Programming 3 Assignment 7

Assignment Overview

- The specifications associated with this assignment will incorporate Batch Record Processing and File I/O to the BIT College application. The batch jobs will execute as a result of receiving one or more files from various programs within BITCollege. These files will all be received in a standard format, described below. Processing will be achieved through *LINQ-to-XML* combined with functionality developed in earlier assignments.
- Work associated with this assignment will reinforce the student's learning in the following areas:
  - Use LINQ to XML Queries to read and process incoming xml files
  - Write to outgoing text files
  - Perform appropriate batch transaction processing
  - Perform extensive validation and error handling
    - Ensuring any errors that take place are logged, but that processing continues
  - Use WCF Web Services in a Windows project

Evaluation

- The evaluation of this assignment will be in the form of a Code Review and Run Time Demonstration.
- During the Code Review evaluation will be based on:
  - Comments
  - Documentation
  - Adherence to Standards – and Pattern Best Practices
  - Coding choices (i.e. refactoring, no repeated code, readability etc…)
  - Implementation of Requirements
  - Verbal discussion of code
- During the Run-Time Demonstration evaluation will be based on:
  - A variety of run-time tests

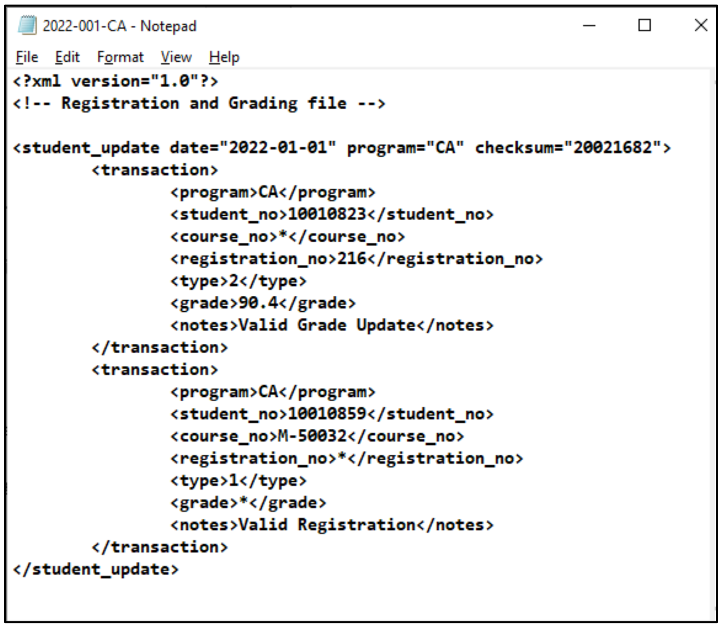
Specifications

Programming 3 Assignment 7

Input Files

- Develop input files to be used for testing the batch processing component of this system. See the sample file below.
- The input files developed should be in the following format. An example is shown below:

    - File type: xml file(s)
    - File Name: 4_digit_year-3_digit_day_of_year-program_acronym.extension
    - Example: 2020-159-VT.xml
    - Elements:
        - **<student_update>** Containing the following attributes:
            - **date**: yyyy-mm-dd (must be today's date)
            - **program**: represents the program acronym
            - **checksum**: represents the sum of the student number items within each transaction element that follows
        - **<transaction>:** This element will repeat for each transaction in the file and will contain child elements as follows:
            - **<program>**: represents the academic program acronym
            - **<student_no>**: represents the student number to which the transaction is to be applied
            - **<course_no>**: represents the course number ('*' if grading type)
            - **<registration_no>**:represents the registration number ('*' if registration type)
            - **<type>**: represents the transaction type (registration = 1, grading = 2)
            - **<grade>**: represents the grade applied to the course ('*' if registration type)
            - **<notes>**: represents a notes field associated with the transaction



```
<?xml version="1.0"?>
<!-- Registration and Grading file -->

<student_update date="2022-01-01" program="CA" checksum="20021682">
	<transaction>
		<program>CA</program>
		<student_no>10010823</student_no>
		<course_no>*</course_no>
		<registration_no>216</registration_no>
		<type>2</type>
		<grade>90.4</grade>
		<notes>Valid Grade Update</notes>
	</transaction>
	<transaction>
		<program>CA</program>
		<student_no>10010859</student_no>
		<course_no>M-50032</course_no>
		<registration_no>*</registration_no>
		<type>1</type>
		<grade>*</grade>
		<notes>Valid Registration</notes>
	</transaction>
</student_update>
```

Figure 1

- Ensure that as you progress through the assignment, you introduce erroneous files and transactions to ensure that the exception handling requirement of this assignment is met.
- Store this (these) file(s) in the bin/debug folder of the BITCollegeWindows project.

