Language SQL: operator Cube

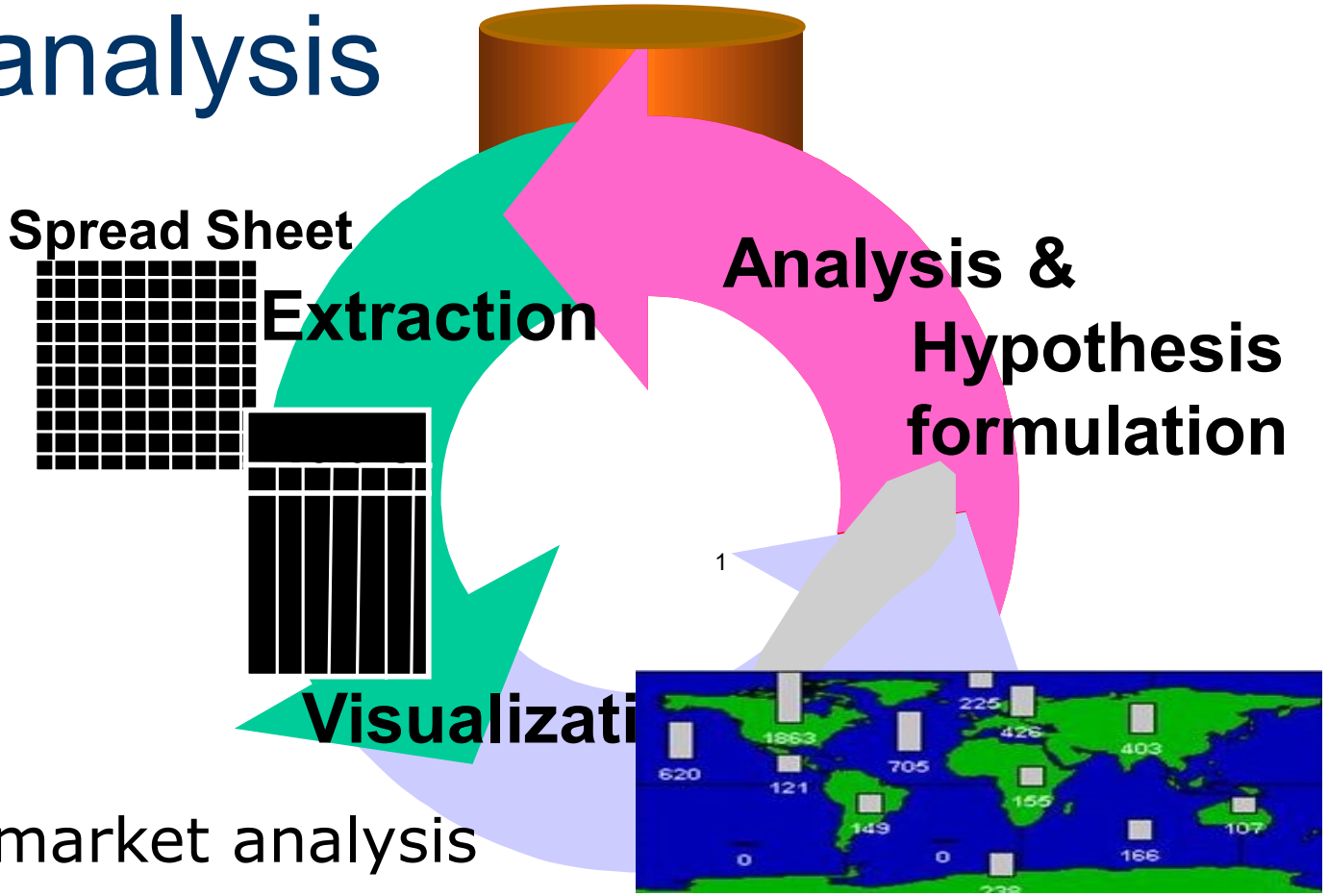
J. Pokorný
MFF UK

Content

- Motivation for CUBE operator
  - GROUP BY limits
  - how to do a
- CUBE and ROLLUP operators
- Conclusions

1

Data analysis



Ex.: Car market analysis

We are interested in the influence of the car model, colour, and year of production to the amount of sold cars. We do not care about the dealer and the date of sale.

