DBCASE 3.0

TRABAJO FIN DE GRADO
CURSO 2022-2023

AUTORES
ARTURO IBÁÑEZ MARTÍNEZ
MIGUEL DERECHO PRIETO

DIRECTOR
FERNANDO SÁENZ PÉREZ

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

"El producto más valioso que conozco es la información." – Gordon Gekko, *Wall Street*

AGRADECIMIENTOS

A nuestros amigos de carrera. Juntos hemos recorrido este largo camino que empezó allá por el año 2017, viviendo tanto los momentos duros como los excelentes. ¡Qué bien lo hemos pasado!

A Fernando, por su ayuda, implicación y tiempo invertido en el proyecto, así como por iniciarnos en el mundo de las bases de datos durante la asignatura correspondiente del año 2019.

Resumen

DBCASE 3.0

El proyecto que aquí se expone se corresponde con el Trabajo de Fin de Grado realizado por los alumnos Arturo Ibáñez Martínez y Miguel Derecho Prieto durante el curso académico 2022/2023, bajo la tutela del profesor Fernando Sáenz Pérez. El trabajo consiste en continuar con el desarrollo de la aplicación de escritorio DBCASE, correspondiéndose esta con la versión 3.0 de la herramienta. Las tareas fundamentales del proyecto son las relativas a la corrección de los errores y excepciones existentes en las versiones previas, así como ampliaciones de las funcionalidades de la aplicación. En esta memoria se detallan las diferentes tareas completadas, junto con otros aspectos importantes relacionados con el aplicativo, como pueden ser las distintas utilidades de la herramienta, la base teórica en la que se ha basado el proyecto, su evolución a lo largo de los años o la estructura que presenta el código implementado. Finalmente, se abordarán las conclusiones del proyecto, en las que se incluyen aspectos relacionados con posibles trabajos futuros sobre DBCASE. Adicionalmente, se incluye como anexo un manual de usuario para ayudar a comprender perfectamente el funcionamiento de la herramienta. Como se podrá observar, esta memoria está también orientada a ayudar a futuros alumnos que continúen con el proyecto en los próximos años.

Palabras clave

Bases de datos, Java, Modelo Entidad-Relación, Modelo Vista-Controlador, Traducción, Funcionalidades, Corrección de errores, Modelo conceptual, Modelo lógico, Modelo físico.

Abstract

DBCASE 3.0

The project presented here corresponds to the Final Degree Project carried out by the students Arturo Ibáñez Martínez and Miguel Derecho Prieto during the 2022/2023 academic year, under the supervision of Professor Fernando Sáenz Pérez. The work consists of continuing with the development of the DBCASE desktop application, this being this the 3.0 version of the tool. The main tasks of the project are those related to the correction of errors and exceptions existing in previous versions, as well as extensions to the application's functionalities. This report details the different tasks completed, as well as other important aspects related to the application, such as the different uses of the tool, the theoretical basis on which the project has been based, its evolution over the years, or the structure of the implemented code. Finally, the conclusions of the project will be discussed, including aspects related to possible future work on DBCASE. In addition, a user manual is included as an appendix in order to help to understand how the tool works. As can be seen, this report is also focused on helping future students who will continue with the project in the coming years.

Keywords

Database, Java, Entity-Relationship Model, Model-View-Controller, Translation, Functionalities, Correction of errors, Conceptual Model, Logical Model, Physical Model.

ÍNDICE DE CONTENIDOS

Índice de figuras

Capítulo 1 - Introducción

Una base de datos es un conjunto organizado y estructurado de información que permite un acceso, actualización y administración eficiente de los datos. Las distintas funcionalidades y aplicaciones de las bases de datos en el mundo real son prácticamente infinitas, estando presentes en un gran número de sistemas informáticos.

Debido a los importantes avances producidos en el análisis y tratamiento de información, así como los beneficios que conllevan estas prácticas, las bases de datos suponen una herramienta fundamental para infinidad de ámbitos distintos. Entre muchos de los posibles campos donde se utilizan grandes cantidades de información y, en consecuencia, se potencia la importancia de las bases de datos, destacan:

- La investigación científica, para predecir y estudiar distintos comportamientos relacionados con fenómenos meteorológicos, estudios demográficos, epidemiología, etc.
- Sistemas de información empresarial, para registrar y gestionar las actividades realizadas, trabajadores, activos, etc.
- Aplicaciones web, como Google, para almacenar y proporcionar información de manera rápida.

En la Facultad de Informática de la Universidad Complutense de Madrid existen gran cantidad de asignaturas relacionadas con el tratamiento y optimización de datos. En este trabajo nos centraremos en aspectos relacionados con la asignatura *Bases de datos*, fundamental para el entendimiento y acercamiento de los alumnos a los distintos métodos existentes para diseñar, acceder y gestionar una base de datos real. Concretamente, nos centraremos en las bases de datos relacionales, es decir, basadas en diagramas Entidad-Relación.

Para ello, se ha continuado con el trabajo realizado por el profesor Fernando Sáenz Pérez, junto con distintos alumnos de la facultad, sobre la aplicación de escritorio DBCASE. Esta herramienta reúne las tres fases claves para la implementación de una base de datos relacional, permitiendo al usuario diseñar mediante diagramas el modelo conceptual y traducir automáticamente su funcionamiento a los modelos lógico y físico.

1.1 Motivación

La motivación personal principal del trabajo se corresponde en extender los conocimientos sobre bases de datos adquiridos durante la asignatura de *Bases de datos.* De este modo, a través del desarrollo y corrección de errores de la herramienta DBCASE se complementan y refrescan los aspectos aprendidos durante la asignatura, concretamente sobre el diseño conceptual de un base de datos relacional y las correspondientes traducciones a los modelos lógico y físico. Además, se ha podido también estudiar aspectos adicionales relacionados con la normalización y el análisis del rendimiento.

DBCASE son las siglas de Database Computer-Aided Software Engineering. La herramienta constituye la continuación de la aplicación DBDT (Database Design Tool). Esta nació en el año 2008 de la mano de Yolanda García Ruiz, continuando Fernando Sáenz Pérez en los años siguientes, ya bajo la denominación de DBCASE. Este trabajo constituye la versión 3.0 de la aplicación de escritorio.

Mediante una interfaz gráfica intuitiva e interactiva, permite al usuario diseñar el comportamiento de una base de datos mediante diagramas Entidad-Relación. Además, a partir del esquema conceptual creado, muestra las correspondientes traducciones a los modelos lógico, pudiendo corroborar además la corrección del diseño del esquema conceptual; y físico, generando el código asociado a la creación de cada tabla.

Esta herramienta está enfocada al ambiente académico, facilitando a los estudiantes el entendimiento de la teoría de bases de datos a partir de casos prácticos. No obstante, su uso no se centra únicamente en este ámbito, la capacidad de generación del código SQL correspondiente permite la ejecución de scripts reales en distintos sistemas de gestión de bases de datos (SGDB), o *Database Management System* (DBMS), como son Oracle, MySQL o Microsoft Access.

1.2 Objetivos

Como se introdujo anteriormente, este proyecto constituye la versión 3.0 de la aplicación de escritorio DBCASE, herramienta utilizada para el diseño e implementación de las bases de datos relacionales.

El objetivo principal del trabajo es el de corregir distintos errores y excepciones existentes en las versiones anteriores de la herramienta, además de incluir nuevas funcionalidades, enriqueciendo aún más las posibilidades de la aplicación.

A continuación, se listan algunas de las tareas marcadas inicialmente como objetivos, relacionadas tanto con correcciones como con extensión de funcionalidades:

- Revisar la generación del esquema relacional, con la intención de identificar los posibles errores y excepciones que puedan existir.
- Incluir las agregaciones en la herramienta.
- Añadir las cadenas deshacer y rehacer (junto con los atajos de teclado Ctrl+Z y Ctrl+Y).
- Añadir una lista de archivos recientes para abrir.
- Se tiene identificado el siguiente error, siendo el objetivo solucionarlo: al añadir una entidad hija a la relación IsA salta un mensaje de error, sin embargo, el proceso crea un enlace entre la relación IsA y la entidad hija seleccionada.
- Tratar de incluir la flecha que representa la restricción de cardinalidad 1 en relaciones recursivas.
- Permitir usar las teclas *Y* y *N* en los cuadros de dialogo Yes/No.
- Introducir un submenú en Menú Help que permita visualizar un manual de usuario.
- Introducir la reingeniería del diseño físico/relacional. A partir de las tablas creadas en los esquemas lógico y/o físico, mostrar el esquema conceptual correspondiente.
- Identificar y resolver los posibles errores y excepciones que puedan existir a la hora de realizar acciones. Por ejemplo, al hacer doble clic sobre elementos, se abre un cuadro para editarlos, pero en algunas ocasiones no se modifican correctamente.

- Permitir especificar manualmente el nivel de zoom del panel del esquema conceptual.
- Permitir incluir zoom en los esquemas lógico y físico.
- Introducir un nuevo menú Edición, con los submenús Copiar, Cortar, Pegar, Deshacer y Rehacer. Incluir también los atajos correspondientes: Ctrl+C, Ctr+V, Ctrl+Z y Ctrl+Y.
- Añadir elementos de menú para todas las acciones. En particular para las acciones del menú contextual y para los botones de la generación de código.

Cabe destacar que conforme se fue avanzando en el proyecto, se fueron identificando diversos errores existentes, cuya solución se consideró prioritaria.

Los objetivos que no se han podido completar se encuentran listados en la sección **3.3.3 Tareas sin concluir y mejoras**, junto con una explicación de las acciones realizadas en el intento de implementarlas, así como una serie de consideraciones e indicaciones de cómo se podrían implementar.

Adicionalmente, el objetivo de la memoria, además de por supuesto mostrar los aspectos realizados, es ayudar a futuros alumnos que continúen con el proyecto a entender el funcionamiento de la aplicación y del código, así como las distintas consideraciones que recomendamos a la hora de organizar el trabajo.

1.3 Plan de trabajo

En esta sección se expone el plan de trabajo a seguir para la consecución de los objetivos indicados anteriormente. Este plan de trabajo se detalla en profundidad en la sección **3.1 Cronología del trabajo**. Los distintos pasos seguidos fueron:

- Contacto con Fernando Sáenz Pérez, tutor del TFG.
- Repaso y estudio de la teoría correspondiente a la asignatura de *Bases de datos*, relacionada con el diseño conceptual de una base de datos relacional y la correspondiente traducción al modelo lógico, así como de procesamiento de consultas. Además, se inspeccionó también teoría fuera del ámbito de la asignatura, relacionada con normalización y análisis de rendimiento.

- Obtención del código y primeros contactos con la interfaz de la aplicación.
- Identificación de tareas prioritarias y división de trabajo.
- Análisis y entendimiento del código.
- Organización entre los alumnos para la compartición de código y fusionar los avances independientes.
- Desarrollos sobre la aplicación y comunicación con el tutor.
- Escritura de esta memoria.

Se adjunta a continuación un diagrama de Gantt donde se muestra la evolución de las distintas etapas del proyecto:

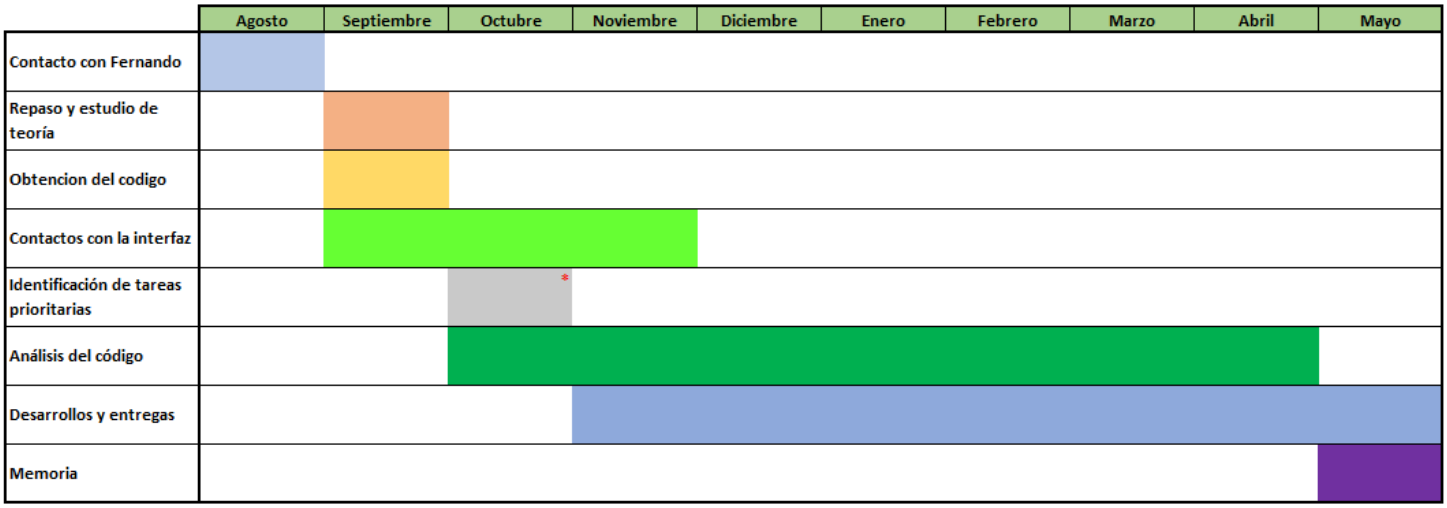

Figura 1. Evolución del plan de trabajo

1.4 Estructura de la memoria

Una vez introducidas las cuestiones previas sobre la importancia y utilidades de las bases de datos, así como la motivación y objetivos del trabajo, donde se describe y expone la herramienta DBCASE, se van a detallar los distintos capítulos que componen la presente memoria.

En primer lugar, en el Capítulo 2 se exponen los antecedentes que componen el estado de la cuestión del proyecto. En este sentido, se introducen las bases teóricas necesarias para poder comprender la asignatura de *Bases de datos* y poder acometer las diferentes tareas de las que se compone el trabajo. Además, al corresponderse este proyecto con la versión 3.0 de la herramienta DBCASE, se realiza un breve resumen de las distintas versiones y *antepasados* de la aplicación, incluyendo su herramienta web análoga DBCASEWeb. El capítulo se cierra exponiendo las diferentes tecnologías empleadas para la implementación, desde los frameworks y herramientas de comunicación y compartición de código utilizadas, hasta las librerías y lenguajes de programación que componen el desarrollo del código.

En el Capítulo 3 nos centraremos en el proyecto en sí. Este capítulo, además de incluir las distintas tareas implementadas, está orientado a ayudar a posibles alumnos que continúen con el trabajo en los años venideros. En este sentido, tras introducir la cronología del trabajo seguida, se muestra una explicación de la estructura del código de la aplicación. A continuación, se detallan las distintas tareas realizadas, divididas en tareas de corrección de errores y ampliación de funcionalidades, junto con una breve descripción de aquellos aspectos en los que se ha trabajado, pero no acaban de ser funcionales.

Por último, se cierra la memoria exponiendo las conclusiones finales del proyecto, así como indicaciones sobre el trabajo futuro a realizar sobre la aplicación.

A modo de anexo, se incluye un manual de usuario de la herramienta con el objetivo de ayudar a entender el comportamiento y funcionalidades de la interfaz de la herramienta.

Capítulo 2 - Estado de la cuestión

2.1 Base teórica del proyecto

En esta sección buscamos introducir la base teórica sobre la que nos hemos apoyado para la realización de las tareas. El objetivo principal se corresponde con facilitar el funcionamiento de la aplicación a posibles usuarios, en el sentido de que se comprenda cada aspecto de la herramienta, tanto relacionado con el diseño, como con las traducciones.

Como se comentó anteriormente, el proyecto está altamente relacionado con la asignatura de *Bases de datos*, por lo que la teoría presente en esta sección se asemeja bastante a la introducida en la asignatura.

Concretamente, nos centraremos en los aspectos necesarios para el correcto entendimiento de DBCASE, que se corresponden con el diseño conceptual basado en diagramas Entidad-Relación, así como las correspondientes representaciones en los modelos lógico y físico.

La bibliografía utilizada se corresponde con las diapositivas de la asignatura proporcionadas por el tutor[1].

2.1.1 Modelo conceptual

El modelo conceptual es una representación abstracta de alto nivel de la estructura de una base de datos que proporciona una visión general sobre los elementos existentes, sus características, y cómo se relacionan entre sí.

Existen diferentes modelos de representación, como diagramas de clases UML, diagramas de flujo de datos, o los diagramas Entidad-Relación, que se tratan de los empleados durante todo el proyecto.

Un modelo Entidad-Relación se caracteriza por ser conciso, eficiente y fácil de comprender y mantener. Los principales aspectos a evitar son la redundancia (existencia de información repetida), la incompletitud (aspectos no modelados) y la inexactitud (aspectos mal modelados).

Cabe destacar que un diseño Entidad-Relación no es único ni directo. A una misma especificación le pueden corresponder diversos diseños.

En un modelo Entidad-Relación, se representan los conceptos y relaciones clave de la base de datos mediante entidades, atributos y relaciones.

- **Entidades** (figura 2): elemento (concreto o abstracto) que existe en el mundo real que puede ser identificado y descrito en términos de sus características. Son representados en el diagrama a través de rectángulos.

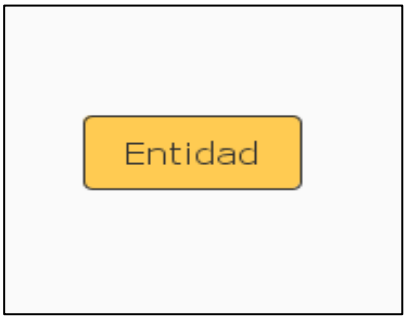

Figura 2: Entidades en DBCASE

- **Relaciones** (figura 3): representan las conexiones entre entidades. No tienen existencia propia en el mundo real, pero son necesarias para reflejar las interacciones existentes. En el diagrama se muestran como rombos, y se conectan a entidades a través de líneas.

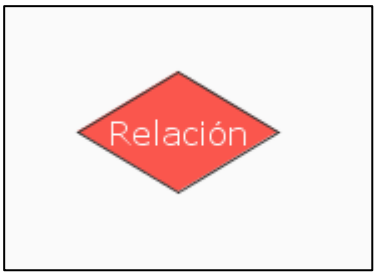

Figura 3: Relaciones en DBCASE

Se pueden clasificar en función de su grado, es decir, del número de entidades que conectan:

- Relación binaria: asocia dos entidades.
- Relación ternaria: asocia tres entidades. En general, pueden asociar más de tres entidades.
- Relación recursiva: asocia una entidad consigo misma.

La función que desempeña una entidad en una relación se denomina rol. En general, los roles están implícitos y no se suelen especificar, pero resultan

útiles para identificar claramente la naturaleza de una relación. Los roles son necesarios en relaciones recursivas para diferenciar las entidades que participan en ellas.

- **Atributos** (figura 4): características de una entidad o relación. Tienen asociado un dominio para representar su tipología (fecha, número, cadena de caracteres, etc.). Se representan en el diagrama a través de óvalos y se unen a otros elementos a través de líneas.

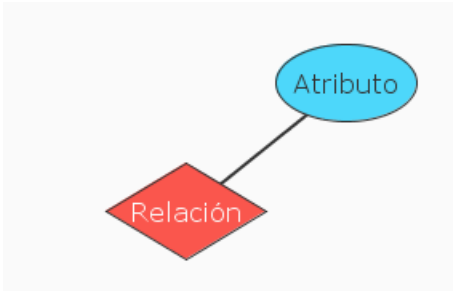

Figura 4: Atributos en DBCASE

Los atributos pueden anidarse para representar propiedades de otras características. En este caso se dice que un atributo compuesto tiene subatributos.

A su vez, se dividen en:

- Atributos monovalor: sólo pueden contener un único valor.
- Atributos multivalor (figura 5): pueden contener más de un valor. Se representan en el esquema como una doble elipse concéntrica.

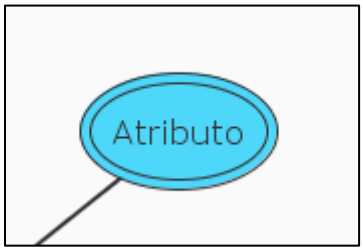

Figura 5: Atributo multivalor

Si un atributo puede ser nulo, se representa con un asterisco al lado del nombre, mientras que, si puede obtenerse mediante combinaciones de otros atributos, se dice que es derivado y se une al elemento al que pertenecen mediante una línea discontinua.

