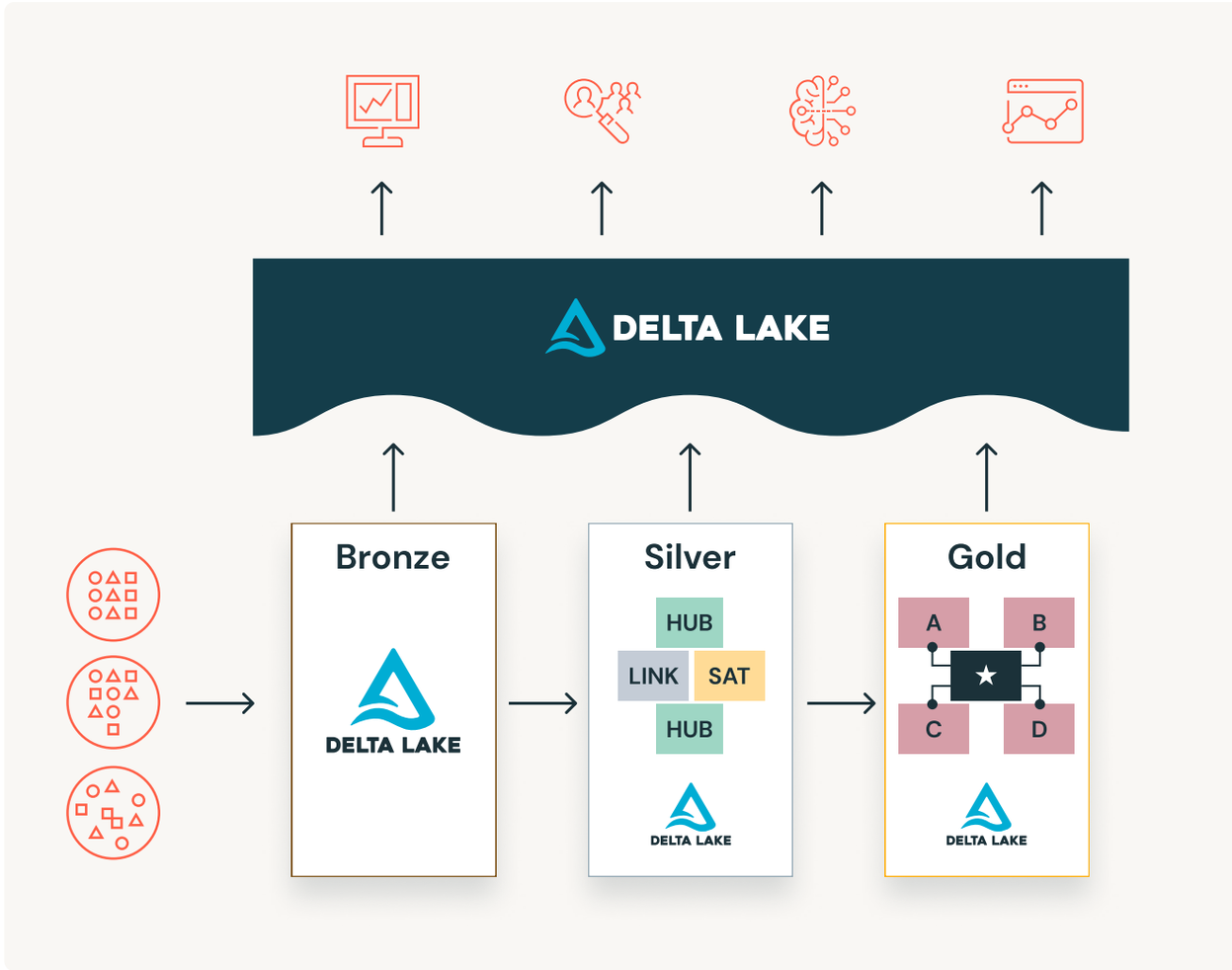
The Lakehouse Data Organization Paradigm

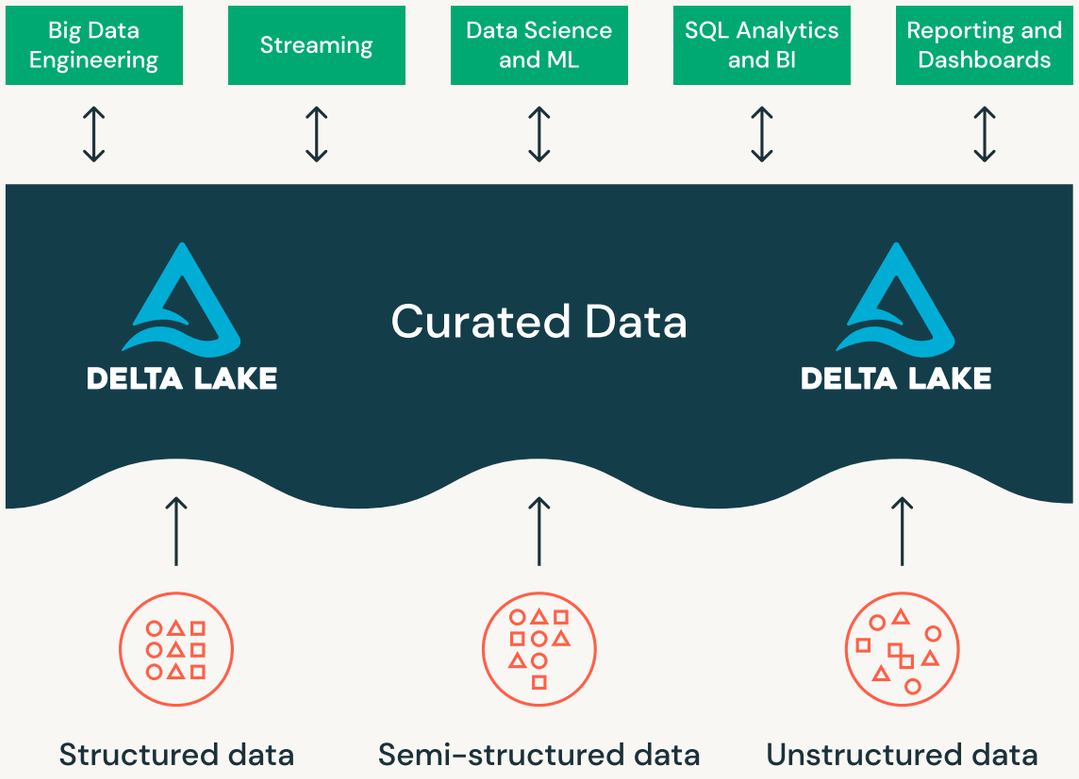
To summarize, data is curated as it moves through the different layers of a data lakehouse architecture.

- The Bronze layer uses the data models of source systems. If data is landed in raw formats, it is converted to DeltaLake format within this layer.
- The Silver layer for the first time brings the data from different sources together and conforms it to create an Enterprise view of the data — typically using a more normalized, write-optimized data models that are typically 3rd-Normal Form-like or Data Vault-like.
- The Gold layer is the presentation layer with more denormalized or flattened data models than the Silver layer, typically using Kimball-style dimensional models or star schemas. The Gold layer also houses departmental and data science sandboxes to enable self-service analytics and data science across the enterprise. Providing these sandboxes and their own separate compute clusters prevents the Business teams from creating their own copies of data outside of the Lakehouse.



