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

1. SQL OLAP Extensions

2. GROUP BY Extensions
   - ROLLUP
   - CUBE
   - GROUPING SETS
   - GROUPING
   - GROUPING_ID
Outline

1. SQL OLAP Extensions

2. GROUP BY Extensions
   - ROLLUP
   - CUBE
   - GROUPING SETS
   - GROUPING
   - GROUPING_ID
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)
Cross-tabular report with (sub)totals

<table>
<thead>
<tr>
<th>Media</th>
<th>Country</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>France</td>
<td>USA</td>
</tr>
<tr>
<td>Internet</td>
<td>9,597</td>
<td>124,224</td>
</tr>
<tr>
<td>Sales</td>
<td>61,202</td>
<td>638,201</td>
</tr>
<tr>
<td>Total</td>
<td>70,799</td>
<td>762,425</td>
</tr>
</tbody>
</table>

- This is space efficient for dense data only (thus, few dimensions).
- Shows data at four 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.

<table>
<thead>
<tr>
<th>Media</th>
<th>Country</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>France</td>
<td>9,597</td>
</tr>
<tr>
<td>Internet</td>
<td>USA</td>
<td>133,821</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>France</td>
<td>61,202</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>USA</td>
<td>638,201</td>
</tr>
<tr>
<td>Internet</td>
<td>ALL</td>
<td>133,821</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>ALL</td>
<td>699,403</td>
</tr>
<tr>
<td>ALL</td>
<td>France</td>
<td>70,799</td>
</tr>
<tr>
<td>ALL</td>
<td>USA</td>
<td>762,425</td>
</tr>
<tr>
<td>ALL</td>
<td>ALL</td>
<td>833,224</td>
</tr>
</tbody>
</table>
Outline

1 SQL OLAP Extensions

2 GROUP BY Extensions
  - ROLLUP
  - CUBE
  - GROUPING SETS
  - GROUPING
  - GROUPING_ID
Aggregation is a fundamental part of OLAP and data warehousing.

Oracle provides the following extensions to the GROUP BY clause:
- CUBE and ROLLUP
- GROUPING SETS expression
- GROUPING functions

The CUBE, ROLLUP, and GROUPING SETS extensions
- make querying and reporting easier and faster, and
- produce a single result set that is equivalent to a UNION ALL of differently grouped rows.

GROUPING functions allow to identify the roll-up (grouping) level
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 gcols_1, CUBE(gcols_2)`
  - Partial roll-up
- `GROUP BY GROUPING SETS (gcols_1, ..., gcols_n)`
  - Explicit specification of roll-ups
- `GROUP BY groupings_1, groupings_2, ...`
  - Cross-product of groupings
- `SELECT ... GROUPING_ID(gcols) ...`
  - Identification of roll-up level
Example Sales Schema/1

- **t(ime)**
  - t_id
  - t_day_name
  - t_cal_month_num
  - t_cal_year
  - t_cal_week_num
  - ...

- **p(roduct)**
  - p_id
  - p_name
  - p_desc
  - p_cat
  - p_subcat
  - p_list_price
  - ...

- **s(ales)**
  - s_p_id
  - s_c_id
  - s_t_id
  - s_m_id
  - s_quant_sold
  - s_amnt_sold

- **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
  - ...
Example Sales Schema/2

- An instance is in Oracle DB on dukeflelet.inf.unibz.it
  - Available in schema bi
  - Write bi.s to access the sales table

- The following view `bi.tpch` is available

```sql
CREATE OR REPLACE VIEW bi.tpch 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;
```
To access the DB with sqlplus, log in to students.inf.unibz.it (or russel.inf.unibz.it)

- SQL*Plus is the most basic Oracle Database utility, with a basic command-line interface, commonly used by users, administrators, and programmers.
- Works also from your notebook if an sqlplus client is installed

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 admtdummy@dukefleet.inf.unibz.it:1521/duke.inf.unibz.it @q1
```

Start sqlplus for a dialog:

