

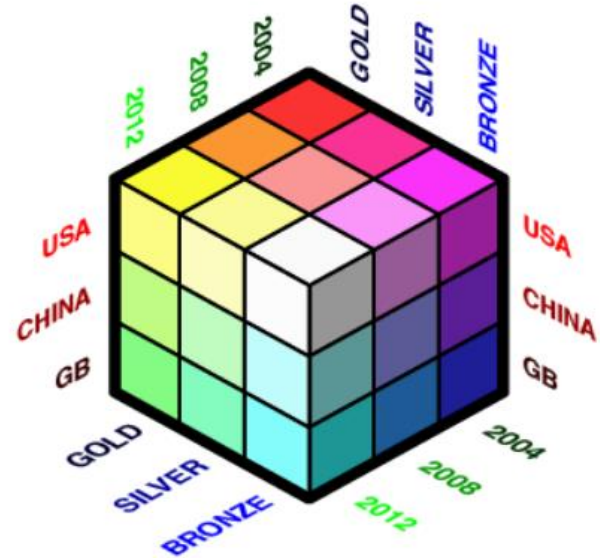
DATA CUBE : A RELATIONAL AGGREGATION OPERATOR GENERALIZING GROUP-BY, CROSS-TAB AND SUB-TOTALS

SNEHA REDDY BEZAWADA

CMPT 843

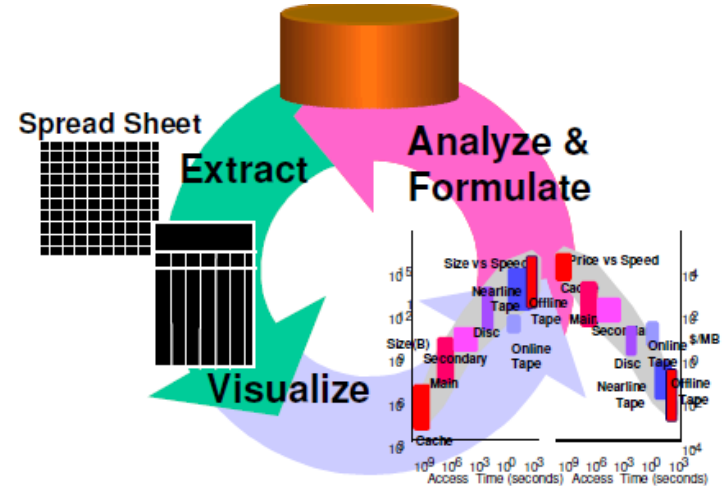
WHAT IS A DATA CUBE?

- The Data Cube or Cube operator produces N-dimensional answers by treating each N aggregate attributes as dimensions of an N-space.



WHY DO WE NEED DATA CUBES?

- Data Analysis applications
 - Summarization
 - Dimensionality reduction
- Multi-dimensional visualizations




CUBES IN RELATIONAL CONTEXT

- How to model N-dimension problem using relational tables?
- What are the operations supported by relational systems for multi-dimensional analysis?

N-DIMENSIONAL DATA AS RELATION

Dimensions

Measurements



Time	Latitude	Longitude	Altitude (m)	Temperature (c)	Pressure (mb)
96/6/1:500	37:58:33:N	122:45:28:W	102	21	1009
96/6/7:1500	34:16:18:N	27:05:55:W	10	23	1024

SQL AGGREGATE FUNCTIONS

- SQL standard provides 5 main aggregate functions – COUNT(), SUM(), AVG(), MIN(), MAX().
- GROUP BY construct to create aggregate values indexed by set of attributes.

```
SELECT AVG(Temp) FROM Weather;
```

```
SELECT Time, Altitude, AVG(Temperature)  
FROM Weather  
GROUP BY Time, Altitude;
```

PROBLEMS WITH GROUP BY

Three common problems with GROUP BY

- Histograms
- Roll-up Totals and Sub-Totals
- Cross-Tab

HISTOGRAMS

- Standard SQL does not allow direct computation of histograms – grouping by computed columns.



```
SELECT day, nation, MAX(Temp)
FROM Weather
GROUP BY Day(Time) as day,
Nation(Latitude, Longitude) as
nation ;
```



```
SELECT day, nation, MAX(Temp)
FROM (
  SELECT Day(Time) as day,
  Nation(Latitude, Longitude) as
  nation, Temp
FROM Weather
)AS foo
GROUP BY day, nation ;
```


ROLL UP AND DRILL DOWNS

- Displays data at multiple levels of aggregation.
- Drill down displays data at a level of increasing detail.
Ex: Month -> Week -> Day
- Roll ups displays data at a level of decreasing detail.
Ex: Day -> Month -> Year

ALL VALUE

'ALL' value introduced to fill in for super aggregate items.

