Muntz, Cade Dean

05/17/2023 |

OSU College of Business – Syllabi Collection and Publication

BA 372 – Process improvement delivery 1, prof. reitsma

Contents

[Chapter 1 – Introduction 2](#_Toc135221905)

[Collection of course syllabi 2](#_Toc135221906)

[Posting the Syllabi 2](#_Toc135221907)

[Chapter 2 – Process Improvement Impact Assessment 4](#_Toc135221908)

[Measures of Success 4](#_Toc135221909)

[Chapter 3 – Process Models 6](#_Toc135221910)

[Existing Business Process 6](#_Toc135221911)

[Chapter 4 – Database Design 9](#_Toc135221912)

[Description of Tables 9](#_Toc135221913)

[Appendix 12](#_Toc135221914)

# Chapter 1 – Introduction

Oregon State University is a university housing about 33,000 students as well as about 1500 professors. Within the College of Business, these professors must make and publish syllabi for their classes using the current process which is set in place. Currently, this process runs through multiple different programs such as Access and SalesForce, with Qualtrics and Box being the main programs used. Using the current process works and is effective, however, it is extremely inefficient and comes with a handful of problems that could be subsided by putting in a different process. The current process is separated into two main parts, the collection of the syllabi as well as the publishing of the syllabi. These processes are as follows.

## Collection of course syllabi

1. John Womack (JW) uses an access database and runs queries against OSU’s data warehouse to gather student information for upcoming courses.
2. Students’ names and their corresponding courses result from these queries and are uploaded to the COB’s SalesForce database using the data loader program within Salesforce.
3. The Salesforce database contains data regarding instructors and which courses they are teaching. By pairing the student enrollment data from earlier and combining it with the instructor data, they are placed into a dataset containing the instructors, their emails, and the courses they teach with at least one student enrolled. This dataset file is then exported from SalesForce as a CSV file.
4. JW then reformats the CSV file manually in Excel and sorts the data by the instructor in the rows.
5. Next JW will upload the CSV file into an existing Qualtrics survey which is used as the main instrument for the collection of the syllabi.
6. Qualtrics will send an email to all instructors on the email list. This email contains a link to the survey which asks the instructor questions about their office hours, and most importantly asks the instructor to upload their syllabus file (.doc, .docx, pdf.) This ends up becoming an unsorted set of files within the Qualtrics survey.
7. Qualtrics at some point will automatically alert instructors who have not responded to the survey.
8. Then at some point, JW will export all the syllabi from Qualtrics, with an average response rate of 80% to the survey. The files have no naming convention put in place.

## Posting the Syllabi

1. Once all the data has been collected from the Qualtrics survey JW runs a program to automatically strip the index codes from the front end of the file names from the Qualtrics survey.
2. These files are then manually gone through to ensure that all files are renamed to follow a specific naming convention. Sometimes there can be up to 300 files per term to go through manually.
3. JW then uses a program to convert all files into a .pdf file.
4. Lastly, JW manually copies each syllabus to its proper location with a publicly viewable box folder.

As mentioned earlier, this process is an effective process as it does work, but it is missing a lot of efficiencies that could make this process a lot easier and faster for Mr. Womack. Going through the list of steps, it can be noted that a lot of the process involves manual intervention which can take a lot of time and effort, but also can cause human error if things are input wrong or moved into the wrong location. A lot of the downsides come down to how the different programs interact with one another, and some of these issues can come to light. Some of the main issues with this process are as follows:

1. There is no consistent naming convention set into place, requiring that manual intervention and renaming of files take place.
2. There is a lack of coordination between Mr. Womack and the professors, as noted above the Qualtrics survey only averages about an 80% response rate.
3. There is no way to search for syllabi information, which leads to having to manually search for syllabi through the box folder by clicking through the different screens until one finds what they are looking for.
4. If a syllabus is missing, the only way to figure this out is by doing an exhaustive search through the entire box folder while also checking through the other data sheets to make sure everything matches up properly.
5. There is no way to link a syllabus with a course automatically, and this must be done manually by going through each syllabus and matching it with its corresponding course.
6. Manual intervention is required to move the information over to Box, which can lead to human error such as placing a syllabus in the wrong place.

# Chapter 2 – Process Improvement Impact Assessment

Based on the previous section, a lot could be done to make this process less exhaustive and time-consuming for Mr. Womack. The current process, while it works, has a lot of issues with efficiency and potential human error which can likely be changed by changing to a relational database process. By implementing a relational database, a lot of the aforementioned steps within the process which required manual intervention could be automated while also allowing for a more user-friendly environment when it comes to searching for specific syllabi.

