

# Data Warehousing and Data Mining

## - OLAP Operations -

---

- SQL

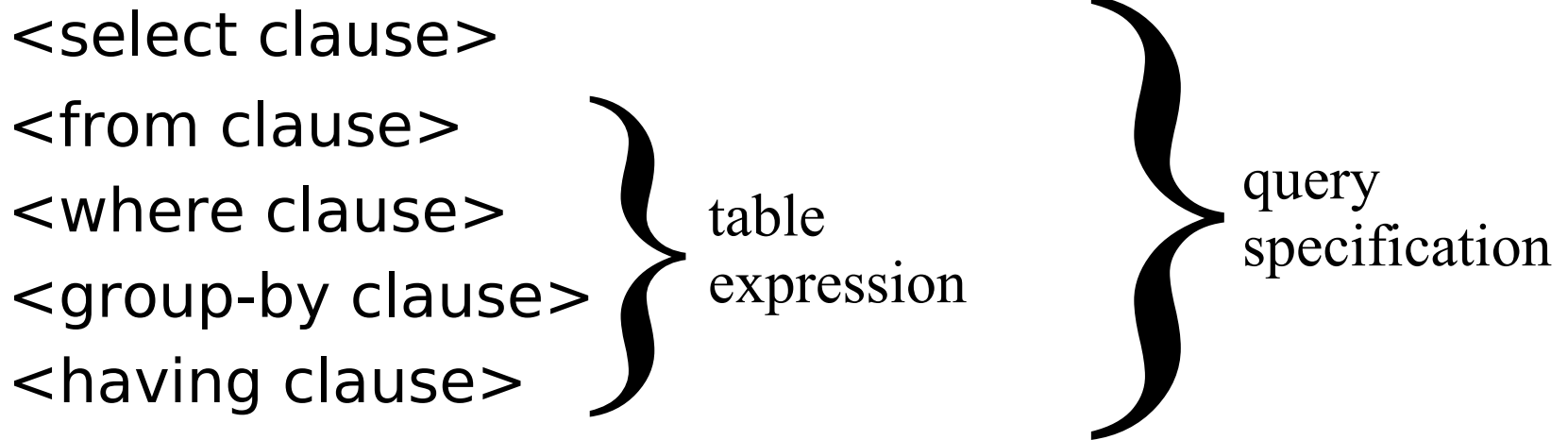
- table expression
- query specification
- query expression

- OLAP

- GROUP BY extensions: rollup, cube, grouping sets
- SQL for analysis and reporting: ranking, moving window

**Acknowledgements:** I am indebted to M. Böhlen for providing me the lecture notes.

# Query Specifications



- **Query specifications** are the building block for most SQL statements. They determine much of the expressive power of SQL.

# Table Expressions

---

- A **table expression** is defined as follows:

from clause

[where clause]

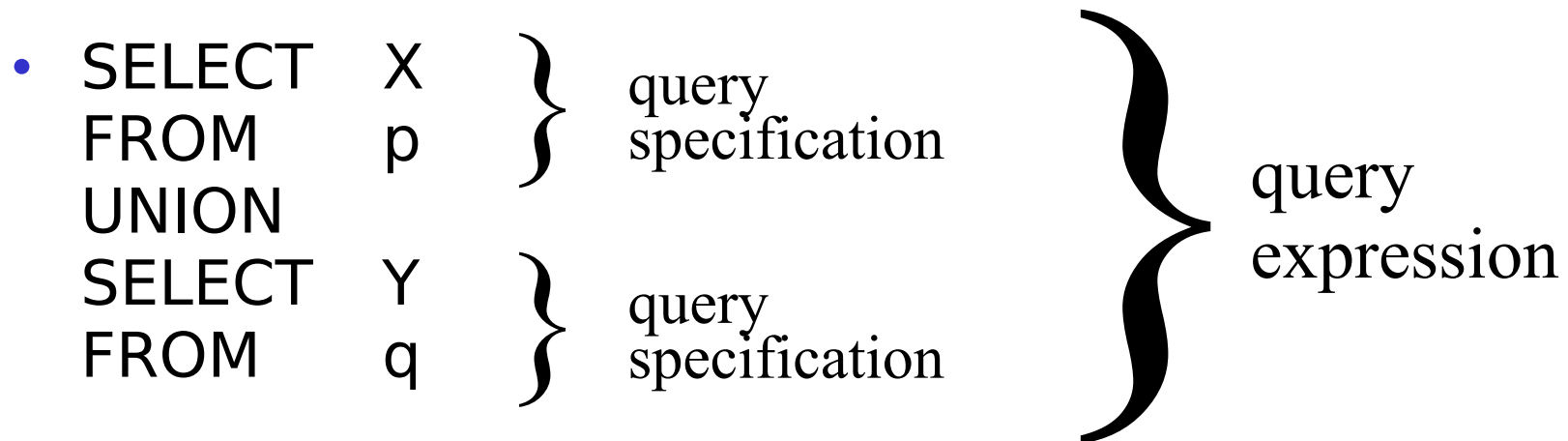
[group-by clause]

[having clause]

- The from clause is always required.
- The clauses are operators that take input and produce output.
- The output of each clause is a virtual table, i.e., a table that is not stored.

