Databricks SQL & Warehousing

1. Databricks SQL Overview

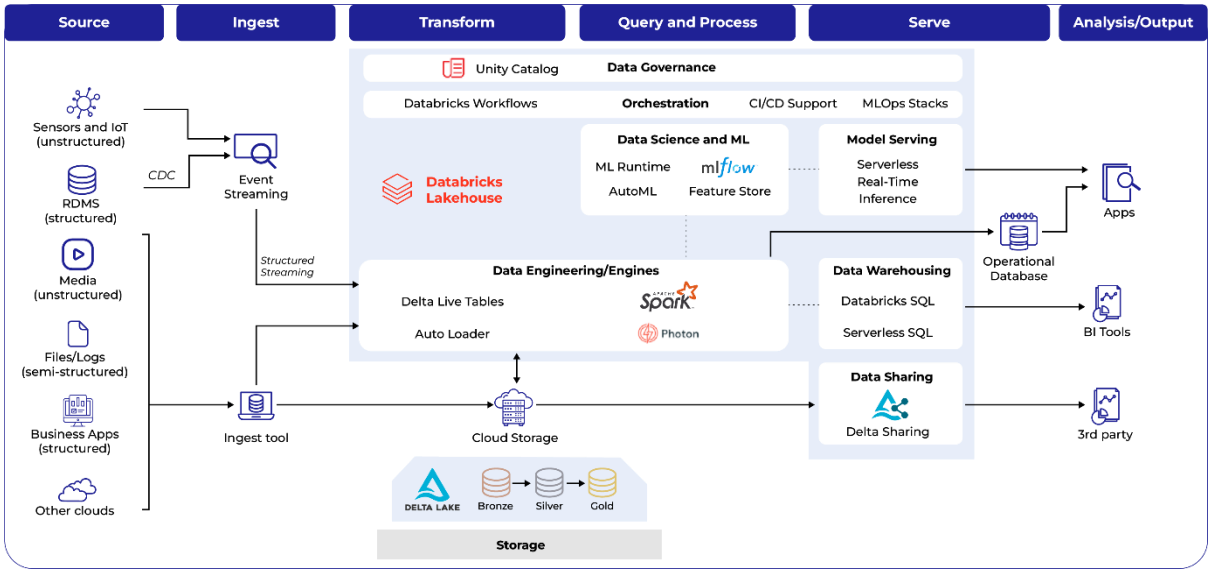
What is Databricks SQL?

A **serverless data warehouse** solution on Databricks Lakehouse Platform that provides:

- ANSI SQL compliant querying
- Visual dashboarding
- Performance-optimized compute
- Direct Delta Lake integration

Key Features

✓ **Serverless SQL Warehouses** - Instant compute without management
✓ **BI Tool Integration** - Native connectors for Tableau/Power BI
✓ **Delta Engine** - High-performance query execution
✓ **Unity Catalog Integration** - Centralized governance



Databricks SQL Architecture (Warehouses → Delta Lake → BI Tools)

---

2. SQL Warehouses

Warehouse Types

Type	Best For	Auto-Termination
Serverless	Ad-hoc analysis	Yes (default 10 min)
Pro	Production workloads	Configurable
Classic	Legacy compatibility	Yes

Creating a Warehouse

sql

```
-- UI Method:
1. Navigate to SQL → Warehouses → Create
2. Configure:
   - Name: "prod-warehouse"
   - Cluster Size: "Medium (4-8 cores)"
   - Auto-stop: 30 minutes
   - Type: "Serverless"

-- API Method:
POST /api/2.0/sql/warehouses
{
  "name": "analytics-wh",
  "cluster_size": "Small",
  "auto_stop_mins": 15,
  "enable_serverless_compute": true
}
```

Size Recommendations

Size	Cores	Memory	Use Case
X-Small	2	8GB	Light queries
Small	4	16GB	Moderate workloads
Medium	8	32GB	Dashboards
Large	16	64GB	ETL pipelines

---

3. SQL Editor

Editor Features

- **Autocomplete**: Table/column suggestions
- **Syntax Highlighting**: SQL, Python, Scala
- **Execution Plans**: Visual query breakdown
- **Parameterization**: {{date}} variables

Query Examples

sql

```
-- Basic query
SELECT user_id, SUM(amount) as total_spend
FROM transactions
WHERE transaction_date > CURRENT_DATE() - INTERVAL 30 DAYS
GROUP BY user_id;

-- CTE and window functions
WITH ranked_orders AS (
  SELECT
    *,
```

```
    RANK() OVER (PARTITION BY customer_id ORDER BY order_date DESC) as rank
  FROM orders
)
SELECT * FROM ranked_orders WHERE rank = 1;

-- Delta Lake time travel
SELECT * FROM transactions TIMESTAMP AS OF '2023-10-01';
```

Query History

sql

```
-- View past queries
SELECT * FROM system.query.history
WHERE query_text LIKE '%transactions%'
ORDER BY start_time DESC
LIMIT 10;
```

