Project Overview:

The goal of this project is to generate random student feed data into three separate feeds (Feed-1, Feed-2, Feed-3), automate the data generation, identify and cleanse duplicate entries, update data with unique rows, compare feeds, and validate the results. The process is built entirely in SQL Server with automation and testing using SQL.

Requirements:

|  |  |
| --- | --- |
| Req1 | Generate a random data input files a) Feed-1  which has 10 columns with 10 rows, b) Feed-2 which has 15 columns with 15 rows , c) Feed-3 which has 20 columns  with 20 rows |
| Req2 | Automate the Req 1 input file generation using SQL  scripts and the parameter will be "Feed name" & Number of Rows to populate Data |
| Req3 | Write SQL script to identify the duplicate (rows) in each of the table Feed-1, 2, 3 |
| Req4 | Write the duplicate records in output file - "duplicates" |
| Req5 | Create a script to replace all the duplicates with Unique rows and update back to respective Feed table |
| Req6 | Execute the duplicate script and check the output is zero |
| Req7 | Create SQL script to compare data from Feed-2,3 to Feed-1 and write in output file on the compared results |
| Req8 | Create Test plan with all kinds of manual test cases in order to test this End to End functionality |
| Req9 | Automate the test cases (if possible) using any method but should be automated… |
| Req 10 | Document everything in word with all screen grabs as proper Project Document |

**Table of Figures**

[Figure 1 : Feed\_1 table structure 3](#_Toc197414364)

[Figure 2: Feed\_2 and Feed\_3 Table structure 4](#_Toc197414365)

[Figure 3: Feed\_1 Stored Procedure for Data Generation 5](#_Toc197414366)

[Figure 4: Stored Procedure For Feed\_2 Data Generation 6](#_Toc197414367)

[Figure 5: Stored Procedure For Feed\_3 Data Generation 6](#_Toc197414368)

[Figure 6: Stored Procedure for Feed\_3 Part 2 7](#_Toc197414369)

[Figure 7: Feed\_1 Table 9](#_Toc197414370)

[Figure 8: Feed\_2 Data 10](#_Toc197414371)

[Figure 9: Feed\_3 Data with Duplicates Generated 10](#_Toc197414372)

[Figure 10: Duplicate 1 11](#_Toc197414373)

[Figure 11: Duplicate\_2 12](#_Toc197414374)

[Figure 12: Duplicate 3 12](#_Toc197414375)

[Figure 13: Duplicate 3 Data 13](#_Toc197414376)

[Figure 14: Stored Procedure for Unique Data 14](#_Toc197414377)

[Figure 15: Checking Duplicates After Insertion of Unique Data 15](#_Toc197414378)

[Figure 16: Compare Feed\_2 to Feed\_1 16](#_Toc197414379)

[Figure 17: Feed 2 and Feed 3 To Feed\_1 Comparison Data 16](#_Toc197414380)

[Figure 18: Save Comparison Data in CSV File Using BCP 17](#_Toc197414381)

[Figure 19: Test Case For Feed 1 Row and Column Count and Handling Null Value 19](#_Toc197414382)

[Figure 20: Test Case for Feed\_1 and Feed 2 Data Error 20](#_Toc197414383)

[Figure 21: Feed 3 Testing 20](#_Toc197414384)

[Figure 22: Feed 3 Duplicate Rows Testing 21](#_Toc197414385)

[Figure 23: Execution of Test Case Procedure 22](#_Toc197414386)

**Feed Specification:**

Student feed generation

Feed\_1 : 10 rows and 10 columns

Feed\_2 : 15 rows and 15 columns

Feed\_3 : 20 rows and 20 columns

**Step 1** : **Student Feed table Creation**

Created tables Feed\_1, Feed\_2 and Feed\_3 and defined the table structure using SQL DDL statements

Feed\_1 ( [StudentID] ,[FirstName] ,[LastName] ,[Email] ,[Phone] ,[Gender] ,[DOB] ,[EntryDate] ,[Major] ,[GPA] )

