DBA to Databricks

This book gives the perspective of a (SQL Server) DBA making the transition to databricks.

Databricks is a loud based data platform for lakehouse in the big three (Azure, Amazon, GCP). Based on the open source Delta table format.

About me: briandill.com (opens in a new window)

Glossary

Glossary of Terms associated with databricks

ADLS2 - Azure Data Lake Storage Gen2; hierarchical namespace–enabled blob storage used for analytics.
autoloader - Databricks feature that incrementally ingests new files from a cloud storage path.
autoscaling - automatic resizing of cluster worker count based on workload demand.
Azure Key Vault - Azure-managed secret store used for storing credentials, keys, and SAS tokens for DBX access.
Azure Managed Identity - Identity assigned to a DBX resource to securely access Azure services without secrets.
barcode - internal Databricks deployment identifier for clusters, notebooks, and jobs.
batch processing - processing data in grouped chunks rather than continuous streams.
blob storage - Azure term for inexpensive storage of any type of file. Amazon term is S3
bronze layer - raw ingestion layer in the medallion architecture; contains minimally processed source data.
catalog - top-level container in Unity Catalog that organizes schemas and tables.
checkpoint - directory used by Structured Streaming and Autoloader to track state for incremental processing.
cluster mode - controls how local vs remote driver processes run (single-node, standard, high-concurrency).
cluster policies - governance rules that restrict cluster configurations that users are allowed to create.
cloudFiles - Autoloader's file discovery mechanism API for streaming ingestion.
compute plane - where the VM nodes actually run—notebooks, jobs, worker tasks, shuffle operations.
concurrency - number of simultaneous queries a cluster can process; high-concurrency clusters optimize this.
control plane - the "brains" of DBX. The DBX UI is in the control plane and issues commands to the node(s) in the data plane
copy-on-write - Delta Lake mechanism where updates create new versions of modified Parquet files.
data lineage - Unity Catalog-tracked history of data flows between tables, notebooks, and jobs.
data plane - location of the user's data - typically in blob storage container in an Azure storage account
data skipping - Delta optimization that uses statistics in data file metadata to prune unnecessary

files during reads.
Databricks Connect - tool allowing you to run local IDE code against a remote Databricks cluster.
Databricks Runtime - versioned environment that defines Spark version, Delta version, and libraries on a cluster.
DBX - abbreviation of databricks
Delta Lake - storage layer that adds ACID transactions, schema enforcement, and time travel to Parquet.
Delta Live Tables - declarative ETL framework in DBX for building pipelines with quality checks and event tracking.
Delta Log - `_delta_log` folder containing JSON transaction files tracking commits and table versions.
driver node - orchestrates Spark tasks, maintains metadata, and coordinates work among executors/workers.
ETL - extract, transform, load; standard data integration pipeline pattern.
event grid - Azure service that publishes notifications for blob storage events to trigger ingestion.
executor node - worker process in a cluster that performs the actual computation for tasks.
Hive metastore - legacy metadata catalog used prior to Unity Catalog, scoped per workspace.
instance pool - pre-warmed VMs used to speed up Databricks cluster startup times and reduce cost.
job cluster - ephemeral cluster automatically spun up for a job, then terminated after completion.
lakehouse - unified architecture combining data lake storage and data warehouse capabilities.
manifest file - JSON export of a Delta table for external tools not natively Delta-aware.
materialized view - table whose results are precomputed and refreshed to improve query performance.
medallion architecture - bronze → silver → gold tiered data modeling design for incrementally refined datasets.
metastore - metadata service containing catalogs, schemas, tables, permissions, and lineage.
MLflow - open-source model tracking and experiment management system native to Databricks.
multi-cluster warehouse - SQL warehouse that can automatically scale out with multiple clusters for concurrency.
notebook - interactive coding environment for Scala, SQL, Python, or R inside the Databricks workspace.
optimize - Delta table maintenance operation to coalesce small files and improve performance.
parquet files - columnar data file format that is compressed. Contains column headers, data types, and some metadata.
photon - next-generation Databricks execution engine written in C++ for faster SQL performance.
pipelines - automated workflows in DBX Jobs or Delta Live Tables for scheduled ETL processes.
power BI connector - direct connection option between Power BI and Databricks SQL warehouses.
query profile - graphical explanation of query stages, tasks, and performance characteristics.
RBAC - role-based access control; permissions model in Unity Catalog for fine-grained governance.
schema - logical grouping of tables within a catalog (similar to a database schema in SQL Server).
schema enforcement - Delta Lake's ability to prevent writes that violate expected column types or names.
schema evolution - Delta Lake's automated capability to add new columns when enabled.
serverless SQL - Databricks-managed SQL compute with instant start and no cluster management.

