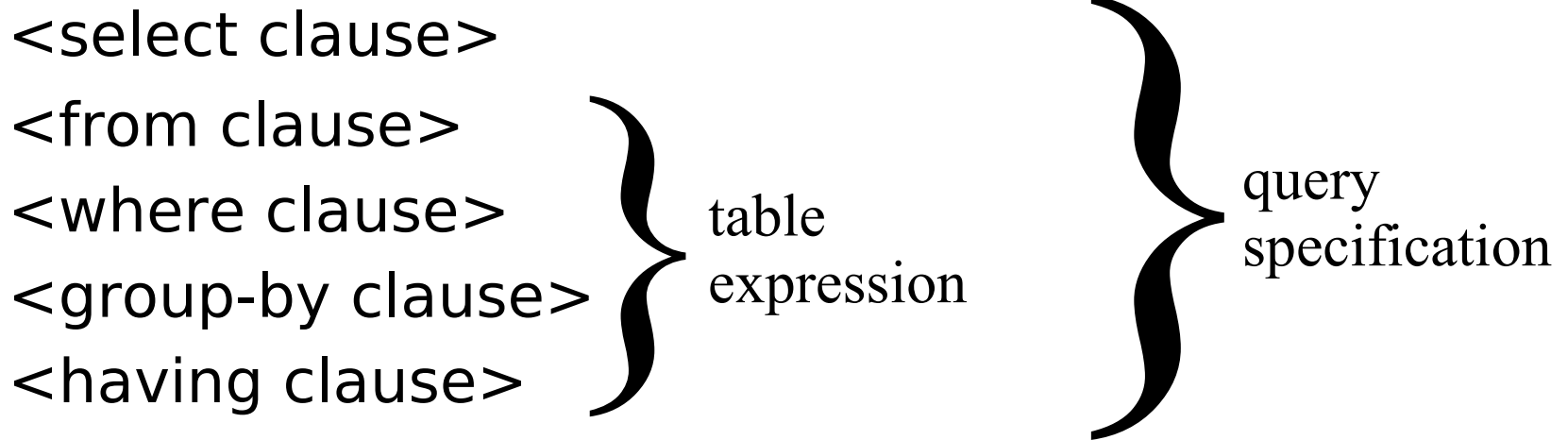# OLAP Operations

- SQL
  - table expression
  - query specification
  - query expression

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

Query Specifications

<select clause>

<from clause>

<where clause>   } table expression

<group-by clause>

<having clause>

} query specification

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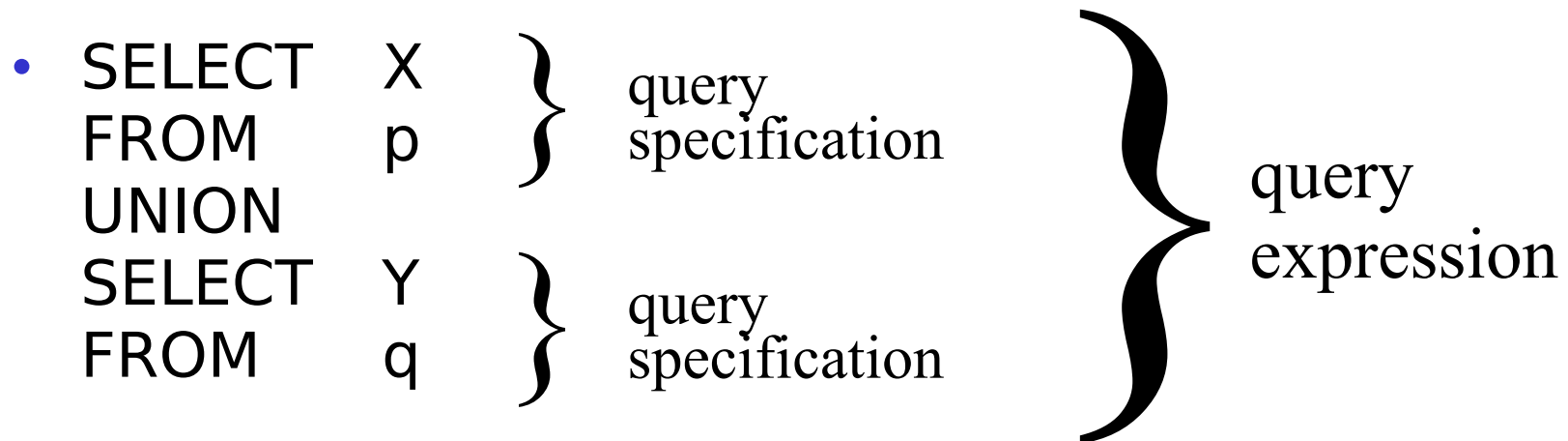
Table Expressions