```
rlwrap sqlplus admtdummy@dukefleet.inf.unibz.it:1521/duke.inf.unibz.it
```
ROLLUP Example/1

SELECT m_desc, t_cal_month_desc, n_iso_code, SUM(s_amount_sold)
FROM bi.tpch
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 4 groupings:
  - m_desc, t_cal_month_desc, n_iso_code
  - m_desc, t_cal_month_desc
  - m_desc
  - -

- Note that the actual number of result tuples (i.e., groups) is typically different and depends on the number of different values of the grouping attributes.
### ROLLUP Example/2

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>T_CAL_MO</th>
<th>N_</th>
<th>SUM(S_amount_sold)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>GB</td>
<td>16569.36</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>US</td>
<td>124223.75</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>GB</td>
<td>14539.14</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>US</td>
<td>137054.29</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>GB</td>
<td>85222.92</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>US</td>
<td>638200.81</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>GB</td>
<td>91925.43</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>US</td>
<td>682296.59</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td></td>
<td>140793.11</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td></td>
<td>151593.43</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td></td>
<td>774222.02</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td></td>
<td>723423.73</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td></td>
<td>292386.54</td>
</tr>
<tr>
<td>Direct Sales</td>
<td></td>
<td></td>
<td>1497645.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1790032.29</td>
</tr>
</tbody>
</table>
Partial ROLLUP Example/1

SELECT m_desc, t_cal_month_desc, n_iso_code, SUM(s_amount_sold)
FROM bi.tpch
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

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

- ROLLUP creates subtotals at \( n + 1 \) levels (i.e., \( n + 1 \) groupings), 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)
  - ROLLUP(country, state, city)
CUBE Example/1

SELECT  m_desc, t_cal_month_desc, n_iso_code, SUM(s_amount_sold)
FROM    bi.tpch
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.
  - 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
  - t_cal_month_desc
  - n_iso_code
  - -
**CUBE Example/2**

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>T_CAL_MO</th>
<th>N_</th>
<th>SUM(S_AMOUNT_SOLD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>GB</td>
<td>16569.36</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>US</td>
<td>124223.75</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>GB</td>
<td>14539.14</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>US</td>
<td>137054.29</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>GB</td>
<td>85222.92</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>US</td>
<td>638200.81</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>GB</td>
<td>91925.43</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>US</td>
<td>682296.59</td>
</tr>
<tr>
<td></td>
<td>2000-09</td>
<td>GB</td>
<td>101792.28</td>
</tr>
<tr>
<td></td>
<td>2000-09</td>
<td>US</td>
<td>762424.56</td>
</tr>
<tr>
<td></td>
<td>2000-10</td>
<td>GB</td>
<td>106464.57</td>
</tr>
<tr>
<td></td>
<td>2000-10</td>
<td>US</td>
<td>819350.88</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td>GB</td>
<td>31108.50</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td>US</td>
<td>261278.04</td>
</tr>
<tr>
<td>Direct Sales</td>
<td></td>
<td>GB</td>
<td>177148.35</td>
</tr>
<tr>
<td>Direct Sales</td>
<td></td>
<td>US</td>
<td>1320497.4</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>GB</td>
<td>140793.11</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>GB</td>
<td>151593.43</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>GB</td>
<td>723423.73</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>GB</td>
<td>774222.02</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td>GB</td>
<td>292386.54</td>
</tr>
<tr>
<td>Direct Sales</td>
<td></td>
<td>GB</td>
<td>1497645.75</td>
</tr>
<tr>
<td></td>
<td>2000-09</td>
<td>GB</td>
<td>864216.84</td>
</tr>
<tr>
<td></td>
<td>2000-10</td>
<td>GB</td>
<td>925815.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GB</td>
<td>208256.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US</td>
<td>1581775.44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1790032.29</td>
</tr>
</tbody>
</table>
CUBE creates \(2^n\) combinations of subtotals (i.e., groupings), 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.
GROUP BY Extensions

GROUPING SETS Example/1

```
SELECT m_desc, t_cal_month_desc, n_iso_code, SUM(s_amount_sold)
FROM bi.tpch
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

