Web-based
Attendance Management System

Integrantes:

- Juan Sebastian Velasquez Acevedo (1744936-3743)
- Edwin Leonardo Cuaran Cuaran (1910252-3743)
- Juan José Angel Perdomo (1825116-3743)

Índice

Sprint No. 1: Design a conceptual diagram using ER model

Challenge No. 1 - Identify the entities.

Entities:

1. User (Id_User, Password, Email, First Name, Last Name, Modified, Last Ip, Last Login, Avatar, Description, Joined, user_type)
    a. Staff
    b. Student
    c. Admin
2. Campus (Id_Campus, Location, Name)
3. Test (Id_Test, Name, Status, Description, Release Date)
4. Course (Id_Couse, Status, Number of credits, Name)
5. Question (Id_Question, Description, Type)
6. Option List (Id_OptionList, Name)
7. Option (Id_Option)

Relationships:

1. Manages // Entities --> Admin - 1 : Campus - N
2. Enrolls (Date) // Entities --> Student - N : Course - M , Staff - N : Course - M
3. Attends (Date) // Entities --> Student - N : Test - M, Student - N : Course - M
4. Creates - Recursive Relationship // Entities --> Admin - 1 - N
5. Creates // Entities --> Staff - 1 : Test - N
6. Creates // Entities --> Staff - 1 : Student - N
7. Assigned To // Entities --> Test - N : Course - 1
8. Contains // Entities --> Test - M : Question - N
9. Contains // Entities --> Question - N : Option List - 1
10. Contains // Entities --> Option List - 1 : Option - N
11. Teaches // Entities --> Staff - 1 : Course : N
12. Answers (Id_Answer, Date, Option selected or Text) // Entities --> Student - N : Question - M

Where student answers will be stored?

- En la relación Answers que tiene atributos propios, y que luego en el modelo relacional se puede representar como una relación-tabla propia.

Where tests and questions will be stored ?

- Cada una de ellas tiene su entidad correspondiente con sus respectivos atributos, que luego se convertirán a tablas en el modelo relacional. Marcando siempre que una Question está contenida en un Test. Es decir, un cuestionario puede tener una o más preguntas.

Do the questions have answer options?

- Sí, cada pregunta tiene una lista de opciones de las cuales puede escoger una respuesta y una opción de respuesta puede estar en varias preguntas. Esto sucede

sí y solo sí el tipo de pregunta es cerrada. Luego de que el estudiante escoge una opción de la lista de opciones, ésta quedará almacenada en answer.

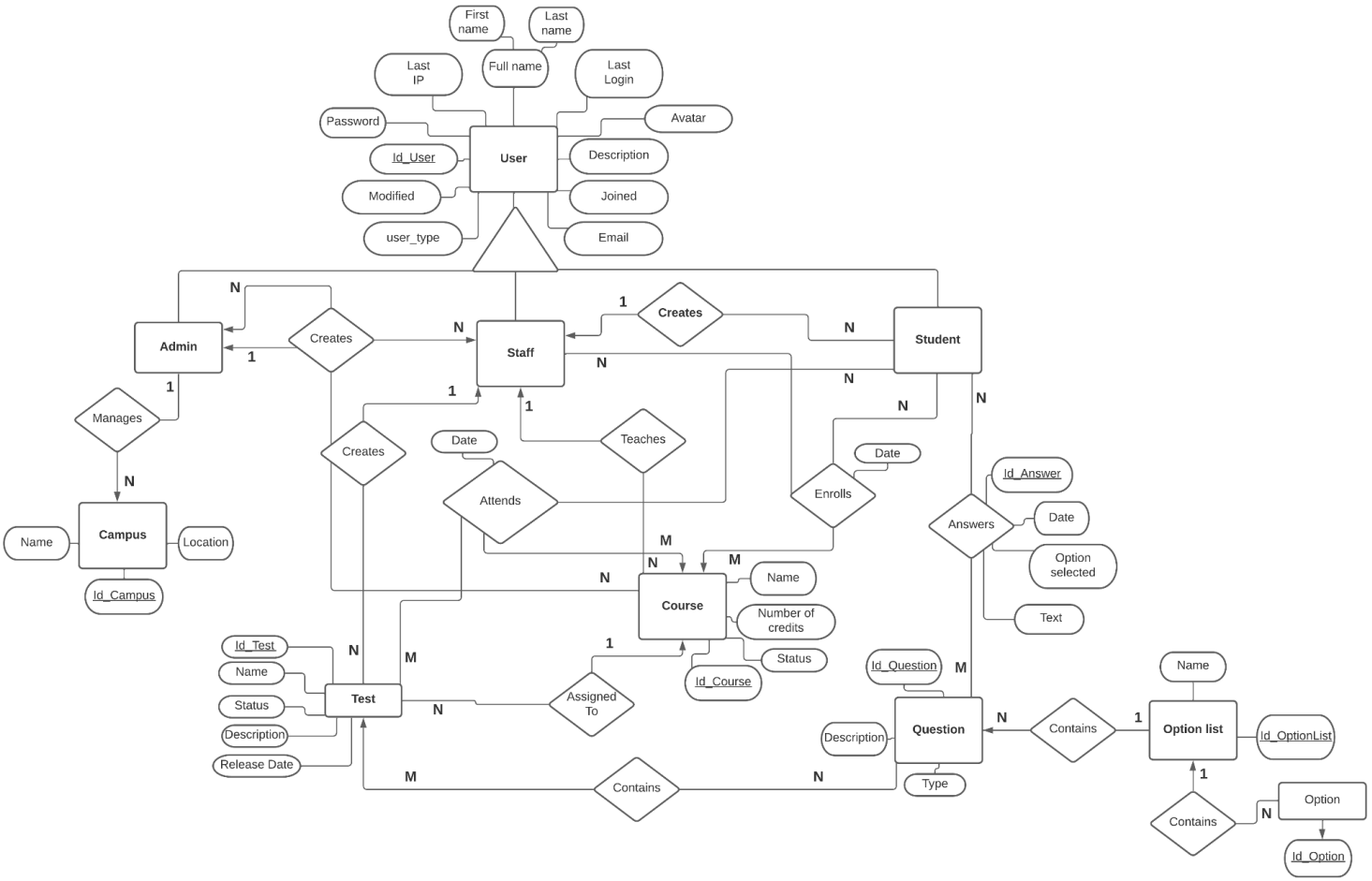
Challenge No. 2 - Semantic relationships between entities.