OLAP

- OLAP (Online Analytical Processing)
- Principle of modelling: dimensions, facts
  - dimensions
    - can be hierarchical
    - have attributes
  - facts
    - attributes dependent on dimensions

Ex.: Car market

Dimensions: Model, Year, Colour

Fact: Amount of sold cars

OLAP

- n-dimensional data structures
- possibilities of representation:
  - one table for all
  - table for each dimension + table of facts
  - data cube
- evaluation:
    - aggregation functions COUNT, SUM, MAX, ...
    - operator GROUP BY

Problems with GROUP BY

- Simple queries: common aggregations like

```
SELECT Model, Country, SUM(Amount)
FROM Sale
GROUP BY Model, Country;
```

- More complex: Which model is a bestseller in Slovakia?
- Limits of aggregation constructions:
  - histograms
  - roll-up
  - cross-tables

Histograms

- Standard SQL has no statements for histograms construction

  Ex.: we have day weather-forecast, we want to aggregate days to weeks or months
- Histograms are computed using nested queries

Histograms

Modern SQL systems support histograms directly (it is not necessary to use nested queries as in SQL92)

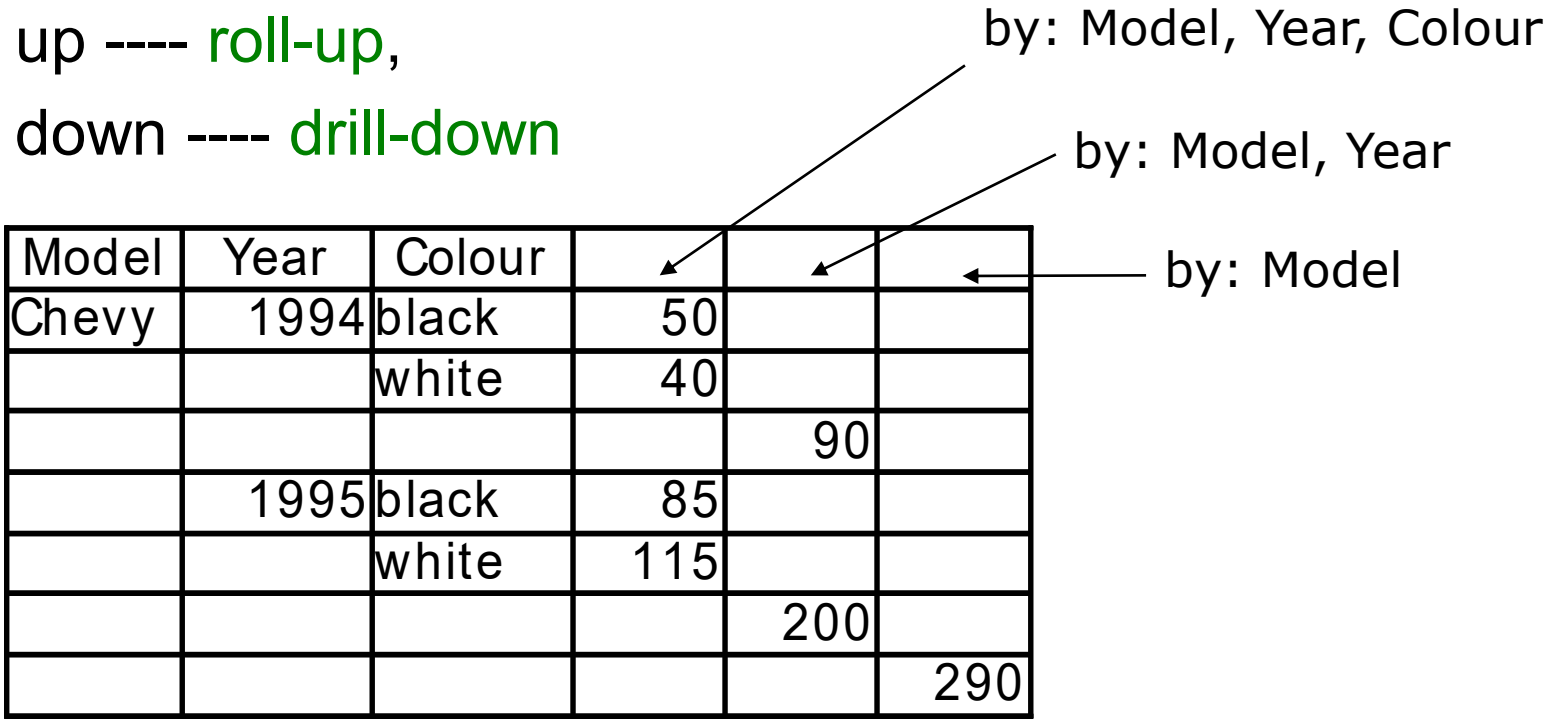
```
SELECT month, area, MIN(temperature)
FROM Weather
GROUP BY Month(time) AS month,
                Area(latitude, longitude) AS area
```

