ITS 462 Term Project

Database Design

and

Dataset Consolidation

Yongbing Tang

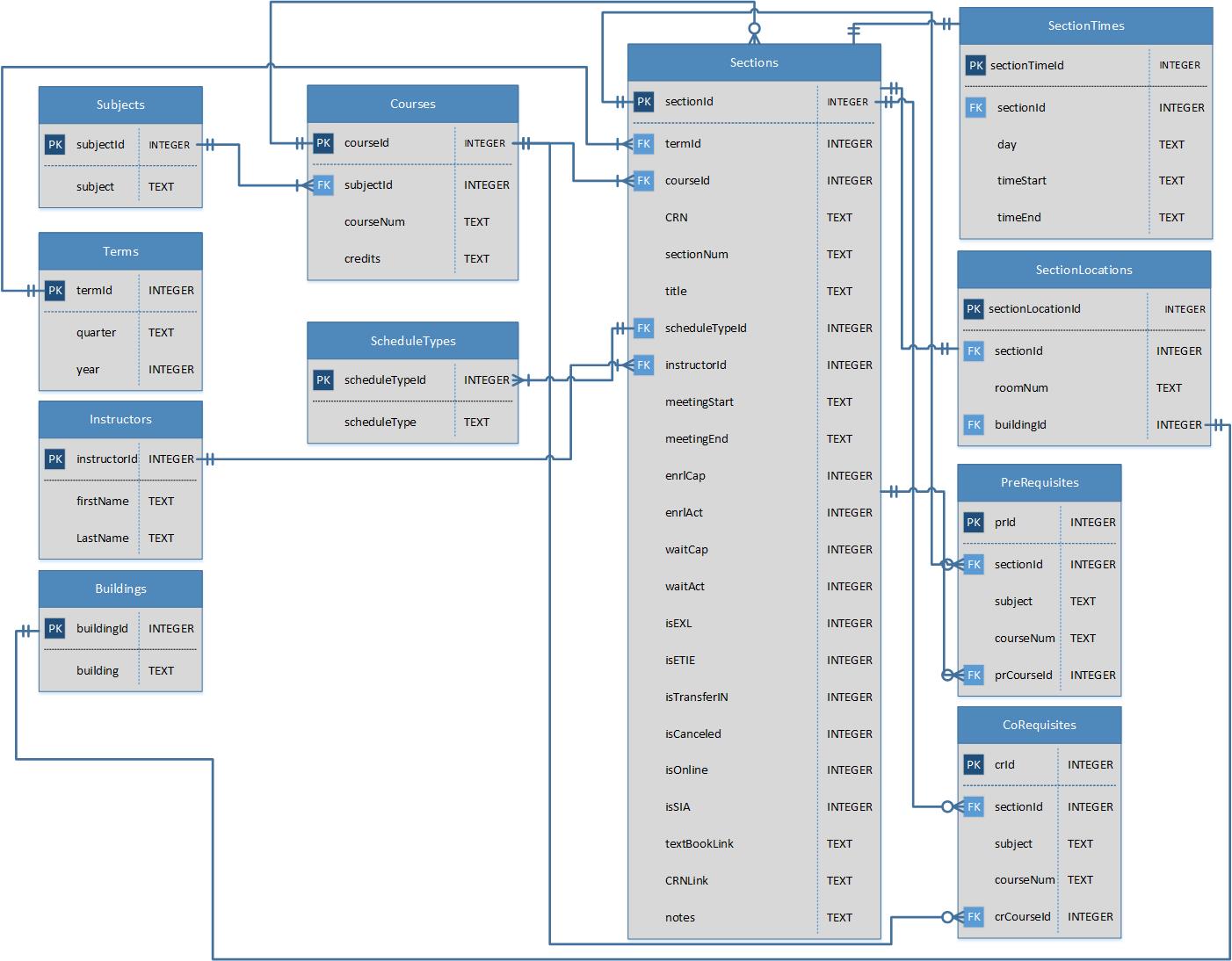
# Monday, May 12, 2015

# Introduction

This project involves redesign of the PUC online class schedule application. The deliverable will be a desktop application for PUC class schedules, with easy to use and more functional features. In this report, the redesigned database for storing course information will be introduced in the following sections. And the consolidation process will be explained in Consolidation section. And the source code of this project has been uploaded to [GitHub](https://github.com/EdwardTang/PUC-ScheduleMaker).

# Database Logical Design

Based on the study of data provided by instructor, the unified schema (ERD) for class schedule is created, shown as the following:



# Data Dictionary

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | | Not Null | Unique |
| Subject | A branch knowledge taught in the school. The table name of the entity is “Subjects”. | subjectId | | Primary key of a subject record in the table “Subjects”. | INTEGER | True | True | | True |
| subject | | Subject name in acronym, e.g. CGT, ITS. | TEXT | False | True | | True |
|  |  |  |  | |  |  |  | |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Terms | The time during which school holds classes. The table name of the entity is “Terms”. | termId | | Primary key of a term record in the table “Terms”. | INTEGER | True | True | True |
| quarter | | Quarter name in each term, e.g. Fall, Spring, Summer. | TEXT | False | True | False |
| year | | Year of the term, e.g. 2014, 2015. | INTEGER | False | True | False |
|  |  |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Instructors | The staffs who teach courses in the school. The table name of the entity is “Instructors”. | instructorId | | Primary key of an instructor record in the table “Instructors”. | INTEGER | True | True | True |
| firstName | | First name of this instructor, e.g. The first name of data “Jiang, Keyuan” is “Keyuan”. | TEXT | False | True | False |
| lastName | | Last name of this instructor, e.g. The last name of data “Jiang, Keyuan” is “Jiang”. | TEXT | False | True | False |
|  |  |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Buildings | The building in which instructors teach classes . The table name of the entity is “Buildings”. | | buildingId | | Primary key of a building record in the table “Buildings”. | INTEGER | True | True | True |
| building | | Name of this building, e.g. The building name of data “Gyte (Millard E) Science Bldg - 002” is “Gyte (Millard E) Science Bldg”. | TEXT | False | True | True |
|  |  | |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Courses | A series of lectures or lessons in a particular subject. The table name of the entity is “Courses”. | courseId | | Primary key of a course record in the table “Courses” | INTEGER | True | True | True |
| subjectId | | Foreign Key of a course record indicates which subject offers this course, associated with the subject record in the table “Subjects”. | TEXT | False | True | False |
| courseNum | | Code of the course. The value of data cells associated with column “#” in project data. e.g. the course number of “AAE 69001” is “69001”. | TEXT | False | True | False |
|  |  | credits | | Credit hours of the course. The value of data cell associated with column  “Cr Hrs” in the raw data, e.g. “3.0”. | INTEGER | False | True | False |
|  |  |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Sections | A course section is a specific instance of a course offered during a specific term | sectionId | | Primary key of a section record in the table “Sections”. | INTEGER | True | True | True |
| termId | | Foreign Key of a section record indicates which term offered this course section, associated with the specific term record in the table “Terms”. | INTEGER | False | True | False |
| courseId | | Foreign Key of a section record indicates which course offered this course section, associated with the specific course record in the table “Courses”. | INTEGER | False | True | False |
|  |  | CRN | | In the raw data, CRN is the value of data cell associated with column “CRN”, e.g. “66103”, which distinguish each course section in each academic year. CRNs are recycling every academic year, which means a course section can have same CRN across years. | TEXT | False | True | False |
|  |  | sectionNum | | Code of a course section under the associated course. In the raw data, “sectionNum” is the value of data cell associated with column “Section”, e.g. “01”. | INTEGER | False | True | False |
|  |  | title | | Title of a course section. In the raw data, “title” is the value of data cell associated with column “Title”, e.g. “Drawing II”. | TEXT | False | True | False |
|  |  | scheduleTypeId | | Foreign Key of a section record indicates what type of schedule is offered with the section, associated with the specific record in table “ScheduleTypes”. | INTEGER | False | True | False |
|  |  | instructorId | | Foreign Key of a section record indicates which instructor teaches the section, associated with the specific record in table “Instructors”. | INTEGER | False | True | False |
|  |  | meetingStart | | The date in which a course section starts. In the raw data, “meetingStart” can be found in the data cell associated with column “Section Meeting Dates”, e.g. “JAN 19, 2010” in “JAN 19, 2010 to MAY 15, 2010”. | TEXT | False | True | False |
|  |  | meetingEnd | | The date in which a course section ends. In the raw data, “meetingEnd” can be found in the data cell associated with column “Section Meeting Dates”, e.g. “MAY 15, 2010” in “JAN 19, 2010 to MAY 15, 2010”. | TEXT | False | True | False |
|  |  | enrlCap | | The number of available seats for enrollment in a course section. “enrlCap” can be found in the value of data cells associated with column “Enrollment  Taken/Avail”, e.g. “3” in “21 / 3” | INTEGER | False | True | False |
|  |  | enrlAct | | The number of taken seats for enrollment in a course section. “enrlAct” can be found in the values of data cells associated with column “Enrollment  Taken/Avail”, e.g. “21” in “21 / 3” | INTEGER | False | True | False |
|  |  | waitCap | | The number of available seats for wait list. If the data cell associated with column “Waitlist  Taken/Avail  ” is shown as “N/A”, then “waitCap” is 0. | INTEGER | False | True | False |
|  |  | waitAct | | The number of taken seats for wait list. If the data cell associated with column “Waitlist  Taken/Avail  ” is shown as “N/A”, then “waitAct” is 0. | INTEGER | False | True | False |
|  |  | isEXL | | The flag that indicates if a course section is included in EXL program. If yes, “isEXL” is 1, otherwise 0. To distinguish an EXL course section, the acronym “EXL” is indicated in the values of the data cells which associated with column “#” in raw data, e.g. “14100(EXL)” | INTEGER | False | True | False |
|  |  | isETIE | | The flag that indicates if a course section is included in ETIE program. If yes, “isETIE” is 1, otherwise 0. To distinguish an ETIE course section, the acronym “ETIE” is indicated in the values of the data cells which associated with column “Title” and column “Important comments about the section.” in raw data, e.g. “ETIE English” in the title and “ETIE” in the comment area. | INTEGER | False | True | False |
|  |  | isTransferIN | | The flag that indicates if a course section is included in TransferIN program. If yes, “isTransferIN” is 1, otherwise 0. To distinguish a TransferIN course section, the acronym “TransferIN” is indicated in the values of the data cells which associated with column “#” in raw data, e.g. “15100(TransferIN)” | INTEGER | False | True | False |
|  |  | isCanceled | | The flag that indicates if a course section canceled. If yes, “isCanceled” is 1, otherwise 0. To distinguish a canceled course section, the status is indicated in the values of the data cells which associated with column “Important comments about the section” in raw data, e.g. “\*\*\*CANCELED\*\*\*” in the comment area. | INTEGER | False | True | False |
|  |  | isOnline | | The flag that indicates if a course section is scheduled as distance learning. If yes, “isOnline” is 1, otherwise 0. To distinguish a distance learning course section, the key word “Distance Learning” is indicated in the values of the data cells which associated with column “Type”, column “Times”, column, column “Building - Room” and column “Important comments about the section.” in raw data, e.g. “Distance Learning” in the time and “Distance Learning Courses” in the column “Building - Room”. | INTEGER | False | True | False |
|  |  | isSIA | | The flag that indicates if a course section is supplemental instruction available. If yes, “isSIA” is 1, otherwise 0. To distinguish a SIA course section, the keyword “supplemental instruction available” is indicated in the values of the data cells which associated with column “Important comments about the section.” in raw data, e.g. “Supplemental Instruction Available” in the comment area. | INTEGER | False | True | False |
|  |  | textBookLink | | The hyper link linked to the textbook information of a course section. To found the link, capture the hyper link with the keyword “View Books” in comment area. | TEXT | False | True | False |
|  |  | CRNLink | | The part of URL linked to the details of a course section, manipulated by JavaScript. To found the URL, capture the <a> tag in data cell associated with column “CRN” e.g. “62373”. Within the <a> tag, only capture the URL that passed to JavaScript function “openWindow”, e.g. “<a href="javascript:openWindow('**/pls/proddad/Webctlg.P\_CtlgProcInput?inputsubjcode=AD&inputsymbol==&inputcrsenumb=22200&inputcoursetype=2&inputreqind=2&callpage=clistquery**')">62373</a>” | TEXT | False | True | False |
|  |  | notes | | The rest information that uncovered by flags “isETIE”, “isOnline”, “isCanceled” and “isSIA”, “**Pre-Requisites**”, “**Co-Requisites**” and “View Books” in the comment area. | TEXT | False | True | False |
|  |  |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Schedule Types | Specific name of the schedule for a course section during the weekdays. | instructorId | | Primary key of scheduleType record in the table “scheduleTypes”. | INTEGER | True | True | True |
| scheduleType | | The type name of a course section’ schedule. “scheduleType” is the value of the data cell associated with column “Type” in raw data, e.g. “Lecture” and “Distance Learning”. | TEXT | False | True | False |
|  |  |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Section Times | The specific time in which the course is taught during scheduled days. The table name of the entity is “SectionTimes”. | sectionTimeId | | Primary key of a sectionTime record in the table “SectionTimes”. | INTEGER | True | True | True |
| sectionId | | Foreign key of a sectionTime record indicates which course section is scheduled at this time, associated with the specific section record in table “Sections” | INTEGER | False | True | False |
| day | | The names of scheduled days, e.g. “M”, “R”, “MW” and “TR” | TEXT | False | True | False |
| timeStart | | The time in which a course section starts during scheduled days. In the raw data, “timeStart” can be found in the data cell associated with column “Times”, e.g. “05:00 PM” in “05:00 PM - 05:50 PM”. | TEXT | False | False | False |
| timeEnd | | The time in which a course section ends during scheduled days. In the raw data, “timeEnd” can be found in the data cell associated with column “Times”, e.g. “05:50 PM” in “05:00 PM - 05:50 PM”. | TEXT | False | False | False |
|  |  |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Section Locations | The specific location at which the course is taught during scheduled days. The table name of the entity is “SectionTimes”. | sectionTimeId | | Primary key of a sectionLocation record in the table “SectionTimes”. | INTEGER | True | True | True |
| sectionId | | Foreign key of a sectionLocation record indicates which course section is scheduled at this location, associated with the specific section record in table “Sections”. | INTEGER | False | True | False |
| roomNum | | Code of room where the course is taught. “roonNum” can be found in the value of data cell associated with column “Building - Room” e.g. “131” in “Powers (Donald S) Building - 131” | TEXT | False | False | False |
| buildingId | | Foreign key of a sectionLocation record indicates which building is scheduled for the course section, associated with the specific building record in table “Buildings”. | INTEGER | False | False | False |
|  |  |  |  | |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Pre-Requisites | A course that a student must pass before enrolling in the more advanced course. The table name of the entity is “PreRequisites”. These data can be found in comment area with keyword “Pre-Requisites:”. | prId | | Primary key of an Prerequisite record in the table “PreRequisites”. | INTEGER | True | True | True |
| sectionId | | Foreign key of a prerequisite record indicates which course section is advanced, associated with the specific course record in table “Courses” | INTEGER | False | True | False |
| subject | | The subject name of this prerequisite, e.g. the subject name of data “Pre-Requisites: MA 15300” is “MA”. | TEXT | False | True | False |
|  |  | courseNum | | The course code of this prerequisite, e.g. the course code of data “Pre-Requisites: MA 15300” is “15300”. | TEXT | False | True | False |
|  |  | prCourseId | | Foreign key of a prerequisite record indicates which course is the pre-requisite, associated with the specific course record in table "Courses". | INTEGER | False | True | False |
|  |  |  |  | |  |  |  |  |

