Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención: tecnologías de la información.

Método de transformación de un esquema Entidad-Relación en una base de datos con MongoDB.

Autor:
Rubén de Diego Varona

Tutora:
Dra. M.ª Mercedes Martínez González

A mi padre y mi hermana,
sin su ayuda, comprensión y cariño
no habría llegado hasta aquí.

Resumen.

La finalidad de este trabajo de fin de grado es el de proponer una serie de reglas para convertir un diagrama Entidad-Relación en una base de datos no relacional, en concreto con *MongoDB*.

Para lograr dicho objetivo se estudiarán los conceptos teóricos de dicho diagrama, así como la funcionalidad y características necesarias de *MongoDB* para posteriormente definir un método de transformación de los elementos presentes en un diagrama ER en sus correspondientes elementos en *MongoDB*, ejemplificando cada una de las transformaciones.

Posteriormente se realizará la transformación de un diagrama Entidad-Relación completo en una base de datos con *MongoDB* utilizando las reglas previamente definidas.

Palabras clave

Diagrama Entidad-Relación, MongoDB, Bases de datos, NoSQL

Abstract.

The purpose of this final degree project is to propose a series of rules to convert an Entity-Relationship diagram into a non-relational database, specifically with *MongoDB*.

To achieve this objective, the theoretical concepts of said diagram will be studied, as well as the functionality and necessary characteristics of *MongoDB* to achieve said conversion.

Subsequently, the transformation of a complete entity-relationship diagram will be carried out in a database with *MongoDB* using the previously defined rules.

Key words

Entity relationship diagram, MongoDB, Databases, NoSQL

Tabla de Contenidos

Lista de figuras.

Lista de tablas.

Capítulo 1 Introducción y objetivos.

1.1 Motivación.

Debido a los avances tecnológicos de los últimos años en todos los ámbitos de las nuevas tecnologías y en concreto en el de las bases de datos, cada vez más programadores, desarrolladores de aplicaciones y organizaciones están apostando por el uso de bases de datos *NoSQL* para almacenar y gestionar la información.

A pesar de que el término se acuñó en el año 1998, no fue hasta finales de la década del año 2000 cuando estos tipos de bases de datos se empezaron a utilizar de forma significativa.

Actualmente, los sistemas gestores de bases de datos más utilizados son *Oracle* (seguido de *MySQL*, *Microsoft SQL Server* y *PostgreSQL*). Sin embargo, y pese a su relativamente corto periodo de existencia (fue desarrollada en 2007 pero no se consideró una base de datos útil para producción hasta su versión 1.4 lanzada en marzo de 2011) *MongoDB* es el quinto gestor de bases de datos más utilizado a nivel mundial, así como el número uno entre los sistemas *NoSQL*. [1]

A pesar de que actualmente sigue aumentando el número de usuarios que hacen uso de esta tecnología, solo hay que buscar información en la red sobre la migración del modelo tradicional a este sistema gestor de bases de datos, para darnos cuenta de que no son pocos los programadores que desean transferir los datos previamente almacenados utilizando el modelo relacional en bases de datos *NoSQL*, en concreto en *MongoDB*.

La documentación oficial del uso de esta base de datos es bastante escasa, y a pesar de que en el año 2022 apareció una herramienta para migrar del modelo relacional a *MongoDB*, denominada *Relational Migrator*[2], los detalles de su implementación no son públicos, e independientemente, no transforma un esquema Entidad-Relación en una base de datos en *MongoDB*, sino que parte directamente del modelo relacional para realizar dicha migración.

Es por ello que nos parece un tema interesante la propuesta de creación de unas reglas de transformación entre dicho diagrama y una base de datos en *MongoDB*.

A título personal, la principal motivación, es el deseo de aprender la funcionalidad de este nuevo tipo de bases de datos, ya que tanto en nuestra carrera profesional como en la académica se han utilizado, para varios proyectos, bases de datos relacionales, y no se ha tenido la posibilidad de trabajar con estos nuevos sistemas, presentándose en la realización del presente TFG la oportunidad de ampliar los conocimientos en esta tecnología y aportar, mediante las reglas de transformación que definiremos a lo largo de este trabajo, un acercamiento a la conversión entre diferentes modelos.

[1] DB-Engine Ranking. [Recurso online] Disponible en: https://db-engines.com/en/ranking
Fecha última consulta: 01/06/2024

[2] MongoDB Relational Migrator. [Recurso online] Disponible en:
https://www.mongodb.com/products/tools/relational-migrator Fecha última consulta:01/06/2024

1.2 Objetivos

Como hemos visto en la introducción, el objetivo principal de este proyecto es el de definir una serie de reglas para convertir un diagrama Entidad-Relación en una base de datos no relacional, en concreto con *MongoDB*. Cabe decir que las reglas que definiremos a lo largo de este proyecto no son la única manera de afrontar la conversión de un diagrama con características relacionales en una base de datos no relacional, sino que gracias a la flexibilidad que nos ofrecen estas últimas, este acercamiento no es único, pero realizaremos una propuesta de las mismas que sirva de guía para afrontar este proceso.

Para cada una de las reglas propuestas (múltiples para una misma transformación en algunos casos debido a la alta flexibilidad ofrecida por *MongoDB*) se realizará un ejemplo de conversión. Estos ejemplos, cuya finalidad es la de cubrir todas las transformaciones posibles, no pertenecerán a un único universo, algo que dejaremos para un capítulo específico de esta memoria, en el que realizaremos la conversión completa de un diagrama Entidad-Relación en su correspondiente base de datos con *MongoDB*.

Posteriormente, y dado que definiremos una serie de criterios generales para dicha conversión, llevaremos a la práctica las reglas previamente definidas mediante el análisis y posterior transformación de un diagrama Entidad-Relación completo para, de este modo, poder ejemplificar las reglas anteriormente definidas para un mismo universo.

Para conseguir obtener estos objetivos será necesario cumplir los siguientes hitos:

- Afianzar los conocimientos teóricos sobre el modelo Entidad-Relación.
- Estudiar y aprender cómo funcionan las bases de datos *NoSQL*, en concreto *MongoDB*.
- Elaborar las reglas de transformación para cada uno de los elementos pertenecientes al diagrama Entidad-Relación en su correspondiente elemento en *MongoDB*.
- Llevar a cabo un ejemplo práctico en el que se representen los conceptos estudiados y las reglas formuladas.

1.3 Planificación.

A continuación, mostramos la planificación temporal que se ha ido siguiendo realizando para llevar a cabo este TFG. [Tabla 1.1]

Nombre de la tarea	**Descripción**	**Fecha de inicio**	**Fecha de fin**	**Nº días**
Plan de desarrollo del proyecto		**01/08/2023**	**02/09/2023**	**12 días**
Determinar el alcance del proyecto	Determinar los objetivos que se desean alcanzar a la hora de desarrollar este TFG.	02/08/2023	12/08/2023	12 días

Planificación del proyecto		**13/08/2023**	**05/09/2023**	**24 días**
Identificación de tareas	Identificar, estudiar y describir cada una de las tareas o hitos que se van a llevar a cabo durante la ejecución del proyecto.	13/08/2023	15/08/2023	3 días
Determinar la duración de las tareas	Determinar la duración de cada una de las secciones de las que va a constar el proyecto.	16/08/2023	17/08/2023	2 días
Identificación de los recursos.	Identificar los recursos necesarios para la elaboración de cada una de las tares del proyecto.	18/08/2023	18/08/2023	1 día
Instalación y análisis de recursos.	Instalar, analizar y probar los recursos necesarios para la ejecución del proyecto.	19/08/2023	24/08/2023	6 días
Identificación y análisis de riesgos.	Identificar los riesgos a los que nos podemos enfrentar durante el desarrollo del proyecto. Documentar, analizar, describir y trazar una estrategia para enfrentarnos a ellos.	25/08/2023	05/09/2023	12 días
Introducción y objetivos.		**06/09/2023**	**18/09/2023**	**13 días**
Documentación "Introducción y objetivos".	Elaborar la documentación referente a esta primera etapa del proyecto.	06/09/2023	18/09/2023	13 días
Diagrama Entidad-Relación.		**15/09/2023**	**21/10/2023**	**37 días**
Búsqueda de información diagrama Entidad-Relación.	Búsqueda de información del modelo Entidad-Relación. En particular del diagrama Entidad-Relación	15/09/2023	23/09/2023	9 días
Estudio diagrama Entidad-Relación.	Estudio del diagrama Entidad-Relación. Base, teoría y usos del mismo.	24/09/2023	24/09/2023	1 día
Adquisición, instalación y estudio de herramientas necesarias.	Adquirir y usar para su mejor entendimiento una serie de herramientas necesarias para el estudio y posterior ejemplificación de los diferentes casos.	25/10/2023	30/10/2023	6 días
Documentación "Diagrama Entidad-Relación".	Elaborar la documentación referente a este segundo capítulo de la memoria.	01/10/2023	21/10/2023	21 días
MongoDB		**28/10/2023**	**21/11/2023**	**26 días**
Búsqueda de información sobre MongoDB	Búsqueda de la información necesaria para familiarizarse con esta base de datos y su funcionalidad	28/10/2023	06/11/2023	10 días

Documentación "MongoDB".	Elaborar la documentación referente a este capítulo de la memoria	07/11/2023	22/11/2023	16 días
Reglas de transformación		**22/11/2023**	**26/02/2024**	**97 días**
Creación de las bases teóricas de transformación. Ejemplificación de cada una de ellas.	Estudio, creación y ejemplificación de cada una de las reglas de transformación del modelo Entidad-Relación a MongoDB	22/11/2023	02/02/2024	73 días
Documentación "Reglas de transformación de un diagrama Entidad-Relación en una base de datos con MongoDB".	Elaborar la documentación referente a este capítulo de la memoria.	03/01/2024	26/02/2023	24 días
Ejemplo transformación completo.		**28/02/2024**	**20/03/2024**	**22 días**
Creación de un diagrama Entidad-Relación completo.	Creación de un diagrama ER para su posterior transformación.	28/02/2024	29/02/2024	2 días
Creación instancias/colecciones/documentos.	Creación de las instancias que van a poblar la base de datos.	01/02/2024	02/03/2024	2 días
Transformación del diagrama creado en una base de datos con MongoDB.	Creación de las colecciones, documentos y referencias en estos para obtener la base de datos final.	03/03/2024	08/03/2024	6 días
Documentación "Ejemplo de transformación de un diagrama ER en una base de datos con MongoDB".	Elaborar la documentación referente a este capítulo de la memoria.	09/03/2024	20/03/2024	**41 días**
Revisión completa		**19/03/2024**	**27/03/2024**	**9 días**
Revisión, corrección de errores, ampliación de ejemplos		19/03/2024	27/03/2024	9 días
Revisión, corrección de errores, ampliación de la memoria		14/05/2024	22/05/2024	8 días

Tabla 1.1 Planificación temporal del proyecto.

1.3.1 Diagrama de Gantt y cronograma.

A continuación, se muestra el cronograma únicamente para las tareas principales de la tabla anterior. Para cada una de las tareas mostramos el nombre de la misma, la duración en días entre la fecha de comienzo y la fecha de fin y una visualización gráfica de la carga de cada tarea principal a lo largo del tiempo de realización del proyecto. [Figura 1.1]

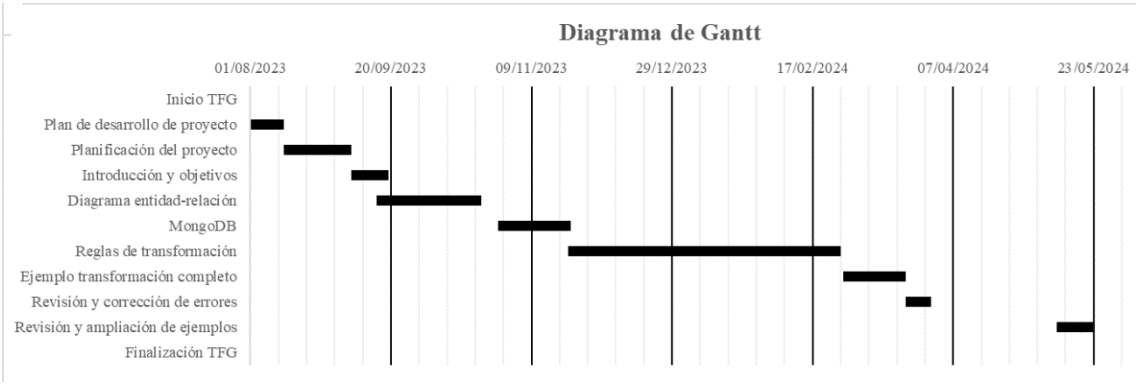

Figura 1.1 Diagrama de Gantt y cronograma.

1.4 Herramientas utilizadas.

Para la realización de este proyecto ha sido necesario el uso de una máquina con doble instalación de sistema operativo. En una partición (usada para realizar los diagramas ER y la memoria de la práctica) *Windows* en su versión 11. En otra partición *Ubuntu*, concretamente la versión 22.04.3 LTS de 64 bits con las características que podemos observar en la tabla [Tabla 1.2] para el proceso de crear la base de datos con *MongoDB* y la realización de todos los ejemplos que acompañan a la misma.

CPU				**RAM**	**Disco**
Marca	**Arquitectura**	**Frec. (GHz)**	**Núcleos**	8 GB	256 GB
Dell Inc. Vostro 3590	x86_64	1.60	8		

Tabla 1.2 Hardware utilizado para la elaboración del proyecto.

Para la creación del cronograma se ha utilizado *Microsoft Project* en su versión 2021.

Para la figura del diagrama de Gantt mostrada en esta memoria se utilizó *Microsoft Office* en su versión 2021 (con los datos previamente creados con *Microsoft Project*) debido a la expiración de la licencia de esta última herramienta.

Para la creación de los distintos diagramas Entidad-Relación se ha decidido utilizar el software *DIA* en su versión 0.97.2

La versión de *MongoDB* con la que se han creado las distintas bases de datos y los ejemplos contenidos en este proyecto ha ido variando a lo largo de su desarrollo siendo la 7.0.7 la utilizada para realizar las conversiones y ejemplos finales incluidos en esta memoria.

Finalmente se ha utilizado una máquina con sistema operativo *Windows* para la creación de la memoria de este TFG.

1.5 Estructura de la memoria.

Capítulo 1: Motivación general y personal sobre la elección del tema de este presente proyecto. Detalles sobre el equipo y software necesarios para su elaboración. Objetivos, planificación general del proyecto y plan de riesgos y soluciones en caso de que se presente alguno de ellos.

Capítulo 2: Introducción y conceptos básicos del diagrama Entidad-Relación. Descripción de los diferentes componentes del mismo, necesarios para posteriormente, a partir de cada uno de ellos, convertirlos en una base de datos con *MongoDB*.

Capítulo 3: Introducción a la base de datos NoSQL *MongoDB*. Terminología, características, componentes y funcionamiento.

Capítulo 4: Estudio de las reglas para transformar cada uno de los componentes de un esquema Entidad-Relación en una base de datos en *MongoDB* manteniendo las restricciones del modelo de partida.

Para cada regla de transformación se realizará un ejemplo de conversión del elemento perteneciente al diagrama ER en su equivalente en *MongoDB*.

Capítulo 5: Conversión de un diagrama Entidad-Relación completo en una base de datos en *MongoDB* aplicando las reglas descritas en el capítulo 4.

Capítulo 6: Conclusiones y líneas de trabajo futuras.

1.6 Gestión de riesgos.

La planificación que se llevará a cabo durante la realización de este proyecto está descrita en el apartado [1.3] de esta misma memoria.

A la hora de realizar dicha planificación se han intentado tener en cuenta todos los posibles riesgos asociados a la elaboración del proyecto.

Una vez identificados los riesgos que pueden aparecer durante el periodo de ejecución del mismo se ha procedido a su división en categorías atendiendo a su tipo y, posteriormente, se ha procedido a la elaboración de una serie de formularios en los que se describen los mismos, se resume su naturaleza, implementa un plan de acción ante su posible aparición y se detallan las posibles actuaciones en caso de producirse. Asimismo, se ha evaluado y asignado una probabilidad de aparición, así como el nivel de impacto que supondría y un plan de contingencia.

1.6.1 Tipos de riesgos asociados al proyecto.

- Riesgos de personal.

  - Enfermedad.
  - Falta de conocimiento.
  - Falta de destreza con las herramientas.

-Riesgos de Hardware.

  - Obsolescencia de materiales.
  - Catástrofe externa.
  - Caída de la red del sistema.

-Riesgos de Seguridad.

  - Desastre natural.

-Riesgos de planificación.

  - Decisiones incorrectas.
  - Recursos insuficientes.

-Riesgos de desarrollo.

  - Cambio de versión de la herramienta

1.6.2 Exposición a los riesgos.

Factor	Probabilidad	Consecuencia	Impacto
Enfermedad.	Media	3-4 días	Medio
Falta de conocimiento.	Alta	6 días	Medio
Falta de destreza con las herramientas.	Media	3 días	Medio
Obsolescencia de los materiales.	Baja	2 días	Bajo
Catástrofe externa.	Baja	15 días	Alto
Caída de la red del sistema.	Media	2 días	Medio
Desastre natural	Baja	10 días	Alto
Decisiones incorrectas.	Media	5 días	Medio

Recursos insuficientes	Media	4 días	Medio
Cambio de versión de la herramienta	Baja	2 días	Bajo
Exposición Total		53 días	

Tabla 1.3 Exposición a riesgos

1.6.3 Formularios de gestión de riesgos.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 01	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de personal
Título de riesgo: Enfermedad.	**Probabilidad:** Media	**Consecuencia:** 3-4 días	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Alto	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: Durante el tiempo de desarrollo del proyecto debido a enfermedad o indisposición del encargado de llevar a cabo el proyecto, es imposible avanzar en el mismo o de realizar las tareas asignadas para ese periodo de tiempo
Contexto del riesgo: Durante el desarrollo del proyecto se produce una parada en el desarrollo del proyecto debido a enfermeda del encargado de llevarlo a cabo.
Análisis del Riesgo: Presente durante todo el tiempo necesario para desarrollar el proyecto.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** El conjunto de tareas que se iban a realizar en el periodo de tiempo descrito son pospuestas, sin que ello lleve un incremento importante en el tiempo destinado al cómputo total del proyecto, ya que la carga de trabajo perdida se repartiría en días posteriores
• Evitación. • Protección. • Reducción. • Investigación. • **Reserva**. • Transferencia.	

Objetivos cuantitativos.	**Plan de acción frente a riesgos:** Reservar el riesgo. La tarea o conjunto de tareas que se iban a realizar en ese periodo de tiempo se realizarán posteriormente aumentando la carga de trabajo hasta la correcta resolución de las mismas.
Indicador: Desde que se para el proyecto debido a Indisposición o enfermedad.	
Umbral: Desde que el encargado de llevar a cabo el proyecto es incapaz de continuar con el debido a una enfermedad o indisposición hasta que se recupera y el desarrollo retoma la normalidad.	
Disipador: Se produce la enfermedad o indisposición y el desarrollo del proyecto se pausa.	

Tabla 1.4 Riesgo 01. Enfermedad.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 02	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de personal
Título de riesgo: Falta de conocimiento.	**Probabilidad:** Alta	**Consecuencia:** 6 días.	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Medio	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: La persona encargada de llevar a cabo el proyecto es incapaz de llevar a cabo alguna de las fases del mismo debido a falta de conocimientos sobre algún determinado aspecto de esa fase
Contexto del riesgo: Llegados a un determinado punto del proyecto, el encargado de llevarlo a cabo es incapaz de realizar alguna de las partes del proyecto debido a la falta de conocimientos sobre el mismo.
Análisis del Riesgo: Riesgo siempre presente debido al tipo de proyecto realizado.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** Estrategia de investigación en fases preliminares de desarrollo del proyecto. En caso de presentarse una vez avanzado reservar tiempo suficiente para afianzar el conocimiento
• Evitación. • Protección. • Reducción. • **Investigación**. • Reserva**.** • Transferencia.	

Objetivos cuantitativos.	**Plan de acción frente a riesgos**: En caso de producirse este riesgo, pedir ayuda externa o aumentar los conocimientos sobre la información relevante a esa parte mediante bibliografía o fuentes externas y necesaria para poder seguir avanzando.
Indicador: el realizador del proyecto es incapaz de realizar una determinada tarea.	
Umbral: Desde el momento en que se conoce el hecho hasta que se soluciona.	
Disipador: Se es consciente de la incapacidad para realizar una determinada tarea.	

Tabla 1.5 Riesgo 02. Falta de conocimiento.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 03	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de personal.
Título de riesgo: Falta de Destreza con las herramientas.	**Probabilidad:** Media	**Consecuencia:** 3 días	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Media	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: Durante la realización del proyecto, la persona que lo está llevando a cabo es incapaz de continuar alguno de los procesos del mismo debido a la falta de destreza o ausencia de conocimiento de alguna herramienta necesaria para la finalización del mismo, más concretamente MongoDB.
Contexto del riesgo: Durante el desarrollo del proyecto se es incapaz de seguir avanzando debido a la falta de destreza en el uso de alguna de las herramientas.
Análisis del Riesgo: Riesgo siempre presente debido a la ausencia de uso en determinadas herramientas, más concretamente MongoDB.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** Investigación y búsqueda de recursos necesarios para solventar dicha carencia fuera de los tiempos previamente previstos en la planificación.
• Evitación. • Protección. • Reducción. • **Investigación**. • Reserva**.** • Transferencia.	

Objetivos cuantitativos.	**Plan de acción frente a riesgos:** Petición de ayuda a una persona con conocimientos superiores y/o aprendizaje activo buscando documentación y literatura referente a MongoDB.
Indicador: El encargado de llevar a cabo el proyecto es consciente de que no puede continuarlo debido a la falta de destreza en alguna de las herramientas.	
Umbral: Desde que se es consciente de la carencia hasta que se solventa por medio del aprendizaje.	
Disipador: Ausencia de capacidades para la realización de alguna de las fases del proyecto.	

Tabla 1.6 Riesgo 03. Falta de destreza con las herramientas.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 04	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de hardware.
Título de riesgo: Obsolescencia de materiales.	**Probabilidad:** baja.	**Consecuencia:** 2 días	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Bajo	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: El equipo utilizado para la realización del proyecto queda obsoleto antes de finalizar el proyecto.
Contexto del riesgo: Durante el desarrollo del proyecto el material utilizado para llevarlo a cabo nos percatamos de la existencia de un problema de obsolescencia en alguno de los materiales necesarios para la consecución de la práctica.
Análisis del Riesgo: Si en la fase de análisis previa al proceso de ejecución del proyecto se tiene en cuenta este riesgo, es poco probable que cause problemas, aunque es un riesgo que va a estar siempre presente.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** Previsión en las fases previas del desarrollo del proyecto.
• **Evitación**. • Protección. • Reducción. • Investigación. • Reserva**.** • Transferencia.	

SEGUIMIENTO DEL RIESGO.

Objetivos cuantitativos.	**Plan de acción frente a riesgos**: mantener el equipo actualizado y prever los elementos hardware y software necesarios para la finalización del proyecto. En caso de obsolescencia o ruptura de componentes hardware reemplazar los mismos en el menor tiempo posible.
Indicador: El equipo es insuficiente para finalizar el proyecto o alguno de los componentes deja de funcionar correctamente.	
Umbral: Desde que el equipo deja de funcionar correctamente o sufre la ruptura de unos sus componentes hasta que el mismo es reemplazado.	
Disipador: El equipo queda obsoleto o alguno de sus componentes imposibilita el correcto desarrollo del proyecto.	

Tabla 1.7 Riesgo 04. Obsolescencia de materiales.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 05	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de hardware.
Título de riesgo: Catástrofe externa.	**Probabilidad:** baja	**Consecuencia:** 15 días.	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Alto	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: Pérdida del material hardware utilizado para la ejecución del proyecto debido a causas externas.
Contexto del riesgo: Durante el desarrollo del proyecto y debido a causas externas (desastre natural, pico de tensión, inundación, incendio…) el equipo se ve comprometido causando su pérdida.
Análisis del Riesgo: Riesgo en principio muy poco probable, pero con consecuencias muy graves que pueden impedir la finalización del proyecto en la fecha prevista.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** No existe plan de acción frente a este determinado riesgo salvo asumir la pérdida del material.
• Evitación. • Protección. • Reducción. • Investigación. • Reserva**.** • Transferencia.	

Objetivos cuantitativos.	**Plan de acción frente a riesgos**: asumir las pérdidas y reemplazar el equipo comprometido.
Indicador: Se produce una catástrofe externa en la que se ve involucrado el hardware necesario para llevar a cabo el proyecto.	
Umbral: Desde que se produce la pérdida del equipo hasta que se reemplazan los materiales comprometidos.	
Disipador: se produce una catástrofe externa y el equipo se ve involucrado.	

Tabla 1.8 Riesgo 05. Catástrofe externa.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 06	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de hardware.
Título de riesgo: Caída en la red del sistema.	**Probabilidad:** media.	**Consecuencia:** 2 días	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Medio	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: la red sufre una caída y deja de funcionar imposibilitando la continuación del proyecto en alguna de las fases en las que es necesaria la misma.
Contexto del riesgo: Durante cualquier fase del proyecto la red de alguno de los sistemas se cae imposibilitando el desarrollo de elaboración del proyecto.
Análisis del Riesgo: Riesgo con alta probabilidad de aparición, pero con muy fácil solución, bien trabajando en local e intentando restaurar la red lo antes posible, o bien buscando otra red distinta.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** Intentar evitar la caída de la red contando con varias redes de conexión, realizando copias de seguridad por si es necesario utilizarlas.
• **Evitación.** • Protección. • Reducción. • Investigación. • Reserva. • Transferencia.	

SEGUIMIENTO DEL RIESGO.

Objetivos cuantitativos.	**Plan de acción frente a riesgos:** tener copias de seguridad en caso de ser necesarias para trabajar en local hasta el restablecimiento de la red.
Indicador: La red deja de estar operativa.	
Umbral: desde que la red deja de estar operativa hasta que se sustituye por otra o esta se recupera.	
Disipador: Ocurre la caída de la red del sistema.	

Tabla 1.9 Riesgo 06. Caída en la red del sistema.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 07	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de seguridad.
Título de riesgo: Desastre natural.	**Probabilidad:** baja	**Consecuencia:** 10 días.	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Alto	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: se pierde material del cual se estaba haciendo uso en proyecto por un desastre natural.
Contexto del riesgo: Sucede un desastre natural que involucra los activos del proyecto.
Análisis del Riesgo: Es un riesgo con una probabilidad baja de aparición, pero con consecuencias desastrosas.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** Asumir las pérdidas físicas ya que no está en nuestra mano prever ciertos desastres.
• Evitación. • Protección. • Reducción. • Investigación. • Reserva**.** • Transferencia.	

Objetivos cuantitativos.	**Plan de acción frente a riesgos:** Asumir las pérdidas físicas ya que no está en nuestra mano prever ciertos desastres.
Indicador: Pérdida de material físico o lógico referente al proyecto.	
Umbral: Desde la pérdida de material hasta la resolución del problema.	
Disipador: Cuando se produce el desastre y debido a ello las pérdidas asociadas.	

Tabla 1.10 Riesgo 07. Desastre natural.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 08	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de planificación.
Título de riesgo: Decisiones incorrectas.	**Probabilidad:** media	**Consecuencia:** 5 días	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Medio.	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: En etapas iniciales del proyecto se toman decisiones que no han sido correctas y que salen a la luz en etapas críticas del proyecto, pudiendo retrasar la entrega final del proyecto o incluso cambiar la planificación inicial.
Contexto del riesgo: En un momento del desarrollo del proyecto se decide no seguir una errónea planificación inicial.
Análisis del Riesgo: Riesgo del que depende el plazo de entrega del proyecto y que, de ocurrir, podría originar cambios en la planificación inicial del mismo. La probabilidad de ocurrir no es muy alta, pero sí sus consecuencias.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** Estrategia de evitación realizando una correcta planificación inicial y realizando comprobaciones periódicas.
• **Evitación**. • Protección. • Reducción. • Investigación. • Reserva**.** • Transferencia.	

Objetivos cuantitativos.	**Plan de acción frente a riesgos:** Realizar un seguimiento continuo del proyecto para confirmar que la planificación inicial está siendo llevada a cabo.
Indicador: Una decisión tomada en fases iniciales del proyecto no ha sido la correcta.	
Umbral: Desde la toma de dicha decisión hasta la resolución del problema causante por la misma.	
Disipador: Problema emergente por la toma de una decisión errónea.	

Tabla 1.11 Riesgo 08. Decisiones incorrectas.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 09	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de planificación.
Título de riesgo: Recursos insuficientes.	**Probabilidad:** media.	**Consecuencia:** 4 días	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Medio.	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: Los recursos son insuficientes para llevar a cabo una determinada tarea.
Contexto del riesgo: Durante la realización del proyecto, y llegados a un determinado punto, se observa que no hay recursos suficientes para llevar a cabo una determinada actividad.
Análisis del Riesgo: Este riesgo no debería ser muy probable si las tareas de planificación han sido correctamente llevadas a cabo, pero siempre existe la posibilidad de producirse debido a un fallo o error en dicha etapa.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** Prevención en etapas iniciales del proyecto (planificación del mismo). En caso de ocurrir intentar obtener el recurso faltante en el menor espacio de tiempo posible.
• **Evitación**. • Protección. • Reducción. • Investigación. • Reserva**.** • Transferencia.	

SEGUIMIENTO DEL RIESGO.

Objetivos cuantitativos.	**Plan de acción frente a riesgos:** Realizar una primera planificación y tenerla siempre presente para identificar la ausencia o necesidad de dicho recurso y reponerlo lo antes posible.
Indicador: Se necesita un recurso del que no se dispone.	
Umbral: Desde la percepción de la ausencia de dicho recurso hasta su obtención.	
Disipador: Momento en el que se detecta la falta del recurso.	

Tabla 1.12 Riesgo 09. Recursos insuficientes.

FORMULARIO DE GESTIÓN DE RIESGOS.

Número: 10	**Fecha:** 09/2023		**Categoría de riesgo:** Riesgos de desarrollo.
Título de riesgo: Cambio de versión de herramienta.	**Probabilidad:** baja	**Consecuencia:** 2 días	**Marco de tiempo:** Todo el proyecto
Proyecto: TFG Rubén de Diego Varona	**Fase:** Todas.	**Impacto:** Bajo	

EVALUACIÓN DEL RIESGO.

Descripción del riesgo: Se puede producir un cambio de versión de alguna de las herramientas objeto de estudio del proyecto, más concretamente MongoDB o algunas de las herramientas necesarias para las diferentes partes del análisis, pudiendo pasar a ser la misma de pago.
Contexto del riesgo: En un momento concreto del desarrollo del proyecto se necesita una versión superior o inferior de una determinada herramienta por problemas que puedan aparecer con dicha herramienta.
Análisis del Riesgo: Es un riesgo directamente proporcional al número de herramientas que se utilicen, se puede evitar manteniendo el software siempre actualizado, o bien manteniéndose en una versión estable.

PLANIFICACIÓN DEL RIESGO.

Estrategia:	**Plan de acción frente a riesgos:** actualizar la herramienta siempre que sea posible y tener un backup con versiones anteriores por si se necesita su uso en algún momento del desarrollo. Investigar posibles problemas de las diferentes versiones.
• **Evitación**. • Protección. • Reducción. • Investigación. • Reserva**.** • Transferencia.	

Objetivos cuantitativos.	**Plan de acción frente a riesgos:** Realizar copias de seguridad (en la medida de lo posible) de las diferentes versiones del software utilizado durante el desarrollo del proyecto y estar informado de los distintos changelogs de las versiones más modernas.
Indicador: Errores debidos a las versiones de las herramientas	
Umbral: Detección del fallo hasta solución del mismo.	
Disipador: Se detecta un fallo debido a la versión del software.	

Tabla 1.13 Riesgo 10. Cambio de versión de la herramienta.

Capítulo 2 Diagrama Entidad-Relación.

La finalidad de este proyecto es definir una serie de reglas para transformar un diagrama Entidad-Relación en una base de datos con *MongoDB*.

Para lograr este objetivo es necesario, en primer lugar, conocer las características de dicho diagrama y los fundamentos teóricos en los que se basa.

En este capítulo explicaremos los conceptos básicos del modelo Entidad-Relación, del cual depende el diagrama asociado, el cual será el punto de partida para la realización de la transformación en una posterior base de datos.

Se estudiarán los dos componentes principales del diagrama: las entidades y las relaciones, así como sus características propias, los distintos tipos de atributos, las relaciones entre entidades dependiendo de su participación en las mismas, las restricciones a dicho diagrama y algunas de las características extendidas que fueron implementándose a lo largo de los años tras la aparición de este modelo.

2.1 Modelo Entidad-Relación.

En el año 1977 Peter Chen, un informático especializado en la teoría de las bases de datos, publicó un artículo *"The entity-relationship model: towards a unified view of data"*.[3] Esta es la primera vez que aparece el concepto Entidad-Relación. En este artículo Chen propuso un nuevo modelo de datos [2]. Dicho modelo describe aspectos de un dominio específico de conocimiento, relacionados entre sí, cuya información es necesaria para almacenarla en una futura base de datos

Su principal finalidad era ofrecer una técnica basada en esquemas como herramienta para diseñar bases de datos. Los componentes del diagrama son dos conceptos principales: el de entidades y el de relaciones, así como una serie de restricciones de ellas.

Hasta ese momento se consideraban tres principales modelos da datos: el modelo de red, el relacional y el de conjunto de entidades. Chen consideraba que cada uno de esos tres modelos tenía una serie de ventajas y desventajas. El *modelo de red* proporcionaba una visión natural de los datos debido a la distinción que realizaba entre las relaciones y las entidades. El *modelo relacional* (basado en la teoría relacional) tenía la desventaja de perder información semántica del mundo real, sin embargo, lograba una gran independencia de los datos, y por último el *modelo de conjuntos* de entidades (basado en la teoría de conjuntos), al igual que el modelo relacional, lograba obtener un alto grado de independencia de los datos, sin embargo, la visión de ciertos valores de esos conjuntos podía no ser natural para ciertos desarrolladores.

Este nuevo modelo propuesto por Chen (basado en la teoría de conjuntos y en la teoría de las relaciones) denominado *modelo Entidad Relación* quería alcanzar también un alto grado de independencia de los datos, así como una visión natural de los mismos. y quería sentar las bases para realizar una visión unificada de los datos. No se desprendía de ellos, ya que estaba basado en los mismos, sino que lo exponía como una extensión de los ya existentes.

Este modelo es un modelo conceptual y, a día de hoy, sigue siendo uno de los más utilizados a hora de diseñar una base de datos en muchas organizaciones.

La expresión gráfica del modelo definido por Chen se denomina esquema Entidad-Relación, el cual utiliza una serie de figuras para representar los componentes del modelo.

A lo largo del presente capítulo se estudiarán los componentes del modelo, así como su representación gráfica.

Las principales características de un esquema Entidad-Relación son:[4]

- **Completo**: El esquema ha de Representar todas las características del dominio de la aplicación.
- **Correcto**: Ha de utilizar correctamente aquellos conceptos definidos en el modelo Entidad-Relación.
- **Mínimo**: Cada uno de los aspectos de los requerimientos aparece una única vez en el esquema, no debe haber duplicados.

[3] Peter Pín-Shan Chen "The entity-relationship model: towards a unified view of data". March 1977
[4] Dra. M.ª Mercedes Martínez González, Carmen Hernández Díez "Apuntes de la asignatura bases de datos" ETSII Uva

- **Expresivo**: Ha de representar todos los requerimientos necesarios de una forma natural.

2.2 Componentes.

Como su propio nombre indica, los dos componentes principales del modelo Entidad-Relación son, las entidades y las relaciones existentes entre esas mismas entidades.

2.2.1 Entidades y conjuntos de entidades.

Las **entidades** (las cuales denotaremos *e*) son aquellos objetos del mundo real sobre los que deseamos almacenar información. Son algo que existe y se puede diferenciar de otras. Podemos decir que las entidades son objetos que forman parte de un universo. Este universo será dado por la organización para la que haremos la base de datos o la aplicación sobre la que deseamos crear una aplicación. Estos objetos pueden ser tanto reales (una persona específica, el actor de una película…) como abstractos (un determinado suceso, la crítica de una película…). Ejemplos de entidades son un actor llamado *Miguel*, una película que se titula *2001* o un cine situado en Valladolid con nombre *Roxy*. Una entidad no es una propiedad concreta, sino un objeto que tiene varias propiedades (a las que llamaremos atributos).

Todas aquellas entidades que tienen las mismas propiedades forman **conjuntos de entidades** (o tipos de entidad) (denotada por *Ei*) [Figura 2.1]. Ejemplos de conjuntos de entidades son el conjunto de todos los Actores, el conjunto de todas las Películas, o el conjunto de las Críticas de una película…

Una **instancia de entidad** es una única ocurrencia de un conjunto de entidad [Figura 2.1]. Por ejemplo, dado el conjunto de todas las *Películas*, una instancia concreta de este conjunto, sería la película con título *2001*.

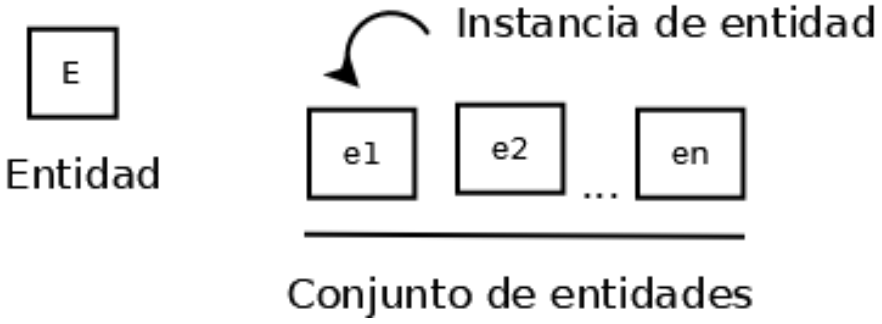

Figura 2.1 Conjuntos de entidades e instancias.

Existe una implicación asociada a este hecho, si sabemos que una determinada película forma parte del conjunto de entidades *Pelicula*, dicha película va a tener un conjunto de características iguales al resto de elementos que forman parte de ese conjunto de *Películas*.

En el diagrama Entidad-Relación las entidades están representadas por un rectángulo. [Figura 2.2]

Figura 2.2 Representación de entidad.

Existen varios tipos de entidades atendiendo a los atributos de las mismas o a las relaciones que estas poseen, pero antes de definirlos necesitamos conocer ciertos conceptos que veremos a lo largo de este capítulo. Identificar las entidades es el primer objetivo a la hora de realizar un esquema Entidad-Relación.

2.2.2 Relaciones, conjuntos de entidades y roles.

Una **relación** es la asociación entre dos o más entidades. También forman parte del universo al que nos hemos referido en la definición de entidad. De forma matemática y utilizando la teoría de conjuntos si tenemos dos conjuntos *A* y *B*, entonces *R* (una relación) es un subconjunto de *A* y *B*. Refiriéndonos a entidades y no conjuntos, *R e*s un subconjunto de las entidades *A* y *B*. Por ejemplo, la relación entre una determinada *película* y el *director* que la *dirige*, o esa misma película y la reseña que han escrito sobre ella. Entre estos ejemplos existe una relación, la película es *dirigida* por un director (o un director *dirige* una determinada *película*) y la reseña es *escrita* por un determinado crítico (o un crítico *escribe* una reseña). Debemos tener en cuenta que los conjuntos de entidades pueden, pero no tienen por qué ser, necesariamente disjuntos. Una entidad *Director* puede ser una entidad *Actor*, una entidad *Actor* puede pertenecer a la vez a *Director* o no serlo. Identificar estas relaciones es el segundo objetivo a la hora de la realizar el esquema Entidad-Relación.

En el diagrama Entidad-Relación las relaciones están representadas por un rombo. [Figura 2.3]

Figura 2.3 Representación de relación.

Si consideramos la asociación entre entidades, un **conjunto de relaciones** (detonada por *Ei*) es una relación matemática entre n entidades (siendo n mayor o igual a 2) (n>=2), cada una de ellas tomada de un conjunto de entidades.

$$\{[e1, e2, ..., en] \mid e1 \in E1, e2 \in E2, ..., en \in En\}$$

Y cada una de las tuplas de entidades **[***e*1, *e*2, …, *e*n], es una relación.

Por ejemplo:

$$(\text{Steven Spielberg, Tiburón}) \in \textit{Dirige}.$$

Dirige es una relación entre la *Película* Tiburón y el *Director* Steven Spielberg.

El **rol** de una entidad (denotado por *r*) en una relación es la función que dicha entidad realiza en esa relación. Los roles representan la función de un conjunto de entidades en un conjunto de relaciones. Mientras que en el diagrama Entidad-Relación los nombres tanto de las relaciones como de las entidades son necesarios, en el caso de los roles esto no es así. Siguiendo el ejemplo anterior, un director tiene el rol de dirigir una película o un actor el de actuar en ella.