Roll-up, drill-down

- data can be aggregated into different dimensions levels
- we want to move through the levels

  up ---- roll-up,

  down ---- drill-down



Model	Year	Colour			
Chevy	1994	black	50		
		white	40		
				90	
	1995	black	85		
		white	115		
				200	
					290

Where to put aggregated values?

- Disadvantages of previous representation:
  - empty values in rows
  - it is not a relation
  - too many attributes (domains)
- Partial solution:
  - it is suitable to store aggregated values directly to the table
  - let us add columns which provide aggregated values for each row
  - disadvantage: it is out of the relational data model

Model	Year/Colour						Total
	1994		Total	1995		Total	
	black	white		black	white		
Chevy	50	40	90	85	115	200	290
Ford	50	10	60	85	75	160	220
Total	100	50	150	170	190	360	510

Where to put aggregated values?

Solution: relational representation

- special value ALL
- ALL means that we want to all values of a domain in this place.
- ALL() defines a set

  Ex.: ALL(Model)={Black, White}

Model	Year	Colour	Amount
Chevy	1994	black	50
Chevy	1994	white	40
Chevy	1994	ALL	90
Chevy	1995	black	85
Chevy	1995	white	115
Chevy	1995	ALL	200
Chevy	ALL	ALL	290

How to use SQL?

```
SELECT 'ALL', 'ALL', 'ALL', SUM(amount)
   FROM sale
   WHERE Model='Chevy'
UNION
SELECT Model, 'ALL', 'ALL', SUM(amount)
   FROM sale
   WHERE Model='Chevy'
   GROUP BY Model
UNION
SELECT Model, Year, 'ALL', SUM(amount)
   FROM sale
   WHERE Model='Chevy'
   GROUP BY Model, Year
UNION ...
```

- or several SELECT statements without ALL

Cross table

- Let us change relational representation and we obtain a cross table.
  - values of dimensions are placed in headings of rows and columns in a “two-dimensional space”
- construction in SQL: GROUP BY + UNION
- Problem: what for example Ford? The next table.