Un atributo con restricción de unicidad (*Unique*) garantiza que los valores en ese atributo sean únicos en toda la tabla.

Un aspecto importante a la hora de diseñar una base de datos es tener en cuenta las restricciones existentes sobre los datos. Las restricciones son reglas o condiciones que se aplican a los elementos del modelo para ampliar el nivel de especificación. Pueden asociarse a un tipo de entidad, relación o atributo; por ejemplo, la unicidad indicada anteriormente. Existen tres tipos principales de restricciones:

- **Restricciones de cardinalidad:** es un tipo de restricción binaria estructural que limita el número máximo de instancias de una entidad asociadas a una ocurrencia dada de la otra entidad. En las relaciones ternarias, la cardinalidad de una entidad se calcula respecto de las otras dos entidades. A su vez, existen distintos tipos de cardinalidad:
    - Uno a uno (1:1) (figura 6): una instancia de la entidad A se relaciona con una única instancia de la entidad B, y viceversa. En el diagrama, si la entidad A figura con cardinalidad 1 en una relación, se representa con una línea con flecha, apuntando a la entidad.

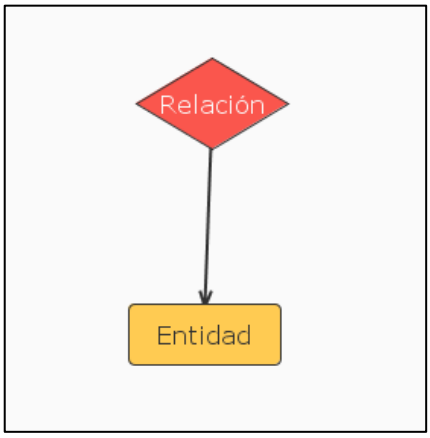

Figura 6: Cardinalidad 1

    - Uno a varios (1:N): una entidad A se relaciona con varias instancias de otra entidad B, pero cada ocurrencia de la entidad B sólo le corresponde una instancia de la entidad A. En el caso de cardinalidad a varios, se considera que no existe restricción y se representa en el diagrama con una línea sin flecha.

    - Varios a varios (N:N): a cada ocurrencia de la entidad A le pueden corresponder varias ocurrencias de la entidad B, y viceversa.
- **Restricciones de participación:** tipo de restricción binaria estructural que se corresponde con el número mínimo y máximo de instancias de una entidad, asociadas a una ocurrencia de la otra entidad. La participación (mínima y máxima) se escribe entre paréntesis en el lado de la entidad a la que corresponda. Es necesario describir la participación mínima y máxima de todas las entidades involucradas en una relación. El concepto de participación se asocia con los términos de "obligatoriedad" y "opcionalidad". Dada una relación en la cual participan un conjunto de entidades, la participación mínima puede ser de dos tipos:
    - Total (figura 7): cada entidad participa en al menos una instancia de la relación. Se puede describir en el diagrama únicamente con la notación (1,N), o a través de una doble línea entre la entidad con participación total y la relación asociada.

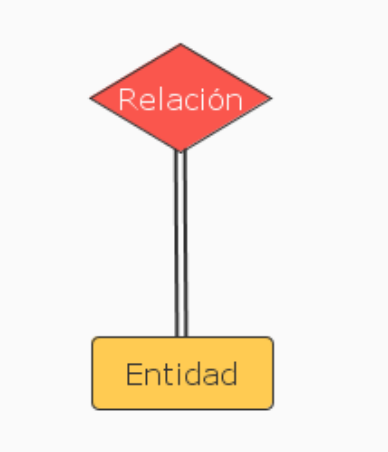

Figura 7: Participación total

    - Parcial: existe una entidad que no participa en ninguna instancia de la relación. En este caso, se considera que no existe restricción y se representa en el diagrama con una línea normal.
- **Unicidad:** las instancias de entidades deben distinguirse unas de otras a través de los valores de sus atributos. Interesa encontrar un conjunto de atributos lo más pequeño posible que permita distinguir entidades. Se llama superclave a cualquier conjunto de atributos que permita distinguir

a todas las instancias de cualquier entidad. Una clave candidata es una superclave que no contiene ningún atributo o conjunto de ellos sin el que también forme superclave; es decir, se corresponde con el conjunto mínimo de atributos que puede identificar una instancia. La clave primaria es la clave candidata finalmente seleccionada por el diseñador para distinguir entre las instancias. En los diagramas Entidad-Relación, se representa una clave primaria subrayando los atributos que la componen.

Relacionado con las claves primarias, se introduce el concepto de **entidad débil** (figura 8), que es un tipo de entidad que no tiene suficientes atributos para formar una clave primaria. Una entidad débil necesita los atributos de otra entidad, denominada fuerte, para poder formar su clave primaria y así poder identificar sus tuplas de manera única. Las entidades débiles se representan en el diagrama mediante rectángulos dobles, y la relación que permiten asociar las entidades fuerte y débil, mediante un doble rombo. La clave de la entidad débil se denomina clave parcial y se representa con un subrayado discontinuo.

En la realidad, existen bases de datos con representaciones más complejas de

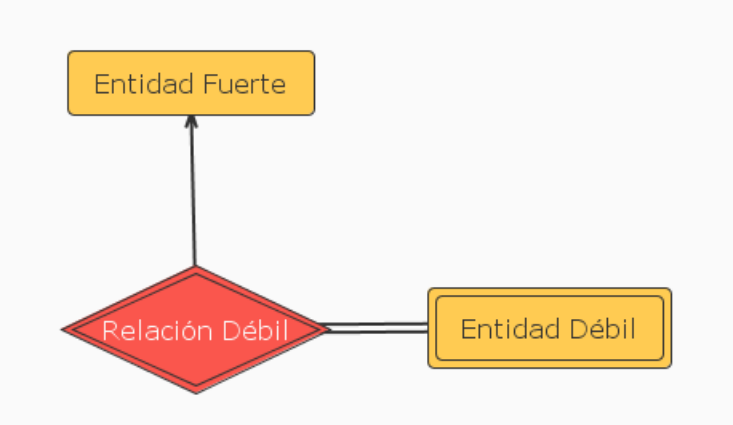

Figura 8: Entidades débiles

la información. De esta forma, existe una extensión del modelo Entidad-Relación, el modelo EER (Entidad-Relación Extendido), que se utiliza para diseñar bases de datos que requieren una mayor complejidad en la representación de las relaciones entre las entidades. Los aspectos añadidos más importantes se corresponden con:

- **<u>Generalización/Especialización</u>** (figura 9): permite modelar relaciones entre entidades que comparten atributos comunes. Una entidad E es una generalización (entidad padre) de una entidad F (especialización / entidad hija) cuando los atributos de E están incluidos en los atributos de F. Se representan con un triángulo que incluye el texto *IsA*.

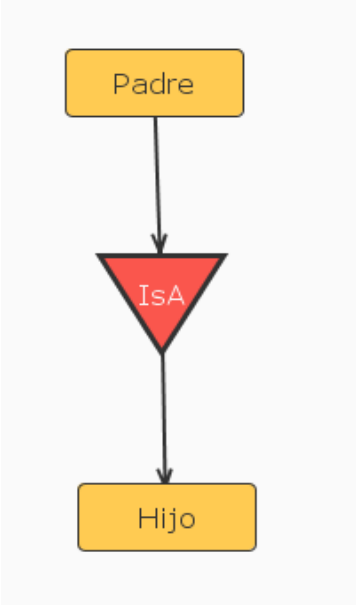

Figura 9. Generalización/Especialización

- **<u>Agregación</u>** (figura 10)**:** consiste en agrupar una relación y sus entidades participantes, como si fueran una única entidad. Generalmente, se representa con un cuadrado que engloba todos los elementos de la agregación. En DBCASE se muestran las agregaciones en el panel de elementos.

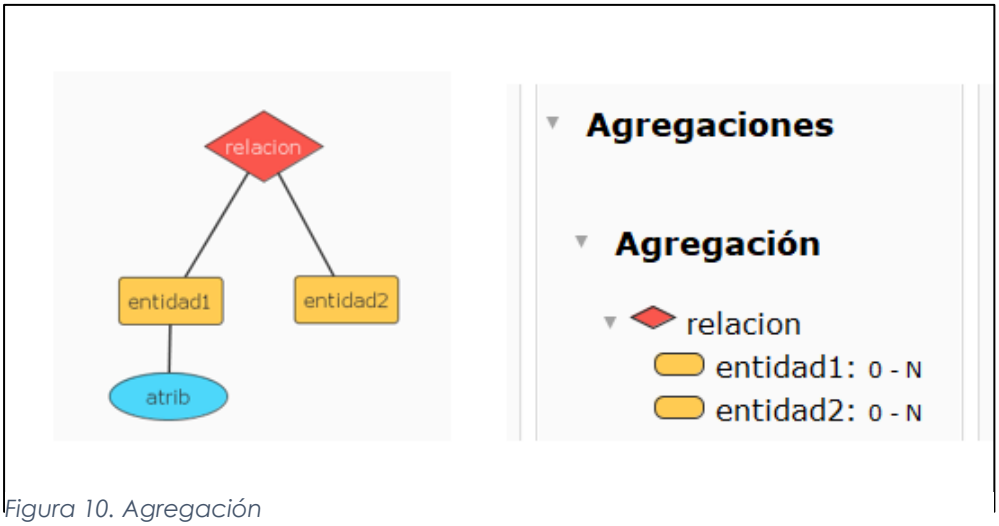

Figura 10. Agregación

2.1.2 Modelo lógico

Se corresponde con la representación del modelo conceptual introducido anteriormente, a través del uso de tablas. De este modo, para generar este modelo se traduce o traspasa la información de cada elemento especificada en el modelo Entidad-Relación. Nuevamente, cabe destacar que este diseño no es único, existiendo distintos modelos que representan un mismo diagrama Entidad-Relación. A continuación, detallaremos los mecanismos concretos para traducir la información de cada elemento existente en el diagrama Entidad-Relación.

- **Entidades**: para cada entidad que no sea débil se crea una tabla con el mismo nombre y conjunto de atributos. La clave primaria será la del diagrama ER.
- **Entidades débiles**: se transforma en una tabla que incluye como atributos aquellos que corresponden a las claves primarias de las entidades fuertes asociadas, junto con los atributos de la propia entidad débil. La clave primaria de la tabla se forma con las claves primarias de las entidades de las que depende, junto con la clave parcial de la entidad débil.
- **Relaciones (N:N)**: para cada tipo de relación con restricción de cardinalidad N a N se crea una tabla cuyos atributos son los atributos de la propia relación junto con atributos los atributos que componen las claves primarias de las entidades que participan en la relación. La clave primaria de la tabla se corresponde con la concatenación de las claves primarias de las entidades que la forman. En algunos casos es necesario renombrar atributos para evitar tener varios con el mismo nombre.
- **Relaciones (1:N)**: para cada tipo de relación con restricción de cardinalidad 1 a N existen dos posibles soluciones:
    - Propagación de clave: consiste en pasar los atributos de la relación y la clave principal de las entidades que tiene la cardinalidad máxima 1 a las que tiene cardinalidad máxima N.
    - Transformar la relación en una tabla como si se tratara de una relación N:N, pero con clave primaria la de las entidades que

tienen cardinalidad N. Esta es la solución implementada en DBCASE.

- **Relaciones (1:1)**: son un caso particular de las relaciones (1:N), por lo que se pueden aplicar las dos soluciones anteriores:
    - Propagación de clave en ambos sentidos. Si solo una de las entidades tiene participación parcial será dicha entidad la que propaga su clave.
    - Transformar la relación en una tabla y elegir como clave primaria la de alguna de las dos entidades con cardinalidad 1. En este caso, las claves primarias de las entidades no seleccionadas figuran como restricción perdida, y así se indica en DBCASE.
- **Atributos multivalor:** se crea una tabla nueva para el atributo multivalor que incluirá como atributos la clave primaria de la entidad (como clave externa) y un atributo monovalor para representar valores individuales del atributo multivalor a clave primaria de la tabla será la conjunción de la clave externa y el atributo monovalor.
- **Generalizaciones**: en primer lugar, la relación *IsA* no se traduce en una tabla. Para las entidades, existen diversas soluciones:
    - Reducir a una sola tabla correspondiente con la entidad padre. Los atributos se corresponden con los de la entidad padre, junto con atributos identificadores de las entidades hijos (su nombre o algún atributo si tuviera). La clave primaria se corresponde con la de la entidad padre.
    - Utilizar una tabla por cada entidad hija, repitiendo en cada una los atributos comunes. Cada entidad hija hereda los atributos y clave primaria de la entidad padre.
    - Crear tablas para la entidad padre y las entidades hijas. En este caso, las entidades hijas sólo heredan la clave primaria del padre, pues los atributos ya están representados en la tabla de la entidad padre. Solución implementada en DBCASE. Nuevamente, suele ser necesario renombramiento de atributos.
- **Agregaciones**: las tablas de las entidades dentro de la agregación se construyen de la manera habitual. Para cada agregación se construye una tabla (correspondiente a la que sería la tabla de la relación

contenida en la agregación) con los atributos de las entidades y la propia relación que contiene. La clave primaria se construye como concatenación de las claves primarias de las entidades y la relación que contiene. Una relación que une una agregación con otra entidad se construye de manera habitual, salvo que solo incorpora de la agregación la clave primaria de la relación sobre la que se creó.

Además, un aspecto importante en el diseño lógico de una base de datos son las **restricciones de integridad referencial**. Este tipo de restricciones aparecen cuando una tabla incorpora claves externas o foráneas, es decir, pertenecientes a otras tablas. De este modo, para poder garantizar la coherencia y la integridad de los datos, una instancia de una tabla que contiene una clave externa se debe corresponder a una instancia de otra tabla que contiene la clave asociada.

Por último, es bien sabido que al pasar del modelo conceptual al lógico se suele perder información. Esto se debe a que los modelos presentan diferentes de abstracción. Mientras que el diseño conceptual representa a alto nivel los datos y las relaciones entre ellos, el diseño lógico proporciona una representación más detallada que describe cómo se estructuran y almacenan los datos en la base de datos.

Para hacer referencia a la información que se pierde en el modelo lógico se suele utilizar el término **restricciones perdidas**. Debido a la manera en la que esta implementada la herramienta DBCASE, las restricciones perdidas se corresponden con:

- Restricciones de participación máxima y mínima
- Restricciones de participación total
- Restricciones de unicidad (claves candidatas en relaciones 1:1)
- Restricciones de cardinalidad

2.1.3 Modelo físico

El modelo físico de una base de datos es una representación detallada y concreta de cómo se implementará y almacenará la base de datos en un sistema de gestión de bases de datos. En DBCASE se obtiene traduciendo los

elementos del esquema conceptual en modo de tablas, que se construyen mediante introducción de consultas básicas.

DBCASE incorpora los siguientes gestores de bases de datos:

- **MySQL**
- **Oracle**
- **Microsoft Access via MDB:** formatos de archivo MDB (*Microsoft Database*), que almacenan los datos en una base de datos relacional.
- **Access via ODCB**: conexión a través del estándar de conectividad ODBC (Open Database Connectivity).

Las consultas a introducir para la traducción al modelo físico se corresponden con consultas básicas del lenguaje SQL. Estas consultas difieren, en cuanto a la estructura, para cada uno de los gestores considerados, pero el funcionamiento es común en todos. Se detallan a continuación las consultas necesarias para la traducción:

- **DROP TABLE** o **DROP TABLE IF EXIST**: Eliminación de una tabla para evitar sobreescritura de los datos.
- **CREATE TABLE**: creación de una tabla. Incluye entre paréntesis los atributos junto con su dominio.
- **ALTER TABLE:** modificación de una tabla. Se utiliza para añadir claves primarias (**PRIMARY KEY**) y externas (**FOREIGN KEY**). Para las claves externas se introduce el comando **REFERENCES,** para referenciar a la clave foránea correspondiente. En los gestores Oracle y Access las claves primarias y externas se añaden como restricciones, mediante el comando **ADD CONSTRAINT**. También se permiten introducir restricciones de unicidad, mediante la opción **UNIQUE,** precedida de **ADD CONSTRAINT**.

Una vez explicados los tres modelos, incluimos un ejemplo de su representación y funcionamiento en DBCASE. En el manual de usuario presente en el anexo adjunto se profundizará en la explicación de los modos de uso y distintas funcionalidades de la herramienta.

