SQL Analytics & Dashboards

Day 9: Building Business Intelligence on Databricks

Databricks 14-Days AI Challenge

January 20, 2026

Agenda

- **SQL Warehouses**
  - Types & Architecture
  - Sizing Guide
- **Complex Analytical Queries**
  - Window Functions
  - Common Table Expressions
- **Dashboard Creation**
  - Visualization Types
  - Design Principles
- **Visualizations & Filters**
  - Filter Implementation
  - Scheduled Refresh
- **Practical Tasks**
  - 7-Day Moving Average
  - Conversion Funnel
  - Customer Tier Segmentation
- **Best Practices**

Introduction to SQL Analytics

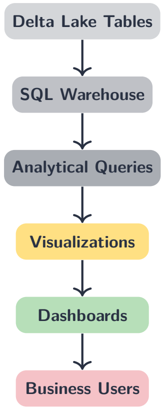
What is SQL Analytics?

A powerful environment for:

- Running **analytical queries**
- Building **interactive dashboards**
- Gaining **insights** from your data lakehouse

Why It Matters:

- Enables analysts without Spark knowledge
- Familiar SQL interface
- Direct lakehouse integration
- Real-time dashboard capabilities



SQL Warehouses Overview

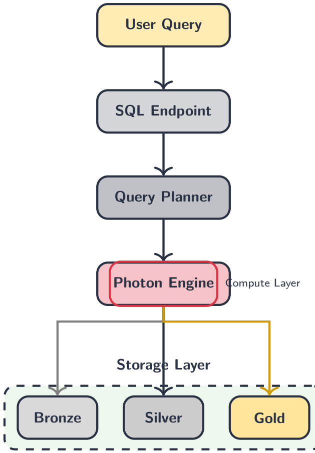
What is a SQL Warehouse? The **compute resource** that executes SQL queries against Delta Lake tables.

Type	Description	Best Use Case	Cost Model
Serverless	Fully managed, instant startup	Ad-hoc queries	Pay per query
Pro	Provisioned with advanced features	Production dashboards	Pay per hour
Classic	Basic provisioned clusters	Development, testing	Pay per hour

Key Benefits:

- **Separation of compute and storage** – Scale independently
- **Photon Engine** – Vectorized query acceleration
- **Auto-scaling & Auto-stop** – Cost optimization

SQL Warehouse Architecture



Warehouse Sizing Guide

Configuration Parameters:

Parameter	Recommendation
Name	Descriptive (e.g., `analytics_prod`)
Cluster Size	Start small, scale up
Min/Max Clusters	Min=1, Max=based on users
Auto Stop	10-30 min for dev

Size Reference:

Size	DBU/Hr	Memory
2X-Small	2	16 GB
X-Small	4	32 GB
Small	8	64 GB
Medium	16	128 GB
Large	32	256 GB

Tip: Start with 2X-Small and scale based on query performance metrics

Understanding Window Functions

What are Window Functions?

Perform calculations across a **set of related rows** while retaining individual rows.

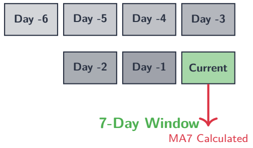
General Syntax:

```
function_name(expression) OVER (
    [PARTITION BY partition_expr]
    [ORDER BY order_expr]
    [frame_clause]
)
```

Frame Clause Example:

```
ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
```

Include current row + 6 previous rows (7 total)



Common Window Functions

Function	Purpose	Example Use Case
`ROW_NUMBER()`	Assigns unique sequential integers	Ranking products
`RANK()`	Assigns rank with gaps for ties	Leaderboards
`DENSE_RANK()`	Assigns rank without gaps	Competition rankings
`LAG()`	Accesses previous row value	Day-over-day comparison
`LEAD()`	Accesses next row value	Forecasting trends
`SUM() OVER`	Running total	Cumulative revenue
`AVG() OVER`	Moving average	Smoothing trends
`FIRST_VALUE()`	First value in window	Baseline comparisons
`LAST_VALUE()`	Last value in window	Latest status

Common Table Expressions (CTEs)

What are CTEs? Temporary named result sets that exist only for the duration of the query.

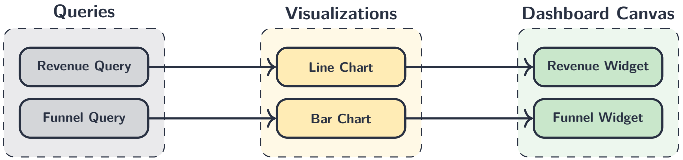
```
WITH cte_name AS (
    -- First query
    SELECT ...
),
another_cte AS (
    -- Can reference previous CTEs
    SELECT ... FROM cte_name
)
SELECT * FROM another_cte;
```

Benefits:

- • Improve query **readability**
- • Allow **recursive** queries
- • Can be referenced **multiple times**
- • Easier to **debug** complex logic

Break complex queries into logical steps

Dashboard Components



Dashboard = Multiple visualizations (widgets) arranged on a canvas Each visualization is powered by a SQL query

