE T.day_id= S.day_id
GROUP BY ROLLUP(T.year_id, T.month_id, T.day_id);
```

Express the above query with GROUPING SETS

```
GROUP BY GROUPING SETS(
(T.year_id, T.month_id, T.day_id),
(y,m),
(y),()
);
```

GROUP BY year_id,month_id with GROUPING SETS?

GROUPING SETS with ROLLUP/CUBE

Complete syntax

```
···GROUP BY GROUPING SETS (
    <attribute sequence_1>,
    <attribute sequence_2>,
    ...
    <attribute sequence_i>,
    ROLLUP <attribute sequence_1>,
    ...
    ROLLUP <attribute sequence_j>,
    CUBE <attribute sequence_1>,
    ...
    CUBE <attribute sequence_k>
);
```

Concatenated groupings

Product of groups

GROUP BY X, Y=GROUP BY GROUPING SETS (
$\{\{ (a_1, \ldots, a_i, b_1, \ldots, b_j) | < a_1, \ldots a_i > \in X, < b_1, \ldots b_j > \in Y\}\}$
);

X, Y: lists of attributes, GROUPING SETS and ROLLUP.

Semantics of GROUP BY is cartesian product of its groups.

Composite columns (b,c)

A collection of columns treated as a unit (i.e., as a single attribute) in the groupings.

Composite column

ROLLUP(a,(b,c)) is equivalent to GROUPING SETS((a,b,c),(a),())

Examples of multiple groups and composite columns

Identify the groups involved

- GROUP BY a, GROUPING SETS ((b), (c,d))

  (a,b) (a,c,d)

- GROUP BY a, ROLLUP (a, b)

  (a,b) (a) (a)

- GROUP BY a, GROUPING SETS ((b), (c), ())

  (a,b) (a,c)

- GROUP BY GROUPING SETS((a,b), (b,c)), GROUPING SETS((d,e),(d),())

  (a,b,d,e) (a,b,d) (a,b) (b,c,d,e) (b,c,d) (b,c)

- GROUP BY CUBE((a,b),(b,c))

  () (a,b) (b,c) (a,b,c)

Table des matières

SQL 99 extensions

Identifying row provenance (groupings)

Additional aggregation commands from SQL 99:

- GROUPING(a)†‡§: is the row aggregated on attribute a
- GROUPING_ID($a_1, \ldots, a_n$)†‡: combines multiple GROUPING(a_i)
- GROUP_ID()†: identify redundant groupings from the query

Supported in Oracle†, SQL Server‡, IBM DB2§.

NULLs identification with GROUPING

GROUPING(*a*)

- returns 1 when the row is a (sub)total for attribute *a*, i.e., when *a* is not part of the group for this row.
- returns 0 otherwise (stored NULL, or non NULL value).

```
SELECT quarter, DECODE(GROUPING(city),1,'multicity',city) city_desc,
  SUM(sales) sales, GROUPING(city), GROUPING(quarter)
FROM Facts F, Time T, Location L
WHERE T.TID = F.TID AND L.LID = F.LID AND L.country='France'
GROUP BY GROUPING SETS (
  (quarter),
  (quarter, city)
);
```

QUARTER	CITY_DESC	SALES	GROUPING(CITY)	GROUPING(QUARTER)
Q1 2010	Lyon	355	0	0
Q1 2010	Paris	385	0	0
Q1 2010	multicity	740	1	0
Q4 2010	Lyon	103	0	0
Q4 2010	multicity	103	1	0

GROUPING_ID

GROUPING_ID($a_1, \ldots, a_n$)

- identifies which attributes belong to the grouping
- returns the base 10 value of the bitvector: GROUPING(a_1) ... GROUPING(a_n)

```
SELECT quarter, city,country, SUM(sales) sales,
  GROUPING_ID(city,quarter,country) GID,
  GROUPING(city) GCT, GROUPING(quarter) GQT , GROUPING(country) GCO
FROM Facts F, Time T, Location L
WHERE T.TID = F.TID AND L.LID = F.LID AND L.country='France'
GROUP BY GROUPING SETS ((country),(quarter, city),(country,city));
```

QUARTER	CITY	COUNTRY	SALES	GID	GCT	GQT	GCO
Q4 2010	Lyon		103	1	0	0	1
Q1 2010	Lyon		355	1	0	0	1
Q1 2010	Paris		385	1	0	0	1
	Lyon	France	458	2	0	1	0
	Paris	France	385	2	0	1	0
		France	843	6	1	1	0

GROUP_ID

GROUP_ID()

- identifies duplicate groupings
- for each grouping, returns 0 for the first set of rows computed according to the grouping, 1 for the second, then 2...