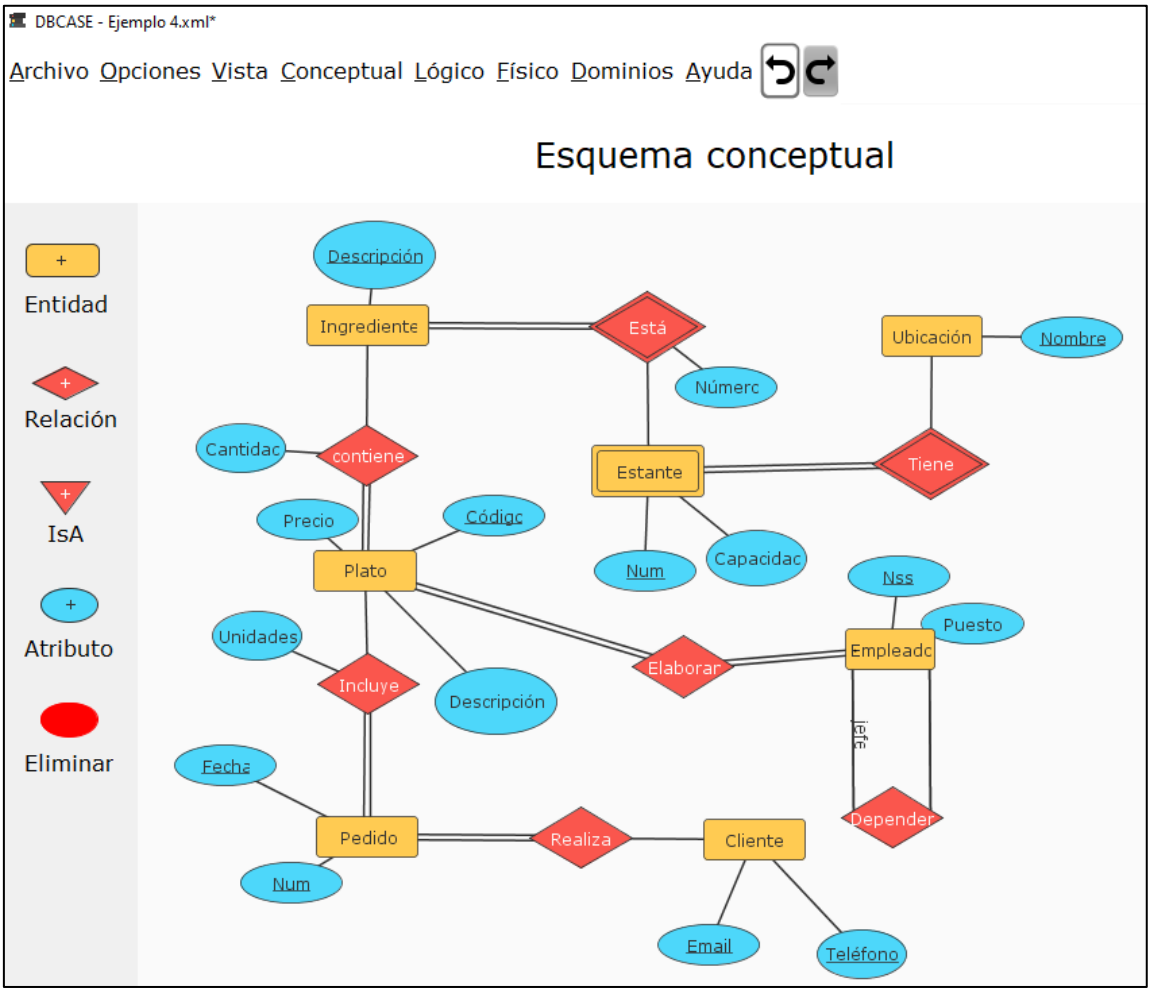

Figura 11: Esquema conceptual. Ejemplo

Figura 12: Tablas modelo lógico. Ejemplo

Figura 13: Restricciones de integridad referencial modelo lógico. Ejemplo

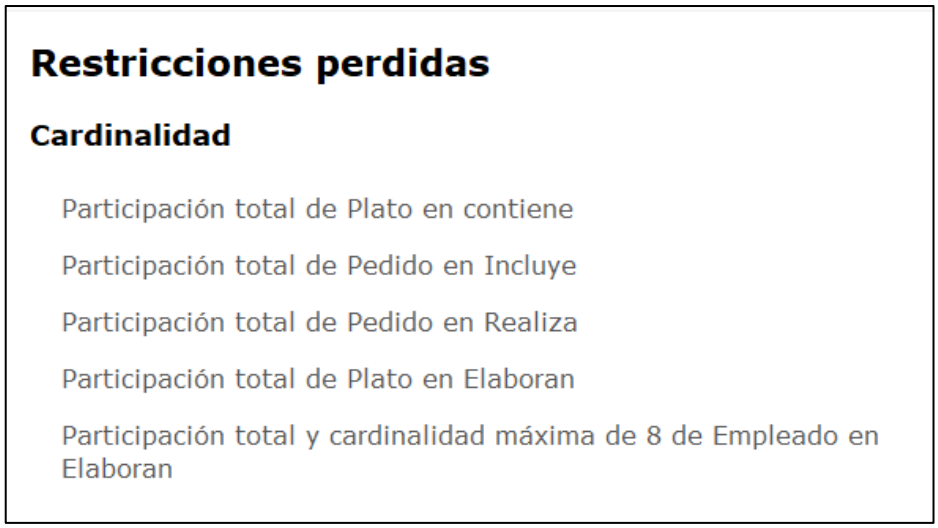

Figura 14. Restricciones perdidas modelo lógico. Ejemplo

Figura 15: Tablas modelo físico I (MySQL). Ejemplo

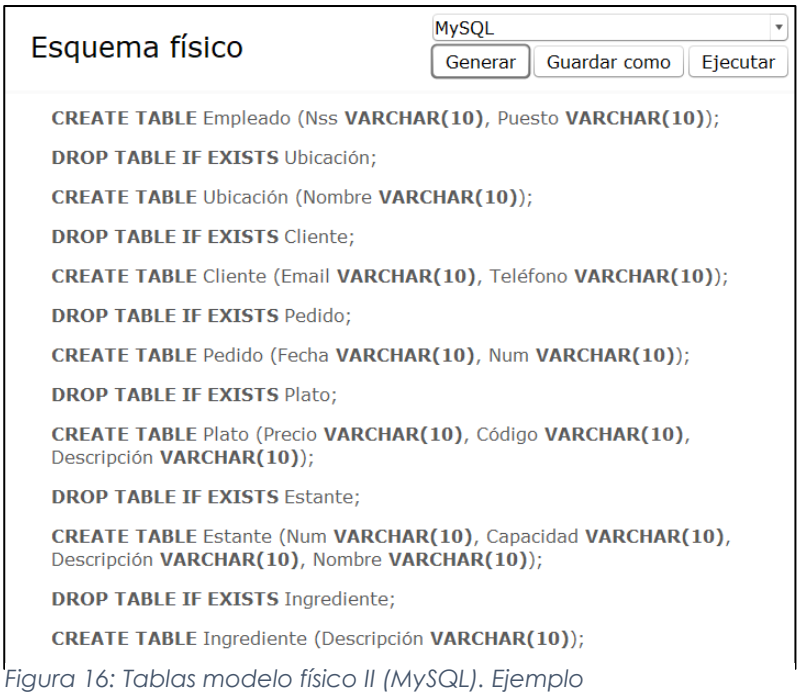

Figura 16: Tablas modelo físico II (MySQL). Ejemplo

Figura 17: Claves modelo físico I (MySQL). Ejemplo

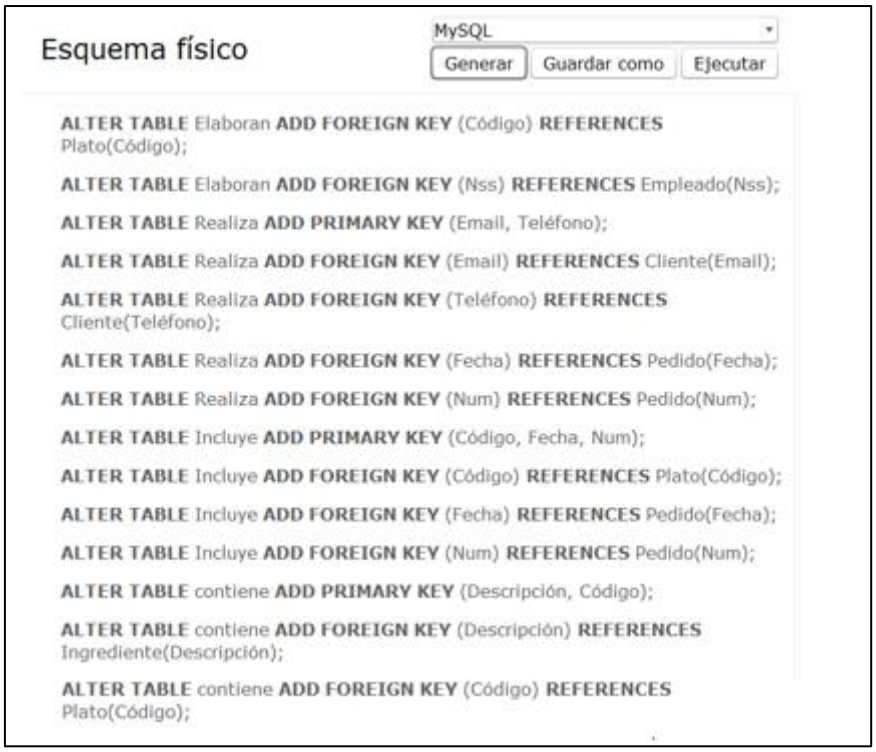

Figura 18: Claves modelo físico II (MySQL). Ejemplo

2.2 DBCASE a través de los años

El proyecto hoy conocido como DBCASE surgió en el año 2008 de la mano de Yolanda García Ruiz, con el nombre DBDT, como se ha comentado al inicio de este documento. Varios alumnos participaron en el desarrollo de la aplicación, sus nombres son Alberto Millán Gutiérrez, Miguel Martínez Segura y Francisco Javier Cáceres González. Se instauró así este proyecto como una opción para los alumnos de la facultad de informática de la Universidad Complutense de Madrid para realizar su proyecto de la por entonces asignatura *Sistemas Informáticos*, debiendo añadir funcionalidades y corregir los defectos encontrados hasta el momento y durante la realización de las pruebas del código nuevo. Al año siguiente, ya bajo la dirección de Fernando Sáenz Pérez, adoptó el nombre DBCASE. Rodrigo Denis Cepeda Mateos, Cristina Marco de Francisco y Tello Serrano Gordillo perfilaron la interfaz que conocemos hoy en día.

A partir del curso 2016-17 se incorporó el proyecto como opción para la realización del Trabajo de Fin de Grado de los alumnos. En el año 2019 el trabajo de Miguel Arriba García dio lugar a la versión DBCASE 2.0. Luis Eduardo Paucar, Yrving David Conde y Roxanne Antonieta se ocuparon durante los dos años venideros a implementar DBCASE WEB, la versión web de esta herramienta, para uso online de los estudiantes de esta universidad. Ya en 2022 Guillermo Garvi Villegas se encargó de añadir modificaciones al proyecto que nos ocupa, dando lugar a la versión DBCASE 2.1. Finalmente, en el presente curso, el proyecto DBCASE 3.0 ha sido trabajado por Arturo Ibáñez Martínez y Miguel Derecho Prieto, tutelados como siempre desde que cogiera la dirección del proyecto en 2009 por Fernando Sáenz Pérez.

2.3 Frameworks y programas. Consideraciones

Eclipse

Comenzamos con el entorno de programación que ambos integrantes del equipo hemos utilizado para el desarrollo del código. Esta plataforma de software está formada por un conjunto de herramientas de programación que han sido usadas habitualmente para desarrollar entornos de desarrollo integrados (en inglés IDE) como el IDE de Java: *Java Development Toolkit* (JDT).

Elegimos esta herramienta porque ambos estábamos familiarizados con su funcionamiento, ya que fue uno de los entornos que más utilizamos durante nuestra estancia en la universidad. Su sistema de coloreado permite una gran visibilidad al programador, permitiendo distinguir de un simple vistazo variables globales de variables locales, métodos, etc. La detección de errores e incluso las sugerencias de corrección no solo facilita la comprensión del fallo al usuario si no que permite automatizar tareas como importar librerías externas o incluso clases del propio proyecto.

Sin embargo, tuvimos que aprender varias cosas de vital importancia para trabajar en este proyecto con Eclipse. Para empezar, en las propiedades del proyecto tuvimos que modificar el apartado *Java Build Path* para especificar a varias librerías empleadas la ruta en la que se encontraban desde nuestro

ordenador (cada uno de nosotros tuvo que especificar la suya propia). Además, como la codificación de caracteres del código se había realizado con el formato *UTF-8*, también descubrimos que para no tener un problema de correlación de caracteres podíamos establecer este formato de codificación en Eclipse, a través de la propia herramienta: *Window -> Preferences -> General -> Workspace -> Text file encoding -> Other: UTF-8.* Por último, debido a que las librerías que ya se utilizaban en DBCASE correspondían a versiones anteriores de Java de las que teníamos instaladas en nuestros equipos, tuvimos que configurar Eclipse para poder utilizarlas correctamente. Descargamos a través de *Oracle* el *Java Development Kit 8 (JDK 8),* pues son las herramientas de desarrollo Java que permite programar, compilar y ejecutar dicho lenguaje en la versión 1.8, algo indispensable para utilizar las librerías ya incluidas en el proyecto. Para poder trabajar con esta versión en Eclipse el proceso es el siguiente: *Project -> Properties -> Java Compiler -> JDK Compliance -> Compiler compliance level: 1.8.*

Para finalizar este apartado cabe mencionar que, para una mejor comunicación con nuestro tutor, a la hora de entregar código correspondiente a los prototipos que se desarrollaban en cada iteración del proceso con mejoras, funcionalidades nuevas o correcciones de errores; se adjuntaba un archivo *.jar* ejecutable para facilitar el acceso a la herramienta al tutor. Tuvimos por tanto que comprender cómo se genera dicho archivo a partir del proyecto utilizando Eclipse: Clic *derecho en la carpeta del proyecto -> Export -> Java -> Runnable Jar File -> Finish.*

GitHub

GitHub nos permitió crear un repositorio privado online al que únicamente nosotros disponíamos de acceso. De esta manera pudimos trabajar en el proyecto de manera cooperativa, haciendo más fácil la repartición de tareas y permitiéndonos en todo momento tener una copia local en nuestro equipo. De esta manera al introducir cambios en local, podíamos actualizar el repositorio online gracias a las herramientas de *Git*. También se puede descargar el contenido del repositorio a la copia local fácilmente.

Describimos a continuación el proceso que hemos seguido para trabajar con esta herramienta. Para los 3 primeros pasos es necesario crear una cuenta en GitHub y trabajar desde ahí. Los siguientes pasos son seguidos por cada uno de los colaboradores del proyecto en su equipo:

1. Creación del repositorio https://github.com/arturoibez/DBCase.
2. Conceder permisos de participación en el proyecto a ambos estudiantes.
3. Subir el proyecto al repositorio online.
4. Descargar las herramientas de Git a través de la página oficial.
5. Seleccionar en el explorador de archivos la carpeta donde se encuentra el proyecto DBCASE: Clic *derecho -> Git Bash here*.
6. Se abre un terminal. Para indicar al equipo que el repositorio seleccionado será un repositorio Git, escribimos el comando *git init*.
7. Para conectar nuestro repositorio local con el repositorio online escribimos el comando:
   git remote add origin https://github.com/arturoibez/DBCase.git.
8. Creación dentro del repositorio local de una rama para trabajar sobre ella: *git branch nombre_rama -> git checkout nombre_rama.*

A partir de este momento cada miembro del equipo está listo para trabajar en su rama local, utilizando Eclipse (establecemos como workspace de la herramienta la carpeta en la que abrimos el terminal en el paso 5). Una vez realizadas las modificaciones oportunas, se siguen los siguientes pasos para trasladar los cambios en local al repositorio online. Comenzamos volviendo a abrir el terminal en nuestro equipo como indica el paso 5 anterior:

1. *git add --all.*
2. *git commit –m "Mensaje que permita identificar qué cambios se han introducido".*
3. *git push origin nombre_rama.*
4. En el repositorio GitHub se ha actualizado la rama del usuario que ha introducido los cambios: *Solicitar pull request -> Confirm -> merge rquest*
5. Merge request consiste en fusionar la rama del repositorio online que hemos actualizado con la principal. Si los cambios introducidos por ambos integrantes del equipo han sido sobre archivos diferentes del proyecto, se hará automáticamente. Si hay algún archivo modificado

por ambos, deberemos resolver a mano los conflictos desde GitHub, accediendo a dichos archivos y comprobando las diferencias. Gracias a una repartición de tareas coherente, a pesar de que con frecuencia tuvimos que resolver estos conflictos porque hay clases (como por ejemplo el controlador) que han de ser modificadas prácticamente siempre que se quiere incluir cualquier cambio, la resolución de conflictos consistía en mantener los fragmentos de código incluidos por ambos.

6. Una vez resuelto el conflicto el proyecto actualizado quedaba guardado en la rama principal, denominada *master*. Para descargar de nuevo la versión más reciente del proyecto y poder así incluir los cambios introducidos por el compañero en nuestro repositorio local se debe acceder como siempre al terminal Git e introducir el comando *git pull origin master*.

Google Drive

Utilizamos una carpeta compartida de Google Drive para comunicarnos durante el desarrollo del proyecto. Nuestro tutor nos compartió una carpeta con el código comprimido en un *.zip*, la teoría correspondiente a la asignatura *Bases de datos* necesaria para comprender el funcionamiento de la herramienta y una lista de tareas para extraer ideas acerca de las modificaciones que podríamos realizar.

Para la comunicación entre nosotros creamos otra carpeta compartida propia en la que disponíamos de documentos para apuntar la modificación de tareas, un diario de modificaciones y, en los pasos finales del proceso, un documento colaborativo para esbozar el borrador de esta memoria.

Otras consideraciones

Remarcamos la dirección del repositorio de GitHub donde se encuentra el código: https://github.com/arturoibez/DBCase. El proyecto, junto con los ejecutables correspondientes, también ha sido entregado mediante la carpeta de Google Drive habilitada por la facultad, comprimidos todos los archivos en un *.zip*. La carpeta incluye un ejecutable dbcase.jar donde se ha

exportado el proyecto desde Eclipse. También cuenta con un archivo por lotes start.bat. Al hacer doble click en este último se ejecuta directamente el proyecto, puesto que encierra el siguiente comando: *start java -Dfile.encoding=utf-8 -jar dbcase.jar.*

Cabe recordar que la versión necesaria para realizar esta ejecución se corresponde con la versión Java 1.8.

2.4 Lenguajes

Java

La aplicación está desarrollada en el lenguaje de programación Java. Este lenguaje está muy generalizado y es utilizado en gran cantidad de aplicaciones. Su fuerte tipado, rapidez y fiabilidad lo convierten en un lenguaje ideal para la programación orientada a objetos. En la siguiente sección se detallan las librerías empleadas en el proyecto que han permitido sacarle partido a este lenguaje y cubrir una gran variedad de funcionalidades.

HTML

Dentro del propio código de Java y empleando Eclipse se utiliza también el lenguaje HTML para crear el texto de las ventanas de la aplicación, tales como el manual de usuario. Este lenguaje es sencillo y muy utilizado sobre todo para la generación de texto. Tanto es así que es utilizado para la confección del contenido de páginas web, permitiendo incluir imágenes, listas, vídeos, etc. Su sencillez permite su utilización por parte de personas sin conocimientos en programación.

XML

Otro lenguaje portable y compatible con Java es el lenguaje XML. Las librerías de Java que permiten la creación y modificación de archivos en este lenguaje mediante la utilización de nodos en esquema de árbol (un nodo puede poseer varios nodos hijos y así sucesivamente) permiten una gran operabilidad. Tanto es así que los ficheros que guardan los proyectos de la herramienta DBCASE

están construidos con este lenguaje. El usuario realiza modificaciones a través de la interfaz gráfica, los cuáles quedan reflejados en los archivos *.XML* gracias al código en lenguaje Java que permite realizar las modificaciones oportunas en estos archivos. El lenguaje XML proporciona una manera de definir elementos formatear y generar un lenguaje personalizado. Posee un diseño simple, general y de fácil utilización.

2.5 Librerías

En esta sección se listan las librerías Java utilizadas a la hora de desarrollar el código de la aplicación. Se han utilizado dos tipos distintos de librerías, aquellas que vienen incorporadas por defecto en Eclipse (JRE System Library: JavaSE-1.8), como por ejemplo *java.util* o *javax.swing*; y librerías externas como por ejemplo *org.json* o *javax.mail*. Cabe destacar que este último tipo de librerías ya estaban importadas cuando se obtuvo el código, y se pueden encontrar en la carpeta *libs* del directorio que compone el trabajo.

La bibliografía empleada para consultar algunas descripciones se corresponde con las páginas web oficiales de las librerías en cuestión. Los enlaces correspondientes se pueden encontrar en la sección **Bibliografía** de esta memoria.

Así, las librerías empleadas para la implementación del trabajo son:

- Librerías relacionadas con la creación y organización de la interfaz gráfica de usuario
    - **java.awt** [2]: entre las numerosas sublibrerías utilizadas destacan:
        - **java.awt.LayoutManager**: clase para la ubicación de los componentes gráficos.
        - **java.awt.Color:** clase que incorpora los métodos necesarios para trabajar con colores.
        - **java.awt.Container**: clase abstracta que proporciona métodos para incluir componentes dentro de otros elementos.
        - **java.awt.event**: proporciona los métodos necesarios para la detección de interacciones con el usuario. Destacan las

sublibrerías *KeyListener, MouseWheelListener* o *WindowListener*.

- **java.awt.geom:** proporciona los métodos necesarios para realizar operaciones con objetos bidimensionales. Destacan las sublibrerías *Ellipse2D, Point2D*
- **java.awt.Graphics:** clase base abstracta para todos los contextos gráficos que permiten dibujar sobre componentes.
- **java.awt.Font**: clase que incorpora los métodos necesarios para trabajar con fuentes de texto.
- **java.awt.Image**: clase que incorpora los métodos necesarios para trabajar con imágenes.

- **javax.swing**[2]**:** librería relacionada con los componentes (paneles) de la interfaz gráfica de usuario. Entre las numerosas sublibrerías utilizadas destacan:
  - **javax.swing.AbstractButton**: incorpora los métodos necesarios para introducir botones.
  - **javax.swing.Icon**: incorpora los métodos necesarios para introducir imágenes como iconos.
  - **import javax.swing.JComboBox:** incorpora los métodos necesarios para introducir menús desplegables con opciones seleccionables.
  - **javax.swing.JFileChooser**: incorpora los métodos necesarios para seleccionar ficheros del repositorio deseado.
  - **javax.swing.JLabel**: incorpora los métodos necesarios para introducir etiquetas.
  - **javax.swing.JMenu**: incorpora los métodos necesarios para introducir elementos de menú.
  - **javax.swing.JOptionPane**: incorpora los métodos necesarios para introducir cuadros de diálogo.
  - **javax.swing.JTextField**: incorpora los métodos necesarios para introducir campos de texto.
  - **javax.swing.event**: proporciona los métodos necesarios para la identificación de eventos sobre los componentes

        - **javax.swing.tree:** proporciona métodos necesarios para la representación de los componentes a modo de árbol.

- Librerías empleadas para las componentes visuales del grafo que constituye el esquema conceptual [4]:
    - **jung.graph:** librería externa de código abierto que proporciona las estructuras de datos necesarias para la representación de grafos y redes.
    - **jung.visualization**: proporciona herramientas avanzadas para la visualización gráfica de los elementos. Entre las sublibrerías utilizadas destacan *VisualizationViewer, EdgeShape, GraphZoomScrollPane, RenderContext, PickedState*.
    - **jung.api**: clases básicas para la construcción y tratamiento de grafos y redes
    - **jung.algorithms**: proporciona métodos para análisis de grafos y redes. Destacan *GraphElementAccessor, Layout*.
    - **collections-generic:** proporciona métodos genéricos para operar sobre datos. Se utiliza para el renderizado de los componentes del grafo.

- Librerías relacionadas con el uso de ficheros:
    - **java.io**: empleada para las operaciones de entrada/salida y lectura/escritura de ficheros. Destacan las sublibreríasn*BufferReader, BufferWriter, File, FileInputStream, FileOutputStream, FileReader, FileWriter, IOException*.
    - **java.nio**: empleada de manera análoga a la anterior, pero sin bloqueos del hilo de ejecución. Se utiliza para la implementación de la tarea correspondiente con reportar incidencia a través de email.
    - **jung.io**: proporciona métodos para leer y escribir grafos y redes como archivos.
    - **org.json**: proporciona una serie de métodos para la creación, tratamiento y análisis de objetos JSON. Los ficheros que almacenan las vistas (*themes*) tienen formato JSON.

