UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Herramienta de modelado de bases de datos relacionales con Crows Foot

Presentado por Juan Romera Pérez
en Universidad de Burgos — 9 de julio de 2024
Tutor: Jesús Manuel Maudes Raedo

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Jesús Maudes Raedo, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Juan Romera Pérez, con DNI 71352256T, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado "Herramienta de modelado de bases de datos relacionales con Crows Foot".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 9 de julio de 2024

Vº. Bº. del Tutor:

D. Jesús Maudes Raedo

Resumen

El uso de sistemas de bases de datos ha aumentado en gran medida, con lo que la necesidad de facilitar la comprensión de su arquitectura y funcionamiento consecuentemente se ha visto en auge. Para suplir esta necesidad, disponemos de los diagramas relacionales con los que representar bases de datos.

En este trabajo se explora la creación de una aplicación web que permita modelar dicho tipo de diagramas utilizando la notación Crows Foot, permitiendo almacenar los diagramas para su reedición y que es capaz de generar un script relacional de creación de tablas a partir del diagrama del usuario.

Descriptores

Aplicación web, diagrama relacionales, base de datos relacional.

Abstract

The use of data base systems has increased greatly, so the need to provide an easy way to understand the architecture and operation has grown. To meet this need, relational diagrams are available to represent databases.

In this work we explore the creation of a web application that allows modeling this type of diagram using the Crows Foot notation, allowing to store the diagrams for their re-edition and that is able to generate a relational script to create tables from the user's diagram.

Keywords

Web application, relational diagram, relational database.

Índice general

Índice general iii

Índice de figuras v

Índice de tablas vi

1. Introducción 1
- 1.1. Estructura de la memoria . . . 1
- 1.2. Estructura de los anexos . . . 2
- 1.3. Materiales adjuntos . . . 2

2. Objetivos del proyecto 5
- 2.1. Objetivos generales . . . 5
- 2.2. Objetivos técnicos . . . 5
- 2.3. Objetivos personales . . . 6

3. Conceptos teóricos 7
- 3.1. Modelo relacional . . . 7
- 3.2. Notación patas de cuervo . . . 10

4. Técnicas y herramientas 17
- 4.1. Metodologías . . . 17
- 4.2. Control de versiones . . . 17
- 4.3. Hosting del repositorio . . . 18
- 4.4. Comunicación . . . 18
- 4.5. Entorno de desarrollo (IDE) . . . 18
- 4.6. Documentación . . . 19

4.7. Documentación del código 19
4.8. Servicios de integración continua 19
4.9. Despliegue 20
4.10. Librerías 20

5. Aspectos relevantes del desarrollo del proyecto 21
5.1. Introducción 21
5.2. Formación 21
5.3. Librería mxGraph 22
5.4. Desarrollo de la aplicación 23
5.5. Documentación 30
5.6. Diferencias entre el ejemplo Schema y este trabajo 30

6. Trabajos relacionados 33
6.1. Artículos científicos 33
6.2. Proyectos 33
6.3. Fortalezas y debilidades del proyecto 35

7. Conclusiones y Líneas de trabajo futuras 37
7.1. Conclusiones 37
7.2. Líneas de trabajo futuras 38

Bibliografía 41

Índice de figuras

3.1. Alumnos[5] . . . 8
3.2. Clave primaria[8] . . . 9
3.3. Clave foránea[8] . . . 10
3.4. Multiplicidad de uno[2] . . . 11
3.5. Multiplicidad de muchos[2] . . . 12
3.6. Obligatoria[2] . . . 12
3.7. Opcional[2] . . . 12
3.8. Cero o muchos[2] . . . 13
3.9. Uno o muchos[2] . . . 13
3.10. Uno y solo uno[2] . . . 13
3.11. Cero o uno[2] . . . 14
3.12. Relación Uno-a-Uno[2] . . . 14
3.13. Relación Uno-a-Muchos[2] . . . 14
3.14. Relación Muchos-a-Muchos[2] . . . 15
5.1. Símbolos 1 . . . 25
5.2. Símbolos 2 . . . 25
5.3. Símbolos 3 . . . 25
5.4. Panel propiedades . . . 28

Índice de tablas

6.1. Comparativa de características 35

1. Introducción

El uso de los sistemas de bases de datos está ampliamente extendido en diferentes ámbitos con múltiples usos. Dentro de los diferentes tipos de bases de datos nos encontramos las bases de datos relacionales[11]. Un modelo comúnmente utilizado para describir este tipo de bases de datos es el modelo relacional (relational model)[12], modelo compuesto por relaciones, las cuales representan la información con la que trabaja la base de datos, y sus atributos.

Entre las notaciones comúnmente utilizadas para representar las relaciones entre las entidades de la base de datos encontramos la notación Crows Foot o patas de cuervo, objeto de este trabajo. Esta notación, que recibe este nombre debido a la similitud de los extremos de las líneas que representan las relaciones con las patas de un cuervo, utiliza tres símbolos para representar la cardinalidad de las relaciones que se utilizan en pares para representar los cuatro tipos de cardinalidad que una entidad podría tener en una relación.

1.1. Estructura de la memoria

La memoria se estructura de la siguiente forma:

- **Introducción:** descripción del problema a resolver y la solución propuesta. Estructura de la memoria y listado de materiales adjuntos.
- **Objetivos del proyecto:** exposición de los objetivos que se persiguen con el proyecto.
- **Conceptos teóricos:** breve explicación de los conceptos teóricos clave para la comprensión de la solución propuesta.

- **Técnicas y herramientas:** listado de técnicas metodológicas y herramientas utilizadas para gestión y desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** exposición de aspectos destacables que tuvieron lugar durante la realización del proyecto.
- **Trabajos relacionados:** estado del arte en el campo del diseño de aplicaciones web para la realización de diagramas de bases de datos.
- **Conclusiones y líneas de trabajo futuras:** conclusiones tras la realización del proyecto y posibilidades de mejora o expansión de la solución aportada.

1.2. Estructura de los anexos

Junto a la memoria se proporcionan los siguientes anexos:

- **Plan del proyecto software:** planificación temporal y estudio de viabilidad del proyecto.
- **Especificación de requisitos del software:** se describe la fase de análisis; los objetivos generales, el catálogo de requisitos del sistema y la especificación de requisitos funcionales y no funcionales.
- **Especificación de diseño:** se describe la fase de diseño; el ámbito del software, el diseño de datos, el diseño procedimental y el diseño arquitectónico.
- **Manual del programador:** recoge los aspectos más relevantes relacionados con el código fuente (estructura, compilación, instalación, ejecución, pruebas, etc.).
- **Manual de usuario:** guía de usuario para el correcto manejo de la aplicación.

1.3. Materiales adjuntos

Los materiales que se adjuntan con la memoria son:

- Memoria
- Anexos

- Código fuente y recursos de la aplicación
- Vídeo presentación y vídeo tutorial
- Enlaces a repositorio y a aplicación de proceso de calidad

2. Objetivos del proyecto

A continuación, se detallan los diferentes objetivos que han motivado la realización del proyecto.

2.1. Objetivos generales

- Desarrollar una aplicación web que permita diseñar bases de datos utilizando la notación de *patas de cuervo*.
- Facilitar la creación de bases de datos generando código *SQL* y *SQLAlchemy* a partir del diagrama creado.
- Permitir exportar el diagrama creado en la aplicación.

2.2. Objetivos técnicos

- Desarrollar una aplicación web utilizando la librería de Javascript *mxGraph*.
- Añadir la capacidad de diseñar *Diagramas relacionales* desde la aplicación.
- Implementar la notación de *Patas de Cuervo* para trabajar con los diagramas.
- Desarrollar la posibilidad de descargar el diagrama en un fichero XML para posteriormente ser capaces de importar el mismo diagrama desde este fichero.