Leer de izquierda a derecha, tomando como referencia siempre a la entidad de la fila luego leyendo la acción y luego revisar hacia cuál entidad fue aplicada. Por ejemplo en la fila 1 - columna 3 están las entidades Admin y Staff, y se lee: Admin Creates Staff. De la fila hacía la columna, para mantener la lógica semántica.

También como anotación, se sabe que el staff crea no solo el Test, sino también Question y Option List. Pero según nuestro modelo, Option List está contenida dentro de Question y Question está contenido dentro de Test, por eso está implícita esta relación.

	Admin	**Staff**	**Students**	**Course**	**Campus**	**Test**	**Question**	**Option List**	**Option**
Admin	Creates	Creates		Creates	Manages				
Staff	Created by		Creates - Enrolls	Teaches		Creates			
Student		Enrolled by		Attends			Answers		
Course	Created by	Taught by	Attended by			Has Assigned			
Campus	Managed by								
Test		Created by		Assigned To			Contains		
Question			Answered by			Contained in		Contains	
Option List							Contained in		Contains
Option								Contained in	

Challenge No. 3 - Entities, relationships and the cardinality of the relationships. Model ER



Link Lucidchart Página 1 - ER:
https://lucid.app/lucidchart/e6699e3f-089f-4560-9490-782b73e6f67a/edit?invitationId=inv_97576286-047e-4529-81a5-1739d1d9f25e

Notation Used:
https://drive.google.com/file/d/1oW8GYqDvLUd-gMz-LxrGoPsqs6umTMLz/view?usp=sharing

Link image - png model ER:
https://drive.google.com/file/d/1W8xCduEVpT1dvszjiBF1ztAH7ayWo5Ek/view?usp=sharing

Cuestionamientos Sprint 1

En el proceso de realización de este Sprint surgieron algunas dudas sobre la implementación de algunas relaciones, porque se cree que algunos enunciados pueden dar pie a ambigüedades. Para esto se trató de contactar al profesor por medio de correo electrónico para recibir algún tipo de retroalimentación, pero como desafortunadamente no se recibió una respuesta, se decide seguir con el proceso de desarrollo de los sprints. Por esto se deja aquí pactado los cuestionamientos y las posibles respuestas o soluciones que dimos como grupo según lo que se entiende del enunciado del proyecto.