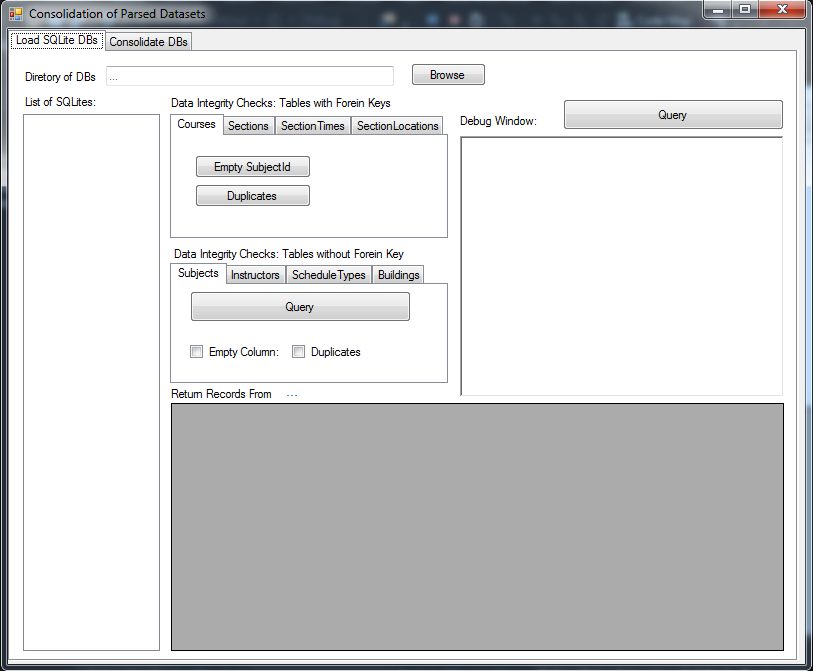
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Entity Description | Column Name | | Column Description | Data Type | Primary Key | Not Null | Unique |
| Co-Requisites | A course that a student must enroll in at the same time as, or in some cases prior to, enrolling in the desired course. The table name of the entity is “CoRequisites”. These data can be found in comment area with keyword “Co-Requisites:”. | prId | | Primary key of a co-requisite record in the table “CoRequisites”. | INTEGER | True | True | True |
| sectionId | | Foreign key of a co-requisite record indicates which course section is required to enroll at same time, associated with the specific course record in table “Courses” | INTEGER | False | True | False |
| subject | | The subject name of this co-requisite, e.g. the subject name of data “Co-Requisites: CHM 26200” is “CHM”. | TEXT | False | True | False |
|  |  | courseNum | | The course code of this co-requisite, e.g. the course code of data “Co-Requisites: CHM 26200” is “26200”. | TEXT | False | True | False |
|  |  | crCourseId | | Foreign key of a prerequisite record indicates which course is the co-requisite, associated with the specific course record in table "Courses". | INTEGER | False | True | False |
|  |  |  |  | |  |  |  |  |

**Important notes for "prCourseId" and "crCourseId":** To find these foreign keys, you have to search "Courses" table with the subject name and the course number stored in the corresponding prerequisite record or co-requisite record. However, you won't find all of matches with single one data set. Because, most prerequisites and some co-requisites are very likely to be taken in the previous semesters. Therefore, I would suggest you not to deal with them until you have consolidated all course schedules in one data store.

## Dataset Consolidation

### Dataset Diagnosis

To start consolidating the datasets collected from the data processors, the first thing we need to do is checking the integrity of each dataset. In order to reduce the workload, I developed a diagnosis program that can check each dataset with specific checkpoints pre-defined. The user interface of this diagnosis program is shown as below:



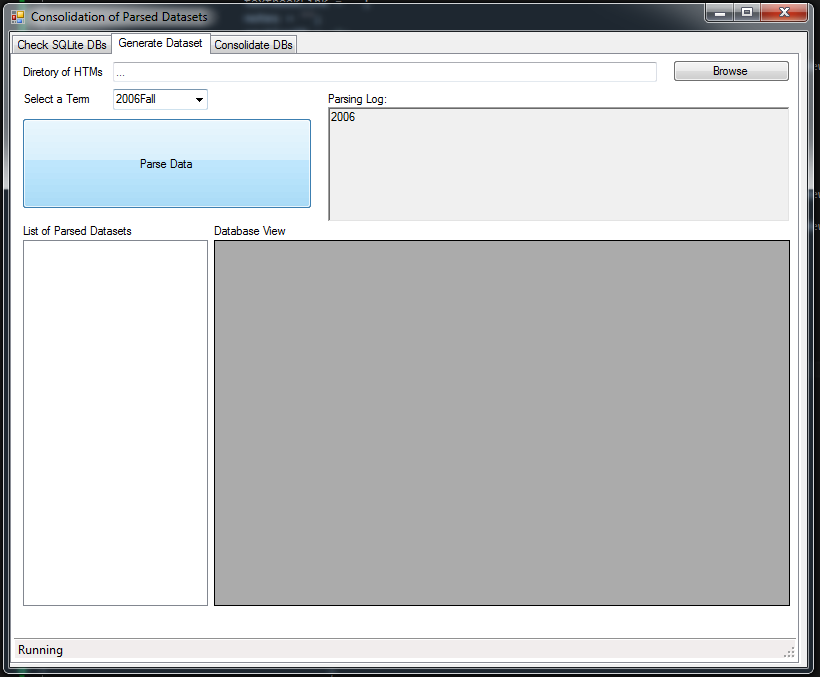
With this diagnosis program, I can diagnosis each dataset with following checkpoints:

1. For the tables with foreign keys:
   1. Empty foreign key columns,
   2. Duplicate records check,
   3. Missed flags specifically in “Sections” table,
2. For the tables without foreign keys:
   1. Duplicates

And we can also customized SQL query in Debug Window.

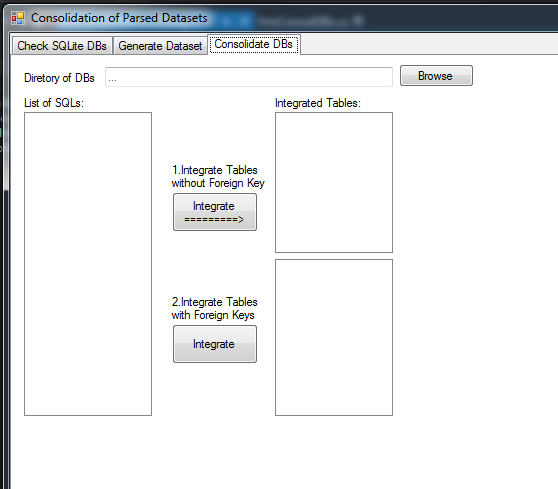
### Generate Dataset

If one of the collected dataset is poor with integrity, the dataset integration cannot be proceeded. To solve this problem, I developed my own data processing program to generate the sufficient dataset.



### Data Integration

The most important part of data consolidation is integrating the datasets. To integrate each dataset together, I first integrated each dataset’s tables without foreign key. Then integrate each table’s tables with foreign keys.



**Comments**

I really enjoyed this project since it is challenging as well as it meets practical development. At the beginning of the project, we did not commutate very closely, which lead to confusion on the understanding of data model. With Prof. Jiang’s helpful hints, we finally made progress on data processing.

## Source Code

using System;

using System.Diagnostics;

using System.IO;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net.NetworkInformation;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Controls;

using System.Data.SQLite;

using HtmlAgilityPack;

namespace ConsolidateDBs

{

public partial class frmConsolDBs : Form

{

private string sSelectedFolder;

private string sSelectedDbName;

private int checkPointNum;

private string sSelectedSqlFolder;

private string sSelectedSqlName;

private string quarter;

private int year;

//string connString = "Data Source=..\\..\\..\\..\\..\\Database\\PUC\_course\_schedule.sqlite;Version=3;";

string connString = "Data source=" + Application.StartupPath + @"\\PUC\_course\_schedule.sqlite";

string roomNumber, subject, instructor, firstName, lastName, schedule, building, CRN, CRNLink, title, startDate, endDate, number, isTransfer, isEXL, isSIA, isETIE, section, crHrs, enrTaken, enrAvail, startTime, endTime, isOnline, isCancelled, waitTaken, waitAvail, textbookLink, notes, days;

string building\_list = "";

string subject\_list = "";

string instructor\_list = "";

string schedule\_list = "";

string course\_list = "";

Int64 courseID;

Int64 sectionID;

List<string> PreReqList = new List<string>();

List<string> CoReqList = new List<string>();