Model	Year	Color	Units
Chevy	1991	Black	20
Chevy	1991	White	70
Chevy	1991	All	90
Chevy	1992	Black	50
Chevy	1992	White	30
Chevy	1992	All	80
Chevy	All	All	170

SQL STATEMENT FOR ROLLUP

```
SELECT 'ALL', 'ALL', 'ALL',  
SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
UNION  
SELECT Model, 'ALL', 'ALL',  
SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
GROUP BY Model  
UNION  
SELECT Model, Year, 'ALL', SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
GROUP BY Model, Year  
UNION  
SELECT Model, Year, Color, SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
GROUP BY Model, Year, Color;
```

This 3 dimensional roll up requires union of 4 SQL statements.

Aggregating over N dimensions require at least N such unions.

SQL STATEMENT FOR CROSS TAB

Chevy	1994	1995	Total
Black	50	80	130
White	40	20	60
Total	90	100	190

```
SELECT 'ALL', 'ALL', 'ALL',  
SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
UNION  
SELECT Model, 'ALL', 'ALL',  
SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
GROUP BY Model  
UNION  
SELECT Model, Year, 'ALL', SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
GROUP BY Model, Year  
UNION  
SELECT Model, 'ALL', Color,  
SUM(Sales) FROM Sales  
WHERE Model = 'Chevy'  
GROUP BY Model, Color;  
UNION  
SELECT Model, Year, Color, SUM(Sales)  
FROM Sales WHERE Model = 'Chevy'  
GROUP BY Model, Year, Color;
```

PROBLEMS WITH SQL REPRESENTATION

- SQL representation of Roll ups and Cross tabs is too complex.
- Difficult to optimize the aggregation queries.

CUBE OPERATOR

- Data cube is generalization of cross tab to N dimensions.
- Core of the cube is data.
- Lower dimensional aggregates appear as points, lines, planes, cubes etc.
- Data cube operator builds a table containing all these aggregate values.
- Cube operator is used in conjunction with SQL GROUP_BY

Aggregate



Sum

Group By (with total)

By Color

RED
WHITE
BLUE



Sum

Cross Tab

Chevy Ford By Color

RED
WHITE
BLUE

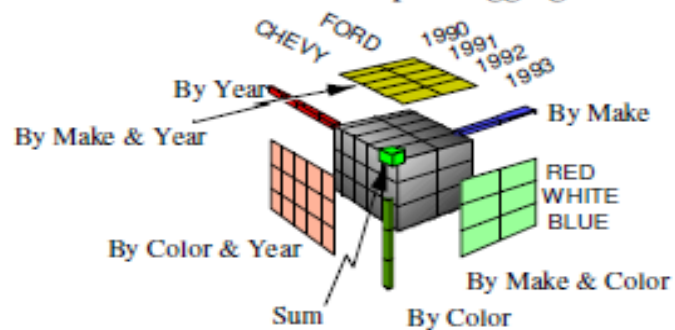


By Make



Sum

The Data Cube and The Sub-Space Aggregates



CUBE SYNTAX

```
SELECT Model, Year, Color,  
SUM(sales) AS Sales  
FROM Sales  
WHERE Model in {'Ford',  
'Chevy'}  
AND Year BETWEEN 1990 AND  
1991  
GROUP BY CUBE Model, Year,  
Color;
```


MODEL	YEAR	COLOR	SALES
Chevy	1990	Red	5
Chevy	1990	White	87
Chevy	1991	Red	54
Chevy	1991	White	95
Ford	1990	Red	64
Ford	1990	White	62
Ford	1991	Red	52
Ford	1991	White	9

CUBE



MODEL	YEAR	COLOR	SALES
Chevy	1990	Red	5
Chevy	1990	White	87
Chevy	1990	All	92
Chevy	1991	Red	54
Chevy	1991	White	95
Chevy	1991	All	149
Chevy	All	Red	59
Chevy	All	White	182
Chevy	All	All	241
Ford	1990	Red	64
Ford	1990	White	62
Ford	1990	All	126
Ford	1991	Red	52
Ford	1991	White	9
Ford	1991	All	61
Ford	All	Red	116
Ford	All	White	71
Ford	All	All	187
All	1990	Red	69
All	1990	White	149
All	1990	All	218
All	1991	Red	106
All	1991	White	104
All	1991	All	210
All	All	Red	175
All	All	White	253
All	All	All	428

CUBE SYNTAX

- CUBE operator aggregate over the select list attributes with SQL GROUP BY. Then it UNIONs all the super-aggregates by substituting ALL in aggregation columns.
- If the cardinality of N attributes is $C_1, C_2 \dots C_n$, the cardinality of resulting relation through cube operator is $\prod(C_i+1)$. Extra value is for 'ALL'

IS ALL VALUE NEEDED?

