Datawarehousing on Databricks

Enabling migrations from legacy data warehouses

Data warehousing and BI - Hands on demo (30 mins)

1. This workshop is designed for all personas that use Datawarehouse technologies primarily with SQL and perform ad-hoc or exploratory data analysis.
2. We will also use advanced data warehouse techniques like stored procedures, multi-statement transactions and temporary tables which are used by data engineers
3. We will cover data catalog and governance techniques on Databricks using Unity Catalog
4. (Optional) We will connect data to BI tools to create visualizations and dashboards and give you a foundational understanding on what and how to leverage Databricks SQL

Data Migration enablement content (10 mins)

5. We will then cover the concepts of data migrations from legacy data warehouses to Databricks Data Intelligence platform
6. For self-implemented migrations: We will introduce tools like Lakebridge to facilitate self-managed migrations
7. For third-party managed migrations (turn-key): We will cover options available for you to leverage Databricks partnerships to facilitate the migration to the Databricks Data Intelligence platform

LakeBridge (30 mins)

8. We will go through the 4 part process, breaking down each, and giving screenshots to indicate what each step involves.
9. We will talk discuss how Lakebridge fits into the larger pattern of code migration and modernization
10. We will end with a discussion of the unvarnished truth on this feature. While extremely useful, it is not a substitute for a full migration or modernization process, and still has gaps which partners and other technical individuals will need to bridge.

Q&A - 15 mins

Note: This workshop doesn't go into implementation details or best practices of building a lakehouse or cloud data warehouse. For more information about deep dive topics please visit Databricks Academy or contact your Databricks representative.

Data warehousing and BI - Hands on demo agenda

- Create tables using Databricks SQL warehouse
- Run queries and create visualizations
- Run advanced data warehousing: Stored procedures, materialized views, transaction management
- Create and customize an AI/BI dashboard
- Parameterized queries
- Create a Genie space
- Refresh data and alerts

DBSQL features

Create tables using Databricks SQL warehouse

In this course, we'll be working with simulated data from an IOT device which is available as one of the default datasets in Databricks under a hidden volume called "databricks-datasets" .

We want to use data warehousing techniques using SQL first approach with stored procedures and multi-statement SQL code to facilitate SQL to SQL migration from legacy data warehouse to Databricks Lakehouse.

Note: Databricks offers full-fledged ETL capabilities as well. Please check out ETL features on Databricks to automate the data loading process.It enables the creation of smooth, automated data pipelines using native Apache Spark, Spark Declarative pipelines (DLT), and optimized SQL. This is out of scope of this workshop.

➕Create tables using SQL Editor

The SQL Editor is specifically designed for authoring, running, and managing SQL queries with features like autocomplete, autoformatting, and query tabs, making it ideal for users focused on SQL analytics and data exploration within a dedicated interface. It allows users to save, schedule, and share queries easily. On the other hand, Notebooks offer a more versatile environment that supports multiple languages (such as SQL, Python, and Scala), enabling complex data processing workflows that integrate SQL with other programming tasks.

STEP 1: Create the catalog, database from existing Databricks datasets. Run the below code in SQL editor

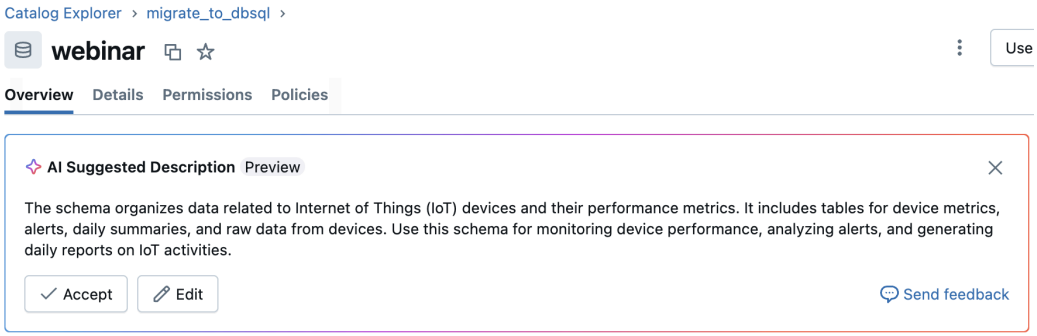
```
USE CATALOG migrate_to_dbsql;
USE SCHEMA webinar;


CREATE OR REPLACE TABLE iot_device_raw AS
```

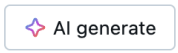
```sql
SELECT * FROM json.`/databricks-datasets/iot-stream/data-device/`;

CREATE OR REPLACE TABLE iot_device_metrics (
 device_id STRING,
 user_id STRING,
 timestamp TIMESTAMP,
 num_steps INT,
 calories_burnt DOUBLE,
 miles_walked DOUBLE
)
TBLPROPERTIES ('delta.feature.catalogOwned-preview' = 'supported');

CREATE OR REPLACE TABLE iot_daily_summary (
 device_id STRING,
 user_id STRING,
 date DATE,
 total_steps INT,
 total_miles DOUBLE,
 total_calories DOUBLE
)
TBLPROPERTIES ('delta.feature.catalogOwned-preview' = 'supported');

CREATE OR REPLACE TABLE iot_alerts (
 alert_id BIGINT GENERATED ALWAYS AS IDENTITY,
 device_id STRING,
 alert_type STRING,
 alert_message STRING,
 created_at TIMESTAMP
)
TBLPROPERTIES ('delta.feature.catalogOwned-preview' = 'supported');

CREATE TABLE migrate_to_dbsql.webinar.device_access (
 device_id STRING,
 user_email STRING
);
```

STEP 2: Use AI-Generate to create metadata for the tables

Table metadata is important for intelligent features like databricks assistant, personalized search, genie spaces etc. to perform better and give relevant results

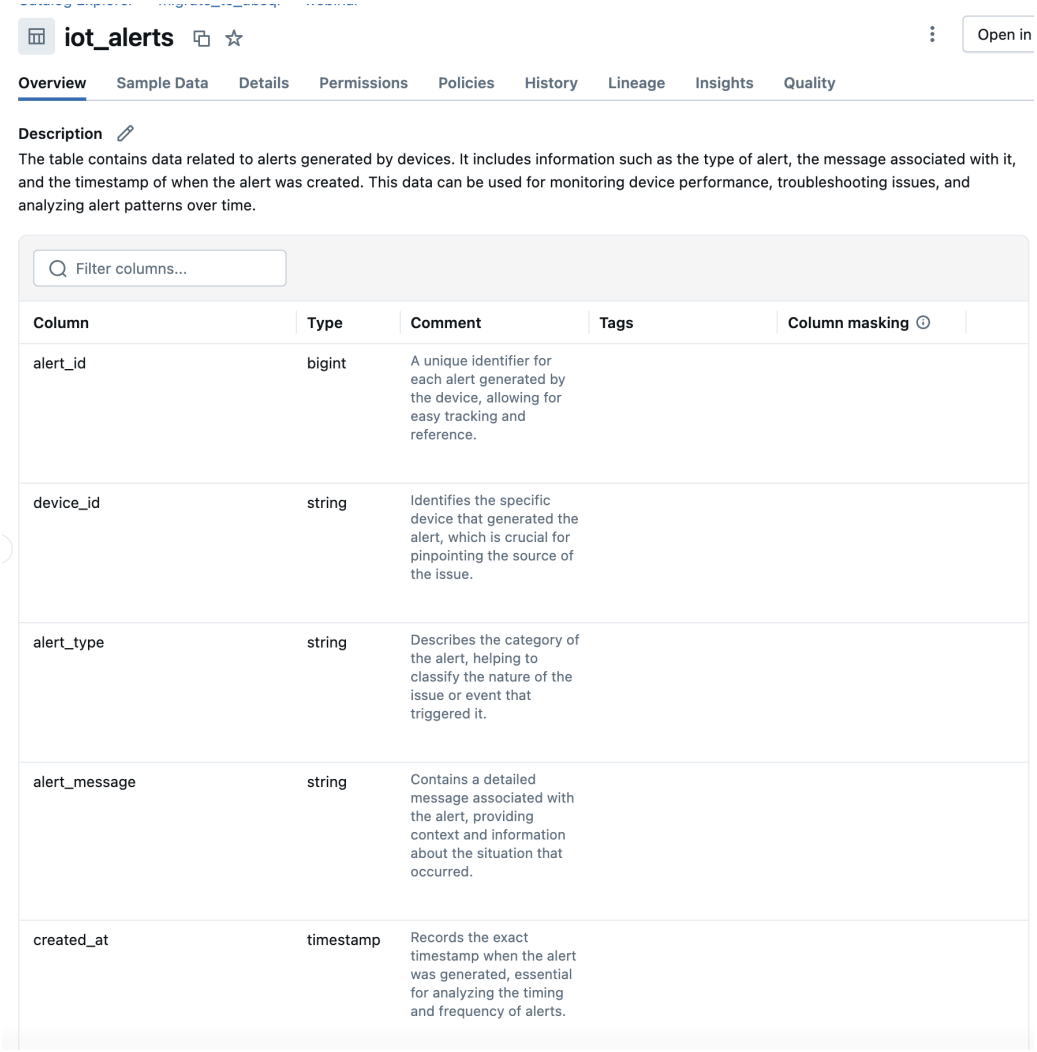
- For the webinar schema, generate description using AI