- **javax.imageio**: permite leer y escribir imágenes a través de archivos.

  Se han utilizado también librerías específicas para el tratamiento de los ficheros XML [6]:

- **org.w3c.dom**: se representa un fichero XML como un árbol de nodos en el que cada nodo representa un elemento, atributo o información.
- **com.sun.org.apache.xml.internal.serialize**: representación de los ficheros como cadenas de caracteres o bytes.
- **javax.xml.parsers**: procesamiento de entrada de los ficheros XML.
- **javax.xml.transform**: proporciona métodos para el tratamiento de los ficheros XML.
- **org.xml.sax.InputSource**: proporciona métodos para la lectura de los ficheros XML.

- Librerías relacionadas con la comunicación con bases de datos
  - **java.sql**: proporciona métodos para interactuar con bases de datos relacionales a través de lenguaje SQL. Destacan las sublibrerías *Connection*, *DriverManager*, *Statement*, *SQLException*.
  - **ojdbc6**[3]: librería proporcionada por Oracle Corporation empleada para conectarse y operar con bases de datos Oracle.
  - **mysql-connector-java**[7]: librería proporcionada por MySQL empleada para conectarse y operar con bases de datos MySQL.

- Librerías relacionadas con las estructuras de datos:
  - **java.util**: proporciona variedad de métodos relacionados con distintas estructuras de datos y utilizados en todos los desarrollos. Destacan las sublibrerías *Objects, ArrayList, Vector, Iterator.*
  - **jung.graph**: introducida anteriormente.

- Librerías relacionadas con el envío de correo de correo electrónico en caso de reportar incidencia:

- **java.nio**: descrita previamente
- **javax.mail**: proporciona los métodos necesarios para el envío de correos electrónicos. Destacan las sublibrerías *Session, Transport, internet.InternetAddress*.
- **java.net.URI**: proporciona métodos para la conexión a internet

- Otras librerías:
  - **concurrent**: proporciona métodos para trabajar con programación paralela y concurrente.
  - **cli-module:** empleada para iniciar la aplicación a través del fichero ejecutable .jar mediante la línea de comandos.
  - **jgoodies-looks**[5]**:** empleada para mejorar la apariencia de las componentes visuales del proyecto en cuanto a diseño y renderizado.

Capítulo 3 - DBCASE 3.0

3.1 Cronología del trabajo

En esta sección se detallan los distintos pasos introducidos en la sección **1.3 Plan de trabajo**, y mediante los cuales se ha obtenido el presente Trabajo de Fin de Grado.

Como se indicó en el Capítulo 1 de la memoria, una de las principales razones por las que iniciamos este proyecto fue la experiencia que tuvimos con la asignatura de *Bases de datos*, que impartimos durante el primer cuatrimestre del curso académico 2019/2020. Además, debido a los importantes avances y funcionalidades que se han producido respecto al tratamiento de datos, decidimos realizar un trabajo relacionado con la asignatura que nos ayudase a ampliar nuestro conocimiento sobre las bases de datos. De este modo, una vez se publicaron los trabajos en la web de la facultad, nos decidimos por continuar con el desarrollo de la aplicación de escritorio DBCASE. Entre las razones de nuestra elección, destaca que durante la carrera hemos programado en Java más que en cualquier otro lenguaje. Además, nos llamó la atención la funcionalidad académica que tiene la aplicación y cómo nos hubiera gustado disponer de ella durante la asignatura cursada. Así, contactamos con Fernando en verano de 2022 y acordamos la realización del trabajo.

A continuación, como se ha comentado también anteriormente, para el desarrollo de las diferentes tareas que componen el proyecto es clave la base teórica sobre las bases de datos relacionales. Por tanto, el primer paso fue refrescar las nociones sobre el diseño conceptual de un base de datos y la correspondiente traducción al modelo lógico, así como cuestiones referentes al procesamiento consultas. Además, pudimos obtener también documentación adicional sobre aspectos fuera del temario de la asignatura, ampliando nuestro conocimiento en relación con análisis de rendimiento y la normalización de bases de datos.

De manera paralela al estudio teórico mencionado previamente, nos reunimos con Fernando y obtuvimos el código correspondiente a la versión anterior del trabajo. Además, obtuvimos un listado de tareas iniciales a realizar, que posteriormente se fue complementando con la identificación de nuevos y errores y tareas adicionales. Acordamos con el tutor decidir nosotros el orden de realización de las tareas, así como la organización de las distintas entregas.
A continuación, el siguiente paso era entender bien la aplicación. Para ello, realizamos un primer contacto con la interfaz de la herramienta para así poder comprender todas las funcionalidades y modos de uso.

Una vez entendida la aplicación nos tocó enfrentarnos al código que implementa la herramienta. Para ello, a través del tutor, intentamos contactar con el último alumno que finalizó el trabajo en septiembre de 2022. No obstante, no obtuvimos contestación alguna. Por suerte, sí pudimos contactar con Miguel Arriba, quien realizó el trabajo años atrás y, a pesar de que se habían introducido algunas modificaciones, nos ayudó a comprender la estructura del código. Sin embargo, para obtener una conclusión clara del funcionamiento del código, fueron necesarias diversas depuraciones sobre el código a la vez que se realizaban acciones sobre la interfaz del aplicativo. Debido a la gran cantidad de código y funciones existentes, el entendimiento y análisis del código se ha realizado durante prácticamente todo el desarrollo del trabajo.

Previamente al comienzo de los desarrollos, fue necesario organizarnos en cuanto a la compartición de código. Para ello, decidimos utilizar la herramienta colaborativa *GitHub* que permite importar, exportar y fusionar de manera óptima modificaciones realizadas por varias personas. Para ello, fue necesario también comprender el funcionamiento de esta aplicación, así como los distintos comandos a introducir para la realización de las acciones correspondientes.

De esta forma, pudimos comenzar con la implementación de las tareas que conforman el proyecto. Como se expone previamente, los aspectos prioritarios se corresponden con la corrección de errores sobre la aplicación y, en segundo lugar, con la ampliación de sus funcionalidades. Así, comenzamos con las tareas correspondientes a errores identificados por el tutor. Sin embargo, un aspecto que fue clave a la hora de encontrar errores y excepciones fue realizar diversos ejemplos de diagramas extensos, como los presentes en las diapositivas y en exámenes de la asignatura. A continuación, decidimos que las tareas de ampliación de funcionalidad más importantes eran la incorporación de agregaciones y la tarea Deshacer/Rehacer, repartiéndolas de manera consecuente. Al tratarse de tareas bastante extensas, de manera intercalada se han ido realizando otros avances, para los cuales no llevamos a cabo una división previa.

Como se ha comentado anteriormente el tutor delegó en nosotros la organización de las entregas. De este modo, tras realizar los distintos pasos de entendimiento y organización del trabajo, acordamos realizar una entrega de manera mensual aproximadamente. Este aspecto es fundamental para la consecución de las distintas tareas, pues con la ayuda de Fernando se han podido identificar y solucionar excepciones y comportamientos atípicos. Además, como se indicó antes, con los avances que fuimos realizando, pudimos identificar y solucionar otros errores y otras posibles tareas adicionales.
Finalmente, de manera análoga, dividimos las secciones de la memoria a realizar.

3.2 Estructura del código

Esta sección está enfocada a nuevos alumnos que continúen con el trabajo, para que tengan más o menos una base de cómo está estructurado el código y cómo funciona (Modelo-Vista-Controlador).

Controlador

La clase controlador contiene el método *Main* y es por tanto la clase que ejecuta la aplicación y controla su hilo de ejecución. En ella se encuentran los métodos que gestionan la recepción y el envío de los mensajes por parte de la vista y el modelo y permite su comunicación. En este paquete se incluye un enumerado para nombrar los distintos mensajes.

Modelo

- En el paquete modelo.conectorDBMS se encuentran las distintas clases que se encargan de la conexión con las bases de datos disponibles para la traducción del esquema.
- En el modelo.servicios las clases servicios implementan la funcionalidad de los elementos del esquema entidad-relación: entidades, relaciones, atributos, dominios, agregaciones. Estas funciones consisten en la gestión

del comportamiento de estos elementos y su inclusión en sus correspondientes DAOs. También están las clases encargadas de traducir el esquema relacional al modelo lógico y de recoger las restricciones perdidas en el proceso.

- En el modelo.transfers están las clases transfer utilizadas para intercambiar información entre los servicios y sus correspondientes DAOs.

Persistencia

Aquí, nos topamos con los DAOs (*Data Access Object),* es decir, las clases que recogen la información perteneciente a cada elemento del esquema del archivo XML relativo al proyecto en que se está trabajando, así como de volver a guardar la información una vez se ha actualizado. Este intercambio de información se realiza a través de los *Transfers*, los cambios en el estado de cada elemento son realizados por los servicios y la comunicación entre estas clases se realiza a través de mensajes gestionados por el controlador.

Vista

Guarda la clase que gestiona la Interfaz Gráfica de usuario (GUI) principal con la que interactúa el usuario de la aplicación.

- En vista.componentes encontramos las clases que gestionan los submenús que aparecen en la GUI principal al ir interactuando con ella, así como las que gestionan los árboles de dominio y de elementos y la clase que guarda un listado de los últimos proyectos abiertos.
- En vista.diagramas aparecen todas las clases que gestionan como se dibuja el esquema entidad-relación.
- En vista.frames se implementan todas las GUI's desplegables que utiliza el usuario para manejar distintas funcionalidades de la aplicación.
- En vista.iconos se gestionan los iconos que aparecen en la interfaz gráfica.
- En vista.imagenes se almacenan las imágenes que aparecen en la interfaz gráfica.

- En vista.tema se gestionan los colores que colorean la interfaz gráfica en función del tema seleccionado. Actualmente podemos elegir entre el tema luminoso o el tema oscuro.
- En vista.lenguaje nos ocupamos del idioma. Actualmente el idioma por defecto es el inglés, pero también podemos seleccionar el español. En esta clase nos encargamos de gestionar, a través de un enumerado, los términos a los que nos queremos referir cuando comunicamos algo al usuario a través de la interfaz. A cada elemento del enumerado le asignamos un valor que relacionamos en cada archivo *.lng* con su idioma correspondiente.

3.3 Desarrollos

En esta sección se incluyen todos los avances realizados con respecto a la versión anterior de la aplicación DBCASE. Hemos dividido las implementaciones entre correcciones de errores y excepciones existentes, y nuevas implementaciones en la herramienta. Cabe destacar que muchas de las correcciones no fueron consideradas como objetivos iniciales, pues se fueron identificando conforme se realizaban pruebas.

Además, en la última subsección se incluyen aquellas tareas en las que se ha trabajado, pero no han acabado de funcionar.

3.3.1 Correcciones

- Se ha solucionado los errores que existían al editar elementos desde el cuadro emergente que surge al hacer doble clic sobre ellos. Concretamente, no se desactivaban los cuadros de entidades y relaciones, y en algunas ocasiones saltaban excepciones al editar atributos de esta forma.
- Si se insertaba un atributo marcado con clave primaria o *Unique*, de manera errónea (nombre vacío o nombre ya existente en la entidad / relación), el programa continuaba con la ejecución para añadir la opción de clave primaria a un atributo inexistente, produciéndose una excepción *nullPointer*.

- Se ha solucionado el error que ocurría al añadir una entidad hija a la relación IsA en DBCASE.
- Al incluir una relación IsA y generar el esquema lógico, saltaba un error cuando existía una entidad hija sin atributos, cuando debería incorporar la clave primaria de la entidad padre.
- Al debilitar una entidad, introduciendo como nombre de relación uno ya presente, saltaba error por renombramiento, pero la entidad se visualizaba y se mantenía dibujaba como débil.
- En este sentido, en la traducción al modelo lógico, cuando una relación incluía atributos de entidades, los cuales eran atributos heredados, existía la posibilidad de repetición en el nombre de un atributo. Este aspecto se ha solucionado introduciendo numeración cuando sucede la repetición.
- Saltaban excepciones al eliminar varios elementos a la vez, si estaban seleccionados. Concretamente:
    - Entidades y atributos de la propia entidad
    - Entidades débiles y la correspondiente relación débil
- En algunos casos, no se quedaban marcadas las propiedades de cardinalidad, participación, min-máx.
- De forma predefinida, al crear una entidad débil se añadía participación total a la entidad fuerte.
- Enlazando con lo anterior, no se permitía la modificación de la cardinalidad y roles de entidades débiles.
- Los atributos multivalorados se traducían sin claves primarias.
- Saltaba una excepción al definir unicidad sobre un atributo, cuando ya existía otro en la entidad con *Unique* activo.
- Se han introducido modificaciones en cuanto al tamaño y texto en el cuadro editar cardinalidad/participación/rol de entidad. Además de en el submenú correspondiente al hacer clic derecho sobre una relación.
- Se ha eliminado la dependencia que existía entre la cardinalidad y la participación máxima al añadir/editar una entidad en una relación.

- Se han introducido modificaciones visuales en el tema *dark*, pues no se veían bien las propiedades marcadas de atributos.
- Se ha identificado y solucionado un error presente a la hora de reportar incidencias. La ventana de reportar reconocía todo el panel como el botón de reportar. Además, en el modo oscuro el color de la fuente coincidía con el del fondo y no se veía el texto a reportar.
- Si se quería modificar que una entidad débil pasase a ser fuerte, no se modificaba la debilidad de la relación.
- Se han corregido de algunos aspectos relacionados con el lenguaje, pues las explicaciones de ciertos botones (*ToolTipText*) aparecían siempre en castellano.
- Al añadir o eliminar un subatributo a un atributo que tuviera activada la opción *Clave Primaria*, desaparecía esta propiedad.
- Al cambiar el tema o el lenguaje de la herramienta en la vista *VerTodo*, se contraía totalmente el panel.

3.3.2 Nuevas funcionalidades

- Se ha añadido la cadena deshacer y rehacer, junto con los correspondientes atajos de teclado (Ctrl+Z y Ctrl+Y). El color de los botones correspondientes se marca en gris cuando se llega al límite. Para implementar esta tarea se crea una carpeta temporal que almacena ficheros conforme se realiza un cambio. De este modo, se permite hacer *Deshacer* y *Rehacer* tantas veces como se desee. Al salir de la aplicación, esta carpeta se elimina automáticamente. Esta tarea a consistido en mucho esfuerzo y trabajo debido a la gran cantidad de aspectos a tener en cuenta. En primer lugar, se ideó mediante el registro del último mensaje de acción enviado al controlador y el posterior envío del mensaje "opuesto". No obstante, esta manera de implementación solo permitía al usuario retroceder en las acciones una única vez. Por tanto, se decidió proceder de la forma actual. La realización de esta tarea no ha sido nada sencilla y ha sido perfeccionada a través de numerosas modificaciones y mejoras. Con la ayuda del tutor, se han

podido identificar errores y comportamientos extraños en el funcionamiento que han podido ser solucionados con mucho esfuerzo y trabajo, hasta la versión final presente. Además, esta tarea permite integrar fácilmente cualquier funcionalidad nueva insertada. Simplemente consiste en añadir el mensaje enviado al controlador que informa de la realización de una acción a una función propia de la clase Controlador.

- Se ha introducido la posibilidad de usar las teclas Y y N en los cuadros de dialogo Yes/No. Además, se ha incluido la posibilidad de moverse sobre los botones con las flechas del teclado.
- Se ha añadido la opción de visualizar un manual de usuario desde la herramienta. Se ha incorporado el submenú correspondiente en el Menú Help.
- Finalmente se consideró que la inclusión del menú Edición supondría una excesiva carga de la barra de menús. En su lugar se incluyen los botones *Deshacer* y *Rehacer*, junto con los atajos de teclado correspondientes. Además, se han implementado las tareas de Copiar y Pegar, junto con sendos atajos: Ctrl+C, Ctrl+V. Implementar esta tarea para atributos causaba algunos problemas y no lo consideramos necesario.
- Se han añadido iconos a las acciones de los menús Archivo y Vista. Para esta tarea, se ha utilizado la siguiente librería de iconos libres de derechos de autor:

  https://icon-icons.com/es/

  Además, para transformarlos en iconos personales se han introducido pequeñas modificaciones (tonalidad del color, fusión de iconos, cambio de forma y tamaño, …).
- Se han añadido elementos de menú para todas las acciones. En particular para las acciones del menú contextual y para los botones de la generación de código.
- Se ha añadido el menú Vista a la barra de menús. A través de este menú se permite cambiar las vistas de la herramienta. Además, se han incluido, pero no implementado, las opciones de modificar manualmente el zoom

de la ventana, y la inclusión de cuadricula para ayudar a la alineación de los elementos. En la siguiente subsección se profundizará en estos aspectos.

- Se ha añadido una carpeta de ejemplos, con ejemplos completos de diseños conceptuales. Esta tarea ha sido indispensable para la identificación de errores. Se recomienda no modificar los esquemas creados, la tarea esta enfocada a probar los tres pasos distintos en la creación de una base de datos (modelo conceptual, lógico y físico).
- Si el proyecto no ha sido guardado durante los últimos diez minutos, se crea un proyecto a modo de *backup*, por si hubiera algún problema inesperado.
- Se ha actualizado el panel de información *About DBCASE* con toda la cronología de los diferentes trabajos.
- En el mismo menú *Help* en el que se encuentra el submenú *About,* se ha incluido un breve *Manual de Usuario* en el que a través de ilustraciones explica las diferencias entre los distintos esquemas que se generan en la aplicación.
- Se ha incluido en el menú Archivo la opción de abrir ficheros recientes. Uno de entre los 10 últimos archivos utilizados desde la aplicación podrá ser abierto desde aquí.
- Se ha incluido un botón Delete/Eliminar en la GUI principal que permite seleccionar una entidad, relación, atributo o agregación que deseemos eliminar. Si se elimina de esta manera una agregación, los elementos que la formaban siguen existiendo, solamente quedan desligados de la agregación. Al pulsar el botón se abre una ventana en la que aparece un desplegable que lista todos los elementos del esquema de la siguiente manera: primero las entidades, seguidas cada una de ellas por sus atributos (de esta manera se puede diferenciar si existen dos atributos de elementos distintos con el mismo nombre cuál estamos borrando), a continuación, las relaciones con el mismo tratamiento en cuanto al listado de atributos; y por último las agregaciones.

- Se ha implementado el código necesario para incluir las agregaciones. Esto implica la creación de clases como la clase transfer, servicios o DAO, así como la modificación de estas clases para los otros elementos de tal forma que tengan en cuenta la existencia de agregaciones.

  Para insertar una agregación, se selecciona una relación haciendo clic en el botón derecho del ratón sobre la misma en el esquema relacional. Si la relación seleccionada no pertenece a ninguna agregación, una de las opciones que aparecen es A*ñadir a agregación*. Al pulsar esta opción, se añade dicha relación y las entidades que participan en ella a una agregación. Para ello, aparece un cuadro que pide al usuario que inserte el nombre de la misma.

  Al pulsar clic derecho sobre una relación que sí pertenece a una agregación, se deshabilita la opción de *Añadir agregación*, y aparece en cambio la posibilidad de *Eliminar agregación*. Al seleccionar esta opción, se elimina la agregación, al igual que al eliminarla desde el botón *Delete*.

  En relación con el punto anterior, si la relación seleccionada pertenece a una agregación, también aparecerá una nueva opción *Renombrar agregación*. Al seleccionarla aparece un cuadro que insta a introducir el nuevo nombre. Una vez introducido, se selecciona el botón confirmar de la interfaz y la agregación correspondiente a la relación seleccionada modifica su nombre.

  Se pueden añadir atributos a una agregación a través del botón *Añadir atributos* de la interfaz principal. El atributo aparece de momento desligado al esquema a falta de la representación gráfica de las agregaciones. Se puede editar y eliminar.

  La agregación se elimina cuando la relación que la forma es eliminada.

  La existencia de las agregaciones queda reflejada en el menú de elementos de la interfaz.