This lakehouse data organization approach is meant to break data silos, bring teams together, and empower them to do ETL, streaming, and BI and AI on one platform with proper governance. Central data teams should be the enablers of innovation in the organization, speeding up the onboarding of new self-service users, as well as the development of many data projects in parallel — rather than the data modeling process becoming the bottleneck. The Databricks Unity Catalog provides search and discovery, governance and lineage on the lakehouse to ensure good data governance cadence.

Build your Data Vaults and star schema data warehouses with Databricks SQL today. →



How data is curated as it moves through the various lakehouse architecture layers.

LEARN MORE

- → Five Simple Steps for Implementing a Star Schema in Databricks With Delta Lake
- → Best practices to implement a Data Vault model in Databricks Lakehouse
- → Dimensional Modeling Best Practice & Implementation on Modern Lakehouse
- → Identity Columns to Generate Surrogate Keys Are Now Available in a Lakehouse Near You!
- → Load an EDW Dimensional Model in Real Time With Databricks Lakehouse

SECTION 4.2

Dimensional Modeling Best Practice and Implementation on a Modern Lakehouse Architecture

by Leo Mao, Abhishek Dey, Justin Breese and Soham Bhatt

A large number of our customers are migrating their legacy data warehouses to Databricks Lakehouse as it enables them to modernize not only their Data Warehouse but they also instantly get access to a mature Streaming and Advanced Analytics platform. Lakehouse can do it all as it is one platform for all your streaming, ETL, BI, and AI needs — and it helps your business and Data teams collaborate on one platform.

As we help customers in the field, we find that many are looking for best practices around proper data modeling and physical data model implementations in Databricks.

In this article, we aim to dive deeper into the best practice of dimensional modeling on the Databricks Data Intelligence Platform and provide a live example of a physical data model implementation using our table creation and DDL best practices.

Here are the high-level topics we will cover in this blog:

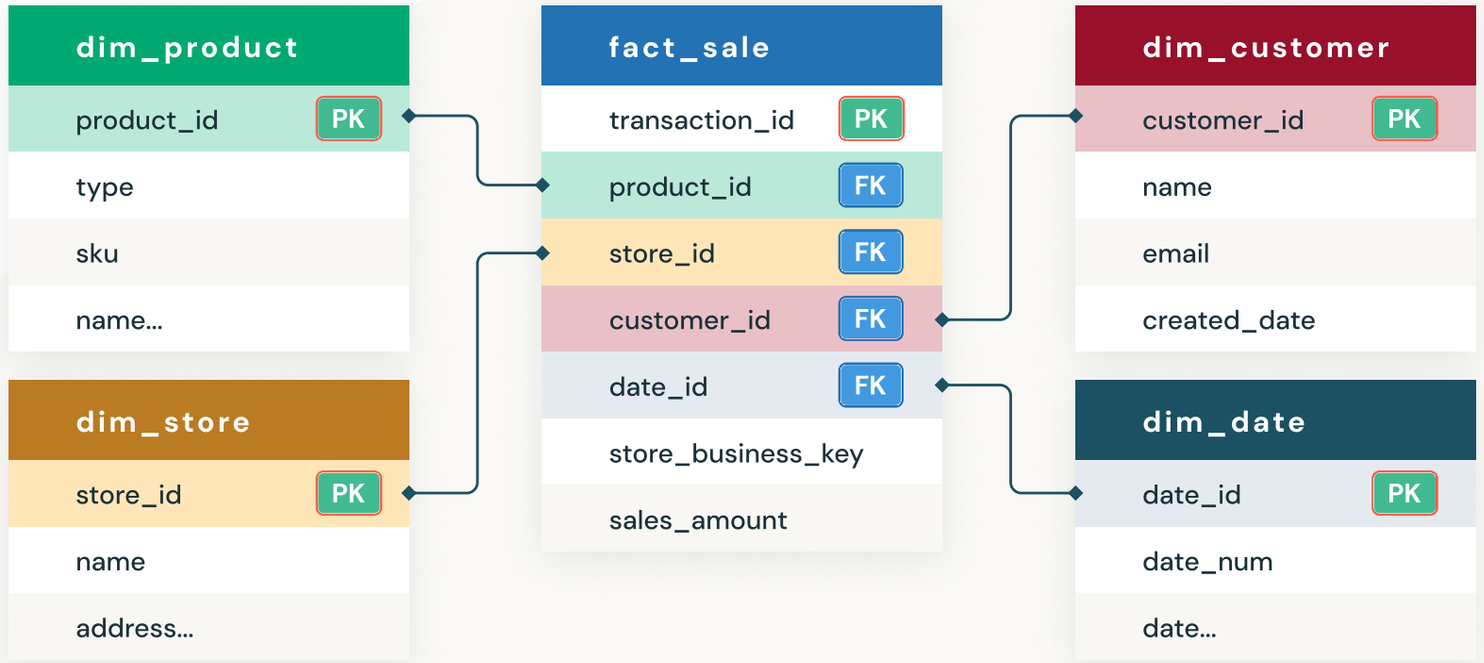
- The Importance of Data Modeling
- Common Data Modeling Techniques
- Data Warehouse Modeling DDL Implementation
- Best Practice and Recommendation for Data Modeling on the Lakehouse

The importance of Data Modeling for Data Warehouse

Data Models are front and center of building a Data Warehouse. Typically the process starts with defending the Semantic Business Information Model, then a Logical data Model, and finally a Physical Data Model (PDM). It all starts with a proper Systems Analysis and Design phase where a Business Information model and process flows are created first and key business entities, attributes and their interactions are captured as per the business processes within the organization. The Logical Data Model is then created depicting how the entities are related to each other and this is a Technology agnostic model. Finally a PDM is created based on the underlying technology platform to ensure that the writes and reads can be performed efficiently. As we all know, for Data Warehousing, Analytics-friendly modeling styles like Star-schema and Data Vault are quite popular.

Best practices for creating a Physical Data Model in Databricks

Based on the defined business problem, the aim of the data model design is to represent the data in an easy way for reusability, flexibility, and scalability. Here is a typical star-schema data model that shows a Sales fact table that holds each transaction and various dimension tables such as customers, products, stores, date etc. by which you slice-and-dice the data. The dimensions can be joined to the fact table to answer specific business questions such as what are the most popular products for a given month or which stores are best performing ones for the quarter. Let's see how to get it implemented in Databricks.



Dimensional model on the lakehouse architecture

Note each dimension table has **__START_AT** and **__END_AT** columns to support SCD Type 2, which are not displayed here because of limited space.

