DATA+AI SUMMIT

BY databricks

Learnings From the Field: “Migration From Oracle DW and IBM DataStage to Databricks on AWS”

Amine Benhamza, Lead Specialist Solutions Architect at Databricks
Himanshu Arora, Resident Solutions Architect at Databricks

Databricks
2023

Agenda

- Migration Pillars
- Data Models' Migration
- Code Migration
- SNCF - EDW Migration Project
  - Target Lakehouse Architecture
  - Data pipelines' implementation details
  - Outcomes
  - Best Practices & recommendations
- Q/A

Migration Pillars

Architecture & Infrastructure

- Establish deployment Architecture
- Implement Security and Governance framework

Data Migration

- Map Data Structures and Layout
- Complete One time load
- Implement incremental load approach

ETL and Pipelines

- **Migrate** Data transformation and pipeline code, orchestration and jobs
- **Validate:** Compare your results with On Prem data and expected results

BI and Analytics

- Re-write reports and analytics for Business Analysts and Business Outcomes
- Connect to reporting and analytics applications

Data Science/ML

- Establish connectivity to ML Tools
- Onboard Data Science teams

Data Modeling

7 Basic Steps to Success

1. **U**se dimensional modeling industry principals (Star Schema, Data Vault)
2. **Use Delta Tables & Databricks SQL (Photon)** - use Delta for your fact and dimension tables. Use DB SQL (Photon) for BI workloads
3. **Optimize file size** - optimize your file sizes for fast file pruning
4. **Z-Order Facts** - create Z-Order on your fact tables, key fields and most likely predicates
5. **Z-Order Dimensions** - create Z-Order on your dimension, key fields and most likely predicates
6. **Analyze Tables** - to gather statistics for Adaptive Query Execution Optimizer
7. **Cache Tables** - cache tables when you can. DBSQL has a great cache

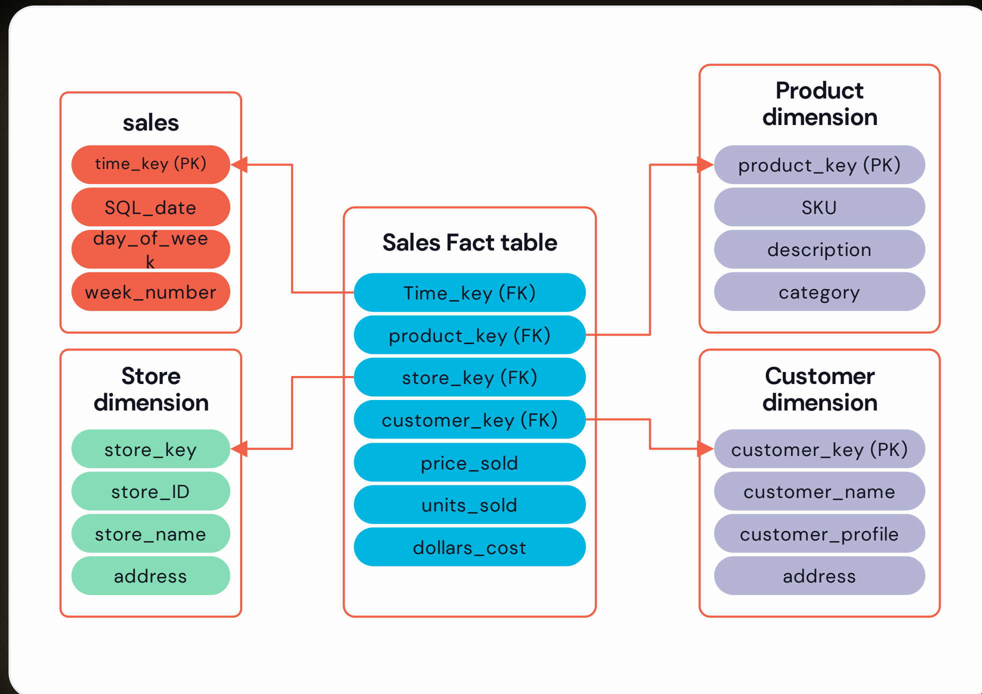
Data Modeling With Constraints

Familiar and easier schema modeling on the lakehouse

Primary + Foreign Key Declaration to allow end users to understand relationships between tables.

IDENTITY Columns automatically generate unique integer values when new rows are added.

Enforced CHECK Constraints to never worry about data quality or data correctness issues sneaking up on you.



Data Modeling: Data Vault 2.0

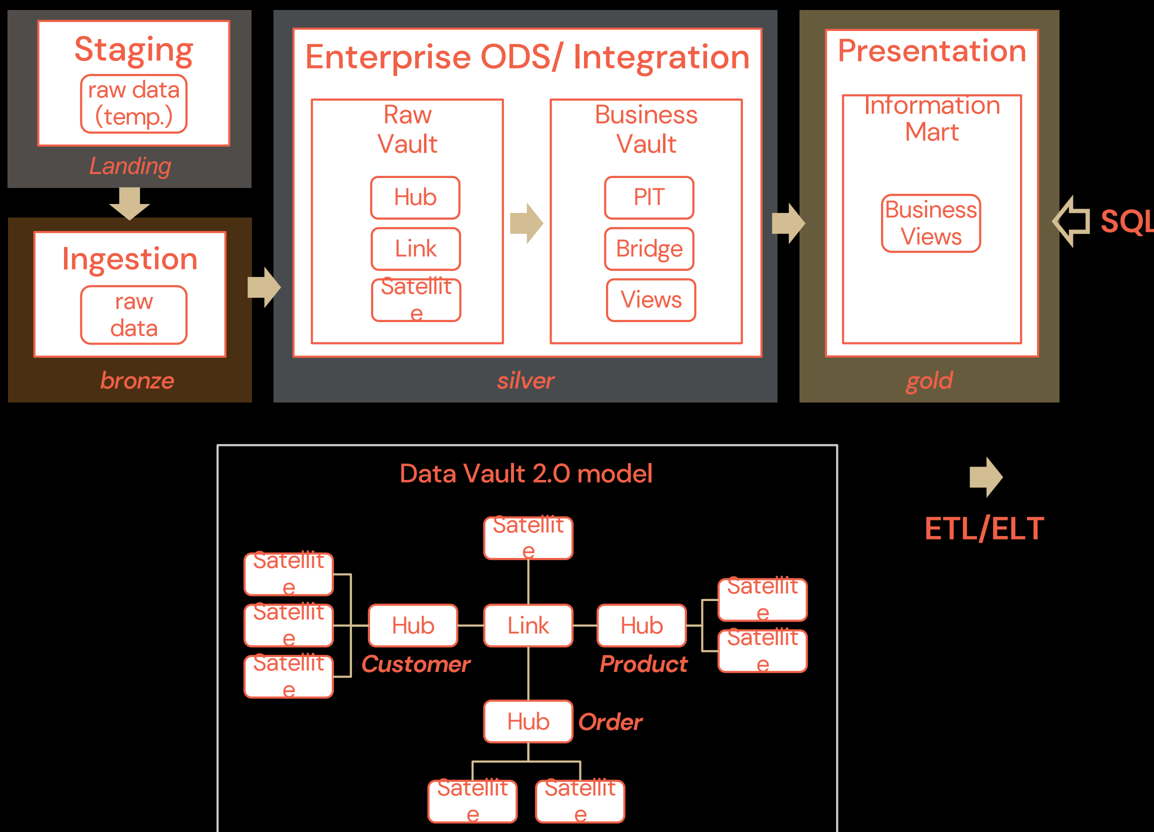
Staging: Raw data in its original format