1. ¿Es necesario la entidad Campus? No la vemos necesaria, tampoco la mencionan en el enunciado. Puesto que la asistencia se marca del estudiante al curso, y nada más. → **La vamos a dejar solo para que un administrador lo gestione**

2. ¿Qué es option list y option? Debería de ir relacionada la entidad Option-List a Question o a Test. ¿Y qué función cumple en la aplicación? Está relacionada con la tercera pregunta que se plantea en el Challenge 1 - 'Do the questions have answer options?' - ¿Qué quiere decir con answer options, las preguntas cerradas? → **Cada pregunta contiene un option list, que se llena solo si la pregunta es cerrada**

3. Si Answer, tiene como atributo 'option selected', entonces ¿Estará bien que relacionamos Student con Question por medio de la relación 'Answers'? O será mejor que relacionamos a Student por medio de 'Answers' con la entidad Option o con Option List → **Está bien que relacionamos student con question porque si no pues no podríamos responder preguntas abiertas**

4. ¿Es necesaria una relación o una entidad 'Attends - Asiste'? O con la entidad Test ya quedaría cubierto lo de Asistencia. Pues según entendemos, habrá un test dedicado (validation test) en cada curso que cumplirá el papel de marcar la asistencia. Si es necesario tener Attendance en alguna parte, es preferible tener la relación Attends como está en este momento, o quedaría mejor crear una entidad que se relacione con Student y con Test. → **Asumimos que todos los tests de la aplicación son para marcar asistencia, y únicamente para eso. Así attendance sería una tabla que se llena cada vez que un estudiante responde un test. Esta tabla tendría llave foránea a curso, a student y a test, posiblemente.**

5. Hasta donde entendemos, la función principal de la app es marcar la asistencia de los estudiantes a cursos manejados por un profesor. Pero según la demostración que se hizo en clase ¿Pueden haber tests de cualquier tipo verdad? No solo el de asistencia. → **Asumimos que todos los tests de la aplicación son para solamente marcar asistencia.**

6. ¿Deberíamos de tener una distinción para el código de estudiante (Student_code), o es mejor que como está en el diagrama. Herede el Id_User ya que Student es una entidad hijo de User? Tenemos esta duda porque para acceder a la plataforma el Admin y el Staff entrarán con su Cédula pero el estudiante con su código de estudiante ¿Hay que hacer alguna distinción de esto en el modelado? O es suficiente con que las tres entidades sean hijas de User y ya se maneja

internamente si es código de estudiante o Cédula. → Probablemente sea necesario un id único para staff, student y admin. Independientemente que as tres relaciones sean hijas de user. Puesto que internamente hay procesos en los que se debe de diferenciar las tres entidades, como por ejemplo qué user puede crear qué en la aplicación**.**

Sprint No. 2: Conceptual diagram using Relational model

Challenge No. 1 - Define primary keys and add other attributes

Llaves Primarias - PK

1. User (**Id_User (PK)**, Password, Email, First Name, Last Name, Modified, Last Ip, Last Login, Avatar, Description, Joined, user_type)
2. Staff (**Id_Staff(PK)**) -- Heredada User
3. Student(**Id_Student (PK)**) -- Heredada User
4. Admin(**Id_Admin (PK))** -- Heredada User
5. Campus (**Id_Campus (PK)**, Location, Name)
6. Test (**Id_Test (PK)**, Name, Status, Description, Release_Date)
7. Attendance(**{Id_Test, Id_Student} (PKC)**, Date)
8. Course (**Id_Course (PK)**, Status, Number_Credits, Name)
9. Enrolls(Date)
10. Question (**Id_Question(PK)**, Description, Type **)**)
11. Question_Test ({**Id_Question , Id_Test} (PKC))**
12. Option List (**Id_OptionList (PK)**, Name)
13. Option (**Id_Option, (PK)**)
14. Answer (**Id_Answer (PK)**, Date)

Si no existiese cada relación heredada de usuario, diríamos que por ejemplo, cada test, course, admin, staff o student pueden ser creados por cualquier usuario, y esto no es verdad. Hay restricciones de qué quién puede crear qué.

Challenge No. 2 - Identify all entity characteristics relevant to the problem domain.

1. **User**: la entidad user en este modelo es una generalización, es decir, es padre para las entidades **Admin, Staff** y **Student,** ya que éstas últimas tres pueden heredar atributos de la entidad User.

2. **Staff**: una entidad con llave primaria (Id_Staff), potencialmente se piensa que tendrá en total 4 relaciones con, Admin, Student, Course y Test, con las cuales establece un vínculo que permite dar solución a los requerimientos del proyecto, tales como creación de estudiantes,de cursos, matricular estudiantes a sus cursos y crear tests para que los estudiantes marquen asistencia. Además tiene un id propio que lo identifica del resto, esto sin olvidar que es hijo de user.

