# IMPLEMENTANDO NUESTRO PROPIO PROYECTO

BD-
ACTIVIDAD 2.3

Trabajo de Alejandro Sainz Sainz

# Contenido

Alejandro Sainz Sainz

# CREANDO EL SCRIP DE NUESTRO PROYECTO

Lo primero es ir creando la BD en el script, e ir fijándonos en cuales son las tablas que no poseen dependencias para ir creándolas primero.

```
#Con esto vamos creando la base de datos y nos aseguramos de que la borremos en caso
#de que ya exista

DROP DATABASE IF exists viajeros;
CREATE DATABASE viajeros;

Use viajeros;

```

Lo primero que voy a hacer es crear la tabla provincia, que no tiene dependencias.

```
#Creo las primeras tablas, aquellas que no tienen dependencias.

CREATE TABLE provincia(
    CP char(2) NOT NULL UNIQUE,
    nombre varchar(15) NOT NULL,
    CONSTRAINT CP primary key (CP)
);
```

Ahora puedo crear la tabla localidad, ya que solo depende de la tabla provincia, así puedo dejar esa parte cerrada y seguir con otras tablas que no tengan dependencia.

```
#Creo la tabla localidad
CREATE table localidad(
    cp_completo char(5) NOT NULL,
    localidad varchar(20),
    cp char(2) NOT NULL,
    CONSTRAINT cp_completo primary key (cp_completo),
    CONSTRAINT cp foreign key (cp) references provincia (CP)
);
```

Al tener completadas estas dos, como luego necesito localidad para otras dos tablas como FK, ya puedo despreocuparme por ellas.

Voy a seguir con otras dos tablas independientes, que son la tabla departamentos y la tabla empresa vehículos.

```
#Creo la tabla departamento
CREATE TABLE departamento(
    cod_dep SMALLINT AUTO_INCREMENT UNIQUE NOT NULL,
    departamento VARCHAR(15) NOT NULL,
    CONSTRAINT cod_dep primary key (cod_dep)
);
```

Esta es la parte del script correspondiente a la tabla departamento.

```
#Creo la tabla empresas_vehiculos
CREATE TABLE empresa_vehiculos(
    cif CHAR(9) UNIQUE NOT NULL,
    nombre_empresa VARCHAR(15) NOT NULL,
    direccion VARCHAR(25) NOT NULL,
    localidad VARCHAR(20) NOT NULL,
    telefono CHAR(9) NOT NULL,
    CONSTRAINT cif primary key (cif)
);
```

Con esta parte del script ya tengo terminada la tabla de empresas de vehículos de alquiler.

Ahora, si no me equivoco, creo que la única tabla que me quede de añadir que no tiene dependencias es la tabla paquetes, aquella que recoge los paquetes y su descripción.

```
#Para terminar con las tablas sin dependencias, salvo que se me haya pasado alguna
#creo la tabla paquetes
CREATE TABLE paquetes(
    cod_paquete CHAR(4) AUTO_INCREMENT UNIQUE NOT NULL,
    nombre_paquete VARCHAR(15) NOT NULL,
    descripcion VARCHAR(30) NOT NULL,
    direccion VARCHAR(25) NOT NULL,
    num_max_personas SMALLINT NOT NULL,
    CONSTRAINT max primary key (num_max_personas)
);
```

Pues esto sería la tabla paquetes. Intento siempre ajustar los valores de los atributos a lo que yo creo que sería realista, por ejemplo, cod_paquete CHAR(4), ya que no creo que el código de paquete ocupe más de cuatro caracteres.

Intento que el atributo dirección sea de la misma longitud en todas las tablas que lo incluyen, por coherencia más que nada.

En el caso de num_max_personas he decidido que sea SMALLINT, simplemente por una cosa. Imaginemos que la empresa escala o necesita otra serie de cosas, como calcular media de clientes por actividad, etc, como incluir el precio de un paquete por persona, etc.

Ahora que lo he comentado, voy a añadir el precio de un paquete, completo, sin contar por grupo o con descuentos, que creo que así completo un poco al modelo.

```
#Para terminar con las tablas sin dependencias, salvo que se me haya pasado alguna
#creo la tabla paquetes
CREATE TABLE paquetes(
    cod_paquete CHAR(4) AUTO_INCREMENT UNIQUE NOT NULL,
    nombre_paquete VARCHAR(15) NOT NULL,
    descripcion VARCHAR(30) NOT NULL,
    direccion VARCHAR(25) NOT NULL,
    precio FLOAT(6,2) NOT NULL,
    num_max_personas SMALLINT NOT NULL,
    CONSTRAINT max primary key (num_max_personas)
);
```

Le añado el campo precio como ya dije antes.

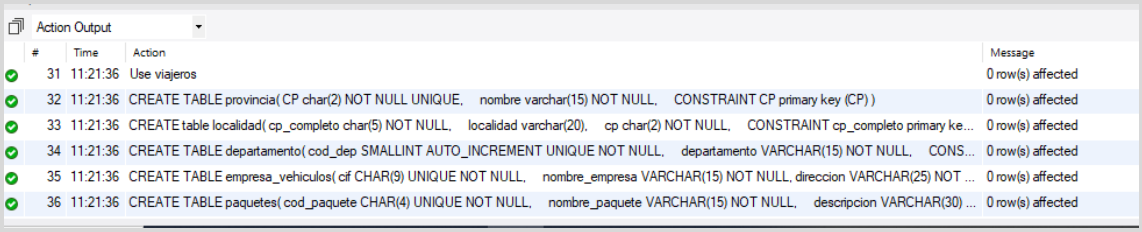
Llegado a este punto voy a ejecutar el script simplemente para ver que no da errores.

```
#Para terminar con las tablas sin dependencias, salvo que se me haya pasado alguna
#creo la tabla paquetes
CREATE TABLE paquetes(
    cod_paquete CHAR(4) UNIQUE NOT NULL,
    nombre_paquete VARCHAR(15) NOT NULL,
    descripcion VARCHAR(30) NOT NULL,
    direccion VARCHAR(25) NOT NULL,
    precio DECIMAL(6,2) NOT NULL,
    num_max_personas SMALLINT NOT NULL,
    CONSTRAINT cod primary key (cod_paquete)
);
```

Gracias a hacer las pruebas de ejecutar el script como comprobación me di cuenta de un par de errores.

1. Puse un campo char como auto increment, cosa bastante improbable.
2. Me indicaba que marcar un campo como float era un practica deprecated y como indicó el profesor en clase lo cambie por decimal.

Una vez realizados los cambios el script quedó así y esto es lo que recoge la ejecución del scritp.




Vemos que ya no da ningún error o warning, así que ahora vamos a realizar un intento de ingeniería inversa, para ver como va quedando el diagrama, aunque me imagino que por ahora será bastante sencillo.

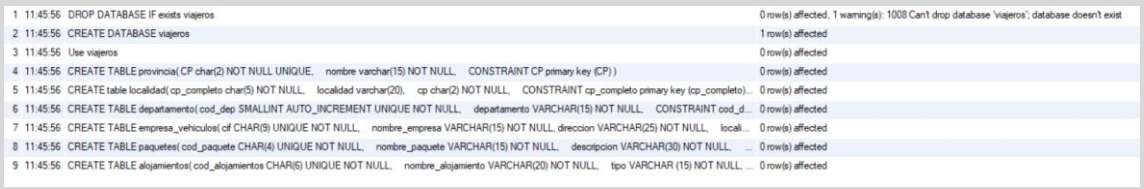
Como última comprobación, me aseguro de que se genera la parte del diagrama correspondiente.