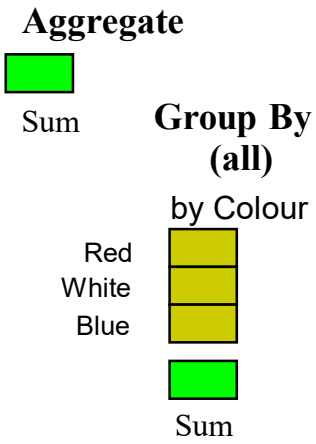
year

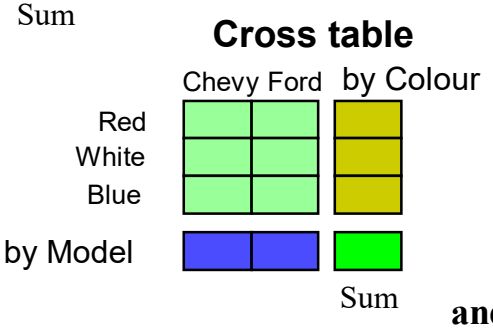
colour

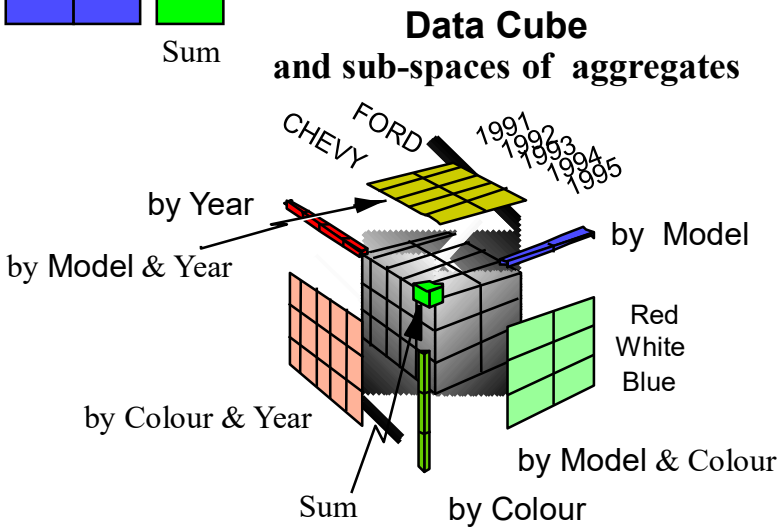
Chevy	1994	1995	ALL
black	50	85	135
white	40	115	155
ALL	90	200	290

Operators CUBE and ROLLUP

- solution: operators ROLLUP and CUBE
- generalization of GROUP BY, or cross table







CUBE – the first idea

- Ex.: we are constructing a data cube from three attributes
  - result is similar to real 3D cube *C*
  - edges of *C* represent the domains of attributes, cells content represent facts
  - each cell corresponds with one SQL group
  - we place aggregated value on each margin of *C;* it is constructed by application of GROUP BY operation in one dimension
  - we place the values aggregated by two dimensions on the edges of *C*, starting from the beginning of the cube
  - the super-aggregation (by all dimensions) is placed in the "origin" of the cube *C*
- Data cube is a multi-dimensional data model, where each domain contains a special value ALL.

CUBE – how it works

- Operator CUBE works like this:
  - it is equivalent to the collection of standard GROUP BY applications for all subsets of specified attributes (groupings),
  - super-aggregates are added to the result
- what is added: if there is N attributes, there are 2^N-1 aggregated values
- if $C_i = |dom(A_i)|$, $i \in <1,N>$, then the size of the cube is $\Pi(C_i + 1)$.
- in CUBE processing, aggregations are processed all-together in one operation for all cells
- Remark: MS SQL Server 2005 - CUBE was 2x faster than GROUP BY and UNION

Syntax

GROUP BY:

GROUP BY <all_attributes_to_aggregate>

<all_attributes_to_aggregate> ::=

{(<column_name> | <expression>)

[AS <name>]

,...}

Reduction of aggregation groups

- Sometimes it is useless to build the whole cube.
- Sometimes any combination of the attributes (dimensions) are unnecessary (example: application of CUBE to attributes day, month, year)
  - GROUPING SETS – grouping by a list
  - ROLLUP – only hierarchical aggregations

GROUPING SETS

- Ex.: Car market

  Dimensions: Model, Year, Colour

  Facts: Amounts of sold cars

- explicit list of of aggregations

```
SELECT Model, Colour, Country, SUM(Amount)
FROM Sale
GROUP BY GROUPING SETS ((),(Model),
                        (Colour, Country))
```

Operator ROLLUP

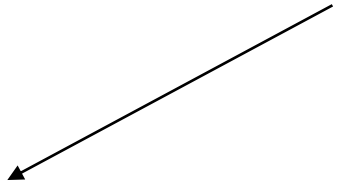
- operator ROLLUP is „low-cost“, it produces only the following aggregates

  $(v_1, v_2 \quad , \ldots , \quad v_k , \quad f()),$
  $(v_1, v_2 \quad , \ldots , \text{ALL}, \quad f()),$
  ...
  $(v_1, \text{ALL} , \ldots , \text{ALL}, \quad f()),$
  $(\text{ALL}, \text{ALL}, \ldots \quad , \text{ALL}, \quad f())$

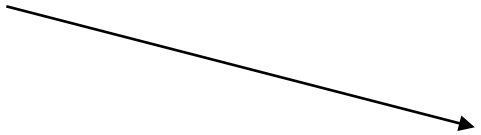
- Subsets with first attribute value ALL are not included into aggregation result (except the super-aggregate)
  - less results than the CUBE operator
  - not applicable for all queries solved by CUBE
    (Q.: „How many white cars were sold?“ .... does not work!)