3. **Student**: Es una de las entidades más relevantes del modelo ya que su comportamiento es el de un usuario final(Persona) que interactúa con el sistema mediante una interfaz gráfica para poder marcar la asistencia. Además tiene un id propio que lo identifica del resto, esto sin olvidar que es hijo de user.

4. **Admin**: Es una entidad que tiene una característica especial, viniendo desde el modelo ER con su relación recursiva, esto es porque una de las acciones de Admin es crear otro Admin para las sedes o campus. Además tiene un id propio que lo identifica del resto, esto sin olvidar que es hijo de user.

5. **Campus**: Esta entidad no es tan relevante para la implementación principal del proyecto, aunque es importante saber por cuál administrador(**entidad Admin**) es manejado cada campus.

6. **Test**: Entidad mediante la cual se puede identificar a un estudiante y su asistencia a un curso. Los atributos de esta entidad (id_test, status, description, release_date) son creados en el sistema por el personal(**Staff**), además se piensa que potencialmente cada ‘Test’ pertenece a un ‘Course’ y debería de tener un vínculo con ‘Question’.

7. **Attendance**: En el modelo ER es una relación para las entidades Student y Test. En el modelo relacional se convierte en una tabla ya que posee un atributo especial que es la fecha y hora en que se hace el registro de la asistencia, además se piensa que esta tabla tiene que estar vinculada con un ‘test’ y un ‘student’, para poder saber qué estudiante marcó asistencia a través de qué test. Se considera que esta es la tabla-relación más importante de todo el proyecto puesto que será la encargada de almacenar los registros de asistencia que es objetivo de la aplicación.

8. **Course**: Esta entidad es una de las que más tiene vínculos con otras entidades, se piensa que debe de relacionarse con Staff, Student, Test y Admin. Es relevante al momento de registrar una asistencia ya que ésta entidad tiene un vínculo directo con el Estudiante que es quien marca su asistencia al curso. Los atributos que creemos necesario por ahora son id_course, status, number_credits, y name.

9. **Enrolls**: Por ahora se cree necesario que esta entidad tenga un atributo Date, para saber en qué fecha se matricula a un estudiante. Se piensa que esta entidad no necesariamente tiene que tener un id, si no que podría tener una llave primaria compuesta, ya que se debe de relacionar de alguna forma con Staff y Student, para permitir agregar un estudiante a un curso, tarea realizada única y exclusivamente por el Staff.

10. **Question**: En ésta entidad se registran las preguntas que el estudiante debe contestar en un test conformado por varias preguntas, que pueden ser abiertas o cerradas, en principio por medio del atributo 'type' queremos diferenciar si con 'close' o 'open' y de acuerdo a esto manejarlas de forma distinta. También se cree necesario que tenga una descripción, que sería el enunciado de la pregunta.

11. **Question_Test:** Se cree necesario crear una tabla que relacione las preguntas y los tests, puesto que en el modelo ER se visualiza una relación de muchos a muchos. Para poder representar esto en el modelo relacional se piensa que lo más adecuado es crear esta nueva tabla que tenga una llave primaria compuesta por el id del test y el id de la pregunta. De esta forma una pregunta puede estar en muchos tests y un test puede tener muchas preguntas.

12. **Option list**: En esta entidad habrá una lista de opciones que se le asigna a cada pregunta que se cree en un cuestionario (entidad Test), cada lista tiene múltiples opciones de respuesta de las cuales 1 o más pueden representar la respuesta correcta. En principio se piensa que cada lista debería de poderse diferenciar a través de un id, y debería de tener un nombre. Hay que recordar que una pregunta puede tener una y solo una lista de opciones si esta pregunta es de tipo cerrada, si es abierta no puede tener una lista de opciones asociada.

13. **Option**: En esta entidad estarán las opciones de respuesta que son indivisibles, estas opciones hacen parte y constituyen una Option List para una pregunta cerrada. En principio se piensa que esta entidad debería de tener una llave primaria solamente, pero se sabe que faltarían más atributos para representar su comportamiento.

14. **Answer**: Esta entidad es importante para identificar la opción que ha marcado un estudiante como posible respuesta si la pregunta fue de tipo cerrada. En caso de ser una respuesta a una pregunta abierta, también se guardará el texto correspondiente a lo que digitó el estudiante por teclado. En principio esta relación debería de tener un id que la identifique y una fecha que marque también el tiempo en el que se responde cada pregunta.

Challenge No. 3 - Revise your ER diagram and transform to Relational

Asignación de Llaves primarias y llaves foráneas.