Me acabo de fijar que la tabla alojamientos no depende de ninguna otra, según el modelo que yo generé, así que voy a crearla ahora.

```
#Creo la tabla alojamientos, que también es independiente.
CREATE TABLE alojamientos(
    cod_alojamientos CHAR(6) UNIQUE NOT NULL,
    nombre_alojamiento VARCHAR(20) NOT NULL,
    tipo VARCHAR (15) NOT NULL,
    direccion VARCHAR(25) NOT NULL,
    localidad VARCHAR(20) NOT NULL,
    telefono CHAR(9) NOT NULL,
    categoria VARCHAR(10),
    CONSTRAINT cod_alojamientos primary key (cod_alojamientos)
    );
```

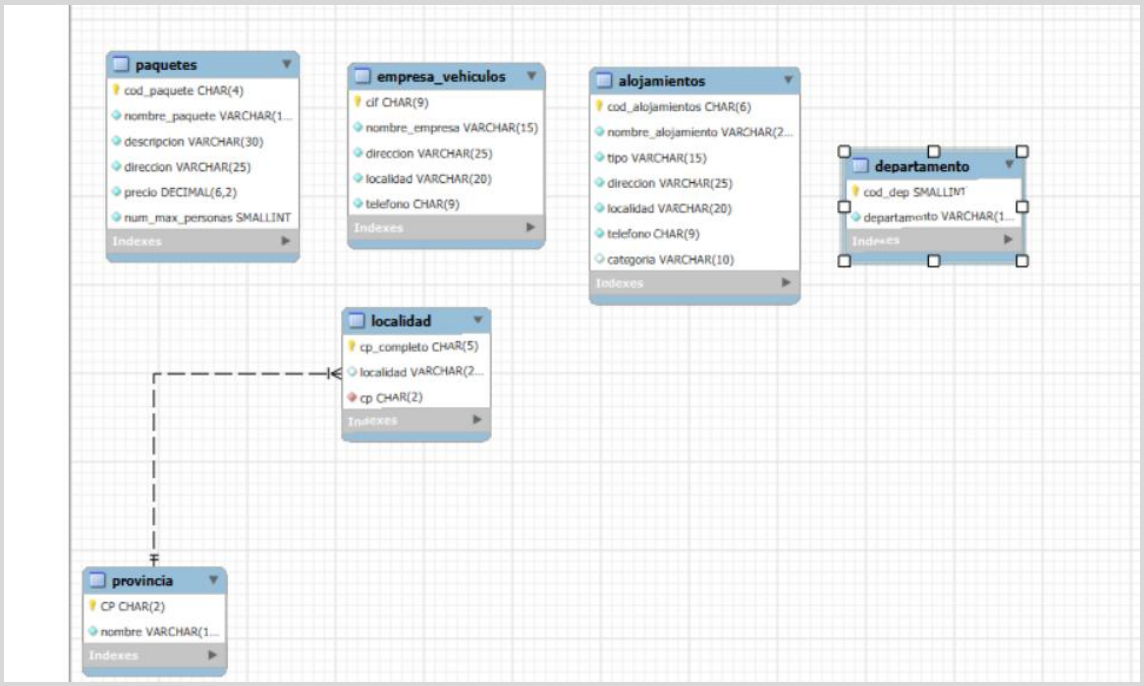
Con esto, ahora sí, creo que ya tengo completadas todas las tablas independientes.

Vuelvo a ejecutar el script para comprobar todo de nuevo.



| | | | |
|---|---|---|---|
| 1 | 11:45:56 | DROP DATABASE IF exists viajeros | 0 row(s) affected, 1 warning(s): 1008 Can't drop database 'viajeros'; database doesn't exist |
| 2 | 11:45:56 | CREATE DATABASE viajeros | 1 row(s) affected |
| 3 | 11:45:56 | Use viajeros | 0 row(s) affected |
| 4 | 11:45:56 | CREATE TABLE provincia( CP char(2) NOT NULL UNIQUE, nombre varchar(15) NOT NULL, CONSTRAINT CP primary key (CP) ) | 0 row(s) affected |
| 5 | 11:45:56 | CREATE table localidad( cp_completo char(5) NOT NULL, localidad varchar(20), cp char(2) NOT NULL, CONSTRAINT cp_completo primary key (cp_completo)... | 0 row(s) affected |
| 6 | 11:45:56 | CREATE TABLE departamento( cod_dep SMALLINT AUTO_INCREMENT UNIQUE NOT NULL, departamento VARCHAR(15) NOT NULL, CONSTRAINT cod_d... | 0 row(s) affected |
| 7 | 11:45:56 | CREATE TABLE empresa_vehiculos( cif CHAR(9) UNIQUE NOT NULL, nombre_empresa VARCHAR(15) NOT NULL, direccion VARCHAR(25) NOT NULL, locali... | 0 row(s) affected |
| 8 | 11:45:56 | CREATE TABLE paquetes( cod_paquete CHAR(4) UNIQUE NOT NULL, nombre_paquete VARCHAR(15) NOT NULL, descripcion VARCHAR(30) NOT NULL, ... | 0 row(s) affected |
| 9 | 11:45:56 | CREATE TABLE alojamientos( cod_alojamientos CHAR(6) UNIQUE NOT NULL, nombre_alojamiento VARCHAR(20) NOT NULL, tipo VARCHAR (15) NOT NULL, ... | 0 row(s) affected |

Ningún warning o error, así que, ahora sí, puedo hacer una prueba para ver cómo va quedando el diagrama.




Por ahora va quedando como esperaba, aunque no era muy difícil, ya que todavía no existe ninguna relación en lo que he creado, salvo en el caso de localidad y provincia.

En este punto, tengo que empezar a crear las tablas que son dependientes de otras, e ir viendo como quedan en comparación con el diagrama que fui desarrollando yo en la realización del ejercicio.

Creo ahora la tabla grupos, que me acabo de dar cuenta en una parte posterior del ejercicio que está sin crear.

```
#Creo la tabla grupos
CREATE TABLE grupos(
    id_grupo INT AUTO_INCREMENT UNIQUE NOT NULL,
    num_personas CHAR(2) NOT NULL,
    CONSTRAINT id_grupo primary key(id_grupo)
);
```

En este caso, he decidido darle a la clave principal, id_grupo, un tipo int que se vaya incrementando según vayamos creando los grupos y a num_personas un varchar(2) ya que no creo que el tamaño de un grupo esté en ningún caso por encima de 99.

Ahora que me doy cuenta, si quiero calcular medias de clientes y grupos, creo que es mejor cambiar el tipo de num_personas a int, por si en algún caso deseo realizar operaciones con esos datos.

```
#Creo la tabla grupos
CREATE TABLE grupos(
    id_grupo INT AUTO_INCREMENT UNIQUE NOT NULL,
    num_personas INT NOT NULL,
    CONSTRAINT id_grupo primary key(id_grupo)
);
```

Lo dejo así.

# AGREGANDO TABLAS DEPENDIENTES

A partir de ahora, hay que crear las tablas que reciben PK de otras tablas como FK. Estas se crean a partir de este momento, pues en un inicio, si no se han creado las tablas de las que se reciben las PK como FK el programa lo reconoce como un error.

Creo que, en este caso, la más importante va a ser la tabla sedes, que es la tabla central de nuestro ejercicio.

```
#Empiezo a crear tablas dependientes, en este caso la tabla sedes
CREATE TABLE sedes(
    cod_sede INT auto_increment NOT NULL UNIQUE,
    direccion VARCHAR(25) NOT NULL,
    telefono CHAR(9) NOT NULL,
    email VARCHAR(25) NOT NULL,
    cp_completo CHAR(5) NOT NULL,
    CONSTRAINT cod_sede primary key (cod_sede),
    CONSTRAINT cp_completo foreign key (cp_completo) references localidad (cp_completo)
);
```

Este sería el fragmento de script que corresponde a la tabla sedes.