En el diagrama Entidad-Relación son las líneas rectas que vinculan las entidades con las relaciones. [Figura 2.4]

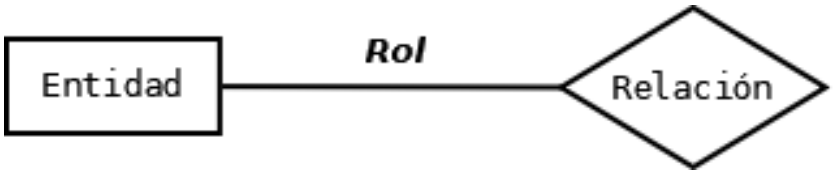

Figura 2.4 Rol de una entidad en una relación.

2.2.3 Atributos, valores y conjuntos de valores.

La información que poseen tanto las relaciones como los atributos se expresan por un **conjunto de atributos**. Estos conjuntos de atributos son comunes a las entidades que pertenecen a un conjunto de entidades o a las relaciones que pertenecen a un conjunto de relaciones. Cuando deseamos almacenar el conjunto de entidades que forman parte de un universo, todo este conjunto tendrá una serie de características (***atributos***), sin embargo, cada una de estas tendrá un **valor** que hará que se diferencien unas de otras instancias dentro de dicho conjunto. Formalmente un atributo se define como una función que esquematiza a un conjunto de entidades o un conjunto de relaciones en un conjunto de valores o un producto cartesiano de conjunto de valores de la siguiente forma:

$$f: E_i \longrightarrow V_i$$

o

$$f: R_i \longrightarrow V_i$$

El conjunto de valores permitidos para cada atributo se denomina **Dominio**. El dominio especifica el tipo de datos a almacenar o las restricciones que hay que aplicar a los valores.

Ejemplos de dominio son; booleano, entero, Double, String…

Por ejemplo, una entidad *Película* está formada por los siguientes atributos: un resumen, el año de producción, el título original, la duración y el idioma (entre otros muchos).

Una entidad *Director* tendrá como atributos el nombre del mismo, la fecha de nacimiento y la nacionalidad.

```
Película = (Resumen, Año de producción, Título original, Duración, Idioma).

Director = (Nombre, Fecha de nacimiento, Nacionalidad).
```

Los atributos en el diagrama asociado se representan como óvalos. Para ligarlos a la entidad a la que pertenecen se utiliza una línea recta [Figura 2.5]

.

Figura 2.5 Representación de atributo.

Debemos tener en cuenta que los nombres de los atributos no se deben repetir, estos han de ser únicos para cada tipo o conjunto de entidad. Si existen dos atributos similares es necesario utilizar **calificadores**. Si necesitamos almacenar para un conjunto de entidades cierta característica con un nombre similar al de otra, como por ejemplo para un conjunto de *Películas*, el nombre original de la película y el nombre en el país de emisión, no podríamos tener dos atributos con el mismo valor, por ejemplo título, por lo que, si necesitamos almacenar dos títulos, utilizamos calificadores, pudiendo separar ambos títulos en *titulo_original* y *título*, manteniendo así la diferencia entre ambos y haciendo que no se repitan.

2.2.3.1 Tipos de atributos.

Los atributos se pueden dividir en varios **tipos** atendiendo a diversas características como pueden ser el número de valores que pueden tomar, la división en partes más simples o su dependencia con otros de la misma entidad**.**

Pueden ser simples o compuestos. *Simple* significa que el valor del mismo es mínimo y ya no se pude dividir en partes con significado propio, como por ejemplo el título de una *Película*. Los *compuestos* son aquellos que, si se pueden dividir en partes [Figura 2.6], cada una de ellas tiene significado propio, como por ejemplo la *dirección* en la que está un cine. Dicha *dirección*, considerada como compuesto se puede dividir a su vez en otros atributos, esta vez simples: *calle*, *número* y *código postal*.

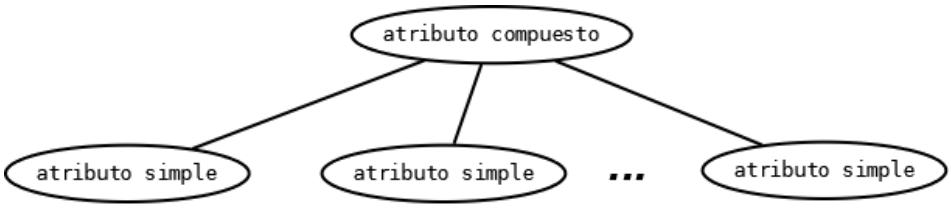

Figura 2.6 Representación de atributo compuesto.

Los atributos también pueden ser mono-valuados y multi-valuados. Un atributo *mono-valuado* es el que tiene un único valor para cada ocurrencia de la entidad o la relación a la que hace referencia mientras que uno *multi-valuado* es el que puede tener varios valores para cada ocurrencia de la entidad o relación.

Por ejemplo, el título de una película puede tener varios valores dependiendo del país en el que se emita. La instancia del conjunto de entidades *Peliculas* con nombre original "*Alien*", tiene en otras regiones el título *"El octavo pasajero*", dicho atributo sería multivaluado, teniendo esos dos valores como posibles. Los atributos multivaluados en el diagrama Entidad-Relación asociado se representan mediante un óvalo doble [Figura 2.7]

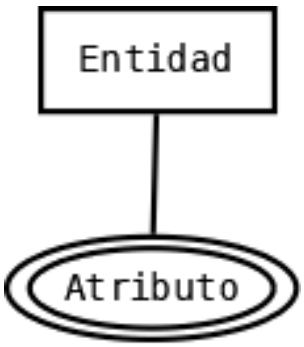

Figura 2.7 Representación de atributo multi-valuado.

Los atributos pueden ser atributos identificadores como veremos en el siguiente apartado. Si un atributo es *identificador* o *clave*, estos se subrayan en el diagrama entidad-relación [Figura 2.8].

Figura 2.8 Representación de atributo clave.

Los atributos pueden ser, debido a la necesidad o no de tener valores, opcionales u obligatorios. Esta distinción no existía en la propuesta original del diagrama Entidad-Relación propuesto por Chen, sin embargo, vamos a explicar este tipo debido a que nos va a resultar de utilidad ya que

en el capítulo 4 [Capítulo 4] a la hora de transformar un diagrama Entidad-Relación completo, veremos que dicha distinción si es posible a la hora de modelar datos con *MongoDB*.

Mientras que los atributos *opcionales* son aquellos que tienen que tener un valor si o si, los *opcionales* son aquellos que no tienen por qué tenerlo. La finalidad de los atributos obligatorios es asegurar que la información para esa característica particular se recoge. Por ejemplo, los identificadores de entidad siempre serán obligatorios, ya que son datos que se necesitan recoger si o si, mientras que por ejemplo la url de un cine puede ser opcional ya que ese determinado cine puede que tenga un sitio web o que no lo haya dado de alta.

Finalmente existen los atributos *derivados*, que son aquellos que se derivan a partir de otros atributos. La edad de un director, calculada a partir de la fecha de su nacimiento, es un atributo derivado.

Los atributos derivados se representan como un óvalo con línea discontinua en el diagrama Entidad-Relación [Figura 2.9].

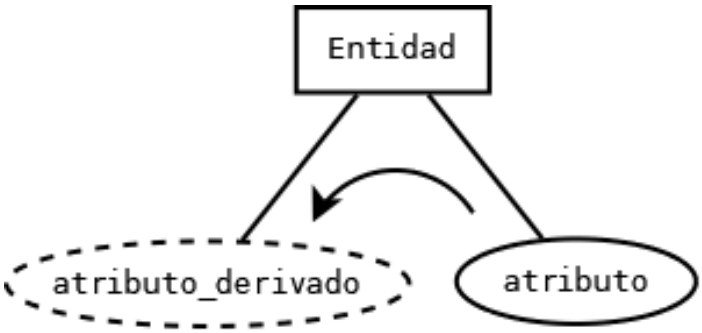

Figura 2.9 Representación de atributo derivado.

Como hemos comentado, un atributo puede ser una propiedad de un conjunto de relaciones y no solo de un conjunto de entidades [Figura 2.10], como por ejemplo el conjunto de relaciones dirigir entre los conjuntos de entidades director y película puede tener el atributo fecha de dirección.

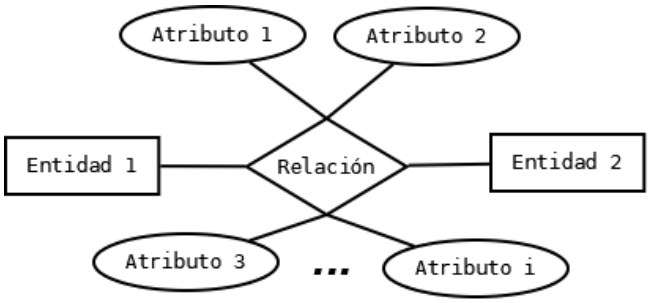

Figura 2.10 Atributos de una relación.

2.2.4 Identificador de entidad.

Las entidades pueden tener un **identificador de entidad**, aunque esto no siempre es necesario como veremos posteriormente. El identificador de entidad son uno o varios atributos cuyos valores permiten distinguir esa entidad del resto del conjunto de entidades, como por ejemplo el nombre completo de un director o el nombre de un cine en una ciudad. Estos atributos que son identificadores se muestran en el diagrama Entidad-Relación subrayados [Figura 2.11].

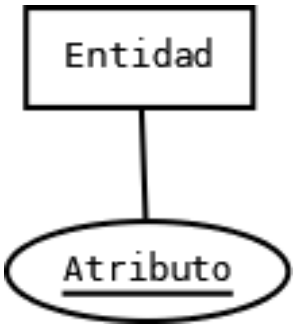

Figura 2.11 Identificador de entidad.

2.2.5 Grado de una relación.

El **grado** de un conjunto de relaciones (también expresado en la literatura referente con el término *aridad*) hace referencia al número de conjuntos de entidades que participan en el conjunto de relaciones. El conjunto de relaciones puede tener cualquier grado.

Las relaciones *binarias* (las más comunes dentro de una base de datos) son aquellas relaciones en las que intervienen dos entidades (como la relación *interpreta* entre actor y película).

Las *ternarias* son las que involucran a tres entidades [Figura 2.12]

Mientras que las *n-arias* son aquellas en las que intervienen *n* entidades.

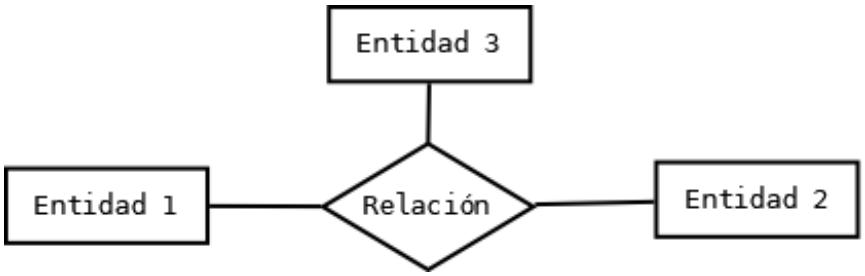

Figura 2.12 Relación ternaria.

2.2.6 Relaciones recursivas.

Las relaciones recursivas son aquellas cuyo mismo conjunto de entidades participa más de una vez en el mismo conjunto de relaciones, pero con diferentes roles.

Comentábamos anteriormente que el rol en el diagrama Entidad-Relación no suele necesario nombrarlo, sin embargo, en las relaciones recursivas pueden aportar información que de otra forma sería imposible comprender [Figura 2.13].

Siguiendo el mismo ejemplo podemos ver una relación recursiva en la que interviene el conjunto de entidades *Directores*, cuando por ejemplo un *director* se dirige a sí mismo en caso de actuar en una determinada *película*.

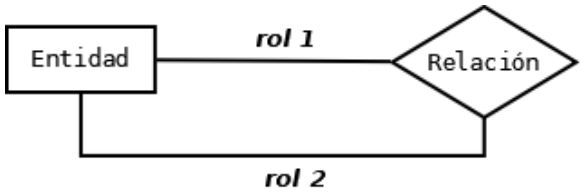

Figura 2.13 Relación recursiva.

2.3 Restricciones.

Existen dos tipos de restricciones que se pueden utilizar en nuestro modelo, estas son las restricciones de cardinalidad y las restricciones de participación.

2.3.1 Restricciones de cardinalidad.

La **cardinalidad** es una restricción estructural del modelo Entidad-Relación. Esta especifica el número de asociaciones en las que una entidad determinada puede estar. En otras palabras, la cardinalidad es el número de instancias de un determinado conjunto de entidades que pueden relacionarse con instancias de otro conjunto de entidades distinto.

Teniendo dos conjuntos de entidades *A* y *B*, e instancias de cada uno de los conjuntos *a* y *b*. La instancia *a* puede no estar relacionada con ninguna entidad *b*, puede estar relacionada con una o puede que lo esté con varias. En este último caso decimos que la cardinalidad es *N*. Por lo tanto, la cardinalidad puede tomar tres valores: *0*, *1* o *N* (siendo *N* siempre mayor que *1*).

2.3.1.1 Cardinalidad máxima y mínima.

A la hora de realizar un diagrama Entidad-Relación y calcular u obtener su cardinalidad, lo que importa son las cardinalidades máximas y mínimas.

Definimos **cardinalidad mínima** al número mínimo de asociaciones que una instancia de un conjunto de entidades presenta una relación con una instancia de un conjunto distinto de entidades.

Asimismo, la **cardinalidad máxima** es el número máximo de asociaciones que una instancia de un conjunto de entidades presenta una relación con una instancia de un conjunto distinto de entidades.[5]

En el diagrama Entidad-Relación, tanto la cardinalidad mínima como la máxima se expresan con su valor correspondiente (0,1 ó N) entre paréntesis sobre el rol y junto a la relación existente. El orden en el que aparecen es la cardinalidad mínima seguida de la máxima separadas por una coma [Figura 2.14].

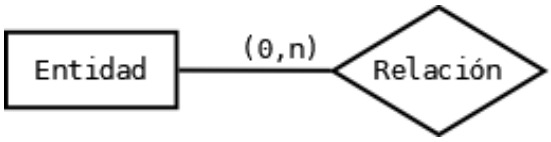

Figura 2.14 Cardinalidad mínima y máxima.

2.3.1.2 Tipo de relación.

El tipo de relación en el diagrama Entidad-Relación puede ser uno a uno, uno a muchos o muchos a muchos.

A la hora de calcular el tipo de relación, tomamos la máxima de cada una de las cardinalidades presentes en dicha relación. En la siguiente figura observamos que la cardinalidad izquierda es de (0, n) mientras que la cardinalidad derecha es de (0, 1). Para calcular el tipo tomamos la mayor de cada una (n y 1) y así obtenemos el tipo de relación. [Figura 2.15].

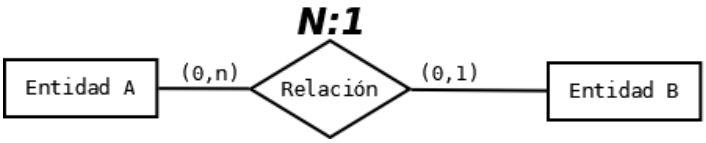

Figura 2.15 Cardinalidad mínima y máxima.

Para las relaciones binarias (aquellas en las que están involucradas dos entidades) la cardinalidad en el modelo original Entidad-Relación tiene la siguiente notación:

- **1: 1** Dada una relación *R* entre los conjuntos de entidades *A* y B, el tipo de relación *1:1*, a una instancia del conjunto de entidades *A* le corresponde ninguna o una única instancia del conjunto de relaciones B [Figura 2.16]. Análogamente, a una instancia del conjunto de entidades *B* le corresponde ninguna o una única instancia del conjunto de entidades *A*. La restricción se aplica tanto para el conjunto de entidades *A* como para el conjunto de entidades *B*.

[5] Eduardo López "¿Qué es cardinalidad de una relación?" [Recurso online] Disponible en: https://www.aulapc.es/lupa_busquedas_posit.html1accesA~A60.00 Fecha última consulta: 01/06/2024

Figura 2.16 Relación 1 a 1.

Ejemplo: La relación poseer entre los conjuntos de entidades *Perro* y *Chip*. Tenemos una instancia de perro, con nombre *Potola* y un chip con número de serie *25.* Potola solo tiene un chip, el *25*. Mientras que el chip con número de serie *25* solo puede ser poseído por *Potola*.

- **1: N** Dada una relación *R* entre los conjuntos de entidades *A* y *B*, el tipo de relación *1:N*, a una instancia del conjunto de relaciones A le corresponden ninguna, una o varias instancias del conjunto de entidades *B*, mientras que a cada instancia del conjunto de entidades *B* le corresponde ninguna o una única instancia del conjunto de entidades A [Figura 2.17]. En este caso la restricción se aplica sobre el conjunto de entidades *B*.

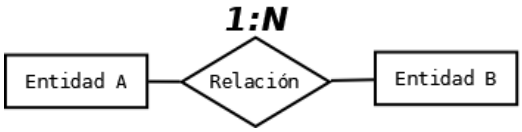

Figura 2.17 Relación 1 a muchos.

Ejemplo: Siguiendo con el universo planteado en el anterior ejemplo. Una instancia de perro, *Potola*, puede tener varios hijos (cuyas instancias son *Chester*, *Triscas* y *Cucas*) mientras que a los hijos le corresponden una única madre, *Potola*.

- **N: M** Dada una relación *R* entre los conjuntos de entidades *A* y *B*, el tipo de relación *N:M,* ninguno de los dos conjuntos de entidades tiene restricciones. Las instancias del conjunto de entidades *A* pueden estar relacionadas con cualquier número de instancias del conjunto de entidades *B*. Análogamente cualquier instancia de entidad del conjunto de relaciones *B* puede estar relacionada con cualquier número de instancias del conjunto de entidades *A* [Figura 2.18].

Figura 2.18 Relación muchos a muchos.

Ejemplo: En el mismo universo planteado, tenemos la relación *ser dueño* entre los conjuntos de entidades Perros y Personas. Tenemos dos instancias de perros, con nombres *Potola* y *Amels* y dos instancias de Personas con nombre *Henar y Cristofer*. Potola tiene como dueños a *Henar y Cristofer*, mientras que *Henar* y Cristofer son dueños de *Potola y Amels*.

2.3.2 Restricciones de participación.

Hablamos de restricciones de participación cuando la existencia de una entidad depende de la existencia de otra entidad distinta. Hace referencia a la obligatoriedad de participación, haciendo referencia de si todos o no participan en ella. Existen dos tipos de participación: *total* y *parcial*. Interesa el número mínimo de instancias de un conjunto de entidades relacionada con una instancia de otro conjunto de entidades distinto.

La **participación total** (también denominada obligatoria) indica que las instancias de un conjunto de entidades A están relacionadas con alguna instancia del conjunto de entidades B. En el diagrama Entidad-Relación el rol entre la entidad y la relación es una línea simple [Figura 2.19].

La **participación parcial** no implica ninguna restricción ya que las instancias de una entidad relacionada con las instancias de otra pueden o no estar relacionadas. En el diagrama Entidad-Relación el rol entre la entidad y la relación es una línea doble [Figura 2.19].

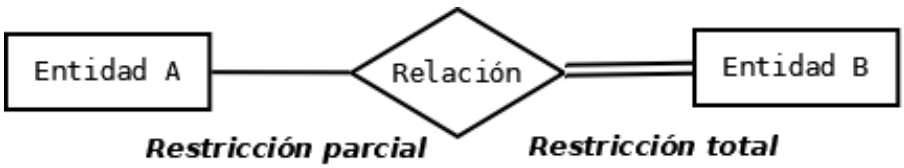

Figura 2.19 Restricción total y parcial.

2.3.3 Restricciones de cardinalidad (min, max).

Este tipo de restricción de cardinalidad no aparecía en el modelo original de Chen, pero desde que lo definió han sido varios los modelos y esquemas Entidad-Relación que utilizan esta notación.

La **restricción de cardinalidad min max** indica el número de instancias de una entidad que pueden pertenecer a la relación [Figura 2.20]. El número mínimo de instancias de la relación

participante es m mientras que el máximo es M. Si por ejemplo tuviésemos una participación (min, max) igual a (0,3) puede que no haya ninguna instancia de la entidad que participase en la relación mientras que el número máximo de instancias participantes es de 3.

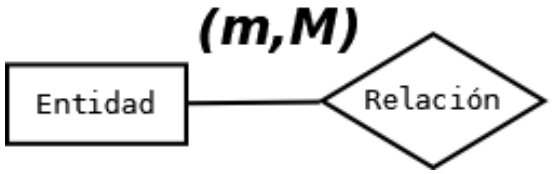

Figura 2.20 Relación min, max.

2.4 Entidades débiles. Relaciones identificantes.

Una vez definidos los conceptos de entidad y del identificador de la misma, podemos diferenciar las entidades en dos tipos: entidades fuertes y entidades débiles.

Diremos que *x* es una **entidad débil** si la existencia de *x* depende de la existencia de otra entidad *y*, entonces diremos que *x* depende por existencia de *y*.[6] En otras palabras, una entidad débil no tiene sentido si no existe la **entidad fuerte** de la que depende. Las entidades débiles carecen de identificador. Existen una serie de atributos discriminantes que diferencian las entidades débiles relacionadas a la misma entidad (o fuerte). En su conversión al modelo relacional, el identificador de entidad débil se forma mediante la unión del identificador de la entidad fuerte con la que está relacionada y los atributos discriminantes.

Por ejemplo, si tenemos una base de datos que almacena la información de los cines de una ciudad, podemos decir que el conjunto de entidades *Salas* son entidades débiles ya que sin la existencia del conjunto de entidades *Cines* estas no tendrían sentido.

En el diagrama Entidad-Relación las entidades débiles se representan con un rectángulo doble, a diferencia de las débiles que lo hacen con un rectángulo simple. [Figura 2.21].

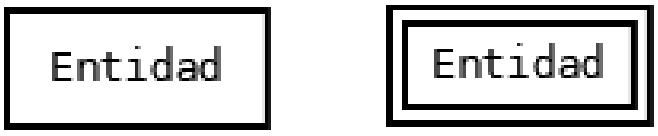

Figura 2.21 Entidades fuertes y débiles.

[6] Atlantic International University "Modelo Entidad-Relación" [Recurso Online] Disponible en: https://cursos.aiu.edu/Base%20de%20Datos/pdf/Tema%203.pdf Fecha última consulta: 01/06/2024

Las entidades débiles participan siempre de forma total con la relación asociada. Como hemos visto en el punto anterior, la línea que representa el rol en el diagrama Entidad-Relación asociado es doble, al igual que el rombo que representa la relación, que también es doble [Figura 2.22]. A este tipo de relaciones se las denomina **relaciones identificantes**. La entidad o entidades que estén asociadas a la relación identificante que no sean débiles pueden tener una participación total o parcial.

Figura 2.22 Relación identificante.

Las entidades débiles tienen un nuevo tipo de atributos, los cuales se representan en el diagrama Entidad-Relación asociado mediante el símbolo de los atributos por defecto, pero subrayados de forma discontinua [Figura 2.23]. Estos atributos se denominan **claves parciales o discriminadores**. Estos atributos son los encargados de diferenciar las distintas instancias de ese conjunto de entidades. Los identificadores de un conjunto de entidades débiles se obtienen concatenando el identificador del conjunto del que dependen con el discriminador del conjunto de entidades débiles.

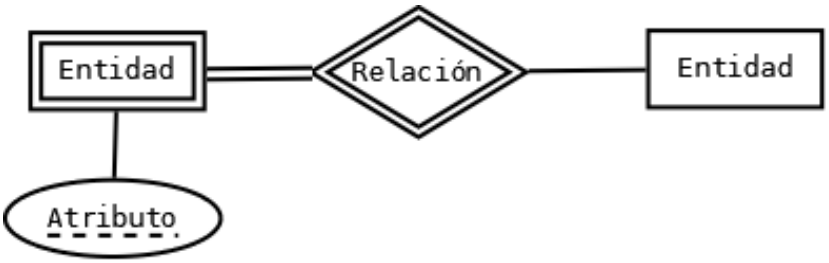

Figura 2.23 Discriminadores de entidades débiles.

2.5 Entidades asociativas.

Las **entidades asociativas** son un tipo de entidades las cuales asocian las instancias de uno o más conjuntos de entidades y pueden contener atributos los cuales pertenecen a la relación entre esas instancias de entidades. En otras palabras, permiten relacionar instancias de varios tipos de entidades. Pueden tener atributos que solo pertenecen a la relación entre esas instancias de entidades

Este tipo de entidades son capaces de relacionarse con otras entidades distintas, cualidad que no tienen aquellas entidades que no son asociativas, las cuales solo se pueden relacionar con otras entidades por medio de relaciones.

Las entidades asociativas se representan mediante un rombo contenido en un rectángulo [Figura 2.24].

Figura 2.24 Entidad asociativa.

2.6 Modelo Entidad-Relación extendido.

Los diagramas Entidad-Relación que definió Chen tienen una serie de restricciones, por lo que es posible que no cumplan de forma correcta su cometido debido a dichas limitaciones siendo complejo representar ciertas características del mundo real con las reglas del modelo inicial.

Es por eso que varios autores han extendido este modelo en repetidas ocasiones. La idea de este proyecto es enunciar y explicar las características básicas y algunas avanzadas para que sirvan de base para transformar diagramas Entidad-Relación en bases de datos con *MongoDB*. [7]

Dentro de las características extendidas del modelo Entidad-Relación extendido se incorporan una serie de conceptos relacionados con la orientación a objetos como pueden ser la generalización, especialización, herencia y agregación.

2.6.1 Especialización.

La **especialización** consiste en designar, dentro de un conjunto de entidades, subconjuntos de entidades pertenecientes a dicho conjunto. En este caso puede existir un determinado subconjunto de entidades que se diferencian de otro en uno o varios atributos los cuales no son compartidos. En el diagrama Entidad-Relación, se representa con un triángulo (o triángulo invertido dependiendo de la bibliografía) con la inscripción ISA (del inglés “is a “, “es un “) [Figura 2.25]

[7] E.F.Cood “Extending the database relational model to capture more meaning” 1979

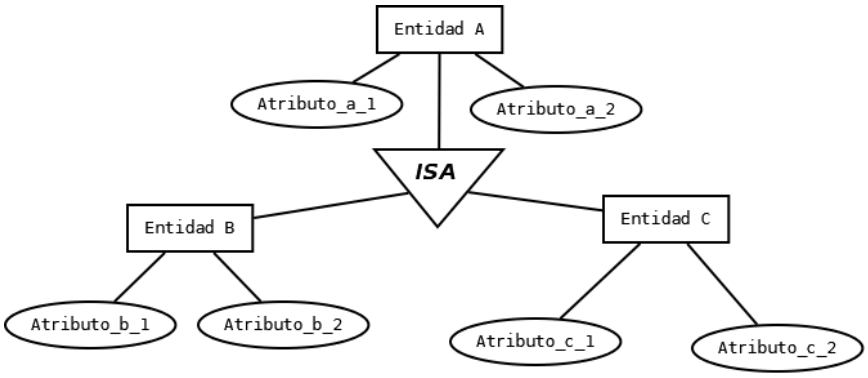

Figura 2.25 Relación ISA.

Ejemplo: Si en un diagrama Entidad-Relación hemos identificado previamente las entidades (dentro del universo de los trabajadores de una Universidad) *Profesores*, *Investigadores*, *Pas* y *Trabajadores* podemos observar fácilmente la relación existente entre dichas entidades. De la entidad *Trabajadores* podemos obtener los siguientes subconjuntos: *Profesores*, *Investigadores* y *Pas*. Todos ellos pertenecen al conjunto de entidades de mayor orden (en este caso *Trabajadores*) y comparten atributos pertenecientes a la misma (como podrían ser nombre, sueldo o centro de trabajo), sin embargo, los subconjuntos *Profesores*, *Investigadores* y *Pas*, aun perteneciendo a *Trabajadores*, poseen atributos que les diferencian unos de otros, como por ejemplo el conjunto de entidades *Profesores* puede tener asociado un atributo con nombre departamento, el cual no posee el conjunto de entidades *Pas*.

2.6.2 Generalización.

La **generalización** consiste en designar una serie de subconjunto de entidades en un conjunto de entidades de mayor orden. Podemos decir que es el proceso contrario al de especialización.

Para el ejemplo que acabamos de ofrecer relativo a los trabajadores de la Universidad, mientras que la especialización sería obtener del conjunto de entidades *Trabajador* los subconjuntos de entidades *Profesores*, *Investigadores* y *Pas*, la generalización sería el proceso opuesto: una vez obtenidos los tres subconjuntos (*Profesores*, *investigadores* y *Pas*) obtener uno de mayor orden que estos, que en el caso descrito sería *Trabajadores*.

En el diagrama Entidad-Relación, estos dos conceptos son indistinguibles, ambos (generalización y especialización) son operaciones inversas. [Figura 2.26]

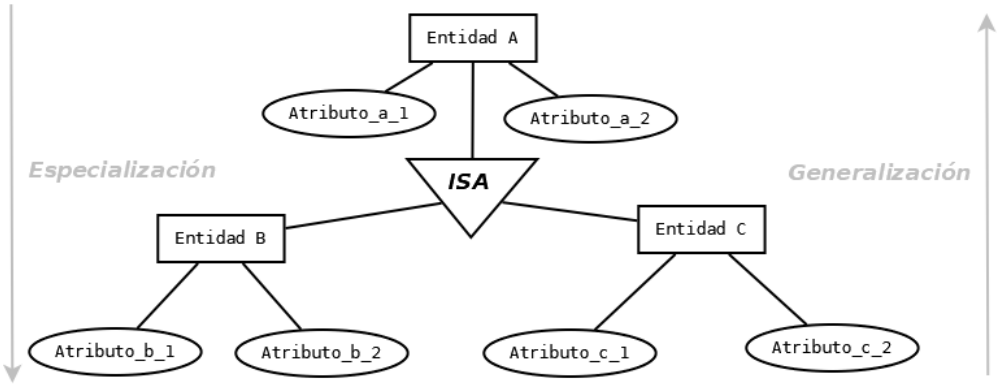

Figura 2.26 Generalización y especialización.

Existen dos tipos de generalización: total y parcial.

En la **especialización total** toda instancia del conjunto de entidades de orden superior ha de estar representada en las de nivel inferior. En el diagrama Entidad-Relación se representa dicha participación con una línea doble que une el conjunto de entidades de mayor orden con el triángulo que representa la especialización ISA [Figura 2.27].

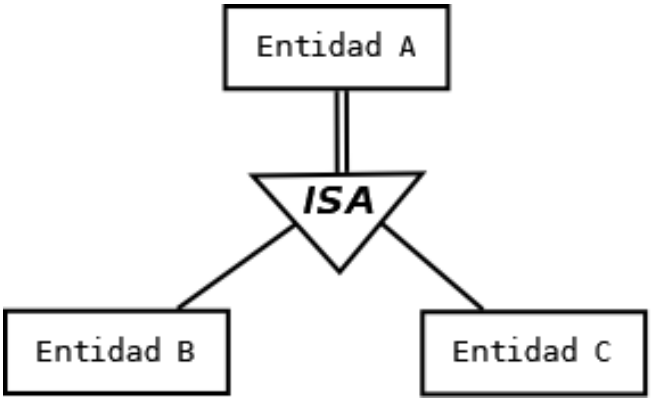

Figura 2.27 Especialización total.

Mientras que en la **especialización parcial** pueden existir instancias del conjunto de entidades del nivel superior las cuales no estén representadas en el conjunto de entidades de nivel inferior. La representación en el diagrama ER se realiza, al contrario que en la especialización total, con una línea simple que une el conjunto de entidades de mayor orden con el triángulo que representa la especialización ISA [Figura 2.28].

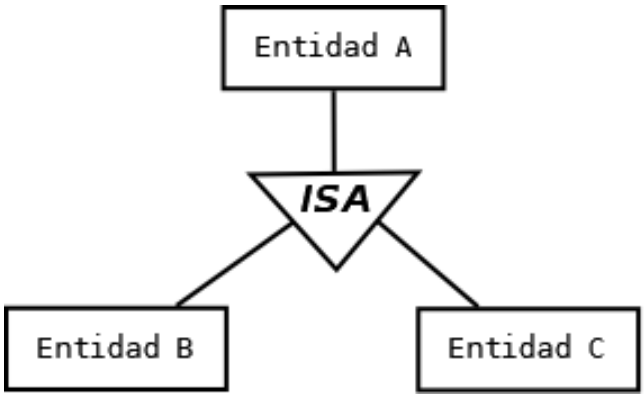

Figura 2.28 Especialización parcial.

2.6.3 Herencia.

Otro concepto importante en el diagrama Entidad-Relación extendido es el de la **herencia**. Definida la especialización y la generalización, los conjuntos de entidades de nivel más bajo heredan los atributos de los conjuntos de entidades de nivel más alto, así como la participación en las relaciones del conjunto de entidades de más alto nivel con el que están relacionadas.

2.6.4 Agregación.

Hay ocasiones en las que existe una relación entre conjuntos de entidades (supongamos *A* y *B*) y se desea relacionar ese conjunto (ambas entidades junto con su relación asociada) con otra entidad diferente *C*.

En el diagrama Entidad-Relación, este tipo de relaciones (denominada **agregación**) se representa introduciendo el conjunto de entidades junto con la relación existente entre ambas en un rectángulo, de tal forma que ese conjunto esté relacionado a su vez con una tercera entidad. [Figura 2.29].

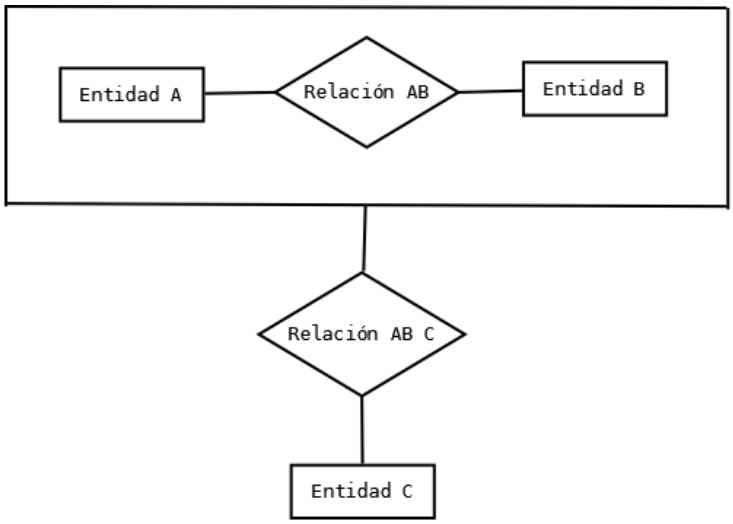

Figura 2.29 Agregación.

Capítulo 3 MongoDB.

Una vez estudiadas las características del modelo Entidad-Relación y su esquema asociado procedemos a ver cuáles son las particularidades de *MongoDB*

En este capítulo veremos las principales características y el surgimiento de las bases de datos no relacionales, para entender el contexto en el que nos encontramos.

Posteriormente nos centraremos en este sistema de bases de datos no relacional, *MongoDB*. Se explicará su funcionamiento, la forma en la que se estructuran los datos y los tipos de los mismos, así como las operaciones elementales de creación, inserción, borrado y modificación de datos necesarios para polar una base de datos.

MongoDB tiene una forma particular de relacionar los datos, pero se estudiará en el siguiente capítulo, cuando veamos las reglas de transformación del esquema Entidad-Relación a esta base de datos.

3.1 NoSQL.

Aunque el término se acuñó en el año 1998, no fue hasta finales de la década del 2000 cuando se instauraron los modelos NoSQL. En ese marco de tiempo fueron las bases SQL las que dominaron el mercado.

Las bases de datos NoSQL nacieron como respuesta a la era de internet, la *Big Data* y la necesidad de procesar datos no estructurados.

Dichos sistemas aparecieron debido a una serie de necesidades que los anteriores modelos no eran capaces de satisfacer. Entre ellas destacaran:

- La necesidad de una mayor escalabilidad que las bases de datos relacionales, incluyendo una cantidad muy grande de datos o un rendimiento en operaciones de escritura muy alto.
- La preferencia por el software gratuito y de código abierto por encima de productos comerciales.
- Operaciones "query" especializadas que no estaban respaldadas completamente por el modelo relacional.
- Los problemas causados por la las restricciones de los modelos relacionales, y un deseo de modelos de datos más dinámicos y expresivos.

Entre las características más importantes que introdujeron este nuevo modelo de bases de datos (y de las que carece el modelo relacional) estas serían las más importantes:

- Ausencia de soporte para el lenguaje Sql. La mayor parte de las bases de datos no relacionales definen su propio lenguaje, como por ejemplo *MongoDb*, sistema en el que profundizaremos en este proyecto, que carece de los "joins" y de las transacciones clásicas de las bases de datos relacionales, o Cassandra que utiliza su propio *CQL* (Cassandra Query Language) un derivado reducido de SQL.
- Ausencia de relaciones como se conocían previamente en el modelo relacional entre los datos.
- Ausencia de transacciones ACID (Atomicity, Consistency, Isolation, Durability).
- Modelos de datos muchísimo más flexibles.

Aun así, a día de hoy Oracle MySQL, Microsoft SQL server y PostgreSQL (todas ellas basadas en el modelo relacional) son las bases de datos más utilizadas en todo el mundo.

Sin embargo, las bases de datos NoSQL están ganando terreno y *MongoDB* (base de datos no relacional) ya es la quinta base de datos más usada del mundo[8] **[**Tabla 3.1].

[8] "DB-Engine Ranking". [Recurso online] Disponible en: https://db-engines.com/en/ranking Fecha última consulta: 01/06/2024

Ranking/mes			Nombre de base de datos	Modelo de base de datos
Nov 2023	Oct 2023	Nov 2022		
1.	1.	1.	Oracle	Relacional, Multi-modelo
2.	2.	2.	MySQL	Relacional, Multi-modelo
3.	3.	3.	Microsoft SQL Server	Relacional, Multi-modelo
4.	4.	4.	PostgreSQL	Relacional, Multi-modelo
5.	5.	5.	MongoDB	Documentos, Multi-modelo
6.	6.	6.	Redis	Clave-Valor, Multi-modelo
7.	7.	7.	Elasticsearch	Motor de búsqueda, Multi-modelo
8.	8.	8.	IBM Db2	Relacional, Multi-modelo
9.	9.	10.	SQLite	Relacional
10.	10.	9.	Microsoft Access	Relacional

Tabla 3.1 Bases de datos más utilizadas en la actualidad.

Entender las diferencias entre los modelos tradicionales y los NoSQL no es uno de los objetivos de este proyecto, ya que vamos a utilizar características o funcionalidades de ambos modelos para la creación de una base de datos.

Estos dos modelos (SQL y NoSQL) no tienen por qué ser completamente antagónicos, es más, cabe recordar el significado de las siglas NoSQL (*Not Only SQL*) por lo que ciertas características del modelo relacional (como puede ser un esquema previo a la creación de la base de datos) junto con la flexibilidad que ofrece este nuevo enfoque, pueden ser características que se pueden conjuntar en el proceso completo de la creación de una base de datos (como veremos en el presente estudio).

En la siguiente tabla [Tabla 3.2] podemos observar, de forma muy resumida, las principales diferencias entre los modelos SQL y NoSQL.

	SQL	NoSQL
Tipo de modelo	Modelo relacional. Almacena los datos en tablas.	Modelo no relacional. Almacena los datos en documentos tipo Json.

Datos	Ideal si cada uno de los registros almacenados tiene las mismas propiedades. El hecho de añadir una nueva propiedad puede ocasionar una modificación del esquema. Útil para datos estructurados. Las relaciones son normalmente plasmadas por datos no normalizados y mostrándolos en un único registro	Ofrece mucha flexibilidad y los registros no tienen por qué almacenar las mismas propiedades. Se pueden ir añadiendo propiedades a medida que estas sean necesarias. Útil para datos semi estructurados. Las relaciones son a menudo plasmadas utilizando joins para determinar referencias.
Escala	Escala de forma correcta verticalmente.	Escala de forma correcta horizontalmente.
Consistencia	Alto nivel de consistencia.	La consistencia varía dependiendo de la implementación adoptada.
Naturaleza	Naturaleza estructurada.	Naturaleza semi-estructurada o no estructurada.
Esquema	Rígido	Flexible
Soporte	Mucho soporte	Depende de la comunidad, aunque cada vez el mismo es mayor.
OLTP (Procesamiento de Transacciones en línea)	Es el recomendado para este tipo de transacciones.	Todavía no
Lenguaje	Lenguaje de consultas estructuradas.	Lenguaje de consultas no estructurado.
Ventajas	Esquema definido de forma precisa, asegura la integridad de los datos. Las relaciones permiten almacenar cada uno de los datos una sola vez. Ausencia de duplicados.	Ausencia de esquema, aunque proporciona una mayor flexibilidad. Los datos se almacenan según las necesidades, hace que la búsqueda de datos sea mucho más rápida.
Desventajas	Modelo menos flexible. Tiene la necesidad de ser planificado de antemano. Algunas relaciones pueden conducir a consultas muy complejas (haciéndolas más lentas).	El aumento de flexibilidad puede acarrear un desarrollo desordenado y posponer decisiones estructurales. El hecho de tener datos duplicados puede implicar el tener que actualizar varias colecciones.