//private string connStr;

public frmConsolDBs()

{

InitializeComponent();

//listBoxDBsMainTab1.Enabled = false;

//tabControlTableCheckpoints.Enabled = false;

dataGridView1.Enabled = false;

tb\_output.AppendText(DateTime.Now.ToShortTimeString() + System.Environment.NewLine);

//checkedListBoxCourses.SelectionMode = System.Windows.Forms.SelectionMode.One;

}

// Process all files in the directory passed in, recurse on any directories

// that are found, and process the files they contain.

public string[] ProcessDirectory(string targetDirectory)

{

// Process the list of files found in the directory.

string[] fileEntries = Directory.GetFiles(targetDirectory);

return fileEntries;

}

private string GetConnectionStr()

{

return (@"Data Source=" + sSelectedFolder + Path.DirectorySeparatorChar + sSelectedDbName + ".sqlite;Version=3;");

}

private void UpdateDataGrid(string con, string sql)

{

SQLiteConnection dbConnection = null;

SQLiteDataAdapter dataAdapter = null;

//Display the SQL it run in rich text box

rtxtBoxSQL.Text = sql;

try

{

// DB code

dbConnection = new SQLiteConnection(con);

dbConnection.Open();

dataAdapter = new SQLiteDataAdapter(sql, con);

DataSet dataSet = new DataSet();

dataAdapter.Fill(dataSet);

// non-DB code

dataGridView1.DataSource = dataSet.Tables[0].DefaultView;

}

catch (SQLiteException sqle)

{

// Handle DB exception

MessageBox.Show("Message: " + sqle.Message + "\n");

}

catch (IndexOutOfRangeException ie)

{

// If you think there might be a problem with index range in the loop, for example

}

catch (Exception ex)

{

// If you want to catch any exception that the previous catches don't catch (that is, if you want to handle other exceptions, rather than let them bubble up to the method caller)

MessageBox.Show("Message: " + ex.Message + "\n");

}

finally

{

// I recommend doing some null-checking here, otherwise you risk a NullReferenceException. There's nothing quite like throwing an exception from within a finally block for fun debugging.

if (dbConnection != null)

dbConnection.Dispose();

if (dataAdapter != null)

dataAdapter.Dispose();

}

}

private void btnBrowse\_Click(object sender, EventArgs e)

{

FolderBrowserDialog fbd = new FolderBrowserDialog();

//fbd.Description = "Custom Description"; //not mandatory

if (fbd.ShowDialog() == DialogResult.OK)

sSelectedFolder = fbd.SelectedPath;

else

sSelectedFolder = string.Empty;

txtDirectory1.Text = sSelectedFolder;

listBoxDBsMainTab1.Enabled = true;

listBoxDBsMainTab1.Items.Clear();

if (Directory.Exists(txtDirectory1.Text))

{

string[] fileEntries = ProcessDirectory(txtDirectory1.Text);

//listBoxDBsTab1.BeginUpdate();

// Loop through and add 50 items to the ListBox.

foreach (string fileName in fileEntries)

{

string extension;

extension = Path.GetExtension(fileName);

if (extension == ".sqlite")

{

string dbName;

dbName = Path.GetFileNameWithoutExtension(fileName);

listBoxDBsMainTab1.Items.Add(dbName);

}

}

// Allow the ListBox to repaint and display the new items.

//listBoxDBsTab1.EndUpdate();

}

else

{

MessageBox.Show(txtDirectory1.Text + " is not a valid directory.");

}

}

private void listBoxDBsMainTab1\_SelectedIndexChanged(object sender, EventArgs e)

{

tabControlTableCheckpoints.Enabled = true;

sSelectedDbName = listBoxDBsMainTab1.Items[listBoxDBsMainTab1.SelectedIndex].ToString();

}

/// <summary>

/// Check if there is duplicates in Courses Table

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnCheckDuplicates\_Click(object sender, EventArgs e)

{

string connStr = GetConnectionStr();

string termPostfix = "\_"+sSelectedDbName;

string courses = " Courses" + termPostfix;

string subjects = " Subjects" + termPostfix;

string cmdStr = "SELECT S.subject, C.courseNum,C.credits, COUNT(\*) AS duplicates"

+ " FROM " + courses + " AS C"

+ " JOIN " + subjects + " AS S ON S.subjectId = C.subjectId"

+ " GROUP BY S.subject, C.courseNum, C.credits"

+ " HAVING duplicates > 1"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

/// <summary>

/// Check if there is empty subjectId in Courses table

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnCheckSubjectId\_Click(object sender, EventArgs e)

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string courses = " Courses" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS EmptySubjectId"

+ " FROM " + courses + " AS C"

+ " WHERE C.subjectId is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

private void btnCheckEmptySectionId1\_Click(object sender, EventArgs e)

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sectionTimes = " SectionTimes" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfEmptySetionId"

+ " FROM " + sectionTimes + " AS STI"

+ " WHERE STI.sectionId is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

private void btnCheckEmptySectionId2\_Click(object sender, EventArgs e)

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sectionLocations = " SectionLocations" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfEmptySetionId"

+ " FROM " + sectionLocations + " AS SL"

+ " WHERE SL.sectionId is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

private void checkBoxSectionEmptyCol\_CheckedChanged(object sender, EventArgs e)

{

if (checkBoxSectionEmptyCol.Checked == true)

{

checkBoxSectionMissedFlg.Checked = false;

cmbBoxSectionEmptyColumn.Enabled = true;

}

else

{

cmbBoxSectionEmptyColumn.SelectedIndex = 0;

cmbBoxSectionEmptyColumn.Enabled = false;

}

}

private void checkBoxSectionMissedFlg\_CheckedChanged(object sender, EventArgs e)

{

if (checkBoxSectionMissedFlg.Checked == true)

{

checkBoxSectionEmptyCol.Checked = false;

cmbBoxSectionMissedFlg.Enabled = true;

}

else

{

cmbBoxSectionMissedFlg.SelectedIndex = 0;

cmbBoxSectionMissedFlg.Enabled = false;

}

}

private void cmbBoxSectionEmptyColumn\_SelectedIndexChanged(object sender, EventArgs e)

{

checkPointNum = cmbBoxSectionEmptyColumn.SelectedIndex;

}

private void cmbBoxSectionMissedFlg\_SelectedIndexChanged\_1(object sender, EventArgs e)

{

checkPointNum = cmbBoxSectionMissedFlg.SelectedIndex+6;

}

private void btnSectionQuery\_Click(object sender, EventArgs e)

{

switch (checkPointNum)

{

case Constants.EmptyCRN:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfEmptyCRN"

+ " FROM " + sections + " AS S"

+ " WHERE S.CRN is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.EmptyCRNLink:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfEmptyCRNLink"

+ " FROM " + sections + " AS S"

+ " WHERE S.CRNLink is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.EmptyCourseId:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfEmptyCourseId"

+ " FROM " + sections + " AS S"

+ " WHERE S.courseId is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.EmptyScheduleTypeId:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfEmptyScheduleId"

+ " FROM " + sections + " AS S"

+ " WHERE S.scheduleTypeId is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.EmptyTextBookLink:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfEmptyTxtBookLink"

+ " FROM " + sections + " AS S"

+ " WHERE S.textBookLink is null"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.MissedIsEXL:

{

if (sSelectedDbName.Contains("2006") || sSelectedDbName.Contains("2007"))

{

MessageBox.Show("There is no EXL progmam in this year");

break;

}

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfIsEXL"

+ " FROM " + sections + " AS S"

+ " WHERE S.isEXL = 1"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.MissedIsTransferIN:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfIsTransferIN"

+ " FROM " + sections + " AS S"

+ " WHERE S.isTransferIN = 1"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.MissedIsETIE:

{

if (sSelectedDbName.Contains("2006") || sSelectedDbName.Contains("2007") || sSelectedDbName.Contains("2008") || sSelectedDbName.Contains("2009"))

{

MessageBox.Show("There is no ETIE progmam in this year");

break;

}

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfIsETIE"

+ " FROM " + sections + " AS S"

+ " WHERE S.isETIE= 1"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.MissedIsCanceled:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfIsCanceled"

+ " FROM " + sections + " AS S"

