TASK 2

SCHEDULED TRIGGER

(The code for the procedure and the table creation is present in ScheduledTrigger folder)

When you create a *schedule trigger*, you specify a schedule like a start date, recurrence, or end date for the trigger and associate it with a pipeline. Pipelines and triggers have a many-to-many relationship. Multiple triggers can kick off a single pipeline. A single trigger can kick off multiple pipelines.

The task performed in the following scheduled trigger task:

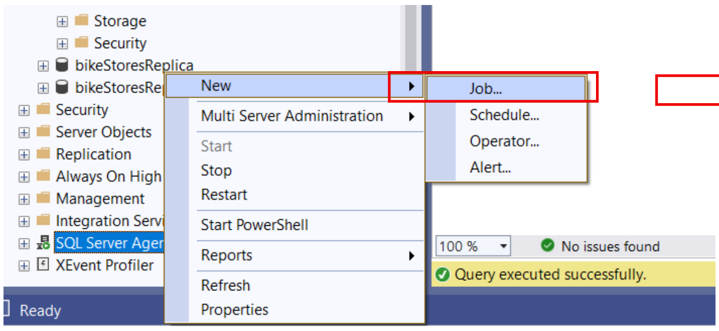
The source table is **production.categories** present in the **bikeStores** database and the destination table is **pr.categories** present in **bikeStoresRep** database. The destination table is updated with the data everyday at specific time from the updated source table.

STEP 1: Create Procedure

```
USE [bikeStoresRep];

GO

CREATE OR ALTER PROCEDURE CopyData

AS

BEGIN

   INSERT INTO pr.categories (category_name)

   SELECT category_name

   FROM bikeStores.production.categories bsc

   WHERE bsc.category_id NOT IN (

      SELECT category_id from pr.categories

   );

END;
```

STEP 2:

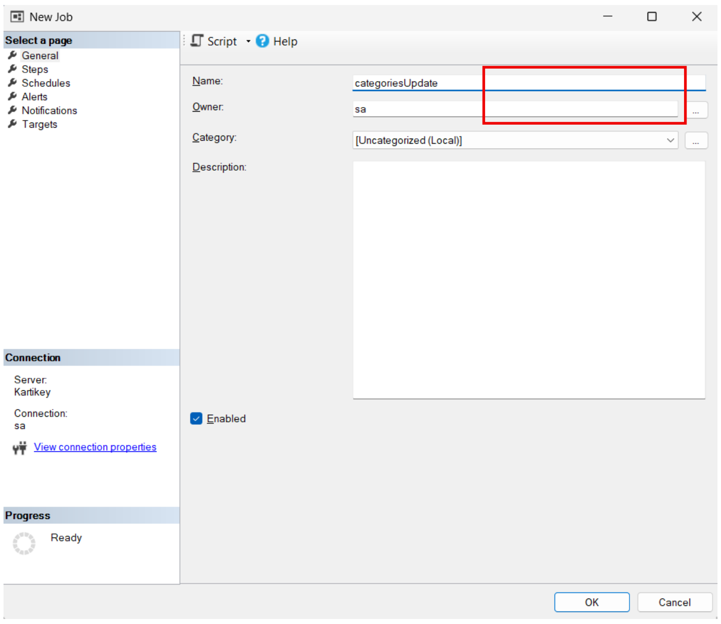
1. Create Server Job



No issues found

Query executed successfully.

2. Set Job Name and Owner



Name: categoriesUpdate

Owner: sa

Category: [Uncategorized (Local)]