- Click on 

to generate metadata for all the columns. Edit and save the metadata generated for each of the columns

- For each table that got created in `webinar` schema, edit or accept the AI suggested description of the table



STEP 3: Load datasets

- Different than Multi-Statement/Multi-Table
- Simple statement - will produce output
- Run from Query editor against SQL Warehouse
- → Set Catalog / Warehouse at top of the query, USE statements did not work

```
BEGIN
  DECLARE rows_inserted INT DEFAULT 0;
  DECLARE alerts_created INT DEFAULT 0;
  DECLARE process_date DATE DEFAULT current_date();
```

```
  -- Insert cleaned data into metrics table
  INSERT INTO iot_device_metrics
  SELECT
    device_id,
    user_id,
    CAST(timestamp AS TIMESTAMP) AS timestamp,
    num_steps,
    calories_burnt,
    miles_walked
  FROM iot_device_raw
  WHERE device_id IS NOT NULL
    AND num_steps > 0;

  SET rows_inserted = (SELECT COUNT(*) FROM iot_device_metrics);

  -- Aggregate to daily summary
  INSERT OVERWRITE iot_daily_summary
  SELECT
    device_id,
    user_id,
    DATE(timestamp) AS date,
    SUM(num_steps) AS total_steps,
    SUM(miles_walked) AS total_miles,
    SUM(calories_burnt) AS total_calories
  FROM iot_device_metrics
  GROUP BY device_id, user_id, DATE(timestamp);

  -- Create alerts for high activity devices
  INSERT INTO iot_alerts (device_id, alert_type, alert_message, created_at)
  SELECT
    device_id,
    'HIGH_ACTIVITY' AS alert_type,
    CONCAT('Device ', device_id, ' recorded ', total_steps, ' steps on ', date)
AS alert_message,
    current_timestamp() AS created_at
```

```
  FROM iot_daily_summary
  WHERE total_steps > 15000;

  SET alerts_created = (SELECT COUNT(*) FROM iot_alerts);

  -- Create users with email Ids. Add your email IDs mapped to a device_id
  INSERT INTO migrate_to_dbsql.webinar.device_access (device_id, user_email)
VALUES
  ('1', 'alice@example.com'),
  ('2', 'bob@example.com'),
  ('3', 'carol@example.com'),
  ('4', 'dave@example.com'),
  ('5', 'abhijit.tilak@databricks.com'),
  ('6', 'eric.begg@databricks.com'),
  ('7', 'zachary.ryan@databricks.com'),
  ('8', 'heidi@example.com'),
  ('9', 'ivan@example.com');
  -- Display summary
  SELECT
    rows_inserted AS metrics_processed,
    alerts_created AS high_activity_alerts,
    process_date AS processing_date;
END;
```

STEP 4: CREATE STORED PROCEDURES IN UC catalog and schema

- Users can create - no need to execute but we can have a Job defined that will call the function, example
  - `CALL <catalog>.<schema>.<procedure_name>(param1, param2)`
- This will create a Function object in UC - so it is governed and controlled - plus how might we tie this into a conversation on UC Metrics
- Run from Query editor against SQL Warehouse
- → Set Catalog / Warehouse at top of the query, USE statements did not work
- Note - SQL SECURITY INVOKER - received errors when not declared

```
CREATE OR REPLACE PROCEDURE process_iot_daily(
 IN target_date DATE,
 OUT records_processed INT
)
SQL SECURITY INVOKER
LANGUAGE SQL
BEGIN
 INSERT INTO iot_daily_summary
 SELECT device_id, user_id, DATE(timestamp),
        SUM(num_steps), SUM(miles_walked), SUM(calories_burnt)
 FROM iot_device_metrics
 WHERE DATE(timestamp) = target_date
 GROUP BY ALL;

 SET records_processed = (
   SELECT COUNT(*)
   FROM iot_daily_summary
   WHERE date = target_date
 );
END;
```

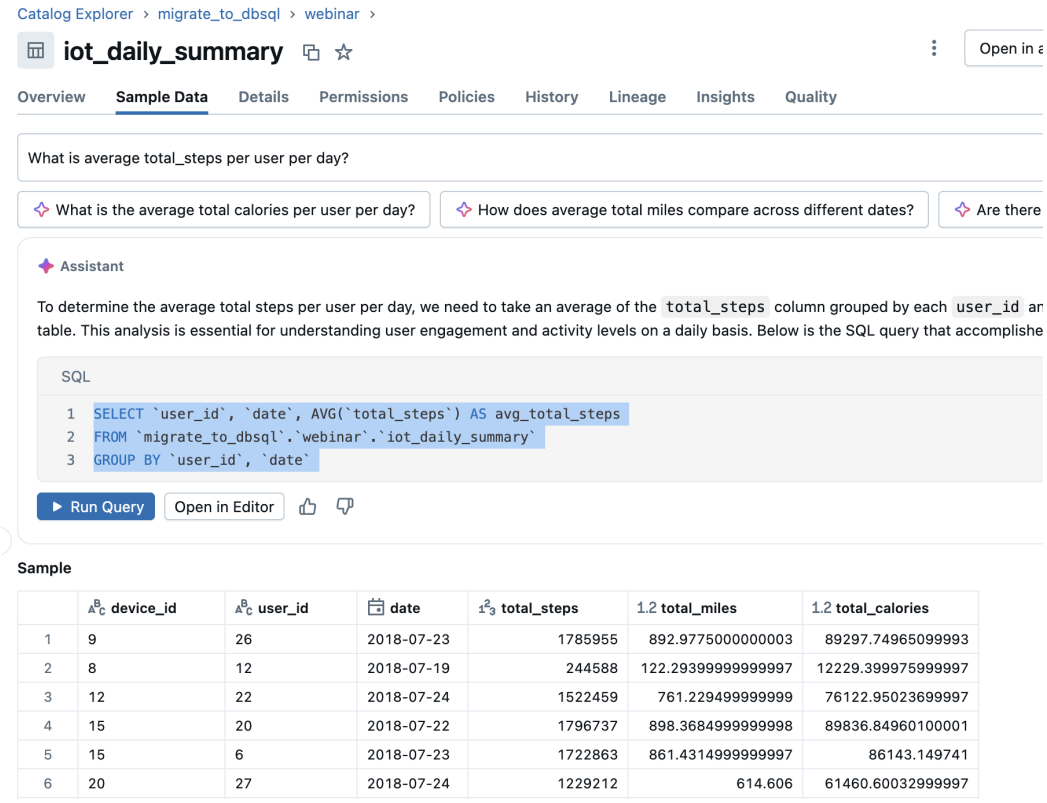
🏃Running Queries for adhoc analytics

Let's try to answer this question with a SQL query:

Navigate to the catalog: migrate_to_dbsql, schema: webinar, table: iot_daily_summary.

Q1: What is the average total_steps per user per day?

Databricks in-built genie assistant at table level can answer this question without you needing to know SQL.

