Wayne Wang

INFO 498 C

26 August 2019

Final Project – Whole Process of ETLs and Backups

Table of Contents

[Introduction 1](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417665)

[Step 1. Attaching/Creating Databases to local SQL server 2](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417666)

[Step 2. ETL Process from Flat Files to OLTP Database 3](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417667)

[Step 3. ETL Process from OLTP Databases to DW Database 5](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417668)

[Step 4. Create a SSIS project for the ETL Process 14](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417669)

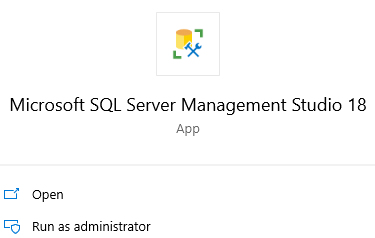
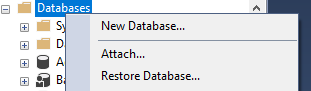
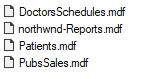
[Step 5. Write Stored Procedures for Back ups 16](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417670)

[Step 6. Put Everything under the SSIS Solution 18](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417671)

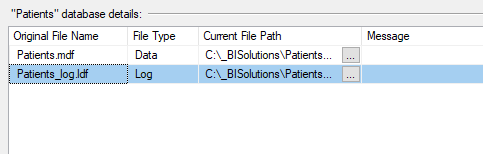
**Introduction**

For this Final Project, we are going to perform the whole process of ETL process. It involves tasks from attaching/creating databases, importing data from flat files to OLTP database with Incremental ETL process, ETL processes (both ‘Flush and Fill’ and Incremental) from OLTP databases to DW Database. Furthermore, we are going to create stored procedures and a job to perform daily backup task for these three final databases.

**Topics**

**Step 1. Attaching/Creating Databases to local SQL server**

**Figure 1 Figure 2 Figure 3**

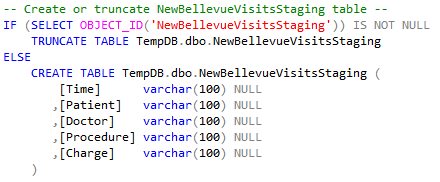
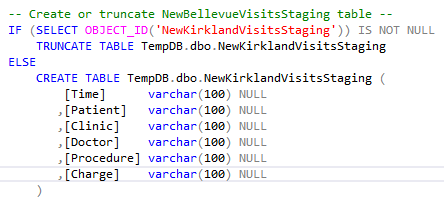
**** We first move the ‘DoctorsSchedules.mdf’ and ‘Patients.mdf’ from the ‘Databases’ folder to the folder ‘\_BISolution’ under C: diver. Then we open the Microsoft SQL Server Management Studio 18 as an administrator (**Figure 1**). After opening the SQL Management Studio and connect to local server, then we right-click ‘Databases’ under ‘Object Explorer’ and click ‘Attach’ (**Figure 2**). Under the Attach databases dialog, select ‘Add’ and direct to the ‘\_BISolution’ folder under C: drive and add ‘DoctorsSchedules.mdf’ and ‘Patients.mdf’ (each at a time) (**Figure 3**). Before click ‘OK’, make sure to remove the ‘ldf’ file from the ‘DoctorsSchedules’/’Patients’ database details box. (**Figure 4**)

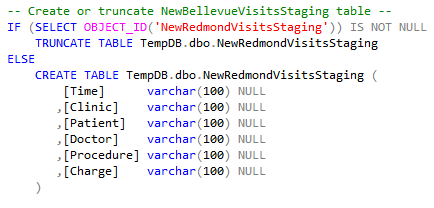
**Figure 4**

After attaching the two OLTP databases, we open the SQL Script in the ‘Databases’ folder to create the DW database, ‘DWClinicReportData’

**Step 2. ETL Process from Flat Files to OLTP Database**

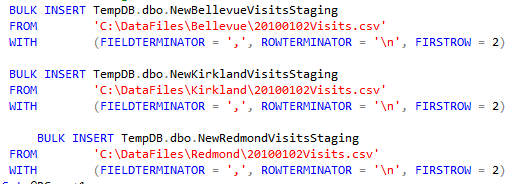
**Figure 5 Figure 6**

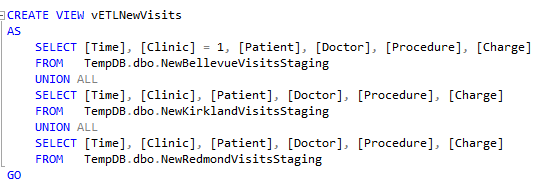
 First, move the ‘DataFile’ from the Solution folder to C: drive. Then we start to write SQL Script to perform the ETL process from the Flat Files to Patients databases. We are using the ‘TempDB’ to create Staging Tables and first import data into the staging tables. (**Figure 5**). First check if there is an existing Procedure named ‘pCreateOrTruncateStagingTables’, if yes, then drop the procedure. (We will write this if condition before every store procedure to minimize the probability of errors, this is a nice to have style.) (**Figure 6**)

