|  |  |
| --- | --- |
|  | **CSCI/ISAT B320**  **Database Management Systems I**  **Fall 2023**  **Project Contributions by Team Member** |

**Purpose:**

Document the contributions of each team member over the course of the project.

**Members and their Contact Information**

|  |  |  |
| --- | --- | --- |
| **Member** | **Email** | **Text** |
| Ian Bickford | [bickfori@email.uscb.edu](mailto:bickfori@email.uscb.edu) | (978) 895-9569 |
| Houston Henderson | [hh41@email.uscb.edu](mailto:hh41@email.uscb.edu) | (854) 345-0275 |

**Overall**

Relative Contribution of each member over the course of the entire project

|  |  |  |
| --- | --- | --- |
| **Member** | **Contribution** | **Total Hours** |
| Ian Bickford | 60.00% | 52.5 |
| Houston Henderson | 40.00% | 37.5 |

**Data Design (i.e., ERD Creation & Revisions)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Contribution** | **Hours** | **Components** |
| Ian Bickford | 60 % | 6 | ERD Creation, attribute identification, restructured ERD to fit specific needs, finalized ERD, mitigated redundancies, and ensured compliance with 3NF. Redesigned original ERD to comply with changes during the development process. |
| Houston Henderson | 40 % | 3.5 | ERD Creation, attribute identification, assisted with the finalized ERD design |

**Create & Populate Script: Entity Creation (i.e., Table, View, Constraint, etc.)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Contribution** | **Hours** | **Components** |
| Ian Bickford | 55% | 20 | Utilized LucidChart built in export to SQL creation script, set null definitions, cleaned LucidChart code to match coding standards. Wrote creation and drop statements for the views added to the script. Created additional constraints for integrity of input values for Acdemicterm table. |
| Houston Henderson | 45% | 15 | Used LucidChart to built framework for early stage database design, followed up with modifications to tables that needed changing during the trials of database operation, for example prerequisites. Modified a Python program to assist with checking for prerequisites and creating prerequisites csv for future population from view, wrote a SQL script for random enrollment creation script, created enrollments for CSCI students and alumni. Wrote python script for generating random-mock information for students. Created student table, gradeinfo, prerequisites, instructors, buildings. |

**Create & Populate Script: Entity Population (i.e., Table Inserts)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Contribution** | **Hours** | **Components** |
| Ian Bickford | 60 % | 20 | Created inserts for CourseCatalog, CourseOfferings, Enrollments, InstructorDepartments, AcademicTerm, and Departments. Refined insert statements for GradeInfo for additions to the table, and refined the insert statements for Instructors table. |
| Houston Henderson | 40% | 12 | Created inserts for Students, GradeInfo, Prerequisites, instructors, and Buildings. Created update for PreRequisiteID in CourseCatalog. |

**Query Script: Query Development**

Note: include here any Views created to support your queries

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Contribution** | **Hours** | **Components** |
| Ian Bickford | 50 % | 5 | Created multi-part view system for the InstructorPerformance view. Tested each table being joined with applicable tables to ensure the database could be used effectively. Created ViewConstraints script |
| Houston Henderson | 50% | 4 | Created Test Queries Script( Unused in final but part of debugging ). Created GPA query. |

**Presentation Preparation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Contribution** | **Hours** | **Components** |
| Ian Bickford | 20 % | 1.5 | Assisted in timeline creation and practiced presentation points. |
| Houston Henderson | 80% | 3 | Created frame work for assumptions, issues and resolutions. Populated half of assumptions with related data . Created bibliography and slides for Powerpoint. |

**Naming**

**Conventions**

**Rules for Naming Tables:**

-We decided to name all our tables by what they represent and in the plural form. The former was done because table names should reflect what the data within them represent and the latter was done to differentiate between the columns and the table itself. For example, a table that represents many buildings would be named “Buildings” and the column that identifies each specific building would be named “Building”. This allows us to reference the whole of the buildings (the table) and one specific building (the column, uniquely represented by the row that contains the value for that column).

-A linking table such as our “InstructorDepartments” table was named based on the idea that 1 instructor could be in zero, one, or many departments. As such, we named the first part of the table Instructor and the latter part Departments to display this idea.

**Rules for Naming Columns:**

-We decided to name all our columns by using the singular form referencing the plural forms used by our tables. We acted out of this naming convention if the column was truly unique across all tables as a way to use the least amount of words possible. An example of this would be between the Students and Teachers tables. They both have a LName column that identify the last name of a given student or teacher. In this case, we named the columns StudentLName and TeacherLName respectively. Doing this we have identified that both of these columns represent a last name, but also allowed the user to identify that the column specifically belongs to a particular table. An advantage of doing this is if you were to join these two tables utilizing another table, you would be able to just use the column name without having to clarify which table the LName belonged to. This avoids the error “ambiguous column name”.

