Week XI

Capstone Project

Intermediate Data Engineer in Azure
Trainer: Balazs Balogh

CUBIX
INSTITUTE OF TECHNOLOGY

Medallion Architecture III. – Gold Layer – Daily Sales Metrics

```python
import pyspark.sql.functions as sf
from pyspark.sql import DataFrame


def get_daily_sales_metrics(wide_sales: DataFrame) -> DataFrame:
    """
    Calculates daily sales metrics from the wide_sales DataFrame.

    Note: In order to get only two decimals for the averages, value is rounded.

    :param wide_sales:  Input DataFrame containing wide sales data.
    :return:            DataFrame with daily metrics including "SalesAmountSum", "SalesAmountAvg",
                        "ProfitSum", and "ProfitAvg" grouped by "OrderDate".
    """

    return (
        wide_sales
        .groupBy(sf.col("OrderDate"))
        .agg(
            sf.sum(sf.col("SalesAmount")).alias("SalesAmountSum"),
            sf.round(sf.avg(sf.col("SalesAmount")), 2).alias("SalesAmountAvg"),
            sf.sum(sf.col("Profit")).alias("ProfitSum"),
            sf.round(sf.avg(sf.col("Profit")), 2).alias("ProfitAvg"),
        )
    )
```

Daily Sales Metrics contains the **sum** and **average** of the Sales Amount and the Profit columns.

These kind of tables provide aggregated insights that support decision making.

Since it’s only five columns, query performance is high.
Enables trend analysis to identify patterns over time.

Checking the Wide Sales and Metrics calculations

```
master_tables = [
    "sales",
    "calendar",
    "customers",
    "products",
    "product_category",
    "product_subcategory",
]

master_dataframes = {
    table: read_file_from_datalake(
        container_name="capstoneproject",
        file_path=f"02_silver/{table}",
        format="delta"
    )
    for table in master_tables
}

wide_sales_df = get_wide_sales(
    sales_master=master_dataframes["sales"],
    customers_master=master_dataframes["customers"],
    products_master=master_dataframes["products"],
    product_category_master=master_dataframes["product_category"],
    product_subcategory_master=master_dataframes["product_subcategory"],
    calendar_master=master_dataframes["calendar"],
)

wide_sales_df.count()

display(wide_sales_df)

wide_sales_df.createOrReplaceTempView("wide_sales")
```

Create the Wide Sales table, with joining all the dimensions and the Sales fact table together.

Checking the Wide Sales and Metrics calculations

The Wide Sales is the best starting point for creating metrics, here are some examples:

```
%sql

-- Total sales and profit by month to identify high-performing months.
SELECT
   MonthName,
   CalendarYear,
   SUM(SalesAmount) AS TotalSales,
   SUM(Profit) AS TotalProfit
FROM
   wide_sales
GROUP BY
   CalendarYear, MonthName, MonthNumberOfYear
ORDER BY
   CalendarYear, MonthNumberOfYear;
```

```
%sql

-- Top high-value customers
SELECT
   CustomerKey,
   Name,
   SUM(SalesAmount) AS TotalSales,
   COUNT(SalesOrderNumber) AS TotalOrders
FROM
   wide_sales
WHERE
   HighValueOrder = true
GROUP BY
   CustomerKey, Name
ORDER BY
   TotalSales DESC
LIMIT 5;
```

