**Lab Title: Advanced Data Flow Techniques in Azure Data Factory**

**Objective:**

Build an optimized, reusable Data Flow in ADF using Flowlets, Derived Column, Conditional Split, Assert-Row for DQ, Sink partitioning, and debugging. Learn performance tuning best practices.

**Pre-requisites:**

* Azure Subscription
* Azure Data Factory (V2) instance
* Azure Storage Account with a container for input and output
* Sample CSV file uploaded to a Blob container (input/employees.csv)

**Sample Data:**

EmployeeID,Name,Department,Salary,JoinDate

1001,John Doe,IT,90000,2018-01-15

1002,Jane Smith,HR,70000,2019-03-22

1003,Bob Lee,IT,,2020-07-12

1004,Alice Brown,Finance,82000,2017-10-02

1005,Mark Taylor,Sales,abc,2021-12-01

**Step-by-Step Lab Instructions**

**Step 1: Create a Data Flow**

1. Go to your ADF instance.
2. In the Author tab, click **+ → Data Flow**.
3. Name it: EmployeeDataFlow.

**Step 2: Add a Source**

1. Add a **Source** transformation.
2. Choose or create a dataset pointing to input/employees.csv.
3. Enable **First row as header**.
4. Set import schema = "From Connection/store".

**Step 3: Add a Derived Column**

1. Add a **Derived Column** transformation after Source.
2. Rename it: CleanSalary.
3. Add a new column: NormalizedSalary.
4. Expression:

iif(isNull(toInteger(Salary)), -1, toInteger(Salary))

1. Add another column: JoinYear → year(JoinDate).

**Step 4: Add a Conditional Split**

1. Add **Conditional Split** transformation after CleanSalary.
2. Name it: DeptFilter.
3. Create two streams:
   * IT\_Department: Department == 'IT'
   * Others: true

**Step 5: Create a Flowlet for Reuse**

1. In the ADF Author pane, click **+ → Flowlet**.
2. Name it: SalaryNormalizationFlowlet.
3. Add:
   * Input
   * Derived Column with logic:

NormalizedSalary = iif(isNull(toInteger(Salary)), -1, toInteger(Salary))

* + Output

1. Publish and close Flowlet.

**Using Flowlet:**

* Go back to the main Data Flow.
* Delete existing derived column CleanSalary.
* Add Flowlet instead and select SalaryNormalizationFlowlet.

**Step 6: Assert-Row Data Quality Transform**

1. Add an **Assert** transformation after Flowlet.
2. Name: CheckSalaryPositive.
3. Condition:
   * Rule: NormalizedSalary > 0
   * Assertion type: Assert → **Error out invalid rows**

**Step 7: Sink with Partitioning and Staging**

1. Add a **Sink** for IT Department.
2. Select Output Dataset → Point to output container /output/IT/
3. Enable **Staging**:
   * Choose Linked Service with staging account.
   * Set staging location: /staging/IT/
4. Under Optimization:
   * Set **Partition by**: Column = JoinYear

**Step 8: Debug & Data Preview**

1. Enable **Data Flow Debug**.
2. Preview each step:
   * Observe transformations
   * Look for salary normalization
   * Check split logic
   * Verify Assert-Row errors

**Step 9: Performance Tuning**

**a. Use Schema Drift:**  
If schema is dynamic, enable schema drift in Source and Sink.

**b. Optimize Partitioning:**

* Sink partitioning by JoinYear helps with parallel write.

**c. Minimize Data Shuffling:**

* Avoid unnecessary Derived Columns before filtering.

**d. Monitor Debug Session Logs:**

* Use Data Preview pane to find bottlenecks (e.g., large shuffles).

**e. Use Flowlets:**

* Reduce duplication and improve modularity.

**Optional Enhancement: Add Logging**

1. After Sink, add an extra sink logging to a logging.csv file.
2. Write metadata like fileName, recordCount, pipelineRunId.

**Validation & Output:**

* Go to the output container and verify partitioned output files by JoinYear.
* Test with a row that has a non-numeric salary → it should be caught by Assert-Row.
* Reuse Flowlet in another pipeline to confirm modularity.