 **Figure 7 Figure 8**

**Figure 9**

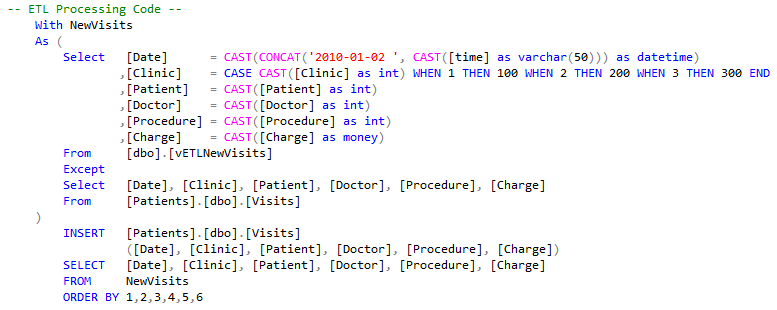
Then we create the Stored Procedure ‘pCreateOrTruncateStagingTables’, within the procedure, we write code to create/truncate the three staging tables: ‘NewBellevueVisitsStaging’ (**Figure 7**), ‘NewKirklandVisitsStaging’ (**Figure 8**) and ‘NewRedmondVisitsStaging’ (**Figure 9**). Then we execute the procedure. **NOTE:** ‘NewBellevueVisitsStaging’ table doesn’t have a ‘Clinic’ column, the order for ‘Clinic’ and ‘Patient’ columns are reversed in the other two tables.

**Figure 10 Figure 11**

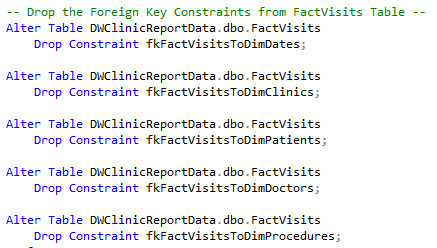
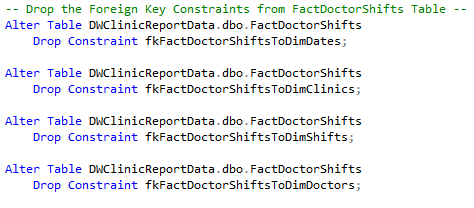
Then, we write a stored procedure ‘pETLCsvToStagingTables’ to perform the ETL process (Check its existence with the if condition before create procedure). Inside the procedure, we use ‘Bulk Insert’ with file paths to insert data from csv files into corresponding staging tables (**Figure 10**). After executing the procedure, ETL from csv files to staging tables is completed.

**Figure 12**

Now we use ‘Patients’ database to continue the rest of tasks (**Figure 11**). Then we create an abstract layer, ‘vETLNewVisits’, for the ‘Visits’ table. We use this view to union data from all three staging tables together, with some normalizations. (Add a ‘Clinic’ column when selecting from the ‘NewBellevueVisitsStaging’ table. Switch ‘Patients’ and ‘Clinics’ columns order when selecting from ‘NewKirklandVisitsStaging’ table).

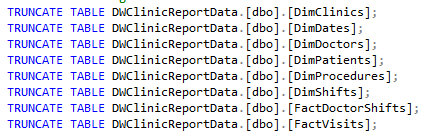
The final step for the ETL process is to perform the incremental ETL into the ‘Visits’ table. We write a procedure named ‘pETLSyncPatientNewVisits’ (Check its’ existence before creating it). For the reason that we only care about inserting data here, so we don’t include the updating and deleting scripts for incremental ETL process. For inserting the data from ‘vETLNewVisits’, we select all rows that are in ‘vETLNewVisits’ but not in the ‘Visits’ table and insert them into ‘Visits’ table. Throughout this process, we will change the column name ‘time’ into ‘Date’, change its content to ‘yyyy-mm-dd hh:mm:ss’ and change the column data type to ‘datetime’. Change the data in ‘Clinic’ column from 1/2/3 to 100/200/300. Then change the ‘charge’ column’s data type to ‘money’. Change the data type of other columns into ‘int’. Then we execute to procedure to finish the final step of this ETL process. (**Figure 13**)