shallow clone - lightweight metadata-only clone of a Delta table referencing the same underlying data files.
silver layer - cleaned and conformed data layer in the medallion architecture.
spark - distributed compute engine used under the hood by Databricks for parallel processing.
SQL warehouse - compute resource optimized for SQL workloads, formerly called SQL endpoints.
table ACLs - access controls that regulate who can query, modify, or manage tables.
table history - list of previous Delta table versions with timestamps and operations.
task - unit of work within a Spark stage executed on a worker.
time travel - Delta feature that lets you query older versions of a table via version number or timestamp.
token - workspace-specific personal access credential used to authenticate external tools to DBX.
Unity Catalog - centralized governance layer for permissions, metadata, auditing, and lineage.
UDF - user-defined function; custom Python/Scala logic registered for use in SQL.
UDI (Update/Delete/Insert) - mutation operations that modify Delta tables atomically.
vacuum - Delta operation that permanently deletes old versions and files older than a retention threshold.
view - saved SQL query definition that appears as a table but does not store its own data.
volume - UC-governed directory for unstructured files, supporting data and code assets.
widget - Notebook UI control (dropdowns, text boxes) enabling parameterization of jobs and dashboards.
workflow - job-based orchestrated set of tasks, dependencies, and triggers.
worker node - cluster node that executes Spark tasks and holds shuffled data.
Z-order - file-level clustering technique in Delta to co-locate related values for faster reads.

Building SELECT lists with SQL

If you are using SQL for ingestion and you want to rename columns

PascalCase to snake_case

```
-- PascalCase to snake_case
-- , OtherNFLStats AS other_nfl_stats

SELECT
    CONCAT(', ', column_name, ' AS ', LOWER(
      REGEXP_REPLACE(
        REGEXP_REPLACE(column_name, '([a-z0-9])([A-Z])', '$1_$2'), -- Pass 1: Lower to Upper
        '([A-Z])([A-Z][a-z])', '$1_$2'                             -- Pass 2: Acronym to Word
      )
    )) AS formatted_column
FROM information_schema.columns
WHERE table_name = 'my_table'
    AND table_schema = 'my_schema'
ORDER BY ordinal_position;
```

PascalCase to snake_case with substitution

```
-- PascalCase to snake_case
-- , OverageAmount AS overage_amt
WITH RawColumns AS (
    SELECT  column_name
        , LOWER(
```

```
        REGEXP_REPLACE(
            REGEXP_REPLACE(column_name, '([a-z0-9])([A-Z])', '$1_$2')
            , '([A-Z])([A-Z][a-z])', '$1_$2'
        )
    ) AS snake_name
  FROM information_schema.columns
  WHERE table_name = 'my_table'
  AND table_schema = 'my_schema'
)
SELECT CONCAT(', ', column_name, ' AS ',
    AGGREGATE( -- AGGREGATE is Spark-specific (thus applicable to databricks)
        -- Define the list of [pattern, replacement]
        ARRAY(
            STRUCT('_amount$'      AS pat, '_amt' AS rep)
          , STRUCT('_percent$'     AS pat, '_pct' AS rep)
          , STRUCT('_flag$'        AS pat, '_flg' AS rep)
          , STRUCT('_code$'        AS pat, '_cd'  AS rep)
          , STRUCT('_number$'      AS pat, '_nbr' AS rep)
          , STRUCT('_name$'        AS pat, '_nm'  AS rep)
          , STRUCT('_description$' AS pat, '_dsc' AS rep)
          , STRUCT('_desc$'        AS pat, '_dsc' AS rep)
          , STRUCT('_date$'        AS pat, '_dts' AS rep)
        )
        , snake_name -- Initial value
        , (acc, x) -> REGEXP_REPLACE(acc, x.pat, x.rep) -- Iterative function
    )
  ) AS formatted_column
FROM RawColumns
ORDER BY column_name;
```