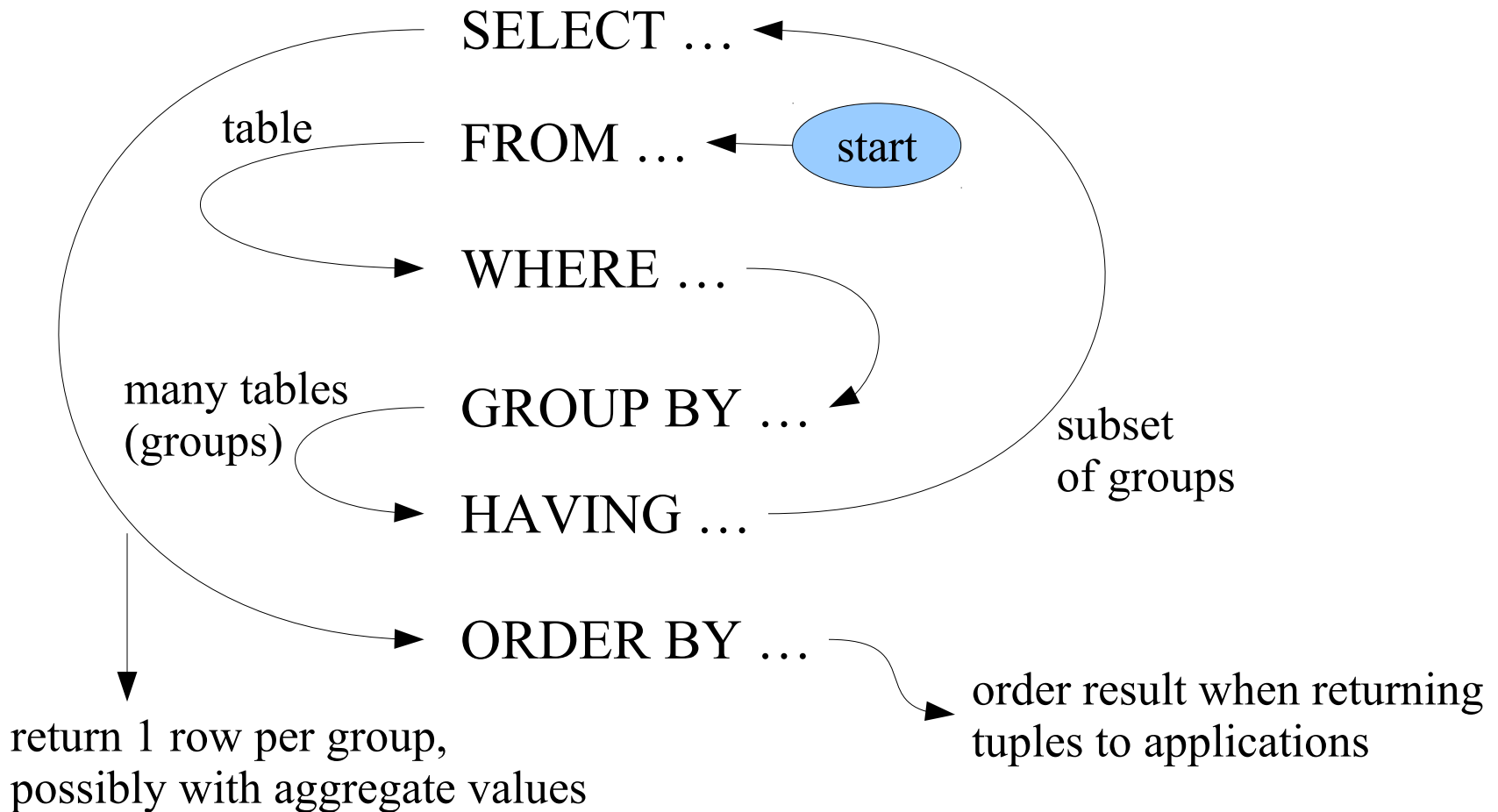
# Query Expressions

- Query specifications can be combined with set operations to form *query expressions*.
- The set operations from the RA are almost directly available in SQL
  - UNION ( $\cup$ ), EXCEPT ( $\setminus$ ), INTERSECT ( $\cap$ )



# Processing of SQL Statement

- Logical order of processing an SQL statement



# The From Clause

- FROM <table reference\_1>, ..., <table reference\_n>
- Computes the Cartesian product of all input tables.
- A table reference is either a table name or a query expression that defines a derived table.

```
FROM r  
WHERE ...
```

```
FROM ( SELECT ...  
      FROM ..  
      WHERE ...  
      ) AS r  
WHERE ...
```

# The From Clause/2

- SQL-92 supports many joins
  - FROM t1 CROSS JOIN t2
    - ◆ Cartesian product
  - FROM t1 NATURAL JOIN t2
    - ◆ Natural join (all identical columns)
  - FROM t1 JOIN t2 ON <join condition>
    - ◆ Theta join
  - FROM t1 JOIN t2 USING (<columns>)
    - ◆ Restricted natural join
  - FROM t1 LEFT OUTER JOIN t2 ON <join cond>
    - ◆ Left outer join (similar for RIGHT and FULL outer join)

# The Where Clause

---

- `WHERE <condition>`
- The where clause takes the virtual table produced by the from clause and filters out those rows that do not meet the condition.
- The where clause is used to specify join and selection conditions.
- Before SQL allowed query expressions in the from clause most of its expressive power came from subqueries in the where clause.

# The Group-by Clause

- GROUP BY <grouping-column1>,  
          <grouping-column2>, ...
- The result of the grouping clause is a grouped table.
- Every row in a group has the same value for the grouping columns.
- Apart from grouping, input and output table are identical.
- A missing GROUP BY clause implies that the entire table is treated as one group.

# Group-by Example

<b>Title</b>	<b>Type</b>	<b>Price</b>
Lethal Weapon	Action	3
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3
Rocky	Action	2

FROM movie\_titles  
GROUP BY Type

<b>Title</b>	<b>Type</b>	<b>Price</b>
Lethal Weapon	Action	3
Rocky	Action	2
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3

# The Having Clause

---

- HAVING <condition>
- The having clause takes a grouped table as input and returns a grouped table.
- The condition is applied to each group. Only groups that fulfill the condition are returned.
- The condition can either reference
  - grouping columns (because they are constant within a group)
  - Aggregated columns (because an aggregate yields one value per group)

# Having Clause Example

<b>Title</b>	<b>Type</b>	<b>Price</b>
Lethal Weapon	Action	3
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3
Rocky	Action	2

FROM movie\_titles  
GROUP BY Type  
HAVING max(Price) <= 3

<b>Title</b>	<b>Type</b>	<b>Price</b>
Lethal Weapon	Action	3
Rocky	Action	2
Star Wars	Fiction	3

# Having Clause Example/2

<b>Title</b>	<b>Type</b>	<b>Price</b>
Lethal Weapon	Action	3
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3
Rocky	Action	2

```
FROM movie_titles  
GROUP BY Type  
HAVING Price <= 3
```

↓  
Illegal SQL

# The Select Clause

- `SELECT [<quantifier>] e1, ..., en`
- The select clause resembles a generalized projection.
- Each item in the select clause is an expression.
- The select clause can contain aggregates.
- If a column in a select clause item is used in an aggregate then all columns have to be used in aggregates. (The exception are column names used in the group-by clause.)
- The quantifier enforces duplicate elimination (`DISTINCT`) or duplicate preservation (`ALL`). The default is `ALL`.

# The Select Clause/2

- `SELECT *` expands to all columns of all tables in the from clause (i.e., no projection in RA).
- `r.*` expands to all columns of table `r`.
- `SELECT *` and `r.*` should be avoided because they cause a statement to change along with schema modifications.
- Columns can be (re)named in the select clause.
  - `SELECT r.A B, r.B+r.C*2 X`
  - `SELECT r.A AS B, r.B+r.C*2 AS X`

# Summary and Outlook

---

- The SQL fragment is fairly powerful.
- Over many years it developed into the “intergalactic data speak” [Stonebraker].
- At some point it was observed that SQL is not good for analytical queries:
  - too difficult to formulate
  - too slow to execute
- OLAP was born
- A huge amount of activities during the last decade.
- GROUP BY extension
- SQL for analysis and reporting

# SQL Extensions for OLAP

- A key concept of OLAP systems is multidimensional analysis: examining data from many dimensions.
- Examples of multidimensional requests:
  - Show total sales across all products at increasing aggregation levels for a geography dimension, from state to country to region, for 1999 and 2000.
  - Create a cross-tabular analysis of our operations showing expenses by territory in South America for 1999 and 2000. Include all possible subtotals.
  - List the top 10 sales representatives in Asia according to 2000 sales revenue for food products, and rank their commissions.
- We use the ISO SQL:2003 OLAP extensions for our illustrations.

# Overview

---

We discuss how SQL:2003 has extended SQL-92 to support OLAP queries.

- crosstab
- group by extensions
  - rollup, cube, grouping sets
  - hierarchical cube
- analytic functions
  - ranking and percentiles
  - reporting
  - windowing
- data densification (partitioned outer join)

# Crosstab

- Cross-tabular report with (sub)totals

Media	Country		
	France	USA	Total
Internet	9,597	124,224	133,821
Direct Sales	61,202	638,201	699,403
Total	70,799	762,425	833,224

- This is space efficient for dense data only (thus, few dimensions)
- Shows data at different “granularities”

# Crosstab/2

- Tabular representation for the cross-tabular report with totals.
- ALL is a dummy value and stands for all or multiple values.
- Probably not as nice to read as the crosstab.
- Information content is the same as in the crosstab.
- Is more space efficient than crosstab if the data is sparse.
- Compatible with relational DB technology.