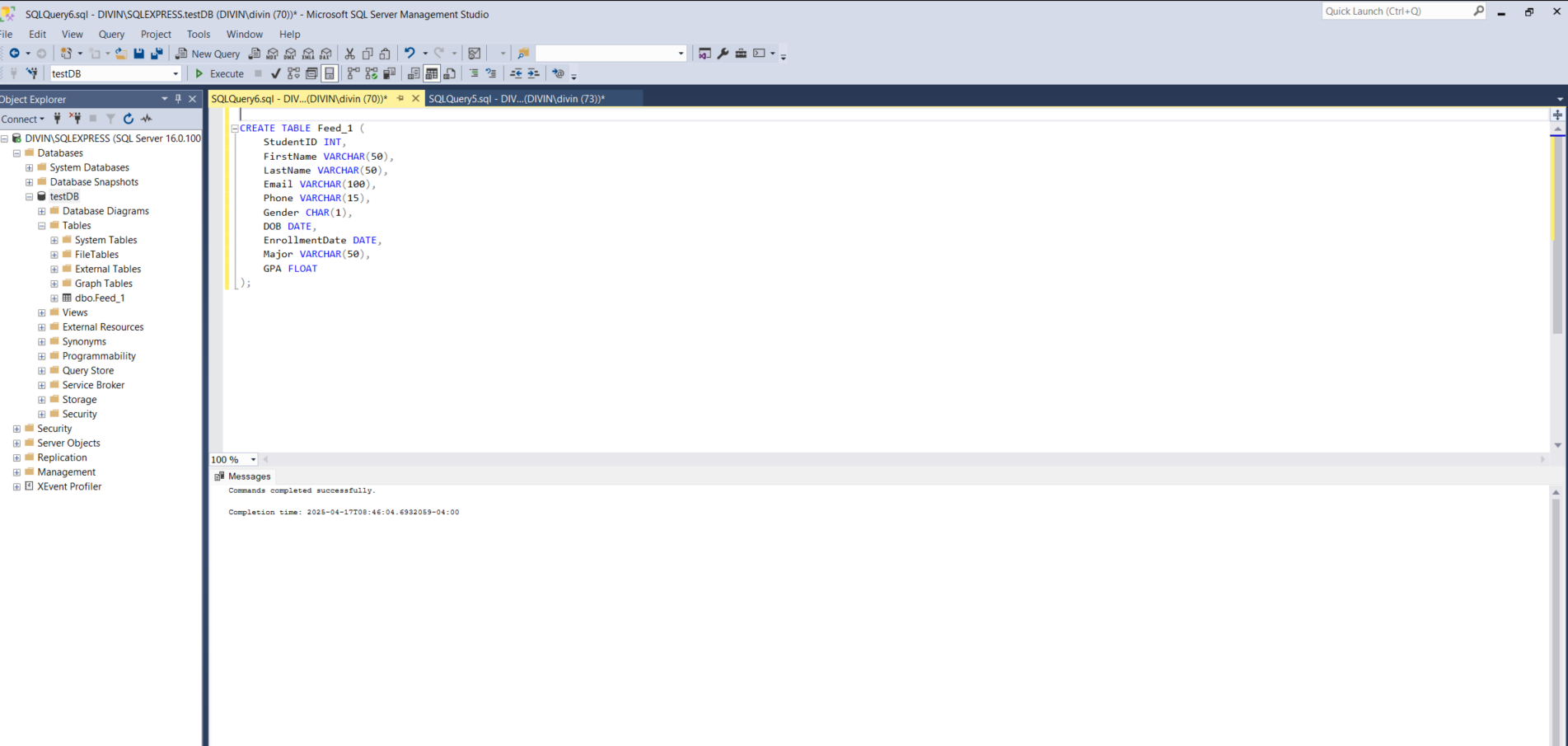


Figure 1 : Feed\_1 table structure

Feed\_2 ( [StudentID] ,[FirstName] ,[LastName] ,[Email] ,[Phone] ,[Gender] ,[DOB] ,[EntryDate] ,[Major] ,[GPA], [Address] ,[City] ,[State] ,[ZipCode] ,[GuardianName] )

Feed\_3 ( [StudentID] ,[FirstName] ,[LastName] ,[Email] ,[Phone] ,[Gender] ,[DOB] ,[EntryDate] ,[Major] ,[GPA] , [Address] ,[City] ,[State] ,[ZipCode] ,[GuardianName] ,[GraduationDate] ,[CreditsEarned] ,[TuitionBalance] ,[IsInternational] ,[Status] )

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 2: Feed\_2 and Feed\_3 Table structure

**Step 2 : Stored procedure for random data generation**

Stored procedures are pre-compiled SQL code that can be executed by name, often accepting input parameters and potentially returning output parameters or result sets.

* Procedure Name : StudentFeed
* Input Parameters:

@FeedName(VARCHAR) : Name of feed

@RowCount(INT): Number of rows to insert

* Key Features:
* Dynamic insertion based on feed type.
* Usage of different functions for realistic data generation.
* All fields are dynamically calculated and generated.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 3: Feed\_1 Stored Procedure for Data Generation

A computer code with many colorful text

AI-generated content may be incorrect.