[✓] Enabled

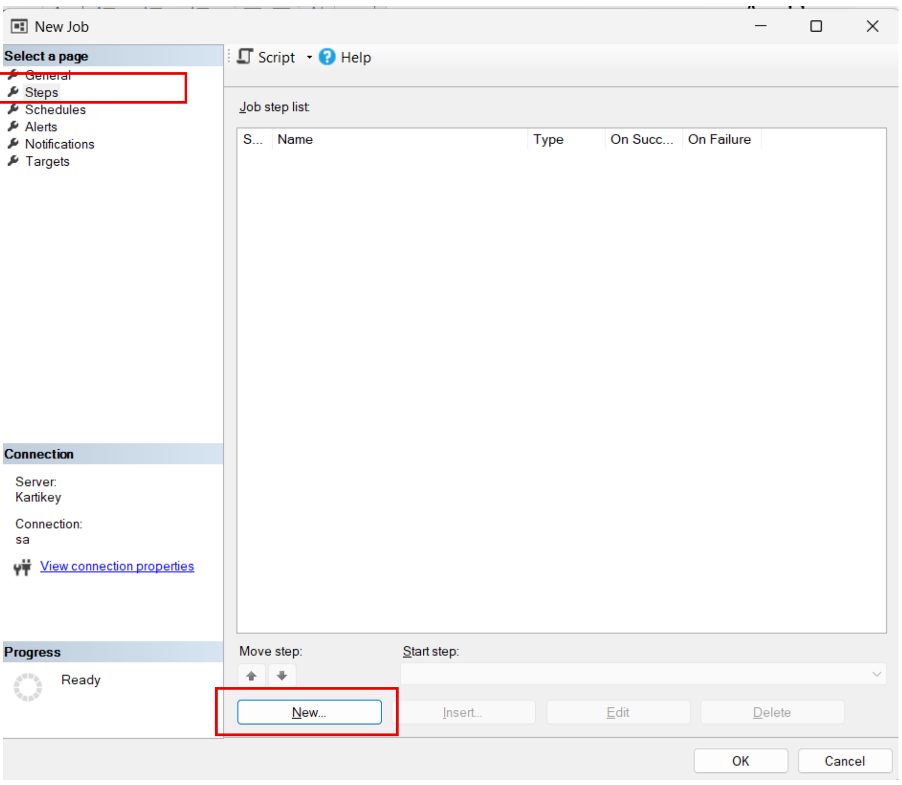
Connection

Server: Kartikey

Connection: sa

Ready

3. Set new Step



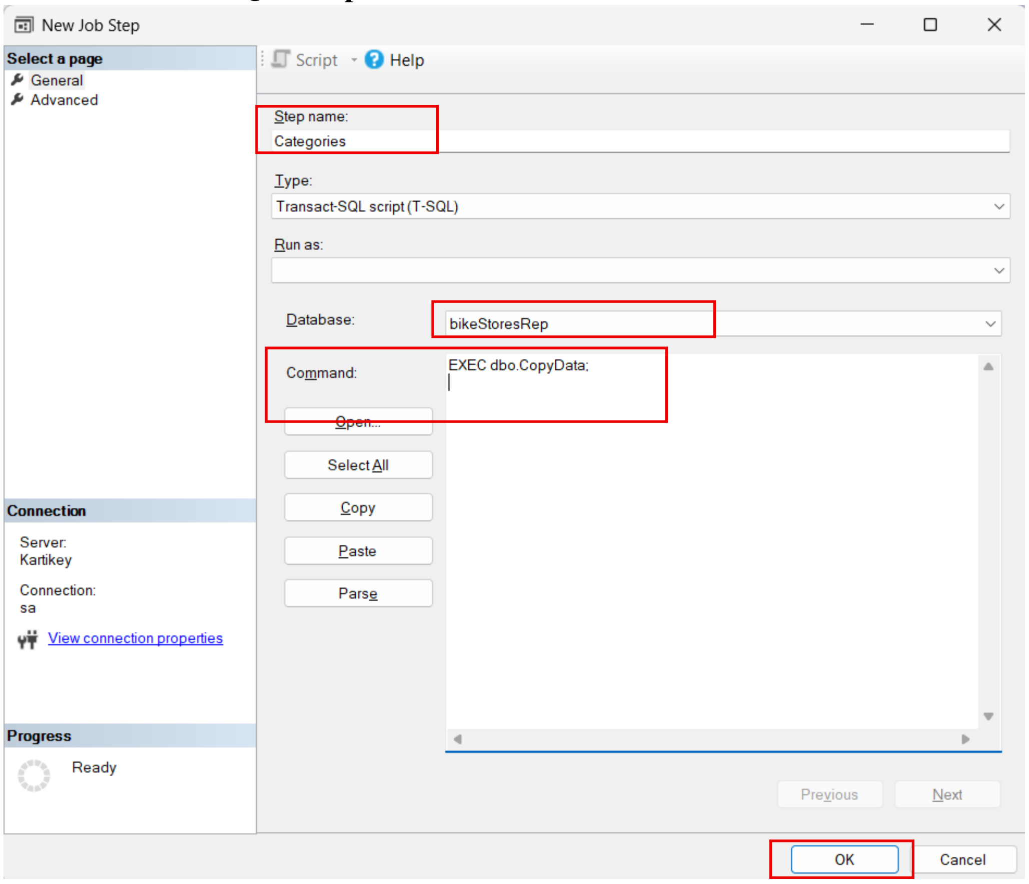
Connection

Server: Kartikey

Connection: sa

Ready

4. Write Transact SQL Script



New Job Step

Select a page
- General
- Advanced

Step name:
Categories

Type:
Transact-SQL script (T-SQL)

Run as:

Database: bikeStoresRep

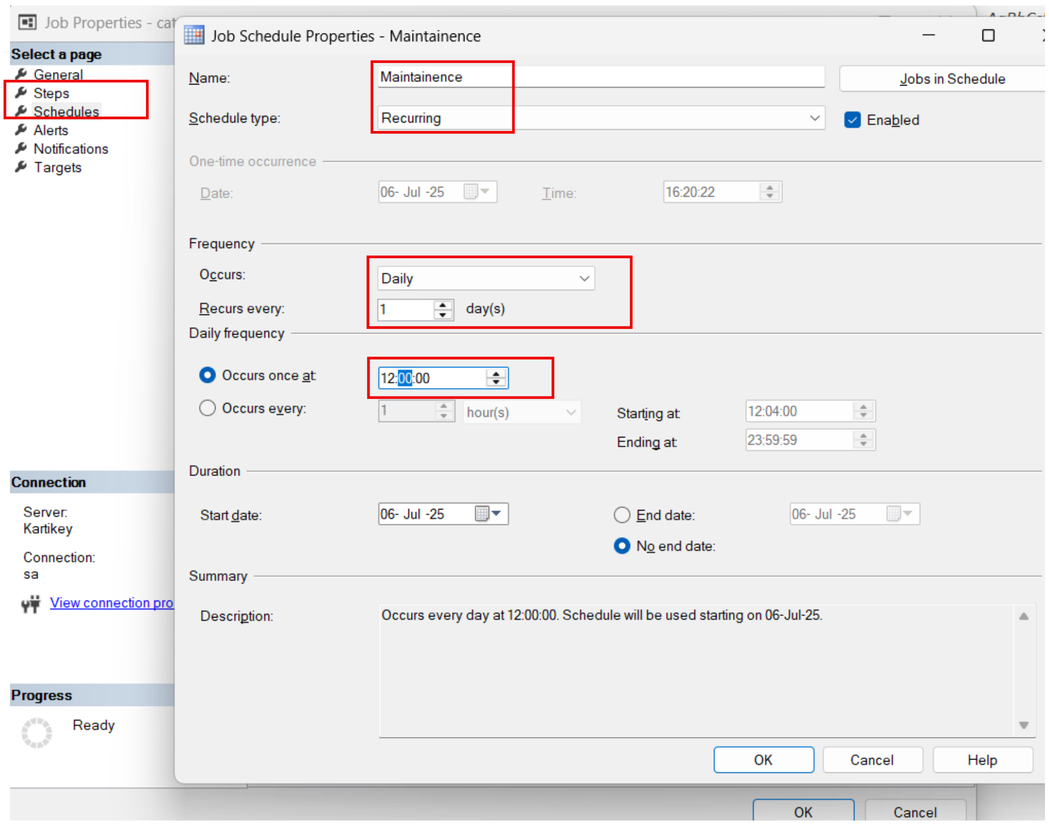
Command:

```
EXEC dbo.CopyData;
```

Connection
Server:
Kartikey
Connection:
sa

Progress
Ready

5. Create Schedule



Job Schedule Properties - Maintainence

Name: Maintainence

Schedule type: Recurring

Enabled ✓

One-time occurrence
Date: 06- Jul -25
Time: 16:20:22

Frequency
Occurs: Daily
Recurs every: 1 day(s)

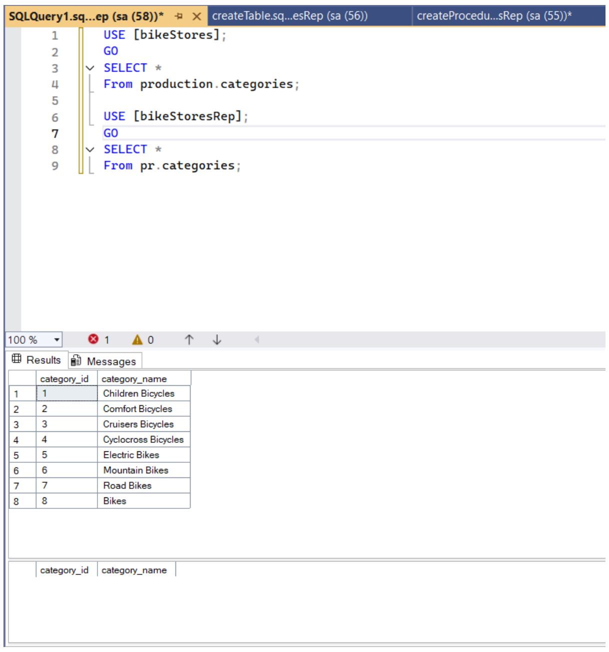
Daily frequency
[● Occurs once at: 12:00:00]
○ Occurs every: 1 hour(s)
Starting at: 12:04:00
Ending at: 23:59:59

Duration
Start date: 06- Jul -25
○ End date: 06- Jul -25
[● No end date:]

Summary
Description: Occurs every day at 12:00:00. Schedule will be used starting on 06-Jul-25.

OUTPUT:

Before

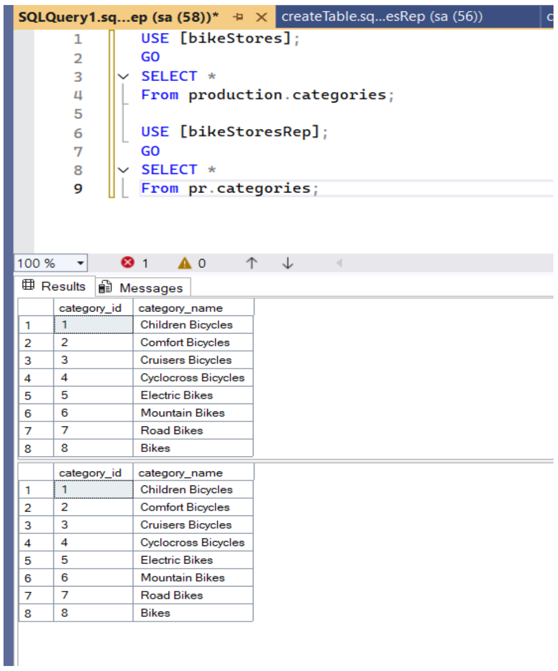


```
USE [bikeStores];
GO
SELECT *
From production.categories;

USE [bikeStoresRep];
GO
SELECT *
From pr.categories;
```

	category_id	category_name
1	1	Children Bicycles
2	2	Comfort Bicycles
3	3	Cruisers Bicycles
4	4	Cyclocross Bicycles
5	5	Electric Bikes
6	6	Mountain Bikes
7	7	Road Bikes
8	8	Bikes

category_id	category_name

After

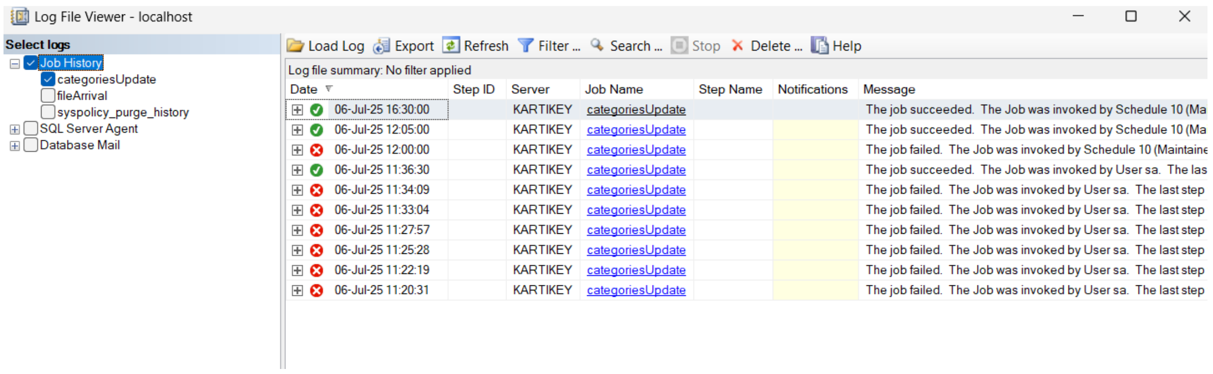


```
USE [bikeStores];
GO
SELECT *
From production.categories;

USE [bikeStoresRep];
GO
SELECT *
From pr.categories;
```