- ALL value return a set over which the aggregates were computed.
Ex: `Model.ALL = ALL(Model) = {Chevy, Ford}`
- There are rules which drive the usage 'ALL'. This adds substantial complexity.
 - ALL ALLOWED has to be added to column definition syntax
 - ALL does not participate in aggregations except COUNT.
 - Relational operators (`>`, `<`, `=`, `IN`) can be applied with ALL.

AVOIDING ALL - GROUPING FUNCTION

- GROUPING returns 1 when it encounters a NULL value created by a ROLLUP or CUBE operation. Any other type of value, including a stored NULL, returns a 0.

```
SELECT fact_1_id,  
       fact_2_id,  
       SUM(sales_value) AS sales_value,  
       GROUPING(fact_1_id) AS f1g,  
       GROUPING(fact_2_id) AS f2g  
FROM   dimension_tab  
GROUP BY CUBE (fact_1_id, fact_2_id)  
ORDER BY fact_1_id, fact_2_id;
```

FACT_1_ID	FACT_2_ID	SALES_VALUE	F1G	F2G
1	1	4363.55	0	0
1	2	4794.76	0	0
1	3	4718.25	0	0
1	4	5387.45	0	0
1	5	5027.34	0	0
1		24291.35	0	1
2	1	5652.84	0	0
2	2	4583.02	0	0
2	3	5555.77	0	0
2	4	5936.67	0	0
2	5	4508.74	0	0
2		26237.04	0	1
	1	10016.39	1	0
	2	9377.78	1	0
	3	10274.02	1	0
	4	11324.12	1	0
	5	9536.08	1	0
		50528.39	1	1

COMPUTING CUBES AND ROLLUPS

- Computing Cubes and Roll ups is similar to computing aggregates using Group-by.
- The result of cube operator is larger than group-by. The ALL value adds one extra value to each dimension of the cube.

IMPLEMENTATION OF AGGREGATE FUNCTIONS

- `start()` call allocates and initializes a scratch pad cell. The scratch pad stores the intermediate results of aggregation operation.
- `next()` call is invoked for each value to be aggregated
- `end()` calculates the final value from scratch pad results, deallocates and returns the result.



2^N - ALGORITHM

- Initialize a handle for each cell of the cube.
- For every row that arrives the `Iter(handle, v)` is called 2^N times, for each cell that matches the value `v`.
- Each attribute can take a value of itself or ALL. Hence it is called 2^N times.
- After all the rows are computed `final(&handle)` is called for each of $\prod(C_i+1)$ nodes of the cube.

TYPES OF AGGREGATE FUNCTIONS

DISTRIBUTIVE

- An aggregate function is distributive if the result obtained by applying the function on n aggregate values derived from data partitioned into n sets is the same as that derived by applying the function to the entire data set.

Ex: `count()`, `sum()`, `min()`, `max()`

DISTRIBUTIVE FUNCTIONS

- Cubes are easy to compute on distributive functions.
- N -1 dimension is computed by aggregating 1 dimension of N cube core.
- Lower dimension aggregates are calculated by aggregating over previously computed aggregates.

ALGEBRAIC FUNCTIONS

- An aggregate function is algebraic if it can be computed by an algebraic function with m arguments each of which is obtained by applying a distributive aggregate function.
- For example, `avg()` can be computed by `sum()/count()`, where both `sum()` and `count()` are distributive

ALGEBRAIC FUNCTIONS

- Cubes of algebraic functions are difficult to compute.
- It must maintain a handle for each input tuple and the result is obtained by calling `final()`.
- Once cube core is computed, the handles are passed to next level super aggregates and so on until (ALL, ..., ALL) aggregate has been computed.

HOLISTIC FUNCTIONS

- A function is said to be holistic if there is no algebraic function with M arguments that characterizes the computation.
- Ex: `median()`, `mode()`

HOLISTIC FUNCTIONS

- It is difficult to compute holistic function efficiently.
- However efficient techniques to approximate the computations of some holistic measures do exist. For example medians can be approximated using statistical techniques rather than computing them exactly.

MAINTAINING CUBES

- How to incrementally compute aggregate functions after cube is created and stored?
- Maintaining cube differs for SELECT, INSERT, UPDATE, DELETE operations.
- For Ex: MAX() is a distributive function. It is easy to compute MAX() on a cube as new data is being inserted into base table. But if a UPDATE or DELETE is performed on base table the entire cube has to be recomputed.

MAINTAINING CUBES

- If a function is algebraic or distributive for insert, update or delete , it is easy to maintain a cube.
- If a function is delete holistic like MAX() it very expensive to maintain the cube.

SUMMARY

- Cube operator computes aggregates in N dimensions.
- Standard SQL aggregation queries with GROUP BY is unsuitable for operations like Histograms, Roll-up, Drill-down, Cross-Tabs.
- Cube operator generalizes all these operations.
- Data cube is easy to compute for wide class of functions.



QUESTIONS?