**A black background with a black square

Description automatically generated with medium confidence**

**Database Specification**

For

**Online Learning Platform**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | Full Name | ID | Dept | SEC# | M#1 | M#2 | M#3 | M#4 | Bonus | Total |
| 1 | Aly Mohamed Salama Elsharkawy | 320220040 | CSC | 1 |  |  |  |  |  |  |
| 2 | Belal Mohamed Elsayed Salem | 320220015 | CSC | 1 |  |  |  |  |  |  |
| 3 | Ahmed Samy Hekal | 320230170 | CSC | 1 |  |  |  |  |  |  |
| 4 | Mostafa Mohamed Essa Mohamed | 320230181 | CSC | 1 |  |  |  |  |  |  |
| 5 | Mohamed Nasser Mohamed Massoud | 320230119 | CSC | 1 |  |  |  |  |  |  |

Total

**Table of Contents:**

[**Business Requirements Specification:** 3](#_Toc185705328)

[**Scope:** 4](#_Toc185705329)

[**Business Rules and Flow:** 4](#_Toc185705330)

[**System Actors:** 6](#_Toc185705331)

[**Use Case Diagrams:** 6](#_Toc185705332)

[**Administrator Use Case:** 6](#_Toc185705333)

[**Instructor Use Case:** 7](#_Toc185705334)

[**Student Use Case:** 8](#_Toc185705335)

[**System Overview:** 9](#_Toc185705336)

[**Student Requirements:** 9](#_Toc185705337)

[**Instructor Requirements:** 10](#_Toc185705338)

[**Admin Requirements:** 12](#_Toc185705339)

[**Shared Requirements:** 12](#_Toc185705340)

[**Non-functional requirements:** 12](#_Toc185705341)

[**Database Overview:** 13](#_Toc185705342)

[**Entity Relationship Diagram:** 14](#_Toc185705343)

[**Database Mapping:** 14](#_Toc185705344)

[**Non-Normalized:** 14](#_Toc185705345)

[**Normalized:** 14](#_Toc185705346)

[**Data Dictionary:** 15](#_Toc185705347)

[**Administrator Tables:** 15](#_Toc185705348)

[**Instructor Tables:** 18](#_Toc185705349)

[**Student Tables:** 22](#_Toc185705350)

[**Organization Tables:** 28](#_Toc185705351)

[**Financial Tables:** 30](#_Toc185705352)

[**Courses Tables:** 32](#_Toc185705353)

[**Modules and Questions:** 35](#_Toc185705354)

[**SolvedQuestions:** 42](#_Toc185705355)

[**Miscellaneous:** 48](#_Toc185705356)

# **Business Requirements Specification:**

This specification covers technical and non-technical aspects of the system, usages, key features, functional and non-functional requirements, and specific UML diagrams. The audience is not limited to software engineers, but also includes instructors and students. This specification will mainly focus on the Online Learning System (OLS) from the user perspective. Specific attention will be brough to the user’s requirements and their effects on the resulting database.

The document is divided into five large chapters. The first chapter is the introduction and will introduce the outline of the document and provide a brief overview of the system. The OLS actors will be introduced and explained through UML Use Case diagrams. The second chapter will be a system overview. It will delve into the details of the functional and non-functional requirements. We will also begin “conceptualizing” the database through the use of a UML Data Flow diagram. The third chapter will talk about the database in great detail. It will begin with the database’s high-level requirements and finish with the database’s data dictionary. The fourth chapter will be a guide on the database’s usage. Select queries will be shown. The queries will be grouped by the kind of actor they were designed for. The fifth and final chapter will contain snapshots of the system. This includes both states of the database at certain intervals of time and screenshots of the database’s frontend.

## **Scope:**

The Online Learning System (OLS) is designed to streamline operations and improve efficiency in the personalized online learning space by establishing a clear user hierarchy and role-based functionality. Key user roles include instructors, administrators, and students. Instructors have the ability to moderate the platform and view analytics about the platform. Furthermore, they can create new instructor users and “Organization” entities. Instructors can either be personal instructors, with their personal courses, or organizational instructors where their created courses are shared between them. Students can enroll in these courses, pay the course fees, and complete the courses.

## **Business Rules and Flow:**

There are three user types in the OLS: Instructors, Students, and Administrators. Administrators are the “managers” of the OLS. There are also two “kinds” of administrators. Only administrator users can create other administrator users. This creation relationship must be logged. The first administrator who was not created by any other administrator users is considered the “root” user of the OLS. This user is the original user of the OLS. Furthermore, the root administrator user has full access to the OLS database. The root administrator user is also responsible for creating other administrator users and instructor users. Instructor users can only be created by administrator users. Administrators create their accounts in advance and send the credentials to an instructor user. The instructors then change their temporary credentials. Instructors are accepted and vetted through a process that is external to the OLS. After successful entry, instructor users can then create courses, either alone or with other instructor users. A course consists of a set of “modules”. These modules can in turn contain various questions such as true/false, multiple choice, and short answer. Furthermore, pre-recorded videos can be uploaded as well as reading texts. Courses also have some publicly available information about them: title, description, tags, and cost. Tags are used by the OLS for recommending the course and for student users to search for the course. Students can also search by title.

Student users are the most common type of user in the OLS. They can enroll in and pay for courses. Courses can only be accessed upon successful payment for course. Bank cards must be used and invoices will be generated and return upon successful payment. Students can also write public reviews about courses they are currently enrolled in. Furthermore, students can wishlist courses.

Organizations also exist on the OLS. These organizations must be registered by an administrator user. These organizations contain a set of instructions. All courses created by these instructors are then collectively by the organization. Lastly, all user types have some information common between them such as profile pictures, security information, contact information, and personal information. Organizations have background images, names, and descriptions which courses have thumbnails and descriptions.