## Measures of Success

Measures of Success are an important step in revising an existing process and attempting to make that process better in the long run. These measures can be used as a guide on how a new process can minimize the issues caused by the existing process. By understanding these measures, we can gain a better insight into why a new process can help relieve the issues of the past. These measures are as follows:

* Reduce time.
* Reduce manual intervention.
* Prevent human errors.
* Allow querying for ease of use.
* Acquire a higher response rate from professors.

One of, if not the most important measure of success is reducing the process time. By reducing the process time, we can ensure that the process is, for one, much easier on Mr. Womack who currently must put a lot of time and effort into this task. But also reducing the process time can also reduce the costs spent on labor for such a simple task of collecting course syllabi. By reducing the process time, we can ensure that organizational resources are spent on more important tasks.

The next measure of success would be to reduce the amount of manual intervention required to run through the entire process. Manual intervention can cause a lot of issues such as reducing overall productivity and making employees such as Mr. Womack spend much more time on the basic task of collecting syllabi when instead this time could possibly be better spent bringing more value to the organization. By automating the process further, we can see a reduction in time, and human errors, as well as fewer delays and costs.

Human error is another huge issue with this process that can be used as a great measure of success. The more automated the process is, the less human error we may have to deal with as a result. One example would be when manually inputting the syllabi into the box folder, a human may have the potential of placing a syllabus into the wrong folder. This can cause a lot of problems as with the current process situation, searching for files is not a possibility, meaning that somebody would have to go through the entire box folder manually to try and find the misplaced file. By automating the process, we could have a program that automatically places these files into the correct folders which would get rid of this issue and other issues like this.

By using a relational database rather than the current viewable box folder, querying will be of use as we can search for files and gather more in-depth information and insights from the syllabus information. As mentioned, sometimes a professor may not get their syllabus into the Qualtrics survey. With querying, one could simply use a query to search for a specific professor and ensure that they have a syllabus placed into the database. This will get rid of the issue where an exhaustive search must take place to ensure that all the syllabi have been sent in.

Lastly, acquiring a higher response rate from professors would be a great step forward to ensure that less work is needed to be done. By using a relational database, we can continuously check whether a professor has submitted their syllabus allowing them to check in with them more often. Compared to the current system where finding out if an instructor’s syllabus is missing is a very grueling task.

# Chapter 3 – Process Models

The purpose of this section is to review the current process used for the collection and publication of Syllabi used by Mr. Womack in the College of Business. By looking at the current process, an understanding can be reached of each step as well as to analyze how the process truly works and what makes each step. To clearly understand the process and effectively analyze why each action is important and needed, we can display the process in a 3-column quasi formatted summary. By using this format, we can display each action, the content for each action, as well as the medium and tool used to implement the action. The existing progress is also displayed in Figure 1, where we can see each action and how it flows throughout a swim lane diagram for a more visual approach.

### Existing Business Process

The current business process of collection and publication of syllabi is as follows:

|  |  |  |
| --- | --- | --- |
| **Actor and Action** | **Medium or Tool** | **Required Information** |
| JW starts the process by running queries within an MS Access Database. These queries are run against the OSU data warehouse | MS Access and OSU’s data warehouse | These queries are solely used to get student enrollment information to understand which courses students are registered for |
| Results of the queries are uploaded to SalesForce | SalesForce | Using the data loader program, the results are uploaded. These results are the students and what courses they are registered for |
| The student enrollment data is then paired with instructor information such as the instructor’s name, email, and the courses they will be teaching | SalesForce | By pairing the student and professor information, we can understand which classes the professors have that at least 1 student is registered for. These are the classes that will require a syllabus to be submitted |
| JW manually reformats the CSV files using MS Excel | MS Excel, SalesForce Import | Once the data is exported to an Excel CSV file, the data is then reformatted to be sorted by the instructor (rows) |
| JW uploads this CSV dataset to the Qualtrics survey | Qualtrics | The Qualtrics survey is a previously existing survey, which is then sent out by instructor email |
| Qualtrics sends the emails out to the professors, the email has a link, the professors submit their syllabi through the survey | Qualtrics, Gmail | If an instructor does not respond to the email within a certain amount of time that is set by JW, then they will receive another automatic email (Only about 80% of instructors respond on average) |
| Instructors will open the survey and provide their names as well as the syllabus. They also provide their office hours. | Qualtrics | The syllabi do not have a consistent file type, some are .docx, .doc, or .pdf. The syllabi are also not indexed by course but are instead an unordered set of files. |
| JW will then export all the syllabi from Qualtrics. | Qualtrics | The time JW does this step is set by him. All file names contain the original file name input by the professor preceded by a fixed-length unique Qualtrics index code. |
| JW then runs a program to automatically strip the Qualtrics index codes from the file names | Unspecified Program | The program is unspecified |
| JW manually inspects and renames the files | Manual | With the current process, this can sometimes be upwards of 300 files for JW to manually go through and rename to a specific naming convention. This could lead to accidental human error |
| JW converts the files to .PDF | Windows Program | A windows program is run which automatically converts all .doc and .docx files into a pdf format to have a standardized format |
| JW manually copies each syllabus to Box | Box / Manual | JW must manually place each file into a specific location within Box. This again leaves a lot of room for human error. |