+ " WHERE S.isCanceled = 1"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.MissedIsOnline:

{

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfMissedIsOnline"

+ " FROM " + sections + " AS S"

+ " WHERE S.isOnline = 0"

+ " AND S.notes like 'Distance Learning'"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

case Constants.MissedIsSIA:

{

if (!sSelectedDbName.Contains("2013") && !sSelectedDbName.Contains("2014") && !sSelectedDbName.Contains("2015"))

{

MessageBox.Show("There is no Supplemental Instruction Available in this year");

break;

}

string connStr = GetConnectionStr();

string termPostfix = "\_" + sSelectedDbName;

string sections = " Sections" + termPostfix;

string cmdStr = "SELECT COUNT(\*) AS NumOfIsSIA"

+ " FROM " + sections + " AS S"

+ " WHERE S.isSIA= 1"

+ ";";

dataGridView1.Enabled = true;

lbDbName.Text = sSelectedDbName;

UpdateDataGrid(connStr, cmdStr);

}

break;

default:

break;

}

}

private void btnBrowse1\_Click(object sender, EventArgs e)

{

FolderBrowserDialog fbd = new FolderBrowserDialog();

//fbd.Description = "Custom Description"; //not mandatory

if (fbd.ShowDialog() == DialogResult.OK)

sSelectedFolder = fbd.SelectedPath;

else

sSelectedFolder = string.Empty;

txtDirectory1.Text = sSelectedFolder;

clBoxSqls.Items.Clear();

if (Directory.Exists(txtDirectory1.Text))

{

string[] fileEntries = ProcessDirectory(txtDirectory1.Text);

foreach (string fileName in fileEntries)

{

string extension;

extension = Path.GetExtension(fileName);

if (extension == ".sql")

{

string dbName;

dbName = Path.GetFileNameWithoutExtension(fileName);

clBoxSqls.Items.Add(dbName);

}

}

// Allow the ListBox to repaint and display the new items.

//listBoxDBsTab1.EndUpdate();

}

else

{

MessageBox.Show(txtDirectory1.Text + " is not a valid directory.");

}

}

private void btnDebugQuery\_Click(object sender, EventArgs e)

{

}

private void cmbSelectTerm\_SelectedIndexChanged(object sender, EventArgs e)

{

string dataName = cmbSelectTerm.SelectedItem.ToString();

if (dataName != "Terms")

{

quarter = Regex.Replace(dataName, @"\d", "");

string currentYearStr = Regex.Replace(dataName, @"\D", "");

year = Convert.ToInt16(currentYearStr);

tb\_output.Text = year.ToString();

}

else

{

quarter = "N/A";

year = Convert.ToInt16("1990");

tb\_output.Text = "Select a term first";

}

}

private void btnBrowse2\_Click(object sender, EventArgs e)

{

FolderBrowserDialog fbd = new FolderBrowserDialog();

//fbd.Description = "Custom Description"; //not mandatory

if (fbd.ShowDialog() == DialogResult.OK)

sSelectedFolder = fbd.SelectedPath;

else

sSelectedFolder = string.Empty;

txtDirectory2.Text = sSelectedFolder;

if (Directory.Exists(txtDirectory2.Text))

{

string[] fileEntries = ProcessDirectory(txtDirectory2.Text);

cmbSelectTerm.Items.Clear();

cmbSelectTerm.Items.Add("Terms");

//listBoxDBsTab1.BeginUpdate();

// Loop through and add 50 items to the ListBox.

foreach (string fileName in fileEntries)

{

string extension;

extension = Path.GetExtension(fileName);

if (extension == ".htm")

{

string dataName;

dataName = Path.GetFileNameWithoutExtension(fileName);

cmbSelectTerm.Items.Add(dataName);

}

}

// Allow the ListBox to repaint and display the new items.

//listBoxDBsTab1.EndUpdate();

}

else

{

MessageBox.Show(txtDirectory2.Text + " is not a valid directory.");

}

}

private void PrepareNumberExtra(string numberExtra)

{

number = numberExtra.Replace(System.Environment.NewLine, "");

string[] words;

words = number.Split(new string[] { "(" }, StringSplitOptions.None);

number = words[0].Trim().Replace(System.Environment.NewLine, "");

words[1] = words[1].Trim().Replace(")", "");

switch (words[1])

{

case "TransferIN":

isTransfer = "1";

break;

case "EXL":

isEXL = "1";

break;

}

//tb\_output.AppendText("TransferIn: Yes" + System.Environment.NewLine);

}

private void PrepareTimes(string times)

{

if (times.Contains("-"))

{

string[] words;

words = times.Split(new string[] { "-" }, StringSplitOptions.None);

startTime = words[0].Trim();

endTime = words[1].Trim();

}

else

{

startTime = times.Trim();

endTime = "";

}

}

private void PrepareSectionDates(string SectionDates)

{

string[] words;

words = SectionDates.Split(new string[] { "to" }, StringSplitOptions.None);

startDate = words[0].Trim().Replace(System.Environment.NewLine, "");

endDate = words[1].Trim().Replace(System.Environment.NewLine, "");

}

private void PrepareWaitlist(string waitList)

{

string[] part = waitList.Trim().Split(new string[] { "/" }, StringSplitOptions.None);

if (part[0].Equals("N")) { waitTaken = "0"; } else { waitTaken = part[0].ToString(); }

if (part[1].Equals("A")) { waitAvail = "0"; } else { waitAvail = part[1].ToString(); }

}

private void PreparePreReq(string PreReq)

{

PreReq = PreReq.Replace(System.Environment.NewLine, "");

string[] words;

words = PreReq.Split(new string[] { "and", "or" }, StringSplitOptions.None);

foreach (string req in words)

{

PreReqList.Add(req.Trim());

}

}

private void PrepareCoReq(string CoReq)

{

CoReq = CoReq.Replace(System.Environment.NewLine, "");

string[] words;

words = CoReq.Split(new string[] { "and", "or" }, StringSplitOptions.None);

foreach (string req in words)

{

CoReqList.Add(req.Trim());

}

}

private void InsertCourseIntoDatabase(SQLiteConnection conn)

{

/\*Insert Subject\*/

if (!subject\_list.Contains("++" + subject))

{

subject\_list = subject\_list + "++" + subject;

SQLiteCommand cmd\_insert\_instructor = new SQLiteCommand(conn);

SQLiteParameter paramSubject = new SQLiteParameter();

cmd\_insert\_instructor.CommandText = "INSERT INTO Subjects(subject) SELECT ? WHERE NOT EXISTS(SELECT 1 FROM Subjects WHERE subject = ?)";

paramSubject.DbType = DbType.String;

paramSubject.Value = subject;

cmd\_insert\_instructor.Parameters.Add(paramSubject);

cmd\_insert\_instructor.Parameters.Add(paramSubject);

cmd\_insert\_instructor.ExecuteNonQuery();

}

if (!course\_list.Contains("++" + subject + number)) //Checks if the Course Number already exists

{

/\*Insert Course\*/

SQLiteCommand cmd\_insert\_course = new SQLiteCommand(conn);

SQLiteParameter paramSubjectId = new SQLiteParameter();

SQLiteParameter paramcourseNum = new SQLiteParameter();

SQLiteParameter paramCredits = new SQLiteParameter();

cmd\_insert\_course.CommandText = "INSERT INTO Courses (subjectid, courseNum, credits) VALUES ((SELECT subjectid FROM Subjects WHERE subject = ?), ?, ?); select last\_insert\_rowid()";

paramSubjectId.DbType = DbType.String;

paramSubjectId.Value = subject;

paramcourseNum.DbType = DbType.String;

paramcourseNum.Value = number;

paramCredits.DbType = DbType.String;

paramCredits.Value = crHrs;

cmd\_insert\_course.Parameters.Add(paramSubjectId);

cmd\_insert\_course.Parameters.Add(paramcourseNum);

cmd\_insert\_course.Parameters.Add(paramCredits);

courseID = (Int64)cmd\_insert\_course.ExecuteScalar();

course\_list = course\_list + "++" + subject + number;

}

else

{

SQLiteCommand cmd\_insert\_course = new SQLiteCommand(conn);

