

Adding measures to Calcite SQL

Julian Hyde (Google)

Apache Calcite virtual meetup, 2023-03-15



SQL vs BI

BI tools implement their own languages on top of SQL. Why not SQL?

Possible reasons:

- Semantic Model
- Control presentation / visualization
- Governance
- Pre-join tables
- Define reusable calculations
- Ask complex questions in a concise way

Processing BI in SQL

Why we should do it

- Move processing, not data
- Cloud SQL scale
- Remove data lag
- SQL is open

Why it's hard

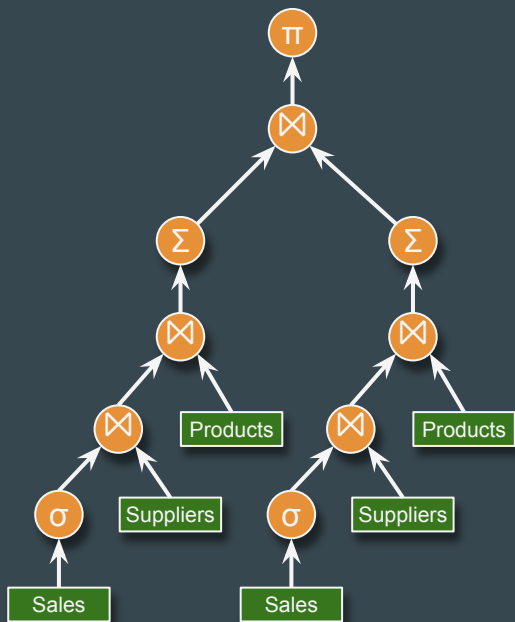
- Different paradigm
- More complex data model
- Can't break SQL

Pasta machine vs Pizza delivery

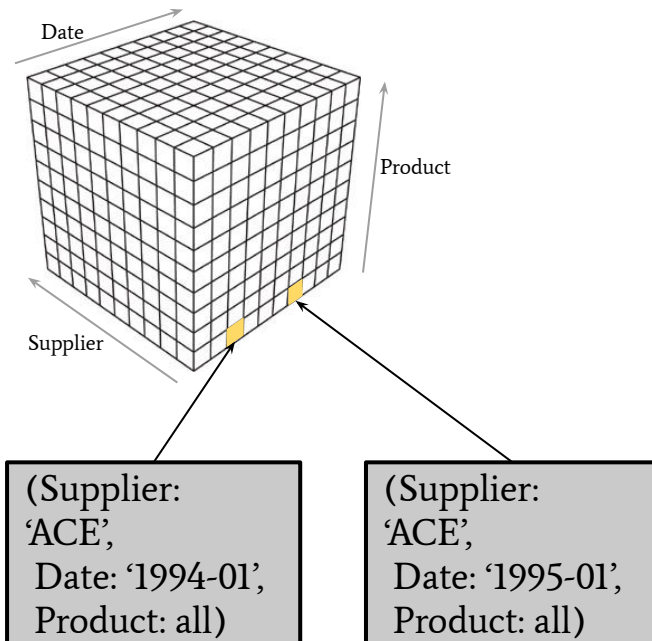


Bottom-up vs Top-down query

Relational algebra (bottom-up)



Multidimensional (top-down)



Some multidimensional queries

- Give the total sales for each product in each quarter of 1995. (Note that quarter is a function of date).
- For supplier “Ace” and for each product, give the fractional increase in the sales in January 1995 relative to the sales in January 1994.
- For each product give its market share in its category today minus its market share in its category in October 1994.
- Select top 5 suppliers for each product category for last year, based on total sales.
- For each product category, select total sales this month of the product that had highest sales in that category last month.
- Select suppliers that currently sell the highest selling product of last month.
- Select suppliers for which the total sale of every product increased in each of last 5 years.
- Select suppliers for which the total sale of every product category increased in each of last 5 years.

From [Agrawal1997]. Assumes a database with dimensions {supplier, date, product} and measure {sales}.)