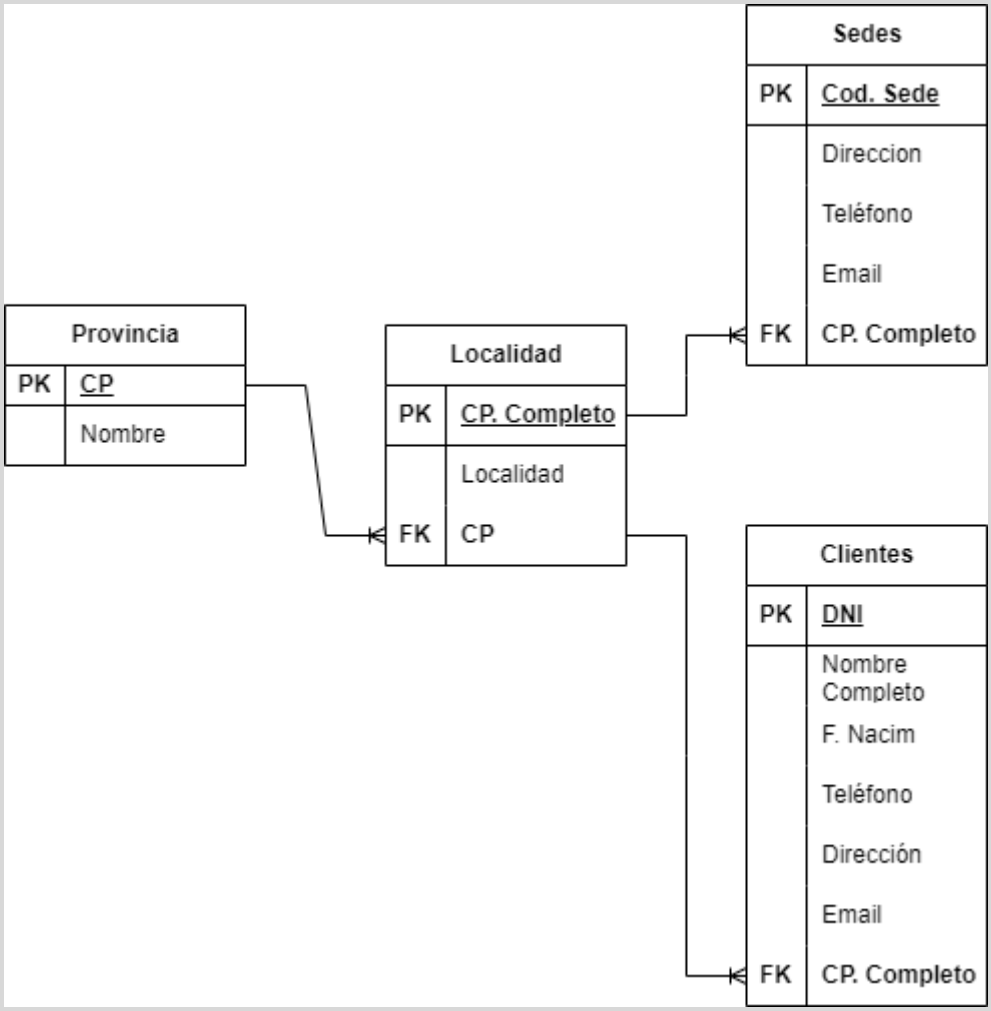
En un mismo orden de cosas, tengo la tabla clientes, que tiene la misma dependencia, de la tabla localidad, y como ya mostré en el informe anterior, se pude incluir en el mismo ejemplo.

```
#Seguimos con la tabla clientes
CREATE TABLE clientes(
    dni CHAR(9) UNIQUE NOT NULL,
    nombre_completo VARCHAR(25) NOT NULL,
    telefono CHAR(9) NOT NULL,
    direccion VARCHAR(25) NOT NULL,
    emali VARCHAR(25) NOT NULL,
    cp_completo CHAR(5) NOT NULL,
    CONSTRAINT dni primary key (dni),
    CONSTRAINT cp_completo foreign key(cp_completo) references localidad(cp_completo)
);
```

Pues aquí tenemos creada la tabla clientes.

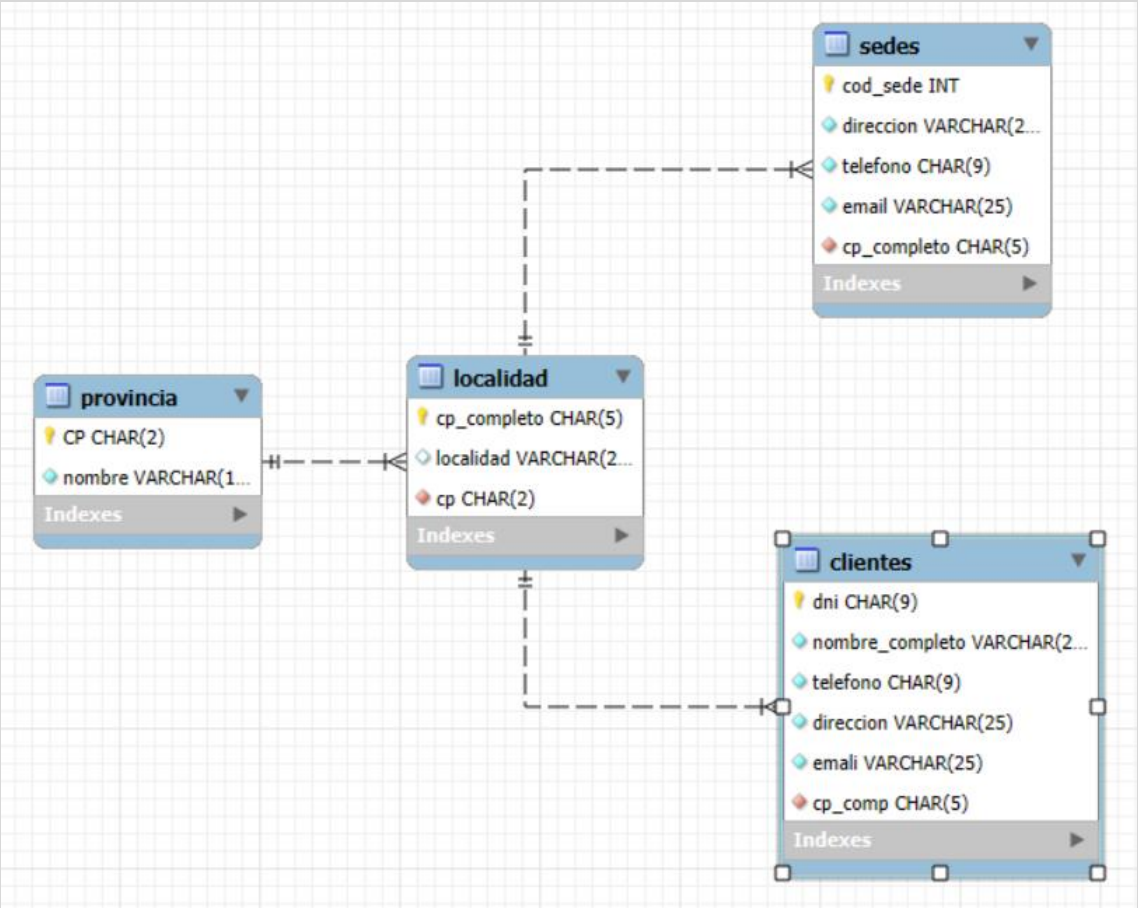
## COMPROBACIÓN DE TABLAS Y DIAGRAMA

Para comprobar que estas dos tablas y la relación con la tabla localidad y provincia queda como resolví en el ejercicio anterior voy a ejecutar el script y la ingeniería inversa, y compararé el resultado.