Llaves Primarias - PK
Llaves Primarias Compuestas- PKC
Llaves Foráneas - FK

1. User (**Id_User (PK)**, Password, Email, First Name, Last Name, Modified, Last Ip, Last Login, Avatar, Description, Joined, user_type)

2. Staff (**Id_Staff(PK), Id_User* (FK), created_ByAdmin* (FK -> Id_Admin)**)

3. Student(**Id_Student (PK), Id_User* (FK), created_ByStaff* (FK -> Id_Staff)**)

4. Admin(**Id_Admin (PK), Id_User* (FK), created_ByAdmin*(FK→Id_Admin))**

5. Campus (**Id_Campus (PK)**, Location, Name, **Id_Admin* (FK)**)

6. Test (**Id_Test (PK)**, Name, Status, Description, Release_Date, **created_ByStaff*** (**FK -> Id_Staff), Id_Course* (FK)**)

7. Attendance(**{Id_Test* (FK) , Id_Course* (FK), Id_Student* (FK)} (PKC)**, Date)

8. Course (**Id_Couse (PK)**, Status, Number_Credits, Name, **Id_staff* (FK), created_ByAdmin* (FK->Id_admin)**)

9. Enrolls(**{Id_Course* (PK), Id_Student* (FK), Id_Staff* (FK)} (PKC)**, Date)

10. Se decide que 'Question' tenga una llave foránea a 'Id_OptionList', puesto que cada pregunta, si es de tipo cerrada, debería tener una y solo una lista de opciones asociada. Esa lista de opciones puede ser de una cantidad cualquiera y no se pueden repetir.

    Question (**Id_Question(PK)**, Description, Type , **Id_OptionList* (FK) (Puede ser NULL)**)

11. Question_Test (**{ Id_Question (FK), Id_Test(FK) } PKC**)

12. Option List (**Id_OptionList (PK)**, Name)

13. Se decide agregar los atributos literal, description y correct a la relación. El primero nos ayuda a determinar que literal está asociado con la pregunta cerrada ('a', 'b', 'c' …), el segundo a guardar el enunciado de cada opción y el tercero es un booleano que nos permite saber si es una opción correcta entre la lista de opciones, una lista de opciones asociada a una pregunta puede tener una o más opciones correctas. Una opción está existe sí y sólo sí está asociada a una pregunta de tipo 'cerrada'.

Option (**Id_Option, literal, Id_question* (FK), description, correct)**

14. Answer (**Id_Answer (PK)**, Date, **OptionSelected*(FK)** Puede ser NULL, Text - Puede ser NULL, **Id_Student*(FK) , Id_question* (FK)**)

Normalización:

En este paso se encuentran algunas llaves redundantes y dependencias transitivas que violan la 3ra forma normal y se decide cambiar un poco esas relaciones.

1. User (**Id_User (PK)**, Password, Email, First Name, Last Name, Modified, Last Ip, Last Login, Avatar, Description, Joined, user_type)

2. Staff (**Id_Staff(PK), Id_User* (FK), created_ByAdmin* (FK -> Id_Admin)**)

3. Student(**Id_Student (PK), Id_User* (FK), created_ByStaff* (FK -> Id_Staff)**)

4. Admin(**Id_Admin (PK), Id_User* (FK), created_ByAdmin*(FK→Id_Admin))**

5. Campus (**Id_Campus (PK)**, Location, Name, **Id_Admin* (FK)**)

6. Test (**Id_Test (PK)**, Name, Status, Description, Release_Date**, Id_Course* (FK)**)

   Se decide eliminar la llave foránea created_ByStaff* (FK -> Id_Staff), puesto que tenemos en la relación 'Curso' una dependencia funcional que indica que Id_Course → Id_Staff. Es decir, en nuestro modelo se asume que cada curso solo puede ser enseñado por un profesor, por lo tanto es lógico que el test sea creado por el profesor que enseña este curso y por nadie más. En otras palabras, un profesor sólo puede crear Tests en los cursos que esté enseñando.

7. Attendance(**{Id_Test* (FK) , Id_Student* (FK)} (PKC), Date**} **(PKC)**)

   Se decide eliminar la llave foránea a Id_Course* (FK), puesto que tenemos en la relación 'Test' una DF que nos indica que Id_test → Id_course. Es decir, en nuestro modelo se asume que si un estudiante marca asistencia mediante un test, este test pertenece ya a un curso. Ya que todo test pertenece a un solo curso y un curso puede tener muchos test.

   Además se decide agregar 'Date' a la llave compuesta, para que permita que un estudiante pueda responder el mismo Test sin problema, pero solo una vez por día.