4. Advanced Query Techniques

COPY INTO (Bulk Load)

sql

```
-- Load CSV to Delta
COPY INTO sales.transactions
FROM 's3://data-lake/raw/transactions/'
FILEFORMAT = CSV
FORMAT_OPTIONS ('header' = 'true', 'inferSchema' = 'true')
COPY_OPTIONS ('mergeSchema' = 'true');

-- Incremental pattern
COPY INTO sales.transactions
FROM (
  SELECT *, current_timestamp() as load_time
  FROM 's3://data-lake/raw/transactions/'
  WHERE file_modification_time > (
    SELECT MAX(file_modification_time)
    FROM sales.transactions_metadata
  )
)
FILEFORMAT = CSV;
```

Materialized Views

sql

```
CREATE MATERIALIZED VIEW sales.daily_summary
REFRESH EVERY 1 HOUR
AS
SELECT
  transaction_date,
  COUNT(*) as transaction_count,
  SUM(amount) as total_volume
FROM sales.transactions
GROUP BY transaction_date;
```

Stored Procedures

sql

```
CREATE PROCEDURE sales.update_metrics(metric_date DATE)
LANGUAGE SQL
AS
$$
```

```
  DELETE FROM sales.daily_metrics WHERE date = metric_date;

  INSERT INTO sales.daily_metrics
  SELECT
    transaction_date as date,
    COUNT(*) as count
  FROM sales.transactions
  WHERE transaction_date = metric_date
  GROUP BY transaction_date;
$$;

CALL sales.update_metrics(CURRENT_DATE());
```

5. Performance Optimization

Warehouse Configurations

sql

```
-- Set session parameters
SET spark.databricks.sql.performance.tips = true;
SET spark.sql.adaptive.enabled = true;

-- Cache frequently used tables
CACHE SELECT * FROM sales.transactions WHERE year = 2023;
```

Query Acceleration

```
-- Enable Delta caching
ALTER TABLE sales.transactions
SET TBLPROPERTIES (delta.enableChangeDataFeed = true);

-- Z-ordering
OPTIMIZE sales.transactions
ZORDER BY (transaction_date, customer_id);
```

Partition Pruning

```
-- Optimal schema
CREATE TABLE sales.transactions (
  id STRING,
  amount DECIMAL(18,2),
  transaction_date DATE
) USING DELTA
PARTITIONED BY (transaction_date);

-- Partition-aware query
SELECT * FROM sales.transactions
WHERE transaction_date BETWEEN '2023-01-01' AND '2023-01-31';
```

6. Dashboards & Alerts

Creating Dashboards

1. **UI Method**:
    - SQL → Dashboards → Create
    - Add widgets from query results

2. **Programmatic**:

python

```
from databricks_api import DatabricksAPI

db = DatabricksAPI(host=host, token=token)
db.dashboards.create(
  name="Sales Performance",
  widgets=[
    {
      "visualization_id": query1_viz_id,
      "width": 3,
      "options": {...}
    }
  ]
)
```

Alerting

sql

```
-- Create alert
CREATE ALERT sales.high_value_transactions
IF EXISTS (
  SELECT * FROM transactions
  WHERE amount > 10000 AND transaction_date = CURRENT_DATE()
)
EVERY 1 HOUR
EMAIL 'fraud-team@company.com';

-- Check alert history
SELECT * FROM system.alerts.history;
```

---

7. Security & Governance

Column-Level Security

sql

```
-- Mask sensitive data
CREATE MASK sales.ssn_mask AS (
  CASE WHEN is_member('hr-team') THEN ssn
       ELSE '***-**-****' END
);

ALTER TABLE customers
ALTER COLUMN ssn SET MASK sales.ssn_mask;

-- Row filtering
CREATE FILTER sales.region_filter AS (
  region IN (SELECT region FROM user_regions WHERE user = current_user())
);

ALTER TABLE sales.transactions
SET ROW FILTER sales.region_filter;
```

Audit Logging

```
-- Monitor access
SELECT * FROM system.access.audit
WHERE table_name = 'customers'
ORDER BY event_time DESC
LIMIT 100;
```

8. BI Integration

Power BI Connection

1. **Get connection details**:
    - Warehouse → Connection Details → JDBC/ODBC
2. **Power BI Desktop**:
    - Get Data → SQL Server
    - Server: adb-<workspace>.azuredatabricks.net
    - Database: default

Tableau Setup

markdown

```
1. Connect using Databricks connector
2. Server: `adb-<workspace>.azuredatabricks.net`
3. Authentication: Personal Access Token
4. Catalog/Schema selection
```

9. Cost Management

Warehouse Monitoring

sql

```
-- Cost analysis
SELECT
  warehouse_id,
  SUM(credits_used) as total_credits
FROM system.billing.warehouse_usage
WHERE start_time > CURRENT_DATE() - INTERVAL 30 DAYS
GROUP BY warehouse_id;
```

Auto-Scaling

sql

```
-- Configure via API
PATCH /api/2.0/sql/warehouses/<id>
{
  "min_num_clusters": 1,
  "max_num_clusters": 3,
  "scaling_policy": {
    "enabled": true,
    "utilization_threshold": 75
  }
}
```

10. Troubleshooting

Issue	Solution
Slow queries	Check execution plan, optimize Z-ordering
Connection limits	Increase warehouse size/cluster count
Permission errors	Verify Unity Catalog grants
COPY INTO failures	Validate file permissions and formats

---

11. Learning Resources

- Databricks SQL Documentation
- SQL Reference Guide
- Performance Tuning