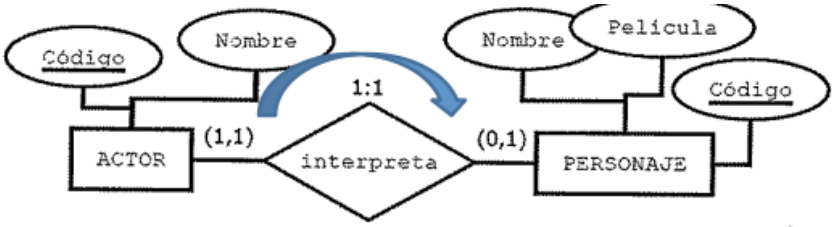
APUNTES

Si un atributo es **clave alternativa** en el modelo relacional se tendrá que indicar que es NOT NULL y UNIQUE

Se pone **NOT NULL** cuando en una cardinalidad en uno de los lados hay (1,1) y **UNIQUE** se pone **en una relación 1 : 1 al arrastrar la PK a la tabla deseada**.

Para pasar un diagrama a tabla: dependerá de las cardinalidades en las relaciones de las entidades, habrá varios casos dentro de cada uno (**1:1**, **1:N**, **N:M**)

- Si es **1:1** y la cardinalidad mínima en uno de los lados es 0 se crean dos tablas de las entidades como siempre y la PK de la tabla con cardinalidad (1 ,1) se pasa a la tabla con cardinalidad (0 ,1) como FK, esta FK se tendrá que indicar como Not Null y UNIQUE:

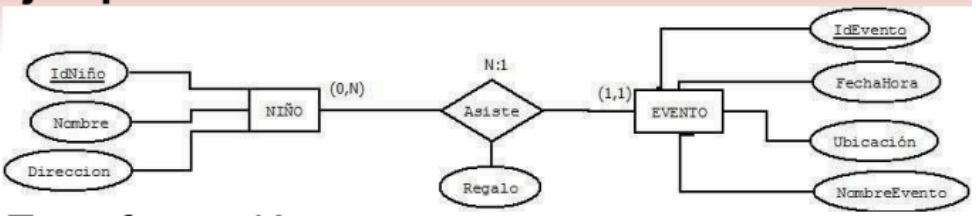


ACTORES(CodigoActor,Nombre)
PERSONAJES(CódigoPersonaje,Nombre,Pelicula,CodigoActor)
Claves ajenas: CodigoActor(ACTORES) Not Null y UNIQUE

En el caso de que en cada parte de la relación tengamos (1 , 1) podemos pasar la PK de una de las dos entidades a la tabla que queramos, estará bien pasemos la que pasemos siempre que pongamos Not Null y UNIQUE después. Si es 0,1 y 0,1 entonces **creamos una tabla nueva**

- Si la cardinalidad es **1:N** y consideramos que no habrá tantos nulos entonces la PK del lado que tenga como máxima 1 se irá a la tabla de la parte con cardinalidad máxima n especificando **NOT NULL** detrás (si en la relación hay algún atributo también se pasará a la tabla):

❖ **Ejemplo:**



❖ Transformación:

NIÑOS(IdNiño,Nombre,Direccion,IdEvento,Regalo)

Clave Primaria: IdNiño

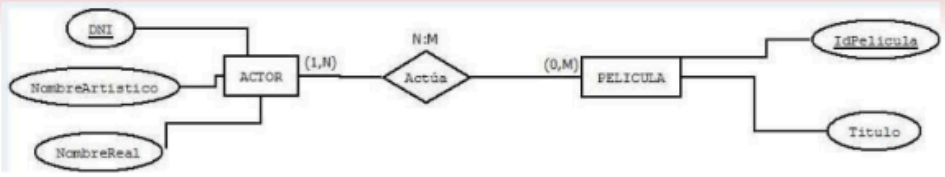
Claves ajenas: IdEvento (EVENTOS). NOT NULL.

