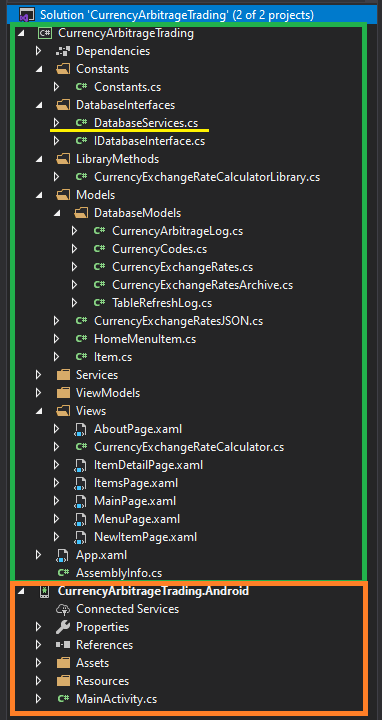
**Currency Arbitrage Trading – Android Application Design & Implementation**

**Solution: “CurrencyArbitrageTrading”** (Visual Studio 2019 Community Edition with Xamarin for Android Mobile Application Development)

We have structured our “CurrencyArbitrageTrading” Solution into 2 Projects.

**Project #1. “CurrencyArbitrageTrading”:** This project is common to all platforms like Android, iOS, Windows, etc. and it is composed the base models, service/library methods, constants, and common view models & views, etc. The primary SQLite database interactions are performed in the “**DatabaseServices.cs**” C# class file and is explained in detail below.

**Project #2. “CurrencyArbitrageTrading.Android”:** This project is specific to releasing the solution to Android platform. This also includes any customizations specific to Android platforms, platform specific navigations, etc.

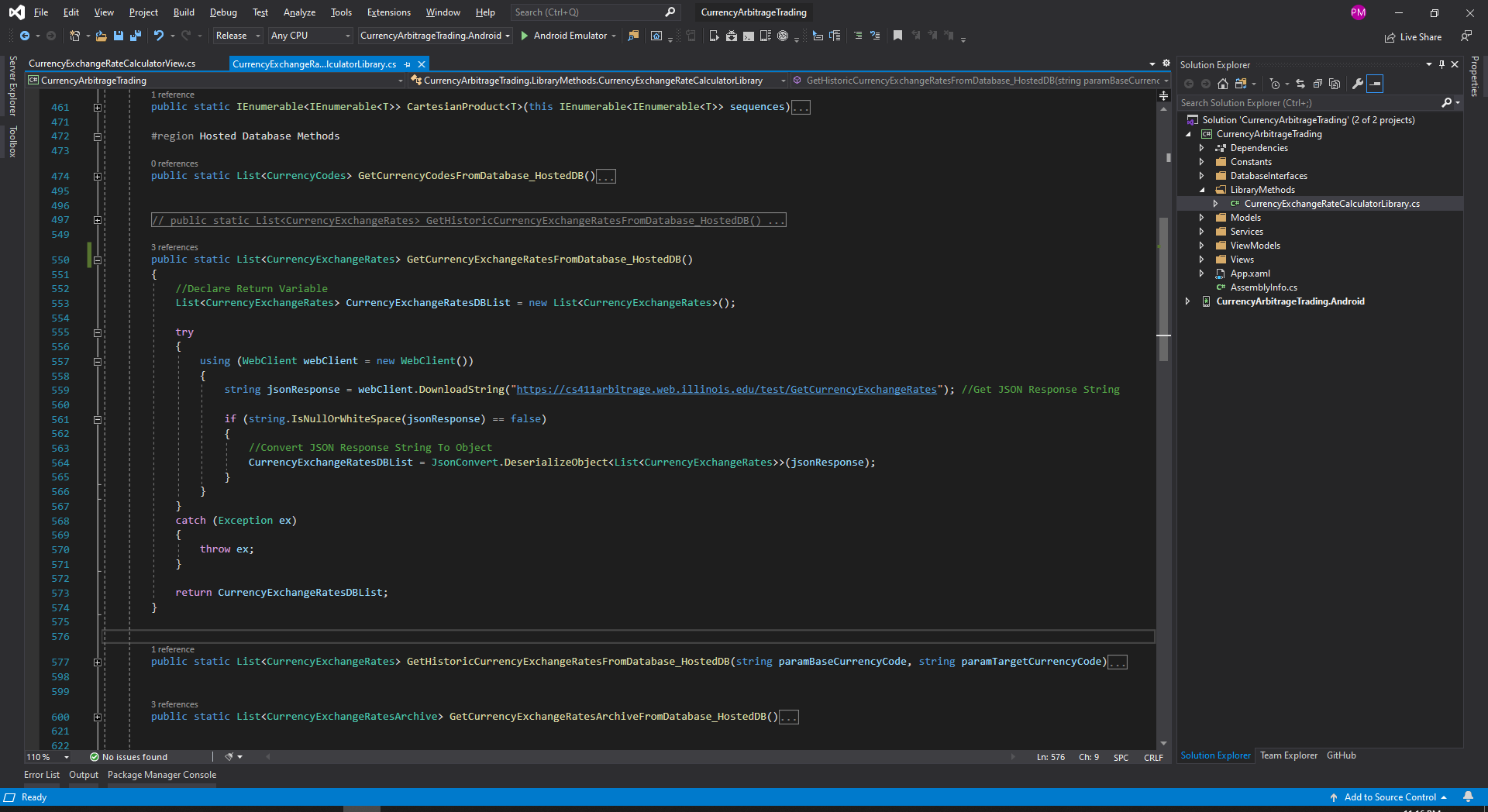


**Hosted Database Service Methods / Endpoints:** The following are the endpoints that interacts with the Hosted SQLite database.

1. **GetCurrencyExchangeRatesFromDatabase\_HostedDB:** This api endpoint gets the Latest Currency Exchange Rates (most recent / latest value) from the CurrencyExchangeRates table on the Hosted database using the API URL [**https://cs411arbitrage.web.illinois.edu/test/GetCurrencyExchangeRates**](https://cs411arbitrage.web.illinois.edu/test/GetCurrencyExchangeRates)

This method selects records from CurrencyExchangeRates table and returns them as a JSON response to the Android application.

Please refer to the below screenshots.