- Comprobar que la aplicación funciona en una amplia gama de navegadores.
- Aplicar la metodología ágil Scrum en el desarrollo software.
- Utilizar Git como sistema de control de versiones, almacenando el repositorio en GitHub.
- Desplegar la aplicación para que pueda ser utilizada.

2.3. Objetivos personales

- Aprender a manejarme en desarrollo web.
- Mejorar mis capacidades en el desarrollo de UI.
- Ampliar mis conocimientos en el diseño de *Bases de Datos* y *diagramas relacionales*, incluyendo la, para mí, nueva notación de *Patas de Cuervo*.

3. Conceptos teóricos

En el apartado teórico hablaremos sobre el *modelo relacional*, modelo en el cual se utiliza la notación de *patas de cuervo*, sobre los *diagramas relacionales* y sobre la notación de *patas de cuervo*.

3.1. Modelo relacional

El *modelo relacional* es un acercamiento a la gestión de datos utilizando una estructura y un lenguaje coherentes con la lógica de predicados de primer orden.

El elemento central del modelo relacional es la Relación. Una relación tiene un nombre, un conjunto de atributos que representan sus propiedades y un conjunto de tuplas que incluyen los valores que cada uno de los atributos toma para cada elemento de la relación. El Dominio es otro elemento importante, el cual describe los valores válidos que un atributo puede tomar[5].

Las relaciones se representan con una tabla de dos dimensiones, siendo las columnas sus diferentes atributos y las filas las tuplas. Ejemplo de relación *Alumnos*:

ALUMNOS

COD_MATRICULA	NOMBRE	CIUDAD	COD_GRUPO
101	Juan Montero	Alcorcón	11
102	Alicia Cristóbal	Leganés	11
202	Ana Vallejo	Leganés	21
300	Ignacio López	Móstoles	31
103	Leticia Martínez	Alcorcón	--

Figura 3.1: Alumnos[5]

Restricciones inherentes

El modelo relacional impone una serie de restricciones inherentes:

- En una relación no puede haber dos tuplas iguales (*obligatoriedad de clave primaria*).
- El orden de las tuplas y el de los atributos no es relevante.
- Cada atributo solo puede tomar un único valor del dominio sobre el cual está definido (*no hay grupos repetitivos*).
- Ningún atributo que forme parte de la clave primaria puede tomar un valor nulo (*regla de integridad de entidad*).

Restricciones semánticas

El modelo relacional también proporciona mecanismos para recoger *restricciones semánticas*:

- **Restricción de *clave primaria* (PRIMARY KEY):** Permite declarar un atributo o conjunto de atributos como la clave primaria de una relación que identifica inequívocamente cada tupla de la relación. Por la restricciones inherentes anteriores, es necesario definir en toda relación una clave primaria que no puede ser nula.
- **Restricción de *unicidad*(UNIQUE):** Permite definir claves de identificación alternativas (los valores de uno o varios atributos no pueden repetirse en la relación).

- **Restricción de *obligatoriedad* (NOT NULL):** Permite declarar si uno o varios atributos de una relación deben tomar siempre un valor, no puede tomar valores nulos.
- **Restricción de *clave ajena*(FOREIGN KEY):** Permite enlazar relaciones de una base de datos. La integridad referencial indica que los valores de la clave ajena en la relación que referencia deben corresponderse con alguno de los valores existentes de clave definida en la relación referenciada, o tener un valor nulo si se permite. Los atributos que son clave ajena en una relación no necesitan tener los mismos nombres que los atributos de la clave primaria con la que se corresponden.

Claves primarias

Este atributo define de forma inequívoca un ejemplar de una relación determinada, de tal manera que no podemos encontrar dos ejemplares distintos con una misma clave primaria. Se representa visualmente de la siguiente forma.

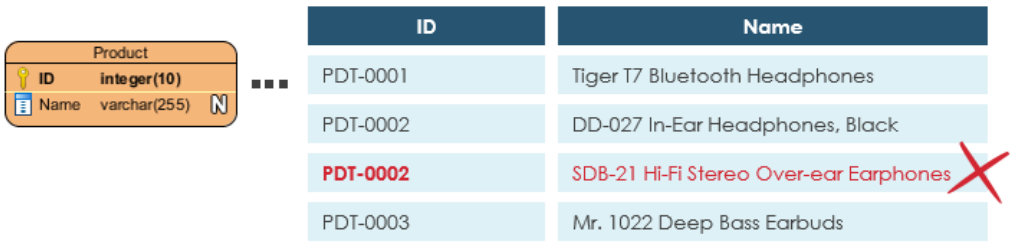

Figura 3.2: Clave primaria[8]

En la imagen vemos como se identifica a un producto por su atributo ID, siendo la tercera entrada no válida debido a que este se encuentra repetido.

Claves foráneas

La clave foránea alamcenará la clave primaria del ejemplar de la otra relación con el que se enlaza.

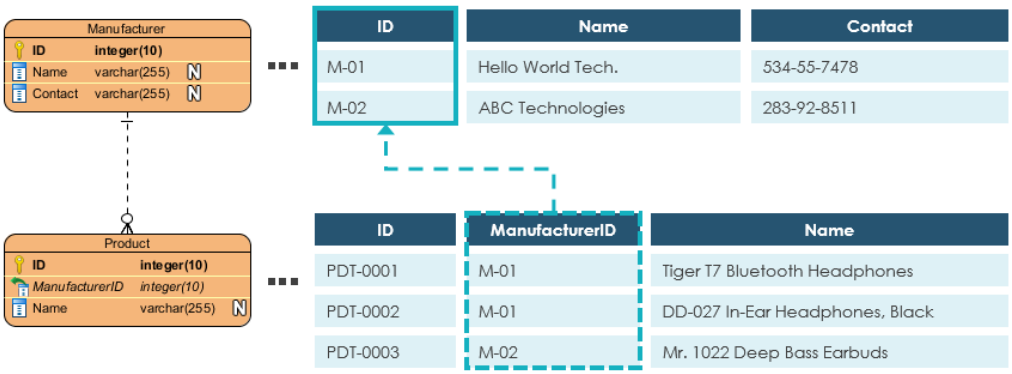

Figura 3.3: Clave foránea[8]

Podemos observar como el atributo *ID* de la relación *Manufacturer* se almacena como clave foránea en la relación *Product*, indicando su enlace.

3.2. Notación patas de cuervo

Historia

La notación de ***patas de cuervo*** nace en 1976, apareciendo por primera vez en un artículo de Gordon Everest[4], aunque la notación ha ido evolucionando a lo largo del tiempo. Llamada por primera vez *flecha invertida* para distinguirla de otras notaciones, se escogió este símbolo porque no implica direccionalidad o un acceso físico, además de ser visualmente intuitivo[2]. Aunque el tema principal del artículo publicado por Gordon Everest era las estructuras de bases de datos explicadas con ejemplos, el uso de esta nueva notación fue fortuito aunque cuidadosamente elegido.

Símbolos en la notación de patas de cuervo

En esta notación encontramos los típicos símbolos de los diagramas relacionales mencionados anteriormente, con algunas modificaciones. Los símbolos en esta notación se dibujan de la siguiente forma:

Entidades

En la notación de patas de cuervo, las *entidades* se representan mediante tablas, con el nombre escrito en la parte superior y sus atributos debajo.

Para la realización de este proyecto se han utilizado las entidades para representar las relaciones o tablas utilizadas en el modelo relacional.

Atributos

Como se ha visto en el modelo relacional, los atributos forman parte de las relaciones, en este caso llamadas entidades, simbolizando sus propiedades. Los atributos de una entidad se mostrarán en la tabla que la representa, presentándose uno debajo de otro. Los atributos que describan la entidad de forma única, como su clave primaria o los atributos unique, se indicarán de forma especial.