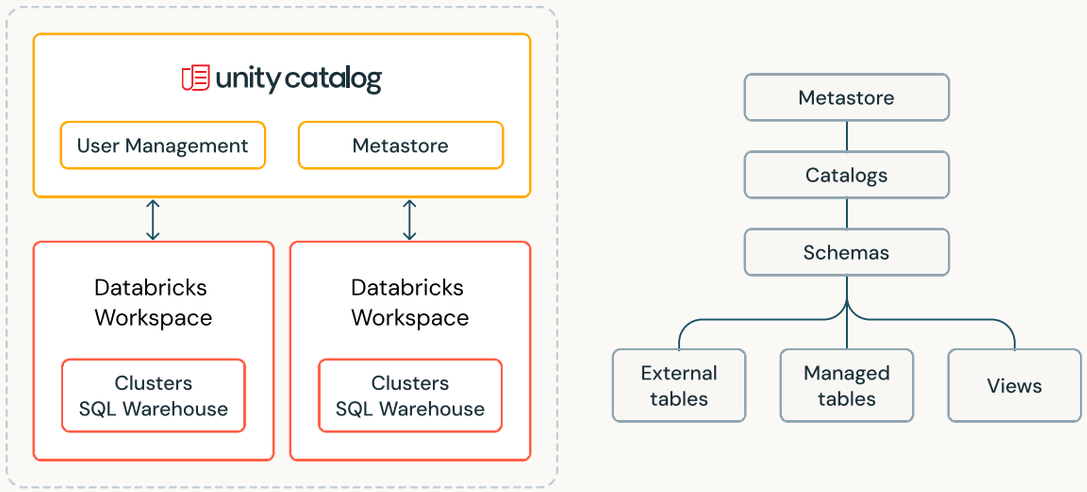
Data Warehouse Modeling DDL Implementation on Databricks

In the following sections, we would demonstrate the below using our examples.

- Creating 3-level Catalog, Database and Table
- Primary Key, Foreign Key definitions
- Identity columns for Surrogate keys
- Column constraints for Data Quality
- Index, optimize and analyze
- Advanced techniques

1. Unity Catalog - 3 level namespace

Unity Catalog is a Databricks Governance layer which lets Databricks admins and data stewards manage users and their access to data centrally across all of the workspaces in a Databricks account using one Metastore. Users in different workspaces can share access to the same data, depending on privileges granted centrally in Unity Catalog. Unity Catalog has 3 level Namespace (catalog.schema(database).table) that organizes your data. Learn more about Unity Catalog here.



Here is how to set up the catalog and schema before we create tables within the database. For our example, we create a catalog **US_Stores** and a schema (database) **Sales_DW** as below, and use them for the later part of the section.

```
CREATE CATALOG IF NOT EXISTS US_Stores;
USE CATALOG US_Stores;
CREATE SCHEMA IF NOT EXISTS Sales_DW;
USE SCHEMA Sales_DW;
```

Setting up the Catalog and Database

Here is an example on querying the **fact_sales** table with a 3 level namespace.

```
SELECT *
FROM US_Stores.Sales_DW.fact_sales
```

▸ (2) Spark Jobs

Table | Data Profile

	transaction_id	date_id	customer_id	product_id	store_id	store_business_key	sales_amount
1	10001	20211001	1	1	1	PER01	50
2	10004	20211003	2	1	2	BNE02	60
3	10005	20211003	3	2	1	PER01	79
4	10002	20211002	2	1	2	BNE02	79
5	10003	20211002	1	2	2	BNE02	79

Showing all 5 rows.

Example of querying table with catalog.database.tablename

2. Primary Key, Foreign Key definitions

Primary and Foreign Key definitions are very important when creating a data model. Having the ability to support the PK/FK definition makes defining the data model super easy in Databricks. It also helps analysts quickly figure out the join relationships in Databricks SQL Warehouse so that they can effectively write queries. Like most other Massively Parallel Processing (MPP), EDW, and Cloud Data Warehouses, the PK/FK constraints are informational only. Databricks does not support enforcement of the PK/FK relationship, but gives the ability to define it to make the designing of Semantic Data Model easy.

Here is an example of creating the **dim_store** table with **store_id** as an Identity Column and it's also defined as a Primary Key at the same time.

```
-- Store dimension
CREATE OR REPLACE TABLE dim_store(
  store_id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  business_key STRING,
  name STRING,
  email STRING,
  city STRING,
  address STRING,
  phone_number STRING,
  created_date TIMESTAMP,
  updated_date TIMESTAMP,
  start_at TIMESTAMP,
  end_at TIMESTAMP
);
```

DDL Implementation for creating store dimension with Primary Key Definitions

After the table is created, we can see that the primary key (store_id) is created as a constraint in the table definition below.

Describe Dim Store table information

```
DESC TABLE EXTENDED US_Stores.Sales_DW.dim_store
```

Table | Data Profile

	col_name	data_type
20	Owner	leo.mao@databricks.com
21	Is_managed_location	true
22	Table Properties	[delta.minReaderVersion=1,delta.minWriterVersion=6]
23		
24	# Constraints	
25	dim_store_pk	PRIMARY KEY (`store_id`)

Showing all 25 rows.

Primary Key store_id shows as table constraint

Here is an example of creating the **fact_sales** table with **transaction_id** as a Primary Key, as well as foreigh keys that are referencing the dimension tables.

```
-- Fact Sales
CREATE OR REPLACE TABLE fact_sales(
  transaction_id BIGINT PRIMARY KEY,
  date_id BIGINT NOT NULL CONSTRAINT dim_date_fk FOREIGN KEY REFERENCES dim_date,
  customer_id BIGINT NOT NULL CONSTRAINT dim_customer_fk FOREIGN KEY REFERENCES
dim_customer,
  product_id BIGINT NOT NULL CONSTRAINT dim_product_fk FOREIGN KEY REFERENCES
dim_product,
  store_id BIGINT NOT NULL CONSTRAINT dim_store_fk FOREIGN KEY REFERENCES dim_
store,
  store_business_key STRING,
  sales_amount DOUBLE
);
```

DDL Implementation for creating sales fact with Foreign Key definitions

After the fact table is created, we could see that the primary key **(transaction_id)** and foreign keys are created as constraints in the table definition below.

Describe Fact Sales Table Info

```
DESC TABLE EXTENDED US_Stores.Sales_DW.fact_sales
```

Table | Data Profile

	col_name	data_type
20	# Constraints	
21	dim_product_fk	FOREIGN KEY (`product_id`) REFERENCES `us_stores`.`sales_dw`.`dim_product` (`product_id`)
22	dim_date_fk	FOREIGN KEY (`date_id`) REFERENCES `us_stores`.`sales_dw`.`dim_date` (`date_id`)
23	dim_customer_fk	FOREIGN KEY (`customer_id`) REFERENCES `us_stores`.`sales_dw`.`dim_customer` (`customer_id`)
24	dim_store_fk	FOREIGN KEY (`store_id`) REFERENCES `us_stores`.`sales_dw`.`dim_store` (`store_id`)
25	fact_sales_pk	PRIMARY KEY (`transaction_id`)

Showing all 25 rows.

Fact table definition with primary key and foreign keys referencing dimensions

3. Identity columns for Surrogate keys

An identity column is a column in a database that automatically generates a unique ID number for each new row of data. These are commonly used to create surrogate keys in the data warehouses. Surrogate keys are system-generated, meaningless keys so that we don't have to rely on various Natural Primary Keys and concatenations on several fields to identify the uniqueness of the row. Typically these surrogate keys are used as Primary and Foreign keys in data warehouses. Details on Identity columns are discussed in this blog. Below is an example of creating an identity column customer_id, with automatically assigned values starting with 1 and increment by 1.

```
-- Customer dimension
CREATE OR REPLACE TABLE dim_customer(
  customer_id BIGINT GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1)
PRIMARY KEY,
  name STRING,
  email STRING,
  address STRING,
  created_date TIMESTAMP,
  updated_date TIMESTAMP,
  start_at TIMESTAMP,
  end_at TIMESTAMP
);
```

DDL Implementation for creating customer dimension with identity column

4. Column constraints for Data Quality

In addition to Primary and Foreign key informational constraints, Databricks also supports column-level Data Quality Check constraints which are enforced to ensure the quality and integrity of data added to a table. The constraints are automatically verified. Good examples of these are NOT NULL constraints and column value constraints. Unlike the other Cloud Data Warehouse, Databricks went further to provide column value check constraints, which are very useful to ensure the data quality of a given column. As we could see below, the **valid_sales_amount** check constraint will verify that all existing rows satisfy the constraint (i.e. **sales amount** > 0) before adding it to the table. More information can be found here.