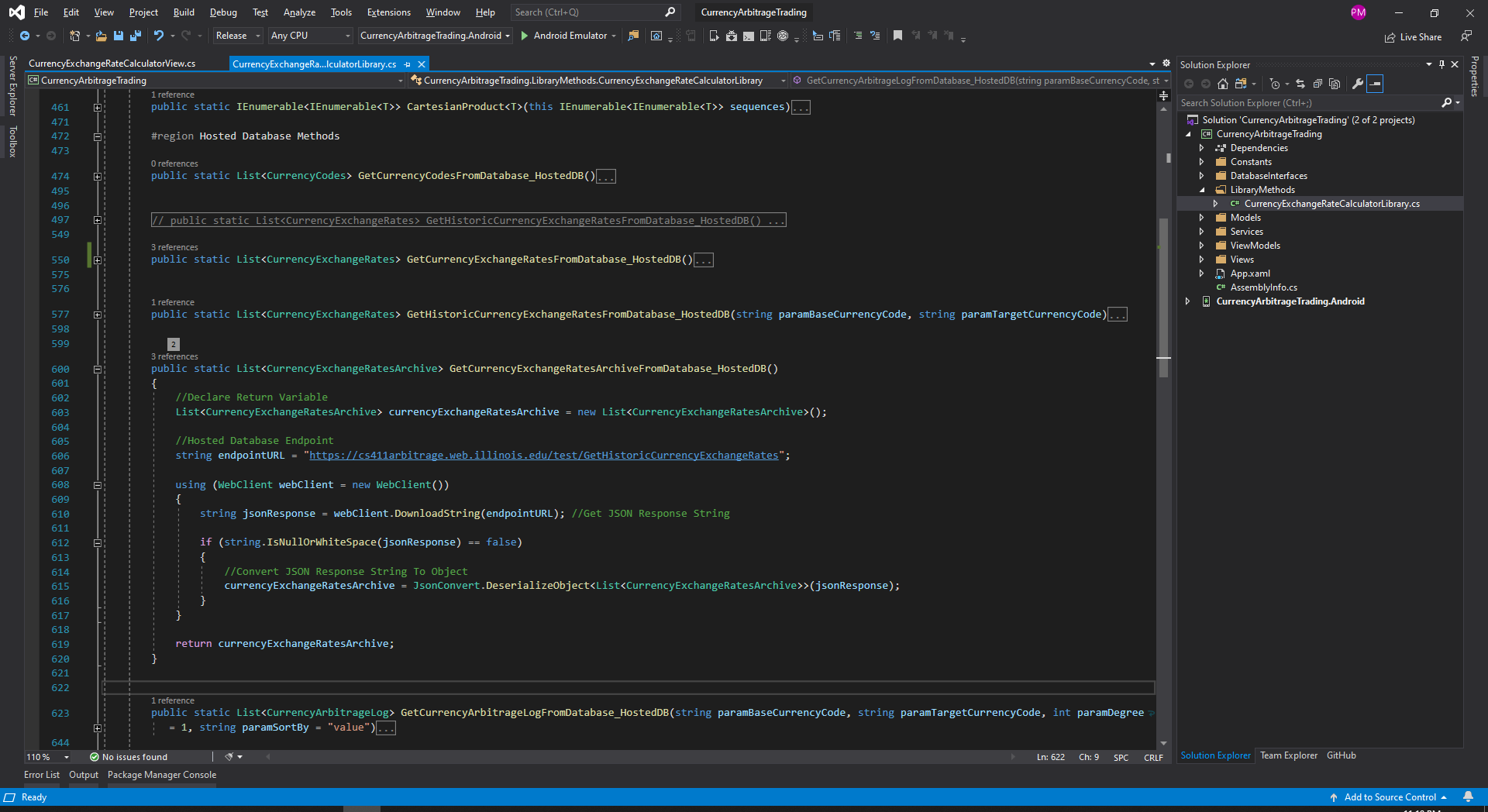


Alternate additional endpoints were also created to get the latest currency exchange rates from the CurrencyExchangeRates table that can filter the records based on the latest date the exchange rates data are refreshed with.

[**https://cs411arbitrage.web.illinois.edu/test/GetLatestCurrencyExchangeRates**](https://cs411arbitrage.web.illinois.edu/test/GetLatestCurrencyExchangeRates)

1. **GetCurrencyExchangeRatesArchiveFromDatabase\_HostedDB:** This api endpoint gets the Historic Currency Exchange Rates (from last 30 days) from the CurrencyExchangeRatesArchive table on the Hosted database using the API URL [**https://cs411arbitrage.web.illinois.edu/test/GetHistoricCurrencyExchangeRates**](https://cs411arbitrage.web.illinois.edu/test/GetHistoricCurrencyExchangeRates)