Ingestion: Raw data converted to Delta

Integration – Raw Vault: Data is modeled as Hubs, Links and Satellites

Integration – Business Vault: Tables with applied business rules, data quality rules, cleansing and conforming rules

Presentation – Information Marts: Similar to a classical Data Mart with consumer oriented models



Code Migration

Main artefacts to consider

- Data Types
- SQL * Loader
- Data Definition Language (DDL) expressions
- Data Manipulation Language (DML) expressions
- PL/SQL Code

Code Migration

Supported Data Types

Data Type Category	Oracle Data Type	Converted Data Type
Character	CHAR, NCHAR, VARCHAR, VARCHAR2, NVARCHAR, NVARCHAR2, CHARACTER, TEXT, CLOB, NCLOB, LONG	STRING
Numeric	NUMBER [(p [, s])] FLOAT [(p)] BINARY_FLOAT BINARY_DOUBLE	BIGINT NUMERIC[(p [, s])] FLOAT FLOAT
DateTime	DATE TIMESTAMP	DATE TIMESTAMP
Byte	BLOB (Binary Large Object)	BINARY

Code Migration

SQL * Loader

- > **Oracle code**	- > **Databricks code**
<pre>LOAD DATA INFILE '/inputfolder/employee.csv' BADFILE '/outfolder/employee.rejected' DISCARDFILE '/outfolder/employee.discarded' INSERT INTO TABLE emp FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS (employee_id, first_name, last_name, email, phone_number, hire_date date 'mm/dd/yyyy');</pre>	<pre>COPY INTO emp FROM (SELECT employee_id, first_name, last_name, email, phone_number, hire_date::date FROM '/inputfolder/employee.csv') FILEFORMAT = CSV FORMAT_OPTIONS ('badRecordsPath' = '/outfolder/employee.rejected', 'delimiter' = ',', 'quote' = '"', 'dateFormat' = 'mm/dd/yyyy');</pre>

Code Migration

DDLs

Identity Column	-- > **Oracle code** `CREATE  TABLE identity_demo (` `    id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY START WITH 10 INCREMENT BY 10,description VARCHAR2(100) not null );` -- > **Databricks code** `CREATE  TABLE identity_demo (` `    id BIGINT GENERATED BY DEFAULT IDENTITY START WITH 10 INCREMENT BY 10,` `    description STRING not NULL);`
UNIQUE	Unique Constraints are not supported on Databricks yet. Required checks to be implemented in ETL processes. -- > **Oracle code** `CREATE TABLE table_x (a1 INTEGER UNIQUE,  a2 CHARACTER(10) );` -- > **Databricks code** `CREATE TABLE table_x (a1 INTEGER,  a2 STRING );`

Code Migration

DDLs

<table>
<tr>
<td>CHECK</td>
<td>

-- > **Oracle code**

```
CREATE TABLE table_x (
     column_1 INTEGER
       CHECK (column_1 > 0)
       CHECK (column_1 < 999)
       CHECK (column_1 NOT IN
(100,200,300))
       CONSTRAINT check_0
       CHECK (column_1 IS NOT
NULL),
     column_2 INTEGER
      CONSTRAINT check_1
       CHECK (column_2 > 0)
       CHECK (column_2 < 999)
);
```

</td>
<td>

-- > **Databricks code**

```
CREATE TABLE table_x (
     column_1 INTEGER,
     column_2 INTEGER
);
ALTER TABLE table_x ADD CONSTRAINT
column_1_checks
CHECK (
  column_1 > 0 AND
  column_1 < 999 AND
  column_1 NOT IN (100,200,300) AND
  column_1 IS NOT NULL
);
ALTER TABLE table_x ADD CONSTRAINT
column_2_checks
CHECK ( (column_2 > 0) AND
       (column_2 < 999)
);
```

</td>
</tr>
</table>

Code Migration

DMLs

UPSERT

-- > **Oracle code**

```sql
MERGE INTO bonuses D USING (
  SELECT Employee_id, salary,
department_id
  FROM
    employees
  WHERE department_id = 80
) S ON (D.employee_id = S.employee_id)
WHEN MATCHED THEN
  UPDATE
  SET D.bonus = D.bonus + S.salary *.01
  DELETE WHERE (S.salary > 8000)
WHEN NOT MATCHED THEN
  INSERT (D.employee_id, D.bonus)
  VALUES (S.employee_id, S.salary *.01)
  WHERE (S.salary <= 8000)
;
```

-- > **Databricks code**

```sql
MERGE INTO bonuses D USING (
  SELECT employee_id, salary, department_id
  FROM
    employees
  WHERE department_id = 80
) S ON (D.employee_id = S.employee_id)
WHEN MATCHED AND (S.salary > 8000) THEN
DELETE
WHEN MATCHED THEN UPDATE
  SET D.bonus = D.bonus + S.salary *.01
WHEN NOT MATCHED AND (S.salary <= 8000)
THEN
  INSERT (D.employee_id, D.bonus)
  VALUES (S.employee_id, S.salary *.01)
;
```