<b>Media</b>	<b>Country</b>	<b>Total</b>
Internet	France	9,597
Internet	USA	133,821
Direct Sales	France	61,202
Direct Sales	USA	638,201
Internet	ALL	133,821
Direct Sales	ALL	699,403
ALL	France	70,799
ALL	USA	762,425
ALL	ALL	833,224

# GROUP BY Extensions

- Aggregation is a fundamental part of OLAP and data warehousing.
- To improve aggregation performance Oracle provides the following extensions to the GROUP BY clause:
  - CUBE and ROLLUP
  - GROUPING SETS expression
  - Three GROUPING functions
- The CUBE, ROLLUP, and GROUPING SETS extensions make querying and reporting easier and faster.
- CUBE, ROLLUP, and GROUPING SETS produce a single result set that is equivalent to a UNION ALL of differently grouped rows.

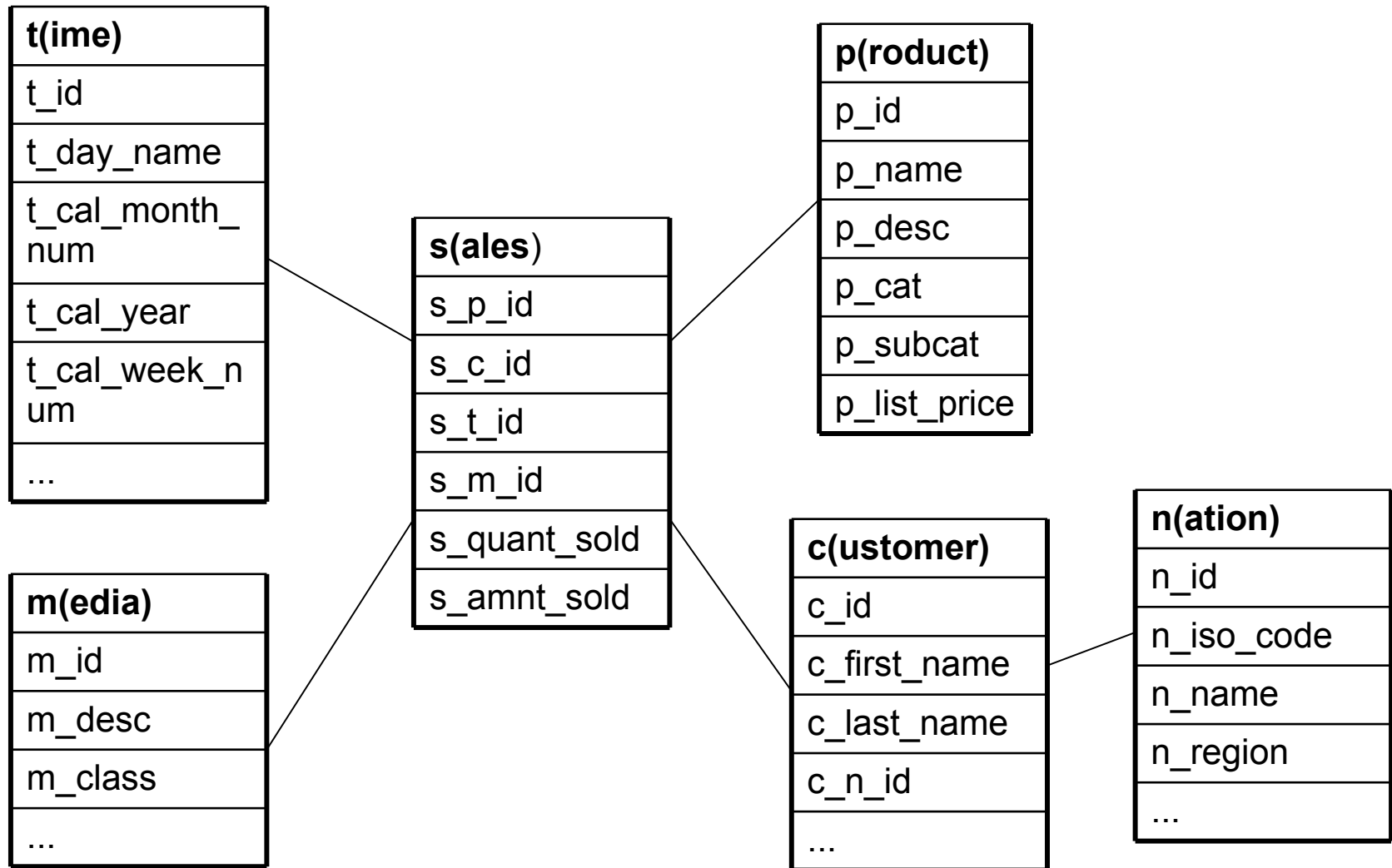
# GROUP BY Extensions/2

- ROLLUP calculates aggregations such as SUM, COUNT, MAX, MIN, and AVG at **increasing levels** of aggregation, from the most detailed up to a grand total.
- CUBE is similar to ROLLUP, enabling a single statement to calculate **all possible combinations** of aggregations.
  - Computing a CUBE creates a heavy processing load,
- The GROUPING SETS extension lets you specify **just the needed groupings** in the GROUP BY clause. This allows efficient analysis across multiple dimensions without performing a CUBE operation.
  - Replacing cubes with grouping sets can significantly increase performance.

# Syntax of GROUP BY Extensions

- GROUP BY ROLLUP(gcols)
  - Roll-up hierarchically
- GROUP BY CUBE (gcols)
  - Roll-up to all possible combinations
- GROUP BY gcols1, CUBE(gcols2)
  - Partial roll-up
- GROUP BY GROUPING SETS (gcols1, ..., gcolsN)
  - Explicit specification of roll-ups
- GROUP BY groupings1, groupings2, ...
  - Cross-product of groupings
- SELECT ... GROUPING\_ID(gcols)...
  - Identification of roll-up level

# Example Sales Schema



# Example Sales Schema/2

- An instance is in Oracle DB on `alcor.inf.unibz.it`
  - Available in schema `bi`
  - Write `bi.s` to access the sales table
- The following view is available

```
CREATE OR REPLACE VIEW spctmn AS
SELECT      *
FROM        bi.s, bi.p, bi.c, bi.t, bi.m, bi.n
WHERE       s_t_id = t_id
AND         s_p_id = p_id
AND         s_c_id = c_id
AND         s_m_id = m_id
AND         c_n_id = n_id;
```

# Example Sales Schema/3

- To access the DB log in to `russe1.inf.unibz.it`
- Environment variables should be

```
export ORACLE_HOME=/usr/lib/oracle/10.2.0.3/client
export ORACLE_SID=orcl
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH
```

- Create a text file with SQL commands, e.g., `q1.sql`

```
-- This is an example file. A semicolon terminates a statement.
-- desc x describes the schema of table x.
-- Comment lines start with two -'s
```

```
select count(*) from bi.s;
desc bi.s;
quit;
```

- Execute the file by running the command:

```
sqlplus dwdm/dummy@ alcor.inf.unibz.it:1521/orcl @q1
```

# ROLLUP Example

```
SELECT      m_desc,   t_cal_month_desc, n_iso_code,
            SUM(s_amount_sold)
FROM        spctmn
WHERE       m_desc IN ('Direct Sales', 'Internet')
AND        t_cal_month_desc IN ('2000-09', '2000-10')
AND        n_iso_code IN ('GB', 'US')
GROUP BY   ROLLUP(m_desc, t_cal_month_desc, n_iso_code);
```

- Rollup from right to left
- Computes and combines the following groupings
  - m\_desc, t\_cal\_month\_desc, n\_iso\_code
  - m\_desc, t\_cal\_month\_desc
  - m\_desc
  - -