Este diagrama es como quedó resuelto mi ejercicio, vamos a comprobar como queda en el Workbench.

Lo primero, al ejecutar el script, me da un error, hay dos FK con el mismo nombre dentro del modelo, cambio la FK de la tabla clientes por cp_comp, para que no se repita con la FK de la tabla sedes. Después de eso, ya puedo comprobar el diagrama.



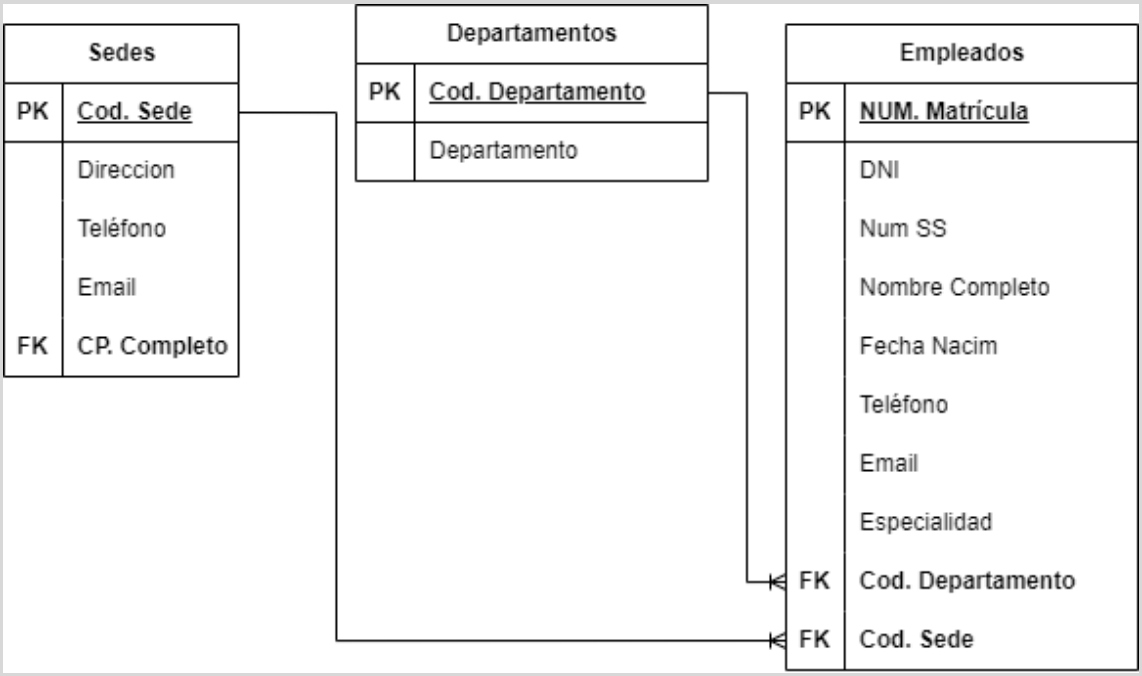

Si lo comparamos con la resolución del ejercicio vemos que quedan iguales.

Ahora prosigamos con los siguientes ejemplos.

## MÁS RELACIONES RESULTAS

Ahora voy a ir resolviendo el caso de los empleados, que creo que lo hice mal en el ejercicio anterior. En el diagrama E/R indiqué una herencia excluyente.

Sin embargo, al crear las tablas en ese mismo ejercicio, creé una tabla departamentos, y luego pasé la clave del departamento como FK a la tabla empleados, así que no sé si está bien resuelto como herencia, aunque como dos tablas independientes que se relacionan creo que si está resuelto de una forma correcta, así que optaré por esta segunda opción.



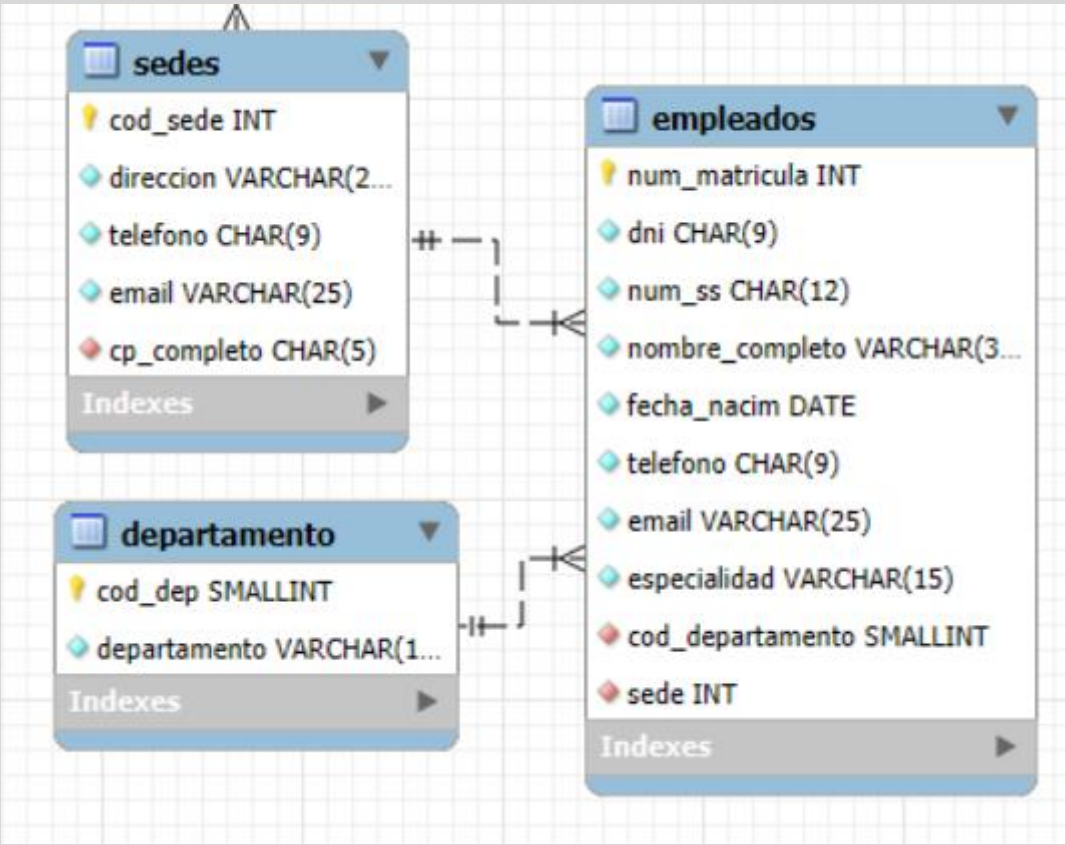

Esta es la forma en que lo resolví en el ejercicio anterior, y esta es la que voy a seguir al crear la base de datos en Workbench.