Code Migration

DMLs

Conditional INSERT	-- > **Oracle code** `INSERT ALL` `  WHEN order_total <= 100000 THEN` `    INTO small_orders` `  WHEN order_total > 100000 AND order_total <= 200000 THEN` `    INTO medium_orders` `  ELSE` `    INTO large_orders` `  SELECT order_id, order_total, sales_rep_id, customer_id` `    FROM orders;`	-- > **Databricks code** `INSERT INTO small_orders` `SELECT order_id, order_total, sales_rep_id, customer_id FROM orders` `WHERE order_total <= 100000 ;` `INSERT INTO medium_orders` `SELECT order_id, order_total, sales_rep_id, customer_id FROM orders` `WHERE order_total > 100000 AND order_total <= 200000 ;` `INSERT INTO large_orders` `SELECT order_id, order_total, sales_rep_id, customer_id FROM orders` `WHERE order_total > 200000 ;`

Code Migration

PL/SQL Code

Sql statements	-- > **Oracle Code**	-- > **Databricks code**
	`` DECLARE l_average_credit l_credit_limit%TYPE; l_max_credit l_credit_limit%TYPE; l_min_credit l_credit_limit%TYPE; BEGIN SELECT MIN(credit_limit), MAX(credit_limit), AVG(credit_limit) INTO l_min_credit, l_max_credit, l_average_credit FROM customers; END; ```	``` %python l_average_credit=None l_max_credit=None l_min_credit=None l_min_credit, l_max_credit, l_average_credit, = spark.sql(f""" SELECT MIN(credit_limit), MAX(credit_limit), AVG(credit_limit) MIN_COL FROM customers; """).first().asDict().values() ```

Code Migration

PL/SQL Code

CASE statements	-- > **Oracle Code**	-- > **Databricks code**
	<pre>DECLARE c_grade CHAR(1); c_rank VARCHAR2(20); BEGIN c_grade := 'B'; CASE c_grade WHEN 'A' THEN c_rank := 'Excellent' ; WHEN 'B' THEN c_rank := 'Very Good' ; WHEN 'C' THEN c_rank := 'Good' ; WHEN 'D' THEN c_rank := 'Fair' ; ELSE c_rank := 'No such grade' ; END CASE; END;</pre>	<pre>%python c_grade=None c_rank=None c_grade = 'B' If c_grade == 'A': c_rank = 'Excellent' elif c_grade == 'B': c_rank = 'Very Good' elif c_grade == 'C': c_rank = 'Good' elif c_grade == 'D': c_rank = 'Fair' else: c_rank = 'No such grade'</pre>

Code Migration

PL/SQL Code

FUNCTIONS

-- > **Oracle Code**

```
CREATE OR REPLACE FUNCTION
get_total_sales(
    in_year PLS_INTEGER
)
RETURN NUMBER
IS
    l_total_sales NUMBER := 0;
BEGIN
    SELECT SUM(unit_price * quantity)
    INTO l_total_sales
    FROM order_items
    INNER JOIN orders USING (order_id)
    WHERE status = 'Shipped'
    GROUP BY EXTRACT(YEAR FROM
order_date)
    HAVING EXTRACT(YEAR FROM
order_date) = in_year;
    RETURN l_total_sales;
END;
```

-- > **Databricks code**

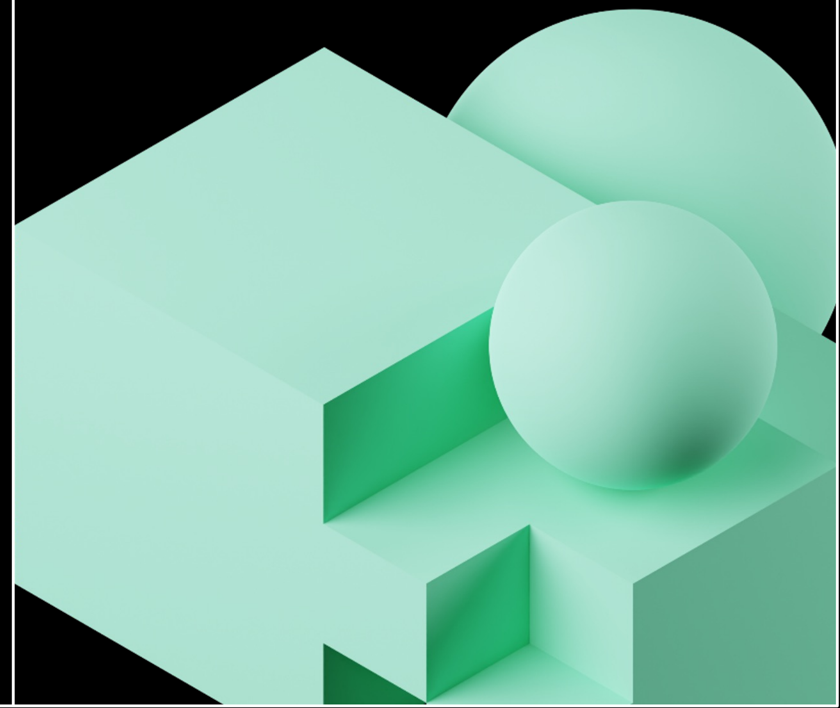
```
def get_total_sales(in_year):
  l_total_sales = 0

  l_total_sales, = spark.sql(f"""
SELECT SUM(unit_price * quantity)
SUM_COL
FROM order_items
INNER JOIN orders USING (order_id)
WHERE status = 'Shipped'
GROUP BY EXTRACT(YEAR FROM order_date)
HAVING EXTRACT(YEAR FROM order_date) =
{in_year};
""").first().asDict().values()

  spark.conf.set("var.l_total_sales",
l_total_sales)

  return l_total_sales;
```