- 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 ($\cap$)

- SELECT   X
  FROM   p   } query specification
  UNION
  SELECT   Y
  FROM   q   } query specification

  } query expression

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              FROM  ( SELECT ...
WHERE  ...                    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

Title	Type	Price
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
```

Title	Type	Price
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

Title	Type	Price
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
AND Type <> 'Western'
```

Title	Type	Price
Lethal Weapon	Action	3
Rocky	Action	2
Star Wars	Fiction	3

Having Clause Example/2

Title	Type	Price
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 funtions
  - 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.

Media	Country	Total
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.
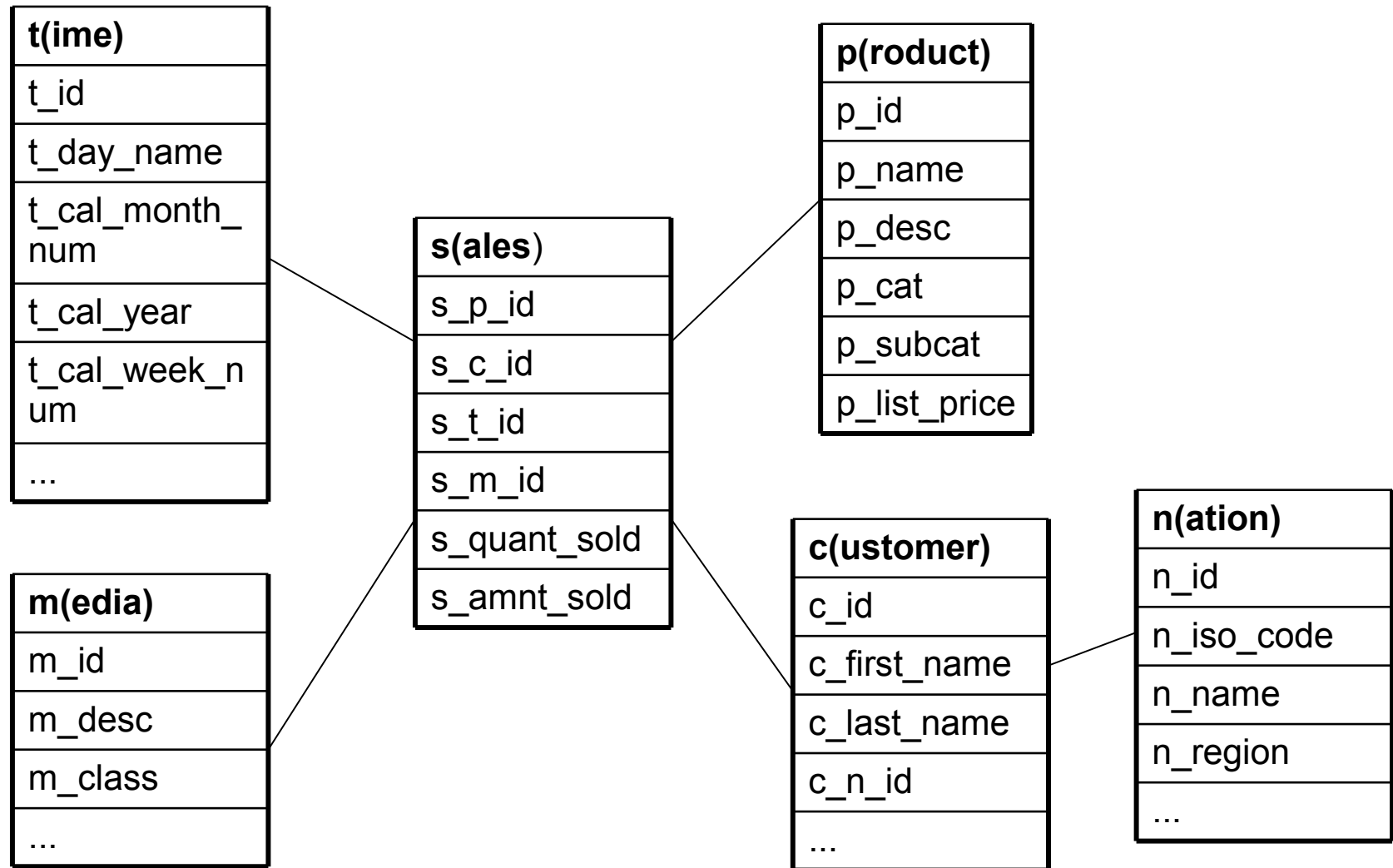
GROUP BY Extensions/2

