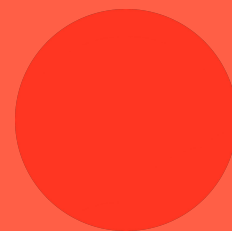


# Unity Catalog Metric Views



# Goal: Turn data & context into insights

## Lakehouse

Customer data



Website/  
application/  
logs



Marketing  
campaigns



Financial data



**Need:**  
Enterprise-wide,  
reliable and  
consistent  
business KPIs and  
semantics

## Data Consumers



Dashboards



SQL access:  
ODBC/JDBC/  
SQL REST



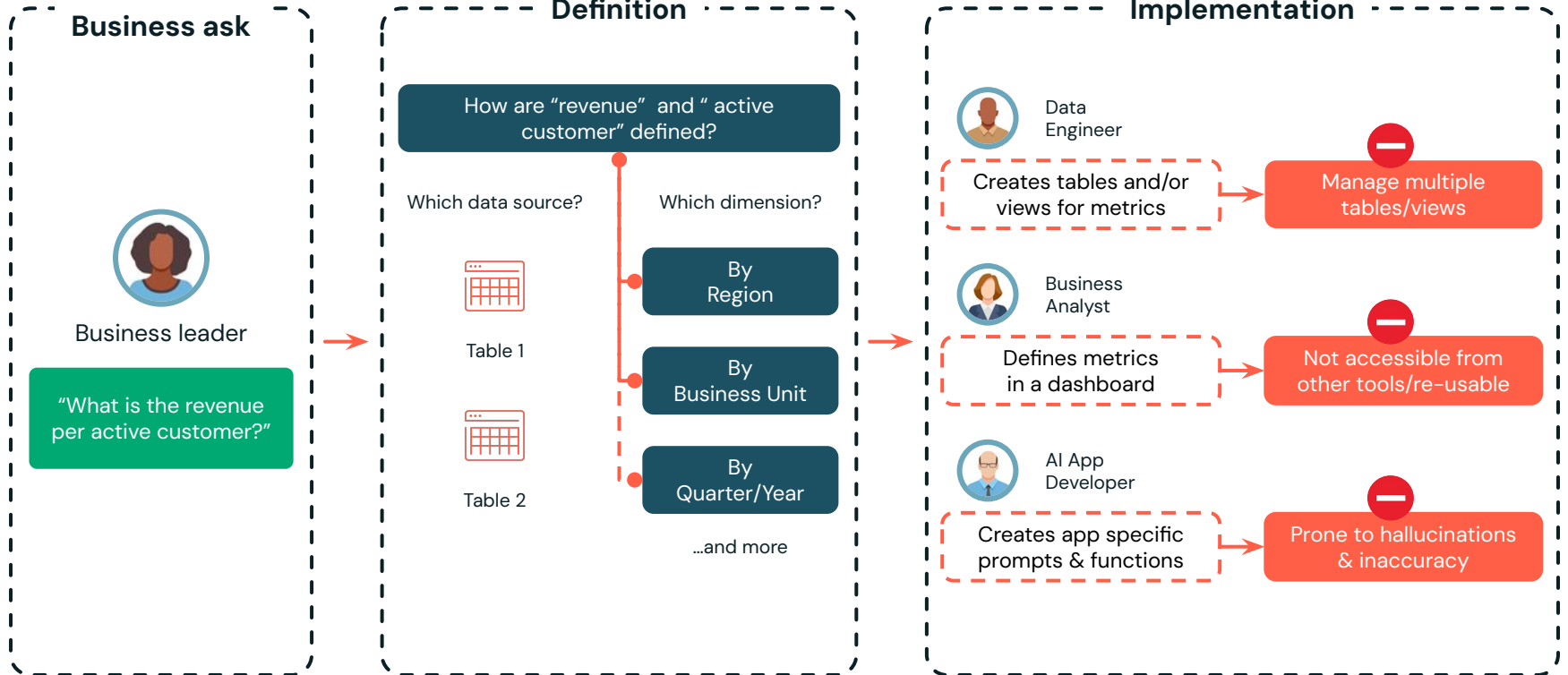
Notebooks



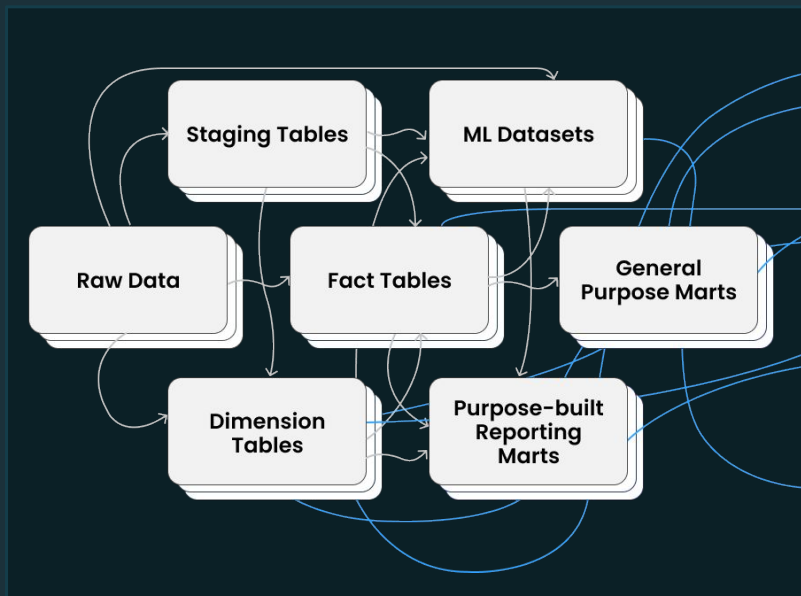
AI apps



# Hard in practice



# Getting the right number is hard

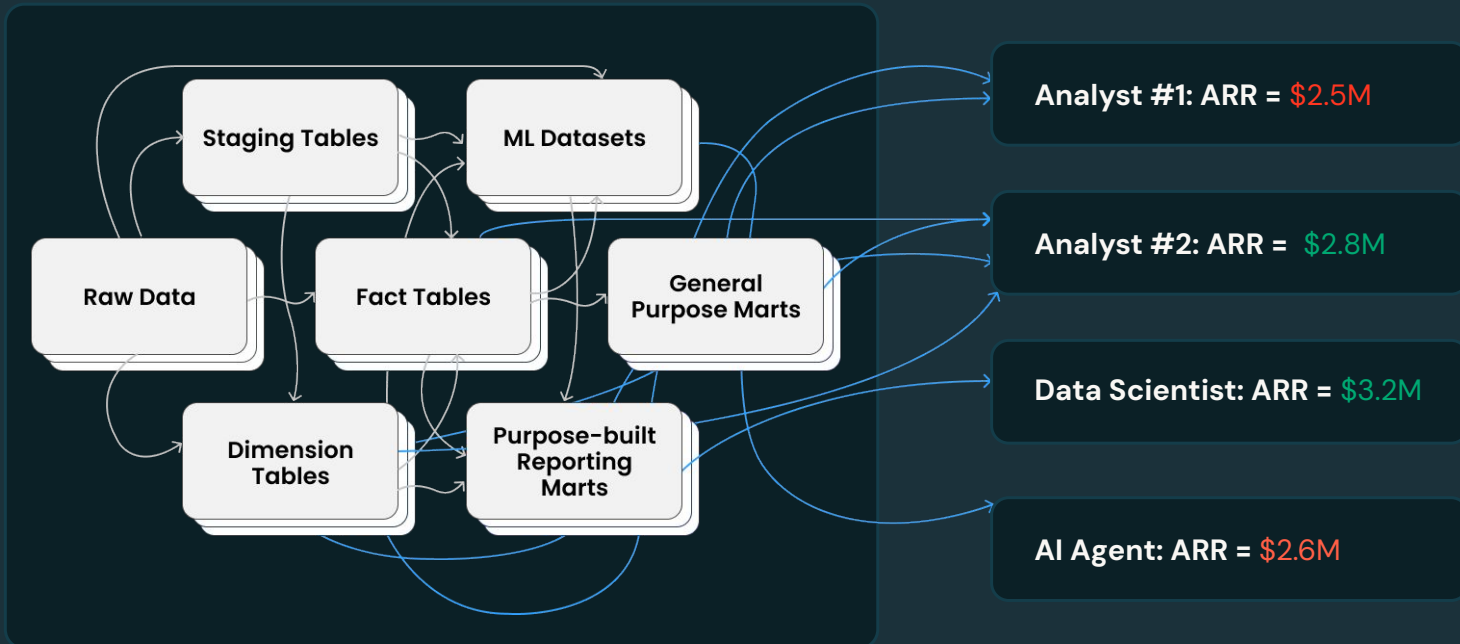


“What is our run-rate ARR?”

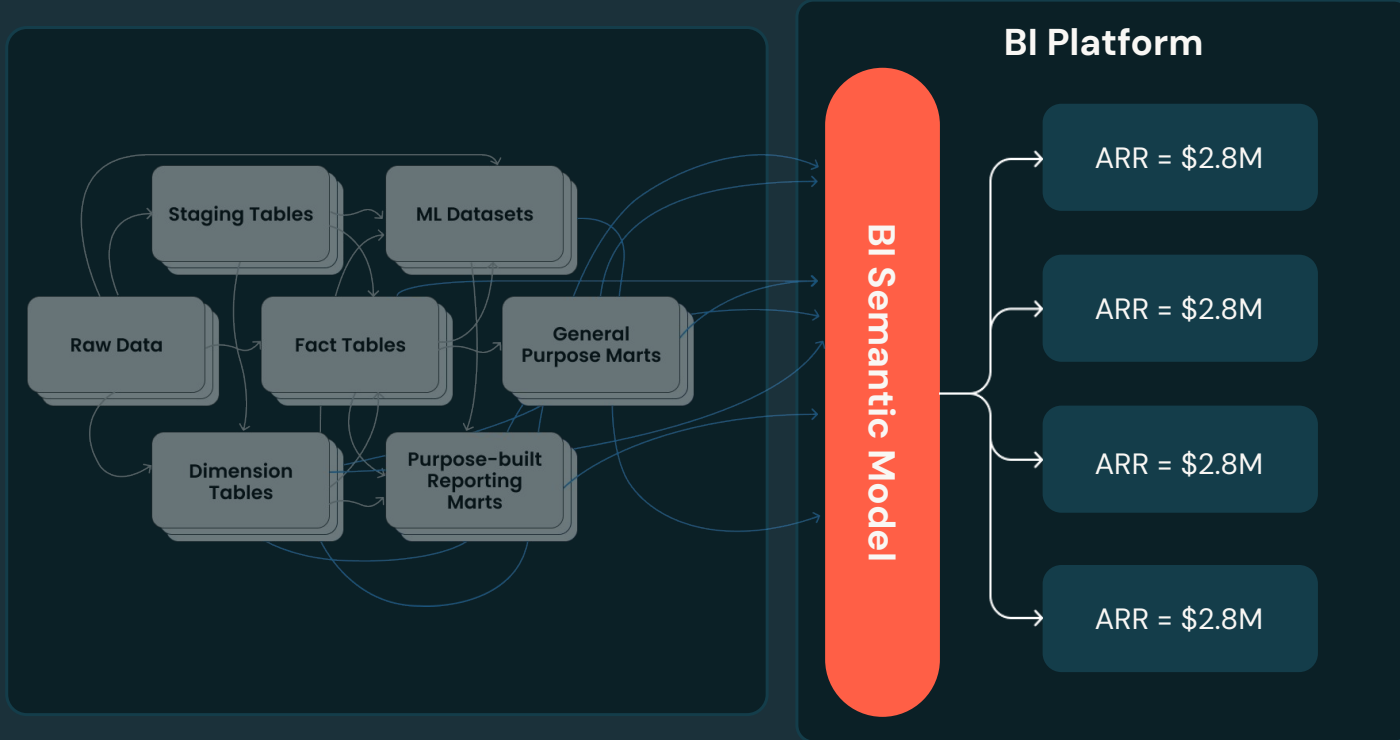
```
With dates AS (Select distinct date from
date_spline),
dailyData as (Select A.date,
sum(opportunities.amount) as bookings
From dates A
Left Join prod.oppo B
ON (B.date between date_sub(A.date, 6) and
A.date)
Group By A.date
Where opportunities.stage = 'Won'
)
Select date,
sum(bookings) OVER (ORDER BY date ASC ROWS 7 -
1 preceding) * 365 / 7 `AS Run-rate`
From dailyData
```