Relaciones

Las relaciones son lo que da nombre a este tipo de notación, puesto los símbolos dibujados en los extremos de las relaciones se asemejan a las patas de un cuervo. Las relaciones se representan como una línea recta uniendo dos entidades, son relaciones binarias, con un nombre escrito sobre la línea que describe la relación.

En los extremos de la relación encontraremos dos indicadores, uno a cada lado, que nos representarán la cardinalidad de cada entidad en la relación. Podemos dividir estos símbolos en dos partes, las cuales nos indican diferentes propiedades.

- **Multiplicidad:** Indica el número máximo de ejemplares que podemos encontrar en la relación de la entidad a la que apunta. En la notación de patas de cuervo disponemos de *multiplicidad de uno* y de *multiplicidad de muchos*. Interpretaremos la multiplicidad según el extremo del símbolo, como se muestra en la imagen:

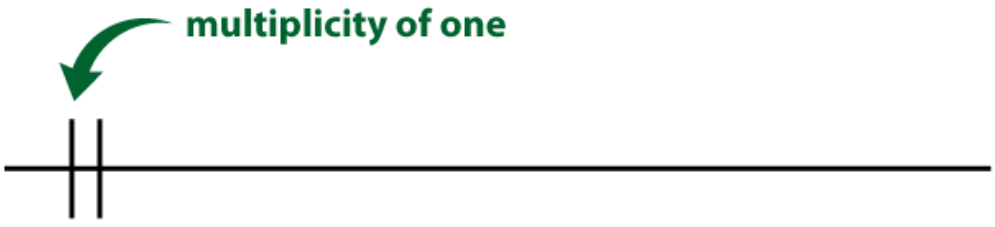

Figura 3.4: Multiplicidad de uno[2]

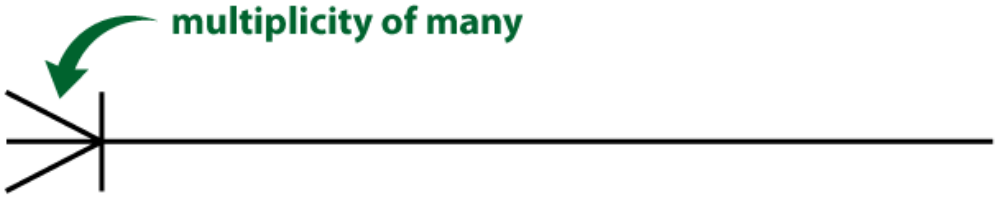

Figura 3.5: Multiplicidad de muchos[2]

- **Opcionalidad:** Indica el número mínimo de ejemplares de una entidad que podemos encontrar, pudiendo ser *obligatoria* (mínimo una ocurrencia) u *opcional* (no hay mínimo de ocurrencias). La opcionalidad se muestra en la parte interior de la relación:

Figura 3.6: Obligatoria[2]

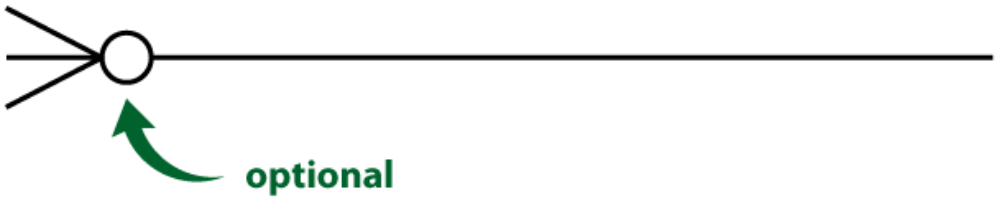

Figura 3.7: Opcional[2]

Mediante la combinación de la multiplicidad y la opcionalidad obtendremos los símbolos que utilizaremos en la notación de patas de cuervo. Tenemos cuatro posibilidades diferentes:

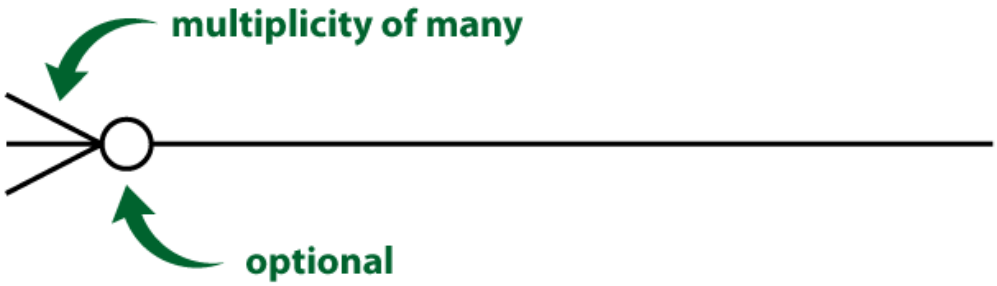

Figura 3.8: Cero o muchos[2]

Figura 3.9: Uno o muchos[2]

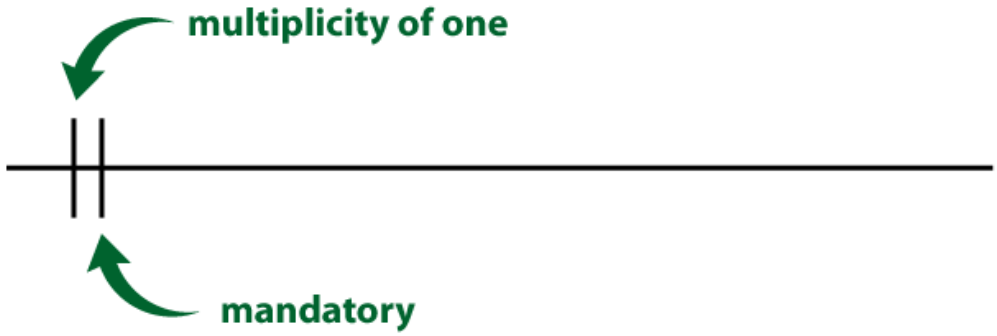

Figura 3.10: Uno y solo uno[2]

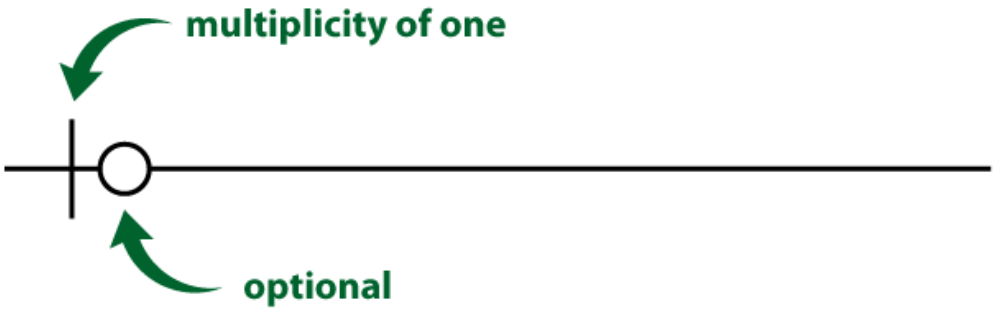

Figura 3.11: Cero o uno[2]

Combinando estos símbolos en los extremos de una relación, obtendremos los diferentes grados[1]:

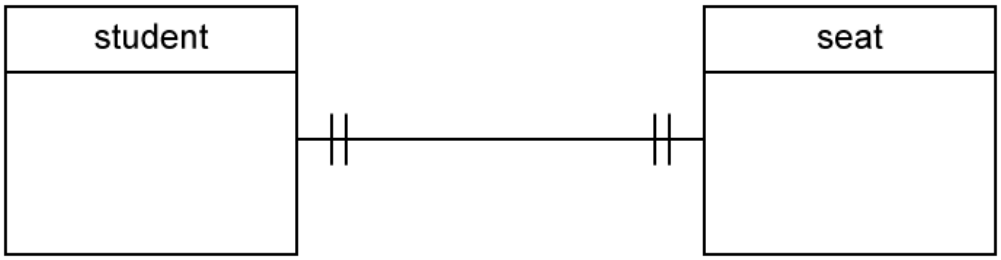

Figura 3.12: Relación Uno-a-Uno[2]