EVENTOS (IdEvento,FechaHora,Ubicación,NombreEvento);

Clave Primaria: IdEvento

- Si la cardinalidad es **N:M** puede ocurrir que haya muchos nulos al crear la tabla, si esto ocurre deberemos de optar por una de las dos soluciones. La primera solución se usará a si no hay muchos nulos y consistirá en crear una tabla por cada entidad y una tabla de la relación donde incluiremos la suma de las dos PK de las entidades relacionadas:

❖ **Ejemplo:**



❖ Transformación:

ACTORES(DNI,NombreArtistico,Nombrereal)

PELICULAS(IdPelicula, Titulo)

ACTORES_PELICULAS(DNI, IdPelicula)

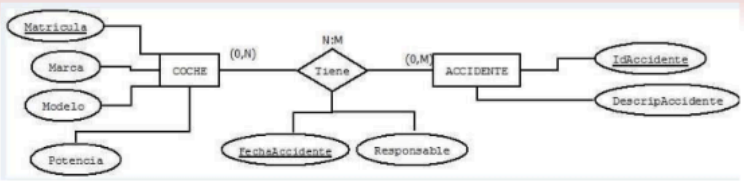
Clave Primaria: DNI, idPelicula

Claves Ajenas: DNI(ACTORES)

IdPelicula(PELICULAS)

Si contamos con un caso en el que pensemos que habrá muchos nulos crearemos (al igual que en el anterior caso) una tabla que contendrá las PK de las dos entidades (y de la relación si tiene) pero esta vez las "sumaremos" para que sean "una sola" PK:

❖ **Ejemplo:**



❖ Transformación:

COCHES(Matricula,Marca,Modelo,Potencia)

ACCIDENTES(IdAccidente, DescripcionAccidente)

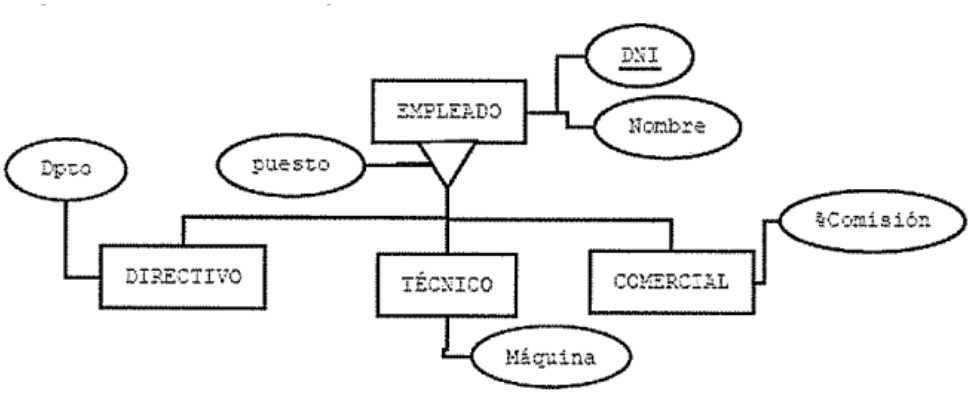
ACCID_COCHES(Matricula, IdAccidente,FechaAccidente,Responsable)

Clave Primaria: Matricula+IdAccidente+FechaAccidente

Claves ajenas: Matricula (COCHES)

IdAccidente(ACCIDENTES)

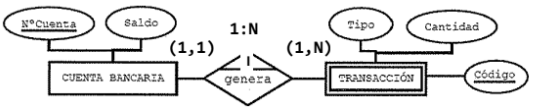
Cuando sea una jerarquía: se crea una tabla de la entidad de la que salen las demás (se le llama “supertabla”) y después se crean otras tablas de los tipos que salen de la entidad, lo que hace que se distinga es que la PK de la “supertabla” se pondrá como FK en todas las tablas que se hagan de los tipos:



EMPLEADOS(<u>DNI</u>,Nombre,Puesto)
DIRECTIVOS(<u>DNI</u>,Dpto)
Claves ajenas: DNI(EMPLEADOS)
TECNICOS(<u>DNI</u>,Máquinas)
Claves ajenas: DNI(EMPLEADOS)
COMERCIALES(<u>DNI</u>,Comisión)
Claves ajenas: DNI(EMPLEADOS)