Tabla 3.2 Diferencias SQL y NoSQL.

3.2 MongoDB.

MongoDB es un sistema de base de datos NoSQL orientada a documentos, de código abierto y escrito en C++ que nació en el año 2007 y fue creada por Dwight Merriman, Eliot Horowitz y Kevin Ryan.

La principal característica de *MongoDB*, en contraposición con las bases de datos relacionales, es que no necesita de un esquema para describir cada uno de los elementos funcionales, esto es, son bases de datos *schemaless*.[9]

Es una plataforma de base de datos flexible y escalable diseñada como comentamos anteriormente, para modificar o ampliar el enfoque de las bases de datos relacionales y las limitaciones ofrecidas por otras bases de datos también NoSQL. Posee un alto grado de escalamiento horizontal y de equilibrio de la carga .[10]

Esto significa que los desarrolladores pueden centrarse en los datos que necesitan almacenar y procesar, en lugar de invertir tiempo en cómo introducir y repartir los datos en tablas *rígidas*. En el siguiente apartado veremos la estructura de las colecciones y sus documentos asociados.

En relación con las consultas, *MongoDB* permite realizarlas *ad hoc,* permitiendo buscar datos o información por campo, consultas de rango y también búsquedas con expresiones regulares.

Es posible indexar cualquier documento. Esta es una de las características más importantes de esta base de datos. No entraremos en sus detalles en este trabajo ya que a la hora de transformar los diagramas ER intentaremos que la base de datos resultante represente de la forma más fiel el diagrama objeto de conversión

Permite también *replicación* (concepto el cual permite replicar y actualizar los datos ya sean estos persistentes o no), así como la replicación y una serie de características adicionales que no estaban disponibles en las anteriores bases de datos.[11]

3.3 Colecciones y documentos.

Mientras que en el modelo relacional la información se almacena en filas y columnas, las cuales están organizadas en tablas, *MongoDB* almacena la información en documentos, los cuales se reúnen en colecciones. Una base de datos en *MongoDB* es un *contenedor de colecciones de documento*s.

[9] "MongoDB Schemaless" [Recurso Online] Disponible en: https://www.mongodb.com/unstructured-data/schemaless Fecha última consulta: 01/06/2024
[10] "MongoDB features" [Recurso Online] Disponible en: https://www.mongodb.com/features Fecha última consulta: 01/06/2024
[11] "Replicación (informática)" [Recurso Online] Disponible en: https://es.wikipedia.org/wiki/Replicaci%C3%B3n_(inform%C3%A1tica) Fecha última consulta: 01/06/2024

Las bases de datos *MongoDB* no poseen esquema, al contrario que las bases de datos relacionales, esto quiere decir que el formato o esquema de un documento dentro de una colección no tiene por qué ser igual al de otro documento que pertenezca a esa misma colección.

Una **base de datos** en *MongoDB* puede almacenar una o varias colecciones.

Las **colecciones** son como las tablas de las bases de datos relacionales, ya que almacenan los datos, solo que en forma de documentos.

Los **documentos** son registros individuales de una colección, estos son la unidad básica de los datos en *MongoDB*.

En la siguiente figura podemos ver cómo es la estructura de datos en *MongoDB*. [Figura 3.1]

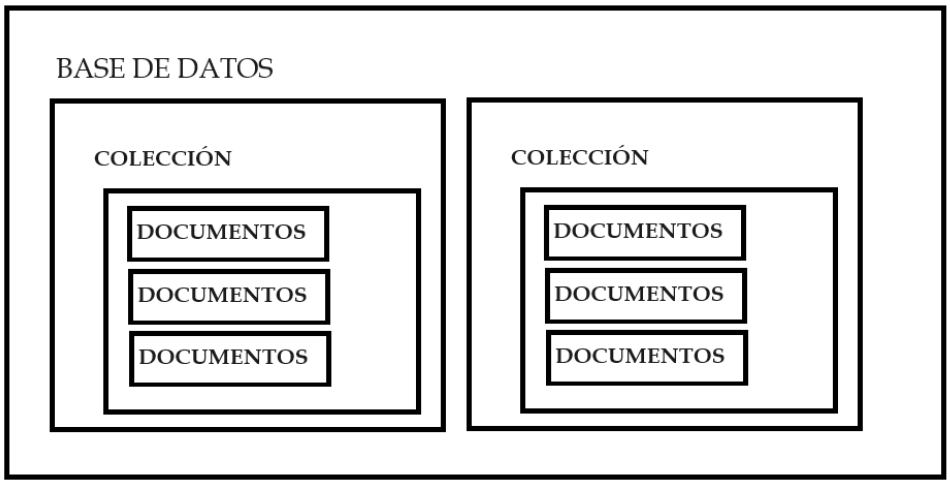

Figura 3.1 Base de datos en MongoDB

BSON es la abreviatura de Binary JSON. En otras palabras, BSON es la codificación en binario de documentos tipo Json (formato de texto para el acceso, almacenamiento e intercambio de datos que forma parte del sistema de JavaScript)

Cada uno de estos documentos (que sería el equivalente a las filas en las bases de datos relacionales) contiene un número indeterminado de "Campos" (atributos en el modelo tradicional), y cada uno de estos campos es un par clave valor, que sustituye a las columnas [Tabla 3.3].

```
// Colección A
// Documento a
{
   _id: <id_a>,
   clave_a_1: valor_a_1,
```

```
    clave_a_2: valor_a_2,
    ...
    clave_a_i: valor_a_i
}
```

Tabla 3.3 Ejemplo de documento.

Hemos de tener en cuenta que los documentos de *MongoDB* tienen un tamaño máximo de 16 Mb. Si a la hora de crear nuestra base de datos, nos encontramos con documentos mayores podemos utilizar la herramienta GridFS[12] la cual divide los archivos que superen ese número de bytes en trozos de 255 KB a excepción de la última parte.

En *MongoDB* el modelado de los datos y la estructura de los documentos deben responder a las necesidades del usuario y la aplicación en la que se desee utilizar este tipo de base de datos.

Por ejemplo, si nuestra aplicación va a recibir muchas consultas sería necesario el uso de índices en el modelo de datos para optimizar el tiempo de ejecución de las mismas, sin embargo, si nuestra aplicación va a tener un gran número de inserciones, borrados o modificaciones, sería necesario utilizar los índices y el sistema de fragmentado de datos para optimizar la ejecución de la misma.

En este proyecto no se va a implementar ninguna aplicación, ya que su finalidad es la de obtener una serie de reglas de transformación del modelo Entidad-Relación (y por ende del modelo relacional) en una base de datos en *MongoDB*

3.4 Tipos de datos en MongoDB.

Acabamos de ver la forma en la que se almacenan los datos en *MongoDB*, ahora bien, ¿qué tipo de datos permite almacenar los documentos Bson?

En estos documentos tenemos un formato muy simple. Para cada uno de los campos tenemos un par clave-valor separados por el símbolo ":". La clave es siempre un String o cadena, la cual identifica cada uno de las pares, mientras que el valor puede pertenecer a varios tipos de datos.

Algunos de estos tipos de datos son los siguientes: [13]

TIPO DE DATO	
Double	Tipo numérico de real de 48 bytes.
String	Cadena de caracteres UTF-8. Aquellos valores que entrecomillemos son por defecto de este tipo.
Object	Tipo object.

[12] "GridFS" [Recurso online] Disponible en: https://www.mongodb.com/docs/manual/core/gridfs/ Fecha última consulta: 01/06/2024

[13] "Bson specs" [Recurso Online] Disponible en: https://bsonspec.org/#/specification Fecha última consulta: 01/06/2024

Binarydata	Array de documentos binarios.
ObjectId	Consta de 12 bytes. Se utiliza normalmente como identificador de documentos para diferenciar de forma única unos de otros.
Booleano	True o false
Date	Consta de 64 bits. Representa el número de milisegundos trascurridos desde el 1 de enero de 1970
Null	1 par
Expresiones regulares	Conjunto de caracteres que especifican un patrón.
Javascript	Documentos de java script.
Integer de 32 bits	Tipo entero de 4 bytes.
Integer de 64 bits	Tipo entero de 8 bytes. Long.
Timestamp	Consta de 64 bits. Los 32 primeros son los milisegundos desde la creación del epoch de Unix y los siguientes diferencian operaciones dentro de un mismo segundo.
Decimal128	Tipo numérico de 16 bytes.
Min Key	Tipo interno de datos en *MongoDB* que representa los extremos teóricos de un intervalo. MinKey es menor que cualquier valor de un intervalo.
Max Key	Tipo interno de datos en *MongoDB* que representa los extremos teóricos de un intervalo. MaxKey es mayor que cualquier valor de un intervalo

Tabla 3.4 Tipos de datos en MongoDB.

3.4.1 ObjectID.

Ampliamos el concepto del tipo de datos **ObjectId**, ya que nos va a ser de gran utilidad a la hora de transformar un esquema Entidad-Relación en una base de datos con *MongoDB*:

_id es un campo requerido en cada uno de los documentos *MongoDB*.[14] El campo _id es del tipo ObjectId. Este tipo de objetos tienen la finalidad de preservar la unicidad en entornos como *MongoDB*. Dicho campo está compuesto por 12 bytes: los 4 primeros bytes son una marca temporal o timestamp que representa la creación del mismo medido en segundos desde el epoch de Unix, los 5 siguientes son un valor generado de forma aleatoria una vez por proceso, el cual es único a la máquina y proceso, los últimos 3 bytes son un contador incremental, el cual está inicializado a un valor aleatorio**.** Este campo garantiza un identificador único por segundo, máquina y proceso, concepto que casa perfectamente con la idea de atributo identificador en el diagrama ER.

[14] "MongoDB ObjectID" [Recurso Online] Disponible en: https://www.mongodb.com/docs/manual/reference/method/ObjectId/ Fecha última consulta: 01/06/2024

3.5 Operaciones MongoDB.

A la hora de convertir nuestro diagrama Entidad-Relación en una base de datos de *MongoDB* va a ser necesario utilizar una serie de operaciones para lograr crear nuestra base de datos. A continuación, enunciamos y ejemplificamos cada una de ellas.

3.5.1 Creación/cambio base de datos.

Mostrar/cambiar base de datos. Para mostrar la base de datos que estamos utilizando utilizamos el comando db.

```
db
```

Esta operación nos devuelve, en caso de no haber seleccionado ninguna base de datos creada anteriormente, *test*, que es la base de datos predeterminada de *MongoDB*.

Crear una base de datos. Para cambiar entre las bases de datos previamente creadas, utilizamos el comando *use* seguido del nombre de la base de datos que deseamos utilizar. Si la base de datos pasada como parámetro al comando use no existe, *MongoDB* la creará con dicho nombre.

```
Use TFG
```

3.5.2 Creación de colecciones.

Para crear una nueva colección se utiliza el método *createCollection( ),* a continuación, como argumento, se le pasa el nombre que deseemos que tenga nuestra colección. Veamos un ejemplo para crear una colección llamada "*Criticas*":

```
db.createCollection("Criticas")
```

3.5.2.1 Json Schema.

De forma opcional *createCollection( )* recibe un parámetro de configuración donde podemos añadir una serie de restricciones y reglas a nuestra colección. Se conoce como *JSON Schema* [3.5]

Un Json Schema es un tipo de medio Json cuya finalidad es la de definir la estructura de los datos Json, está destinado a definir tanto la validación, documentación y navegación por hipervínculos como el control de la interacción de los datos Json.

Existen varios esquemas para documentos Json, sin embargo, en la documentación oficial de *MongoDB* se nos emplaza a utilizar este si queremos crear las colecciones con una serie de restricciones base para posteriormente, a la hora de introducir documentos, estos tengan que tener una serie de características propuestas en las restricciones de los esquemas.

Las únicas restricciones que tiene este esquema son la imposibilidad de especificar dicha validación para las conexiones de la base de datos admin, local y config ni para las colecciones previamente creadas en el sistema. A continuación, podemos ver un ejemplo de JSON Schema:

```
"CriticasJson" {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Criticas validación",
         required: [ "resumen_critica", "numero_critica", "nombre_critico",
"estrellas","fecha_critica"],
         properties: {
            resumen_critica: {
               bsonType: "string",
               description: "'resumen_critica' ha de ser un string y es
necesario"
            },
            numero_critica: {
               bsonType: "int",
               minimum: 1,
               maximum: 100,
               description: "'numero_critica' ha de ser un integer en el rango [
1, 100] y es necesario"
            },
            nombre_critico: {
               bsonType: "string",
               description: "'nombre_critico' ha de ser un string y es necesario"
            },
            estrellas: {
               bsonType: "int",
               minimum: 1,
               maximum: 5,
               description: "'estrellas' ha de ser un entero en el rango [1,5] y
es necesario"
            },
           fecha_critica: {
```

```
                bsonType: "string",
                description: "'fecha_critica' ha de ser un string y es necesario"
            }
        }
    }
  }
}
```

3.5.3 Creación/inserción de documentos.

La **creación** o **inserción** de documentos consiste en añadir uno o varios documentos a una colección. *MongoDB* creará la colección en caso de no existir al realizar dichas operaciones. *MongoDB* proporciona los siguientes métodos para insertar documentos en una colección:

- *insertOne( )* método que **inserta un documento** a nuestra colección, como argumento se le pasa la estructura que deseemos almacenar. Por ejemplo, para añadir un único documento a la colección *Realizadores*:

```
db.Realizadores.insertOne(
 {
"nombre": "Quentin Tarantino",
"nacionalidad": "Estados Unidos",
 "fecha_nac": "02-03-1963"
}
)
```

- *insertMany( )* método que **inserta múltiples documentos** en una colección, como argumento se le pasa un array con las estructuras que deseemos almacenar. Para añadir varios (en este caso dos) documentos a la colección *Realizadores*:

```
db.Realizadores.insertMany(
[{
 "nombre": "David Fincher",
 "nacionalidad": "Estados Unidos",
 "fecha_nac": "28-08-1962"
},{
  "nombre": "Ridley Scott",
 "nacionalidad": "Reino Unido",
  "fecha_nac": "30-10-1937"}]
 )
```

3.5.4 Actualización de documentos.

Las operaciones de **actualización** modifican documentos existentes en una colección. *MongoDB* proporciona los siguientes métodos para actualizar nuestros documentos:

- *updateOne( )* **Actualiza un documento** de nuestra colección. Como argumento se le proporciona la información que hace de filtro de búsqueda y los elementos a actualizar con el operador *$set.* Mostramos como se actualizaría un documento de nuestra colección Realizadores añadiendo el campo *num_oscars*: 2

```
db.Realizadores.updateOne(
{"nombre": "Quentin Tarantino","nacionalidad": "Estados Unidos","fecha_nac":
"02-03-1963"},
 {$set: {"numero_oscars": 2}},
)
```

- *updateMany( )* Funciona de la misma manera que *updateOne( ),* solo que **actualiza más de un documento** de una determinada colección. Mientras que *updateOne( )* simplemente actualiza un único documento, este método actualizará todas las coincidencias. En el siguiente ejemplo actualizaremos todos los documentos de la colección realizadores con un campo nuevo llamado *num_oscars* inicializado a null.

```
db.Realizadores.updateMany(
 {},
 {$set: {"numero_oscars": null}},
)
```

- *replaceOne()* Nos permite **reemplaza un documento** entero de una colección excepto el campo *_id.* Recibe dos argumentos, el primero de ellos el documento a actualizar, el segundo el documento actualizado que se desea almacenar. Este nuevo documento solo tiene que tener pares clave valor, no se utiliza ningún operador de actualización.

  El documento actualizado puede tener campos distintos al del que se desea actualizar.
  A continuación, vemos un ejemplo en el que se actualiza un documento de la colección Realizadores.

```
db.Realizadores.replaceOne(
{"nombre": "Quentin Tarantino"},
{
 "nombre": "Quentin Tarantino",
"nacionalidad": "Estados Unidos",
```

```
"fecha_nac": "02-03-1963"
})
```

3.5.5 Borrado de documentos.

Las operaciones de **borrado** eliminan documentos de una colección. *MongoDB* nos proporciona dos métodos para eliminar documentos de nuestras colecciones. Tienen como objetivo una única colección, y nos permite utilizar filtros. Al igual que otros métodos que hemos visto anteriormente *MongoDB* nos permite borrar un único documento, varios o todos los documentos de una colección.

- *deleteOne( )* **Elimina un único documento** de nuestra colección. Como argumento se le pasa el filtro de búsqueda. A continuación, mostramos la forma de eliminar un determinado realizador de nuestra colección Realizadores. En vez de aplicar el filtro nombre como en anteriores ejemplos utilizaremos como argumento el *ObjectId* del mismo, algo que nos será de gran utilidad cuando más adelante veamos la forma de relacionar nuestras colecciones mediante referencias.

```
db.Realizadores.deleteOne(
 {_id: ObjectId("6557a6452b113681d0ed0d89")}
)
```

- *deleteMany( )* Funciona de la misma manera que *deleteOne( )* pero **elimina varios documentos**. En el siguiente ejemplo eliminamos todos los documentos que coincidan con el filtro pasado como argumento, todos cuya *fecha_nac* sea *02-03-1986*:

```
db.Realizadores.deleteMany(
 {"fecha_nac": "02-03-1963"}
)
```

3.5.6 Operaciones de consulta.

Este tipo de operaciones son necesarias para **buscar** documentos de nuestras colecciones.

En las bases de datos relacionales se tiene un lenguaje propio para realizar las consultas, SQL (*Structured Query Language*). En *MongoDB* no tenemos esta opción, ya que las búsquedas se

realizan mediante las funciones *find( )* y *findOne( ).* Estas devuelven un subconjunto de elementos de una determinada colección.

- *find( )* tiene dos argumentos, el primero de ellos es un documento que especifica los criterios de **consulta**, el segundo especifica el campo a devolver de entre los documentos que coincidan.

  Si el primer argumento de consulta es vacío (es decir, el argumento es {}), nos devolverá todos los elementos de la colección seleccionada. En el siguiente ejemplo:

```
db.Realizadores.find()
```

  La consulta nos devolverá todos los documentos de la colección *Realizadores*.

  Sin embargo, si utilizados la siguiente clave/valor en la misma consulta:

```
db.Realizadores.find({"nombre": "Ridley Scott"})
```

  La consulta nos devolverá todos los realizadores que cumplan con el filtro: nombre: *Ridley Scott*.

- *findOne()* Funciona de forma idéntica a *find()* salvo en que devuelve un único documento que satisfaga los criterios de búsqueda utilizados. En caso de que sean más de uno, devuelve el primero de ellos según el orden natural que en el que estén localizados los documentos en el disco. Si ningún documento coincide con los criterios introducidos devuelve null.

```
db.Realizadores.findOne({"nacionalidad": "Estados Unidos"})
```

  La consulta nos devolverá un único documento que cumpla el filtro: nacionalidad: *Estados Unidos.*

Capítulo 4 Reglas de transformación de un esquema Entidad-Relación a una base de datos con MongoDB.

Tras el estudio de las características que definen el modelo Entidad-Relación y habernos familiarizado con los conceptos y funcionalidad de la base de datos *MongoDB*, en este capítulo definiremos una serie de reglas para convertir un diagrama Entidad-Relación en esta base de datos.

Lo primero que debemos aclarar es que, como hemos visto en capítulos anteriores, las bases de datos *NoSQL* en general y de *MongoDB* en particular, surgieron con una serie de objetivos, entre los que destaca eliminar las restricciones del modelo relacional.

Es por ello que a priori parezca que no tiene mucho sentido sentar una base de reglas para transformar esquemas pertenecientes al modelo relacional en bases de datos que huyen de los principios de estos mismos y que aparecieron gracias a avances tecnológicos que ampliaban la forma de almacenar los datos. Sin embargo, y a pesar de cada vez más aplicaciones utilizan bases de datos *NoSQL* para almacenar la información, el estándar relacional sigue muy presente a día de hoy, siendo la base de la gestión y almacenamiento de datos en buena parte de organizaciones.

Sin embargo, solo hace falta consultar foros de desarrolladores de software en internet para percatarse de que muchos de estos realizaron su software teniendo como base de almacenamiento de datos una base de datos relacional y desean trasladarlos o migrarlos a *MongoDB* para beneficiarse de ciertas ventajas de estos nuevos modelos, pero manteniendo características del modelo tradicional, como puede ser la forma en la que desean almacenar los datos.

Puede que sea una tarea muy compleja la obtención, a partir de una base de datos *NoSQL* (en la que posiblemente no se haya creado un modelo de la misma previo a su implementación) un esquema gráfico en el que se vea exactamente la forma en la que están estructurados los datos. Sin embargo, como veremos a continuación, sí que es posible obtener a partir de un esquema relacional una base de datos implementada con esta nueva tecnología, pero manteniendo la mayor parte de las características de dicho esquema.

Gracias a la ausencia de restricciones que proponía el modelo tradicional, la base de datos resultante podría implementar, sin necesidad de realizar modificaciones en su modelo, una serie de características pertenecientes a esta nueva tecnología lo cual la dotaría en parte, de los beneficios del esquema tradicional y a su vez, de las características adicionales de este nuevo tipo de bases de datos.

A continuación, procedemos a definir una serie de reglas de transformación para plasmar de la forma más precisa (y, por ende, con una serie de restricciones) la información presente en un diagrama Entidad-Relación en una base de datos no relacional (*MongoDB*) lo cual es algo completamente opuesto a la naturaleza de estas últimas.

Asimismo, para cada una de las reglas de conversión de los diferentes componentes que forman un diagrama Entidad-Relación, se ejemplificará con casos concretos dicha conversión. En este capítulo dichos ejemplos no formarán parte de un único universo con la finalidad de abarcar el total de casos propuestos. En el siguiente capítulo se realizará la conversión de un diagrama

Entidad-Relación completo (no solo de sus componentes individuales) en una base de datos con *MongoDB*.

Junto a esta memoria se adjunta un Script (Hacer clic) [TFG - Método de transformación - Rubén de Diego Varona Capítulo 4.js][15] en el que se puede consultar el código de todos los ejemplos realizados. Haciendo clic sobre el enlace de este mismo párrafo se redirigirá al repositorio *GitHub* en el que se puede ver el código utilizado.

4.1 Entidades.

En el diagrama Entidad-Relación [Capítulo 2] nos encontramos con dos tipos de entidades, las denominadas *entidades fuertes* y las *entidades débiles*. Las fuertes están representadas en el mismo como un rectángulo, mientras que las débiles lo están con un doble rectángulo o un rectángulo con doble línea. [Figura 4.1]

Figura 4.1 Entidades débiles y fuertes.

Recordemos la definición de entidad débil: Si la existencia de la entidad *x* (entidad débil) depende de la existencia de la entidad *y*, entonces decimos que *x depende por existencia* de y. En otras palabras, la existencia de las entidades débiles depende de la existencia de otras entidades con las que se relacionan.

Como hemos comentado anteriormente, es posible encontrarnos con diagramas en los que las entidades fuertes y débiles aparecen representadas únicamente con un rectángulo. En este caso, y para diferenciar las débiles de las fuertes, basta con observar si la entidad posee o no un *atributo identificador* (el cual aparece subrayado) [Figura 4.02].

[15] Rubén de Diego Varona "Código capítulo 4 TFG" [Recurso online] Disponible en: https://github.com/rubdedi/MongoDB/blob/main/TfgCap%C3%ADtulo4.js Fecha última consulta: 01/06/2024

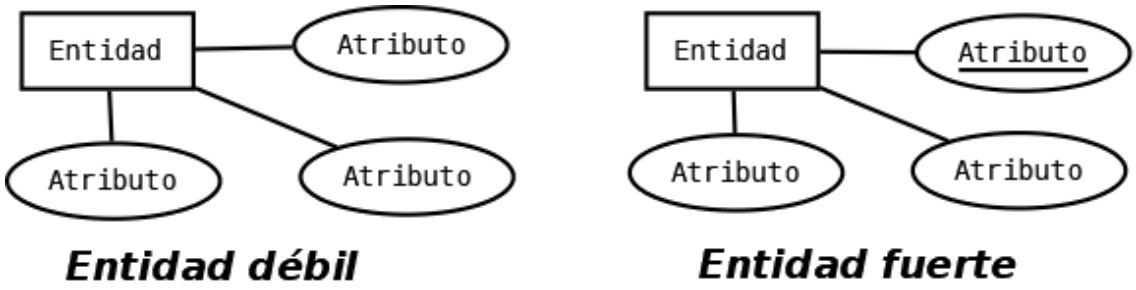

Figura 4.2 Entidades débiles y fuertes.

El primer paso que tendremos que realizar a la hora de convertir el esquema Entidad-Relación en una base de datos con *MongoDB* será *identificar las entidades fuertes y débiles en el diagrama*.

4.1.1 Transformación de entidades en MongoDB.

4.1.1.1 Entidades fuertes.

En *MongoDB* un **conjunto de entidades fuertes** se convertirá en una colección cuyo nombre será el del conjunto de entidades. Como veremos posteriormente, en general, los atributos se corresponderán con los campos de los documentos. [Figura 4.03].

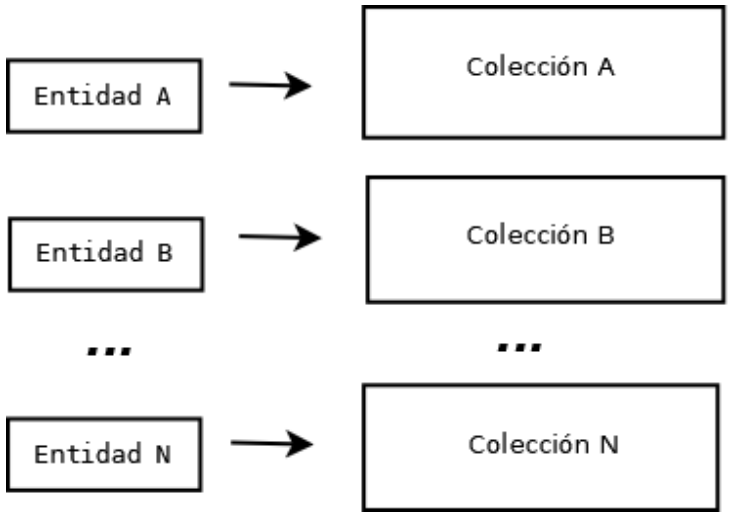

Figura 4.3 Entidades a colecciones.

Ejemplo: Veamos cómo se transformarían las entidades fuertes en el siguiente ejemplo: Tenemos un conjunto de entidades *Clientes* las cuales se relacionan mediante la relación *Tener* con un conjunto de entidades *Préstamos*: [Figura 4.4].

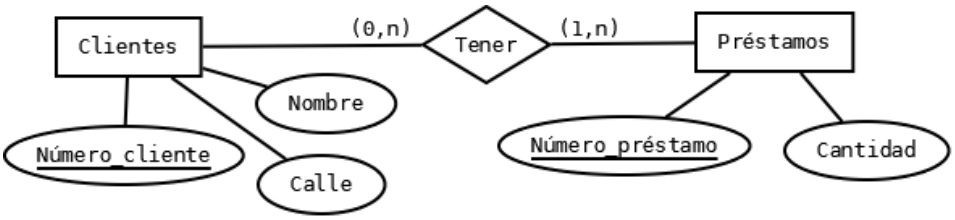

Figura 4.4 Ejemplo conversión entidades.

El primer paso sería identificar las entidades en el diagrama, las cuales serían *Clientes* y *Préstamos*.

El conjunto de entidades *Clientes* se convertiría en la colección *Clientes* en *MongoDB*, de igual forma el conjunto de entidades *Préstamos* se convertiría en la colección *Préstamos* en *MongoDB*.

Bastaría con crear las colecciones en *MongoDB* con los comandos adecuados como vimos en el capítulo anterior.

```
Ejemplo> db.createCollection("Clientes")
```

```
Ejemplo> db.createCollection("Préstamos")
```

4.1.1.2 Entidades débiles.

Una de las características de las **entidades débiles** es que, al contrario que aquellas de las que dependen, no tienen identificador propio o atributo identificador.

Sin embargo, si en *MongoDB* creamos un documento sin un campo *_id*, será *MongoDB* el que automáticamente lo cree y le asigne un ObjectID de tipo Bson único. Como vimos en el capítulo anterior, no puede existir ningún documento perteneciente a una colección sin *_id* en *MongoDB*.

Al igual que hemos hecho con las entidades fuertes, podríamos convertir las entidades débiles en colecciones, pero a la hora de crearlas, y aunque no introdujésemos un *ObjectID* para cada una de ellas, *MongoDB* lo crearía de forma predeterminada, con lo que tendríamos información redundante y no estaríamos modelando con el concepto base del esquema Entidad-Relación. Si siguiésemos este patrón, posteriormente deberíamos referenciar dichos documentos para establecer la relación existente en el modelo ER. En una sección posterior de este documento estudiaremos la forma de referenciar documentos.

Para transformar un **conjunto de entidades débiles** en *MongoDB* se crea un campo clave valor dentro del documento perteneciente a la colección que representa la entidad fuerte con la que esté relacionada. Este campo tendrá como clave el nombre del conjunto de entidades débiles y como valor un array con tantos pares clave valor como número de atributos tuviese el conjunto de

entidades débiles. [Figura 4.5]. A este proceso en *MongoDB* se le denomina incrustar documentos, proceso que veremos más adelante.

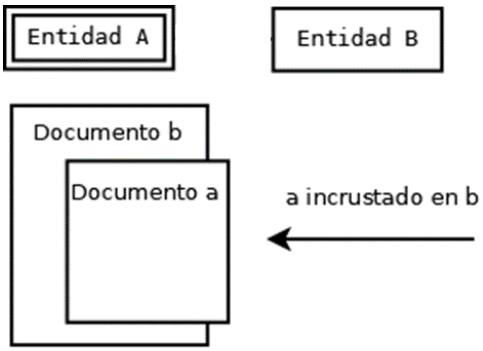

Figura 4.5: Entidades débiles incrustadas.

Sean *A* y *B* conjuntos de entidades (*A* entidad fuerte y *B* entidad débil) relacionadas entre sí. Cada una de ellas con *i* y *j* atributos respectivamente. [Figura 4.6].

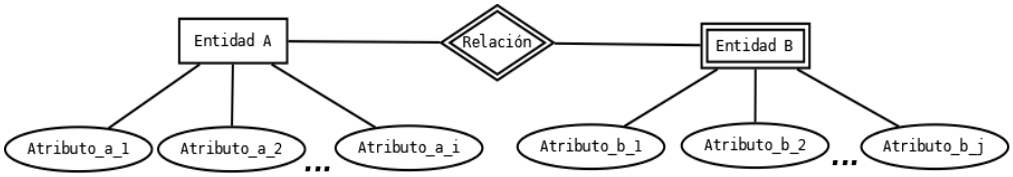

Figura 4.6 Entidades débiles incrustadas.

En su paso a *MongoDB* el conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* con *i+1* campos clave valor de la siguiente forma: [Tabla 4.1].

```
// Colección A
// Documento a
{
   _id: < ObjectId_a >,
   Atributo_a_1: valor_a_1,
   Atributo_a_2: valor_a_2,
   ...
   Atributo_a_i: valor_a_i
}
```

Tabla 4.1 Documento a.

Al incrustar el conjunto de entidades débiles *B* relacionadas con el conjunto de entidades fuertes *A*, la transformación resultante sería el mismo documento *A* más un clave valor con clave el nombre de la colección *B* y valor un array conteniendo tantos campos clave valor como atributos tuviese el conjunto de entidades débiles *B* (*j* campos) con sus respectivos nombres y valores de atributos. [Tabla 4.2].

Nota: Solo para el caso de las entidades débiles en su paso a *MongoDB*, a la hora de incrustarlas en el documento padre (correspondiente a la entidad fuerte del diagrama *ER*), omitiremos el *ObjectID* de la entidad débil, para, de esta forma mantener la idea de que las entidades débiles no han de tener atributo identificador en el diagrama *ER*. No crearemos colecciones para las entidades débiles, ya que serán directamente datos incrustados en las entidades fuertes correspondientes.

```
// Colección A
// Documento a
{
   _id: < ObjectId_a >,
   Atributo_a_1: valor_a_1,
   Atributo_a_2: valor_a_2,
   ...
   Atributo_a_i: valor_a_i
   Nombre_colección_B: [{
      Atributo_b_1: valor_b_1,
      Atributo_b_2: valor_b_2,
      ...
      Atributo_b_j: valor_b_j,
     }]
}
```

Tabla 4.2 Documento b incrustado en a.

Ejemplo: En la siguiente figura podemos observar que existen dos conjuntos de entidades distintas. El conjunto de entidades fuertes *Préstamos* y el Conjunto de entidades débiles *Pagos*. [Figura 4.7].

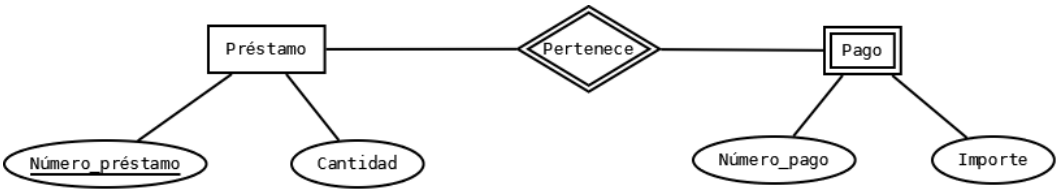

Figura 4.7 Ejemplo entidad débil.

Como acabamos de ver el conjunto de entidades *Préstamos* se transformaría en la colección *Préstamos*. Ahora bien, *Pagos* no sería una nueva colección en *MongoDB* ya que se trata de una entidad débil. Cada instancia del conjunto de entidades *Préstamos* sería un nuevo documento, mientras que la información del *pago* asociado a dicho documento estaría incrustada en el mismo.

Supongamos una instancia del conjunto de entidades *Préstamo* con los siguientes valores de atributos: [Tabla 4.3].

Préstamo	Instancia 1
Número_préstamo	123456
Cantidad	10

Tabla 4.3 Instancia Préstamo.

Y una instancia del conjunto de entidades Pago con los valores de atributos: [Tabla 4.4].

Pago	Instancia 1
Número_pago	654321
Importe	100

Tabla 4.4 Instancia Pago.

Tras convertir la entidad fuerte *Préstamo* en su correspondiente colección con igual nombre, la instancia de *pago* se incrustaría en la correspondiente instancia *préstamo* mediante un nuevo campo clave valor con clave nombre *Pago* y valor el array que contiene el nombre de los atributos y su respectivo valor. [Tabla 4.05].

```
// Colección Préstamos
// Documento préstamo1
{
   _id: ObjectId('65c899073342ee53cbf8f866'),
   Número_préstamo: 123456,
   Cantidad: 10,
   Pago: [{
        Número_pago: 654321,
        Importe: 100
      }]
}
```

Tabla 4.5 Ejemplo conversión entidad débil.

4.2 Atributos.

Una vez identificadas las entidades es necesario hacerlo con los atributos. [Figura 4.8].

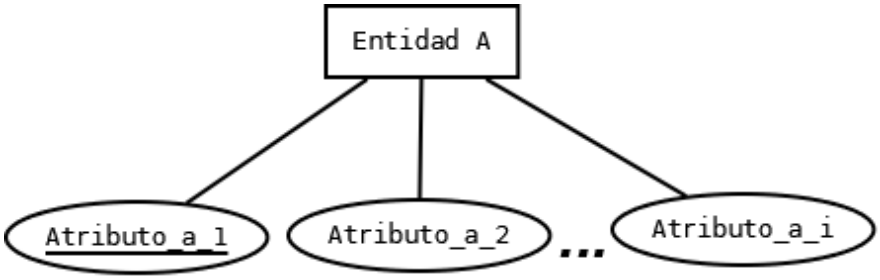

Figura 4.8 Ejemplo atributos diagrama ER.

En general, los atributos del diagrama Entidad-Relación pasan a ser *campos* (del tipo clave valor) en un documento de *MongoDB*. Si una entidad tiene *i* atributos, en *MongoDB* el documento asociado a dicha entidad tendrá *i + 1* campos clave valor: *i* correspondientes a los atributos de la entidad más un campo adicional que contendrá el *_id* del documento. Las claves del documento serán los nombres de los atributos, mientras que los valores serán los mismos asociados a los atributos de la entidad [Tabla 4.06].

```
// Colección A
// Documento a
{
   _id: < ObjectId_a >,
   Atributo_a_1: valor_a_1,
   Atributo_a_2: valor_a_2,
   ...
   Atributo_a_i: valor_a_i
}
```

Tabla 4.6 Ejemplo atributos en MongoDB.

Ejemplo: Tenemos el conjunto de entidades *Clientes* con los siguientes atributos: Número, Nombre y Calle. [Figura 4.9].

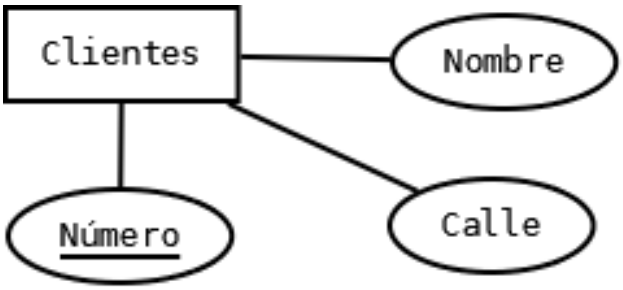

Figura 4.9 Ejemplo atributos diagrama ER.

Y una instancia del conjunto de entidades *Clientes* con los siguientes valores de atributos [Tabla 4.7].:

Clientes	Instancia 1
Número	14523
Nombre	Rubén de Diego
Calle	Inventada

Tabla 4.7 Ejemplo instancia Clientes.

En *MongoDB* el conjunto de entidades Clientes se transformaría en la colección con mismo nombre y la instancia en un documento con la siguiente información. [Tabla 4.8].

```
// Colección Cliente
// Documento cliente
{
   _id: ObjectId('65c642e2ffdb3aec9dbf0d84'),
   Número: 14526,
   Nombre: Rubén de Diego,
   Calle: Inventada
}
```

Tabla 4.8 Ejemplo transformación de atributos en MongoDB.

4.2.1 Atributo identificador.

Los **atributos identificadores o clave** son aquellos que nos permiten identificar una determinada identidad unívocamente dentro de un conjunto de estas. En el modelo Entidad-Relación son aquellos atributos que aparecen subrayados. [Figura 4.10].

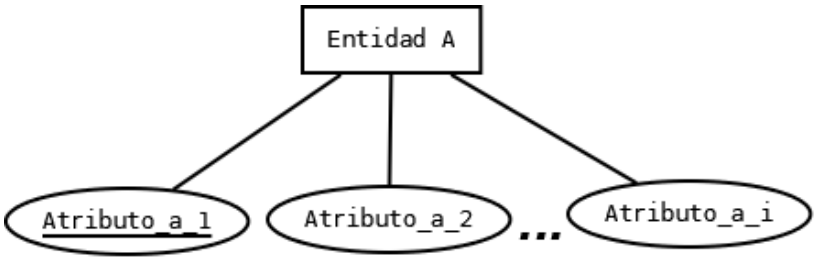

Figura 4.10 Ejemplo atributo identificador diagrama ER.

En su paso del diagrama Entidad-Relación al modelo relacional, el atributo clave o atributo identificador se correspondería con la primary key de la tabla. Aunque no estemos transformando el modelo relacional en *MongoDB* (sino que partimos de un esquema ER y directamente deseamos migrarlo a *MongoDB*) en este último el concepto primary key también aparece, con nombre *_id* [16][Capítulo 3]

El campo *_id* posee, al igual que el atributo identificador, un valor único. En las bases de datos relacionales las *PK* son habitualmente un número entero almacenado en cada fila del campo id (el identificador de la tabla). En *MongoDB*, el campo *_id* almacenaría el atributo identificador del documento el cual es un ObjectID Bson.

Por lo tanto, podríamos equiparar el *atributo identificador* de un diagrama ER como el campo *_id* de un documento en *Mongodb* [Tabla 4.09].

```
// Colección A
// Documento a
{
   _id: <valor_a_1>, // _id: <ObjectId_a> = atributo identificador de ER
   Atributo_a_2: valor_a_2,
   ...
   Atributo_a_i: valor_a_i
}
```

Tabla 4.9 Ejemplo atributos identificador como campo _id.

Por defecto es *MongoDB* el encargado de aplicar el valor de *_id* de forma predetermina a los diferentes documentos que insertemos, a menos que seamos nosotros los que indiquemos dicho valor siempre y cuando sea único para cada uno de los documentos de una colección.