```
SELECT quarter, month,
  SUM(sales) AS sales, GROUP_ID()
FROM Facts F, Time T,
  Product P, Location L
WHERE P.PID = F.PID
  AND T.TID = F.TID
  AND L.LID = F.LID
GROUP BY quarter, ROLLUP(quarter,month)
;
```

QUARTER	MONTH	SALES	GROUP_ID()
Q1 2010	fev10	422	0
Q1 2010	jan10	675	0
Q1 2010	mar10	400	0
Q4 2010	dec10	253	0
Q1 2010		1497	0
Q4 2010		253	0
Q1 2010		1497	1
Q4 2010		253	1

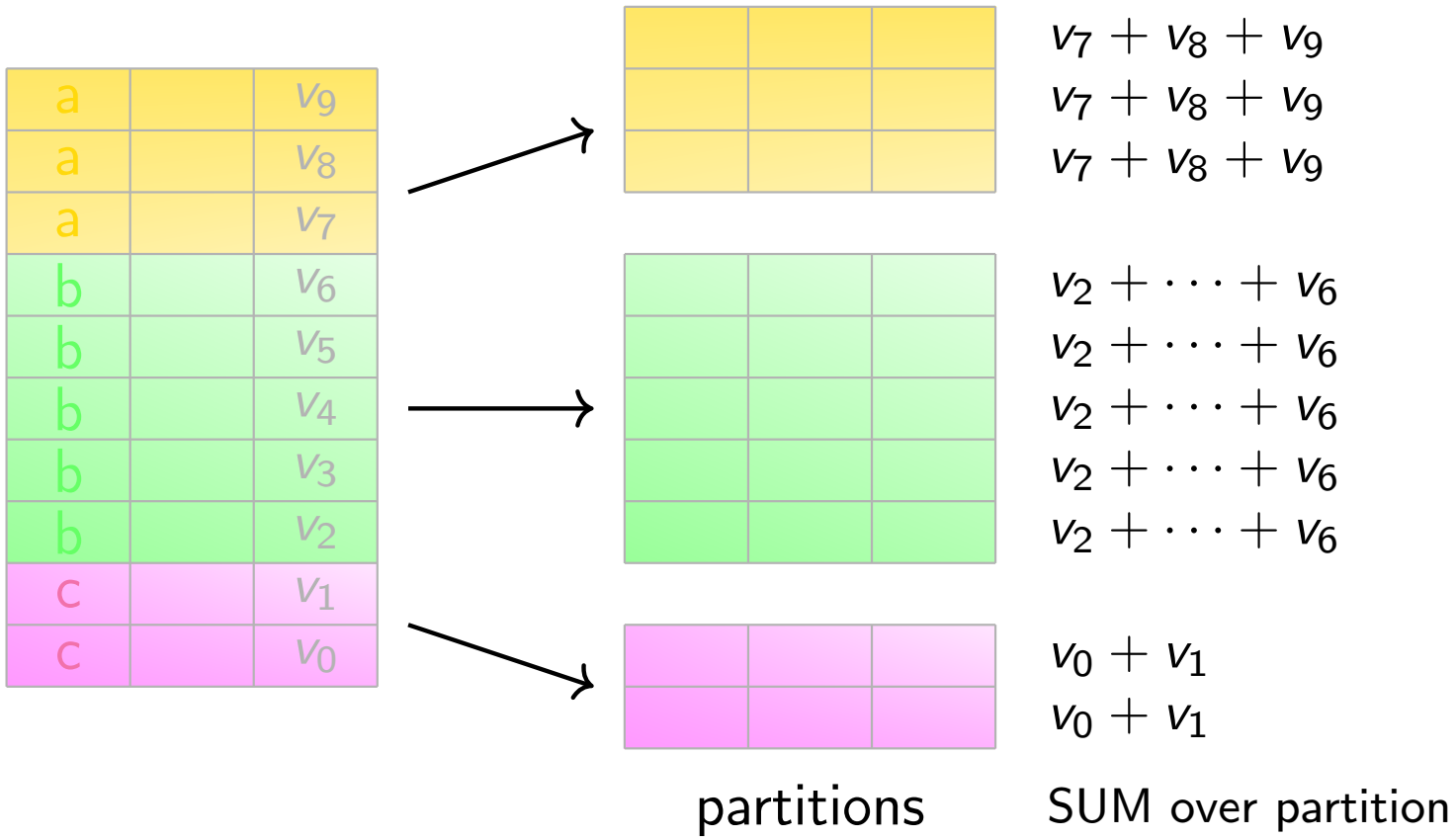
Eliminate duplicate groupings.

Table des matières

Window partitioning

Window partitioning

partitions input tuples, then aggregate independently each partition. Each original row is preserved; aggregated value comes as additional column.



Ex: add to each row the total sales (SUM) within each partition

Supported by Oracle, SQL Server, DB2, PostgresQL, MariaDB(dev) but not MySQL.

Window partitioning

window_function OVER ([PARTITION BY *attribute_sequence*])

partitions tuples, then aggregate independently each partition.

Contribution to sales per city and period