Log Files

- A log file (.txt format) will be produced for each Academic Program in which a transmission has been executed. The log file will contain details of every successful and unsuccessful transaction processed within the transmission.
- When writing to the log file, use appropriate escape sequences to make the log file readable. The table below shows some commonly used escape sequences:

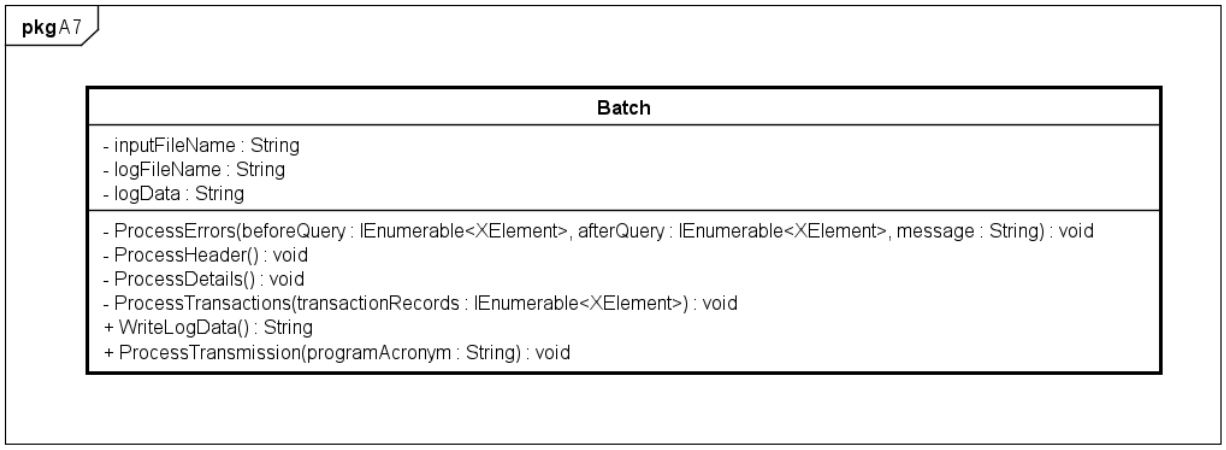
Programming 3 Assignment 7

Table 1

Escape Sequence	*Description*
\b	*Backspace*
\t	*(Horizontal) Tab*
\n	*New Line Feed*
\v	*Vertical Tab*
\f	*Form Feed*
\r	*Carriage Return*
\	*Back slash*
\'	*Single quote (grave accent)*
\"	*Double quote*

BITCollegeWindows Class: Batch

Given in the BITCollegeWindows project is a class called Batch based on the figure below.



```
- inputFileName : String
- logFileName : String
- logData : String
- ProcessErrors(beforeQuery : IEnumerable<XElement>, afterQuery : IEnumerable<XElement>, message : String) : void
- ProcessHeader() : void
- ProcessDetails() : void
- ProcessTransactions(transactionRecords : IEnumerable<XElement>) : void
+ WriteLogData() : String
+ ProcessTransmission(programAcronym : String) : void
```

Figure 2 – Batch class (electronic version in the Assignment 7 Files folder)

Attribute Descriptions

- inputFileName: This string will represent the name of the file being processed.
- logFileName: This string will represent the name of the log file that corresponds with the file being processed.
- logData: This string will represent all data to be written to the log file that corresponds with the file being processed.

- Note: Feel free to define additional private variables as necessary when proceeding through the requirements below. Only declare these variables if class scope is needed.