-For columns that would not be confused with any other table, we were ok with dropping the singular form of the table name from the column name. An example of this would be CRN in the CourseOfferings table. Since CRN appears nowhere else in any of the other tables, we felt it was unnecessary to name this column CourseOfferingCRN.

**Rules Regarding Primary Keys (PK):**

-Not every single table in our database uses a primary key. We have a linking table that does not use a primary key because it represents a many-many relationship. One of the linking tables in question is the InstructorDepartments table. This table does not have a column that uniquely identifies a row in every instance, as such it contains just 2 foreign keys that complete the table. This is also the case for our enrollments table, as it does not have a primary key either and is made up of foreign keys.

-In 1 singular case we have a primary key that is not its table name. This would be the PreRequisites table. This was because we ran into an issue with redesigning our database and with future work could be refined to have a primary key named after its specific table. (See Assumptions/Issues portion for more info)

-Every other table in our database has a primary key and it is represented by its table name/ identity column. An example of what we mean by this would be in the Students table we have the singular form StudentID to represent the identity column and primary key.

**Rules for Naming Constraints:**

-We named constraints by adding a prefix abbreviation (indicating what type of key it is) and an underscore to an existing {tablename}.columnname except for primary keys, which are indicated just by PRIMARY KEY {columnname}. An example of this would be FK\_CourseOfferings.CourseCatalogID to indicate that the constraint is a foreign key referencing the primary key that exists in CourseCatalog named CourseCatalogID (the naming statement is in the create script attached to the table creation statement).

-All abbreviations in our tables that reference a type of key are FK and AK. FK stands for foreign key and AK stands for alternate key.

-AK was used for the AcademicTerm table in the Season and TermLength columns to constrain input values to only values that made sense to reside in those columns. An example of this is the Season column is restricted by AK\_AcademicTerm.Season to values ‘Fall’, ‘Spring’, and ‘Summer’ as these are the only seasons that you can attend classes at USCB.

**Rules for Naming Views:**

-We named our views by what they represent with a ‘vw’ at the beginning. For example, the view that calculates the GPA for students is vwGPA.

Assumptions, Issues and Resolutions

**PreRequisites** –

**Issue**: Prerequisites were not very well documented in an easy to extract format within the provided data. Instead, the required course numbers and four-digit course name were contained within a larger string that offered this information and various other requirements.

**Resolution**: Using a python script, we searched course descriptions where the strings containing prerequisites were stored. We were looking for four-digit alphabetical values followed by three-digit numerical values, and then stored this information in a new `.xlsx` file that contained a newly populated prerequisites column. This was done using the `re` and `pandas` libraries` as imports*.*

**---------------------------------------------------------------------------------------------**

**Assumption**: If query contained “Research” or “Master” in the column “TitleShortDescription”, then the class would be a graduate level class.

**Issue**: No Issue.

**Resolution**: Filter out the classes contain the words “Research” or “Master” in the column “TitleShortDescription”. The remaining classes were the classes that were available for the undergraduate student.

**---------------------------------------------------------------------------------------------**

**Issue**: In the original design of the database, we sought out to use the PreReqID column as a primary key in the above table. However, after a redesign as to how we searched for the prerequisites required for a course, the PreReqID had to be changed to a foreign key to allow for multiple entries for each course. Due to this, the PreReqID column is a redundant column to CourseCatalogID.

**Resolution**: Although it is possible to remove the PreReqID, it became evident that this proved to be far more time consuming than we had to alleviate the issue. Instead, we are documenting that with further time and future work, this column could and would be removed from the above table. The time-consuming issue we ran into was unavoidable. This issue stemmed from downloading the contents of the PreReq table into in .xslx file, removing the column entirely from the data, saving as a .CSV, altering the create/populate script accordingly, and finally running the .CSV file through an online “convert CSV to SQL”. However, when this approach was attempted, we continued to get around 500 insert statements that were not formatted correctly at all. We would either have to manually edit 500 insert statements in 3 separate locations or allow for this issue to remain and focus on more important parts of the project. We chose the latter option and focused on the more vital portions of our work instead.

**---------------------------------------------------------------------------------------------**

**Assumption**: Finding all possible prerequisites for each course could be done with simple processing of data based on a columns value, allow us to make realistic mock data.