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>T_CAL_MO</th>
<th>N</th>
<th>SUM(S_AMOUNT_SOLD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>GB</td>
<td>16569.36</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>GB</td>
<td>85222.92</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>US</td>
<td>124223.75</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>US</td>
<td>638200.81</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>GB</td>
<td>14539.14</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>GB</td>
<td>91925.43</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>US</td>
<td>137054.29</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>US</td>
<td>682296.59</td>
</tr>
<tr>
<td></td>
<td>2000-09</td>
<td>GB</td>
<td>101792.28</td>
</tr>
<tr>
<td></td>
<td>2000-09</td>
<td>US</td>
<td>762424.56</td>
</tr>
<tr>
<td></td>
<td>2000-10</td>
<td>GB</td>
<td>106464.57</td>
</tr>
<tr>
<td></td>
<td>2000-10</td>
<td>US</td>
<td>819350.88</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td>GB</td>
<td>31108.5</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td>US</td>
<td>261278.04</td>
</tr>
<tr>
<td>Direct Sales</td>
<td></td>
<td>GB</td>
<td>177148.35</td>
</tr>
<tr>
<td>Direct Sales</td>
<td></td>
<td>US</td>
<td>1320497.4</td>
</tr>
</tbody>
</table>
Equivalences

- \( \text{CUBE}(a, b) \equiv \text{GROUPING SETS} ((a, b), (a), (b), (())) \)
- \( \text{ROLLUP}(a, b, c) \equiv \text{GROUPING SETS} ((a, b, c), (a, b), (a), (())) \)
- \( \text{GROUP BY GROUPING SETS}(a, b, c) \)
  \[ \equiv \text{GROUP BY } a \text{ UNION ALL GROUP BY } b \text{ UNION ALL GROUP BY } c \]
- \( \text{GROUP BY GROUPING SETS}((a, b, c)) \equiv \text{GROUP BY } a, b, c \)
- \( \text{GROUP BY GROUPING SETS}(a, b, (b, c)) \)
  \[ \equiv \text{GROUP BY } a \text{ UNION ALL GROUP BY } b \text{ UNION ALL GROUP BY } b, c \]
- \( \text{GROUP BY GROUPING SETS}(a, \text{ROLLUP}(b, c)) \)
  \[ \equiv \text{GROUP BY } a \text{ UNION ALL GROUP BY } \text{ROLLUP}(b, c) \]
Identification of Groupings

- Two challenges arise with the use of ROLLUP and CUBE:
  1. How to programmatically determine which rows in the result set are subtotals?
  2. How to find the exact level of aggregation for a given subtotal?
- Subtotals are often needed to calculate percent-of-totals.
- Distinction between NULL values created by ROLLUP or CUBE and NULL values stored in the data is often important.
- SQL provides functions for this:
  - GROUPING
  - GROUPING_ID
GROUPING Function Example/1

```
SELECT m_desc, t_cal_month_desc, n_iso_code, SUM(s_amount_sold)
GROUPING(m_desc) AS M,
GROUPING(t_cal_month_desc) AS T,
GROUPING(n_iso_code) AS N
FROM bi.tpch
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);
```

- **GROUPING** has a single column as argument and returns
  - 1 when it encounters a NULL value created by a ROLLUP or CUBE operation;
    - NULL indicates that the row is a subtotal
  - 0 for any other type of value, including a stored NULL value.
### GROUPING Function Example/2

<table>
<thead>
<tr>
<th>M_DESC</th>
<th>T_CAL_MO</th>
<th>N_</th>
<th>SUM(S_amount_sold)</th>
<th>M</th>
<th>T</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>GB</td>
<td>16569.36</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td>US</td>
<td>124223.75</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>GB</td>
<td>14539.14</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td>US</td>
<td>137054.29</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>GB</td>
<td>85222.92</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td>US</td>
<td>638200.81</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>GB</td>
<td>91925.43</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td>US</td>
<td>682296.59</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-09</td>
<td></td>
<td>140793.11</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Internet</td>
<td>2000-10</td>
<td></td>
<td>151593.43</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-10</td>
<td></td>
<td>774222.02</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>2000-09</td>
<td></td>
<td>723423.73</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td></td>
<td>292386.54</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Direct Sales</td>
<td></td>
<td></td>
<td>1497645.75</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1790032.29</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Detail rows have ‘0 0 0’, first level subtotals ‘0 0 1’, second level subtotals ‘0 1 1’, and overall total ‘1 1 1’.
GROUPING_ID