Here are examples to add constraints for dim_store and fact_sales respectively to make sure store_id and sales_amount have valid values.

```
-- Add constraint to dim_store to make sure column store_id is between 1 and 9998
ALTER TABLE US_Stores.Sales_DW.dim_store ADD CONSTRAINT valid_store_id CHECK
(store_id > 0 and store_id < 9999);

-- Add constraint to fact_sales to make sure column sales_amount has a valid value
ALTER TABLE US_Stores.Sales_DW.fact_sales ADD CONSTRAINT valid_sales_amount CHECK
(sales_amount > 0);
```

Add column constraint to existing tables to ensure data quality

5. Index, Optimize, and Analyze

Traditional Databases have b-tree and bitmap indexes, Databricks has much advanced form of indexing - multi-dimensional Z-order clustered indexing and we also support Bloom filter indexing. First of all, the Delta file format uses Parquet file format, which is a columnar compressed file format so it's already very efficient in column pruning and on top of it using z-order indexing gives you the ability to sift through petabyte scale data in seconds. Both Z-order and Bloom filter indexing dramatically reduce the amount of data that needs to be scanned in order to answer highly selective queries against large Delta tables, which typically translates into orders-of-magnitude runtime improvements and cost savings. Use Z-order on your Primary Keys and foreign keys that are used for the most frequent joins. And use additional Bloom filter indexing as needed.

```
-- Optimise fact_sales table by customer_id and product_id for better query and
join performance
OPTIMIZE US_Stores.Sales_DW.fact_sales
ZORDER BY (customer_id, product_id);
```

Optimize fact_sales on customer_id and product_id for better performance

```
-- Create a bloomfilter index to enable data skipping on store_business_key
CREATE BLOOMFILTER INDEX
ON TABLE US_Stores.Sales_DW.fact_sales
FOR COLUMNS(store_business_key)
```

Create a Bloomfilter Index to enable data skipping on a given column

And just like any other Data warehouse, you can ANALYZE TABLE to update statistics to ensure the Query optimizer has the best statistics to create the best query plan.

```
-- collect stats for all columns for better performance
ANALYZE TABLE US_Stores.Sales_DW.fact_sales COMPUTE STATISTICS FOR ALL COLUMNS;
```

Collect stats for all the columns for better query execution plan

6. Advanced Techniques

While Databricks support advanced techniques like Table Partitioning, please use these feature sparingly, only when you have many Terabytes of compressed data - because most of the time our OPTIMIZE and Z-ORDER indexes will give you the best file and data pruning which makes partitioning a table by date or month almost a bad practice. It is however a good practice to make sure that your table DDLs are set for auto optimization and auto compaction. These will ensure your frequently written data in small files are compacted into bigger columnar compressed formats of Delta.

Are you looking to leverage a visual data modeling tool? Our partner erwin Data Modeler by Quest can be used to reverse engineer, create and implement Star-schema, Data Vaults, and any Industry Data Models in Databricks with just a few clicks.

Databricks Notebook Example

With the Databricks Platform, one can easily design and implement various data models with ease. To see all of the above examples in a complete workflow, please look at this example.

Please also check out our related blog:

Five Simple Steps for Implementing a Star Schema in Databricks With Delta Lake →

SECTION 4.3

Loading a Data Warehouse Data Model in Real Time With the Databricks Data Intelligence Platform

Dimensional modeling implementation on the modern lakehouse using Delta Live Tables

by Leo Mao, Soham Bhatt and Abhishek Dey

Dimensional modeling is one of the most popular data modeling techniques for building a modern data warehouse. It allows customers to quickly develop facts and dimensions based on business needs for an enterprise. When helping customers in the field, we found many are looking for best practices and implementation reference architecture from Databricks.

In this article, we aim to dive deeper into the best practice of dimensional modeling on the lakehouse architecture and provide a live example to load an EDW dimensional model in real-time using Delta Live Tables.

Here are the high-level steps we will cover in this blog:

- Define a business problem
- Design a dimensional model
- Best practices and recommendations for dimensional modeling
- Implementing a dimensional model on a lakehouse architecture
- Conclusion

Define a business problem

Dimensional modeling is business-oriented; it always starts with a business problem. Before building a dimensional model, we need to understand the business problem to solve, as it indicates how the data asset will be presented and consumed by end users. We need to design the data model to support more accessible and faster queries.

The Business Matrix is a fundamental concept in Dimensional Modeling, below is an example of the business matrix, where the columns are shared dimensions and rows represent business processes. The defined business problem determines the grain of the fact data and required dimensions. The key idea here is that we could incrementally build additional data assets with ease based on the Business Matrix and its shared or conformed dimensions.

Shared Dimensions →

Business Processes ↓	Date	Customer	Product	Vendor	Promotion	Reseller	Sales Territory	Employee	Account	Organization
Internet Sales	✓	✓	✓	✓	✓		✓			
Reseller Sales	✓		✓		✓	✓	✓	✓		
General Ledger	✓								✓	✓
Sales Plan	✓		✓				✓			
Inventory	✓		✓	✓					✓	
Customer Surveys	✓	✓								
Customer Service Calls	✓	✓	✓					✓		

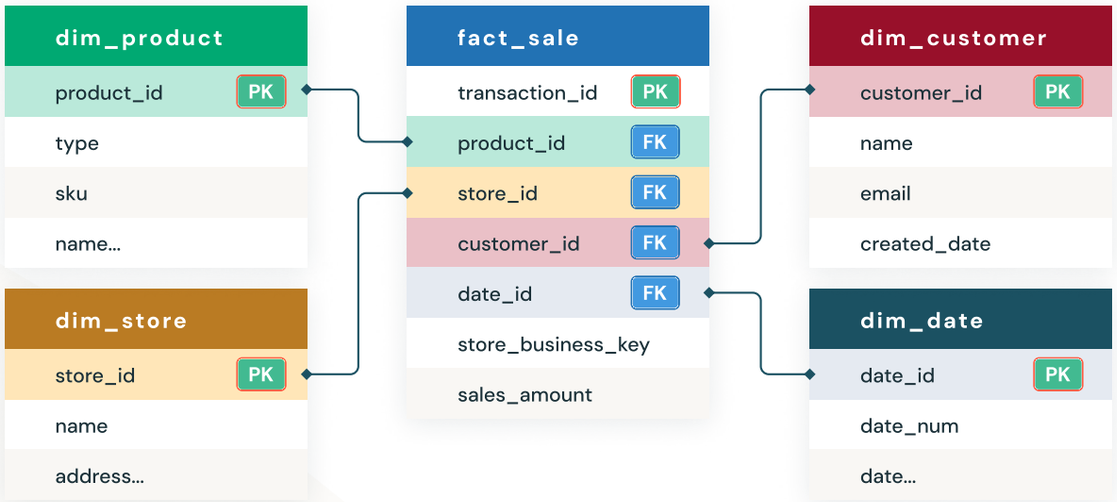
A Business Matrix with Shared Dimensions and Business Processes

Here we assume that the business sponsor would like to team to build a report to give insights on:

- What are the top selling products so they can understand product popularity
- What are the best performing stores to learn good store practices

Design a dimensional model

Based on the defined business problem, the data model design aims to represent the data efficiently for reusability, flexibility and scalability. Here is the high-level data model that could solve the business questions above.