A screenshot of a computer screen

Description automatically generated with low confidence

Fig. 1 – CURRENT SYLLABI COLLECTION / PUBLICATION PROCESS

*The flow chart describes the flow of the current syllabi collection and publication process used in the College of Business at OSU. Note the loop that regards whether an instructor responded to the survey or not.*

## Chapter 4 – Database Design

In this chapter, we go over the proposed database design which will hold all the data for collection and publication of syllabi throughout the College of Business at Oregon State University. The relationship diagram produced by the database is shown in Figure 2, which shows how the database defines the relationships within it.

A picture containing text, screenshot, diagram, font

Description automatically generated

FIGURE 2 – ENTITY RELATIONSHIP DIAGRAM

This figure displays the 9 tables included within the database for Syllabus collection and publication. Note the primary keys in each table and how they are referenced by other tables. Also, note the relationships these references create between the different tables.

### Description of Tables

**Student** is a table holding the informational data that pertains to the students within the College of Business. This table includes columns such as *StudentID* (which would be the students OSU Identification Number) as well as *FName*, and *LName*.

* Primary Key: *StudentID* is an integer representation for the student which is unique and only belongs to 1 student. This will get rid of any confusion if any students have the same first name, last name, or both. This will ensure that no duplicates are held within the database.

**Professor** is a table which holds all the information that pertains to the professors within the College of Business. This table includes columns such as *ProfessorID*, *FName*, *LName*, and *Email*. This table essentially holds the professor’s name as well as contact information.

* Primary key: *ProfessorID* is an integer representation for the student which is unique and only belongs to 1 professor. Identifying professors in this sense will get rid of any confusion if there were to be two professors with the same name, this way we can uniquely identify each professor.

**Department** is an object table that holds all the information relating to the department. This table only holds two columns, the *DepartmentID* as well as the *Name* of said department. This table would be already filled with specific Department names and IDs that can efficiently connect different courses to specific departments within the College of Business.

* Primary Key: *DepartmentID* is the unique id associated with a specific department. This key refers to the specific department within the college and using numbers to uniquely identify each department can cause less confusion.

**Course** is a table that holds all the information pertaining to the different courses within the college. This table boasts columns such as *CourseID*, *CourseCode*, *CourseTitle*, and *DepartmentID*.

* Primary Key: *CourseID* is a unique way to identify each course by a specific and unique number. Some classes will have the same name and be different courses, so it is important to ensure that each course has a unique identifier.
* Foreign Key: *DepartmentID* is a foreign key within the course table. This allows us to identify a course with a specific department.

**AcademicYear** is a category table that holds the academic year information. It essentially just contains the values *AcademicYearID* and *Year*. Realistically, you could have just *year* as the only column, but adding an ID makes it a little easier when relating to other tables.

* Primary Key: *AcademicYearID* essentially uniquely identifies each year so that it can be referenced in other tables in the future.

**CourseOffering** is a transaction table that holds all the information about when courses are offered. Some courses are only offered in specific years or terms, so it is important to hold this information when collecting syllabi. This table has columns such as *OfferingID*, *Term*, *AcademicYearID*, and *CourseID*.

* Primary Key: *OfferingID* is a way to uniquely identify different statuses of offering based on term, academic year, etc.
* Foreign Key: *AcademicYearID* is referenced here to understand which academic years certain courses may be offered or not.
* Foreign Key: *CourseID* is referenced here to connect whether a course is offered or not.