3.3.3 Tareas sin concluir y mejoras

- Al insertar un atributo con las opciones *Clave Primaria* o *Unique* activadas, por cómo está implementación la aplicación, se mandan dos mensajes al controlador, uno para insertar el atributo, y otro para modificar la opción *Clave Primaria* o *Unique*. De este modo, al deshacer o rehacer esta acción, es necesario dar dos veces al botón, una para la modificación de la propiedad, y otra para la creación del atributo. Este aspecto se solucionó en un primer momento, introduciendo un *flag* en el mensaje de modificación de la propiedad *Clave Primaria* y en el de modificación de la propiedad *Unique*, de forma que el controlador identificaba cuando estos mensajes se enviaban por acciones directas o debido a consecuencias de otras acciones. El fichero auxiliar utilizado para *Deshacer* y *Rehacer* únicamente se creaba cuando la modificación de las propiedades era directa. Sin embargo, esto provocaba que las propiedades marcadas del atributo no se guardaran en este fichero cuando se creaba un atributo. Por lo tanto, cuando se aplicaba cualquier acción que requiriera la apertura de ficheros, como *Deshacer*, *Rehacer*, cambiar tema o cambiar lenguaje, las restricciones se perdían. Finalmente, no se pudo solucionar este aspecto y se decidió mantener la necesidad de realizar dos acciones si el usuario quiere deshacer o rehacer la inserción de un atributo con estas restricciones.
- Enlazando con lo anterior, algo similar sucede a la hora de insertar una entidad débil, pues esta acción conlleva la creación de la relación débil asociada, junto con la adición de las entidades fuerte y débil a la relación. En total, esta acción requiere el envío de 4 mensajes al controlador.
- Al relacionar una entidad consigo misma de manera recursiva con cardinalidad 1, no se pintaba una flecha sino una línea recta. Hemos identificado el problema ya que hay clases distintas que pintan, o bien una flecha o bien una recta, y hemos podido permitir que se pinte la flecha, pero no hemos sido capaces de manejar las funciones matemáticas utilizadas para proyectar las líneas y el dibujo queda algo descuadrado. Además, en algunas ocasiones pueden producirse

comportamientos extraños al seleccionar la cardinalidad 1 de las entidades en relaciones recursivas, como que no se visualice la flecha, o que las dos entidades se marquen con esta restricción.

- Por otro lado, una posible mejora de la tarea Copiar/Pegar es permitir copiar y pegar varios elementos a la vez, lo cual causaba algunos errores en el reconocimiento de los elementos copiados y decidimos no implementarlo. Del mismo modo sucedía con los atributos, pues no se reconocía bien a que elementos pertenecían. Estas implementaciones se encuentran comentadas en el código fuente, incluyendo el mantenimiento de los atributos al copiar y pegar entidades o relaciones, por si pudiera servir de ayuda a futuros alumnos.
- Se ha trabajado en permitir especificar manualmente el nivel de zoom del panel del esquema conceptual. En el menú Vista se ha añadido un submenú *Zoom*. Al seleccionarlo, se abre una ventana que permite al usuario seleccionar el nivel de zoom deseado en el panel conceptual. Esta ventana reconoce las teclas + y – del teclado (figura 19). El valor de zoom introducido se guarda entre ejecuciones a través del fichero de configuración.

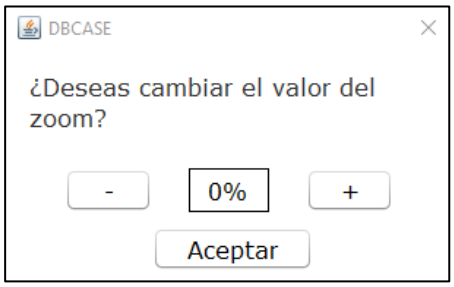

Figura 19. Ventana Zoom manual

No obstante, al darle a aceptar, a pesar de variar el nivel de zoom, en cuanto el usuario realiza una acción se vuelve al valor de zoom previo (el que se modifica con la rueda del ratón). En este sentido, no conseguimos asociar el valor de zoom marcado con el ratón con el introducido manualmente por el usuario. Finalmente, si se activaba manualmente un valor de zoom, identificamos que en algunas ocasiones el programa reconocía acciones extrañas realizadas con el ratón, devolviendo una excepción. El código fuente programado se encuentra comentado, por si pudiera servir como guía para futuros alumnos. La ventana y el submenú se mantienen en la interfaz de la herramienta, para que futuros

alumnos puedan continuar con la implementación. Se ha añadido una etiqueta "En desarrollo" al submenú, que emerge como *ToolTipText*.

- Para ayudar con la alineación de los elementos se ha trabajado en la introducción de un modo *Cuadrícula,* que insertara una cuadricula de fondo en el panel conceptual. Este modo se podía seleccionar también a través del menú Vista. La tarea se pensó mediante la inserción de una imagen como fondo. Se programaron los mensajes correspondientes entre vista y controlador, pero no se acabó de conseguir que se mostrase la imagen como fondo. Al igual que en la tarea anterior se deja comentado en el código las funciones utilizadas en las pruebas. En este sentido, se creó una clase que en esta versión no es útil, pero puede ayudar a enfocar esta tarea. Se trata de la clase *ImagePanel*, que extiende *JPanel* junto con nuevos modos de visualización.
- Se ha trabajado en permitir al usuario hacer zoom en los esquemas lógico y físico. Está tarea se ideó mediante el conjunto de acciones: pulsar la tecla CTRL + rueda del ratón, pero al igual que en la tarea del zoom manual, no conseguimos que el panel donde se muestran estos diseños reconociera las acciones realizadas. Al igual que en el apartado anterior, se mantiene el submenú en la interfaz de la herramienta, para que futuros alumnos puedan continuar con la implementación. Del mismo modo, se ha añadido una etiqueta emergente "En desarrollo".
- Se identificó el siguiente error: cuando se traduce al modelo lógico con restricción perdida de clave (en tablas de relaciones con entidades con cardinalidad 1), se indica que las claves primarias de la entidad que no transfiere los atributos a la relación son claves candidatas de manera individual, cuando debería ser una combinación de ellas. El código está implementado correctamente para reconocer la clave candidata como concatenación de las claves individuales (como se puede observar en el modelo físico), no obstante, separa las restricciones para mostrarlas. Al intentar solucionarlo se producían errores en cuanto al reconocimiento de restricciones que sí se mostraban bien inicialmente. Esto, junto con el aspecto de que el error fue identificado poco antes de entregar la memoria, hizo que decidiéramos indicar el matiz mediante palabra (figura 20).

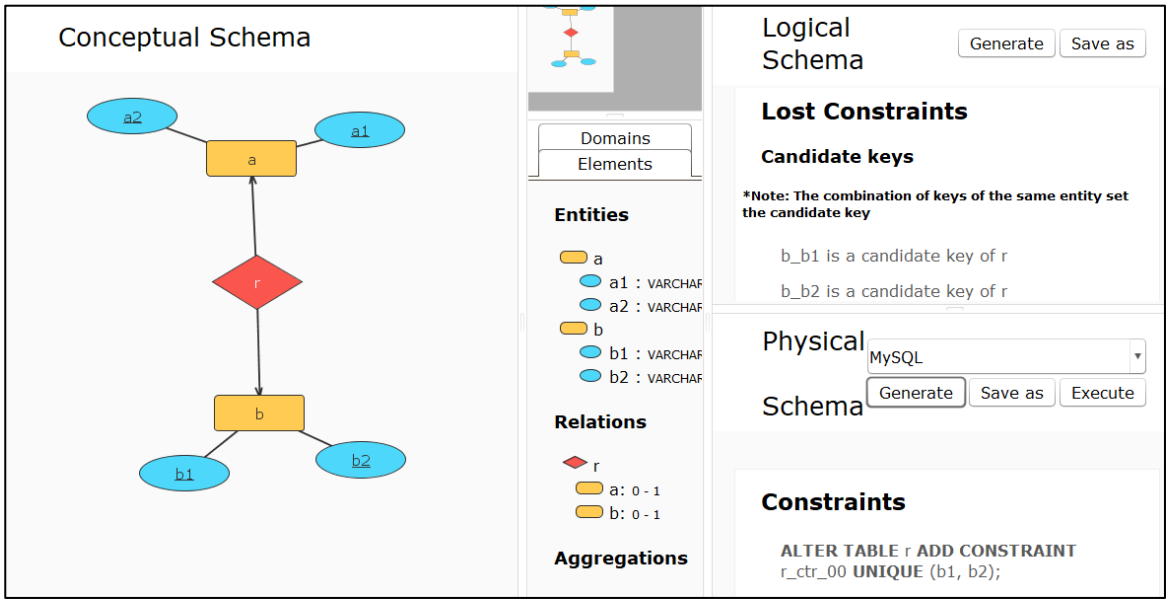

Figura 20. Restricciones perdidas. Clave candidata

- Se ha identificado un error presente en la versión anterior que provoca que con cualquier acción que conlleve apertura de ficheros, como son pulsar a los botones *Deshacer* o *Rehacer*, cambiar el tema o cambiar el lenguaje, se pierden las restricciones de cardinalidad 1 de las entidades. Inicialmente se pensó que este aspecto había sido introducido indirectamente en la corrección de la dependencia que existía entre cardinalidad y participación máxima. No obstante, el error se encontraba presente en la versión anterior. Se ha intentado solucionar identificando la cardinalidad 1 de las entidades de otros modos distintos al que estaba implementado, como añadir la restricción al fichero XML que representa un proyecto.
- Por el momento las agregaciones no se representan en el esquema relacional. Deberían quedar fijados sus elementos y recuadrados por un rectángulo. Para ello, habría que implementar el método abstracto de la clase transfer en el transfer de agregaciones.
- Se debe permitir añadir una agregación a una relación. Para ello intentamos que el transfer de las agregaciones heredase del de las entidades. Tuvimos que descartarlo debido a que, aunque en la traducción del esquema e incluso a la hora de añadirlas a una relación

las agregaciones se comporten como entidades, presentan tales diferencias y existen tantos fragmentos del código en que se hacen distinciones del flujo de ejecución según estemos tratando una entidad/relación/atributo/agregación que existían demasiadas complicaciones. En la clase *EntidadYAridad* se guarda como atributos las relaciones para saber qué entidades y con qué aridad están ligadas a ellas, una idea para implementar esta tarea puede ser permitir que en vez de una entidad pueda ser una agregación.

- Se deben generar las tablas de una agregación en los esquemas físico y lógico. Esto consiste en que la tabla generada para la relación de la agregación lleve las claves primarias de las entidades relacionadas y de la propia relación. En esta tarea se ha trabajado y se le ha dedicado una considerable cantidad de tiempo. Sin embargo, aunque ese esfuerzo no haya sido traducido en la aplicación de manera que pueda percibirse, sí que queda reflejado en el código. La clase encargada de generar este esquema recibe adecuadamente la información necesaria para la traducción de las agregaciones por parte del controlador. También hay código que, si bien se optó por dejarlo como comentario, servirá como base para completar esta tarea en el futuro.
- También se debe implementar la forma en que se crea la tabla de una relación que tiene una agregación ligada a ella, llevando las claves primarias de las entidades de la agregación y del resto de entidades ligadas a esta relación. Nótese que previamente se debe realizar el punto anterior.

Capítulo 4 - Conclusiones y trabajo futuro

Este proyecto ha sido un desafío para nosotros a la par que una experiencia enriquecedora en muchos aspectos. Ambos alumnos acabamos muy satisfechos, tanto con la experiencia de haber participado en el proyecto como con el resultado final. La cantidad de errores encontrados en la aplicación por nosotros mismos, así como la corrección tanto de estos errores como aquellos que ya se conocían, suponen una mejora notable en el funcionamiento de la herramienta DBCASE. Además, se han añadido varias funcionalidades que facilitan la utilización de la aplicación por parte del usuario. Entre ellas cabe destacar la opción de Deshacer/Rehacer y la inclusión de las agregaciones. Al principio del proceso, cuando identificamos las tareas que realizar a lo largo del proyecto, reparamos en estas tareas, pues las concebimos como aquellas que supondría un mayor avance para la aplicación DBCASE. Son las tareas que han supuesto más esfuerzo y más cantidad de código. Además, al ser tareas empezadas desde cero sin base previa, quedó en nosotros la responsabilidad de elegir la manera en qué deberíamos implementar el nuevo código. Consideramos que las ideas que acabamos llevando a cabo no sólo respetan la estructura del código si no que suponen una solución elegante y coherente con el funcionamiento interno de la aplicación.

Como se ha comentado previamente, la consecución de los objetivos, tanto los marcados inicialmente como los identificamos durante el transcurso del proyecto, ha consistido en un trabajo incremental. Es decir, aunque en la memoria final aparezcan muchas tareas listadas, la realización de muchas de ellas ha consistido en numerosos avances y retrocesos. Identificación de errores de implementación, elaboración de los distintos procesos para probar los avances, excepciones extrañas en ciertos casos, dificultades a la hora de integrar el trabajo paralelo o modificaciones de otras acciones al introducir o modificar código existente han sido algunos de los aspectos y retos que nos hemos encontrado durante la realización de las distintas tareas, tanto de corrección de funcionamiento como de extensión de la funcionalidad.

Adicionalmente, en un primer momento, recibir un proyecto tan complejo con tantas clases y código tan extenso resulta nos resultó algo abrumador. No obstante, un análisis general del código en las primeras fases del proceso nos ayudó a formar una idea general de cómo funciona internamente la aplicación, siendo muy fácil de apreciar que se utiliza el patrón de programación Modelo-Vista-Controlador. La encapsulación del código propia de este patrón de la programación orientada a objetos permite de un simple vistazo comprender qué paquete del proyecto agrupa las clases que se encargan de implementar cada funcionalidad de la aplicación. Sin embargo, dentro de un mismo paquete existen una enorme cantidad de clases distintas, cada una bien diferenciada, las cuales son necesarias de comprender.

A pesar de este primer análisis, la manera de entender en profundidad la estructura del código y generar una noción sobre dónde hay que incluir o modificar los aspectos necesarios para una nueva tarea, consiste en proceder a intentar las distintas tareas propuestas y trabajar con la aplicación para comprender todas sus funcionalidades. A medida que íbamos trabajando en el proyecto desarrollábamos una mayor familiarización con la herramienta, así como un mayor conocimiento del código, reduciendo el tiempo de comprensión y análisis de cada nueva tarea y optimizando la implementación utilizando todos los recursos existentes en el software de partida.

Además, un aspecto adicional que hemos obtenido con esta manera de proceder es que nos ha permitido conocer nuevas formas de interactuar con la herramienta Eclipse, descubriendo técnicas y comandos para buscar a lo largo

del proyecto fragmentos que deseábamos modificar, permitiéndonos automatizar el proceso de detección del módulo con el que deseábamos trabajar para realizar una tarea en concreto. De igual modo, tanto para localizar este punto, como para localizar errores o comprobar el comportamiento de la aplicación durante su ejecución (llamadas a funciones, estado de las variables, comportamiento de las mismas, tipo de elementos que almacena cada colector …), hemos trabajado mucho la depuración del código gracias al modo *Debug* que ofrece Eclipse.

El hecho de que el proyecto se haya ido desarrollando a lo largo de diversos años supone dos cosas. La primera es que ha ido evolucionando de manera incremental, es decir, las funcionalidades se han ido añadiendo una tras otra teniendo en cuenta siempre el estado del proyecto en el pasado. La segunda es que han sido muchas las personas que han colaborado a la implementación de DBCASE. Por estos motivos aparecen complicaciones a la hora de la comprensión del código habituales en este tipo de proyectos. Es recomendable adaptarse a la manera en que está implementada la aplicación, utilizar una nomenclatura similar en el nombre de métodos o variables, que aclaren de un simple vistazo cuál es su función, para una mayor legibilidad por parte de futuros programadores. También es vital respetar la estructura para mantener la coherencia y contribuir a esta comprensión del código por otras personas, pero también para no romper la encapsulación y respetar el patrón Modelo-Vista-Controlador, de tal manera que para modificar un aspecto de una clase o de un módulo no sea necesario realizar grandes cambios en el resto del proyecto.

También es importante conocer cómo trabaja la herramienta DBCASE, cómo gestiona los proyectos a través de los archivos *.XML*, la forma de abrirlos, editarlos y guardarlos, así como localizar las rutas del equipo en que se encuentran. La realización de la depuración de tareas mencionada con anterioridad no sólo ayuda comprobar el flujo del código de manera dinámica y a localizar los errores, también es de gran utilidad para comprobar cómo se van modificando estos archivos, que son los que guardan la información útil con la que trabaja la aplicación.

El desarrollo del proyecto da lugar a un proceso de mucho desgaste, de ensayo y error, de realizar muchas pruebas tanto del código como de la propia aplicación. La curva de aprendizaje sufre un crecimiento demasiado lento en

las fases iniciales. La localización de errores es costosa y nunca se tiene la certeza de si han sido producidos por un error en el código de nueva creación o era un fallo que arrastraba el software de las versiones anteriores, lo que complica la manera en que se ha de resolver. Sin embargo, resulta una gran satisfacción llevar a cabo cada tarea y comprobar la evolución real del proyecto, con sus nuevas funcionalidades y la reducción de errores pertinente. DBCASE es una aplicación con numerosas funcionalidades y con intención de ser utilizada en la Universidad Complutense de Madrid, lo cual motiva a seguir buscando la forma de perfeccionarla.

A nivel personal, consideramos que este proyecto es ideal para la realización de un trabajo de estas características, como puede ser un Trabajo de Fin de Grado, puesto que permite un amplio repaso de las distintas maneras de representar y comprender una base de datos, así como una ampliación de los conocimientos en este campo. También nos ha permitido tener una experiencia muy similar a la que se adquiere al comenzar a trabajar en un proyecto de cualquier empresa. Hemos tenido que realizar una planificación, reparto de tareas, comprensión de código ajeno y amoldarnos a este. Como hemos mencionado a lo largo de este documento también hemos adquirido conocimientos importantes en cuanto al uso de herramientas tales como Git/GitHub (también utilizadas en el mundo empresarial) o Eclipse. También hemos conocido un sinfín de maneras de trabajar con Java y posibilidades que ofrecen este lenguaje hasta ahora desconocidas por nosotros. Es por ello que consideramos que el proyecto presenta tanto una parte de investigación como una parte más clara de implementación. Por todos los conocimientos reflejados y adquiridos durante la realización de este proyecto, así como por el apego personal que se genera con el mismo, recomendamos a aquellos estudiantes en búsqueda de un Trabajo de Fin de Grado que se planteen continuar contribuyendo a la mejora de DBCASE.

Debido a esto precisamente, hemos elaborado esta memoria con la idea de que sirva a futuros alumnos como guía para comprender de manera general la estructura del código, así como para desarrollar una noción para completar las tareas que han quedado sin terminar o que son susceptibles de mejora.

Como se comentó al inicio de esta sección, ambos alumnos estamos muy satisfechos con el resultado final del proyecto, pues consideramos que hemos

extendido y perfeccionado una herramienta realmente útil para la comprensión, diseño y manipulación de bases de datos. No obstante, como es propio en los proyectos de desarrollo software, no todo el esfuerzo y tiempo invertido se puede reflejar en las tareas finales. De ahí que hayamos decidido incluir la subsección **3.3.3 Tareas sin concluir y mejoras**, en la que explicamos las distintas ideas y procedimientos llevados a cabo para intentar finalizar o perfeccionar los aspectos listados. Esta subsección tiene la intención adicional de ayudar a posibles alumnos que continúen con el proyecto a enfocar la realización de estas tareas.

Por último, de cara a introducir el posible trabajo futuro, como acabamos de indicar, animamos a que se completen las tareas presentes en la sección **3.3.3 Tareas sin concluir y mejoras**, teniendo en cuenta todas las consideraciones indicadas.

Por supuesto recomendamos también seguir explorando la aplicación tanto para comprender el funcionamiento como para encontrar posibles errores que deban corregirse. Como se ha comentado en diversas secciones las prioridades del proyecto se dividen de la siguiente manera, en función de su prioridad:

- Corrección de errores de funcionamiento y bugs.
- Adición de nuevas funcionalidades.

Por otro lado, en cuanto a extensiones de funcionalidad, algunas de las posibles tareas adicionales a realizar pueden ser:

- Dar soporte a la notación gráfica de Elmasri para la cardinalidad de entidades en relaciones. Esta notación consiste en indicar la cardinalidad mediante un 1 en vez de una flecha, y mediante una N en caso contrario. Se podría incluir en la herramienta una opción para cambiar entre la notación actual y la de Elmasri.
- Incluir reingeniería de bases de datos. Esta tarea consiste en poder diseñar una base de datos desde el esquema lógico o físico, introduciendo las tablas o consultas necesarias. A continuación, se podría mostrar el esquema conceptual correspondiente. Para acometer esta tarea, es necesario codificar un procesamiento sintáctico entre los datos introducidos en los paneles lógico o físico y generar los mensajes necesarios para la inserción de los elementos en el esquema conceptual.