# Getting it right *everywhere* is even harder





# BI Semantic Layers aim to solve these problems



# BI Semantic Layers are tool-specific



Semantic Layer #1	Power BI 	ARR = \$2.5M
Semantic Layer #2	 + a b l e a u	ARR = \$2.4M
Semantic Layer #3	 Looker	ARR = \$2.8M
Custom SQL	  	ARR = \$3.1M

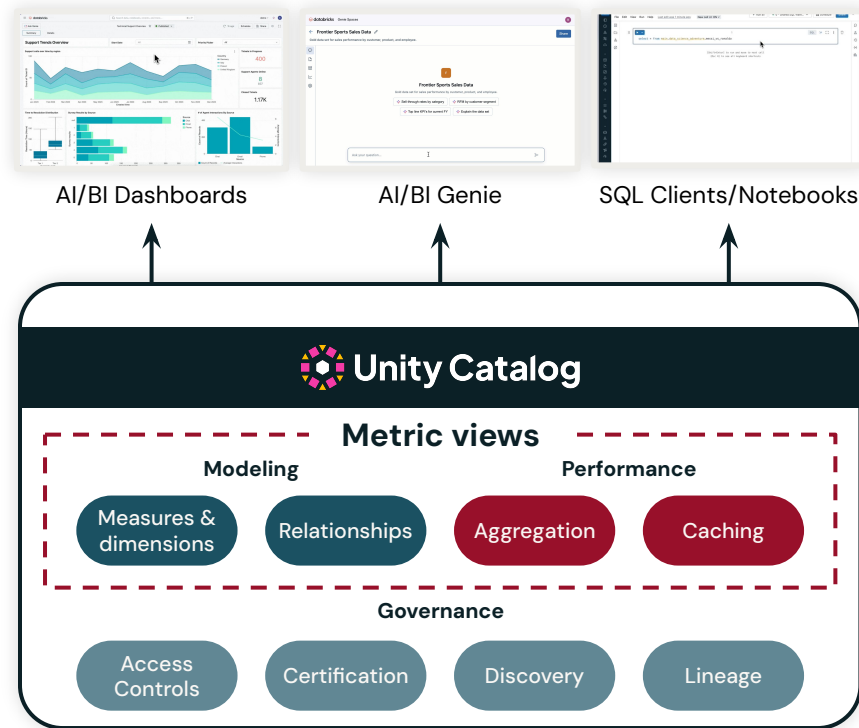


# Introducing Unity Catalog Metric Views

**Define** your organization's business metrics in one place

**Establish trust and reliability** through centralized governance and certification

**Access** from AI/BI Dashboards, AI/BI Genie and SQL with the right semantics



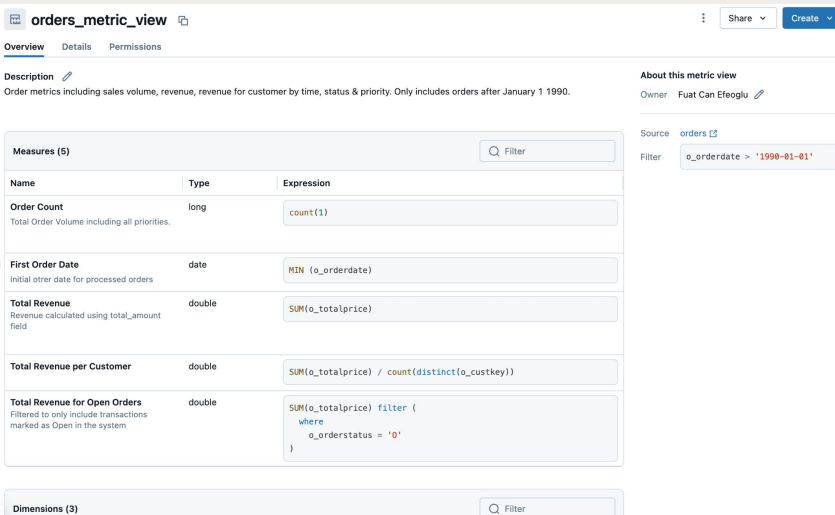
# Reliable and consistent business metrics