	category_id	category_name
1	1	Children Bicycles
2	2	Comfort Bicycles
3	3	Cruisers Bicycles
4	4	Cyclocross Bicycles
5	5	Electric Bikes
6	6	Mountain Bikes
7	7	Road Bikes
8	8	Bikes

	category_id	category_name
1	1	Children Bicycles
2	2	Comfort Bicycles
3	3	Cruisers Bicycles
4	4	Cyclocross Bicycles
5	5	Electric Bikes
6	6	Mountain Bikes
7	7	Road Bikes
8	8	Bikes

LOGS:



Log File Viewer - localhost

Select logs
- Job History
  - categoriesUpdate
  - fileArrival
  - syspolicy_purge_history
- SQL Server Agent
- Database Mail

Log file summary: No filter applied

Date	Step ID	Server	Job Name	Step Name	Notifications	Message
✓ 06-Jul-25 16:30:00		KARTIKEY	categoriesUpdate			The job succeeded. The Job was invoked by Schedule 10 (Ma
✓ 06-Jul-25 12:05:00		KARTIKEY	categoriesUpdate			The job succeeded. The Job was invoked by Schedule 10 (Ma
✗ 06-Jul-25 12:00:00		KARTIKEY	categoriesUpdate			The job failed. The Job was invoked by Schedule 10 (Maintaine
✓ 06-Jul-25 11:36:30		KARTIKEY	categoriesUpdate			The job succeeded. The Job was invoked by User sa. The las
✗ 06-Jul-25 11:34:09		KARTIKEY	categoriesUpdate			The job failed. The Job was invoked by User sa. The last step
✗ 06-Jul-25 11:33:04		KARTIKEY	categoriesUpdate			The job failed. The Job was invoked by User sa. The last step
✗ 06-Jul-25 11:27:57		KARTIKEY	categoriesUpdate			The job failed. The Job was invoked by User sa. The last step
✗ 06-Jul-25 11:25:28		KARTIKEY	categoriesUpdate			The job failed. The Job was invoked by User sa. The last step
✗ 06-Jul-25 11:22:19		KARTIKEY	categoriesUpdate			The job failed. The Job was invoked by User sa. The last step
✗ 06-Jul-25 11:20:31		KARTIKEY	categoriesUpdate			The job failed. The Job was invoked by User sa. The last step

There are several unsuccessful execution, as I was learning at the time. The correct exection runs, i.e, top 2, are the successful runs conducted at 4:30pm and 12:05pm.

The schedule was modified accordingly.

EVENT TRIGGER

(The codes are present in their respective folders)

Event Triggers are special types of database triggers that automatically execute or fire in response to certain events or database-wide events, such as:

- Creating, altering, or dropping tables, views, or other database objects
- User login or logout (in some DBMS)
- Transactions starting or committing (in some systems)
- Data Manipulation Language events like INSERT, UPDATE, or DELETE on tables.

1. **Database Changes**

   Table which will be tracked for changes is **pr.categories** present in **bikeStoresRep** database. If any record is inserted or deleted from the table, the changes are audited or logged in the **pr.category_audits** table automatically using trigger.

   STEP 1: Create audit table

```
CREATE TABLE pr.category_audits(
   change_id INT IDENTITY PRIMARY KEY,
   category_id INT NOT NULL,
   category_name VARCHAR(255) NOT NULL,
   updated_at DATETIME NOT NULL,
   operation CHAR(3) NOT NULL,
   CHECK(operation = 'INS' or operation='DEL')
);
```

   STEP 2: Create trigger

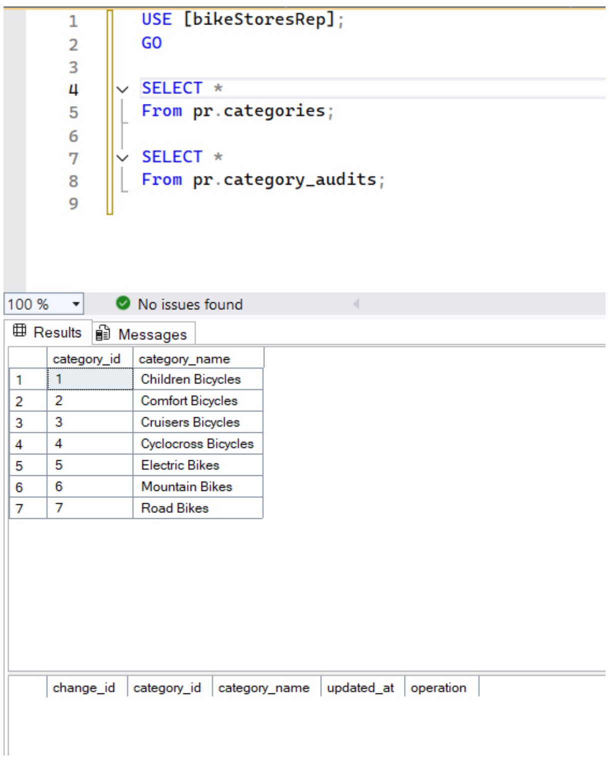
```
CREATE TRIGGER
        pr.trg_category_audit
ON
        pr.categories
AFTER INSERT, DELETE
AS
BEGIN
        SET NOCOUNT ON;
        INSERT INTO pr.category_audits(
                category_id,
                category_name,
                updated_at,
                operation
        )
        SELECT
                i.category_id,
                category_name,
                GETDATE(),
                'INS'
        FROM
                inserted as i
        UNION ALL
        SELECT
                d.category_id,
                category_name,
                GETDATE(),
                'DEL'
        FROM
                deleted as d;
END
```

