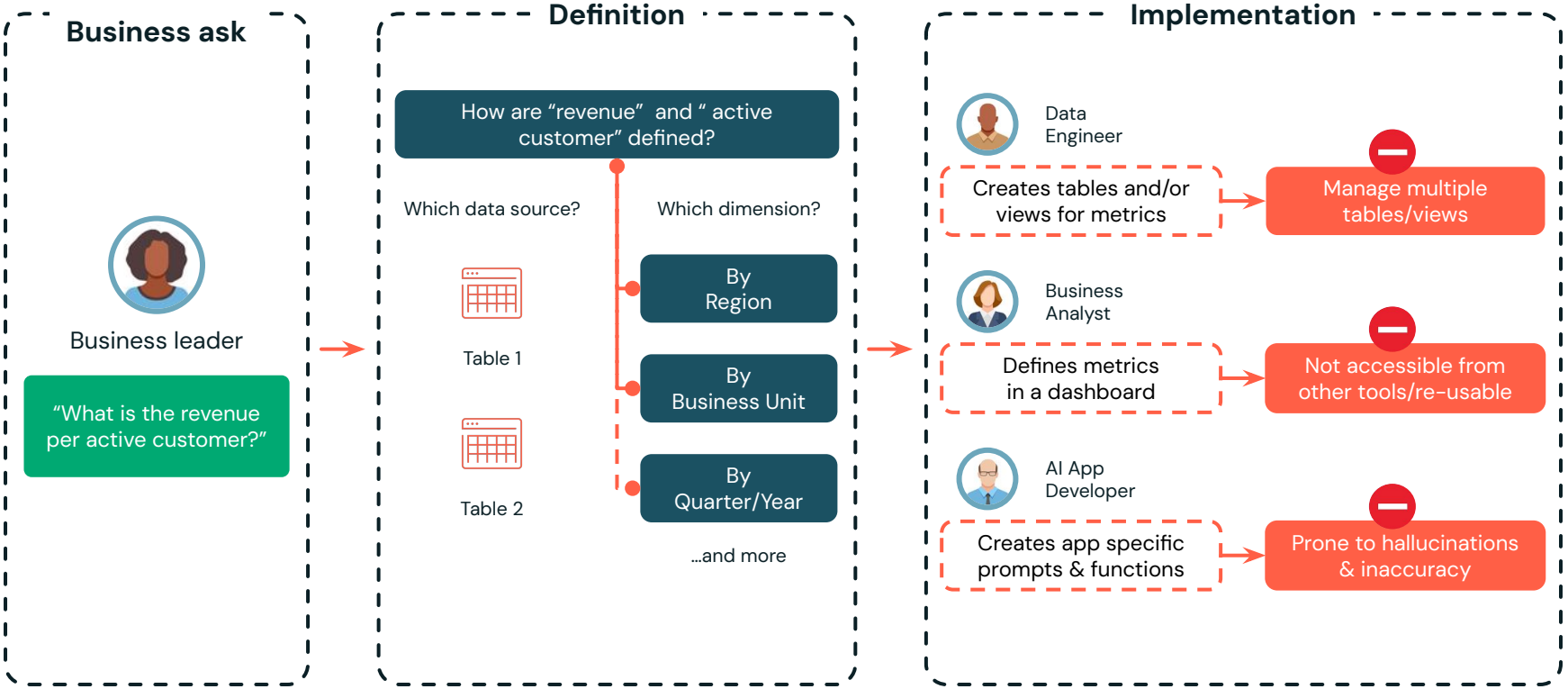
databricks

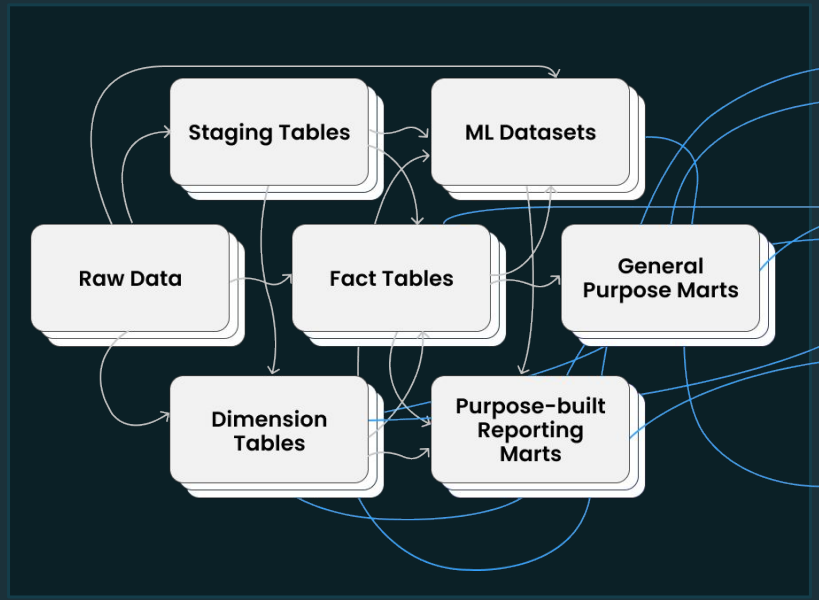
Unity Catalog Metric Views