La tabla departamentos ya la tengo creada en el apartado anterior, de la misma forma que clientes, así que ahora sólo me queda crear la tabla empleados y relacionarlo todo.

```
#Creando la tabla empleados
CREATE TABLE empleados(
    num_matricula INT AUTO_INCREMENT UNIQUE NOT NULL,
    dni CHAR (9) NOT NULL,
    num_ss CHAR(12) NOT NULL,
    nombre_completo VARCHAR(30) NOT NULL,
    fecha_nacim DATE NOT NULL,
    telefono CHAR(9) NOT NULL,
    email VARCHAR(25) NOT NULL,
    especialidad VARCHAR(15) NOT NULL,
    cod_departamento SMALLINT NOT NULL,
    sede INT NOT NULL,
    CONSTRAINT num_matricula primary key(num_matricula),
    CONSTRAINT cod_departamento foreign key(cod_departamento) references departamento(cod
    CONSTRAINT sede foreign key(sede) references sedes(cod_sede)
);
```

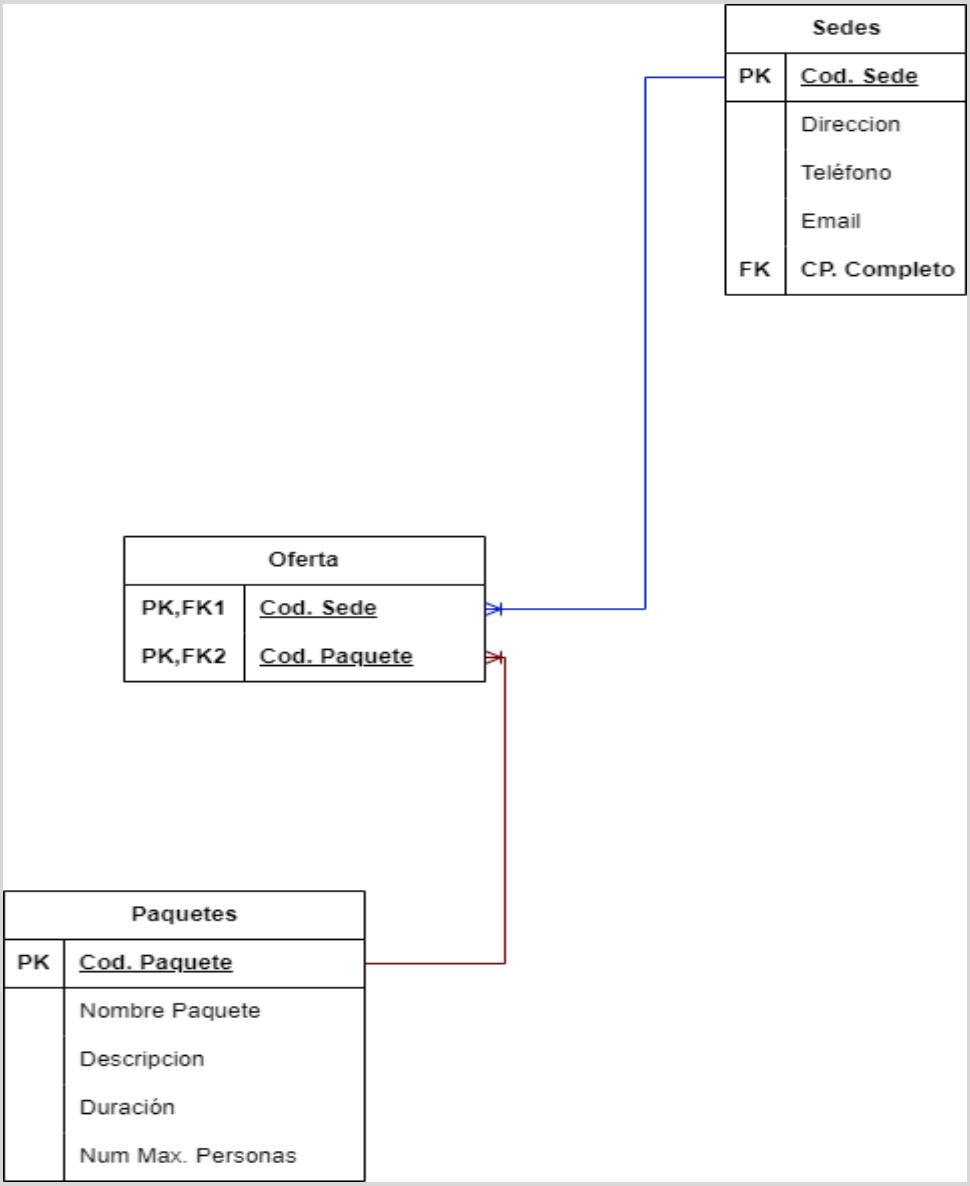
Esta es la creación de la tabla empleados, que es una de las más largas que tenía en el ejercicio. De nuevo voy a ejecutar el script y a ver el diagrama en Workbench.

De nuevo compruebo que al ejecutar el script no hay fallos (no incluyo captura para no alargarlo todo demasiado).



La colocación es diferente a la que creé yo en draw.io pero el resultado es el mismo.

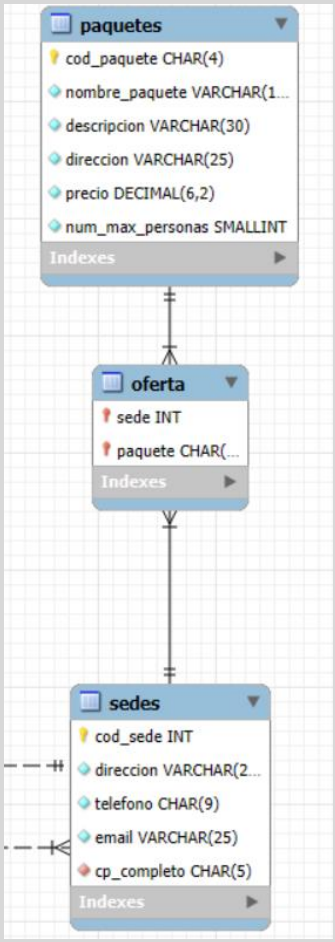
Continuo resolviendo tablas que tienen que ver con sedes, en este caso voy a resolver la relación entre sedes y paquetes. Como la relación era N:N debo de crear la tabla intermedia.




Como se ve aquí, en vez de crear un código para la relación, use las dos FK para crear un PK compuesta. Vamos a resolverlo en el Script.

```
CREATE TABLE oferta(
    sede INT NOT NULL,
    paquete CHAR(4) NOT NULL,
    CONSTRAINT oferta_sedes primary key(sede, paquete),
    CONSTRAINT sedes foreign key(sede) references sedes(cod_sede),
    CONSTRAINT paquetes foreign key(paquete) references paquetes(cod_paquete)
);
```

Este es la porción de Script. Veamos el resultado.



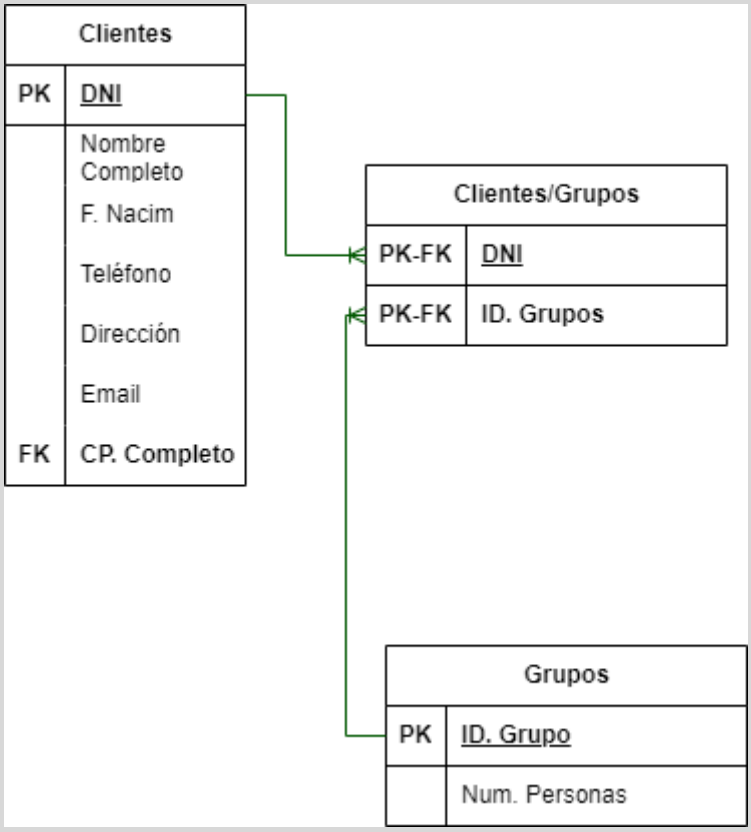

Las dos entidades quedan relacionadas así. Prosigamos.