SNCF - EDW Migration Project - Architecture & Best Practices



Context of Project

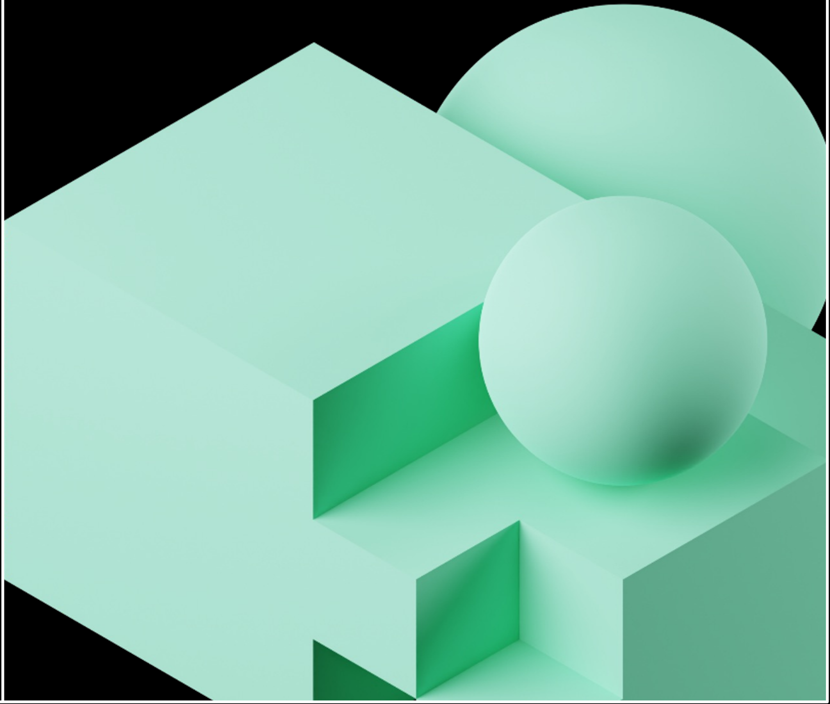
- SNCF - French national railway & transportation company
  - Project was for the Real Estate entity
- Migrate from on-prem Oracle EDW and IBM DataStage to Databricks on AWS due to
  - High 💰 of Oracle EDW & IBM DataStage
  - Rigid & non-scalable solution
  - No support for streaming, ML & AI

Context of Project

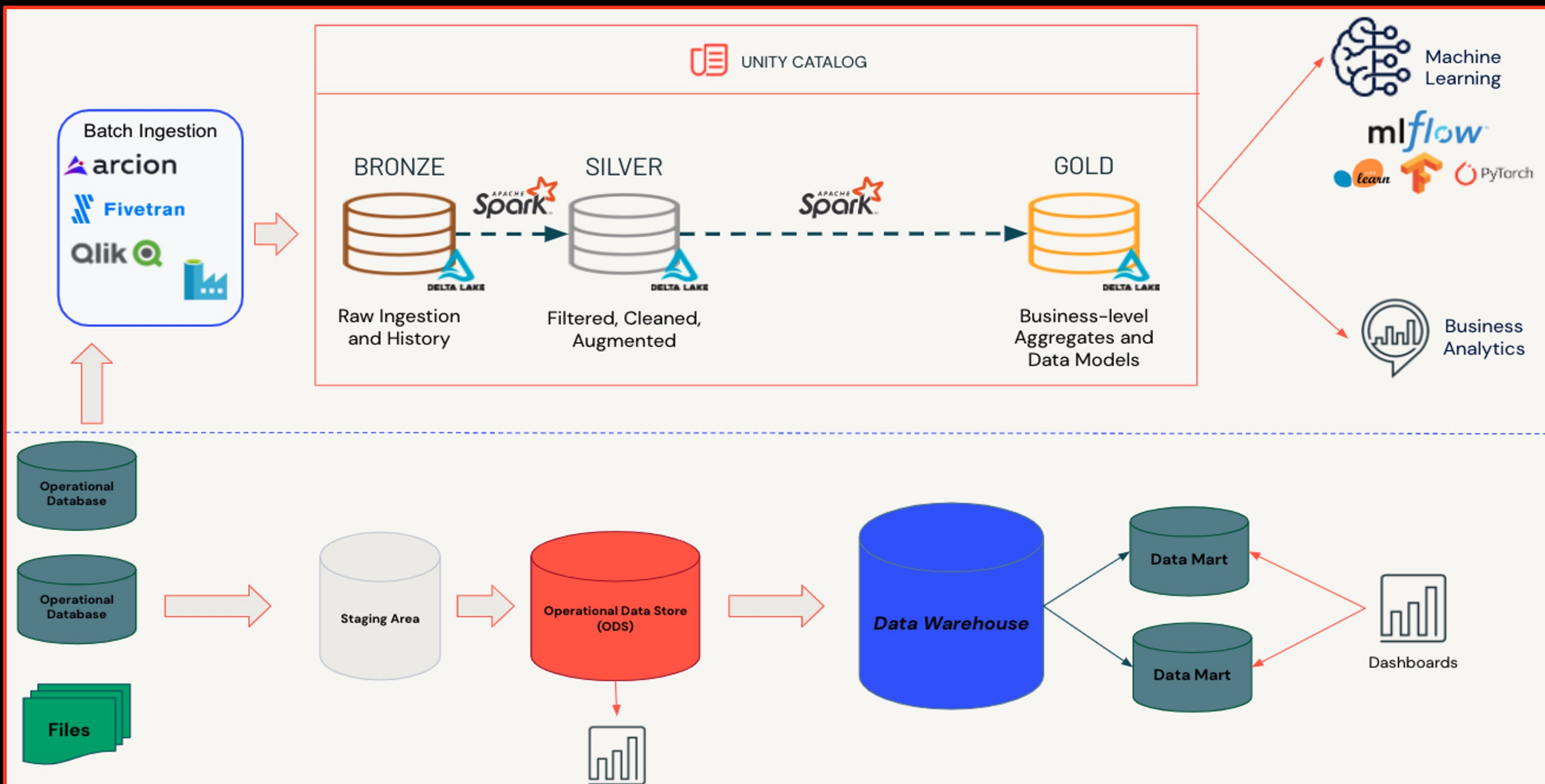
Databricks professional services team partnered with SNCF to:

- Migrate 1st data application (approx 30 DW tables) to Lakehouse
- Lead the data lake architecture design
- Lead and oversee the data pipelines implementation
- Provide best practices of pyspark, delta, databricks & software development