Asimismo, sería necesario introducir los posibles mensajes de errores o avisos.

- Incluir la opción de visualizar un análisis de rendimiento, con el objetivo de optimizarla base de datos diseñada. Esta tarea se puede implementar llevando la cuenta de los elementos presentes de cada tipo y asignar las funciones de coste correspondiente. Se podría mostrar al usuario el coste total de la base de datos diseñada, para que pueda acometer variaciones si fuese necesario.
- Por último, otra tare de extensión posible puede ser la normalización automática de la base de datos diseñada por el usuario. El objetivo principal de esta tarea es reducir las redundancias, garantizar la integridad de los datos y eliminar las dependencias incoherentes.

Introduction

A database is an organized and structured set of information that allows efficient access, update and management of data. The different functionalities and applications of databases in the real world are almost infinite, being present in all computer systems.

Due to the significant progress in the analysis and processing of information, as well as the benefits that these practices bring, databases are a fundamental tool for many different areas. Among many of the possible fields where large amounts of information are used and, consequently, the importance of databases is enhanced, the following stand out:

- Scientific research, in order to predict and study different behaviours related to meteorological phenomena, demographic studies, epidemiology, etc.
- Business information systems, in order to record and manage the activities carried out, workers, incomes, etc.
- Web applications, such as Google, to store and provide information quickly.

In the Faculty of Computer Science at the Universidad Complutense de Madrid there are a large number of subjects related to data processing and optimisation. In this project we will focus on aspects related to the subject *Bases de datos* (*Databases*), which is necessary for the understanding and approach of the students to the different existing methods to design, access and manage

a real database. Specifically, we will focus on relational databases, i.e. based on Entity-Relationship diagrams.

We have continued with the work carried out by Professor Fernando Sáenz Pérez, together with various students of the faculty, on the DBCASE desktop application. This tool brings together the three key phases for the implementation of a relational database, allowing the user to design the Conceptual Model, by means of diagrams, and automatically translate it to the Logical and Physical Models.

Motivation

The principal personal motivation of the project corresponds to extending the knowledge about databases acquired during the subject of *Bases de datos*. In this way, through the development and correction of the DBCASE tool, the aspects learned during the course have been complemented and refreshed, specifically those related with the conceptual design of a relational database and the corresponding translations into logical and physical models. Furthermore, it has also been possible to study additional aspects related to normalisation and performance analysis.

DBCASE stands for Database Computer-Aided Software Engineering. The tool was created in 2008 by Yolanda García Ruiz, continuing Fernando Sáenz Pérez in the next years. This work is the version 3.0 of the desktop application.

By means of an intuitive and interactive graphical interface, it allows the user to design the behaviour of a database using Entity-Relationship diagrams. In addition, from the Conceptual Schema created, it shows the corresponding translations to the Logical Model, being able to corroborate the correctness of the design; and Physical Model, by generating the code associated with the creation of each table.

This tool is focused on the academic environment, making it easier for students to understand Database Theory from practical cases. However, its use is not only focused on this area, the ability to generate the corresponding SQL code allows the execution of real scripts in different database managemet systems (DBMS) such as Oracle, MySQL or Microsoft Access.

Goals

As introduced above, this project constitutes the version 3.0 of the DBCASE desktop application, a tool that is used for the design and implementation of relational databases.

The main objective of the work is to correct different the errors and exceptions that existed in the previous versions of the tool, as well as to include new functionalities, in order to enrich the possibilities of the application.

Below, there is a list of some of the tasks initially set as objectives, related to both corrections and the extension of functionalities:

- Review the generation of the relational schema, with the intention of identifying possible errors and exceptions that may exist.
- Include de functionality to add aggregations in the tool.
- Add undo and redo strings (together with the keyboard shortcuts Ctrl+Z and Ctrl+Y).
- Add a list of recent files to open.
- The following error has been identified, and the objective is to solve it: when adding a child entity to the IsA relationship, an error message pops up, however, the process creates a link between the IsA relationship and the selected child entity.
- Try to include the arrow representing the cardinality constraint 1 in recursive relationships.
- Allow the use of the Y and N keys in the Yes/No dialogue boxes.
- Introduce a submenu in the Help menu that allows to display a user manual.
- Introduce re-engineering of the physical/relational design. From the tables created in the logical and/or physical schemas, display the corresponding conceptual schema.
- Identify and solve possible errors and exceptions that may exist when performing actions. For example, double-clicking on elements opens a box to edit them, but sometimes the modifcations are not correctly done.

- Allow to manually specify the zoom level of the concept diagram panel.
- Allow zooming in the logical and physical schemas.
- Introduce a new Edit menu, with Copy, Cut, Paste, Undo and Redo submenus. Also include the corresponding shortcuts: Ctrl+C, Ctrl+V, Ctrl+Z and Ctrl+Y.
- Add menu items for all actions. In particular, for the context menu actions and for code generation buttons.

It should be noted that as the project progressed, several existing errors were identified, and their solution was considered a priority.

The objectives that could not be completed are listed in section **3.3.3 Tareas sin concluir y mejoras**, together with an explanation of the actions taken in the attempt to implement them, as well as some considerations and indications of how they could be implemented.

Additionally, the goal of the report, besides of course showing what has been implemented, is to help future students who will continue with the project to understand how the application and the code work, as well as the different considerations we recommend when organising the work.

Work plan

This section sets out the work plan followed in order to reach the objectives indicated above. The work plan followed for the realisation of this project is detailed in section **3.1 Cronología del trabajo.** The different steps followed were:

- Contact with Fernando Sáenz Pérez, tutor of the TFG.
- Study of the theory corresponding to the subject of *Databases*, related to the conceptual design of a relational database and the corresponding translation to the logical model, as well as query processing. In addition, theory outside the scope of the subject was also inspected, related to normalisation and performance analysis.
- Obtaining the code and first contacts with the application interface.
- Analysis and understanding of the code.

- Identification of priority tasks and division of work
- Organisation between students for code sharing and merge of independent developments.
- Developments on the application and communication with the tutor.

- Writing this report

Structure of the Report

Once the previous questions about the importance and usefulness of databases have been introduced, as well as the motivation and objectives of the project, where the DBCASE tool is described and presented, the different chapters that make up this report will be detailed.

Firstly, Chapter 2 sets out the background to the state of the art of the project. In this sense we introduce the theoretical bases necessary to understand the subject of *Databases* and to be able to face the different tasks of which the work is composed. Furthermore, as this project corresponds to the DBCASE 3.0 version of the tool, a brief summary is made of the different versions and ancestors of the application, including its analogue web tool DBCASEWeb. The chapter closes by outlining the different technologies used for the implementation, from the frameworks and tools used to communication and code sharing, to the libraries and programming languages that base the code development.

In Chapter 3, we will focus on the project itself. This chapter, as well as including the different tasks implemented, is oriented to help potential students who will continue with the work in the coming years. In this sense, after introducing the chronology of the work followed, an explanation of the structure of the application code is shown. Then, we detail the various tasks carried out, divided into bug fixes and functionality extensions, together with a brief description of those aspects that have been worked on but are not functional.

Finally, the report closes with the final conclusions of the project, as well as indications on the future work to be carried out on the application.

As an annex, a user manual of the tool is included, with the aim of helping to understand the behaviour and functionalities of the tool's interface.

Conclusions and future work

This project has been a challenge for us as well as an enriching experience in many aspects. Both students finish the experience very satisfied, with the experience of having participated in the project as well as with the final result. The number of bugs found in the application by ourselves, as well as the correction of both these bugs and those that were already known, have led to a significant improvement in the operation of the DBCASE tool. In addition, several functionalities have been added to facilitate the use of the application by the user. These include the Undo/Redo option and the inclusion of aggregations. At the beginning of the process, when we identified the tasks to be carried out throughout the project, we marked these tasks, as we conceived them as those that would represent the greatest progress for the DBCASE application. They are the tasks that have involved the most effort and the largest amount of code and time. Moreover, as these tasks were started with no previous code base, it was left to us to choose the way in which we should implement them. We consider that the ideas we ended up implementing not only respect the structure of the code but also represent an elegant and coherent solution with the internal functioning of the application.

As previously mentioned, the achievement of the objectives, both those initially set and those identified during the course of the project, has consisted of an incremental work. This means that, although many tasks are listed in the final report, the completion of many of them has consisted of numerous advances and backward steps. Identification of implementation errors, elaboration of the different processes to test the advances, strange exceptions in certain cases, difficulties when integrating parallel work or modifications of other actions when introducing or modifying existing code have been some of the aspects and challenges that we have faced during the performance of the different tasks, both in terms of correcting operation and extending the functionality.

In addition, at first, receiving such a complex project with so many classes and such extensive code was a bit overwhelming. However, a general analysis of the code in the early stages of the process helped us to form a general idea of how the application works internally, and it was easy to identify that the pattern used is the Model-View-Controller programming. The encapsulation of the code typical of this object-oriented programming pattern allows us to understand which package of the project groups the classes that are responsible for implementing each functionality of the application. However, within the same package there are a huge number of different classes, each one well differentiated, which are necessary to understand.

Despite this first analysis, the way to understand in depth the structure of the code and generate a notion of where to include or modify the necessary aspects for a new task, is to proceed to try the different tasks proposed and work with the application to understand all its functionalities. As we worked on the project, we developed a greater familiarity with the tool, as well as a greater knowledge of the code, reducing the time to understand and analyse each new task and optimising the implementation by using all the existing resources in the initial software.

Moreover, an additional aspect that we have obtained with this way of proceeding is that it has allowed us to learn new ways of interacting with the Eclipse tool, discovering techniques and commands to search throughout the project for fragments that we wanted to modify, allowing us to automate the process of detecting the module we wanted to work with to carry out a specific task. Similarly, both to locate this point and to locate errors or check the behaviour of the application during execution (function calls, variable status, variable behaviour, type of elements stored in each collector, etc.), we have worked a lot on debugging the code thanks to the Debug mode offered by Eclipse.

The fact that the project has been developed over several years implies two things. The first is that it has evolved incrementally, this means that many functionalities have been added one after the other, always considering the state of the project in the past. The second is that many people have collaborated in the implementation of DBCASE. For these reasons, complications

could arise when it comes to understanding the code that are common in this type of project. It is advisable to adapt to the way in which the application is implemented, to use a similar nomenclature in the name of methods or variables, to clarify what their function is, for greater readability by future programmers. It is also vital to respect the structure and to maintain coherence and contribute to the understanding of the code by other people, but also to avoid breaking the encapsulation and respect the Model-View-Controller pattern, so that to modify an aspect of a class or a module it is not necessary to make major changes in the rest of the project.

It is also important to know how the DBCASE tool works, how it manages projects through .XML files, how to open, edit and save them, as well as how to locate the paths to the computer where they are located. Performing the task debugging mentioned above not only helps to check the flow of the code dynamically and to locate errors, but it is also very useful to check how these files, which are the ones that store the useful information with which the application works, are being modified.

The development of the project is a process of much waste, of trial and error, including several testing processes of both the code and the application itself. The learning curve grows too slowly in the early stages. The identification of errors is a really expensive task, and it is never certain whether an error was caused by a bug in the newly created code or was a bug carried over from previous versions of the software, which complicates the way in which it has to be resolved. However, it is a great satisfaction to carry out each task and to see the real evolution of the project, with its new functionalities and the relevant reduction of errors. DBCASE is an application with numerous functionalities and with the intention of being used at the Complutense University of Madrid, which motivates us to continue looking for ways to improve it.

On a personal level, we consider this project to be ideal for a work of this nature, such as a Final Degree Project, as it allows a broad review of the different ways of representing and understanding a database, as well as a broadening of knowledge in this field. It has also allowed us to have an experience very similar to that acquired when starting to work on a project in any company. We have had to plan, distribute tasks, understand other people's code and adapt to it. As

we have mentioned throughout this document, we have also acquired important knowledge regarding the use of tools such as Git/GitHub (also used in the business world) or Eclipse. We have also learned a lot of ways of working with Java and possibilities offered by this language unknown to us until now. That is why we consider that the project has both a research part and a clearer implementation part. For all the knowledge reflected and acquired during the completion of this project, as well as for the personal attachment that is generated with it, we recommend those students in search of a Final Degree Project to continue contributing to the improvement of DBCASE.

Because of this, we have written this report with the idea that it will work as a guide for future students to understand the structure of the code in a general way, as well as to develop a notion for completing the tasks that have been left unfinished or that are susceptible to improvement.

As mentioned at the beginning of this section, both students are very satisfied with the end result of the project, as we feel that we have extended and refined a really useful tool for understanding, designing and manipulating databases. However, as is typical in software development projects, not all the effort and time invested can be reflected in the final tasks. Hence, we have decided to include the sub-section **3.3.3 Tareas sin concluir y mejoras**, in which we explain the different ideas and procedures carried out to try to finalise or refine the listed aspects. This sub-section is further intended to help potential students continuing with the project to approach the completion of these tasks.

Finally, in order to introduce possible future work, as indicated above, we encourage the completion of the tasks in section **3.3.3 Tareas sin concluir y mejoras**, taking into account all of the above considerations.

Of course, we also recommend further exploration of the application both to understand how it works and to find possible bugs that need to be fixed. As discussed in several sections, the priorities of the project are divided as follows, according to their priority:

- Correction of bugs and bugs

- Addition of new functionalities.

On the other hand, in terms of functionality extensions, some of the possible additional tasks to be carried out could be:

- Supporting Elmasri's graphical notation for the cardinality of entities in relations. This notation consists of indicating cardinality by a 1 instead of an arrow, and by an N otherwise. An option to switch between the current notation and the Elmasri notation could be included in the tool.

- Include database re-engineering. This task consists of being able to design a database from the logical or physical schema, introducing the necessary tables or queries. Then, the corresponding conceptual schema could then be shown. To undertake this task, it is necessary to code a syntactic processing between the data entered in the logical or physical panels and to generate the necessary messages for the insertion of the elements in the conceptual schema. It would also be necessary to introduce possible error or warning messages.

- Include the option of displaying a performance analysis, with the aim of optimising the designed database. This task can be implemented by keeping track of the elements present of each type and assigning the corresponding cost functions. The user could be shown the total cost of the designed database, so that he can undertake variations if necessary.

- Finally, another possible extension task could be the automatic normalisation of the database designed by the user. The main purpose of this task is to reduce redundancies, ensure data integrity and eliminate inconsistent dependencies.

Contribuciones personales

Miguel Derecho Prieto

Como contribuciones personales, Miguel ha identificado todos los errores y excepciones existentes en la aplicación, que inicialmente no se tenían localizados. Por consiguiente, ha sido el encargado de solucionarlos. Además, Miguel ha solucionado los errores que fueron identificados inicialmente y marcados como objetivos. En definitiva, Miguel ha realizado todas las tareas presentes en la subsección **3.3.1 Correcciones.**

Por otro lado, Miguel ha sido el encargado de idear e implementar íntegramente la tarea de la cadena *Deshacer/Rehacer*. Esta tarea ha sufrido diversas modificaciones a lo largo del trabajo, empezando con su versión inicial, implementada a través de la identificación del último mensaje de acción realizada por el usuario, y el posterior envío al controlador del mensaje "opuesto". Además, junto con la ayuda del tutor, se han ido identificando y solucionando diversos errores y mejoras, hasta la versión final presentada.

Continuando con las tareas de extensión de la funcionalidad de la herramienta, Miguel ha realizado las siguientes mejoras:

- Permitir usar las teclas Y y N en los cuadros de dialogo Yes/No. Además, se ha incluido la posibilidad de moverse sobre los botones con las flechas del teclado.
- Introducir la opción de abrir el manual de usuario en el menú Help, salvo la inserción de las imágenes que se muestran.
- Tarea de Copiar y Pegar elementos, junto con los atajos Ctrl+C y Ctrl+V.
- Añadir barra de iconos
- Añadir elementos de menú para todas las acciones.
- Añadir menú Vista a la barra de menús.
- Adición de la carpeta de ejemplos, con ejemplos completos de diseños conceptuales.

- Guardar un proyecto de *backup* si el proyecto actual no ha sido guardado durante los últimos 10 minutos.
- Actualizar el panel de información *About DBCASE* con toda la cronología de los diferentes trabajos.

Además, Miguel ha trabajado en las siguientes tareas no concluidas:

- Introducción manual del valor de zoom.
- Modo *Cuadrícula*.
- Inserción de zoom en los paneles lógico y físico.
- Intentar indicar la clave candidata presente como restricción perdida en el modelo lógico, como combinación de claves candidatas de una misma entidad.
- Solucionar el error existente sobre la restricción de cardinalidad 1 de entidades en relaciones.

Por otro lado, Miguel ha ayudado en la identificación de aspectos e ideas de implementación para la inclusión de las agregaciones, pero no participó en el desarrollo de código.

En cuanto a la memoria, Miguel ha redactado las siguientes secciones y capítulos:

- Resumen (Abstract)
- Capítulo 1: Introducción (Introduction)
    - 1.1 Motivación (Motivation)
    - 1.2 Objetivos (Goals)
    - 1.3 Plan de trabajo (Work plan)
    - 1.4 Estructura de la memoria (Structure of the report)
- Sección 2.1 Base teórica del proyecto
- Sección 2.5 Librerías
- Sección 3.1 Cronología del trabajo
- De la sección 3.3 Desarrollos, Miguel ha documentado las explicaciones de las implementaciones realizadas por él, así como aquellas no concluidas satisfactoriamente.
- Manual de usuario

Arturo Ibáñez Martínez

En la parte inicial del proyecto, Arturo se encargó de familiarizarse con la herramienta Git, así como de entender el funcionamiento de GitHub para poder trabajar de forma cooperativa en el proyecto, permitiendo una exportación e importación más eficiente del código.

La primera tarea relativa al código del proyecto se corresponde con la inclusión del manual de usuario con imágenes ilustrativas del funcionamiento del mismo.

También se encargó de la tarea referida al problema: "No aparece la flecha en relaciones recursivas con cardinalidad 1", lo cual se corresponde con un error de las versiones anteriores.

En cuanto a nuevas funcionalidades, Arturo creó el menú que permite abrir archivos recientes, así como el botón de la interfaz gráfica que permite eliminar entidades, relaciones, atributos y agregaciones. Esto último no se correspondía con una tarea propuesta pero mientras trabajaba con las agregaciones pensó en la manera de eliminar una agregación sin tener que borrar los elementos incluidos en ella y se le ocurrió esta idea. Además, permite borrar en menos pasos y más rápido los elementos deseados.

Por último, Arturo se dedicó los últimos meses del proyecto a incluir todas las funcionalidades posibles de las agregaciones. Al contar con una nula base en cuanto a código para esta tarea, resultó imposible una repartición de tareas para este apartado, ya que eran muchas las clases que había que ir creando en paralelo para ir comprendiendo cómo incluir estas funcionalidades en el proyecto, sobre todo la cantidad de mensajes que debían crearse para comunicar las nuevas clases con el controlador, e incluso cómo modificar las clases ya existentes para que tengan en cuenta la existencia de las nuevas. Por tanto, las funcionalidades relativas a la existencia de agregaciones se corresponden con esta tarea:

- Añadir, renombrar y eliminar agregación en menú conceptual.
- Representación en el panel de elementos.
- Añadir atributo a agregación desde el botón *Añadir atributo*.
- Editar y eliminar atributos.

- Eliminar la agregación cuando se ha eliminado su relación.

En cuanto a la memoria, Arturo ha redactado las siguientes secciones y capítulos:

- Sección 2.2 DBCase a través de los años
- Sección 2.3 Frameworks y programas
- Sección 2.4 Lenguajes
- De la sección 3.3 Desarrollos, Arturo ha documentado las explicaciones de las implementaciones realizadas por él, así como aquellas no concluidas satisfactoriamente.
- Capítulo 4 – Conclusiones y trabajo futuro
- Participación en Manual de usuario

Bibliografía

[1] Teoría proporcionada por el tutor Fernando Sáenz Pérez

[2] https://docs.oracle.com/search/?category=java&q=

[3] https://www.oracle.com/database/technologies/appdev/jdbc.html

[4] https://jung.sourceforge.net/

[5] https://www.jgoodies.com/freeware/libraries/looks/

[6] https://jsoup.org/apidocs/org/jsoup/parser/package-summary.html

[7] https://dev.mysql.com/doc/connector-j/8.0/en/

Anexo - Manual de usuario

Manual de usuario

El objetivo principal de la presente sección es permitir al lector entender el funcionamiento de la herramienta DBCASE, así como todas sus funcionalidades y posibilidades.