Me acabo de dar cuenta de que tenía otra tabla independiente que tenía que crear (esto me pasa por dejarme llevar y hacer un ejercicio muy largo).

Lo que voy a hacer es volver al apartado anterior a crearla y ponerlo allí y después resolver la relación correspondiente.

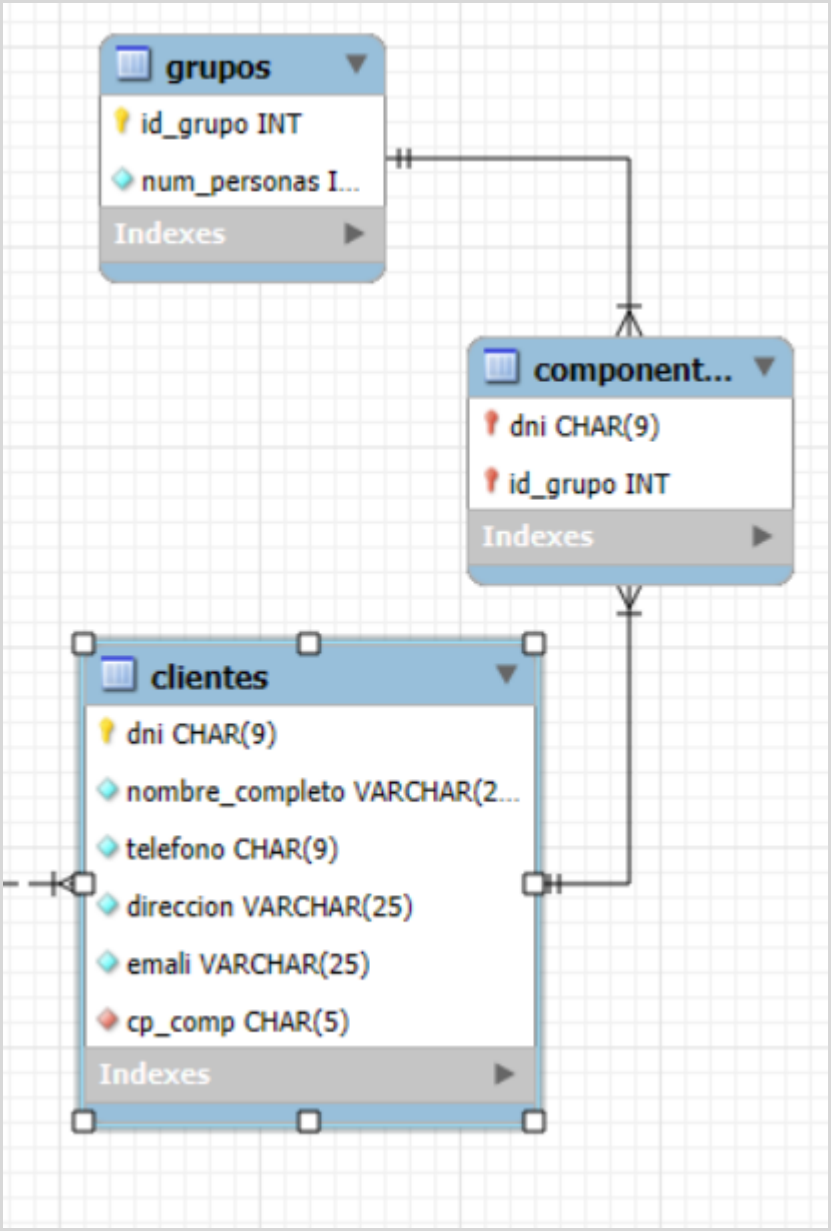
La tabla en cuestión es la tabla grupos.

Una vez subsanado esto, procedo a crear la tabla intermedia tal y como resolví el ejercicio anterior.



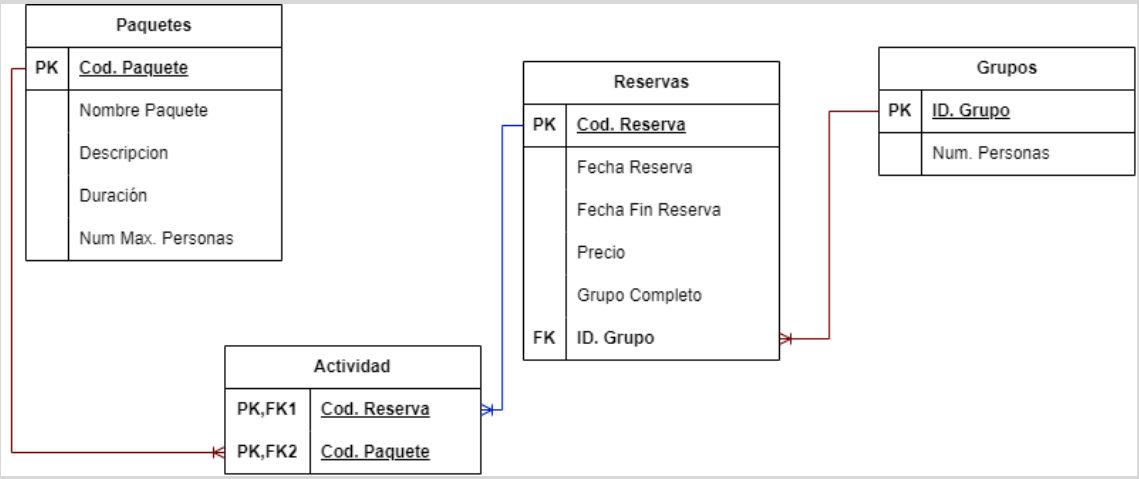

```
#Genero la tabla componentes para resolver la relación entre clientes y grupos
CREATE TABLE componentes(
    dni CHAR(9) NOT NULL,
    id_grupo INT NOT NULL,
    CONSTRAINT componentes primary key(dni, id_grupo),
    CONSTRAINT miembro foreign key(dni) references clientes(dni),
    CONSTRAINT grupo foreign key(id_grupo) references grupos(id_grupo)
);
```

Creando nombres distintos para la PK y las FK se genera esta tabla. El script se ejecuta de forma correcta. Vamos a ver el diagrama, que ya se empieza a liar un poco.




Ya tengo la relación resuelta, y como podemos observar, este fragmento del diagrama se parece bastante a la resolución del diagrama que yo creé.

Ahora que ya tenemos resulto el tema de los grupos, podemos pasar a resolver las reservas, que incluye todo esto que he hecho hasta ahora.