- 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

t(ime)
| t_id |
| t_day_name |
| t_cal_month_ num |
| t_cal_year |
| t_cal_week_n um |
| ... |

s(ales)
| s_p_id |
| s_c_id |
| s_t_id |
| s_m_id |
| s_quant_sold |
| s_amnt_sold |

p(roduct)
| p_id |
| p_name |
| p_desc |
| p_cat |
| p_subcat |
| p_list_price |

m(edia)
| m_id |
| m_desc |
| m_class |
| ... |

c(ustomer)
| c_id |
| c_first_name |
| c_last_name |
| c_n_id |
| ... |

n(ation)
| n_id |
| n_iso_code |
| n_name |
| n_region |
| ... |

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

```
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
Internet      2000-09                 140793.11
Internet      2000-10                 151593.43
Direct Sales  2000-10                 774222.02
Direct Sales  2000-09                 723423.73
Internet                              292386.54
Direct Sales                         1497645.75
                                     1790032.29
```

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

```
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
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
```

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
```

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

M_DESC	T_CAL_MO	N_	SUM(S_AMOUNT_SOLD)
Internet	2000-09	GB	16569.36
Direct Sales	2000-09	GB	85222.92
Internet	2000-09	US	124223.75
Direct Sales	2000-09	US	638200.81
Internet	2000-10	GB	14539.14
Direct Sales	2000-10	GB	91925.43
Internet	2000-10	US	137054.29
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

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)

a	b	Bit vector	GROUPING_ID(a,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

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

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:
  - country-city, year-month-day
  - country/year, city/year, country/month, city/month, country/day, city/day

- In SQL we specify hierarchies explicitly. They are not "known" to the system.

Hierarchical Cube Example

- Complex OLAP queries are handled by enclosing the hierarchical cube in an outer query.

  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 Cube Example/2

- Query optimizers can be tuned to efficiently handle nested hierarchical cubes.
- The previous statement would be optimized to:

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 Funtions/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 sepcification:
  - 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 Funtions/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 Funtions/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 window (i.e., the rank is reset for each window).

RANK with Partitioning Example/2

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

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
  (rank of row - 1) / (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

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

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

```
M_DESC              SALES        TOTAL_SALES      RATIO
---------------- ---------      -----------    ----------
Direct Sales      14447.23       23183.45      .623169977
Internet            345.02       23183.45      .014882168
Partners           8391.2        23183.45      .361947855
```

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

```
T_ID            SALES    CENTERED_3_DAY_AVG
--------  ----------    ------------------
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
```

LAG and LEAD

- LAG and LEAD give access to rows that are at a certain distance form 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

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

Densification: Problem Description

- Example data:

```
PROD         YEAR      WEEK      SALES
-------   ----------   -------   ------
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).

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

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

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.