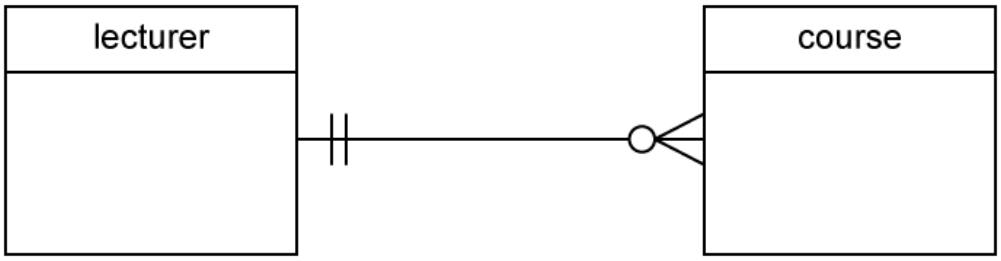

Figura 3.13: Relación Uno-a-Muchos[2]

[1]Las imágenes utilizadas se han escogido para mostrar los diferentes grados, no representan una implementación real

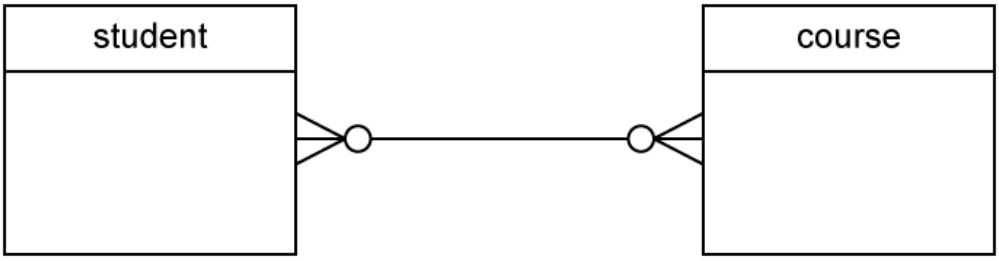

Figura 3.14: Relación Muchos-a-Muchos[2]

Claves primarias y claves foráneas

Para distinguir los diferentes ejemplares y sus relaciones de cada entidad, en los diagramas de patas de cuervo relacionales utilizamos un atributo conocido como claves, siendo las claves primarias para diferenciar cada ejemplar de una entidad y las claves foráneas para determinar con que otros ejemplares de otras entidades se relaciona el ejemplar con el que estemos trabajando.

Tratamiento de las cardinalidades en el modelo relacional

Una vez hemos visto las diferentes relaciones que podemos encontrar en la notación de patas de cuervo y el uso de las claves primarias y claves foráneas para identificar dichas relaciones, podemos continuar con la cardinalidad de las relaciones y el manejo de las claves en función de esta[3].

Cardinalidad Uno-a-Uno (1:1)

En una relación Uno-a-Uno encontramos que un ejemplar de una entidad se relaciona únicamente con un ejemplar de la otra entidad.

Teniendo en cuenta la opcionalidad de la que disponemos en la notación de patas de cuervo, encontramos diferentes casos con esta cardinalidad.

- **Ambos lados obligatorios u opcionales**: En este caso, la entidad en la que situar la clave foránea queda a criterio del programador.
- **Un lado opcional y otro obligatorio**: Cuando uno de los lados sea opcional, colocaremos la clave foránea en el lado opcional, asumiendo que siempre que exista el lado opcional existirá el lado obligatorio, pero no viceversa.

En la cardinalidad uno a uno, la clave foránea será *UNIQUE*, evitando así que los registros de un lado de la relación se relacionen más de una vez con registros del otro lado de la relación.

En el caso que el lado de la relación contrario al lado donde se almacena la clave foránea sea obligatorio, entonces la clave foránea deberá ser *NOT NULL*, puesto que siempre va a existir un registro de dicho lado de la relación (es obligatorio) cuando exista uno con el que se relaciona de este lado.

Las relaciones uno a uno en la que tengamos ambos lados de la relación obligatorios es un caso complejo que deberemos tratar con especial cuidado, debido a que al tener que cumplir la integridad referencial en ambos lados de la relación, cuando tratemos de insertar un nuevo registro en una de las tablas obtendremos un error al no existir el otro registro. Debido a esto, el tratamiento de este tipo de relaciones queda al criterio del programador, ya sea bien utilizando valores nulos temporales (no cumpliendo la restricción de obligatoriedad) que se actualizarán posteriormente, utilizando transacciones atómicas para insertar ambos registros secuencialmente o triggers para manejar la inserción automática en la tabla relacionada, entre algunas de las opciones.

Cardinalidad Uno-a-Muchos (1:M)

En una relación Uno-a-Muchos un ejemplar de una entidad determinada se relaciona con diversos ejemplares de otra entidad, pero los ejemplares de dicha entidad se relacionan únicamente con un ejemplar de la primera entidad.

En este caso, la clave foránea se situará en el lado muchos.

Si el lado a uno de la relación es obligatorio, entonces la clave foránea será *NOT NULL*, obligando a que siempre exista un registro en el otro lado de la relación con el que se asocia.

Cardinalidad Muchos-a-Muchos (M:M)

Una relación Muchos-a-Muchos muchos ejemplares de una entidad se relacionan con muchos ejemplares de otra entidad.

Para implementar esta cardinalidad, se utiliza una tercera tabla intermedia, la cual contiene como claves foráneas las claves primarias de las dos tablas a las que hace referencia. La combinación de las dos claves foráneas será la clave primaria de dicha tabla intermedia.

4. Técnicas y herramientas

4.1. Metodologías

Scrum

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, obteniendo el mejor resultado posible. En Scrum se realizan entregas parciales y regulares del producto. Estas entregas ocurren en interacciones conocidas como sprints. [13].

Gitflow

Gitflow es un modelo alternativo de creación de ramas en Git en el que se utilizan ramas de función y varias ramas principales. Con este flujo de trabajo disponemos de la rama *main*, donde se encuentra la última versión estable del proyecto, la rama *develop*, en la que se realizan las nuevas implementaciones para la próxima release. Podemos utilizar diferentes ramas *feature*, por cada característica que queramos desarrollar. En caso de encontrar un fallo en el proyecto que necesite un rápido parcheado, utilizaremos la rama *hotfix*[1].

4.2. Control de versiones

Herramienta utilizada: *Git*

Git es un sistema de control de versiones distribuido, pensado en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones[9]. Git es la herramienta de control de versiones más utilizada actualmente, que además nos permite almacenar una versión local del repositorio.

4.3. Hosting del repositorio

Herramienta utilizada: *Github*

Github es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git[10]. En Github almacenaremos los recursos utilizados en el proyecto.

4.4. Comunicación

Herramienta utilizada: *email* y *Teams*

Para mantener la comunicación durante el desarrollo del proyecto se ha utilizado el email para mantener una comunicación mediante mensajes directos y la herramienta Teams para tener un contacto directo mediante llamadas de vídeo.

4.5. Entorno de desarrollo (IDE)

Herramienta utilizada: *Visual Studio Code*

Visual Studio Code es un IDE desarrollado por Microsoft bastante ligero y comúnmente utilizado para el desarrollo de aplicaciones web mediante Javascript.

Este IDE nos permite instalar multitud de extensiones que nos permitirán desarrollar nuestra aplicación de manera más cómoda y eficiente. Algunas de las extensiones que se han utilizado son las siguientes.

Extensiones VSCode

Live Server

La extensión *Live Server* nos permite desplegar un servidor de manera local en el que ejecutar nuestra aplicación web. De esta forma, podemos tener una rápida visión de como se verá nuestra aplicación en el momento de un despliegue real.