SQLiteParameter paramCourseId = new SQLiteParameter();

cmd\_insert\_course.CommandText = "SELECT courseId FROM Courses WHERE courseNum = ?";

paramCourseId.DbType = DbType.String;

paramCourseId.Value = number;

cmd\_insert\_course.Parameters.Add(paramCourseId);

courseID = (Int64)cmd\_insert\_course.ExecuteScalar();

}

}

private void InsertSectionIntoDatabase(SQLiteConnection conn)

{

/\*Insert Instructor\*/

SQLiteCommand cmd\_insert\_instructor = new SQLiteCommand(conn);

cmd\_insert\_instructor.CommandText = "INSERT INTO Instructors(firstName,lastName) SELECT ?, ? WHERE NOT EXISTS(SELECT 1 FROM Instructors WHERE firstName = ? AND lastName = ?)";

SQLiteParameter paramfirstName = new SQLiteParameter();

SQLiteParameter paramlastName = new SQLiteParameter();

if (instructor.Equals("To Be Announced"))

{

paramfirstName.DbType = DbType.String;

firstName = "To Be Announced";

paramfirstName.Value = firstName;

paramlastName.DbType = DbType.String;

lastName = "To Be Announced";

paramlastName.Value = lastName;

}

else

{

string[] words;

words = instructor.Split(new string[] { "," }, StringSplitOptions.None);

paramfirstName.DbType = DbType.String;

firstName = words[1].Trim();

paramfirstName.Value = firstName;

paramlastName.DbType = DbType.String;

lastName = words[0].Trim();

paramlastName.Value = lastName;

}

if (!instructor\_list.Contains("++" + firstName + "," + lastName))

{

instructor\_list = instructor\_list + "++" + firstName + "," + lastName;

cmd\_insert\_instructor.Parameters.Add(paramfirstName);

cmd\_insert\_instructor.Parameters.Add(paramlastName);

cmd\_insert\_instructor.Parameters.Add(paramfirstName);

cmd\_insert\_instructor.Parameters.Add(paramlastName);

cmd\_insert\_instructor.ExecuteNonQuery();

}

/\*Insert Schedule Type\*/

if (!schedule\_list.Contains("++" + subject))

{

subject\_list = subject\_list + "++" + subject;

SQLiteCommand cmd\_insert\_type = new SQLiteCommand(conn);

SQLiteParameter paramType = new SQLiteParameter();

cmd\_insert\_type.CommandText = "INSERT INTO ScheduleTypes(scheduleType) SELECT ? WHERE NOT EXISTS(SELECT 1 FROM ScheduleTypes WHERE scheduleType = ?);";

paramType.DbType = DbType.String;

paramType.Value = schedule;

cmd\_insert\_type.Parameters.Add(paramType);

cmd\_insert\_type.Parameters.Add(paramType);

cmd\_insert\_type.ExecuteNonQuery();

}

/\*Insert Building\*/

if (building != string.Empty)

{

building = building.Replace(System.Environment.NewLine, "");

SQLiteCommand cmd\_insert\_building = new SQLiteCommand(conn);

SQLiteParameter paramBuilding = new SQLiteParameter();

cmd\_insert\_building.CommandText = "INSERT INTO Buildings(building) SELECT ? WHERE NOT EXISTS(SELECT 1 FROM Buildings WHERE building = ?)";

if (building.Contains("-"))

{

string[] words;

words = building.Split(new string[] { "-" }, StringSplitOptions.None);

building = words[0].Trim();

roomNumber = words[1].Trim();

paramBuilding.DbType = DbType.String;

paramBuilding.Value = words[0].Trim();

}

else

{

paramBuilding.DbType = DbType.String;

paramBuilding.Value = building.Trim();

}

if (!building\_list.Contains("++" + building))

{

building\_list = building\_list + "++" + building;

cmd\_insert\_building.Parameters.Add(paramBuilding);

cmd\_insert\_building.Parameters.Add(paramBuilding);

cmd\_insert\_building.ExecuteNonQuery();

}

}

if (sectionID == -1)

{

/\*Insert Section\*/

SQLiteCommand cmd\_insert\_section = new SQLiteCommand(conn);

SQLiteParameter paramTermYear = new SQLiteParameter();

SQLiteParameter paramTermQuarter = new SQLiteParameter();

SQLiteParameter paramCRN = new SQLiteParameter();

SQLiteParameter paramCourseID = new SQLiteParameter();

SQLiteParameter paramNumber = new SQLiteParameter();

SQLiteParameter paramTitle = new SQLiteParameter();

SQLiteParameter paramSchedule = new SQLiteParameter();

SQLiteParameter paramInstructorFn = new SQLiteParameter();

SQLiteParameter paramInstructorLn = new SQLiteParameter();

SQLiteParameter paramMeetingStart = new SQLiteParameter();

SQLiteParameter paramMeetingEnd = new SQLiteParameter();

SQLiteParameter paramEnrlCap = new SQLiteParameter();

SQLiteParameter paramEnrlAct = new SQLiteParameter();

SQLiteParameter paramWaitCap = new SQLiteParameter();

SQLiteParameter paramWaitAct = new SQLiteParameter();

SQLiteParameter paramIsExl = new SQLiteParameter();

SQLiteParameter paramIsEtie = new SQLiteParameter();

SQLiteParameter paramIsTransferIn = new SQLiteParameter();

SQLiteParameter paramIsCancelled = new SQLiteParameter();

SQLiteParameter paramIsOnline = new SQLiteParameter();

SQLiteParameter paramIsSia = new SQLiteParameter();

SQLiteParameter paramTextBookLink = new SQLiteParameter();

SQLiteParameter paramCRNLink = new SQLiteParameter();

SQLiteParameter paramNotes = new SQLiteParameter();

paramTermYear.DbType = DbType.Int16;

paramTermQuarter.DbType = DbType.String;

paramCRN.DbType = DbType.String;

paramCourseID.DbType = DbType.Int16;

paramNumber.DbType = DbType.String;

paramTitle.DbType = DbType.String;

paramSchedule.DbType = DbType.String;

paramInstructorFn.DbType = DbType.String;

paramInstructorLn.DbType = DbType.String;

paramMeetingStart.DbType = DbType.String;

paramMeetingEnd.DbType = DbType.String;

paramEnrlCap.DbType = DbType.Int16;

paramEnrlAct.DbType = DbType.Int16;

paramWaitCap.DbType = DbType.Int16;

paramWaitAct.DbType = DbType.Int16;

paramIsExl.DbType = DbType.Int16;

paramIsEtie.DbType = DbType.Int16;

paramIsTransferIn.DbType = DbType.Int16;

paramIsCancelled.DbType = DbType.Int16;

paramIsOnline.DbType = DbType.Int16;

paramIsSia.DbType = DbType.Int16;

paramTextBookLink.DbType = DbType.String;

paramCRNLink.DbType = DbType.String;

paramNotes.DbType = DbType.String;

cmd\_insert\_section.CommandText = "INSERT INTO Sections (termId, CRN, courseId, sectionNum, title, scheduleTypeId, instructorId, meetingStart, meetingEnd, enrlCap, enrlAct, waitCap, waitAct, isEXL, isETIE, isTransferIN, isCanceled, isOnline, isSIA, textBookLink, CRNLink, notes) VALUES ((SELECT termId FROM Terms WHERE quarter = ? AND year = ?), ?, ?,?,?,(SELECT scheduleTypeId FROM ScheduleTypes WHERE scheduleType = ?), (SELECT instructorId FROM Instructors WHERE firstName = ? AND lastName = ?),?,?,?,?,?,?,?,?,?,?,?,?,?,?,?); select last\_insert\_rowid()";

paramTermQuarter.Value = quarter;

paramTermYear.Value = year;

paramCRN.Value = CRN;

paramCourseID.Value = courseID;

paramNumber.Value = section;

paramTitle.Value = title;

paramSchedule.Value = schedule;

paramInstructorFn.Value = firstName;

paramInstructorLn.Value = lastName;

paramMeetingStart.Value = startDate;

paramMeetingEnd.Value = endDate;

paramEnrlCap.Value = int.Parse(enrAvail);