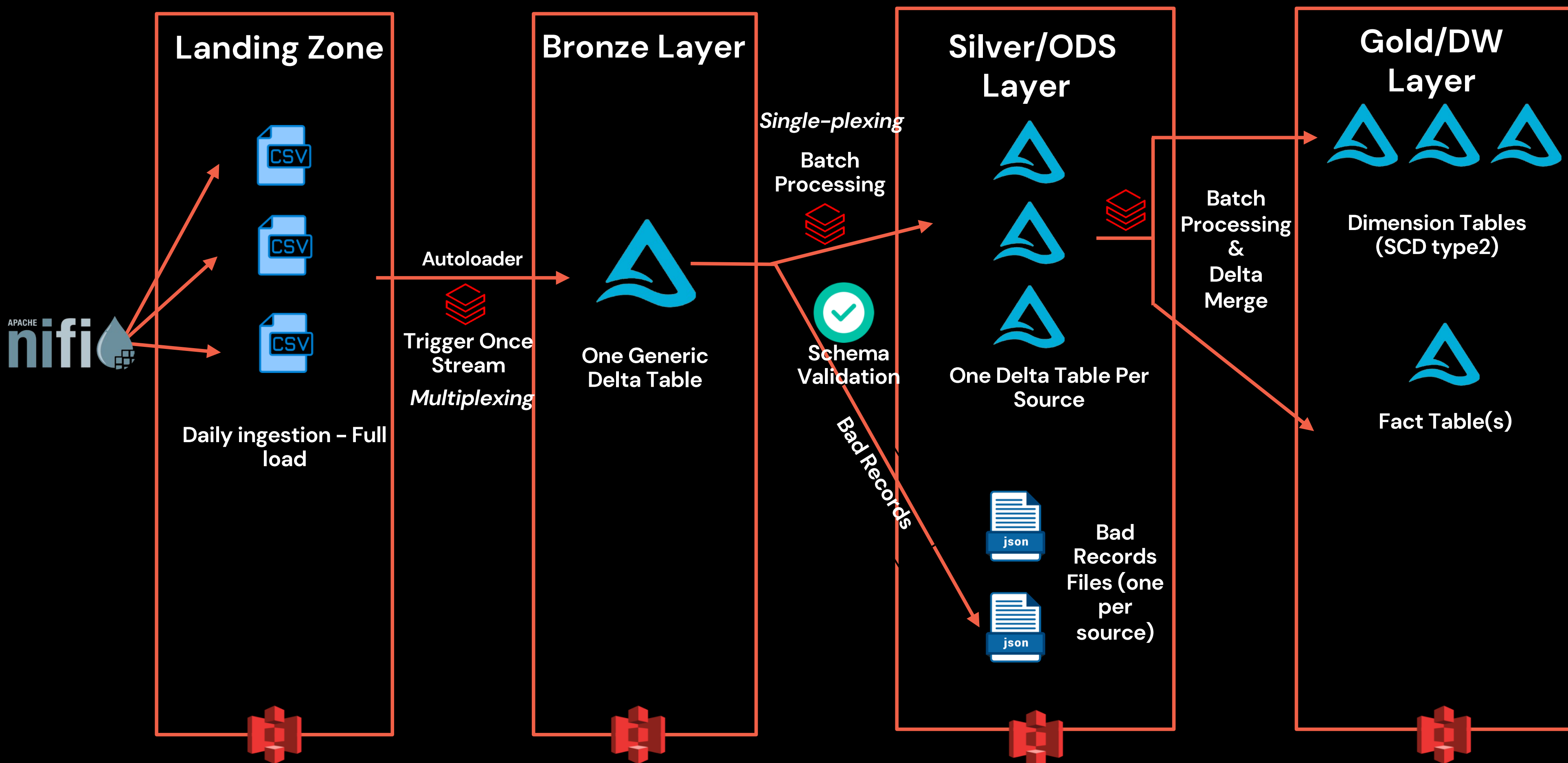
Target Architecture



Lakehouse & EDW - High Level Mapping



SNCF - Target DataLake Architecture



Landing Zone

- Daily ingestion in full load in landing zone by Apache Nifi
  - Constraint from source apps – can’t provide incremental load
    - Not an issue as daily volume < 100GB
- Input file format is CSV (non-standard CSV)
  - Encoding ISO-8859-1, 2 header rows, double quotes and semicolons in each data row & “|” used as column separator

```
HEADER|         ACTIVITES|09022022|348
BIEN_CODE_COMPLET|BIEN_ACTIVITE_DATE|BIEN_ACTIVITE_ECHEANCE|BIEN_ACTIVITE_DESCRIPTION|BIEN_ACTIVITE_PRIORITE|BIEN_ACTIVITE_TYPE|BIEN_ACTIVITE_ETAT|BIEN_ACTIVITE_LOT
"108440S_B 001_ET -1_L 330.0|||||||L 330.0 POL Placard";
"108440S_B 001_ET -1_L 340.0|||||||L 340.0  Armoire electrique";
"108440S_B 001_ET -1_L 350.0|||||||L 350.0 Armoire electrique";
"108440S_B 001_ET -1_L 360.0|||||||L 360.0 POL- Bureau Police";
"108440S_B 001_ET -1_L 370.0|||||||L 370.0 POL- Cellule";
"108440S_B 001_ET -1_L 380.0|||||||L 380.0 POL- Accueil Police";
"108440S_B 001_ET -1_L 390.0|||||||L 390.0 COM- concession";
"108440S_B 001_ET -1_L 400.0|||||||L 400.0 Reserve Bar";
"108440S_B 001_ET -1_L 410.0|||||||L 410.0 Issue de secours";
```

Landing to Bronze

Autoloader with text file format

- Spark only supports UTF-8 for text format (LineRecordReader.java)

Autoloader with CSV file format

- Encoding set to ISO-8859-1
- Header option set to false
- First header deleted as source name extracted from `input_file_name()`
- Second header row ingested,needed in silver layer to add column structure
- Delimiter set to `"@|@"`
- All data ended up in *_c0* column of generic delta table as *string*
- *source type* and *execution date* columns added in bronze delta table
- Trigger once mode – once per day

Bronze Layer

- One generic notebook ingesting all source files
- Single generic bronze table (one per real-estate app)
  - Partitioned by *`execution date`* and *`source type`* columns for downstream partition pruning
- Write mode set to *append*
  - Re-run the ingestion job to process the late arriving data
- Manually *`delete`* partitions for more than 7 days
- Run *`vacuum`* command right after delete