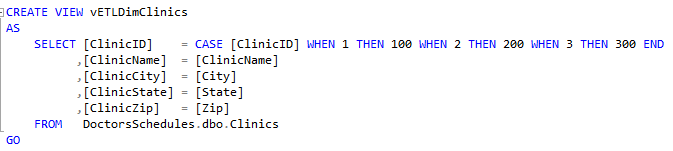
**Figure 13**

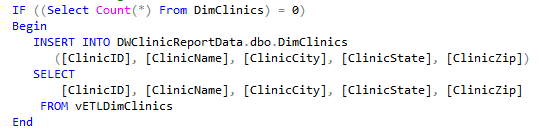
**Step 3. ETL Process from OLTP Databases to DW Database**

**Figure 14 Figure 15**

For this task, we will be using the ‘DWClinicReportData’ database. Before performing the ETL Process, we need to drop all Foreign Key Constraints from the Fact Tables in the database and then truncate all tables in the database. We write stored procedures ‘pETLDropForeignKeyConstraints’ (**Figure 14 & 15**) and ‘pETLTruncateTables’ (**Figure 16**)

to perform these tasks (Check their existences before creating them).

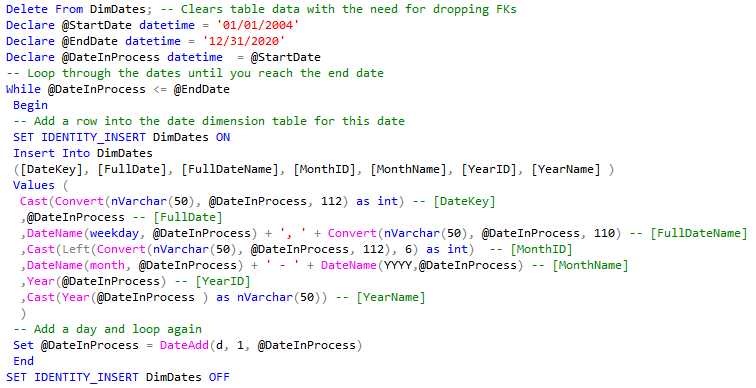
**Figure 16**

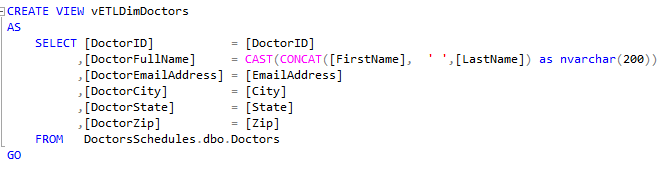
**Figure 17**

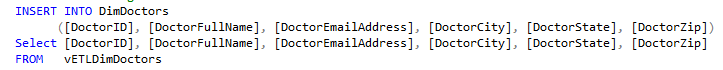
**Figure 18**

After the preparation work is done, we can start the ETL process one table at a time. (**NOTE**: The order varies, however, remember to always perform the ETL process for Fact tables after the ETL process es are completed for all Dimension tables that connect to them)

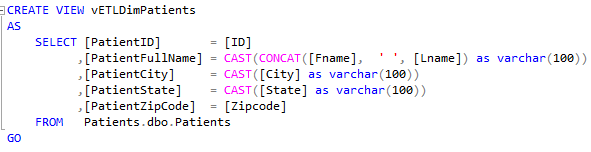
First, we create an abstract layer, ‘vETLDimClinics’, for ‘DimClinics’ table. Import data from ‘Clinics’ table in ‘DoctorsSchedules’ database. During this process, we need to normalize the ‘ClinicID’ from 1/2/3 to 100/200/300 (**Figure 17**). Then we create a stored procedure named ‘pETLFillDimClinics’ to load data from ‘vETLDimClinics’ into ‘DimClinics’. (**Figure 18**)

**Figure 19**

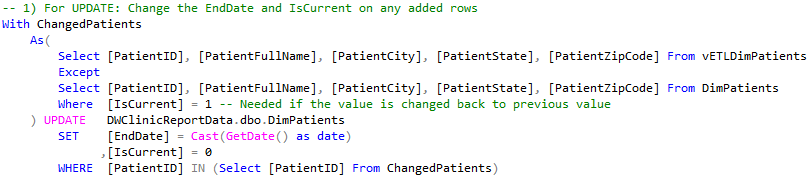
 Then we create a stored procedure named ‘pETLFillDimDates’to generate data in the ‘DimDates’ table (**Figure 19**). We set the ‘@StartDate’ to ‘01/01/2004’ and ‘@EndDate’ to ‘12/31/2020’ to make sure all data from the OLTP databases will be covered within this range. (**NOTE**: In order to insert the ‘DateKey’ into ‘DimDates’ table, we need to set ‘Identity\_insert’ to ‘ON’ before inserting and set it back to ‘OFF’ after inserting all data into the table).