## **System Actors:**

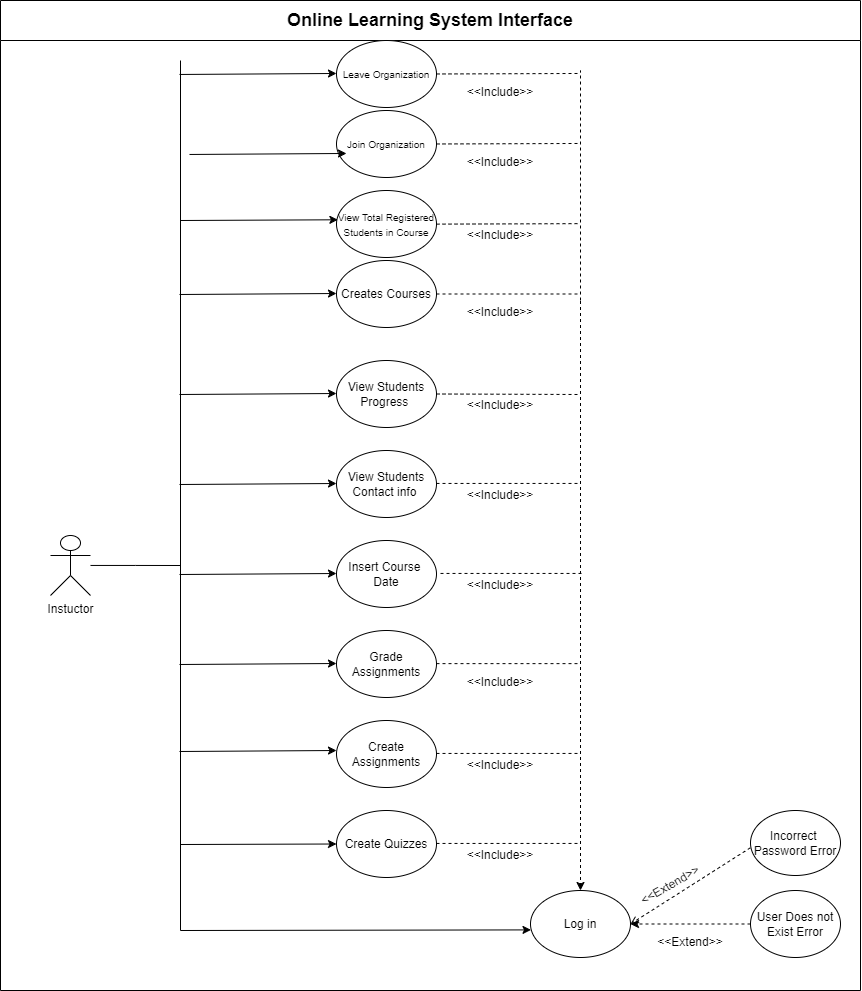
|  |  |
| --- | --- |
| **Administrator** | These are considered the “Business Owners” of the OLS. They are responsible for the creation of new Instructor users. Furthermore, they will be able to view analytics about the health and popularity of the OLS |
| **Instructor** | Instructors create courses on the OLS. They can either be personal instructors or organizational instructors. Furthermore, several instructors can choose to collaborate on a course. Furthermore, they can directly contact students if needed. |
| **Student** | Students are the most common type of user in the system. They can enroll in courses, pay for said courses, and complete said courses. They can also contact instructors and be contacted by them. |

## **Use Case Diagrams:**

### **Administrator Use Case:**

### **Instructor Use Case:**

### **Student Use Case:**



# **System Requirements Specification:**

Based on the high-level actor descriptions and project scope defined above, we will now be able to delve into the functional and non-functional requirements of the database and product as a whole. Furthermore, we can visualize the flow of information through the database and product with the aid of a data flow diagram. The functional and non-functional requirements will be listed first, each with a unique ID and priority. Then, the dataflow diagram will be shown.

## **Student Requirements:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Priority** |
| SR-1 | The System must allow the student to provide details (name, email, phone number, password, profile picture) to sign up on the platform. | HIGH |
| SR-2 | The system must allow the student to sign in using (email, password) on the platform. | HIGH |
| SR-3 | The system must show a list of available courses to students whether they are signed in or not. | HIGH |
| SR-4 | The system must allow students to enroll in a selected course. | HIGH |
| SR-5 | The system must show a student their enrolled course outline (modules). | HIGH |
| SR-6 | The system must allow the student to click on a quiz to start it. | HIGH |
| SR-7 | If a student is taking a quiz, then the system must allow the students to answer questions by selecting the correct answer from available answers represented as buttons. | HIGH |
| SR-8 | The system must track answered quizzes. | HIGH |
| SR-9 | The system must record and show the student his or her quiz score. | HIGH |
| SR-10 | The system must track course progress for the student. | HIGH |
| SR-11 | The system must allow the student to register for more than one course. | HIGH |
| SR-12 | The system must allow the student to withdraw from a selected course. | MEDIUM |
| SR-13 | The system must receive a certificate at the end of a course showing details as (Certificate ID, student name, course name). | MEDIUM |
| SR-14 | The system should notify a student if they haven’t accessed an enrolled course for 30 days. | LOW |
| SR-15 | The system must show completed courses and allow students to access them. | HIGH |
| SR-16 | For assignments the system must allow the student to upload their work and view grades they took. | HIGH |
| SR-17 | The system must allow students to search using tags or course names to find relevant courses. | HIGH |

## **Instructor Requirements:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Priority** |
| IR-1 | The system must allow instructors to sign in using their predetermined credentials (email, password). | HIGH |
| IR-2 | The system must allow instructors to create new courses and add  needed data (see “IR-6” to “IR-10”). | HIGH |
| IR-3 | The system must allow an instructor to view how many students are registered in each created course by them. | HIGH |
| IR-3 | The system must allow instructors to view each student progress in a specific course. | Medium |
| IR-4 | The system must allow the Instructor to view a student's contact information. | LOW |
| IR-5 | The system must allow instructors to view total students registered across all their created courses. | Medium |
| IR-6 | While creating a course. The system must allow instructors to upload thumbnails for their courses. | HIGH |
| IR-7 | While creating a course. The system must allow instructors to insert course data (name, price, description). | HIGH |
| IR-8 | While creating a course, the system must allow instructors to insert tags related to the course. | HIGH |
| IR-9 | While creating a course, the system must allow instructors to create assignments and insert their data. | HIGH |
| IR-10 | While creating a course. The system must allow instructors to create quizzes and insert their data (questions, prompt, and correct answer) and the question type (MCQ or T/F). | HIGH |
| IR-11 | The system must allow instructors to view and grade assignments. | HIGH |
| IR-12 | The system must allow instructors to view and provide feedback on quizzes. | LOW |

## **Admin Requirements:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Priority** |
| AR-1 | The system must allow admin to register accounts for instructors. | HIGH |
| AR-2 | The system must allow admin to view specific student details (Name, contact information). | HIGH |
| AR-3 | The system must allow admin to view all students who are registered in a specific course with semi-anonymous information. | HIGH |
| AR-4 | The system must allow admin to view the count of students who register on the platform every day. | HIGH |

## **Shared Requirements:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **PRIORITY** |
| SHR-1 | The system must allow any user to change their passwords. | HIGH |
| SHR-2 | The system must allow any user to change their profile details (profile picture, name, contact info). | HIGH |

## **Non-functional requirements:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **PRIORITY** |
| NFR-1 | The system must use hashing and salting during password authentication | HIGH |

## **Assumptions and Dependencies:**

The OLS is designed to be easy to use and deploy. The database must be hosted on a Linux server with the MariaDB database management software. However, the frontend of the OLS is currently available on Windows. It was created using the WinForms UI framework. However, it should work on all versions of Windows. All members of the OLS must be using the Windows operating system and have the OLS frontend installed on their computer.