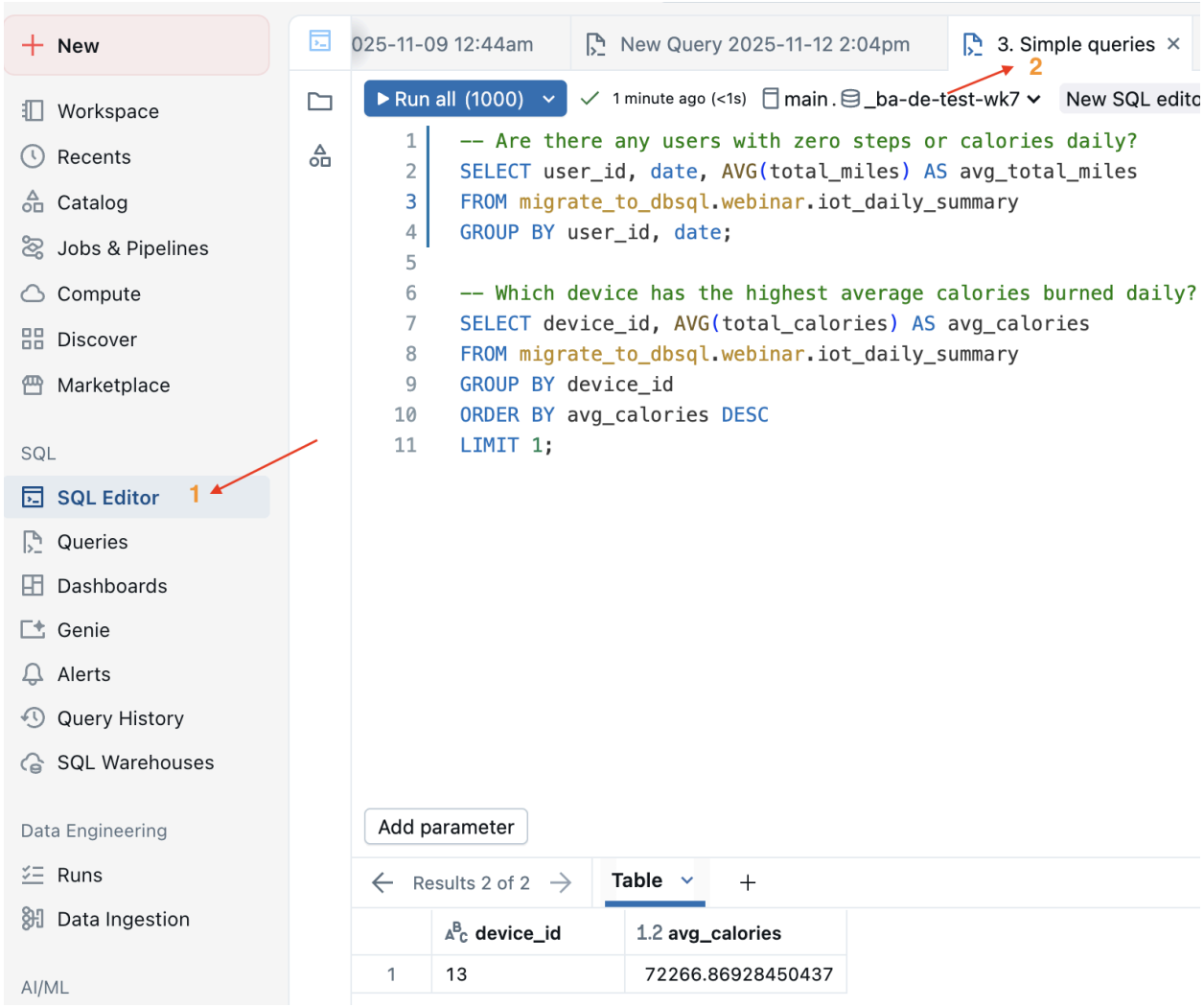


Sample

	device_id	user_id	date	total_steps	total_miles	total_calories
1	9	26	2018-07-23	1785955	892.9775000000003	89297.74965099993
2	8	12	2018-07-19	244588	122.29399999999997	12229.399975999997
3	12	22	2018-07-24	1522459	761.229499999999	76122.95023699997
4	15	20	2018-07-22	1796737	898.3684999999998	89836.84960100001
5	15	6	2018-07-23	1722863	861.4314999999997	86143.149741
6	20	27	2018-07-24	1229212	614.606	61460.60032999997

Q2: Are there any users with zero steps or calories daily?

```
SELECT `user_id`, `date`
FROM `migrate_to_dbsql`.`webinar`.`iot_daily_summary`
WHERE `total_steps` = 0 OR `total_calories` = 0
```


New
Workspace
Recents
Catalog
Jobs & Pipelines
Compute
Discover
Marketplace
SQL
SQL Editor
1
Queries
Dashboards
Genie
Alerts
Query History
SQL Warehouses
Data Engineering
Runs
Data Ingestion
AI/ML
025-11-09 12:44am
New Query 2025-11-12 2:04pm
3. Simple queries
2
Run all (1000)
1 minute ago (<1s)
main . _ba-de-test-wk7
New SQL editc
-- Are there any users with zero steps or calories daily?
SELECT user_id, date, AVG(total_miles) AS avg_total_miles
FROM migrate_to_dbsql.webinar.iot_daily_summary
GROUP BY user_id, date;
-- Which device has the highest average calories burned daily?
SELECT device_id, AVG(total_calories) AS avg_calories
FROM migrate_to_dbsql.webinar.iot_daily_summary
GROUP BY device_id
ORDER BY avg_calories DESC
LIMIT 1;
Add parameter
Results 2 of 2
Table
device_id
avg_calories
1
13
72266.86928450437

Compound Statements

- Doing analysis on morning records via timestamp
- Highlights if...then, merge
- - ✅ Variables - morning_records, avg_morning_steps, target_date
- - ✅ IF/THEN/ELSE - Conditional processing based on data availability
- - ✅ SET statements - Dynamic variable assignments
- - ✅ Time-based filtering - Focus on 6-9 AM window
- - ✅ INSERT, MERGE, SELECT - Multiple SQL operations
- - ✅ Business logic - Alerts for above-average activity

```
BEGIN
  DECLARE rows_inserted INT DEFAULT 0;
  DECLARE alerts_created INT DEFAULT 0;
  DECLARE process_date DATE DEFAULT current_date();

  -- Insert cleaned data into metrics table
  INSERT INTO iot_device_metrics
  SELECT
    device_id,
    user_id,
    CAST(timestamp AS TIMESTAMP) AS timestamp,
    num_steps,
    calories_burnt,
    miles_walked
  FROM iot_device_raw
  WHERE device_id IS NOT NULL
    AND num_steps > 0;

  SET rows_inserted = (SELECT COUNT(*) FROM iot_device_metrics);

  -- Aggregate to daily summary
  INSERT OVERWRITE iot_daily_summary
  SELECT
    device_id,
    user_id,
    DATE(timestamp) AS date,
```

```
    SUM(num_steps) AS total_steps,
    SUM(miles_walked) AS total_miles,
    SUM(calories_burnt) AS total_calories
  FROM iot_device_metrics
  GROUP BY device_id, user_id, DATE(timestamp);

  -- Create alerts for high activity devices
  INSERT INTO iot_alerts (device_id, alert_type, alert_message, created_at)
  SELECT
    device_id,
    'HIGH_ACTIVITY' AS alert_type,
    CONCAT('Device ', device_id, ' recorded ', total_steps, ' steps on ', date)
AS alert_message,
    current_timestamp() AS created_at
  FROM iot_daily_summary
  WHERE total_steps > 15000;

  SET alerts_created = (SELECT COUNT(*) FROM iot_alerts);

  -- Display summary
  SELECT
    rows_inserted AS metrics_processed,
    alerts_created AS high_activity_alerts,
    process_date AS processing_date;
END;
```

Multi-Statement/Multi-Table (Private Preview Feature)

STEP 4: Multi-statement multi-table statements

✅ Multi-Statement: 3 separate SQL statements (INSERT, MERGE, INSERT)
✅ Multi-Table: Touches 4 tables:
- iot_device_raw (reads)
- iot_device_metrics (writes)
- iot_daily_summary (writes)
- iot_alerts (writes)

✅ Atomic: BEGIN ATOMIC ... END ensures:
- All 3 statements succeed together, OR
- All 3 statements rollback together (none commit)

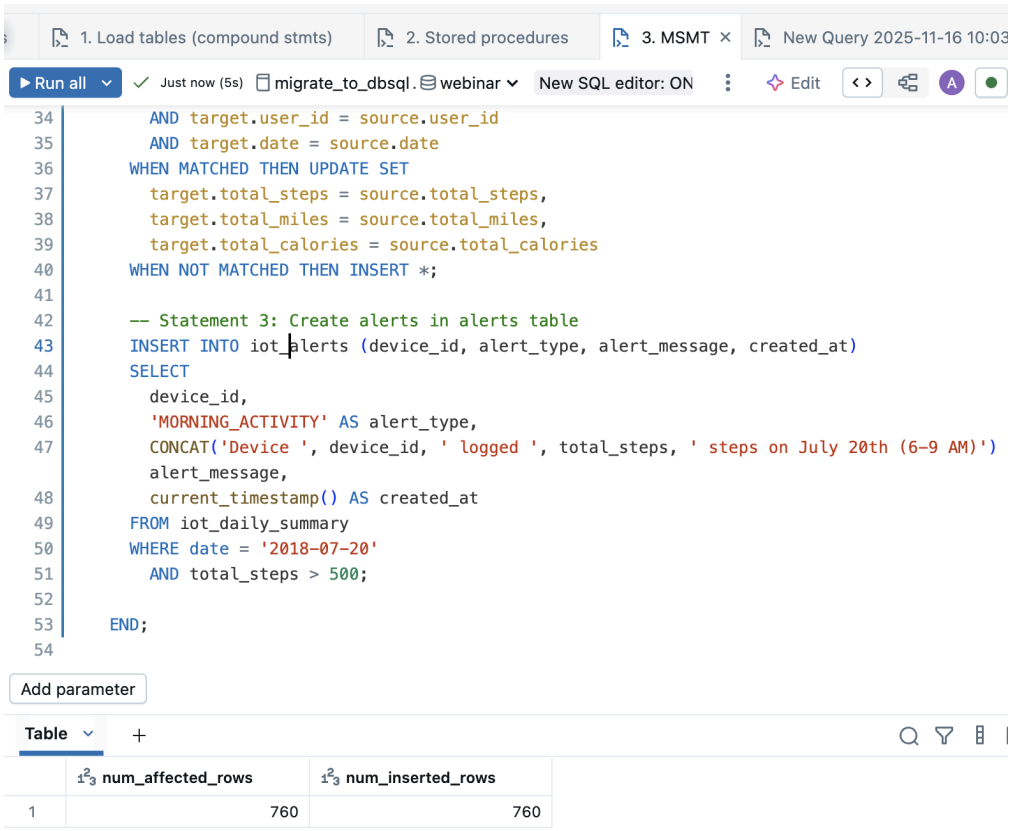
Why does this matter for Warehouse Migrations:

