Measures in SQL

arxiv.org/abs/2406.00251

- Composable calculations are what is missing from SQL. We propose a new kind of column, called a measure, that attaches a calculation to a table.

- Like regular tables, tables with measures are composable and closed when used in queries.

- SQL-with-measures has the power,
  - conciseness
  - and reusability

  of multidimensional languages – but retains SQL semantics.

- Measure invocations can be expanded in place to simple, clear SQL.

Extending relational model

Measure calculations are holographic values defined as columns with **AS MEASURE**.

They are context sensitive expressions.

The **context transformation operator** AT modifies the evaluation context.

Syntax:

expression AT (*contextModifier*…)

contextModifier ::= WHERE *predicate*
| ALL
| ALL *dimension*
| SET *dimension* = [CURRENT] *expression*
| VISIBLE

Orders

prodName	custName	orderDate	revenue	cost
Happy	Alice	2023/11/28	6	4
Acme	Bob	2023/11/27	5	2
Happy	Alice	2024/11/28	7	4
Whizz	Celia	2023/11/25	3	1
Happy	Bob	2022/11/27	4	1

Composing measures

```
SELECT *,
  SUM(cost) AS MEASURE sumCost,
  SUM(revenue) AS MEASURE sumRevenue,
  (sumRevenue - sumCost) / sumRevenue
    AS MEASURE profitMargin,
  sumRevenue
    - sumRevenue AT (SET YEAR(orderDate)
        = CURRENT YEAR(orderDate) - 1)
    AS MEASURE revenueGrowthYoY,
  ARRAY_AGG(prodName
    ORDER BY sumRevenue DESC LIMIT 5)
      AT (ALL prodName)
      AS MEASURE top5Products,
  ARRAY_AGG(custName
    ORDER BY sumRevenue DESC LIMIT 3)
    AT (ALL custName
      SET productId MEMBER OF top5Products
        AT (SET YEAR(orderDate)
          = CURRENT YEAR(orderDate) - 1))
    AS MEASURE top3CustomersOfTop5Products
FROM Orders;
```

Grain locking

A measure is locked to the grain of its defining table.

Joining another table does not introduce double-counting.

Measure invocations can still be rewritten into SQL without measures via a semi-join (**WHERE EXISTS**).

Customers

custName	custAge
Alice	23
Bob	41
Celia	17

SELECT * FROM Orders JOIN Customers USING (custName)

prodName	custName	orderDate	revenue	cost	custAge
Happy	Alice	2023/11/28	6	4	23
Acme	Bob	2023/11/27	5	2	41
Happy	Alice	2024/11/28	7	4	23
Whizz	Celia	2023/11/25	3	1	17
Happy	Bob	2022/11/27	4	1	41

Alice appears twice affecting the average of custAge after the JOIN

JOIN query with measure

```
WITH EnhancedCustomers AS (
  SELECT *,
    AVG(custAge) AS MEASURE avgAge
  FROM Customers)
SELECT o.prodName,
  AVG(c.custAge) AS weightedAvgAge,
  c.avgAge AS avgAge
FROM Orders AS o
JOIN EnhancedCustomers AS c USING
(custName)
GROUP BY o.prodName;
```

Output: avgAge measure keeps grain

prodName	weightedAvgAge	avgAge
Acme	41	41
Happy	**29** (Alice counted twice)	32
Whizz	17	17

Measure rewritten to semi-JOIN

```
SELECT
  o.prodName,
  AVG(c.custAge) AS weightedAvgAge,
  (
    SELECT AVG(c2.custAge)
    FROM Customers AS c2
    WHERE
      EXISTS(
        SELECT TRUE FROM Orders AS o2
        WHERE c2.custName = o2.custName join condition
          AND o.prodName = o2.prodName  grouping key filter context
      )
  ) AS avgAge
FROM Orders AS o
JOIN Customers AS c
  USING (custName)
GROUP BY o.prodName
```

Rewriting

Implementing measures & context sensitive expressions as SQL rewrites

simple ↓ complex

Complexity	Query	Expanded query

Simple measure can be inlined	`SELECT prodName, avgRevenue`
	`FROM OrdersCube`
	`GROUP BY prodName`
		`SELECT prodName, AVG(revenue)`
		`FROM orders`
		`GROUP BY prodName`

Join requires grain-locking	`SELECT prodName, avgAge`
	`FROM OrdersCube`
	`GROUP BY prodName`
		`SELECT o.prodName,`
		`AVG(ANY_VALUE(c.custAge) GROUP BY c.custName)`
		`FROM orders JOIN customers GROUP BY prodName`
		→ (something with `GROUPING SETS`)

Period-over- period	`SELECT prodName, avgAge -`
	`avgAge AT (SET year =`
	`CURRENT year - 1)`
	`FROM OrdersCube`
	`GROUP BY prodName`
		(something with window aggregates)

Scalar subquery can accomplish anything	`SELECT prodName, prodColor`
	`avgAge AT (ALL custState`
	`SET year = CURRENT year - 1)`
	`FROM OrdersCube`
	`GROUP BY prodName, prodColor`
		`SELECT prodName, prodColor,`
		`(SELECT … FROM orders`
		`WHERE <evaluation context>)`
		`FROM orders`
		`GROUP BY prodName, prodColor`

Let's connect!

Topic ideas to chat about (more SQL innovations in the pipes!)

- Formal semantics for filter context transformations
- Using measures with LLM assistants
- Operators for managing grain
- Implementation strategies (efficient grain locking)
- Changing temporal grain for forecasts and timeseries
- Adding measures to nested structures typical in log files
- Sequential processing and measures

Julian Hyde
@julianhyde @ApacheCalcite
julianhyde@google.com

John Fremlin
@johnfremlin
fremlin@google.com