**** **Figure 20**

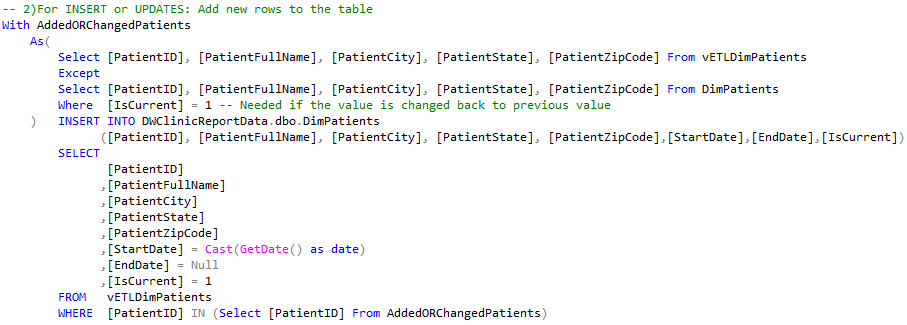
**Figure 21**

Then we start the ETL process for ‘DimDoctors’ table. First, we create an abstract layer, ‘vETLDimDoctors’ for the ‘DimDoctors’ table. For creating this view, we select data from ‘Doctors’ table from ‘DoctorsSchedules’ database. To transform the data, we combine the ‘FirstName’ and ‘LastName’ columns into ‘DoctorFullName’. In addition, we will not extract the ‘DoctorAddress’ column from the source table (**Figure 20**). Then we create a stored procedure named ‘pETLFillDimDoctors’ to insert data from ‘vETLDimDoctors’ into ‘DimDoctors’ table. (**Figure 21**)

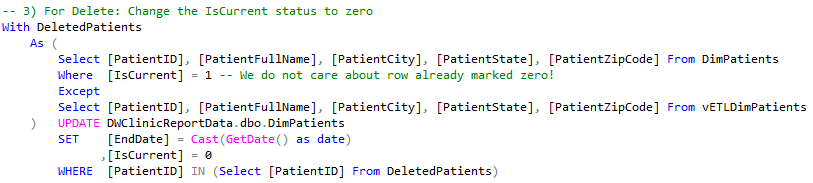
**Figure 22**

 Then we start the ETL process for ‘DimPatients’ table. The ETL process for ‘DimPatients’ table is different from the ETL processes for other tables. Instead of using ‘Flush and Fill’ process, we are going to do an incremental ETL process for this table. First, we create an abstract layer for ‘DimPatients’ to normalize/transform the data. This view selects data from ‘Patients’ table from ‘Patients’ database. We combine the ‘Fname’ and ‘Lname’ columns from ‘Patients’ table into ‘PatientFullName’ column in the view. We then add ‘Patient’ in front all column names and change the data type of ‘PatientFullName’, ‘PatientCity’ and ‘PatientState’ columns into ‘varchar(100)’. We don’t need to extract the ‘Email’ and ‘Address’ columns from the source table (**Figure 22**).

**Figure 23**

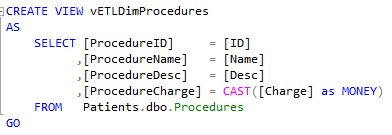
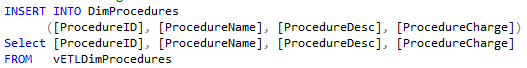
Then we create a stored procedure named ‘pETLSyncDimPatients’ to perform the Incremental ETL process. Inside the procedure, first we write scripts to check if there are any update to the current patients. If yes, then change these row’s ‘EndDate’ to the current date and change the value of ‘IsCurrent’ to 0. (**Figure 23**)