# ROLLUP Example/2

<b>M_DESC</b>	<b>T_CAL_MO</b>	<b>N_</b>	<b>SUM(S_amount_sold)</b>
-----	-----	--	-----
<b>Internet</b>	<b>2000-09</b>	<b>GB</b>	<b>16569.36</b>
<b>Internet</b>	<b>2000-09</b>	<b>US</b>	<b>124223.75</b>
<b>Internet</b>	<b>2000-10</b>	<b>GB</b>	<b>14539.14</b>
<b>Internet</b>	<b>2000-10</b>	<b>US</b>	<b>137054.29</b>
<b>Direct Sales</b>	<b>2000-09</b>	<b>GB</b>	<b>85222.92</b>
<b>Direct Sales</b>	<b>2000-09</b>	<b>US</b>	<b>638200.81</b>
<b>Direct Sales</b>	<b>2000-10</b>	<b>GB</b>	<b>91925.43</b>
<b>Direct Sales</b>	<b>2000-10</b>	<b>US</b>	<b>682296.59</b>
<b>Internet</b>	<b>2000-09</b>		<b>140793.11</b>
<b>Internet</b>	<b>2000-10</b>		<b>151593.43</b>
<b>Direct Sales</b>	<b>2000-10</b>		<b>774222.02</b>
<b>Direct Sales</b>	<b>2000-09</b>		<b>723423.73</b>
<b>Internet</b>			<b>292386.54</b>
<b>Direct Sales</b>			<b>1497645.75</b>
			<b>1790032.29</b>

# Partial ROLLUP Example

```
SELECT      m_desc,   t_cal_month_desc, n_iso_code,
            SUM(s_amount_sold)
FROM        spctmn
WHERE       m_desc IN ('Direct Sales', 'Internet')
AND         t_cal_month_desc IN ('2000-09', '2000-10')
AND         n_iso_code IN ('GB', 'US')
GROUP BY    m_desc, ROLLUP(t_cal_month_desc, n_iso_code);
```

- m\_desc is always present and not part of the rollup hierarchy
- Computes and combines the following groupings
  - m\_desc, t\_cal\_month\_desc, n\_iso\_code
  - m\_desc, t\_cal\_month\_desc
  - m\_desc

# Partial ROLLUP Example/2

<b>M_DESC</b> -----	<b>T_CAL_MO</b> -----	<b>N_</b> --	<b>SUM(S_amount_sold)</b> -----
<b>Internet</b>	<b>2000-09</b>	<b>GB</b>	<b>16569.36</b>
<b>Internet</b>	<b>2000-09</b>	<b>US</b>	<b>124223.75</b>
<b>Internet</b>	<b>2000-10</b>	<b>GB</b>	<b>14539.14</b>
<b>Internet</b>	<b>2000-10</b>	<b>US</b>	<b>137054.29</b>
<b>Direct Sales</b>	<b>2000-09</b>	<b>GB</b>	<b>85222.92</b>
<b>Direct Sales</b>	<b>2000-09</b>	<b>US</b>	<b>638200.81</b>
<b>Direct Sales</b>	<b>2000-10</b>	<b>GB</b>	<b>91925.43</b>
<b>Direct Sales</b>	<b>2000-10</b>	<b>US</b>	<b>682296.59</b>
<b>Internet</b>	<b>2000-09</b>		<b>140793.11</b>
<b>Internet</b>	<b>2000-10</b>		<b>151593.43</b>
<b>Direct Sales</b>	<b>2000-09</b>		<b>723423.73</b>
<b>Direct Sales</b>	<b>2000-10</b>		<b>774222.02</b>
<b>Internet</b>			<b>292386.54</b>
<b>Direct Sales</b>			<b>1497645.75</b>

# ROLLUP

- ROLLUP creates subtotals at  $n+1$  levels, where  $n$  is the number of grouping columns
  - Rows that would be produced by GROUP BY without ROLLUP
  - First-level subtotals
  - Second-level subtotals
  - ...
  - A grand total row
- It is very helpful for subtotaling along a hierarchical dimensions such as time or geography
  - ROLLUP(y, m, day) or ROLLUP(country, state, city)

# CUBE Example

```
SELECT      m_desc,   t_cal_month_desc, n_iso_code,
            SUM(s_amount_sold)
FROM        spctmn
WHERE       m_desc IN ('Direct Sales', 'Internet')
AND         t_cal_month_desc IN ('2000-09', '2000-10')
AND         n_iso_code IN ('GB', 'US')
GROUP BY    CUBE(m_desc, t_cal_month_desc, n_iso_code);
```

- Produces all possible roll-up combinations
- Computes and combines the following groupings
  - m\_desc, t\_cal\_month\_desc, n\_iso\_code
  - m\_desc, t\_cal\_month\_desc
  - m\_desc, n\_iso\_code
  - t\_cal\_month, n\_iso\_code
  - m\_desc
  - ...

# CUBE Example/2

M_DESC	T_CAL_MO	N_	SUM(S_AMOUNT_SOLD)
-----	-----	--	-----
Internet	2000-09	GB	16569.36
Internet	2000-09	US	124223.75
Internet	2000-10	GB	14539.14
Internet	2000-10	US	137054.29
Direct Sales	2000-09	GB	85222.92
Direct Sales	2000-09	US	638200.81
Direct Sales	2000-10	GB	91925.43
Direct Sales	2000-10	US	682296.59
	2000-09	GB	101792.28
	2000-09	US	762424.56
	2000-10	GB	106464.57
	2000-10	US	819350.88
Internet		GB	31108.5
Internet		US	261278.04
Direct Sales		GB	177148.35
Direct Sales		US	1320497.4
Internet	2000-09		140793.11
Internet	2000-10		151593.43
Direct Sales	2000-09		723423.73
Direct Sales	2000-10		774222.02
Internet			292386.54
Direct Sales			1497645.75
	2000-09		864216.84
	2000-10		925815.45
		GB	208256.85
		US	1581775.44
			1790032.29

# CUBE

- CUBE creates  $2^n$  combinations of subtotals, where  $n$  is the number of grouping columns
  - Includes all the rows produced by ROLLUP
- CUBE is typically most suitable in queries that use columns from multiple dimensions rather than columns representing different levels of a single dimension
  - e.g., subtotals for all combinations of month, state, and product
- Partial CUBE similar to partial ROLLUP

# GROUPING SETS Example

```
SELECT    m_desc, t_cal_month_desc, n_iso_code,
          SUM(s_amount_sold)
FROM      spctmn
WHERE     m_desc IN ('Direct Sales', 'Internet')
AND       t_cal_month_desc IN ('2000-09', '2000-10')
AND       n_iso_code IN ('GB', 'US')
GROUP BY GROUPING SETS ((m_desc, t_cal_month_desc, n_iso_code),
                        (m_desc, n_iso_code),
                        (t_cal_month_desc, n_iso_code));
```

- Grouping sets produce just the specified groupings.
- No (automatic) rollup is performed.