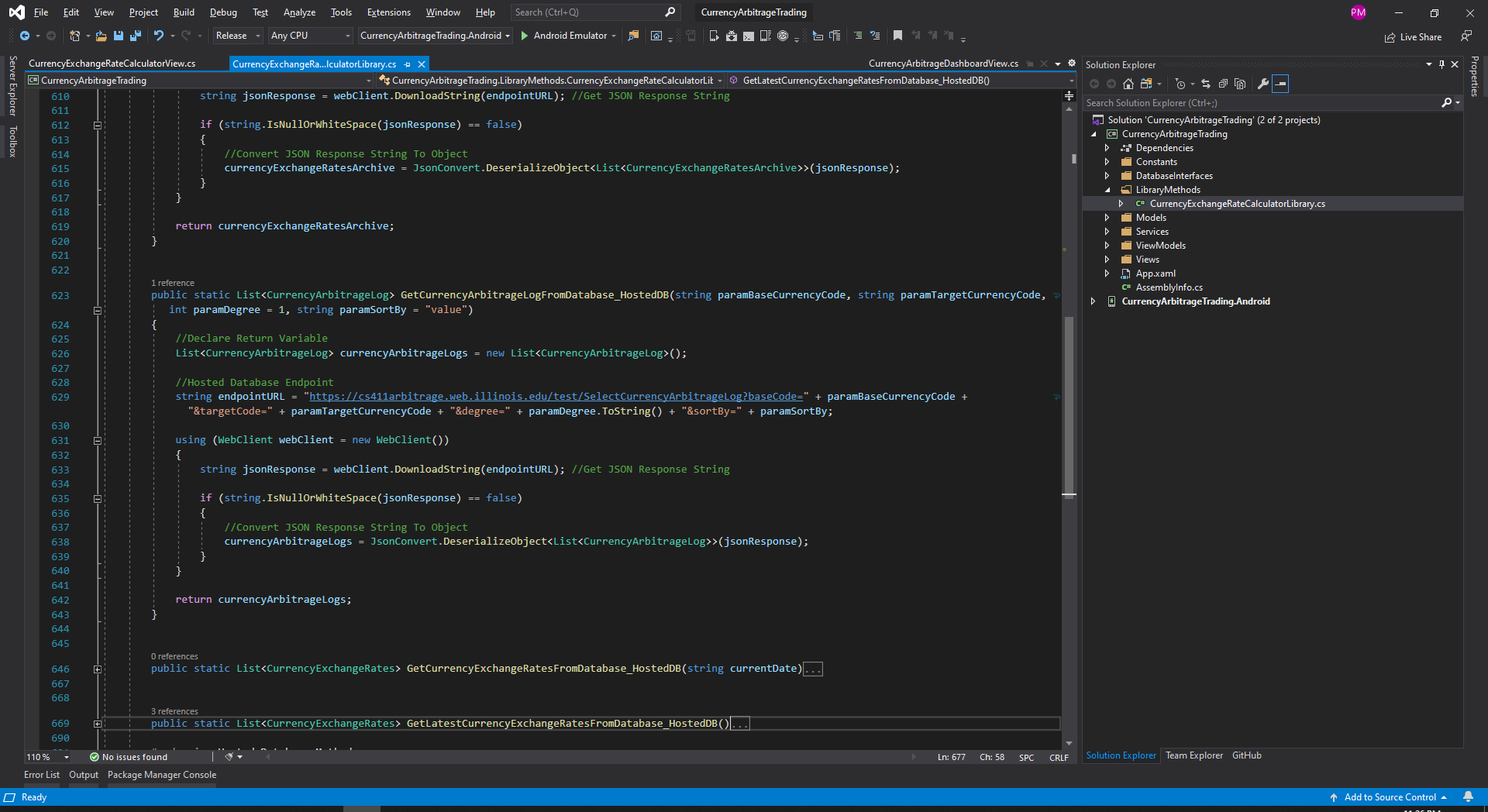
This method selects records from CurrencyExchangeRatesArchive table and returns them as a JSON response to the Android application.

Please refer to the below screenshots.

1. **GetCurrencyArbitrageLogFromDatabase\_HostedDB:** This api endpoint gets the latest Currency Arbitrage Log Rates (most recent / latest value) from the CurrencyArbitrageLog table & CurrencyExchangeRates tables (using CurrencyArbitrageLogView) on the Hosted database using the API URL [**https://cs411arbitrage.web.illinois.edu/test/SelectCurrencyArbitrageLog**](https://cs411arbitrage.web.illinois.edu/test/SelectCurrencyArbitrageLog)

This method accepts the BaseCurrencyCode, TargetCurrencyCode, Degree for Arbitrage and Sort By Column Name as its input parameters and selects the latest arbitrage records from CurrencyArbitrageLogView and returns them as a JSON response to the Android application.

Example: <https://cs411arbitrage.web.illinois.edu/test/SelectCurrencyArbitrageLog?baseCode=EUR&targetCode=USD&degree=3>

Please refer to the below screenshots.

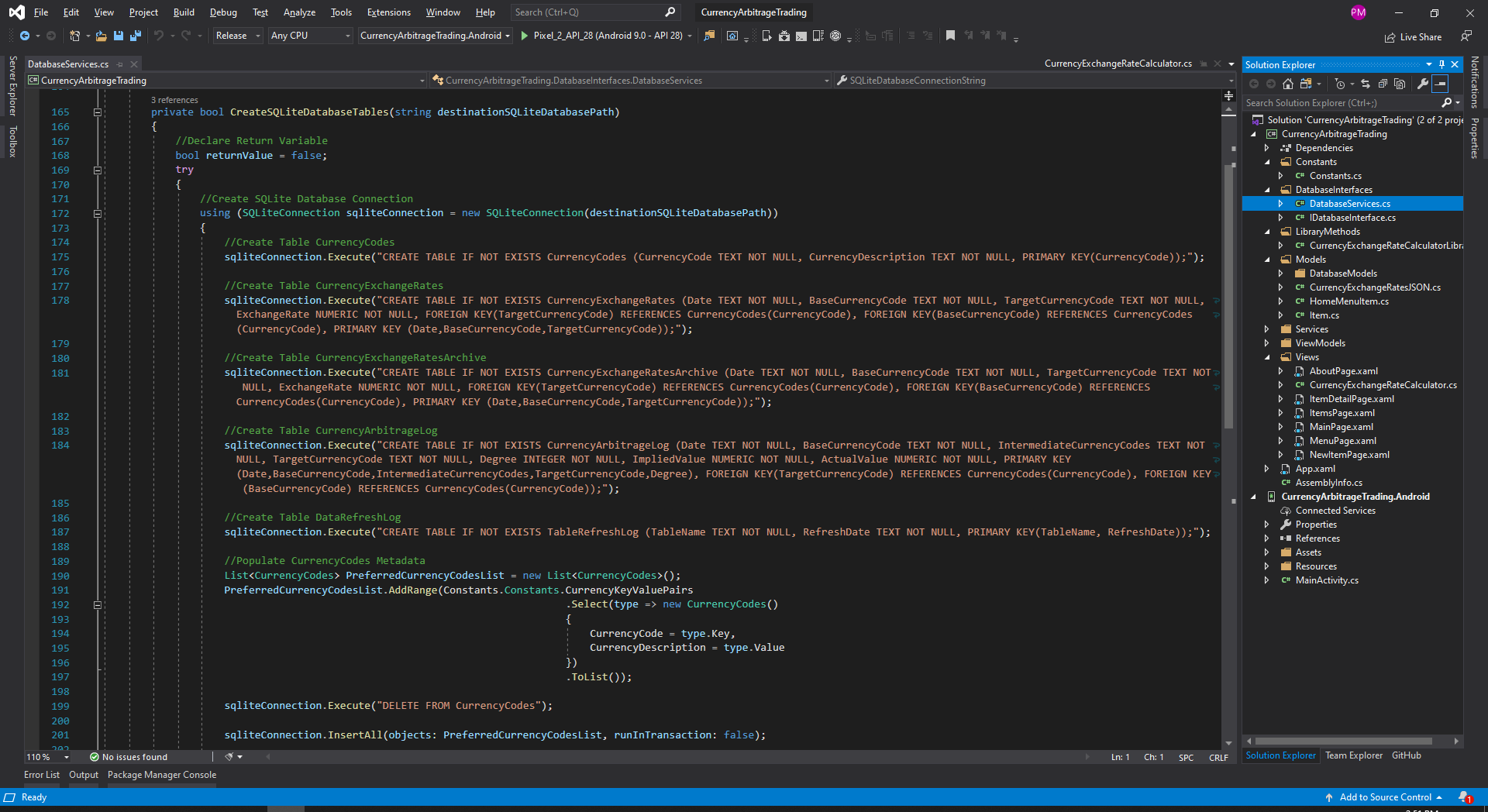
**Local Database Service Methods:** The following are the methods that interacts with the Local SQLite database.