Methods

- Use the given method skeletons to complete the requirements below:

ProcessTransmission

- This method will initiate the batch process by determining the appropriate filename and then proceeding with the header and detail processing.
- Formulate the inputFileName that is to be processed based on the filename format provided above.
    - Use properties of the DateTime class along with the programAcronym argument to create the file name.
    - Ensure that the .xml extension is included in this filename.
    - The format of the file is:
        - 4DigitYear-3DigitDayOfYear-AcademicProgramAcronym.xml
        - (E.g.: “2022-001-VT.xml”)
- Formulate logFileName that is to represent the name of the log file
    - The logFileName property is to be set to the value of the string “LOG” concatenated with the file name defined in the previous step, replacing the extension with .txt
        - (E.g.: “LOG 2022-001-VT.txt”)
- Use the File class to determine whether a file exists matching the inputFileName created above (not the LOG filename).
- If the file does not exist:
    - Append a relevant message to logData indicating that the file does not exist. Include the inputFileName in this message.
- If the file does exist:
    - Call the ProcessHeader method defined below.
    - Call the ProcessDetails method defined below.
    - If an exception occurs, append a relevant message to logData indicating the reason for the exception (use the Message property of the exception instance).

ProcessHeader

- This method is used to verify the attributes of the xml file's root element. If any of the attributes produce an error, the file is NOT to be processed.
  - If any of the validation described below fails, throw an exception with an appropriate message otherwise proceed with the validation.
  - **NOTE**: DO NOT catch the exception here. The exception will be caught up the call stack.

- Define an XDocument object and populate this object with the contents of the current file (inputFileName).
- Define an XElement object and populate this object with the data found within the root element (**student_update**) of the xml file.

Validation Requirements

- The XElement object must have 3 attributes.
  - **Note**: You can assume if there are 3 attributes, the attribute names are correct.
- The **date** attribute of the XElement object must be equal to today's date.
- The **program** attribute must match one of the academic program acronym values within the AcademicPrograms table of your BITCollege_FLContext database.
- The checksum attribute must match the sum of all **student_no** elements in the file.
  - **Note**: You may assume that all **student_no** element values in the file are numeric.
- If any of the above validation fails, throw an exception with an appropriate message.

ProcessDetails

- This method is used to verify the contents of the detail records in the input file. If any of the records produce an error, that record will be skipped, but the file processing will continue.
- Define an XDocument object and populate this object with the contents of the current file (inputFileName).

- Define an IEnumerable<XElement> *LINQ-to-XML* query against the XDocument object:
  - The result set should only include **transaction** elements.
  - This result set will be the basis of the next query.

Programming 3 Assignment 7

Validation Requirements

- The following validation must be completed in the order listed.
- After each round of validation, use the ProcessErrors method (defined below) to write all erroneous records to the logData property.
    - When calling ProcessErrors, include the result set of the previous query and the result set of the current query as arguments.
    - When calling ProcessErrors, include an appropriate error message based on the round of validation that has taken place.
    - **<u>NOTE</u>**: The ProcessErrors method has not yet been developed.
        - For now: Use the method stub created earlier.

- Each **transaction** element in the xml file must have 7 child elements. Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous query results
    - The result set should only include **transaction** elements that have 7 child elements.
    - **<u>Hint</u>**: use the Nodes().Count() methods
    - **<u>Note</u>**: You can assume if there are 7 child elements, the element names are correct.
    - This result set will be the basis of the next query.

- The **program** element must match the **program** attribute of the root element. Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous query results.
    - The result set should only include **transaction** elements whose **program** element matches the **program** attribute of the root element.
    - This result set will be the basis of the next query.

- The **type** element within each **transaction** element must be numeric. Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous query results.
    - The result set should only include **transaction** elements whose **type** element is numeric.
    - **<u>Reminder</u>**: The Utility project has an IsNumeric method that can be used.
    - This result set will be the basis of the next query.

- The **grade** element within each **transaction** element must be numeric **<u>or</u>** have the value of '*'. Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous query results.
    - The result set should only include **transaction** elements whose **grade** element is numeric or has a value of "*".
    - This result set will be the basis of the next query.

- The **type** element within each **transaction** element must have a value of 1 or 2. Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous query results
    - The result set should only include **transaction** elements whose **type** element is a 1 or 2.
    - This result set will be the basis of the next query.

- The **grade** element for course registrations (**type** element = 1) within each **transaction** element must have a value of "*". The **grade** element for course grading (**type** element = 2) within each **transaction** element must have a value between 0 and 100 inclusive.
- Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous query results.
    - The result set should only include **transaction** elements whose **type** element is 1 and **grade** element is "*", OR whose **type** element is 2 and **grade** element is between 0 and 100 inclusive.
    - This result set will be the basis of the next query.