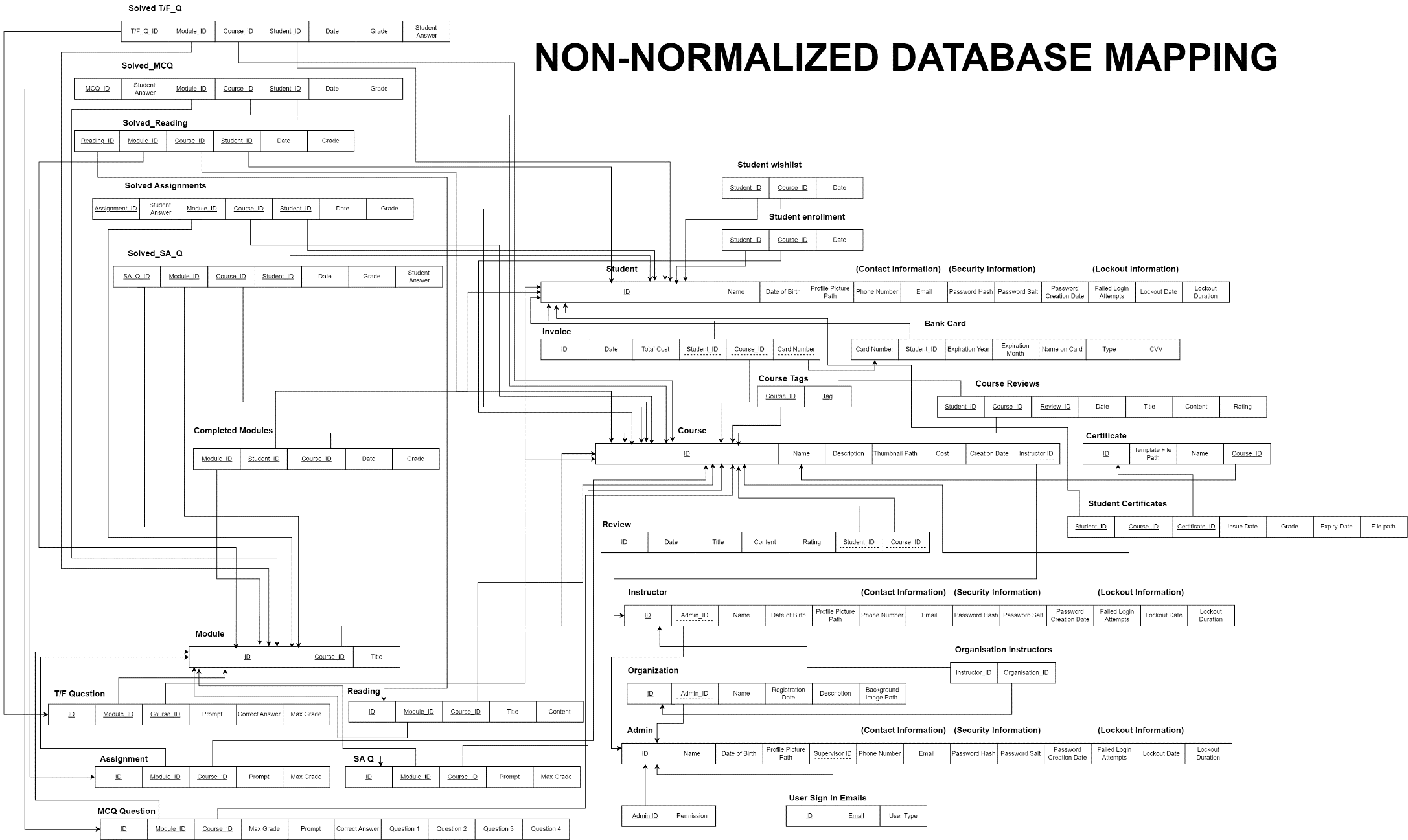
No prior knowledge is required of any actors to use the OLS. All database interactions will be abstracted through the use of the WinForms frontend. The only user who *could* have databases knowledge is the root administrator user. The root administrator user has access to the database. However, this power is only used in emergency situations.