**Figure 24**

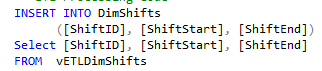
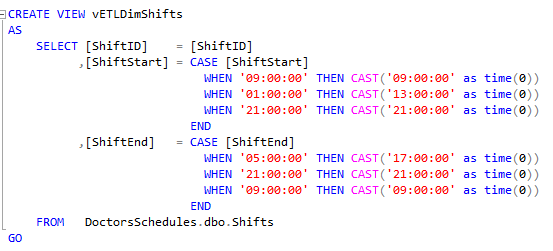
 Then we write scripts to perform the inserting task. First we select all rows that are in the ‘vETLDimPatients’ but not in the ‘DimPatients’ table, and then insert these rows into ‘DimPatients’. While inserting the new rows, we need to add three more column values to each row: ‘StartDate’ with the current date, also change the datatype to ‘date’; ‘EndDate’ with ‘Null’ as its default value; ‘IsCurrent’ with value 1 to indicate this patient is current. (**Figure 24**)

**Figure 25**

The final task within the procedure is to perform the deleting task. We select all rows that are in the ‘DimPatients’ table but not in ‘vETLDimPatients’. These rows indicate the patients who are not current patients. Then we with update these selected rows in ‘DimPatients’ table by changing their the ‘EndDate’ value to current date and ‘IsCurrent’ value to 0 to indicate that they are not current anymore. (**Figure 25**)

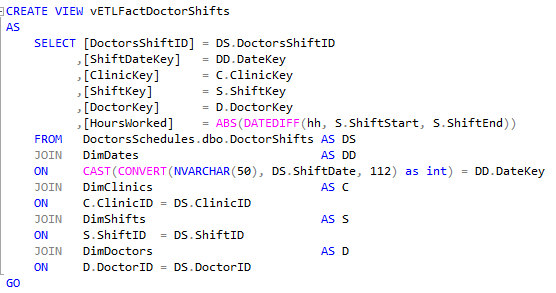
 **Figure 26**

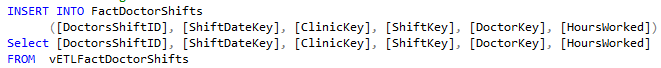
**Figure 27**

Then we start the ETL process for ‘DimProcedures’ table. We first create an abstract layer, ‘vETLDimProcedures’, for ‘DimProcedure’. In this view, we select data from the ‘Procedures’ table form ‘DoctorsSchedules’ database. During the process, we change the datatype of ‘Charge’ to ‘Money’, and add ‘Procedure’ in front every column names (**Figure 26**). After creating the abstract layer, we create a stored procedure to insert data from ‘vETLDimProcedures’ into ‘DimProcedures’ to finish the ETL process for ‘DimProcedures’. (**Figure 27**)

**Figure 28 Figure 29**

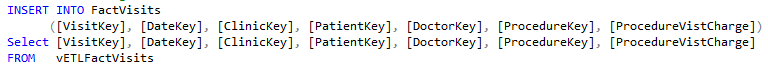
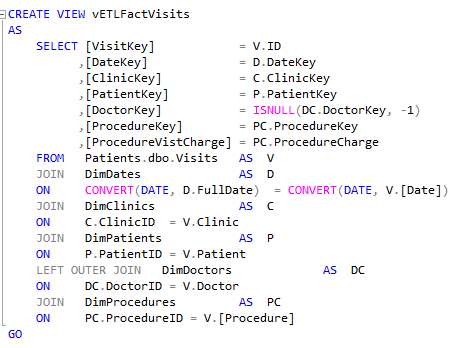
Then we start the ETL process for ‘DimShifts’ table. First, we create an abstract layer, ‘vETLDimShifts’, for ‘DimShifts’. In the view, we select data from ‘Shifts’ table from ‘DoctorsSchedules’ database and normalize the data. When change the ‘ShiftStart’ and ‘ShiftEnd’ into military time to reduce the confusions (Example: 01:00:00 🡪 13:00:00) (**Figure 28**). After creating the abstract layer, we create a stored procedure to insert data from ‘vETLDimShifts’ to ‘DimShifts’ to finish the ETL process (**Figure 29**).

Then we move on to the ETL processes for the two Fact tables. For these two tables, we will still be using ‘Flush and Fill’ ETL process.