Sin embargo, siguiendo este último enfoque, cuantos más documentos introduzcamos en la base de datos mayor será la probabilidad de que dos *_id* coincidan en la misma colección, e

[16] "Primary key" – MongoDB MondoDB Glossary [Recurso Online] Disponible en: https://www.mongodb.com/docs/manual/reference/glossary/#std-term-primary-key Fecha última consulta: 01/06/2024

independientemente, esta acción tiene un gran impacto en el rendimiento: al insertar documentos uno por uno el hecho de ser nosotros los que definamos el valor del _id no tiene apenas efecto en el rendimiento, sin embargo, al introducirlos en grandes volúmenes, el impacto es significativo.[17]

Durante la realización de este TFG se han realizado varias transformaciones de elementos contenidos en los diagramas Entidad-Relación en documentos y colecciones en *MongoDB*. En un principio, y para mantener el mayor número de características presentes en el esquema ER a la hora de migrarlo a esta base de datos, se decidió tomar como campo *_id* de los documentos, el atributo identificador de las entidades, sin embargo, e independientemente del impacto comentado en el párrafo anterior, a la hora de presentar los ejemplos (tanto los generales como los particulares) de transformación, al equiparar el *_id* con el atributo identificador estos carecían de cohesión y algunos entraban en conflicto con una de las pocas "normas" de *MongoDB*: cada documento ha de tener un único *_id*. Es por ello que se ha decidido, para todas las transformaciones, incluir los atributos identificadores de las entidades del diagrama Entidad-Relación como un campo clave valor más y permitir que sea *MongoDB* el encargado de crear el *_id* de los documentos.

Podemos concluir por tanto que el concepto de atributo identificador del esquema Entidad-Relación se convierte en un campo *clave* valor con clave el nombre del atributo identificador y valor el propio del atributo identificador en *MongoDB* y que será la propia base de datos la que cree el propio *_id* del documento [Tabla 4.10].

```
// Colección A
// Documento a
{
   _id: < ObjectId_a >,
   Atributo_a_1: valor_a_1, // Atributo identificador en ER
   Atributo_a_2: valor_a_2,
   ...
   Atributo_a_i: valor_a_i
}
```

Tabla 4.10 : Ejemplo atributos identificador como campo adicional _id.

4.2.2 Atributos simples y compuestos.

Los atributos **simples** en el esquema Entidad-Relación son aquellos cuyo valor es indivisible en partes con significado propio. [Figura 4.11].

[17] Richard Gupta "Mongodb primary key using the default objectid vs picking a custom value" [Recurso Online] Disponible en: https://medium.com/@rishabh011/mongodb-primary-key-using-the-default-objectid-vs-picking-a-custom-value-12399d828e33 Fecha última consulta: 01/06/2024

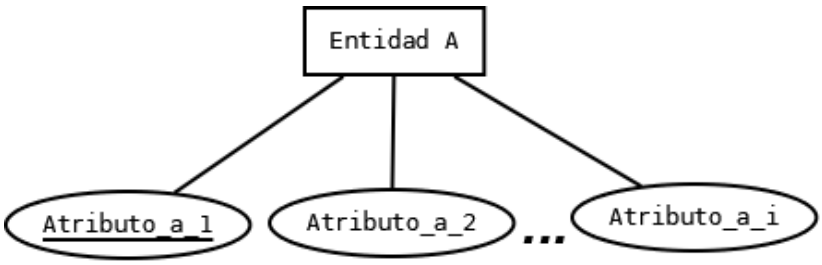

Figura 4.11 Atributos simples ER.

En *MongoDB* los atributos simples pasan a ser campos clave valor con clave el nombre del atributo y valor el valor de la instancia [Tabla 4.11].

```
// Colección A
// Documento a
{
   _id: < ObjectId_a >,
   Atributo_a_1: valor_a_1,
   Atributo_a_2: valor_a_2,
   ...
   Atributo_a_i: valor_a_i
}
```

Tabla 4.11 Atributos simples MongoDB.

Los atributos **compuestos** son aquellos los cuales pueden dividirse en varias partes con significado propio. [Tabla 4.12].

.

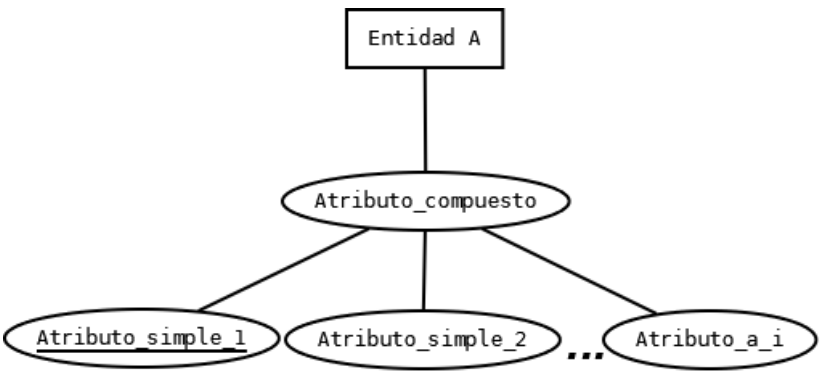

Figura 4.12 Atributos compuestos ER.

En *MongoDB* los atributos compuestos pasan a ser un campo clave valor con clave el nombre del atributo compuesto y valor tantos pares clave valor como atributos simples compongan dicho atributo [Tabla 4.12].

```
// Colección A
// Documento a
_id: <ObjectId_a>,
Atributo_compuesto: {
  Atributo_simple_1: valor_1,
  Atributo_simple_2: valor_2,
  ...
  Atributo_simple_i: valor_i,
}
...
```

Tabla 4.12 Atributos compuestos MongoDB.

Siguiendo el ejemplo utilizado para la conversión de entidades utilizado en el apartado anterior, si el atributo de Cliente *Dirección* fuese tomado como atributo compuesto de *Calle*, *Número*, *Piso*, *Letra*, *Ciudad* y *Código Postal*: [Figura 4.13].

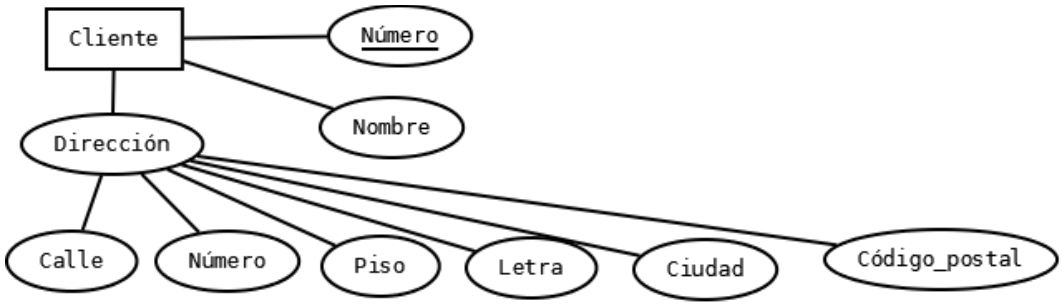

Figura 4.13 Atributos compuestos ER.

Al convertirlo en *MongoDB* con la regla descrita para los atributos compuestos obtendríamos un documento de la siguiente forma*:* [Tabla 4.13].

```
// Colección Cliente
// Documento cliente
_id: ObjectId('65c6436fffdb3aec9dbf0d85'),
Número: 123456,
Nombre: Rubén,
```

```
Dirección: {
    Calle: Inventada,
    Número: 1,
    Piso: 4,
    Letra: F,
    Ciudad: Valladolid,
    Código_postal: 47009
}
```

Tabla 4.13 Ejemplo atributos compuestos MongoDB.

4.2.3 Atributos mono valuados y multivaluados.

Los atributos mono valuados son aquellos que solo admiten un valor, mientras que los multi valuados admiten uno o más valores. [Figura 4.14].

Figura 4.14 Atributos multi-valuados ER.

Los atributos **mono-valuados** se representan de la misma manera que los atributos simples, mientras que los atributos **multi-valuados** se representan como un campo clave valor cuya clave es el nombre del atributo multi-valuado y valor un array conteniendo *n* pares clave valor (siendo *n* el número de posibles valores del atributo multi-valuado). La clave de cada uno de los componentes del array será el nombre del atributo multivaluado seguido de un número entero partiendo de 1 hasta *n*, mientras que el valor será el posible de entre los valores del atributo multi-valuado [Tabla 4.14].

```
// Colección A
// Documento a
{
...
Atributo_multi_valuado: {[
    Atributo_multi_valuado_1: Valor_1,
    Atributo_multi_valuado_2: Valor_2,
    ...
    Atributo_multi_valuado_n: Valor_n
  ]}
```

```
...
}
```

Tabla 4.14 Atributos multi-valuados MongoDB.

Por ejemplo, la entidad *Cliente*, la cual posee los siguientes atributos: *Número*, *Nombre* y *Teléfono*, los dos primeros atributos simples, mientras que *Teléfono* multivaluado. [Figura 4.15].

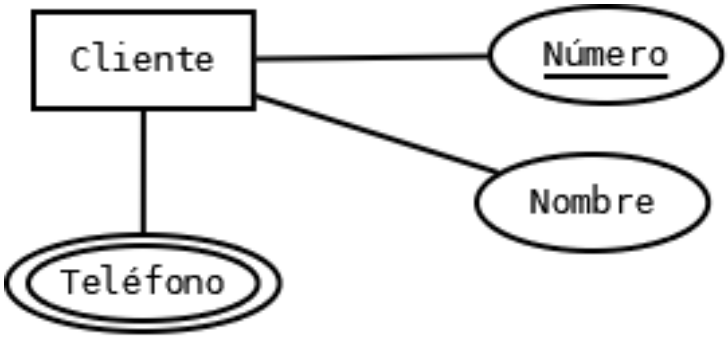

Figura 4.15 Ejemplo atributo multi-valuado ER.

Una instancia de dicha entidad posee los siguientes valores para los tres atributos [Tabla 4.15].

Clientes	Instancia 1
Número	143214
Nombre	Henar
Teléfono	983123456 983987654 689123456

Tabla 4.15 Instancia Cliente.

En su paso a *MongoDB*, el documento resultante de la transformación contendría la siguiente información: [Tabla 4.16].

```
// Colección Cliente
// Documento cliente
{
_id: ObjectId('65c6436fffdb3aec9dbf0d85'),
Número: 143214
Nombre: Henar,
Teléfono: [{
```

```
    Teléfono1: 983123456,
    Teléfono2: 983987654,
    Teléfono3: 689123456
    ]}
}
```

Tabla 4.16 Ejemplo Atributos multi-valuados MongoDB.

4.2.4 Atributos derivados.

Los atributos derivados son aquellos cuyo valor puede ser calculado mediante el valor de otros atributos. [Figura 4.16].

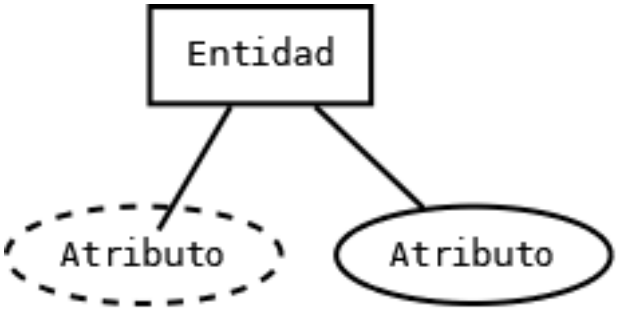

Figura 4.16 Ejemplo atributo derivado ER.

En *MongoDB* se utilizan las denominadas *Aggregation operations*, término que podemos traducir como *operaciones de suma* u *operaciones de adición*. Estas operaciones permiten agrupar valores de varios documentos, realizar operaciones sobre un conjunto de datos y devolver un único resultado y analizar el cambio de los datos a través del tiempo.

No podemos definir un método general de transformación de atributos derivados en un esquema ER en su correspondiente par clave valor en *MongoDB* ya que dependiendo del atributo derivado que se desee calcular (y el atributo del que se desee "derivar") habrá que hacer una serie de operaciones mediante las posibilidades que nos ofrece *MongoDB* con las *Aggregation operations*.

4.2.5 Atributos requeridos y opcionales.

Como su propio nombre indica definiremos atributos requeridos como aquellos que han de estar presentes en nuestra base de datos, mientras que los opcionales pueden aparecer o no. [Figura

4.17]. En los diagramas Entidad-Relación todos los atributos que aparecen en el mismo son requeridos.

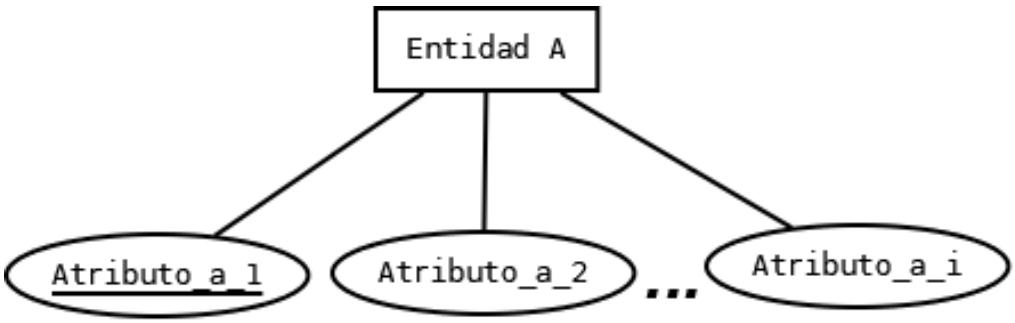

Figura 4.17 Ejemplo atributo requerido ER.

Sin embargo, *MongoDB* nos permite a la hora de crear colecciones, la posibilidad de aplicar una serie de restricciones a los futuros documentos que formen parte de esa determinada colección mediante los denominados *JSON Schema* que vimos en capítulo referente a *MongoDB*. [Capítulo 3].

Para definir los atributos como requeridos, a la hora de transformarlos en campos en *MongoDB,* basta con marcarles como *required* en dicho esquema. De esta forma el esquema validará los documentos que introduzcamos en esa determinada colección y, si a la hora de realizarlo, falta algún campo marcado como required en el mismo, no nos permitirá crear la colección devolviéndonos un error para informarnos. [Tabla 4.17].

```
// Colección A
Validator: {
    $jsonSchema: {
        bsonType: “object”,
        title: “Título”,
        required: [atributo_a_1, atributo_a_2, ..., atributo_a_i]

...
...
    }
}
```

Tabla 4.17 Ejemplo JsonSchema Validator

Siguiendo con el ejemplo planteado con anterioridad tomemos el siguiente componente del diagrama Entidad-Relación. [Figura 4.18].

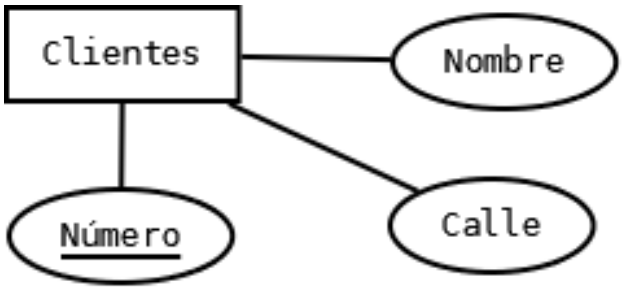

Figura 4.18 Ejemplo entidad ER.

La entidad *cliente* tiene tres atributos: *Número*, *Nombre* y *Calle*.

A la hora de crear una base de datos en *MongoDB* podemos marcar los campos correspondientes a los atributos que posee esta entidad como requeridos como vemos en el siguiente ejemplo:

```
// Creación de la colección Cliente con atributos obligatorios
db.createCollection("Cliente", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Cliente",
         required: [ "numero", "nombre", "calle"],
         properties: {
            numero: {
               bsonType: "int",
               minimum: 1,
               maximum: 100000000000,
               description: "'numero' ha de ser un integer en el rango [1,100000000000]
y es necesario"
            },
            nombre: {
               bsonType: "string",
               description: "'nombre' ha de ser un string y es necesario"
            },
            calle: {
               bsonType: "string",
               description: "'calle ha de ser un string y es necesario"
            }
         }
      }
   }
} )
```

4.3 Relaciones.

En *MongoDB* la finalidad a la hora de diseñar el modelo de la base de datos se basa en la estructura de los documentos y el comportamiento de la posible aplicación para la que estemos

diseñando esa base de datos para representar las relaciones existentes entre los datos. Como comentamos en la introducción de este TFG el objetivo de este estudio no es obtener una base de datos lo más eficiente posible, sino transformar un esquema Entidad-Relación en una base de datos en este sistema, manteniendo, en la medida de lo posible, las restricciones del diagrama Entidad-Relación.

En las bases de datos relacionales las relaciones se efectúan utilizando las claves primaria y foránea. En *MongoDB* no se utiliza esta técnica, sino que las relaciones se crean mediante dos técnicas diferentes: *Incrustando documentos* dentro de otros y *mediante referencias* entre documentos.

a) Mediante **documentos incrustados** (también denominados embebidos) – Es una forma de aunar en un único documento datos relacionados. En este modelo introducimos la información de uno o varios documentos(s) dentro de otro(s). Como resultado, en vez de tener documentos independientes, obtenemos un único documento con toda la información referente a todos los documentos relacionados.

b) Mediante **referencias** – De esta forma mantenemos la estructura de documentos independientes. Las relaciones se crean mediante referencias manuales, las cuales consisten en incluir el (los) campo(s) *_id* del (los) documento(s) relacionado(s) con otro(s) como un campo clave valor.

En él [Capítulo 2] vimos que no solo existen *relaciones binarias* en los diagramas Entidad-Relación, sino que dependiendo del número de entidades que estén relacionadas entre sí nos podemos encontrar relaciones ternarias, cuaternarias… n-arias. En este capítulo nos vamos a centrar solo en las relaciones binarias, ya que una vez conocidas estas (y al no existir para ninguno de sus transformaciones un único método) podemos extrapolar las reglas de transformación que enunciaremos a relaciones en las que intervengan más de dos entidades (pudiendo bien referenciar o incrustar como veremos en las binarias). Independientemente, en el diagrama ER, siempre se puede disminuir el grado de una relación, si este es superior a dos, con las transformaciones necesarias.

A continuación, se expondrán ambas formas (incrustar documentos y referenciarlos) para los tipos de relaciones existentes en un diagrama Entidad-Relación (*1:1*, *1: N* y *N:M*). Para cada una de las reglas generales se realizará un ejemplo práctico para todas y cada una de las posibilidades de transformación.

4.3.1 Relaciones uno a uno.

Sean *A* y *B* conjuntos de entidades relacionadas uno a uno. Cada una de ellas con *i* y *j* atributos respectivamente [Figura 4.19].

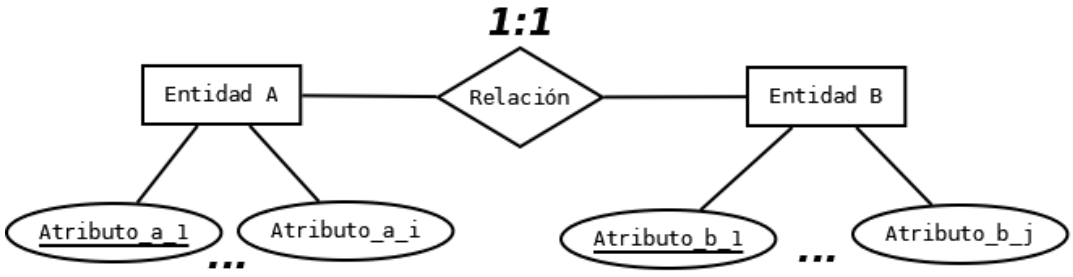

Figura 4.19 Relación uno a uno ER.

En su paso a *MongoDB* el conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* el cual tendría *i+1* campos clave valor. El conjunto de entidades *B* sería la colección *B* y una instancia de *b* sería un documento *b* con *j+1* campos clave valor.

4.3.1.1 Relaciones uno a uno con documentos incrustados.

Una de las formas de crear relaciones del tipo uno a uno entre datos es utilizando documentos incrustados unos dentro de otros.

Dado un documento *a* resultante de la transformación del conjunto de entidades *A* en la colección *A,* con *i +1* campos clave valor [Tabla 4.18].

```
// Colección A
// Documento a
{
   _id: <ObjectId(id_a)>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   ...
   atributo_a_i: valor_a_i
}
```

Tabla 4.18 Documento a.

Y un documento *b* resultante de la transformación del conjunto de entidades *B* en la colección *B, con j+1* campos clave valor [Tabla 4.19].

```
// Colección B
// Documento b
{
```

```
    _id: <ObjectId(id_b)>,
    atributo_b_1: valor_b_1,
    atributo_b_2: valor_b_2,
    ...
    atributo_b_j: valor_b_j
}
```

Tabla 4.19 Documento b

Y ambos relacionados entre sí en el diagrama Entidad-Relación mediante una relación tipo uno a uno:

Incrustando el documento *b* en el documento *a* obtendríamos un único documento *a*, de la colección *A*, de la siguiente forma. El documento resultante *a* contendría toda la información de sí mismo y la del documento *b* [Tabla 4.20].

```
// Colección A
// Documento a
{
   _id: <ObjectId(id_a)>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   ...
   atributo_a_i: valor_a_i
   Nombre_colección_B: // Documento b incrustado
   {
      _id: <ObjectId(id_b)>,
      atributo_b_1: valor_b_1,
      atributo_b_2: valor_b_2,
      ...
      atributo_b_j: valor_b_j
      }
}
```

Tabla 4.20 Documento b incrustado en a

La posición en la que introduzcamos nuestro documento a incrustar es indiferente siempre y cuando mantenga la estructura definida, pero para mantener una estructura igual en todas las transformaciones que hagamos de aquí en adelante introduciremos el documento incrustado en la última posición del documento principal.

De igual forma podríamos incrustar el documento *a* en el documento *b* obteniendo de esta forma un único documento *b*, de la colección *B*, con la información de ambos, solo que con diferente estructura [Tabla 4.21].

```
// Colección B
// Documento b
{
   _id: <ObjectId(id_b)>,
   atributo_b_1: valor_b_1,
   atributo_b_2: valor_b_2,
   ...
   atributo_b_j: valor_b_j,
   Nombre_coleccion_A: // Documento a incrustado
   {
      atributo_a_1: valor_a_1,
      atributo_a_2: valor_a_2,
      ...
      atributo_a_i: valor_a_i
      }
}
```

Tabla 4.21: Documento a incrustado en b

Ejemplo: Veamos un ejemplo concreto de la transformación de una relación uno a uno en un diagrama Entidad-Relación en su respectivas colecciones y documentos en *MongoDB*.

Tenemos la entidad *Estudiante* (cuyos atributos son *Número_estudiante*, *Nombre*, *Curso* y *Centro*) y la entidad *Tarjeta_Uva* (con atributos *Número_Tarjeta* y *Tipo*, pudiendo ser este último de crédito, débito o solo identificativa). Ambas entidades relacionadas entre sí mediante *Posee*. Un *estudiante* solo va a poder poseer una *tarjeta _Uva*, mientras que una *tarjeta_Uva* solo puede pertenecer a un *estudiante*. [Figura 4.20].

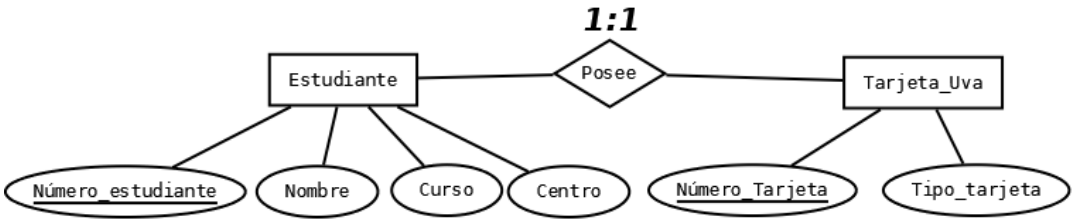

Figura 4.20 Ejemplo relación uno a uno ER.

Tenemos una instancia del conjunto de entidades *Estudiantes* con los siguientes valores de atributos: [Tabla 4.22].

Estudiante	Instancia 1
Número_estudiante	542531987
Nombre	Miguel Ángel
Curso	Cuarto
Centro	ETSII

Tabla 4.22 Instancia 1 estudiante.

Y una instancia del conjunto de entidades *Tarjeta_Uva* con los siguientes valores de atributos: [Tabla 4.23].

Tarjeta_Uva	Instancia 1
Número_Tarjeta	VA1133545
Tipo_tarjeta	Identificativa

Tabla 4.23 Instancia 1 de tarjeta uva

Aplicando las reglas de transformación definidas, en *MongoDB* tendríamos dos documentos: *estudiante_1* y *tarjeta_uva_1* pertenecientes a las colecciones *Estudiante* y *Tarjeta_Uva* respectivamente con la siguiente información: [Tabla 4.24]. [Tabla 4.25].

```
// Colección Estudiante
// Documento estudiante_1
{
   _id: ObjectId ('65c640a4ffdb3aec9dbf0d73 '),
   Número_estudiante: 542531987,
   Nombre: Miguel Ángel,
   Curso: Cuarto,
   Centro: ETSII
}
```

Tabla 4.24 Documento estudiante_1.

```
// Colección Tarjeta_Uva
// Documento tarjeta_uva_1
{
   _id: ObjectId('65c65065ffdb3aec9dbf0d87'),
   Número_Tarjeta: VA1133545,
   Tipo_tarjeta: Identificativa
```

```
}
```

Tabla 4.25 Documento tarjeta_uva_1.

Incrustando el documento *tarjeta_uva_1* en el documento *estudiante_1* obtendríamos un único documento perteneciente a la colección *Estudiante* el cual contendría tanto la información de *estudiante* como la de la *tarjeta_Uva* con la que está relacionada: [Tabla 4.26].

```
// Colección Estudiante
// Documento estudiante_1
{
   _id: ObjectId ('65c640a4ffdb3aec9dbf0d73 '),
   Número_estudiante: 542531987,
   Nombre: Miguel Ángel,
   Curso: Cuarto,
   Centro: ETSII,
   Tarjeta_uva: // Documento tarjeta_uva_1 incrustado
   {
      _id: ObjectId('65c65065ffdb3aec9dbf0d87'),
      Número_tarjeta: VA1133545,
      Tipo_tarjeta: Identificativa,
}
```

Tabla 4.26 Documento tarjeta_uva_1 incrustado en estudiante_1.

Como hemos comentado anteriormente también se podría incrustar el documento *estudiante_1* en el documento *tarjeta_uva_1*. La decisión de incrustar un determinado documento en otro dependerá del desarrollador de la base de datos y como desea que estén estructurados los mismos.

Incrustando el documento *estudiante_1* en el documento *tarjeta_uva_1* obtendríamos un único documento *tarjeta_uva_1* el cual contendría tanto la información de la tarjeta como la del estudiante con la que está relacionada: [Tabla 4.27].

```
// Colección Tarjeta_Uva
// Documento tarjeta_uva_1
{
   _id: ObjectId ('65c65065ffdb3aec9dbf0d87'),
   Número_tarjeta: VA1133545,
   Tipo_tarjeta: Identificativa
   Estudiante: // Documento estudiante_1 incrustado
```

```
    {
        _id: ObjectId ('65c640a4ffdb3aec9dbf0d73 '),
        Número_estudiante: 542531987,
        Nombre: Miguel Ángel,
        Curso: Cuarto,
        Centro: ETSII
}
```

Tabla 4.27 Documento estudiante_1 incrustado en tarjeta_uva_1.

4.3.1.2 Relaciones uno a uno con documentos referenciados.

La segunda forma de transformar relaciones uno a uno en *MongoDB* es referenciando documentos.

Dados los documentos resultantes de la transformación de entidades en Colecciones en *MongoDB* descritos en las tablas [Tabla 4.18] y [Tabla 4.19].

Si un documento *a* se relaciona con un documento *b* referenciamos el ObjectId de *b* en el documento *a*, añadiendo un campo clave valor con clave el nombre de la colección *B* y valor el *ObjectId* del documento *b* con el que está relacionado. De igual modo referenciaríamos el documento *a* en el documento *b,* referenciando el ObjectId de *a* en el documento *b* mediante un par clave valor con clave el nombre de la colección *A* y valor el ObjectId del documento *a* con el que está relacionado. La posición en la que se añadiría el nuevo campo clave valor referenciando al documento es indiferente, pero la introduciremos en último lugar para mantener la misma estructura en todos los ejemplos de este trabajo.

Obtendríamos de esta forma el documento *b* referenciado dentro del documento *a* con los siguientes campos: [Tabla 4.28]

```
// Colección A
// Documento a
{
   _id: <ObjectId(id_a)>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   atributo_a_3: valor_a_3,
   ...
   atributo_a_i: valor_a_i,
```

```
    Nombre_coleccion_B: <ObjectId(id_b)> // Referencia a documento b
}
```

Tabla 4.28 Documento b referenciado en documento a.

Y el documento *a* referenciado en el documento *b* con los siguientes campos: [Tabla 4.29]

```
// Colección B
// Documento b
{
   _id: <ObjectId(id_b)>,
   atributo_b_1: valor_b_1,
   atributo_b_2: valor_b_2,
   atributo_b_3: valor_b_3,
   ...
   atributo_b_j: valor_b_j
   Nombre_coleccion_A: <ObjectId(id_a)> // Referencia a documento a
}
```

Tabla 4.29 Documento b.

Ejemplo: Utilicemos el diagrama Entidad-Relación propuesto en [Figura 4.20] así como las instancias de *Estudiante* y *Tarjeta_Uva* propuestos en las tablas [Tabla 4.22] y [Tabla 4.23].

Referenciamos el documento *tarjeta_Uva_1* perteneciente a la colección *Tarjeta_Uva* dentro del documento *estudiante_1* de la colección *Estudiante* obteniendo: [Tabla 4.30].

```
// Colección Estudiante
// Documento estudiante_1
{
   _id: ObjectId ('65c640a4ffdb3aec9dbf0d73 '),
   Número_estudiante: 542531987,
   Nombre: Miguel Ángel,
   Curso: Cuarto,
   Centro: ETSII,
   Tarjeta_uva: '65c65065ffdb3aec9dbf0d87'    // Referencia a tarjeta_uva_1
```

Tabla 4.30 Documento tarjeta_uva_1 referenciado en estudiante_1.

De igual modo, referenciamos el documento *estudiante_1* perteneciente a la colección *Estudiante* dentro del documento *tarjeta_Uva*_1 de la colección *Tarjeta_Uva*: [Tabla 4.31].

```
// Colección tarjeta_uva_1
// Documento estudiante_1 referenciado en Documento tarjeta_uva_1
{
   _id: ObjectId('65c65065ffdb3aec9dbf0d87'),
   Número_tarjeta: VA1133545,
   Tipo_tarjeta: Identificativa
   Estudiante: '65c640a4ffdb3aec9dbf0d73' // Referencia a estudiante_1
}
```

Tabla 4.31 Documento estudainet_1 referenciado en tarjeta_uva_1.

4.3.2 Relaciones uno a muchos.

Sean *A* y *B* conjuntos de entidades relacionadas uno a muchos. Cada una de ellas con *i* y *j* atributos respectivamente [Figura 4.21].

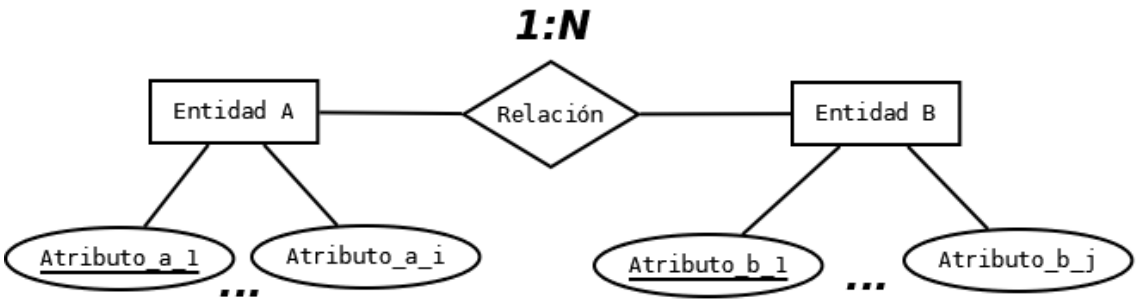

Figura 4.21 Relación uno a muchos ER.

En su paso a *MongoDB* el conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* el cual tendría *i+1* campos clave valor. El conjunto de entidades *B* sería la colección *B* y una instancia de *b* sería un documento *b* con *j+1* campos clave valor.

Para las transformaciones generales de este apartado suponemos que una instancia (documento *a*) de la colección *A* se relaciona con *n* instancias (documentos *b*) de la colección *B*.

4.3.2.1 Relaciones uno a muchos con documentos incrustados.

Una de las dos posibles formas de crear relaciones del tipo uno a muchos entre entidades es utilizando documentos incrustados dentro de otros.

Dado un documento *a* resultante de la transformación del conjunto de entidades *A* en la colección *A,* con *i +1* campos clave valor [Tabla 4.32].

```
// Colección A
// Documento a
{
   _id: <ObjectId_a>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   atributo_a_2: valor_a_3,
   ...
   atributo_a_i: valor_a_i
}
```

Tabla 4.32 Documento a.

Y dados *n* documentos *b* resultantes de la transformación del conjunto de entidades *B* en la colección *B, con* j+1 campos clave valor cada uno [Tabla 4.33].

```
// Colección B
// Documento b_1
{
   _id: <ObjectId_b1>,
   atributo_b1_1: valor_b1_1,
   atributo_b1_2: valor_b1_2,
   ...
   atributo_b1_j: valor_b1_j
}
// Documento b_2
{
   _id: <ObjectId_b2>,
   atributo_b2_1: valor_b2_1,
   atributo_b2_2: valor_b2_2,
   ...
```

```
   atributo_b2_j: valor_b2_j
}
...
// Documento b_n
{
   _id: <ObjectId_bn>,
   atributo_bn_1: valor_bn_1,
   atributo_bn_2: valor_bn_2,
   ...
   atributo_bn_j: valor_bn_j
}
```

Tabla 4.33 n Documentos b

Incrustando los documentos *b* en el documento *a,* al igual que hemos visto en el apartado anterior para las relaciones uno a uno, obtendríamos un único documento *a* perteneciente a la colección *A* con sus propios campos clave valor y un campo adicional con clave el nombre de la colección *B* y valor un array conteniendo el conjunto de todos los campos clave valor de todos los documentos *b* con los que está relacionado: [Tabla 4.34].

```
// Colección A
// Documento a
{
   _id: <ObjectId_a>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   ...
   atributo_a_i: valor_a_i
   nombre_colección_B: [ // Documentos b relacionados incrustados
   {
      _id: <ObjectId_b1>,
      atributo_b1_1: valor_b1_1,
      atributo_b1_2: valor_b1_2,
      ...
      atributo_b1_j: valor_b1_j
      },
   {
      _id: <ObjectId_b2>,
      atributo_b2_1: valor_b2_1,
```

```
        atributo_b2_2: valor_b2_2,
        ...
        atributo_b2_j: valor_b2_j
        },
        ...
    {
        _id: <ObjectId_bm>,
        atributo_bm_1: valor_bm_1,
        atributo_bm_2: valor_bm_2,
        ...
        atributo_bm_j: valor_bm_j
        }]
}
```

Tabla 4.34 Documentos b incrustados en a.

Ejemplo: Veamos un ejemplo práctico de la conversión de una relación uno a muchos entre entidades en un diagrama Entidad-Relación con documentos incrustados en *MongoDB*. En la siguiente figura mostramos un diagrama ER en el que el conjunto de entidades *Grados* está relacionado mediante *Pertenece* con el conjunto de entidades *Asignaturas*. La relación es del tipo 1: N. [Figura 4.22].

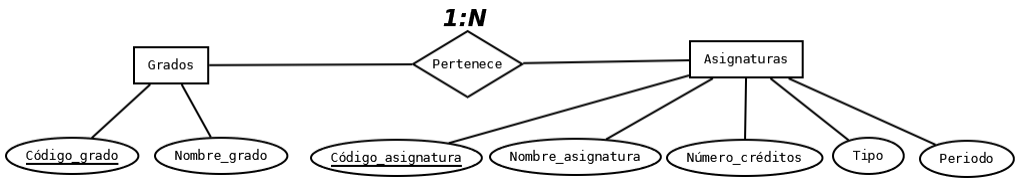

Figura 4.22 Ejemplo relación uno a muchos ER.

Tomemos una instancia *grado_1* del conjunto de entidades *Grados* y tres instancias *asignatura_1*, *asignatura_2* y *asignatura_3* del conjunto de entidades *Asignaturas* con los siguientes valores de atributos [Tabla 4.35] [Tabla 4.36].

Grados	Instancia 1
Código_grado	123456
Nombre_grado	Grado en ingeniería informática

Tabla 4.35 Instancia grado 1.

Asignatura	Instancia 1	Instancia 2	Instancia 3
Código_asignatura	46962	46901	46977
Nombre_asignatura	Administración de bases de datos	Fundamentos de matemáticas	Trabajo Fin de grado
Número_créditos	6	6	12
Tipo	Optativa	Obligatoria	Obligatoria
Periodo	1C	1C	2C

Tabla 4.36 Instancias asignatura.

La instancia de *grado_1* está relacionada con las tres instancias de *Asignatura*.

Tras realizar las transformaciones de entidades en colecciones vistas anteriormente obtendríamos 4 documentos: Uno de ellos perteneciente a la colección *Grados*: el documento *grado_1* [Tabla 4.37] y tres pertenecientes a la colección *Asignatura*, los documentos: *asignatura_1,* [Tabla 4.38] *asignatura_2* [Tabla 4.39] *y asignatura_3* [Tabla 4.40] conteniendo la siguiente información:

```
// Colección Grados
// Documento grado_1
{
   _id: ObjectId ('65c640a4ffdb3aec9dbf0d73 '),
   Número_grado: 123456,
   Nombre: Grado en ingeniería informática,
}
```

Tabla 4.37 Documento grado_1.

```
// Documento asignatura_1
{

   _id: ObjectId('65c64223ffdb3aec9dbf0d7e'),
   Código_asignatura: 46962,
   Nombre_asignatura: Administración de bases de datos,
   Número_créditos: 6,
   Tipo: Optativa,
   Periodo: 1C
}
```

Tabla 4.38 Documento asignatura_1.

```
// Documento asignatura_2
{
   _id: ObjectId('65c6425affdb3aec9dbf0d7f'),
```

```
    Código_asignatura: 46901,
    Nombre_asignatura: Fundamentos de matemáticas,
    Número_créditos: 6,
    Tipo: Obligatoria,
    Periodo: 1C
}
```

Tabla 4.39 Documento asignatura_2.

```
// Documento asignatura_3
{
    _id: ObjectId('65c642abffdb3aec9dbf0d80'),
    Código_asignatura: 46977,
    Nombre_asignatura: Trabajo Fin de grado,
    Número_créditos: 12,
    Tipo: Obligatoria,
    Periodo: 2C
}
```

Tabla 4.40 Documento asignatura_3.

Incrustando los documentos de la colección *Asignaturas* en los documentos de la colección *Grados* obtendríamos un documento perteneciente a la colección *Grados* con toda la información referente a dicho grado, así como la de las asignaturas con las que está relacionado [Tabla 4.41].

```
// Colección Grados
// Documento grado_1
{
    _id: ObjectId ('65c640a4ffdb3aec9dbf0d73 '),
    Número_grado: 123456,
    Nombre: Grado en ingeniería informática,
    Asignaturas: [ // Asignaturas relacionadas incrustadas
    {
        _id: ObjectId('65c64223ffdb3aec9dbf0d7e'), // Incrustamos asignatura 1
        Código_asignatura: 46962,
        Nombre_asignatura: Administración de bases de datos,
        Número_créditos: 6,
        Tipo: Optativa,
        Periodo: 1C
```

```
    },
    {
       _id: ObjectId('65c6425affdb3aec9dbf0d7f'), // Incrustamos asignatura 2
        Código_asignatura: 46901,
        Nombre_asignatura: Fundamentos de matemáticas,
        Número_créditos: 6,
        Tipo: Obligatoria,
        Periodo: 1C
    },
    {
        _id: ObjectId('65c642abffdb3aec9dbf0d80'), // Incrustamos asignatura 3
        Código_asignatura: 46977,
        Nombre_asignatura: Trabajo Fin de grado,
        Número_créditos: 12,
        Tipo: Obligatoria,
        Periodo: 2C
    }
   }
}
```

Tabla 4.41 Documentos asignatura incrustados en estudiante.

4.3.2.2 Relaciones uno a muchos con documentos referenciados.

Al igual que hemos hecho con las relaciones tipo uno a uno podemos referenciar el tipo uno a muchos.

Dados los documentos resultantes de la transformación de entidades en Colecciones en *MongoDB* descritos en las tablas [Tabla 4.32] y [Tabla 4.33].

Si un documento *a* se puede relacionar con varios documentos *b* y un documento *b* se puede relacionar con un documento *a* debemos realizar dos operaciones:

1. La primera es referenciar los documentos *b* relacionados con los documentos *a*. Para ello introducimos un campo clave valor adicional en los documentos *a* cuya clave será el nombre de la colección *B* y valor un array conteniendo los ObjectId de todos los documentos *b* relacionados.