```
SELECT quarter, date, city, sales,
   SUM(sales) OVER(PARTITION BY quarter) AS SALES_QT
FROM Facts F, Time T, Product P, Location L
WHERE P.PID = F.PID
   AND T.TID = F.TID
   AND L.LID = F.LID;
```

QUARTER	DATE	CITY	SALES	SALES_QT
Q1 2010	01Jan10	Berlin	120	1497
Q1 2010	04Feb10	Berlin	120	1497
Q1 2010	20Mar10	Berlin	145	1497
Q1 2010	01Jan10	Lyon	355	1497
Q1 2010	10Mar10	Paris	385	1497
Q1 2010	10Jan10	Stuttgart	372	1497
Q4 2010	18Nov10	Berlin	50	253
Q4 2010	19Nov10	Berlin	100	253
Q4 2010	16Dec10	Lyon	103	253

Window partitioning

window_function OVER ([PARTITION BY *attribute_sequence*])

combined with `GROUP BY`:

Contribution to sales per city and period

```
SELECT quarter, city,
   SUM(SUM(sales)) OVER(PARTITION BY quarter) AS SALES_QT,
   SUM(sales) AS salesC
FROM Facts F, Time T, Product P, Location L
WHERE P.PID = F.PID
   AND T.TID = F.TID
   AND L.LID = F.LID
GROUP BY quarter, city;
```

```
QUARTER  CITY       SALES_QT SALESC
-------- --------- -------- ------
Q1 2010  Berlin         1497    385
Q1 2010  Lyon           1497    355
Q1 2010  Paris          1497    385
Q1 2010  Stuttgart      1497    372
Q4 2010  Berlin          253    150
Q4 2010  Lyon            253    103
```

Window ordering

window_function OVER ([PARTITION BY ...] [ORDER BY ...])

- orders rows within each partition
- aggregation restricted to preceding rows within the partition. 

Rank of city w.r.t. sales, within trimester

```
SELECT quarter, city,
   RANK() OVER(PARTITION BY quarter ORDER BY SUM(sales)) AS RG,
   SUM(sales) AS sales
FROM Facts F, Time T, Product P, Location L
WHERE P.PID = F.PID AND T.TID = F.TID AND L.LID = F.LID
GROUP BY quarter, city;
```

QUARTER	CITY	RG	SALES
Q1 2010	Lyon	1	355
Q1 2010	Stuttgart	2	372
Q1 2010	Berlin	3	385
Q1 2010	Paris	3	385
Q4 2010	Lyon	1	103
Q4 2010	Berlin	2	150

Window framing: ROWS

window_function OVER ([PARTITION BY ...] [ORDER BY ...] [*window_frame*])

- framing clause limits the scope of aggregation within each partition
- In ROW mode, frame is specified as nb of rows around current row.

Average sales of the city over last 3 months