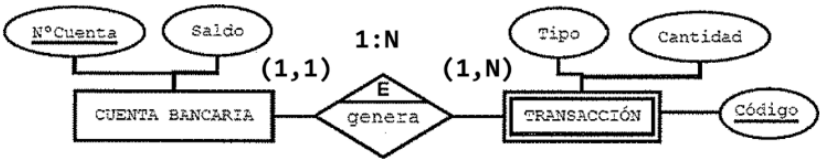
Cuando es entidad débil: si queremos hacer el diagrama extendido y nos encontramos con una entidad débil deberemos de saber primero si es entidad débil de identificación o de existencia:

- En el caso que sea entidad débil de identificación la PK de la entidad fuerte pasará como **parte** de la PK de la entidad débil. También deberemos indicar después de decir que es una FK Update/Delete on cascade:



CUENTAS(<u>Nro_cuenta</u>,Saldo)
TRANSACCIONES(<u>Codigo, Nro_cuenta</u>,tipo,Cantidad,)
Claves Ajenas: Nro_cuenta(CUENTAS).
(Update/Delete on cascade.)

- En el caso de que sea una entidad débil por existencia la PK de la entidad fuerte pasará como clave **ajena** en la tabla de la entidad débil, se deberá indicar que sea Not Null y Update/Delete on cascade:

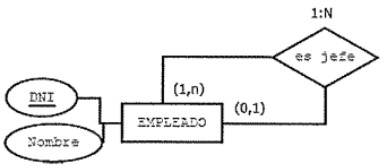


CUENTAS(Nro_cuenta,Saldo)
TRANSACCIONES(Codigo,Nro_cuenta, tipo,Cantidad)
Claves Ajenas: Nro_cuenta(CUENTAS)
Not Null y Update/Delete on cascade.

Si una entidad es reflexiva (tiene una relación con ella misma): pueden pasar tres cosas, que la cardinalidad sea **1:1**, **1:N** o que sea **N:M**.

- Si es **1:N** podrá ser una relación en donde la cardinalidad mínima por una de las partes sea 1 y otra 0, si esto es así entonces creamos una tabla con el nombre de la entidad que se relaciona con ella misma y creamos una PK inventada que hará referencia a la relación:

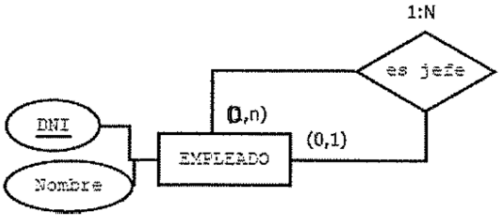
Es igual que el caso anterior



EMPLEADOS(DNI, Nombre,DNI_JEFE),
Clave ajena: DNI_JEFE(EMPLEADO)

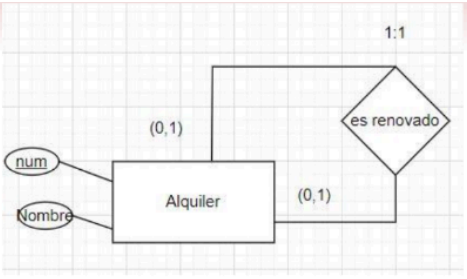
En el caso de que consideremos que habrá muchos nulos podemos crear una tabla de la relación (con el nombre de la relación) teniendo como PK la PK de la entidad y como FK la creada anteriormente:

Caso 2:
(0,1) y (**0**,n): Para evitar nulos, se crea una tabla con su PK y una nueva tabla con PK del lado N(Empleado) y FK del lado 1(Jefe)



EMPLEADOS(DNI, Nombre)
JEFES(DNI,DNI_JEFE);
Clave ajena: DNI_JEFE(EMPLEADO) Not Null
DNI (Empleados)

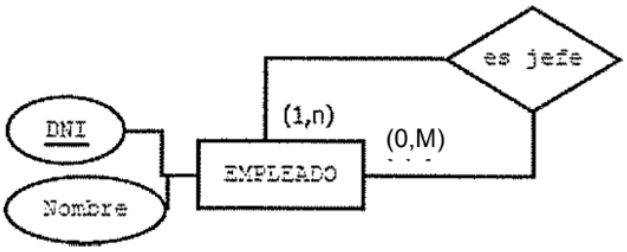
- Si es **1:1** se crea una tabla con la PK de la entidad y se crea una FK de ella misma indicando siempre que esta FK es UNIQUE:



Se crea una única tabla con la PK de la entidad y una FK de ella misma (se ha propagado la clave).