Goal: Turn data & context into insights

Hard in practice



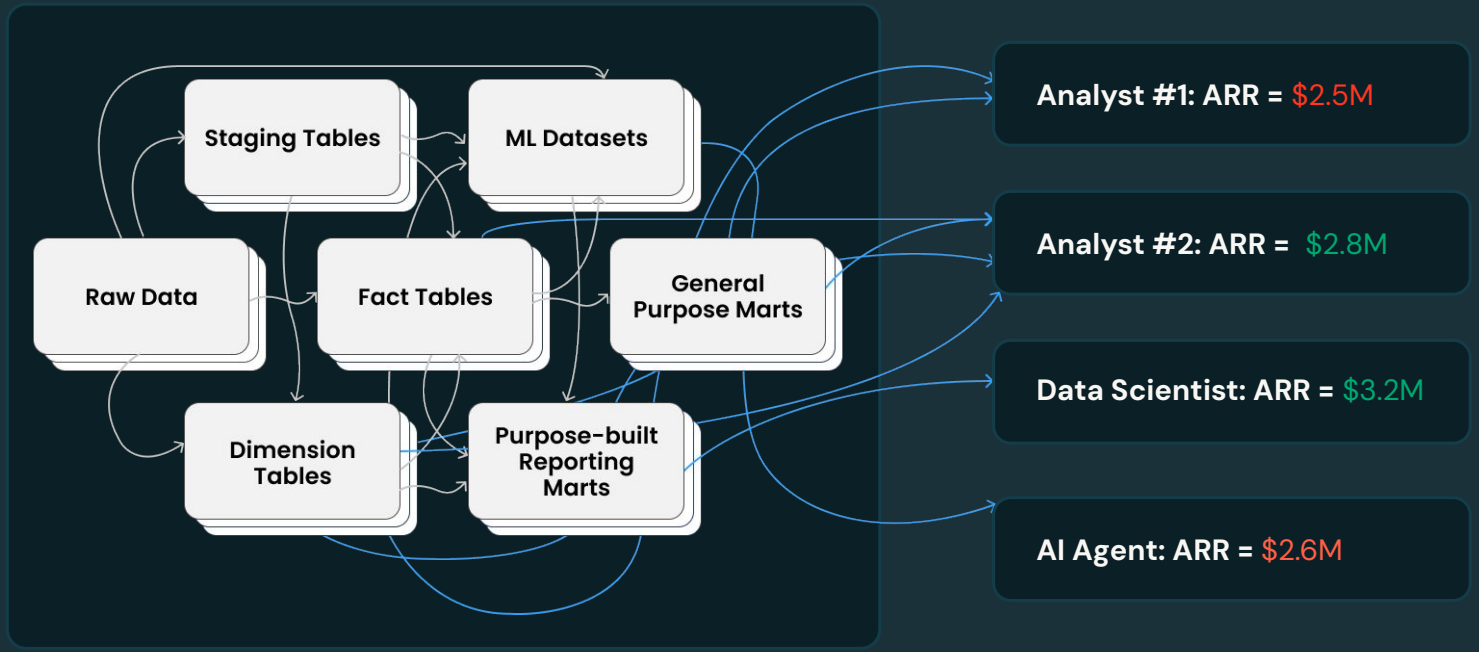
