**Lab 1: Transformation Types – Join, Aggregate, Conditional Split, Derived Column**

**Objective:** Learn to design Mapping Data Flows with common transformation types.

**Steps:**

1. **Create Source Datasets**:
   * Employee\_CSV\_Dataset → EmployeeID, DepartmentID, Salary.
   * Department\_JSON\_Dataset → DepartmentID, DepartmentName.
2. **Add Join Transformation**:
   * Join on DepartmentID.
   * Join type = Inner.
3. **Add Aggregate Transformation**:
   * Group by DepartmentName.
   * Aggregates:
     + AvgSalary = avg(Salary)
     + EmpCount = count(EmployeeID)
4. **Add Conditional Split Transformation**:
   * Condition: Salary > 100000 → HighSalary.
   * Else → RegularSalary.
5. **Add Derived Column Transformation**:
   * AnnualBonus = Salary \* 0.10.
6. **Add Sink Transformations**:
   * Write results into SQL tables:
     + DeptSalarySummary (aggregates).
     + HighSalaryEmployees.
     + RegularSalaryEmployees.
7. Debug, Validate, Publish, and Run Pipeline.

**Outcome:** Build complex ETL flows with joins, aggregations, splits, and derived columns.

**Lab 2: Parameterization in Pipelines & Datasets**

**Objective:** Make pipelines reusable by passing dynamic values.

**Steps:**

1. Create **Pipeline Parameter**: FileName.
2. In dataset, define FileNameParam.
3. Configure file path expression:
   * Example: @dataset().FileNameParam.
4. In Copy Activity, map:
   * Pipeline parameter → Dataset parameter.
   * @pipeline().parameters.FileName.
5. Trigger pipeline multiple times with different values:
   * employee\_jan.csv
   * employee\_feb.csv

**Outcome:** Single pipeline processes multiple files dynamically.

**Lab 3: Control Flow Activities – If, ForEach, Until**

**Objective:** Implement control flow for conditional execution and loops.

**Steps:**

1. **If Condition Activity**:
   * Expression: @equals(variables('RowCount'),0).
   * If true → Send notification.
   * If false → Continue ETL.
2. **ForEach Activity**:
   * Items: @activity('Get Metadata1').output.childItems.
   * Inside loop: Copy Data activity → Process each file.
3. **Until Activity**:
   * Condition: @greaterOrEquals(variables('RetryCount'),3).
   * Inside: Retry copy with increment counter.

**Outcome:** Pipelines behave like **program logic** with loops, retries, and branching.

**Lab 4: Pipeline Triggers – Schedule, Tumbling Window, Event-based**

**Objective:** Automate pipeline execution.

**Steps:**

1. **Schedule Trigger**:
   * Run pipeline every day at 2 AM.
   * Example: Refresh daily reports.
2. **Tumbling Window Trigger**:
   * Interval = 1 hour.
   * Window dependency = true.
   * Example: Hourly incremental loads.
3. **Event-based Trigger**:
   * Event Grid → Blob Storage → New file event.
   * Trigger pipeline immediately when employee\_data.csv is uploaded.

**Outcome:** Understand automation via **time-driven, window-based, and real-time triggers**.

**Lab 5: Incremental Data Load – Watermarking Strategy**

**Objective:** Implement incremental loading to avoid full reloads.

**Steps:**

1. Ensure SQL source table has LastModifiedDate.
2. Create **Pipeline Parameter**: LastWatermark.
3. In source dataset, parameterize query:
4. SELECT \*
5. FROM Employee
6. WHERE LastModifiedDate > '@{pipeline().parameters.LastWatermark}'
7. Copy incremental rows to ADLS/incremental\_load/.
8. After successful load, update LastWatermark using Lookup + Set Variable activity.

**Outcome:** Only new/updated records are processed in each run.

**Lab 6: Git Integration & CI/CD**

**Objective:** Enable version control for ADF pipelines.

**Steps:**

1. Open **ADF Studio → Manage hub → Git Configuration**.
2. Select Azure DevOps or GitHub.
3. Provide:
   * Repo URL.
   * Collaboration branch.
   * Root folder path.
4. Save configuration.
5. Make pipeline change → Save All → Commit with message.
6. Verify changes appear in Git repo.

**Outcome:** Pipelines are **source-controlled** and ready for CI/CD.

**Lab 7: Logging & Monitoring Best Practices**

**Objective:** Learn how to monitor pipelines and configure alerts.

**Steps:**

1. Open **Monitor hub** → Check pipeline run history.
2. Enable **Diagnostic Settings**:
   * Send logs to Log Analytics.
   * Export metrics to Azure Monitor.
3. Configure **alerts** (example: failure alerts to email/MS Teams).
4. Implement **retry policies** for Copy Activities.
5. Add **custom logging** (e.g., output row counts with Set Variable).

**Outcome:** Proactive monitoring ensures reliability and observability.