OUTPUT:

Insert Record

```
INSERT INTO pr.categories(
    category_name
)
VALUES (
    'Test product'
);
```

Before



```
USE [bikeStoresRep];
GO

SELECT *
From pr.categories;

SELECT *
From pr.category_audits;
```

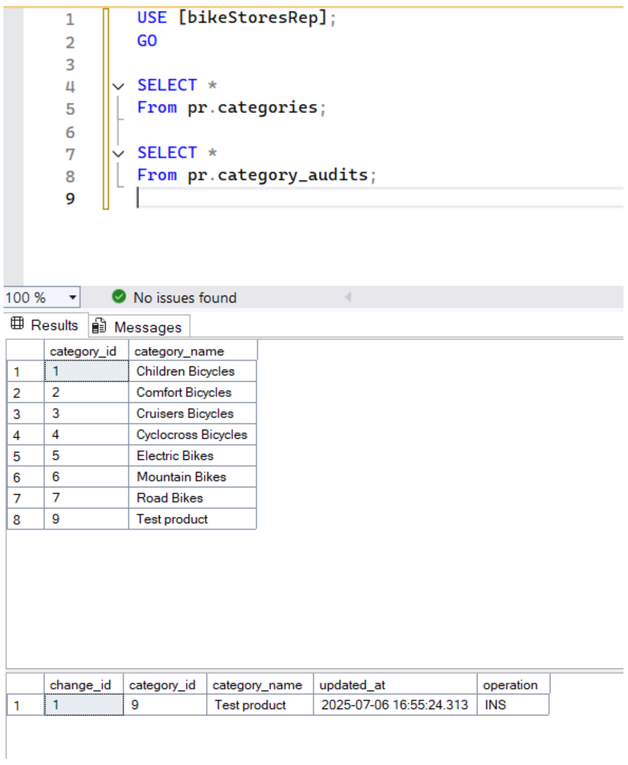
100 % No issues found

Results Messages

	category_id	category_name
1	1	Children Bicycles
2	2	Comfort Bicycles
3	3	Cruisers Bicycles
4	4	Cyclocross Bicycles
5	5	Electric Bikes
6	6	Mountain Bikes
7	7	Road Bikes

change_id	category_id	category_name	updated_at	operation

After



```
USE [bikeStoresRep];
GO

SELECT *
From pr.categories;

SELECT *
From pr.category_audits;
```

100 % No issues found

Results Messages

	category_id	category_name
1	1	Children Bicycles
2	2	Comfort Bicycles
3	3	Cruisers Bicycles
4	4	Cyclocross Bicycles
5	5	Electric Bikes
6	6	Mountain Bikes
7	7	Road Bikes
8	9	Test product

	change_id	category_id	category_name	updated_at	operation
1	1	9	Test product	2025-07-06 16:55:24.313	INS

2. File Arrival

A **powerchell** script listens for **.csv** files in the **fileArrival** folder. When new csv file is identified, the shell runs the sqlcmd command, connects to the server and runs the job to add the data to the MS SQL Server.

STEP 1: Create tables

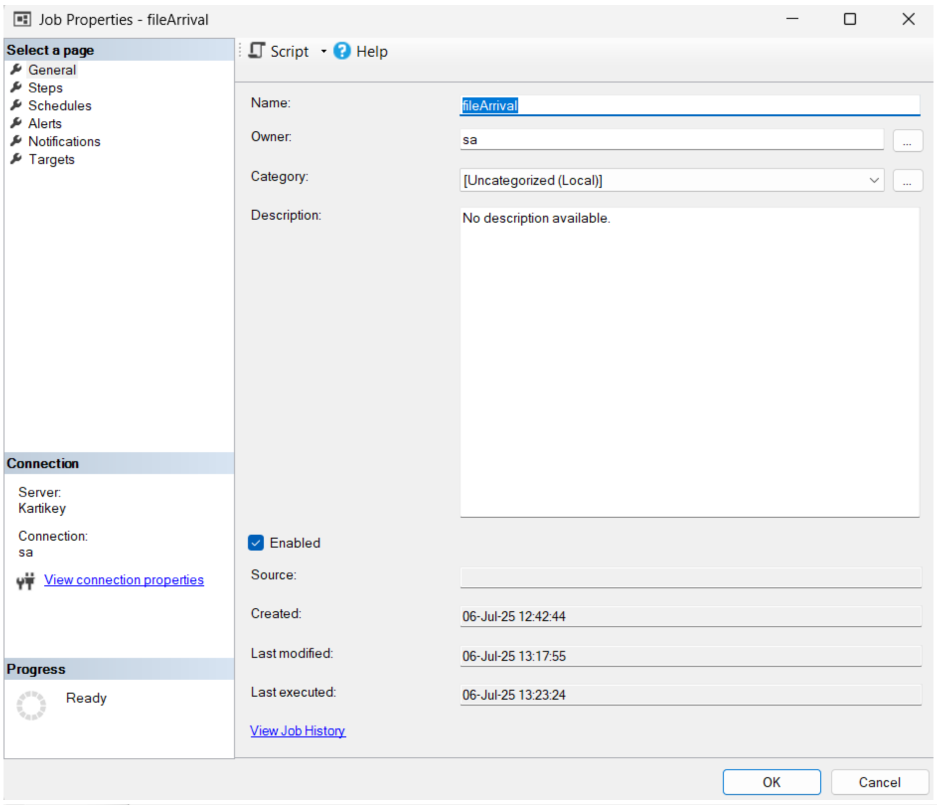
```
CREATE SCHEMA fl;
GO

IF OBJECT_ID('fl.stores', 'U') IS NULL
BEGIN
   CREATE TABLE fl.stores (
         store_id INT IDENTITY (1, 1) PRIMARY KEY,
            store_name VARCHAR (255) NOT NULL,
            phone VARCHAR (25),
            email VARCHAR (255),
            street VARCHAR (255),
            city VARCHAR (255),
            state VARCHAR (10),
            zip_code VARCHAR (5)
   );
END;
GO

IF OBJECT_ID('dbo.FileQueue', 'U') IS NULL
BEGIN
   CREATE TABLE fl.FileQueue (
      id INT IDENTITY(1,1) PRIMARY KEY,
      file_path NVARCHAR(500),
      processed BIT DEFAULT 0,
      created_at DATETIME DEFAULT GETDATE()
   );
END;
```