Getting the right number is hard



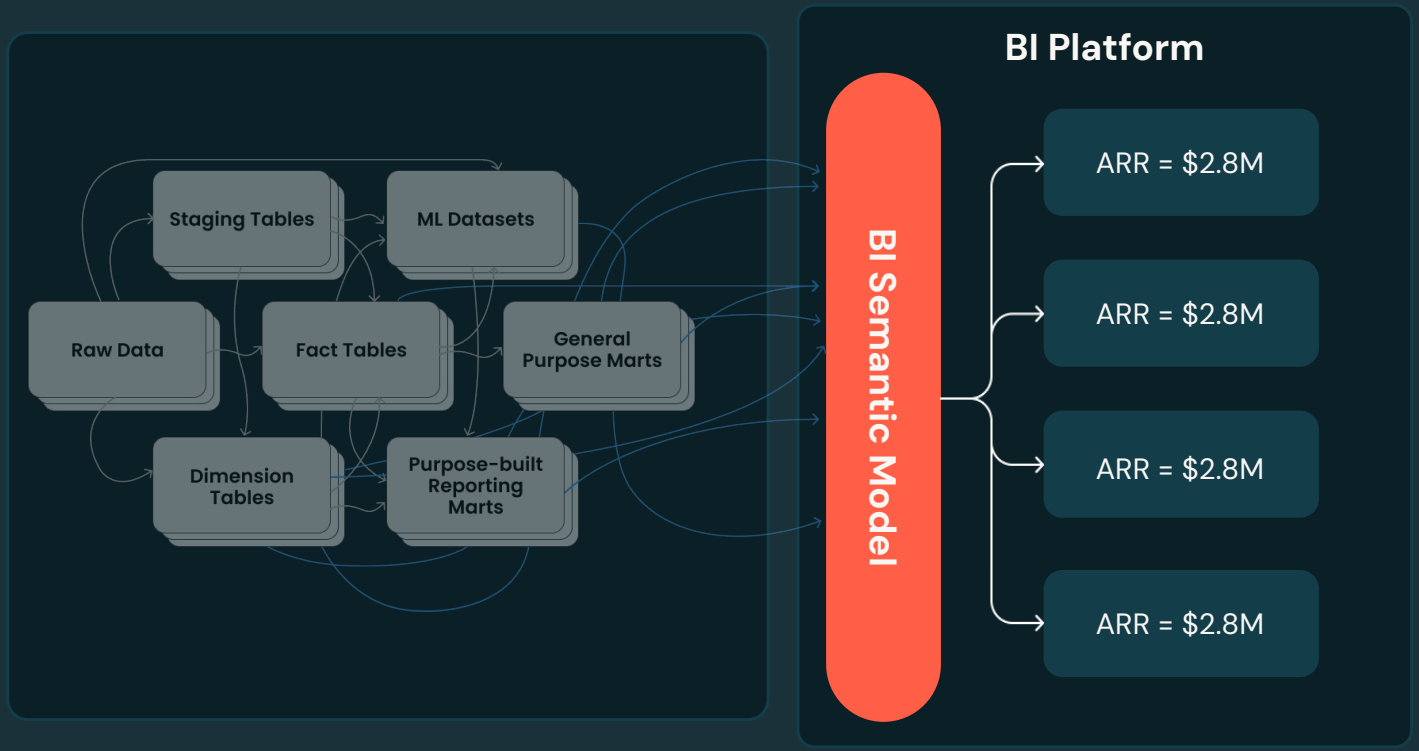
“What is our run-rate ARR?”