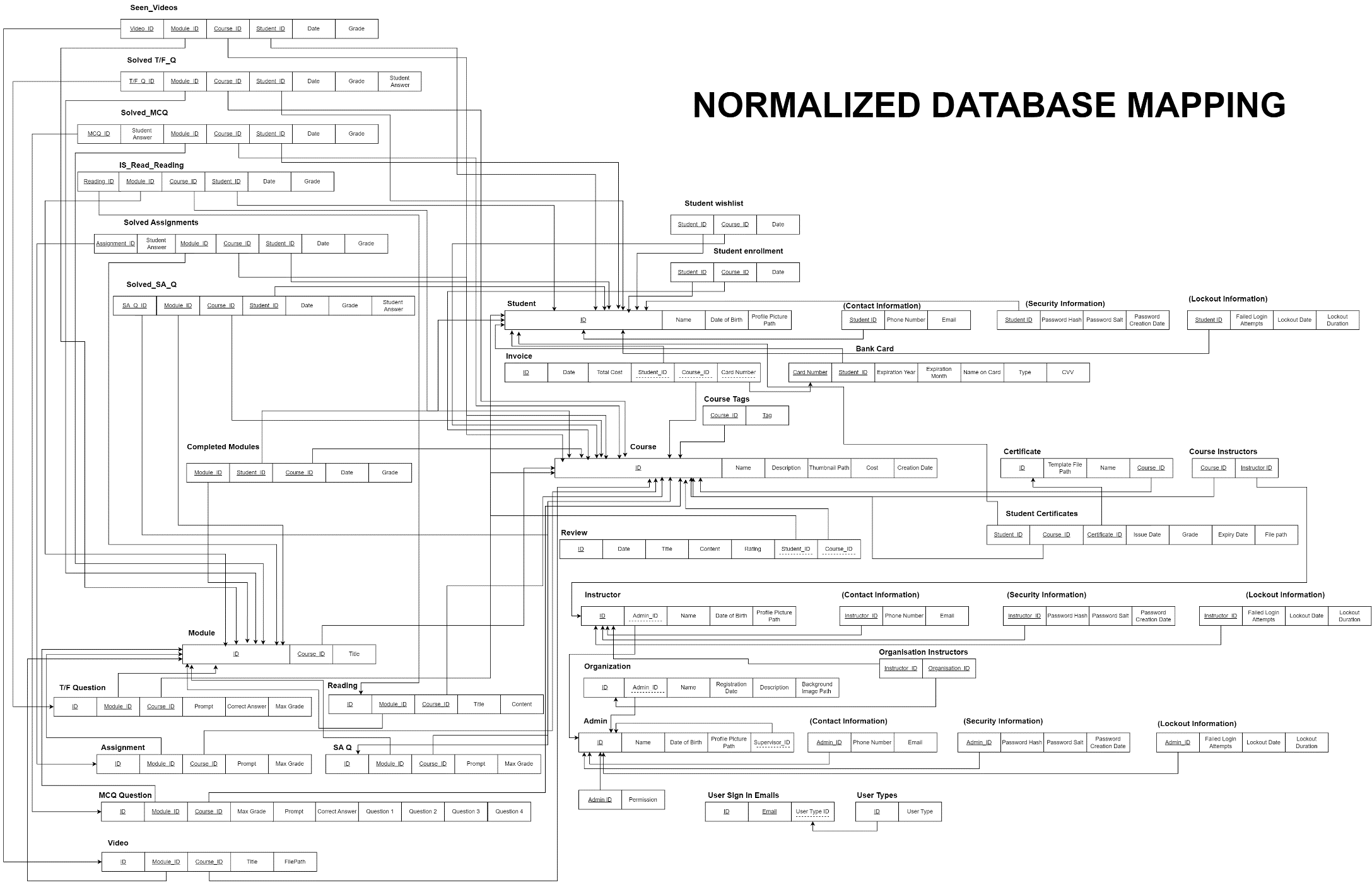
# **Database Overview:**

With the functional and non-functional requirements clearly illustrated alongside the flow of information in the OLS, it is now time to start discussing the details of the OLS’s database. Firstly, the ERD will be shown. This will be followed by the database’s mapping and schema. The mapping will be shown in various forms of normalization. The first diagram will have no normalization. The second diagram will be normalized until the 5th normal form where it is practical to do so. Lastly, a data dictionary of the database will be shown. It will discuss in great detail the tables and columns of the database. Furthermore, the SQL query used to create each table will also be shown. Please note that the DBMS used was MariaDB. This will affect the syntax used in the DDL SQL queries shown.

## **ERD and Mapping:**

### 





## **Data Dictionary:**

With the tables of the database clearly defined from the previous section, it is now time to start defining the low-level implementation details of said tables. This section will be separated into several sub-chapters: Instructor, Administrator, Student, Courses, Modules and Questions, Solved Questions, Invoices, Organizations, Bank Cards, and Log In. Each section will contain several DDL queries and tables related to the various tables that roughly make up that section. Then, a data dictionary entry will be shown for that table. The entry will contain the name of the field, its data type, and constraints and constraining attributes, its default value, and a short description.

### **Administrator Tables:**

**AdminInformation:**

This table contains all of the essential information that an admin is guaranteed to use when signed into the OLS. Other non-essential information has been normalized into other tables.

CREATE TABLE AdminInformation(

ID BIGINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(48) NOT NULL,

BirthDate DATE NOT NULL,

ProfilePicturePath VARCHAR(128) NOT NULL,

SupervisorID BIGINT UNSIGNED,

FOREIGN KEY(SupervisorID) REFERENCES AdminInformation(ID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID of user. |
| Name | VARCHAR(48) | NOT NULL | N/A | Name of user. |
| Birthdate | DATE | NOT NULL | N/A | Birthdate of user. |
| ProfilePicturePath | VARCHAR(128) | NOT NULL | N/A | File path to an image of the user stored on the server. |
| SupervisorID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key self-refrence to ID. This is the admin who created this admin user. |

**AdminContactInformation:**

This contains all of the non-essential contact information for the admin user type. The admin user type is only allowed to have 1 email and phone number.

CREATE TABLE AdminContactInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES AdminInformation(ID),

PhoneNumber VARCHAR(11) NOT NULL,

Email VARCHAR(32) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, AUTO\_INCREMENT, PRIMARY KEY,  FOREIGN KEY | AUTO\_INCREMENT | Foreign key to the admin ID in AdminInformation |
| PhoneNumber | VARCHAR(11) | NOT NULL | N/A | Phone number of the specified admin user. |
| Email | VARCHAR(32) | NOT NULL | N/A | Email address of the specified admin user |

**AdminSecurityInformation:**

This table contains the information needed for an admin user to sign in. This table was normalized because it is only needed during log in and sign up.

CREATE TABLE AdminSecurityInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES AdminInformation(ID),

PasswordHash CHAR(64) NOT NULL,

PasswordSalt CHAR(64) NOT NULL,

PasswordCreationDate DATE NOT NULL DEFAULT CURRENT\_DATE

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | N/A | Foreign key to the admin ID in AdminInformation. |
| PasswordHash | CHAR(64) | NOT NULL | N/A | String of 64 chars representing the SHA256 digest of the salted password |
| PasswordSalt | CHAR(64) | NOT NULL | N/A | Random string of 64 ASCII characters to be added to the end of admin’s password for hashing. |
| PasswordCreationDate | DATE | NOT NULL | CURRENT\_DATE | Date at which the specified admin user’s password was created and/or updated. |

**AdminLockoutInformation:**

This table contains all of the information needed for when an admin user fails to sign in. The failed login attempt be logged. If too many logins fail, then the admin will be locked out of their account temporarily.

CREATE TABLE AdminLockoutInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES AdminInformation(ID),

FailedLoginAttempts TINYINT UNSIGNED NOT NULL DEFAULT 0,

LockoutDate DATE,

LockoutDuration TINYINT UNSIGNED NOT NULL DEFAULT 0

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | A foreign key to the admin ID in AdminInformation. |
| FailedLoginAttempts | TINYINT | UNSIGNED, NOT NULL | 0 | Total amount of failed login attempts for the specified admin user. |
| LockoutDate | DATE | N/A | CURRENT\_DATE | When the user account was logged from too many failed login attempts. |
| LockoutDuration | TINYINT | UNSIGNED, NOT NULL | 0 | Lockout duration in minutes for the specified admin user. |

### **Instructor Tables:**

**InstructorInformation:**

This table contains all of the essential information that an instructor is guaranteed to use when signed in to the OLS. Other non-essential information has been normalized into other tables.

CREATE TABLE InstructorInformation(

ID BIGINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(48) NOT NULL,

BirthDate DATE NOT NULL,

ProfilePicturePath VARCHAR(128) NOT NULL,

AdminID BIGINT UNSIGNED NOT NULL,

FOREIGN KEY(AdminID) REFERENCES AdminInformation(ID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO INCREMENT | AUTO\_INCREMENT | Internal ID for instructor user. |
| Name | VARCHAR(48) | NOT NULL | N/A | File path to the profile picture of the instructor user on the server. |
| BirthDate | DATE | NOT NULL | N/A | Birthdate of the specified instructor user. |
| ProfilePicturePath | VARCHAR(128) | NOT NULL | N/A | File path on the server to the specified instructor’s user profile picture |
| AdminID | BIGINT | UNSIGNED, NOT NULL, FOREIGN KEY | N/A | Foreign key to ID in AdminInformation. |

**InstructorContactInformation:**

This contains all of the non-essential contact information for the instructor user type. The instructor user type is only allowed to have 1 email and phone number.

CREATE TABLE InstructorContactInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES InstructorInformation(ID),

PhoneNumber VARCHAR(11) NOT NULL,

Email VARCHAR(32) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key to admin ID in AdminInformation. |
| PhoneNumber | VARCHAR(11) | NOT NULL | N/A | Phone number for specified instructor user. |
| Email | VARCHAR(32) | NOT NULL | N/A | Email address for instructor user. |

**InstructorSecurityInformation:**

This table contains the information needed for an admin user to sign in. This table was normalized because it is only needed during sign in and sign up.