**Syllabus** is a table that holds all the important information regarding the syllabi themselves. This table contains columns *SyllabusID*, *SyllabusDesc*, and *OfferingID*.

* Primary Key: *SyllabusID* is a way to uniquely identify each syllabus as its own entity. Some syllabi are likely to have the same or similar names, so it is important to ensure that each syllabus has a number to uniquely identify it.
* Foreign Key: *OfferingID* is referenced in this table to ensure that a syllabus would be offered in a specific term. This is to ensure that we know if we need to get a professor’s syllabus for a specific term or not.

**Professor\_Course** is a junction table that holds the many-to-many relationships that courses here have at OSU. Sometimes professors teach multiple courses, but sometimes courses have many professors as well, for example, the blueprint courses. This table contains columns *ProfessorID* and *OfferingID*

* Primary Key: (*ProfessorID, OfferingID*) are both used to uniquely identify this table as a joint connection. This will uniquely identify Professors with the OfferingID which also holds course information. Because of this, we can identify multiple professors with multiple courses, and vice versa.
* Foreign Key: *ProfessorID* is referenced in this table so that we can match the professor with an *OfferingID*.
* Foreign Key: *OfferingID* is referenced in this so that we can match the Course with a *ProfessorID.*

**Student\_Course** like the previous table, is also a junction table that holds the many-to-many relationship that courses have with students. Students can take many classes, and classes can also have many students. This will give us an opportunity to see what students are taking, what classes, and what classes have how many students as this is necessary information for understanding whether a syllabus is necessary from a professor or not.

* Primary Key: (*StudentID, OfferingID*) are both used to uniquely identify this table as a joint connection. This will uniquely identify Students with the OfferingID which also holds course information. Because of this, we can identify multiple students with multiple courses, and vice versa.
* Foreign Key: *StudentID* is referenced in this table so that we can match the student with an OfferingID.
* Foreign Key: *OfferingID* is referenced in this so that we can match the Course with a StudentID.

## Chapter 5 – Prototype Design and Implementation

For JW’s process it was noted that JW had to manually inspect around 300 syllabi per term and rename those files to a specific naming convention. Because of this, I figured it would be a fantastic idea to try and make a program which automates this process. This program searches through all Syllabi within a folder (either .docx, .doc, or .pdf) given by the user and searches for the terms BA, BIS, MRKT, MGMT, and ECON as well as the 3-digit code after these terms. For example, if the course was BIS372, the program would then return BIS372 to the user. It will also search for any emails ending with @oregonstate.edu or @bus.oregonstate.edu and return the beginning part – for example, an email being [muntzc@oregonstate.edu](mailto:muntzc@oregonstate.edu) would then be returned as muntzc. With those examples, it would then rename the syllabus file to BIS372muntzc. While I am unsure about the exact naming convention that JW typically uses, I figured this is a fine naming convention that can be easily used for this process.

This program will also print to the screen what file was renamed to what, as well as totals for how many files were successfully renamed and totals for how many files were unable to be renamed. The files that were unable to be renamed are then moved to a folder titled “Incomplete.” If this folder does not already exist, the program will create the folder in the folder that the program is run from.

Overall this program may not be perfect and in the past has messed up on a few files, although I have worked out a lot of the kinks recently. Out of the albeit small sample of 13 syllabus files used for testing, all of them have been renamed correctly with a 100% success rate and hopefully with a larger sample size this will not change too drastically. However, even with the potential faults, I believe this program will overall make JW’s life easier by automatically renaming most of the files correctly which will allow JW to not have to manually rename every file any longer.

### Program Source Code:

(Python Script)

*#AutoRenamer.py*

*#Created by Cade Muntz for the use of the College of Business at Oregon State University.*

import os

import docx2txt *# Extracts text from docx files*

import PyPDF2 *# extracts text from PDF files*

import textract *#extract texts from DOC files*

import re

import shutil

*#Function to extract information from specifically docx files*

*def* extract\_from\_docx(filename):

    text = docx2txt.process(filename) *#turns doc into text*

    lines = text.split("\n") *#splits text into individual lines*

*#establishes both as None to start, only replaces if something is found to replace with*

    course\_number = None

    instructor = None

*# Look for course number using "Course Number:" and alternative pattern*

    pattern = r"(BA|BIS|MRKT|MGMT|ECON)[\s\_]\*(\d{3})" *#establishes the pattern to look for*