ALQUILERES(num, Nombre, numAlquilerAnterior)
Clave ajena: numAlquilerAnterior(ALQUILERES). UNIQUE

- Por último, si fuera **N:M** entonces deberemos de crear una nueva tabla con la PK repetida dos veces (si la PK es DNI pues pondremos DNI_NOSEQUE, DNI_NOSECUANTOS):



EMPLEADOS(DNI, Nombre)
JEFES(DNI_EMPLE,DNI_JEFE)
Clave ajena: DNI_EMPLE(EMPLEADOS) es el empleado
DNI_JEFE(EMPLEADOS) son los jefes

Si la relación es ternaria: como en los demás casos el resultado podrá variar dependiendo de la cardinalidad que tengan, podrán ser **N:M:P**, **1:N:M**, **1:1:N** y **1:1:1**.

- Si la relación es **N:M:P** entonces crearemos una tabla para cada entidad que está relacionada (con sus respectivas PK, atributos, etc.) y a parte una tabla de la relación y sus atributos serán las PK de las demás entidades (además de los atributos propios que tenga la relación si es que tiene)

- Si la relación es **1:N:M** entonces tendremos que crear una tabla de la relación, con sus PK siendo las PK de las entidades con cardinalidad N

❖ Transformación

AULAS (Numero, Planta, Situación) Clave Primaria: Numero, Planta

ESTUDIANTES(N°Matricula,Nombre,Direccion) Clave Primaria: NºMatricula

ASIGNATURAS (Nombre, Ciclo,Descripcion) Clave Primaria: Nombre, Ciclo

ESTUDIOS(Numero, Planta, Nombre, Ciclo, NºMatrícula, Hora)

Clave Primaria: Numero,Planta,Nombre,Ciclo

Claves Ajenas: Numero+Planta (AULA) Nombre+Ciclo (ASIGNATURAS)
Nº Matricula(ESTUDIANTES) Not Null

- Si la relación es **1:1:N** entonces tendremos que crear una tabla con la PK de la entidad con cardinalidad N y con la PK de una de las entidades con cardinalidad 1 (aunque solo pasamos una PK como PK en la nueva tabla eso no significa que la PK de la otra entidad 1 no deba de pasar a la tabla también, pasará pero en forma de atributo):

CLIENTES(ID_CLI,NOMBRE,DIRECCION) Clave Primaria:ID_CLI

CUENTAS(ID_CUENTA, FECHA_APERTURA) Clave Primaria: ID_CUENTA

PRESTAMOS(ID_PRESTAMO,FECHA_CONCESIÓN) CPrimaria: ID_PRSTAMO

CONCEDEN(ID_PRESTAMO,ID_CLI, ID_CUENTA**,** FECHA_CONCESION)
Clave Primaria:ID_PRESTAMO, ID_CLI ó ID_CUENTA
Clave Ajena: ID_PRESTAMO(PRÉSTAMO)
ID_CLI(CLIENTE) Not Null la que no sea PK
ID_CUENTA(CUENTA)

- Por último si la relación es **1:1:1** haremos prácticamente lo mismo que si fuera **1:1:N**, ya que crearemos otra tabla en donde las PK serán las PK de dos de las entidades con cardinalidad 1 (podemos escoger las que nosotros queramos, no influye) y pasando la otra PK como atributo.