* **Creating SQLite Database Tables & Inserting Base Records InTo Tables**

1. CreateSQLiteDatabaseTables: This method creates the database tables (only if the tables doesn’t exists) in the SQLite database using the "CREATE TABLE IF NOT EXISTS” statements.

This method also deletes and inserts data into the metadata table like “CurrencyCodes” using the DELETE and Insert and InsertAll statements. Please refer to the below screenshots.

|  |
| --- |
| **Statements Executed On SQLite Database** |
| //Create Table CurrencyCodes  sqliteConnection.Execute("CREATE TABLE IF NOT EXISTS CurrencyCodes (CurrencyCode TEXT NOT NULL, CurrencyDescription TEXT NOT NULL, PRIMARY KEY(CurrencyCode));");  //Create Table CurrencyExchangeRates  sqliteConnection.Execute("CREATE TABLE IF NOT EXISTS CurrencyExchangeRates (Date TEXT NOT NULL, BaseCurrencyCode TEXT NOT NULL, TargetCurrencyCode TEXT NOT NULL, ExchangeRate NUMERIC NOT NULL, FOREIGN KEY(TargetCurrencyCode) REFERENCES CurrencyCodes(CurrencyCode), FOREIGN KEY(BaseCurrencyCode) REFERENCES CurrencyCodes(CurrencyCode), PRIMARY KEY (Date,BaseCurrencyCode,TargetCurrencyCode));");  //Create Table CurrencyExchangeRatesArchive  sqliteConnection.Execute("CREATE TABLE IF NOT EXISTS CurrencyExchangeRatesArchive (Date TEXT NOT NULL, BaseCurrencyCode TEXT NOT NULL, TargetCurrencyCode TEXT NOT NULL, ExchangeRate NUMERIC NOT NULL, FOREIGN KEY(TargetCurrencyCode) REFERENCES CurrencyCodes(CurrencyCode), FOREIGN KEY(BaseCurrencyCode) REFERENCES CurrencyCodes(CurrencyCode), PRIMARY KEY (Date,BaseCurrencyCode,TargetCurrencyCode));");  //Create Table CurrencyArbitrageLog  sqliteConnection.Execute("CREATE TABLE IF NOT EXISTS CurrencyArbitrageLog (Date TEXT NOT NULL, BaseCurrencyCode TEXT NOT NULL, IntermediateCurrencyCodes TEXT NOT NULL, TargetCurrencyCode TEXT NOT NULL, Degree INTEGER NOT NULL, ImpliedValue NUMERIC NOT NULL, ActualValue NUMERIC NOT NULL, PRIMARY KEY (Date,BaseCurrencyCode,IntermediateCurrencyCodes,TargetCurrencyCode,Degree), FOREIGN KEY(TargetCurrencyCode) REFERENCES CurrencyCodes(CurrencyCode), FOREIGN KEY (BaseCurrencyCode) REFERENCES CurrencyCodes(CurrencyCode));");  //Create Table DataRefreshLog  sqliteConnection.Execute("CREATE TABLE IF NOT EXISTS TableRefreshLog (TableName TEXT NOT NULL, RefreshDate TEXT NOT NULL, PRIMARY KEY(TableName, RefreshDate));");  //Populate CurrencyCodes Metadata  List<CurrencyCodes> PreferredCurrencyCodesList = new List<CurrencyCodes>();  PreferredCurrencyCodesList.AddRange(Constants.Constants.CurrencyKeyValuePairs  .Select(type => new CurrencyCodes()  {  CurrencyCode = type.Key,  CurrencyDescription = type.Value  })  .ToList());  sqliteConnection.Execute("DELETE FROM CurrencyCodes");  sqliteConnection.InsertAll(objects: PreferredCurrencyCodesList, runInTransaction: false); |



* **Select Records From SQLite Tables**

1. **GetCurrencyCodesFromDatabase:** Gets the list of CurrencyCodes using the statement below from the SQLite Database and assigns them to local object CurrenCodes within the application and returns them to the caller.

|  |
| --- |
| currencyCodes = sqliteConnection.Query<CurrencyCodes>("SELECT CurrencyCode, CurrencyDescription FROM CurrencyCodes"); |

1. **GetHistoricCurrencyExchangeRatesFromDatabase:** Gets the list of ALL Historic Exchange Rates using the statement below and assigns them to the local object CurrencyExchangeRates within the application and returns them to the caller.

|  |
| --- |
| CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates"); |

1. **GetHistoricCurrencyExchangeRatesFromDatabase with Base & Target Currency as Input parameters:** Gets the list of ALL Historic Exchange Rates for the supplied input parameters - Base & Target Currency using the statement below and assigns them to the local object CurrencyExchangeRates within the application and returns them to the caller.

|  |
| --- |
| CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates WHERE BaseCurrencyCode = \"" + paramBaseCurrencyCode + "\" AND TargetCurrencyCode = \"" + paramTargetCurrencyCode + "\" ORDER BY DATE(Date) LIMIT " + Constants.Constants.HistoryDays.ToString()); |

1. **GetCurrencyExchangeRatesArchiveFromDatabase:** Gets the list of Archive Exchange Rates using the statement below from the SQLite Database and assigns them to local object CurrencyExchangeRatesArchive within the application and returns them to the caller.

|  |
| --- |
| currencyExchangeRatesArchive = sqliteConnection.Query<CurrencyExchangeRatesArchive>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRatesArchive"); |

1. **GetCurrencyArbitrageLogFromDatabase:** Gets the list of Arbitrage Log Data using the statement below from the SQLite Database and assigns them to local object CurrencyArbitrageLog within the application and returns them to the caller.