# GROUPING SETS Example/2

<b>M_DESC</b>	<b>T_CAL_MO</b>	<b>N_</b>	<b>SUM(S_AMOUNT_SOLD)</b>
-----	-----	--	-----
<b>Internet</b>	<b>2000-09</b>	<b>GB</b>	<b>16569.36</b>
<b>Direct Sales</b>	<b>2000-09</b>	<b>GB</b>	<b>85222.92</b>
<b>Internet</b>	<b>2000-09</b>	<b>US</b>	<b>124223.75</b>
<b>Direct Sales</b>	<b>2000-09</b>	<b>US</b>	<b>638200.81</b>
<b>Internet</b>	<b>2000-10</b>	<b>GB</b>	<b>14539.14</b>
<b>Direct Sales</b>	<b>2000-10</b>	<b>GB</b>	<b>91925.43</b>
<b>Internet</b>	<b>2000-10</b>	<b>US</b>	<b>137054.29</b>
<b>Direct Sales</b>	<b>2000-10</b>	<b>US</b>	<b>682296.59</b>
	<b>2000-09</b>	<b>GB</b>	<b>101792.28</b>
	<b>2000-09</b>	<b>US</b>	<b>762424.56</b>
	<b>2000-10</b>	<b>GB</b>	<b>106464.57</b>
	<b>2000-10</b>	<b>US</b>	<b>819350.88</b>
<b>Internet</b>		<b>GB</b>	<b>31108.5</b>
<b>Internet</b>		<b>US</b>	<b>261278.04</b>
<b>Direct Sales</b>		<b>GB</b>	<b>177148.35</b>
<b>Direct Sales</b>		<b>US</b>	<b>1320497.4</b>

# Equivalences

- $\text{CUBE}(a,b) \equiv \text{GROUPING SETS}((a,b), (a), (b), ())$
- $\text{ROLLUP}(a,b,c) \equiv \text{GROUPING SETS}((a,b,c), (a,b), (a), ())$
- $\text{GROUP BY GROUPING SETS}(a,b,c) \equiv$   
     $\text{GROUP BY } a \text{ UNION ALL GROUP BY } b \text{ UNION ALL GROUP BY } c$
- $\text{GROUP BY GROUPING SETS}((a,b,c)) \equiv \text{GROUP BY } a, b, c$
- $\text{GROUP BY GROUPING SETS}(a,b,(b,c)) \equiv$   
     $\text{GROUP BY } a \text{ UNION ALL GROUP BY } b \text{ UNION ALL GROUP BY } b, c$
- $\text{GROUP BY GROUPING SETS}(a, \text{ROLLUP}(b,c)) \equiv$   
     $\text{GROUP BY } a \text{ UNION ALL GROUP BY ROLLUP}(b, c)$

# Identification of Groupings

- With rollup and cube we must provide a possibility to programmatically determine the rollup level.
- The GROUPING\_ID function is designed for this.
  - GROUPING\_ID takes a list of grouping columns as an argument.
  - For each column it returns 1 if its value is NULL because of a rollup, and 0 otherwise.
  - The list of binary digits is interpreted as a binary number and returned as a base-10 number.
- Example: GROUPING\_ID(a,b) for CUBE(a,b)

<b>a</b>	<b>b</b>	<b>Bit vector</b>	<b>GROUPING_ID(a,b)</b>
1	2	0 0	0
1	NULL	0 1	1
NULL	1	1 0	2
NULL	NULL	1 1	3

# GROUPING\_ID Example

```
SELECT
  CASE WHEN GROUPING_ID(m_desc)=1 THEN '*' ELSE m_desc END,
  CASE WHEN GROUPING_ID(n_iso_c)=1 THEN '*' ELSE n_iso_c END,
  SUM(s_amount_sold)
FROM      spctmn
WHERE     m_desc IN ('Direct Sales', 'Internet')
AND       t_cal_month_desc= '2000-09'
AND       n_iso_code IN ('GB', 'US')
GROUP BY  CUBE(m_desc, n_iso_code);
```

- Replaces all NULL from rollup with string '\*'.
- Leaves NULL that are not the result of rollup untouched.
- Could easily make selective replacements of NULL.

# GROUPING\_ID Example/2

<b>CASEWHENGROUPING(M_D</b>	<b>CAS</b>	<b>SUM(S_AMOUNT_SOLD)</b>
<b>-----</b>	<b>---</b>	<b>-----</b>
<b>Internet</b>	<b>GB</b>	<b>16569.36</b>
<b>Internet</b>	<b>US</b>	<b>124223.75</b>
<b>Direct Sales</b>	<b>GB</b>	<b>85222.92</b>
<b>Direct Sales</b>	<b>US</b>	<b>638200.81</b>
<b>Direct Sales</b>	<b>*</b>	<b>723423.73</b>
<b>Internet</b>	<b>*</b>	<b>140793.11</b>
<b>*</b>	<b>GB</b>	<b>101792.28</b>
<b>*</b>	<b>US</b>	<b>762424.56</b>
<b>*</b>	<b>*</b>	<b>864216.84</b>

# Composite Columns

- A **composite column** is a collection of columns that are treated as a unit for the grouping.
- Allows to skip aggregation across certain levels
- Example: `ROLLUP (year, (quarter, month), day)`
  - `(quarter, month)` is treated as a unit
  - produces the groupings  
`(year, quarter, month, day)`  
`(year, quarter, month)`  
`(year)`  
`()`

# Concatenated Groupings

- A **concatenated grouping** is specified by listing multiple grouping sets, cubes, and rollups, and produces the cross-product of groupings from each grouping set
- Example:
  - `GROUP BY GROUPING SETS(a,b), GROUPING SETS(c,d)` produces  $(a,c)$ ,  $(a,d)$ ,  $(b,c)$ ,  $(b,d)$
- A concise and easy way to generate useful combinations of groupings
  - A small number of concatenated groupings can generate a large number of final groups
  - One of the most important uses for concatenated groupings is to generate the aggregates for a hierarchical cube

# Hierarchical Cubes

- A hierarchical cube is a data set where the data is aggregated along the rollup hierarchy of each of its dimensions.
- The aggregations are combined across dimensions
- **Example:**
  - `ROLLUP(year, quarter, month),`  
`ROLLUP(category, subcategory, name),`  
`ROLLUP(region, subregion, country, state, city)`
  - Produces a total of  $4 \times 4 \times 6 = 96$  aggregate groups
    - ◆ Compare to  $2^{12} = 4096$  groupings by CUBE and 96 explicit group specifications
  - Groups: (year,category,region), (quarter,category,region), (month,category,region), ...
- In SQL we specify hierarchies explicitly. They are not “known” to the system.

# Hierarchical Cubes/2

- Hierarchical cubes (logical cubes) form the basis for many analytic applications, but frequently only certain slices are needed.
- Complex OLAP queries are handled by enclosing the **hierarchical cube** in an outer query that specifies the exact slice that is needed.

```
SELECT month, division, sum_sales
FROM (SELECT year, quarter, month, division, brand,
            item, SUM(sales) sum_sales,
            GROUPING_ID(grouping-columns) gid
      FROM sales, products, time
      GROUP BY ROLLUP(year, quarter, month),
              ROLLUP(division, brand, item))
WHERE division = 25 AND month = 200201
      AND gid = gid-for-Division-Month;
```

# Hierarchical Cubes/3

- Query optimizers can be tuned to efficiently handle nested hierarchical cubes.
- Based on the outer query condition, unnecessary groups are not generated.
  - In the example, 15 out of 16 groups are removed (i.e., all groups involving year, quarter, brand, and item)
- Optimized query:

```
SELECT month, division, sum_sales
FROM (SELECT null, null, month, division, null, null,
          SUM(sales) sum_sales,
          GROUPING_ID(grouping-columns) gid
FROM sales, products, time
GROUP BY month, division)
WHERE division = 25 AND month = 200201
      AND gid = gid-for-Division-Month
```

# Window Functions

- ISO SQL:2003 has enhanced SQL's analytical processing capabilities by introducing *window functions*.
- All major database systems (have to) support window functions.
- These window functions permit things such as:
  - Rankings and percentiles: cumulative distributions, percent rank, and N-tiles.
  - Reporting aggregate functions (nested aggregations, moving averages)
  - Data densification
  - Linear regression

# Window Functions/2

- Basic syntax:  
`WFctType(expr) OVER (WPartitioning WOrdering Wframe)`
- Window functions may only be used in the select (or ordering) clause
- Processing order within a query specification:
  - FROM, WHERE, GROUP BY, and HAVING are computed as usual
  - The window functions are computed last (after groupings and aggregations)
  - The final ORDER BY clause is evaluated and determines the output ordering

# Window Functions/3

- Basic syntax:  
`WFctType(expr) OVER (WPartitioning WOrdering Wframe)`
- WPartitioning
  - Divides the table into windows, i.e., groups of rows.
  - Windows are created after groupings and aggregations, and can refer to any aggregate results.
- WOrdering
  - Determines the ordering in which the rows are passed to the window function
  - Many window functions are sensitive to the ordering of rows

# Window Functions/4

- Basic syntax:  
`WFctType(expr) OVER (WPartitioning WOrdering Wframe)`
- WFrame
  - For each row within a window a sliding frame of data can be defined.
  - The window frame determines the rows that are used to calculate values for the current row.
  - Windows can be defined as a
    - ◆ physical number of rows or
    - ◆ logical range of rows
- Each calculation with an window function is based on the current row within a window.