MODELO FÍSICO

A tener en cuenta:

	USO	EJEMPLO
Crear una base de datos	Podremos crear una base de datos	`CREATE DATABASE [nombre]`

Usar una base de datos	Podremos usar una base de datos	`USE [nombre]`
		`GO`
		`[nombre]`
		`GO`

Eliminar una base de datos	Podremos eliminar una base de datos	`DROP DATABASE [nombre]`
Crear una tabla	Nos permite crear una nueva tabla en nuestra base de datos	`CREATE TABLE [nombre] ()`
Definir primary key	Nos permitirá establecer un atributo como primary key	`CONSTRAINT PK_[nombretabla]` `PRIMARY KEY ([campo])`

Definir una clave ajena	Nos permite establecer una relación de foreign key entre dos campos	`CONSTRAINT`
		`FK_[nombretabla]_[nombretabla]`
		`FOREING KEY ([campo])`
		`REFERENCES [tabla] ([campo])`

Establecer una restricción	Podemos establecer restricciones en los campos	`CONSTRAINT CK_[nombre]` `CHECK ([restricción])`

Insertar datos	Podemos meter los datos en las tablas y en las columnas creadas para ver que todo funcione. Si quieres insertar datos en todas las columnas de la tabla no haría falta especificar los campos, solo el nombre de la tabla.	`INSERT INTO [nombre tabla]`
		`([campo], [campo])`
		`VALUES ([dato], [dato])`

Mostrar datos	Podemos mostrar todos los datos que haya en una tabla. El * sirve para mostrar todos los datos, si solo queremos que nos muestre unos datos concretos deberemos de especificar las columnas en vez de poner *	**`SELECT * FROM [nombre tabla]`** **`SELECT t.* FROM tabla as t`**
Eliminar datos	Podemos eliminar los datos que hemos insertado previamente. Si no se pone "where" se eliminarán todos los datos introducidos en las columnas.	**`DELETE FROM [nombre tabla]`** **`WHERE [condición]`** **`//...WHERE id = 3`**
Modificar el dato	Si queremos modificar el dato pero no lo queremos borrar siempre podemos usar update	**`UPDATE ALUMNO SET [campo] = [dato nuevo] WHERE [campo pk] = [dato pk];`**

	USO
`ON DELETE` **`ON UPDATE`** **`NO ACTION`**	Generará un error y revierte la acción o el intento de eliminación en la fila de la tabla principal.
`ON DELETE` **`ON UPDATE`** **`CASCADE`**	Elimina o actualiza todo lo relacionado, es decir que actualiza y elimina la tabla secundaria automáticamente.
`ON DELETE` **`ON UPDATE`** **`SET NULL`**	En vez de actualizar o eliminar automáticamente cuando haya algún cambio en el campo asociado el de la tabla principal se establece las filas a NULL. No se le puede aplicar a una Primary Key
`ON DELETE` **`ON UPDATE`** **`SET DEFAULT`**	Establece las filas de las tablas en sus valores predeterminados si se eliminan las filas de la principal. Si el campo puede tener valor NULL el predeterminado será este.

COMANDO	USO	EJEMPLO
Añadir una restricción	Podemos usar esto para añadir una primary key, foreign key o una restricción check.	**`ALTER TABLE [nombre tabla] ADD CONSTRAINT …`**
Eliminar una restricción	Podemos eliminar una restricción si nos hemos equivocado al crearla y ejecutarla	**`ALTER TABLE [nombre tabla] DROP CONSTRAINT [nombre const.]`**
Añadir un nuevo campo	Podemos usar este comando para añadir un campo a la tabla si se nos ha olvidado (por ejemplo)	**`ALTER TABLE [nombre tabla] ADD [nombre columna] [tipo columna]`**
Cambiar un campo	Si necesitamos cambiar algo de una columna	**`ALTER TABLE [nombre tabla] MODIFY COLUMN [nombre columna] [tipo columna]`**
Eliminar un campo	Si tenemos que eliminar una columna creada	**`ALTER TABLE [nombre tabla] DROP [nombre columna] [tipo columna]`**