 **Figure 30**

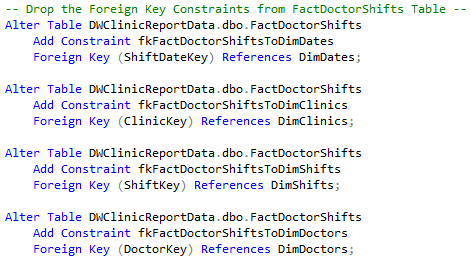
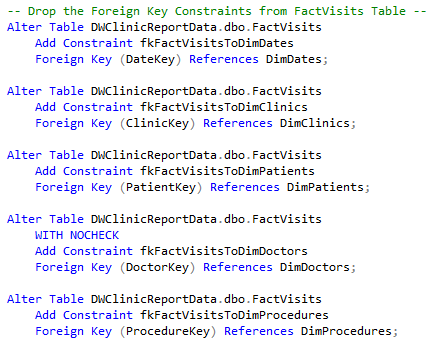
**Figure 31**

First, we start the ETL process for ‘FactDoctorShifts’ table. We create an abstract layer, ‘vETLFactDoctorShifts’ for ‘FactDoctorShifts’. In the view, we will select data from different tables. We will join the ‘DoctorShifts’ table from ‘DoctorsSchedules’ database with the ‘DimDates’, ‘DimClinics’, ‘DimDoctors’ tables to get the data columns we needed. Furthermore, for the ‘HoursWorked’ column, we will take the absolute value of the time difference (in hours) between the ‘ShiftStart’ and ‘ShiftEnd’ values from the ‘DimShifts’ table (**Figure 30**). After creating the abstract layer, we create a stored procedure named ‘pETLFillFactDoctorShifts’ to insert all data from ‘vETLFactDoctorShifts’ into ‘FactDoctorShifts’ to complete the ETL process for ‘FactDoctorShifts’ table. (**Figure 31**)

**Figure 32**

**Figure 33**

Now, we start the ETL process for the last table left in this DW database. We create an abstract layer, ‘vETLFactVisits’ for the table ‘FactVisits’. In the view, we will select data from different tables. We will join the ‘Visits’ table from ‘Patients’ database with the ‘DimDates’, ‘DimClinics’, ‘DimPatients’, ‘DimProcedures’ tables. Then we Left Outer Join the ‘DimDoctors’ table as well. When Joining the ‘DimDates’ table, we need to convert ‘FullDate’ from ‘DimDates’ and the ‘Date’ from ‘Visits’ into ‘DATE’ data type before make the comparison. Furthermore, when selecting the columns, for the ‘DoctorKey’, we need to change any Null values to ‘-1’ to handle several ‘weird’ rows. (**Figure 32**). After creating the abstract layer, we then create a stored procedure to insert all data from ‘vETLFactVisits’ into ‘FactVisits’ table to complete the ETL process for ‘FactVisits’ table. (**Figure 33**)

 **Figure 34** **Figure 35**

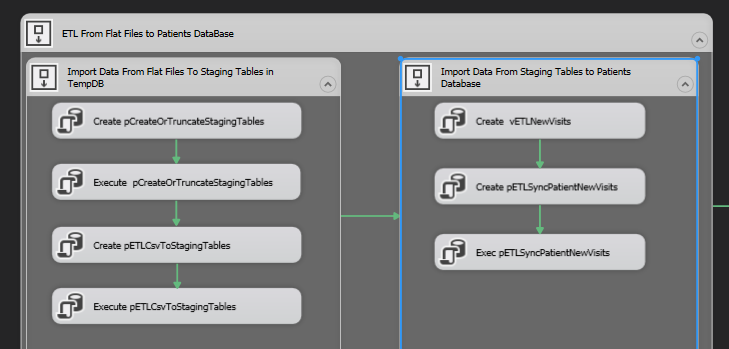
After finishing all the stored procedures for the ETL process, then we write a stored procedure named ‘pETLAddForeignKeyConstraints’ to add all Foreign Key Constraints back to the ‘FactDoctorShifts’ table (Figure 34) and ‘FactVisits’ table (Figure 35).

The final step is to execute all the stored procedure to make sure everything runs successfully and the result is what we expected.

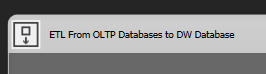
**The ETL process from Flat Flies 🡪 OLTP databases 🡪 DW database is completed.**

Step 4. Create a SSIS project for the ETL Process

**Figure 36 Figure 37**

For this step, we are going to create an SSIS project to perform the ETL Process. First of all, open visual studio and create a new project. Under the ‘Business Intelligence’, select ‘Integration Services Project’. Then choose the path for your project and name the project.