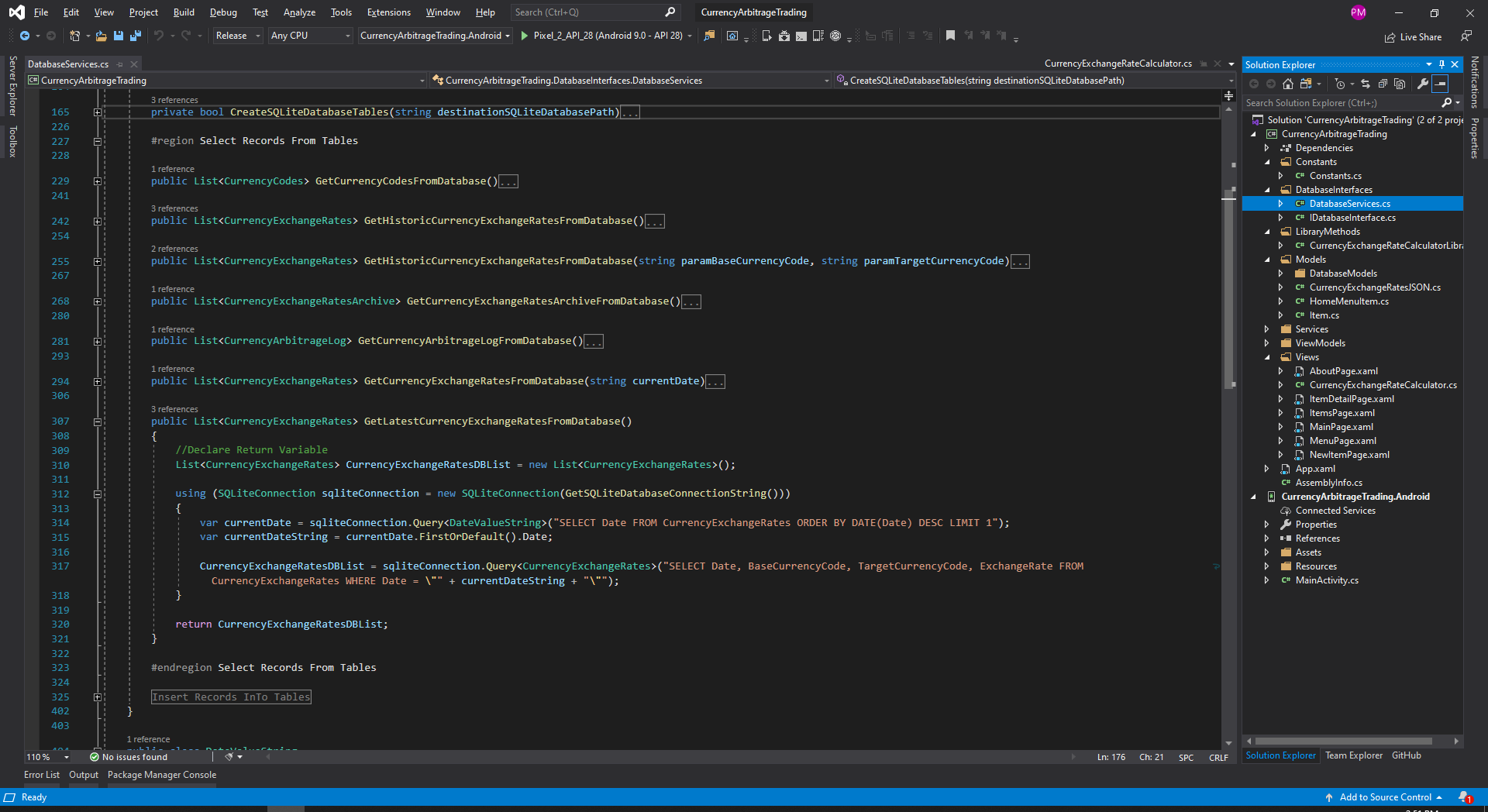
|  |
| --- |
| currencyArbitrageLogs = sqliteConnection.Query<CurrencyArbitrageLog>("SELECT Date, BaseCurrencyCode, IntermediateCurrencyCodes, TargetCurrencyCode, Degree, ImpliedValue, ActualValue FROM CurrencyArbitrageLog"); |