This demonstrates ACID transaction guarantees that traditional data warehouses provide:
- If the alert creation fails, the metrics and summary inserts are rolled back
- Data consistency is maintained across all 3 tables
- No partial updates or orphaned records

```
BEGIN ATOMIC

  -- Statement 1: Insert morning activity into metrics table
  INSERT INTO iot_device_metrics
  SELECT
    device_id,
    user_id,
    CAST(timestamp AS TIMESTAMP) AS timestamp,
    num_steps,
    calories_burnt,
    miles_walked
  FROM iot_device_raw
  WHERE device_id IS NOT NULL
    AND num_steps > 0
```

```
    AND DATE(CAST(timestamp AS TIMESTAMP)) = '2018-07-20'
    AND HOUR(CAST(timestamp AS TIMESTAMP)) BETWEEN 6 AND 8;

  -- Statement 2: Update daily summary table
  MERGE INTO iot_daily_summary AS target
  USING (
    SELECT
      device_id,
      user_id,
      DATE(timestamp) AS date,
      SUM(num_steps) AS total_steps,
      SUM(miles_walked) AS total_miles,
      SUM(calories_burnt) AS total_calories
    FROM iot_device_metrics
    WHERE DATE(timestamp) = '2018-07-20'
      AND HOUR(timestamp) BETWEEN 6 AND 8
    GROUP BY device_id, user_id, DATE(timestamp)
  ) AS source
  ON target.device_id = source.device_id
    AND target.user_id = source.user_id
    AND target.date = source.date
  WHEN MATCHED THEN UPDATE SET
    target.total_steps = source.total_steps,
    target.total_miles = source.total_miles,
    target.total_calories = source.total_calories
  WHEN NOT MATCHED THEN INSERT *;

  -- Statement 3: Create alerts in alerts table
  INSERT INTO iot_alerts (device_id, alert_type, alert_message, created_at)
  SELECT
    device_id,
    'MORNING_ACTIVITY' AS alert_type,
    CONCAT('Device ', device_id, ' logged ', total_steps, ' steps on July
20th (6-9 AM)') AS alert_message,
    current_timestamp() AS created_at
```

```
  FROM iot_daily_summary
  WHERE date = '2018-07-20'
    AND total_steps > 500;

END;
```



Review differences between compound statement and Multi-statment multi-table statements

Compare MultiStatement (PrPr) and Compound Statements

Compare Table

Feature	Compound Statement	Multi-Statement Transaction
Syntax	BEGIN ... END	BEGIN ATOMIC ... END
Atomicity	❌ Each statement commits independently	✅ All-or-nothing execution
Rollback	❌ Can't rollback - if statement 2 fails,	✅ Auto-rollback - if any statement fails,

	statement 1 stays committed	ALL rollback
Variables	✅ Supported	✅ Supported
Control Flow	✅ IF/CASE/WHILE/FOR/LOOP	✅ IF/CASE/WHILE/FOR/LOOP
Stored in UC	❌ Ad-hoc only (not persisted)	❌ Ad-hoc only (not persisted)
Availability	✅ Generally available	⚠️ Limited availability contact account team)
Use Case	ETL pipelines, data processing workflows	Critical multi-table updates requirig consistency

How Stored Procedure Fit In

Feature	Compound Statement	Multi-Statement Transaction	Stored Procedure
Syntax	BEGIN ... END	BEGIN ATOMIC ... END	CREATE PROCEDURE ... BEGIN ... END
Atomicity	❌ No	✅ Yes	✅ Yes (if body uses ATOMIC)
Stored in UC	❌ No	❌ No	✅✅ YES
Reusable	❌ Copy/paste	❌ Copy/paste	✅ CALL procedure_name()
UC Governance	❌ Only on tables	❌ Only on tables	✅ On procedure + tables
Parameters	❌ No	❌ No	✅ IN/OUT/INOUT
Versioning	❌ No	❌ No	✅ CREATE OR REPLACE
UC Lineage	Table-level only	Transaction-level	Procedure → Table lineage
UC Audit	Table operations	Transaction operations	Procedure execution + table ops

Now that you know how to run a query and create a visualization, let's try to replicate a few things you might usually do in Excel.

Advanced Data Warehousing features — workshop material

UC metric views: operational and governance insights

Build simple "metric views" from information_schema to track objects and grants.

```sql
-- Count objects by type in a catalog
SELECT table_type, COUNT(*) AS objects
```

```
FROM my_catalog.information_schema.tables
GROUP BY table_type
ORDER BY objects DESC;
```

SQL

```
-- Who has SELECT on what (table-level)
SELECT table_schema, table_name, grantee, privilege_type
FROM my_catalog.information_schema.table_privileges
WHERE privilege_type = 'SELECT'
ORDER BY table_schema, table_name, grantee;
```

SQL

```
-- Create a reusable “governance metrics” view
CREATE OR REPLACE VIEW
my_catalog.webinar.uc_governance_metrics AS
SELECT 'tables' AS metric, COUNT(*) AS value FROM
my_catalog.information_schema.tables
UNION ALL
SELECT 'views', COUNT(*) FROM
my_catalog.information_schema.views
UNION ALL
SELECT 'materialized_views', COUNT(*) FROM
my_catalog.information_schema.materialized_views;
```

Variant datatype

Show both a VARIANT-first approach and a portable JSON fallback.

SQL

```
-- VARIANT (if enabled in your workspace)
CREATE OR REPLACE TABLE my_catalog.webinar.events_v
(payload VARIANT);
```

```
INSERT INTO my_catalog.webinar.events_v VALUES
(parse_json('{"user_id":42,"attrs":{"country":"US","tier":"
gold"}}'));

SELECT
  CAST(payload:user_id AS INT)            AS user_id,
  CAST(payload:attrs.country AS STRING)   AS country,
  CAST(payload:attrs.tier AS STRING)      AS tier
FROM my_catalog.webinar.events_v;
```

SQL

```
-- Fallback without VARIANT: store string JSON and use
from_json with schema
CREATE OR REPLACE TABLE my_catalog.webinar.events_json
(payload STRING);

INSERT INTO my_catalog.webinar.events_json VALUES
('{"user_id":42,"attrs":{"country":"US","tier":"gold"}}');

WITH typed AS (
  SELECT from_json(
           payload,
           'user_id INT, attrs STRUCT<country: STRING,
tier: STRING>'
         ) AS j
  FROM my_catalog.webinar.events_json
)
SELECT j.user_id, j.attrs.country AS country, j.attrs.tier
AS tier
FROM typed;
```

SQL stored procedures

An idempotent daily upsert procedure using MERGE and an OUT parameter.

```sql
CREATE OR REPLACE PROCEDURE
my_catalog.webinar.upsert_daily_summary(
  IN target_date DATE,
  OUT rows_affected INT
)
SQL SECURITY INVOKER
LANGUAGE SQL
BEGIN
  MERGE INTO my_catalog.webinar.iot_daily_summary AS t
  USING (
    SELECT device_id, user_id, DATE(timestamp) AS date,
           SUM(num_steps) AS total_steps,
           SUM(miles_walked) AS total_miles,
           SUM(calories_burnt) AS total_calories
    FROM my_catalog.webinar.iot_device_metrics
    WHERE DATE(timestamp) = target_date
    GROUP BY device_id, user_id, DATE(timestamp)
  ) s
  ON t.device_id = s.device_id AND t.user_id = s.user_id
AND t.date = s.date
  WHEN MATCHED THEN UPDATE SET
    total_steps = s.total_steps,
    total_miles = s.total_miles,
    total_calories = s.total_calories
  WHEN NOT MATCHED THEN INSERT *;

  SET rows_affected = (SELECT COUNT(*) FROM
my_catalog.webinar.iot_daily_summary WHERE date =
target_date);
END;
```

```sql
-- Example call
CALL my_catalog.webinar.upsert_daily_summary('2016-07-19',
rows_affected => ?);
```

