**Implementing SCD Type 2 in Azure Data Factory**

Implementing a Slowly Changing Dimension (SCD) Type 2 in Azure Data Factory (ADF) allows for the preservation of historical data by creating new records for changes in source data. This guide provides a detailed, step-by-step process to build an SCD Type 2 solution using Mapping Data Flows in ADF, tailored to a source with id and name columns.

## 1. Prerequisites

Before you begin, ensure you have the following:

* An active Azure subscription.
* An Azure Data Factory instance.
* An Azure SQL Database or other supported data store to serve as your dimension table.
* A source dataset. For this example, we will use a source with two records:
  + id: 1, name: king
  + id: 2, name: queen

**2. Setting up the Dimension Table**

First, create a dimension table in your Azure SQL Database. This table will store the historical and current records. In addition to the source columns (id and name), you'll need columns to manage the history:

* **SurrogateKey**: A unique identifier for each record in the dimension table (typically an auto-incrementing integer).
* **StartDate**: The date from which the record is valid.
* **EndDate**: The date until which the record was valid. A null or future date (e.g., '9999-12-31') indicates the current record.
* **IsActive**: A boolean or integer flag to easily identify the current record (e.g., 1 for active, 0 for inactive).

Here's a sample SQL script to create the dimension table:

CREATE TABLE Dim\_Customer (

SurrogateKey INT IDENTITY(1,1) PRIMARY KEY,

id INT,

name VARCHAR(255),

StartDate DATETIME,

EndDate DATETIME,

IsActive BIT

);

**3. Creating the Azure Data Factory Pipeline and Data Flow**

1. **Create a new pipeline** in your Azure Data Factory.
2. Within the new pipeline, add a **Data Flow** activity to the canvas.
3. Open the Data Flow to begin configuring the transformations.

**4. Configuring the Mapping Data Flow**

The core of the SCD Type 2 implementation lies within the Mapping Data Flow. Here’s a breakdown of the necessary transformations:

**Step 4.1: Add Source Transformations**

You will need two sources:

* **Source for New Data**: Add a source transformation that points to your incoming data (e.g., a CSV file in Azure Blob Storage containing the id and name columns).
* **Source for Existing Dimension Data**: Add another source transformation that connects to your Dim\_Customer table in Azure SQL Database.

**Step 4.2: Lookup Existing Records**

1. Add a **Lookup** transformation after your new data source.
2. **Left Stream**: This will be your new data source.
3. **Right Stream**: This will be your existing Dim\_Customer source.
4. **Lookup conditions**: Match the id column from the new data source with the id column from the dimension table. This will check if the incoming records already exist in the dimension table.

**Step 4.3: Add a Derived Column for Hash Comparison (Optional but Recommended) -** To efficiently detect changes in multiple columns, it's a best practice to create a hash column.

1. Add a **Derived Column** transformation after the new data source (before the Lookup).
2. Create a new column, for example, NewDataHash, and use an expression like md5(toString(name)) to create a hash of the columns you want to track for changes.
3. Similarly, add a Derived Column transformation after the Dim\_Customer source to create a hash of the corresponding columns, for example, ExistingDataHash using md5(toString(name)).

* Your lookup condition would then also include comparing these hash columns to identify records where the name has changed.

**Step 4.4: Conditional Split for New, Changed, and Unchanged Records**

Add a **Conditional Split** transformation after the Lookup to route the data based on whether a record is new, has changed, or remains unchanged.

* **New Records**: Use a condition like isNull(Dim\_Customer@id) to identify rows from the new data source that did not have a match in the dimension table.
* **Changed Records**: Use a condition that checks if the record exists but the data has changed. If using a hash, the condition would be something like !isNull(Dim\_Customer@id) && NewDataHash != ExistingDataHash.
* **Unchanged Records**: This will be the default stream for records that do not meet the "New" or "Changed" criteria.

**Step 4.5: Handling New Records**

For the "New Records" stream from the Conditional Split:

1. a **Derived Column** transformation.
2. Create the following new columns:
   * StartDate: currentUTC()
   * EndDate: toTimestamp('9999-12-31 23:59:59.000', 'yyyy-MM-dd HH:mm:ss.SSS')
   * IsActive: 1

**Step 4.6: Handling Changed Records -** This requires two separate streams to update the old record and insert the new one.

**Stream 1: Expire Old Records**

1. From the "Changed Records" stream of the Conditional Split, add a **Derived Column** transformation.
2. Create the following columns to mark the existing record as inactive:
   * EndDate: currentUTC()
   * IsActive: 0
3. Add an **Alter Row** transformation. Set the update condition to true(). This will flag these rows for an update in the sink.
4. Add a **Sink** transformation. In the settings, allow "Update" and specify the key column as SurrogateKey.

**Stream 2: Insert New Version of Changed Records**

1. From the "Changed Records" stream of the Conditional Split, add another **Derived Column** transformation.
2. Create the new version of the record with the updated name and new validity columns:
   * StartDate: currentUTC()
   * EndDate: toTimestamp('9999-12-31 23:59:59.000', 'yyyy-MM-dd HH:mm:ss.SSS')
   * IsActive: 1

**Step 4.7: Union New and Newly Versioned Records -** Add a **Union** transformation to combine the "New Records" stream and the "Insert New Version of Changed Records" stream.

**Step 4.8: Final Sink for Inserts**

1. Add a **Sink** transformation after the Union transformation.
2. Configure this sink to point to your Dim\_Customer table.
3. In the settings, allow "Insert".
4. In the Mapping tab, ensure the source columns are correctly mapped to the destination columns. The SurrogateKey will be generated by the database.

**5. Executing and Monitoring the Pipeline**

Once the Data Flow is configured, return to the pipeline view. You can now debug the pipeline to test the logic with a small dataset. After successful debugging, you can trigger a pipeline run to process the full dataset. Monitor the pipeline run in the "Monitor" tab of your Azure Data Factory to ensure it completes successfully.

By following these steps, you can effectively implement an SCD Type 2 solution in Azure Data Factory, ensuring that you maintain a complete history of your dimension data. This allows for powerful historical analysis and reporting.