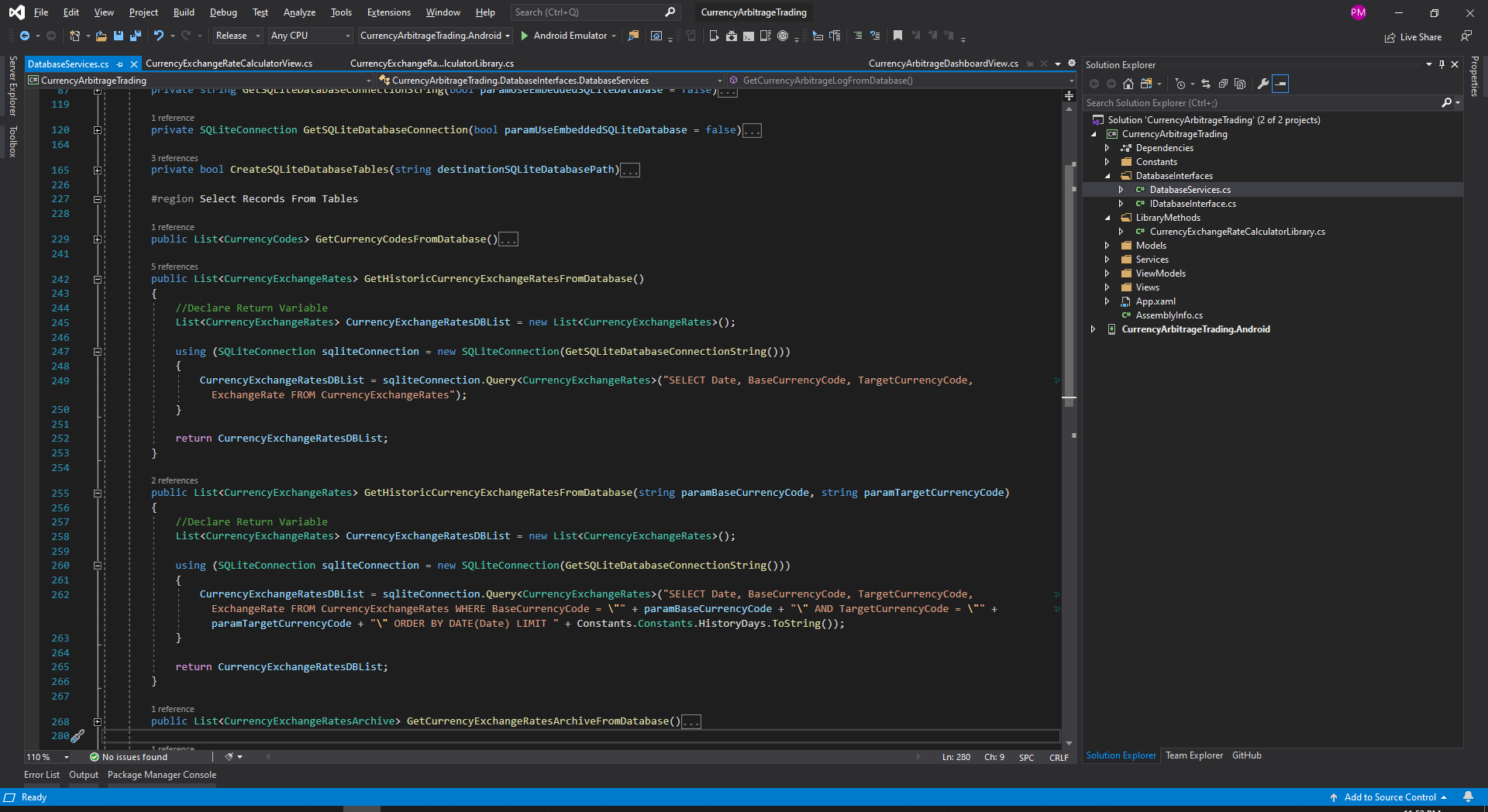
1. **GetCurrencyExchangeRatesFromDatabase with Current Date as Input parameter:** Gets the list of ALL latest / current Currency Exchange Rates based on supplied input date as filter condition / parameter using the statement below from the SQLite Database and assigns them to local object CurrencyExchangeRates within the application and returns them to the caller.

|  |
| --- |
| CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates WHERE Date = \"" + currentDate + "\""); |

1. **GetLatestCurrencyExchangeRatesFromDatabase:** Gets the list of ALL latest / current Currency Exchange Rates based on latest Date value on the table using the statement below from the SQLite Database and assigns them to local object CurrencyExchangeRates within the application and returns them to the caller.

|  |
| --- |
| var currentDate = sqliteConnection.Query<DateValueString>("SELECT Date FROM CurrencyExchangeRates ORDER BY DATE(Date) DESC LIMIT 1");  var currentDateString = currentDate.FirstOrDefault().Date;  CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates WHERE Date = \"" + currentDateString + "\""); |