Some multidimensional queries

- Give the total sales for each product in each quarter of 1995. (Note that quarter is a function of date).
- **For supplier “Ace” and for each product, give the fractional increase in the sales in January 1995 relative to the sales in January 1994.**
- For each product give its market share in its category today minus its market share in its category in October 1994.
- Select top 5 suppliers for each product category for last year, based on total sales.
- For each product category, select total sales this month of the product that had highest sales in that category last month.
- Select suppliers that currently sell the highest selling product of last month.
- Select suppliers for which the total sale of every product increased in each of last 5 years
- Select suppliers for which the total sale of every product category increased in each of last 5 years.

From [Agrawal1997]. Assumes a database with dimensions {supplier, date, product} and measure {sales}.)

Query:

- For supplier “Ace” and for each product, give the fractional increase in the sales in January 1995 relative to the sales in January 1994.

SQL

```
SELECT p.prodId,  
       s95.sales,  
       (s95.sales - s94.sales) / s95.sales  
FROM (  
  SELECT p.prodId, SUM(s.sales) AS sales  
  FROM Sales AS s  
  JOIN Suppliers AS u USING (suppId)  
  JOIN Products AS p USING (prodId)  
  WHERE u.name = 'ACE'  
  AND FLOOR(s.date TO MONTH) = '1995-01-01'  
  GROUP BY p.prodId) AS s95  
LEFT JOIN (  
  SELECT p.prodId, SUM(s.sales) AS sales  
  FROM Sales AS s  
  JOIN Suppliers AS u USING (suppId)  
  JOIN Products AS p USING (prodId)  
  WHERE u.name = 'ACE'  
  AND FLOOR(s.date TO MONTH) = '1994-01-01'  
  GROUP BY p.prodId) AS s94  
USING (prodId)
```

MDX

```
WITH MEMBER [Measures].[Sales Last Year] =  
  ([Measures].[Sales],  
   ParallelPeriod([Date], 1, [Date].[Year]))  
  MEMBER [Measures].[Sales Growth] =  
    ([Measures].[Sales]  
     - [Measures].[Sales Last Year])  
    / [Measures].[Sales Last Year]  
SELECT [Measures].[Sales Growth] ON COLUMNS,  
       [Product].Members ON ROWS  
FROM [Sales]  
WHERE [Supplier].[ACE]
```


Query:

- For supplier “Ace” and for each product, give the fractional increase in the sales in January 1995 relative to the sales in January 1994.

SQL

```
SELECT p.prodId,  
       s95.sales,  
       (s95.sales - s94.sales) / s95.sales  
FROM (  
  SELECT p.prodId, SUM(s.sales) AS sales  
  FROM Sales AS s  
  JOIN Suppliers AS u USING (suppId)  
  JOIN Products AS p USING (prodId)  
  WHERE u.name = 'ACE'  
  AND FLOOR(s.date TO MONTH) = '1995-01-01'  
  GROUP BY p.prodId) AS s95  
LEFT JOIN (  
  SELECT p.prodId, SUM(s.sales) AS sales  
  FROM Sales AS s  
  JOIN Suppliers AS u USING (suppId)  
  JOIN Products AS p USING (prodId)  
  WHERE u.name = 'ACE'  
  AND FLOOR(s.date TO MONTH) = '1994-01-01'  
  GROUP BY p.prodId) AS s94  
USING (prodId)
```

SQL with measures

```
SELECT p.prodId,  
       SUM(s.sales) AS MEASURE sumSales,  
       sumSales AT (SET FLOOR(s.date TO MONTH)  
                   = '1994-01-01')  
       AS MEASURE sumSalesLastYear  
FROM Sales AS s  
  JOIN Suppliers AS u USING (suppId)  
  JOIN Products AS p USING (prodId))  
WHERE u.name = 'ACE'  
AND FLOOR(s.date TO MONTH) = '1995-01-01'  
GROUP BY p.prodId
```

Self-joins, correlated subqueries, window aggregates, measures

Window aggregate functions were introduced to save on self-joins.

