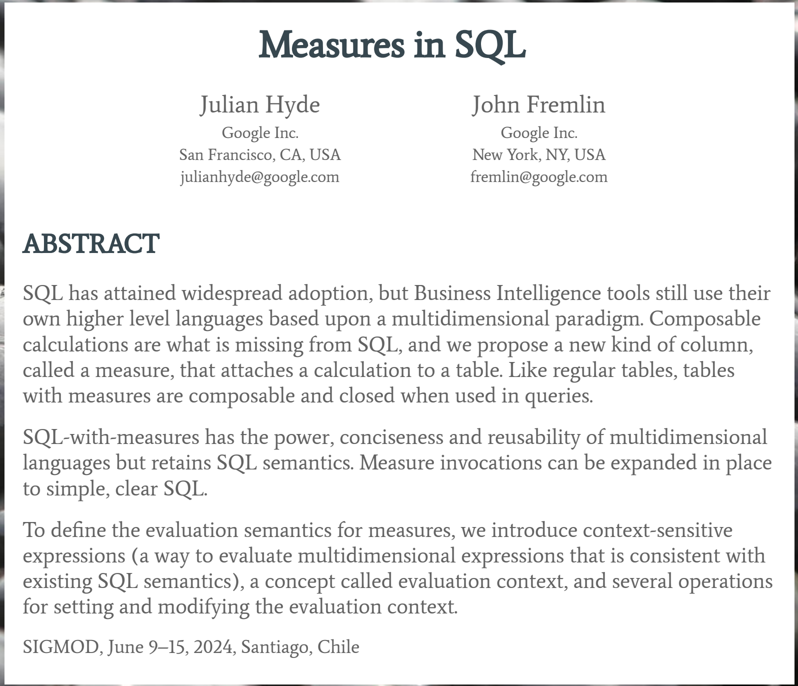
Measures in SQL

Julian Hyde (Google)

2024-05-22
SF Distributed Systems Meetup
San Francisco, California



Measures in SQL

Julian Hyde
Google Inc.
San Francisco, CA, USA
julianhyde@google.com

John Fremlin
Google Inc.
New York, NY, USA
fremlin@google.com

ABSTRACT

SQL has attained widespread adoption, but Business Intelligence tools still use their own higher level languages based upon a multidimensional paradigm. Composable calculations are what is missing from SQL, and 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.

To define the evaluation semantics for measures, we introduce context-sensitive expressions (a way to evaluate multidimensional expressions that is consistent with existing SQL semantics), a concept called evaluation context, and several operations for setting and modifying the evaluation context.

SIGMOD, June 9–15, 2024, Santiago, Chile

1. Problem

Tables are broken!

Tables are unable to provide reusable calculations.

Problem: Calculate profit margin of 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

```
SELECT prodName,
    (SUM(revenue) - SUM(cost))
      / SUM(revenue) AS profitMargin
FROM Orders
WHERE prodName = ‘Happy’;

profitMargin
============
0.47
```

Attempted solution: Create a view

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

```
CREATE VIEW SummarizedOrders AS
SELECT prodName, orderDate,
    (SUM(revenue) - SUM(cost))
      / SUM(revenue) AS profitMargin
FROM Orders
GROUP BY prodName, orderDate;
```

```
SELECT prodName,
    (SUM(revenue) - SUM(cost))
      / SUM(revenue) AS profitMargin
FROM Orders
WHERE prodName = ‘Happy’;

profitMargin
============
0.47
```

```
SELECT AVG(profitMargin) AS profitMargin
FROM SummarizedOrders
WHERE prodName = ‘Happy’;

profitMargin
============
0.50
```

2. Theory

Extend the relational model with measures

1. Allow tables to have measures

```
DESCRIBE EnhancedOrders;

column       type
============ =====
prodName     STRING
custName     STRING
orderDate    DATE
revenue      INTEGER
cost         INTEGER
profitMargin DOUBLE MEASURE
```

2. Operators for evaluating measures

```
SELECT prodName, profitMargin
FROM EnhancedOrders
GROUP BY prodName;

prodName profitMargin
======== ============
Acme             0.60
Happy            0.47
Whizz            0.67
```

3. Syntax to define measures in a query

```
SELECT *,
  (SUM(revenue) - SUM(cost)) / SUM(revenue)
   AS MEASURE profitMargin
FROM Orders
GROUP BY prodName;
```

Definitions

A **context-sensitive expression** (CSE) is an expression whose value is determined by an evaluation context.