Temporary tables/views

Session-scoped working sets for exploration without polluting the schema.

```sql
-- Temp view for last 7 days of metrics
CREATE OR REPLACE TEMP VIEW last7_metrics AS
SELECT *
FROM my_catalog.webinar.iot_device_metrics
WHERE DATE(timestamp) >= date_sub(current_date(), 7);

-- Use it in follow-on queries
SELECT device_id, SUM(num_steps) AS steps_7d
FROM last7_metrics
GROUP BY device_id
ORDER BY steps_7d DESC
LIMIT 10;
```

Multi-statement transactions (concise exercise)

Atomic multi-table change with BEGIN ATOMIC.

```sql
BEGIN ATOMIC
  -- Example date param
  SET target_date = DATE('2016-07-19');

  INSERT INTO my_catalog.webinar.iot_device_metrics
```

```
  SELECT device_id, user_id, CAST(timestamp AS TIMESTAMP),
num_steps, calories_burnt, miles_walked
  FROM my_catalog.webinar.iot_device_raw
  WHERE DATE(CAST(timestamp AS TIMESTAMP)) = target_date
    AND num_steps > 0;

  MERGE INTO my_catalog.webinar.iot_daily_summary AS t
  USING (
    SELECT device_id, user_id, DATE(timestamp) AS date,
           SUM(num_steps) AS total_steps,
           SUM(miles_walked) AS total_miles,
           SUM(calories_burnt) AS total_calories
    FROM my_catalog.webinar.iot_device_metrics
    WHERE DATE(timestamp) = target_date
    GROUP BY device_id, user_id, DATE(timestamp)
  ) s
  ON t.device_id = s.device_id AND t.user_id = s.user_id
AND t.date = s.date
  WHEN MATCHED THEN UPDATE SET
    total_steps = s.total_steps,
    total_miles = s.total_miles,
    total_calories = s.total_calories
  WHEN NOT MATCHED THEN INSERT *;

  INSERT INTO my_catalog.webinar.iot_alerts (device_id,
alert_type, alert_message, created_at)
  SELECT device_id,
         'MORNING_ACTIVITY',
         CONCAT('Device ', device_id, ' logged ',
total_steps, ' steps on ', CAST(target_date AS STRING)),
         current_timestamp()
  FROM my_catalog.webinar.iot_daily_summary
  WHERE date = target_date AND total_steps > 500;
END;
```

Recursive CTE

Hierarchical drill-up or sequence generation.

```sql
-- Category hierarchy example
-- my_catalog.webinar.categories(id INT, parent_id INT,
name STRING)

WITH RECURSIVE hierarchy AS (
  SELECT id, parent_id, name, CAST(name AS STRING) AS path,
0 AS depth
  FROM my_catalog.webinar.categories
  WHERE parent_id IS NULL
  UNION ALL
  SELECT c.id, c.parent_id, c.name,
         CONCAT(h.path, ' > ', c.name) AS path,
         h.depth + 1 AS depth
  FROM my_catalog.webinar.categories c
  JOIN hierarchy h ON c.parent_id = h.id
)
SELECT * FROM hierarchy ORDER BY path;
```

(Optional): Replicate things usually done in Excel

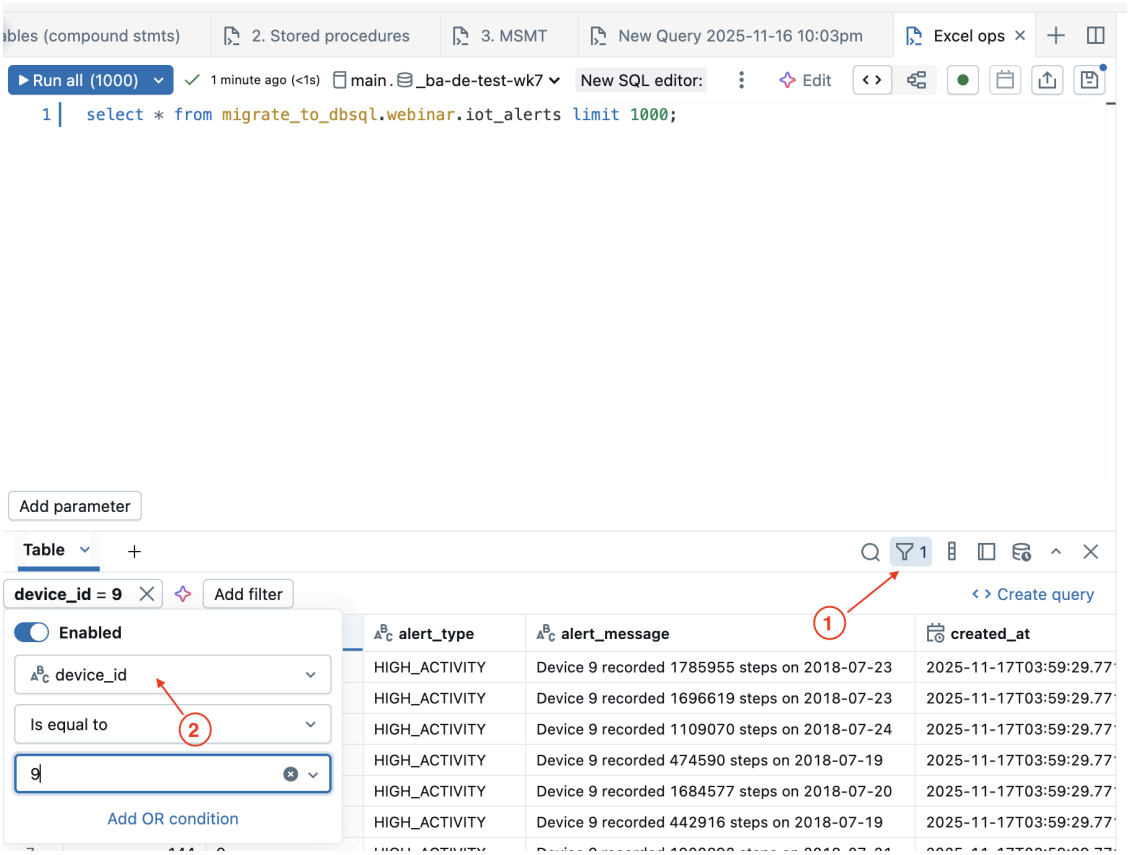
Viewing Data in a table

```
select * from migrate_to_dbsql.webinar.iot_alerts limit 100;
```

- Explore the options which show up in the UI when you can see the data in the table
    - Filtering
    - Sorting
    - Pinning Columns

Filtering data

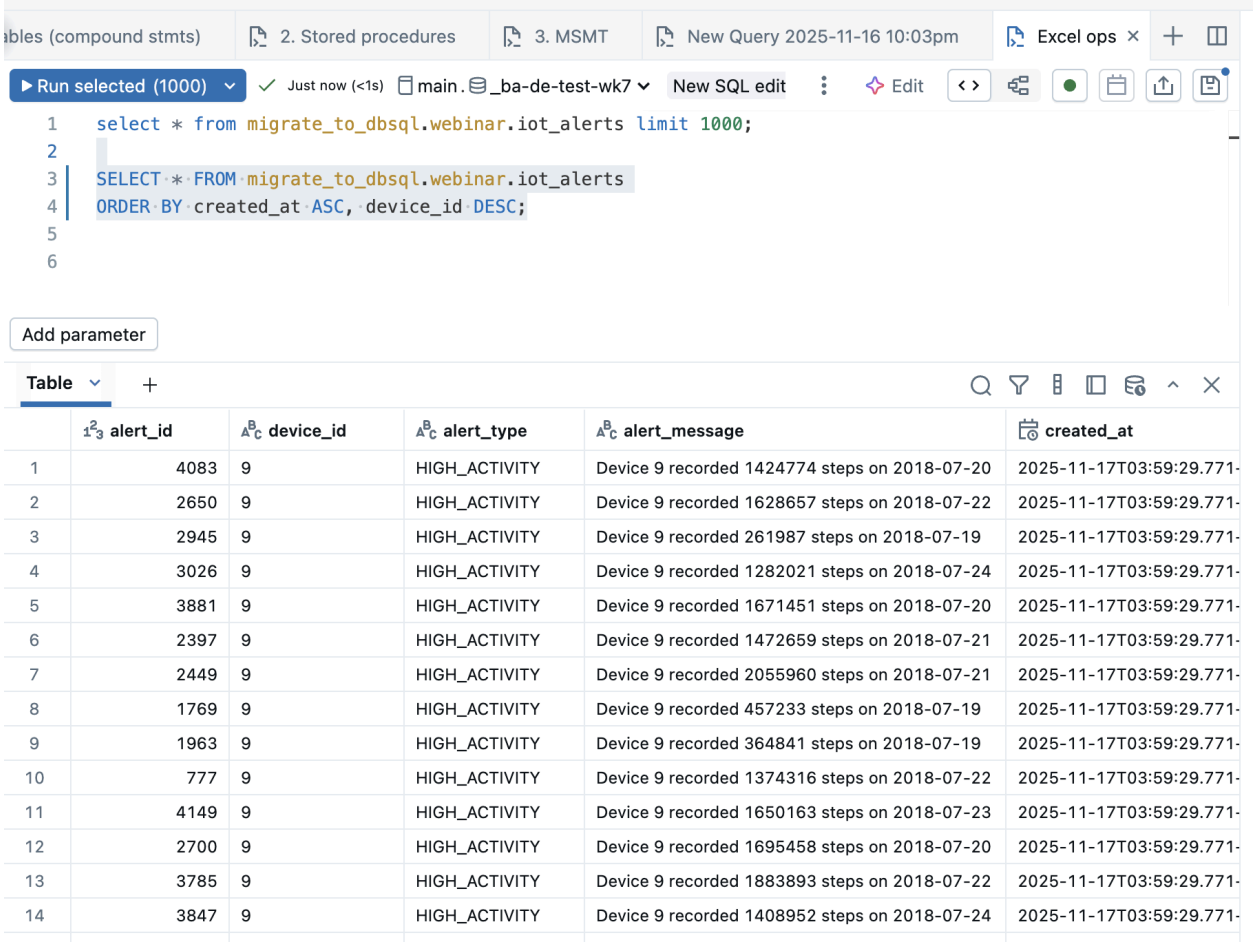
1. Filter by device_id = 9 using visual SQL editor