- The **student_no** element within each **transaction** element must exist in the database.
- Define an IEnumerable<long> *LINQ-to-SQL Server* query against the Students table retrieving a list of all Student Numbers. The results of this query will be used to verify the transactions within the input file apply to valid students.
    - **<u>NOTE</u>**: It will be necessary to apply the .ToList() method to this result set.
- Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous *LINQ-to-XML* query results.
    - The result set should only include **transaction** elements whose **student_no** exists in the *LINQ-to-SQL Server* query containing a list of all Student Numbers in the database.
        - **<u>Hint</u>**: Use the Contains() query method.

- The **course_no** element within each **transaction** element must be a “*” for grading (type 2) or it must exist in the database for a registration (type 1).
- Define an IEnumerable<string> *LINQ-to-SQL Server* query against the Courses table retrieving a list of all Course Numbers. The results of this query will be used to verify the transactions within the input file apply to valid courses.
    - **NOTE**: It will be necessary to apply the .ToList() method to this result set.
- Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous *LINQ-to-XML* query results
    - The result set should only include **transaction** elements whose **course_no** is a “*” for type 2 transactions, or exists in the *LINQ-to-SQL Server* query containing a list of all Course Numbers in the database for type 1 transactions.
        - **Hint**: Use the Contains() query method

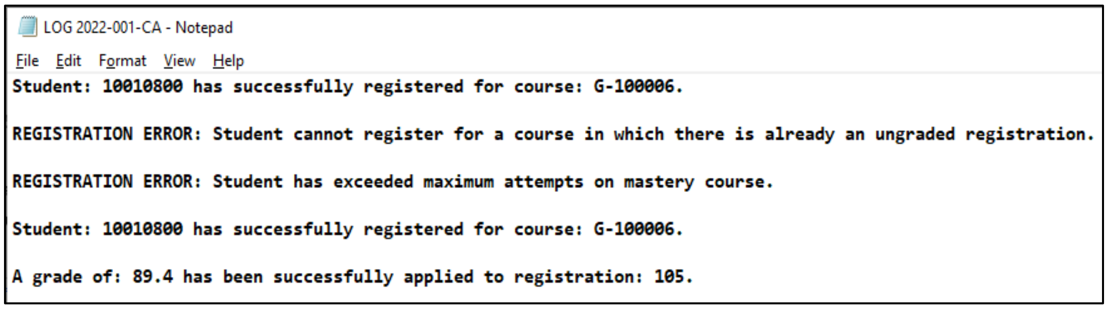
- The **registration_no** element within each **transaction** element must be a “*” for course registration (type 1) or it must exist in the database for grading (type 2).
- Define an IEnumerable<long> *LINQ-to-SQL Server* query against the Registrations table retrieving a list of all Registration Numbers. The results of this query will be used to verify the transactions within the input file apply to valid registrations.
    - **NOTE**: It will be necessary to apply the .ToList() method to this result set.
- Define an IEnumerable<XElement> *LINQ-to-XML* query against the previous *LINQ-to-XML* query results.
    - The result set should only include **transaction** elements whose **registration_no** is a “*” for type 1 transactions, or exists in the *LINQ-to-SQL Server* query containing a list of all Registration Numbers in the database for type 2 transactions.
        - **Hint**: Use the Contains() query method

- At this point, the records that remain after all of the above validation, should be error-free
- Call the ProcessTransactions method passing the error free result set.
    - **NOTE**: The ProcessTransactions method will be created next.
        - For now: Use the method skeleton created earlier.