   Los documentos a de la colección *A* tendrían la siguiente estructura [Tabla 4.42]:

```
// Colección A
// Documento a
```

```
{
   _id: < ObjectId_a>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   atributo_a_2: valor_a_3,
   ...
   atributo_a_i: valor_a_i
   Nombre_colección_B: [<ObjectId_b1>, …, <ObjectId_bn>] // Referencia a n documentos b
}
```

Tabla 4.42 Documentos b referenciados en a.

2. La segunda es referenciar el documento a relacionado con el documento b. Para ello introducimos un campo clave valor adicional en los documentos *b* cuya clave será el nombre de la colección *A* y valor el ObjectID del documento *a* relacionado.

   Así los documentos de la colección *B* tendrían la siguiente estructura [Tabla 4.43]:

```
// Colección B
// Documento b_1
{
   _id: <ObjectId_b1>,
   atributo_b1_1: valor_b1_1,
   atributo_b1_2: valor_b1_2,
   ...
   atributo_b1_j: valor_b1_j,
   Nombre_coleccion_A: < ObjectId_a> // Referencia documento a
}
// Documento b_2
{
   _id: <ObjectId_b2>,
   atributo_b2_1: valor_b2_1,
   atributo_b2_2: valor_b2_2,
   ...
   atributo_b2_j: valor_b2_j,
   Nombre_coleccion_A: < ObjectId_a> // Referencia documento a
}
...
```

```
// Documento b_n
{
   _id: <ObjectId_bn>,
   atributo_bn_1: valor_bn_1,
   atributo_bn_2: valor_bn_2,
   ...
   atributo_bn_j: valor_bn_j
   Nombre_coleccion_A: < ObjectId_a> // Referencia documento a
}
```

Tabla 4.43 Documento b incrustado en a.

Ejemplo: Veamos un ejemplo práctico de la conversión de una relación uno a muchos entre entidades en un diagrama Entidad-Relación referenciando los documentos resultantes de sus respectivas transformaciones en *MongoDB*.

Vamos a volver a utilizar el ejemplo presentado a la hora de crear relaciones uno a muchos mediante documentos incrustados para observar la diferencia de utilizar uno u otro método. [Figura 4.23].

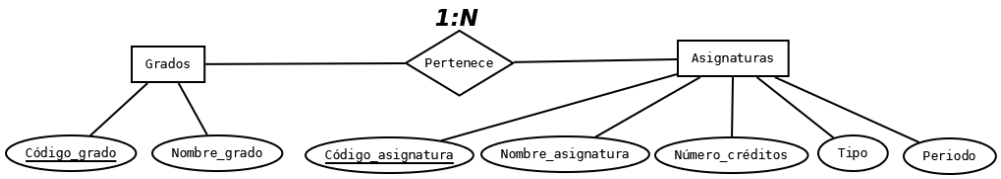

Figura 4.23 Ejemplo relación uno a muchos ER.

Las colecciones y documentos resultantes de la transformación de las entidades y sus respectivos atributos son los mismos que obtuvimos en el apartado anterior: *Grados* (*grado_1*) [Tabla 4.37] *asignatura_1,* [Tabla 4.38] *asignatura_2* [Tabla 4.39] *y asignatura_3* [Tabla 4.40]

Sin embargo, al referenciar los documentos de la colección *Grados* en el documento de la colección *Asignaturas* obtenemos 4 documentos: los 3 pertenecientes a *Asignaturas* con el campo adicional referenciando al grado con el que están relacionadas, mientras que el correspondiente a *Grados* tendrá el campo adicional con clave Asignaturas y valor un array conteniendo todos los ObjectId de las asignaturas.

```
// Colección Asignaturas
// Documento asignatura_1
```

```
{
   _id: ObjectId('65c64223ffdb3aec9dbf0d7e'),
   Código_asignatura: 46962,
   Nombre_asignatura: Administración de bases de datos,
   Número_créditos: 6,
   Tipo: Optativa,
   Periodo: 1C,
   Grados: '65c640a4ffdb3aec9dbf0d73' // Referencia a grado Ingeniería informática
}
```

Tabla 4.44 Instancia 1 asignatura.

```
// Colección Asignaturas
// Documento asignatura_2
{
   _id: ObjectId('65c6425affdb3aec9dbf0d7f'),
   Código_asignatura: 46901,
   Nombre_asignatura: Fundamentos de matemáticas,
   Número_créditos: 6,
   Tipo: Obligatoria,
   Periodo: 1C,
   Grados: '65c640a4ffdb3aec9dbf0d73'// Referencia a grado Ingeniería informática
}
```

Tabla 4.45 Instancia 2 asignatura.

```
// Colección Asignaturas
// Documento asignatura_3
{
   _id: ObjectId('65c642abffdb3aec9dbf0d80'),
   Código_asignatura: 46977,
   Nombre_asignatura: Trabajo Fin de grado,
   Número_créditos: 12,
   Tipo: Obligatoria,
   Periodo: 2C,
   Grados: '65c640a4ffdb3aec9dbf0d73'// Referencia a grado Ingeniería informática
}
```

Tabla 4.46 Instancia 3 asignatura.

```
// Colección Grados

// Documento grado_1

{

   _id: ObjectId ('65c640a4ffdb3aec9dbf0d73 '),

   Número_grado: 542531987,

   Nombre: Grado en ingeniería informática,

   Asignaturas: [ '65c640a4ffdb3aec9dbf0d73 ', '65c6425affdb3aec9dbf0d7f',
'65c642abffdb3aec9dbf0d80']

}
```

Tabla 4.47 Instancia Grados sin cambios

4.3.3 Relaciones muchos a muchos.

Cuando nos encontramos en un diagrama Entidad-Relación con relaciones muchos a muchos podemos utilizar los dos métodos vistos anteriormente para convertirlos en una base de datos con *MongoDB*: mediante documentos incrustados o mediante referencias entre documentos.

Explicaremos ambos procesos de transformación, sin embargo, para este tipo siempre es recomendable hacerlo mediante documentos referenciados. Como hemos explicado en el punto referente a las relaciones uno a muchos con documentos referenciados [4.3.2.2] nuestros documentos pueden crecer de tamaño a medida que introduzcamos información en nuestra futura base de datos, haciendo que la información pueda ser extremadamente redundante y el rendimiento de la misma se vea afectado, además de ser uno de los pocos patrones de diseño a evitar o "*anti patrones*" (término utilizado por los propios desarrolladores de *MongoDB*).[18]

En las relaciones tipo muchos a muchos, varias instancias de una entidad pueden estar relacionadas con varias instancias de otra entidad. Varias instancias del conjunto de entidades *A* pueden estar relacionadas con varias instancias del conjunto de entidades *B*, mientras que varias instancias del conjunto de entidades *B* pueden estar relacionadas con varias instancias del conjunto de entidades *A*. [Figura 4.24].

[18] Lauren Schaefer, Daniel Coupal "Schema design anti pattern massive arrays" [Recurso online] Disponible en: https://www.mongodb.com/developer/products/mongodb/schema-design-anti-pattern-massive-arrays/ Fecha última consulta: 01/06/2024

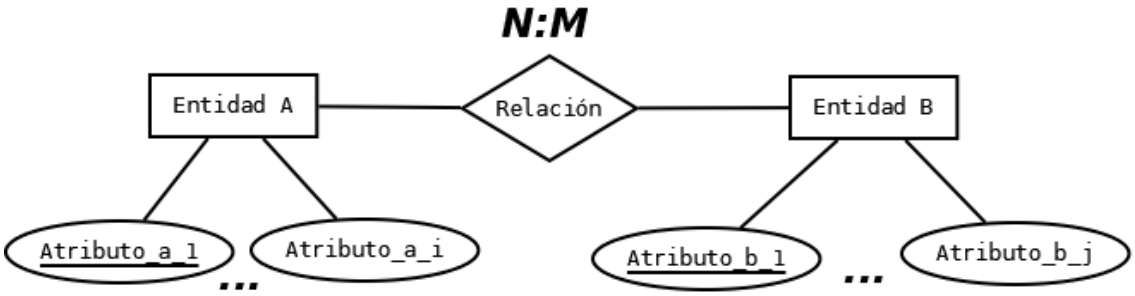

Figura 4.24 Relación muchos a muchos ER.

4.3.3.1 Relaciones muchos a muchos con documentos incrustados.

Sean *A* y *B* conjuntos de entidades relacionadas muchos a muchos.

Sean *a* y *b* documentos de las colecciones *A* y *B* respectivamente, resultantes de la transformación de dichas entidades en sus correspondientes colecciones. Si tenemos *n* documentos (con *i* campos cada uno) de la colección *A* relacionados con *m* documentos (con *j* campos cada uno) de la colección *B*. Y del mismo modo, los *m* documentos de la colección *B* relacionados con *n* documentos de la relación *A*.

Dados *n* documentos *a* pertenecientes a la colección *A*: [Tabla 4.48]

```
// Colección A
// Documento a1
{
   _id: <ObjectId_a1>,
   atributo_a1_1: valor_a1_1,
   atributo_a1_2: valor_a1_2,
   atributo_a1_3: valor_a1_3,
   ...
   atributo_b1_j: valor_a1_i
```

```
}
...
// Documento an
{
   _id: <ObjectId_an>,
   atributo_an_1: valor_an_1,
   atributo_an_2: valor_an_2,
   atributo_an_3: valor_an_3,
   ...
   atributo_an_j: valor_an_i,
}
```

Tabla 4.48 Documentos colección A.

Y dados *m* documentos *b* pertenecientes a la colección *B*: [Tabla 4.49]

```
// Colección B
// Documento b1
{
   _id: <ObjectId_b1>,
   atributo_b1_1: valor_b1_1,
   atributo_b1_2: valor_b1_2,
   atributo_b1_3: valor_b1_3,
   ...
   atributo_b1_j: valor_b1_j
}
...
// Documento bm
{
   _id: <ObjectId_bm>,
   atributo_bm_1: valor_bm_1,
   atributo_bm_2: valor_bm_2,
   atributo_bm_3: valor_bm_3,
   ...
   atributo_bm_j: valor_bm_j,
}
```

Tabla 4.49 Documentos colección B.

Incrustando los documentos *b* en los documentos *a* con los que están relacionados obtendríamos documentos *a* de la colección *A* de la siguiente forma: [Tabla 4.50]

```
// Colección A
// Documento a1
{
   _id: <ObjectId_a1>,
   atributo_a1_1: valor_a1_1,
   atributo_a1_2: valor_a1_2,
   atributo_a1_3: valor_a1_3,
   ...
   atributo_a1_j: valor_a1_i,
   Nombre_colección_B: [ // Documentos b relacionados incrustados
   {
      _id: <ObjectId_b1>,
      atributo_b1_1: valor_b1_1,
      atributo_b1_2: valor_b1_2,
      atributo_b1_3: valor_b1_3,
      ...
      atributo_b1_j: valor_b1_j
      },
      ...
   {
      _id: <ObjectId_bm>,
      atributo_bm_1: valor_bm_1,
      atributo_bm_2: valor_bm_2,
      atributo_bm_3: valor_bm_3,
      ...
      atributo_bm_j: valor_bm_j
      }
   ]
}
...
// Documento an
{
   _id: <ObjectId_an>,
   atributo_an_1: valor_an_1,
   atributo_an_2: valor_an_2,
```

```
    atributo_an_3: valor_an_3,
    ...
    atributo_an_i: valor_an_i,
    Nombre_colección_B: [ // Documentos b relacionados incrustados
    {
       _id: <ObjectId_b1>,
       atributo_b1_1: valor_b1_1,
       atributo_b1_2: valor_b1_2,
       atributo_b1_3: valor_b1_3,
       ...
       atributo_b1_j: valor_b1_j
       },
       ...
    {
       _id: <ObjectId_bm>,
       atributo_bm_1: valor_bm_1,
       atributo_bm_2: valor_bm_2,
       atributo_bm_3: valor_bm_3,
       ...
       atributo_bm_j: valor_bm_j
       }
    ]
}
```

Tabla 4.50 Documentos b incrustados en a

De igual modo si incrustásemos los documentos de la colección *A* en la colección *B*. [Tabla 4.51]

```
// Colección B
// Documento b1
{
    _id: <ObjectId_b1>,
    atributo_b1_1: valor_b1_1,
    atributo_b1_2: valor_b1_2,
    atributo_b1_3: valor_b1_3,
    ...
```

```
    atributo_b1_j: valor_b1_j,
    Nombre_colección_A: [ // Documentos a relacionados incrustados
    {
       _id: <ObjectId_a1>,
       atributo_a1_1: valor_a1_1,
       atributo_a1_2: valor_a1_2,
       atributo_a1_3: valor_a1_3,
       ...
       atributo_a1_i: valor_a1_i
       },
       ...
    {
       _id: <ObjectId_an>,
       atributo_an_1: valor_an_1,
       atributo_an_2: valor_an_2,
       atributo_an_3: valor_an_3,
       ...
       atributo_an_i: valor_an_j
       }
    ]
}
...
// Documento bm
{
    _id: <ObjectId_bm>,
    atributo_bm_1: valor_bm_1,
    atributo_bm_2: valor_bm_2,
    atributo_bm_3: valor_bm_3,
    ...
    atributo_bm_j: valor_bm_j,
    Nombre_colección_A: [ // Documentos a relacionados incrustados
    {
       _id: <ObjectId_b1>,
       atributo_a1_1: valor_a1_1,
       atributo_a1_2: valor_a1_2,
       atributo_a1_3: valor_a1_3,
       ...
```

```
        atributo_a1_i: valor_a1_i
        },
        ...
    {
        _id: <ObjectId_an>,
        atributo_an_1: valor_an_1,
        atributo_an_2: valor_an_2,
        atributo_an_3: valor_an_3,
        ...
        atributo_an_j: valor_an_i
        }
    ]
}
```

Tabla 4.51 Documento a incrustado en b.

Ejemplo: Veamos un ejemplo concreto de una relación muchos a muchos en un diagrama Entidad-Relación. Tenemos el conjunto de entidades *Profesor*, cuyos atributos son *Número_profesor*, *Nombre*, *Email* y *Teléfono* y el conjunto de entidades *Asignatura*, con atributos *Código_asignatura*, *Nombre_asignatura*, *Número_créditos*, *Tipo* y *Periodo* (Primer o Segundo cuatrimestre). Ambas entidades relacionadas por la relación *Imparte*.

Un *profesor* puede impartir una o varias *Asignaturas*, mientras que una asignatura puede ser impartida por un único *profesor* o por varios (bien porque la misma tenga parte teórica y práctica o bien porque está dividida en módulos impartidos por *profesores* especializados en ellos). [Figura 4.25].

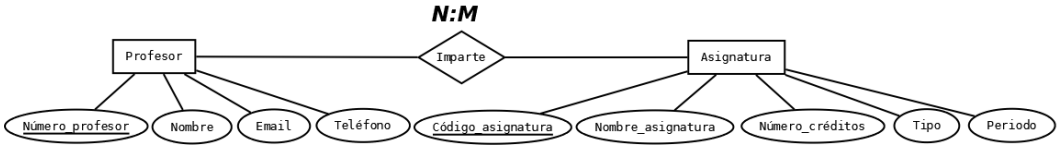

Figura 4.25 Ejemplo relación muchos a muchos ER.

Tomemos tres instancias *profesor_1*, *profesor_2* y *profesor_3* del conjunto de entidades *Profesores* con los siguientes valores de atributos: [Tabla 4.52]

Profesor	Instancia 1	Instancia 2	Instancia 3
Número_profesor	159261	159262	159263
Nombre	Fernando	Ángela	María Luz
Email	fernandouva@inf.uva.es	angelaouva@inf.uva.es	marialuzuva@inf.uva.es
Teléfono	983222222	983111111	983333333

Tabla 4.52 Instancias profesor.

Y tres instancias *asignatura_1*, *asignatura_2* y *asignatura_3* del conjunto de entidades *Asignatura* con los siguientes valores de atributos: [Tabla 4.53]

Asignatura	Instancia 1	Instancia 2	Instancia 3
Código_asignatura	46962	46901	46977
Nombre_asignatura	Administración de bases de datos	Fundamentos de matemáticas	Trabajo Fin de grado
Número_créditos	6	6	12
Tipo	Optativa	Obligatoria	Obligatoria
Periodo	1C	1C	2C

Tabla 4.53 Instancias asignatura.

Los documentos resultantes en *MongoDB* pertenecientes a la colección *Profesores* tras realizar las oportunas transformaciones serían los siguientes: [Tabla 4.54] [Tabla 4.55] [Tabla 4.56]

```
// Colección Profesor
// Documento profesor_1
{
   _id: ObjectId('65c67d05ffdb3aec9dbf0d91'),
   Número_profesor: 159261,
   Nombre: Fernando,
   Email: fernandouva@inf.uva.es,
   Teléfono: 983111111
}
```

Tabla 4.54 Documento profesor 1.

```
// Colección Profesor
// Documento profesor_2
{
   _id: ObjectId('65c67dc6ffdb3aec9dbf0d93'),
   Número_profesor: 159262,
   Nombre: Ángela,
   Email: angelauva@inf.uva.es,
   Teléfono: 983222222
}
```

Tabla 4.55 Documento profesor 2.

```
// Colección Profesor
// Documento profesor_3
{

   _id: ObjectId('65c67e35ffdb3aec9dbf0d94'),
   Número_profesor: 159263,
   Nombre: María Luz,
   Email: marialuzuva@inf.uva.es,
   Teléfono: 983333333
}
```

Tabla 4.56 Documento profesor 3.

Los pertenecientes a la colección *Asignaturas:* [Tabla 4.57] [Tabla 4.58] [Tabla 4.59].

```
// Colección Asignatura
// Documento asignatura_1
{
   _id: ObjectId('65c64223ffdb3aec9dbf0d7e'),
   Código_asignatura: 46962,
   Nombre_asignatura: Administración de bases de datos,
   Número_créditos: 6,
   Tipo: Optativa,
   Periodo: 1C
}
```

Tabla 4.57 Documento asignatura 1.

```
// Colección Asignatura
// Documento asignatura_2
{
   _id: ObjectId('65c6425affdb3aec9dbf0d7f'),
   Código_asignatura: 46901,
   Nombre_asignatura: Fundamentos de matemáticas,
   Número_créditos: 6,
   Tipo: Obligatoria,
   Periodo: 1C
}
```

Tabla 4.58 Documento asignatura 2.

```
// Colección Asignatura
// Documento asignatura_3
{
   _id: ObjectId('65c642abffdb3aec9dbf0d80'),
   Código_asignatura: 46977,
   Nombre_asignatura: Trabajo Fin de grado,
   Número_créditos: 12,
   Tipo: Obligatoria,
   Periodo: 2C
}
```

Tabla 4.59 Documento asignatura 3.

Las relaciones existentes entre estas instancias son las siguientes: la instancia 1 de *profesor* (Fernando) está relacionada mediante *imparte* con las instancias de *Asignatura*: Administración de bases de datos, Fundamentos de matemáticas Y Trabajo Fin de grado.

La instancia 2 de *profesor* (Ángela) está relacionada con Trabajo Fin de Grado y Fundamentos de matemáticas.

Por último, la *instancia* 3 de *profesor* (María Luz) está relacionada con Trabajo Fin de Grado y Administración de bases de datos.

Incrustando los documentos relacionados entre sí dentro del documento principal (en este caso *profesor*) obtendríamos:

Para el documento profesor 1. [Tabla 4.60]

```
// Colección Profesor
// Documento instancia 1 profesor
{
   _id: ObjectId('65c67d05ffdb3aec9dbf0d91'),
   Número_profesor: 159261,
   Nombre: Fernando,
   Email: fernandouva@inf.uva.es,
   Teléfono: 983111111,
   Asignaturas: [ // Asignaturas relacionadas incrustadas

      {// Asignatura Fundamentos de matemáticas incrustada
      _id: ObjectId('65c6425affdb3aec9dbf0d7f'),
      Código_asignatura: 46901,
      Nombre_asignatura: Fundamentos de matemáticas,
```

```
        Número_créditos: 6,
        Tipo: Obligatoria,
        Periodo: 1C

        },{// Asignatura administración de bases de datos incrustada

        _id: ObjectId('65c64223ffdb3aec9dbf0d7e'),
        Código_asignatura: 46962,
        Nombre_asignatura: Administración de bases de datos,
        Número_créditos: 6

        },{

        _id: ObjectId('65c642abffdb3aec9dbf0d80'),
        Código_asignatura: 46977,
        Nombre_asignatura: Trabajo Fin de grado,
        Número_créditos: 12,
        Tipo: Obligatoria,
        Periodo: 2C
        ]
    }
}
```

Tabla 4.60 Resultado incrustar instancia 1 profesor.

Para el documento profesor 2: [Tabla 4.61]

```
// Colección Profesores
// Documento instancia 2 profesor
{
    _id: ObjectId('65c67dc6ffdb3aec9dbf0d93'),
    Número_profesor: 159263,
    Nombre: Ángela,
    Email: angelauva@inf.uva.es,
    Teléfono: 983222222,
    Asignaturas: [ // Asignaturas relacionadas incrustadas
```

```
      {// Asignatura fundamentos de matemáticas incrustada
      _id: ObjectId('65c6425affdb3aec9dbf0d7f'),
      Código_asignatura: 46901,
      Nombre_asignatura: Fundamentos de matemáticas,
      Número_créditos: 6,
      Tipo: Obligatoria,
      Periodo: 1C

      },{ // Asignatura TFG incrustada
      _id: ObjectId('65c642abffdb3aec9dbf0d80'),
      Código_asignatura: 46977,
      Nombre_asignatura: Trabajo Fin de grado,
      Número_créditos: 12,
      Tipo: Obligatoria,
      Periodo: 2C
      ]
   }
}
```

Tabla 4.61 Resultado incrustar instancia 2 profesor.

Y, por último, para el documento profesor 3: [Tabla 4.62]

```
// Colección Profesores
// Documento instancia 3 profesor
{
   _id: ObjectId('65c67e35ffdb3aec9dbf0d94'),
   Número_profesor: 159263,
   Nombre: María Luz,
   Email: marialuzuva@inf.uva.es,
   Teléfono: 983333333,
   Asignaturas: [ // Asignaturas relacionadas incrustadas

      {// Asignatura Administración de bases de datos incrustada
       _id: ObjectId('65c64223ffdb3aec9dbf0d7e'),
      Código_asignatura: 46962,
      Nombre_asignatura: Administración de bases de datos,
```

```
        Número_créditos: 6


        },{ // Asignatura TFG incrustada
        _id: ObjectId('65c642abffdb3aec9dbf0d80'),
        Código_asignatura: 46977,
        Nombre_asignatura: Trabajo Fin de grado,
        Número_créditos: 12,
        Tipo: Obligatoria,
        Periodo: 2C
        ]
    }
}
```

Tabla 4.62 Resultado incrustar instancia 3 profesor.

De igual forma podríamos incrustar documentos de la colección *Profesor* en documentos de la colección *Asignatura* si están relacionados entre sí. Por ejemplo, para la asignatura TFG los tres *Profesores* imparten dicha *Asignatura*: [Tabla 4.63]

```
// Colección Asignaturas
// Documento asignatura_3
{
    _id: ObjectId('65c642abffdb3aec9dbf0d80'),
    Código_asignatura: 46977,
    Nombre_asignatura: Trabajo Fin de grado,
    Número_créditos: 12,
    Profesores: [ // Profesores relacionados incrustados
        {
        _id: ObjectId('65c67d05ffdb3aec9dbf0d91'),
        Número_profesor: 159261,
        Nombre: Fernando,
        Email: fernandouva@inf.uva.es,
        Teléfono: 983111111

        },{
        _id: ObjectId('65c67dc6ffdb3aec9dbf0d93'),
        Nombre: Ángela,
        Email: angelauva@inf.uva.es,
```

```
        Teléfono: 983222222


        },{
        _id: ObjectId('65c67e35ffdb3aec9dbf0d94'),
        Número_profesor: 159263,
        Nombre: María Luz,
        Email: marialuzuva@inf.uva.es,
        Teléfono: 983333333
```

Tabla 4.63 Resultado incrustar instancia 3 asignatura.

El resto de relaciones y documentos se crearían de la misma forma que hemos descrito tomando como documento principal *Profesor*, solo que esta vez insertando los ObjectID correspondientes a las relaciones de los documentos de la colección *Asignatura*.

4.3.3.1 Relaciones muchos a muchos con documentos referenciados.

Al igual que hemos hecho con las relaciones tipo uno a uno y uno a muchos, podemos referenciar las relaciones existentes en el tipo muchos a muchos.

Dados los documentos resultantes de la transformación de entidades en Colecciones en *MongoDB* descritos en las tablas [Tabla 4.48] y [Tabla 4.49].

Si un documento *a* se puede relacionar con varios documentos *b* y varios documentos *b* se pueden relacionar con varios documentos *a* debemos realizar dos operaciones:

1. La primera es referenciar los documentos *a* relacionados con los documentos *b*. Para ello introducimos un campo adicional en los documentos *b* cuya clave será el nombre de la colección *A* y valor un array conteniendo los ObjectId de todos los documentos *a* relacionados.

   Obteniendo la siguiente estructura para los documentos de la colección *B*: [Tabla 4.64]

```
// Colección B
// Documentos b
{
   _id: <ObjectId_b1>,
   atributo_b1_1: valor_b1_1,
   atributo_b1_2: valor_b1_2,
   atributo_b1_2: valor_b1_3,
```

```
    ...
    atributo_b1_i: valor_b1_j
    Nombre_coleccion_a: [ObjectId_a1,...,ObjectId_an] //Referencias a documentos a
}
...
// Documento bm
{
    _id: <ObjectId_bm>,
    atributo_bm_1: valor_bm_1,
    atributo_bm_2: valor_bm_2,
    atributo_bm_3: valor_bm_3,
    ...
    atributo_bm_j: valor_bm_j,
    Nombre_coleccion_A: [ObjectId_a1,...,ObjectId_an] //Referencias a documentos a
}
```

Tabla 4.64 Resultado referenciar n a m, documentos b.

2. La segunda es referenciar los documentos *b* relacionados con los documentos *a*. Para ello introducimos un campo adicional en los documentos *a* cuya clave será el nombre de la colección *B* y valor un array conteniendo los ObjectId de todos los documentos *b* relacionados.

   Obteniendo así los documentos de la colección A con la siguiente estructura: [Tabla 4.65]

```
// Colección A
// Documentos a
{
   _id: <ObjectId_a1>,
   atributo_a1_1: valor_a1_1,
   atributo_a1_2: valor_a1_2,
   atributo_a1_2: valor_a1_3,
   ...
   atributo_a1_i: valor_a1_i
   Nombre_coleccion_B: [ObjectId_b1,..., ObjectId_bm] //Referencias a documentos b
```

```
}
...
// Documento an
{
   _id: <ObjectId_an>,
   atributo_an_1: valor_an_1,
   atributo_an_2: valor_an_2,
   atributo_an_3: valor_an_3,
   ...
   atributo_an_j: valor_an_j,
   nombre_colección_B: [ObjectId_b1, ..., ObjectId_bm] //Referencias a documentos b
}
```

Tabla 4.65 Resultado referenciar n a m, documentos a.

Ejemplo: Veamos un ejemplo práctico de la conversión de una relación muchos a muchos entre entidades en un diagrama Entidad-Relación referenciando documentos en *MongoDB*.

Vamos a volver a utilizar el ejemplo presentado a la hora de crear relaciones muchos a muchos mediante documentos incrustados para observar la diferencia de utilizar uno u otro método. [Figura 4.26].

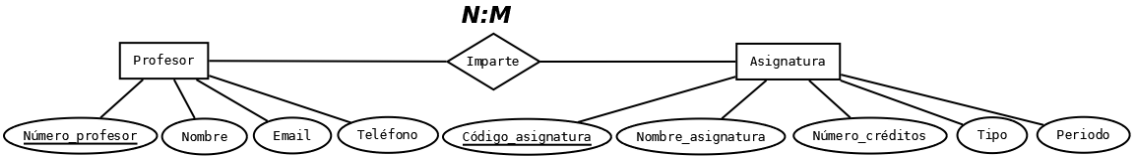

Figura 4.26 Relación muchos a muchos ER.

Igualmente utilizaremos las instancias de *Profesor* y *Asignatura* utilizadas en la anterior transformación mediante documentos incrustados: [Tabla 4.52] y [Tabla 4.53]

La instancia 1 de *Profesor* (Fernando) está relacionada mediante *imparte* con las instancias de *Asignatura* Fundamentos de matemáticas, Administración de bases de Datos y Trabajo Fin de Grado. Añadimos un campo extra en el documento de dicha instancia con clave el nombre de la colección con la que está relacionada (*Asignatura*) y tres valores, los ObjectId de dichas asignaturas.

Añadimos al documento profesor_1 un campo con clave el nombre Asignaturas y valor los tres ObjectID de las asignaturas con las que está relacionado. [Tabla 4.66]

```
// Colección Profesor
// Documento profesor_1
{
   _id: ObjectId('65c67d05ffdb3aec9dbf0d91'),
   Número_profesor: 159261,
   Nombre: Fernando,
   Email: fernandouva@inf.uva.es,
   Teléfono: 983111111
   Asignatura: ['65c6425affdb3aec9dbf0d7f', '65c64223ffdb3aec9dbf0d7e',
'65c642abffdb3aec9dbf0d80']
}
```

Tabla 4.66 Resultado referenciar instancia 1 de profesor

La instancia 2 de *Profesor* (Ángela) está relacionada mediante imparte con las *Asignaturas* Fundamentos de matemáticas y Trabajo Fin de Grado. Realizamos el mismo proceso, esta vez añadiendo los ObjectId correspondientes. [Tabla 4.67]

```
// Colección Profesor
// Documento profesor_2
{
   _id: ObjectId('65c67dc6ffdb3aec9dbf0d93'),
   Número_profesor: 159262,
   Nombre: Ángela,
   Email: angelaouva@inf.uva.es,
   Teléfono: 983222222
   Asignaturas: ['65c6425affdb3aec9dbf0d7f', '65c642abffdb3aec9dbf0d80']
}
```

Tabla 4.67 Resultado referenciar instancia 2 de profesor

Por último, la instancia 3 de *Profesor* (María Luz) está relacionada mediante imparte con Trabajo Fin de Grado y Administración de bases de datos. Referenciamos en el documento correspondiente los ObjectID de las dos asignaturas. [Tabla 4.68]

```
// Colección Profesor
// Documento profesor_3
{
   _id: ObjectId('65c67e35ffdb3aec9dbf0d94'),
```

```
    Número_profesor: 159263,
    Nombre: María Luz,
    Email: marialuzuva@inf.uva.es
    Teléfono: 983333333
    Asignatura: ['65c6425affdb3aec9dbf0d7f', '65c64223ffdb3aec9dbf0d7e']
}
```

Tabla 4.68 Resultado referenciar instancia 3 de profesor

De esta forma quedarían referenciadas las asignaturas con las que están relacionados los profesores dentro de estos últimos.

Ahora tendríamos que insertar dentro de los documentos de la colección *Asignaturas* las referencias a los profesores con los que están relacionados: [Tabla 4.69] [Tabla 4.70] [Tabla 4.71]

```
// Colección Asignatura
// Documento asignatura_1
{
    _id: ObjectId('65c64223ffdb3aec9dbf0d7e'),
    Código_asignatura: 46962,
    Nombre_asignatura: Administración de bases de datos,
    Número_créditos: 6,
    Tipo: Optativa,
    Periodo: 1C
    Profesor: ['65c67d05ffdb3aec9dbf0d91', '65c67e35ffdb3aec9dbf0d94']
}
```

Tabla 4.69 Resultado referenciar instancia 1 de asignatura.

```
// Colección Asignatura
// Documento asignatura_2
{
    _id: ObjectId('65c6425affdb3aec9dbf0d7f'),
    Código_asignatura: 46901,
    Nombre_asignatura: Fundamentos de matemáticas,
    Número_créditos: 6,
    Tipo: Obligatoria,
    Periodo: 1C
    Profesor: ['65c67d05ffdb3aec9dbf0d91', '65c67dc6ffdb3aec9dbf0d93']
}
```

Tabla 4.70 Resultado referenciar instancia 2 de asignatura.

```
// Colección Asignatura
// Documento asignatura_3
{
   _id: ObjectId('65c642abffdb3aec9dbf0d80'),
   Código_asignatura: 46977,
   Nombre_asignatura: Trabajo Fin de grado,
   Número_créditos: 12,
   Tipo: Obligatoria,
   Periodo: 2C
   Profesor: ['65c67d05ffdb3aec9dbf0d91','65c67dc6ffdb3aec9dbf0d93',
'65c67e35ffdb3aec9dbf0d94']
}
```

Tabla 4.71 Resultado referenciar instancia 3 de asignatura.

Una vez expuestos ambos ejemplos de transformación de relaciones N:M a sus correspondientes colecciones y documentos en *MongoDB* y aun siendo únicamente 3 documentos por colección, podemos observar que es mucho más beneficioso, en términos del crecimiento de los arrays contenedores de la información, referenciar los documentos antes que incrustar la información de unos en otros.

4.4 Transformación jerarquía ISA (generalización/especialización).

Dado un conjunto de entidades de mayor orden (llamémosle *A*) con *i* atributos, y un número *x* de entidades de menor orden (el conjunto de entidades *B* con *j* atributos… el conjunto de entidades *X* con *k* atributos) las cuales están relacionadas mediante jerarquía ISA con la entidad *A* de mayor orden. [Figura 4.27]

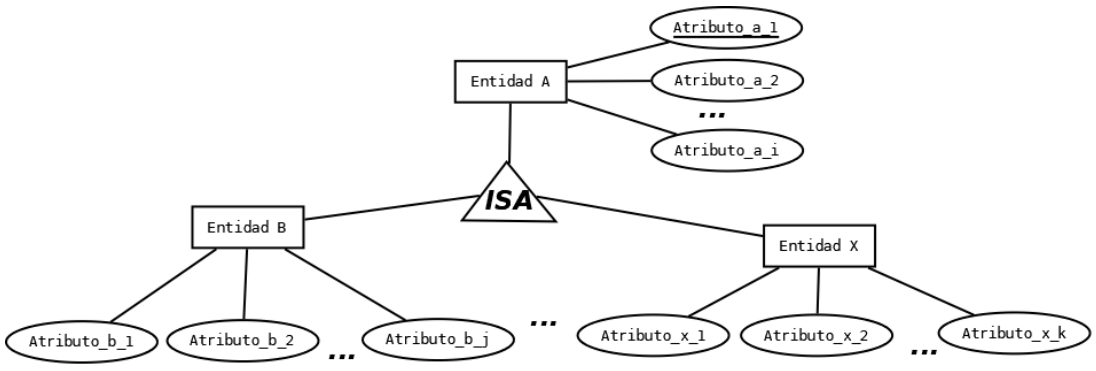

Figura 4.27 jerarquía ISA ER.

Sean *A*, *B*...*X* conjuntos de entidades relacionadas muchos a muchos.

Sean *a*, *b, ... x* documentos de sus respectivas colecciones, resultantes de la transformación de dichas entidades en sus correspondientes colecciones. Cada uno de los documentos con un número determinado de campos clave valor (*i* campos para los documentos *a*, *j* campos para los documentos *b*,…,*k* campos para los documentos *x*).

Podemos abordar la transformación de dicha jerarquía ISA en sus respectivas colecciones, documentos y relaciones en *MongoDB* de tres maneras distintas con diferentes resultados:

1. *Integrando la jerarquía de generalización en una sola colección*. Los documentos pertenecientes a dicha colección tendrán como campos clave valor todos los atributos de cada uno de los conjuntos de entidades participantes de la relación ISA.

   Se mantendrá una única colección con nombre la de mayor orden cuyos documentos tendrán tantos campos clave valor como atributos tuviesen todas las entidades participantes (*i* + *j*+ … +*k*) más uno que será el objectId del documento *a*. [Tabla 4.72]

   De esta forma perderíamos el concepto de generalización/especialización.

```
// Colección A
// Documento a
{
   _id: <ObjectId_a>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   ...
   atributo_a_i: valor_a_i,
   atributo_b_0: valor_b_1,
   atributo_b_1: valor_b_2,
    ...
   atributo_b_j: valor_b_j,
   ...
   atributo_x_1: valor_x_1,
   atributo_x_2: valor_x_2,
   ...
   atributo_x_k: valor_x_k
}
```

Tabla 4.72 Resultado primera transformación ISA.

2. *Eliminando la entidad de orden superior y transformando las entidades de inferior orden en colecciones*, de esta forma se propagarían los atributos de la de orden superior en las colecciones de orden inferior.

   Se crearían tantas colecciones como número de entidades de menor orden existiesen en la jerarquía ISA. Cada una de ellas con nombre el del conjunto de entidades de menor orden al que pertenecen y con documentos con tantos pares clave valor como atributos tuviese dicha entidad, los de la entidad de mayor orden y uno adicional con el ObjectID del propio documento. [Tabla 4.73]

   Mediante esta operación perderíamos igualmente el concepto de generalización/especialización.

```
// Colección B
// Documento b
{
   _id: <ObjectId_b>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   ...
   atributo_a_i: valor_a_i,
   atributo_b_1: valor_b_1,
   atributo_b_2: valor_b_2,
   ...
   atributo_b_j: valor_b_j
}
...
// Colección X
// Documento x
{
   _id: <ObjectId_x>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_2,
   ...
   atributo_a_i: valor_a_i,
   atributo_x_1: valor_x_1,
   atributo_x_2: valor_x_2,
   ...
   atributo_x_k: valor_x_k
```

```
}
```

Tabla 4.73 Resultado segunda transformación ISA.

3. La tercera opción: se *pueden transformar todos los conjuntos de entidades participantes en la relación ISA, cada uno de ellos en una colección*. La transformación consistiría en crear una colección por cada conjunto de entidades participantes en la jerarquía ISA con nombre el del propio conjunto de entidades. Cada uno de los documentos pertenecientes a sus respectivas colecciones tendría tantos campos clave valor como atributos tuviesen las entidades a las que pertenecen y aquellas de menor orden un campo clave valor adicional con clave el nombre de la entidad de mayor orden y valor una referencia al *_id* de la entidad de mayor orden. [Tabla 4.74].

   Es el único método en el que no se pierde el concepto de generalización/especialización, aunque hay riesgo de redundancia de datos.

```
// Colección A
// Documento a
{
   _id: <ObjectId_a>,
   atributo_a_1: valor_a_1,
   atributo_a_2: valor_a_1,
   atributo_a_2: valor_a_2,
   ...
   atributo_a_i: valor_a_i
}
// Colección B
// Documento b
{
   _id: <ObjectId_b>,
   atributo_b_1: valor_b_1,
   atributo_b_2: valor_b_2,
   atributo_b_3: valor_b_3,
   ...
   atributo_b_j: valor_b_j
   Nombre_coleccion_a: < ObjectId_a> // Referencia a mayor orden
}
...
// Colección X
// Documento x
{
```

```
    _id: <ObjectId_x>,
    atributo_x_1: valor_x_1,
    atributo_x_2: valor_x_2,
    atributo_x_2: valor_x_3,
    ...
    atributo_x_k: valor_x_k
    Nombre_coleccion_a: < ObjectId_a> // Referencia a mayor orden
}
```

Tabla 4.74 Resultado tercera transformación ISA.

Esta última es la mejor aproximación a la hora de transformar jerarquías ISA en *MongoDB*. Aunque también podemos encontrarnos con repetición de datos, dadas las características de esta base de datos no supone un problema como si ocurre con el modelo relacional.

Ejemplo: Veamos el siguiente ejemplo de un diagrama Entidad-Relación con jerarquía ISA: [Figura 4.28]

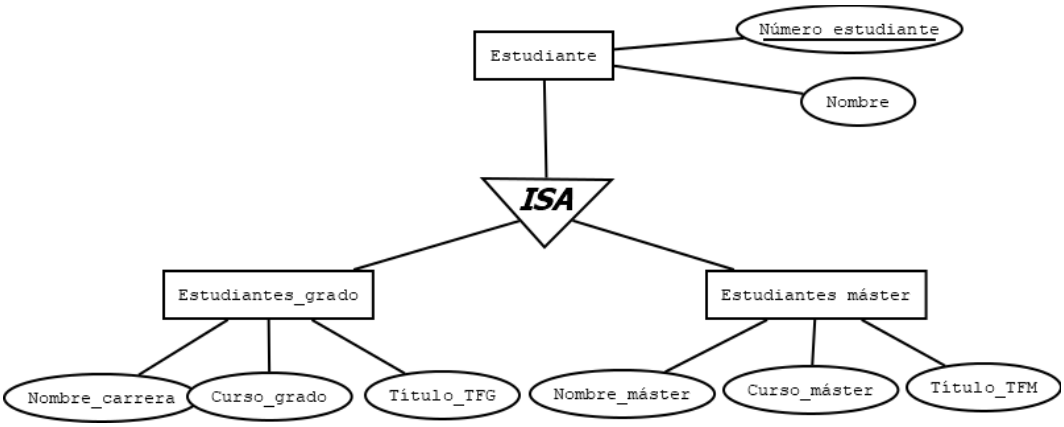

Figura 4.28 Ejemplo jerarquía ISA ER.