```
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

tableau

ARR = $2.4M

Semantic Layer #3

Looker

ARR = $2.8M

Custom SQL

jupyter HEX plotly

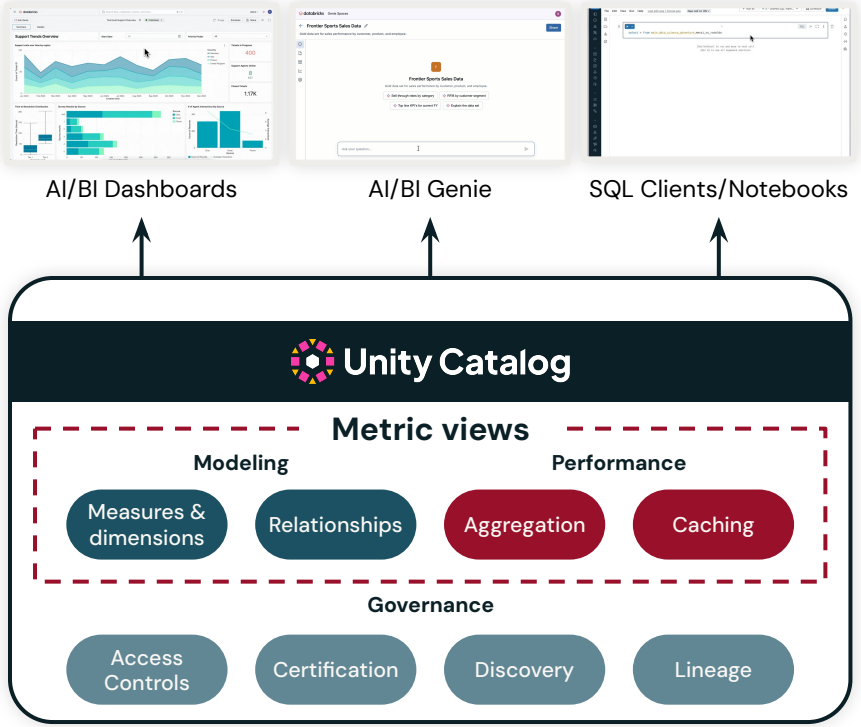
ARR = $3.1M

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



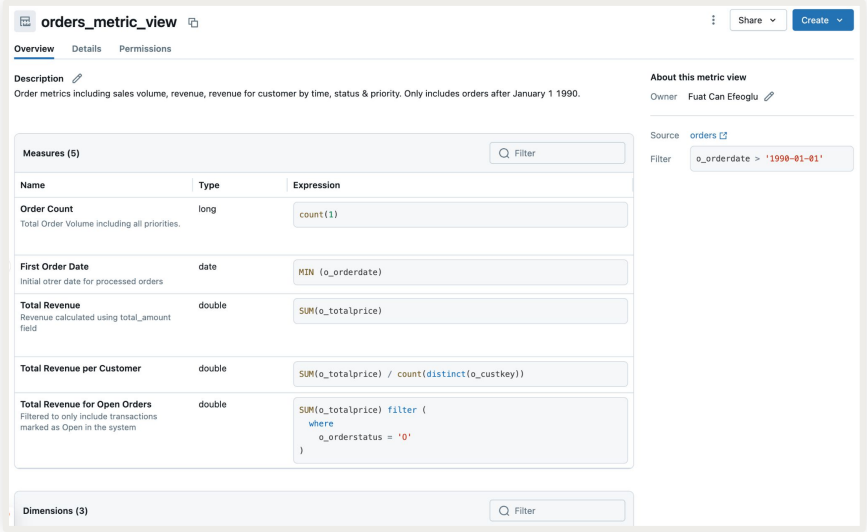
Reliable and consistent business metrics

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



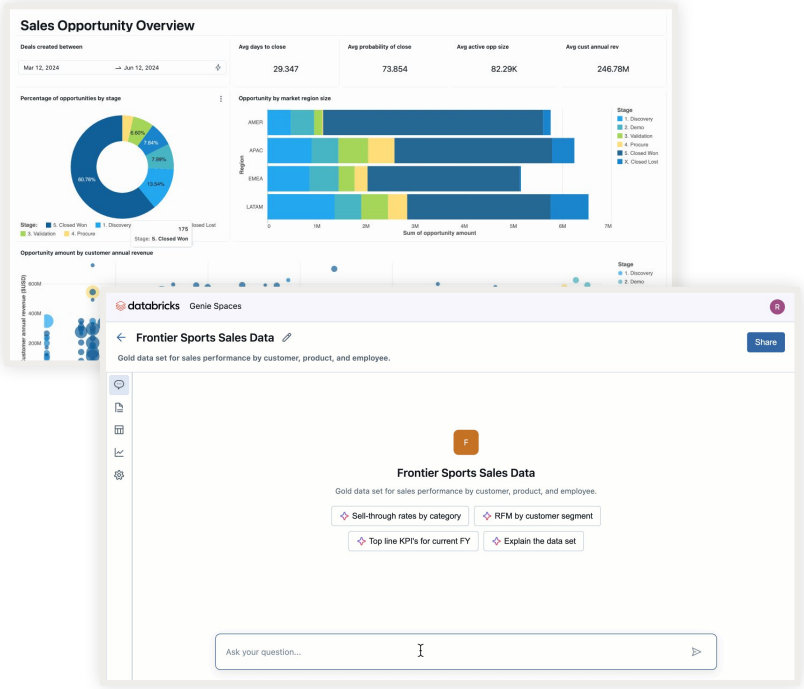
Accuracy across any Databricks tool, any workload