- Si alteramos una tabla para ponerle PK a un campo hay que antes declararlo como NOT NULL
- Para DNI se usa char.
- Para calcular la edad:
  `Year(CURRENT_TIMESTAMP- CAST (BirthDate AS SmallDateTime))-1900`

```
-- Verificamos si se ha asignado una posición al caballo en la carrera
IF EXISTS (SELECT 1 FROM LTApuestas WHERE ID = @IDApuesta)
BEGIN
    DECLARE @Posicion TINYINT;


    -- Obtenemos la posición del caballo en la carrera
    SELECT @Posicion = CC.Posicion
                       FROM LTApuestas A INNER JOIN LTCarreras CARR ON A
                       INNER JOIN LTCaballosCarreras CC ON CARR.ID = CC.
```

TRIGGER

```
CREATE OR ALTER TRIGGER trg_ActualizarStockVentas
ON Ventas
AFTER UPDATE
AS
BEGIN
    -- Solo si se cambió UnidadesVendidas
    IF UPDATE(UnidadesVendidas)
    BEGIN
        -- Aumento de unidades vendidas (nuevo > antiguo)
        UPDATE p
        SET p.Stock = p.Stock - (i.UnidadesVendidas - d.UnidadesVendidas)
        FROM Productos p
        INNER JOIN inserted i ON p.CodProducto = i.CodProducto
        INNER JOIN deleted d ON d.CodProducto = i.CodProducto
        WHERE i.UnidadesVendidas > d.UnidadesVendidas;

        -- Devolución de unidades (nuevo < antiguo)
        UPDATE p
        SET p.Stock = p.Stock + (d.UnidadesVendidas - i.UnidadesVendidas)
        FROM Productos p
        INNER JOIN inserted i ON p.CodProducto = i.CodProducto
        INNER JOIN deleted d ON d.CodProducto = i.CodProducto
        WHERE i.UnidadesVendidas < d.UnidadesVendidas;

        -- Borrar ventas si se devuelven todas las unidades
        DELETE v
        FROM Ventas v
        INNER JOIN inserted i ON v.CodVenta = i.CodVenta
        WHERE i.UnidadesVendidas = 0;
    END
END
```

CURSOR

```
    --declaro el cursor que iterará en la tabla de ventas
    DECLARE cursorVentas CURSOR FOR
    /*seleccionará la línea de producto, su nombre, el número de unidades
    vendidas por producto y el importe de dichas unidades vendidas que será el precio
    por unidad de cada producto multiplicado por el numero total de unidades vendidas*/
    SELECT P.LineaProducto, P.Nombre, SUM(V.UnidadesVendidas) AS 'Nº unidades vendidas', (P.PrecioUnitario * SUM(V.UnidadesVendidas)) AS 'Importe total' FROM ventas AS V
    INNER JOIN productos AS P ON P.CodProducto = V.CodProducto
    GROUP BY P.LineaProducto, P.Nombre, P.PrecioUnitario

    --abro el cursor
    OPEN cursorVentas
    --hago la primera iteración
    FETCH cursorVentas INTO @lineaProducto, @nombreProducto, @unidadesTotales, @importeTotalProducto

    --empiezo a recorrer toda la tabla con el while
    WHILE (@@FETCH_STATUS = 0)
    BEGIN

        PRINT 'Línea producto: ' + @lineaProducto
        --casteo las variables que contienen numeros porque pueden llegar a dar errores
        PRINT 'Nombre del producto          Unidades totales           Importe total'
        PRINT @nombreProducto + '                        ' + CAST(@unidadesTotales AS varchar(20)) + '                        ' + CAST(@importeTotalProducto AS varchar(20))

        --voy sumandole a la variable el importe total por producto para calcular el importe total de todos los productos vendidos
        SET @importeTotalLineas += @importeTotalProducto

        --salto al siguiente producto
        FETCH cursorVentas INTO @lineaProducto, @nombreProducto, @unidadesTotales, @importeTotalProducto

    END

    PRINT 'Total ventas: ' + CAST(@importeTotalLineas AS varchar(20))

    --cierro el cursor
    CLOSE cursorVentas
    DEALLOCATE cursorVentas

END
```