A.1 Ventana general

Al ejecutar la aplicación, suponiendo que se tiene seleccionada la vista completa (más adelante se explicarán los tipos de vistas), se abre una ventana dividida en 6 paneles distintos, como se muestra en la figura 21:

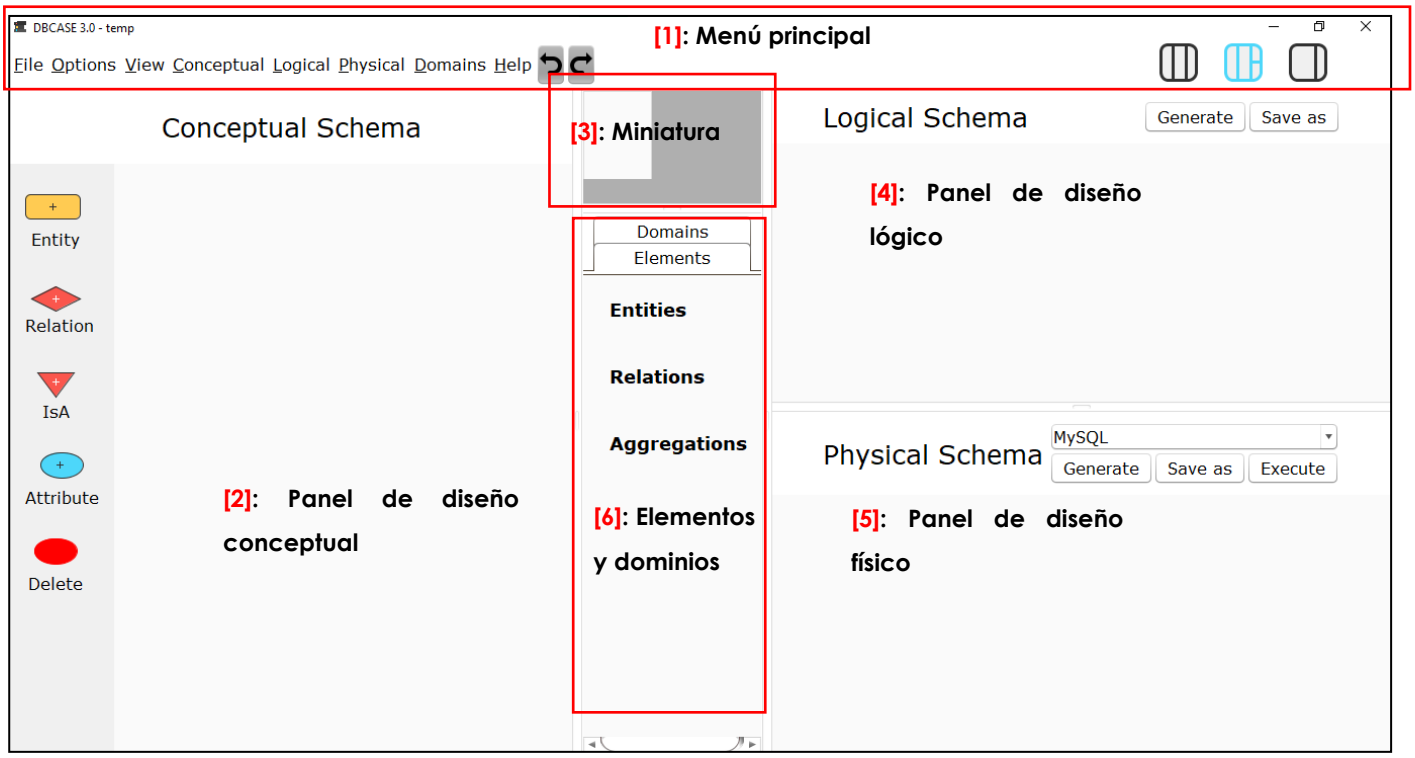

Figura 21. Vista Completa de DBCASE

1. En el menú principal se muestran los menús habituales de las aplicaciones gráficas (File, Options, View Help), así como menús propios de cada uno de los esquemas presentes. Además, incluye los botones para deshacer y rehacer modificaciones y para cambiar la vista de la aplicación.
2. El esquema conceptual se corresponde con el panel principal de la herramienta. Es donde se modela la base de datos a través de los elementos descritos en la sección 2.1 Base teórica del proyecto.

3. En el panel miniatura se muestra una representación del esquema conceptual, permitiendo mover el esquema a los límites no visibles en el panel anterior.
4. El esquema lógico muestra la traducción, del modelo Entidad-Relación introducido en el esquema conceptual.
5. El esquema físico muestra las consultas necesarias para las construcciones de las tablas que componen la base de datos modelada en el panel conceptual.
6. Esta ventana muestra los elementos (entidades, relaciones, atributos y agregaciones) existentes en el modelo. También se incluyen los dominios que se pueden asociar a un atributo, tanto los incluidos por defecto en la aplicación, como los creados por el usuario.

A.2 Menú principal

Como se ha indicado anteriormente este menú está compuesto por los siguientes submenús:

- **Submenú File (Archivo)** (figura 22): DBCASE funciona con ficheros XML. Este menú permite ejecutar las acciones habituales sobre archivos: Nuevo proyecto, Abrir, Guardar, Guardar Como, Imprimir, Exportar (como imagen JPEG), Abrir proyectos pendientes y Abrir Ejemplos, donde se muestran ejemplos completos sobre distintas bases de datos. Además, permite salir de la aplicación.

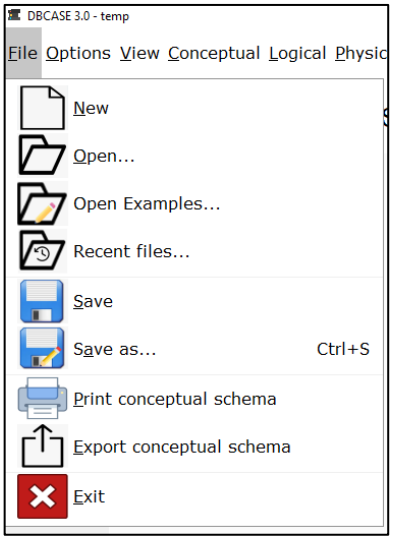

Figura 22. Menú File

- **<u>Submenú Options (Opciones)</u>** (figura 23): permite cambiar el tema de la herramienta (dark o light), cambiar el idioma de la aplicación (español o inglés), marcar los atributos que pueden ser nulos (mediante un asterisco), modo confirmar las eliminaciones (con cada eliminación de un elemento surge una ventana para que el usuario confirme si desea la eliminación) y acceder en modo soporte.

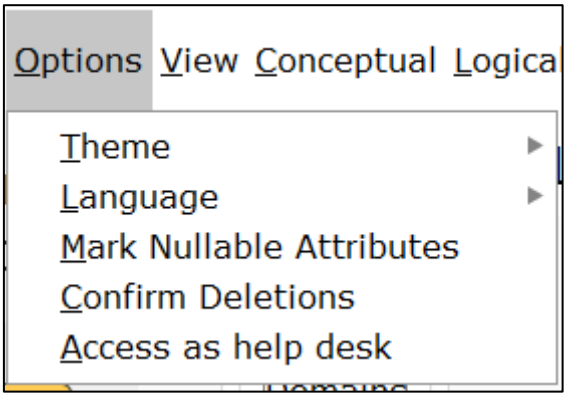

Figura 23. Menú Options

En la figura 21 anterior se muestra la interfaz de la herramienta en tono claro, a continuación, se introduce como es la vista en modo oscuro:

Figura 24. Vista oscura de DBCASE

- **<u>Submenú View (Vista)</u>** (figura 25): permite modificar las vistas del aplicativo en cuanto a mostrar u ocultar los diferentes paneles. En las figuras previas se muestra la vista completa. A continuación, en las figuras 26 y 27, se muestran las otras dos vistas. La opción insertar cuadricula y variar el nivel de zoom se corresponden con tareas sin completar, como se indicó en el apartado **3.3.3 Tareas sin concluir y mejoras**.

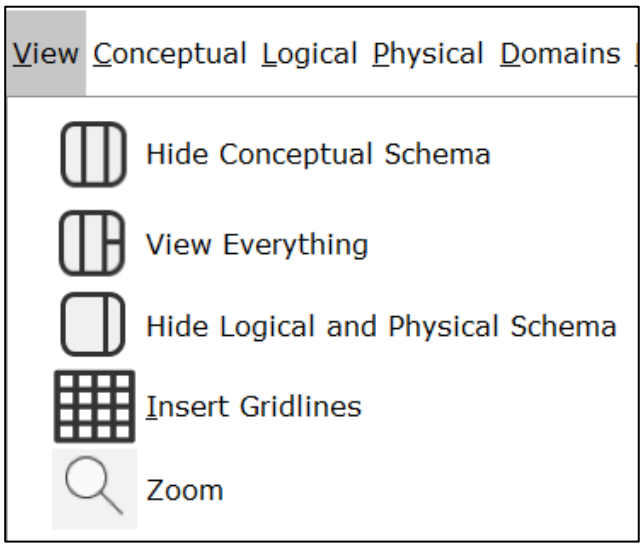

Figura 25. Menú View

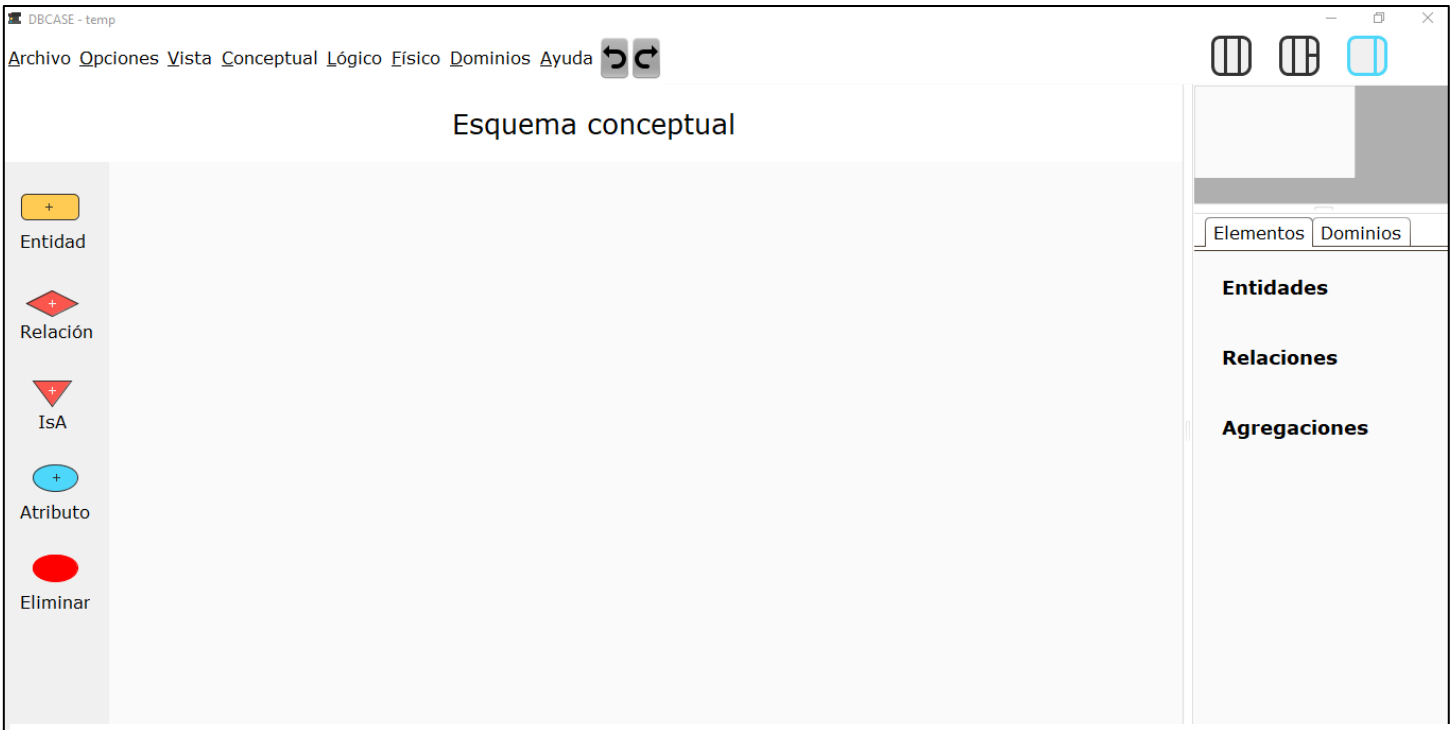

Figura 26. Vista Conceptual

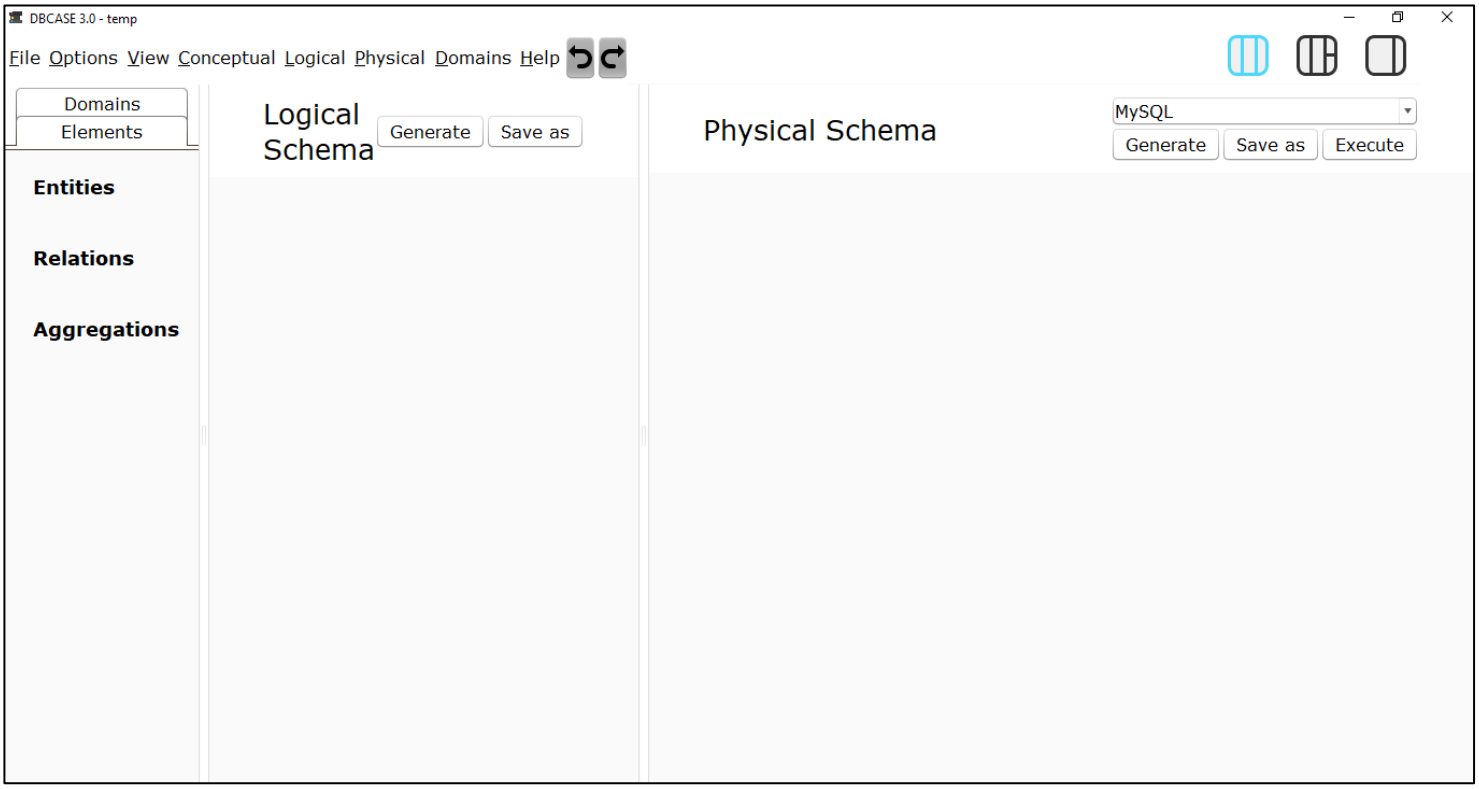

Figura 27. Vista Programador

- Los submenús **Conceptual, Logical (Lógico), Physical (Físico) y Domains (Dominios)**, permiten realizar las acciones propias de los paneles correspondientes. Posteriormente se ahondará en sus funcionalidades.

- **<u>Submenú Help (Ayuda)</u>** (figura 28): permite mostrar al usuario una breve introducción a DBCASE (similar a la sección **2.2 DBCASE a través de los años**), un breve manual de usuario y comunicar posibles incidencias identificadas a través de correo electrónico.

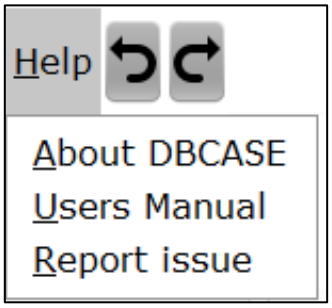

Figura 28. Menú Help

- Los botones para Deshacer y Rehacer se muestran en la figura anterior, mientras que el menú principal también presenta botones para cambiar las vistas explicadas previamente (figura 29).

Figura 29. Botones Vistas

A.3 Panel de diseño conceptual.

A continuación, se muestran las siguientes acciones posibles a la hora de modelar una base de datos a través de diagramas Entidad-Relación.

- Para insertar elementos, existen tres formas posibles:
    - Pulsando sobre el elemento que se desee introducir, en el menú de la izquierda.
    - Haciendo clic derecho sobre el ratón en el panel y seleccionando la opción correspondiente. De este modo también se pueden crear dominios propios.
    - Utilizando el submenú Conceptual introducido anteriormente

  Además, los atributos pueden ser también insertados haciendo clic derecho sobre el elemento para el que se quieren crear. Esto se muestra posteriormente, en la figura 31.

También es posible insertar nuevos elementos copiando y pegando elementos ya existentes.

En la figura 30 se muestran las diferentes opciones existentes para insertar elementos en DBCASE.

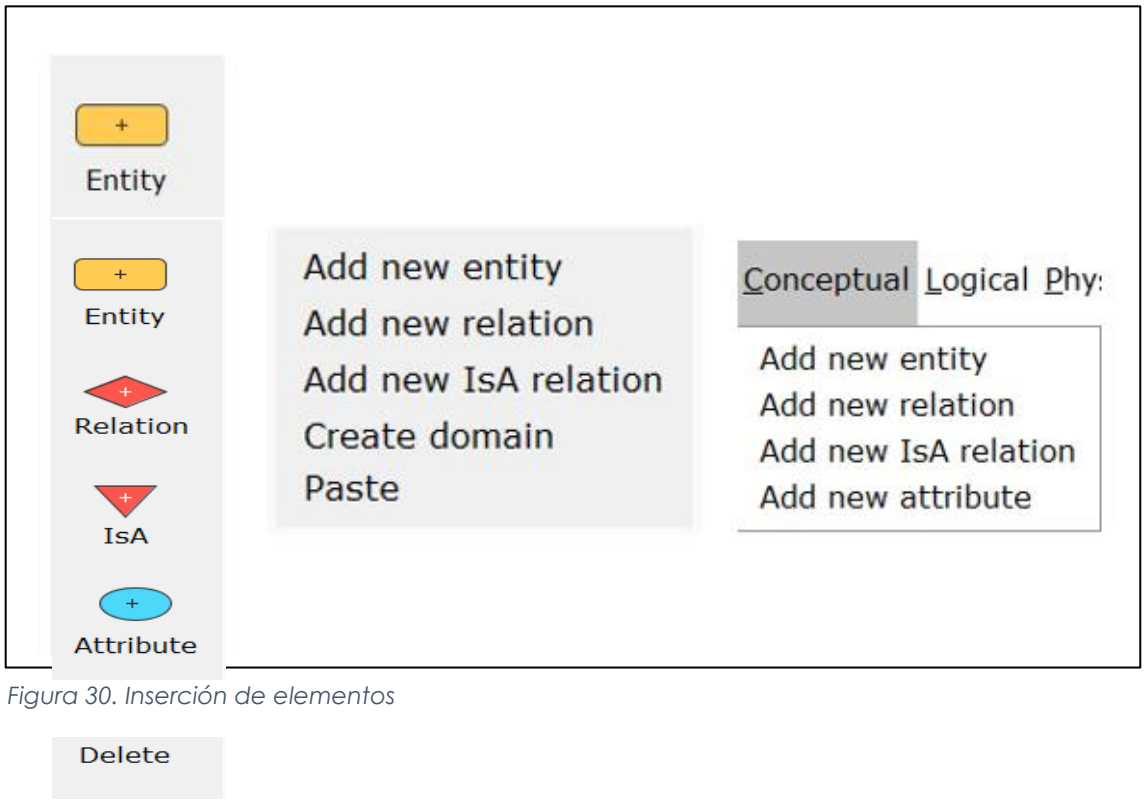

Figura 30. Inserción de elementos

- Para eliminar elementos, se selecciona el elemento (o elementos) que se desee eliminar, y se pulsa sobre la tecla *Supr* del teclado. Además, se puede también eliminar un elemento haciendo clic derecho sobre el elemento a eliminar, y seleccionando la opción correspondiente. Por último, se puede pulsar el botón Delete/Borrar y seleccionar cualquier elemento para ser eliminado. En la figura 31 insertada a continuación se muestra también la forma adicional para insertar atributos, comentada anteriormente.