**Define** your data model, relationships.  
Build measures & dimensions.

**Centrally govern** with search, usage  
insights, lineage & auditing built-in

**Certify metrics** for increased trust  
and reliability

**Materialize for performance** and  
pre-compute to reduce latency  
for frequent queries



The screenshot displays a web interface for a data catalog view titled "orders\_metric\_view". The interface includes a description, a table of measures, and a list of dimensions. The measures table lists five metrics: Order Count, First Order Date, Total Revenue, Total Revenue per Customer, and Total Revenue for Open Orders, each with its type and a corresponding SQL expression. The dimensions section is partially visible at the bottom.

Name	Type	Expression
Order Count <small>Total Order Volume including all priorities.</small>	long	count(1)
First Order Date <small>Initial order date for processed orders</small>	date	MIN (o_orderdate)
Total Revenue <small>Revenue calculated using total_amount field</small>	double	SUM(o_totalprice)
Total Revenue per Customer	double	SUM(o_totalprice) / count(distinct(o_custkey))
Total Revenue for Open Orders <small>Filtered to only include transactions marked as Open in the system</small>	double	SUM(o_totalprice) filter ( where o_orderstatus = '0' )



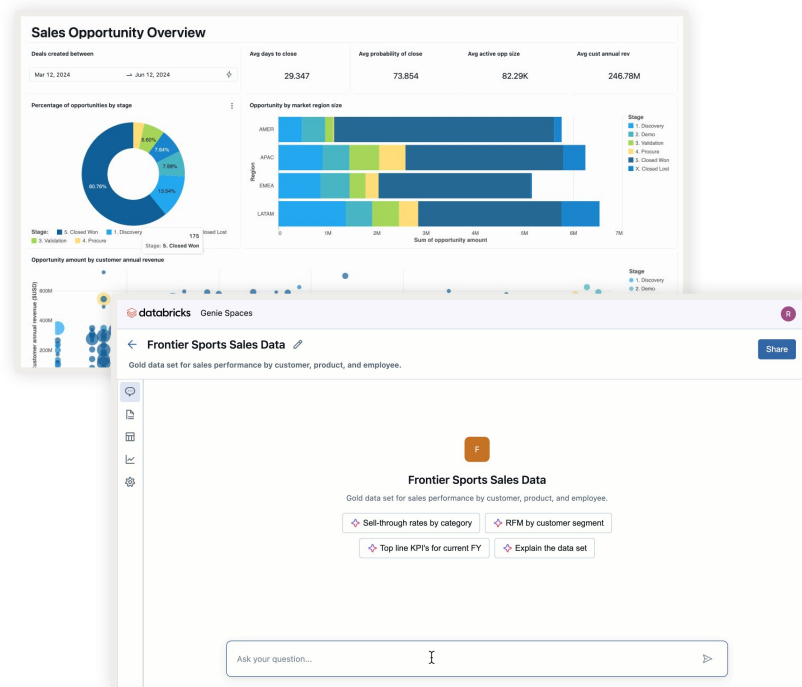
# Accuracy across any Databricks tool, any workload