*#Loop to iterate through the lines*

    for line in lines:

            if course\_number is None:

                match = re.search(pattern, line) *#Otherwise, we look for the pattern and the 3 digits after the pattern*

                if match:

                    course\_number = match.group(0) *#if the pattern matches me match the pattern with the 3 digits*

                    line = line.replace(course\_number, "").strip() *#and replace*

                    course\_number = course\_number.replace("\_", "")

*# Extract instructor email*

            if instructor is None and re.search(r"([\w.-]+)[\s]?@(oregonstate\.edu|bus\.oregonstate\.edu)", line): *#search for either @oregonstate or @bus.oregonstate*

                email = re.search(r"([\w.-]+)[\s]?@(oregonstate\.edu|bus\.oregonstate\.edu)", line).group(1) *#find only the first occurence*

                instructor = email.split('@')[0] *#split first half from the email half*

    return course\_number, instructor *#return the values :)*

*#Same thing but for PDF files*

*def* extract\_from\_pdf(filename):

    course\_number = None

    instructor = None

    with open(filename, "rb") as f:

        reader = PyPDF2.PdfReader(f)

        for page in reader.pages:

            text = page.extract\_text()

            lines = text.split("\n")

            pattern = r"(BA|BIS|MRKT|MGMT|ECON)[\s\_]\*(\d{3})"

            for line in lines:

                if course\_number is None:

                    match = re.search(pattern, line) *#Otherwise, we look for the pattern and the 3 digits after the pattern*

                    if match:

                        course\_number = match.group(0) *#if the pattern matches me match the pattern with the 3 digits*

                        line = line.replace(course\_number, "").strip() *#and replace*

                        course\_number = course\_number.replace("\_", "")

*# Extract instructor email*

                if instructor is None and re.search(r"([\w.-]+)[\s]?@(oregonstate\.edu|bus\.oregonstate\.edu)", line): *#search for either @oregonstate or @bus.oregonstate*

                    email = re.search(r"([\w.-]+)[\s]?@(oregonstate\.edu|bus\.oregonstate\.edu)", line).group(1) *#find only the first occurence*

                    instructor = email.split('@')[0] *#split first half from the email half*

    return course\_number, instructor *#return the values :)*

*#Same thing for DOC files*

*def* extract\_from\_doc(filename):

    text = docx2txt.process(filename)  *# Extract text from DOC file*

    lines = text.split("\n")

    course\_number = None

    instructor = None

*# Look for course number using "Course Number:" and alternative pattern*

    pattern = r"(BA|BIS|MRKT|MGMT|ECON)[\s\_]\*(\d{3})"

    for line in lines:

        if course\_number is None:

            match = re.search(pattern, line)

            if match:

                course\_number = match.group(0)

                line = line.replace(course\_number, "").strip()

                course\_number = course\_number.replace("\_", "")

*# Extract instructor email*

        if instructor is None and re.search(r"([\w.-]+)[\s]?@(oregonstate\.edu|bus\.oregonstate\.edu)", line):

            email = re.search(r"([\w.-]+)[\s]?@(oregonstate\.edu|bus\.oregonstate\.edu)", line).group(1)

            instructor = email.split('@')[0]

    return course\_number, instructor

*#Calls the individual functions to extract from the file itself*

*def* extract\_from\_file(filename):

    \_, extension = os.path.splitext(filename) *#splits the filename from the extension (mysyllabus.pdf becomes .pdf i.e., extension = .pdf)*

    if extension == ".docx": *#if docx use docx function*

        return extract\_from\_docx(filename)

    elif extension == ".pdf": *# if pdf use pdf function*

        return extract\_from\_pdf(filename)

    elif extension == ".doc": *#if doc use doc function*

        return extract\_from\_doc(filename)

    else: *#otherwise, return nothing*

        return None, None

*# Path to the folder containing the files*

*#... While loop to ensure that the path is correct and repeatedly asks for a new path if the path is not a real path on their system*

while True:

    folder\_path = input("Enter the folder path: ")

*# Verify if the provided folder path exists*

    if os.path.isdir(folder\_path):

        break

    print("Invalid folder path:", folder\_path)

    print("Please ensure that you are entering a correct folder path where the files you would like to be re-named are kept")

print(" ")

print("File changes noted below!")