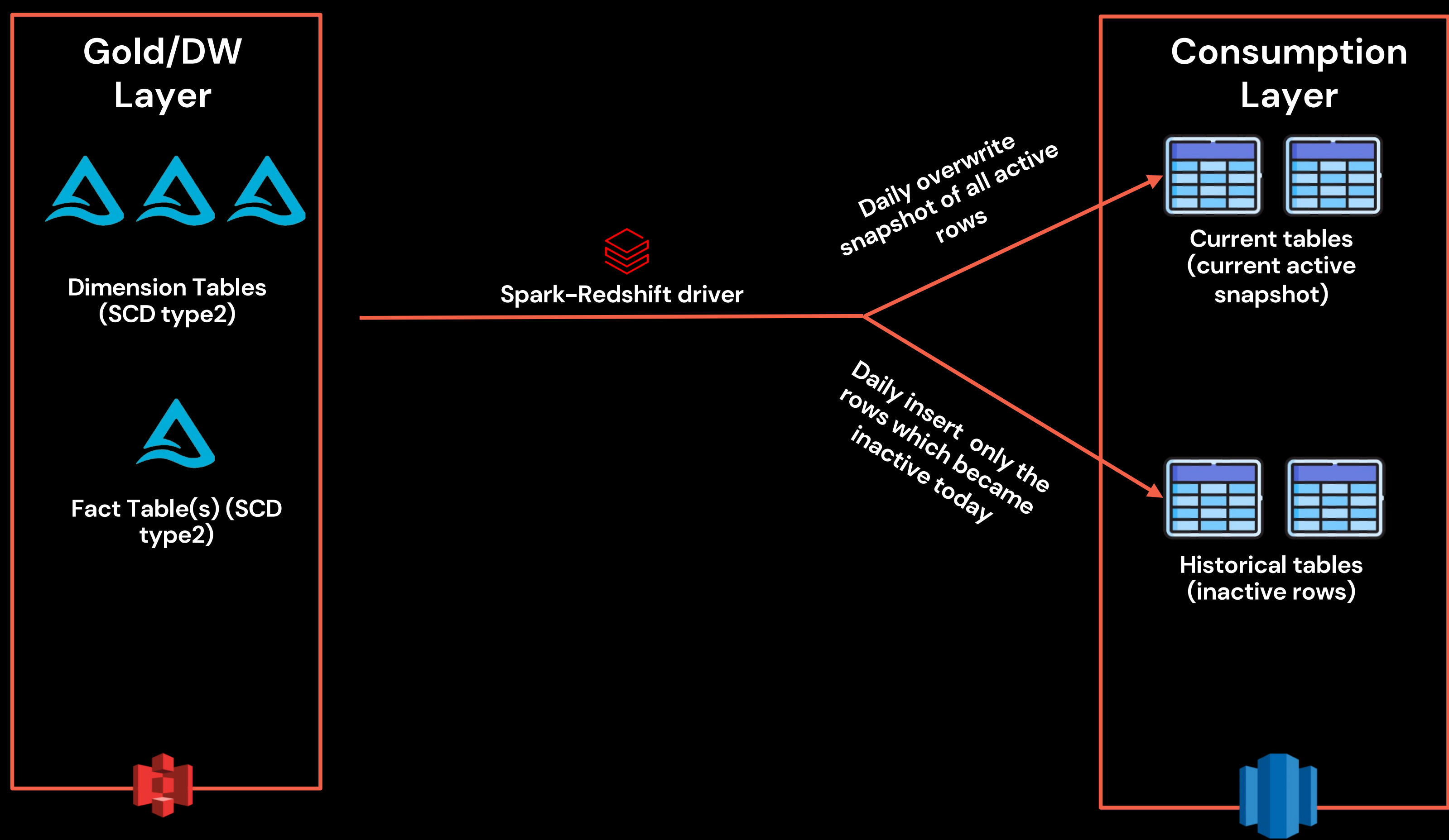
Silver/ODS Layer

- Adds structure to the raw data
  - As many delta tables in silver layer as number of different sources/tables in app
- One generic notebooks looping for all table/source names
- *`execution date`* and *`source type`* partition filter pushed to bronze table
- Validation the schema
  - Target schemas stored in Json format in S3 bucket (metadata files)
  - Corrupted rows ended up in bad record files
- Write mode set to *`overwrite`*
  - Bronze table daily *`execution date`* partition contains full data, so can re-run

Gold/DW Layer

- DW Star Schema implemented
- Dimension tables keep the historical data (SCD type 2)
  - Delta merge to implement SCD type 2 tables (example)
- Dimension tables needed surrogate keys
  - MD5 of business cols to uniquely identify each row
  - Delta identity column was not available back then, it's a better choice
    - Auto-increment integer better for data-skipping (w/ Zorder) than MD5 hex string
- Full load merge from silver to gold can be slow
  - Historical records are immutable, only active records are updated/deleted
  - Gold tables partitioned on a boolean flag column `RecordActive`
  - `RecordActive = True` used as filter in merge clause for partition pruning

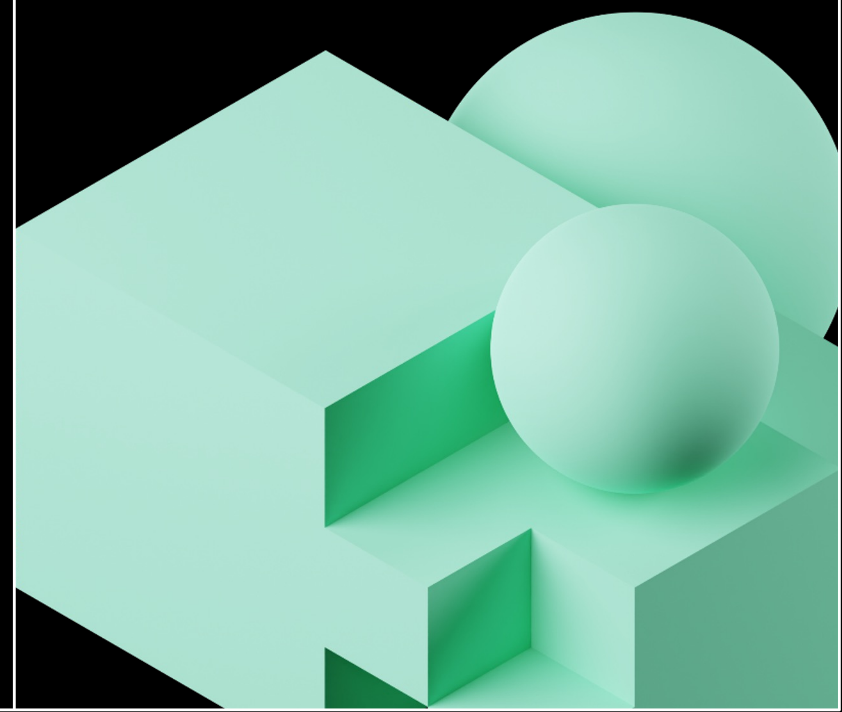
Consumption via Redshift



Consumption via Redshift

- Redshift imposed by central IT team – to consume data via Rest API
  - Databricks SQL Statement Execution API was under development back then, in public preview now.
- Spark-redshift driver can only perform insert and overwrite
  - Current tables: containing all the current active rows
  - Historical tables: containing only the historical inactive rows
- A daily batch job
  - Overwrites active rows snapshot from gold delta tables to current data tables in Redshift
  - Inserts only the rows that transitioned to inactive today to historical data tables in Redshift