Some DBs rewrite scalar subqueries and self-joins to window aggregates [Zuzarte2003].

Window aggregates are more concise, easier to optimize, and often more efficient.

However, window aggregates can only see data that is from the same table, and is allowed by the **WHERE** clause.

Measures overcome that limitation.

```
SELECT *  
FROM Employees AS e  
WHERE sal > (  
    SELECT AVG(sal)  
    FROM Employees  
    WHERE deptno = e.deptno)
```



```
SELECT *  
FROM Employees AS e  
WHERE sal > AVG(sal)  
    OVER (PARTITION BY deptno)
```

A measure is... ?

... a column with an aggregate function.

`SUM(sales)`

A measure is... ?

... a column with an aggregate function.

`SUM(sales)`

... a column that, when used as an expression, knows how to aggregate itself.

`(SUM(sales) - SUM(cost))
/ SUM(sales)`

A measure is... ?

... a column with an aggregate function.

```
SUM(sales)
```

... a column that, when used as an expression, knows how to aggregate itself.

```
(SUM(sales) - SUM(cost))  
/ SUM(sales)
```

... a column that, when used as expression, can evaluate itself in any context.

```
(SELECT SUM(forecastSales)  
FROM SalesForecast AS s  
WHERE predicate(s))
```

```
ExchService$ClosingRate(  
  'USD', 'EUR', sales.date)
```

A measure is...

... a column with an aggregate function.

```
SUM(sales)
```

... a column that, when used as an expression, knows how to aggregate itself.

```
(SUM(sales) - SUM(cost))  
/ SUM(sales)
```

... **a column that, when used as expression, can evaluate itself in any context.**

```
(SELECT SUM(forecastSales)  
FROM SalesForecast AS s  
WHERE predicate(s))
```

Its value depends on, and only on, the predicate placed on its dimensions.

```
ExchService$ClosingRate(  
  'USD', 'EUR', sales.date)
```

Table model

Tables are SQL's fundamental model.

The model is closed – queries consume and produce tables.

Tables are opaque – you can't deduce the type, structure or private data of a table.

```
SELECT MOD(deptno, 2) = 0 AS evenDeptno, avgSal2  
FROM Employees3  
WHERE deptno < 30
```


Table model

Tables are SQL's fundamental model.

The model is closed – queries consume and produce tables.

Tables are opaque – you can't deduce the type, structure or private data of a table.

```
SELECT MOD(deptno, 2) = 0 AS evenDeptno, avgSal2
FROM
  SELECT deptno, AVG(avgSal) AS avgSal2
  FROM
    SELECT deptno, job,
      AVG(sal) AS avgSal
    FROM Employees
    GROUP BY deptno, job
  GROUP BY deptno
WHERE deptno < 30
```

Table model with measures

We propose to allow any table and query to have measure columns.

The model is closed – queries consume and produce tables-with-measures.

Tables-with-measures are semi-opaque – you can't deduce the type, structure or private data, but you can evaluate the measure in any context that can be expressed as a predicate on the measure's dimensions.

```
SELECT e.deptno, e.job, d.dname, e.avgSal / e.deptAvgSal  
FROM AnalyticEmployees2  
  
AS e  
JOIN Departments AS d USING (deptno)  
WHERE d.dname <> 'MARKETING'  
GROUP BY deptno, job
```

Table model with measures

We propose to allow any table and query to have measure columns.