CREATE TABLE InstructorSecurityInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES InstructorInformation(ID),

PasswordSalt CHAR(64) NOT NULL,

PasswordHash CHAR(64) NOT NULL,

PasswordCreationDate DATE NOT NULL DEFAULT CURRENT\_DATE

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key to the admin ID in InstructorInformation. |
| PasswordSalt | CHAR(64) | NOT NULL | N/A | Random string of 64 ASCII characters to be added to the end of instructor’s password for hashing. |
| PasswordCreationDate | DATE | NOT NULL | CURRENT\_DATE | Date at which the specified admin user’s password was created and/or updated. |
| PasswordHash | CHAR(64) | NOT NULL | N/A | String of 64 chars representing the SHA256 digest of the salted password. |

**InstructorLockoutInformation:**

This table contains all of the information needed for when an instructor user fails to sign in. The failed login attempt be logged. If too many logins fail, then the admin will be locked out of their account temporarily. The timeout will be as severe as it is for admin users.

CREATE TABLE InstructorLockoutInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES InstructorInformation(ID),

FailedLoginAttempts TINYINT UNSIGNED NOT NULL DEFAULT 0,

LockoutDuration TINYINT UNSIGNED NOT NULL DEFAULT 0,

LockoutDate DATE

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key to admin ID in InstructorInformation. |
| FailedLoginAttempts | TINYINT | UNSIGNED, NOT NULL | 0 | Total amount of failed login attempts for the specified admin user. |
| LockoutDate | DATE | N/A | CURRENT\_DATE | When the user account was logged from too many failed login attempts. |
| LockoutDuration | TINYINT | UNSIGNED, NOT NULL | 0 | Represents the lockout duration in minutes for the specified admin user. |

### **Student Tables:**

**StudentInformation:**

This table contains all of the essential information that a student user is guaranteed to use upon being signed into the OLS. Other non-essential information has been normalized into other tables. This information needed is similar/identical to that of a instructor user.

CREATE TABLE StudentInformation(

ID BIGINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(48) NOT NULL,

BirthDate DATE NOT NULL,

ProfilePicturePath VARCHAR(128) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID used for the student user. |
| Name | VARCHAR(48) | NOT NULL | N/A | Name for specified student user. |
| BirthDate | DATE | NOT NULL | N/A | Birthdate for specified student user. |
| ProfilePicturePath | VARCHAR(128) | NOT NULL | N/A | File path to a profile picture on the server for the student user. |

**StudentContactInformation:**

This contains all of the non-essential contact information for the student user type. The student user type is only allowed to have 1 email and phone number. This information will be query-able by instructor user types if the student enrolls in one of their courses.

CREATE TABLE StudentContactInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES StudentInformation(ID),

Email VARCHAR(32) NOT NULL,

PhoneNumber VARCHAR(11) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key to ID in StudentInformation. |
| Email | VARCHAR(32) | NOT NULL | N/A | Email address for student user. |
| PhoneNumber | VARCHAR(11) | NOT NULL | N/A | Phone number for student user. |

**StudentSecurityInformation:**

This table contains the information needed for a student user to sign in. This table was normalized because it is only needed during log in. This table is identical to the security information tables for student and admin user types.

CREATE TABLE StudentSecurityInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES StudentInformation(ID),

PasswordSalt CHAR(64) NOT NULL,

PasswordHash CHAR(64) NOT NULL,

PasswordCreationDate DATE NOT NULL DEFAULT CURRENT\_DATE

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key to ID in StudentInformation. |
| PasswordSalt | CHAR(64) | NOT NULL | N/A | Random string of 64 ASCII characters to be added to the end of student’s password for hashing. |
| PasswordHash | CHAR(64) | NOT NULL | N/A | String of 64 chars representing the SHA256 digest of the salted password |
| PasswordCreationDate | DATE | NOT NULL | CURRENT\_DATE | Date at which the student user’s password was created and/or updated. |

**StudentLockoutInformation:**

This table contains all of the information needed for when a student user fails to sign in. The failed login attempt be logged. If too many logins fail, then the student will be locked out of their account temporarily. The timeout will be slightly less harsh than those of instructor and admin user types.

CREATE TABLE StudentLockoutInformation(

ID BIGINT UNSIGNED PRIMARY KEY,

FOREIGN KEY(ID) REFERENCES StudentInformation(ID),

LockoutDate DATE,

LockoutDuration TINYINT UNSIGNED NOT NULL DEFAULT 0,

FailedLoginAttempts TINYINT UNSIGNED NOT NULL DEFAULT 0

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key to ID in StudentInformation. |
| FailedLoginAttempts | TINYINT | UNSIGNED, NOT NULL | 0 | Total amount of failed login attempts for the student user. |
| LockoutDate | DATE | N/A | CURRENT\_DATE | When the user account was logged from too many failed login attempts. |
| LockoutDuration | TINYINT | UNSIGNED, NOT NULL | 0 | Represents the lockout duration in minutes for the specified student user. |

**StudentWishList:**

This simple table contains the courses that the student has wishlisted. It also stores when this action happened.

CREATE TABLE StudentWishList(

StudentID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

PRIMARY KEY (StudentID, CourseID),

Date DATE DEFAULT CURRENT\_DATE);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY, | N/A | Foreign Key to ID in StudentInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key to ID in CourseInformation. |
| Date | DATE | N/A | CURRENT\_DATE | Date when course was wishlisted by student user. |

**StudentEnrollment:**

This simple table stores the courses that a student has enrolled in. It also stores when the student enrolled in the course.

CREATE TABLE StudentEnrollment(

StudentID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

Date DATE DEFAULT CURRENT\_DATE,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

PRIMARY KEY (StudentID, CourseID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key to ID in StudentInformation. |
| CourseID | BIGINT | UNSIGNED | N/A | Foreign key to ID in CourseInformation |
| Date | DATE | N/A | CURRENT\_DATE | Date at which student enrolled in specified course. |

**StudentCompletedModules:**

Courses are composed of multiple modules. Students complete these modules individually. This table records when students’ complete modules along with the grade that they achieved.

CREATE TABLE StudentCompletedModules(

ID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

PRIMARY KEY(ID, StudentID, CourseID),

CompletionDate DATE NOT NULL DEFAULT CURRENT\_DATE,

Grade DECIMAL(5,2) NOT NULL CHECK (Grade > 0)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key refrence to ID in ModuleInformation. |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in StudentInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseID. |
| CompletionDate | DATE | NOT NULL | CURRENT\_DATE | Date at which student completed specified course module. |
| Grade | DECIMAL(5,2) | NOT NULL, CHECK (Grade > 0) | N/A | Grade received by user upon module completion. |

**StudentEarnedCertificates:**

Students, upon course completion, will earn a certificate. This table stores the *personalized* relation between a certificate and its student receiver. Information such as the completed certificates file path, its unique ID, expiry and issue date, and grade will be stored.