Sorting data

- This can be done through the UI after you run a query, but doing it in the SQL query is more permanent

```
SELECT * FROM migrate_to_dbsql.webinar.iot_alerts
ORDER BY created_at ASC, device_id DESC;
```



	alert_id	device_id	alert_type	alert_message	created_at
1	4083	9	HIGH_ACTIVITY	Device 9 recorded 1424774 steps on 2018-07-20	2025-11-17T03:59:29.771-
2	2650	9	HIGH_ACTIVITY	Device 9 recorded 1628657 steps on 2018-07-22	2025-11-17T03:59:29.771-
3	2945	9	HIGH_ACTIVITY	Device 9 recorded 261987 steps on 2018-07-19	2025-11-17T03:59:29.771-
4	3026	9	HIGH_ACTIVITY	Device 9 recorded 1282021 steps on 2018-07-24	2025-11-17T03:59:29.771-
5	3881	9	HIGH_ACTIVITY	Device 9 recorded 1671451 steps on 2018-07-20	2025-11-17T03:59:29.771-
6	2397	9	HIGH_ACTIVITY	Device 9 recorded 1472659 steps on 2018-07-21	2025-11-17T03:59:29.771-
7	2449	9	HIGH_ACTIVITY	Device 9 recorded 2055960 steps on 2018-07-21	2025-11-17T03:59:29.771-
8	1769	9	HIGH_ACTIVITY	Device 9 recorded 457233 steps on 2018-07-19	2025-11-17T03:59:29.771-
9	1963	9	HIGH_ACTIVITY	Device 9 recorded 364841 steps on 2018-07-19	2025-11-17T03:59:29.771-
10	777	9	HIGH_ACTIVITY	Device 9 recorded 1374316 steps on 2018-07-22	2025-11-17T03:59:29.771-
11	4149	9	HIGH_ACTIVITY	Device 9 recorded 1650163 steps on 2018-07-23	2025-11-17T03:59:29.771-
12	2700	9	HIGH_ACTIVITY	Device 9 recorded 1695458 steps on 2018-07-20	2025-11-17T03:59:29.771-
13	3785	9	HIGH_ACTIVITY	Device 9 recorded 1883893 steps on 2018-07-22	2025-11-17T03:59:29.771-
14	3847	9	HIGH_ACTIVITY	Device 9 recorded 1408952 steps on 2018-07-24	2025-11-17T03:59:29.771-

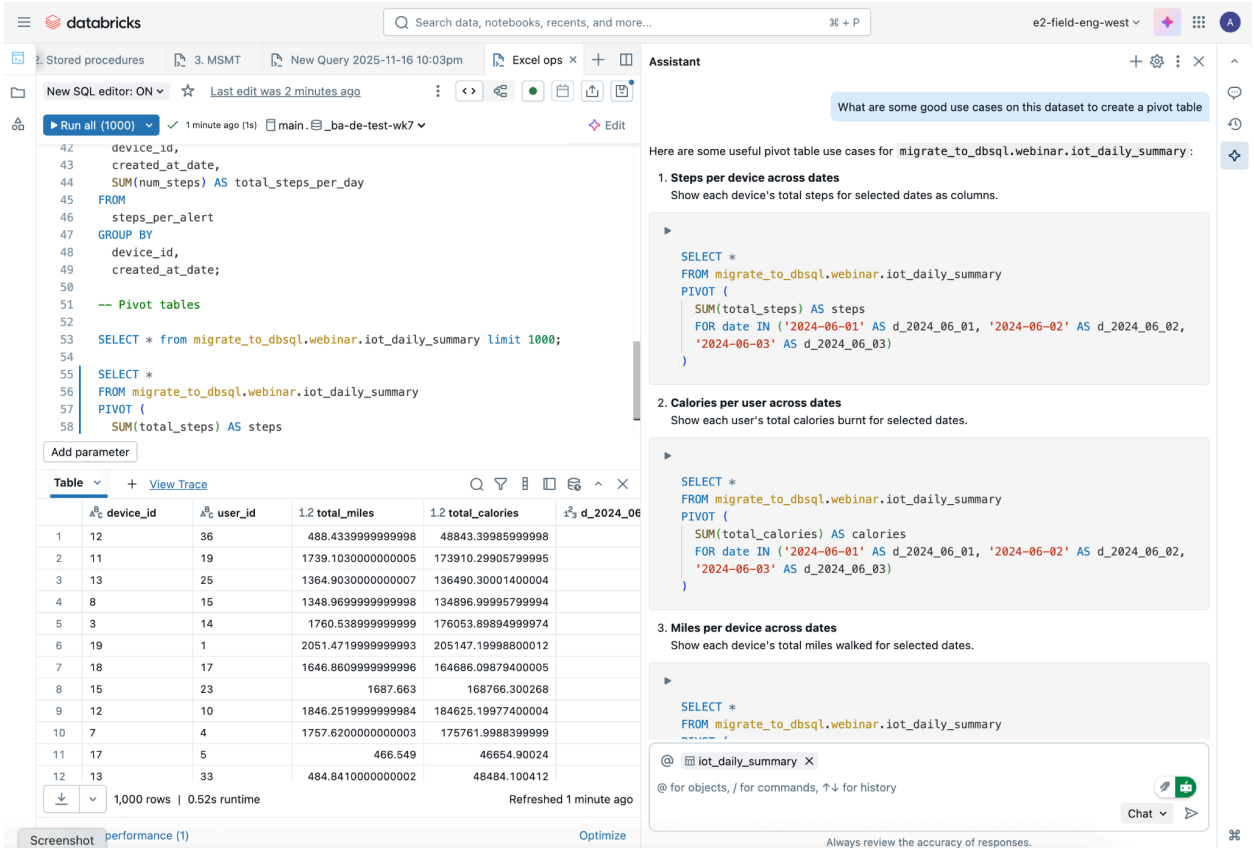
Aggregations (Sum, Average, Count)

```
-- Calculate total steps per day per device
select
 user_id,
 date_format(date, 'yyyy-MM-dd') as date_dt,
 sum(total_steps) as steps
from
 migrate_to_dbsql.webinar.iot_daily_summary
```

```
group by
 user_id,
 date_dt
order by
 user_id,
 date_dt;
```

Pivot Tables

- Run the query "Select * from `migrate_to_dbsql.webinar.iot_daily_summary` LIMIT 100;" and wait for the results
- Try asking Databricks Assistant "What are some good use cases on this dataset to create a pivot table". Explore running some of these queries
- We use the PIVOT clause from DBSQL to run these queries



VLOOKUP

- In SQL, the equivalent of Excel's VLOOKUP function is typically achieved using JOIN operations

