databricks 01.2 - Delta Live Tables

(https://databricks.com)

Data engineering with Databricks - Building our C360 database

Building a C360 database requires to ingest multiple datasources.

It's a complex process requiring batch loads and streaming ingestion to support real-time insights, used for personalization and marketing targeting among other.

Ingesting, transforming and cleaning data to create clean SQL tables for our downstream user (Data Analysts and Data Scientists) is complex.

John, as Data engineer, spends immense time....

- Hand-coding data ingestion & transformations and dealing with technical challenges:
  Supporting streaming and batch, handling concurrent operations, small files issues, GDPR requirements, complex DAG dependencies...

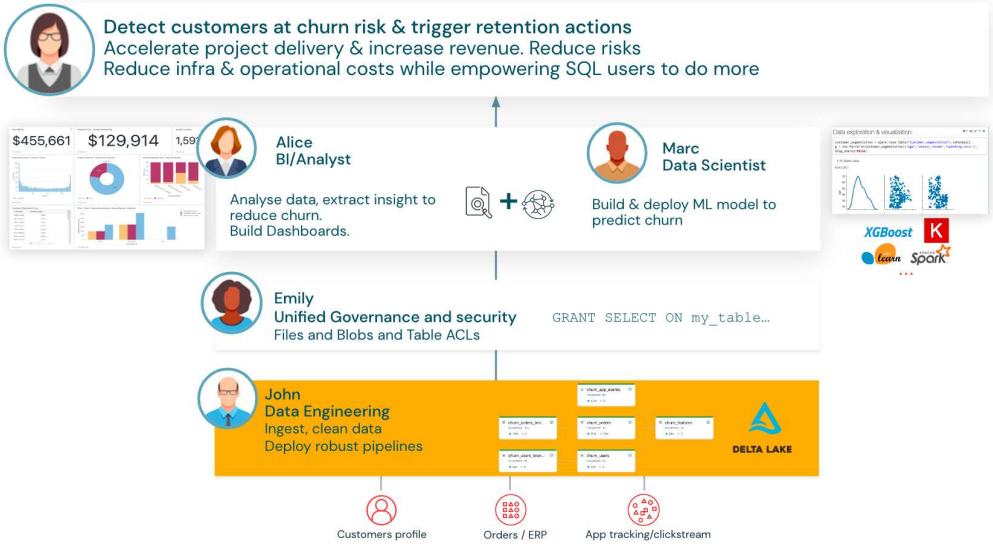
- Building custom frameworks to enforce quality and tests

- Building and maintaining scalable infrastructure, with observability and monitoring

Source: Forrester

Simplify Ingestion and Transformation with Delta Live Tables

In this notebook, we'll work as a Data Engineer to build our c360 database.
We'll consume and clean our raw data sources to prepare the tables required for our BI & ML workload.



We have 3 data sources sending new files in our blob storage (`/demos/retail/churn/`) and we want to incrementally load this data into our Datawarehousing tables:

- Customer profile data *(name, age, adress etc)*
- Orders history *(what our customer bough over time)*
- Streaming Events from our application *(when was the last time customers used the application, typically a stream from a Kafka queue)*

Databricks simplify this task with Delta Live Table (DLT) by making Data Engineering accessible to all.

DLT allows Data Analysts to create advanced pipeline with plain SQL.

Delta Live Table: A simple way to build and manage data pipelines for fresh, high quality data!

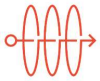
Accelerate ETL development

Trust your data

Enable analysts and data engineers to innovate rapidly with simple pipeline development and maintenance

With built-in quality controls and quality monitoring to ensure accurate and useful BI, Data Science, and ML

Remove operational complexity
By automating complex administrative tasks and gaining broader visibility into pipeline operations



Simplify batch and streaming
With self-optimization and auto-scaling data pipelines for batch or streaming processing

Re-building the Data Engineering pipeline with Delta Live Tables

In this example we will re-implement the pipeline we just created using DLT.

Examine the source.

A DLT pipeline can be implemented either in SQL or in Python.

- DLT pipeline definition in SQL ($./01.2%20-%20Delta%20Live%20Tables%20-%20SQL)
- DLT pipeline definition in Python ($./01.2%20-%20Delta%20Live%20Tables%20-%20Python)

Define the pipeline

Use the UI to achieve that:

- Go to **Workflows / Delta Live Tables / Create Pipeline**
- Specify **churn_data_pipeline** as the name of the pipeline
- As a source specify **one** of the above notebooks. **Either** the SQL or the Python one would work.
- Specify the parameters for the DLT job with the values below:

```
%run ./includes/SetupLab
```

Parameters for the DLT job

```
# Create the tables in a database in the hive metastore with data on dbfs
print("Specify the following database/schema when defining the DLT pipeline:\n" + databaseForDLT + "\n")
print("Specify the following storage location for the DLT pipeline tables:\n" + dltPipelinesOutputDataDirectory +
"\n")
```

```
Specify the following database/schema when defining the DLT pipeline:
odl_user_1237583_databrickslabs_com_retail_dlt

Specify the following storage location for the DLT pipeline tables:
/Users/odl_user_1237583_databrickslabs_com/retail/dlt_pipelines
```

Run the following after having set up and run the DLT job

Count the rows in the churn_features table

```
sqlStatement = "select count(*) from hive_metastore." + databaseForDLT + ".churn_features"
print("Executing:\n" + sqlStatement)
display(spark.sql(sqlStatement))
```

```
Executing:
select count(*) from hive_metastore.odl_user_1237583_databrickslabs_com_retail_dlt.churn_features
```

```
AnalysisException: [TABLE_OR_VIEW_NOT_FOUND (https://docs.databricks.com/error-messages/error-classes.html#table_or_view_not_found)] The table or view `hive_metastore`.`odl_user_1237583_databrickslabs_com_retail_dlt`.`churn_features` cannot be found. Verify the spelling and correctness of the schema and catalog.
If you did not qualify the name with a schema, verify the current_schema() output, or qualify the name with the correct schema and catalog.
To tolerate the error on drop use DROP VIEW IF EXISTS or DROP TABLE IF EXISTS.; line 1 pos 21;
'Aggregate [unresolvedalias(count(1), None)]
+- 'UnresolvedRelation [hive_metastore, odl_user_1237583_databrickslabs_com_retail_dlt, churn_features], [], false
```

Retrieve the table details

```
# Scroll the output to verify the storage location of the table
sqlStatement = "DESCRIBE EXTENDED hive_metastore." + databaseForDLT + ".churn_features"
print("Executing:\n" + sqlStatement)
display(spark.sql(sqlStatement))
```

```
Command skipped
```

Retrieve the table history

```
sqlStatement = "DESCRIBE HISTORY hive_metastore." + databaseForDLT + ".churn_features"
print("Executing:\n" + sqlStatement)
display(spark.sql(sqlStatement))
```

```
Command skipped
```

Rerun the DLT pipeline

As not new data are uploaded on the blob storage, there will be only a recalculation of the last table

Count the rows in the churn_features table again. It should be the same

```
sqlStatement = "select count(*) from hive_metastore." + databaseForDLT + ".churn_features"
print("Executing:\n" + sqlStatement)
display(spark.sql(sqlStatement))
```

```
Command skipped
```

Retrieve the table history. There is an additional entry now

```
sqlStatement = "DESCRIBE HISTORY hive_metastore." + databaseForDLT + ".churn_features"
print("Executing:\n" + sqlStatement)
display(spark.sql(sqlStatement))
```

```
Command skipped
```

Next up

Build and train a Machine Learning model ($./02%20-%20Machine%20Learning%20with%20MLflow)