Integrated with **AI/BI Genie** spaces for **natural language querying**

Drag & drop **dashboarding** using **AI/BI Dashboards**

Query Metric Views using SQL from **notebooks, Gen AI or other applications**

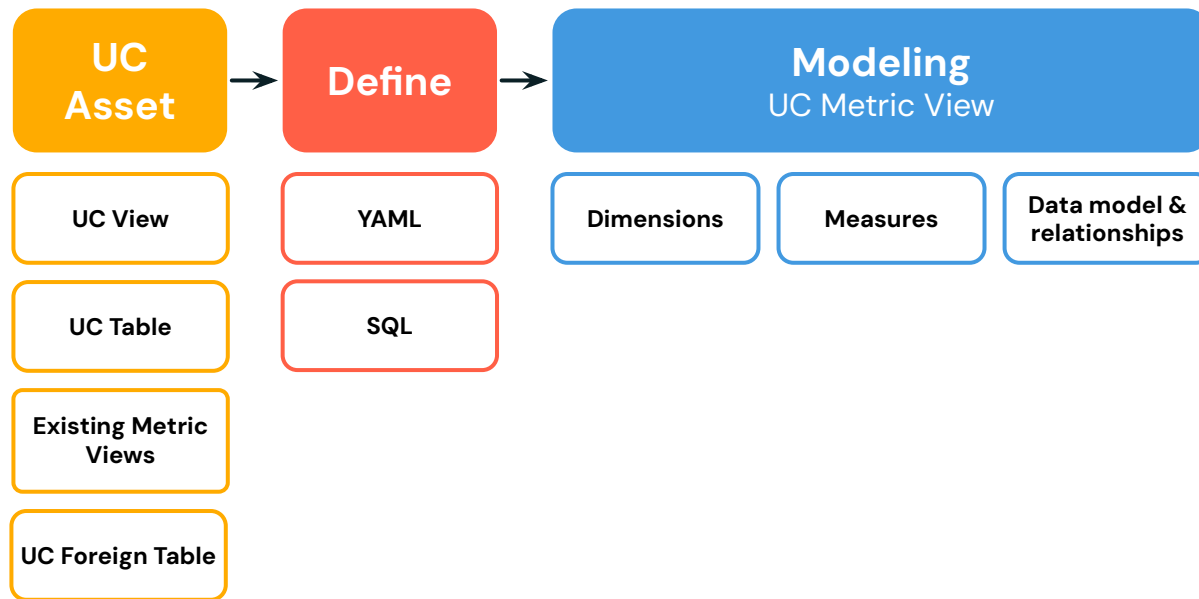
Query using various **clients via SQL**:  
SQL REST API, ODBC, JDBC, Python,  
GO, Node.js



# How it works



# Define



# Simplified, consistent insights

## Business question

"What's the average revenue per customer per segment for fulfilled orders since Jan 1, 2024?"

### Equivalent query

```
SELECT
  c.customer_segment,
  SUM(o.revenue) / COUNT(DISTINCT
  c.customer_id) AS revenue_per_customer
FROM orders o
JOIN customer c ON o.c_id=c.id
WHERE order_date > '2024-01-01' AND
o.status = 'fulfilled' AND o.check IS
NOT NULL
GROUP BY c.customer_segment;
```

- ✗ Repeated logic
- ✗ Easy to make mistakes
- ⚠ Need to understand underlying tables

### UC Metric View Query

```
SELECT
  'Customer Segment',
  MEASURE('Revenue per Customer')
FROM customer_metrics
WHERE 'Order Category' = "Valid
Fulfilled"
GROUP BY ALL
```

- ✓ Simpler queries
- ✓ Defined once, reused everywhere
- ✓ Fully governed & centralized



# Defining Metric View

## YAML to define a Metric View

A metric view is made up of:

- Source: Table or view containing raw data
  - ✓ e.g., sales\_transactions
- Dimensions: Fields to filter or group by
  - ✓ e.g., region, product, channel
- Measures: Aggregations or KPIs
  - ✓ e.g., total\_sales, avg\_discount, order\_count

One metric view = Many questions answered

- Mix & match measures and dimensions
- Supports slicing and filtering without redefining
- No need to create new assets for every slice or filter