Operator CUBE

Model	Country	Colour	Amount
Chevy	CZ	white	45
Chevy	CZ	yellow	18
Chevy	CZ	black	78
Chevy	SK	white	41
Chevy	SK	yellow	52
Chevy	SK	black	61
Ford	CZ	white	28
Ford	CZ	yellow	47
Ford	CZ	black	30
Ford	SK	white	21
Ford	SK	yellow	46



```
SELECT agg_amount = SUM(amount),
    Model, Country, Colour
FROM Sale
GROUP BY CUBE
(Model, Country, Colour);
```



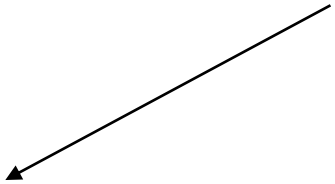
Operator CUBE

36 rows

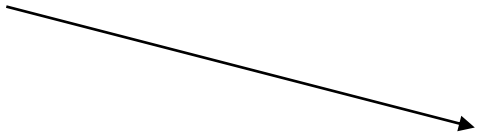
Agg_am	Model	Country	Colour
45	Chevy	CZ	white
18	Chevy	CZ	yellow
78	Chevy	CZ	black
141	Chevy	CZ	ALL
41	Chevy	SK	white
52	Chevy	SK	yellow
61	Chevy	SK	black
154	Chevy	SK	ALL
295	Chevy	ALL	ALL
28	Ford	CZ	white
47	Ford	CZ	yellow
30	Ford	CZ	black
105	Ford	CZ	ALL
21	Ford	SK	white
46	Ford	SK	yellow
8	Ford	SK	black
75	Ford	SK	ALL
180	Ford	ALL	ALL
475	ALL	ALL	ALL
73	ALL	CZ	white
65	ALL	CZ	yellow
108	ALL	CZ	black
246	ALL	CZ	ALL
62	ALL	SK	white
98	ALL	SK	yellow
69	ALL	SK	black
229	ALL	SK	ALL
86	Chevy	ALL	white
49	Ford	ALL	white
135	ALL	ALL	white
70	Chevy	ALL	yellow
93	Ford	ALL	yellow
163	ALL	ALL	yellow
139	Chevy	ALL	black
38	Ford	ALL	black
177	ALL	ALL	black

Operator ROLLUP

Model	Country	Colour	Amount
Chevy	CZ	white	45
Chevy	CZ	yellow	18
Chevy	CZ	black	78
Chevy	SK	white	41
Chevy	SK	yellow	52
Chevy	SK	black	61
Ford	CZ	white	28
Ford	CZ	yellow	47
Ford	CZ	black	30
Ford	SK	white	21
Ford	SK	yellow	46



```
SELECT agg_amount = SUM(amount),
    Model, Country, Colour
FROM Sale
GROUP BY ROLLUP
(Model, Country, Colour);
```



ROLLUP

Agg_am	Model	Country	Colour
45	Chevy	CZ	white
18	Chevy	CZ	yellow
78	Chevy	CZ	black
141	Chevy	CZ	ALL
41	Chevy	SK	white
52	Chevy	SK	yellow
61	Chevy	SK	black
154	Chevy	SK	ALL
295	Chevy	ALL	ALL
28	Ford	CZ	white
47	Ford	CZ	yellow
30	Ford	CZ	black
105	Ford	CZ	ALL
21	Ford	SK	white
46	Ford	SK	yellow
8	Ford	SK	black
75	Ford	SK	ALL