CREATE TABLE StudentEarnedCertificates(

StudentID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

CertificateID BIGINT UNSIGNED,

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(CertificateID) REFERENCES CourseCertificate(ID),

PRIMARY KEY (StudentID, CourseID, CertificateID),

IssueDate DATE NOT NULL DEFAULT CURRENT\_DATE,

Grade DECIMAL(4,2) NOT NULL CHECK (Grade > 0),

ExpiryDate DATE NOT NULL,

FilePath VARCHAR(128) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in StudentInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation |
| CertificateID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseCertificate |
| IssueDate | DATE | NOT NULL | CURRENT\_DATE | Date at which certificate was issued to student. |
| Grade | DECIMAL(4,2) | NOT NULL, CHECK (GRADE > 0) | N/A | Percentage grade of course for which certificate is issued. |
| ExpiryDate | DATE | NOT NULL | N/A | Date at which certificate will expire. |
| FilePath | VARCHAR(128) | NOT NULL | N/A | File path on server to filled in certificate. |

### **Organization Tables:**

**Organizations:**

Instructors can be part of “Organizations”. Organizations are simply collections of related instructors. Organizations also have homepages in which the collective courses of all their instructors will be listed. Organizations must be registered by admin users. This tables stores all of this information.

CREATE TABLE Organizations(

ID BIGINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

AdminID BIGINT UNSIGNED,

FOREIGN KEY(AdminID) REFERENCES AdminInformation(ID),

RegistrationDate DATE NOT NULL DEFAULT CURRENT\_DATE,

BackgroundImagePath VARCHAR(128)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for organization entity. |
| AdminID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in InstructorInformation. This is ID for admin who created this organization |
| RegistrationDate | DATE | NOT NULL | CURRENT\_DATE | Date at which organization was created. |
| BackgroundImagePath | VARCHAR(128) | N/A | N/A | File path to image on server used as background on organization page. |

**Organization Instructors:**

As stated before, organizations are groups of instructors. This table records this relation.

CREATE TABLE OrganizationInstructors(

ID BIGINT UNSIGNED,

InstructorID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES AdminInformation(ID),

FOREIGN KEY(InstructorID) REFERENCES InstructorInformation(ID),

PRIMARY KEY(ID, InstructorID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in AdminInformation. |
| InstructorID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in Organizations. |

### **Financial Tables:**

**BankCard:**

Not all courses are free on the OLS. Courses must be paid for with Bank Cards (Credit or Debit). Furthermore, an invoice must be generated and sent to the purchaser of the course. This table stores the Bank Card’s information.

CREATE TABLE BankCard(

ID BIGINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

StudentID BIGINT UNSIGNED,

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

ExpirationYear SMALLINT UNSIGNED NOT NULL CHECK (ExpirationYear > 2024),

ExpirationMonth SMALLINT UNSIGNED NOT NULL CHECK (ExpirationMonth > 0 AND ExpirationMonth < 13),

NameOnCard VARCHAR(32) NOT NULL,

Type VARCHAR(16) NOT NULL,

CVV CHAR(3) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for Bank Card. |
| StudentID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in StudentInformation. |
| ExpirationYear | SMALLINT | UNSIGNED, NOT NULL,CHECK (ExpirationYear > 2024), | N/A | Year at which the Bank Card will expire. |
| ExpirationMonth | SMALLINT | UNSIGNED, NOT NULL, CHECK (ExpirationMonth > 0 and ExpirationMonth < 13) | N/A | Month at which the Bank Card will expire. |
| Type | VARCHAR(16) | NOT NULL | N/A | Type of bank card (credit/debit). |
| CVV | CHAR(3) | NOT NULL | N/A | Three digits used for payments. |

**InvoiceInformation:**

As stated above, courses must be paid for bank cards. Furthermore, bank cards must be generated. This table stores bank cards. Invoices will be stored even if their respective courses and student users have been deleted.

CREATE TABLE InvoiceInformation(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

CourseID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

CardID BIGINT UNSIGNED,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

FOREIGN KEY(CardID) REFERENCES BankCard(ID),

PRIMARY KEY(ID),

InvoiceDate DATE DEFAULT CURRENT\_DATE,

TotalCost DECIMAL(7,2) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY | N/A | Internal ID for Invoice. |
| CourseID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation |
| StudentID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in StudentInformation. |
| CardID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in CardInformation |
| invoiceDate | DATE | N/A | CURRENT\_DATE | Date the transaction of the invoice occurred |
| TotalCost | DECIMAL(7,2) | NOT NULL | N/A | Total amount of money involved in invoice |

### **Courses Tables:**

**CourseInformation:**

This table stores essential information about courses. This information is visible to student users who are searching for courses on the OLS.

CREATE TABLE CourseInformation(

ID BIGINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(48) NOT NULL,

Description VARCHAR(1024) NOT NULL,

ThumbnailPath VARCHAR(128) NOT NULL,

Cost DECIMAL(5,2) NOT NULL CHECK (Cost > 0),

CreationDate DATE NOT NULL DEFAULT CURRENT\_DATE

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for course. |
| Name | VARCHAR(48) | NOT NULL | N/A | Name visible for course. |
| Description | VARCHAR(1024) | NOT NULL | N/A | Description for course and its contents. |
| Cost | DECIMAL(5,2) | NOT NULL, CHECK (COST > 0) | N/A | Cost that must be paid before course is accessible. |
| CreationDate | DATE | NOT NULL | CURRENT\_DATE | Date at which this course was created by instructor user. |

**CourseTags:**

Courses can also be recommended to students on the OLS by the OLS itself. Courses have a set of tags. This is used by the OLS to recommend courses and for users to search by courses.

CREATE TABLE CourseTags(

ID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES CourseInformation(ID),

Tag VARCHAR(24),

PRIMARY KEY(ID, Tag)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, FOREIGN KEY, PRIMARY KEY | N/A | Foreign key reference to ID in CourseInformation |
| Tag | VARCHAR(24) | PRIMARY KEY | N/A | Tag for course. |

**CourseInstructors:**

As stated before, courses must be taught by an instructor. Furthermore, they can be taught by multiple instructors. This table records this relation.

CREATE TABLE CourseInstructors(

CourseID BIGINT UNSIGNED,

InstructorID BIGINT UNSIGNED,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(InstructorID) REFERENCES InstructorInformation(ID),

PRIMARY KEY(CourseID, InstructorID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | PRIMARY KEY,FOREIGN KEY | N/A | Foreign key reference to CourseID in CourseInformation. |
| InstructorID | BIGINT | PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to InstructorID in InstructorInformation. |

**CourseCertificate:**

Upon course completion, students will be awarded a certificate. This certificate will contain information such as their grade, date of issue, etc. This table will simply store the template’s file path and the certificates name.

CREATE TABLE CourseCertificate(

ID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

PRIMARY KEY(ID, CourseID),

TemplateFilePath VARCHAR(128) NOT NULL,

Name VARCHAR(48) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| CourseID | BIGINT | PRIMARY KEY, FOREIGN KEY | N/A | Foreign key |

**CourseReviews:**

During the course, students will be able to freely write a review and leave a rating for the course in which they are currently enrolled. This table stores this behaviour.