```
SELECT T.TID AS T, month, city,
  AVG(SUM(sales)) OVER(
    PARTITION BY city
    ORDER BY T.TID
    ROWS 2 PRECEDING) sales3,
  SUM(sales) AS sales
FROM Facts F, Time T,
  Product P, Location L
WHERE P.PID = F.PID
  AND T.TID = F.TID
  AND L.LID = F.LID
GROUP BY month,T.TID, city
ORDER BY city,T.TID;
```

T	MONTH	CITY	SALES3	SALES
1	jan10	Berlin	163	163
2	fev10	Berlin	142.5	122
3	mar10	Berlin	128.333333	100
4	dec10	Berlin	124	150
1	jan10	Lyon	155	155
2	fev10	Lyon	127.5	100
3	mar10	Lyon	118.333333	100
4	dec10	Lyon	101	103
1	jan10	Paris	185	185
2	fev10	Paris	142.5	100
3	mar10	Paris	128.333333	100
1	jan10	Stuttgart	172	172
2	fev10	Stuttgart	136	100
3	mar10	Stuttgart	124	100

Window framing: RANGE

window_function OVER ([PARTITION BY ...] [ORDER BY ...] [*window_frame*])

- `RANGE` mode: frame specified by de/in-crementing `ORDER BY` expr. of current row. ↪ `ORDER BY` must be single expr. (ex: numeric or date).
- called "logical offset" as opposed to physical.

Counting cities with sales amount between 90% and 100% of current city, per trimester

```
SELECT quarter, city,
  COUNT(*) OVER(
    PARTITION BY quarter
    ORDER BY SUM(sales)
    RANGE 0.1*SUM(sales) PRECEDING) NB,
  SUM(sales) AS sales
FROM Facts F, Time T,
  Product P, Location L
WHERE P.PID = F.PID
  AND T.TID = F.TID
  AND L.LID = F.LID
GROUP BY quarter, city;
```

QUARTER	CITY	NB	SALES
Q1 2010	Lyon	1	355
Q1 2010	Stuttgart	2	372
Q1 2010	Berlin	4	385
Q1 2010	Paris	4	385
Q4 2010	Lyon	1	103
Q4 2010	Berlin	1	150

Scope of aggregation

- `OVER()`: all tuples
- `OVER(PARTITION BY)`: all tuples in same partition
- `OVER(ORDER BY)`: all tuples smaller than current tuple,
  `=RANGE UNBOUNDED PRECEDING`

- `ROWS UNBOUNDED PRECEDING`: all preceding tuples in same partition
- `ROWS k PRECEDING`: k preceding tuples in same partition
- `UNBOUNDED FOLLOWING`
  `k FOLLOWING`
  `CURRENT ROW`
  `BETWEEN`
  `ROWS BETWEEN 2 PRECEDING AND CURRENT ROW = ROWS 2 PRECEDING`

- `RANGE`: like `ROWS` but allow to define scope w.r.t. value instead of rank (number of lines)

`ROWS, RANGE` require order be defined.

Aggregation functions

- Analytical functions (set semantics $\Longrightarrow$ relational):

  `SUM, COUNT, MIN, MAX, AVG, EVERY, ANY,`
  `STDDEV_POP, VAR_SAMP, PERCENTILE_CONT...`

- Ranking functions (w.r.t. some order):

  `RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST,`
  `ROW_NUMBER, NTILE, LEAD, LAG, FIRST_VALUE...`

 `DISTINCT` aggregates and most ranking functions (but not `FIRST_VALUE...`) are not affected by framing clause.

Ranking functions

ROW_NUMBER: ties ranked arbitrarily

RANK vs DENSE_RANK: no value gap after ties