Figure 4: Stored Procedure For Feed\_2 Data Generation

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 5: Stored Procedure For Feed\_3 Data Generation

The provided SQL code snippet is designed to insert data into the Feed\_3 table. It uses a conditional logic based on the value of the variable @i to populate the table with student data, including some simulated duplicate entries for testing and to fulfill project requirement of displaying duplicate data handling.

Conditional Insertion: The code uses an IF...ELSE IF statement to determine how data is inserted into Feed\_3.

* @i <= 16: When @i is less than or equal to 16, the code inserts a set of student records with @i as the primary identifier.
* @i > 16: When @i is greater than 16, the code inserts a set of student records where most of the data is similar to the first 16 records, but with @i - 16 used in the data generation. This creates a set of duplicate records, but with a different primary key.

A computer code with many text

AI-generated content may be incorrect.

Figure 6: Stored Procedure for Feed\_3 Part 2

Code snippet of @i>16 for feed\_3 condition is written to generate a dataset that includes a set of duplicate data which could be useful for testing and handling the duplicate record.

A screenshot of a computer

AI-generated content may be incorrect.

After writing and running Script for data generation as a stored procedure with FeedName and RowCount as input parameters.

**EXEC StudentFeed FeedName, RowCount** command executes a stored procedure named “StudentFeed” to generate data in the Feed\_1, Feed\_2, and Feed\_3 tables, passing the values 10, 15, and 20, respectively, as the number of rows to be generated for each table.

* **EXEC (or EXECUTE)**: This is a SQL command used to run a stored procedure or a string containing dynamic SQL.
* **StudentFeed**: This is the **name of the stored procedure** that is being called.
* FeedName and RowCount are parameters passed

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7: Feed\_1 Table

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8: Feed\_2 Data

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: Feed\_3 Data with Duplicates Generated

**Step 3: Check for duplicates**

The SQL code identifies and extracts duplicate records from the feed\_1 table and stores those duplicates in a new table called duplicate\_feed1.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 10: Duplicate 1

The code efficiently identifies records in the feed\_1 table that have duplicate values across the FirstName, LastName, Email, Phone, DOB, and Major columns. It then creates a new table, duplicate\_feed1, containing only these duplicate records.

CTE common table expression is used to create a temporary view (duplicate\_1) with row number with partitioning by FirstName, LastName, Email, Phone, DOB, and Major columns and finding duplicates by filtering the temporary view by row number.

Feed\_1 shows zero rows affected since there are no duplicates in the table.

The same process is followed with Feed\_2 and Feed\_3 and duplicates found are moved into new tables named duplicate\_2 and duplicate\_3.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11: Duplicate\_2

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12: Duplicate 3

Feed\_3 had 4 duplicates and It gets added to the new file duplicate\_3

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13: Duplicate 3 Data

**Step 4: Replacing duplicates with unique values**

To replace duplicates with unique values, the duplicates must first be removed using a **DELETE FROM table\_name WHERE** condition statement. In this script, data is deleted from the Duplicates\_3 temporary view where the row number is greater than 1, and new unique values are inserted using a stored procedure.

A screenshot of a computer

AI-generated content may be incorrect.

**A computer code with many text

AI-generated content may be incorrect.**

Figure 14: Stored Procedure for Unique Data

Unique rows for Feed\_3 are inserted using the standard data generation script, with data fields dynamically calculated based on the passed parameter @i. This script generates new data after the 16th row because the StudentFeed data generation procedure adds unique rows to Feed\_3 only up to the 16th row ( reference Figure 6).

After insertion of the new unique rows, the Duplicates\_3 script is run to confirm that no more duplicates exist, as shown in Figure 15."

A computer screen shot

AI-generated content may be incorrect.

Figure 15: Checking Duplicates After Insertion of Unique Data

**Step 5: Feed Comparison**

Compared data from Feed\_2 and Feed\_3 to Feed\_1 using common fields by JOIN statement. SQL code provided below is used to find rows in Feed\_2 and Feed\_3 that do not have matching row in Feed\_1 based on StudentID.

Feed\_2 is compared with Feed\_1 and 5 affected row data is moved into FEED2\_TO\_1 new table.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 16: Compare Feed\_2 to Feed\_1

A screenshot of a computer

AI-generated content may be incorrect.

Figure 17: Feed 2 and Feed 3 To Feed\_1 Comparison Data

A screenshot of a computer

AI-generated content may be incorrect.