SonarLint

Extensión relacionada con una de las herramientas de integración continua utilizada, Sonarcloud, nos permite ver en el editor las reglas de So-

narcloud que no estamos cumpliendo, permitiendo corregir rápidamente posibles fallos.

Codacy

Igual que anteriormente con SonarLint, esta extensión conecta con otra herramienta de integración continua con el mismo nombre.

4.6. Documentación

Herramienta utilizada: *Overleaf*

Overleaf es una plataforma online para la edición de LaTeXbasado en la nube. La documentación presentada se ha realizado en esta plataforma.

4.7. Documentación del código

Herramienta utilizada: *JSDoc*

JSDoc es la sintaxis utilizada para documentar el código de Javascript mediante comentarios, además de un módulo que genera automáticamente la documentación del código a partir de dichos comentarios. La sintaxis es muy similar a la utilizada en Javadoc, lo que facilita su uso. Se puede descargar como un módulo de node, lo que nos genera los ficheros HTML con la documentación.

4.8. Servicios de integración continua

Calidad del código

SonarCloud

SonarCloud es un servicio de análisis estático de código basado en la nube ofrecido por la empresa SonarQube. Esta herramienta es capaz de detectar problemas de código en 25 lenguajes de programación diferentes. Tiene fácil integración con GitHub y nos permite medir la seguridad de nuestras aplicaciones y mejorar la confiabilidad y mantenibilidad de nuestro código[6].

Codacy

Codacy es otra herramienta de integración continua que monitorizará la calidad de nuestro código, además de duplicaciones y problemas de seguridad. Añade más cobertura además de la que nos pueda proporcionar SonarCloud.

4.9. Despliegue

Herramienta utilizada: *Vercel*

Vercel es una plataforma de servicio en la nube que permite a los desarrolladores desplegar sus aplicaciones y servicios web. Vercel se integra con GitHub, permitiendo desplegar la aplicación desde el repositorio actualizando cada vez que se realiza un commit en la rama principal.

4.10. Librerías

mxGraph

mxGraph es una librería de Javascript empleada para la realización de diagramas que utiliza SVG y HTML para renderizado.

Esta la librería es el pilar sobre el que se sustenta este trabajo, realizando la gestión de los datos que se muestran y cómo se muestran. mxGraph se divide en 8 paquetes, teniendo en el nivel más alto la clase *mxClient* la cual se encarga de incluir o importar de forma dinámica el resto de elementos de la librería. Estos paquetes nos proporcionarán las clases necesarias para generar un diagrama en el que añadir los elementos que deseemos, dentro de unos parámetros dados[7].

5. Aspectos relevantes del desarrollo del proyecto

En este capítulo se comentan los aspectos más relevantes a la hora de desarrollar el proyecto, qué decisiones se tomaron y qué llevó a ellas.

5.1. Introducción

Antes del comienzo del desarrollo de esta aplicación mi trabajo con desarrollo web, y más concretamente con *Javascript*, había sido prácticamente nulo, más allá de algún trabajo concreto realizado durante el curso. Esto ha supuesto varias dificultades que se han tenido que resolver a lo largo del desarrollo.

El trabajo se seleccionó principalmente por falta de alternativas, ya que se solicitó la realización de otro trabajo, pero ante la larga espera, la falta de respuesta a dicha solicitud hizo que cuando se presentó la oportunidad de realizar este proyecto se aceptase rápidamente.

5.2. Formación

Puesto que se partía de una base casi nula en el trabajo con Javascript, el periodo de formación se extendió algo más de lo esperado. Se realilzó un trabajo de formación en Javascript, Node y React, aunque más adelante se descartó el uso de estos frameworks.

Javascript

Para la formación en Javascript se hizo uso de tutoriales disponibles en webs como W3Schools, además de algún ejemplo obtenido de otras webs. Se realizó trabajo con clases, objetos, funciones básicas de Javascript, además de utilidades como fetch.

Node

Igual que como se ha comentado en la sección anterior, los tutoriales de w3schools fueron bastante útiles, además de la formación disponible en la propia web de Node. Para la formación, se realizaron ejemplos para trabajar principalmente con ficheros, además de identificar que recursos mostrar al usuario en función de los parámetros obtenidos.

React

En la formación con React también se siguió la formación de la web mencionada anteriormente, pero en este caso se trabajó más con ejemplos, como el mostrado en la propia web de React, con el que se construye un tres en raya báscio, además de un ejemplo integrando React y Node en el que construimos una aplicación de notas en la web que almacenaba y obtenía los datos con los que el usuario estuviera trabajando desde una base de datos.

5.3. Librería mxGraph

El pilar fundamental de este proyecto es la librería de Javascript *mxGraph*, la cual se encarga de la gestión de gran parte de los elementos de nuestra aplciación. Una vez considreado terminado el apartado de formación anterior, se comenzó a trabajar con esta librería.

Formación

La formación en esta librería consistió basicamente en la lectura y realización de algunos de los diferentes ejemplos que podemos encontrar en directorio de ejemplos de mxGraph, ejemplos disponibles gracias a los propios creadores. También se hizo un uso constante de la documentación disponible en la especificación de la API de mxGraph, además de los manuales que podemos encontrar en la web de mxGraph. Más allá de esto, no se encontró mucho material con el que se formarse en la librería, sin tener en cuenta secciones de código encontradas plataformas de preguntas y respuestas en

línea destinadas al intercambio de conocimiento entre desarrolladores, como *Stackoverflow*.

Ejemplos

Teniendo en cuenta que la mayoría de la formación en la librería se hizo con los ejemplos encontrados, a continuación se listarán algunos de los más relevantes.

- **Hello world:** Ejemplo básico de hola mundo para ir familiarizándome con la librería.
- **User object:** Ejemplo para aprender a trabajar con objetos personalizados dentro de las celdas de mxGraph.
- **Toolbar:** Ejemplo que nos muetra como implementar una barra de herramientas desde la que añadir elementos a nuestro grafo utilizando drag and drop.
- **Fileio:** Ejemplo en el que se trata como obtener un grafo a partir de un fichero, en nuestro un fichero XML.
- **Markers:** Ejemplo que nos enseña como implementar símbolos personalizados en los extremos de los enlaces del grafo.
- **Schema:** En este ejemplo se muestra como se puede implementar un editor muy sencillo de diagramas de base de datos, con la capacidad de añadir tablas, columnas y relaciones.

Una vez trabajado con todos los ejemplos y entendidos estos, se comenzó a trabajar en la aplicación, tomando el ejemplo *Schema*[2] como base desde la que empezar a construir.

El seguimiento de esta etapa de formación se puede encontrar en el repositorio del proyecto en la rama *Formación*.

5.4. Desarrollo de la aplicación

Una vez considerada completa la formación, comienza el desarrollo de la aplicación. Se consideró que el ejemplo *Schema* podría servir como base sobre la que trabajar, ya que contaba con funcionalidades como incluir

[2] https://jgraph.github.io/mxgraph/javascript/examples/schema.html

tablas y columnas arrastrando, además de la posiblidad de crear relaciones entre las tablas, aunque, a pesar de ser un buen primer punto de contacto, carecía de diversas funcionalidades como los símbolos de patas de cuervo que necesitamos, considerar más de una cardinalidad en la relación, la gestión automática de las claves o una correcta generación de código.

Metodologías

Para el desarrollo del proyecto se intento llevar un enfoque los más profesional posible, utilizando metodologías que se siguen en un entorno empresarial.

Se ha tratado de llevar la metodología ágil *Scrum* en el proyecto, siguiendo el enfoque de desarrollo incremental. Los pasos seguidos fueron:

- Se dividió el desarrollo incremental en diferentes Sprints.
- Los sprints duraban normalmente entre 1-2 semanas.
- Al comienzo de cada sprint se creaba una pila de tareas que se iban a realizar en dicho sprint.
- Durante el sprint se podían añadir nuevas tareas en función de problemas o cuestiones encontradas.