Outcomes



Outcomes

70%

Cost Reduction

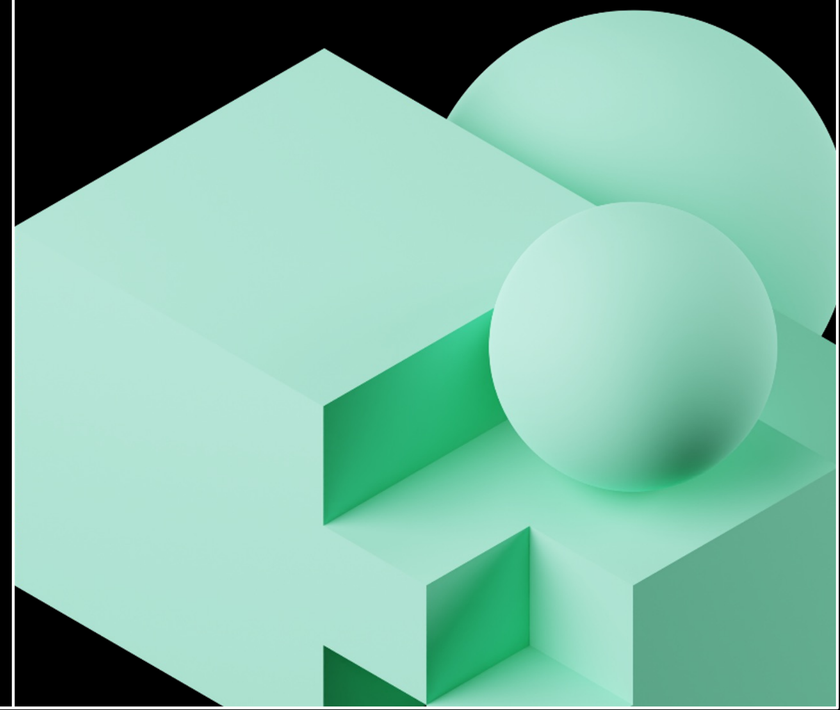
≈ 1 year's

worth of Acceleration with only

13 days

of consulting with Databricks Professional Services

Best Practices & Recommendations



Best Practices & Recommendations

- Code modularisation & unit testing
- Code documentation & indentation
- Monitor rejected bad records
- Spark Optimizations – Broadcast Join
- Spark Optimizations – Shuffle Partitions
- Delta Optimizations – Slow Merges
- Miscellaneous Recommendations

Code modularisation & unit testing

- Pyspark code in modular manner
  - Small single responsibility deterministic functions for business transformations
  - Take dataframe or configuration in and return dataframe out
  - Easier to unit test

Fonction correction fichier csv

Python

Dataframe In

Deterministic Business Logic

Dataframe Out

```python
def raw_processing(di):
    di = di.withColumnRenamed("_c0", "Valeur").withColumn("Source", input_file_name()) #adds column with complete path of file

    data_df = (
            di.withColumn("DATE_CHARGEMENT",current_date())
              .withColumn("Valeur", regexp_replace('Valeur', '"', ''))
              .withColumn("Valeur", regexp_replace('Valeur', ';', ''))
             )

    #Ajout d'un ID_Source incrémenté suivant la source de la donnée. Cela nous permettra de supprimer les headers inutiles
    w = Window.partitionBy("Source")
    data_di = data_df.withColumn("count", count("*").over(w))\
        .withColumn("ID_Source", row_number().over(w.orderBy("Source"))).drop("count").filter(col("ID_Source") > 1)

    #data_di contient : Source,Date_Chargement,Valeur,ID_Source
    #Modification du champ Source en enlevant le path + les numéros de lot + l'extension
    data_di = data_di.withColumn("Source", when(expr("substring(substring_index(Source,  '/', -1),1,length(substring_index(Source,  '/', -1))-length(substring_index(Source,'_', -2))-1)") == "IMMOSIS_DEC_INVENTAIRE",expr("substring(substring_index(Source,  '/', -1),1,length(substring_index(Source,  '/', -1))-length(substring_index(Source,'_', -2))-1)"))
                                  .when(expr("substring(substring_index(Source,  '/', -1),1,length(substring_index(Source,  '/', -1))-length(substring_index(Source,'_', -2))-1)") == "IMMOSIS_DEC_UDJ",expr("substring(substring_index(Source,  '/', -1),1,length(substring_index(Source,  '/', -1))-length(substring_index(Source,'_', -2))-1)"))
                                  .otherwise(expr("substring(substring_index(Source,  '/', -1),1,length(substring_index(Source,  '/', -1))-length(substring_index(Source,'_', -1))-1)")))

    data_di = data_di.drop("_rescued_data")

    return data_di
```

Code modularisation & unit testing

```python
import unittest
from chispa import assert_df_equality
from pyspark.sql.types import StructType,StructField, StringType, IntegerType, DateType

# Ingestion Modules' unit tests suite
class IngestionModulesTests(unittest.TestCase):

    # Test name should start with 'test_' to be considered as part of test suite
    def test_raw_processing(self):

        # Create input file
        df = spark.createDataFrame(["header1\";", \
                                    "Col1|Col2|Col3|Col4|Col5\";", \
                                    "Zoning||Modifications données Gares|Clôturé|L 010.0 Hall d'acces par parvis Ste Devote\";"], StringType())
        df.repartition(1).write.format("csv").mode("overwrite").save("/tmp/raw_processing_test/input/")

        # Read inout dataframe
        input_df = spark.read.csv("/tmp/raw_processing_test/input/")

        # Generate output dataframe
        output_df = raw_processing(input_df)

        # Create expected dataframe
        expected_df = spark.createDataFrame([("Col1|Col2|Col3|Col4|Col5", 2), \
                                             ("Zoning||Modifications données Gares|Clôturé|L 010.0 Hall d'acces par parvis Ste Devote", 3)\
                                            ], \
                                            StructType([ \
                                                StructField("Valeur", StringType(),True), \
                                                StructField("ID_Source", IntegerType(),False) \
                                              ]) \
                                           )
        expected_df = expected_df.withColumn("DATE_CHARGEMENT", current_date())

        # Assertions
        assert_df_equality(output_df.select("Valeur", "ID_Source", "DATE_CHARGEMENT"), expected_df)  # not checking "Source" column as it comes from this unit test
        self.assertEqual(output_df.count(), 2)
```