Dimensional model on the lakehouse architecture

Note each dimension table has **__START_AT** and **__END_AT** columns to support SCD Type 2, which are not displayed here because of limited space.

The design should be easy to understand and efficient with different query patterns on the data. From the model, we designed the sales fact table to answer our business questions; as you can see, other than the foreign keys (FKs) to the dimensions, it only contains the numeric metrics used to measure the business, e.g. sales_amount.

We also designed dimension tables such as Product, Store, Customer, Date that provide contextual information on the fact data. Dimension tables are typically joined with fact tables to answer specific business questions, such as the most popular products for a given month, which stores are the best-performing ones for the quarter, etc.

Best practices and recommendations for dimensional modeling

With the Databricks Data Intelligence Platform, one can easily design and implement dimensional models, and simply build the facts and dimensions for the given subject area.

Below are some of the best practices recommended while implementing a dimensional model:

- One should denormalize the dimension tables. Instead of the third normal form or snowflake type of model, dimension tables typically are highly denormalized with flattened many-to-one relationships within a single dimension table.
- Use conformed dimension tables when attributes in different dimension tables have the same column names and domain contents. This advantage is that data from different fact tables can be combined in a single report using conformed dimension attributes associated with each fact table.
- A usual trend in dimension tables is around tracking changes to dimensions over time to support as-is or as-was reporting. You can easily apply the following basic techniques for handling dimensions based on different requirements.
  - The type 1 technique overwrites the dimension attribute's initial value.
  - With the type 2 technique, the most common SCD technique, you use it for accurate change tracking over time.

  This can be easily achieved out of the box with Delta Live Tables implementation.
  - One can easily perform SCD type 1 or SCD type 2 using Delta Live Tables using APPLY CHANGES INTO
- Primary + Foreign Key **Constraints** allow end users like yourselves to understand relationships between tables.
- **Usage of IDENTITY Columns** automatically generates unique integer values when new rows are added. Identity columns are a form of surrogate keys. Refer to the blog link for more details.
- **Enforced CHECK Constraints** to never worry about data quality or data correctness issues sneaking up on you.

Implementing a dimensional model on a lakehouse architecture

Now, let us look at an example of Delta Live Tables based dimensional modeling implementation:

The example code below shows us how to create a dimension table (dim_store) using SCD Type 2, where change data is captured from the source system.

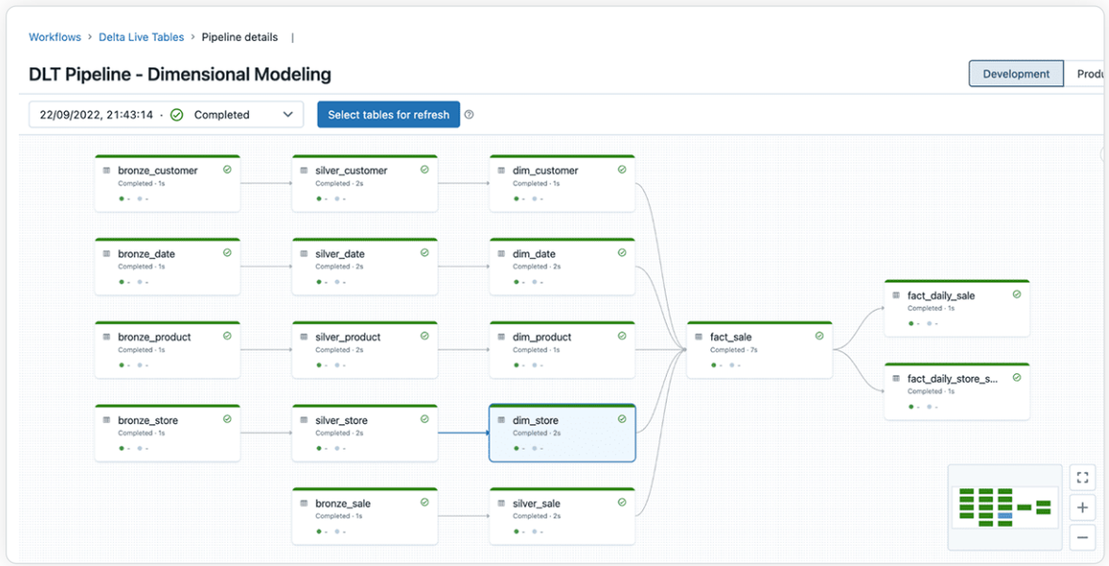
```
-- create the gold table
CREATE INCREMENTAL LIVE TABLE dim_store
TBLPROPERTIES (“quality” = “gold”)
COMMENT “Slowly Changing Dimension Type 2 for store dimension in the gold layer”;

-- store all changes as SCD2
APPLY CHANGES INTO live.dim_store
FROM STREAM(live.silver_store)
  KEYS (store_id)
  SEQUENCE BY updated_date
  COLUMNS * EXCEPT (_rescued_data, input_file_name)
  STORED AS SCD TYPE 2;
```

The example code below shows us how to create a fact table (fact_sale), with the constraint of valid_product_id we are able to ensure all fact records that are loaded have a valid product associated with it.

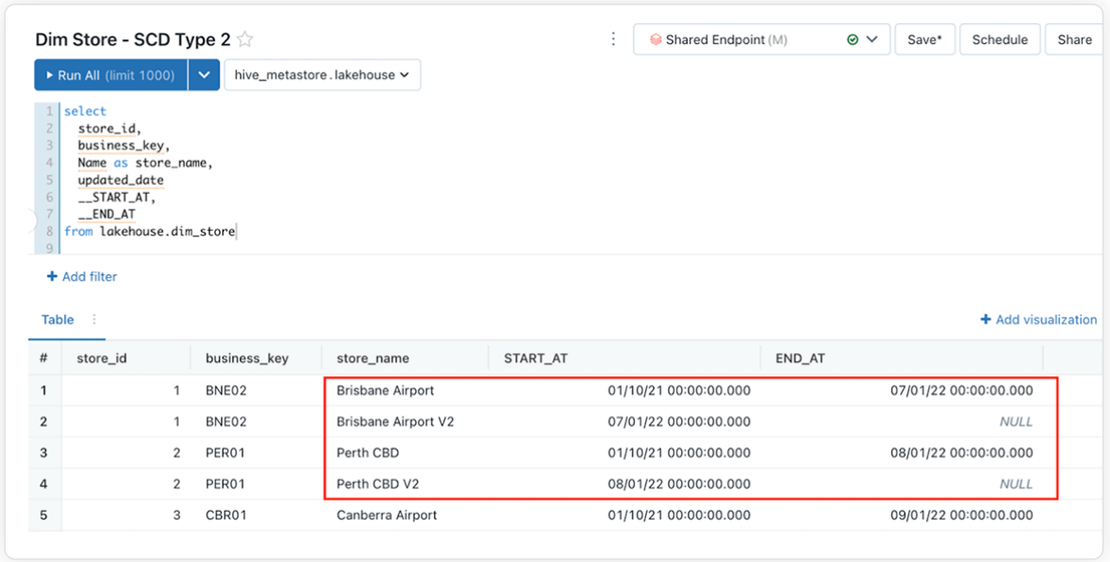
```
-- create the fact table for sales in gold layer
CREATE STREAMING LIVE TABLE fact_sale (
  CONSTRAINT valid_store_business_key EXPECT (store_business_key IS NOT NULL) ON
VIOLATION DROP ROW,
  CONSTRAINT valid_product_id EXPECT (product_id IS NOT NULL) ON VIOLATION DROP
ROW
)
TBLPROPERTIES (“quality” = “gold”, “ignoreChanges” = “true”)
COMMENT “sales fact table in the gold layer” AS
  SELECT
    sale.transaction_id,
    date.date_id,
    customer.customer_id,
    product.product_id AS product_id,
    store.store_id,
    store.business_key AS store_business_key,
    sales_amount
  FROM STREAM(live.silver_sale) sale
  INNER JOIN live.dim_date date
  ON to_date(sale.transaction_date, ‘M/d/yy’) = to_date(date.date, ‘M/d/yyyy’)
  -- only join with the active customers
  INNER JOIN (SELECT * FROM live.dim_customer WHERE __END_AT IS NULL) customer
  ON sale.customer_id = customer.customer_id
  -- only join with the active products
  INNER JOIN (SELECT * FROM live.dim_product WHERE __END_AT IS NULL) product
  ON sale.product = product.SKU
  -- only join with the active stores
  INNER JOIN (SELECT * FROM live.dim_store WHERE __END_AT IS NULL) store
  ON sale.store = store.business_key
```