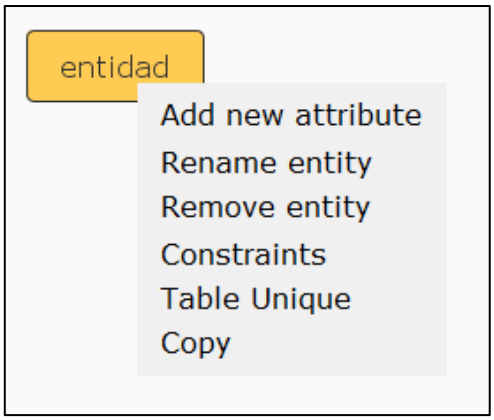

Figura 31. Eliminación de elementos

- Para modificar elementos ya existentes, se puede hacer doble clic sobre el elemento deseado a modificar, o nuevamente a través de hacer doble clic sobre el elemento a modificar. Las modificaciones sobre entidades y relaciones únicamente se corresponden con renombramientos o adición de restricciones manuales, como se muestra en las opciones de la figura 31. Por otro lado, los atributos presentan gran cantidad de opciones de edición. En la figura 32 se muestran las distintas opciones que existen al hacer clic derecho sobre un atributo, además de la ventana de modificación que surge al hacer doble clic sobre un atributo. Esta ventana es análoga a la que surge para la creación.

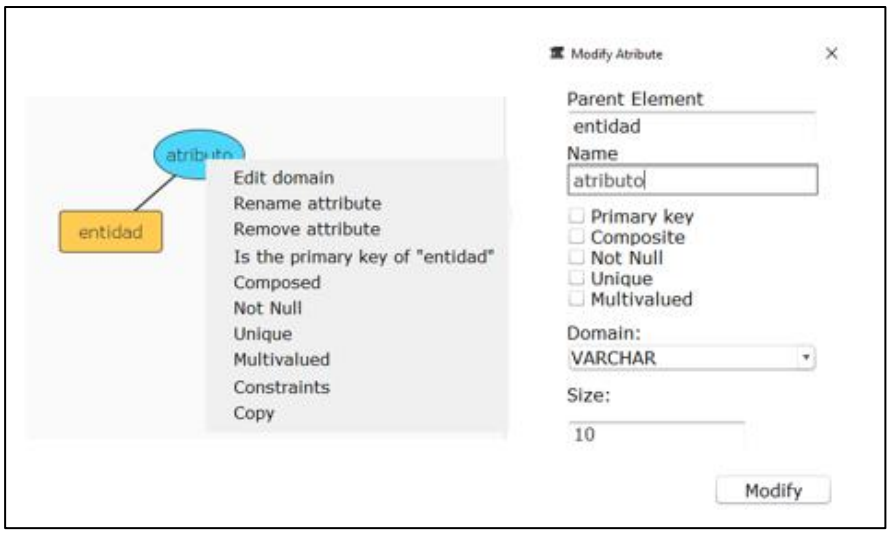

Figura 32. Modificación de atributos

- Para asociar entidades a relaciones, es necesario hacer clic derecho sobre la relación correspondiente y, en la ventana emergente,

seleccionar la entidad deseada, junto con las restricciones de cardinalidad, participación y rol que considere el usuario. Para editar las restricciones y los roles, se procede del mismo modo. A continuación, en la figura 33, se muestra el menú desplegable asociado a relaciones junto con la ventana de edición emergente.

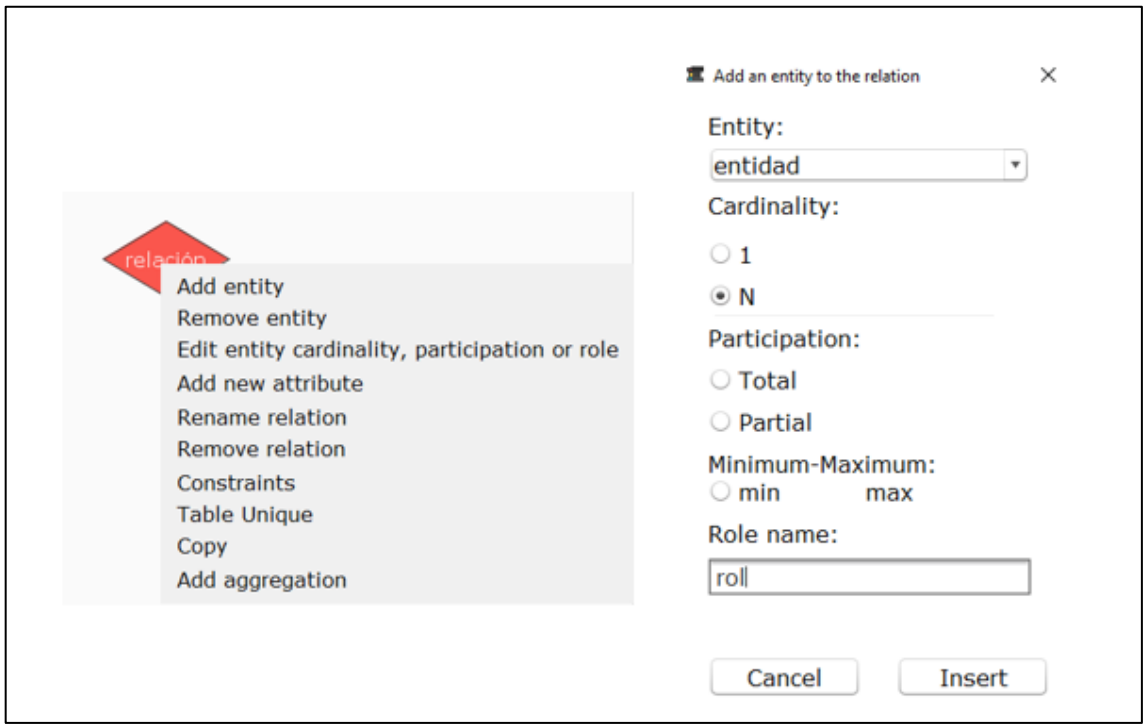

Figura 33. Añadir entidad a relación

- Para añadir una entidad débil se procede de una de las formas que se introdujo anteriormente. A continuación, en el cuadro de diálogo, se selecciona la opción de entidad débil, se indica la entidad fuerte correspondiente, así como la relación débil que se creará y a través de la cual se asociaran las entidades. En la figura 34 se muestra la ventana emergente donde se indica que una entidad es débil.

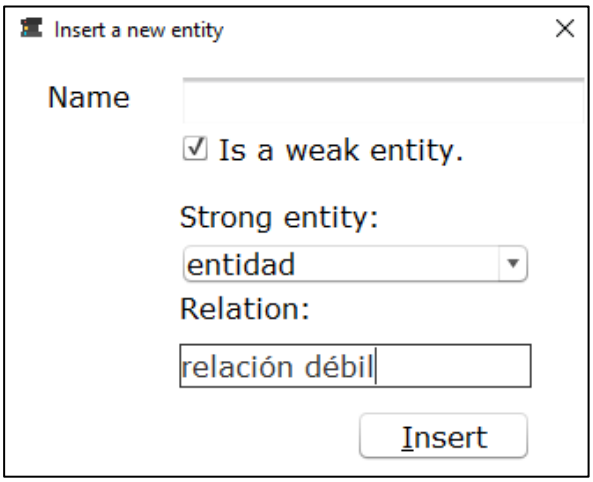

Figura 34. Insertar entidad débil

- Para añadir una agregación se debe seleccionar una relación (que no pertenezca a una agregación) y hacer clic derecho. A continuación, pulsamos en *Añadir agregación* y aparece una ventana para introducir el nombre de la misma. Al pulsar el botón confirmar de dicha ventana se agrega la relación y sus entidades a la nueva agregación.

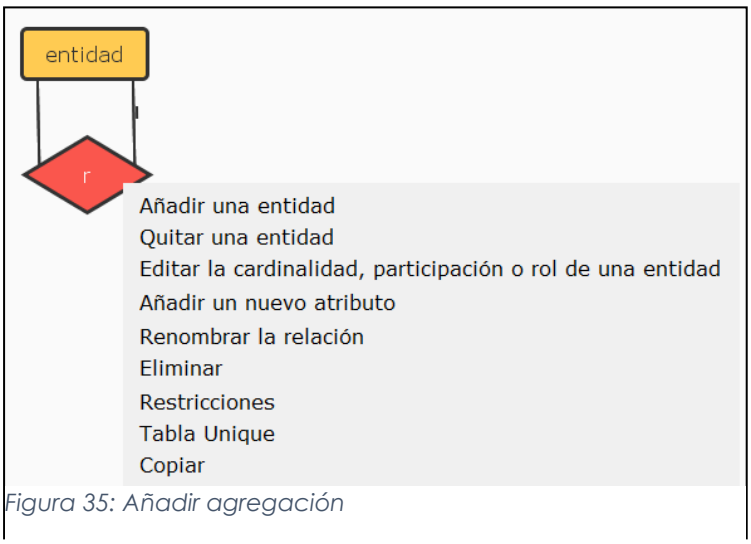

Figura 35: Añadir agregación

Para eliminar una agregación, a parte de la opción del botón *Delete*/Eliminar, se puede hacer clic derecho en la relación de la agregación que se desea eliminar. Al pertenecer a una agregación, no aparece la opción *Añadir agregación*, sino que permite eliminar esta agregación sin borrar ningún elemento de la misma. También permite

modificar el nombre de la agregación mediante la opción *Renombrar agregación*. Al pulsar en este submenú aparece un cuadro que insta a introducir el nuevo nombre. Al pulsar en *Confirmar* se cambia el nombre en caso de que cumpla con las mismas condiciones que se exigían para poner nombre a una agregación recién creada.

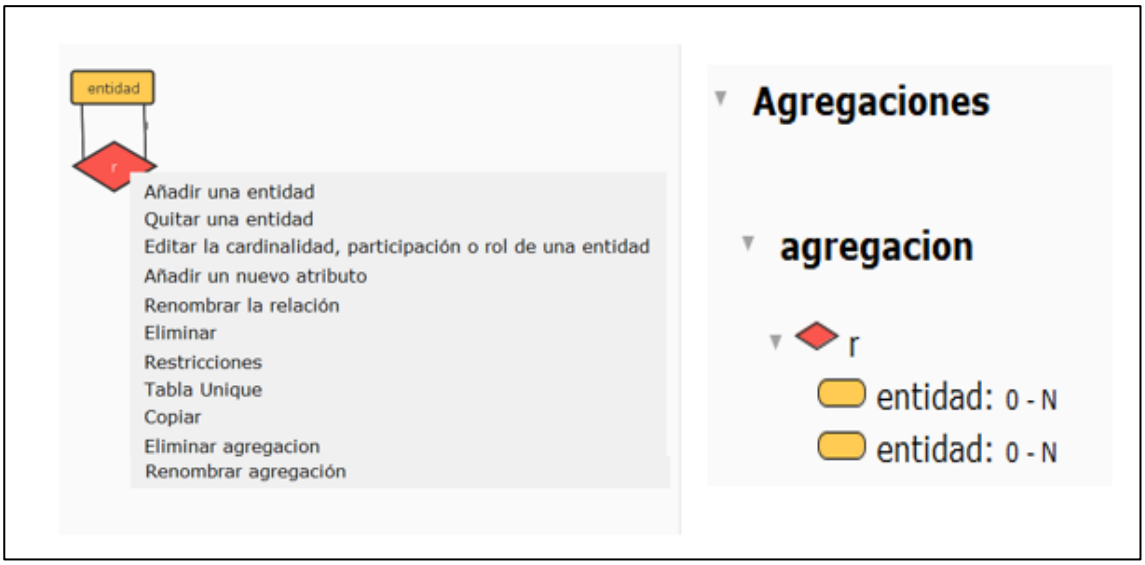

Figura 36: Eliminar y renombrar agregación

A.4 Panel de diseño lógico

Como se explicó anteriormente, este panel muestra la traducción del diagrama Entidad-Relación modelado en el panel conceptual**.** Para ello, se debe pulsar en el botón *Generar* de este panel, o a través del submenú *Logical* presentado anteriormente. Asimismo, se permiten guardar las tablas generadas en formato txt mediante el botón *Guardar como*.

En este panel también se muestran ciertos avisos existentes a la hora de generar el modelo lógico, así como los posibles errores existentes en ciertas tablas, como se muestra en la figura 37 adjunta a continuación.

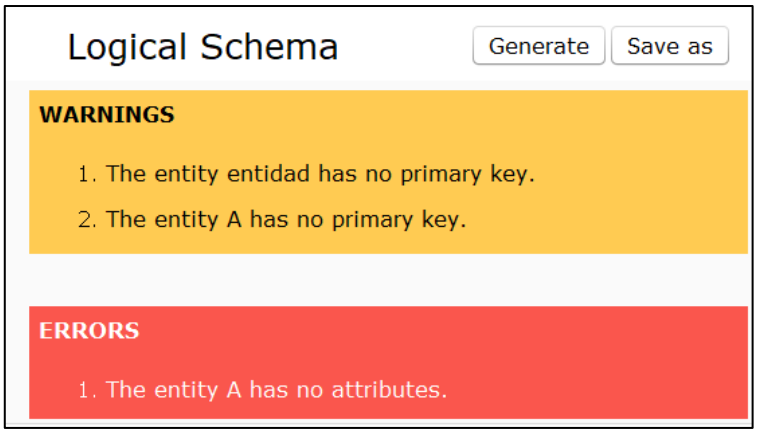

Figura 37. Esquema lógico. Avisos y errores

Además de las tablas traducidas, también se muestran, en caso de existir tablas con claves externas, las restricciones de integridad referencial. Se muestra un ejemplo en la figura 38 adjunta a continuación.

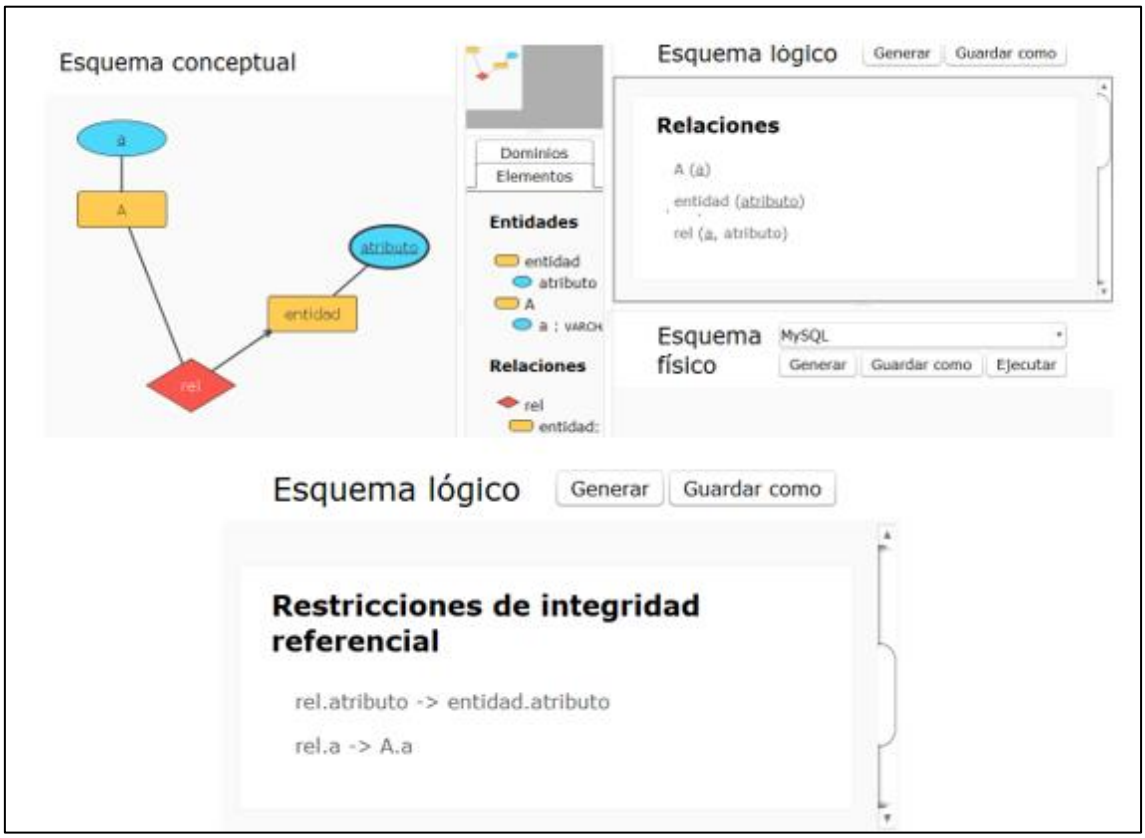

Figura 38. Restricciones de integridad referencial

Por último, este panel también muestra las restricciones perdidas, como es el caso de relaciones con participación (1:1). Nuevamente, adjuntamos un ejemplo (figura 39) donde se muestran las restricciones perdidas.

Figura 39. Restricciones perdidas

A.5 Panel de diseño físico

Al igual que el panel lógico, en esta ventana se muestran las consultas necesarias para representar físicamente la base de datos modelada en el esquema conceptual. Incluye los botones análogos de *Generar* y *Guardar Como d*el panel lógico, y del mismo modo puede usarse el submenú *Physical* (Físico) introducido anteriormente. A través de un menú desplegable, permite seleccionar el gestor de base de datos deseado. Además, este panel incluye la opción de ejecutar las consultas sobre bases de datos reales, para lo que es necesario, tras pulsar el botón *Ejecutar*, añadir una conexión a SGDB. Se adjunta a continuación un ejemplo donde se añade una conexión (figura 40).

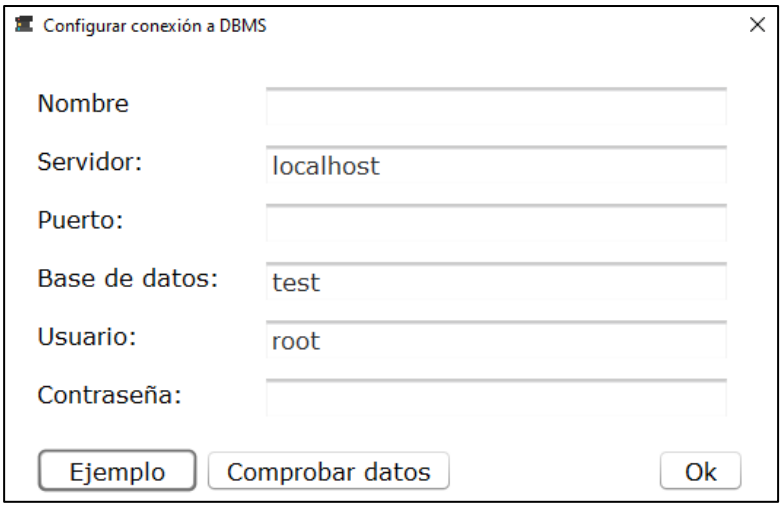

Figura 40. Conexión a DBMS

Las consultas mostradas en este panel se dividen en:

- **Tablas**: asociadas a la eliminación y creación de las tablas
- **Tipos enumerados**: para asociar los dominios creados por el usuario
- **Claves**: consultas para añadir restricciones claves primarias y foráneas.
- **Restricciones**: consultas para añadir restricciones de unicidad.

Al igual que el panel lógico, también muestra los posibles avisos y errores existentes a la hora de modelar el diseño conceptual.