BASES DE DATOS

## **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-Relacion como del Modelo relacional ha sido llevado a cabo mediante la herramienta **draw.io**

### **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**)
* Gestor de la tienda
* Dirección web de la tienda física

Dado que la longitud de cualquiera de los tres campos es variable, el dominio de cada columna consistirá en una **cadena de caracteres de longitud variable**, tal y como se especificará en el modelo relacional.

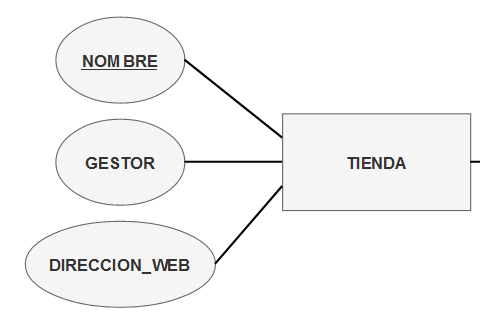


Ilustración 1. Entidad Tienda

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

* VAT (**clave primaria**)
* Nombre
* Pais tributario
* Año de creación
* Correo electrónico
* Página web

El dominio del campo **VAT** se reducirá a una cadena de caracteres de longitud variable hasta 12 caracteres como máximo estipulado, dado que el código de cada país europeo es de diferente 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)

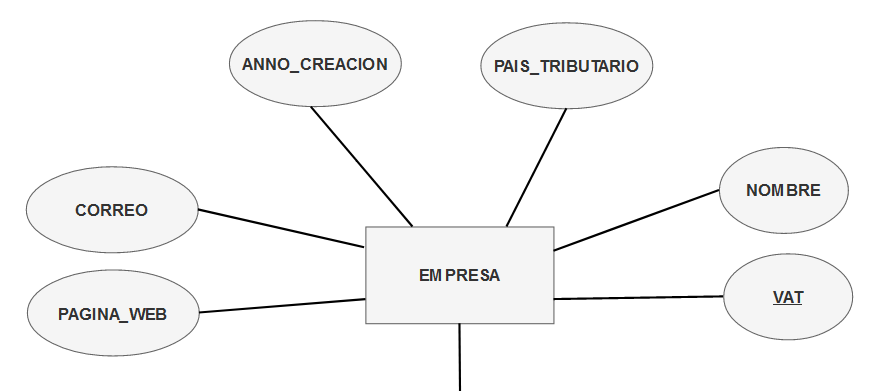


Ilustración 2. Entidad Empresa

### **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.*

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
* Dirección, un atributo **compuesto** 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
* Correo electrónico, un campo candidato a ser clave primaria (dado que es único para cada empleado dentro de una empresa), por lo que deberá ser único
* Teléfono, un atributo multivalorado, formado por los campos teléfono fijo y teléfono móvil.

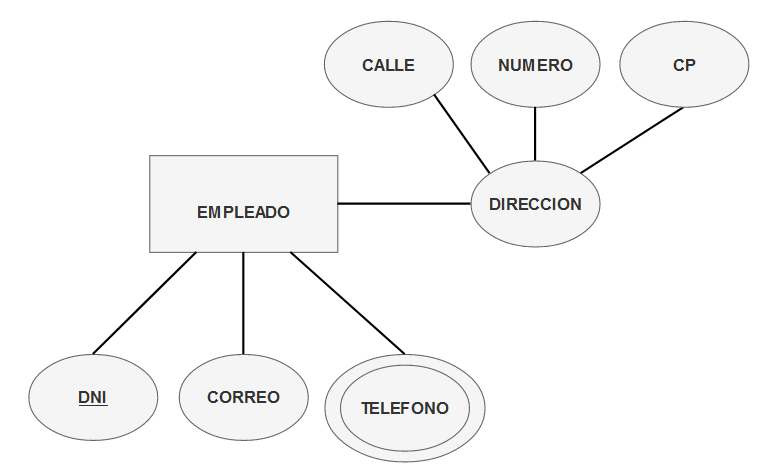


Ilustración 3. Tabla empleado

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** y, por otra parte, en una empresa pueden trabajar uno o varios empleados. Considerando que un empleado debe haber trabajado, como mínimo, en una empresa, así como en una empresa debe trabajar como mínimo un empleado, la cardinalidad de la relación será **N:N**, tal y como se muestra a continuación:

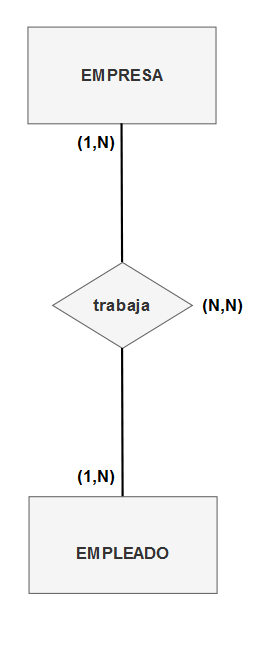


Ilustración 4. Relación trabaja (I)

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 relacional, no lo permitiría, ya que no se tiene constancia de las diferentes fechas en las que ha trabajado el empleado, por lo que faltaría por añadir dos nuevos campos a la relación N:N: la **fecha de inicio** y la **fecha de fin** en el que un empleado trabajó en una empresa:

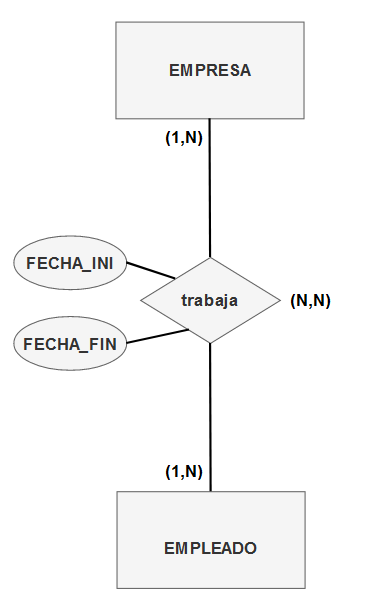


Ilustración 5. Relación trabaja (II)

### **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.*

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
* Código de la aplicación, un campo único, de cara al modelo relacional
* La fecha de inicio de desarrollo de la aplicación
* La fecha de fin de desarrollo de la aplicación
* La/s categoría/s de la aplicación. Dado que una aplicación puede pertenecer a varias categorías (entretenimiento, social, educativo etc.) se trata de un atributo multivalorado.
* El espacio de memoria
* El precio de la aplicación. Dado que puede haber aplicaciones gratuitas, el valor por defecto será cero

Por otro lado, cabe destacar que la entidad **Aplicación** es una entidad débil, en relación con la entidad **Empleado**. Como se detallará a continuación, una aplicación es dirigida por un empleado, que a su vez puede dirigir múltiples aplicaciones. Como consecuencia, la existencia de una aplicación depende de quién dirige su desarrollo.

Diagrama

Descripción generada automáticamente

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 y, como máximo, N aplicaciones. Por otro lado, una aplicación puede estar alojada en 1 o N tiendas como mínimo y máximo, respectivamente. Por tanto, la cardinalidad será N:N.

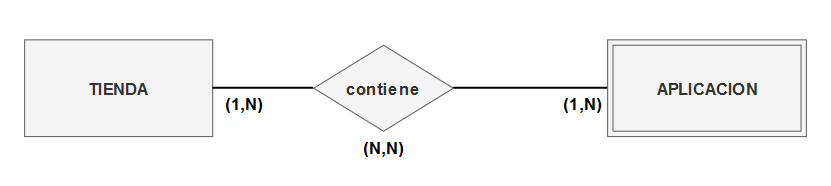


Ilustración 7. Relación tienda – aplicación

*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, y 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 un proyecto). Sin embargo, una aplicación es dirigida por uno y solo un empleado, por lo que la cardinalidad entre ambas entidades será 1:N