El desarrollo del código se ha realizado siguiendo una metodología de enseayo y error, añadiendo las funcionalidades de forma paulatina probando hasta obtener el resultado deseado.

Aspecto de tablas y columnas

El ejemplo Schema anteriomente mencionado ya partía con un diseño del aspecto de tablas, columnas y relaciones que tenía buena pinta. Debido a que el tema de este trabajo era trabajar con la notación de patas de cuervo y no el diseño de tablas y columnas en sí, se decidió mantener este diseño, aunque se realizaron algunos cambios a lo largo del desarrollo para conseguir que el diseño se ajuste mejor al proyecto. Algunos de estos cambios son los siguientes.

- Ajuste de la etiqueta de la tabla a la izquierda y no centrado
- Borde para todas las columnas

- Icono personalizado para indicar que una columna es clave foránea

Además de estos cambios, más adelante introdujo en la aplicación un apartado que permite modificar algunos de estos aspectos estéticos de las tablas, columnas y relaciones, del que se hablará más adelante.

Símbolos de patas de cuervo

Al comienzo del desarrollo de la aplicación, se trabajó en la implementación de los símbolos correspondientes a la notación de patas de cuervo. Para una correcta visualización, se implementaron los símbolos personalizados de tal forma que estos sean consistentes y fácilmente identificables independientemente del ángulo del enlace que representa la relación. Esto supuso un inconveniente al principio, pero se solventó rápidamente.

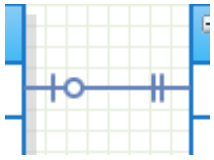

Figura 5.1: Símbolos 1

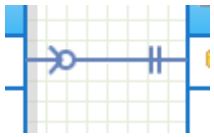

Figura 5.2: Símbolos 2

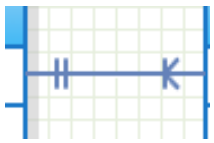

Figura 5.3: Símbolos 3

Identificando los símbolos por el tipo de multiplicidad y obligatoriedad de cada uno, implementamos los símbolos *solo_uno*, *cero_o_uno*, *cero_o_mas* y *uno_o_mas*.

Desde el apartado de propiedades de la relación podemos seleccionar qué símbolo se muestra en cada extremo de la relación, actualizando automáticamente las claves foráneas en caso de que se necesario.

Diccionario de datos

Se decidió implementar las propiedades *Título* y *Descripción* en los objetos que representaban a las columnas para permitir al usuario ampliar la información con respecto a los elementos de su diagrama, mostrando toda esta información de forma ordenada desde el botón correspondiente.

Tipo de datos de las columnas

Durante el desarrollo se tomó la decisión de mantener el tipo de dato como una cadena de texto introducida por el usuario, dejando bajo su responsabilidad introducir dicho valor correctamente correspondiendo con la base de datos que esté utilizando, ya que dicho atributo de los objetos de las columnas será utilizado más adelante en la generación de código.

Se decidió mantener el atributo así ya que se consideró que el resultado obtenido con la cantidad de trabajo requerido no merecía la pena, además de no ser este el enfoque principal del trabajo. Dejarlo bajo responsabilidad del usuario fue la opción mejor valorada.

Ajuste de etiqueta

Al principio del desarrollo surgió la idea de ajustar la anchura de las tablas y las columnas a la anchura de la etiqueta que se va a mostrar. Esta idea se descartó, en parte debido a que visualmente no es la mejor solución, pero principalmente debido a que la librería contiene problemas sin resolver sobre el ajuste automático del tamaño de sus celdas cuando estas forman parte de un layout de grupo, como en nuestro caso swimlane, provocando que una celda pueda hacerse más grande pero no pueda hacerse más pequeña o recuperar su tamaño inicial. El layout Swimlane, o layout de carriles, organiza la información en diferentes carriles, similares a los que podrías encontrar en una piscina. En la aplicación, utilizamos este layout para representar las tablas utilizando carriles verticales, teniendo como título del carril el nombre de la tabla y dentro del carril situamos cada columna de esta. Para organizar las columnas dentro del carril se utiliza un *Stacklayout*, en el cual apilamos de forma vertical cada columna en el interior de la tabla. Así, no necesitamos realizar un control exacto de la posición de las columnas, puesto que estas se apiliarán unas debajo de otras cada vez que se incluya una nueva. Además, utilizar el layout Swimlane nos facilita expandir y colapsar las tablas en el diagrama, mostrando el título de la tabla y todas las columnas o solo el título, respectivamente.

Después de esto, se decidió modificar como se mostraba la etiqueta para que esta se ajuste a la anchura de la celda. Con la función *getTextWidth* calculamos la anchura de la cadena correspondiente a la etiqueta en píxeles, permitiéndonos calcular que parte de la etiqueta mostrar, añadiendo tres puntos suspensivos (...) al final de la etiqueta indicado que esta continua.

Gestión de claves

Durante el desarrollo se consideró importante el hecho de que, cuando se crea o actualiza una relación entre dos tablas, las claves foráneas necesarias se añaden y eliminan de forma automática de las tablas correspondientes. Además, los parámetros *UNIQUE* y *NOT NULL* de estas nuevas columnas se gestionan de tal forma que corresponda con la multiplicidad y la obligatoriedad de los símbolos del enlace.

Para que esto funcionase correctamente, se implementó en el objeto de las relaciones un array que almacenase las claves foráneas asociadas, de tal forma que en caso de eliminar o actualizar una relación entre dos tablas, las claves foráneas antiguas se eliminen. Para identificar fácilmente en que tabla se encuentran las claves foráneas, se tomó la decisión de diseño de que todos los enlaces que representan las relaciones tengan en su lado origen la tabla que contiene las claves foráneas y en su lado destino la tabla con la clave primaria referenciada. Además, de esta forma podemos identificar rápidamente desde una tabla qué enlaces insertan columnas en ellas y cuáles hacen una copia de sus claves primarias en otra, ayudando a la hora de actualizar posibles referencias o tener que añadir o eliminar celdas.

Además de lo anteriormente mencionado, se introdujo la posibilidad de añadir Uniques compuestos en las tablas con dos columnas o más, permitiendo el manejo de claves candidatas también.

Teniendo en cuenta que las relaciones M:M se implementan utilizando una tabla intermedia, cuando el usuario establezca una relación de este tipo entre dos tablas, el sistema incluirá automáticamente una tabla intermedia representando a dicha relación incluyendo como claves foráneas las claves primarias de estas dos tablas.

Hasta el momento, la aplicación es capaz de trabajar únicamente con claves primarias y claves foráneas simples, aunque se permite la declaración de Uniques compuestos, con los que ya trabaja la aplicación.

Panel propiedades

Para mantener un aspecto visual más moderno, se implementó un panel lateral que muestar las propiedades del elemento del diagrama seleccionado. El panel está dividido en dos pestañas, el partado *Estilos* y el apartado *Datos*.

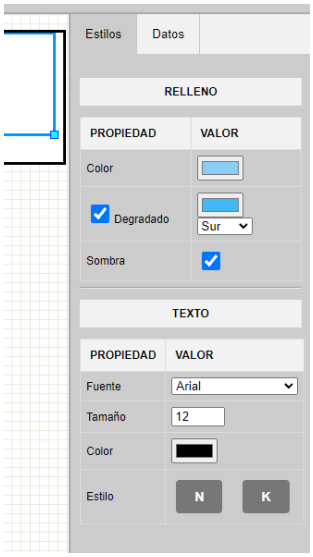

Figura 5.4: Panel propiedades

Estilos

Ya que se adoptó la estética de los elementos del diagrama Schema, para obtener una diferenciación más clara se introdujo este panel para permitir al usuario cambiar la apariencia de los elementos de su diagrama.

