ETL & Transformations

1. Data Extraction from ADLS

Mounting ADLS Gen2

```
configs = {
  "fs.azure.account.auth.type": "OAuth",
  "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
  "fs.azure.account.oauth2.client.id": "<service-principal-id>",
  "fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="key-vault",
key="adls-sp-secret"),
  "fs.azure.account.oauth2.client.endpoint":
"https://login.microsoftonline.com/<tenant-id>/oauth2/token"
}

dbutils.fs.mount(
  source = "abfss://container@storage.dfs.core.windows.net/",
  mount_point = "/mnt/adls",
  extra_configs = configs
)
```

Reading Different File Formats

```
# CSV
df_csv = spark.read.csv("/mnt/adls/raw/sales.csv", header=True, inferSchema=True)

# JSON
df_json = spark.read.json("/mnt/adls/raw/logs.json")

# Parquet
df_parquet = spark.read.parquet("/mnt/adls/raw/users.parquet")

# Delta
df_delta = spark.read.format("delta").load("/mnt/adls/processed/transactions")
```

Incremental Load Pattern

```
from pyspark.sql.functions import max

# Get last processed timestamp
last_processed = spark.sql("SELECT MAX(updated_at) FROM
bronze.sales").collect()[0][0]

# Read only new data
new_data = spark.read.parquet("/mnt/adls/raw/sales") \
  .filter(f"updated_at > '{last_processed}'")
```

2. Data Transformations

SQL Transformations

```
# Register temp view
df.createOrReplaceTempView("raw_data")

# Execute SQL
transformed_df = spark.sql("""
  SELECT
    user_id,
    COUNT(*) as transaction_count,
    SUM(amount) as total_spend
  FROM raw_data
  WHERE transaction_date > CURRENT_DATE() - INTERVAL 30 DAYS
  GROUP BY user_id
""")
```

Python DataFrame Transformations

```
from pyspark.sql.functions import col, when, to_date

transformed_df = (df
  .withColumn("clean_amount",
              when(col("amount") > 1000, 1000)
              .otherwise(col("amount")))
  .withColumn("transaction_date", to_date("timestamp"))
  .filter(col("status") == "COMPLETED")
  .groupBy("user_id", "transaction_date")
  .agg({"amount": "sum", "transaction_id": "count"})
```

Advanced Patterns

Slowly Changing Dimensions (SCD) Type 2

```
from delta.tables import *
from pyspark.sql.functions import current_timestamp

deltaTable = DeltaTable.forPath(spark, "/mnt/delta/customers")

updatesDF = spark.read.parquet("/mnt/adls/updates/customers")

deltaTable.alias("target").merge(
    updatesDF.alias("source"),
    "target.customer_id = source.customer_id AND target.current = true") \
  .whenMatchedUpdate(
    set = {
      "current": "false",
      "end_date": current_timestamp()
    }
  ) \
  .whenNotMatchedInsertAll() \
  .execute()
```

Data Quality Checks

```
from pyspark.sql.functions import lit

# Add data quality metrics
result_df = (transformed_df
  .withColumn("is_valid_amount", col("amount") > 0)
  .withColumn("is_valid_email", col("email").rlike(".+@.+\\..+"))
)

# Track metrics
valid_count = result_df.filter("is_valid_amount AND is_valid_email").count()
total_count = result_df.count()
dbutils.notebook.exit(f"Data Quality Score: {valid_count/total_count:.2%}")
```

3. Loading Data to Databases

JDBC Write to SQL Database

```
jdbc_url = "jdbc:sqlserver://server.database.windows.net:1433;database=sales_db"

(transformed_df.write
  .format("jdbc")
  .option("url", jdbc_url)
  .option("dbtable", "processed.transactions")
  .option("user", dbutils.secrets.get(scope="kv", key="db-user"))
  .option("password", dbutils.secrets.get(scope="kv", key="db-pwd"))
  .mode("append")
  .save())
```

Optimized Delta Lake Writes

python

```
(transformed_df.write
  .format("delta")
  .mode("overwrite")
  .option("replaceWhere", "transaction_date >= '2023-01-01'")
  .partitionBy("year", "month")
  .save("/mnt/delta/processed/transactions"))
```

CDC (Change Data Capture) Pattern

python

```
# Write changes only
changes_df = spark.read.format("delta") \
  .option("readChangeFeed", "true") \
  .option("startingVersion", 12) \
  .load("/mnt/delta/source_table")

changes_df.write \
  .format("jdbc") \
  .option("dbtable", "changes_log") \
  .save()
```

4. Complete ETL Pipeline Example

End-to-End Production Pipeline

```
# 1. Extract
raw_df = (spark.read
  .format("csv")
  .option("header", "true")
  .load("/mnt/adls/raw/sales_*.csv"))

# 2. Transform
from pyspark.sql.functions import col, to_date, year, month

transformed_df = (raw_df
  .filter(col("amount").isNotNull())
  .withColumn("transaction_date", to_date("timestamp"))
  .withColumn("year", year("transaction_date"))
  .withColumn("month", month("transaction_date"))
  .dropDuplicates(["transaction_id"]))

# 3. Load
(transformed_df.write
  .format("delta")
  .partitionBy("year", "month")
  .mode("append")
  .option("mergeSchema", "true")
  .save("/mnt/delta/silver/sales"))

# 4. Aggregate
spark.sql("""
  CREATE TABLE gold.sales_aggregates
  USING DELTA
  LOCATION '/mnt/delta/gold/sales_aggregates'
  AS
  SELECT
    customer_id,
    SUM(amount) as lifetime_value,
    COUNT(*) as transaction_count
  FROM silver.sales
  GROUP BY customer_id
""")
```