paramEnrlAct.Value = int.Parse(enrTaken);

paramWaitCap.Value = int.Parse(waitAvail);

paramWaitAct.Value = int.Parse(waitTaken);

paramIsExl.Value = int.Parse(isEXL);

paramIsEtie.Value = int.Parse(isETIE);

paramIsTransferIn.Value = int.Parse(isTransfer);

paramIsCancelled.Value = int.Parse(isCancelled);

paramIsOnline.Value = int.Parse(isOnline);

paramIsSia.Value = int.Parse(isSIA);

paramTextBookLink.Value = textbookLink;

paramCRNLink.Value = CRNLink;

paramNotes.Value = notes;

cmd\_insert\_section.Parameters.Add(paramTermQuarter);

cmd\_insert\_section.Parameters.Add(paramTermYear);

cmd\_insert\_section.Parameters.Add(paramCRN);

cmd\_insert\_section.Parameters.Add(paramCourseID);

cmd\_insert\_section.Parameters.Add(paramNumber);

cmd\_insert\_section.Parameters.Add(paramTitle);

cmd\_insert\_section.Parameters.Add(paramSchedule);

cmd\_insert\_section.Parameters.Add(paramfirstName);

cmd\_insert\_section.Parameters.Add(paramlastName);

cmd\_insert\_section.Parameters.Add(paramMeetingStart);

cmd\_insert\_section.Parameters.Add(paramMeetingEnd);

cmd\_insert\_section.Parameters.Add(paramEnrlCap);

cmd\_insert\_section.Parameters.Add(paramEnrlAct);

cmd\_insert\_section.Parameters.Add(paramWaitCap);

cmd\_insert\_section.Parameters.Add(paramWaitAct);

cmd\_insert\_section.Parameters.Add(paramIsExl);

cmd\_insert\_section.Parameters.Add(paramIsEtie);

cmd\_insert\_section.Parameters.Add(paramIsTransferIn);

cmd\_insert\_section.Parameters.Add(paramIsCancelled);

cmd\_insert\_section.Parameters.Add(paramIsOnline);

cmd\_insert\_section.Parameters.Add(paramIsSia);

cmd\_insert\_section.Parameters.Add(paramTextBookLink);

cmd\_insert\_section.Parameters.Add(paramCRNLink);

cmd\_insert\_section.Parameters.Add(paramNotes);

sectionID = (Int64)cmd\_insert\_section.ExecuteScalar();

}

/\*Insert Section Time\*/

SQLiteCommand cmd\_insert\_section\_time = new SQLiteCommand(conn);

SQLiteParameter paramSectionID\_time = new SQLiteParameter();

SQLiteParameter paramDay = new SQLiteParameter();

SQLiteParameter paramTimeStart = new SQLiteParameter();

SQLiteParameter paramTimeEnd = new SQLiteParameter();

paramSectionID\_time.DbType = DbType.Int64;

paramDay.DbType = DbType.String;

paramTimeStart.DbType = DbType.String;

paramTimeEnd.DbType = DbType.String;

cmd\_insert\_section\_time.CommandText = "INSERT INTO SectionTimes (sectionId, day, timeStart, timeEnd) VALUES (?, ?, ?, ?);";

paramSectionID\_time.Value = sectionID;

paramDay.Value = days;

paramTimeStart.Value = startTime;

paramTimeEnd.Value = endTime;

cmd\_insert\_section\_time.Parameters.Add(paramSectionID\_time);

cmd\_insert\_section\_time.Parameters.Add(paramDay);

cmd\_insert\_section\_time.Parameters.Add(paramTimeStart);

cmd\_insert\_section\_time.Parameters.Add(paramTimeEnd);

cmd\_insert\_section\_time.ExecuteNonQuery();

/\*Insert Section Locations\*/

SQLiteCommand cmd\_insert\_section\_location = new SQLiteCommand(conn);

SQLiteParameter paramSectionID\_loc = new SQLiteParameter();

SQLiteParameter paramRoomNum = new SQLiteParameter();

SQLiteParameter paramBuilding\_loc = new SQLiteParameter();

paramSectionID\_loc.DbType = DbType.Int64;

paramRoomNum.DbType = DbType.String;

paramBuilding\_loc.DbType = DbType.String;

cmd\_insert\_section\_location.CommandText = "INSERT INTO SectionLocations (sectionId, roomNum, buildingId) VALUES (?, ?, (SELECT buildingId FROM Buildings WHERE building = ?));";

paramSectionID\_loc.Value = sectionID;

paramRoomNum.Value = roomNumber;

paramBuilding\_loc.Value = building;

cmd\_insert\_section\_location.Parameters.Add(paramSectionID\_loc);

cmd\_insert\_section\_location.Parameters.Add(paramRoomNum);

cmd\_insert\_section\_location.Parameters.Add(paramBuilding\_loc);

cmd\_insert\_section\_location.ExecuteNonQuery();

/\*Insert PreRequisites\*/

if (PreReqList.Count > 0)

{

foreach (string PreReqSingle in PreReqList)

{

string[] words;

words = PreReqSingle.Split(new string[] { " " }, StringSplitOptions.None);

SQLiteCommand cmd\_insert\_PreReq = new SQLiteCommand(conn);

SQLiteParameter paramSectionID\_Pre = new SQLiteParameter();

SQLiteParameter paramSubject\_Pre = new SQLiteParameter();

SQLiteParameter paramCourseNum\_Pre = new SQLiteParameter();

paramSectionID\_Pre.DbType = DbType.Int64;

paramSubject\_Pre.DbType = DbType.String;

paramCourseNum\_Pre.DbType = DbType.String;

cmd\_insert\_PreReq.CommandText = "INSERT INTO PreRequisites (sectionId, subject, courseNum, prCourseId) VALUES (?, ?, ?, (SELECT courseId FROM Courses WHERE subjectId = (SELECT subjectId FROM Subjects WHERE subject = ?) AND courseNum = ?));";

paramSectionID\_Pre.Value = sectionID;

paramSubject\_Pre.Value = words[0].Trim();

paramCourseNum\_Pre.Value = words[1].Trim();

cmd\_insert\_PreReq.Parameters.Add(paramSectionID\_Pre);

cmd\_insert\_PreReq.Parameters.Add(paramSubject\_Pre);

cmd\_insert\_PreReq.Parameters.Add(paramCourseNum\_Pre);

cmd\_insert\_PreReq.Parameters.Add(paramSubject\_Pre);

cmd\_insert\_PreReq.Parameters.Add(paramCourseNum\_Pre);

cmd\_insert\_PreReq.ExecuteNonQuery();

}

PreReqList.Clear();

}

/\*Insert CoRequisites\*/

if (CoReqList.Count > 0)

{

foreach (string CoReqSingle in CoReqList)

{

string[] words;

words = CoReqSingle.Split(new string[] { " " }, StringSplitOptions.None);

SQLiteCommand cmd\_insert\_CoReq = new SQLiteCommand(conn);

SQLiteParameter paramSectionID\_Co = new SQLiteParameter();

SQLiteParameter paramSubject\_Co = new SQLiteParameter();

SQLiteParameter paramCourseNum\_Co = new SQLiteParameter();

paramSectionID\_Co.DbType = DbType.Int64;

paramSubject\_Co.DbType = DbType.String;

paramCourseNum\_Co.DbType = DbType.String;

cmd\_insert\_CoReq.CommandText = "INSERT INTO CoRequisites (sectionId, subject, courseNum, crCourseId) VALUES (?, ?, ?, (SELECT courseId FROM Courses WHERE subjectId = (SELECT subjectId FROM Subjects WHERE subject = ?) AND courseNum = ?));";

paramSectionID\_Co.Value = sectionID;

paramSubject\_Co.Value = words[0].Trim();

paramCourseNum\_Co.Value = words[1].Trim();

cmd\_insert\_CoReq.Parameters.Add(paramSectionID\_Co);

cmd\_insert\_CoReq.Parameters.Add(paramSubject\_Co);

cmd\_insert\_CoReq.Parameters.Add(paramCourseNum\_Co);

cmd\_insert\_CoReq.Parameters.Add(paramSubject\_Co);

cmd\_insert\_CoReq.Parameters.Add(paramCourseNum\_Co);

cmd\_insert\_CoReq.ExecuteNonQuery();