An **evaluation context** is a predicate whose terms are one or more columns from the same table.

- This set of columns is the **dimensionality** of the CSE.

A **measure** is a special kind of column that becomes a CSE when used in a query.

- A measure's dimensionality is the set of non-measure columns in its table.
- The data type of a measure that returns a value of type *t* is *t* MEASURE, e.g. `INTEGER MEASURE`.

```
SELECT prodName,
    profitMargin
FROM EnhancedOrders
GROUP BY prodName;
```

`profitMargin` is a measure (and a CSE)

Dimensionality is {`prodName`, `custName`, `orderDate`, `revenue`, `cost`}

```
prodName profitMargin
======== ============
Acme             0.60
Happy            0.50
Whizz            0.67
```

Evaluation context for this cell is `prodName = 'Acme'`

```
SELECT (SUM(revenue) - SUM(
 / SUM(revenue) AS profitMargin
FROM Orders
WHERE prodName = 'Acme';

profitMargin
============
0.60
```

AT operator

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

Syntax:

expression `AT` (*contextModifier…*)

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

```
SELECT prodName,
    profitMargin,
    profitMargin
      AT (SET prodName = ‘Happy’)
      AS happyMargin,
    profitMargin
      AT (SET custName = ‘Bob’)
      AS bobMargin
FROM EnhancedOrders
GROUP BY prodName;
```

```
prodName profitMargin happyMargin bobMargin
======== ============ =========== =========
Acme             0.60        0.50      0.60
Happy            0.50        0.50      0.75
Whizz            0.67        0.50      NULL
```

```
SELECT (SUM(revenue) - SUM(cost))
    / SUM(revenue) AS m
FROM Orders
WHERE prodName = ‘Whizz’
AND custName = ‘Bob’;

m
====
NULL
```

Evaluation context for this cell is
`prodName = ‘Whizz’`
`AND custName = ‘Bob’`

3. Consequences

Grain-locking

What is the average age of the customer who would ordered each product?

When we use an aggregate function in a join query, it will 'double count' if the join duplicates rows.

This is generally not we want for measures – except if we want a weighted average – but is difficult to avoid in SQL.

Measures are locked to the grain of the table that defined them.

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

custName	custAge
Alice	23
Bob	41
Celia	17

```
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;

prodName weightedAvgAge avgAge
======== ============== ======
Acme                 41     41
Happy                29     32
Whizz                17     17
```

Alice (age 23) has two orders; Bob (age 41) has one order.

Composition & closure

Just as tables are closed under queries, so tables-with-measures are closed under queries-with-measures

Measures can reference measures

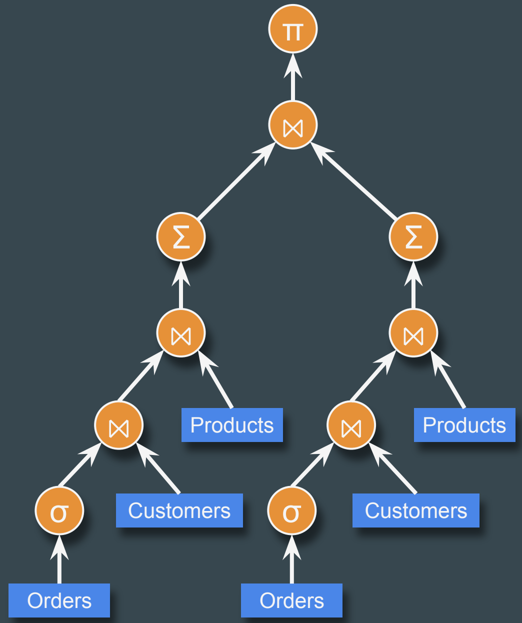
Complex analytical calculations without touching the `FROM` clause