The Delta Live Tables pipeline example could be found here. Please refer to Delta Live Tables quickstart on how to create a Delta Live Tables pipeline. As seen below, DLT offers full visibility of the ETL pipeline and dependencies between different objects across Bronze, Silver and Gold layers following the lakehouse medallion architecture.



End to End DLT Pipeline

Here is an example of how the dimension table dim_store gets updated based on the incoming changes. Below, the Store Brisbane Airport was updated to Brisbane Airport V2, and with the out-of-box SCD Type 2 support, the original record ended on Jan 07 2022, and a new record was created which starts on the same day with an open end date (NULL) - which indicates the latest record for the Brisbane airport.



#	store_id	business_key	store_name	START_AT	END_AT
1	1	BNE02	Brisbane Airport	01/10/21 00:00:00.000	07/01/22 00:00:00.000
2	1	BNE02	Brisbane Airport V2	07/01/22 00:00:00.000	NULL
3	2	PER01	Perth CBD	01/10/21 00:00:00.000	08/01/22 00:00:00.000
4	2	PER01	Perth CBD V2	08/01/22 00:00:00.000	NULL
5	3	CBR01	Canberra Airport	01/10/21 00:00:00.000	09/01/22 00:00:00.000

SCD Type 2 for Store Dimension

For more implementation details, please refer to here for the full notebook example.

Conclusion

In this blog, we learned about dimensional modeling concepts in detail, best practices, and how to implement them using Delta Live Tables.

Learn more about dimensional modeling at Kimball Technology.

SECTION 4.4

What's New With Databricks SQL?

AI-driven optimizations, lakehouse federation, and more for enterprise-grade BI

by Alex Lichen, Miranda Luna, Can Efeoglu and Cyrielle Simeone

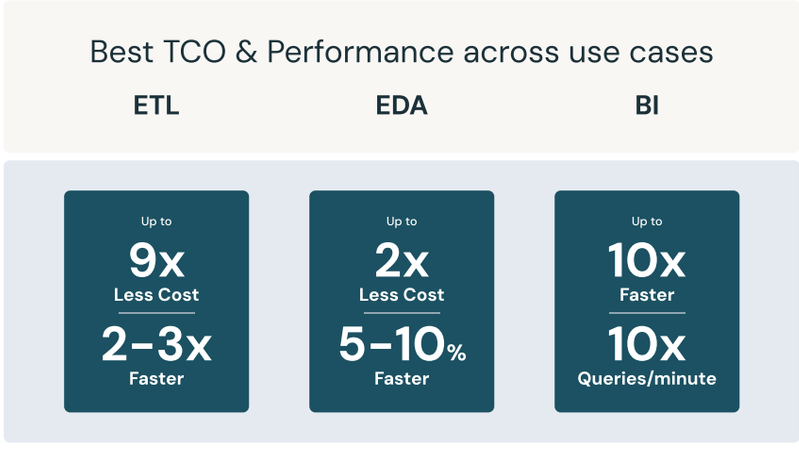
At this year's Data + AI Summit, Databricks SQL continued to push the boundaries of what a data warehouse can be, leveraging AI across the entire product surface to extend our leadership in performance and efficiency, while still simplifying the experience and unlocking new opportunities for our customers. In parallel, we continue to deliver improvements to our core data warehousing capabilities to help you unify your data stack under the lakehouse architecture on the Databricks Data Intelligence Platform.

In this blog post, we are thrilled to share the highlights of what's new and coming next in Databricks SQL:

- AI-driven performance optimizations like Predictive I/O that deliver leading performance and cost-savings, without manual tuning required.
- New user experiences with AI Functions, SQL Warehouses in Notebooks, new Dashboards, and Python User Defined Functions.
- Rich external data source support with Lakehouse Federation.
- New ways to access your data with the SQL Statement Execution API.
- Simple and efficient data processing with Streaming Tables, Materialized Views and Workflows integration.
- Intelligent assistance powered by DatabricksIQ, our knowledge engine.
- Enhanced administration tools with Databricks SQL System Tables and Live Query Profile.
- Additional features for our partner integrations with Fivetran, dbt labs and PowerBI

The AI-optimized warehouse: Ready for all your workloads — no tuning required

We believe that the best data warehouse is a lakehouse; therefore, we continue to extend our leadership in ETL workloads and harnessing the power of AI. Databricks SQL now also delivers industry-leading performance for your EDA and BI workloads, while improving cost savings — with no manual tuning.



Say goodbye to manually creating indexes. With Predictive I/O for reads (GA) and updates (Public Preview), Databricks SQL now analyzes historical read and write patterns to intelligently build indexes and optimize workloads. Early customers have benefited from a remarkable 35x improvement in point lookup efficiency, impressive performance boosts of 2-6x for MERGE operations and 2-10x for DELETE operations.

With Predictive Optimizations (Public Preview), Databricks will seamlessly optimize file sizes and clustering by running OPTIMIZE, VACUUM, ANALYZE and CLUSTERING commands for you. With this feature, Anker Innovations benefited from a 2.2x boost to query performance while delivering 50% savings on storage costs.

> **Databricks' Predictive Optimizations intelligently optimized our Unity Catalog storage, which saved us 50% in annual storage costs while speeding up our queries by >2x. It learned to prioritize our largest and most-accessed tables. And, it did all of this automatically, saving our team valuable time.**
>
> — ANKER INNOVATIONS

Tired of managing different warehouses for smaller and larger workloads or fine tuning scaling parameters? Intelligent Workload Management is a suite of features that keeps queries fast while keeping cost low. By analyzing real time patterns, Intelligent Workload Management ensures that your workloads have the optimal amount of compute to execute incoming SQL statements without disrupting already running queries.