}

CoReqList.Clear();

}

}

private void btnParseData\_Click(object sender, EventArgs e)

{

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

toolStripLabel.Text = "Running";

statusStrip.Refresh();

SQLiteConnection conn = new SQLiteConnection();

conn.ConnectionString = connString;

/\*Insert New Term\*/

SQLiteCommand cmd\_insert\_term = new SQLiteCommand(conn);

SQLiteParameter paramQuarter = new SQLiteParameter();

SQLiteParameter paramYear = new SQLiteParameter();

cmd\_insert\_term.CommandText = "INSERT INTO Terms(quarter, year) SELECT ? , ? WHERE NOT EXISTS(SELECT 1 FROM Terms WHERE quarter= ? AND year = ?)";

paramQuarter.DbType = DbType.String;

paramQuarter.Value = quarter;

paramYear.DbType = DbType.Int16;

paramYear.Value = year;

cmd\_insert\_term.Parameters.Add(paramQuarter);

cmd\_insert\_term.Parameters.Add(paramYear);

cmd\_insert\_term.Parameters.Add(paramQuarter);

cmd\_insert\_term.Parameters.Add(paramYear);

conn.Open();

cmd\_insert\_term.ExecuteNonQuery();

/\*Finish Insert\*/

HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();

// doc.Load("../../2009 Fall - On-Line Class Schedule Query Results\_original.htm");

string dataPath = txtDirectory2.Text + Path.DirectorySeparatorChar + cmbSelectTerm.SelectedItem + ".htm";

doc.Load(dataPath);

HtmlNode base\_table = doc.DocumentNode.SelectSingleNode("//table [@class='datadisplaytable']");

foreach (HtmlNode row in base\_table.ChildNodes)

{

tb\_output.Text = row.ToString();

if (row.InnerHtml.Equals(System.Environment.NewLine) || row.OuterHtml.Contains("ddheader")) { }

else if (row.ChildNodes.Count == 0)

{

}

else

{

switch (row.ChildNodes[1].ChildNodes.Count)

{

case 19: //Normal first row will have 19 children. This includes returns and data

subject = "";

CRN = "";

title = "";

startDate = "";

endDate = "";

number = "";

isTransfer = "0";

isEXL = "0";

isETIE = "0";

isSIA = "0";

section = "";

crHrs = "";

enrAvail = "";

enrTaken = "";

sectionID = -1;

CRN = row.ChildNodes[1].ChildNodes[1].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

CRNLink =

row.ChildNodes[1].ChildNodes[1].Attributes["href"].Value.Replace(

"javascript:openWindow('", "").Replace("')", "").Trim();

subject =

row.ChildNodes[1].ChildNodes[3].InnerText.Trim().Replace(System.Environment.NewLine, "");

number = row.ChildNodes[1].ChildNodes[5].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

section =

row.ChildNodes[1].ChildNodes[7].InnerText.Trim().Replace(System.Environment.NewLine, "");

crHrs = row.ChildNodes[1].ChildNodes[9].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

enrTaken = row.ChildNodes[1].ChildNodes[11].ChildNodes[0].InnerText.Replace("/", "").Trim();

enrAvail = row.ChildNodes[1].ChildNodes[11].ChildNodes[1].InnerText.Replace("/", "").Trim();

PrepareWaitlist(row.ChildNodes[1].ChildNodes[13].InnerText);

title = row.ChildNodes[1].ChildNodes[15].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

PrepareSectionDates(row.ChildNodes[1].ChildNodes[17].InnerText);

InsertCourseIntoDatabase(conn);

break;

case 6:

//Non-Normal first rows will have 6 children. The paranthesis in the # column messes up the counting.

subject = "";

CRN = "";

title = "";

startDate = "";

endDate = "";

number = "";

isTransfer = "0";

isEXL = "0";

isETIE = "0";

isSIA = "0";

section = "";

crHrs = "";

enrTaken = "";

enrAvail = "";

sectionID = -1;

roomNumber = "";

CRN = row.ChildNodes[1].ChildNodes[1].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

subject =

row.ChildNodes[1].ChildNodes[3].InnerText.Trim().Replace(System.Environment.NewLine, "");

PrepareNumberExtra(row.ChildNodes[1].ChildNodes[5].ChildNodes[1].InnerText);

//Needed to fix paranthesis issue. Has to go one more child deeper

section =

row.ChildNodes[1].ChildNodes[5].ChildNodes[3].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

crHrs =

row.ChildNodes[1].ChildNodes[5].ChildNodes[5].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

enrTaken =

row.ChildNodes[1].ChildNodes[5].ChildNodes[7].ChildNodes[0].InnerText.Replace("/", "")

.Trim();

enrAvail =

row.ChildNodes[1].ChildNodes[5].ChildNodes[7].ChildNodes[1].InnerText.Replace("/", "")

.Trim();

PrepareWaitlist(row.ChildNodes[1].ChildNodes[5].ChildNodes[9].InnerText);

title =

row.ChildNodes[1].ChildNodes[5].ChildNodes[11].InnerText.Trim()

.Replace(System.Environment.NewLine, "");

PrepareSectionDates(row.ChildNodes[1].ChildNodes[5].ChildNodes[13].InnerText);

InsertCourseIntoDatabase(conn);

break;

default: //For non first row children

schedule = "";

instructor = "";

building = "";

startTime = "";

endTime = "";

isOnline = "0";

isCancelled = "0";

textbookLink = "";

notes = "";

sectionID = -1;

schedule =

row.ChildNodes[1].ChildNodes[0].InnerText.Trim().Replace(System.Environment.NewLine, "");

if (schedule.Equals("Distance Learning"))

{

isOnline = "1";

}

days = row.ChildNodes[3].ChildNodes[0].InnerText.Trim();

PrepareTimes(row.ChildNodes[5].ChildNodes[0].InnerText.Trim());

building =

row.ChildNodes[7].ChildNodes[0].InnerText.Trim().Replace(System.Environment.NewLine, "");

instructor =

row.ChildNodes[9].ChildNodes[0].InnerText.Trim().Replace(System.Environment.NewLine, "");

if (row.ChildNodes.Count >= 12)

{

int skip = 0;

foreach (HtmlNode comment in row.ChildNodes[11].ChildNodes)

//The comment row can have numerous things.

{

if (skip == 0)

{

if (comment.InnerText.Equals("Pre-Requisites:"))

{

PreparePreReq(comment.NextSibling.InnerHtml.ToString().Trim());

skip = 1;

}

else if (comment.InnerText.Equals("Co-Requisites:"))

{

PrepareCoReq(comment.NextSibling.InnerHtml.Trim());

skip = 1;

}

else if (comment.InnerText.Contains("\*\*\*CANCELED\*\*\*"))

{

isCancelled = "1";

skip = 0;

}

else if (comment.InnerText.Contains("View Books"))

{

textbookLink = comment.Attributes["href"].Value;

skip = 0;

}

else

{

if (comment.InnerText.Trim() != string.Empty)

{

notes = comment.InnerText;

skip = 0;

}

}

}

else

{

skip = 0;

continue;

}

}

}

InsertSectionIntoDatabase(conn);

break;

}

}

}

conn.Close();

toolStripLabel.Text = "Finished";

statusStrip.Refresh();

stopwatch.Stop();

tb\_output.AppendText(DateTime.Now.ToShortTimeString() + System.Environment.NewLine);

tb\_output.AppendText(stopwatch.Elapsed.ToString());

}

private void txtDirectory2\_TextChanged(object sender, EventArgs e)

{

}

}

static class Constants

{

public const int EmptyCRN = 1;

public const int EmptyCRNLink = 2;

public const int EmptyCourseId = 3;

public const int EmptyScheduleTypeId = 4;

public const int EmptyTextBookLink = 5;

public const int MissedIsEXL = 7;

public const int MissedIsTransferIN = 8;

public const int MissedIsETIE = 9;

public const int MissedIsCanceled = 10;

public const int MissedIsOnline = 11;

public const int MissedIsSIA = 12;

}

}