19 rows

180	Ford	ALL	ALL
475	ALL	ALL	ALL
~~73~~	~~ALL~~	~~CZ~~	~~white~~
~~65~~	~~ALL~~	~~CZ~~	~~yellow~~
~~108~~	~~ALL~~	~~CZ~~	~~black~~
~~246~~	~~ALL~~	~~CZ~~	~~ALL~~
~~62~~	~~ALL~~	~~SK~~	~~white~~
~~98~~	~~ALL~~	~~SK~~	~~yellow~~
~~69~~	~~ALL~~	~~SK~~	~~black~~
~~229~~	~~ALL~~	~~SK~~	~~ALL~~
~~86~~	~~Chevy~~	~~ALL~~	~~white~~
~~49~~	~~Ford~~	~~ALL~~	~~white~~
~~135~~	~~ALL~~	~~ALL~~	~~white~~
~~70~~	~~Chevy~~	~~ALL~~	~~yellow~~
~~93~~	~~Ford~~	~~ALL~~	~~yellow~~
~~163~~	~~ALL~~	~~ALL~~	~~yellow~~
~~139~~	~~Chevy~~	~~ALL~~	~~black~~
~~38~~	~~Ford~~	~~ALL~~	~~black~~
~~177~~	~~ALL~~	~~ALL~~	~~black~~

Relationships of GROUP BY, CUBE, and ROLLUP

- The following algebraic laws hold:
  - CUBE(ROLLUP) = CUBE
  - CUBE(GROUP BY) = CUBE
  - ROLLUP(GROUP BY) = ROLLUP
- Meaningful hierarchical order of the operators:

GROUP BY <attributes_to_aggregate>

ROLLUP <attributes_to_aggregate>

CUBE <attributes_to_aggregate>

Syntax

From CUBE to ROLLUP:

GROUP BY [<attributes_to_aggregate>]
[ROLLUP <attributes_to_aggregate>]
[CUBE <attributes_to_aggregate>]

- after GROUP BY it is allowed to use ROLLUP and CUBE
- each operator generates lists of attributes for aggregations (groups); then their Cartesian product included in the result

More aggregations

```
SELECT Model, Colour, Country, SUM(Amount)
FROM Sale
GROUP BY ROLLUP (Model),
   ROLLUP(Colour, Country)
```

generates groupings:

{Model, ()} X {(Colour, Country), (Colour), ()}

= { (Model, Colour, Country), (Model, Colour), (Model), (Colour, Country), (Colour), ( ) }

Value ALL

- problems with ALL as a special value:
  - many special cases
  - if ALL represents the set, then the remaining values of the domain have to be of simple types
- the implementations of ALL is therefore as follows:
  - it is used NULL instead of ALL
  - function ALL() is implemented
  - function GROUPING() is implemented to differentiate between NULL and ALL

Value ALL

- former: value ALL
- now: in data space the value NULL
- value TRUE in the corresponding field expresses that the NULL means ALL
- former : (ALL, ALL, ALL, 941)
- now : (NULL,NULL,NULL,941,TRUE,TRUE,TRUE)

GROUPING

- NULL value in the place of ALL is called grouping (grouping NULL)
- Function GROUPING differentiates grouping NULL value from normal (non-grouping) NULL
  - returns 1, if it is the grouping NULL (i.e. ALL)
  - returns 0, if it is the non-grouping NULL or there is a non-NULL value there.

GROUPING

- We can write:

```
SELECT Model, Year, Colour, SUM(Amount),
                GROUPING(Model),
                GROUPING(Year),
                GROUPING(Colour)
FROM Sale
GROUP BY CUBE Model, Year, Colour.
```

GROUPING()

- INSERT INTO Sale

  VALUES (NULL, ‘SK’, NULL, 229);
  - it is impossible to differentiate this new row from another one which express aggregations of CUBE
  - the only possibility is the GROUPING() function

GROUPING()

```
SELECT Agg_amount = SUM(Amount),
    Model, Colour, Country
FROM Sale
GROUP BY Model, Colour, Country
WITH CUBE;
```