Como ya tengo creada la tabla grupos, puedo proceder a crear la tabla reservas. Vamos a ello.

```
#Después de crear la tabla componentes puedo crear la tabla reservas
CREATE TABLE reservas(
    cod_reserva INT AUTO_INCREMENT UNIQUE NOT NULL,
    fecha_reserva DATE NOT NULL,
    fecha_fin_reserva DATE,
    precio DECIMAL(6,2) NOT NULL,
    grupo_completo BOOLEAN default false,
    id_grupo INT NOT NULL,
    CONSTRAINT cod_reserva primary key(cod_reserva),
    CONSTRAINT grupo_reserva foreign key(id_grupo) references grupos(id_grupo)
);
```

De esta forma he creado yo la tabla. Fecha_fin_reserva permito que sea null por si en la reserva, por el motivo que sea no se especifica fecha de fin. Incluyo el campo precio, un decimal, grupo completo (que sirve para indicar si un grupo completa el número de personas máximas que permite la actividad a realizar) indico que es del tipo boolean y por defecto su valor será false.

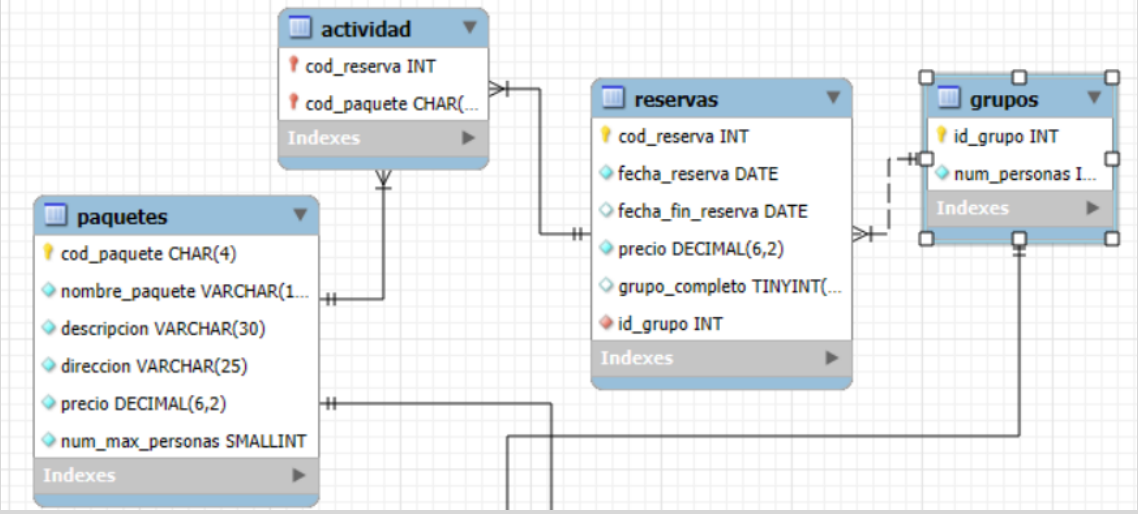
Después de esto indico la PK y la FK que forman parte de esta tabla. Al ejecutar el script todo correcto (que no quiere decir que esté bien, pero que se ejecuta sin ningún problema evidente).

Voy a crear ahora la tabla intermedia que usé para terminar de resolver todo este bloque de relaciones.

Esta tabla se llama actividad, ya que sería un listado de las actividades que se llevan a cabo como parte de lo contratado por un grupo en una reserva, indicando los paquetes que están contratados en una reserva en concreta. Vamos a ello.

```
#Creo la tabla actividad para terminar de cerrar este bloque
CREATE TABLE actividad(
    cod_reserva INT NOT NULL,
    cod_paquete CHAR(4) NOT NULL,
    CONSTRAINT actividades primary key(cod_reserva, cod_paquete),
    CONSTRAINT reserva foreign key (cod_reserva) references reservas(cod_reserva),
    CONSTRAINT paquete foreign key (cod_paquete) references paquetes(cod_paquete)
);
```

Así es como resuelvo la creación de esta tabla intermedia. La ejecución en principio sin problema, veamos el diagrama para confirmar.




En principio todo correcto, el único problema, es que ya se empieza a hacer complicado que no haya líneas de relación por todas partes. Intento colocarlo todo de una forma en la que se vean de forma clara cuales son las líneas que corresponden a la relación resuelta en cada caso. Lo que me he dado cuenta al revisar, es que el campo boolean me le cambia el propio programa por un tinyint, que ya había visto en los apuntes que se considera lo mismo.

## AVANZANDO CON LA RESOLUCIÓN

Ahora que ya hemos ido resolviendo gran parte de lo que es el núcleo de nuestra BD, vamos a seguir con el resto de las relaciones.

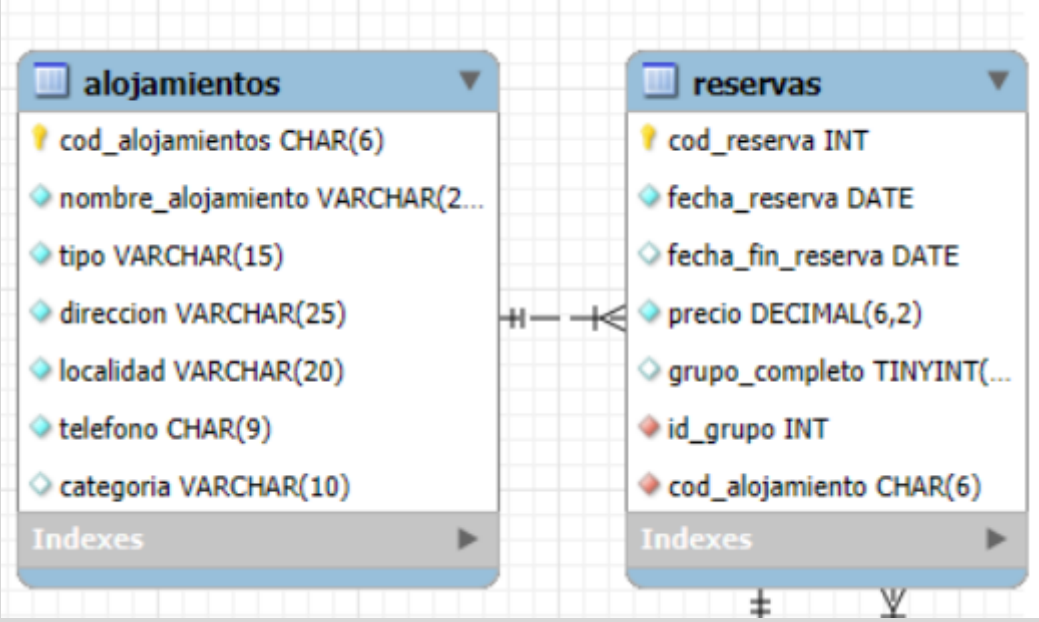
Bien, este caso que viene ahora, lo dejé yo para este momento para probar comandos.

Como sabemos, en mi ejemplo, en la tabla reservas también se indica que alojamiento corresponde a cada reserva. Vamos a incluir eso ahora, y así tengo que usar comandos para modificar la tabla reservas.