With AI-powered optimizations, Databricks SQL provides industry leading TCO and performance for any kind of workload, without any manual tuning needed. To learn more about available optimization previews, watch Reynold Xin's keynote and Databricks SQL Serverless Under the Hood: How We Use ML to Get the Best Price/Performance from the Data+AI Summit.

Unlock siloed data with Lakehouse Federation

Today's organizations face challenges in discovering, governing and querying siloed data sources across fragmented systems. With Lakehouse Federation, data teams can use Databricks SQL to discover, query and manage data in external platforms including MySQL, PostgreSQL, Amazon Redshift, Snowflake, Azure SQL Database, Azure Synapse, Google's BigQuery (coming soon) and more.

Furthermore, Lakehouse Federation seamlessly integrates with advanced features of Unity Catalog when accessing external data sources from within Databricks. Enforce row and column level security to restrict access to sensitive information. Leverage data lineage to trace the origins of your data and ensure data quality and compliance. To organize and manage data assets, easily tag federated catalog assets for simple data discovery.

Finally, to accelerate complicated transformations or cross-joins on federated sources, Lakehouse Federation supports Materialized Views for better query latencies.

For more details, watch our dedicated session Lakehouse Federation: Access and Governance of External Data Sources from Unity Catalog from the Data+AI Summit.

Develop on the lakehouse architecture with the SQL Statement Execution API

The SQL Statement Execution API enables access to your Databricks SQL warehouse over a REST API to query and retrieve results. With HTTP frameworks available for almost all programming languages, you can easily connect to a diverse array of applications and platforms directly to a Databricks SQL Warehouse.

The Databricks SQL Statement Execution API is available with the Databricks Premium and Enterprise tiers. To learn more, watch our session, follow our tutorial (AWS | Azure), read the documentation (AWS | Azure), or check our repository of code samples.

Streamline your data processing with Streaming Tables, Materialized Views, and DB SQL in Workflows

With Streaming Tables, Materialized Views, and DB SQL in Workflows, any SQL user can now apply data engineering best practices to process data. Efficiently ingest, transform, orchestrate, and analyze data with just a few lines of SQL.

Streaming Tables are the ideal way to bring data into "Bronze" tables. With a single SQL statement, scalably ingest data from various sources such as cloud storage (S3, ADLS, GCS), message buses (EventHub, Kafka, Kinesis), and more. This ingestion occurs incrementally, enabling low-latency and cost-effective pipelines, without the need for managing complex infrastructure.

```
CREATE STREAMING TABLE web_clicks
AS
SELECT *
FROM STREAM
  read_files('s3://mybucket')
```

Materialized Views reduce cost and improve query latency by pre-computing slow queries and frequently used computations, and are incrementally refreshed to improve overall latency. In a data engineering context, they are used for transforming data. But they are also valuable for analyst teams in a data warehousing context because they can be used to (1) speed up end-user queries and BI dashboards, and (2) securely share data. In just four lines of code, any user can create a materialized view for performant data processing.

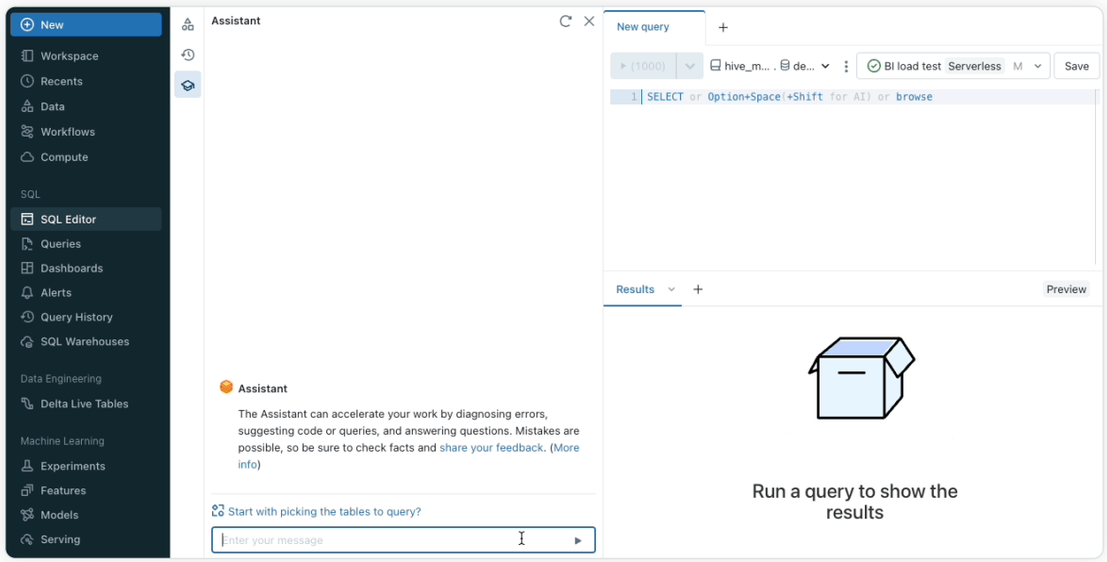
```
CREATE MATERIALIZED VIEW customer_orders
AS
SELECT
  customers.name,
  sum(orders.amount),
  orders.orderdate
FROM orders
  LEFT JOIN customers ON
    orders.custkey = customers.c_custkey
GROUP BY
  name,
  orderdate;
```

Need orchestration with DB SQL? Workflows now allows you to schedule SQL queries, dashboards and alerts. Easily manage complex dependencies between tasks and monitor past job executions with the intuitive Workflows UI or via API.

Streaming Tables and Materialized Views are now in public preview. To learn more, read our dedicated blog post. To enroll in the public preview for both, enroll in this form. Workflows in DB SQL is now generally available, and you can learn more by reading the documentation (AWS | Azure).

Databricks Assistant: Write better and faster SQL with natural language

Databricks Assistant is a context-aware AI assistant embedded inside Databricks Notebooks and the SQL Editor. Databricks Assistant can take a natural language question and suggest a SQL query to answer that question. When trying to understand a complex query, users can ask the Assistant to explain it using natural language, enabling anyone to understand the logic behind query results.



Databricks Assistant uses a number of signals to provide more accurate, relevant results. It uses context from code cells, libraries, popular tables, Unity Catalog schemas and tags to map natural language questions into queries and code.

In the future, we will be adding integration with DatabricksIQ to provide even more context for your requests.

Manage your data warehouse with confidence

Administrators and IT teams need the tools to understand data warehouse usage. With System Tables, Live Query Profile, and Statement Timeouts, admins can monitor and fix problems when they occur, ensuring that your data warehouse runs efficiently.

Gain deeper visibility and insights into your SQL environment with System Tables. System Tables are Databricks-provided tables that contain information about past statement executions, costs, lineage, and more. Explore metadata and usage metrics to answer questions like “What statements were run and by whom?”, “How and when did my warehouses scale?” and “What was I billed for?”. Since System Tables are integrated within Databricks, you have access to native capabilities such as SQL alerts and SQL dashboards to automate the monitoring and alerting process.

As of today, there are three System Tables currently in public preview: Audit Logs, Billable Usage System Table, and Lineage Sytem Table (AWS | Azure). Additional system tables for warehouse events and statement history are coming soon.