The data from csv file will be stored in **fl.stores** table and the file_path, path, where the file was identified will be stored in the **fl.FileQueue** table to track the processed and unprocessed files.

STEP 2: Create Procedure
(Comments have been added to the code for documenting the process)

```
CREATE OR ALTER PROCEDURE fl.ProcessLatestFile
AS
BEGIN
   SET NOCOUNT ON; -- Prevents SQL to output the number of rows effected
   DECLARE @file_path NVARCHAR(500); -- Declare file_path variable to store the
path of the file

   -- Extract the file_path from fl.FileQueue and store it in file_path variable

   SELECT TOP 1 @file_path = file_path
   FROM fl.FileQueue
   WHERE processed = 0
   ORDER BY created_at DESC;
   IF @file_path IS NULL
   BEGIN
      PRINT 'No unprocessed files found.'; -- If file_path not present, return
      RETURN;
   END
   BEGIN TRY
      TRUNCATE TABLE fl.stores;
      -- Else, run the sql command, using delimiter, record terminator and skip the first
row( or start from second row), since first row is usually headers in csv file.
      DECLARE @sql NVARCHAR(MAX);
      SET @sql = '
         BULK INSERT fl.stores
         FROM ''' + @file_path + '''
         WITH (
            FIELDTERMINATOR = '','',
            ROWTERMINATOR = ''\n'',
            FIRSTROW = 2
         );
      ';
      EXEC sp_executesql @sql;
      UPDATE fl.FileQueue SET processed = 1 WHERE file_path = @file_path;
      PRINT 'File imported: ' + @file_path;
   END TRY
   BEGIN CATCH
      PRINT 'Error: ' + ERROR_MESSAGE(); -- Print error, if encountered.
   END CATCH
END;
GO
```

SQL SERVER JOB SETUP (Similar to job explained from page 2)