8. Course (**Id_Couse (PK)**, Status, Number_Credits, Name, **Id_staff* (FK), created_ByAdmin* (FK->Id_admin)**)

9. Enrolls(**{Id_Course* (FK), Id_Student* (FK)} (PKC)**, Date)

Se decide eliminar de esta relación la llave foránea a Id_staff porque ya con Id_course sabemos el Id_staff (DF : Id_course → Id_Staff). Es decir, en nuestro modelo tenemos que un curso puede ser enseñado sólo por un profesor, por lo tanto es lógico que un estudiante solo pueda ser matriculado a un curso por el mismo profesor que enseña este curso. Un profesor solo puede matricular estudiantes en los cursos que enseña.

10. Question (**Id_Question (PK)**, Description, Type , **Id_OptionList* (FK) (Puede ser NULL)**)

11. Question_Test (**{ Id_Question* (FK), Id_Test*(FK) } PKC**)

12. Option List (**Id_OptionList (PK)**, Name)

13. Option (**Id_Option (PK), literal, Id_question* (FK), description, correct)**

14. Answer_Close (**Id_Answer_Close (PK)**, Date, **OptionSelected*(FK)**, **Id_Student*(FK)**)

15. Answer_Open (**Id_Answer_Open (PK)**, Date, Text, **Id_Student*(FK) , Id_question* (FK)**)

    Se decide separar la relación 'Answer' = Answer (Id_Answer (PK), Date, OptionSelected*(FK) Puede ser NULL, Text - Puede ser NULL, Id_Student*(FK) , Id_question* (FK)), en dos relaciones diferentes. Una relación que guarde las respuestas a preguntas cerradas y otra relación que guarde las respuestas a preguntas abiertas.

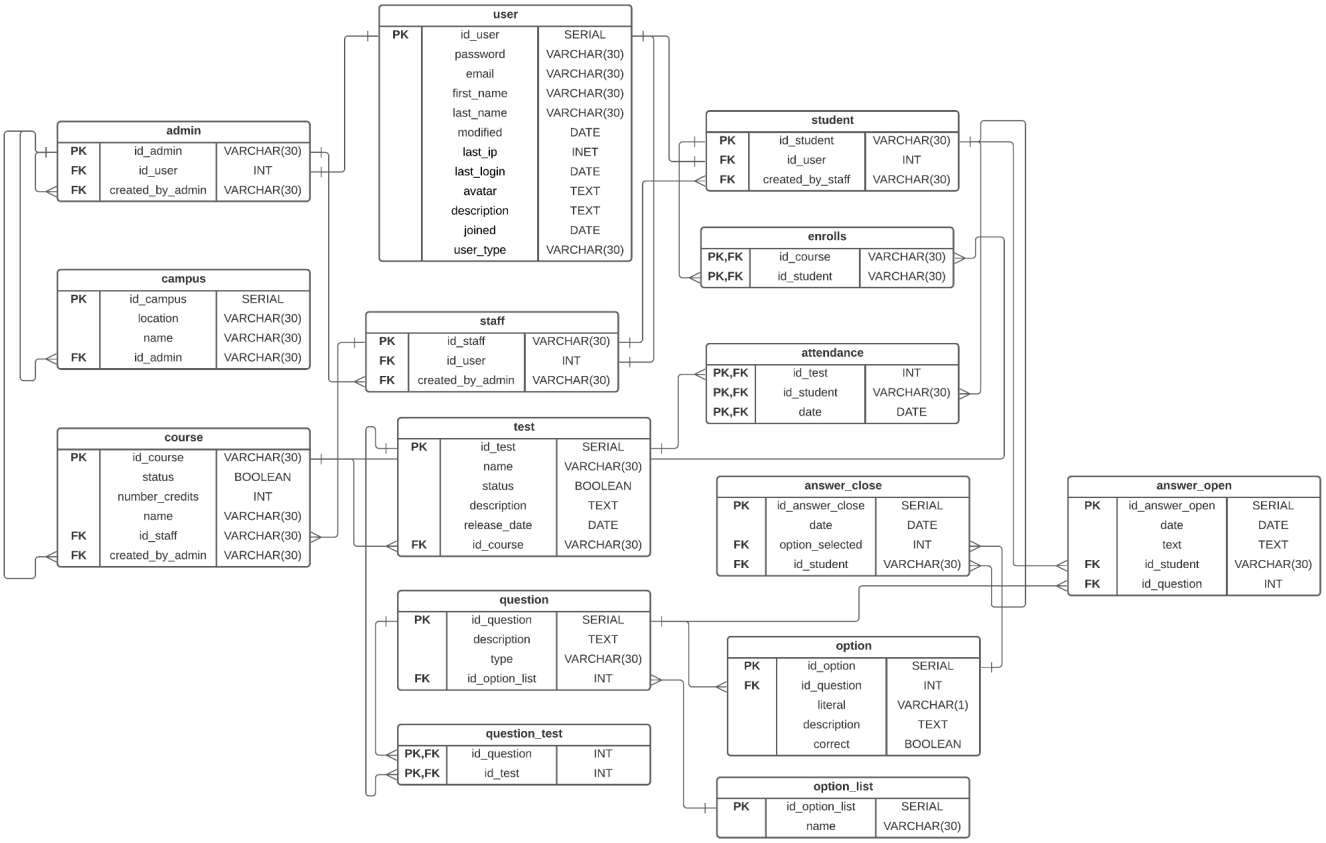
    Esto se hace primero para evitar que las llaves foráneas tomen un valor NULL, y segundo porque existía una dependencia transitiva, teníamos que Id_Answer(PK) → OptionSelected(FK) → Id_question (FK). De la manera que lo separamos, no hay dependencias transitivas y el Id_question sólo es necesario para las respuestas de preguntas abiertas.