Entre las características que se permiten cambiar podemos encontrar el color de relleno, el color del texto, fuente, tamaño de fuente, botones de negrita y cursiva, ...

Datos

Al comienzo del desarrollo se permitía al usuario editar la información de las columnas con una pequeña ventana modal en la que se podía editar algunas de sus características a la cual se accedía desde el botón de la parte superior o desde el menú popup. Más adelante se decidió eliminar estos botones, moviendo el contenido al panel lateral y ampliándolo.

Se implementó la posibilidad de modificar los datos de las tablas y relaciones, siendo esto indispensable para seleccionar qué símbolos de la relación se deben utilizar y, por tanto, modificando toda la relación.

Añadir columnas

Además de poder añadir columnas nuevas a las tablas arrastrando el icono correspondiente, se consideró otras opciones para agilizar la creación de tablas. Las dos opciones que se consideraron y se terminaron implementando son:

- Añadir una opción en el menu popup de la tabla que permita añadir una columna
- Añadir un atajo de teclado (F4) que incluya una nueva columna en la tabla seleccionada

Mover columnas

Al utilizar un stacklayout, un layout fijo, para mostrar las columnas y estas tomaban las posición en la tabla según se fueran añadiendo a esta, se tomó la decisión de añadir la posibilidad de mover las columnas de posición dentro de las tablas, moviendo estas hacia arriba o hacia abajo.

Ya que mover las columnas de posición dentro del layout se entendió que no tenía mucho viabilidad, se decidió tomar otro enfoque, en el que en vez de mover "físicamente"de posición las celdas de las columnas, lo que se hace es intercambiar el objeto asociado y su apariencia con la celda de arriba o de abajo. De esta forma, la información sigue siendo la misma, solo se modifica la celda que la contiene. Como las columnas tienen relaciones cruzadas con otros elementos del diagrama, uno de los apartados más destacados fue la modificación de estas referencias.

Generación de código

Como otra funcionalidad importante para la aplicación, se decidió tener un apartado de generación de código en el que el sistema generaría el código equivalente al diagrama que ha creado el usuario.

La generación de código que se implementó fue para SQL y SQLAlchemy, adaptando todas las mejoras mencionadas anteriormente para que se tengan en cuenta cosas como claves foráneas y uniques compuestos.

Las principales dificultades surgieron sobre como tratar claves compuestas, además de referenciar correctamente las claves primarias desde las claves foráneas.

5.5. Documentación

La documentación del proyecto se encuentra en los documentos entregados. Se ha realizado tanto la memoria como los anexos utilizando LaTeXdesde la plataforma *Overleaf*.

Además de estos documentos, se ha generado la documentación *JSDoc* a partir de los comentarios del código de la aplicación. Esta documentación también se incluye.

5.6. Diferencias entre el ejemplo Schema y este trabajo

Como se ha comentado anteriormente, al comienzo del trabajo se tomo como base sobre la que empezar a construir el ejemplo Schema[3]. A continuación, se listan las diferencias o mejoras diferenciales implementadas en el proyecto.

- Panel lateral derecho, el cual cambia según el elemento del diagrama seleccionado, que permite editar tanto la apariencia de cada uno de los elementos, como sus datos.
- Se han implementado propiedades a las tablas y relaciones, las cuales pueden ser modificadas desde el panel lateral mencionado anteriormente.
- La aplicación permite declarar Uniques compuestos en aquellas tablas con dos o más columnas.
- Las claves foráneas se añadirán y eliminarán de forma automática, ya sea cuando cambien las claves de las tablas asociadas, cuando cambie el tipo de relación o cuando una relación entre dos tablas se elimine.
- Se han implementado los símbolos correspondientes a la notación de patas de cuervo, con la posibilidad de cambiar entre ellos.

[3]Ejemplo desarrollado por mxGraph que se puede encontrar en https://jgraph.github.io/mxgraph/javascript/examples/schema.html

- Cuando el usuario establece una relación N:M entre dos tablas, automáticamente se genera una tabla intermedia que represente dicha relación.
- El usuario tiene la posibilidad de añadir título y descripción para cada una de las columnas de su diagrama, cuya información podrá consultar en el diccionario de datos que tiene disponible.
- Se permite tanto la descarga del fichero XML correspondiente al diagrama creado por el usuario, como la posterior importación del diagrama a partir de dicho fichero.
- La etiqueta de tablas y columnas se ajusta al ancho de estas.
- Es posible mover de posición verticalmente las columnas dentro de una tabla.
- Es posible añadir una columna a una tabla o bien utilizando el menú popup, o el atajo de teclado F4.
- El botón borrar todo de la barra de herramientas superior eliminará todos los elementos del diagrama del usuario.
- Se ha mejorado la generación de código SQL, mejorando las referencias a las claves primarias y foráneas y añadiendo el manejo de uniques compuestos, además de haber implementado la posibilidad de generar el código para SQLAlchemy, teniendo en cuenta todo lo anterior.

6. Trabajos relacionados

6.1. Artículos científicos

Basic Data Structure Models Explained With A Common Example

Este artículo escrito en 1976, se trata de la primera aparición de las *patas de cuervo*. Conocidas primero como *flechas invertidas*, su autor, Gordon C. Everest, menciona que el artículo no trata sobre la notación, aunque esta fue escogida cuidadosamente. Conocida primero como *fork* (tenedor), fue seleccionada por su facilidad para representarse mediante caracteres estándar (—<) y debido a que no implica ni direccionalidad ni un acceso físico.[4].

En este artículo no se utiliza la notación de patas de cuervo con bases de datos relacionales, ya que estas acababan de nacer, pero es la primera referencia que se tiene de dicha notación, la cual evolucionaría con el paso del tiempo para trabajar con bases de datos relacionales.

6.2. Proyectos

mxGraph Schema

Ejemplo de los creadores de la librería *mxGraph*, muestra las funcionalidades básicas para implementar una aplicación de diseño de diagramas de bases de datos. La aplicación desarrollada toma como base este ejemplo, mejorándolo y ampliándolo con la notación de patas de cuervo. En el ejemplo únicamente disponemos de relaciones direccionales sin cardinalidad y un

simple manejo de claves foráneas añadiendo la clave primaria de la tabla con la que se relaciona. En la aplicación desarrollada se han extendido las funcionalidades, tanto en las relaciones con la nueva notación, como en las columnas ampliando sus atributos.

- Web del ejemplo: https://jgraph.github.io/mxgraph/javascript/examples/schema.html

Microsoft Visio

La aplicación *Visio* de Microsoft es utilizada para el diseño de diagramas y gráficos vectoriales. Esta aplicación nos permite, entre otros, diseñar diagramas relacionales disponiendo de diferentes notaciones. Entre las notaciones que podemos utilizar encontramos la notación de patas de cuervo, aquí denominada notación de patas de gallo. Con Visio podemos diseñar nuestros diagramas, añadiendo las tablas, columnas y relaciones que necesitemos. Microsoft Visio cuenta con diferentes licencias, gratuitas y de pago, lamentablemente, la notación de patas de cuervo se encuentra únicamente disponible en la versión de pago.

- Web de la aplicación: https://www.microsoft.com/es-es/microsoft-365/visio/flowchart-software.

draw.io

draw.io es una aplicación web pensada para el diseño de multitud de diagramas. Esta aplicación, desarrollada por *JGraph* los creadores de la librería *mxGraph*, también nos permite diseñar diagramas relacionales utilizando la notación de patas de cuervo, teniendo a disposición las tablas y columnas que representarán a las entidades y sus atributos, y todas las posibles relaciones que podemos encontrar entre las tablas.