Figure18: Compare Feed\_3 To Feed\_1

Figure 18 shows the SQL script used to compare the Feed\_3 table to Feed\_1. Feed\_3 contains 20 rows of data, while Feed\_1 has 10 rows. Therefore, the data from StudentID 11 to 20, which is not present in Feed\_1, is moved into Feed3\_to\_1 (affecting 10 rows).

A black screen with white text

AI-generated content may be incorrect.

Figure 18: Save Comparison Data in CSV File Using BCP

BCP (Bulk Copy Program) is a command-line utility in SQL Server that is commonly used to export large amounts of data to a file. It can be very efficient for this purpose. To export the data to a CSV file, you would typically use a BCP command with the following options:

* out: Specifies that you are exporting data.
* <filename>.csv: Specifies the name of the CSV file you want to create.
* -c: Specifies that the data should be exported in character format.
* -t,: Specifies the comma (,) as the field terminator, which is what separates the values in a CSV file.
* -T : Specifies the authentication method. -T is for a trusted connection (using your Windows credentials).

Figure 19 shows the usage of BCP to save Feed2\_to\_1 and Feed3\_to\_1 data as CSV file to Local folder on the PC.

**Step 6: Manual Test Plan for End-to-End Validation**

Project : Student feed Generator

Env : SQL Server

Tools : SSMS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Test Case | Input | Expected Result | Test Result |
| 1. | Define Feed\_1, Feed\_2 and Feed\_3 structure with 10 ,15 and 20 columns each | EXEC sp\_help (stored procedure help) 'Feed\_1',’Feed\_2’ and ‘Feed\_3’ | Table structure including, Column name, data types and length. | Pass |
| 2. | Generate Feed\_1 with 10 rows | Execute Stored procedure by passing table name as Feed\_1, and row count 10 | Table Feed\_1 has 10 rows with 10 columns | Pass |
| 3. | Generate Feed\_2 with 15 rows | Feed\_2, 15 | Table Feed\_2 has 15 rows with 15 columns | Pass |
| 4. | Generate Feed\_3 with 20 rows | Feed\_3, 20 | Table Feed\_3 has 20 rows with 20 columns | Pass |
| 5. | |  | | --- | | Validate data types for each column | | Select statement | All columns match expected data types | Pass |
| 6. | Check for duplicates using Script | Feed\_1, Feed\_2, Feed\_3 | Outputs duplicate records | Pass |
| 7. | Clean duplicates using script | Delete duplicates and replace with unique records | Table updated with unique records only | Pass |
| 8. | Re-run duplicate check script | Feed\_1, Feed\_2, Feed\_3 | Duplicate count = 0 | Pass |
| 9. | Compare Feed\_2 and Feed\_3 to Feed\_1 | Match on common columns | Outputs differences | Pass |
| 10. | Generate CSV/Excel from results | Bulk copy program (BCP) using Cmd | CSV files contain expected content | Pass |

**Step 7: Automate test cases**

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 19: Test Case For Feed 1 Row and Column Count and Handling Null Value

**A screenshot of a computer code

AI-generated content may be incorrect.**

Figure 20: Test Case for Feed\_1 and Feed 2 Data Error

**A screenshot of a computer program

AI-generated content may be incorrect.**

Figure 21: Feed 3 Testing

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 22: Feed 3 Duplicate Rows Testing

Figure 19 to 22 are the parts of Testing Procedure script, It performs Automated unit testing for Student data feed (Feed\_11, Feed\_2 and Feed\_3) to validate row counts, column counts, data errors and duplicates.

* Schema Validation:
* Confirms that each feed has the correct number of columns.
* Verifies that the correct number of rows were inserted.
* Data Quality Checks:
* Looks for NULL values in LastName or StudentID.
* Ensures email format has an “@” .
* Validates phone numbers have exactly 10 digits (excluding hyphens).
* Duplicate Detection:
* Groups rows by all key columns (StudentID, FirstName, etc.) and checks for duplicates.

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 23: Execution of Test Case Procedure

All test cases passed with PASS status in SSMS message window.

**Summary:**

The Student Feed generation and validation project demonstrates a fully automated, scalable data pipeline using Microsoft SQL Server. Randomized feed creation, integrity validation, duplicate handling, and automated testing—were all achieved within a modular and reusable architecture.

The solution ensures data quality by including checks for nulls, format discrepancies, and duplicates. Test automation through stored procedures enables rapid regression testing, increasing confidence in the integrity of generated feeds. All tests passed successfully, confirming that the system performs accurately across diverse scenarios.