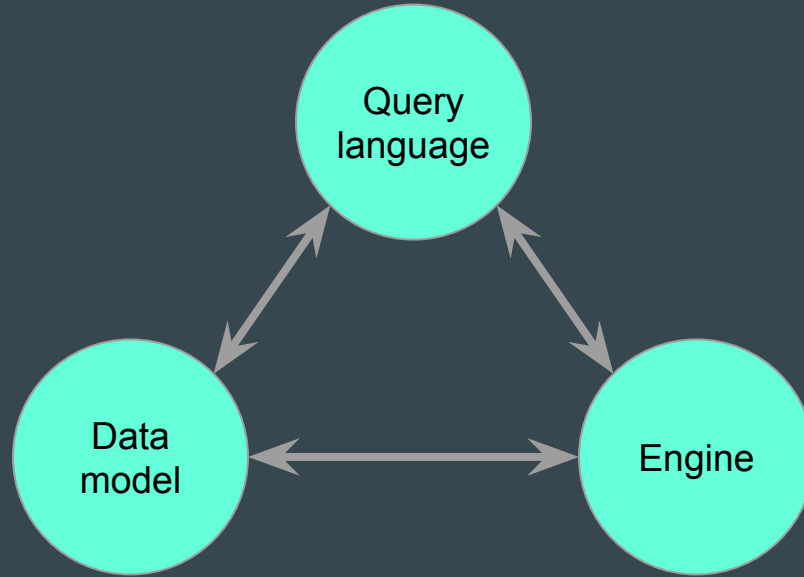
The model is closed – queries consume and produce tables-with-measures.

Tables-with-measures are semi-opaque – you can't deduce the type, structure or private data, but you can evaluate the measure in any context that can be expressed as a predicate on the measure's dimensions.

```
SELECT e.deptno, e.job, d.dname, e.avgSal / e.deptAvgSal
FROM
  SELECT *,
    avgSal AS MEASURE avgSal,
    avgSal AT (CLEAR deptno) AS MEASURE deptAvgSal
  FROM
    SELECT *,
      AVG(sal) AS MEASURE avgSal
    FROM Employees

  AS e
JOIN Departments AS d USING (deptno)
WHERE d.dname <> 'MARKETING'
GROUP BY deptno, job
```

Model + Query + Engine = Data system



Syntax

expression **AS MEASURE** – defines a measure in the **SELECT** clause

AGGREGATE(*measure*) – evaluates a measure in a **GROUP BY** query

expression **AT** (*contextModifier*...) – evaluates expression in a modified context

contextModifier ::=

CLEAR *dimension*

 | **SET** *dimension* = [**CURRENT**] *expression*

 | **VISIBLE**

 | **ALL**

aggFunction(*aggFunction*(*expression*) **PER** *dimension*) – multi-level aggregation

Plan of attack

1. Add measures to the table model, and allow queries to use them
 - ◆ Measures are defined only via the Table API
2. Define measures using SQL expressions (**AS MEASURE**)
 - ◆ You can still define them using the Table API
3. Context-sensitive expressions (**AT**)

Semantics

0. We have a measure M , value type V , in a table T .

1. System defines a row type R with the non-measure columns.

2. System defines an auxiliary function for M . (Function is typically a scalar subquery that references the measure's underlying table.)

```
CREATE VIEW AnalyticEmployees AS
  SELECT *, AVG(sal) AS MEASURE avgSal
  FROM Employees
```

```
CREATE TYPE R AS
  ROW (deptno: INTEGER, job: VARCHAR)
```

```
CREATE FUNCTION computeAvgSal(
  rowPredicate: FUNCTION<R, BOOLEAN>) =
  (SELECT AVG(e.sal)
   FROM Employees AS e
   WHERE APPLY(rowPredicate, e))
```


Semantics (continued)

3. We have a query that uses M .

```
SELECT deptno,  
       avgSal  
       / avgSal AT (CLEAR deptno)  
FROM AnalyticEmployees AS e  
GROUP BY deptno
```

4. Substitute measure references with calls to the auxiliary function with the appropriate predicate

```
SELECT deptno,  
       computeAvgSal(r  $\square$  (r.deptno = e.deptno))  
       / computeAvgSal(r  $\square$  TRUE))  
FROM AnalyticEmployees AS e  
GROUP BY deptno
```