**Issue**: A non-uniform method of formatting prerequisites for each course which had strings of various size text and isolation of course numbers into specific columns. Also, non collegiate level prerequisites were stored as various sized strings and did not have a standard numerical representation, which is the opposite of how collegiate course numbers were stored in the data, as the course numbers are a 4 letter abbreviation followed by numerical values.

**Resolution**: When creating mock data, we aimed at the very least for students to have all prerequisites for BIOL, CSCI, and ISAT major courses were correct. Enrollments generated with the random enrollment script required classes with prerequisites to have prerequisite #1 met.

**---------------------------------------------------------------------------------------------**

**Enrollments** -

**Issue**: Time conflicts in population of classes.

**Resolution**: Class times were ignored regarding mock data.

**---------------------------------------------------------------------------------------------**

**Issue**: ISAT was not offered more than 3 years ago.

**Resolution**: CSCI courses populated most of the enrollments, who graduated with ISAT, if the student was enrolled in classes before 2021.

**---------------------------------------------------------------------------------------------**

**Issue**: Manually inputting student's information leaves a large margin for error, while also requiring a huge amount of time just to input the data. To input the student information with perfect accuracy would typically require external software that supports looping while also conditionals that follow rules for prerequisites, duplicate classes, passing grades, etc.

**Resolution**: While we manually input a fair amount of the data by hand, a script was also created to attempt to follow the conditional guidelines for checking for prerequisites and selecting classes directly out of certain majors, while checking if previous classes were passed and if so, then students were not allowed to enroll in the class again. This was done with TSQL declaration of variables and multiple if statements checking various conditions. In its current form this is a work in progress but works well enough to populate, using multiple executions. If we were to continue to build the TSQL `RandomEnrollmentPopulation.sql` script, then we could ideally make it recursive until a specific number of credits in each semester have been obtained. This was beyond the scope of the project but worked well enough to populate the alumni fields with a loose set of rules revolving around credits, semester, previous classes taken, and grades.

**---------------------------------------------------------------------------------------------**

**GradeInfo** -

**Issue**: An explicit column was not created for cumulative `GPA` of all courses completed for each student.

**Resolution**: A query calculating the `GPA` using the cumulative amount of `GradePoints` earned and `MinimumCredits` in a query to result in the `GPA`.

**---------------------------------------------------------------------------------------------**

**Student**-

**Assumption**: Having a way to generate student info with code not only saves times and ensures that at minimum, the structure of the data follows a specified format. It also creates a frame work for future projects which will allow random-fake data to be populated in a rapid manor on this project, while creating reuse-ability.

**Issue**: Writing random information by manual input is tedious, error prone, and time consuming. This issue is not a result of the assumption, instead the issue was precursor for the assumption leading to the resolution.

**Resolution**: To curb the chance for making mistakes with manual input and also save time in the future when faced with this task again, I researched how to create fake student information in python. The result was using the faker library to create a CSV of fake student information.

**---------------------------------------------------------------------------------------------**

**Instructors**-

**Assumption**: `Professor` data could be split into left and right strings at the comma to return the first and last name in their respective columns.

**Issue**: Some professors have a middle name initial with a period in their title.

**Resolution**: The middle name is included as part of the `ProfessorFName', therefore accurate queries could be made on `ProfessorLName` which is a common method of sorting, i.e. by last name.

**---------------------------------------------------------------------------------------------**

**Buildings**-

**Assumption**: We can provide the city for each building city by the real location on Bluffton/HHI/Beaufort.

**Issue**: The data offers the course campus in terms of institution main office location , but no dedicated column for the sub-campus location or building location such as “Building 312 on Hilton Head”, therefore without doing deep research into where each building is physically located, we could not ensure accurate assignment of `BuildingCity`.

**Resolution**: Due to the time required to research the physical location related to `BuildingID`, we determined this falls outside of the scope of the project. If the building was on the USCB campus, it was assigned to Beaufort.

**---------------------------------------------------------------------------------------------**

**CourseCatalog**-

**Assumption**: All courses with the ScheduleType of “Other” were removed from the data. Often, this “other” schedule type was redundant data. We cannot say for sure all of the “other” types were redundant, but we can assume they are as there are too many to inspect.

**Issue**: No issue.

---------------------------------------------------------------------------------------------

**Assumption**: We assumed that all courses without a MaximumCredit value would be equal to the MinimumCredit value within that course. As such, we ran a query to test for null values in the max credit field and updated the null value to the minimum credit.