- To find the GROUP BY level of a row, a query must return the GROUPING function information for each GROUP BY column.
  - Inconvenient and long SQL code
  - Extra storage space if stored
- The GROUPING_ID function addresses this problem.
  - 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 or CUBE, 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)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Bit vector</th>
<th>GROUPING_ID(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>NULL</td>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1 0</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>1 1</td>
<td>3</td>
</tr>
</tbody>
</table>
GROUPING_ID Example/1

```sql
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),
          GROUPING_ID(m_desc, n_iso_code) AS MN
FROM       bi.tpch
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

<table>
<thead>
<tr>
<th>CASEWHEN GROUPING(M_D)</th>
<th>CAS</th>
<th>SUM(S_AMOUNT_SOLD)</th>
<th>MN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>GB</td>
<td>16569.36</td>
<td>0</td>
</tr>
<tr>
<td>Internet</td>
<td>US</td>
<td>124223.75</td>
<td>0</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>GB</td>
<td>85222.92</td>
<td>0</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>US</td>
<td>638200.81</td>
<td>0</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>*</td>
<td>723423.73</td>
<td>1</td>
</tr>
<tr>
<td>Internet</td>
<td>*</td>
<td>140793.11</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>GB</td>
<td>101792.28</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>US</td>
<td>762424.56</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>864216.84</td>
<td>3</td>
</tr>
</tbody>
</table>
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 following 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:**
  
  ```sql
  GROUP BY GROUPING SETS((a),(b)), GROUPING SETS((c),(d))
  ```

  - Produces the groupings (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.
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:** Hierarchical cube across 3 dimensions

```
GROUP BY ROLLUP(year, quarter, month),
       ROLLUP(category,subcategory,name),
       ROLLUP(region,subregion,country,state,city)
```

- Groupings: (year,category,region),
  (year,quarter,category,region),
  (year,quarter,month,category,region), ...
- Produces a total of $4 \cdot 4 \cdot 6 = 96$ groupings.
- Compare to $2^{11} = 2048$ groupings by CUBE and 96 explicit group specifications.

In SQL we specify hierarchies explicitly. They are not “known” to the system.
Hierarchical Cubes/2

- Hierarchical 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 = 2002-01
AND gid = 3; -- grouping by month and division
```

- Bit vector for grouping by month and division: 0 0 0 0 1 1
Hierarchical Cubes/3

- Query optimizers are 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 groupings are removed (i.e., all groupings involving year, quarter, brand, and item)
- Optimized query:

```sql
SELECT month, division, sum_sales
FROM ( SELECT year, quarter, month, division, null, null
       SUM(sales) sum_sales,
       GROUPING_ID(grouping_columns) gid
       FROM sales, products, time
       GROUP BY year, quarter, month, division
     )
WHERE division = 25
AND month = 2002-01
AND gid = gid-for-Division-Month;
```
ISO SQL:2003 has added a lot of support for **OLAP operations**. This was triggered by intensive DW research during the last 1-2 decades.

**Grouping and aggregation** is a major part of SQL.

**OLAP extensions** of GROUP BY clause since SQL:2003

- **ROLLUP, CUBE, GROUPING SETS** are compact ways to express different groupings over the grouping attributes
  - ROLLUP generates results at \( n + 1 \) increasing levels of aggregation
  - CUBE generates results for all \( 2^n \) possible combinations of the grouping attributes
  - GROUPING SETS generates results exactly for the specified groupings

- **Composite columns** allow to skip certain levels of aggregation
- **Concatenated groupings** build the cross-product of a list of specified groupings (by ROLLUP, CUBE, etc.)
- **GROUPING** and **GROUPING_ID** allow to programmatically identify groupings

There exist a number of equivalence rules between the different forms of groupings.