Evaluation contexts can be nested

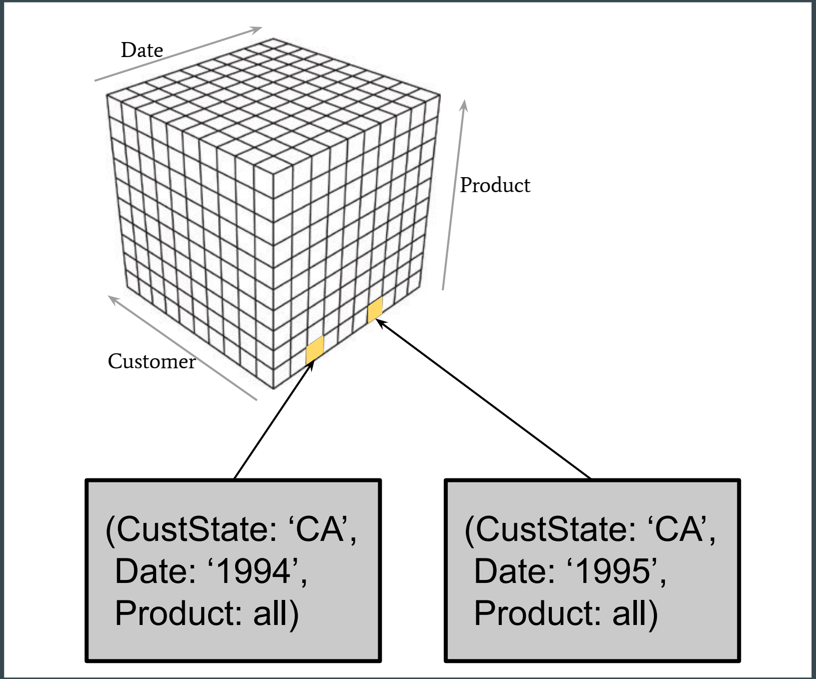
```
SELECT *,
  SUM(cost) AS MEASURE sumCost,
  SUM(revenue) AS MEASURE sumRevenue,
  (sumRevenue - sumCost) / sumRevenue
    AS MEASURE profitMargin,
  sumRevenue
    - sumRevenue AT (SET YEAR(orderDate)
        = CURRENT YEAR(orderDate) - 1)
    AS MEASURE revenueGrowthYoY,
  ARRAY_AGG(productId
 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

Bottom-up vs Top-down query

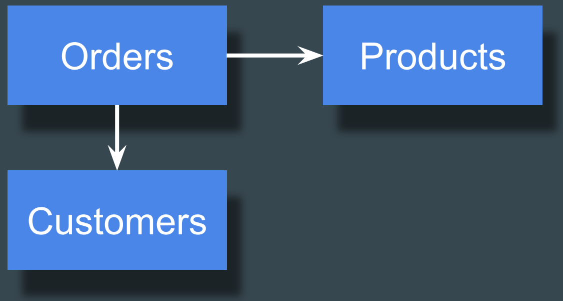
Relational algebra (bottom-up)



Multidimensional (top-down)


Date
Product
Customer
(CustState: ‘CA’, Date: ‘1994’, Product: all)
(CustState: ‘CA’, Date: ‘1995’, Product: all)

Represent a Business Intelligence model as a SQL view



- SQL planner handles view expansion
- Grain locking makes it safe to use a star schema
- Users can define new models simply by writing queries

```
CREATE VIEW OrdersCube AS
SELECT *
FROM (
  SELECT o.orderDate AS `order.date`,
    o.revenue AS `order.revenue`,
    SUM(o.revenue) AS MEASURE `order.sum_revenue`
  FROM Orders) AS o
LEFT JOIN (
  SELECT c.custName AS `customer.name`,
    c.state AS `customer.state`,
    c.custAge AS `customer.age`,
    AVG(c.custAge) AS MEASURE `customer.avg_age`
  FROM Customers) AS c
ON o.custName = c.custName
LEFT JOIN (
  SELECT p.prodName AS `product.name`,
    p.color AS `product.color`,
    AVG(p.weight) AS MEASURE `product.avg_weight`
  FROM Products) AS p
ON o.prodName = p.prodName;
```

```
SELECT `customer.state`, `product.avg_weight`
FROM OrdersCube
GROUP BY `customer.state`;
```

Implementing measures & CSEs as SQL rewrites

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(c.custAge PER 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`

Summary

Top-down evaluation makes queries concise

Measures make calculations reusable

Measures don’t break SQL

Thank you!
Any questions?

@julianhyde
@JohnFremlin
@ApacheCalcite
https://calcite.apache.org
https://issues.apache.org/jira/browse/CALCITE-4496
http://www.hydromatic.net/hyde2024measures.pdf