For example, to compute the monthly DBUs used per SKU, you can query the Billiable Usage System Tables.

```
SELECT sku_name, usage_date, sum(usage_quantity) as `DBUs`
    FROM system.billing.usage
WHERE
    month(usage_date) = month(NOW())
    AND year(usage_date) = year(NOW())
GROUP BY sku_name, usage_date
```

With Live Query Profile, users gain real-time insights into query performance to help optimize workloads on the fly. Visualize query execution plans and assess live query task executions to fix common SQL mistakes like exploding joins or full table scans. Live Query Profile allows you to ensure that running queries on your data warehouse are optimized and running efficiently. Learn more by reading the documentation (AWS | Azure).

Looking for automated controls? Statement Timeouts allow you to set a custom workspace or query level timeout. If a query's execution time exceeds the timeout threshold, the query will be automatically halted. Learn more by reading the documentation (AWS | Azure)

Compelling new experiences in DBSQL

Over the past year, we've been hard at work to add new, cutting-edge experiences to Databricks SQL. We're excited to announce new features that put the power of AI in SQL users hands such as, enabling SQL warehouses throughout the entire Databricks platform; introducing a new generation of SQL dashboards; and bringing the power of Python into Databricks SQL.

Democratize unstructured data analysis with AI Functions

With AI Functions, DB SQL is bringing the power of AI into the SQL warehouse. Effortlessly harness the potential of unstructured data by performing tasks such as sentiment analysis, text classification, summarization, translation and more. Data analysts can apply AI models via self-service, while data engineers can independently build AI-enabled pipelines.

Using AI Functions is quite simple. For example, consider a scenario where a user wants to classify the sentiment of some articles into Frustrated, Happy, Neutral, or Satisfied.

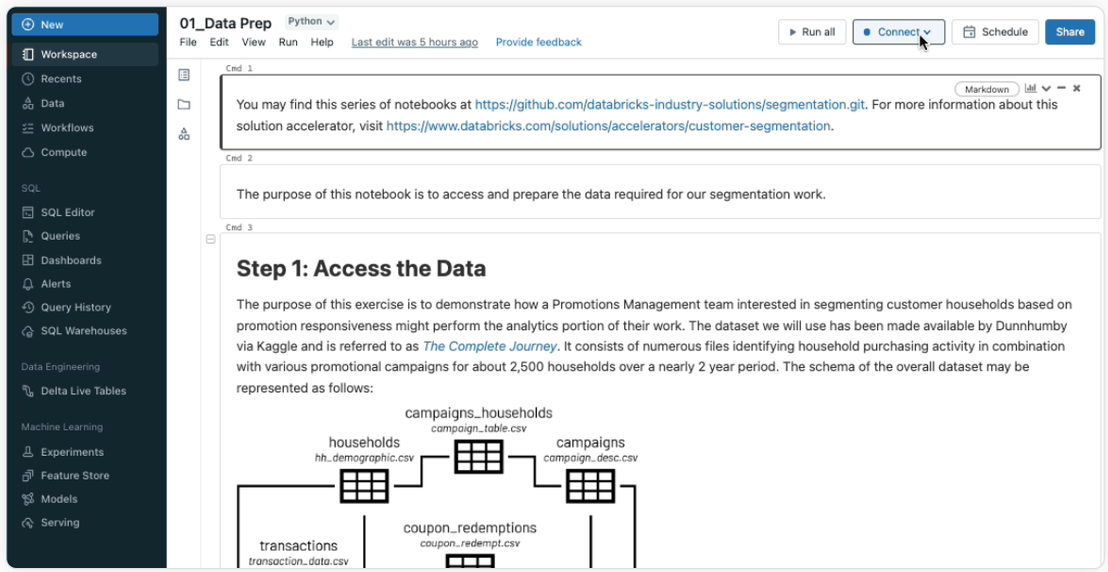
```
-- create a udf for sentiment classification
CREATE FUNCTION classify_sentiment(text STRING)
  RETURNS STRING
  RETURN ai_query(
    'Dolly', -- the name of the model serving endpoint
    named_struct(
      'prompt',
      CONCAT('Classify the following text into one of four categories [Frustrated,
Happy, Neutral, Satisfied]:\n',
        text),
      'temperature', 0.5),
    'returnType', 'STRING');
```

```
-- use the udf
SELECT classify_sentiment(text) AS sentiment
FROM reviews;
```

AI Functions are now in Public Preview. To sign up for the Preview, fill out the form here. To learn more, you can also read our detailed blog post or review the documentation (AWS | Azure).

Bring the power of SQL warehouses to notebooks

Databricks SQL warehouses are now public preview in notebooks, combining the flexibility of notebooks with the performance and TCO of Databricks SQL Serverless and Pro warehouses. To enable SQL warehouses in notebooks, simply select an available SQL warehouse from the notebooks compute dropdown.



Connecting serverless SQL warehouses from Databricks notebooks

Find and share insights with a new generation of dashboards

Discover a revamped dashboarding experience directly on the lakehouse architecture. Users can simply select a desired dataset and build stunning visualizations with a SQL-optional experience. Say goodbye to managing separate queries and dashboard objects - an all-in-one content model simplifies the permissions and management process. Finally, publish a dashboard to your entire organization, so that any authenticated user in your identity provider can access the dashboard via a secure web link, even if they don't have Databricks access.

New Databricks SQL Dashboards are currently in Private Preview. Contact your account team to learn more.

Leverage the flexibility of Python in SQL

Bring the flexibility of Python into Databricks SQL with Python user-defined functions (UDFs). Integrate machine learning models or apply custom redaction logic for data processing and analysis by calling custom Python functions directly from your SQL query. UDFs are reusable functions, enabling you to apply consistent processing to your data pipelines and analysis.

For instance, to redact email and phone numbers from a file, consider the following CREATE FUNCTION statement.

```
CREATE FUNCTION redact(a STRING)
RETURNS STRING
LANGUAGE PYTHON
AS $$
import json
keys = [“email”, “phone”]
obj = json.loads(a)
for k in obj:
  if k in keys:
    obj[k] = “REDACTED”
return json.dumps(obj)
$$;
```

Learn more about enrolling in the private preview here.

Integrations with your data ecosystem

At Data+AI Summit, Databricks SQL announced new integrations for a seamless experience with your tools of choice.

Databricks + Fivetran

We're thrilled to announce the general availability of Fivetran access in Partner Connect for all users including non-admins with sufficient privileges to a catalog. This innovation makes it 10x easier for all users to ingest data into Databricks using Fivetran. This is a huge win for all Databricks customers as they can now bring data into the lakehouse architecture from hundreds of connectors Fivetran offers, like Salesforce and PostgreSQL. Fivetran now fully supports serverless warehouses as well!

Learn more by reading the blog post here.

Databricks + dbt Labs

Simplify real-time analytics engineering on the lakehouse architecture with Databricks and dbt Labs. The combination of dbt's highly popular analytics engineering framework with the Databricks Platform provides powerful capabilities:

- dbt + Streaming Tables: Streaming ingestion from any source is now built-in to dbt projects. Using SQL, analytics engineers can define and ingest cloud/streaming data directly within their dbt pipelines.
- dbt + Materialized Views: Building efficient pipelines becomes easier with dbt, leveraging Databricks' powerful incremental refresh capabilities. Users can use dbt to build and run pipelines backed by MVs, reducing infrastructure costs with efficient, incremental computation.

To learn more, read the detailed blog post.