**Issue**: No issue.

**Resolution:** All course offerings have a minimum and maximum credit hours available that are accurate to the course.

---------------------------------------------------------------------------------------------

**AcademicTerm**-

**Assumption**: We assumed that all the semesters’ start dates and end dates for each entry would be the same for the year before it. We did this to save time from manually searching for the term state and end date for the 10 years we were given. This would not cause a significant issue as terms start and end +- 2 days from each other each year.

**Issue**: The start and end dates are not 100% accurate.

**Resolution**: The start and end dates are reasonably close enough to the actual values.

---------------------------------------------------------------------------------------------

**Departments**-

**Assumption**: We can provide the city for each building city by the real location on Bluffton/HHI/Beaufort.

**Issue**: The data offers the course campus in terms of institution main office location, but no dedicated column for the sub-campus location or building location such as `Building 312 on Hilton Head`, therefore without doing deep research into where each building is physically located, we could not ensure accurate assignment of `BuildingCity`.

**Resolution**: Due to the time that would be required to research the physical location related to building ID, we determined this falls outside of the scope of the project. If the building was on the USCB campus, it would be assigned to Beaufort.

---------------------------------------------------------------------------------------------

**CourseOfferings**-

**Assumption**: Modality values were assumed based on the building that the course was offered at. If the building was equal to “Web2”, we considered this value to be an online class. If the value was anything other than this value, it would be an in-person class.

**Issue**: If there is a value other than “Web2” that represents an online class, it may be misidentified as an in person class.

**Resolution**: It is far too time consuming to verify all 17,000+ rows are the correct modality, this solution satisfies the aim of the project and wasn’t required.

**---------------------------------------------------------------------------------------------**

**InstructorDepartments**-

**Assumption**: Instructors may be assigned more than one department resulting in duplicates of the `InstructorID` as it is associated to multiple departments.

**Issue**: No issue, just an acknowledgment in the building process.

**Resolution**: This assumption resulted in the table not having a primary key in `InstructorDepartments`.

**---------------------------------------------------------------------------------------------**

*Note: Items surrounded with ` ` refer to direct names from the code base and database.*

**Bibliography**

*Generating random numbers for each row in various SQL databases*. Programming Code Examples. https://www.code4example.com/sql/generating-random-numbers-for-each-row-in-various-sql-databases/

*Creating fake data with Python Faker*. Udacity. (2023, March 22). https://www.udacity.com/blog/2023/03/creating-fake-data-in-python-using-faker.html

*Easily convert files into SQL databases*. SQLizer. (n.d.). https://sqlizer.io/

*How case when works*. The Data School. (n.d.). https://dataschool.com/how-to-teach-people-sql/how-case-when-works/

MatthewMartin, SQLMenace, & Jeremy SmythJeremy Smyth. (2009, June 1). *How do I generate a random number for each row in a T-SQL select?.* Stack Overflow. https://stackoverflow.com/questions/1045138/how-do-i-generate-a-random-number-for-each-row-in-a-t-sql-select

Michael Stum, RichardTheKiwi. (1956, December 1). *Turning a comma separated string into individual rows*. Stack Overflow. https://stackoverflow.com/questions/5493510/turning-a-comma-separated-string-into-individual-rows

Michael Stum, & RichardTheKiwi. (2012, March 1). *Turning a comma separated string into individual rows*. Stack Overflow. https://stackoverflow.com/questions/5493510/turning-a-comma-separated-string-into-individual-rows

Ravi. (n.d.). *How to create fake user profiles using Faker in python*. Educative. https://www.educative.io/answers/how-to-create-fake-user-profiles-using-faker-in-python

rwestMSFT. (n.d.-a). *If...else (transact-SQL) - SQL server*. SQL Server | Microsoft Learn. https://learn.microsoft.com/en-us/sql/t-sql/language-elements/if-else-transact-sql?view=sql-server-ver16

rwestMSFT. (n.d.-b). *Variables (transact-SQL) - SQL server*. SQL Server | Microsoft Learn. https://learn.microsoft.com/en-us/sql/t-sql/language-elements/variables-transact-sql?view=sql-server-ver16

Syverson, B., & Murach, J. (2023). *Murach’s SQL Server 2022 FOR DEVELOPERS: Training & reference*. Mike Murach & Associates, Inc.

*Welcome to Faker’s documentation!* Welcome to Faker’s documentation! - Faker 24.14.1 documentation. (n.d.). https://faker.readthedocs.io/en/master/

Chaudhary, E. (n.d.). Prerequisite Parser Python. computer software. Retrieved April 14, 2024.

Insert statements from existing tables made possible by Iambacck on GitHub: <https://github.com/lambacck/generate_inserts>