# Ranking and Percentiles

- RANK() OVER ( [WPartitioning] WOrdering )
- DENSE\_RANK() OVER ( [WPartitioning] WOrdering )
- CUME\_DIST() OVER ( [WPartitioning] WOrdering )
- PERCENT\_RANK () OVER  
([WPartitioning] WOrdering)
- NTILE(expr) OVER ([WPartitioning] WOrdering)
- ROW\_NUMBER() OVER  
( [WPartitioning] WOrdering )

# RANK Example

```
SELECT    m_desc, SUM(s_amount_sold),  
          RANK() OVER (ORDER BY SUM(s_amount_sold))  
FROM      bi.spctmn  
WHERE     m_desc IN ('Direct Sales', 'Internet')  
AND       t_cal_month_desc IN ('2000-09', '2000-10')  
AND       n_iso_code = 'US'  
GROUP BY m_desc;
```

- RANK() assigns the rank to each row according to the order of the total amount sold.
- The ordering attribute or expression must be specified.
- The ordering can be ASC (default) or DESC.

# RANK Example/2

```
SELECT      m_desc, SUM(s_amount_sold),
            RANK() OVER (ORDER BY SUM(s_amount_sold))
FROM        bi.spctmn
WHERE       m_desc IN ('Direct Sales', 'Internet')
AND         t_cal_month_desc IN ('2000-09', '2000-10')
AND         n_iso_code = 'US'
GROUP BY   m_desc;
```

M_DESC	SUM(S_AMNT_SOLD)	RANK
Internet	261278.04	1
Partners	800871.37	2
Direct Sales	1320497.4	3

# RANK with Partitioning Example

```
SELECT      m_desc, t_cal_month_desc, SUM(s_amnt_sold),
            RANK() OVER (PARTITION BY m_desc
                        ORDER BY SUM(s_amnt_sold) DESC)
            AS RANK_BY_MEDIA
FROM        bi.spctmn
WHERE       t_cal_month_desc IN
            ('2000-08', '2000-09', '2000-10', '2000-11')
AND        m_desc IN ('Direct Sales', 'Internet')
GROUP BY   m_desc, t_cal_month_desc;
```

- If a PARTITION BY clause is specified the rank is computed independently for each group specified by the partitioning, i.e., the rank is reset for each group.

# RANK with Partitioning Example/2

<b>Media</b>	<b>Month</b>	<b>AmntSold</b>	<b>RankByMedia</b>
-----	-----	-----	-----
<b>Direct Sales</b>	<b>2000-08</b>	<b>1236104.31</b>	<b>1</b>
<b>Direct Sales</b>	<b>2000-10</b>	<b>1225584.31</b>	<b>2</b>
<b>Direct Sales</b>	<b>2000-09</b>	<b>1217807.75</b>	<b>3</b>
<b>Direct Sales</b>	<b>2000-11</b>	<b>1115239.03</b>	<b>4</b>
<b>Internet</b>	<b>2000-11</b>	<b>284741.77</b>	<b>1</b>
<b>Internet</b>	<b>2000-10</b>	<b>239236.26</b>	<b>2</b>
<b>Internet</b>	<b>2000-09</b>	<b>228241.24</b>	<b>3</b>
<b>Internet</b>	<b>2000-08</b>	<b>215106.56</b>	<b>4</b>

# Multiple RANK Functions

- A query block can contain more than 1 ranking function, each partitioning the data into different groups
  - e.g., rank products based on their dollar sales within each month and within each channel

```
RANK() OVER (PARTITION BY cal_month_desc ORDER BY SUM(amount_sold)),  
RANK() OVER (PARTITION BY channel_desc ORDER BY SUM(amount_sold))
```

# DENSE\_RANK

- DENSE\_RANK leaves no gaps in the ranking sequence when there are ties
  - e.g., rank and dense rank of amount sold

```
RANK() OVER (ORDER BY SUM(amount_sold)) AS Rank,  
RANK() OVER (ORDER BY SUM(amount_sold)) AS Dense_Rank
```

Media	Month	AmntSold	Rank	Dense_Rank
-----	-----	-----	----	-----
Direct Sales	2000-09	1200000	1	1
Direct Sales	2000-10	1200000	1	1
Partners	2000-09	600000	3	2
Partners	2000-10	600000	3	2
Internet	2000-09	200000	5	3
Internet	2000-10	200000	5	3

# Ranking Example/1

- Rank the media ('Internet' versus 'Direct sales') used for selling products according to their dollar sales. Use the number of unit sales to break ties. Do the analysis for August until November 2000.

```
SELECT  m_desc, t_cal_month_desc,
        SUM(s_amnt_sold), SUM(s_quantity_sold),
        RANK() OVER (ORDER BY SUM(s_amnt_sold) DESC,
                      SUM(s_quantity_sold) DESC) AS Rank
FROM    bi.spctmn
WHERE   m_desc IN ('Direct Sales', 'Internet'),
AND    t_cal_month_desc IN ('2000-08', '2000-09',
                           '2000-10', '2000-11')
GROUP BY m_desc, t_cal_month_desc;
```

# Ranking Example/2

- Determine the two least and most successful sales media, respectively (in terms of total amount sold).

```
SELECT      *
FROM        ( SELECT m_desc, SUM(s_amnt_sold),
                  RANK() OVER (ORDER BY SUM(s_amnt_sold)) worst,
                  RANK() OVER (ORDER BY SUM(s_amnt_sold) DESC) best
              FROM      bi.spctmn
              GROUP BY m_desc )
WHERE       worst < 3 OR best < 3;
```

M_DESC	SUM(S_AMNT_SOLD)	Worst	Best
-----	-----	-----	-----
Direct Sales	57875260	4	1
Partners	26346342	3	2
Internet	13706802	2	3
Tele Sales	277426	1	4

# Ranking Example/3

- Rank sales per media and country, per media, and per country, respectively. Consider US, JP, and DK during September 2000.

```
SELECT  m_desc, n_iso_code, SUM(s_amnt_sold),
        RANK() OVER (PARTITION BY GROUPING_ID(m_desc, n_iso_code)
                    ORDER BY SUM(s_amnt_sold) DESC) AS RankPerGroup,
FROM    bi.spctmn
WHERE   t_cal_month = '2000-09'
AND     n_iso_code IN ('DK', 'US', 'JP')
GROUP BY CUBE(m_desc, n_iso_code)
HAVING  GROUPING_ID(m_desc, n_iso_code) <> 3
ORDER BY GROUPING_ID(m_desc, n_iso_code);
```