**Figure 38**

After creating and opening the SSIS package, we will be working under the control flow. We will first create a ‘Sequence Container’ from the SSIS toolbox, for the ETL process from Flat Files to the ‘Patients’ database. Inside the Sequence Container named ‘ETL from Flat Files to Patients Database’, we create two Sequence Containers, one for ‘Import Data From Flat Files To Staging Tables in TempDB’, one for ‘Import Data from Staging tables to Patients’. Under these two Sequence Containers, we create ‘Execute SQL Task’ with the SQL scripts we wrote from previous sections for Importing data from Flat Files to Staging tables and Importing data from staging tables to ‘Patients’ database. (Figure 38)

**Figure 39**

Afterwards, we create another Sequence Container for the ETL process from OLTP databases to the DW database, we name the Sequence Container ‘ETL From OLTP Databases to DW Database’. Inside this container, we create a Sequence Container for each table’s ETL process: ‘DimClinics’, ‘DimDate’, ‘DimDoctors’, ‘DimPatients’, ‘DimProcedures’, ‘DimShifts’, ‘FactDoctorShifts’, ‘FactVisits’. We also create Sequence Containers for the process of Dropping Foreign Key Constraints, Truncating Tables and Adding Foreign Key Constraints back. In each of the Containers, we create several ‘Execute SQL Task’ tasks with corresponding SQL code that we wrote in previous sections. (**NOTE**: Remember to add OLE DB Connections to the project before creating the ‘Execute SQL Task’. When creating the ‘Execute SQL Task’, connect to corresponding database connections under the ‘Connection’ )

After finishing adding SQL codes to each ‘Execute SQL Task’ and organize them in their belonged Sequence Containers. We connected all the components. Connect the two large Containers together, under each Containers, connect the sub-containers together as well. Under each sub-containers, connect all the ‘Execute SQL Task’ based on a reasonable order. Basic Rule to follow: ETL from Flat File to OLTP databases before ETL from OLTP databases to DW database; ETL from flat files to staging tables before ETL from staging tables to ‘Patients’ databases; Drop Foreign Keys before Truncate the tables, before ETL processes. ETL for Dimension tables before ETL for Fact Tables; Add Foreign Key Constraints back in the end.

**A picture containing black

Description automatically generated**A possible work flow for this project look similar to **Figure 40.**

**Figure 40**

After finishing this project, we use ‘Debug’ 🡪 ‘Start Debugging’ to make sure there is no error while debugging. Then we can trace back to SQL Server Management Studio to see if the data in OLTP databases and DW database are as we expected.

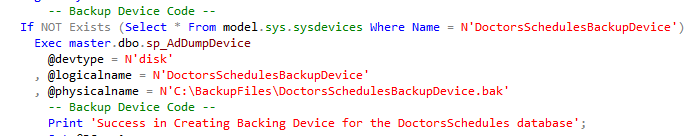
After confirmations, we completed this step of creating SSIS project.

[Step 5. Write Stored Procedures for Back ups](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417670)

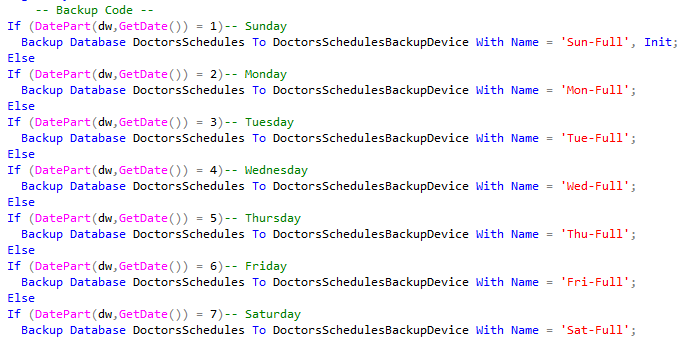
For this step, we need to Automate the Backup of all three fain databases.

First, we need to create stored procedures for preformatting the backup tasks. The stored procedures for backup these three databases are very similar. Here we will show the process for creating stored procedures for the ‘DoctorsSchedules’ databases.

Like in the previous sections, we always check the stored procedure’s existence in the database before we create the stored procedure.