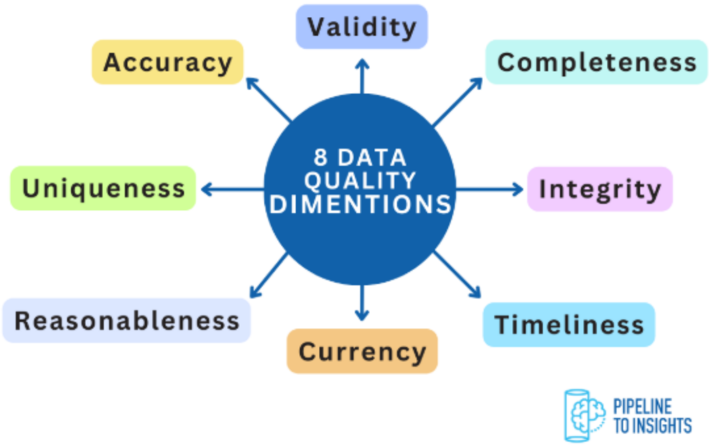
```
%sql

-- Sales by customer demographics
SELECT
   MaritalStatus,
   Gender,
   AVG(SalesAmount) AS AvgSales,
   COUNT(SalesOrderNumber) AS TotalOrders
FROM
   wide_sales
GROUP BY
   MaritalStatus, Gender
ORDER BY
   AvgSales DESC;
```

Data Quality

Data Quality is a major challenge for companies, especially with the rise of AI and other data driven products. According to Monte Carlo's recent survey, 68% of data leaders don't feel completely confident about the quality of their data.

But what is Data Quality? To put in simple terms, it's about how well data meets specific needs.

DQ has 8 dimensions we can measure across:



Picture is from pipeline to insights

Data Quality dimensions

Validity

Definition: Validity measures **how well** data aligns with the **expected** business logic.
Example: a valid phone number should include the country code and a nine-digit number, depending on the business requirements. If the business operates locally, including a country code might not be necessary. However, for international operations, such as in Australia and New Zealand, country codes become essential.

Completeness

Definition: Completeness ensures that **all necessary** data is **present**.
Example: a complete address should include the apartment or building number and street name to ensure accurate delivery and proper service to the customer.

Integrity

Definition: Integrity ensures that data is plausible and matches reality.
Example: if a company's address lists a location in New South Wales (NSW) but the company is based in Victoria, this creates a data integrity issue, as the information doesn’t match the real-world context.

Timeliness

Definition: Timeliness refers to how **quickly data is refreshed** according to **business expectations**.
Example: If data is expected to be refreshed within two weeks, any delay beyond that would be considered untimely and may impact decision-making.

Data Quality dimensions

Currency

Definition: Currency focuses on **how current the data is.**
Example: When the status changes, the approval date should reflect the most recent update. However, if the approval date is listed before the "won" date, it indicates a currency issue, as the data is not aligned with the correct timeline.

Reasonableness

Definition: Reasonableness ensures that data values are logical and meet expected business logic.
Example: In an HVAC materials purchase order, we should only see materials. An entry like "AC hire" doesn’t make sense here, as it should be captured under a different concept, such as a service or rental, rather than a purchase. This is tied to the business logic and expected data values.

Uniqueness

Definition: Uniqueness ensures that each data value is unique, which is crucial for merging data from different systems.
Example: unique IDs are essential to avoid duplicates.

Accuracy

Definition: Accuracy measures the correctness of data.
Example: In an HVAC-related job, a technician should spend a reasonable amount of time preparing, fixing, or replacing equipment. If we see a belt replacement job that took **20 hours**, this would indicate an accuracy issue, as the time spent is unusually high and doesn't reflect the actual time required for such a task.

Measuring Data Quality with Great Expectations

```
!pip install great_expectations

from great_expectations.core.batch import Batch
from great_expectations.validator.validator import Validator
from great_expectations.execution_engine.sparkdf_execution_engine import SparkDFExecutionEngine
from great_expectations import get_context

# 1. Create a context:
context = get_context()

# 2. Create a Spark Execution Engine
execution_engine = SparkDFExecutionEngine()

# 3. Create a Batch from the DataFrame
batch = Batch(data=wide_sales_df)

# 4. Create a Validator with the batch and execution engine
validator = Validator(execution_engine=execution_engine, batches=[batch])

# 5. Add expectations
validator.expect_column_values_to_not_be_null(
    column="SalesOrderNumber",
)
```

```
validator.expect_column_values_to_be_in_set(
    column="Gender",
    value_set=["Male", "Female"])

validator.expect_column_values_to_be_between(
    column="BirthDate",
    min_value="1999-01-01",
    max_value=None
)

# 6. Run validation and get results
results = validator.validate()

# 7. Process results
if results["success"]:
    print("All validations passed!")
else:
    print("Some validations failed.")
    for result in results["results"]:
        print(f"Expectation: {result['expectation_config']['type']}")
        print(f"Success: {result['success']}")
        if not result["success"]:
            print(f"Details: {result['result']}")
```