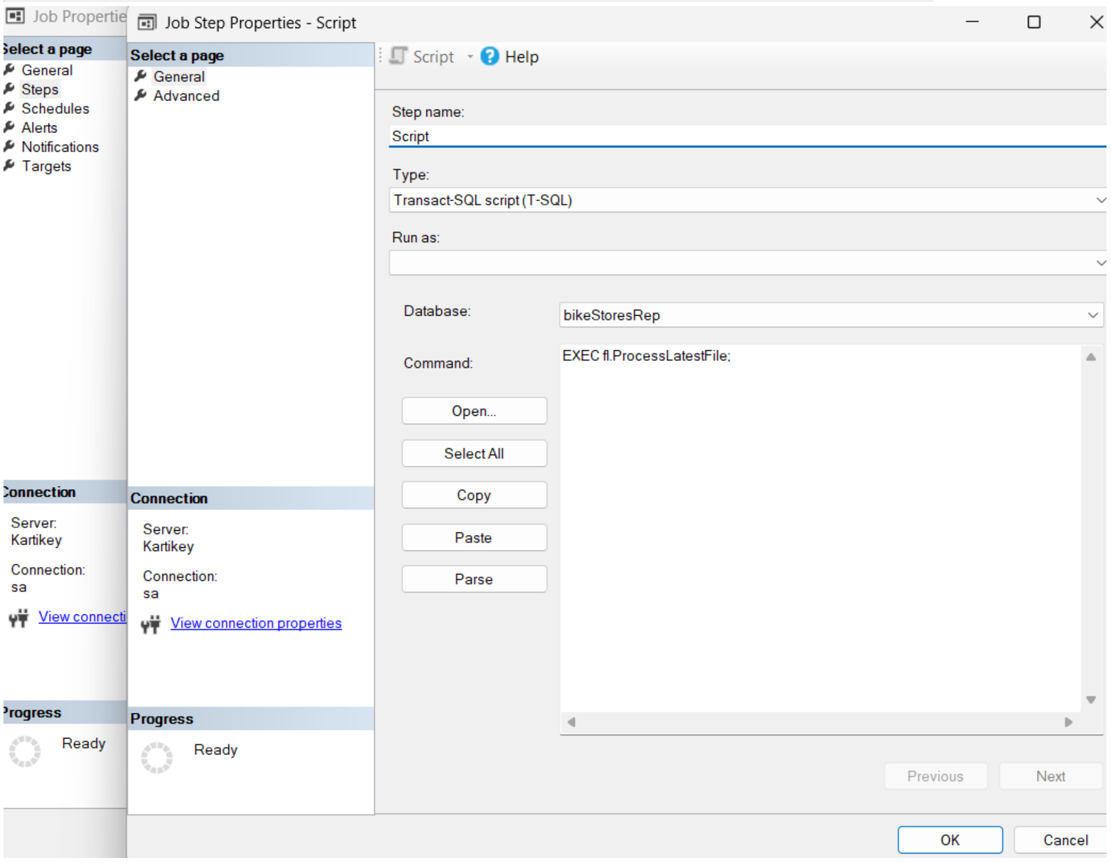

Job Properties - fileArrival

Select a page: General, Steps, Schedules, Alerts, Notifications, Targets

Name: fileArrival
Owner: sa
Category: [Uncategorized (Local)]
Description: No description available.
Enabled ✓
Source:
Created: 06-Jul-25 12:42:44
Last modified: 06-Jul-25 13:17:55
Last executed: 06-Jul-25 13:23:24
View Job History

Connection
Server: Kartikey
Connection: sa
View connection properties

Progress
Ready


Job Step Properties - Script

Select a page: General, Advanced

Step name: Script
Type: Transact-SQL script (T-SQL)
Run as:
Database: bikeStoresRep
Command:

```
EXEC fl.ProcessLatestFile;
```

Connection
Server: Kartikey
Connection: sa
View connection properties

Progress
Ready

Schedule has **not been added**, as the event handling is being handled by powershell that continously listens for the csv file instead of updating at regular intervals.

STEP 3: Create Powershell Script

(I am using windows, therefore used powershell. There are multiple ways to achieve the same operation, the following script is just an example to depict the file handling trigger in MS SQL)

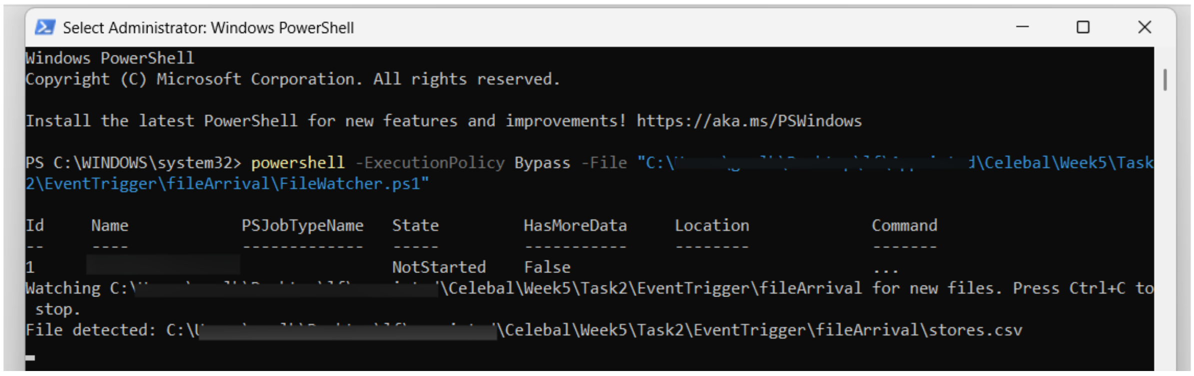
(Comments added for readability and understanding)

```
$watchFolder = "C: \EventTrigger\fileArrival"
$filter = "*.csv"
$sqlServer = "localhost"
$database = "bikeStoresRep"
$jobName = "fileArrival"
# Load SQL Server module
Import-Module SqlServer -ErrorAction SilentlyContinue
# Attach the path, file extension, event to the powershell and monitoring control
$watcher = New-Object System.IO.FileSystemWatcher
$watcher.Path = $watchFolder
$watcher.Filter = $filter
$watcher.EnableRaisingEvents = $true
$watcher.IncludeSubdirectories = $false
# Event handler when new file is created
Register-ObjectEvent $watcher "Created" -Action {
   # Extracting file name and file path
   $fileName = $Event.SourceEventArgs.Name
   $fullPath = Join-Path $watchFolder $fileName
   Write-Host "File detected: $fullPath"


   try {
      # Insert file path into SQL Server queue
      $insertQuery = @"
INSERT INTO fl.FileQueue (file_path)
VALUES (N'$fullPath');
"@
      Invoke-Sqlcmd -ServerInstance $sqlServer -Database $database -Query
$insertQuery
```

```
            # Trigger the SQL Server Agent job
            Invoke-Sqlcmd -ServerInstance $sqlServer -Database "msdb" -Query "
                EXEC msdb.dbo.sp_start_job @job_name = N'$jobName';
            "
            Write-Host "Job triggered for file: $fileName"
        }
        catch {
            # If any error occurs, display it on the console
            Write-Host "ERROR: $_"
        }
    }
    Write-Host "Watching $watchFolder for new files. Press Ctrl+C to stop."
    while ($true) { Start-Sleep -Seconds 5 }
```