Catalog Explorer > can > metric\_view\_testing >

### Update definition for orders\_metric\_view

```
1  version: 1.0
2
3  source: samples.tpch.orders
4  filter: o_orderdate > '1990-01-01'
5
6  dimensions:
7  - name: Order Month
8    expr: date_trunc('MONTH', o_orderdate)
9    type: date
10
11 - name: Order Status
12   expr: >
13     case
14     when o_orderstatus = 'O' then 'Open'
15     when o_orderstatus = 'P' then 'Processing'
16     when o_orderstatus = 'F' then 'Fulfilled'
17     end
18   type: string
19
20 - name: Order Priority
21   expr: split(o_orderpriority, '-') [1]
22   type: string
23
24 measures:
25 - name: Order Count
26   expr: count(1)
27   type: bigint
28
29 - name: First Order Date
30   expr: MIN(o_orderdate)
31   type: date
32
33 - name: Total Revenue
34   expr: SUM(o_totalprice)
35   type: double
36
37 - name: Total Revenue per Customer
38   expr: SUM(o_totalprice) / count(distinct(o_custkey))
39   type: double
40
```



# Define – Dimensions

```
dimensions:  
  - name: Order Date  
    expr: orders.o_orderdate  
  
  - name: Order Status  
    expr: >  
      case  
        when orders.o_orderstatus = 'O' then 'Open'  
        when orders.o_orderstatus = 'P' then 'Processing'  
        when orders.o_orderstatus = 'F' then 'Fulfilled'  
      end  
  
  - name: Order Priority  
    expr: >  
      case  
        when split(orders.o_orderpriority, '-') [0] = '1' then 'Urgent'  
        when split(orders.o_orderpriority, '-') [0] = '2' then 'High'  
        when split(orders.o_orderpriority, '-') [0] = '3' then 'Medium'  
        when split(orders.o_orderpriority, '-') [0] = '4' then 'Low'  
        else 'Unknown'  
      end  
end
```

## Supports:

- Direct column references
- Any valid scalar SQL expression
- Scalar subqueries for advanced usage



# Define – Measures

```
measures:  
  - name: Total Revenue  
    expr: SUM(l_extendedprice * (1 - l_discount))  
  
  - name: Order Count  
    expr: COUNT(l_orderkey)  
  
  - name: Average Discount %  
    expr: AVG(l_discount) * 100  
  
  - name: Average Order Value  
    expr: MEASURE(`Total Revenue`) / MEASURE(`Order Count`)  
  
  - name: Fulfilled Revenue  
    expr: MEASURE(`Total Revenue`)  
      FILTER (WHERE orders.o_orderstatus = 'F')  
  
  - name: Trailing 7-Day Revenue  
    expr: MEASURE(`Total Revenue`)  
    window:  
      - order: Order Date  
        range: trailing 7 day  
        semiadditive: last  
  
  - name: Cumulative Revenue  
    expr: MEASURE(`Total Revenue`)  
    window:  
      - order: Order Date  
        range: cumulative  
        semiadditive: last
```

## Supports:

- Any valid aggregate SQL expression
- Common windowing scenarios:
  - Trailing windows
  - Cumulative windows
  - Current / sub-total
- Composability: Use existing measures and/or dimensions to create new measures
- Scalar subqueries for advanced usage