CREATE TABLE CourseReviews(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

StudentID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

 KEY(StudentID) REFERENCES StudentInformation(ID),

PRIMARY KEY(ID),

ReviewDate DATE NOT NULL DEFAULT CURRENT\_DATE,

Title VARCHAR(48) NOT NULL,

CONTENT VARCHAR(2048) NOT NULL,

Rating TINYINT UNSIGNED NOT NULL CHECK (RATING < 11)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID used for course reviews. |
| StudentID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in StudentInformation. |
| CourseID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| Title | VARCHAR(48) | NOT NULL | N/A | Publicly visible title for the review |
| Content | VARCHAR(2048) | NOT NULL | N/A | Publicly visible text for the review. |
| Rating | TINYINT | UNSIGNED, NOT NULL,  CHECK (RATING < 11) | N/A | Publicly visible numerical rating for the course out of 10. |

### **Modules and Questions:**

**Module Information:**

While creating courses, instructors must specify a set of modules. These modules will contain questions of various types: True/Fase, MCQ, short answer, and assignments (file upload). This table will simply store the required attributes for a module.

CREATE TABLE ModuleInformation(

ID BIGINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

CourseID BIGINT UNSIGNED,

Title VARCHAR(48) NOT NULL,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for module. |
| CourseID | BIGINT | UNSIGNED, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation |
| Title | VARCHAR(48) | NOT NULL | N/A | Title visible to student completing module. |

**QuestionTrueFalse:**

The first type of question that a student can solve is a True/False question. These questions consist of a prompt, and a correct answer.

CREATE TABLE QuestionTrueFalse(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

PRIMARY KEY(ID, ModuleID, CourseID),

Prompt VARCHAR(128) NOT NULL,

CorrectAnswer BOOLEAN NOT NULL,

MaxGrade TINYINT UNSIGNED NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for True/False questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation |
| Prompt | VARCHAR(128) | NOT NULL | N/A | Prompt visible to student solving question. |
| CorrectAnswer | BOOLEAN | NOT NULL | N/A | Correct answer to question. |
| MaxGrade | TINYINT | NOT NULL | N/A | Grade given upon successful completion of question |

**QuestionReading:**

The second type of question a student can solve is not a question. Instead, it is educational text that the student must read.

CREATE TABLE QuestionReading(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

Title VARCHAR(64) NOT NULL,

Content VARCHAR(4096),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

PRIMARY KEY(ID, ModuleID, CourseID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for reading questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| Title | VARCHAR(64) | NOT NULL | N/A | Title shown to student completing the reading question. |
| Content | VARCHAR(4096) | NOT NULL | N/A | Content to be shown to student solving the reading question. |

**QuestionAssignment:**

The third type of question a student can complete is called the assignment. This is solved by uploading a file which the instructor will manually review and grade.

CREATE TABLE QuestionAssignment(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

Prompt VARCHAR(256) NOT NULL,

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

PRIMARY KEY(ID, ModuleID, CourseID),

MaxGrade TINYINT UNSIGNED NOT NULL CHECK (MaxGrade > 0)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for assignment questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY,  FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation |
| Prompt | VARCHAR(256) | NOT NULL | N/A | Prompt visible to student solving the assignment question. |
| MaxGrade | TINYINT | UNSIGNED, NOT NULL, CHECK (MaxGrade > 0) | N/A | Maximum possible grade upon successful assignment completion. |

**QuestionShortAnswer:**

The fourth type of question the student can solve. In this question type, the student types a short essay to answer the prompt given to them. The instructor will then manually grade it.

CREATE TABLE QuestionShortAnswer(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

PRIMARY KEY(ID, ModuleID, CourseID),

Prompt VARCHAR(256) NOT NULL,

MaxGrade TINYINT UNSIGNED NOT NULL CHECK (MaxGrade > 0)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | AUTO\_INCREMENT | Internal ID for short answer questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| Prompt | VARCHAR(256) | NOT NULL | N/A | Prompt visible to student solving the short answer question. |
| MaxGrade | TINYINT | UNSIGNED, NOT NULL, CHECK (MaxGrade > 0) | N/A | Maximum possible grade upon successful assignment completion. |

**QuestionMultipleChoice:**

The fifth type of question the student can solve is the multiple-choice question. This question has a prompt and four possible answers. Only one answer is correct. Correctly solving this question will give the student the maximum possible mark.

CREATE TABLE QuestionMultipleChoice(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

CourseID BIGINT UNSIGNED,

ModuleID BIGINT UNSIGNED,

MaxGrade TINYINT UNSIGNED NOT NULL CHECK (MaxGrade > 0),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

PRIMARY KEY(ID, ModuleID, CourseID),

Prompt VARCHAR(256) NOT NULL,

CorrectAnswer TINYINT UNSIGNED NOT NULL CHECK (CorrectAnswer > 0 AND CorrectAnswer < 5),

QuestionOne VARCHAR(128) NOT NULL,

QuestionTwo VARCHAR(128) NOT NULL,

QuestionThree VARCHAR(128) NOT NULL,

QuestionFour VARCHAR(128) NOT NULL);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for multiple choice questions. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| Prompt | VARCHAR(256) | NOT NULL | N/A | Prompt shown to student answering the multiple-choice question. |
| QuestionOne | VARCHAR(128) | NOT NULL | N/A | Answer shown to first question. |
| QuestionTwo | VARCHAR(128) | NOT NULL | N/A | Answer shown to second question. |
| QuestionThree | VARCHAR(128) | NOT NULL | N/A | Answer shown to third question. |
| QuestionFour | VARCHAR(128) | NOT NULL | N/A | Answer shown to fourth question. |

**QuestionVideo:**

The fifth type of question the student can solve is also not a question. Rather, it is an educational video that the student must watch.

CREATE TABLE QuestionVideo(

ID BIGINT UNSIGNED AUTO\_INCREMENT,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

PRIMARY KEY(ID, ModuleID, CourseID),

Title VARCHAR(64) NOT NULL,

FilePath VARCHAR(128) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT | AUTO\_INCREMENT | Internal ID for video questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| Title | VARCHAR(64) | NOT NULL | N/A | Title shown to student solving question. |
| Filepath | VARCHAR(128) | NOT NULL | N/A | File path on server to video displayed to student. |

### **SolvedQuestions:**

**SolvedQuestionsTrueFalse:**

This table stores the student’s answer and grade in a True/False question, along with date on which it was solved.

CREATE TABLE SolvedQuestionTrueFalse(

ID BIGINT UNSIGNED,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES QuestionTrueFalse(ID),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

PRIMARY KEY (ID, ModuleID, CourseID, StudentID),

DateSolved DATE DEFAULT CURRENT\_DATE NOT NULL,

Grade TINYINT UNSIGNED NOT NULL,

StudentAnswer BOOLEAN NOT NULL,

Grade DECIMAL(4,2) NOT NULL CHECK (Grade > 0)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Internal ID for video questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key refrerence to ID in StudentInformation. |
| DateSolved | DATE | NOT NULL | N/A | Date when student solved the question. |
| Grade | DECIMAL(4,2) | NOT NULL | N/A | Grade received when student solved question. |

**SolvedQuestionMultipleChoice:**

This table is the same as the one above but for multiple choice questions.