ProcessTransactions

- This method is used to process all valid transaction records.
- Iterate through the collection of **transaction** elements passed to this method.
  - For the requirements that follow:
    - Extract values from the XElement objects in the collection as necessary in order to proceed.
    - Use *LINQ-to-SQL Server* as necessary to obtain additional values from the database in order to proceed.
  - Evaluate the **type** element:
    - If the **type** is a 1 (indicating registration)
      - Use the WCF Service to register the student into the specified course.
        - When calling the WCF Service method, use the Notes from the input file for the Notes argument.
      - If the transaction was successful (**Hint**: check the return code):
        - Append a relevant message to logData indicating that the transaction was successful. See the successful registration entries in the log file below.
      - If the transaction was unsuccessful:
        - Append a relevant message to logData indicating that the transaction was unsuccessful. **Reminder**: The Utility project has the RegisterError method with predefined error messages associated with failed registrations. See the unsuccessful registration entries in the log file (figure 3) below.
    - If the **type** is a 2 (indicating grading)
      - **Reminder**: Valid grades within the input file are between the values of 0 and 100. However, grades must be stored in the database must have values between 0 and 1. Ensure the appropriate values are updated to the database.
      - Use the WCF Service to update the student's grade.
        - When calling the WCF Service method, use the Notes from the input file for the Notes argument.
      - If the transaction was successful:

        - Append a relevant message to logData indicating that the transaction was successful. See the grade update in the log file below.
    - If the transaction was unsuccessful:
    - **Note**: No exceptions should take place.
        - Append a message to log data providing enough details about the exception to help you troubleshoot the cause.



```
Student: 10010800 has successfully registered for course: G-100006.

REGISTRATION ERROR: Student cannot register for a course in which there is already an ungraded registration.

REGISTRATION ERROR: Student has exceeded maximum attempts on mastery course.

Student: 10010800 has successfully registered for course: G-100006.

A grade of: 89.4 has been successfully applied to registration: 105.
```

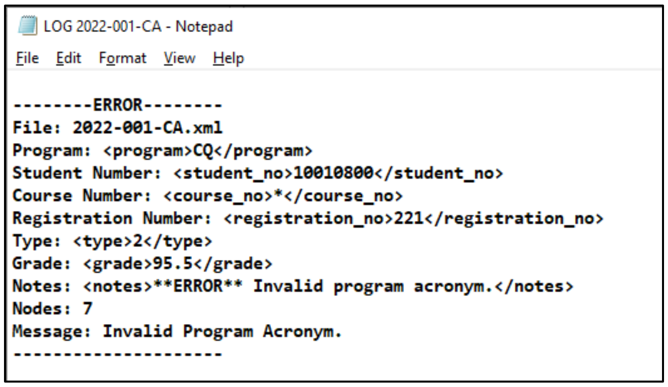
Figure 3

ProcessErrors

- This method will process all detail errors found within the current file being processed.
- This method will be called after each round of detail record validation in ProcessDetails (above).
- beforeQuery: represents the records that existed before the round of validation.
- afterQuery: represents the records that remained following the round of validation.
- message: represents the error message that is to be written to the log file based on the record failing the round of validation.

- Compare the records from the beforeQuery with those from the afterQuery. Any records that do not exist in both failed that round of validation. **Hint**: use the Except() method.
- Process each of the records that failed validation by appending relevant information to the logData instance variable.
    - When writing to logData, ensure the error stands out and data is readable. See the figure below which includes the following:
        - InputFileName (class property)
        - Program
        - Student Number
        - Course Number
        - RegistrationNumber

        - Type
        - Grade
        - Notes
        - Number of Nodes
        - The string message passed to this method
    - **<u>Hint</u>**: Do not use the Value property as this will cause an exception if the error is a result of a missing Element. Using the Element itself will not cause an exception – but will display a value if one exists.



```
--------ERROR--------
File: 2022-001-CA.xml
Program: <program>CQ</program>
Student Number: <student_no>10010800</student_no>
Course Number: <course_no>*</course_no>
Registration Number: <registration_no>221</registration_no>
Type: <type>2</type>
Grade: <grade>95.5</grade>
Notes: <notes>**ERROR** Invalid program acronym.</notes>
Nodes: 7
Message: Invalid Program Acronym.
---------------------
```

Figure 4

WriteLogData

- This method will be called upon completion of a file being processed.
- Note: Processing of a single transmission file may be completed under two circumstances:

    1. When a transmission is processed (whether successfully or unsuccessfully).
    2. When the attributes of the transmission file indicate that the file is not to be processed (that is, when the file fails the ProcessHeader validation).

- Instantiate a StreamWriter associated with the value of the logFileName instance variable.
- Write the accumulated logging data (logData) to the log file.
- Close the StreamWriter.
- Capture in a local variable, the contents of the logging instance variable (logData) to return from this method (NOTE: logData will be cleared in the next step so that it may be reused for subsequent input files – thereby making it necessary to capture the contents into a local variable).
- Set logData to an empty String so that it may be used to capture data for the next transmission file.