Business Function to Test

Comparing with Expected Dataframe

Command took 0.04 seconds -- by ext.himanshu.arora@sncf.fr at 04/04/2022, 16:33:19 on dbkscluster-dev-01

nd 4

```python
suite = unittest.TestLoader().loadTestsFromTestCase(IngestionModulesTests)
runner = unittest.TextTestRunner(verbosity=2)
runner.run(suite)
```

Running test with python UnitTest

Code documentation & indentation

- Python documentation conventions
- Comments in between of code blocks for complex transformation
- Online python code formatter to properly indent the code
  - Now available in the product itself, feature called – ***new notebook editor***

```python
def inventaire_process_final(inventaire_df: DataFrame, inventaire_df_doublon: DataFrame) -> DataFrame:
    """Prepares the final inventory dataframe which is the union of
    inventory deduplicated dataframe, archived & inactive inventory dataframe
    and non-archived and active inventory dataframe

    Parameters
    ----------
    inventaire_df : pyspark.sql.DataFrame
        The inventory dataframe
    inventaire_df : pyspark.sql.DataFrame
        The inventory dataframe containing duplicates

    Returns
    -------
    pyspark.sql.DataFrame
        The final inventory datafram

    Raises
    ------
    """

    inventaire_df_join = inventaire_df.join(inventaire_df_doublon, "CODE_COMPLET_BIEN", "left")
    inventaire_df_join_doublon = inventaire_df_join.filter(F.col("ID_CODE_COMPLET") >= 2)
    inventaire_df_join_no_doublon = inventaire_df_join.filter(F.col("ID_CODE_COMPLET") < 2)

    inventaire_df_join_doublon = inventaire_df_join_doublon.withColumn(
        "BIEN_ACTIF",
        F.when(
            (F.expr("DATE_FIN_EXISTENCE is null"))
            | (F.col("DATE_FIN_EXISTENCE") > F.col("DATE_CHARGEMENT")),
            F.when((F.col("BIEN_ARCHIVE") == 0), F.lit(1)).otherwise(F.lit(0)),
        ).otherwise(F.lit(0)),
    )

    inventaire_df_actif_non_archive = inventaire_df_join_doublon.filter(
        (F.col("BIEN_ARCHIVE") == 0) | (F.col("BIEN_ACTIF") == 1)
    )
    inventaire_df_non_actif_archive = inventaire_df_join_doublon.filter(
        (F.col("BIEN_ARCHIVE
    )

    # Window aggregate to only select the first row per CODE_COMPLET_BIEN (deduplication)
    w = Window.partitionBy("CODE_COMPLET_BIEN")
    window_df = inventaire_df_non_actif_archive.withColumn(
        "ID_CODE_COMPLET_NON_ACTIVE", F.row_number().over(w.orderBy("CODE_COMPLET_BIEN"))
    )
    inventaire_df_non_actif_archive_no_doublon = window_df.filter(
        F.col("ID_CODE_COMPLET_NON_ACTIVE") == 1
    ).drop("ID_CODE_COMPLET_NON_ACTIVE")

    inventaire_df_final = inventaire_df_join_no_doublon.unionByName(
        inventaire_df_non_actif_archive_no_doublon.drop("BIEN_ACTIF")
    ).unionByName(inventaire_df_actif_non_archive.drop("BIEN_ACTIF"))

    return inventaire_df_final
```

Python Docstring

Additional business comments

Monitor rejected bad records

- Bad records rejected in silver layer end up in json files stored at *`badrecordpath`* (a spark option)
- A daily batch job triggers after silver layer jobs to
  - Append all the bad record from json files a target delta table
  - With additional columns *`execution date`* and *`source type`*
- Compute some technical KPIs & display them on dashboarding tools like DBSQL, PowerBI
  - Number of total rejected rows today
  - Number of total rejected rows today per source type
  - Timeseries graph of number of total rejected rows per day and source type

Spark Optimizations – Broadcast Join

- Joins/merges in Gold layer induce data shuffling
  - Avoid some of the shuffle for smaller tables/dataframes by broadcasting them to worker nodes
- Driver with 32 GB+ RAM, safe to broadcast any table or dataframe of size <= 200MB

```
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "209715200")
```

- Driver can collect up to 1GB by default, change it to 8GB
  - Set before the cluster starts hence put this in advance cluster options

```
spark.driver.maxResultSize 8g
```

Spark Optimizations – Shuffle Partitions

- Join/aggregations induce shuffle in spark
  - By default number of shuffle partitions = 200 (which is almost never the right value)
- Recommendation for tuning the # shuffle partitions
  - Either fine tune it based on shuffle stage size (refer spark summit talk from Daniel Tomes)
  - Or as a rule of thumb, set it to 2x or 3x of number of total worker cores, to fully leverage all cpu cores during shuffle stages

```
spark.conf.set("spark.sql.shuffle.partitions",3*sc.defaultParallelism)
```

Delta Optimizations – Slow Merges

- Leverage LowShuffleMerge
  - Enable by default in DBR 10.4+
- For optimized file sizes, use AutoOptimize features of Delta lake by using `delta.autoOptimize.optimizeWrite` & `delta.autoOptimize.autoCompact` options
  - Target file size 128MB
  - Smaller file size implies less data rewrite during merges
  - Further reduce file size using `delta.targetFileSize` option
- Broadcast the source dataframe being merged
- Refer DAIS talk from Justin Breese to learn more

Other Recommendations

- Leverage the databricks jobs workflow feature to schedule and orchestrate the notebooks' execution.
- Use Databricks Repos feature for seamless integration with Git repository
  - Comes with Rest API to integrate into CI/CD setup
- BladeBridge is technology partner of Databricks which can generate Pyspark code from IBM DataStage pipelines
- Delta live tables (managed ELT feature) of Databricks can also be a great choice to speed up the migration project
  - Takes care of pipeline operational tasks and allows developers to focus on business code
  - Enforces and monitors data quality rules

Q/A

databricks