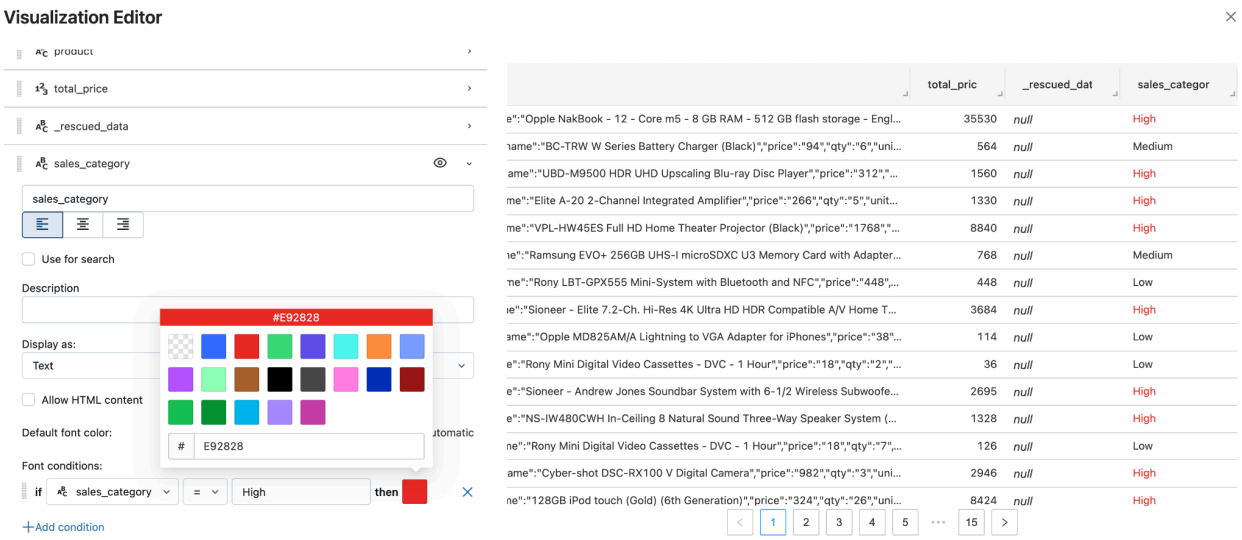
```
--Total sales by loyalty segment
SELECT c.loyalty_segment, SUM(sg.total_price) AS total_sales
   FROM dbsql_migration_enablement.retail360.sales_gold sg
   JOIN dbsql_migration_enablement.retail360.customers c ON sg.customer_id =
c.customer_id
   GROUP BY c.loyalty_segment
```

Conditional Formatting

- Can be done through the UI, after writing SQL Code

```
-- Bucket total sales into high (>1000), medium (>500) and low (<500)
SELECT *,
 CASE
   WHEN total_price > 1000 THEN 'High'
   WHEN total_price BETWEEN 500 AND 1000 THEN 'Medium'
   ELSE 'Low'
 END as sales_category
FROM dbsql_migration_enablement.retail360.sales_gold
```

- Create a visualization with visualization type “Table” and add font conditions to the required columns



Explore creating relevant visualizations for all the above queries.

🏛️Data Governance with Unity Catalog

GRANT statements (Unity Catalog)

Use groups synced from your IdP. Replace placeholders with your catalog and schema.

```sql
-- Catalog and schema access
GRANT USAGE ON CATALOG migrate_to_dbsql TO `analysts`;
GRANT USAGE ON SCHEMA  migrate_to_dbsql.webinar TO `analysts`;

-- Table-level permissions
GRANT SELECT ON TABLE migrate_to_dbsql.webinar.iot_daily_summary TO
`analysts`;
GRANT SELECT ON TABLE migrate_to_dbsql.webinar.iot_device_metrics TO
`analysts`;

-- Limit write to engineering only
GRANT MODIFY ON TABLE migrate_to_dbsql.webinar.iot_device_metrics TO
`data_engineers`;
```

```
-- Revoke example
REVOKE MODIFY ON TABLE migrate_to_dbsql.webinar.iot_device_metrics FROM
`analysts`;
```

Row-level security (RLS) via dynamic view

Show only rows the current user is entitled to see based on a mapping table.

```
SQL
-- Access mapping: which devices each user can see
-- migrate_to_dbsql.webinar.device_access(user_email
STRING, device_id STRING)

CREATE OR REPLACE VIEW migrate_to_dbsql.webinar.iot_daily_summary_rls AS
SELECT s.*
FROM migrate_to_dbsql.webinar.iot_daily_summary s
JOIN migrate_to_dbsql.webinar.device_access a
 ON a.device_id = s.device_id
WHERE LOWER(a.user_email) = LOWER(current_user());

-- Grant access to the RLS view (not the base table)
GRANT SELECT ON VIEW migrate_to_dbsql.webinar.iot_daily_summary_rls TO
`analysts`;
```

Column-level security (CLS) via masking in a view

Mask sensitive columns unless the user is in a privileged group.

```
SQL
CREATE OR REPLACE VIEW
migrate_to_dbsql.webinar.iot_device_metrics_cls AS
SELECT
```

```
  device_id,
  user_id,
  timestamp,
  CASE
    WHEN is_member('pii-cleared') THEN num_steps
    ELSE NULL
  END AS num_steps,
  CASE
    WHEN is_member('pii-cleared') THEN calories_burnt
    ELSE NULL
  END AS calories_burnt,
  miles_walked
FROM migrate_to_dbsql.webinar.iot_device_metrics;

GRANT SELECT ON VIEW
migrate_to_dbsql.webinar.iot_device_metrics_cls TO
`analysts`;
```

ABAC (attribute-based access control) using groups/tags

A practical pattern is to use groups as attributes and encode policies in views.

SQL

```
-- Region-restricted access: a user must be in the matching
region group
-- migrate_to_dbsql.webinar.user_regions(user_email STRING,
region STRING)

CREATE OR REPLACE VIEW
migrate_to_dbsql.webinar.iot_daily_summary_abac AS
SELECT s.*
FROM migrate_to_dbsql.webinar.iot_daily_summary s
JOIN migrate_to_dbsql.webinar.user_regions ur
  ON LOWER(ur.user_email) = LOWER(current_user())
WHERE s.user_id IN (
```

```
  SELECT user_id
  FROM migrate_to_dbsql.webinar.user_regions
  WHERE region = ur.region
);

GRANT SELECT ON VIEW
migrate_to_dbsql.webinar.iot_daily_summary_abac TO
`analysts`;
```

(Optional) 📊Business Intelligence

STEP 5: Create New Dashboard

- Click on the Dashboards tab, and then click on New Dashboard.
- Name your new dashboard “IOT analysis”
- All visualizations we will create going forward will be added to this dashboard

STEP 6: Click + to Add Visualization

- Click + and then select Visualization
- Change the Visualization Name to “Steps by device per day”
- The default visualization type is Bar. Change the visualization type to Pie.
- Ensure X column and Y column are selected appropriately
- Click Save
- Add it to the IOT Dashboard

(Optional) ❓ Parameterized Queries

Now, let's explore more features in the query editor that will allow us to create dashboards that our users can interact with. This dataset represents sales over a period of time. So far, we have created queries and visualizations that capture metrics that include all of the data. Now, we'll create a query and visualization that will allow the viewer to drill down to some relevant details. For this, we will write a parameterized query. Parameterized

queries allow the viewer to substitute values into a query at runtime without having to edit the query source.
The next few steps will walk through an example of how you might use parameterized queries in practice.

STEP 1:
First, let's write a query to figure out the number of months represented in the data.

- Create a new query in the SQL Editor
- Execute and save this query.

```
[TODO] - add example query
```

STEP 2:

- Create a new query in the SQL Editor
- Copy and paste the code below into the editor
- Execute the query

```
[TODO] - add example query
```

STEP 3:
The results of this query show a daily sales total for each day in the data. If we wanted to view a single month, we could simply add a WHERE clause and identify the month we want to view.

- Copy and paste this **above** the GROUP BY clause.
- Execute and save the query.

```
[TODO] - add example query for parameterized query
```

🤖AI Functions

Databricks AI Functions are built-in SQL functions that allow users to leverage large language models (LLMs) directly within SQL queries. These functions are designed to simplify the process of deriving insights from unstructured data, making it easier for

analysts to interact with LLMs using SQL. The AI Functions include capabilities such as sentiment analysis, classification, information extraction, text generation etc.
Some examples of AI functions available are ai_analyze_sentiment( ), ai_classify( ), ai_fix_grammar( ) etc.

STEP 1: Forecasting

Let's create a sample dataset on which we will do forecast analysis. In a notebook or SQL editor run the below queries

This table should already be created as part of the workshop.
Copy the below query to create a forecast on the pumpkin demand using the ai_forecast( ) function

```
-- Overall daily steps series
WITH series AS (
  SELECT
    DATE(date) AS ds,
    SUM(total_steps) AS y
  FROM my_catalog.webinar.iot_daily_summary
  GROUP BY DATE(date)
)
-- Forecast to a fixed horizon date used in the workshop
examples
SELECT *
FROM AI_FORECAST(
  TABLE(series),
  horizon   => '2016-08-01',       -- pick a date ~2 weeks
beyond your max(ds)
  time_col  => 'ds',
  value_col => 'y'
);
```

STEP 3: ai_query()

```
-- calculate total steps per day based on alerts data
WITH steps_per_alert AS (
 SELECT
```

```
   device_id,
   date_format(created_at, 'yyyy-MM-dd') AS created_at_date,
   ai_query(
     'databricks-gpt-5-mini',
     'Extract number of steps from this text. Only output the number: ' ||
alert_message
   ) AS num_steps
 FROM
   migrate_to_dbsql.webinar.iot_alerts
)
SELECT
 device_id,
 created_at_date,
 SUM(num_steps) AS total_steps_per_day
FROM
 steps_per_alert
GROUP BY
 device_id,
 created_at_date;
```

STEP 4: ai_parse_document()

[TODO]: Provide an example query based on PDF files

Genie Spaces

Genie spaces enable self-service data analytics by allowing users to interact with data using natural language.