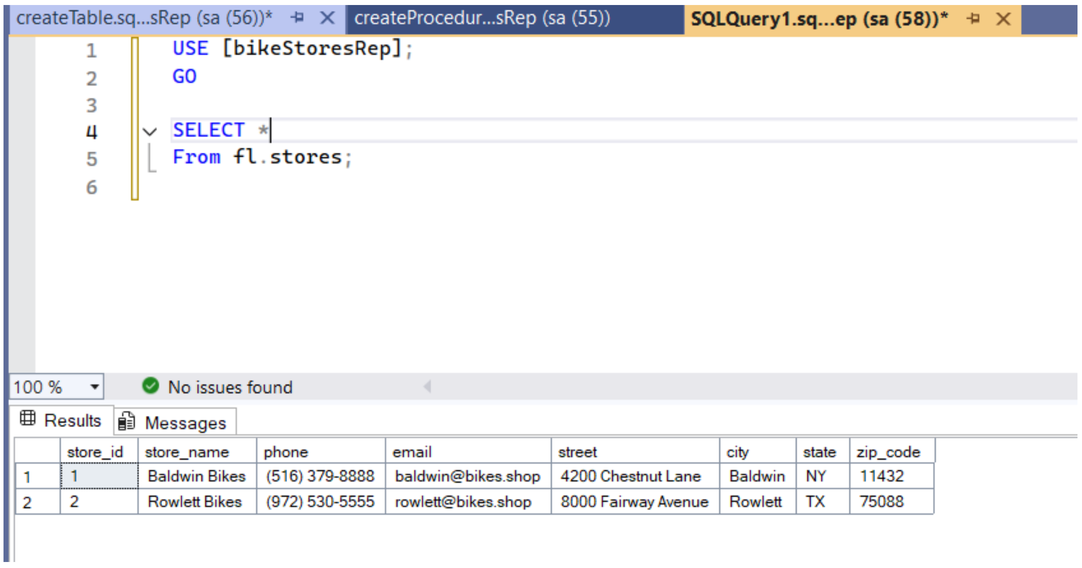
<u>OUTPUT:</u>

Running Powershell script



```
Select Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> powershell -ExecutionPolicy Bypass -File "C:\[redacted]\Celebal\Week5\Task
2\EventTrigger\fileArrival\FileWatcher.ps1"

Id     Name            PSJobTypeName   State         HasMoreData     Location             Command
--     ----            -------------   -----         -----------     --------             -------
1      [redacted]                      NotStarted    False                                ...
Watching C:\[redacted]\Celebal\Week5\Task2\EventTrigger\fileArrival for new files. Press Ctrl+C to
 stop.
File detected: C:\[redacted]\Celebal\Week5\Task2\EventTrigger\fileArrival\stores.csv
```

SQL Table

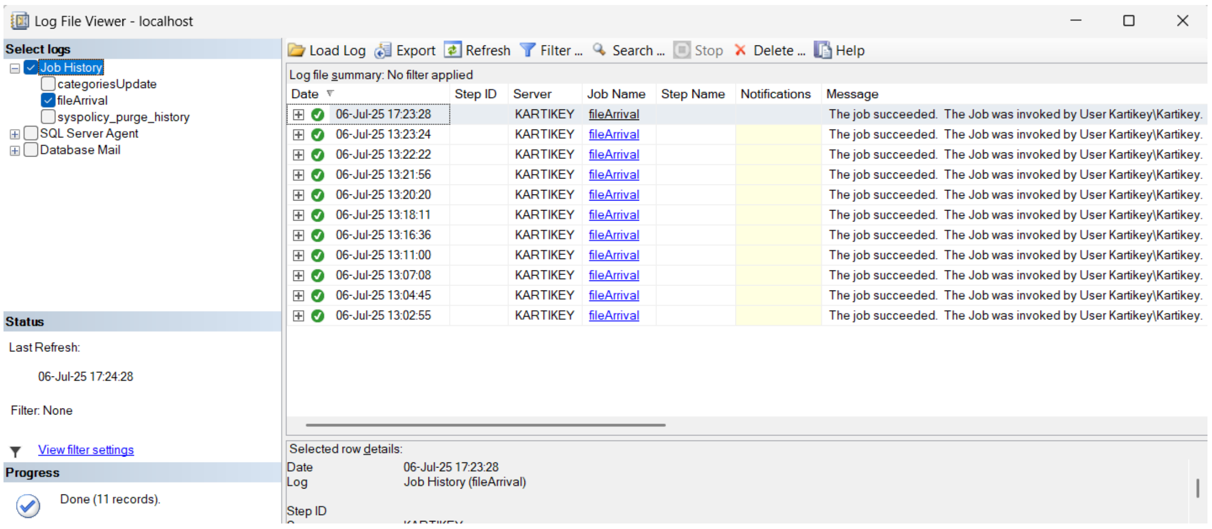


```
USE [bikeStoresRep];
GO

SELECT *
From fl.stores;
```

No issues found

	store_id	store_name	phone	email	street	city	state	zip_code
1	1	Baldwin Bikes	(516) 379-8888	baldwin@bikes.shop	4200 Chestnut Lane	Baldwin	NY	11432
2	2	Rowlett Bikes	(972) 530-5555	rowlett@bikes.shop	8000 Fairway Avenue	Rowlett	TX	75088

LOGS:



Log File Viewer - localhost

Select logs: Job History (categoriesUpdate, ✓ fileArrival, syspolicy_purge_history), SQL Server Agent, Database Mail

Log file summary: No filter applied

```
Date                Step ID  Server    Job Name     Step Name  Notifications  Message
06-Jul-25 17:23:28           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:23:24           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:22:22           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:21:56           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:20:20           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:18:11           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:16:36           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:11:00           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:07:08           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:04:45           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
06-Jul-25 13:02:55           KARTIKEY  fileArrival                            The job succeeded.  The Job was invoked by User Kartikey\Kartikey.
```

Status
Last Refresh:
06-Jul-25 17:24:28
Filter: None
View filter settings

Progress
Done (11 records).

Selected row details:

```
Date        06-Jul-25 17:23:28
Log         Job History (fileArrival)

Step ID
```

The log for above execution corresponds to the timestamp: **06-Jul-25-25 17:23:28**