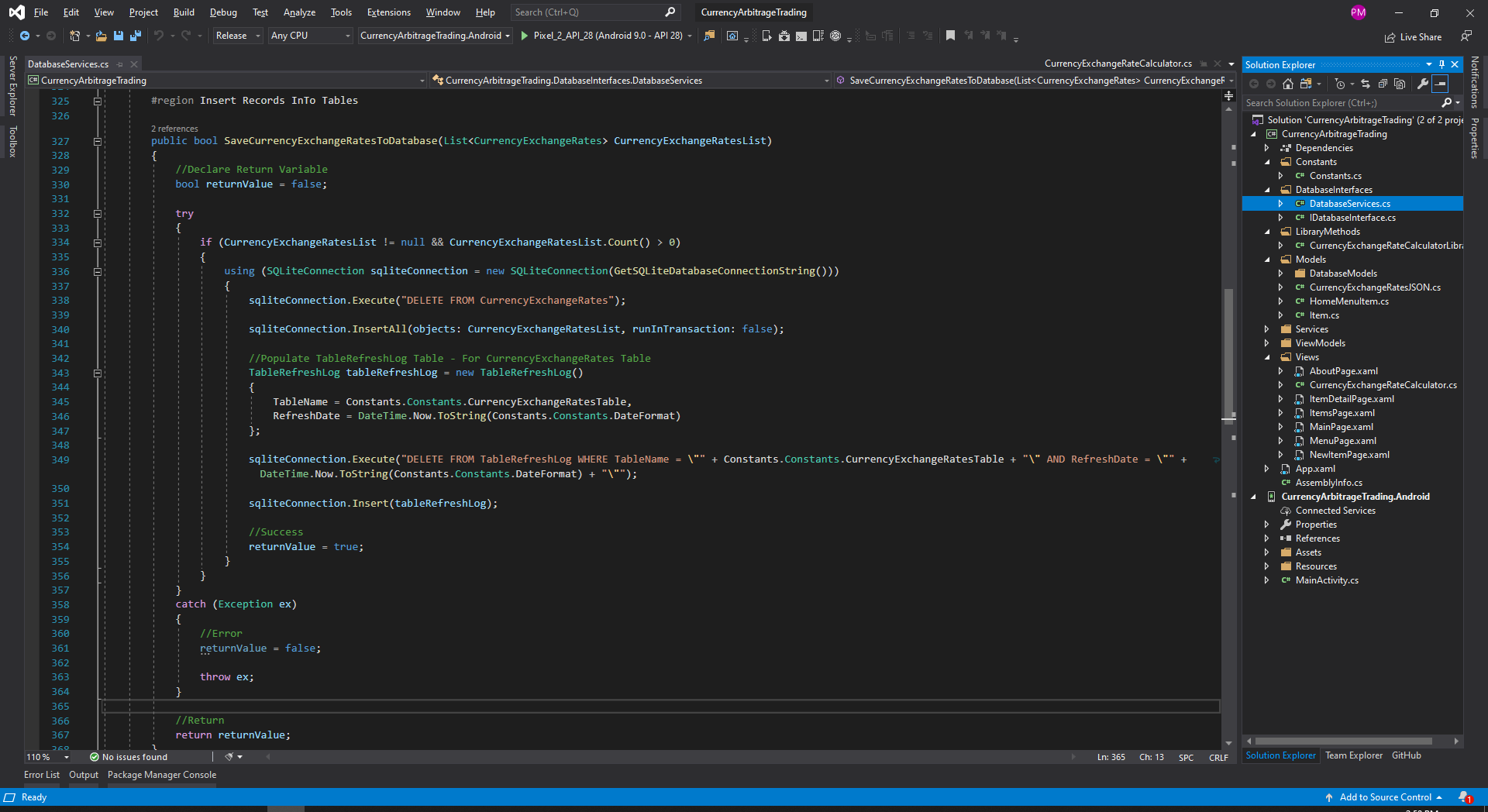
* **Insert Records InTo SQLite Tables**

1. **SaveCurrencyExchangeRatesToDatabase with Input parameter CurrencyExchangeRates:** This method Saves the supplied list of CurrencyExchangeRates into the SQLite database and inserts them to the CurrencyExchangeRates table. Before insert this method deletes the existing records and then performs the inserts. This method also updates the RefreshLog table with the latest RefreshDate value.

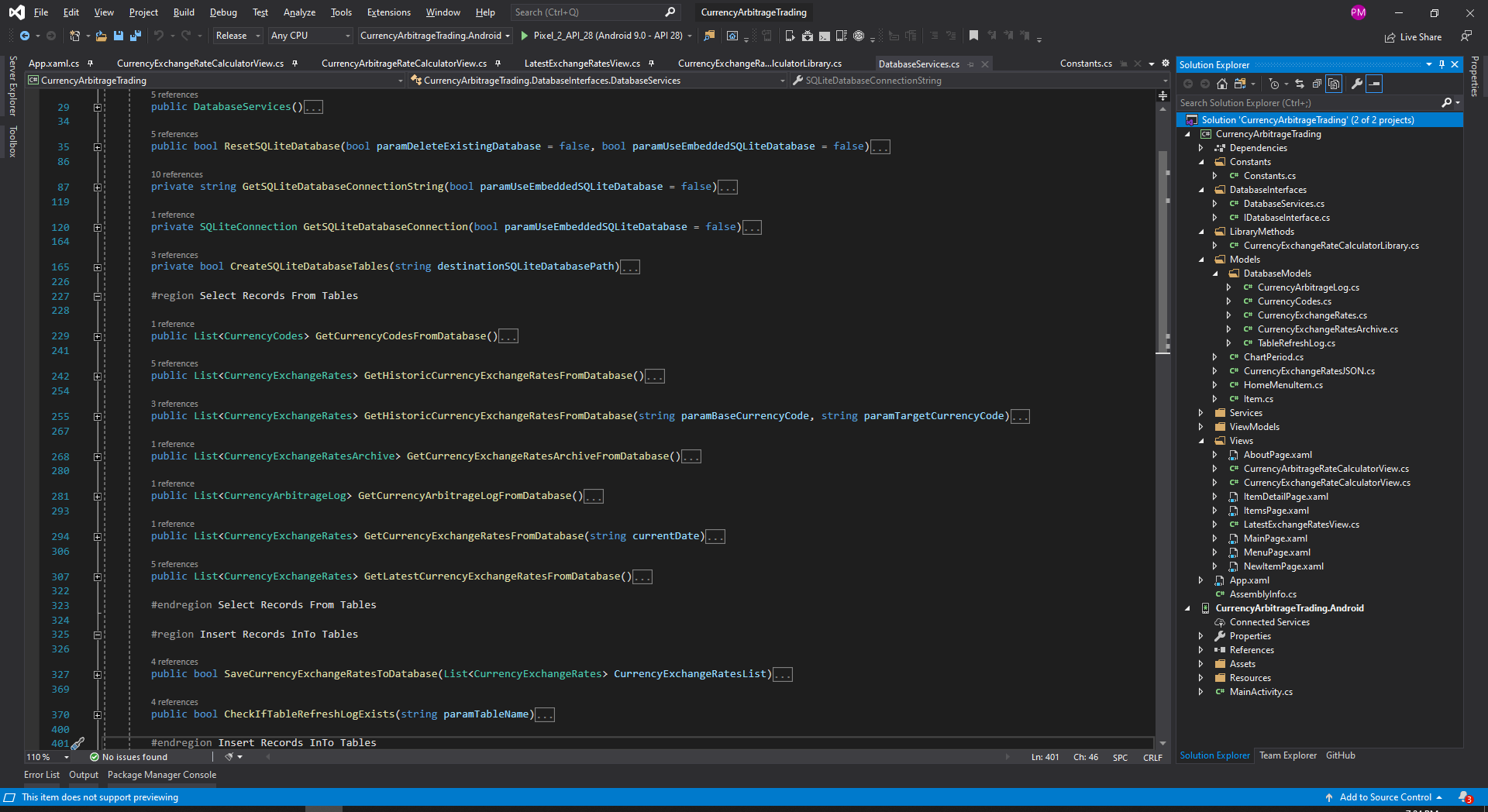
|  |
| --- |
| sqliteConnection.Execute("DELETE FROM CurrencyExchangeRates");  sqliteConnection.InsertAll(objects: CurrencyExchangeRatesList, runInTransaction: false);  ...  ...  sqliteConnection.Execute("DELETE FROM TableRefreshLog WHERE TableName = \"" + Constants.Constants.CurrencyExchangeRatesTable + "\" AND RefreshDate = \"" + DateTime.Now.ToString(Constants.Constants.DateFormat) + "\"");  sqliteConnection.Insert(tableRefreshLog); |

1. **CheckIfTableRefreshLogExists:** This method checks the RefreshLog table for the presence of latest record and returns the integer value - count of records, if exists.

|  |
| --- |
| recordCount = sqliteConnection.ExecuteScalar<int>("SELECT COUNT(1) FROM TableRefreshLog WHERE TableName = \"" + paramTableName + "\" AND RefreshDate = \"" + currentDate + "\""); |