Measuring Data Quality with Great Expectations

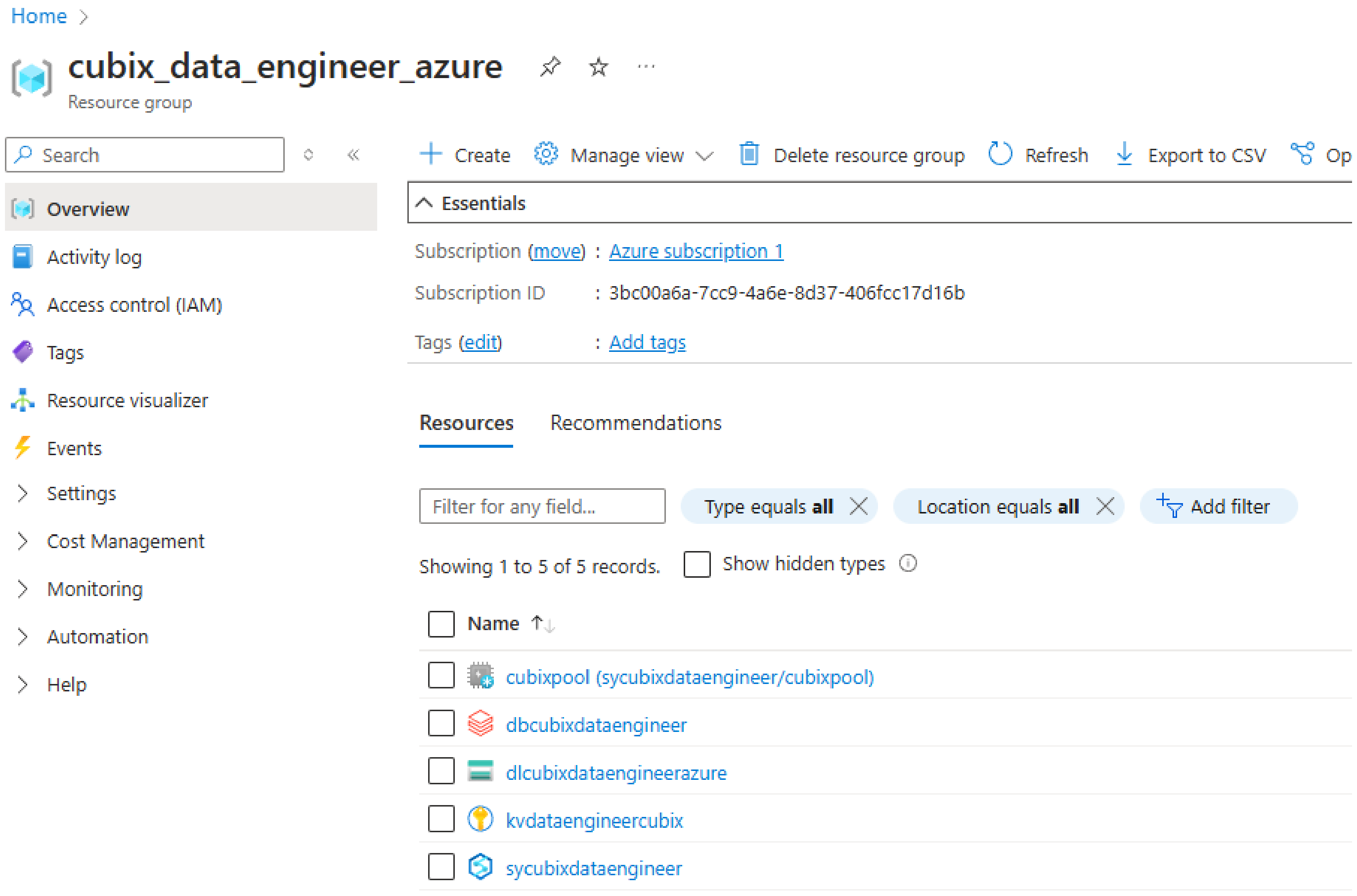
```
validation_results = results["results"]

results_data = [
    {
        "Expectation": res["expectation_config"]["type"],
        "Column": res["expectation_config"]["kwargs"].get("column"),
        "Success": res["success"],
        "Count": res["result"].get("element_count", "N/A"),
        "Failed Records Count": res["result"].get("unexpected_count", "N/A"),
        "Failed Records %": res["result"].get("unexpected_percent", "N/A"),
    }
    for res in validation_results
]

validation_results_df = spark.createDataFrame(results_data)

write_file_to_datalake(validation_results_df, "capstoneproject", "03_gold/wide_sales_validation_results", "parquet")

display(validation_results_df)
```