# Ranking Example/4

- Determine the output of the following statement:

```
SELECT c_id, p_id,  
       RANK() OVER (ORDER BY p_id) AS r1,  
       RANK() OVER (ORDER BY c_id) AS r2,  
       RANK() OVER (ORDER BY 1) AS r3,  
       RANK() OVER (PARTITION BY c_id ORDER BY p_id) AS r4,  
       RANK() OVER (PARTITION BY p_id ORDER BY c_id) AS r5  
FROM bi.spctmn  
WHERE c_id in (214, 608, 699)  
AND p_id in (42, 98, 123)  
GROUP BY c_id, p_id;
```

C_ID	P_ID	R1	R2	R3	R4	R5
214	123					
608	42					
608	123					
699	42					
699	123					

# CUME Example

```
SELECT    t_cal_month_desc AS MONTH, m_desc, SUM(s_amnt_sold),
          CUME_DIST() OVER (PARTITION BY t_cal_month_desc
                             ORDER BY SUM(s_amnt_sold))
          AS CUME_DIST_BY_MEDIA
FROM      bi.spctmn
WHERE     t_cal_month_desc IN ('2000-09', '2000-07', '2000-08')
GROUP BY t_cal_month_desc, m_desc;
```

- CUME\_DIST() computes the position of a value relative to a set of values.
- Thus, CUME\_DIST(x) is the number of values smaller or equal to x divided by the total number of values.
- PERCENT\_RANK() is similar and computes  $(\text{rank of row} - 1) / (\text{number of rows} - 1)$

# CUME Example/2

MONTH	M_DESC	SUM(S_AMNT	CUME_DIST_BY_MEDIA
-----	-----	-----	-----
2000-07	Internet	140423.34	.333333333
2000-07	Partners	611064.35	.666666667
2000-07	Direct Sales	1145275.13	1
2000-08	Internet	215106.56	.333333333
2000-08	Partners	661044.92	.666666667
2000-08	Direct Sales	1236104.31	1
2000-09	Internet	228241.24	.333333333
2000-09	Partners	666171.69	.666666667
2000-09	Direct Sales	1217807.75	1

# NTILE Example

```
SELECT    t_cal_month_desc AS MONTH , SUM(s_amnt_sold),
          NTILE(4) OVER (ORDER BY SUM(s_amnt_sold)) AS TILE4
FROM      bi.spctmn
WHERE     t_cal_year=2000
AND       p_cat = 'Electronics'
GROUP BY t_cal_month_desc;
```

- NTILE(n) divides the data into n equal sized buckets. Each bucket is assigned a number.
- Each bucket shall contain the same number of rows.
- If the rows cannot be distributed evenly then the highest buckets have one row less.

# NTILE Example/2

<b>MONTH</b>	<b>SUM(S_AMNT_SOLD)</b>	<b>TILE4</b>
-----	-----	-----
<b>2000-02</b>	<b>242416.38</b>	<b>1</b>
<b>2000-01</b>	<b>257285.89</b>	<b>1</b>
<b>2000-03</b>	<b>280010.94</b>	<b>1</b>
<b>2000-06</b>	<b>315950.95</b>	<b>2</b>
<b>2000-05</b>	<b>316824.18</b>	<b>2</b>
<b>2000-04</b>	<b>318105.67</b>	<b>2</b>
<b>2000-07</b>	<b>433823.77</b>	<b>3</b>
<b>2000-08</b>	<b>477833.26</b>	<b>3</b>
<b>2000-12</b>	<b>553534.39</b>	<b>3</b>
<b>2000-10</b>	<b>652224.76</b>	<b>4</b>
<b>2000-11</b>	<b>661146.75</b>	<b>4</b>
<b>2000-09</b>	<b>691448.94</b>	<b>4</b>

# ROW\_NUMBER Example

```
SELECT    m_desc, t_cal_month_desc,  
          SUM(s_amnt_sold),  
          ROW_NUMBER() OVER (ORDER BY SUM(s_amnt_sold) DESC)  
          AS Row_Number  
FROM      bi.spctmn  
WHERE     t_cal_month_desc IN ('2001-09', '2001-10')  
GROUP BY m_desc, t_cal_month_desc;
```

- ROW\_NUMBER assigns a unique number (sequentially, starting from 1, as defined by ORDER BY) to each row within the partition.

# ROW\_NUMBER Example/2

<b>M_DESC</b>	<b>MONTH</b>	<b>SUM(S_AMNT_SOLD)</b>	<b>ROW_NUMBER</b>
-----	-----	-----	-----
<b>Direct Sales</b>	<b>2001-09</b>	<b>1100000</b>	<b>1</b>
<b>Direct Sales</b>	<b>2001-10</b>	<b>1000000</b>	<b>2</b>
<b>Internet</b>	<b>2001-09</b>	<b>500000</b>	<b>3</b>
<b>Internet</b>	<b>2001-10</b>	<b>700000</b>	<b>4</b>
<b>Partners</b>	<b>2001-09</b>	<b>600000</b>	<b>5</b>
<b>Partners</b>	<b>2001-10</b>	<b>600000</b>	<b>6</b>

# Nested Aggregates

---

- After a query has been processed, aggregate values like the number of rows or an average value in a column can be made available to window functions.
- This yields nested aggregations
- Nested aggregate functions return the same value for each row in a window.
- Reporting functions relate partial totals to grand totals, etc. They are based on nested aggregations.

# RATIO\_TO\_REPORT Example

```
SELECT  m_desc,  
        SUM(s_amnt_sold) AS SALES,  
        SUM(SUM(s_amnt_sold)) OVER () AS TOTAL_SALES,  
        RATIO_TO_REPORT(SUM(s_amnt_sold)) OVER () AS RATIO  
FROM    bi.spctmn  
WHERE   s_t_id = to_DATE('11-OCT-2000')  
GROUP BY m_desc;
```

- For each media compute the total amount sold and the ration wrt the overall total amount sold (across all media).

# RATIO\_TO\_REPORT Example/2

<b>M_DESC</b>	<b>SALES</b>	<b>TOTAL_SALES</b>	<b>RATIO</b>
-----	-----	-----	-----
<b>Direct Sales</b>	<b>14447.23</b>	<b>23183.45</b>	<b>.623169977</b>
<b>Internet</b>	<b>345.02</b>	<b>23183.45</b>	<b>.014882168</b>
<b>Partners</b>	<b>8391.2</b>	<b>23183.45</b>	<b>.361947855</b>

# Window Frame

- Syntax:  
**WFctType(ValueExpr)**  
**OVER ([WPartitioning] [WOrdering] [WFrame])**
- Window frames are used to compute cumulative, moving and centered aggregates.
- Window frames return a value for each row that depends on the other rows in the window.
- Window frames provide access to more than one row without a self join.
- `FIRST_VALUE` and `LAST_VALUE` return the first and last value of the window, respectively.

# Window Frames/2

- Examples of window frame specifications:
  - **ROWS UNBOUNDED PRECEDING**
    - ◆ Takes all rows in the window up to and including the current row
  - **ROWS 2 PRECEDING**
    - ◆ Takes the 2 preceding rows
  - **RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND INTERVAL '1' DAY FOLLOWING**
    - ◆ Takes all rows that fall within the given logical offset (wrt the expression in the ordering clause). In this example rows with a timestamp that differs by at most 1 day.
  - **RANGE BETWEEN INTERVAL '10' DAY PRECEDING AND CURRENT ROW**
    - ◆ Takes all rows with a timestamp that is at most 10 days before the timestamp of the current row.