Step: Create a Genie Space

- In the left navigation bar, select Genie and click “New” on the top right
- Name the Genie Space [Your Name]-IOT-Data
- Add all the datasets from this workshop

- Add these sample questions "**What is the average total_steps per user per day**" and "**Are there any users with zero steps or calories daily?**"
- Create the genie space
- Ask Questions such as "Explain the dataset", "What were the total steps each day", "Which device_id has the highest number of steps in the last quarter?" etc.
- Click on "Auto Visualize" to get visualizations. You can also see the query that was generated and make changes to it.

When publishing a dashboard, you get an option to enable "Genie Spaces". This gives the users the ability to ask more questions to the underlying dataset, beyond the visualizations in the dashboard.

(Optional) 🪟Views, Temporary Views, Materialized Views, CTEs, Copy

Type	Definition	Use Case	Persistence	Sample SQL Query
Views	Read-only object composed from one or more tables or views, storing only the SQL definition.	Encapsulate complex SQL logic for reuse	Permanent and stored in the database schema.	CREATE OR REPLACE VIEW experienced_employee AS SELECT id, name FROM all_employee WHERE working_years > 5;
Temporary Views	View with session scope, existing only in the session where it's created.	Perform ad-hoc analysis within session without persistence.	Ephemeral, lost once the session ends.	CREATE TEMPORARY VIEW subscribed_movies AS SELECT mo.member_id, mb.full_name, mo.movie_title FROM movies AS mo INNER JOIN members AS mb ON mo.member_id = mb.id;
Materialized Views	Stores results physically as a table, updated based on data changes.	Improve performance for complex, frequently run queries on static data.	Persists data, enhancing performance by storing precomputed results.	CREATE MATERIALIZED VIEW sales_summary_mv AS SELECT customer_id, SUM(order_amount) AS total_sales FROM sales_table GROUP BY customer_id;

CTEs	Temporary result set within a SQL statement, existing in the statement.	Simplify complex queries; enhance readability and maintainability.	Transient, existing only during the single query execution.	WITH sales_summary AS (SELECT customer_id, SUM(order_amount) AS total_sales FROM sales_table GROUP BY customer_id) SELECT customer_id, total_sales FROM sales_summary;
Copy	Command to duplicate data from one table to another, creating a new table.	Create a persistent duplicate of a dataset for backup/snapshot purposes.	Creates a physical table in the database.	sql COPY INTO target_table FROM (SELECT * FROM source_table) FILEFORMAT = PARQUET;

(Optional) 📈Power BI, Excel plugin

Databricks connects seamlessly with PowerBI online and PowerBI Desktop, where you can publish your data sets directly to PowerBI from the databricks UI

Excel can access data stored with Databricks through the ODBC connector for excel provided.

(Optional) ⏱️Schedule Queries and Set up Alerts

Queries can be scheduled to run on a schedule and alerts can be created which are fired when a certain metric crosses the threshold defined by you. Explore scheduling query refreshes and alerts in the SQL editor

Data Migration Enablement with Databricks 🧱

Migration methodology

- **Self-implemented migrations**: Use Lakebridge for free/open migration assistance; pair with migration guides and a hands-on workshop with the account team.
- **Turn-key migrations:** Align Databricks account team with a migration partner; explore Databricks accelerate and cloud funding early to de-risk cost and timelines.

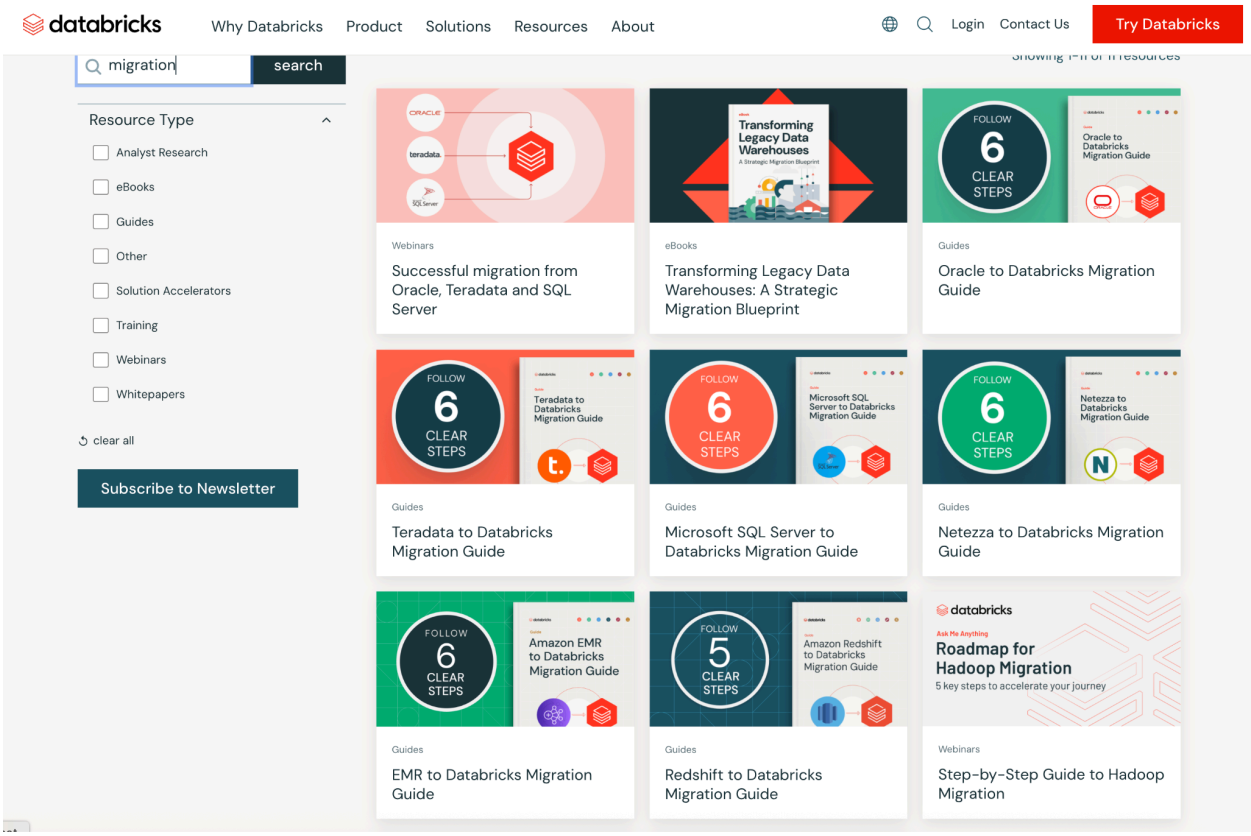
Why migrate now? What are the various methods of migration? How can I migrate from legacy warehouse to Databricks?

Key pillars: performance, governance, AI-readiness, cost

- Patterns:
  - Lift-and-improve (SQL compat, semantic equivalence tests)
  - Incremental domain-by-domain
  - Side-by-side validation with data quality contracts
- Tooling and collateral:
  - Lakebridge overview and demo path
  - Migration guides gallery and playbooks (explore industry based data migration pathways).
- Engagement models:
  - Self-implement with Databricks STS
  - Partner-led turn-key
  - Co-delivery for the first workload (funding options permitting)

Self implementation migration services:

1. STS DW migrator services: Shared Technical Services (go/sts)
2. Migration guides: https://www.databricks.com/resources - search for migration



3. Work with Databricks account team to conduct a migration consultation workshop

Funded migrations via Databricks partners or Cloud partners or Databricks Professional services:

1. A certified partner leads delivery using Brickbuilder Migration Solutions and Lakebridge for assessment, code conversion, and validation, with Databricks Professional Services (PS) providing migration assurance and design oversight as needed.
2. The Databricks Professional Services Migration Team can run structured assessments and offer full-service migration or assurance services if you prefer Databricks-led execution.
3. Funding is typically a blend of Databricks programs (DCIF in-deal or Accelerate out-of-deal) and cloud provider programs (AWS, Microsoft, GCP), coordinated by the Databricks account team with partners.