Model	Country	Colour	Amount
NULL	SK	NULL	229
Chevy	CZ	white	45
Chevy	CZ	yellow	18
Chevy	CZ	black	78
Chevy	SK	white	41
Chevy	SK	yellow	52
Chevy	SK	black	61
Ford	CZ	white	28
Ford	CZ	yellow	47
Ford	CZ	black	30
Ford	SK	white	21
Ford	SK	yellow	46
Ford	SK	black	8

45	Chevy	white	CZ
41	Chevy	white	SK
86	Chevy	white	**NULL**
....			
229	**NULL**	**NULL**	SK
....			
229	**NULL**	**NULL**	SK

ALL ..... Grouping(Model) = 1

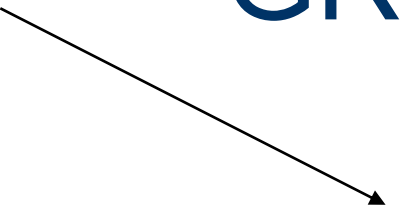
NULL ..... Grouping(Model) = 0

GROUPING()

```
SELECT Ag_amount = SUM(Amount),
   Model,
   'all_models'=grouping(Model),
   Country,
   'all_countries'=grouping(Country),
   Colour,
   'all_colours'=grouping(Colours)
FROM Sale
GROUP BY CUBE Model, Colour, Country;
```

Model	Country	Colour	Amount
NULL	SK	NULL	229
Chevy	CZ	white	45
Chevy	CZ	yellow	18
Chevy	CZ	black	78
Chevy	SK	white	41
Chevy	SK	yellow	52
Chevy	SK	black	61
Ford	CZ	white	28
Ford	CZ	yellow	47
Ford	CZ	black	30
Ford	SK	white	21
Ford	SK	yellow	46
Ford	SK	black	8

GROUPING()



ag_amount	Model	all_models	Country	all_countries	Colour	all_colours
45	Chevy	0	CZ	0	white	0
41	Chevy	0	SK	0	white	0
86	Chevy	0	NULL	1	white	0
...						
229	NULL	0	NULL	0	NULL	0
...						
229	NULL	1	SK	0	NULL	1

Non-standard: from NULL to ALL in T-SQL

```
SELECT Ag_amount = SUM(Amount),
Model,
Colour=ISNULL(Model,'ALL'),
Country
FROM Sale
GROUP BY
CUBE Model, Colour, Country;
```

Model	Country	Colour	Amount
NULL	SK	NULL	229
Chevy	CZ	white	45
Chevy	CZ	yellow	18
Chevy	CZ	black	78
Chevy	SK	white	41
Chevy	SK	yellow	52
Chevy	SK	black	61
Ford	CZ	white	28
Ford	CZ	yellow	47
Ford	CZ	black	30
Ford	SK	white	21
Ford	SK	yellow	46
Ford	SK	black	8

141	Chevy	NULL	CZ
154	Chevy	NULL	SK
295	Chevy	NULL	NULL
105	Ford	NULL	CZ
....			

Non-standard: from NULL to ALL in T-SQL

```
SELECT Units = SUM(Amount),
    Model = CASE WHEN (grouping(Model)=1) THEN 'ALL'
    ELSE ISNULL(Model, 'N/A')
    END,
    Country = CASE WHEN (grouping(Country)=1) THEN 'ALL'
    ELSE ISNULL(Country, 'N/A')
    END,
    colour = CASE WHEN (grouping(Colour)=1) THEN 'ALL'
    ELSE ISNULL(colour, 'N/A')
    END
FROM Sale
GROUP BY ROLLUP Model, Country, Colour
```

Nt.: N/A - Not-Applicable

Conclusions

- Operator CUBE generalizes and unifies:
  - aggregates
  - group by
  - roll-up and drill-down
  - cross tables
- Interesting problems:
  - evaluating CUBE for different aggregation functions
  - implementation (hashing, 2^N algorithm, CUBE algorithm)

Conclusions

- Operators CUBE and ROLLUP are standardized in SQL:1999.
- Creation of a data cube requires a special implementation.
- Querying strategy: restriction of queried data by specialized query (WHERE), then application of CUBE operator
- The next extension in practise: mainly Microsoft – MDX (**M**ulti**D**imensional E**X**pressions)