Por qué está Normalizado:
1nf: El modelo está en 1nf, ya que todas las columnas de las tablas son datos atómicos, datos no divisibles.
2nf y 3nf: Para que se cumpla la segunda forma normal y la tercera forma normal, todos los datos deben depender funcionalmente de una llave primaria, y se deben eliminar las dependencias transitivas.

Tabla user: La tabla cuenta con atributos como nombre, contraseña, email, última vez conectado y el tipo de usuario; cada uno de estos atributos dependen funcionalmente de la llave primaria id_user, cada uno de estos atributos no existiría sin la llave id_user, que es un identificador único para cada usuario.
Tabla staff: La tabla cuenta con atributos creado por el staff y una llave foránea id_user a la tabla user, nos damos cuenta que todos los atributos dependen funcionalmente de id_staff.

Tabla student: La tabla cuenta con atributos creado por el staff y una llave foránea id_user a la tabla user, nos damos cuenta que todos los atributos dependen funcionalmente de id_student.
Tabla admin: La tabla cuenta con atributos creado por el staff y una llave foránea id_user a la tabla user, nos damos cuenta que todos los atributos dependen funcionalmente de id_admin.
Tabla campus: Tiene una llave primaria (id_campus) y atributos como ubicación, nombre y admin (llave foranea a la tabla admin). Admin depende funcionalmente de campus ya que cada campus tiene un solo admin.
Tabla test: Tiene una llave primaria (id_test), además atributos como nombre, status, descripción, creado por, fecha de lanzamiento y creado por. Todos estos atributos son dependencias funcionales ya que estos atributos no existirían si no existiera id_test, donde id_test es un identificador único para cada test.
Tabla attendance: Como tal la tabla tendria tres atributos que conforman una llave primaria (id_test, id_student, id_course), y un solo atributo date, este dependerá funcionalmente de los tres atributos en conjunto que forman la llave primaria.
Tabla enrolls: Como tal la tabla tendria tres atributos que conforman una llave primaria (id_staff, id_student, id_course), y un solo atributo date, este dependerá funcionalmente de los tres atributos en conjunto que forman la llave primaria.
Tabla question: Tiene una llave primaria id question, con atributos como descripción, tipode
Tabla question test: Relaciona las preguntas y al test, por lo que estas dos forman una llave primaria.
Tabla option list: Tiene una llave primaria (id option list) y cada option list tiene un nombre asociado.
Tabla option: Tiene una llave primaria (id option), la cual tiene campos que dependen funcionalmente, como id question, literal, descripción y correcta.
Tabla answer: Hay dos tipos de respuesta (respuesta abierta y respuesta cerrada), estas tablas tienen una llave primaria id answer, de esta llave primaria van a haber dependencias funcionales como la fecha, la respuesta que dio el estudiante, el id de la pregunta asociada y el id del estudiante que respondió.

Challenge No. 4 - Draw the Relational design.