```
SELECT month, city, SUM(sales) AS sales,
  RANK() OVER (ORDER BY SUM(sales) DESC) RK,
  DENSE_RANK() OVER (ORDER BY SUM(sales) DESC) DENSE_RK,
  ROW_NUMBER() OVER (ORDER BY SUM(sales) DESC) ROW_NUM
FROM Facts F, Time T, Location L
WHERE T.TID = F.TID AND L.LID = F.LID
GROUP BY month,city;
```

MONTH	CITY	SALES	RK	DENSE_RK	ROW_NUM
jan10	Paris	185	1	1	1
jan10	Berlin	172	2	2	2
jan10	Stuttgart	172	2	2	3
jan10	Lyon	155	4	3	4
dec10	Berlin	150	5	4	5
...					

Reporting functions

1 by default

if offset exceeds partition

LAG/LEAD (*value_expr* [, *offset*] [, *default*]) [RESPECT NULLS | IGNORE NULLS]
OVER ([*query_partition_clause*] *order_by_clause*)

```
SELECT month,city, SUM(sales) AS sales,
  LAG(SUM(sales),1,-1) OVER (
    PARTITION by city
    ORDER BY to_date(month,'monYY'))
  AS sales_prev
FROM Facts F, Time T, Location L
WHERE T.TID = F.TID  AND L.LID = F.LID
GROUP BY month,city;
```

MONTH	CITY	SALES	SALES_PREV
jan10	Berlin	163	-1
feb10	Berlin	122	163
mar10	Berlin	100	122
dec10	Berlin	150	100
jan10	Lyon	155	-1
feb10	Lyon	100	155
mar10	Lyon	100	100
dec10	Lyon	103	100
jan10	Paris	185	-1
feb10	Paris	100	185
mar10	Paris	100	100
jan10	Stuttgart	172	-1
feb10	Stuttgart	100	172
mar10	Stuttgart	100	100

LEAD : same but offset follows instead of precede current row

Reporting functions

FIRST_VALUE, LAST_VALUE : select FIRST/LAST row of aggregation scope.

```
SELECT month,city, SUM(sales) AS sales,
  FIRST_VALUE(SUM(sales)) OVER (
    PARTITION by month
    ORDER BY SUM(sales) DESC)
  AS sales_max
FROM Facts F, Time T, Location L
WHERE T.TID = F.TID  AND L.LID = F.LID
GROUP BY month,city;
```

MONTH	CITY	SALES	SALES_MAX
dec10	Berlin	150	150
dec10	Lyon	103	150
feb10	Berlin	122	122
feb10	Lyon	100	122
feb10	Paris	100	122
feb10	Stuttgart	100	122
jan10	Paris	185	185
jan10	Stuttgart	172	185
jan10	Berlin	163	185
jan10	Lyon	155	185
mar10	Lyon	100	100
mar10	Paris	100	100
mar10	Berlin	100	100
mar10	Stuttgart	100	100


What result if we replace FIRST_VALUE with LAST_VALUE?

Pivoting

Principle: on fact table with sales measure, *GROUP BY quarter,city* would return 1 line per city. Instead, *pivot* query returns a single line per *quarter* with one column per city.

```
SELECT * FROM
  (SELECT quarter, city, sales
   FROM SALES_TRIM_CITY
   )
  PIVOT (SUM(sales)
    FOR city IN ('Paris' AS lutetia, 'Lyon' AS Lugdunum)
  )
;
```

```
QUARTER  CITY       SALES
-------- ---------- -----
Q1 2010  Berlin        385
Q1 2010  Paris         385
Q1 2010  Stuttgart     372
Q1 2010  Lyon          355
Q4 2010  Lyon          103
Q4 2010  Berlin        150
```

```
QUARTER  LUTETIA LUGDUNUM
-------- ------- --------
Q1 2010      385      355
Q4 2010               103
```

Supported in Oracle, SQL Server.

References

Oracle DataWarehousing guide (inspired the whole section):
http://docs.oracle.com/cd/E11882_01/server.112/e25554/analysis.htm

http://sqlpro.developpez.com/article/olap-clause-window/

http://www.dba-oracle.com/t_advanced_sql_windowing_clause.htm

http://msdn.microsoft.com/fr-fr/library/ms189461.aspx

http://www.vldb.org/pvldb/vol8/p1058-leis.pdf