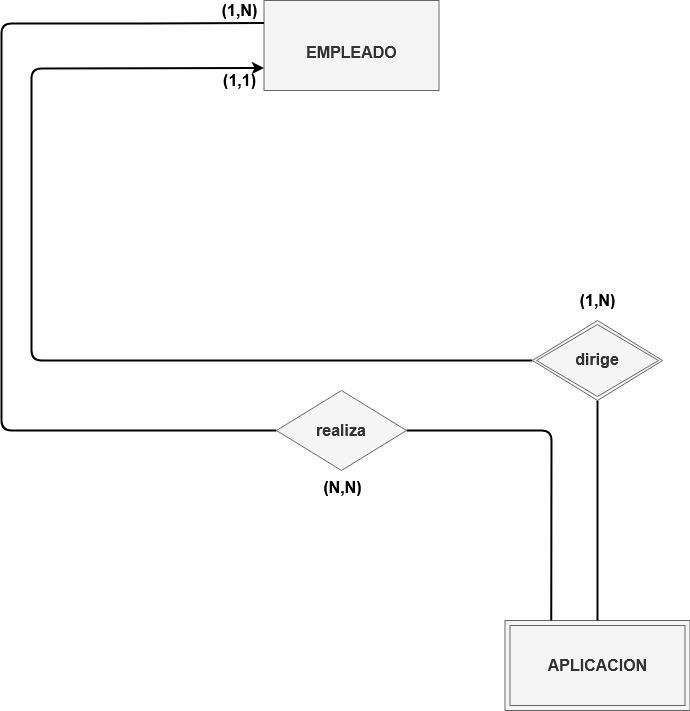


Ilustración 8. Relaciones realiza, dirige

### **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
* Nombre del usuario, un campo único, de cara al modelo relacional
* Dirección que, al igual que en la entidad Empleado, se trata de un campo **compuesto** por calle, número y código postal
* País del usuario

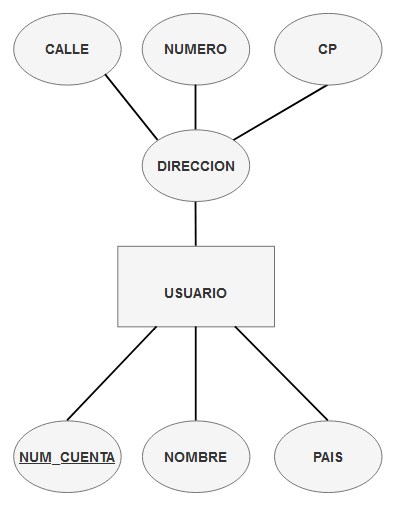


Ilustración 9. Entidad usuario

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. Dado que un usuario puede puntuar la aplicación descargada, realizar comentarios sobre la misma, su número de móvil en caso de descarga así como la fecha en la que se realizó, sobre la relación N:N añadimos dichos campos:

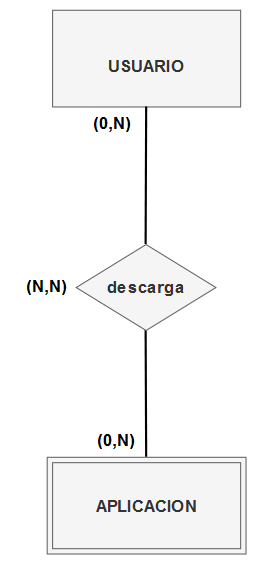
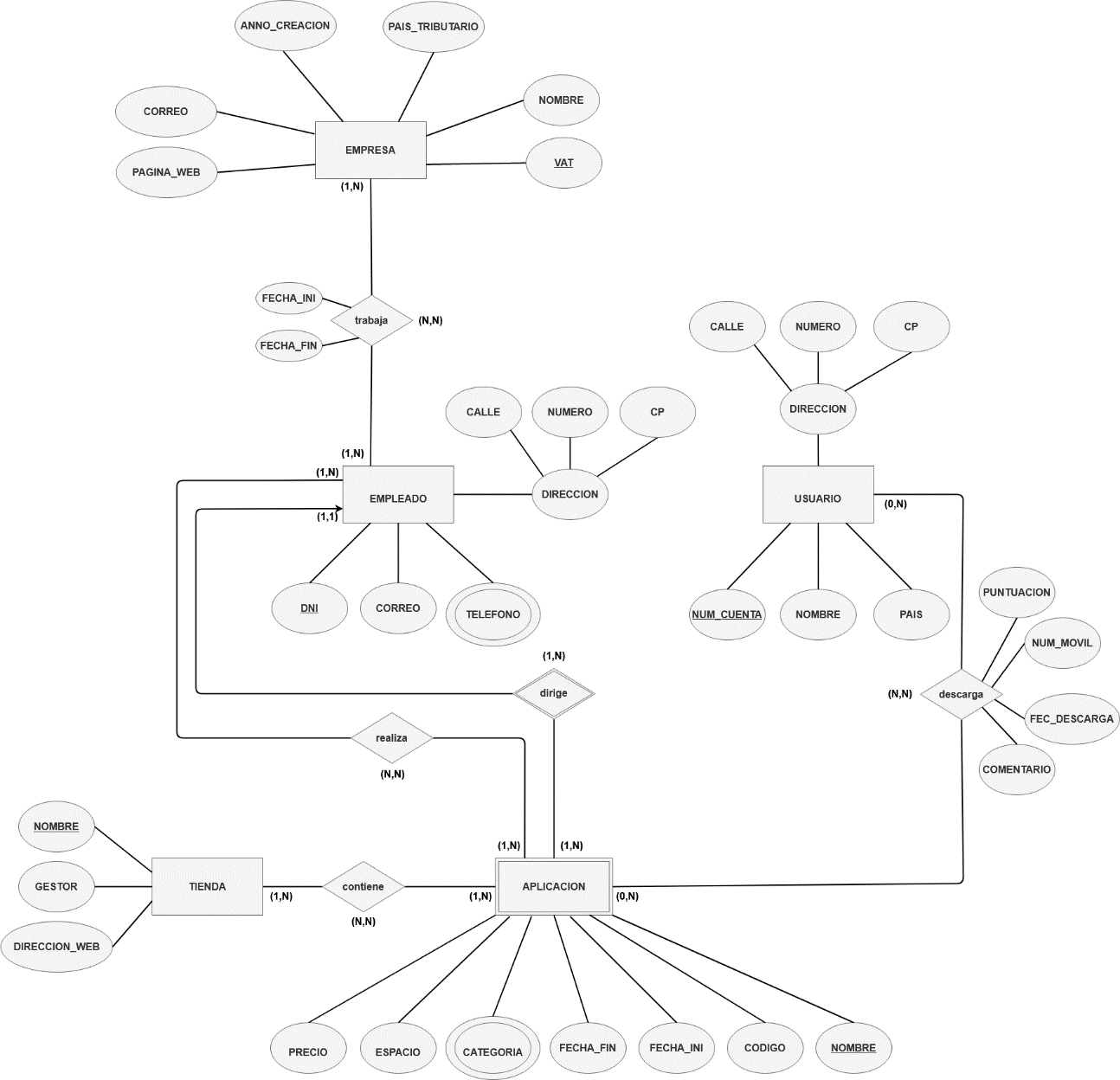


Ilustración 10. Relación descarga

Añadiendo cada entidad y relación anterior, el modelo queda del siguiente modo:



## **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 una *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 un *Varchar* de tamaño medio (20 caracteres).

En cualquiera de los tres atributos, **ninguno de ellos puede ser NULL**.

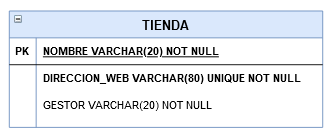


Ilustración 11. Tabla tienda

### **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 debe ser único para cada empresa, lo marcamos como clave candidata
* **Página web**: un *Varchar* de 80 caracteres (al igual que la tabla Tienda), un campo único para cada empresa (pues la dirección de cada página es única), por lo que lo marcamos como clave candidata
* 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

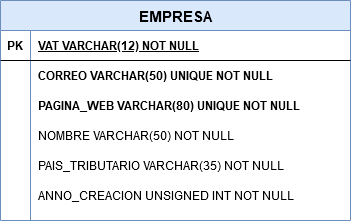


Ilustración 12. Tabla empresa

En cualquiera de los atributos, **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 de caracteres fija de longitud 9 (8 dígitos más letra).
* **Correo**: un *Varchar* de mayor longitud (50 caracteres). Dado que el nombre de correo debe ser único para cada empleado, lo marcamos como clave candidata
* **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, se trata de una clave candidata, por lo que se marca como UNIQUE.
* Teléfono fijo: un campo entero **sin signo**. A diferencia del campo anterior, un teléfono fijo puede ser compartido por varios empleados (familiares), por lo que puede repetirse.

El campo 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 hasta 3 dígitos). Dado que una calle puede no tener número, en caso de no insertar un número (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 en lugar de un entero por la omisión de ceros en algunos códigos. A modo de ejemplo, el código postal 00610, en caso de tratarse de un entero, quedaría insertado en la tabla como 610. Por el contrario, mediante un *Varchar* se conservan ambos ceros en la tabla.

Salvo el campo número, en cualquiera de los atributos **ninguno de ellos puede ser NULL**.

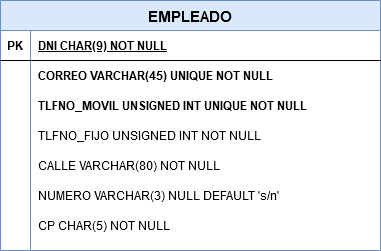


Ilustración 13. Tabla empleado

### **TRABAJA**

Una vez diseñada tanto la tabla Empresa como Empleado, deben unirse por medio de relaciones. Dado que la relación definida en el modelo anterior N:N, 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 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 empleado en la misma empresa, siempre y cuando la fecha de inicio sea diferente:

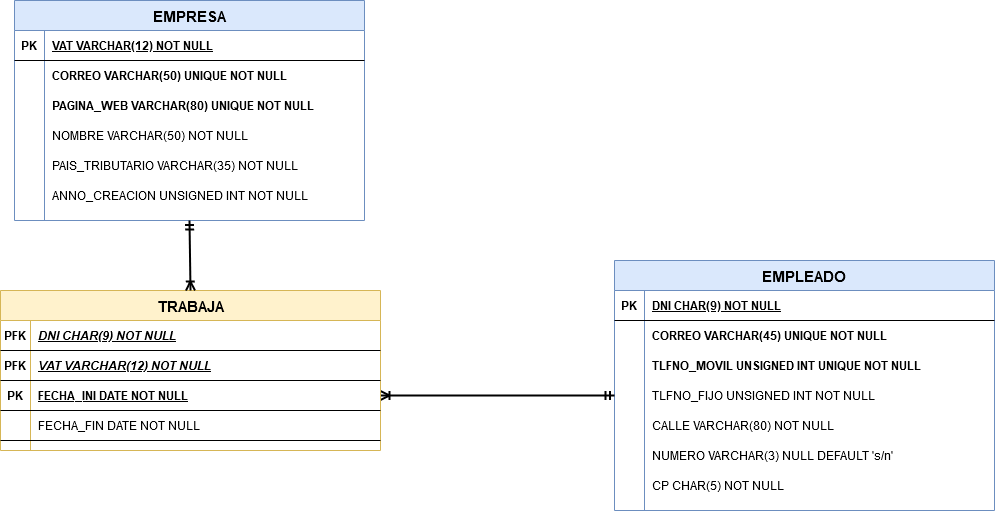


Ilustración 14. Relación trabaja

### **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: 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, se trata de una relación 1:N, por lo que en lugar de una tabla intermedia, la tabla Aplicación deberá almacenar, como clave foránea, el DNI del empleado encargado de gestionarla

En cualquiera de los atributos, **ninguno de ellos puede ser NULL**. Por otro lado, dado que la clave foránea *Empleado \_DNI* no forma parte de la clave primaria de la tabla Aplicación, la relación es **no identificativa**, marcado mediante líneas discontinuas:

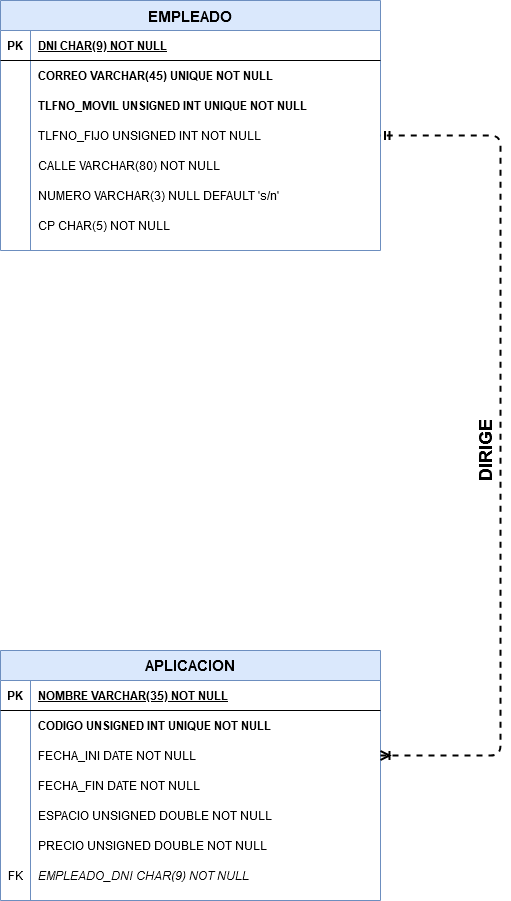


Ilustración 15. Entidad aplicación

### **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 múltiples. Por ello, en lugar de descomponer el atributo en múltiples campos, la categoría se traducirá en una nueva tabla, relacionada con la tabla 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.

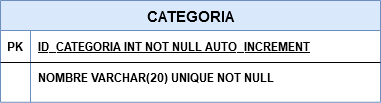
****

Ilustración 16. Entidad categorías

### **CATEGORIA\_APLICACION**

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

* ***NOMBRE***: Aplicación
* ***ID\_CATEGORÍA***: Aplicación

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:

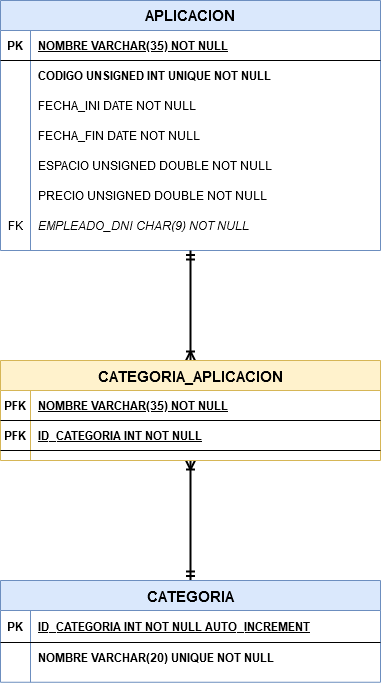


Ilustración 17. Relación categoría\_aplicacion

### **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 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:

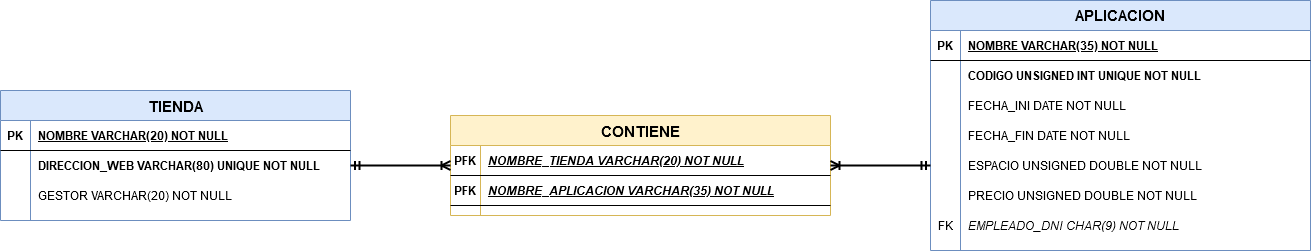


Ilustración 18. Relación contiene

### **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.

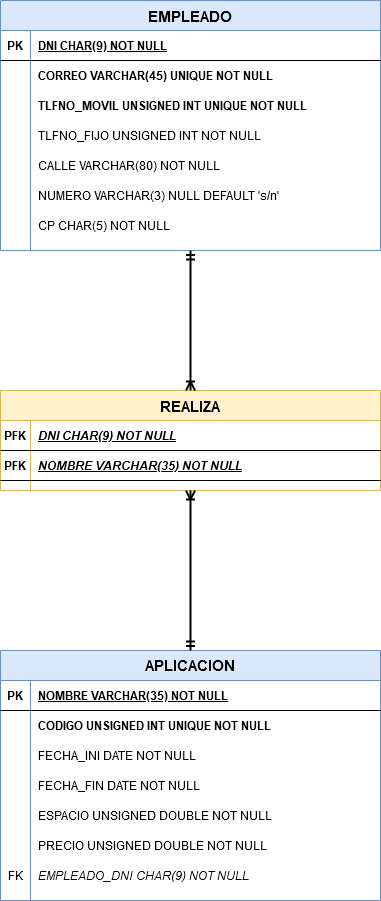


Ilustración 19. Relación realiza

### **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:

* Calle: un *Varchar* de mayor longitud (80 caracteres)
* Número: un *Varchar* de hasta 3 caracteres (permitiendo un número de calle hasta 3 dígitos). Dado que una calle puede no tener número, en caso de no insertar un número (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 en lugar de un entero por la omisión de ceros en algunos códigos
* País: un *Varchar* de tamaño medio (35 caracteres)

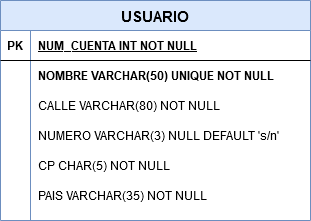


Ilustración 20. Tabla usuario

### **DESCARGA**

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

* ***NUM\_CUENTA***: Usuario
* ***NOMBRE***: Descarga

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 mediante un tipo *Text*, lo que permite no limitar el campo a un número de caracteres

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

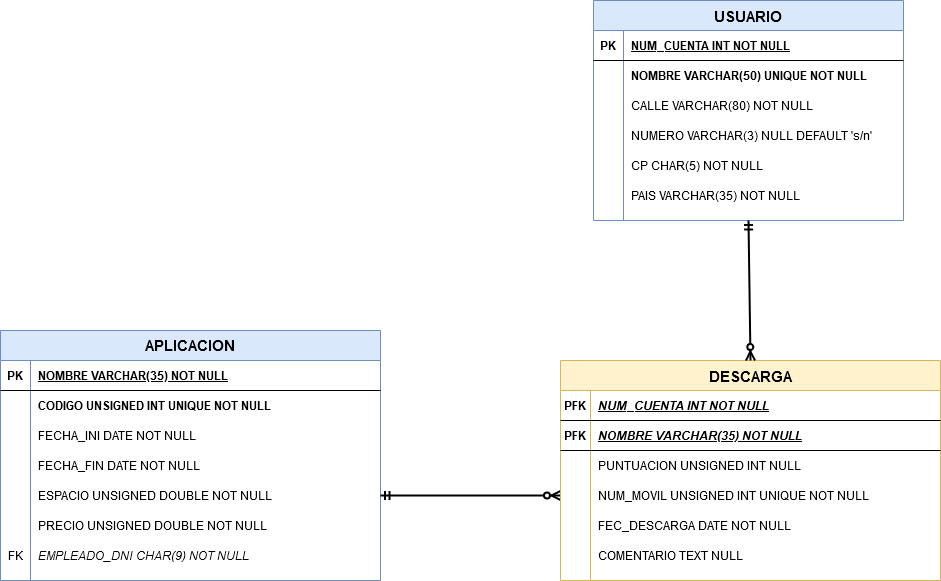
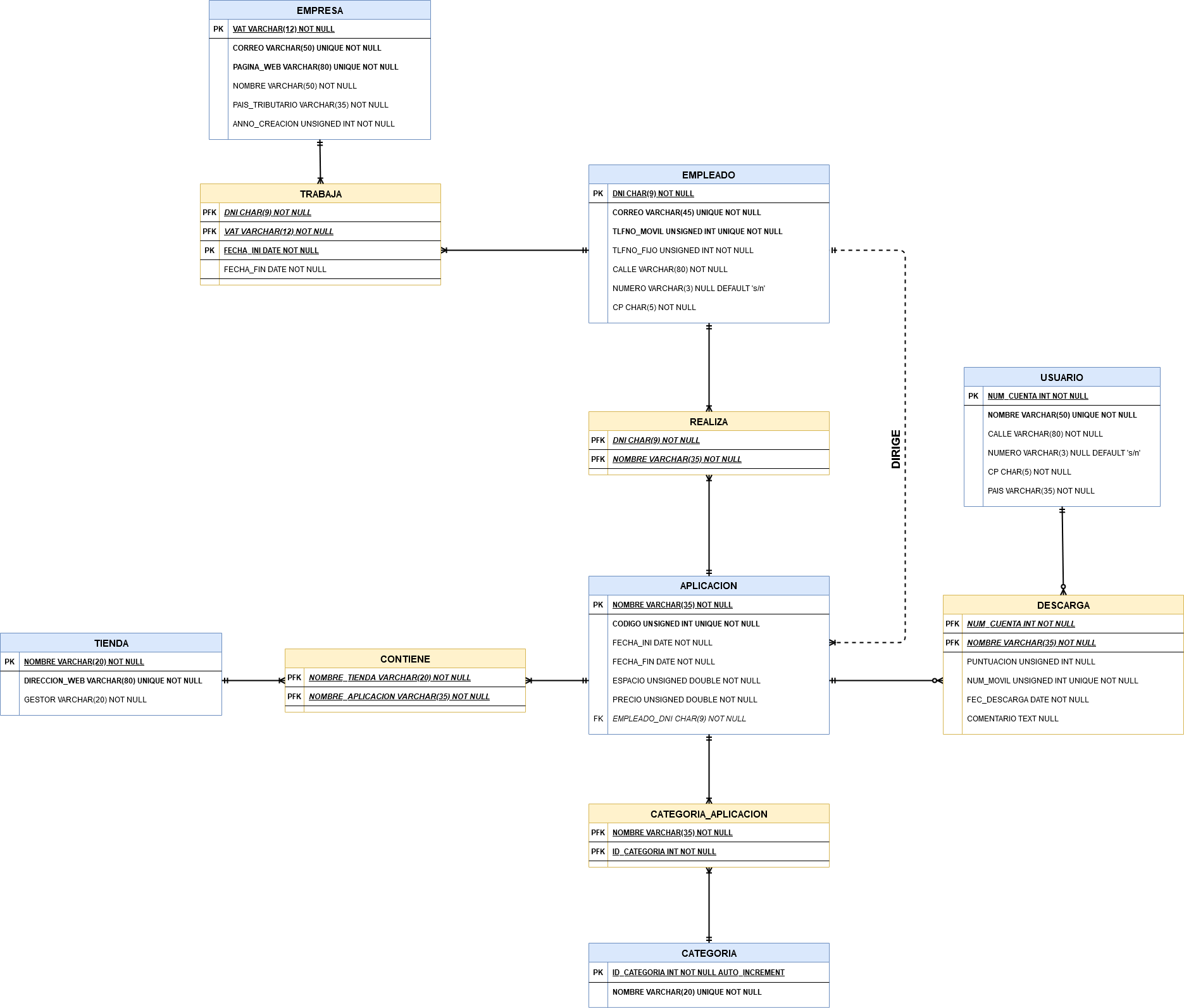


Ilustración 21. Relación descarga

Tras diseñar cada una de las tablas y relaciones, el modelo relacional resultante quedará de la siguiente forma:



## **3. IMPLEMENTACIÓN SQL**

La implementación en SQL se ha realizado conforme al modelo relacional previamente diseñado, mediante el *script* FernandezHernandezAlberto.sql. En el 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 la tabla trabaja, por cada inserción debe comprobarse que el campo FECHA\_INI sea estrictamente menor a FECHA\_FIN. Del mismo modo, debe comprobarse que el espacio de una aplicación sea estrictamente mayor que cero (a diferencia del precio, el cual puede ser cero, esto es, gratuito):

**CHECK** **(**FECHA\_INI **<** FECHA\_FIN**),**

**CHECK** **(**ESPACIO **>** 0**)**

#### **RELACIÓN DESCARGA**

Por último, tal y como se mencionó en los modelos anteriores, el valor de puntuación de una aplicación debe estar comprendida 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 de un empleado (RESTRICT).

##### 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 del empleado que gestiona la aplicación.

##### ON DELETE

Según el modelo Entidad-Relación definido, si el empleado es borrado de la base de datos, al tratarse de una entidad débil, la aplicación debería borrarse. 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. Por el contrario, al marcar el proceso de borrado como RESTRICT permitiría mantener la aplicación, teniendo que modificar únicamente la clave foránea, asociándolo con un nuevo jefe de proyecto.

##### 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** **RESTRICT**

**ON** **UPDATE** **CASCADE);**

#### **RELACIÓN REALIZA**

##### ON DELETE

Al igual que la relación trabaja, si el DNI del empleado desapareciera, implica que dejaría trabajar en la aplicación correspondiente, por lo que los datos serían inconsistentes. Por ello, si un empleado es borrado de la tabla, el proceso se propaga hasta la tabla realiza (CASCADE). 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), 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 CONTIENE**

##### ON DELETE

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 nativas que alojaba (exclusivas de Apple), dejarían de estar disponibles, por lo que borrado de la tienda implicaría eliminar todas aquellas aplicaciones relacionadas con dicha tienda. 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 CATEGORIAS\_APLICACION**

##### ON DELETE

Si una categoría en particular desapareciera, todas las entradas que hagan referencia a dicha categoría en la tabla categorias\_aplicacion deben ser borradas en cascada, del mismo modo que si una aplicación dejara de existir, todas su 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** CATEGORIAS **(**ID\_CATEGORIA**)**

**ON** **DELETE** **CASCADE**

**ON** **UPDATE** **CASCADE);**

#### **RELACIÓN DESCARGA**

##### ON DELETE

En el caso del borrado, existe una diferencia en relación con el resto de relaciones con la tabla aplicación (realiza, contiene). Si un usuario dejara de existir, al igual que ocurría con el resto de 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 de descargas inferior al real, pues algunos de estos usuarios han podido haber cerrado 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 no 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 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 INSERT de prueba se demostrará su correcto funcionamiento.

### **PROCEDURE**

Dicho procedimiento, denominado **comprobar\_fecha**, evalua 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 el *trigger*, comprobar si la letra calculada es la misma a la que aparece en el DNI a insertar:

**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 ejecutan **antes de la inserción**.

##### 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, lo llamará dentro del *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'**);**





##### 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'**);**





##### 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 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!!!!!!'**);**





**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!!!!!!'**);**





##### 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 valido"**;**

**END** **IF;**

**END**$$

**Comprobación (DNI No válido)**:

**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'**);**





##### 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 previamente en esta tabla. Para ello, comprueba si el DNI del responsable está incluida 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 esta en la tabla Realiza"**;**

**END** **IF;**

**END**$$

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





**Los ejemplos anteriores están incluidos al final del *script***

## **4. CONSULTAS**