snake_case to PascalCase + custom rules + suffix

```
-- snake_case to PascalCase with suffix substitution
WITH base AS (
  SELECT
```

```
      column_name
    , CASE
        WHEN column_name LIKE '%\_%' THEN  ARRAY_JOIN( TRANSFORM(SPLIT(LOWER(column_name), '_'), x ->
INITCAP(x) ), '' )  -- underscore to PascalCase
        ELSE INITCAP(column_name)
      END AS pascal_name
    , ordinal_position
  FROM my_catalog.information_schema.columns
  WHERE table_schema = 'my_schema'
    AND table_name = 'my_table'
)
SELECT
  CASE
    -- Parsed Excel files
    WHEN column_name = '_FILENAME' THEN CONCAT(', ', 'regexp_extract(', column_name, ', \'[^/]+$\', 0)', '
AS FileNM')
    WHEN column_name = '_ROW_NUM' THEN CONCAT(', ', column_name, ' AS RowNBR')
    WHEN column_name = '_SHEETNAME' THEN CONCAT(', ', column_name, ' AS SheetNM')
    -- metadata cols
    WHEN column_name = 'hcsys_file_name' THEN CONCAT(', ', 'regexp_extract(', column_name, ', \'[^/]+$\',
0)', ' AS FileNM') -- SourceHCSystemFileNM?
    WHEN column_name = 'hcsys_row_num' THEN CONCAT(', ', column_name, ' AS SourceHCSystemRowNBR')
    WHEN column_name = 'meta_updated' THEN CONCAT(', ', column_name, ' AS  EDWLastModifiedDTS')
    WHEN column_name = 'meta_deleted' THEN CONCAT(', ', column_name, ' AS MetaDeletedFLG')
    WHEN column_name = 'meta_surrogate_key' THEN CONCAT(', ', column_name, ' AS MetaSurrogateKeyID')
    WHEN column_name = 'meta_checksum' THEN CONCAT(', ', column_name, ' AS MetaChecksumID')
    WHEN column_name = 'meta_location' THEN CONCAT(', ', column_name, ' AS MetaLocationDSC')
    -- Replace end word w/ suffix
    WHEN pascal_name RLIKE '(?i)Amount$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Amount$', 'AMT'))
    WHEN pascal_name RLIKE '(?i)Percent$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Percent$', 'PCT'))
    WHEN pascal_name RLIKE '(?i)Flag$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Flag$', 'FLG'))
    WHEN pascal_name RLIKE '(?i)Code$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Code$', 'CD'))
    WHEN pascal_name RLIKE '(?i)Number$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Number$', 'NBR'))
    WHEN pascal_name RLIKE '(?i)Name$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Name$', 'NM'))
```

```
        WHEN pascal_name RLIKE '(?i)Description$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Description$', 'DSC'))
        WHEN pascal_name RLIKE '(?i)Desc$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Desc$', 'DSC'))
        WHEN pascal_name RLIKE '(?i)Date$' THEN CONCAT(', try_to_timestamp(', column_name, ', \'yyyy-MM-dd
HH:mm:ss\') AS ', REGEXP_REPLACE(pascal_name, '(?i)Date$', 'DTS'))  -- chg to try_to_date if no time data
        WHEN pascal_name RLIKE '(?i)Count$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Count$', 'CNT'))
        WHEN pascal_name RLIKE '(?i)Id$' THEN CONCAT(', ', column_name, ' AS ', REGEXP_REPLACE(pascal_name,
'(?i)Id$', 'ID'))
        WHEN pascal_name RLIKE '(?i)Address1$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Address1$', 'Address01TXT'))
        WHEN pascal_name RLIKE '(?i)Address2$' THEN CONCAT(', ', column_name, ' AS ',
REGEXP_REPLACE(pascal_name, '(?i)Address2$', 'Address02TXT'))
        -- Appended suffixes
        WHEN pascal_name RLIKE '(?i)Gender$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name,
'CD'))
        WHEN pascal_name RLIKE '(?i)Age$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name, 'NBR'))
        WHEN pascal_name RLIKE '(?i)Address$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name,
'TXT'))
        WHEN pascal_name RLIKE '(?i)City$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name, 'NN'))
        WHEN pascal_name RLIKE '(?i)State$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name, 'NN'))

        WHEN pascal_name RLIKE '(?i)Zip$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name, 'CD'))
        WHEN pascal_name RLIKE '(?i)Phone$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name,
'NBR'))
        WHEN pascal_name RLIKE '(?i)Fax$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name, 'NBR'))
        WHEN pascal_name RLIKE '(?i)Active$' THEN CONCAT(', ', column_name, ' AS ', CONCAT(pascal_name,
'FLG'))
        ELSE CONCAT(', ', column_name, ' AS ', pascal_name, 'TXT')
    END AS c
FROM base
ORDER BY ordinal_position;
```