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

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

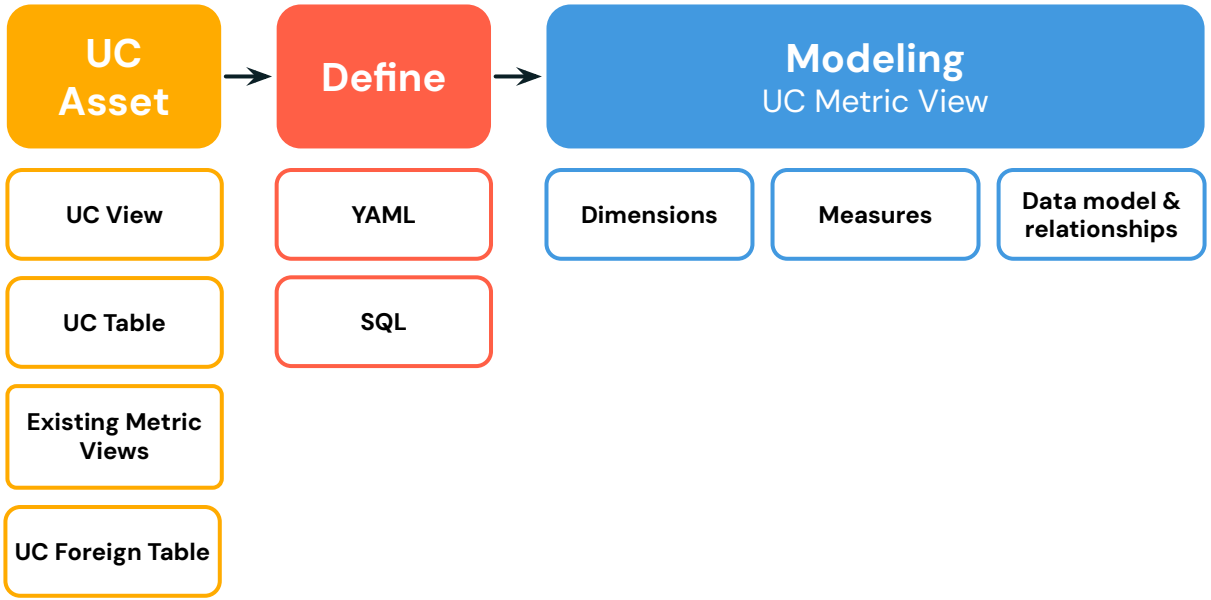
Query Metric Views using SQL from **notebooks, Gen AI or other applications**

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



How it works

Define


UC Asset
Define
Modeling
UC Metric View
UC View
UC Table
Existing Metric Views
UC Foreign Table
YAML
SQL
Dimensions
Measures
Data model & relationships

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