**List of Methods In Database Service Layer**



|  |  |
| --- | --- |
| **Method Name** | **C# Code Reference** |
| **GetCurrencyCodesFromDatabase**  Executes the highlighted SQL select statement and returns back the response to the caller | public List<CurrencyCodes> GetCurrencyCodesFromDatabase()  {  //Declare Return Variable  List<CurrencyCodes> currencyCodes = new List<CurrencyCodes>();  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  currencyCodes = sqliteConnection.Query<CurrencyCodes>("SELECT CurrencyCode, CurrencyDescription FROM CurrencyCodes");  }  return currencyCodes;  } |
| **GetHistoricCurrencyExchangeRatesFromDatabase**  Executes the highlighted SQL select statement and returns back the response to the caller | public List<CurrencyExchangeRates> GetHistoricCurrencyExchangeRatesFromDatabase()  {  //Declare Return Variable  List<CurrencyExchangeRates> CurrencyExchangeRatesDBList = new List<CurrencyExchangeRates>();  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates");  }  return CurrencyExchangeRatesDBList;  } |
| **GetHistoricCurrencyExchangeRatesFromDatabase**  Input Parameters:  paramBaseCurrencyCode  paramTargetCurrencyCode  Executes the SQL select statement with the supplied input parameters and returns back the response to the caller | public List<CurrencyExchangeRates> GetHistoricCurrencyExchangeRatesFromDatabase(string paramBaseCurrencyCode, string paramTargetCurrencyCode)  {  //Declare Return Variable  List<CurrencyExchangeRates> CurrencyExchangeRatesDBList = new List<CurrencyExchangeRates>();  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates WHERE BaseCurrencyCode = \"" + paramBaseCurrencyCode + "\" AND TargetCurrencyCode = \"" + paramTargetCurrencyCode + "\" ORDER BY DATE(Date) LIMIT " + Constants.Constants.HistoryDays.ToString());  }  return CurrencyExchangeRatesDBList;  } |
| **GetCurrencyExchangeRatesArchiveFromDatabase** Executes the highlighted SQL select statement and returns back the response to the caller | public List<CurrencyExchangeRatesArchive> GetCurrencyExchangeRatesArchiveFromDatabase()  {  //Declare Return Variable  List<CurrencyExchangeRatesArchive> currencyExchangeRatesArchive = new List<CurrencyExchangeRatesArchive>();  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  currencyExchangeRatesArchive = sqliteConnection.Query<CurrencyExchangeRatesArchive>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRatesArchive");  }  return currencyExchangeRatesArchive;  } |
| **GetCurrencyArbitrageLogFromDatabase**  Executes the highlighted SQL select statement and returns back the response to the caller | public List<CurrencyArbitrageLog> GetCurrencyArbitrageLogFromDatabase()  {  //Declare Return Variable  List<CurrencyArbitrageLog> currencyArbitrageLogs = new List<CurrencyArbitrageLog>();  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  currencyArbitrageLogs = sqliteConnection.Query<CurrencyArbitrageLog>("SELECT Date, BaseCurrencyCode, IntermediateCurrencyCodes, TargetCurrencyCode, Degree, ImpliedValue, ActualValue FROM CurrencyArbitrageLog");  }  return currencyArbitrageLogs;  } |
| **GetCurrencyExchangeRatesFromDatabase**  Input Parameters:  currentDate  Executes the SQL select statement with the supplied input parameters and returns back the response to the caller | public List<CurrencyExchangeRates> GetCurrencyExchangeRatesFromDatabase(string currentDate)  {  //Declare Return Variable  List<CurrencyExchangeRates> CurrencyExchangeRatesDBList = new List<CurrencyExchangeRates>();  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates WHERE Date = \"" + currentDate + "\"");  }  return CurrencyExchangeRatesDBList;  } |
| **GetLatestCurrencyExchangeRatesFromDatabase** Executes the highlighted SQL select statement and returns back the response to the caller. Gets the Latest ExchangeRates from the database that is currently available in the database. | public List<CurrencyExchangeRates> GetLatestCurrencyExchangeRatesFromDatabase()  {  //Declare Return Variable  List<CurrencyExchangeRates> CurrencyExchangeRatesDBList = new List<CurrencyExchangeRates>();  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  var currentDate = sqliteConnection.Query<DateValueString>("SELECT Date FROM CurrencyExchangeRates ORDER BY DATE(Date) DESC LIMIT 1");  var currentDateString = currentDate.FirstOrDefault().Date;  CurrencyExchangeRatesDBList = sqliteConnection.Query<CurrencyExchangeRates>("SELECT Date, BaseCurrencyCode, TargetCurrencyCode, ExchangeRate FROM CurrencyExchangeRates WHERE Date = \"" + currentDateString + "\"");  }  return CurrencyExchangeRatesDBList;  } |
| **DATABASE INSERT OPERATION** | |
| **SaveCurrencyExchangeRatesToDatabase** | Insert ExchangeRates InTo **CurrencyExchangeRates** Table  Insert/Update RefreshLog InTo **TableRefreshLog** Table - If NO Records Exists for “CurrencyExchangeRates” table, then Insert with latest Refresh Date. If Records Exists, then Update just the RefreshDate Column. |
| **CheckIfTableRefreshLogExists**  Input Parameters:  paramTableName  Executes the SQL select statement with the supplied input parameters and returns back the boolean response to the caller.  **If any record exists for the supplied input parameter (TableName) for the Current Date, then return TRUE, else return FALSE.** | public bool CheckIfTableRefreshLogExists(string paramTableName)  {  //Declare Return Variable  bool returnValue = false;  int recordCount = 0;  string currentDate = DateTime.Now.Date.ToString(Constants.Constants.DateFormat);  try  {  using (SQLiteConnection sqliteConnection = new SQLiteConnection(GetSQLiteDatabaseConnectionString()))  {  recordCount = sqliteConnection.ExecuteScalar<int>("SELECT COUNT(1) FROM TableRefreshLog WHERE TableName = \"" + paramTableName + "\" AND RefreshDate = \"" + currentDate + "\"");  if (recordCount > 0)  {  //Success  returnValue = true;  }  }  }  catch (Exception ex)  {  //Error  returnValue = false;  throw ex;  }  //Return  return returnValue;  } |