CREATE TABLE SolvedQuestionMultipleChoice(

ID BIGINT UNSIGNED,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES QuestionMultipleChoice(ID),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

PRIMARY KEY (ID, ModuleID, CourseID, StudentID),

DateSolved DATE DEFAULT CURRENT\_DATE NOT NULL,

StudentAnswer TINYINT UNSIGNED NOT NULL CHECK (StudentAnswer > 0 AND StudentAnswer < 5),

Grade DECIMAL(4,2) NOT NULL CHECK (Grade > 0)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Internal ID for video questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key refrerence to ID in StudentInformation. |
| StudentAnswer | TINYINT | UNSIGNED, NOT NULL, CHECK (StudentAnswer > 0 AND StudentAnswer < 5) | N/A | Answer student chose while solving the question. |
| DateSolved | DATE | NOT NULL | CURRENT\_DATE | Date in which student solved the multiple choice question. |
| Grade | DECIMAL(4,2) | NOT NULL, CHECK (Grade > 0) | N/A | Grade received upon student solving multiple choice question. |

**SolvedQuestionReading:**

This table is the same as the ones above it. However, it instead uses a BOOLEAN instead of a DECIMAL for storing the “student’s grade”. There are no grades for reading. Thus, it is replaced with a “IsDone” attribute.

CREATE TABLE SolvedQuestionReading(

ID BIGINT UNSIGNED,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

FOREIGN KEY(ID) REFERENCES QuestionReading(ID),

PRIMARY KEY (ID, ModuleID, CourseID, StudentID),

DateSolved DATE DEFAULT CURRENT\_DATE NOT NULL,

IsDone BOOLEAN NOT NULL DEFAULT TRUE);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Internal ID for video questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key refrerence to ID in StudentInformation. |
| DateSolved | DATE | NOT NULL | CURRENT\_DATE | Date in which student solved the reading question. |
| IsDone | BOOLEAN | NOT NULL | TRUE | Indicates whether the student read the reading question or not. |

**SolvedAssignment:**

This table is the same as the ones above it. However, it instead uses a VARCHAR to store the path to the student’s uploaded file.

CREATE TABLE SolvedAssignment(

ID BIGINT UNSIGNED,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES QuestionAssignment(ID),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

PRIMARY KEY (ID, ModuleID, CourseID, StudentID),

StudentAnswer VARCHAR(128) NOT NULL,

Grade DECIMAL(4,2) NOT NULL CHECK (Grade > 0),

SolvedDate DATE NOT NULL DEFAULT CURRENT\_DATE

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY | N/A | Internal ID for video questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key refrerence to ID in StudentInformation. |
| DateSolved | DATE | NOT NULL | CURRENT\_DATE | Date in which student solved the assignment question. |
| Grade | DECIMAL(4,2) | NOT NULL, CHECK (Grade > 0) | N/A | Grade received upon student solving multiple choice question. |
| StudentAnswer | VARCHAR(128) | NOT NULL | N/A | File path to uploaded file created by student solving the question. |

**SolvedShortAnswerQuestion:**

This table is similar to the ones before it. However, it stores the student’s answer in a VARCHAR. Furthermore, the grade is determined by instructor and not automatic.

CREATE TABLE SolvedShortAnswerQuestion(

ID BIGINT UNSIGNED,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES QuestionShortAnswer(ID),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

PRIMARY KEY (ID, ModuleID, CourseID, StudentID),

SolvedDate DATE NOT NULL DEFAULT CURRENT\_DATE,

Grade DECIMAL(4,2) NOT NULL CHECK (Grade > 0),

StudentAnswer VARCHAR(2048)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY | N/A | Internal ID for video questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key refrerence to ID in StudentInformation. |
| DateSolved | DATE | NOT NULL | CURRENT\_DATE | Date in which student solved the short answer question. |
| Grade | DECIMAL(4,2) | NOT NULL, CHECK (Grade > 0) | N/A | Grade received upon student solving multiple choice question. |
| StudentAnswer | VARCHAR(2048) | NOT NULL | N/A | Student’s answer upon solving the question. |

**SolvedQuestionVideo:**

This table is identical to the one used for reading questions. This is because no grades are assigned to videos.

CREATE TABLE SolvedQuestionVideo(

ID BIGINT UNSIGNED,

ModuleID BIGINT UNSIGNED,

CourseID BIGINT UNSIGNED,

StudentID BIGINT UNSIGNED,

FOREIGN KEY(ID) REFERENCES QuestionVideo(ID),

FOREIGN KEY(ModuleID) REFERENCES ModuleInformation(ID),

FOREIGN KEY(CourseID) REFERENCES CourseInformation(ID),

FOREIGN KEY(StudentID) REFERENCES StudentInformation(ID),

PRIMARY KEY(ID, ModuleID, CourseID, StudentID),

DateSolved DATE NOT NULL DEFAULT CURRENT\_DATE,

IsDone BOOLEAN NOT NULL DEFAULT TRUE

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY | N/A | Internal ID for video questions. |
| ModuleID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in ModuleInformation. |
| CourseID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key reference to ID in CourseInformation. |
| StudentID | BIGINT | UNSIGNED, PRIMARY KEY, FOREIGN KEY | N/A | Foreign key refrerence to ID in StudentInformation. |
| DateSolved | DATE | NOT NULL | CURRENT\_DATE | Date in which student solved the video question. |
| IsDone | BOOLEAN | NOT NULL | TRUE | Indicates whether the student read the video question or not. |

### **Miscellaneous:**

**User Types:**

This table stores a unique integer ID that maps to a user type in the OLS. This is used for normalization and to save on database size.

CREATE TABLE UserTypes(

ID TINYINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

Type VARCHAR(32) NOT NULL

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | TINYINT | UNSIGNED, PRIMARY KEY, AUTO INCREMENT | AUTO\_INCREMENT | Internal ID used for user types. |
| Type | VARCHAR(32) | NOT NULL | N/A | String representing the user type. |

**UserEmails:**

This table maps unique email addresses to non-unique and internal user IDs along with their user type. This allows users to sign in with their email. This also allows us to safely use AUTO\_INCREMENT on all primary keys in the database.

CREATE TABLE UserEmails(

ID BIGINT UNSIGNED,

Email VARCHAR(32) UNIQUE NOT NULL,

UserType TINYINT UNSIGNED NOT NULLN,

PRIMARY KEY(ID, Email),

FOREIGN KEY(UserType) REFERENCES UserTypes(ID)

);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Datatype** | **Constraints** | **Default** | **Description** |
| ID | BIGINT | UNSIGNED, PRIMARY KEY | N/A | ID of the user used internally in the database. |
| Email | VARCHAR(32) | UNIQUE, NOT NULL | N/A | Email of the user, usually used for signed in purposes. |
| UserType | TINYINT | UNSIGNED, NOT NULL | N/A | Numeric ID representing the type of the user. |