Putting it all together

Pipeline flow:

- Specify ingest job (batch 1-4)
- Bronze Layer
- Silver Layer
- Gold Layer
- DQ measurement

Deleting resources



Subscription (move) : Azure subscription 1

Subscription ID : 3bc00a6a-7cc9-4a6e-8d37-406fcc17d16b

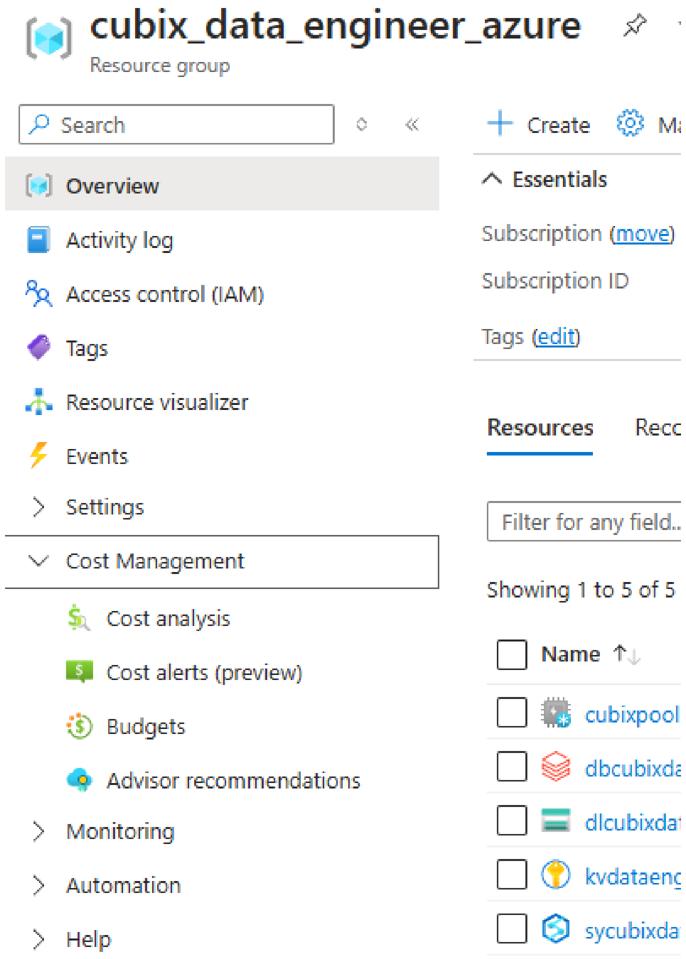
Tags (edit) : Add tags

Showing 1 to 5 of 5 records.

Name ↑↓
cubixpool (sycubixdataengineer/cubixpool)
dbcubixdataengineer
dlcubixdataengineerazure
kvdataengineercubix
sycubixdataengineer

Go to your resource group, and select “Delete resource group”.

Checking the costs



Clicking the **“Cost analysis”** under “Cost Management”, you will be directed to the resource group’s cost analysis, where you can see each resource’s cost.