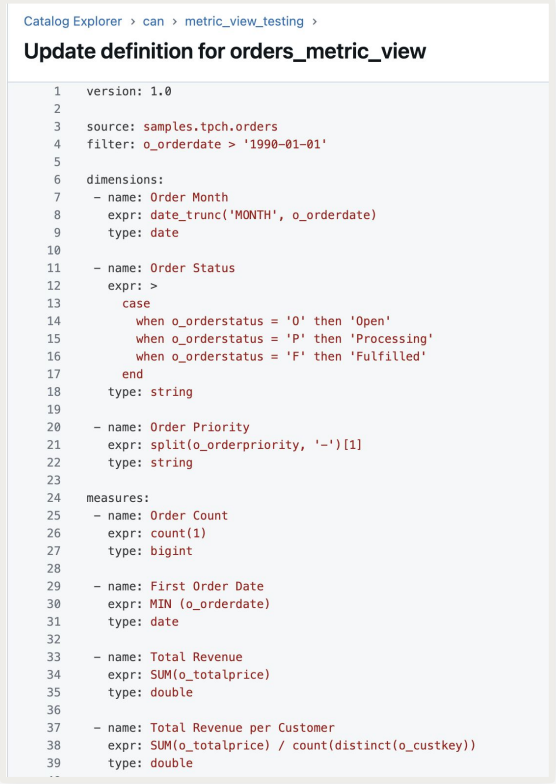
YAML to define a Metric View

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



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
```

Supports:

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

Define – Measures

```yaml
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

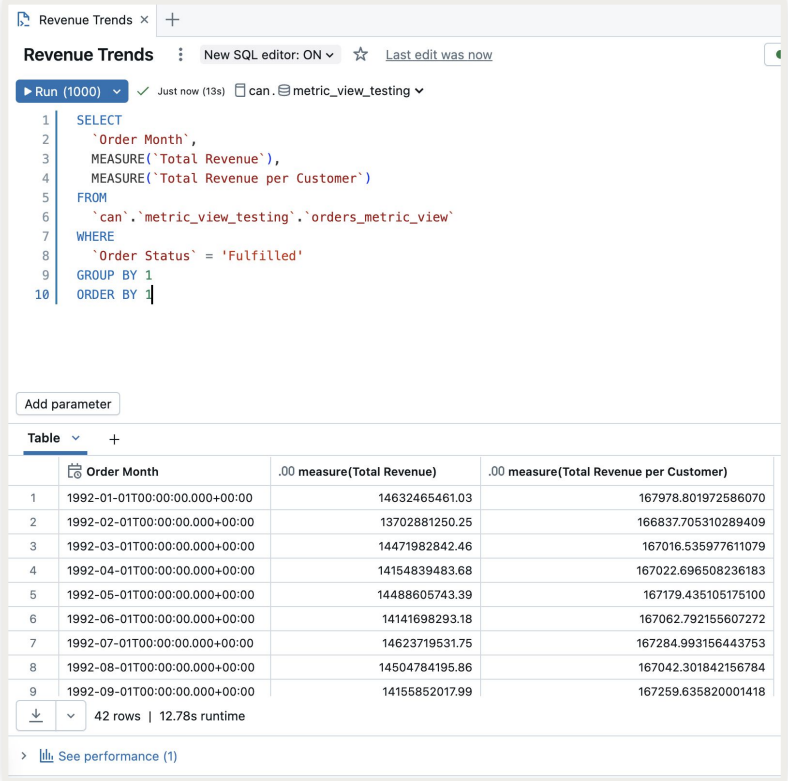
Query using MEASURE() Function

- 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



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)