Visualization Types

Visualization	Best For	Example
Line Chart	Trends over time	Revenue over months
Bar Chart	Comparing categories	Sales by region
Pie/Donut Chart	Part-to-whole relationships	Market share
Counter	Single KPI display	Total revenue
Table	Detailed data display	Top products list
Funnel	Stage-based conversion	Purchase funnel
Scatter Plot	Correlation analysis	Price vs. quantity
Heatmap	Matrix relationships	Activity by day/hour

Design Principles:

- **Hierarchy:** Important metrics at top-left
- **Grouping:** Related vizs together
- **Context:** Provide comparison points
- **Clarity:** One question per widget

Filters & Interactivity

Filter Types:

Type	Example
Dropdown	Category selection
Date Range	Report period
Text	Search by name
Query-based	Dynamic list

Implementation:

```
SELECT * FROM gold.products
WHERE category_code =
      '{{ category_filter }}'
AND event_date BETWEEN
    '{{ start_date }}'
    AND '{{ end_date }}'
```

Scheduled Refresh:

Interval	Use Case
1 minute	Real-time monitoring
1 hour	Operational dashboards
Daily	Executive reports
Weekly	Summary dashboards

More frequent = Higher cost

Task 1: Revenue with 7-Day Moving Average

Goal: Calculate daily revenue and smooth with 7-day moving average

```
-- Revenue with 7-day moving average
WITH daily AS (
    SELECT
        event_date,
        SUM(revenue) as rev
    FROM gold.products
    GROUP BY event_date
)
SELECT
    event_date,
    rev,
    AVG(rev) OVER (
        ORDER BY event_date
        ROWS BETWEEN 6 PRECEDING
             AND CURRENT ROW
    ) as ma7
FROM daily;
```

Mathematical Formula:

$$MA_7(t) = \frac{1}{7}\sum_{i=0}^{6} R_{t-i}$$

Where:

- $MA_7(t)$ = 7-day MA at time t
- R_{t-i} = Revenue on day $t - i$

Why 7-Day?

- ▷ Smooths daily volatility
- ▷ Captures weekly patterns
- ▷ Identifies trends

Task 2: Conversion Funnel Analysis

Goal: Measure how effectively categories convert views into purchases

```
-- Conversion funnel
SELECT
    category_code,
    SUM(views) as views,
    SUM(purchases) as purchases,
    ROUND(SUM(purchases) * 100.0
          / SUM(views), 2)
        as conversion_rate
FROM gold.products
GROUP BY category_code;
```

Formula:

$$\text{Rate} = \frac{\text{Purchases}}{\text{Views}} \times 100$$

Interpreting Results:

Rate	Status
$< 1\%$	Poor
1-3%	Average
3-5%	Good
$> 5\%$	Excellent

Use `100.0` for float division!

Task 3: Customer Tier Segmentation

Goal: Segment customers into tiers based on purchase frequency

```sql
SELECT
    CASE
        WHEN cnt >= 10 THEN 'VIP'
        WHEN cnt >= 5 THEN 'Loyal'
        ELSE 'Regular'
    END as tier,
    COUNT(*) as customers,
    AVG(total_spent) as avg_ltv
FROM (
    SELECT user_id, COUNT(*) cnt, SUM(price) total_spent
    FROM silver.events WHERE event_type = 'purchase' GROUP BY user_id
)
GROUP BY tier;
```

Tier Definitions:

VIP	10+ purchases
Loyal	5-9 purchases
Regular	1-4 purchases

LTV Formula:

$$\text{Avg LTV} = \frac{\sum \text{TotalSpent}}{n}$$

Best Practices

Query Optimization:

- **Use Appropriate Data Layers:**
  - ▷ Bronze: Avoid for analytics
  - ▷ Silver: Detailed analysis
  - ▷ **Gold: Best for dashboards**
- **Leverage Partitioning**
- **Use Delta Lake Features:**
  - ▷ Z-ordering
  - ▷ OPTIMIZE
  - ▷ VACUUM

Cost Management:

- • **Right-size** warehouses
- • Use **auto-stop**
- • **Monitor** query history
- • Schedule during **off-peak**

Dashboard Performance:

- Pre-aggregate data
- Limit result sets
- Enable query caching

Quick Reference: Useful SQL Patterns

Running Total:

```
SUM(amount) OVER (
    ORDER BY date
    ROWS UNBOUNDED PRECEDING)
```

Rank within Group:

```
ROW_NUMBER() OVER (
    PARTITION BY category
    ORDER BY sales DESC)
```

Year-over-Year:

```
LAG(revenue, 12) OVER (
    ORDER BY month)
    as prev_year_revenue
```

Percent of Total:

```
revenue * 100.0 /
    SUM(revenue) OVER ()
    as pct_of_total
```

Filter Parameters:

```
WHERE col = '{{ param }}'
WHERE col IN ({{ param }})
WHERE date BETWEEN ...
```

Summary

Component	Purpose
SQL Warehouse	Compute engine for query execution
Query Editor	Write and test SQL queries
Visualizations	Transform data into charts
Dashboards	Combine visualizations into reports
Filters	Enable interactive exploration
Scheduling	Automate refresh cycles

Key Patterns Covered:

1. **Moving Average** – Trend analysis with window functions
2. **Conversion Funnel** – Performance metrics with aggregation
3. **Customer Tiers** – Segmentation with CASE statements

Thank You!

Day 9 Complete

SQL Analytics & Dashboards in Databricks

Yash Kavaiya | Gen AI Guru