- Set logFileName to an empty String so that it may be used to store the next log file name.
- Return the local variable containing the captured logging data to the calling routine.

Windows Form: BatchUpdate

Design

- Use the DataSource to modify the design of the given BatchUpdate form as shown below:
  - Ensure bindings are appropriately set for the ComboBox such that the Description displays but the ProgramAcronym is stored internally.
  - Ensure that the generated BindingNavigator is removed from the form.
  - Ensure the ComboBox is not editable.

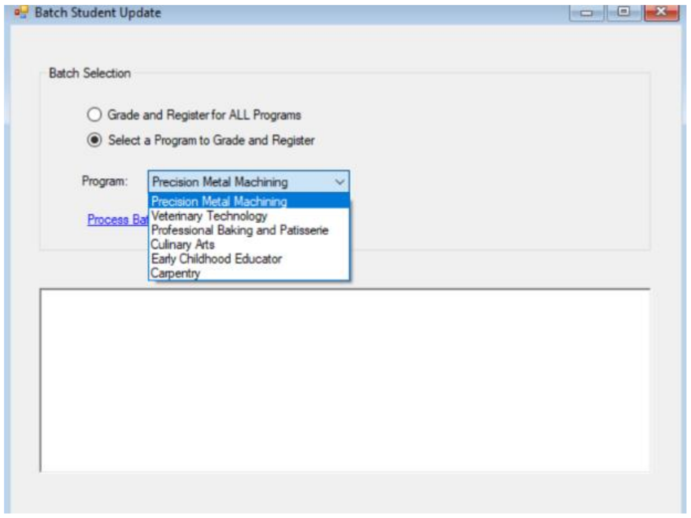

Batch Student Update
Batch Selection
Grade and Register for ALL Programs
Select a Program to Grade and Register
Program:
Precision Metal Machining
Precision Metal Machining
Veterinary Technology
Professional Baking and Patisserie
Culinary Arts
Early Childhood Educator
Carpentry
Process Bat

Figure 5

Script

- Define a private instance of the BIT_College Context, and Batch classes.

BatchUpdate_Load

- Populate the BindingSource object associated with the AcademicProgram ComboBox with a *LINQ-to-SQL Server* query retrieving all records from the AcademicPrograms table.

radAll CheckedChanged Event

- Ensure that the AcademicProgram ComboBox is enabled only when the “Select a Transmission” RadioButton is checked.

Process LinkLabel: linkClicked Event

- Evaluate the RadioButton selection:
    - If a **single transmission** selection has been made:
        - Call the ProcessTransmission method of the Batch class passing appropriate arguments.
        - Call the WriteLogData method of the Batch class to write all logging information associated with this transmission file.
            - Capture the return value and <u>append</u> this value to the richText control's Text property.
    - If **all transmissions** have been selected
        - Iterate through each item in the ComboBox collection. For each iteration:
            - Call the ProcessTransmission method of the Batch class passing appropriate arguments.
            - Call the WriteLogData method of the Batch class to write all logging information associated with this transmission file.
                - Capture the return value and <u>append</u> this value in the richText control's Text property.

Test

- Test your solution to ensure it meets all of the requirements as defined above.
- **<u>Note</u>**: Official Assignment Test Data will be distributed to students a few days prior to evaluation week.

Programming 3 Assignment 7

Backup

- Backup your Solution.
- Create a backup of your database and its corresponding log file.

Prior to Evaluation

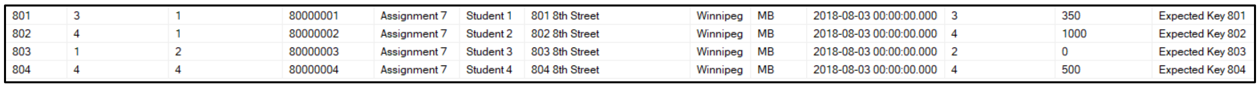
- Specific (sanitized) test data will be used for the evaluation of this assignment.
- In the Assignment 07 Files/SQL Server Script folder you will find a SQL Server script. Use this script in the instructions that follow:

1. Ensure you have created a backup of your database.

2. Modify the script called ***A7TestData.sql*** found in the Assignment 7 Files\SQL Server Scripts folder by replacing the [DATABASENAME] placeholder with the name of your BITCollege_FLContext database.