# Centered Aggregate Example

```
SELECT t_id, SUM(s_amnt_sold) AS SALES,  
        AVG(SUM(s_amnt_sold)) OVER (ORDER BY t_id  
                                     RANGE BETWEEN INTERVAL '1' DAY PRECEDING  
                                     AND          INTERVAL '1' DAY FOLLOWING)  
FROM    bi.spctmn  
WHERE   t_cal_week_num = 51  
AND     t_cal_year=1999  
GROUP BY t_id  
ORDER BY t_id;
```

- The centered 3 day moving average of all sales during week 51 in 1999.

# Centered Aggregate Example/2

<b>T_ID</b>	<b>SALES</b>	<b>CENTERED_3_DAY_AVG</b>
20-DEC-99	134336.84	106675.93
21-DEC-99	79015.02	102538.713
22-DEC-99	94264.28	85341.7533
23-DEC-99	82745.96	93322.3067
24-DEC-99	102956.68	82936.7
25-DEC-99	63107.46	87062.2167
26-DEC-99	95122.51	79114.985

# Ranking Example

- Rewrite the following statement to a semantically equivalent one that does not use the RANK function

```
SELECT    m_desc, t_cal_month_desc,  
          RANK() OVER (ORDER BY SUM(s_amnt_sold) DESC) AS rank  
FROM      bi.spctmn  
WHERE     t_cal_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-11')  
AND       m_desc IN ('Direct sales', 'Internet')  
GROUP BY m_desc, t_cal_month_desc;
```

# LAG and LEAD

- LAG and LEAD give access to rows that are at a certain distance from the current row.
- Report with amounts sold between 10.8.2000 and 14.8.2000. Include with each row the amount of the previous and following day.

```
SELECT    s_t_id, SUM(s_amnt_sold),  
          LAG(SUM(s_amnt_sold),1) OVER (ORDER BY s_t_id),  
          LEAD(SUM(s_amnt_sold),1) OVER (ORDER BY s_t_id)  
FROM      bi.spctmn  
WHERE     s_t_id >= TO_DATE('10-OCT-2000')  
AND       s_t_id <= TO_DATE('14-OCT-2000')  
GROUP BY s_t_id;
```

# LAG and LEAD Example/2

<b>S_T_ID</b>	<b>SUM(S_AMNT</b>	<b>LAG1</b>	<b>LEAD1</b>
-----	-----	-----	-----
<b>10-OCT-00</b>	<b>238479.49</b>		<b>23183.45</b>
<b>11-OCT-00</b>	<b>23183.45</b>	<b>238479.49</b>	<b>24616.04</b>
<b>12-OCT-00</b>	<b>24616.04</b>	<b>23183.45</b>	<b>76515.61</b>
<b>13-OCT-00</b>	<b>76515.61</b>	<b>24616.04</b>	<b>29794.78</b>
<b>14-OCT-00</b>	<b>29794.78</b>	<b>76515.61</b>	

# Densification: Problem Description

- Example data:

<b>PROD</b>	<b>YEAR</b>	<b>WEEK</b>	<b>SALES</b>
-----	-----	-----	-----
Deluxe	2001	25	5560
Mouse P	2001	24	2083
Mouse P	2001	26	2501
Standar	2001	24	2394
Standar	2001	26	1280

- Goal: produce dense report for weeks 24, 25, and 26.
- Can be important for reports or subsequent aggregations (3 months average), time series analysis, etc.

# Densification

- Data is often stored in sparse form.
- For reporting or analysis purposes it can make sense to selectively densify data.
- Data densification is the process of converting sparse data into dense form.
- The key technique is a **partitioned outer join**.
- A partitioned outer join extends the regular outer join, and applies the outer join to each partition.
- This allows to fill in values for the partitioned attributes.
- Note that a regular outer join permits no assumption about missing attribute values.

# Densification Example

```
SELECT p_Name, t.Year, t.Week, NVL(Sales,0) dense_sales
FROM ( SELECT      P_Name, T_Cal_Year Year, t_Cal_Week_num Week,
                SUM(S_Amnt_Sold) Sales
        FROM      bi.spctmn
        WHERE     p_name LIKE '%Mouse%'
        GROUP BY  p_Name, T_Cal_Year, t_Cal_Week_num ) v
PARTITION BY (v.p_Name)
RIGHT OUTER JOIN
( SELECT DISTINCT t_Cal_Week_num Week, T_Cal_Year Year
  FROM bi.spctmn
  WHERE T_Cal_Year IN (2000, 2001)
        AND t_Cal_Week_num BETWEEN 24 AND 26 ) t
ON (v.week = t.week AND v.Year = t.Year)
ORDER BY p_name, year, week;
```

# Densification Example/2

<b>PROD</b>	<b>YEAR</b>	<b>WEEK</b>	<b>DENSE_SALES</b>
<b>Deluxe</b>	<b>2000</b>	<b>24</b>	<b>0</b>
<b>Deluxe</b>	<b>2000</b>	<b>25</b>	<b>0</b>
<b>Deluxe</b>	<b>2000</b>	<b>26</b>	<b>0</b>
<b>Deluxe</b>	<b>2001</b>	<b>24</b>	<b>2260.72</b>
<b>Deluxe</b>	<b>2001</b>	<b>25</b>	<b>1871.3</b>
<b>Deluxe</b>	<b>2001</b>	<b>26</b>	<b>5560.51</b>
<b>Mouse P</b>	<b>2000</b>	<b>24</b>	<b>1685.52</b>
<b>Mouse P</b>	<b>2000</b>	<b>25</b>	<b>494.91</b>
<b>Mouse P</b>	<b>2000</b>	<b>26</b>	<b>1548.2</b>
<b>Mouse P</b>	<b>2001</b>	<b>24</b>	<b>2083.29</b>
<b>Mouse P</b>	<b>2001</b>	<b>25</b>	<b>0</b>
<b>Mouse P</b>	<b>2001</b>	<b>26</b>	<b>2501.79</b>
<b>Standar</b>	<b>2000</b>	<b>24</b>	<b>1007.37</b>
<b>Standar</b>	<b>2000</b>	<b>25</b>	<b>339.36</b>
<b>Standar</b>	<b>2000</b>	<b>26</b>	<b>183.92</b>
<b>Standar</b>	<b>2001</b>	<b>24</b>	<b>2394.04</b>
<b>Standar</b>	<b>2001</b>	<b>25</b>	<b>0</b>
<b>Standar</b>	<b>2001</b>	<b>26</b>	<b>1280.97</b>

# Reporting Example

- Use the reporting functions to determine the answers to the following queries:
  - Media that contributed with more than  $1/3$  to the total sales. Formulate with and without analytic functions.
  - For customer 6510 determine the 3 month moving average of sales (current month plus preceding two months) in 1999.
  - For each product category find the region in which it had maximum sales on Oct 11, 2001.
  - On October 11, 2000, find the 5 top-selling products for each product subcategory that contributes more than 20% of the sales within its category.

# Summary/1

---

- **Extensions of GROUP BY clause**
  - ROLLUP, CUBE, GROUPING SETS
  - GROUPING\_ID
- **Window Functions**
  - WFuncType(Expr) OVER (WPartition WOrder WFrame)
  - RANK, CUME, NTILE
  - RATIO\_TO\_REPORT, LAG, LEAD
  - CURRENT ROW AND INTERVAL '1' DAY FOLLOWING
- **Densification**
  - Partitioned outer join

# Summary/2

---

- ISO SQL:2003 has added a lot of support for OLAP operations.
- This was triggered by intensive data warehouse research during the last decade.
- Make sure you understand SQL. It is much more than syntax.
  - Start with example input and output tables
  - SQL code comes later
- SQL is not select-from-where.
- Grouping and aggregation is a major part of SQL.