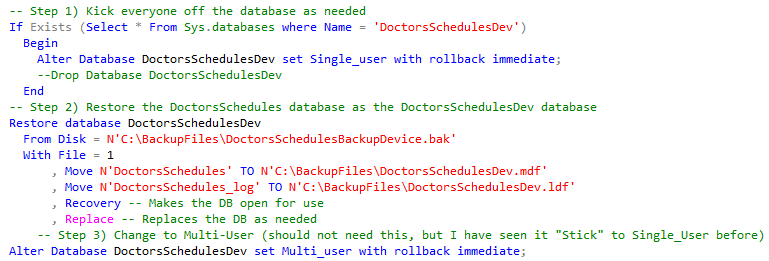
 First, we create a stored procedure named ‘pCreateDoctorsSchedulesBackupDevice’ to create a backup device for the DoctorsSchedules database if there isn’t a backup device created previously. (**Figure 41**)

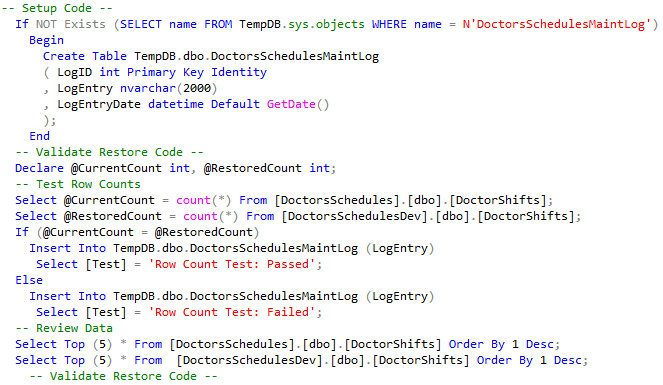
**Figure 41**

 Then we write a stored procedure named ‘pMaintBackupDoctorSchedules’ to performs database backups on the DoctorsSchedules Databases. In this procedure, we will base on the current data of backup to name the backup, it will be easy for us to know when was a certain backup created. This is necessary since we are doing a daily backup. (**Figure 42**)

**Figure 42**

After making the backup tasks, we need two more stored procedures to perform post backup tasks to test it.

 **Figure 43**

We now create a stored procedure named ‘pRefreshDoctorsSchedulesDev’ to create/refresh a Dev Database based on the Sunday ‘First File’ back up of the DoctorsSchedules Database. First, we need to rollback all users off the database. Then we restore the DoctorsSchedules databases as the DoctorsSchedulesDev database. We can then execute this stored procedure and using ‘Select \* From DoctorsSchedulesDev.sys.Tables’ to say if the backup is restored successfully. (**Figure 43**)

**Figure 44**

The final step is to create a stored procedure ‘pTestRestoreFromDoctorsSchedules’ to compare the data in ‘DoctorsSchedules’ and in ‘DoctorsSchedulesDev’ to see if they are exactly the same. (**Figure 44**) (**NOTE**: for this step, for ‘DoctorsSchedules’ databases, we compare the data with ‘DoctorShifts’ tables; for ‘Patients’ database, we compare the data with ‘Visits’ tables; for ‘DWClinicReportData’ database, we compare the data with ‘FactVisits’ tables).

The steps above is an example using ‘DoctorsSchedules’ databases, similar process will also be applied to other two databases.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated **Figure 45**

**Figure 46**

After creating all the stored procedures, we create a job with the SQL Server Agent to automate the daily backup. We set the name as ‘BackupFinalDatabases’ and the owner as ‘sa’ (**Figure 45**). For the steps, we create one step for backup each database (totally three backups for three databases) (**Figure 46**). Under each step, insert SQL code to execute the four stored procedure for the backup process. Finally, we set the job to perform once a day under schedule.

[Step 6. Put Everything under the SSIS Solution](file:///C:\Users\Wayne\Desktop\INFO498-Final-WayneWang\Documents\INFO%20498%20Final.doc#_Toc521417671)

**A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated**

**Figure 47 Figure 48**

The final step for this final project is to put every files needed for this project put into corresponding folders and put these folders inside the real local folder of this Visual Studio solution. Then in Visual Studio, we add these files into the solution and create corresponding virtual folders in the solution. (**Figure 47 & 48**)

After organizing all files in the local folders and in the virtual folders, make sure the path of each file is indicated based on the solution folder, NOT local paths.

**NOW, WE ARE ALL DONE WITH THE FINAL PROJECT**

**Summary**

For this final project, we perform ETL process from flat files to the ‘Patients’ database. Then we perform ETL process from ‘Patients’ and ‘DoctorsSchedules’ databases to the DW database, ‘DWClinicReportData’. After checking the expected results, we create a SSIS project to perform the same ETL tasks. Then we create 4 stored procedures that perform the backup tasks for each of the three databases. After creating the stored procedures, we create a SQL job to perform daily backup tasks. In the end, we put all files and their corresponding folders in both the local folder of the solution and the solution folder within Visual Studio.

After all, this is a typical and relatively standard process of real life ETL process use.