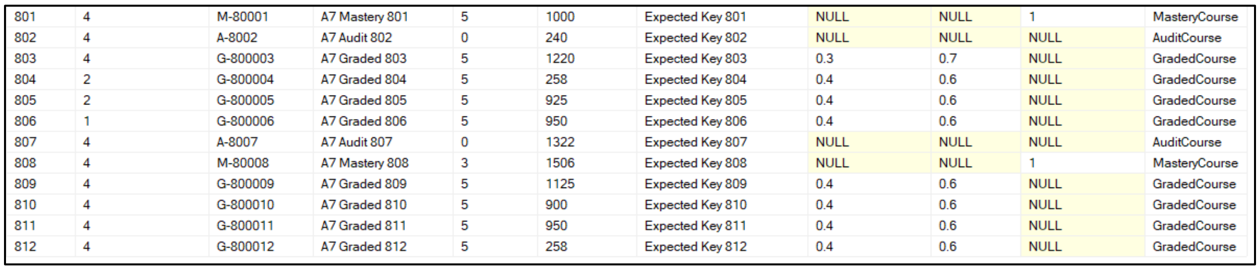
3. Run the script.

Four records should be inserted into the Students table with the primary key value of 801.



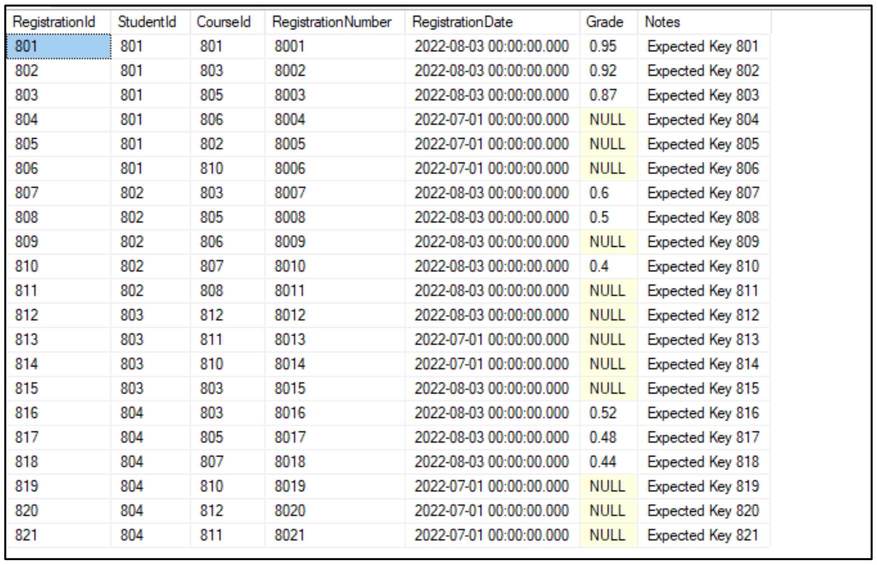
801	3	1	80000001	Assignment 7	Student 1	801 8th Street	Winnipeg	MB	2018-08-03 00:00:00.000	3	350	Expected Key 801
802	4	1	80000002	Assignment 7	Student 2	802 8th Street	Winnipeg	MB	2018-08-03 00:00:00.000	4	1000	Expected Key 802
803	1	2	80000003	Assignment 7	Student 3	803 8th Street	Winnipeg	MB	2018-08-03 00:00:00.000	2	0	Expected Key 803
804	4	4	80000004	Assignment 7	Student 4	804 8th Street	Winnipeg	MB	2018-08-03 00:00:00.000	4	500	Expected Key 804

Twelve records should be inserted into the Courses table starting with the primary key value of 801.