5. Planner inlines `computeAvgSal` and scalar subqueries

```
SELECT deptno, AVG(sal) / MIN(avgSal)  
FROM (  
  SELECT deptno, sal,  
         AVG(sal) OVER () AS avgSal  
  FROM Employees)  
GROUP BY deptno
```

Calculating at the right grain

Example	Formula	Grain
Computing the revenue from units and unit price	<code>units * pricePerUnit</code> AS <code>revenue</code>	Row
Sum of revenue (additive)	<code>SUM(revenue)</code> AS <code>MEASURE sumRevenue</code>	Top
Profit margin (non-additive)	<code>(SUM(revenue) - SUM(cost))</code> / <code>SUM(revenue)</code> AS <code>MEASURE profitMargin</code>	Top
Inventory (semi-additive)	<code>SUM(LAST_VALUE(unitsInStock)</code> PER <code>inventoryDate)</code> AS <code>MEASURE sumInventory</code>	Intermediate
Daily average (weighted average)	<code>AVG(sumRevenue PER orderDate)</code> AS <code>MEASURE dailyAvgRevenue</code>	Intermediate

Subtotals & visible

```
SELECT deptno, job,  
       SUM(sal), sumSal  
FROM (  
  SELECT *,  
         SUM(sal) AS MEASURE sumSal  
  FROM Employees)  
WHERE job <> 'ANALYST'  
GROUP BY ROLLUP(deptno, job)  
ORDER BY 1,2
```

Measures by default sum ALL rows;
Aggregate functions sum only VISIBLE rows

deptno	job	SUM(sal)	sumSal
10	CLERK	1,300	1,300
10	MANAGER	2,450	2,450
10	PRESIDENT	5,000	5,000
10		8,750	8,750
20	CLERK	1,900	1,900
20	MANAGER	2,975	2,975
20		4,875	10,875
30	CLERK	950	950
30	MANAGER	2,850	2,850
30	SALES	5,600	5,600
30		9,400	9,400
		20,750	29,025

Visible

Expression	Example	Which rows?
Aggregate function	<code>SUM(sal)</code>	Visible only
Measure	<code>sumSal</code>	All
AGGREGATE applied to measure	<code>AGGREGATE(sumSal)</code>	Visible only
Measure with VISIBLE	<code>sumSal AT (VISIBLE)</code>	Visible only
Measure with ALL	<code>sumSal AT (ALL)</code>	All

Semantic models versus databases

Shouldn't the semantic model be *outside* the database?

(I don't want to be tied to one DBMS vendor.)

I have a great semantic model already. Why do I need a query language? My users don't want to write SQL.

What even *is* a semantic model?

In my opinion, a semantic model...

- ... is the place to share data and calculations
- ... needs a really good query language
 - (So you don't have to change the model every time someone has a new question)
- ... doesn't become a database just because it speaks SQL
- ... should do other things too
 - (Access control, governance, presentation defaults, guide data exploration, transform data, tune data, ...)

Summary

Concise queries without self-joins

Top-down evaluation

Reusable calculations

Doesn't break SQL

References

Papers

- [Agrawal1997] “Modeling multidimensional databases” (Agrawal, Gupta, and Sarawagi, 1997)
- [Zuzarte2003] “WinMagic: Subquery Elimination Using Window Aggregation” (Zuzarte, Pirahash, Ma, Cheng, Liu, and Wong, 2003)

Issues

- [CALCITE-4488] WITHIN DISTINCT clause for aggregate functions (experimental)
- [CALCITE-4496] Measure columns ("SELECT ... AS MEASURE")
- [CALCITE-5105] Add MEASURE type and AGGREGATE aggregate function
- [CALCITE-5155] Custom time frames
- [CALCITE-xxxx] PER
- [CALCITE-xxxx] AT

Thank you!
Any questions?

@julianhyde

@ApacheCalcite

<https://calcite.apache.org>

Slides and recording will be posted at @ApacheCalcite.



APACHE

calcite™