- Web de la aplicación: https://app.diagrams.net/

Visual Paradigm

Visual Paradigm cuenta con una herramienta online para diseñar infinidad de gráficos y diagramas. Entre la multitud de opciones de las que dispone, podemos diseñar diagramas relacionales utilizando la notación de patas de cuervo. Aunque las opciones que dispone son algo limitadas, es

una buena opción a considerar teniendo en cuenta la posibilidad de trabajar online y su acceso gratuito.

- Web de la aplicación: https://online.visual-paradigm.com/es/

6.3. Fortalezas y debilidades del proyecto

Características	Crow's Foot	Schema	Visio	draw.io	VP
Añadir elementos[4]	✓	✓	✓	✓	✓
Patas de cuervo	✓	×	✓	✓	✓
Claves automática	✓	×	×	×	×
Panel propiedades	✓	×	✓	✓	×
Generación SQL	✓	Parcial	×	×	×
Generación SQLAlchemy	✓	×	×	×	×
Múltiples relaciones	×	✓	✓	✓	✓
Relaciones reflexivas	×	×	✓	✓	✓
Mover columnas	✓	×	✓	✓	✓
Mover relaciones	×	×	✓	✓	✓
Modelo relacional	✓	✓	✓	✓	✓
Modelo E/R	×	×	✓	✓	✓
Plataformas	Web App	Web App	Desk App	Web App	Web App

Tabla 6.1: Comparativa de características

Principales fortalezas

- Es posible acceder a la aplicación desde cualquier dispositivo[5] que disponga de conexión a Internet.
- No necesita instalación, más allá del propio navegador con el que se acceda a la aplicación.
- El sistema gestiona automáticamente cuando tiene que añadir o eliminar claves de cada tabla.

[4]Los elementos necesarios para la notación de patas de cuervo, es decir, tablas, columnas y relaciones.

[5]Es posible acceder desde dispositivos móviles a la aplicación, aunque al no estar pensada para controles táctiles, aunque funcionan, presentan problemas.

- Se permite personalizar los elementos del diagrama visualmente, teniendo la posibilidad de cambiar color de relleno, degradado, fuente, etc.
- La posibilidad de importar y exportar el diagrama nos permite continuar con el desarrollo en cualquier momento.
- Podemos conseguir una rápida implementación de nuestros diagramas gracias a la generación de código.
- Se cumplen muchos de los requisitos de la notación de patas de cuervo de forma automática por el sistema, como añadir las claves foráneas en las tablas correspondientes con los parámetros adecuados o incluir una tabla intermedia de forma automática para las relaciones N:M.

Principales debilidades

- Los diagramas se tienen que almacenar en un fichero XML y volver a importar desde dicho fichero, lo que entorpece el diseño.
- El usuario no tiene el control total sobre las claves de sus tablas al estar estas decididas por el sistema.
- La generación de código tiene únicamente dos posibilidades.
- La aplicación está disponible únicamente en castellano, limitando el número de usuarios que pueden hacer uso de ella.

7. Conclusiones y Líneas de trabajo futuras

En este apartado se comentan las conclusiones extraídas después de completar el proyecto, además de las posibles líneas de trabajo futuras por las que se podría continuar el proyecto más adelante.

7.1. Conclusiones

Las conclusiones obtenidas después del desarrollo de la aplicación son las siguientes:

- La aplicación desarrollada cumple con el objetivo propuesto, permitiendo al usuario diseñar diagramas relacionales utilizando la notación de patas de cuervo (Crow's Foot), además de permitir al usuario importar y exportar sus diagramas en formato XML y obtener el código SQL y SQLAlchemy necesario para la creación de las tablas del diagrama.

- Se ha utilizado para todo el desarrollo la librería de Javascript *mxGraph*. Esta librería utilizada para el diseño de grafos en aplicaciones web se lleva desarrollando desde el año 2005, por lo que se puede entender la profundidad de esta. Debido a esta profundidad, aunque se ha trabajado de forma intensiva con ella, considero que aún queda mucho por aprender y mejorar. Entendiendo el control que otorgaba la librería, a lo largo del desarrollo se decidió entregarle el control de la aplicación a mxGraph y no integrar ningún framework adicional. mxGraph ha permitido una implementación fluida y con buen rendimiento.

- Durante el desarrollo se han obtenido gran cantidad de nuevos conocimientos, principalmente en el desarrollo web y el lenguaje de programación Javascript, con el que no se había trabajado a este nivel con anterioridad. Este falta de trabajo previo probablemente haya afectado a la calidad del producto final, puesto que el aprendizaje ha sido continuo durante todo el desarrollo, mejorando paso a paso.

7.2. Líneas de trabajo futuras

Como posible línea de trabajo a seguir en versiones futuras de la aplicación, se podrían incluir las siguientes funcionalidades y mejores:

- Implementar la gestión necesaria para las relaciones reflexivas, permitiendo al usuario dibujar estas.
- Introducir el manejo de múltiples relaciones entre las mismas dos tablas.
- Mejorar el manejo de claves de la aplicación, proporcionando al usuario más control sobre estas.
- Implementar un backend que permita al usuario registrarse y almacenar sus diagramas en la nube.
- Permitir la colaboración en tiempo real entre múltiples usuarios, siendo una herramienta colaborativa al estilo de *Google Docs*.
- Ampliar las posibilidades de la generación de código, incluyendo soporte para mayor número de lenguajes de bases de datos, plataformas y librerías diferentes.
- Incluir una sección de *Ayuda*, que guíe al usuario en sus primeros pasos en la aplicación, además de una sección de *Soporte*, en caso de que algo salga mal.
- Introducir la IA en la aplicación, proporcionando al usuario una guía en tiempo real mientras desarrolla su diagrama, así como validación de que el diagrama es correcto.
- Ampliar la variedad de formatos en la que se pueden importar y exportar los diagramas, fomentando la compatibilidad con otras herramientas.

- Traducir la aplicación a diferentes idiomas, consiguiendo de esta forma que sea más accesible.

Bibliografía

[1] Atlassian. Flujo de trabajo gitflow. https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow, 2024. [Internet; descargado 25-mayo-2024].

[2] Patricia Dybka. Crow's foot notation. https://vertabelo.com/blog/crow-s-foot-notation/, 2016. [Internet; descargado 16-mayo-2024].

[3] Patricia Dybka. Crow's foot notation in vertabelo. https://vertabelo.com/blog/crow-s-foot-notation-in-vertabelo/, 2024. [Internet; descargado 25-mayo-2024].

[4] Gordon C. Everest. Basic data structure models explained with a common example. *IEEE Computer Society*, 1976.

[5] Dolores; Castro Elena Iglesias, Ana Maria; Cuadra. *Desarrollo de bases de datos*. RA-MA Editorial, 2014.

[6] Juanjo. Qué es sonarcloud y cómo mejora la calidad de tu código. https://platzi.com/blog/sonarcloud-mejora-codigo-sast/, 2022. [Internet; descargado 20-mayo-2024].

[7] mxGraph. Api specification. https://jgraph.github.io/mxgraph/docs/js-api/files/index-txt.html, 2024. [Internet; descargado 22-mayo-2024].

[8] Visual Paradigm. What is an entity relationship diagram (erd). https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/, 2024. [Internet; descargado 14-mayo-2024].

[9] Wikipedia. Git. https://es.wikipedia.org/wiki/Git, 2024. [Internet; descargado 20-mayo-2024].

[10] Wikipedia. Github. https://es.wikipedia.org/wiki/GitHub, 2024. [Internet; descargado 20-mayo-2024].

[11] Wikipedia. Relational database. https://en.wikipedia.org/wiki/Relational_database, 2024. [Internet; descargado 7-abril-2024].

[12] Wikipedia. Relational model. https://en.wikipedia.org/wiki/Relational_model, 2024. [Internet; descargado 7-abril-2024].

[13] Proyectos Ágile. Qué es scrum. https://proyectosagiles.org/que-es-scrum/, 2024. [Internet; descargado 20-mayo-2024].