# Querying Metric View

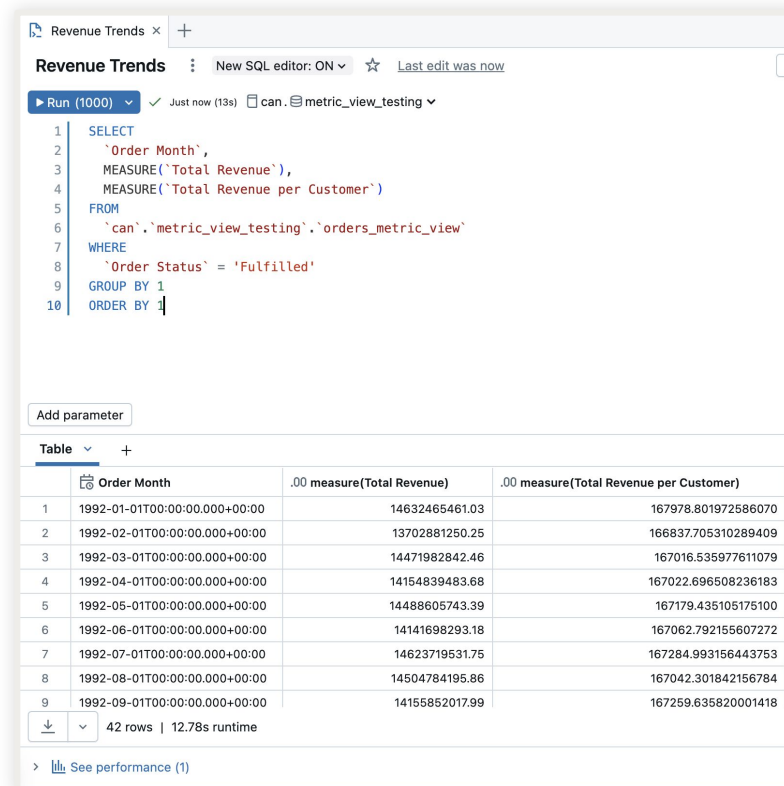
## Query using MEASURE() Function

- Query business metrics directly using SQL
- No need to understand the underlying data model or schema layout

## Metric Views automatically generate:

- Correct aggregations & expressions
- Required joins across tables
- Materialized views for performance optimization (if available and applicable)

**Just use SELECT MEASURE(...)—  
let Metric Views handle  
the complexity**



The screenshot shows a SQL query editor window titled "Revenue Trends". The query is as follows:

```
1 SELECT
2   `Order Month`,
3   MEASURE(`Total Revenue`),
4   MEASURE(`Total Revenue per Customer`)
5 FROM
6   `can`.`metric_view_testing`.`orders_metric_view`
7 WHERE
8   `Order Status` = 'Fulfilled'
9 GROUP BY 1
10 ORDER BY 1
```

Below the query editor, there is a table with the following data:

Order Month	.00 measure(Total Revenue)	.00 measure(Total Revenue per Customer)
1992-01-01T00:00:00.000+00:00	14632465461.03	167978.801972586070
1992-02-01T00:00:00.000+00:00	13702881250.25	166837.705310289409
1992-03-01T00:00:00.000+00:00	14471982842.46	167016.535977611079
1992-04-01T00:00:00.000+00:00	14154839483.68	167022.696508236183
1992-05-01T00:00:00.000+00:00	14488605743.39	167179.435105175100
1992-06-01T00:00:00.000+00:00	14141698293.18	167062.792155607272
1992-07-01T00:00:00.000+00:00	14623719531.75	167284.993156443753
1992-08-01T00:00:00.000+00:00	14504784195.86	167042.301842156784
1992-09-01T00:00:00.000+00:00	14155852017.99	167259.635820001418

The table also shows a summary row: 42 rows | 12.78s runtime.



# Views vs Metric Views | What's the difference?

## Examples

### Views or tables

```
CREATE VIEW view_1 AS
SELECT
  SUM(sale_amount) / COUNT(DISTINCT customer_id)
FROM orders
GROUP BY order_date
```

```
CREATE VIEW view_2 AS
SELECT
  SUM(sale_amount) / COUNT(DISTINCT customer_id)
FROM orders
GROUP BY region
```

```
CREATE VIEW view_3 AS
SELECT
  SUM(sale_amount) / COUNT(DISTINCT customer_id)
FROM orders
GROUP BY product_line
```

```
CREATE VIEW view_N AS
...
```

### Metric view

Definition (*Single metric view instead of N views*)

```
order_metrics
```

```
Dimensions: order_date, region, product_line
```

```
Measures : SUM(sale_amount) / COUNT(DISTINCT customer_id)
```

Users pick GROUP BY at query time

```
SELECT
  product_line,
  MEASURE(rev)
FROM order_metrics
WHERE order_date > '01-01-2024'
GROUP BY 1
```



# Unity Catalog Metrics Partners

Upcoming Integrations  
*(pending partner development)*