801	4	M-80001	A7 Mastery 801	5	1000	Expected Key 801	NULL	NULL	1	MasteryCourse
802	4	A-8002	A7 Audit 802	0	240	Expected Key 802	NULL	NULL	NULL	AuditCourse
803	4	G-800003	A7 Graded 803	5	1220	Expected Key 803	0.3	0.7	NULL	GradedCourse
804	2	G-800004	A7 Graded 804	5	258	Expected Key 804	0.4	0.6	NULL	GradedCourse
805	2	G-800005	A7 Graded 805	5	925	Expected Key 805	0.4	0.6	NULL	GradedCourse
806	1	G-800006	A7 Graded 806	5	950	Expected Key 806	0.4	0.6	NULL	GradedCourse
807	4	A-8007	A7 Audit 807	0	1322	Expected Key 807	NULL	NULL	NULL	AuditCourse
808	4	M-80008	A7 Mastery 808	3	1506	Expected Key 808	NULL	NULL	1	MasteryCourse
809	4	G-800009	A7 Graded 809	5	1125	Expected Key 809	0.4	0.6	NULL	GradedCourse
810	4	G-800010	A7 Graded 810	5	900	Expected Key 810	0.4	0.6	NULL	GradedCourse
811	4	G-800011	A7 Graded 811	5	950	Expected Key 811	0.4	0.6	NULL	GradedCourse
812	4	G-800012	A7 Graded 812	5	258	Expected Key 812	0.4	0.6	NULL	GradedCourse

Twenty-one records should be inserted into the Registrations table starting with the primary key value of 801.



RegistrationId	StudentId	CourseId	RegistrationNumber	RegistrationDate	Grade	Notes
801	801	801	8001	2022-08-03 00:00:00.000	0.95	Expected Key 801
802	801	803	8002	2022-08-03 00:00:00.000	0.92	Expected Key 802
803	801	805	8003	2022-08-03 00:00:00.000	0.87	Expected Key 803
804	801	806	8004	2022-07-01 00:00:00.000	NULL	Expected Key 804
805	801	802	8005	2022-07-01 00:00:00.000	NULL	Expected Key 805
806	801	810	8006	2022-07-01 00:00:00.000	NULL	Expected Key 806
807	802	803	8007	2022-08-03 00:00:00.000	0.6	Expected Key 807
808	802	805	8008	2022-08-03 00:00:00.000	0.5	Expected Key 808
809	802	806	8009	2022-08-03 00:00:00.000	NULL	Expected Key 809
810	802	807	8010	2022-08-03 00:00:00.000	0.4	Expected Key 810
811	802	808	8011	2022-08-03 00:00:00.000	NULL	Expected Key 811
812	803	812	8012	2022-08-03 00:00:00.000	NULL	Expected Key 812
813	803	811	8013	2022-07-01 00:00:00.000	NULL	Expected Key 813
814	803	810	8014	2022-07-01 00:00:00.000	NULL	Expected Key 814
815	803	803	8015	2022-08-03 00:00:00.000	NULL	Expected Key 815
816	804	803	8016	2022-08-03 00:00:00.000	0.52	Expected Key 816
817	804	805	8017	2022-08-03 00:00:00.000	0.48	Expected Key 817
818	804	807	8018	2022-08-03 00:00:00.000	0.44	Expected Key 818
819	804	810	8019	2022-07-01 00:00:00.000	NULL	Expected Key 819
820	804	812	8020	2022-07-01 00:00:00.000	NULL	Expected Key 820
821	804	811	8021	2022-07-01 00:00:00.000	NULL	Expected Key 821

4. **Note**: If any testing based on the records inserted above takes place prior to the evaluation:

   a. Delete the newly inserted Student record from the database.
   ```
   DELETE FROM [dbo].[Students]
         WHERE StudentId > 800
   ```

      i. This will cause a cascade delete of related Registrations records.

   b. Delete the newly inserted Course record from the database.
   ```
   DELETE FROM [dbo].[Courses]
         WHERE CourseId > 800
   ```

   c. Re-run the A7TestData.sql script.

5. Once the xml files have been received from your instructor, copy the files to the bin\debug folder of your BITCollegeWindows Project.
   - **Note:** The file name of each xml file will need to be changed to today's date. The header date within the xml file will also need to be changed to today's date, except for the file for the program ECE. It is intended to have an incorrect header date.

Evaluation: In Person/Hybrid Delivery

- This assignment must be evaluated on or before the due date/time specified in Learn.
    - The assignment is due at the beginning of class on the due date specified. Your instructor will evaluate the assignment on the students' machine in a random order. Remote evaluations will be based on a first come, first served basis.