Tenemos las siguientes instancias para un alumno de Grado: [Tabla 4.75]

Estudiante grado	Instancia
Número_estudiante	123456
Nombre	Fernando
Carrera	Grado en ingeniería informática
Curso_grado	Cuarto
Título_TFG	Método de transformación de un diagrama ER

Tabla 4.75 Instancia estudiante de grado.

Y otro de máster: [Tabla 4.76]

Estudiante máster	Instancia
Número_estudiante	123457
Nombre	Henar
Carrera	Grado en ingeniería informática
Curso_máster	Segundo
Título_TFM	Método de transformación de una base de datos con MongoDB

Tabla 4.76 Instancia estudiante de máster.

Solución 1: *Integrando la jerarquía de generalización en una sola colección*. Crearíamos una única colección con nombre *Estudiante.* Cada uno de los documentos pertenecientes a dicha colección tendría los campos clave valor de *estudiante*, los de *estudiante_grado* y los de *estudiante_master*.

De esta forma el documento de la colección Estudiante con la información relevante a la instancia del estudiante de grado tendría la siguiente estructura: [Tabla 4.77]

```
// Colección Estudiante
// Documento estudiante 1
{
   _id: ObjectId('65d1f88ab55dcd9eddd12e70'),
   Numero_estudiante: 123456,
   Nombre: Rubén,
   Carrera: Grado en ingeniería informática,
   Curso_grado: Cuarto,
   Título_TFG: Método de transformación de un diagrama ER,
   Curso_máster: null, // Se podría omitir el campo
   Título_TFM: null // Se podría omitir el campo
}
```

Tabla 4.77 Documento estudiante de grado.

Y el documento de la misma colección con la información del estudiante de máster: [Tabla 4.78]

```
// Colección Estudiante
// Documento estudiante 2
{
```

```
    _id: ObjectId('65d1f8e5b55dcd9eddd12e71'),
    Numero_estudiante: 123457,
    Nombre: Henar,
    Carrera: Grado en ingeniería informática,
    Curso_grado: null, // Se podría omitir el campo
    Título_TFG: null, // Se podría omitir el campo
    Curso_máster: Segundo,
    Título_TFM: Método de transformación de una base de datos con MongoDB
}
```

Tabla 4.78 Documento estudiante de máster.

Podemos observar que dependiendo de la instancia (si pertenece a una especialización o a otra) los valores aparecen como *null*.

Solución 2: Eliminando la entidad de orden superior y transformando las entidades de inferior orden en colecciones. Para las mismas instancias del ejemplo anterior [Tabla 4.73] y [Tabla 4.74] obtendríamos, en vez de una, dos coleccione distintas con nombre *Estudiante_grado* [Tabla 4.79]y *Estudiante_máster* [Tabla 4.80].

```
// Colección Estudiante_grado
// Documento estudiante_grado 1
{

    _id: ObjectId('65d1f96db55dcd9eddd12e73'),
    Numero_estudiante: 123456,
    Nombre: Rubén,
    Carrera: Grado en ingeniería informática,
    Curso_grado: Cuarto,
    Título_TFG: Método de transformación de un diagrama ER
}
```

Tabla 4.79 Documento estudiante_grado.

```
// Colección Estudiante_master
// Documento estudiante 2
{

    _id: ObjectId('65d1f9d5b55dcd9eddd12e75'),
    Numero_estudiante: 123457,
```

```
    Nombre: Henar,
    Carrera: Grado en ingeniería informática,
    Curso_máster: Segundo,
    Título_TFM: Método de transformación de una base de datos con MongoDB
}
```

Tabla 4.80 Documento estudiante_máster.

Solución 3: transformar todos los conjuntos de entidades participantes en la relación ISA, cada uno de ellos en una colección. Continuamos utilizando las instancias [Tabla 4.75] y [Tabla 4.76] Tendríamos tres colecciones con nombre *Estudiante*, *Estudiante_grado* y *Estudiante_máster*. Cada uno de los documentos pertenecientes a su colección tendría como campos los de los atributos propios de Estudiante [Tabla 4.81] y [Tabla 4.82]. En *Estudiante_grado* [Tabla 4.83] y *Estudiante_máster* [Tabla 4.84] añadimos un campo adicional referenciando a *Estudiante*.

```
// Colección Estudiante
// Documento estudiante 1
{
    _id: ObjectId('65d1fa41b55dcd9eddd12e77'),
    Numero_estudiante: 123456,
    Nombre: Rubén,
    Carrera: Grado en ingeniería informática
}
```

Tabla 4.81 Documento estudante_1.

```
// Colección Estudiante
// Documento estudiante 2
{
    _id: ObjectId('65d1fa86b55dcd9eddd12e78'),
    Numero_estudiante: 123457,
    Nombre: Henar,
    Carrera: Grado en ingeniería informática
}
```

Tabla 4.82 Documento estudiante_2.

```
// Colección Estudiante_grado
// Documento estudiante_grado
{
    _id: ObjectId('65d1fbb6b55dcd9eddd12e7b'),
```

```
    Curso_grado: Cuarto,
    Título_TFG: Método de transformación de un diagrama ER,
    Estudiante: '65d1fa41b55dcd9eddd12e77'// Referencia
}
```

Tabla 4.83 Documento estudiante_grado.

```
// Colección Estudiante_master
// Documento estudiante_máster
{
    _id: ObjectId('65d1f9d5b55dcd9eddd12e75'),
    Estudiante: '65d1fa86b55dcd9eddd12e78'
    Curso_máster: Segundo,
    Título_TFM: Método de transformación de una base de datos con MongoDB,
    Estudiante: '65d1f9d5b55dcd9eddd12e75' //Referencia
}
```

Tabla 4.84 Documento estudiante_máster.

4.5 Entidades asociativas.

Las **entidades asociativas** relacionan las instancias de uno o más conjuntos de entidades y contiene atributos los cuales pertenecen a dicha relación entre esas instancias de entidades. [Figura 4.29]

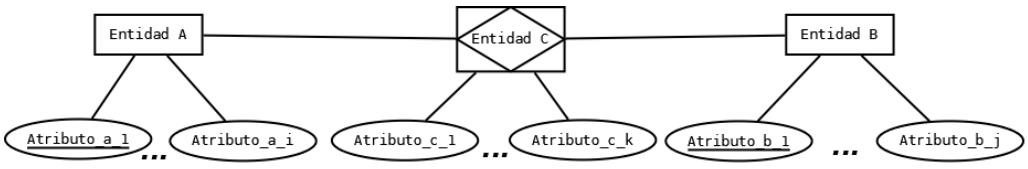

Figura 4.29 Entidad asociativa modelo ER.

Sean *A* y *B* conjuntos de entidades (cada una de ellas con *i* y *j* atributos) relacionadas por un conjunto de entidades asociativas *C* (con *k* atributos).

En su paso a *MongoDB* el conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* el cual tendría *i+1* campos clave valor. [Tabla 4.85]

```
// Colección A
// Documento a
{
   _id: <ObjectId_a>,
   Atributo_a_1: valor_a_1,
   Atributo_a_2: valor_a_2,
   ...
   Atributo_a_i: valor_a_i
}
```

Tabla 4.85 Documento colección A.

El conjunto de entidades *B* sería la colección *B* y una instancia de *b* sería un documento *b* con *j+1* campos clave valor. [Tabla 4.86]

```
// Colección B
// Documento b
{
   _id: <ObjectId_b>,
   Atributo_b_1: valor_b_1,
   Atributo_b_2: valor_b_2,
   ...
   Atributo_b_j: valor_b_j
}
```

Tabla 4.86 Documento colección B.

El conjunto de entidades asociativas *C* sería una colección *C* y una instancia de *c* sería un documento *c* con *k+3* campos clave valor: uno de ellos correspondiente al propio *ObjectID* del documento, otro con clave el nombre de la colección *A* y valor el *ObjectID* del documento *a* con el que está relacionado y por último uno con clave el nombre de la colección *B* y valor el *ObjectID* del documento *b* con el que está relacionado. [Tabla 4.87]

```
// Colección C
// Documento c
{
   _id: <ObjectId_c>,
   Atributo_c_1: valor_c_1,
   Atributo_c_2: valor_c_2,
```

```
    ...
    Atributo_c_k: valor_c_k,
    Nombre_coleccion_A: ObjectId_a, // Referencia a documento a
    Nombre_colección_B: ObjectId_b // Referencia a documento b
}
```

Tabla 4.87 Documento colección C.

Ejemplo: Veamos el siguiente ejemplo de un diagrama Entidad-Relación con una entidad asociativa: [Figura 4.30]

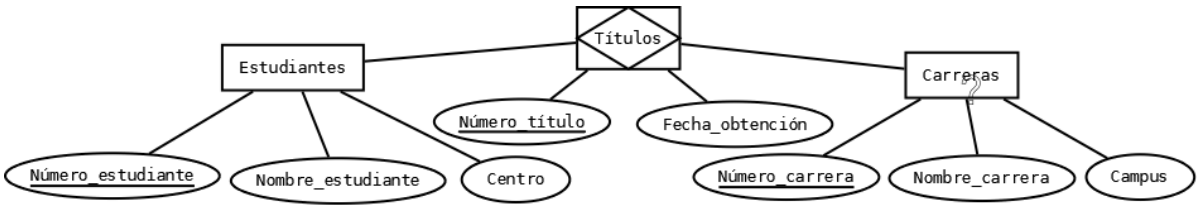

Figura 4.30 Ejemplo entidad asociativa modelo ER.

Tomemos tres instancias: *estudiante_1*, *estudiante _2* y *estudiante _3* del conjunto de entidades *Estudiantes* con los siguientes valores de atributos: [Tabla 4.88]

Estudiante	Instancia 1	Instancia 2	Instancia 3
Número_estudiante	321261	321262	321263
Nombre_estudiante	Rubén	Henar	Miguel
Centro	ETSII	ESS	ESA

Tabla 4.88 Instancias estudiante.

Tomemos tres instancias *carrera_1*, *carrera_2* y *carrera_3* del conjunto de entidades *Carreras* con los siguientes valores de atributos: [Tabla 4.89]

Carreras	Instancia 1	Instancia 2	Instancia 3
Número_carrera	159687	159345	159276
Nombre_carrera	Ingeniería informática	Psicología	Periodismo
Campus	Valladolid	Palencia	Soria

Tabla 4.89 Instancias carreras.

Tenemos la siguiente información a introducir en nuestra base de datos:

- El estudiante *Rubén* obtuvo el título en *Periodismo* el año *2000*, así como el de *Ingeniería informática* en el *2023*.

- La estudiante *Henar* obtuvo el título en *Psicología* en el año *2024*.
- Por último, el estudiante *Miguel* obtuvo el título en *Periodismo* en el año *2000*.

Tomemos tres instancias *títulos_1*, *títulos_2* y *títulos_3* del conjunto de entidades asociativa *Títulos*, asociadas a sus respectivas asociaciones con *Estudiantes* y *Carreras* con los siguientes valores de atributos: [Tabla 4.90]

Títulos	Instancia 1	Instancia 2	Instancia 3	Instancia 4
Número_título	21	22	23	24
Fecha_obtencíon	2024	2023	2022	2000

Tabla 4.90 Instancias Títulos.

Los documentos resultantes en *MongoDB* pertenecientes a la colección *Profesores* tras realizar las oportunas transformaciones serían los siguientes: [Tabla 4.91] [Tabla 4.92] [Tabla 4.92]

```
// Documento estudiante_1
{
   _id: ObjectId ('65d9be24c9215662e68ca821'),
   Número_estudiante: 321261,
   Nombre: Rubén,
   Centro: ETSII
}
```

Tabla 4.91 Instancia 1 estudiante.

```
// Documento estudiante_2
{
   _id: ObjectId (65d9be93c9215662e68ca822),
   Número_estudiante: 321262,
   Nombre: Henar,
   Centro: ESS
}
```

Tabla 4.92 Instancia 2 estudiante.

```
// Documento estudiante_3
{
   _id: ObjectId (65d9bf54c9215662e68ca823),
   Número_estudiante: 321263,
```

```
    Nombre: Miguel,
    Centro: ESA
}
```

Tabla 4.93 Instancia 3 estudiante.

Mientras que los documentos resultantes en *MongoDB* pertenecientes a la colección *Carreras* tras realizar las oportunas transformaciones serían los siguientes: [Tabla 4.94] [Tabla 4.95] [Tabla 4.96]

```
// Documento carrera_1
{
    _id: ObjectId (65d9cb50c9215662e68ca825),
    Número_carrera: 159687,
    Nombre_carrera: Ingeniería informática,
    Campus: Valladolid
}
```

Tabla 4.94 Instancia 1 carrera.

```
// Documento carrera_2
{
    _id: ObjectId (65d9cbadc9215662e68ca826),
    Número_carrera: 159345,
    Nombre_carrera: Psicología,
    Campus: Palencia
}
```

Tabla 4.95 Instancia 2 carrera.

```
// Documento carrera_3
{
    _id: ObjectId (65d9cbd5c9215662e68ca827),
    Número_carrera: 159276,
    Nombre_carrera: Periodismo,
    Campus: Soria
}
```

Tabla 4.96 Instancia 3 carrera.

Utilizando las reglas de transformación para las entidades asociativas propuestas en este punto, los documentos asociados a la colección *Títulos* tendrían la siguiente información:

El estudiante *Rubén* obtuvo el título en *Periodismo* el año *2000*, así como el de *Ingeniería informática* en el *2023*. Se crean dos documentos, uno por título obtenido [Tabla 4.97], y [Tabla 4.98] referenciando tanto el Estudiante que lo obtuvo como la *Carrera* a la que pertenece el título.

```
// Documento título_1
{
   _id: ObjectId (65d9de6bc9215662e68ca836),
   Número_título: 24,
   Fecha_obtención: 2000,
   Estudiantes: '65d9be24c9215662e68ca821', // Estudiante Rubén
   Carreras: '65d9cbd5c9215662e68ca827' // Carrera periodismo
}
```

Tabla 4.97 Instancia 1 título.

```
// Documento título_2
{
   _id: ObjectId (65d9de4ec9215662e68ca834),
   Número_título: 22,
   Fecha_obtención: 2023,

   Estudiantes: '65d9be24c9215662e68ca821', // Estudiante Rubén
   Carreras: '65d9cb50c9215662e68ca825' // Carrera ingeniería
}
```

Tabla 4.98 Instancia 2 título.

La estudiante *Henar* obtuvo el título en *Psicología* en el año *2024*. Se crea el documento perteneciente a la colección *Título* referenciando tanto a la estudiante como a la carrera asociada. [Tabla 4.99]

```
// Documento título_3
{
   _id: ObjectId (65d9de0bc9215662e68ca833),
   Número_título: 21,
   Fecha_obtención: 2024,
   Estudiantes: '65d9be93c9215662e68ca822', // Estudiante Henar
   Carreras: '65d9cbadc9215662e68ca826' // Carrera Psicología
```

```
}
```

Tabla 4.99 Instancia 3 título.

Por último, el estudiante *Miguel* obtuvo el título en *Periodismo* en el año *2023*. Referenciamos tanto al estudiante *Miguel* como la carrera *Periodismo* en el documento de *Título*. [Tabla 4.98]

```
// Documento título_4
{
   _id: ObjectId (65d9de6bc9215662e68ca836),
   Número_título: 23,
   Fecha_obtención: 2000,
   Estudiantes: 65d9bf54c9215662e68ca823, //Estudiante Miguel
   Carreras: 65d9cbd5c9215662e68ca827//Carrera periodismo
}
```

Tabla 4.100 Instancia 4 título.

4.6 Relaciones recursivas.

Son aquellas relaciones en las que una entidad se relaciona consigo misma. Este tipo de relaciones requieren un nombre en el vínculo (rol). [Figura 4.31]

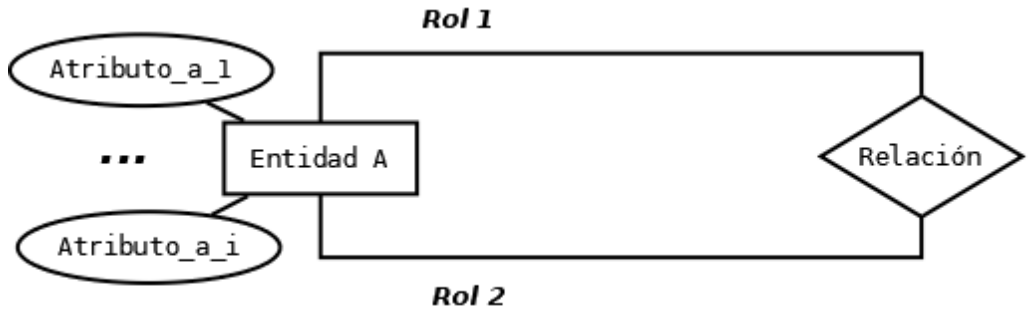

Figura 4.31 Ejemplo entidad recursiva.

Para anteriores transformaciones se han estudiado a lo largo de este trabajo diferentes formar de relacionar los documentos. Las relaciones recursivas pertenecientes a un diagrama *ER* también podríamos convertirlas a *MongoDB* con los instrumentos estudiados: podríamos crear dos colecciones para posteriormente incrustarlas o referenciarlas, o utilizar los mecanismos de herencia o relaciones jerárquicas vistos en las transformaciones ISA. Sin embargo, para continuar

con la estructura de conversiones que hemos utilizado hasta ahora y mantener una única colección por entidad representada en el diagrama (que en este caso representa la entidad o relación recursiva) decidimos crear una única colección y realizar la referencia dentro de los documentos que la componen.

Sea *A* un conjunto de entidades relacionadas con el mismo conjunto de entidades *A* (entidad recursiva) con *i* atributos.

En su paso a *MongoDB* el conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* con *i+3* campos clave valor. Uno para el *ObjectID* del propio documento y otros dos con clave el nombre del *rol 1* y valor el *ObjectId* con el que está relacionado, el último con clave el nombre del *rol 2* y valor el *ObjectId* con el que está relacionado. [Tabla 4.101]

Definida esta estructura de documentos, si en la entidad recursiva existen instancias que no tengan valor para alguno de los dos roles bastaría con marcar el valor del campo como null o no introducir el campo clave-valor referente a ese rol. Además, esta estructura nos permite recursividad en *árbol* (en el sentido de que una instancia *a* puede estar relacionada de forma recursiva con una *b* y esta a su vez con una *c* o la misma *a*) simplemente referenciando la instancia o documento con el que esté relacionado.

```
// Colección A
// Documento a1
{
   _id: <ObjectId_a1>,
   Atributo_a1_1: valor_a1_1,
   Atributo_a1_2: valor_a1_2,
   ...
   Atributo_a1_i: valor_a1_i,
   Nombre_rol1: <ObjectId_ax>, // Siendo ax el documento relacionado con nombre rol 1
   Nombre_rol2: <ObjectId_ay> // Siendo ay el documento relacionado con nombre rol 2
}
```

Tabla 4.101 Transformación entidad recursiva.

Ejemplo: Dentro del entorno de la Universidad existe un programa llamado *Programa mentor* en el que alumnos veteranos ayudan en diferentes facetas del entorno universitario a aquellos que llegan por primera vez a la Universidad. Ambos son estudiantes con una serie de atributos comunes, la única diferencia es que unos pueden hacer de mentor de otros (los denominaremos Aprendices). En la siguiente figura vemos el componente del diagrama ER en el que se representa dicha entidad recursiva [Figura 4.32]

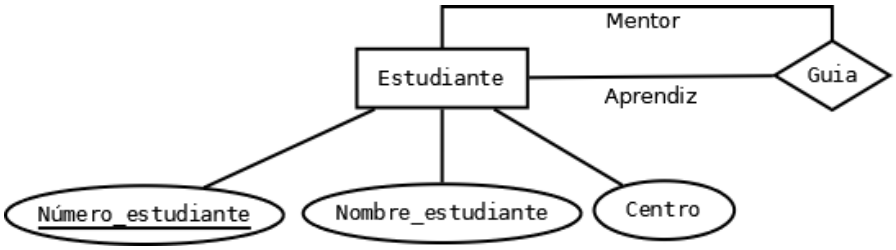

Figura 4.32 Ejemplo entidad recursiva ER.

Tomemos dos instancias: *estudiante_1* y *estudiante _2* del conjunto de entidades recursivas *Estudiantes* con los siguientes valores de atributos: [Tabla 4.102]

Estudiante	Instancia 1	Instancia 2
Número_estudiante	321261	321262
Nombre_estudiante	Rubén	Henar
Centro	ETSII	ESS

Tabla 4.102 Instancias estudiante.

La estudiante *Henar* es mentora del estudiante Rubén, mientras que *Rubén* es mentor de la estudiante *Henar*.

Solo tendríamos que crear una colección con nombre *Estudiantes* y referenciar los documentos con la información de *Rubén* y *Henar* como comentábamos en la regla de transformación. [Tabla 4.103] y [Tabla 4.104]

```
// Colección Estudiante
// Documento estudiante_1
{
   _id: ObjectId ('65d9be24c9215662e68ca821'),
   Número_estudiante: 321261,
   Nombre: Rubén,
   Centro: ETSII
   Mentor: null, // o podemos omitir el campo
   Aprendiz: '65d9be93c9215662e68ca822'// Aprendiz de Henar
}
```

Tabla 4.103 Instancia 1 estudiante.

```
// Colección Estudiante
// Documento estudiante_2
{
```

```
    _id: ObjectId (65d9be93c9215662e68ca822),
    Número_estudiante: 321262,
    Nombre: Henar,
    Centro: ESS,
    Mentor: '65d9be24c9215662e68ca821', // Mentor de Rubén
    Aprendiz: null, // o podemos omitir el campo
}
```

Tabla 4.104 Instancia 2 estudiante.

4.7 Agregación.

Hay ocasiones en las que existe una relación entre conjuntos de entidades (supongamos *A* y *B*) y se desea relacionar ese conjunto (ambas entidades junto con su relación asociada) con otro conjunto de entidades diferente *C*. [Figura 4.33]

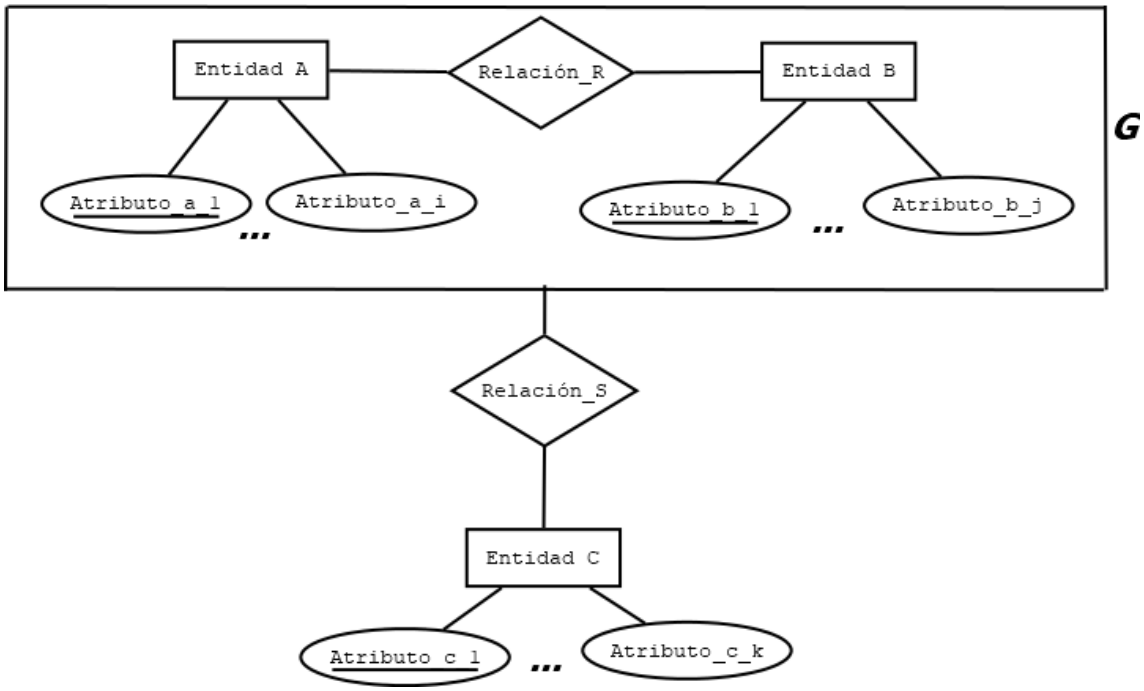

Figura 4.33 Ejemplo agregación ER

Trataremos la relación entre entidades (en el ejemplo la relación *R* entre el conjunto de entidades *A* y *B*) como un conjunto de entidades de orden superior o más compleja que llamaremos *G*.

A la hora de transformar el concepto de **agregación** en *MongoDB* el primer paso a realizar es transformar el conjunto de entidades superior (*G*) con las técnicas vistas ahora dependiendo del tipo de relación existente entre ambos conjuntos de entidades 1:1 [4.3.1] , 1:N [4.3.2] o N:M [4.3.3]

Posteriormente transformaremos el conjunto de entidades *C* del mismo modo que hemos transformado las entidades. [4.1]

Por último, añadimos dos nuevos campos clave valor a los documentos de la colección *C*. Las claves serán los nombres de las colecciones *A* y *B* mientras que el valor asociado dependerá del tipo de relación (1:1, 1:N o N:M) de *S*. Dichos valores serán referencias a los documentos relacionados de sus respectivas colecciones.

No expondremos todos los casos ya que han sido definidos anteriormente en esta memoria, pero veamos el caso general para una relación 1:1 (La relación contenida en *G*) y 1:1 (La relación del conjunto de entidades *C* (mediante *s*) con el conjunto de entidades *G*).

Obtendríamos de esta forma tres colecciones, con documentos *a* de la colección *A* con la siguiente información [Tabla 4.105]:

```
// Colección A
// Documento a
{
   _id: <ObjectId_a>,
   Atributo_a_1: valor_a_1,
    ...
   Atributo_a_i: valor_a_i,
   Nombre_colección_B: <ObjectId_b>
}
```

Tabla 4.105 Documento a

Documentos *b* de la colección *B* con la siguiente información [Tabla 4.106]:

```
// Colección B
// Documento b
{
   _id: <ObjectId_b>,
   Atributo_b_1: valor_b_1,
    ...
   Atributo_b_j: valor_b_j,
   Nombre_colección_A: <ObjectId_a>
}
```

Tabla 4.106 Documento b

Y por último documentos *c* de la colección *C* con la siguiente información [Tabla 4.107]:

```
// Colección C
// Documento c
{
   _id: <ObjectId_c>,
   Atributo_c_1: valor_c_1,
    ...
   Atributo_c_k: valor_c_k,
   Nombre_colección_A: <ObjectId_a>,
   Nombre_colección_B: <ObjectId_b>,
}
```

Tabla 4.107 Resultado colección S agregación

Ejemplo: Supongamos que un profesor solo puede hacer de tutor para un único *alumno* (algo que en el mundo real no es correcto, pero nos sirve para ejemplificar la regla general descrita anteriormente) e igualmente un alumno solo puede tener un *tutor*. Ambos colaboran para *elaborar* un *TFG* [Figura 4.32]

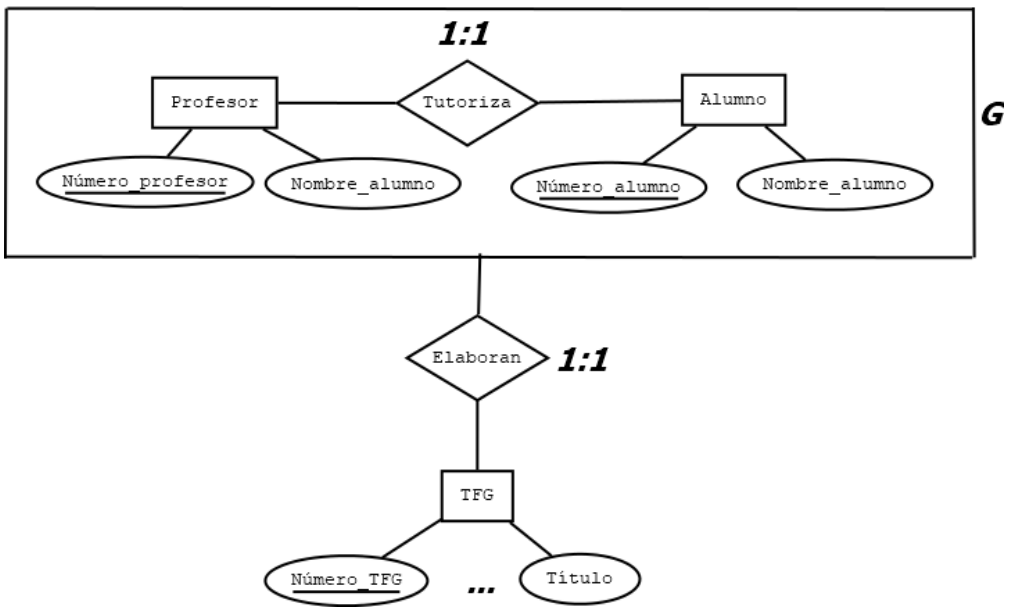

Figura 4.34 Ejemplo agregación ER

Tomemos una instancia: *alumno_1* del conjunto de entidades *Alumno* con los siguientes valores de atributos: [Tabla 4.106]

Alumno	Instancia 1
Número_alumno	321261
Nombre_alumno	Miguel Ángel

Tabla 4.108 Instancia alumno

Y tomemos una instancia: *profesor_1* del conjunto de entidades *Profesor* con los siguientes valores de atributos: [Tabla 4.107]

Profesor	Instancia 1
Número_profesor	987321
Nombre_profesor	Henar

Tabla 4.109 Instancia profesor

Por último, una instancia *tfg_1* del conjunto de entidades *TFG* con la siguiente información: [Tabla 4.108]

TFG	Instancia 1
Número_TFG	159267
Título	Método de transformación

Tabla 4.110 Instancia TFG

Siguiendo las reglas definidas transformamos el conjunto *G* en sus respectivas colecciones y documentos obteniendo de esta manera las colecciones *Profesor* y *Alumno* con los documentos *profesor_1* [Tabla 4.109]:

```
// Colección Profesor
// Documento profesor_1
{
   _id: ObjectId (664d9d26d9b8c163dca26a39),
   Número_profesor: 987321,
   Nombre_profesor: Henar,
   Alumno: ‘664d9d26d9b8c163dca26a38’ // Referencia a alumno
}
```

Tabla 4.111 Documento profesor_1

Y *alumno_1* [Tabla 4.110]*:*

```
// Colección Alumno
// Documento alumno_2
{
   _id: ObjectId (‘664d9d26d9b8c163dca26a38’),
   Número_alumno: 321261,
   Nombre_alumno: Miguel Ángel,
   Profesor: ‘664d9d26d9b8c163dca26a39’ // Referencia a profesor
}
```

Tabla 4.112 Documento alumno_1

Por último, transformamos el conjunto de instancias *TFG* en su respectiva colección con el documento *Tfg_1* añadiendo las referencias a los documentos *alumno_1* y *profesor_1* con los que está relacionado, obteniendo la siguiente información:

```
// Colección TFG
// Documento TFG_1
{
   _id: ObjectId ('664d9d28d9b8c163dca26a3a'),
   Número_TFG: 159267,
   Título: Método de transformación,
   Profesor: ‘664d9d26d9b8c163dca26a39’, // Referencia a profesor
   Alumno: ‘664d9d26d9b8c163dca26a38’ // Referencia a alumno
}
```

Tabla 4.113 Resultado transformación TFG

4.8 Tabla resumen reglas de transformación.

A continuación, mostramos un pequeño resumen con las principales reglas de transformación de los componentes de un diagrama ER en sus respectivos elementos de una base de datos con *MongoDB*. [Tabla 105]

Entidades.
Se utilizará siempre un Json squema a la hora de crear las colecciones resultantes de las transformaciones, en el que se definirá el tipo de dato de los campos, así como el requerimiento de los mismos y las restricciones a implementar.
Sean *A*, *B* y *C* conjuntos de entidades. Cada una de ellas con *i*, *j y k* atributos respectivamente.

Fuertes	Se convertirá en una colección cuyo nombre será el del conjunto de entidades. En general, los atributos se corresponderán con los campos de los documentos. Una entidad fuerte *A* será una colección *A*.
Débiles	Se crea un campo clave valor dentro del documento de la entidad fuerte con la que está relacionada. Dicho campo tendrá como clave el nombre de la entidad débil y tantos valores clave valor, como atributos tuviera la entidad débil, sin necesidad de añadir o crear el ObjectID de esta última. Una entidad débil *B* pasará a ser un documento incrustado en la entidad fuerte *A* con la que está relacionada.
Atributos	
Simples	Campos clave valor con clave el nombre del atributo de la entidad y valor el mismo de la instancia
Compuestos	Campo clave valor con clave el nombre del atributo compuesto y valor tantos pares clave valor como atributos simples contuviera el atributo compuesto.
Mono-valuados	Campos clave valor con clave el nombre del atributo de la entidad y valor el mismo de la instancia
Multi-valuados	Campo clave valor cuya clave es el nombre del atributo de la entidad y valor un array conteniendo tantos valores como posibles valores tuviera el atributo multi-valuado
Derivados	Requieren de la existencia de otro atributo. Para cada atributo derivado se necesita implementar dicho campo clave valor en *MongoDB mediante las denominadas Aggregation operations*.
Requeridos	En el diagrama ER todos los atributos que aparecen son requeridos. Conviene utilizar siempre un Json Squema a la hora de crear las colecciones marcando como *required* aquellos atributos que aparezcan en la entidad.
Opcionales	Todos aquellos que se añadan de forma posterior a la creación de las colecciones.

Relaciones

El conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* el cual tendría *i+1* campos clave valor.

El conjunto de entidades *B* sería la colección *B* y una instancia de *b* sería un documento *b* con *j+1* campos clave valor.

Y sean a y b documentos pertenecientes a sus respectivas colecciones.

	Incrustando documentos	**Referenciando documentos**
Uno a uno	Incrustando el documento b (resp. a) en el documento a (resp. *b*) obtendríamos un único documento	Referenciamos el ObjectId de *b* en el documento *a*, añadiendo un campo clave

	a (resp. *b*), de la colección *A* (resp. *B*) con los campos de *a* y los campos de *b*.	valor con clave el nombre de la colección *B* y valor el *ObjectId* del documento *b* con el que está relacionado. De igual modo referenciaríamos el documento *a* en el documento *b,* referenciando el ObjectId de *a* en el documento *b* mediante un par clave valor con clave el nombre de la colección *A* y valor el ObjectId del documento *a* con el que está relacionado.
Uno a muchos	Incrustando los documentos *b* (resp. *a*) en el documento *a (*resp. *b),* obtendríamos un único documento *a (*resp. *b)* perteneciente a la colección *A* (resp. *B*) con sus propios campos clave valor y un campo adicional con clave el nombre de la colección *B* (resp. *A*) y valor un array conteniendo el conjunto de todos los campos clave valor de todos los documentos *b (*resp. *a)* con los que está relacionado.	***1.*** Referenciar los documentos *b (resp. a)* relacionados con los documentos *a (resp. b)*. Añadiendo un campo clave valor en los documentos *a* (resp. *b*) cuya clave será el nombre de la colección *B (*resp. *A)* y valor un array conteniendo los ObjectId de todos los documentos *b (*resp. *a)* relacionados. ***2.***Referenciar el documento a (resp. *b*) relacionado con el documento b (resp. *a*). Introducimos un campo clave valor adicional en los documentos *b* (resp. *a*) cuya clave será el nombre de la colección *A (*resp. *B)* y valor el ObjectID del documento *a (*resp. *b)* relacionado.
Muchos a muchos	Si tenemos *n* documentos (con *i* campos cada uno) de la colección *A* relacionados con *m* documentos (con *j* campos cada uno) de la colección *B*. Y del mismo modo, los *m* documentos de la colección *B* relacionados con *n* documentos de la relación *A*. Incrustando los documentos *b* en los documentos *a* obtendríamos *n* documentos *a* y *m* documentos *b*. Cada documento *a* contendría además un nuevo campo clave valor con clave el nombre de la colección *B* y valor un array conteniendo todos los campos clave valor de cada uno de los documentos de la colección b con los que está relacionado. Del mismo modo cada documento b contendría un nuevo campo clave valor con clave el nombre de la colección A y valor un array	1.Referenciamos los documentos *a (resp. b)* relacionados con los documentos *b (resp. a)*. introduciendo un campo adicional en los documentos *b (resp. a)* cuya clave será el nombre de la colección *A (resp. B)* y valor un array conteniendo los ObjectId de todos los documentos *a* (resp. *b*) relacionados. 2.Referenciamos los documentos *b (*resp. *a)* relacionados con los documentos *a (*resp. *b)*. Para ello introducimos un campo adicional en los documentos *a (*resp. *b)* cuya clave será el nombre de la colección *B (*resp. *A)* y valor un array conteniendo los ObjectId de todos los documentos *b (*resp. *a)* relacionados.

	conteniendo todos los campos clave valor de cada uno de los documentos de la colección a con los que está relacionado.	
Generalización y especialización		
Jerarquía ISA (Generalización y especificación)	Tres posibles soluciones: 1. Integrando la jerarquía de generalización en una sola colección. 2. Eliminando el conjunto de entidades de orden superior y transformando el conjunto de entidades de inferior orden en sus respectivas colecciones. 3. Transformando todos los conjuntos de entidades participantes en la relación ISA, cada uno de ellos en una colección. (Mejor aproximación)	
Relaciones recursivas		
Recursivas	El conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* con *i+3* campos clave valor. Uno para el *ObjectID* del propio documento y otros dos adicionales. Uno con clave el nombre del *rol 1* y tantos valores como documentos con los que esté relacionado conteniendo los *ObjectId* de los mismos, el último con clave el nombre del *rol 2* y tantos valores como documentos con los que esté relacionado conteniendo los *ObjectId* de los mismos	
Entidades asociativas		
Asociativas	Siendo C una entidad asociativa. En su paso a MongoDB el conjunto de entidades *A* sería la colección *A* y una instancia de *A* sería un documento *a* el cual tendría *i+1* campos clave valor. El conjunto de entidades *B* sería la colección *B* y una instancia de *b* sería un documento *b* con *j+1* campos clave valor. El conjunto de entidades asociativas *C* sería una colección *C* y una instancia de c sería un documento *c* con *k+3* campos clave valor: uno de ellos correspondiente al propio *ObjectID* del documento, otro con clave el nombre de la colección *A* y valor el *ObjectID* del documento *a* con el que está relacionado y por último uno con clave el nombre de la colección *B* y valor el *ObjectID* del documento *b* con el que está relacionado.	

Agregación	
Sea una relación entre conjuntos de entidades (supongamos *A* y *B*) y se desea relacionar ese conjunto *G* (ambas entidades junto con su relación asociada) superior con otro conjunto de entidades diferente *C* mediante la relación *s*	
Agregación	Transformamos el conjunto de entidades superior *G* con las técnicas vistas ahora dependiendo del tipo de relación existente entre ambos conjuntos de entidades 1:1, 1:N o N:M Posteriormente transformaremos el conjunto de entidades *C* del mismo modo que transformamos los conjuntos de entidades. Por último, añadimos dos nuevos campos clave valor a los documentos *c* de la colección *C*. Las claves serán los nombres de las colecciones *A* y *B* mientras que el valor asociado dependerá del tipo de relación (1:1, 1:N o N:M) de *S*. Dichos valores serán referencias a los documentos relacionados de sus respectivas colecciones.

Tabla 4.114 Tabla resumen reglas de transformación.

Capítulo 5 Ejemplo de transformación.

Una vez definidas las reglas de transformación para los diferentes componentes de un diagrama Entidad-Relación, en este capítulo se realizará la transformación de un diagrama ER completo en su correspondiente base de datos en *MongoDB* aplicando las reglas enunciadas en el capítulo anterior. [Capítulo 4]

Debido a la extensión que supondría incluir las transformaciones de todas las instancias en sus respectivas colecciones, documentos y relaciones, solo vamos a incluir en esta memoria una de ellas para cada uno de los componentes que aparecen en el diagrama ER objeto de transformación a modo de ejemplo.

En el script adjunto en este TFG se puede consultar todo el código necesario para crear la base de datos completa utilizando una máquina con una instalación de *MongoDB*.[19]

Para este capítulo, se ha decidido crear una base de datos que alberga la información referente a la proyección de películas en un conjunto de cines de una ciudad. Debido a que, en el apartado anterior, los datos de instancias utilizados han sido elegidos por el autor de este trabajo, se ha creído conveniente que la información del ejemplo de transformación completo sea referente a datos que pueden ser fácilmente accesibles con una simple conexión a internet (como pueden ser que directores o actores han llevado a cabo una determinada película, a que género pertenecen las mismas, o diferente información sobre algunos de los datos que vamos a exponer a continuación).

Independientemente, en el primer anexo de esta memoria [Anexo 1] se adjuntan tablas con las relaciones existentes entre las distintas instancias con las que hemos poblado nuestra base de datos para que sea más sencillo, una vez consultado el código, ver la relaciones que existen entre ellas.

[19] Rubén de Diego Varona "Código capítulo 5 TFG" [Recurso online] Disponible en: https://github.com/rubdedi/MongoDB/blob/main/TfgCap%C3%ADtulo5.js Fecha última consulta: 01/06/2024

5.1 Requisitos.

Se desea crear una base de datos para una aplicación llamada *Cine_TFG* de la que se desea almacenar la siguiente información:

Los cines de la ciudad de Valladolid, cada uno de los cuales tienen un número de cine independiente del resto, un nombre, una dirección completa (que consta del nombre de la calle, el número la ciudad y el código postal) así como el o los números de teléfono de contacto del cine.

Cada uno de estos cines proyectará una o más películas, pero siempre están en activo y tienen alguna proyección.

Se desea almacenar así un conjunto de películas. Estas no tienen por qué estar siendo emitidas en alguno de los cines, sino que también se guardarán aquellas que ya hayan sido proyectadas o de las que se sepa que van a proyectarse próximamente.

De cada una de estas películas se quiere guardar su número de película, el título, tanto original como con el que llegó a nuestro país, la duración en minutos, el país en el que se produjo y la fecha de estreno internacional. Igualmente se desea conocer el género al que pertenece cada película, teniendo en cuenta que una película puede ser de un único género o de varios.

Por cada película se almacenarán el o los directores que se encargaron de llevarla a cabo, así como los actores que la interpretaron, teniendo en cuenta que aquellas películas de no ficción o documentales no cuentan con actores. De cada uno de ellos (tanto de los actores como de los directores) queremos saber el número que les identifica, así como su nombre, fecha de nacimiento y nacionalidad.

También se desea (en caso de que exista) almacenar una cita perteneciente a la película y por la que se haya hecho famosa o sea conocida. De la misma se desea almacenar la cita textual tal cual se expresó en la película y el personaje (que no actor) que la entonó.

Es posible que las películas hayan recibido críticas. Si es así se desea almacenar hasta un máximo de 100 críticas por película, y de cada una de ellas su número, un breve resumen y las estrellas (a modo de nota) con las que se las valoró (siendo el número mínimo de estrellas 1 y un máximo de 5).

Sabemos que estas críticas están escritas o publicadas por críticos (cada uno con su número y nombre) los cuales publican las mismas en un solo medio cada uno (no existen críticos que escriban en dos o más medios) distintos, y que estos medios pueden de dos tipos: el medio tradicional impreso (periódicos o revistas) en cuyo caso se desea saber su tirada y si esta es nacional o internacional, o bien pueden ser medios digitales, en cuyo caso se desea saber su Url, además del número de medio, nombre y país del cual procede dicho medio.

Nota: Las instancias utilizadas para crear esta base de datos se han tomado de la web *Filmaffinity,* tanto la información de las películas como la de los actores y directores de las mismas. [20]

[20] Filmaffinity España [Recurso online] Disponible en: https://www.filmaffinity.com/es/main.html Fecha última consulta: 01/06/2024

5.2 Diagrama Entidad-Relación.

Dados los requisitos proporcionados en el apartado obtenemos el diagrama Entidad-Relación que se muestra en la [Figura 5.1]

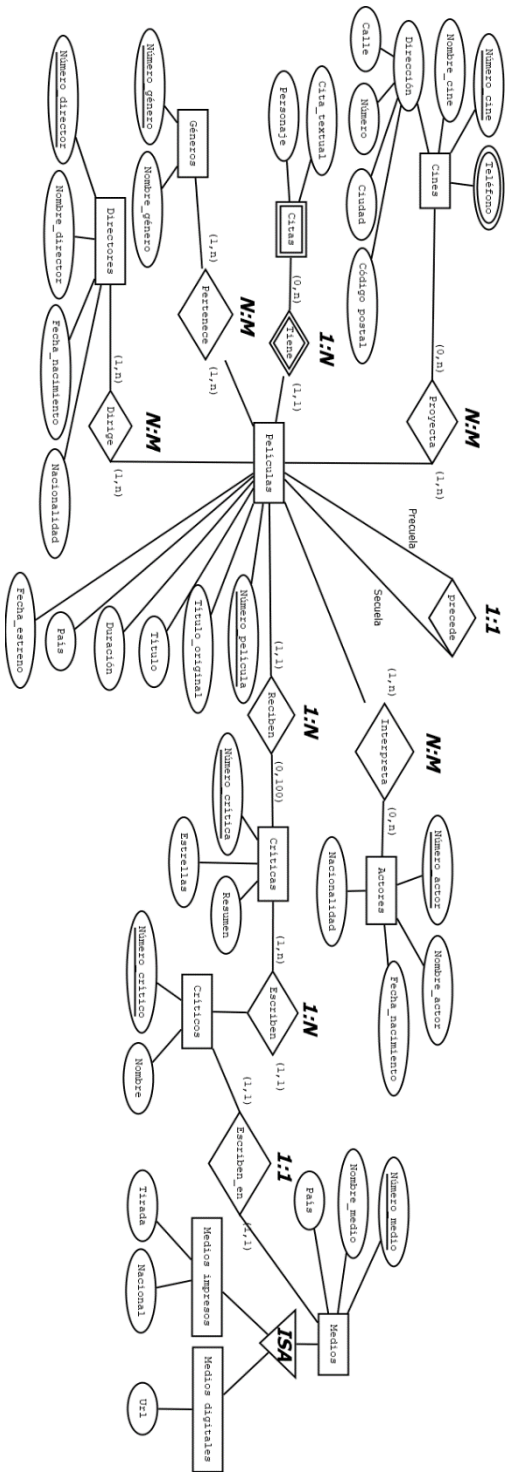

Figura 5.1 Diagrama Entidad-Relación Cines

5.2.1 Entidades y atributos.

Veamos cual ha sido el proceso que hemos llevado a cabo para obtener este diagrama E-R.

El primer paso a la hora de crear un diagrama Entidad-Relación es identificar las **entidades** y sus respectivos **atributos**. Atendiendo a los requisitos proporcionados plasmamos las siguientes entidades en el diagrama E-R de la [Figura 5.1]:

- **Cines**: Representa el conjunto de todos los cines en el universo planteado (en este caso Valladolid). Esta entidad va a poseer los siguientes atributos: *Número_cine* (el cual va a ser el atributo identificador, ya que este es único para el conjunto de todas ellas. Cada una de sus instancias va a tener su propio número de cine, el cual no se puede repetir e identifica de forma unívoca al mismo). *Nombre_Cine* indica el nombre del mismo, *Dirección* atributo compuesto a su vez de los siguientes atributos simples: *Calle*, *Número*, Ciudad (sabemos que es Valladolid ya que nos limitamos a ese universo, pero se nos pide almacenar la misma). Teléfono, atributo multivaluado ya que su valor puede ser múltiple y por último Código_postal.

- **Películas:** Esta entidad representa todas las películas emitidas, en emisión o por emitir en los diferentes cines de Valladolid. Sus atributos serán *Número_película* (atributo identificador del conjunto de entidades), *Título* (representará el nombre de la misma con el que se estrenó en nuestro país), *Título_original* (pudiendo coincidir con el anterior, pero almacenándolo como un atributo diferente), *Duración* (en minutos) *País* (indicará la nacionalidad de producción) y *Fecha_estreno* (el día en el que se presentó en salas).

- **Géneros:** Cada película pertenece a uno (o varios) de ellos. Decidimos identificarlo como entidad. Podríamos tratarlos como atributos de la entidad Película (con tipo multi valor), pero decidimos crear una entidad independiente para representarlos ya que nos parece una forma más natural de ver los datos (característica de este tipo de diagramas). Sus atributos serán *Número_género* (identificador) y *Nombre_género*.

- **Directores:** Representa el conjunto de directores que, valga la redundancia, dirigen películas o documentales**.** Sus atributos serán *Número_director* (atributo identificador o identificativo), *Nombre_director*, *Fecha_ nacimiento* y *Nacionalidad*. Todos ellos atributos simples ya que solo pueden tener un único valor.

- **Actores**: Representa el conjunto de actores de las diferentes películas. Los atributos de esta entidad serán: *Número_actor*, *Nombre*, *Fecha_ nacimiento* y *Nacionalidad*. Al igual que con la entidad Directores, todos sus atributos serán de tipo simple.

- **Citas:** Esta entidad tendrá dos atributos, la *Cita_textual* nombrada en la película a la que pertenece, así como el *Personaje* que la interpreta en la película. Observamos que esta entidad no existiría de no ser de la existencia de la entidad Películas, por lo que tenemos una entidad débil (y así la plasmaremos en el diagrama) y, por lo tanto, carece de atributo identificador.

- **Críticas:** Con tres atributos diferentes, uno de ellos el atributo identificador (*Número_crítica*), así como *Estrellas* y *Resumen* de la misma.
- **Críticos:** Esta entidad representará el conjunto de críticos que valoran las diferentes películas que almacenaremos en nuestra base de datos. Posee dos atributos: *Número_crítico* (atributo identificador de la entidad) y *Nombre* (atributo simple).

- **Medios:** Entidad que representa los diferentes medios para los que trabajan los críticos de películas. Con tres atributos todos ellos simples: *Número_medio* (identificador), *Nombre_medio* y *País* cuyo valor será el mismo al que pertenece.

A la hora de analizar los requisitos y crear el diagrama Entidad-Relación hemos identificado en primer lugar la entidad de orden superior **Medios** (con los atributos antes nombrados para todo el conjunto de entidades), para posteriormente identificar dos entidades de orden inferior **Medios_impresos** y **Medios_digitales**. Este proceso es el de la *especialización*. Si hubiésemos identificado las entidades de orden inferior en primer lugar (para posteriormente hacerlo con una de orden superior con un conjunto de atributos comunes para todas), hablaríamos de Generalización. Podemos observar que el resultado final a la hora de implementar las entidades en el diagrama no variaría.

- **Medios_impresos:** Entidad perteneciente a la jerarquía ISA identificada con dos atributos simples: *Tirada* que representará el número de ejemplares repartidos para su venta y *Nacional* cuyos valores podrán ser "sí" o "no" (dependiendo de si se distribuye en nuestro país o en el extranjero).

- **Medios_digitales:** Con un único atributo *Url* que tendrá como valor la dirección web del mismo.

5.2.2 Relaciones.

Una vez identificadas y representadas las entidades de nuestro diagrama Entidad-Relación debemos hacerlo con las **relaciones** existentes entre las mismas para, posteriormente indicar la cardinalidad entre las relaciones y las entidades y, a partir de ellas, obtener el tipo de la relación.

De esta forma las relaciones que identificamos partiendo de los requisitos presentados son las siguientes:

- **Proyecta:** Relación existente entre las entidades *Cines* y *Películas*. Un cine puede proyectar una o varias películas (1,n), mientras que una película puede no ser proyectada en un cine (porque ya ha sido proyectada o lo hará en un futuro, o incluso no se proyecta nunca) o puede ser proyectada (esa misma instancia) en varios cines de forma simultánea (0,n).

  Tomamos la mayor de las cardinalidades identificadas en esta relación (como haremos en las restantes) y obtenemos un tipo de relación **N:M** (muchos a muchos).

- **Tiene:** Es la relación entre la entidad débil *Cines* y la entidad fuerte de la que depende esta última, *Películas*. Es por ello que es una relación identificante y la representamos como tal en el diagrama asociado (con un rombo doble o rombo con doble línea). Una película puede o no tener citas (0,n), mientras que una cita solo puede pertenecer a una película (y además se nos expresa en los requisitos que no se desea almacenar más de una), siendo así esta cardinalidad de (1,).

  Así obtenemos el tipo de la relación *Tiene*: 1:N (uno a muchos).

- **Pertenece:** Esta relación existe entre las entidades Géneros y Películas. Una película ha de pertenecer, mínimo, a un determinado género (1,n), al igual que a un género le pueden corresponder una o varias películas (1,n). No existen películas sin género.

  El tipo de la relación *Pertenece* es por tanto del tipo **N:M** (muchos a muchos).

- **Dirige:** Relación existente entre las entidades *Directores* y *Películas*. Una película puede tener uno o varios directores (no existen películas sin ellos) (1,n), mientras que un director puede dirigir una o más películas (1,n).

  Tenemos una relación del tipo **N:M** (Muchos a muchos).

- **Precede:** Relación existente entre diferentes instancias de *Películas*. Es por ello que es una relación recursiva y debemos nombrar sus roles, que serán *Precuela* y *Secuela*. Una película solo puede tener una precuela (1,1), al igual que una única secuela (1,1)

  Relación recursiva tipo **1:1** (uno a uno).

- **Interpreta:** Relación existente entre *Películas* y *Actores*. Un actor puede interpretar una o varias películas (1,n) mientras que una película puede no tener actores si es un documental (como se nos indica en los requisitos) o puede que tenga varios (0,n).

  Tipo de relación **N:M** (muchos a muchos).

- **Reciben:** Las *Películas* reciben *Críticas*. Una película puede o no tener críticas. En caso de tenerlas, se nos especifica que el máximo número a almacenar en nuestra base de datos es de 100 (identificamos una cardinalidad min, max) (0,100). Mientras que una determinada crítica pertenece a una determinada película.

  Tipo de relación de Reciben **1:N** (uno a muchos).

- **Escriben:** Relación entre *Críticas* y *Críticos*. Un crítico puede escribir una o varias críticas (1,n), pero una crítica es escrita por un determinado crítico (1,1).

  Tipo de relación **1:N** (uno a muchos).

- **Escriben_en:** Por último, identificamos la relación *Escriben_en* entre las entidades *Críticos* y *Medios*. Analizando los requisitos sabemos que un crítico solo puede

pertenecer a un medio (1,1) (aunque esto no sea así en la vida real), y que un determinado medio solo tiene contratado a un crítico (1,1)

5.3 Creación de la base de datos.

Iniciamos el servidor *MongoDB* con el siguiente comando:

```
sudo systemctl start mongod
```

El primer paso es crear la base de datos en la que alojaremos las colecciones, documentos y relaciones resultantes del diagrama Entidad-Relación proporcionado. Se decide llamar a la base de datos *TFG_Capítulo5*. La creamos mediante el comando necesario. [3.5.1]

```
test> use TFG
switched to db TFG
```

5.4 Conversión de entidades.

5.4.1 Entidades fuertes.

Aplicamos las reglas de transformación enunciadas en este trabajo y procedemos en primer lugar a transformar las entidades en las colecciones correspondientes aplicando un *Json squema* que vimos en el apartado [3.5.3]

Tenemos un total de 6 entidades fuertes que se convertirán en 6 colecciones con idéntico nombre al de la entidad del diagrama original y una jerarquía ISA que consta de 3 entidades, cuya transformación abordaremos tras transformar las 6 anteriores. [5.2.2]

Cada una de ellas serán implementadas con un *Json squema*, marcando todos los atributos como requeridos (*required*). Como vimos en el punto [3.5.2] de este trabajo, a la hora de poblar con instancias nuestra base de datos, si no se introduce alguno de los campos marcados como *required* a la hora de crear las colecciones nos devolverá un error, sin embargo, en el siguiente paso, a la hora de crear las relaciones existentes no tendremos problemas en añadir más campos clave valor.

Para el conjunto de entidades *Película* [Figura 5.2]

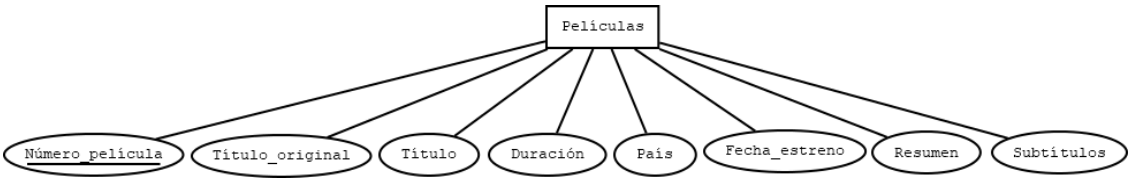

Figura 5.2 Conjunto de entidades Películas

Creamos la colección *Películas* con sus correspondientes campos, todos ellos Strings excepto *Número_película* y *Duración*, ambos Integer, el último con la condición añadida de que esté comprendido entre los valores 1 y 999:

```
// Creación colección Películas
db.createCollection("Películas", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Peliculas Object Validation",
         required: [ "Número_película",
"Título_original","Título","Duración","País","Fecha_estreno","Resumen","Subtítulos"],
         properties: {
            Número_película: {
               bsonType: "int",
               description: "'Número_película' ha de ser un string y el campo es
necesario"
            },
            Título_original: {
               bsonType: "string",
               description: "'Título_original' ha de ser un string y el campo es
necesario"
            },
            Título: {
               bsonType: "string",
               description: "'Título' ha de ser un string y el campo es necesario"
            },
            Duración: {
               bsonType: "int",
               minimum: 1,
               maximum: 999,
               description: "'Duración' ha de ser un integer dentro del intervalo [ 1,
999] y es necesario"
            },
            País: {
               bsonType: "string",
               description: "'País' ha de ser un string y el campo es necesario"
            },
            Fecha_estreno: {
               bsonType: "date",
               description: "'Fecha_estreno' ha de ser string y el campo es necesario"
            },
            Resumen: {
               bsonType: "string",
               description: "'Resumen' ha de ser string y el campo es necesario"
            },
            Subtítulos: {
```

```
                bsonType: "bool",
                description: "'Subtitulos' ha de ser un booleano y el campo es
necesario"
            }
        }
    }
  }
} )
```

A continuación, vemos un ejemplo de inserción de un documento *películas* dentro de su colección *Películas*, adaptándose al esquema Json creado:

```
//Poblamos la base de datos con instancias de películas
db.Películas.insertOne(
  {
    "Número_película": 1,
    "Título_original": "Alien",
    "Título": "Alien el octavo pasajero",
    "Duración": 116,
    "País": "Estados Unidos",
    "Fecha_estreno": new Date ("1979"),
    "Resumen": "De regreso a la Tierra, la nave de carga Nostromo interrumpe su viaje
y despierta a sus siete tripulantes. El ordenador central, MADRE, ha detectado la
misteriosa transmisión de una forma de vida desconocida, procedente de un planeta
cercano aparentemente deshabitado. La nave se dirige entonces al extraño planeta para
investigar el origen de la comunicación.",
    "Subtítulos": true
  }
)
```

Para el conjunto de entidades *Cines* [Figura 5.3].

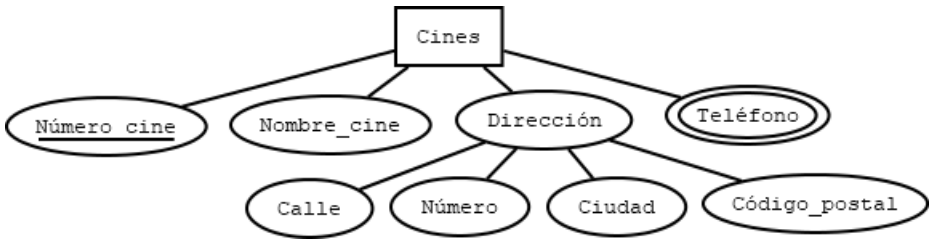

Figura 5.3 Conjunto de entidades Cines

Creamos la colección *Cines*. Realizamos dicha conversión al igual que la anterior con la particularidad de que *Dirección*, en el esquema ER es un atributo compuesto. En su paso al correspondiente campo introducimos en un array los atributos simples de los que consta el mismo [4.2.2]. Teléfono es un atributo compuesto, pero a la hora de realizar el esquema no es necesario indicarlo, ya que el hecho de que sea compuesto no quiere decir necesariamente que una instancia tenga que tener si o si más de un valor.

```
// Creación colección Cines
db.createCollection("Cines", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Cines Object Validation",
         required: [ "Número_cine", "Nombre_cine","Dirección","Teléfono"],
         properties: {
            Número_cine: {
               bsonType: "int",
               description: "'Número_cine' ha de ser un string y el campo es
necesario"
            },
            Nombre_cine: {
               bsonType: "string",
               description: "'Nombre_cine' ha de ser un string y el campo es
necesario"
            },
            Dirección: {
               bsonType: "object",
               required: [ "Calle", "Número","Ciudad","Código postal"],
               properties: {
               "Calle": {bsonType: "string"},
               "Número": {bsonType: "int"},
               "Ciudad": {bsonType: "string"},
               "Código postal": {bsonType: "string"}
           }
            },
            Teléfono: {
               bsonType: "string",
               description: "'Dirección' ha de ser un string y el campo es necesario"
            }
         }
      }
   }
} )
```

Veamos un ejemplo de inserción de un documento *cine* dentro de su colección *Cines*, adaptándose al esquema Json creado:

```
//Poblamos la base de datos con instancias de cines
db.Cines.insertOne(
         {
         "Número_cine": 1,
         "Nombre_cine": "Broadway",
         "Dirección":{
                  "Calle":"C. de Leopoldo Cano",
                  "Número":8,
                  "Ciudad":"Valladolid",
                  "Código postal":"47003"
         },
         "Teléfono": [{"983377134","983377134","983377135"}]
         }
```

```
)
```

Para el conjunto de entidades *Directores*. [Figura 5.4]

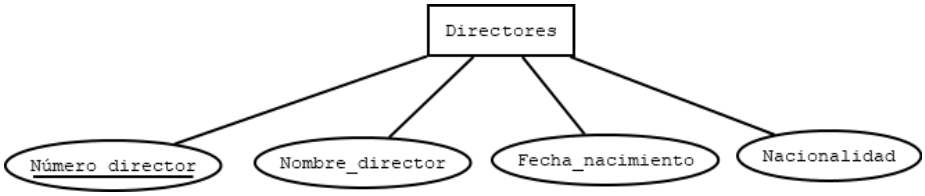

Figura 5.4 Conjunto de entidades Directores.

Creamos la colección *Directores*:

```
// Creación colección Directores
db.createCollection("Directores", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Directores Object Validation",
         required: [ "Número_director",
"Nombre_director","Fecha_nacimiento","Nacionalidad"],
         properties: {
            Número_director: {
               bsonType: "int",
               description: "'Número_director' ha de ser un string y el campo es
necesario",
            },
            Nombre_director: {
               bsonType: "string",
               description: "'Nombre_director' ha de ser un string y el campo es
necesario",
            },
            Fecha_nacimiento: {
               bsonType: "date",
               description: "'Fecha_nacimiento' ha de ser un date y el campo es
necesario"
            },
            Nacionalidad: {
               bsonType: "string",
               description: "'Nacionalidad' ha de ser un string y el campo es
necesario"
            }
         }
      }
   }
} )
```

A continuación, vemos un ejemplo de inserción de un documento *directores* dentro de su colección *Directores*, adaptándose al esquema Json creado:

```
//Poblamos la base de datos con instancias de directores
db.Directores.insertOne(
{
          "Número_director": 1,
          "Nombre_director": "Quentin Tarantino",
          "Fecha_nacimiento":new Date ("1963-02-03"),
          "Nacionalidad": "Estados Unidos"
  }
)
```

Para el conjunto de entidades Actores [Figura 5.5].

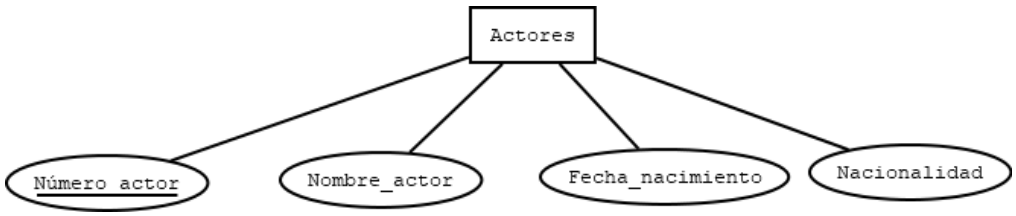

Figura 5.5 Conjunto de entidades Actores

La colección *Actores* tendría la siguiente estructura:

```
// Creación colección Actores
db.createCollection("Actores", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Actores Object Validation",
         required: [ "Número_actor",
"Nombre_actor","Fecha_nacimiento","Nacionalidad"],
         properties: {
            Número_actor: {
               bsonType: "int",
               description: "'Número_director' ha de ser un string y el campo es
necesario",
            },
            Nombre_actor: {
               bsonType: "string",
               description: "'Nombre_director' ha de ser un string y el campo es
necesario",
            },
            Fecha_nacimiento: {
               bsonType: "date",
```

```
                description: "'Fecha_nacimiento' ha de ser un date y el campo es
necesario"
            },
            Nacionalidad: {
                bsonType: "string",
                description: "'Nacionalidad' ha de ser un string y el campo es
necesario"
            }
         }
      }
   }
} )
```

Con un documento *actores* de la colección *Actores* con la siguiente información:

```
//Poblamos la base de datos con instancias de actores
db.Actores.insertOne(
{
          "Número_actor": 1,
          "Nombre_actor": "Bruce Willis",
          "Fecha_nacimiento":new Date("1955"),
          "Nacionalidad": "Estados Unidos"
  }
)
```

Para el conjunto de entidades *Géneros* [Figura 5.6].

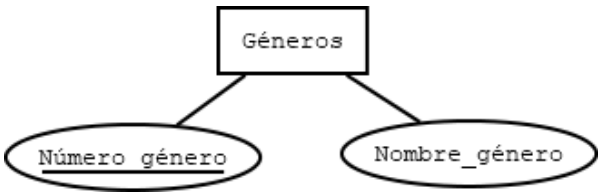

Figura 5.6 Conjunto de entidades Géneros

La colección *Géneros* solo posee dos campos, el número del mismo y el nombre. Dicho nombre solo puede pertenecer a uno de los 10 definidos en los requisitos. Introducimos en el esquema dicho campo como *requerido*, y perteneciente a un tipo de dato *enum* en Bson. Los posibles valores de *enum* se introducen en un array, de tal forma que si a la hora de poblar la base de datos, un documento de la colección *Géneros* se introduce con un campo *Nombre_género* distinto a los pertenecientes al array, nos devolverá un error.

```
// Creación colección Géneros
db.createCollection("Géneros", {
```

```
    validator: {
       $jsonSchema: {
          bsonType: "object",
          title: "Géneros Object Validation",
          required: [ "Número_género", "Nombre_género"],
          properties: {
             Número_género: {
                bsonType: "int",
                description: "'Número_género' ha de ser un string y el campo es
necesario",
             },
             Nombre_género: {
                enum: ["Acción","Aventuras","Ciencia ficción","Comedia","No-
ficción","Drama","Fantasía","Musical","Suspense","Terror"],
                description: "'Nombre_género' ha de pertenecer a alguno de los géneros
enumerados",
             }
           }
       }
    }
} )
```

De esta forma, uno de los 10 posibles documentos *géneros* pertenecientes a la colección *Géneros tendría la siguiente información:*

```
// Poblamos con las instancias de géneros
db.Géneros.insertOne(
{
          "Número_género": 1,
          "Nombre_género": "Acción"
  }
)
```

Para el conjunto de entidades *Críticas* [Figura 5.7].

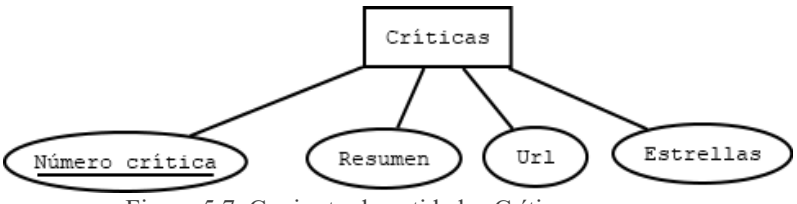

Figura 5.7 Conjunto de entidades Críticas

La colección *Críticas* se transforma de igual modo que las vistas hasta ahora. El único requisito pedido es que el número de críticas de una determinada película no sea superior a 100. Para ello definimos el tipo de dato *Bson* como Integer y marcamos el mínimo y el máximo de las mismas con *mínimum* y *máximum* dentro de las propiedades del mismo.

```
// Creación colección Críticas
db.createCollection("Críticas", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Críticas Object Validation",
         required: [ "Número_crítica", "Resumen","Url","Estrellas"],
         properties: {
            Número_crítica: {
               bsonType: "int",
               minimum: 1,
               maximum: 99,
               description: "'Número_crítica ha de ser un int en el intervalo [1 ,99]
y el campo es necesario",
            },
            Resumen: {
               bsonType: "string",
               description: "'Resumen' ha de ser un string y el campo es necesario",
            },
            Url: {
               bsonType: "string",
               description: "'Url' ha de ser un string y el campo es necesario",
            },
            Estrellas: {
               bsonType: "int",
               minimum: 1,
               maximum: 5,
               description: "'Estrellas' ha de ser un int en el intervalo [1 ,5] y el
campo es necesario",
            }
          }
      }
   }
} )
```

Así un documento *crítica* de la colección *Críticas* se introduciría en la base de datos de la siguiente manera:

```
//Poblamos la base de datos con instancias de críticas
db.Críticas.insertOne(
{
          "Número_crítica": 1,
          "Resumen": "Sin lugar a dudas, el thriller de los años noventa. Un
formidable ejercicio de suspense psicológico con toques de terror que arrasó en las
taquillas, consiguió un hito pocas veces visto en los Oscar (se llevó los 5 premios
principales... ¡cuando llevaba un año estrenada!) y demostró la maestría de Anthony
Hopkins al interpretar de forma prodigiosa a un psiquiatra caníbal que fascinó a
todos. Aunque, para ser justos, lo que hace que The Silence of the Lambs destaque
sobre la mayoría de películas del género no es sólo el magnético personaje de Lecter,
sino la inteligencia de su guión y la habilidad de su directa y poco glamourosa
dirección. El texto alterna interesantísimos detalles sobre la investigación en curso
con sorpresas tan impredecibles como angustiosas. Y la realización se apoya en escenas
```

```
impactantes y multitud de primeros planos, consiguiendo exprimir la mayor potencia
posible del relato. Mención especial para las escenas -y diálogos- entre Clarice y el
Dr. Lecter, todas absolutamente memorables. Un clásico imprescindible.",
          "Url":"https://www.filmaffinity.com/es/film768790.html",
          "Estrellas": 5
  }
)
```

Para el conjunto de entidades *Críticos* [Figura 5.8].

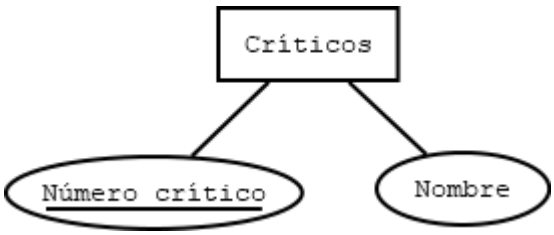

Figura 5.8 Conjunto de entidades Críticos

De igual modo creamos la colección *Críticos*, aplicando la restricción (*min, max*) pedida en el número de Críticos.

```
// Creación colección Críticos
db.createCollection("Críticos", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Críticos Object Validation",
         required: [ "Número_crítico", "Nombre"],
         properties: {
            Número_crítico: {
               bsonType: "int",
               minimum: 1,
               maximum: 99,
               description: "'Número_crítico' ha de ser un int en el intervalo [1 ,99]
y el campo es necesario",
            },
            Nombre: {
               bsonType: "string",
               description: "'Nombre' ha de ser un string y el campo es necesario",
            }
          }
      }
   }
} )
```

Introducimos un *crítico* en su correspondiente colección *Críticos*:

```
db.Críticos.insertOne(
{
          "Número_crítico": 1,
          "Nombre": "Pablo Kurt"
  }
)
```

5.4.2 Entidades de la jerarquía ISA.

En el diagrama ER objeto de transformación nos encontramos con una jerarquía ISA en la que intervienen tres conjuntos de entidades: Una de ellas de orden superior *Medios*, y dos de orden inferior *Medios impresos* y *Medios digitales*. *Medios* [Figura 5.9]

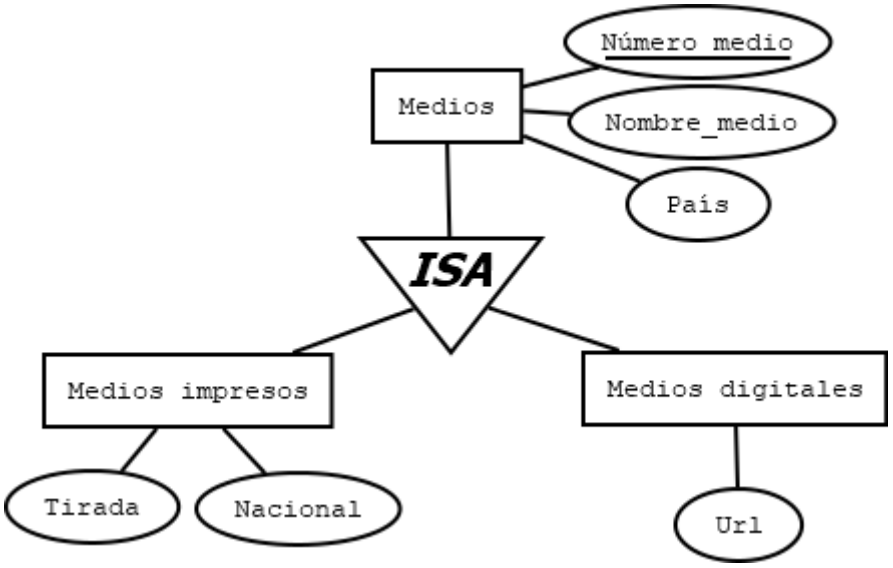

Figura 5.9 Jerarquía ISA Medios

Como vimos en el capítulo anterior, tenemos tres formas de afrontar esta transformación, pero nos decantaremos por la tercera de ellas: se *pueden transformar todos los conjuntos de entidades participantes en la relación ISA, cada uno de ellos en una colección*. Así que crearemos una colección *Medios* una colección *Medios impresos* y una tercera *Medios digitales*.

Así la colección *Medios* tendría el siguiente Json squema asociado a la hora de crearla:

```
// Creación colección Medios
db.createCollection("Medios", {
```

```
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Medios Object Validation",
         required: [ "Número_medio", "Nombre_medio","País"],
         properties: {
            Número_medio: {
               bsonType: "int",
               minimum: 1,
               maximum: 99,
               description: "'Número_crítico' ha de ser un int en el intervalo [1 ,99]
y el campo es necesario",
            },
            Nombre_medio: {
               bsonType: "string",
               description: "'Nombre_medio' ha de ser un string y el campo es
necesario",
            },
            País: {
               bsonType: "string",
               description: "'País' ha de ser un string y el campo es necesario",
            }
          }
      }
   }
} )
```

La colección Medios impresos:

```
// Creación colección Medios_impresos
db.createCollection("Medios_impresos", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Medios_impresos Object Validation",
         required: [ "Tirada", "Nacional"],
         properties: {
            Tirada: {
               bsonType: "int",
               description: "'Tirada' ha de ser un int en el intervalo y el campo es
necesario",
            },
            Nacional: {
               bsonType: "bool",
               description: "'Nacional' ha de ser un boolean y el campo es necesario",
            }
          }
      }
   }
} )
```

Y por último la colección Medios digitales:

```
// Creación colección Medios digitales
db.createCollection("Medios_digitales", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Medios_digitales Object Validation",
         required: [ "Url"],
         properties: {
            Tirada: {
               bsonType: "string",
               description: "'Url' ha de ser un int en el intervalo y el campo es
necesario",
            }
          }
      }
   }
} )
```

Mostramos dos ejemplos de documentos pertenecientes a la colección *Medios* (uno digital, el primero que mostramos, y otro tradicional, el segundo:

```
// Poblamos con las instancias de medios
// Medio digital
db.Medios.insertOne(
{
          "Número_medio": 1,
          "Nombre_medio": "SFGATE",
          "País": "Estados Unidos"
  }
)
// Medio impreso
db.Medios.insertOne(
{
          "Número_medio": 13,
          "Nombre_medio": "El País",
          "País": "España"
  }
)
```

Ahora simplemente tendríamos que referenciar en las instancias tanto del medio digital como el medio tradicional asociados, la instancia de mayor orden (*Medios*) a la que pertenecen. Para ello obtenemos los ObjectID de los documentos de mayor orden como vemos en un punto posterior de este capítulo [5.3.1]

```
//Medios digitales
//SFGATE - medio1
db.Medios_digitales.insertOne(
{
          "Url": "https://www.sfgate.com/",
          "Medio": ObjectId(Medio1)
  }
)
//Medios_impresos
//El país - medio13
db.Medios_impresos.insertOne(
{
          "Tirada": 1000000,
          "Nacional": true,
          "Medio": ObjectId(Medio13)
  }
)
```

5.4.3 Entidades débiles.

Una vez convertidas todas las entidades fuertes procedemos a convertir las entidades débiles.

En el diagrama presentado existe una relación entre las entidades *Películas* y *Citas.* Siendo *Películas* la entidad fuerte y *Citas* la débil relacionada con ella mediante *Tienen.* [Figura 5.10]

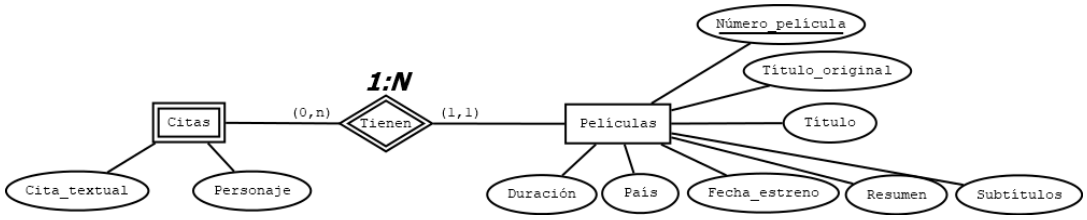

Figura 5.10 Relación Tienen entre Citas y Películas

Aplicando las reglas de transformación definidas [4.1.2.2] incrustamos la información relevante a las *Citas* dentro de los documentos relacionados de la colección *Películas*. Este paso se podría realizar a la hora de crear las colecciones con *Json Schema*, pero decidimos realizarlo de esta forma para que se tenga una mejor visión de las conversiones realizadas.

Con la información de las *Citas* obtenida, por ejemplo, para la película *El padrino*, se tienen 2 *citas*, cada una de ellas con dos campos: el primero la *cita textual* que se desea almacenar y el segundo el *personaje* de la película que dice esa determinada cita.

Para incrustar esta información en la ya existente colección *Películas* de nuestra base de datos, utilizamos el comando update para integrar (mediante set) un array con nombre *Citas* (el de la entidad débil) y dos campos con clave *Cita_textual* y *Personaje* y valor, el de cada uno de ellos.

```
db.Películas.updateOne({"Número_película": 12}, {$set: {Citas:[{"Cita_textual": "Le
haré una oferta que no podrá rechazar","Personaje": "Michael Corleone"},
{"Cita_textual": "No me gusta la violencia, Tom. Soy un hombre de negocios; la sangre
resulta muy costosa","Personaje": "Sollozzo"}]}} )
```

5.5 Conversión de relaciones.

5.5.1 Obtención del valor ObjectID.

Una vez creadas las colecciones y pobladas de documentos, el primer paso que vamos a realizar será obtener el ObjectID de cada uno de los documentos de las diferentes colecciones.

Esto nos será de extrema utilidad para entablar las relaciones, ya que a la hora de referenciar unos documentos en otros será por medio de sus objetos ObjectID. [3.3.1]

Para ello crearemos una variable con nombre el de la colección de la que deseamos obtener el ObjectID del documento seguido de un número entero que representa el número de instancia que deseamos posteriormente referenciar.

Así en el caso por ejemplo de la colección *Medios* almacenamos en la variable *Medio1* el *_id* del documento que coincida con el primer parámetro de *findOne*(). El segundo parámetro de dicho comando indica aquellos campos que deseamos sean visibles o se almacenen en la variable en caso de crearla. Para este caso solo deseamos el campo id, así que como segundo parámetro lo seleccionamos (el campo a mostrar con valor 1 se muestra, con valor 0 no lo hace). Es necesario utilizar el método predefinido toString() de *MongoDB* [16], el cual convierte el valor del objeto ObjectID en un String, el cual introduciremos posteriormente de forma manual a la hora de referenciar los documentos relacionados.

```
Medio1 = db.Medios.findOne({ 'Número_medio': 1 }, { '_id': 1 } )
Medio1['_id']
Medio1['_id'].toString
Medio1 = ""+Medio1['_id']
```

Para el caso de Medios realizaríamos la misma operación:

```
Crítico1 = db.Críticos.findOne({ 'Número_crítico': 1 }, { '_id': 1 } )
Crítico1['_id']
Crítico1['_id'].toString
Crítico1= ""+Crítico1['_id']
```

5.5.2 Relaciones uno a uno.

5.5.2.1 Relación entre Críticos y Medios.

En el diagrama Entidad-Relación proporcionado nos encontramos la relación de tipo uno a uno *Escribe_en* entre los conjuntos de entidades *Críticos* y *Medios.* [Figura 5.11]

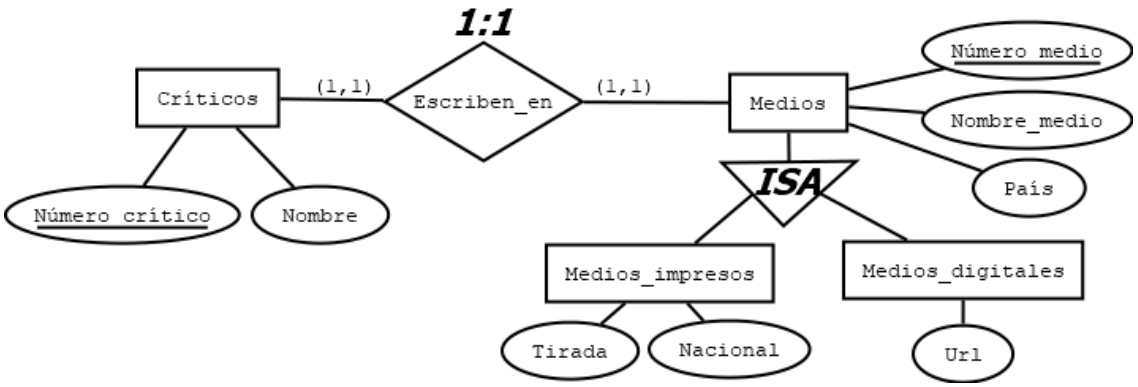

Figura 5.11 Relación Escriben_en entre Críticos y Medios

Aplicando las reglas de transformación definidas en el apartado [4.1.2.2] podríamos bien incrustar la información de una colección en otra o bien referenciar los documentos de ambas colecciones.

Para mantener el mismo número de entidades que de colecciones, al igual que hemos hecho en la mayoría de los casos de este estudio, decidimos referenciar la información de *Críticos* en *Medios* y la de *Medios* en *Críticos*.

Una vez poblada la base de datos con las instancias de Críticos y Medios obtenemos los ObjectID de aquellas instancias que están relacionadas con otras. Para ello utilizamos los siguientes comandos en *MongoDB* (este paso se realizará cada vez que queramos obtenerlas).

Teniendo la instancia de Medios con *Número_ medio:* 1 y la instancia de *Críticos* con *Número_crítico*: *1 y*, añadimos un nuevo campo clave valor con nombre el de la entidad con la que se relaciona (bien *Críticos* o *Medios*) seguido del ObjectId de la instancia con la que se relaciona:

Obteniendo para la instancia de Medio:

```
db.Medios.updateOne( { "Número_medio":16}, {$set:{"Críticos":ObjectId(Crítico1)}})
```

Y para la instancia de Críticos:

```
db.Críticos.updateOne( { "Número_crítico":1}, {$set:{"Medios":ObjectId(Medio16)}})
```

También tenemos la relación recursiva Películas, que posee una relación *Precede con los roles Secuela y Precuela.* Esta transformación la veremos más adelante una vez hayamos estudiado la transformación del conjunto de entidades *Películas* con todas sus relaciones. [5.3.5]

5.5.3 Relaciones uno a muchos.

En el diagrama Entidad-Relación proporcionado nos encontramos varias relaciones del tipo uno a muchos (o muchos a uno, las cuales son idénticas a las nombradas, solo que intercambiando la forma en la que están representadas la relaciones, izquierda-derecha o derecha-izquierda).

5.5.3.1 Relaciones entre Críticos y Críticas.

Tenemos la relación *Reciben* entre *Películas* y *Críticas* y la relación *Escriben* entre *Críticos* y *Críticas*.

Veamos la transformación de la relación *Pertenece* entre *Críticos* y *Críticas*: [Figura 5.12]

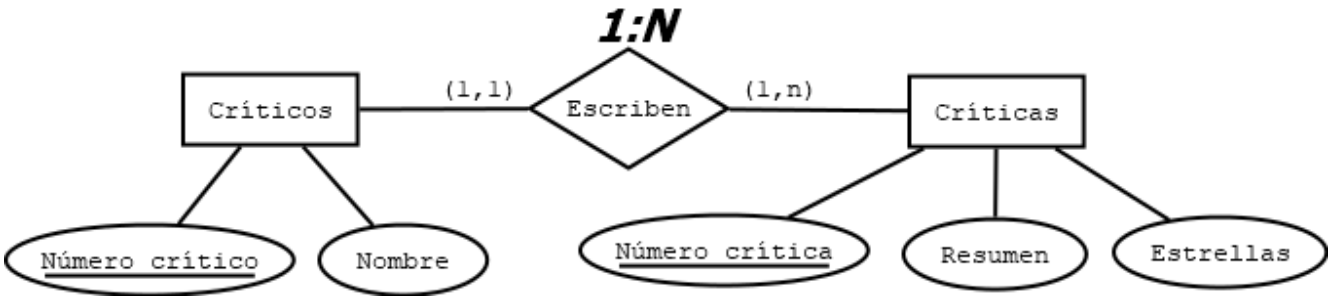

Figura 5.12 Relación Escriben entre Críticos y Críticas

Para transformar esta relación podríamos incrustar los documentos resultantes de la conversión de ambas Entidades bien incrustando documentos en otros o referenciándolos [4.1.2.2]

Al igual que para las relaciones uno a uno decidimos referenciar la información de *Críticos* en *Críticas* y la de *Críticas* en *Críticos*.

Una vez poblada la base de datos con las instancias de *Críticos* y *Críticas* obtenemos los ObjectID de aquellas instancias que están relacionadas con otras como vimos en el apartado anterior y creamos los campos clave valor correspondientes.

Por ejemplo, tenemos que las críticas con *Número_crítica*: *4, 7, 9* y *16* están relacionadas con el crítico con *Número_crítico* igual 4.

Obtenemos el valor de los ObjectID de las cuatro críticas deseadas. [5.3.1] e igualmente el del crítico, que posee *Número_crítico*: 4 [5.3.1]

Para posteriormente referenciar dichos valores tanto en los documentos del crítico:

```
db.Críticos.updateOne( { "Número_crítico":4},
{$set:{"Críticas":[ObjectId(Crítica4),ObjectId(Crítica7),ObjectId(Crítica9),ObjectId(C
rítica16)]}})
```

Así como en los de las críticas:

```
db.Críticas.updateOne( { "Número_crítica":4}, {$set:{"Críticos":ObjectId(Crítica4)}})
db.Críticas.updateOne( { "Número_crítica":7}, {$set:{"Críticos":ObjectId(Crítica7)}})
db.Críticas.updateOne( { "Número_crítica":9}, {$set:{"Críticos":ObjectId(Crítica9)}})
db.Críticas.updateOne( { "Número_crítica":16},
{$set:{"Críticos":ObjectId(Crítica16)}})
```

5.5.3.2 Relación entre Películas y Críticas.

Procedemos de igual modo con la relación Reciben entre Películas y Críticas, la cual es igualmente del tipo uno a muchos. [Figura 5.13]

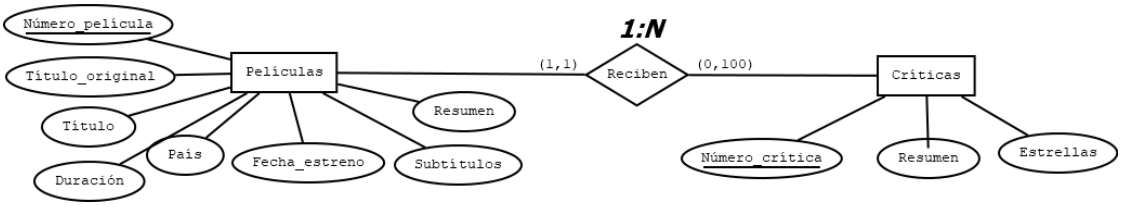

Figura 5.13 Relación Reciben entre Películas y Críticas

Procedemos de idéntico modo que en la relación anterior. La única diferencia es que podemos observar una restricción de cardinalidad (*Min, Max*) [2.3.3] en la cardinalidad a izquierdas de la relación *Reciben*.

Si atendemos al Json squema utilizado a la hora de crear la colección *Críticas*, observamos que la cantidad máxima de número de críticas que podemos insertar es de 100 (siempre que no se

introduzcan instancias diferentes con idéntico número de crítica, algo que queda en manos del desarrollador ya que *MongoDB* no limita el número de instancias con mismo valor de campo).

Una vez obtenidos los ObjectID tanto de los documentos *Películas* como de *Críticas*, solo bastaría referenciar en los documentos de la colección películas tantas críticas como tengan y en los documentos de la colección Críticas el ObjectID a la que pertenece.

Teniendo los documentos cuyo *Número_película* es igual a 1 y las críticas con *Número_crítica*: 4, 5 y 6, referenciamos en dicho documento de *Película*s los ObjectID de las tres críticas relacionadas.

```
db.Películas.updateOne( { "Número_película":1},
{$set:{"Críticas":[ObjectId(Crítica4),ObjectId(Crítica5),ObjectId(Crítica6)]}})
```

Mientras que para cada una de las tres críticas referenciamos el ObjectID de la película con la que se relaciona:

```
db.Críticas.updateOne({Número_crítica: 4}, {$set:{"Películas":ObjectId(Película1)}})
db.Críticas.updateOne({Número_crítica: 5}, {$set:{"Películas":ObjectId(Película1)}})
db.Críticas.updateOne({Número_crítica: 6}, {$set:{"Películas":ObjectId(Película1)}})
```

5.5.4 Relaciones muchos a muchos.

En el diagrama Entidad-Relación proporcionado nos encontramos las siguientes relaciones muchos a muchos: La relación *Interpreta* entre los conjuntos de entidades *Actores* y *Películas*, la relación *Dirige* entre los conjuntos de entidades *Directores* y *Películas*, la relación *Proyecta* entre *Cines* y *Películas* y por último la relación *Pertenece* entre *Géneros* y *Películas*.

5.5.4.1 Relación entre Actores y Películas

Procedemos con la relación existente entre *Actores* y *Películas*: [Figura 5.14]

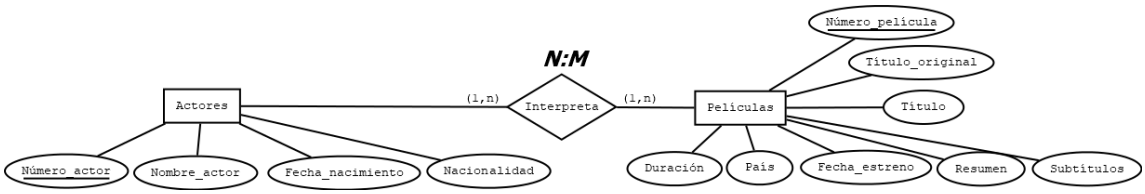

Figura 5.14 Relación Interpreta entre Actores y Películas

Tras obtener el valor de los ObjectID como se ha visto en anteriores transformaciones, simplemente tenemos que insertar un campo clave valor en los documentos de la colección *Actores* con clave *Películas* y tantos valores como ObjectID con los que está relacionada. De idéntico modo se añade en *Actores* un campo con clave *Películas* y valor tantos ObjectID como número de películas que interprete ese actor.

Tenemos, entre otras, las siguientes relaciones entre documentos de ambas colecciones:

El *Actor* con *Número_actor: 11* interpreta 3 películas, aquellas con *Número_película*: 1,2,3,4.

Las 4 películas mencionadas tienen a su vez (además de al actor con *Número_actor:* 11) los siguientes: la número 1 al *Actor* 11, la número 2 al *Actor* 13, la número 3 al *Actor* 14 y por último la número 4 al *Actor* 15

Para crear dichas relaciones introducimos en el documento que posee *Número_actor: 11* un nuevo campo cuya clave es *Películas* y valor un array conteniendo los 3 ObjectID de las películas con las que se relaciona.

```
db.Actores.updateOne( { "Número_actor":11},
{$set:{"Películas":[ObjectId(Película1),ObjectId(Película2),ObjectId(Película3),Object
Id(Película4)]}})
```

Mientras que en los documentos de las *Películas* cuyos *Número_película* son 1, 2, 3 y 4, añadimos un nuevo campo con clave *Actores* y valor el ObjectID de aquellos *Actores* que interpretan dicha película.

```
db.Películas.updateOne( { "Número_película":1},
{$set:{"Actores":[ObjectId(Actor11),ObjectId(Actor10)]}})
db.Películas.updateOne( { "Número_película":2},
{$set:{"Actores":[ObjectId(Actor11),ObjectId(Actor13)]}})
db.Películas.updateOne( { "Número_película":3},
{$set:{"Actores":[ObjectId(Actor11),ObjectId(Actor14)]}})
db.Películas.updateOne( { "Número_película":4},
{$set:{"Actores":[ObjectId(Actor11),ObjectId(Actor15)]}})
```

Este paso se realizaría para todas las relaciones existentes entre los documentos de ambas colecciones. Solo hemos mostrado un pequeño ejemplo, pero al igual que el resto de transformaciones de este trabajo, en el script adjunto se pueden consultar la creación de todas ellas.

5.5.4.2 Relación entre Directores y Películas.

Para la relación *Interpreta* entre los conjuntos de entidades *Directores* y *Películas*: [Figura 5.15]

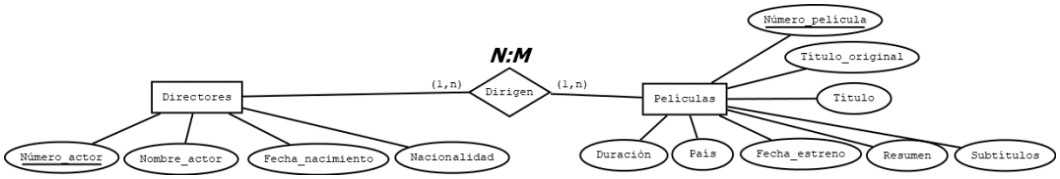

Figura 5.15 Relación Dirigen entre Directores y Películas

Si, por ejemplo, algunas de las relaciones existentes son las siguientes:

El *Director* con *Número_director: 8* dirige 3 películas, aquellas con *Número_película*: 3,5 y 6.

Para crear dichas relaciones introducimos en el documento que posee *Número_director: 8* un nuevo campo cuya clave es *Películas* y valor un array conteniendo los 3 ObjectID de las películas con las que se relaciona.

```
db.Directores.updateOne( { "Número_director":8},
{$set:{"Películas":[ObjectId(Película3),ObjectId(Película5),ObjectId(Película6)]}})
```

Igualmente, para cada una de las películas (aquellas con *Número_película*: 3,5 y 6) se crea un nuevo campo con clave *Director* y valor el ObjectID del director con el que están relacionadas.

```
db.Películas.updateOne( { "Número_película":3},
{$set:{"Directores":ObjectId(Director8)}})
db.Películas.updateOne( { "Número_película":5},
{$set:{"Directores":ObjectId(Director8)}})
db.Películas.updateOne( { "Número_película":6},
{$set:{"Directores":ObjectId(Director8)}})
```

Recordemos que estamos transformando una relación tipo muchos a muchos. A pesar de que la inmensa mayoría de *Películas* tienen un único director, es posible que una *Película* sea dirigida por más de uno, como es el caso de la película con *Número_película*: 11 creada en nuestra base de datos, la cual tiene dos directores.

```
db.Películas.updateOne( { "Número_película":11},
{$set:{"Directores":[ObjectId(Director7),ObjectId(Director11)]}})
```

Siendo necesario referenciar dicha *Película* en los documentos de *Directores* relacionados:

```
db.Directores.updateOne( { "Número_director":7},
{$set:{"Películas":[ObjectId(Película9),ObjectId(Película11)]}})
db.Directores.updateOne( { "Número_director":11},
{$set:{"Películas":ObjectId(Película11)}})
```

5.5.4.3 Relación entre Cines y Películas.

Para la relación *Proyecta* entre los conjuntos de entidades *Cines* y *Películas*: [Figura 5.16]

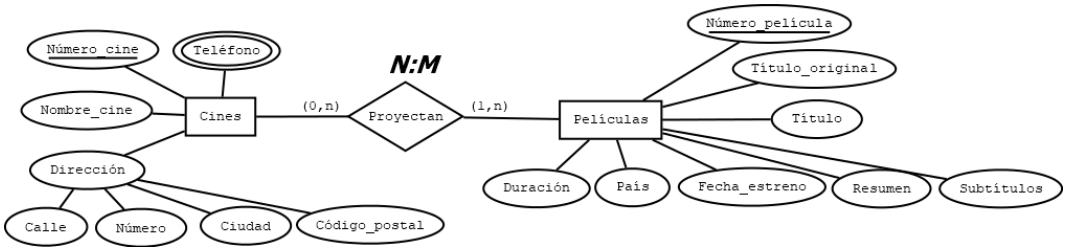

Figura 5.16 Relación Proyectan entre Cines y Películas

Si tenemos el *cine* con *Número_cine*:1, el cual proyecta las películas con *Número_película*: 1, 3, 5, 7, 8 y 9. Referenciamos dentro de dicho documento de cine los ObjectId de esas películas:

```
db.Cines.updateOne( { "Número_cine":1},
{$set:{"Películas":[ObjectId(Película1),ObjectId(Película3),ObjectId(Película5),Object
Id(Película7),ObjectId(Película8),ObjectId(Película9)]}})
```

Y del mismo modo referenciamos en cada una de las películas los cines en los que se proyectan. Para las películas con número de película 1 y 2:

```
db.Películas.updateOne( { "Número_película":1},
{$set:{"Cines":[ObjectId(Cine1),ObjectId(Cine5)]}})
db.Películas.updateOne( { "Número_película":2}, {$set:{"Cines":ObjectId(Cine4)}})
```

5.5.4.4 Relación entre Géneros y Películas.

La última de las relaciones muchos a muchos presentes en el diagrama ER es Pertenece la cual relaciona los conjuntos de entidades *Géneros* y *Películas*: [Figura 5.16].

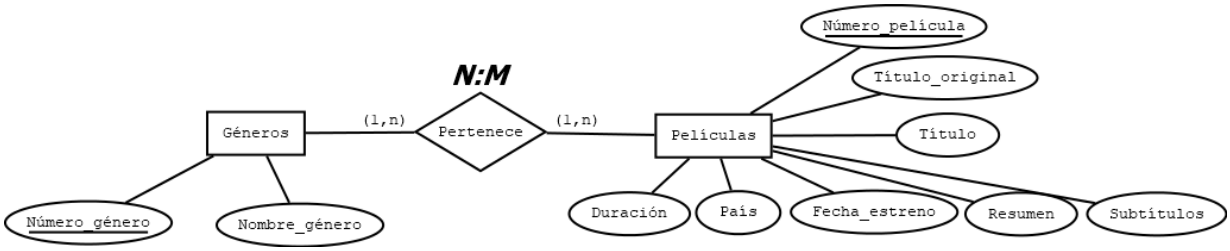

Figura 5.17 Relación Pertenece entre Géneros y Películas

Para el *Género* con *Número_género: 1* tenemos 5 Películas relacionadas con el mismo, aquellas con *Número_película: 2, 4, 5, 8 y 10*. Por lo que añadimos un nuevo campo en el documento número 1 de géneros con clave Películas y como valores los 5 ObjectID de esas Películas.

```
db.Géneros.updateOne( { "Número_género":1},
{$set:{"Películas":[ObjectId(Película2),ObjectId(Película4),ObjectId(Película5),Object
Id(Película8),ObjectId(Película10)]}})
```

De igual modo añadimos un campo con clave Géneros y valor el ObjectID del que pertenecen, en cada una de las 5 películas relacionadas (las cuales a su vez pueden pertenecer también a otros géneros distintos).

```
db.Películas.updateOne( { "Número_película":2},
{$set:{"Género":[ObjectId(Género1),ObjectId(Género2),ObjectId(Género3),ObjectId(Género
10)]}})
db.Películas.updateOne( { "Número_película":4},
{$set:{"Género":[ObjectId(Género1),ObjectId(Género3),ObjectId(Género10)]}})
db.Películas.updateOne( { "Número_película":5},
{$set:{"Género":[ObjectId(Género1),ObjectId(Género6),ObjectId(Género9)]}})
db.Películas.updateOne( { "Número_película":8},
{$set:{"Género":[ObjectId(Género1),ObjectId(Género2),ObjectId(Género3),ObjectId(Género
6),ObjectId(Género7),ObjectId(Género9)]}})
db.Películas.updateOne( { "Número_película":10},
{$set:{"Género":[ObjectId(Género1),ObjectId(Género4)]}})
```

5.5.5 Relaciones recursivas.

En el diagrama ER objeto de estudio solo nos encontramos una relación recursiva.

5.5.5.1 Relación precede entre películas.

En el diagrama proporcionado aparece una relación recursiva con nombre *Precede* en la entidad *Películas*. Los roles de la relación son *Precuela* y *Secuela*. [Figura 5.17]

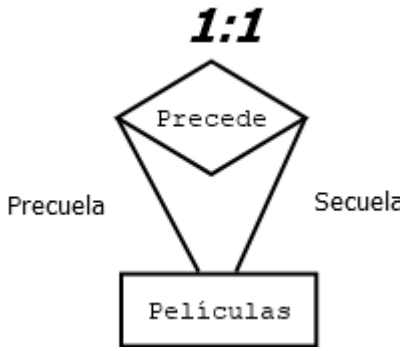

Figura 5.18 Relación Precede entre Películas

Aplicamos la regla de transformación definida en el punto [4.3.6] de esta memoria para el siguiente ejemplo de nuestra base de datos:

Tenemos la película con *Número_película: 1*, la cual tiene una *Secuela*: la película con *Número_película: 2*, la cual a su vez tiene secuela, aquella con *Número_película: 3*.

Añadimos en los documentos relacionados un nuevo campo con valor *Secuela* y clave el ObjectID de la película con la que está relacionada.

```
db.Películas.updateOne( { "Número_película":1},
{$set:{"Secuela":ObjectId(Película2)}})
db.Películas.updateOne( { "Número_película":2},
{$set:{"Secuela":ObjectId(Película3)}})
db.Películas.updateOne( { "Número_película":3},
{$set:{"Secuela":ObjectId(Película4)}})
```

De igual modo y por definición aquellas *Películas* que son poseen *Secuela*, son *Precuela* de estas últimas. Añadimos en las mismas un nuevo campo con clave *Precuela* y valor el ObjectID de la película con la que está relacionada.

```
db.Películas.updateOne( { "Número_película":2},
{$set:{"Precuela":ObjectId(Película1)}})
db.Películas.updateOne( { "Número_película":3},
{$set:{"Precuela":ObjectId(Película2)}})
```

5.6 Tabla resumen conversión.

Una vez realizadas todas las conversiones del diagrama Entidad-Relación propuesto en su equivalente en *MongoDB*, mostramos en las siguientes tablas un resumen de las mismas: [Tabla 5.1] y [Tabla 5.2]

Diagrama ER	**MongoDB**
Entidad *Películas*	Colección *Películas*
Entidad *Cines*	Colección *Cines*
Entidad *Directores*	Colección *Directores*
Entidad *Actores*	Colección *Actores*
Entidad *Géneros*	Colección *Géneros*
Entidad *Críticas*	Colección *Críticas*
Entidad *Críticos*	Colección *Críticos*
Jerarquía ISA *Medios*	Colección *Medios*
	Colección *Medios impresos*
	Colección *medios digitales*
Entidad débil *Citas*	Documento incrustado en colección *Películas*

Tabla 5.1 Resumen transformación entidades

Diagrama ER	**MongoDB**
Escriben_en	Ninguna o una referencia a documentos de la colección *Críticos* en documentos de la colección *Medios*
	Una referencia a documentos de la colección *Medios* en documentos de la colección *Críticos*
Escriben	Única referencia a documentos de la colección *Críticos* en documentos de la colección *Críticas*
	Una o más referencias a documentos de la colección *Críticas* en documentos de la colección *Críticos*
Reciben	Una referencia a documentos de la colección *Películas* en documentos de la colección *Críticas*
	Ninguna o un máximo de 100 referencias a documentos de la colección *Películas* en documentos de la colección *Críticas*
Interpretan	Una o varias referencias a documentos de la colección *Películas* en documentos de la colección *Actores*
	Una o varias referencias a documentos de la colección *Actores* en documentos de la colección *Películas*
Dirigen	Una o varias referencias a documentos de la colección *Películas* en documentos de la colección *Directores*
	Una o varias referencias a documentos de la colección *Directores* en documentos de la colección *Películas*
Proyectan	Ninguna o varias referencias a documentos de la colección *Cines* en documentos de la colección *Películas*
	Una o varias referencias a documentos de la colección *Películas* en documentos de la colección *Cines*
Pertenece	Una o varias referencias a documentos de la colección *Películas* en documentos de la colección *Géneros*
	Una o varias referencias a documentos de la colección *Géneros* en documentos de la colección *Películas*
Precede	Ninguna o una referencia a documentos de la colección *Películas* con clave *Secuela*
	Ninguna o una referencia a documentos de la colección *Películas* con clave *Precuela*

Tabla 5.2 Resumen transformación relaciones

Capítulo 6 Conclusiones y Trabajo Futuro.

6.1 Objetivos alcanzados.

A lo largo de este trabajo de fin de grado hemos estudiado de forma teórica y práctica cómo es posible transformar un esquema ER en una base de datos en *MongoDB* manteniendo las restricciones presentes en el diagrama origen.

Para conseguir lograrlo se han cumplido los siguientes hitos propuestos al inicio del mismo:

- Estudio pormenorizado de los elementos que forman un diagrama Entidad-Relación y las conexiones entre ellos.
- Estudio del funcionamiento de una base de datos *MongoDB*.
- Creación de una serie de reglas para transformar un diagrama ER en una base de datos con *MongoDB*.
- Ejemplos de cada una de ellas y recomendaciones de cuando utilizarlas en caso de existir varias aproximaciones.
- Transformación completa de un diagrama ER en una base de datos con *MongoDB*.

6.2 Conclusiones del trabajo

Una vez superados estos hitos hemos demostrado que es posible transformar un esquema o diagrama Entidad-Relación en una base de datos en *MongoDB* manteniendo las restricciones del esquema de partida.

Podemos afirmar que el uso de ambas técnicas a la hora de llevar a cabo el proceso de creación de una base de datos no es incompatible, siendo incluso recomendable si se desea tener un modelo conceptual a la hora de abordar su desarrollo. De esta forma tendríamos las ventajas de partir de un modelo gráfico relacional a la hora de diseñarla (con las ventajas que implica y hemos expuesto a lo largo de este trabajo) y todas las características adicionales de una base de datos no relacional (en este caso *MongoDB*) a la hora de implementarla y para futuras actualizaciones o modificaciones.

6.3 Experiencia personal.

A lo largo de la realización de este TFG se han afianzado conocimientos sobre materias previamente estudiadas y puestas en práctica, como son las bases de datos y en particular el esquema Entidad-Relación y el modelo relacional.

Se ha aprendido a utilizar una nueva tecnología: *MongoDB*, de la cual solo teníamos conceptos teóricos muy básicos.

El mayor reto al que nos hemos enfrentado ha sido a la hora de diseñar las reglas teóricas de transformación definidas en el [Capítulo 4].

A lo largo de todo nuestro periodo tanto académico como laboral, siempre hemos presente, sea de forma directa o indirecta, las bases del modelo relacional a la hora de manipular la información, tanto para crear bases de datos independientes como en el momento de planificar y diseñar aplicaciones de mayor envergadura.

Esto es debido a que nuestros primeros pasos en el mundo de la informática fueron marcados por el estudio del modelo relacional, así como de los conceptos que se apoyan en el (entre los que está el esquema Entidad-Relación) lo cual ocasiona que tengamos tan interiorizadas dichas ideas, que haya sido complicado abordar los mismos problemas, pero con un enfoque diferente.

A la hora de abordar el estudio teórico de la conversión de un diagrama cuya base está sustentada en una serie de restricciones de un modelo cuya funcionalidad apenas necesita de estas, ha sido bastante complejo intentar mantener la mayor cantidad de conceptos del diagrama ER de partida en la base de datos resultante de la transformación, decidiendo prescindir de algunos de ellos debido a la naturaleza de esta última. No por no estar familiarizados con esta nueva herramienta, que como comentamos a lo largo de este trabajo en varias ocasiones, es extremadamente flexible, sino por intentar integrar las restricciones del modelo relacional en una base de datos no relacional.

La documentación existente de *MongoDB*, sobre todo en lo referente a las relaciones entre los datos es extremadamente escasa. Desconocemos el porqué de ello, aun siendo una de las bases de datos más utilizadas en el mundo (y la más utilizada de entre las NoSQL). Puede ser debido a la extremada flexibilidad que propone o simplemente no desean condicionar a los desarrolladores, pero el hecho es que a pesar de llevar varios años instaurada y con frecuentes actualizaciones la documentación sigue siendo muy pobre.

Otra de las dificultades ha sido trasladar de la manera más precisa posible el trabajo realizado en esta memoria. Este es un TFG con un gran porcentaje de parte teórica y resultó bastante complejo resumir toda la información que se ha deseado expresar en un número limitado de páginas, pero intentando transmitir, de la mejor manera posible, el trabajo realizado y las reglas de transformación obtenidas tras múltiples abordamientos diferentes a dicho tema. Todo ello con un lenguaje cuya finalidad ha sido la de intentar ser lo más preciso y homogéneo posible para todo el documento.

6.4 Trabajo futuro.

Dada una base de datos en *MongoDB* creada a partir de las reglas definidas en el presente estudio sería posible obtener el diagrama Entidad-Relación con el que se obtuvo la misma si se tienen los suficientes datos (colecciones, documentos y relaciones) como para abarcar todos los elementos presentes en el diagrama Entidad-Relación de partida. No hemos llevado a cabo dicha conversión inversa, pero creemos que sería interesante realizar el proceso inverso al desarrollado en este TFG (utilizando el presente como base).

Hemos abordado la transformación de un esquema ER en una base de datos con *MongoDB*. Existen otros tipos de bases de datos NoSQL y no basadas en documentos como es el caso de estudio que poco a poco se van imponiendo en el mercado, como pueden ser *Redis*, una base de datos basada en clave valor o *Neo4j*, una base de datos orientada a grafos.

Buscando información referente a este tema hemos encontrado que *Neo4j* (una base de datos orientada a grafos) permite visualizar un esquema de la base de datos creado con esta tecnología mediante el método predefinido *db.Schema()*[21] De esta forma se podría, de forma gráfica, ver la información que este contenida en esta base de datos, así como las relaciones entre sus instancias, por lo que creemos que sería posible realizar el proceso inverso: con la información de un diagrama Entidad-Relación podríamos crear un esquema perteneciente a *Neo4j* y, a partir de ahí crear y poblar una base de datos.

Por su parte Redis (siglas de **RE**mote **DI**ctionary **S**erver) es una base de datos basada en almacenamiento clave valor utilizada principalmente como una base de datos de respuesta rápida debido a que almacena los datos en memoria.[22] En Redis las relaciones se representan por *sets*, los cuales son colecciones de Strings únicos (a los que denomina miembros), mediante los cuales se pueden realizar operaciones de intersección y unión según su propia documentación.[23]

Conociendo esta información creemos que también sería posible la transformación de un esquema Entidad-Relación en una base de datos basada en clave valor mediante el profundo estudio de esta última como hemos hecho a lo largo de este trabajo con una base de datos orientada a documentos.

Es por ello que en un futuro sería interesante valorar el estudio de las reglas de transformación de un diagrama ER en bases de datos NoSQL no basadas en documentos.

[21] Noe4j Docs "Defining an Schema" [Recurso Online] Disponible en: https://neo4j.com/docs/getting-started/cypher-intro/schema/ Fecha última consulta: 01/06/2024

[22] Ibm Docs "Qué es Redis" [Recurso Online] Disponible en: https://www.ibm.com/es-es/topics/redis Fecha última consulta: 01/06/2024

[23] Redis Docs "Sets in Redis" [Recurso Online] Disponible en: https://redis.io/docs/data-types/sets/ Fecha última consulta: 01/06/2024

Referencias.

1. "Relational Migrator" [Recurso Online] Disponible en: https://www.mongodb.com/docs/relational-migrator/
2. "The entity-relationship model: towards a unified view of data". Peter Pín-Shan Chen – March 1977.
3. "¿Qué es cardinalidad de una relación?" [Recurso Online] Disponible en: https://www.aulapc.es/lupa_busquedas_posit.html1accesA~A60.00
4. "Modelo Entidad-Relación" – Atlantic International University [Recurso Online] Disponible en: https://cursos.aiu.edu/Base%20de%20Datos/pdf/Tema%203.pdf
5. "Extending the database relational model to capture more meaning" – E.F.Cood (Autor) 1979
6. "MongoDB Schemaless" [Recurso Online] Disponible en: https://www.mongodb.com/unstructured-data/schemaless
7. "MongoDB features" [Recurso Online] Disponible en: https://www.mongodb.com/features
8. "Replicación (informática)" [Recurso Online] Disponible en: https://es.wikipedia.org/wiki/Replicaci%C3%B3n_(inform%C3%A1tica)#:~:text=La%20replicaci%C3%B3n%20es%20el%20proceso,sirvientes%20o%20esclavos%20(slaves)
9. "GridFS Use" [Recurso Online] Disponible en: https://www.mongodb.com/docs/manual/core/gridfs/
10. "Bson specs" [Recurso Online] Disponible en: https://bsonspec.org/#/specification
11. "MongoDB ObjectID" [Recurso Online] Disponible en: https://www.mongodb.com/docs/manual/reference/method/ObjectId/
12. "Primary key – MongoDB MondoDB Glossary" [Recurso Online] Disponible en: https://www.mongodb.com/docs/manual/reference/glossary/#std-term-primary-key
13. "Mongodb primary key using the default objectid vs picking a custom value" [Recurso Online] Disponible en:
14. https://medium.com/@rishabh011/mongodb-primary-key-using-the-default-objectid-vs-picking-a-custom-value-12399d828e33
15. "Schema design anti pattern massice arrays" [Recurso online] Disponible en: https://www.mongodb.com/developer/products/mongodb/schema-design-anti-pattern-massive-arrays/
16. "toString in MongoDB" [Recurso online] Disponible en: https://www.mongodb.com/docs/manual/reference/operator/aggregation/toString/
17. "Fundamentos de bases de datos" Ramez A Elmasri (Autor) Shamkant B. Navathe (Autor)
18. "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems" Martin Kleppmann (Autor)
19. "Adaptabase. Adaptative machine learning based database croos- technology selection - Shay Horovitz, Alon Ben-Lavi, Refael Auerbach, Bar Brownshtein" Shay Horovitz, Alon Ben-Lavi (Autor), Refael Auerbach (Autor), Bar Brownshtein (Autor).
20. "Adaptabase. Adaptative machine learning based database croos- technology selection - Shay Horovitz", Alon Ben-Lavi (Autor), Refael Auerbach (Autor), Bar Brownshtein

(Autor) - School of Computer Science, College of Management Academic Studies. 2018
21. “Correlation and comparison of NoSWL specimen with Relational data store” Sangeeta Gupta G. Narsimha (Autor). IJRET: International Journal of Research in Engineering and Technology. 2015
22. “RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's” Rabi Prasad Padhy (Autor), Manas Ranjan (Autor), Patra Suresh (Autor),Chandra Satapathy (Autor) 2011
23. “Techniques for Converting Big Data from SQL to NoSQL Databases” Md. Saiful Islam (Autor). 2016
24. “Apuntes de la asignatura bases de datos” Dra. M. Mercedes Martínez González (Autora)
25. “Apuntes de la asignatura bases de datos” Carmen Hernández Díez (Autora)
26. **“**Ranking DB” [Recurso Online**]** Disponible en: https://db-engines.com/en/ranking
27. “Ranking bases de datos” [Recurso Online] Disponible en: https://db-engines.com/en/ranking
28. “Synthetic foreign key – MongoDB Relational migrator” [Recurso Online] Disponible en: https://www.mongodb.com/docs/relational-migrator/mapping-rules/synthetic-foreign-key/synthetic-foreign-keys/
29. “Defining an Schema” [Recurso Online] Disponible en: https://neo4j.com/docs/getting-started/cypher-intro/schema/
30. “Qué es Redis” [Recurso Online] Disponible en: https://www.ibm.com/es-es/topics/redis
31. “Sets in Redis” [Recurso Online] Disponible en: https://redis.io/docs/data-types/sets/
32. “Página web Filmaffinity” [Recurso Online] Disponible en: https://www.filmaffinity.com/es/main.html

Anexo A. Tablas con relaciones existentes entre las instancias.

Medios Número_medio	Relación Escribe_en	Críticos Número_crítico
1	-	17
3	-	15
4	-	13
5	-	12
6	-	11
7	-	10
8	-	9
9	-	8
10	-	7
11	-	6
12	-	5
13	-	4
14	-	3
15	-	2
16	-	1

Tabla Anexo.1: Relaciones entre Medios y Críticos

Críticos Número_crítico	Relación Escriben	Críticas Número_crítica
1	-	1
3	-	2,15,20
4	-	3,8,12,14,24
5	-	4,7,9,16
6	-	6
7	-	-
8	-	10
9	-	11
10	-	11,23
11	-	17
12	-	18

13	-	21
14	-	22
15	-	25
16	-	1

Tabla Anexo 2: Relaciones entre Críticos y Críticas.

Películas Número_película	**Relación** Dirigen	**Director** Número_director
1	-	4
2	-	6
3	-	8
4	-	10
5	-	8
6	-	8
7	-	2
8	-	9
9	-	9
10	-	10
11	-	7,11
12	-	3
13	-	3
14	-	5
15	-	5

Tabla Anexo 3: Relaciones entre Películas y Directores.

Películas Número_película	**Relación** Reciben	**Críticas** Número_crítica
1	-	4,5,6
2	-	7,8
3	-	
4	-	9,10,11
5	-	12,13
6	-	14,15
7	-	6
8	-	7

9	-	8
10	-	9
11	-	10
12	-	11
13	-	3
14	-	3
15	-	5
16	-	5

Tabla Anexo 4: Relaciones entre Películas y Críticas.

Películas Número_película	**Relación** Pertenecen	**Géneros** Número_género
1	-	3,10
2	-	1,2,3,10
3	-	2,3,10
4	-	1,3,10
5	-	1,6,9
6	-	6,9,10
7	-	6,9,10
8	-	1,2,3,6,7,9
9	-	8,6
10	-	1,4
11	-	5
12	-	6,9
13	-	6,9
14	-	3
15	-	6,9

Tabla Anexo 5: Relaciones entre Películas y Géneros.

Películas Número_película	**Relación** Interpretan	**Actores** Número_actor
1	-	11,10
2	-	11,13
3	-	11,14
4	-	11,15

5	-	7,16
6	-	6,9,10
7	-	17,18
8	-	19,20
9	-	8,9
10	-	1,2,3
11	-	21,22
12	-	21,22
13	-	21,22
14	-	23,24
15	-	25,26

Tabla Anexo 6: Relaciones entre Películas y Actores.

Películas Número_película	**Relación** Proyectan	**Cines** Número_cine
1	-	1,5
2	-	4
3	-	1.4
4	-	4
5	-	1
6	-	-
7	-	1
8	-	1,5
9	-	1
10	-	3,4
11	-	3
12	-	3,5
13	-	2
14	-	2,4
15	-	2,5

Tabla Anexo 7: Relaciones entre Películas y Cines.

Películas Número_película	**Relación** Precuela	**Películas** Número_película
2	-	1

3	-	2
4	-	3
4	-	4
13	-	12

Tabla Anexo 8: Relación precuela entre Películas.

Películas Número_película	**Relación** Secuela	**Películas** Número_película
1	-	2
2	-	3
3	-	4
12	-	13

Tabla Anexo 9: Relación secuela entre Películas.

Medios Número_medio	**Relación** ISA	**Medio digital or Medio tradicional**
1	-	Medio digital
3	-	Medio digital
4	-	Medio digital
5	-	Medio impreso
6	-	Medio digital
7	-	Medio digital
8	-	Medio digital
9	-	Medio digital
10	-	Medio digital
11	-	Medio digital
12	-	Medio impreso
13	-	Medio digital
14	-	Medio digital
15	-	Medio impreso
16	-	Medio digital

Tabla Anexo 10: Relación ISA entre Medios.

Anexo B.

Enlaces adicionales.

Los enlaces útiles de interés para este Trabajo Fin de Grado son:

- Repositorio del código utilizado para creación de los ejemplos del Capítulo 4: https://github.com/rubdedi/MongoDB/blob/main/TfgCap%C3%ADtulo4.js
- Repositorio del código utilizado para la transformación completa del diagrama ER del Capítulo 5: https://github.com/rubdedi/MongoDB/blob/main/TfgCap%C3%ADtulo5.js