29-10-2020

PROYECTO FINAL

CREACIÓN DE UNA BASE DE DATOS

FERNÁNDEZ HERNÁNDEZ, ALBERTO
UNIVERSIDAD COMPLUTENSE DE MADRID

Índice

1. MODELO ENTIDAD-RELACION

Nota: para la explicación del modelo Entidad-Relación, cada sección del diagrama se explicará a partir de fragmentos del texto de la práctica. El diseño tanto del modelo Entidad-Relación como del Modelo relacional ha sido llevado a cabo mediante la herramienta de diseño **draw.io**[1]

TIENDA

De la tienda sabemos el nombre que es distinto para cada tienda, quien la gestiona (Android, Apple, Amazon, ….) y dirección web.

Nos encontramos con la primera entidad: **Tienda**, formada por:

- Nombre de la tienda (dado que es único será la **clave primaria**).
    - Dominio: reducido al conjunto de tiendas existentes (cadena de caracteres de longitud variable):
        - App Store, AppStore, Google Play Store, AppWorld, MarketPlace, OVITienda, AppCatalog.
- Gestor de la tienda.
    - Dominio: reducido al conjunto de gestores de cada tienda (cadena de caracteres de longitud variable):
        - Apple Inc., Amazon.com Inc., Google LLC., Blackberry Limited, Microsoft Corp., Nokia Corporation, Hewlett-Packard.
- Dirección web de la tienda física.
    - Dominio: reducido a la dirección web de cada tienda, en formato URL (http, https), mediante una cadena de caracteres de longitud variable.

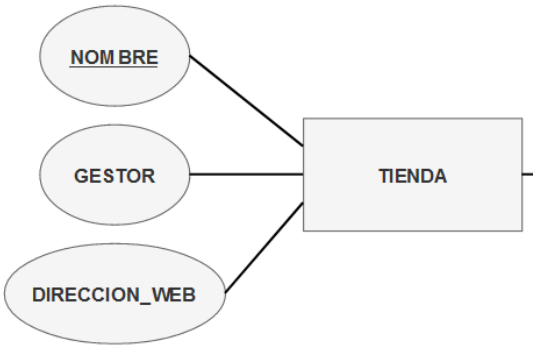

Ilustración 1. Entidad Tienda

EMPRESA

De las empresas que realizan las aplicaciones (apps), conocemos su nombre, país en el que paga sus impuestos, año de creación, correo electrónico y pagina web.

Por otro lado, la siguiente entidad, **Empresa**. En este caso nos encontramos ante un problema ¿Cuál sería la clave primaria más adecuada? Una posibilidad sería escoger el campo **nombre**, ya que el nombre de una empresa, en caso de estar registrada, es único. Sin embargo, puede ocurrir que el nombre de dos empresas coincida en caso de que el nombre no esté registrado y/o patentado. Para evitar inconvenientes, se ha añadido un campo adicional: **VAT**, un número de identificación fiscal que deben tener todas las empresas que deseen realizar operaciones a nivel europeo, único para cada entidad, por lo que evitamos

[1] https://app.diagrams.net/

con ello posibles **coincidencias** con el nombre de la empresa[2]. Por tanto, la entidad **Empresa** estará conformada por los siguientes campos:

- VAT (**clave primaria**).
- Nombre.
    - Dominio: reducido al conjunto de posibles nombres de una empresa (estén o no patentados), mediante una cadena de caracteres de longitud variable.
- País tributario.
    - Dominio: reducido al conjunto de países existentes mediante una cadena de caracteres de longitud variable (España, Francia, Gran Bretaña, Irlanda), por lo que quedan excluidos los nombres propios de personas.
- Año de creación.
    - Dominio: reducido al conjunto de años positivos, a partir del año1900.
- Correo electrónico.
    - Domino: reducido a una cadena de caracteres de longitud variable (única para cada empresa) con la característica extensión de correo (@gmail, @outlook, @yahoo etc.)
- Página web.
    - Dominio: correspondiente a una cadena de caracteres variable con el formato de una URL, al igual que ocurría con la página web de la tienda (http, https).

El dominio del campo **VAT** se reducirá a una cadena de caracteres de longitud variable hasta 12 caracteres como máximo estipulado por ley, dado que el código de cada país europeo es de diferente formato y longitud. A modo de ejemplo:

- **España**: ES12345678 (Uno o dos caracteres seguido de ocho dígitos).
- **Alemania**: DE123456789 (Siglas del país más nueve dígitos).
- **Países bajos:** 123456789B01 (12 caracteres. El décimo siempre es la letra B).
- **Hungría**: HU12345678 (Siglas del país más ocho dígitos).

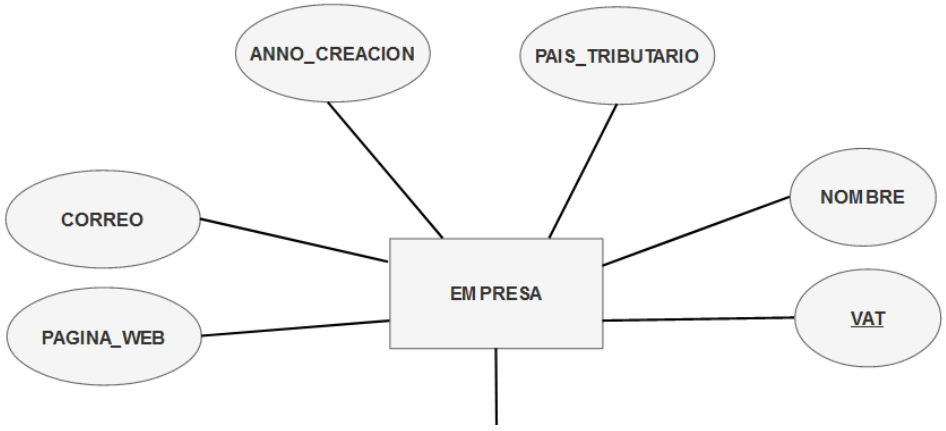

Ilustración 2. Entidad Empresa

EMPLEADO

En la empresa hay empleados, pero debido al dinamismo en estos tipos de trabajo y, a la oferta y la demanda en el sector, el empleado puede haber trabajado en varias empresas del sector e incluso puede trabajar en la misma empresa en distintos periodos de tiempo, nos interesa conocer la experiencia profesional del empleado. Además, del empleado nos interesa el DNI, dirección (calle, número, código postal), correo electrónico y teléfono.

[2] https://www.gov.uk/guidance/vat-eu-country-codes-vat-numbers-and-vat-in-other-languages

A continuación, la siguiente entidad a diseñar, **Empleado**, se compone de los siguientes atributos:

- DNI. Dado que es único para cada persona, será la **clave primaria** de la entidad.
    - Dominio: se reduce a una cadena de longitud 9, compuesto por 8 dígitos más letra.
- Dirección, un atributo **compuesto** formado por los campos calle, número y código postal. En relación con el número de calle, puede tratarse de un atributo nulo (calles sin un número asociado), por lo que deberá tenerse en cuenta este aspecto en el diseño del modelo relacional.
    - Dominio (Calle): se reduce a una cadena de caracteres de longitud variable, incluyendo los tipos de vía (Calle, Avenida, Plaza, Rotonda, Paseo, Carretera etc.)
    - Dominio (Número): para evitar valores nulos en el modelo de tablas, su dominio queda reducido a una cadena de caracteres de longitud 3 (máxima), de forma que, en caso de que el empleado resida en una calle sin número, se marque como 's/n' **por defecto** en lugar de NULL.
    - Dominio (Código Postal): reducido a un conjunto de caracteres de longitud 5 (tamaño del código postal en España).
- Correo electrónico, un campo candidato a ser clave primaria (dado que es único para cada empleado dentro de una empresa), por lo que también deberá ser único.
    - Dominio: se reduce a una cadena de caracteres de longitud variable (único para cada empleado) con la característica extensión de correo (@).
- Teléfono, un atributo numérico **multivalorado**, pudiendo ser el teléfono fijo o el teléfono móvil.
    - Dominio: reducido a un entero positivo. Dado que el formato de teléfono es diferente en cada país, no se reduce a un intervalo específico.

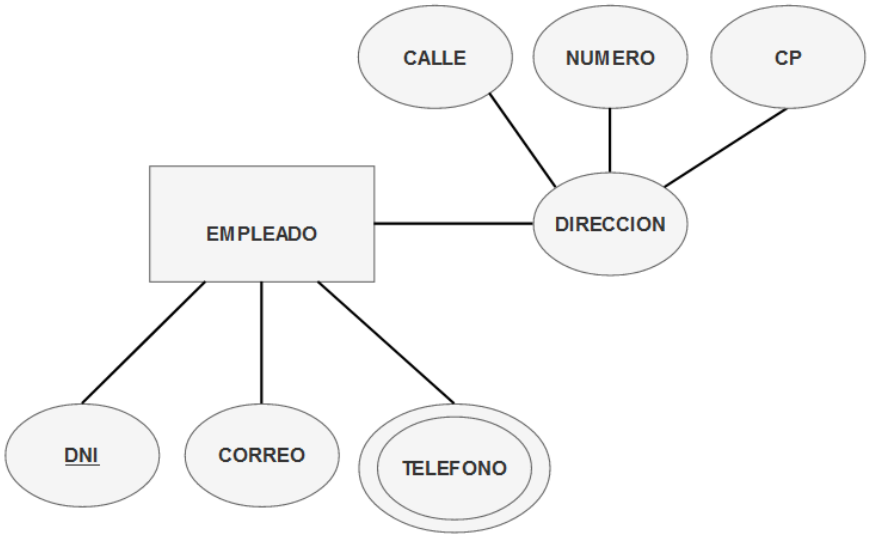

Ilustración 3. Tabla empleado

Una vez diseñada tanto la entidad Empresa como la entidad Empleado, deben unirse por medio de relaciones. Tal y como asegura el enunciado, **un empleado puede trabajar en varias empresas**, tanto en diferentes periodos de tiempo como incluso, aunque no lo indique el enunciado, trabajar en más de una empresa a la vez (dado que en la mayoría de los países es posible el pluriempleo siempre y cuando no supere el número de horas diarias/semanales estipulado por ley).

Por otra parte, en una empresa pueden trabajar uno o varios empleados: un empleado debe haber trabajado en 1 o N empresas (mínimo y máximo), así como en una empresa debe trabajar como mínimo un empleado y máximo N, dado que no tendría sentido decir que en una empresa trabajan cero empleados. Por tanto, la cardinalidad de la relación será **N:N**, tal y como se muestra a continuación:

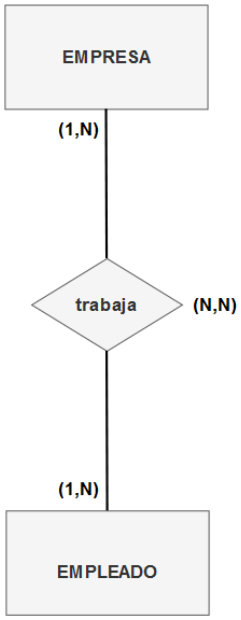

Ilustración 4. Relación trabaja (I)

Sin embargo, nos falta un detalle: **un empleado puede trabajar en una misma empresa en diferentes periodos de tiempo**. La relación anterior, en un modelo de tablas, no lo permitiría, ya que no se tiene constancia de las diferentes fechas en las que ha trabajado el empleado, por lo que faltaría añadir dos nuevos campos a la relación N:N: la **fecha de inicio** y la **fecha de fin** (ambos de tipo fecha) en el que un empleado trabajó en una empresa. Sin embargo, tal y como se verá en el modelo relacional, no bastará con asociar como clave primaria en la tabla N:N a las parejas codigo_empresa, DNI_empleado:

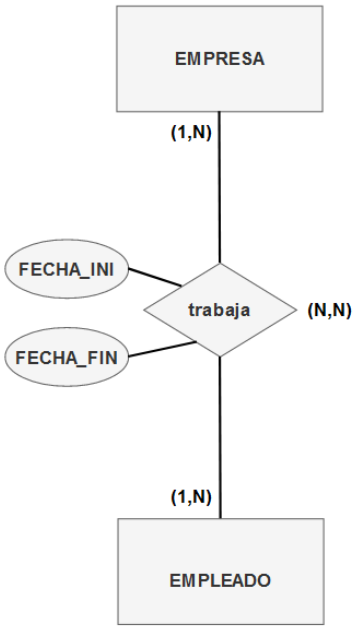

Ilustración 5. Relación trabaja (II)

APLICACION

En la empresa se realizan aplicaciones, de las que conocemos su nombre que es único, el código de aplicación, la fecha en la que se comenzó a realizar y fecha de terminación, la categoría o categorías en las que se puede incluir (entretenimiento, social, educación, ...), espacio de memoria, precio.

Partiendo de lo leído, la siguiente entidad, **Aplicación**, se compone de los siguientes campos:

- Nombre de la aplicación. Dado que es único, será la **clave primaria** de la entidad.
  - Dominio: reducido a una cadena de caracteres de longitud variable.
- Código de la aplicación, un campo único, de cara al modelo relacional.
  - Dominio: para facilitar la posterior inserción de datos, su dominio queda reducido a un conjunto de valores enteros positivos (ejemplo: 61768).
- La fecha de inicio de desarrollo de la aplicación.
- La fecha de fin de desarrollo de la aplicación.
  - Dominio: reducido a un campo fecha (en formato año/mes/dia).
- La/s categoría/s de la aplicación. Dado que una aplicación puede pertenecer a varias categorías se trata de un atributo **multivalorado**.
  - Dominio: reducido a una cadena de caracteres de longitud variable con los tipos de categorías más comunes en aplicaciones (entretenimiento, social, educativo, belleza, arte, fotografía, automoción, compras, libros, empresas, finanzas, estilo de vida, casa y hogar)
- El espacio de memoria.
  - Dominio: deberá tenerse en cuenta que debe ser estrictamente mayor que cero (no existen aplicaciones que pesen 0 MB), reduciéndose a un valor decimal mayor a cero.
- El precio de la aplicación. Dado que puede haber aplicaciones gratuitas, el valor por defecto será cero.
  - Dominio: al igual que el espacio de memoria, se reduce a un valor decimal positivo, aunque en el caso del precio puede ser cero.

Por otro lado, cabe destacar que la entidad **Aplicación** no es una entidad débil dado que, pese a que necesita del DNI del responsable (como se verá a continuación en el modelo relacional), su existencia no depende exclusivamente del empleado. A modo de ejemplo, el responsable que dirigió la aplicación Twitter puede dejar la empresa en la que trabajó y pese a ello la aplicación seguiría estando disponible a los usuarios, es decir, porque el empleado desaparezca de la base de datos no implica que la aplicación deje también de existir:

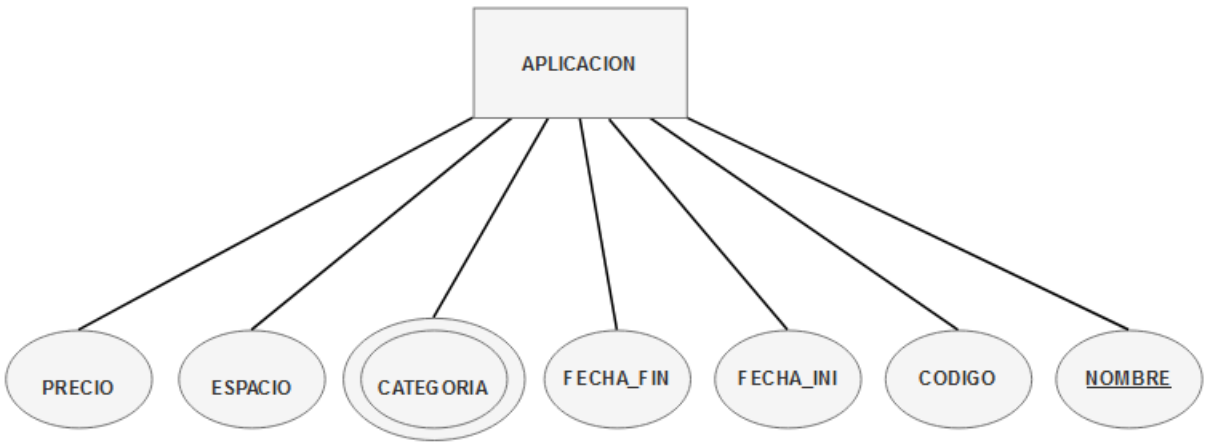

Ilustración 6. Entidad aplicación

Las aplicaciones son subidas a las tiendas o plataformas. Una misma aplicación puede ser subida a varias tiendas, por supuesto una tienda tiene muchas aplicaciones.

Como mínimo, una tienda puede contener una única aplicación (para ser considerada una tienda) y, como máximo, N. Por otro lado, una aplicación puede estar alojada en 1 o N tiendas como mínimo y máximo, respectivamente, dado que para estar disponible al público debe haber sido subida al menos a una tienda, e incluso pueden tratarse de aplicaciones nativas, es decir, que solo estén disponibles para un tipo de móvil en particular. Por tanto, la cardinalidad será N:N.

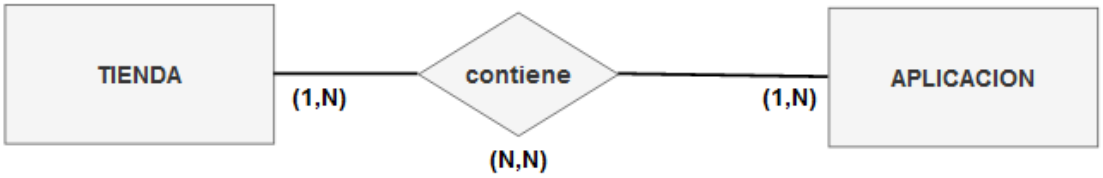

Ilustración 7. Relación tienda – aplicación

Cada aplicación está realizada por un grupo de empleados y un empleado es el jefe o responsable de cada aplicación. Un empleado puede dirigir varias aplicaciones.

Entre las entidades **Aplicación** y **Empleado**, nos encontramos con dos relaciones:

- **realiza**: un empleado puede realizar 1 o N aplicaciones (es decir, puede formar parte de varios proyectos al mismo tiempo, del mismo modo que se indicó que un empleado puede trabajar en más de una empresa a la vez). A su vez, una aplicación es realizada por 1 o N empleados, de forma que la cardinalidad resultante es N:N
- **dirige**: un empleado puede dirigir, ser jefe de proyecto, de 0 o N aplicaciones (puede o no dirigir una aplicación). Sin embargo, una aplicación es dirigida por uno y solo un empleado, por lo que la cardinalidad entre ambas entidades será 1:N, es decir, de cara al modelo relacional se traducirá en la presencia del DNI del responsable de la aplicación como clave foránea.

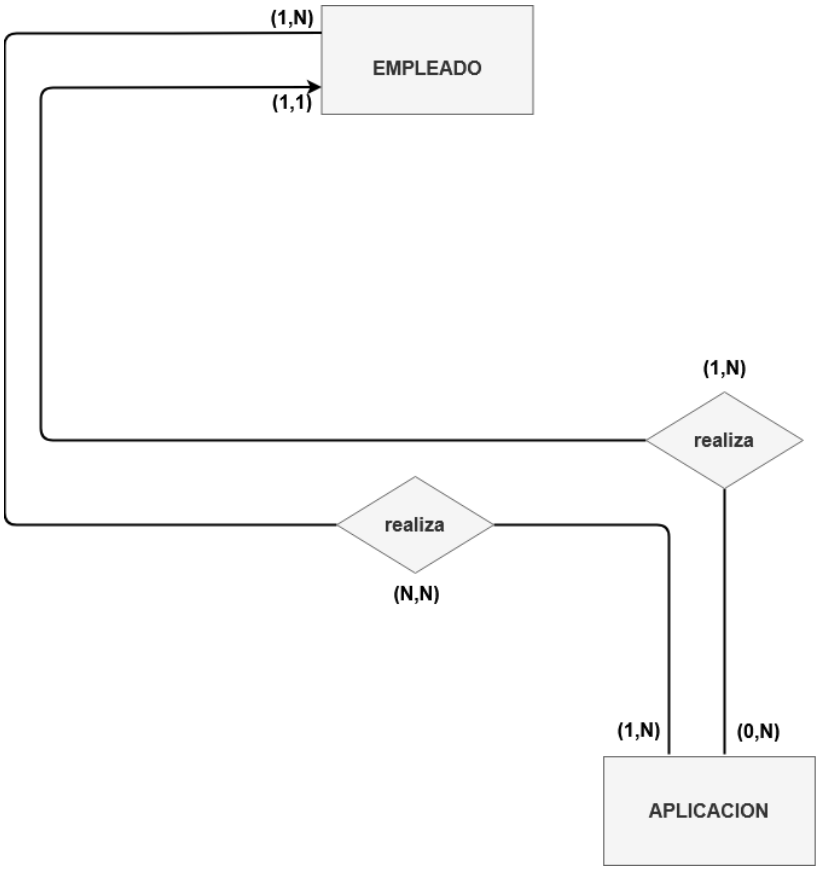

Ilustración 8. Relaciones realiza, dirige

A continuación, debemos preguntarnos ¿Realmente es necesario crear una relación entre **Empresa** y **Aplicación**? Es decir, ya están relacionadas de forma indirecta gracias a la entidad Empleado. Sin embargo, esto no es del todo cierto. A modo de ejemplo, supongamos una empresa ' A ' en la que trabajó un empleado entre el 01/01 y el 31/05. Un día más tarde, pasó a trabajar a otra empresa ' B ' entre el 01/06 y el 31/12. Dicho empleado trabajó en una aplicación entre el 02/06 y el 31/12. Con el modelo actual, al consultar las tablas Empresa, Empleado y Aplicación, la consulta nos indicaría que la empresa ' A ' formó parte del desarrollo de dicha aplicación (junto con la empresa ' B '), debido a que el empleado trabajó en algún momento en esta empresa, aunque fuese antes de que comenzase el proyecto. Este problema es debido a que, como eje intermedio en la consulta, estamos utilizando el DNI del empleado que, como se ha indicado, puede trabajar en varias empresas, asociando erróneamente el desarrollo de una aplicación a una empresa. Para facilitar este tipo de consultas (aplicaciones realizadas por una empresa), merece la pena disponer de una relación adicional (**crea**) que relacione a las empresas y aplicaciones.

Bien es cierto que puede tratarse de información redundante, pero de este modo permite relacionar a las empresas con las aplicaciones que hayan realizado, de forma directa y sin necesidad de pasar por los empleados.

De este modo, y al igual que **realiza**, una empresa puede crear 1 o N aplicaciones (mínimo y máximo), dado que pueden colaborar varias empresas en el desarrollo; de la misma forma que una aplicación es creada por 1 o N empresas:

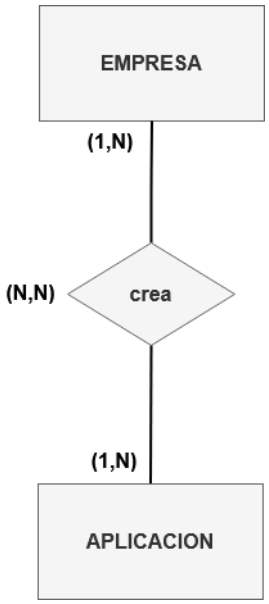

Ilustración 9. Relación crea

USUARIO

Un usuario puede descargar o no aplicaciones, pero no puede descargar dos veces la misma aplicación. El usuario puede puntuar de 0 a 5 cada una de las aplicaciones que se descarga y hacer comentarios referentes a la misma. Del usuario conocemos el número de cuenta que es único, nombre, dirección y si se descarga la aplicación en el teléfono conocemos el número de móvil.

Además, nos interesa saber la fecha en la que se realizan más descargas y el país de los usuarios que más aplicaciones se han descargado.

En el siguiente fragmento nos encontramos con una nueva entidad: **Usuario**, formada por los siguientes campos:

- Número cuenta: dado que es único, será la **clave primaria** de la entidad.
    - Dominio: reducido a un conjunto de valores enteros positivos (mayores o iguales a cero).
- Nombre del usuario, un campo único, de cara al modelo relacional.
    - Dominio: reducido a una cadena de caracteres de longitud variable, incluyendo tanto letras como números.
- Dirección que, al igual que en la entidad Empleado, se trata de un campo **compuesto** por calle, número y código postal. Sigue el mismo dominio que en la entidad Empleado.
- País del usuario. Dado que lo que se quiere conocer es el país de origen de los usuarios y no el país donde se realicen más descargas, el campo país se ha colocado directamente sobre la entidad Usuario, no en la relación.
    - Dominio: reducido al conjunto de países existentes mediante una cadena de caracteres de longitud variable (España, Francia, Gran Bretaña, Irlanda), por lo que quedan excluidos los nombres propios de personas.

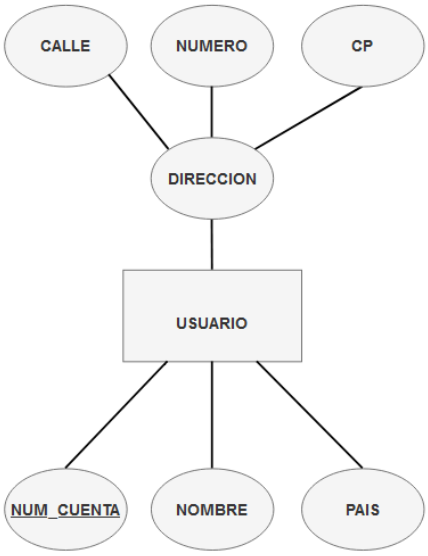

Ilustración 10. Entidad usuario

Por otro lado, un usuario puede descargarse, como mínimo, ninguna aplicación y, como máximo, N aplicaciones, del mismo modo que una aplicación puede ser descargada 0 o N veces, como mínimo y máximo, respectivamente. Por tanto la relación **descarga** tendrá cardinalidad N:N. No obstante, dado que un usuario puede puntuar la aplicación descargada, realizar comentarios sobre la misma, incluir su número de móvil en caso de descarga así como la fecha en la que se realizó, sobre la relación N:N deben añadirse dichos campos. De cara al modelo relacional debe tenerse en cuenta que, tanto la puntuación como el comentario, pueden estar vacíos, así como la puntuación debe estar comprendida entre 0 y 5 puntos.

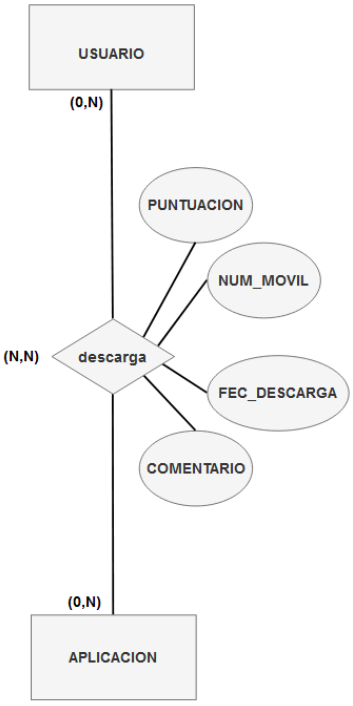

Ilustración 11. Relación descarga

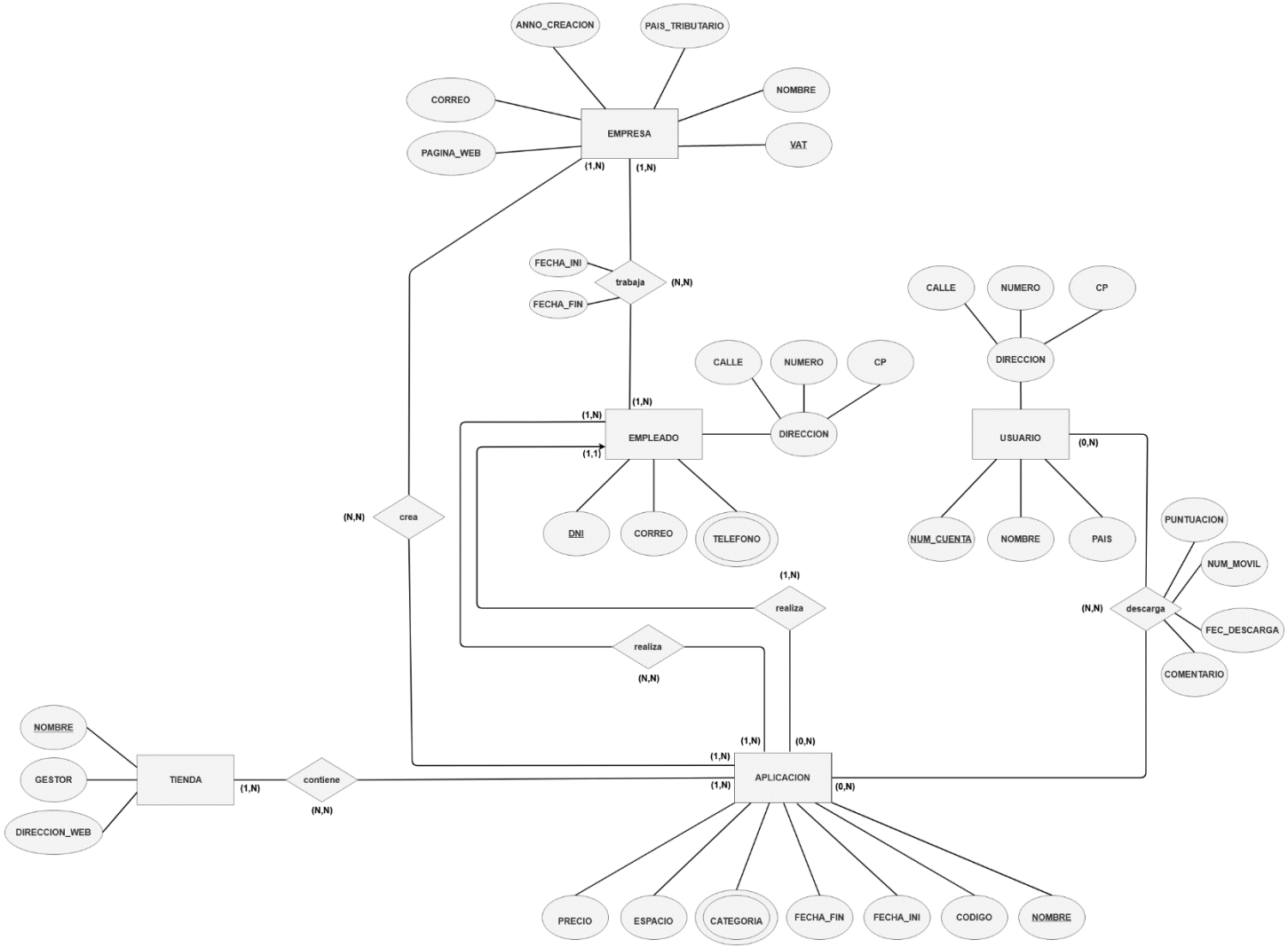

Ilustración 12. Modelo Entidad-Relación final

UNIVERSIDAD COMPLUTENSE MADRID

2. MODELO RELACIONAL

Una vez diseñado el modelo Entidad-Relación, realizamos su paso al modelo de tablas.

TIENDA

- Nombre: dado que es único, será la clave primaria de la tabla, definiendo un *Varchar* de tamaño medio (20 caracteres).
- **Dirección web**: dado que la dirección a la página web de cada tienda es única, la marcamos como clave candidata (**subrayada en negrita**), definiendo un *Varchar* de mayor tamaño (80 caracteres).
- Por último, el campo Gestor. Dado que varias tiendas pueden ser gestionadas por una misma empresa en un futuro (en caso de ser adquiridas), no se trata de una clave candidata, definiendo con ello un *Varchar* de tamaño medio (20 caracteres).

En cualquiera de los tres campos, **ninguno de ellos puede ser NULL**, dado que debe disponer obligatoriamente de una empresa que la gestione, así como un sitio web.

TIENDA	
PK	NOMBRE VARCHAR(20) NOT NULL
	DIRECCION_WEB VARCHAR(80) UNIQUE NOT NULL
	GESTOR VARCHAR(20) NOT NULL

Ilustración 13. Tabla tienda

EMPRESA

Según lo definido anteriormente, la entidad Empresa se compone de los siguientes campos:

- VAT, la clave primaria de la tabla, empleando para ello un *Varchar* de 12 caracteres como máximo (máxima longitud estipulada).
- **Correo**: un *Varchar* de mayor longitud (50 caracteres). Dado que el nombre de correo es único para cada empresa, lo marcamos como clave candidata (UNIQUE).
- **Página web**: un *Varchar* de 80 caracteres (al igual que en la tabla Tienda). Se trata de un campo único para cada empresa (pues la dirección de cada página es única), por lo que lo marcamos como clave candidata (UNIQUE).
- Nombre de la empresa: dado que varias empresas pueden presentar un mismo nombre, no puede tratarse de una clave candidata, empleando para ello un *Varchar* de tamaño medio (50 caracteres).
- País tributario, un *Varchar* de tamaño medio (35 caracteres).
- Año de creación de la empresa, un entero **sin signo**, dado que el año no puede ser negativo.

EMPRESA	
PK	VAT VARCHAR(12) NOT NULL
	CORREO VARCHAR(50) UNIQUE NOT NULL
	PAGINA_WEB VARCHAR(80) UNIQUE NOT NULL
	NOMBRE VARCHAR(50) NOT NULL
	PAIS_TRIBUTARIO VARCHAR(35) NOT NULL
	ANNO_CREACION UNSIGNED INT NOT NULL

Ilustración 14. Tabla empresa

Todos y cada uno de los campos anteriores se consideran esenciales y de carácter obligatorio con el fin de obtener una información completa de la empresa, por lo que **ninguno de ellos puede ser NULL**.

EMPLEADO

Según lo definido en el modelo Entidad-Relación, la tabla Empleado se compone de los siguientes atributos:

- DNI: la clave primaria de la tabla, empleando para ello una cadena fija de caracteres de longitud 9 (8 dígitos más letra).
- **Correo**: un *Varchar* de mayor longitud (45 caracteres). Dado que el nombre de correo debe ser único para cada empleado, lo marcamos como clave candidata (UNIQUE).

El campo multivariado teléfono se descompone, en el modelo relacional, en dos columnas: teléfono móvil y teléfono fijo.

- **Teléfono móvil**: un campo entero **sin signo** (dado que no puede ser negativo). Por otro lado, el número de teléfono móvil para cada empleado es único, por lo que se trata de una clave candidata, marcando dicho campo como UNIQUE.
- Teléfono fijo: un campo entero **sin signo**. A diferencia del caso anterior, un teléfono fijo puede ser compartido por varios empleados (familiares), por lo que puede repetirse.

El campo compuesto dirección, definido en el modelo Entidad-Relación, se descompone en cada uno de sus atributos:

- Calle: un *Varchar* de mayor longitud (80 caracteres).
- Número: un *Varchar* de hasta 3 caracteres (permitiendo un número de calle de hasta 3 dígitos). Dado que una calle puede no tener número, en caso de no insertarlo (NULL) por defecto se incluye *s/n* (sin número), con el objetivo de evitar en la medida de lo posible valores nulos.
- Código postal: un *Char* de tamaño fijo (5 caracteres). Se ha decidido utilizar una cadena de caracteres en lugar de un entero por la omisión de ceros en algunos códigos postales. A modo de ejemplo, el código 00610, en caso de tratarse de un entero, quedaría insertado en la tabla como 610. Por el contrario, mediante un *Varchar* (al tratarse de una cadena de caracteres) se conservan ambos ceros en la tabla.

Salvo el campo número, el resto de campos se consideran obligatorios para obtener la información completa del Empleado, por lo que **ninguno de ellos puede ser NULL**.

EMPLEADO	
PK	DNI CHAR(9) NOT NULL
	CORREO VARCHAR(45) UNIQUE NOT NULL
	TLFNO_MOVIL UNSIGNED INT UNIQUE NOT NULL
	TLFNO_FIJO UNSIGNED INT NOT NULL
	CALLE VARCHAR(80) NOT NULL
	NUMERO VARCHAR(3) DEFAULT 's/n'
	CP CHAR(5) NOT NULL

Ilustración 15. Tabla empleado

TRABAJA

Una vez diseñada tanto la tabla Empresa como Empleado, deben unirse por medio de relaciones. Dada la relación N:N definida en el modelo anterior (un empleado puede trabajar en varias empresas, del mismo modo que en una empresa pueden trabajar varios empleados), la relación se convierte en una tabla formada por las claves foráneas (FK) de ambas tablas:

- *DNI*: Empleado.
- *VAT*: Empresa.

Sin embargo, un empleado puede haber trabajado en una misma empresa durante dos periodos de tiempo, por lo que hay que añadir como atributos los campos FECHA_INI y FECHA_FIN, ambos de tipo DATE (no pueden ser nulos).

Una vez definidos los atributos, debemos preguntarnos ¿Cuál debe ser la/s clave/s primaria/s de la tabla? Si solo definimos como PK la pareja DNI, VAT, no permitiríamos que un empleado pudiera trabajar en la misma empresa en diferentes ocasiones. Por ello, la clave primaria debe extenderse un atributo más, incluyendo FECHA_INI, lo que permitiría varias entradas de un mismo empleado en la misma empresa, siempre y cuando la fecha de inicio sea diferente.

En relación con la cardinalidad, dado que la relación es N:N, implica que una empresa puede aparecer 1 o N veces en la tabla trabaja (mínimo y máximo), del mismo modo que un empleado puede también aparecer 1 o N veces.

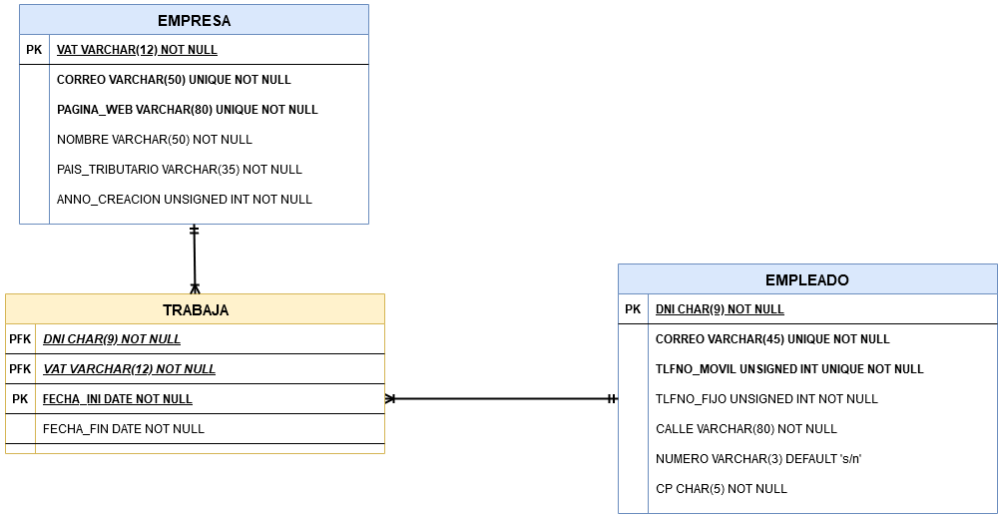

Ilustración 16. Relación trabaja

APLICACION

En su paso al modelo de tablas, Aplicación se compone de los siguientes atributos:

- Nombre: la clave primaria de la tabla, definiendo para ello un *Varchar* de tamaño medio (35 caracteres).
- **Código**: un entero sin signo. Dado que es único para cada aplicación, se trata de una clave candidata, por lo que debe marcarse como UNIQUE.
- Fecha_ini y Fecha_fin: fecha de inicio y fin del diseño de la aplicación, ambos de tipo *Date*.
- Espacio (en MB): un campo de tipo *Double* sin signo. A diferencia del precio, una aplicación no puede ocupar 0 MB, por lo que deberá tenerse en cuenta en la creación del *script* SQL.
- Precio: un campo de tipo *Double* sin signo.
- *Empleado_DNI*: por último, dado que una aplicación debe ser dirigida por un empleado y un empleado puede dirigir varias aplicaciones (relación 1:N), en lugar de una tabla intermedia, la tabla Aplicación deberá almacenar, como clave foránea, el DNI del empleado encargado de gestionarla. No obstante, dicho DNI no puede formar parte de la clave primaria, dado que un empleado puede dirigir varias aplicaciones. Además, en caso de que el empleado desaparezca (como veremos en la integridad referencial), su clave foránea en Aplicación se elimina, marcándose como NULL.

Por tanto, y salvo la clave foránea, **ninguno de los atributos puede ser NULL**, con el objetivo de almacenar toda la información de la aplicación. Por otro lado, la clave foránea *Empleado_DNI* no forma

parte de la clave primaria de la tabla, dado que un empleado puede dirigir más de una aplicación. Por tanto, la relación es **no identificativa**, marcado mediante líneas discontinuas:

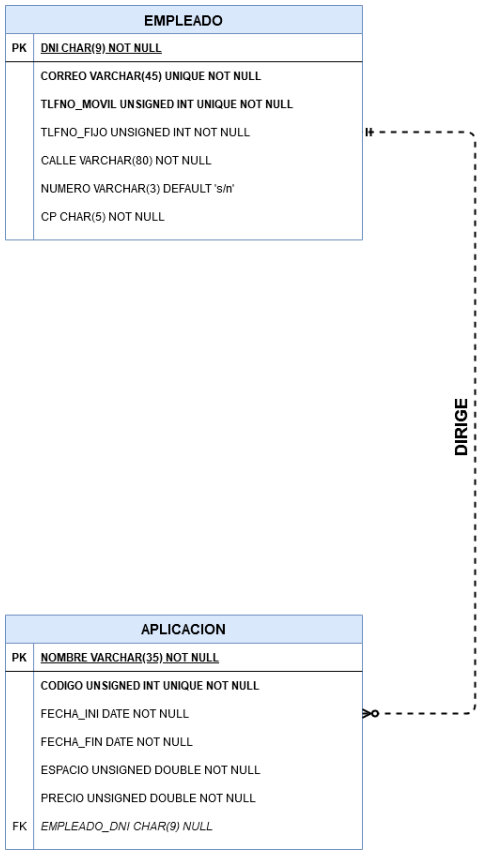

Ilustración 17. Entidad aplicación

En el resto de las relaciones, las claves foráneas si forman parte de la clave primaria, por lo que se tratan de relaciones identificativas, marcadas mediante una línea continua.

CATEGORIA

Sin embargo, nos falta el campo categorías. A diferencia del campo compuesto teléfono, donde puede ser fijo o móvil, una aplicación puede tener asociada múltiples categorías, bien 1,2,3 o más de 3 (no es una longitud fija). Por ello, en lugar de descomponer el atributo en múltiples columnas, la categoría se traducirá en una nueva tabla, relacionada con Aplicación. Dicha tabla está conformada por los siguientes campos:

- Id_categoría: campo identificativo (**clave primaria**), cuyo valor se autoincrementa por cada inserción.
- **Nombre**: campo con el nombre de la categoría, empleando para ello un *Varchar* de tamaño medio (20 caracteres). Dado que cada categoría tiene un nombre único, se trata de una **clave candidata**, por lo que debe marcarse como UNIQUE.

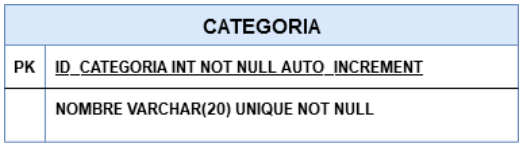

Ilustración 18. Entidad categorías

CATEGORIA_APLICACION

Dado que una aplicación puede tener una o varias categorías (mínimo y máximo), así como una categoría puede pertenecer a una o varias aplicaciones (1,N), la relación N:N resultante se traduce en una tabla intermedia, denominada **Categoria_aplicacion**, formada por las claves foráneas de ambas tablas:

- ***NOMBRE***: Aplicación.
- ***ID_CATEGORÍA***: Categoría.

Ambas claves foráneas son, a su vez, **clave primaria** de la tabla intermedia. De lo contrario, se permitiría que una aplicación tuviese la misma categoría en múltiples ocasiones, generando duplicidades. Dado que la cardinalidad es 1:N en ambos lados, una aplicación puede aparecer 1 o N veces en la tabla **Categoria_aplicacion**, del mismo modo que una categoría puede aparecer en 1 o N ocasiones.

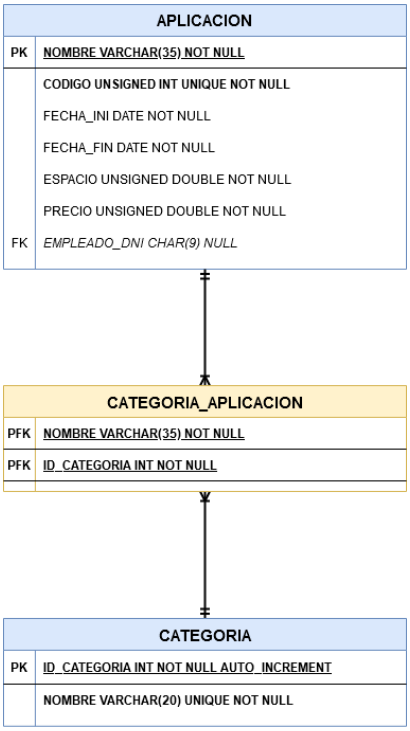

Ilustración 19. Relación categoría_aplicacion

CONTIENE

Según el modelo Entidad-Relación, una aplicación puede ser subida a varias tiendas (1,N), del mismo modo que una tienda puede contener múltiples aplicaciones (1,N). Por tanto, la relación N:N se traduce en una tabla intermedia, denominada **contiene**, formada por las parejas de claves foráneas:

- ***NOMBRE_TIENDA***: Tienda.
- ***NOMBRE_APLICACION***: Aplicación.

Ambas claves foráneas son, a su vez, **clave primaria** de la tabla intermedia. De lo contrario, se permitiría que una tienda alojase la misma aplicación múltiples veces:

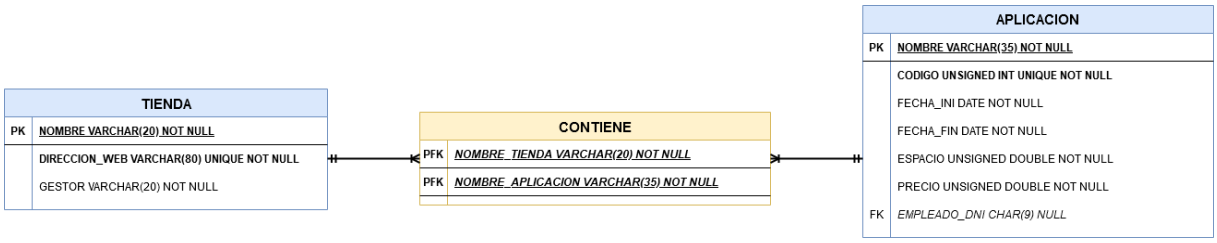

Ilustración 20. Relación contiene

REALIZA

Una vez diseñada tanto la tabla Empleado como la tabla Aplicación, nos falta añadir una última relación entre ambos: un empleado puede realizar una o varias aplicaciones, de la misma forma que una aplicación es realizada por uno o varios empleados (1,N); por lo que la relación resultante (N:N) es una tabla intermedia, denominada **realiza**, formada por las claves foráneas de ambas tablas:

- ***DNI***: Empleado.
- ***NOMBRE***: Aplicación.

Ambas claves foráneas son, a su vez, **clave primaria** de la tabla intermedia. De lo contrario, se permitiría que un empleado participase en un mismo proyecto en más de una ocasión.

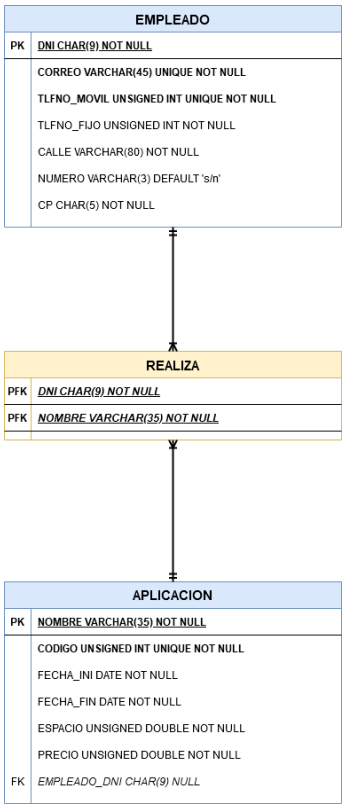

Ilustración 21. Relación realiza

CREA

A continuación, debemos relacionar **Empleado** con **Aplicación**. La relación varios a varios (N:N) **crea** obtenida en el modelo Entidad-Relación, se convierte nuevamente en otra tabla intermedia, formada por las parejas de claves foráneas de ambas tablas:

- ***VAT***: Empresa.
- ***NOMBRE***: Aplicación.

Ambas claves foráneas son, a su vez, claves primarias, lo que permitiría no solo conocer qué aplicaciones ha realizado cada empresa (sin necesidad de pasar por Empleado), sino además evitar que una empresa desarrolle la misma aplicación en múltiples ocasiones, es decir, evitar duplicidades. Dado que la cardinalidad es 1:N en ambos lados de la tabla intermedia, una empresa puede aparecer 1 o N veces en la tabla intermedia, del mismo modo que una aplicación puede aparecer en 1 o N ocasiones:

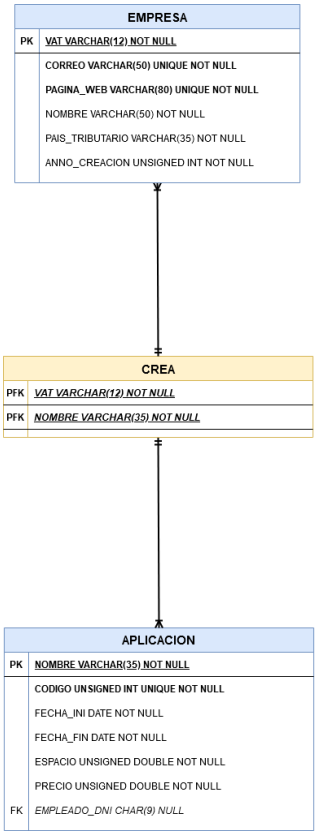

Ilustración 22. Relación crea

USUARIO

En su paso al modelo relacional, Usuario se compone de los siguientes atributos:

- Num_cuenta: clave primaria de la tabla, empleando para ello un entero sin signo.
- **Nombre**: definido mediante un *Varchar* de tamaño medio (50 caracteres). Dado que el nombre de usuario es único, se trata de una clave candidata, por lo que debe marcarse como UNIQUE.

El campo dirección, definido en el modelo Entidad-Relación, se descompone en cada uno de sus atributos (al igual que sucedía con la tabla Empleado):

- Calle: un *Varchar* de mayor longitud (80 caracteres).
- Número: un *Varchar* de hasta 3 caracteres (permitiendo un número de calle de hasta 3 dígitos). Dado que una calle puede no tener número, en caso de no insertar ningún valor (NULL), por defecto se incluye *s/n* (sin número), con el objetivo de evitar valores nulos.
- Código postal: un *Char* de tamaño fijo (5 caracteres). Se ha decidido utilizar una cadena de caracteres por la omisión de ceros en algunos códigos postales si se tratase de un valor entero (ejemplo: 00610).
- País: un *Varchar* de tamaño medio (35 caracteres).

Salvo el campo número, y con el objetivo de almacenar por completo la información del usuario, **ninguno de los campos anteriores puede ser NULL**.

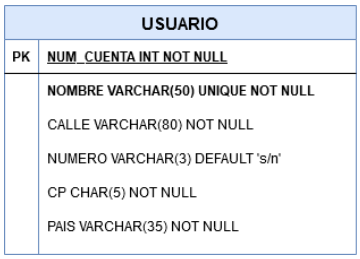

Ilustración 23. Tabla usuario

DESCARGA

Un usuario puede descargar, como mínimo, 0 aplicaciones, y como máximo N, del mismo modo que una aplicación puede ser descargada por 0 o N usuarios. Como consecuencia, la relación resultante (N:N) se traduce nuevamente en una tabla intermedia, denominada **descarga**, formada tanto por las claves de ambas tablas (foráneas) que a su vez serán claves primarias:

- ***NUM_CUENTA***: Usuario.
- ***NOMBRE***: Aplicación.

Como, además, los siguientes campos:

- Puntuacion: mediante un entero sin signo. De cara a su implementación en SQL, deberá tenerse en cuenta que el valor debe estar comprendido entre 0 y 5.
- Número de movil: mediante un entero sin signo. Dado que el teléfono de cada usuario es único, debe marcarse como UNIQUE.
- Fecha de descarga: definido mediante un campo *Date*.
- Comentario: para este campo, se define un tipo *Text*, lo que permite no limitar el campo a un determinado número de caracteres.

Tanto la puntuación como el campo comentario, dado que el usuario puede no puntuar u opinar, **pueden estar a NULL**.

Partiendo de la cardinalidad varios a varios (N:N) definida en el modelo Entidad-Relación, al tratarse de una tabla intermedia el número de cuenta de un usuario puede aparecer 0 o N veces en la tabla descarga (puede no descargar ninguna aplicación), del mismo modo que el nombre de una aplicación puede aparecer en 0 o N ocasiones (una aplicación puede no ser descargada).

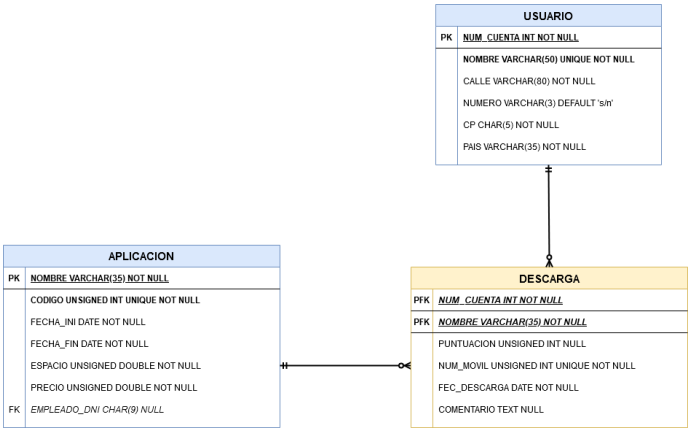

Ilustración 24. Relación descarga

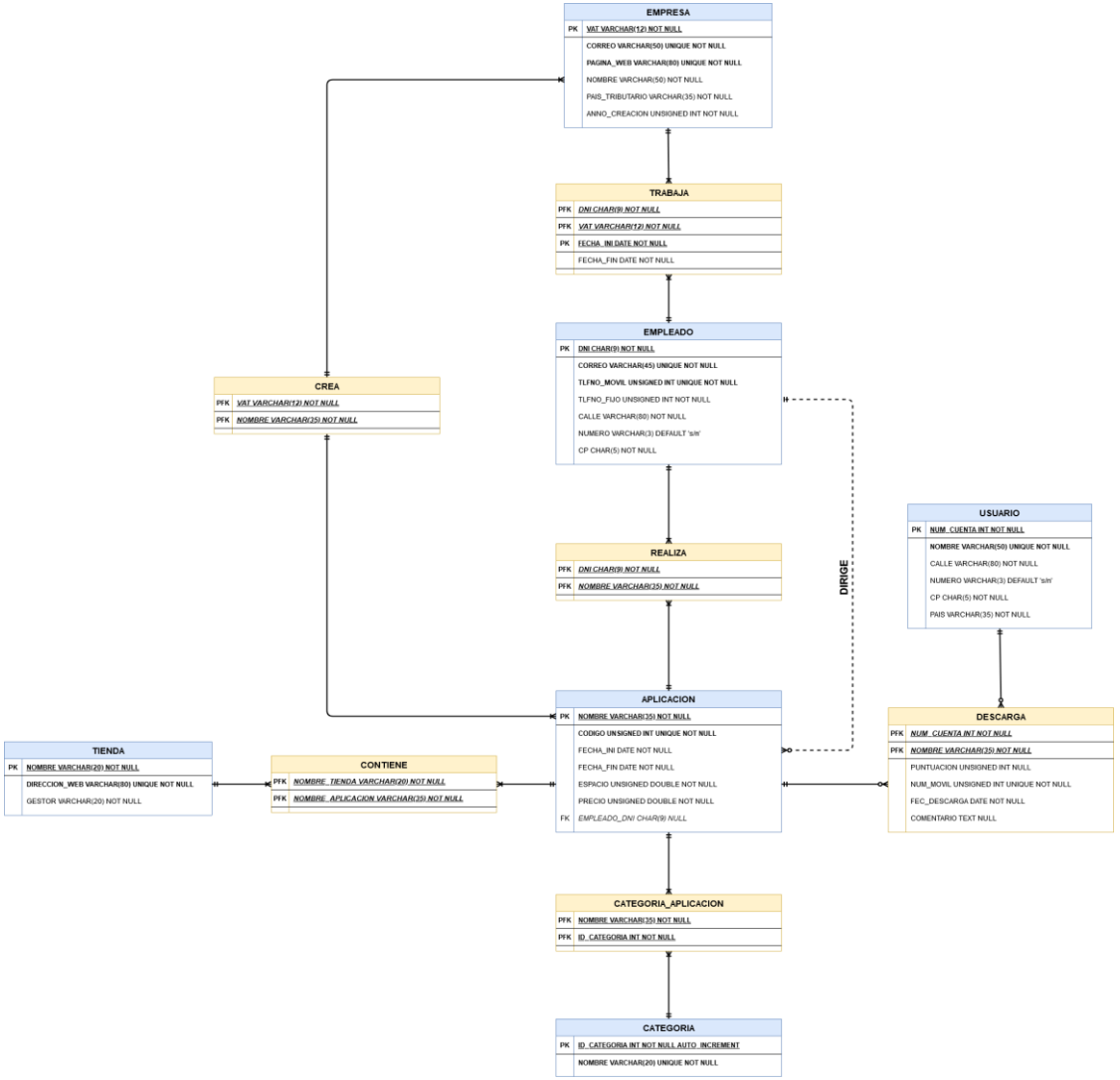

Ilustración 25. Modelo relacional final

UNIVERSIDAD COMPLUTENSE MADRID

3. IMPLEMENTACIÓN SQL

La implementación en SQL se ha realizado conforme al modelo relacional previamente diseñado, mediante el *script* FernandezHernandezAlberto.sql. En él se detalla la definición de cada una de las tablas anteriores, empleando las mismas columnas y tipos de datos. Sin embargo, quisiera remarcar varios aspectos importantes definidos en el *script*: las comprobaciones (**CHECK**), integridad referencial, procedimientos (**PROCEDURE**), funciones (**FUNCTION**) y *triggers*.

CHECK

RELACIÓN TRABAJA

En la definición de la tabla trabaja, cada vez que se inserta una nueva fila debe comprobarse que el campo FECHA_INI sea estrictamente menor a FECHA_FIN:

```
CHECK (FECHA_INI < FECHA_FIN)
```

TABLA APLICACION

En la tabla aplicación, al igual que en la tabla trabaja, por cada inserción debe comprobarse que el campo FECHA_INI sea estrictamente menor a FECHA_FIN. Por otro lado, debido a problemas de versiones en MySQL Server 8.x, los tipos de datos FLOAT y DOUBLE sin signo (UNSIGNED) serán eliminados en futuras versiones. Para evitar problemas y asegurar que tanto el campo ESPACIO es mayor a 0 como el campo PRECIO mayor o igual a cero, empleamos un CHECK:

```
CHECK (FECHA_INI < FECHA_FIN),
CHECK (ESPACIO > 0),
CHECK (PRECIO >= 0)
```

RELACIÓN DESCARGA

Por último, tal y como se mencionó en los modelos anteriores, el valor de puntuación de una aplicación debe estar comprendido entre 0 y 5 (salvo que sea NULL), por lo que deberá comprobarse:

```
CHECK (PUNTUACION BETWEEN 0 AND 5)
```

INTEGRIDAD REFERENCIAL

En el presente apartado se definen las restricciones de integridad aplicadas para cada tabla en la que aparezca cualquier clave foránea (tanto en DELETE como en UPDATE).

RELACIÓN TRABAJA

ON DELETE

Si el DNI del empleado desapareciera, las entradas correspondientes en la tabla trabajan estarían haciendo referencia a un empleado que ya no existe, por lo que son también eliminadas en cascada (CASCADE). No obstante, si una empresa desapareciera, no necesariamente deben borrarse a todos los empleados que han trabajado en ella, de lo contrario no se podría conocer el historial completo de un empleado (RESTRICT), aunque la empresa ya no exista, lo que impediría conocer no solo las empresas en las que ha trabajado sino los años de experiencia.

ON UPDATE

Por el contrario, si el DNI o VAT se actualizan, es necesario actualizar los campos en la tabla N:N, de lo contrario no podría identificarse al empleado o empresa correctamente. Por ello, la actualización se realiza en cascada (CASCADE).

```
FOREIGN KEY (DNI)
REFERENCES EMPLEADO (DNI)
ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY (VAT)
REFERENCES EMPRESA (VAT)
ON DELETE RESTRICT
```

```
ON UPDATE CASCADE);
```

TABLA APLICACIÓN

La tabla aplicación, tal y como se ha mostrado en el modelo relacional contiene (como clave foránea) el DNI del empleado que dirige o dirigió la aplicación. A continuación, se analiza el comportamiento de la tabla en caso de borrado o actualizado de dicha clave foránea.

ON DELETE

Una posible alternativa sería que, si el empleado/responsable es borrado de la base de datos, la aplicación también debería eliminarse. Sin embargo, no tiene porqué ocurrir de este modo y por ello hay que tener especial cuidado. A modo de ejemplo, supongamos una aplicación ya terminada y lanzada al mercado, gestionada por el empleado con el DNI 'A'. Si el empleado 'A' ya no estuviera en la empresa, no implicaría automáticamente que la aplicación dejase de existir, en especial cuando está disponible a los usuarios en tienda. Por el contrario, al marcar el proceso de borrado como SET NULL permitiría mantener la aplicación, eliminando únicamente la clave foránea del jefe de proyecto, pues ya no existe en la base de datos.

ON UPDATE

Al contrario que en el proceso de borrado, si el DNI del empleado se actualiza, la clave foránea en Aplicación deberá actualizarse automáticamente, por lo que la actualización se realiza en cascada (CASCADE).

```
FOREIGN KEY (EMPLEADO_DNI)
REFERENCES EMPLEADO (DNI)
    ON DELETE SET NULL
    ON UPDATE CASCADE);
```

RELACIÓN REALIZA

ON DELETE

Al igual que la relación trabaja, si el DNI del empleado desapareciera, implica que dejaría de trabajar en la aplicación correspondiente, por lo que los datos serían inconsistentes. Por ello, si un empleado es borrado, el proceso se propaga hasta la tabla realiza (CASCADE), del mismo modo que se propaga sobre la tabla trabaja. No obstante, si una aplicación desaparece, al igual que ocurría con la empresa, no podríamos borrar con la misma facilidad, dado que impediría conocer la experiencia profesional de un empleado (en qué aplicaciones ha trabajado) e incluso el número de empleados que participaron en su desarrollo, por lo que en esta situación sería RESTRICT.

ON UPDATE

Por el contrario, si el DNI o el nombre de la aplicación se actualizan, es necesario actualizar los campos en la tabla N:N, de lo contrario no podría identificarse al empleado o aplicación correctamente. Por ello, la actualización se realiza en cascada (CASCADE).

```
    FOREIGN KEY (DNI)
REFERENCES EMPLEADO (DNI)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (NOMBRE)
REFERENCES APLICACION (NOMBRE)
    ON DELETE RESTRICT
    ON UPDATE CASCADE);
```

RELACIÓN CREA

ON DELETE

De la misma forma que en la relación trabaja, si el nombre de la empresa desapareciese, no necesariamente deben borrarse todas las entradas en la relación crea, dado que una aplicación podría haber sido desarrollada por varias empresas. A modo de ejemplo, si una aplicación ha sido creada por tres empresas y una de ellas desaparece, al consultar en un futuro qué empresas participaron solo serían dos, cuando realmente no ha sido así, por lo que se marca como RESTRICT. Por otro lado, si el nombre de la aplicación desapareciese, se estaría haciendo referencia a una aplicación que no existe, por lo que el borrado en este caso si debe realizarse en cascada (CASCADE).

ON UPDATE

En contraposición, si el VAT o el nombre de la aplicación se actualizan, sus campos en la tabla crea también deben modificarse (CASCADE).

```
FOREIGN KEY (VAT)
REFERENCES EMPRESA (VAT)
ON DELETE RESTRICT
ON UPDATE CASCADE,
FOREIGN KEY (NOMBRE)
REFERENCES  APLICACION (NOMBRE)
ON DELETE CASCADE
ON UPDATE CASCADE);
```

RELACIÓN CONTIENE

ON DELETE

A diferencia de las relaciones N:N anteriores, si la aplicación o la tienda desaparecen, en cualquiera de los dos casos deben borrarse sus correspondientes entradas en la tabla contiene (CASCADE). A modo de ejemplo, si la App Store dejase de existir, todas las aplicaciones que alojaba (exclusivas o no de Apple), dejarían de estar disponibles para dispositivos iOS/MacOS, por lo que el borrado de la tienda implicaría eliminar también todas aquellas aplicaciones relacionadas con la AppStore. Por otro lado, si una aplicación dejara de existir, ninguna tienda podría ofrecerla, por lo que el borrado también debe ser en cascada.

ON UPDATE

Al igual que en el borrado, si el nombre de la aplicación o de la tienda cambian, el proceso de actualizado también se produciría en cascada.

```
FOREIGN KEY (NOMBRE_TIENDA)
REFERENCES TIENDA (NOMBRE)
ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY (NOMBRE_APLICACION)
REFERENCES APLICACION (NOMBRE)
ON DELETE CASCADE
ON UPDATE CASCADE);
```

RELACIÓN CATEGORIA_APLICACION

ON DELETE

Si una categoría en particular desapareciera, todas las entradas que hagan referencia a dicha categoría en la tabla categoria_aplicacion deben ser borradas en cascada, del mismo modo que si una aplicación dejara de existir, todas sus entradas en la tabla deben ser eliminadas (CASCADE), de lo contrario se estaría haciendo referencia a una categoría/aplicación que no existe.

ON UPDATE

Al igual que en el proceso de borrado, si una categoría o aplicación cambian de nombre, el proceso de actualizado también se realiza en cascada.

```
        FOREIGN KEY (NOMBRE)
    REFERENCES APLICACION (NOMBRE)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
      FOREIGN KEY (ID_CATEGORIA)
    REFERENCES CATEGORIA (ID_CATEGORIA)
        ON DELETE CASCADE
        ON UPDATE CASCADE);
```

RELACIÓN DESCARGA

ON DELETE

En el caso del borrado, existe una diferencia con respecto al resto de relaciones con la tabla Aplicación: si un usuario dejara de existir, al igual que ocurría con el resto de las relaciones, debería eliminarse en cascada. No obstante, debemos recordar un campo: **la fecha de descarga**. Si el borrado fuese en cascada, en caso de querer consultar el número de descargas realizadas en una determinada fecha, obtendríamos un número inferior al real, pues algunos de estos usuarios han podido cerrar su cuenta y, como consecuencia, su borrado en la tabla. Del mismo modo, si una aplicación desapareciera, el borrado no debería propagarse a la tabla descargas, pues impediría conocer el número real de descargas realizadas en una determinada fecha (RESTRICT).

ON UPDATE

En contraposición, si el usuario o la aplicación actualizan sus claves primarias, la actualización si debe propagarse a la tabla descargas, evitando con ello posibles inconsistencias en los datos (CASCADE).

```
        FOREIGN KEY (NUM_CUENTA)
    REFERENCES USUARIO (NUM_CUENTA)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
      FOREIGN KEY (NOMBRE)
    REFERENCES APLICACION (NOMBRE)
        ON DELETE RESTRICT
        ON UPDATE CASCADE);
```

Una vez realizada la carga de datos, (salvo la tabla categorías la carga se ha realizado por medio de ficheros .csv), quedan definir los *triggers*. Para evitar la redundancia de código, se definen inicialmente un procedimiento, encargado de comprobar si una fecha es mayor a la fecha actual; así como una función encargada de validar un DNI.

Nota: dado que hay algunos *triggers* que requieren de algunas consultas sobre datos ya insertados, los *triggers* se añadirán una vez cargados todos los datos, y mediante un INSERT de prueba se demostrará su correcto funcionamiento.

PROCEDURE

Dicho procedimiento, denominado **comprobar_fecha**, evalúa si la fecha pasada como parámetro es mayor o no a la fecha actual. En caso afirmativo, mostrará un mensaje de error (mediante la variable del sistema MESSAGE_TEXT) con el código de error 45000, un código en SQL que indica que se trata de un error definido por el usuario:

```
DELIMITER $$
CREATE PROCEDURE comprobar_fecha(fecha DATE)
BEGIN
      IF fecha > CURDATE() THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error. La fecha es
superior a la fecha actual';
      END IF;
END$$
```

FUNCTION

Como función, **comprobar_letra_dni** permite comprobar si un DNI, pasado como parámetro, es o no válido. Para ello, recupera el valor numérico del documento (los 8 primeros caracteres) y, a continuación, calcula su letra mediante la operación módulo. Una vez recuperada la letra, la devuelve para, desde un *trigger*, comprobar si la letra calculada corresponde con la que aparece en el DNI original:

```
CREATE FUNCTION comprobar_letra_dni(dni CHAR(9))
RETURNS CHAR(9)
BEGIN
      SET @dni_sin_letra = cast(substring(dni,1,8) as UNSIGNED);
      SET @letra = substring('TRWAGMYFPDXBNJZSQVHLCKE', @dni_sin_letra % 23
+ 1,  1);
    RETURN @letra;
END$$
```

TRIGGERS

Nota: todos los *triggers* creados se lanzan **antes de la inserción** (BEFORE INSERT).

COMPROBAR_FECHA_FIN_TRABAJA_BI

Este *trigger* se encargará de comprobar si la fecha de fin en el que un empleado deja de trabajar es mayor a la fecha actual y, en caso afirmativo, devuelve un error. Dado que existe un procedimiento encargado de comparar fechas, se llamará dentro del mismo *trigger*:

```
CREATE TRIGGER comprobar_fecha_fin_trabaja_BI
BEFORE INSERT
ON trabaja FOR EACH ROW
BEGIN
    CALL comprobar_fecha(NEW.FECHA_FIN);
END$$
```

Comprobación:

```
INSERT INTO trabaja
VALUES('56813892M','ES12345600','2013/07/10','2021/10/21');
```

73 17:24:54 INSERT INTO trabaja VALUES('56813892M','ES12345600','2013/07/10','2021/10/21')

Error Code: 1644. Error. La fecha es superior a la fecha actual

COMPROBAR_FECHA_FIN_APLICACION_BI

De forma similar al *trigger* anterior, se encargará de comprobar si la fecha de finalización de una aplicación es mayor a la fecha actual, en cuyo caso devolverá un error.

```
CREATE TRIGGER comprobar_fecha_fin_aplicacion_BI
BEFORE INSERT
ON aplicacion FOR EACH ROW
BEGIN
    CALL comprobar_fecha(NEW.FECHA_FIN);
END$$
```

Comprobación:

```
INSERT INTO aplicacion
VALUES('Instagram',16568,'2013/07/10','2021/10/21',54,0,'56813892M');
```

75 17:28:35 INSERT INTO aplicacion VALUES('Instagram',16568,'2013/07/10','2021/10/21',54,0,'56813892M')

Error Code: 1644. Error. La fecha es superior a la fecha actual

COMPROBAR_FECHA_DESCARGA_BI

Al igual que los *triggers* anteriores, comprobará si la fecha de descarga de una aplicación es mayor a la fecha actual. Sin embargo, también deberá comprobar si la fecha de descarga es inferior a la fecha de finalización de la aplicación, es decir, antes de haber sido lanzada en tienda. Para ello, dentro del *trigger* realizará una consulta sobre la fecha de fin de la aplicación en particular y, en caso de que la fecha de descarga sea menor, mostrará un error por pantalla:

```
CREATE TRIGGER comprobar_fecha_descarga_BI
BEFORE INSERT
ON descarga FOR EACH ROW
BEGIN
	IF NEW.FEC_DESCARGA < (SELECT FECHA_FIN FROM APLICACION WHERE NOMBRE
= NEW.NOMBRE) THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error. La fecha es
superior a la fecha de fin de la aplicacion';
	ELSE
		CALL comprobar_fecha(NEW.FEC_DESCARGA);
	END IF;
END$$
```

Comprobación (la fecha de descarga es menor a la fecha de fin):

```
INSERT INTO descarga
VALUES('663295','RTNoticias',0,43000744,'2019/08/01','Definitivamente odio
RTNoticias!!!!!!');
```

71 17:42:35 INSERT INTO descarga VALUES('663295','RTNoticias',0,43000744,'2019/08/01','Definitivamente odio RTNoticias!!!!!!')

Error Code: 1644. Error. La fecha es superior a la fecha de fin de la aplicacion

Comprobación (la fecha de descarga es mayor a la fecha actual):

```
INSERT INTO descarga
VALUES('663295','RTNoticias',0,43000744,'2021/10/21','Definitivamente odio
RTNoticias!!!!!!');
```

73 17:45:37 INSERT INTO descarga VALUES('663295','RTNoticias',0,43000744,'2021/10/21','Definitivamente odio RTNoticias!!!!!!')

Error Code: 1644. Error. La fecha es superior a la fecha actual

COMPROBAR_LETRA_DNI_BI

Gracias a la función **comprobar_letra_DNI** definida anteriormente, mediante un *trigger* se comprobará si el DNI insertado es o no válido, en función de la letra calculada por dicha función. En caso de no coincidir, devolverá un mensaje de error:

```
CREATE TRIGGER comprobar_letra_dni_BI
BEFORE INSERT
ON empleado FOR EACH ROW
BEGIN
    SET @letra = comprobar_letra_dni(NEW.DNI);
    IF @letra <> substring(NEW.DNI,9,9) THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Error. EL DNI NO es
válido";
	END IF;
END$$
```

Comprobación (DNI No válido):

```
INSERT INTO empleado
VALUES('54053101S','Salvadoeer23@gmail.com',990252340,650829990,'C.
Comercial Espacio Leon','122','09019');
```

```
75  17:49:16  INSERT INTO empleado VALUES('54053101S','Salvadoeer23@gmail.com',990252340,650829990,'C. Comercial Espacio Leon','122','09019')
```

Comprobación (DNI Válido):

```
INSERT                                  INTO                                  empleado
VALUES('10195062J','Alberto23@gmail.com',990252340,650829997,'C.   Comercial
Espacio Leon','122','09019');
```

```
78  17:51:18  INSERT INTO empleado VALUES('10195062J','Alberto23@gmail.com',990252340,650829997,'C. Comercial Espacio Leon','122','09019')

1 row(s) affected
```

COMPROBAR_TRABAJA_EN_APLICACION_BI

Este último *trigger* se encargará de comprobar si el responsable de una aplicación **ha sido previamente insertado en la tabla *realiza***, es decir, comprueba si el jefe de proyecto realmente está trabajando en dicha aplicación, por lo que deberá estar en la tabla. Para ello, comprueba si el DNI del responsable está incluido en la tabla *realiza*, mediante una consulta. En caso negativo, devuelve un mensaje de error:

```
CREATE TRIGGER comprobar_trabaja_en_aplicacion_BI BEFORE INSERT ON
aplicacion
    FOR EACH ROW
    BEGIN
        IF NEW.EMPLEADO_DNI NOT IN (
            SELECT distinct(r.DNI)
            FROM realiza AS r
            WHERE (NEW.EMPLEADO_DNI = r.DNI)
        ) THEN
           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Error. El jefe de
proyecto no está en la tabla Realiza";
      END IF;
END$$
```

Comprobación (mediante el DNI válido insertado anteriormente en Empleado):

```
79  18:00:42  INSERT INTO aplicacion VALUES('Instagram',16568,'2013/07/10','2020/10/20',54,0,'10195062J')

Error Code: 1644. Error. El jefe de proyecto no esta en la tabla Realiza
```

Los ejemplos anteriores están incluidos en el *script* pruebas_triggers.sql

4. CONSULTAS

Consulta 1

Obtener la fecha en la que se realizan más descargas

Para realizar esta consulta, mediante un ORDER BY se ordena el número de descargas de forma descendente, escogiendo la primera fila, es decir, la fecha con más descargas:

```
SELECT FEC_DESCARGA, count(FEC_DESCARGA) as NUM_DESCARGAS
FROM descarga
GROUP BY FEC_DESCARGA
ORDER BY NUM_DESCARGAS DESC
LIMIT 1;
```

FEC_DESCARGA	NUM_DESCARGAS
2014-03-26	36

Ilustración 26. Salida consulta 1

Consulta 2

Obtener los datos de las 3 empresas que hayan participado más veces en el desarrollo de aplicaciones

Dado que se dispone de una tabla intermedia (crea), mediante la función **count** se cuenta el número de apariciones en dicha tabla por cada empresa, ordenando de forma descendente el resultado, recuperando la primera fila:

```
SELECT e.*, count(c.VAT) AS PARTICIPACIONES
FROM empresa AS e INNER JOIN crea AS c USING(VAT)
GROUP BY c.VAT
ORDER BY PARTICIPACIONES DESC
LIMIT 3;
```

VAT	NOMBRE	PAIS_TRIBUTARIO	ANNO_CREACION	CORREO	PAGINA_WEB	PARTICIPACIONES
ES12345679	DigitalValue	Espana	2012	contact@digvalue.es	https://www.dig-value.com/	4
IR1232647	AppReborn	Irlanda	2009	contact@appreb.ir	https://www.app-reborn.com/	3
U12345679	MajorApp	Austria	2013	contact@majorapp.aut	https://www.major-app.com	3

Ilustración 27. Salida consulta 2

Consulta 3

Obtener aquellos empleados que hayan estado en más de una empresa (o en la misma empresa más de una vez), que tengan extensión de correo @gmail

Para ello, mediante una subconsulta se obtendrán aquellos DNI que aparezcan en más de una ocasión en la tabla *trabaja*, lo que implica que el empleado o bien ha trabajado en más de una empresa o bien en la misma en diferentes periodos de tiempo. Por otro lado, mediante la instrucción LIKE se filtran aquellos empleados con correo *gmail*:

```
SELECT emp.*
FROM empleado AS emp INNER JOIN (SELECT DNI FROM trabaja GROUP BY DNI
HAVING count(DNI) > 1) AS t
ON emp.DNI = t.DNI
WHERE CORREO LIKE '%@gmail%';
```

DNI	CORREO	TLFNO_FIJO	TLFNO_MOVIL	CALLE	NUMERO	CP
00804278Z	Cristina212@gmail.com	994118330	629883140	Callejon Cuesta	13	28999
17158772J	MaitePozo@gmail.com	923190330	649139890	C. Comercial Espacio Leon	120	09190
28780713P	AlbertoFGT@gmail.com	994203230	600315930	Alameda del Cipres	s/n	44110
29395097Q	Juan12112@gmail.com	999414950	735509100	Paseo de la Castellana	59	00001
29707354W	AntonioFGTE@gmail.com	999399080	633313210	Plaza de la Habana	13	00001
36903975S	MariaUAH@gmail.com	924991380	619594330	Acceso glorieta Santander	120	44910
40501746B	Alberto8959@gmail.com	994989430	755299350	Glorieta Lorem ipsum	1	01111
51129590S	Remedios232311@gmail.com	900510990	788993410	Carrera Lorem ipsum	32	50322
63774917B	Mari_Paz1212@gmail.com	919299980	648045990	Plaza de la Habana	122	01419
73307151X	Mari_Paz2398@gmail.com	991122810	601509950	Alameda del Cipres	54	44910
80249243N	GuillermoHernandez@gmail....	990915910	623114430	Plaza de la Habana	13	10921
82728719L	Alfonso23@gmail.com	989489990	709998010	Avenida Lorem	13	01111
87293105D	Jorgefh@gmail.com	942414990	649181420	Callejon Cuesta	99	39259
93155413P	Antonio232@gmail.com	994341490	652215230	Avenida Lorem	122	01111

Ilustración 28. Salida consulta 3

Consulta 4

Obtener el DNI del empleado con menos experiencia laboral (en meses)

Para este apartado, se ha empleado la función **timestampdiff**, propia de MySQL, que permite calcular la diferencia de tiempo (pudiendo elegir entre años, meses o días, entre otros) entre dos fechas dadas como parámetro. Mediante dicha función, se calcula el tiempo trabajado por cada empleado para finalmente, mediante la función de agregación **sum**, sumar cada uno de estos tiempos, agrupados por el DNI del empleado:

```
SELECT e.DNI, sum(timestampdiff(month, t.FECHA_INI, t.FECHA_FIN)) as MESES
FROM empleado AS e INNER JOIN trabaja AS t USING(DNI)
GROUP BY e.DNI
ORDER BY MESES
LIMIT 1;
```

DNI	MESES
61191262S	12

Ilustración 26. Salida consulta 4

Consulta 5

Obtener el país de los usuarios que más aplicaciones se han descargado (y el que menos)

Para este apartado, se han realizado dos consultas, unidas mediante la sentencia UNION. Cada una realiza a su vez una subconsulta, la cual obtiene el número total de descargas por país, ordenadas de forma de descendente y ascendente, respectivamente. A continuación, de cada una se recuperará tanto el valor máximo como mínimo, mediante las operaciones de agregación **max** y **min**:

```
SELECT descargas.PAIS, max(descargas.NUM_DESCARGAS) AS DESCARGAS
FROM
(SELECT PAIS, count(PAIS) as NUM_DESCARGAS
FROM usuario INNER JOIN descarga USING(NUM_CUENTA)
GROUP BY PAIS
ORDER BY NUM_DESCARGAS DESC) as descargas

UNION

SELECT descargas.PAIS, min(descargas.NUM_DESCARGAS) AS DESCARGAS
FROM
(SELECT PAIS, count(PAIS) as NUM_DESCARGAS
FROM usuario INNER JOIN descarga USING(NUM_CUENTA)
GROUP BY PAIS
ORDER BY NUM_DESCARGAS) as descargas;
```

	PAIS	DESCARGAS
▶	Espana	91
	Irlanda	2

Ilustración 27. Salida consulta 5

Consulta 6

Obtener el DNI, correo y móvil de aquellos empleados que NO hayan sido responsables de ninguna aplicación y que pertenezcan o hayan pertenecido a la empresa ItalicSystems

Para realizar esta consulta, se realiza un INNER JOIN entre las tablas **Empresa**, **trabaja**, **Empleado** y **realiza**. Por otro lado, con la tabla **Aplicacion** se realiza un LEFT JOIN, pues el objetivo es consultar qué empleados están desarrollando aplicaciones que no estén como clave foránea en Aplicacion:

```
SELECT distinct(e.DNI), e.CORREO, e.TLFNO_MOVIL
FROM empresa AS emp INNER JOIN trabaja AS t ON emp.VAT = t.VAT
INNER JOIN empleado AS e ON t.DNI = e.DNI
INNER JOIN realiza AS r ON e.DNI = r.DNI
LEFT JOIN aplicacion AS a ON r.DNI = a.EMPLEADO_DNI
WHERE a.EMPLEADO_DNI IS NULL AND emp.NOMBRE = 'ItalicSystems';
```

	DNI	CORREO	TLFNO_MOVIL
▶	29395097Q	Juan12112@gmail.com	735509100
	93155413P	Antonio232@gmail.com	652215230

Ilustración 28. Salida consulta 6

Consulta 7

Obtener la aplicación que menos tiempo haya requerido (en meses) y que haya obtenido un mayor número de descargas

Para esta consulta, se calcula el tiempo requerido por cada aplicación (mediante la función **timestampdiff**), así como contar el número de descargas mediante la función de agregación **count**. Una vez obtenidos, la tabla se ordena en orden ascendente en función del tiempo requerido y en orden descendente en función del número de descargas:

```
SELECT a.NOMBRE, timestampdiff(month, a.FECHA_INI, a.FECHA_FIN) as MESES,
count(d.NOMBRE) as DESCARGAS
FROM aplicacion AS a INNER JOIN descarga AS d USING(NOMBRE)
GROUP BY a.nombre
ORDER BY MESES ASC, DESCARGAS DESC
LIMIT 1;
```

	NOMBRE	MESES	DESCARGAS
▶	RadarCOVID	4	14

Ilustración 29. Resultado consulta 7

Consulta 8

Obtener el número de descargas, agrupadas por el nombre de la tienda (de menos a más descargas)

Al igual que en la consulta anterior, el número de descargas se calcula mediante la función **count**:

```
SELECT c.NOMBRE_TIENDA, count(d.NOMBRE) as NUM_DESCARGAS
FROM contiene AS c INNER JOIN aplicacion AS a ON c.NOMBRE_APLICACION =
a.NOMBRE
INNER JOIN descarga AS d ON a.NOMBRE = d.NOMBRE
GROUP BY c.NOMBRE_TIENDA
ORDER BY NUM_DESCARGAS;
```

	NOMBRE_TIENDA	NUM_DESCARGAS
▶	AppCatalog	113
	OVITienda	127
	MarketPlace	164
	AppStore	217
	AppWorld	217
	Google Play Store	220
	App Store	226

Ilustración 30. Resultado consulta 8

Consulta 9

Obtener los ingresos totales de las empresas gracias a las aplicaciones (solo de pago) descargadas, cuyo precio este por debajo de la media

Por medio de una subconsulta, se cuenta el número de descargas por cada aplicación para, finalmente, multiplicarlo por el precio correspondiente. Por otro lado, también se filtran aquellas aplicaciones cuyo precio sea distinto de cero y esté por debajo de la media, obtenida a través de una segunda subconsulta.

```
SELECT c.VAT, round(a.PRECIO * descargas_por_app.DESCARGAS,2) as INGRESOS
FROM crea AS c INNER JOIN aplicacion AS a ON c.NOMBRE = a.NOMBRE
INNER JOIN

(SELECT NOMBRE, count(NOMBRE) as DESCARGAS
FROM descarga
GROUP BY NOMBRE) as descargas_por_app ON c.NOMBRE =
descargas_por_app.NOMBRE

WHERE a.PRECIO != 0 AND a.PRECIO < (SELECT avg(PRECIO) FROM aplicacion
WHERE PRECIO != 0);
```

	VAT	INGRESOS
▶	ES12345600	67.10
	ES12345679	23.97
	U12345679	23.97

Ilustración 31. Resultado consulta 9

Consulta 10

Obtener el precio y el espacio de memoria medio de las aplicaciones de pago, que NO sean nativas (NO estén en una única tienda)

Para esta consulta, junto con la función de agregación **avg**, mediante una subconsulta se obtienen aquellas aplicaciones que solo aparezcan una vez en la tabla **contiene**, es decir, solo aparezcan en una única tienda, comprobando de este modo si el nombre de la aplicación **no se encuentra en dicho subconjunto**:

```
SELECT round(avg(PRECIO),2) as PRECIO_MEDIO, round(avg(ESPACIO),2) as
ESPACIO_MEDIO
FROM aplicacion
WHERE PRECIO != 0 AND NOMBRE NOT IN
(SELECT NOMBRE_APLICACION
FROM contiene
GROUP BY NOMBRE_APLICACION
HAVING count(NOMBRE_TIENDA) = 1);
```

	PRECIO_MEDIO	ESPACIO_MEDIO
▶	8.54	64.34

Ilustración 32. Resultado consulta 10

Consulta 11

Obtener el VAT y nombre de las empresas que han participado en el desarrollo de aplicaciones nativas (esten en una unica tienda)

Para esta consulta, se recorren las tablas **Empresa**, **crea** y **Aplicacion**, contando el número de participaciones por empresa, además de (mediante una subconsulta) comprobar que la aplicación solo se encuentra en una única tienda:

```
SELECT e.VAT, e.NOMBRE
FROM empresa AS e INNER JOIN crea AS c ON e.VAT = c.VAT
INNER JOIN aplicacion AS a ON c.NOMBRE = a.NOMBRE
WHERE a.NOMBRE IN

(SELECT NOMBRE_APLICACION FROM contiene GROUP BY NOMBRE_APLICACION HAVING
count(NOMBRE_TIENDA) = 1);
```

VAT	NOMBRE
ES12345679	DigitalValue
U12345679	MajorApp
DE1232645120	WunderDev

Ilustración 33. Salida consulta 11

Consulta 12

Obtener las categorías con una puntuación acumulada en sus aplicaciones mayor a 100, cuyo país de descarga sea España

En esta consulta, mediante las tablas **Categoria**, **categoría_aplicacion**, **Aplicación** y **descarga** se obtiene la suma de las puntuaciones de cada aplicación, agrupadas por cada categoría, con la condición (HAVING) de ser mayor a 100. Por último, desde la tabla **Usuario** se filtran aquellas filas cuyo país de origen sea España:

```
SELECT c.NOMBRE, sum(d.PUNTUACION) as PUNTUACION_ACUMULADA
FROM categoria AS c INNER JOIN categoria_aplicacion AS c_a ON
c.ID_CATEGORIA = c_a.ID_CATEGORIA
INNER JOIN aplicacion AS a ON c_a.NOMBRE = a.NOMBRE
INNER JOIN descarga AS d ON a.NOMBRE = d.NOMBRE
INNER JOIN usuario AS u ON d.NUM_CUENTA = u.NUM_CUENTA
WHERE u.PAIS = 'Espana'
GROUP BY c.NOMBRE
HAVING PUNTUACION_ACUMULADA > 100
ORDER BY PUNTUACION_ACUMULADA DESC;
```

NOMBRE	PUNTUACION_ACUMULADA
Social	226
Entretenimiento	181
Estilo de vida	148

Ilustración 34. Salida consulta 12

Consulta 13

Obtener el DNI y teléfono de aquellos empleados que se hayan descargado 7 aplicaciones o menos

Dado que la tabla **descarga** contiene el número de teléfono, mediante un EXISTS se comprueba si existen filas en la tabla descarga cuyo número de teléfono coincide con alguno de la tabla Empleado, con la condición adicional de haberse descargado 7 aplicaciones o menos:

```
SELECT e.TLFNO_MOVIL, e.DNI
FROM empleado AS e
WHERE EXISTS

(SELECT NUM_CUENTA, count(NOMBRE) as DESCARGAS
FROM descarga AS d
WHERE d.NUM_MOVIL = e.TLFNO_MOVIL
GROUP BY NUM_CUENTA
HAVING DESCARGAS <= 7);
```

TLFNO_MOVIL	DNI
601509950	73307151X
652215230	93155413P
699219990	14719687D
709998010	82728719L
730993990	04227807Q
793093440	37276876V
799991210	81307095W

Ilustración 35. Salida consulta 13

Nota: si se quisiera consultar aquellos empleados que no se han descargado alguna aplicación, bastaría con un LEFT JOIN entre Usuarios y descargas (donde NUM_CUENTA sea NULL en descargas):

NUM_CUENTA	NOMBRE	CALLE	NUMERO	CP	PAIS
974552	remedios2	Travesia Lorem	59	10921	Espana

Ilustración 36. Salida LEFT JOIN

Consulta 14

Consultar las aplicaciones con más de 2 categorías, realizadas entre los años 2014 y 2020, cuyo espacio en memoria no supere los 60 MB, y que el porcentaje de descargas (con respecto al total de usuarios) sea mayor al 40 %

Inicialmente se filtran aquellas aplicaciones con más de 2 categorías, a través de una subconsulta en *categoría_aplicacion*, filtrando aquellas cuyo nombre se repita más de 2 veces en la tabla intermedia.

A continuación, una subconsulta desde el WHERE recupera el número de filas de las tablas **Usuarios** y **descargas** mediante **count(*)**, con el objetivo de calcular el porcentaje de descargas (x 100), filtrando aquellos valores mayores a 40, además de aquellas aplicaciones cuya fecha de inicio y fin esté comprendida entre 2014 y 2020, así como el espacio de memoria (inferior a 60):

```
SELECT distinct(a.NOMBRE)
FROM aplicacion AS a INNER JOIN descarga AS d ON a.NOMBRE = d.NOMBRE
INNER JOIN

(SELECT NOMBRE FROM categoria_aplicacion GROUP BY NOMBRE HAVING
count(NOMBRE) > 2) AS t ON d.NOMBRE = t.NOMBRE

WHERE year(a.FECHA_INI) >= 2014 AND year(a.FECHA_FIN) <= 2020
AND a.ESPACIO < 60 AND a.NOMBRE IN
```

```
(SELECT NOMBRE
FROM descarga
GROUP BY NOMBRE
HAVING count(*) * 100 / (SELECT count(*) FROM usuario) > 40);
```

NOMBRE
PhotoPills
Twitter

Ilustración 37. Salida consulta 14

Consulta 15

Consultar empleados cuyas aplicaciones en las que hayan participado tengan un número de descargas superior a 30 y una puntuación media mayor a 3. Además, dichos empleados deben tener entre 2 y 5 años de experiencia en alguna empresa.

Para conocer aquellas aplicaciones con más de 30 descargas y con una puntuación media mayor a 3, desde el FROM se realiza una subconsulta, recuperando aquellos nombres de aplicación cuyo número de descargas (count(NOMBRE)) sea superior a 30 y con una media de puntuación mayor a 3 puntos.

Por otro lado, mediante una segunda subconsulta se recuperan aquellos empleados (de la tabla **Empleado**), sumando los años de experiencia de cada uno, filtrando aquellos comprendidos entre 2 y 5 años (mediante **timestampdiff**).

```
SELECT e.*
FROM empleado AS e INNER JOIN realiza AS r ON e.DNI = r.DNI
INNER JOIN

(SELECT NOMBRE FROM descarga GROUP BY NOMBRE HAVING count(NOMBRE) > 30 AND
avg(PUNTUACION) > 3) as descarga ON r.NOMBRE = descarga.NOMBRE

WHERE e.DNI IN

(SELECT distinct(t.DNI)
FROM empleado AS e INNER JOIN trabaja AS t ON e.DNI = t.DNI
GROUP BY t.DNI
HAVING sum(timestampdiff(year, t.FECHA_INI, t.FECHA_FIN)) BETWEEN 2 AND 5)
GROUP BY e.DNI;
```

DNI	CORREO	TLFNO_FIJO	TLFNO_MOVIL	CALLE	NUMERO	CP
17158772J	MaitePozo@gmail.com	923190330	649139890	C. Comercial Espacio Leon	120	09190
29707354W	AntonioFGTE@gmail.com	999399080	633313210	Plaza de la Habana	13	00001

Ilustración 37. Salida consulta 15

Consulta 16

Consultar aquellas empresas que hayan participado en dos o más aplicaciones de tipo Social o de Entretenimiento, que tengan menos de 4 ceros en las puntuaciones por parte de los usuarios

Para esta última consulta, se han realizado en varias subconsultas. En primer lugar, desde las tablas **Aplicacion**, **categoria_aplicacion** y **categoria** filtrar aquellas aplicaciones de tipo Social o de Entretenimiento, mediante la sentencia IN. A continuación, de estas aplicaciones filtrar aquellas que tengan menos de 4 ceros o (UNION) ninguno. Finalmente, comprobamos qué empresas han trabajado en más de dos de estas aplicaciones obtenidas. En este caso, la consulta devuelve una única empresa.

```
SELECT c.VAT
FROM crea AS c
WHERE c.NOMBRE IN
(
```

```
SELECT distinct(a.NOMBRE)
FROM aplicacion AS a
INNER JOIN categoria_aplicacion AS c_a ON a.NOMBRE = c_a.NOMBRE
INNER JOIN categoria AS ca ON c_a.ID_CATEGORIA = ca.ID_CATEGORIA
WHERE ca.NOMBRE IN ('Social', 'Entretenimiento') AND a.NOMBRE IN

(SELECT distinct(NOMBRE) FROM descarga WHERE PUNTUACION = 0 GROUP BY NOMBRE
HAVING count(PUNTUACION) < 4

UNION

SELECT distinct(d.NOMBRE) FROM descarga AS d WHERE NOT EXISTS (SELECT
d_1.NOMBRE FROM descarga AS d_1 WHERE d_1.NOMBRE = d.NOMBRE AND
d_1.PUNTUACION = 0))

)
GROUP BY c.VAT
HAVING count(c.VAT) >= 2;
```

VAT
ES12345679

Ilustración 38. Salida consulta 16