5. Performance Optimization

Partitioning Strategies

python

```
# Good for time-series
df.write.partitionBy("date").save(...)

# Good for high-cardinality dimensions
df.write.partitionBy("country", "region").save(...)

# Z-ordering for point queries
spark.sql("OPTIMIZE sales ZORDER BY (customer_id)")
```

Write Optimization

python

```
# Control file sizes
spark.conf.set("spark.sql.files.maxRecordsPerFile", 1000000)

# Parallel writes
df.repartition(16).write...  # Match to cluster cores

# Delta Lake optimizations
spark.conf.set("spark.databricks.delta.optimizeWrite.enabled", True)
```

6. Error Handling & Monitoring

ETL Monitoring Dashboard

sql

```
CREATE TABLE etl_audit (
  pipeline_name STRING,
  records_processed LONG,
  processing_time DOUBLE,
  success BOOLEAN,
  error_message STRING,
  timestamp TIMESTAMP
) USING DELTA;

-- Log after each run
INSERT INTO etl_audit VALUES (
  "daily_sales_etl",
  45231,
  125.7,
  true,
  NULL,
  CURRENT_TIMESTAMP()
);
```

Automatic Retry Logic

```
max_retries = 3
retry_count = 0

while retry_count < max_retries:
    try:
        dbutils.notebook.run("data_validation", 300)
        break
    except Exception as e:
        retry_count += 1
        if retry_count == max_retries:
            raise Exception(f"Failed after {max_retries} attempts: {str(e)}")
```

7. Advanced Patterns

Medallion Architecture Implementation

python

```
# Bronze (raw)
(spark.readStream
  .format("cloudFiles")
  .option("cloudFiles.format", "json")
  .load("/mnt/adls/raw/")
  .writeStream
  .format("delta")
  .option("checkpointLocation", "/checkpoints/bronze")
  .start("/mnt/delta/bronze/events"))

# Silver (validated)
(spark.readStream
  .format("delta")
  .load("/mnt/delta/bronze/events")
  .filter("amount > 0 AND user_id IS NOT NULL")
  .writeStream
  .format("delta")
  .option("checkpointLocation", "/checkpoints/silver")
  .start("/mnt/delta/silver/events"))

# Gold (aggregated)
(spark.readStream
  .format("delta")
  .load("/mnt/delta/silver/events")
  .groupBy(window("timestamp", "1 day"), "user_id")
  .agg(sum("amount").alias("daily_spend"))
  .writeStream
  .format("delta")
  .option("checkpointLocation", "/checkpoints/gold")
  .start("/mnt/delta/gold/user_spending"))
```

Data Vault Modeling

python

```
# Hubs
spark.sql("""
  CREATE TABLE vault.hub_customer
  USING DELTA
  AS
  SELECT
    md5(customer_id) AS customer_key,
    customer_id,
    load_timestamp
  FROM staging.customers
""")

# Links
spark.sql("""
  CREATE TABLE vault.link_transaction
  USING DELTA
  AS
  SELECT
    md5(concat(t.transaction_id, c.customer_id)) AS transaction_customer_key,
    md5(t.transaction_id) AS transaction_key,
    md5(c.customer_id) AS customer_key,
    CURRENT_TIMESTAMP() AS load_timestamp
  FROM staging.transactions t
  JOIN staging.customers c ON t.customer_id = c.customer_id
""")
```

8. CI/CD for ETL Pipelines

Terraform Deployment

hcl

```
resource "databricks_job" "nightly_etl" {
  name = "Nightly ETL Pipeline"

  notebook_task {
    notebook_path = "/ETL/nightly_processing"
  }

  schedule {
    quartz_cron_expression = "0 0 22 * * ?"
  }
}

resource "databricks_pipeline" "customer_cdc" {
  name = "Customer CDC Pipeline"

  configuration = {
    pipeline_type = "CDC"
    source = "/mnt/adls/raw/customers"
    target = "/mnt/delta/silver/customers"
  }
}
```

Unit Testing Framework

python

```
import unittest
from pyspark.sql import SparkSession

class TestETLTransformations(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.spark = SparkSession.builder.getOrCreate()
        test_data = [(1, "valid@email.com"), (2, "invalid")]
        cls.test_df = cls.spark.createDataFrame(test_data, ["id", "email"])

    def test_email_validation(self):
        from transforms import validate_emails
        result = validate_emails(self.test_df)
        self.assertEqual(result.filter("is_valid_email").count(), 1)

if __name__ == "__main__":
    unittest.main()
```

9. Security & Governance

Column-Level Encryption

python

```
from pyspark.sql.functions import sha2

secure_df = df.withColumn("ssn_hashed", sha2(col("ssn"), 256))
```

Dynamic View Permissions

sql

```
CREATE VIEW finance.transactions AS
SELECT * FROM raw.transactions
WHERE department = current_user();
```

10. Performance Benchmarking

ETL Metrics Tracking

python

```
# Start timer
start_time = time.time()

# Run transformation
result = transform_large_dataset()

# Calculate metrics
duration = time.time() - start_time
records_per_second = result.count() / duration

# Log metrics
spark.sql(f"""
  INSERT INTO etl_metrics VALUES (
    'customer_etl',
    {duration},
    {records_per_second},
    CURRENT_TIMESTAMP()
  )
""")
```