**ETL Process for Export CSV File**

**Name: Vikas Reddy Gorantla**

**Date: 08-12-2024**

**Extracting Data from Export CSV File**

# Step 1: Extract the data from CSV

from pyspark.sql import SparkSession

# Initialize Spark Session

spark = SparkSession.builder.appName('delta first program').getOrCreate()

# 1. Query Delta table using table name from the metastore

data\_from\_table = spark.table("default.export")

data\_from\_table.show()

# 2. Query Delta table using path from DBFS or cloud storage

data = spark.read.format("delta").load("dbfs:/user/hive/warehouse/export")

data.show()

A screenshot of a computer

Description automatically generated

**Transfromation of Data(Export CSV File)**

from pyspark.sql.functions import col

data = spark.read.format("delta").load("dbfs:/user/hive/warehouse/export")

# 1. \*\*Filtering data\*\*: Filter employees who have a salary greater than 10000

filtered\_data = data.filter(col("salary") > 10000)

print("Filtered Data (salary > 10000):")

filtered\_data.show(5)

A screenshot of a computer

Description automatically generated

from pyspark.sql.functions import year,current\_date

# 2. \*\*Adding a new column\*\*: Calculate age from birthdate

# Get the current year and subtract birth year to calculate age

current\_year = year(current\_date())

data\_with\_age = filtered\_data.withColumn("age", current\_year - year(col("birthDate")))

print("Data with Calculated Age:")

data\_with\_age.show(5)

A screenshot of a computer code

Description automatically generated

# 3. Adding another new column: Calculate salary after a 20% tax deduction

data\_with\_tax = data\_with\_age.withColumn("salary\_after\_tax", col("salary") \* 0.8)

print("Data with Salary After Tax Deduction:")

data\_with\_tax.show(5)

A screenshot of a computer

Description automatically generated

# 4. Renaming columns: Rename 'ssn' to 'social\_security\_number' and 'salary' to 'annual\_salary'

renamed\_data = data\_with\_tax.withColumnRenamed("ssn", "social\_security\_number").withColumnRenamed("salary", "annual\_salary")

print("Data with Renamed Columns:")

renamed\_data.show(5)

A screenshot of a computer

Description automatically generated

from pyspark.sql.functions import when

# 5. Conditional Columns: Create a new column to categorize salary

categorized\_data = renamed\_data.withColumn(

    "salary\_category",

    when(col("annual\_salary") > 50000, "High")

    .when(col("annual\_salary") > 20000, "Medium")

    .otherwise("Low"))

print("Data with Salary Categories:")

categorized\_data.show(5)

A screenshot of a computer

Description automatically generated

from pyspark.sql.functions import avg

# 6. GroupBy and Aggregation: Calculate the average salary by gender

salary\_by\_gender = renamed\_data.groupBy("gender").agg(

    avg("annual\_salary").alias("avg\_salary"))

print("Average Salary by Gender:")

salary\_by\_gender.show(5)

A close-up of a number

Description automatically generated

# 7. Sorting: Sort the data by salary in descending order

sorted\_by\_salary = renamed\_data.orderBy(col("annual\_salary").desc())

print("Data Sorted by Salary (Descending):")

sorted\_by\_salary.show(5)

A screenshot of a computer

Description automatically generated

# 8. Selecting specific columns: Select only 'id', 'firstName', 'lastName', and 'salary'

selected\_columns = renamed\_data.select("id", "firstName", "lastName", "annual\_salary")

print("Selected Columns:")

selected\_columns.show(5)

A screenshot of a computer code

Description automatically generated

from pyspark.sql.functions import date\_diff

# 9. Date Difference: Calculate the number of days since the employee's birthdate

data\_with\_days\_since\_birth = renamed\_data.withColumn(

    "days\_since\_birth", date\_diff(current\_date(), col("birthDate")))

print("Data with Days Since Birth:")

data\_with\_days\_since\_birth.show(5)

A screenshot of a computer

Description automatically generated

# 10. Filtering based on age: Filter employees who are older than 30

filtered\_by\_age = data\_with\_days\_since\_birth.filter(col("age") > 30)

print("Filtered Employees Older Than 30:")

filtered\_by\_age.show(5)

A screenshot of a computer

Description automatically generated

# Apply transformations to the data in this Notebook

# Transformation steps (filter, add columns, etc.) ...

# Final transformed data (filtered\_by\_age) should be saved to a Delta table

filtered\_by\_age.write.format("delta").mode("overwrite").saveAsTable("default.transformed\_employees")

**Loading Transformed Data (Export)**

# \*\*Load Data (Save as Delta Table)\*\*: Save the final transformed data to a Delta table

transformed\_data = spark.table("default.transformed\_employees")

transformed\_data.show(5)

A screenshot of a computer

Description automatically generated

After executing the code, the new table transformed\_employees is successfully added to the **Hive Metastore(in default database)** in Databricks. This table now holds the transformed data, and it can be queried like any other table in the database.

**ETL Job Creation and Execution Overview**

An **ETL (Extract, Transform, Load)** job was created in Databricks to automate the entire data pipeline process. The job consists of three sequential tasks, each performing a key operation in the data pipeline: **Extraction**, **Transformation**, and **Loading**. Each task was associated with a corresponding notebook that contains the specific code for its operation.

**Tasks Overview:**

1. **Task 1 - Extraction**:  
   The **Extraction** task is responsible for pulling data from the source system or external data source. It runs a notebook that contains the **extraction code** to fetch the required data and bring it into the Databricks environment. This is the first step in the ETL pipeline.
2. **Task 2 - Transformation**:  
   The **Transformation** task processes and refines the extracted data. It executes a notebook containing the **transformation code**, which applies various data manipulation techniques such as filtering, cleaning, and aggregating the data to meet the necessary business rules.
3. **Task 3 - Loading**:  
   The **Loading** task is responsible for saving the transformed data into the target data storage. This task runs a notebook with the **loading code**, which writes the transformed data into a Delta table within the Databricks Hive Metastore, making it available for further querying and analysis.

**Job Execution:**

The ETL job was executed on a Databricks cluster. Upon running the job, the execution was **successful**, meaning all three tasks—Extraction, Transformation, and Loading—were completed without any errors. The final outcome was the creation of a Delta table(transformed\_employees) that contains the transformed data, ready for use in downstream applications.

**Summary:**

The ETL job successfully automated the entire pipeline process, handling the extraction, transformation, and loading of data in a seamless flow. By organizing the process into three tasks within a single job, we ensured that the pipeline runs in a structured, error-free manner. The **successful result** confirms that the data was successfully extracted, transformed, and loaded into the target Delta table in the Databricks environment.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated