



Material de Apoyo TC2

Diagrama Entidad Relación (E/R)

Un diagrama entidad relación es una herramienta ampliamente utilizada en la modelación de una base de datos. Su principal utilidad es describir de manera sencilla cómo se ve un modelo de datos. Para esto se señalan las entidades en **rectángulos**, las relaciones entre entidades en **rombos** y los atributos en **óvalos**. Consideremos la siguiente figura:

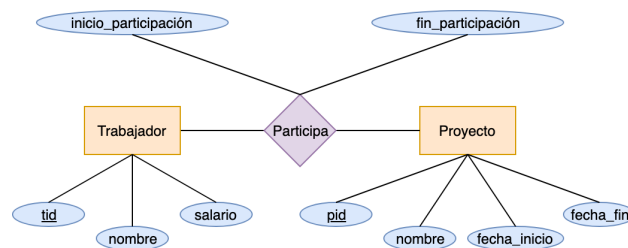


Figura 1: Ejemplo diagrama E/R.

Este diagrama representa el modelo de una empresa que tiene trabajadores y proyectos. Un trabajador puede participar en varios proyectos y un proyecto puede tener muchos trabajadores. En este caso las entidades con sus respectivos atributos son:

- **Trabajador.** Sus atributos son:
 - **tid:** es un identificador del trabajador. Este atributo es la *primary key*, que recordemos, es el atributo que debe ser distinto para cada una de las filas de la tabla.
 - **nombre:** representa el nombre del trabajador.
 - **salario:** representa el salario del trabajador.
- **Proyecto.** Sus atributos son:
 - **pid:** es un identificador del proyecto. Este atributo es la *primary key*.
 - **nombre:** representa el nombre del proyecto.
 - **fecha_inicio:** representa la fecha de inicio del proyecto.
 - **fecha_fin:** representa la fecha de fin del proyecto.

Además tenemos la relación **Participa**. Esta relación nos dice qué trabajador participó en qué proyecto. Notemos que esta relación también tiene atributos, que son:

- **inicio_participación**: representa la fecha en que un trabajador comenzó a trabajar en un proyecto en particular.
- **fin_participación**: representa la fecha en que un trabajador terminó de trabajar en un proyecto en particular.

¿Y dónde está la *primary key* de la relación **Participa**? Es importante notar que **siempre** la *primary key* de una relación es la combinación de la *primary key* de todas las entidades que se ven involucradas gracias a ella. Además, no es necesario volver a señalarlas explícitamente. En este caso sería el par (**tid**, **pid**). Vamos a explicar más adelante qué significa que la *primary key* sean dos valores.

Para el ejemplo anterior, tenemos que su transformación a tablas sería la siguiente¹:

```
Trabajador(tid INT PRIMARY KEY, nombre VARCHAR(100), salario INT)
```

```
Proyecto(pid INT PRIMARY KEY, nombre VARCHAR(100),  
         fecha_inicio DATE, fecha_fin DATE)
```

```
Participa(tid INT, pid INT,  
          inicio_participación DATE, fin_participación DATE,  
          PRIMARY KEY(tid, pid))
```

Es importante notar que las entidades y las relaciones son tablas. Como dijimos anteriormente, del diagrama se desprende que **tid** y **pid** son atributos de la relación **Participa** y que no necesitamos escribirlo explícitamente en el diagrama E/R. Además, estos dos atributos forman la *primary key* de esta tabla. Eso quiere decir que no pueden haber dos filas de la tabla **Participa** en las que coincidan el **tid** y el **pid** juntos.

Por ejemplo, podemos tener dos tuplas:

```
(1, 1, 01/01/2010, 01/01/2011) y (1, 2, 01/04/2011, 01/05/2012)
```

que indican que el trabajador 1 trabajó en el proyecto 1 y el 2. Sin embargo, no podremos tener una tupla:

```
(1, 2, 01/04/2012, 01/05/2013)
```

Porque ya se indicó anteriormente que el trabajador 1 participó en el proyecto 2 en otras fechas. Si te estás preguntando si hay una forma en la que un trabajador pueda volver a participar de un proyecto, la hay. Veremos al final de esta guía como hacer ese estilo de modelaciones.

¹Notemos que no vamos a hablar de *foreign keys* en esta ayudantía, ya que ese tópico es más de lo que se busca enseñar en este curso.

Tipos de relaciones

Muchos a muchos

En el caso anterior describimos una relación de tipo **muchos a muchos**, porque un trabajador puede participar en muchos proyectos, y un proyecto puede tener muchos trabajadores. Este tipo de relaciones también son llamadas **N a N**. Este tipo de cardinalidades las podemos denotar en el diagrama como se muestra en la siguiente figura²:



Figura 2: Ejemplo de relación N a N.

Uno a muchos

Uno no siempre quiere tener relaciones N a N. A veces queremos que una entidad se relacione únicamente una vez con otra. Por ejemplo, consideremos equipos de fútbol y jugadores. Un jugador puede ser parte solamente de un equipo de fútbol, pero un equipo tiene muchos jugadores. Este es un ejemplo de una relación **uno a muchos**, que se denota en el diagrama como **1 a N**. Veamos la siguiente figura:

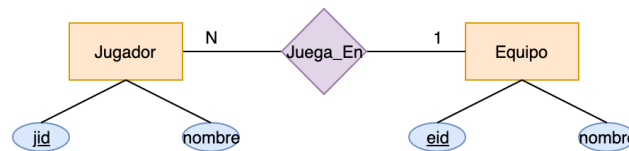


Figura 3: Ejemplo de relación 1 a N.

²Hemos borrado los atributos por simplicidad.

Como dijimos anteriormente, un jugador juega en 1 equipo, pero un equipo puede tener **N** jugadores. Nota cómo se señala esto en el diagrama. Ahora bien, aquí señalamos las llaves primarias de cada entidad para hacer notar algo: en la relación **Juega_En** cada jugador puede aparecer a lo más una vez, por lo que la llave primaria de la tabla **Juega_En** es solamente el atributo **jid**. En este caso, la tabla se vería de la siguiente forma:

```
Juega_En(jid INT, eid INT, PRIMARY KEY(jid))
```

Es importante que entiendas por qué cada jugador puede aparecer a lo más en una única tupla de la tabla **Juega_En**. Una vez que lo entiendas, la siguiente afirmación te hará sentido: podemos omitir la tabla **Juega_En** y solo quedarnos con **Jugador** y **Equipo**. A **Jugador** le vamos a anexar un atributo **eid** y las tablas resultantes del diagrama serían solamente dos:

```
Jugador(jid INT PRIMARY KEY, nombre VARCHAR(100), eid INT)
```

```
Equipo(eid INT PRIMARY KEY, nombre VARCHAR(100))
```

Ahora el equipo en el que participa un jugador se indica en la misma tabla de **Jugador**.

Restricciones de cardinalidad

Tomemos el ejemplo anterior. Podemos añadir restricciones de cardinalidad más específicas. Por ejemplo considera la siguiente figura:

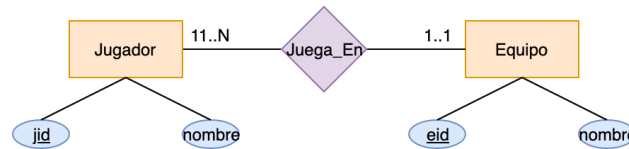


Figura 4: Ejemplo de más restricciones de cardinalidad.

En este ejemplo estamos señalando que cada jugador debe estar en a lo menos un equipo y a lo más un equipo. Si cambiamos el **1..1** por **0..1** podemos tener jugadores cesantes (es decir, no están en ningún equipo). Mientras que estamos señalando que cada equipo tiene al menos 11 jugadores, hasta una cantidad ilimitada.

Roles

Supongamos el ejemplo de una red social, en la que un usuario puede seguir a otro usuario. En este caso debemos relacionar una entidad consigo misma. Esto se puede denotar usando roles, como se muestra en la siguiente figura:

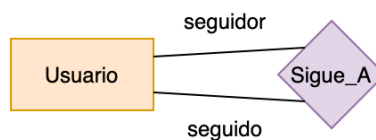


Figura 5: Ejemplo de roles en un diagrama E/R.

En este caso, el usuario participa de la relación con dos roles: como seguidor (el que sigue a una persona) y como seguido (el usuario que es seguido). Una posible forma de las tablas resultantes serían las siguientes:

```
Usuario(uid INT PRIMARY KEY, nombre VARCHAR(100))
```

```
Sigue_A(uid_seguidor INT, uid_seguido INT)
```

Modelación más compleja

Supongamos el ejemplo de una tienda en línea, que tiene usuarios y productos a la venta. Para poder modelar las compras tenemos que relacionar los usuarios con los productos. Una primera idea sería hacer el siguiente diagrama:

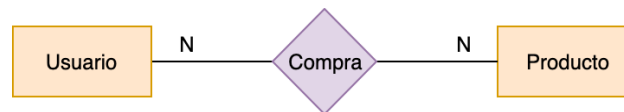


Figura 6: Primera versión tienda en línea.

Sin embargo, ya sabemos que la tabla **Compra** tiene que tener como *primary key* el *id* del usuario junto al *id* del producto. Pero esto significa que cada usuario puede comprar un producto una única vez, lo que no hace mucho sentido. Una primera idea que nos gustaría intentar sería agregar una *primary key* a la relación, algo así como un *id_compra*. Sin embargo, esto no está permitido en el diagrama entidad relación.

Para resolver este problema, una solución típica es el siguiente diagrama:

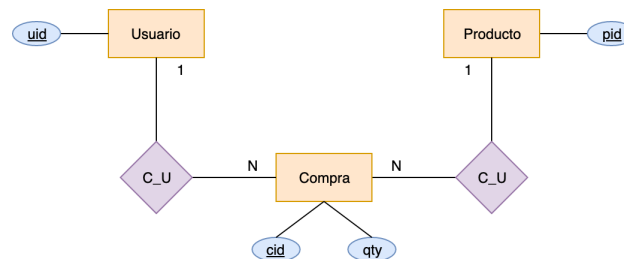


Figura 7: Versión final tienda en línea.

Como vemos, creamos una entidad compra que tiene una relación 1 a N con los usuarios y 1 a N con los productos. Gracias a esto, nosotros no creamos tablas para las relaciones, y las tablas resultantes tienen la siguiente forma³:

```
Usuario(uid INT PRIMARY KEY, nombre VARCHAR(100))
```

```
Producto(pid INT PRIMARY KEY, nombre_producto VARCHAR(100), precio FLOAT)
```

³Estamos creando unos atributos que intuitivamente estarían en una tienda.

`Compra(cid INT PRIMARY KEY, uid INT, pid INT, qty INT, fecha DATE)`

Notando que el atributo `qty` es la cantidad de veces que se compro el producto en aquella compra. Así, un usuario puede volver a adquirir el producto, porque eso genera otra compra. Una posible pregunta es qué pasa si en una compra queremos tener varios productos. En ese caso, la relación entre `Compra` y `Producto` es N a N, lo que modificaría las tablas resultantes. Es un buen ejercicio que describieras el esquema resultante en este caso.