Link Lucidchart: Página 2 - Relacional
https://lucid.app/lucidchart/e6699e3f-089f-4560-9490-782b73e6f67a/edit?invitationId=inv_97576286-047e-4529-81a5-1739d1d9f25e

Link png:
https://drive.google.com/file/d/158xDbQmZ90uxyY5yruzI5q9BcSEjap1e/view?usp=sharing

Sprint No. 3: Improvement of the system prototype for attendance management

Challenge No. 1 - Create SQL scripts to create database in PostgreSQL and fill the database with dummy data

https://github.com/Odzen/FinalProject_Databases/tree/main/Sprint3/Challenge1

Challenge No. 2 - Create three triggers to support some situations in the problem domain

https://github.com/Odzen/FinalProject_Databases/tree/main/Sprint3/Challenge2

1. Una opción está asociada a una pregunta si y solo si esta pregunta es de tipo cerrada. Además se tiene que verificar que el literal se inserta o se actualiza solo si este literal no está asociado antes con una pregunta ya anteriormente.
   https://github.com/Odzen/FinalProject_Databases/blob/main/Sprint3/Challenge2/trigger_1.sql

2. No dejar matricular estudiantes en un curso que está inactivo.
   https://github.com/Odzen/FinalProject_Databases/blob/main/Sprint3/Challenge2/trigger_2.sql

3. No dejar responder un test para marcar asistencia a un curso, si el test está inactivo.
   https://github.com/Odzen/FinalProject_Databases/blob/main/Sprint3/Challenge2/trigger_3.sql

Challenge No. 3 - Create two stored procedures to support some situations in the problem domain.

https://github.com/Odzen/FinalProject_Databases/tree/main/Sprint3/Challenge3

1. Actualiza la última vez conectado y la última ip desde la que se conectó un usuario.
   https://github.com/Odzen/FinalProject_Databases/blob/main/Sprint3/Challenge3/1_procedure.sql
2. Cada vez que se crea un usuario (Admin, Staff or Student) ,analiza el tipo y añade el registro en la tabla respectiva.
   https://github.com/Odzen/FinalProject_Databases/blob/main/Sprint3/Challenge3/2_procedure.sql
3. Inserta una respuesta a una pregunta. Luego mira cuántas preguntas ha respondido el estudiante de ese test, y cuántas le quedan por responder.
   Si con la respuesta que quiero insertar ya no le quedan al estudiante más preguntas por responder de ese test, entonces insertar el récord solo en answers. Si respondió la última pregunta, entonces insertar el récord en answers y además insertar un nuevo récord en attendance, porque significa que el estudiante ya completó un cuestionario de attendance.
   Como hay dos tipos de preguntas (abiertas y cerradas), hay un procedimiento para cada tipo de pregunta.
   https://github.com/Odzen/FinalProject_Databases/blob/main/Sprint3/Challenge3/3_procedure.sql

Challenge No. 4 - Create a view to support the report of student attendance and answers.

Para ver los estudiantes que han marcado asistencia mediante la aplicación, en el código fuente fué necesario usar múltiples JOINs para lograr obtener y ver toda la información de cada estudiante, por ejemplo, ver su código, nombre, preguntas y respuestas marcadas, con esto se cumple el objetivo de consultar la asistencia.

https://github.com/Odzen/FinalProject_Databases/tree/main/Sprint3/Challenge4

Challenge No. 5 - Prototype implementation improvements.

Recursos dados por el profesor:
https://github.com/japeto/attendance-skeleton
https://github.com/ADRE9/bunk-manager-mern

Este challenge no se realizó, por diferentes razones. Las tres principales fueron:

1. Desconocimiento de las tecnologías en el backend como: Express, Node, Python. Y en el frontend como: React. Y se considera que son tecnologías que es imposible aprender en un periodo de tiempo tan corto. Considerando que el proyecto se dejó asignado en las últimas semanas y que la carga estudiantil fue demasiado grande al final del semestre. Además, fueron tecnologías que no se vieron en clase, ni siquiera de manera introductoria, como por ejemplo ORM.
2. Mal entendido entre profesor y estudiantes sobre la entrega del código de ejemplo y guía, puesto que la fecha de entrega de este código se pospuso varias veces, y se cree que era necesario mucho antes para familiarizarnos con tecnologías que no conocíamos e intentar implementar nuestro modelo.

Al final cada integrante del grupo intentó estudiar por su cuenta estos temas, pero al ver que no era posible en tan poco tiempo, se decidió priorizar el desarrollo de otros sprints y challenges, puesto que este último challenge solo tiene un valor de 10 puntos en la nota total.

Repositorio de Github

https://github.com/Odzen/FinalProject_Databases

Sustentación en formato video

Link drive:
https://drive.google.com/file/d/1neF1pxsMdNvkYOplyxoTHzb8RZPmpJOA/view?usp=sharing
Link Youtube: https://youtu.be/vOYQ7LQ5T7c

Se recomienda ver los videos en una velocidad x1.5. Se cree que es bastante extenso y subiendo la velocidad del mismo sigue siendo comprensible.