print("↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ ")

print("--------------------")

incomplete\_folder\_path = os.path.join(folder\_path, "Incomplete")

movedcount = 0

renamedcount = 0

*# Create the "Incomplete" folder if it doesn't exist*

if not os.path.exists(incomplete\_folder\_path):

    os.makedirs(incomplete\_folder\_path)

*# Iterate over files in the folder*

for file\_name in os.listdir(folder\_path):

    file\_path = os.path.join(folder\_path, file\_name) *#gets the folder path and filename*

    if os.path.isfile(file\_path):

        \_, extension = os.path.splitext(file\_name) *#splits the extension*

        if extension in ['.doc', '.docx', '.pdf']: *#makes sure the extension is one of these three*

            course\_number, instructor = extract\_from\_file(file\_path) *#assigns the variables*

            if course\_number is not None and instructor is not None: *#if the file has the correct information then the file will be renamed with said information*

                new\_file\_name = f"{course\_number.replace(' ', '')}{instructor}{extension}"

                new\_file\_path = os.path.join(folder\_path, new\_file\_name)

                os.rename(file\_path, new\_file\_path) *#Renames the file :)*

*#useful little print statement so the user knows which file got renamed to what*

                print("Renamed file:", file\_name)

                print("New file name:", new\_file\_name)

                print("--------------------")

                renamedcount+=1

*#if the information is not valid then it will let the user know the program couldnt find the course number or instructor information from said file*

            else:

                new\_file\_path = os.path.join(incomplete\_folder\_path, file\_name)

                shutil.move(file\_path, new\_file\_path)

                print("Skipping file:", file\_name)

                print("Reason: Course number or instructor information not found.")

                print("--------------------")

                movedcount+=1

*#if the extension isnt one of the three used then the program will skip the file*

        else:

            print("Skipping file:", file\_name)

            print("Reason: File format not supported.")

            print("--------------------")

print("Number of files renamed = ",renamedcount,"!!")

print("Number of files that were unable to be renamed= ",movedcount)

## Appendix

CREATE TABLE Department (

DepartmentID INT PRIMARY KEY NOT NULL,

DepartmentName VARCHAR(50) NOT NULL

);

CREATE TABLE Course (

CourseID INT PRIMARY KEY NOT NULL,

CourseCode VARCHAR(10) NOT NULL,

CourseTitle VARCHAR(255) NOT NULL,

DepartmentID INT NOT NULL,

FOREIGN KEY (DepartmentID) REFERENCES Department (DepartmentID)

);

CREATE TABLE AcademicYear (

AcademicYearID INT PRIMARY KEY NOT NULL,

Year INT NOT NULL

);

CREATE TABLE CourseOffering (

OfferingID INT PRIMARY KEY NOT NULL,

Term VARCHAR(10) NOT NULL,

AcademicYearID INT NOT NULL,

CourseID INT NOT NULL,

FOREIGN KEY (AcademicYearID) REFERENCES AcademicYear (AcademicYearID),

FOREIGN KEY (CourseID) REFERENCES Course (CourseID)

);

CREATE TABLE Professor (

ProfessorID INT PRIMARY KEY NOT NULL,

FName VARCHAR(50) NOT NULL,

LName VARCHAR(50) NOT NULL,

Email VARCHAR(50) NOT NULL

);

CREATE TABLE Student (

StudentID INT PRIMARY KEY NOT NULL,

FName VARCHAR(50) NOT NULL,

LName VARCHAR(50) NOT NULL

);

CREATE TABLE Syllabus (

SyllabusID INT PRIMARY KEY NOT NULL,

SyllabusDesc VARCHAR(250) NOT NULL,

OfferingID INT NOT NULL,

FOREIGN KEY (OfferingID) REFERENCES CourseOffering (OfferingID)

);

CREATE TABLE Professor\_Course (

ProfessorID INT NOT NULL,

OfferingID INT NOT NULL,

PRIMARY KEY (ProfessorID, OfferingID),

FOREIGN KEY (ProfessorID) REFERENCES Professor (ProfessorID),

FOREIGN KEY (OfferingID) REFERENCES CourseOffering (OfferingID)

);

CREATE TABLE Student\_Course (

StudentID INT NOT NULL,

OfferingID INT NOT NULL,

PRIMARY KEY (StudentID, OfferingID),

FOREIGN KEY (StudentID) REFERENCES Student (StudentID),

FOREIGN KEY (OfferingID) REFERENCES CourseOffering (OfferingID)

);