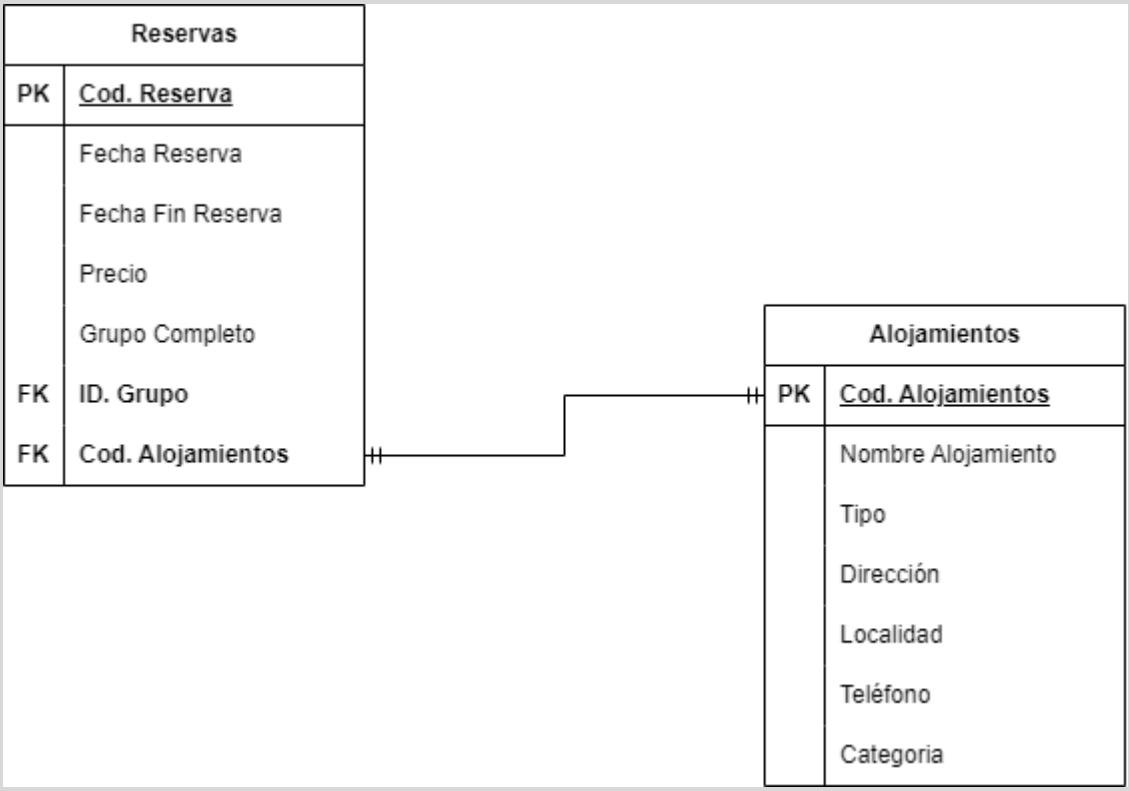
Voy a usar el comando para añadir el cod_alojamiento a la tabla reservas.

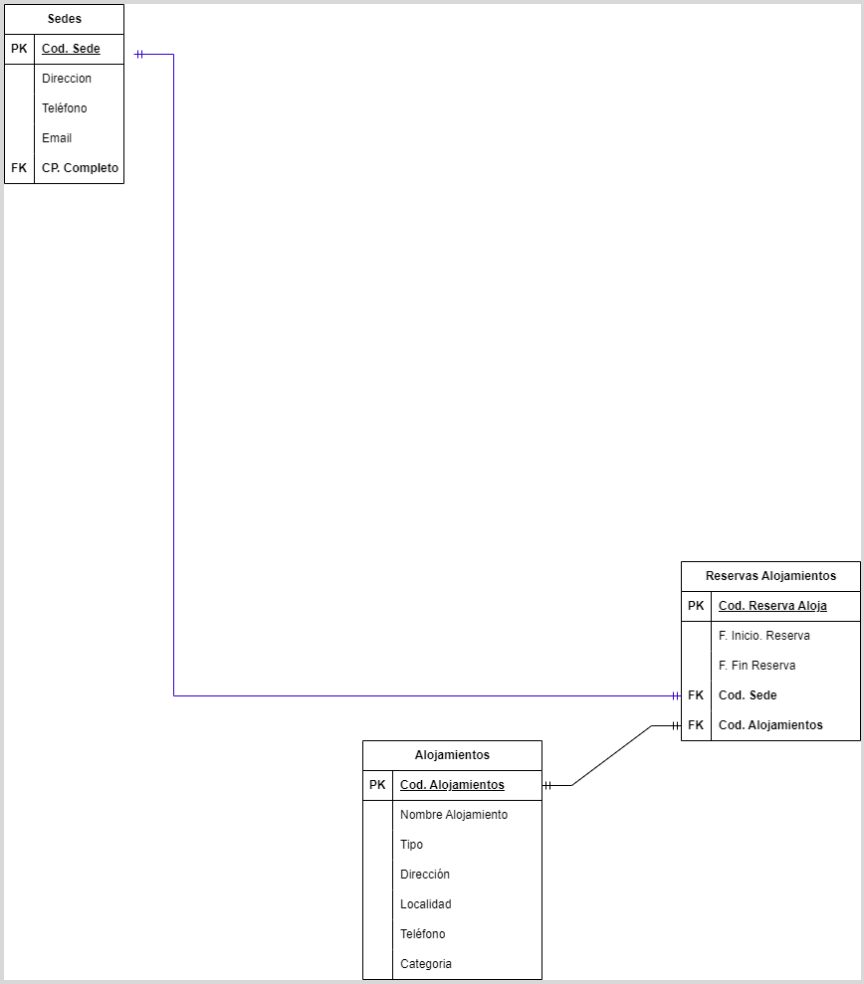
```
#Modifico la tabla reservas para añadir el cod_alojamiento
ALTER TABLE reservas ADD cod_alojamiento CHAR(6) NOT NULL;
ALTER TABLE reservas ADD CONSTRAINT alojamiento foreign key(cod_alojamiento) references alojamientos(cod_alojamientos);
```

Con estos dos comandos, primero añado el nuevo campo a la tabla, después ya le añado a la tabla el CONSTRAINT indicando que ese nuevo campo es la foreign key y a que referencia. No se si se podría hacer en una sola línea, pero creo que así también es válido. Vamos a comprobar ahora el diagrama.




El atributo se ha añadido y se ha generado la relación, así que por ahora todo correcto.

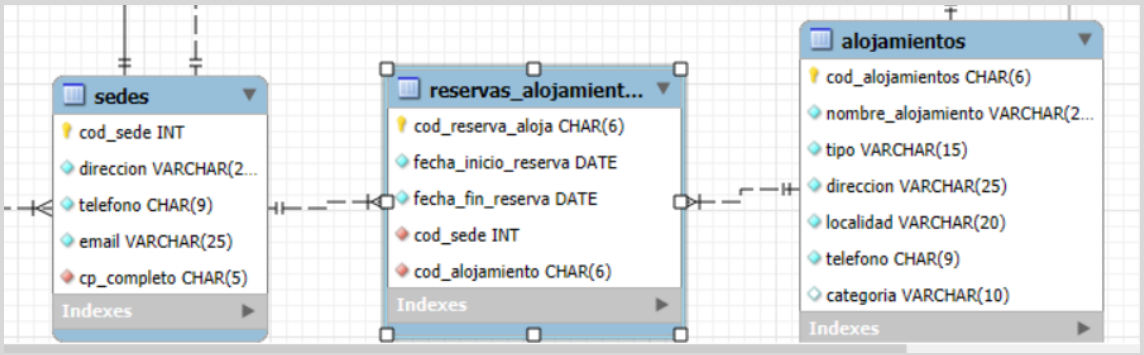

Reservas
PK
Cod. Reserva
Fecha Reserva
Fecha Fin Reserva
Precio
Grupo Completo
FK
ID. Grupo
FK
Cod. Alojamientos
Alojamientos
PK
Cod. Alojamientos
Nombre Alojamiento
Tipo
Dirección
Localidad
Teléfono
Categoria


Sedes
PK
Cod. Sede
Direccion
Teléfono
Email
FK
CP. Completo
Reservas Alojamientos
PK
Cod. Reserva Aloja
F. Inicio. Reserva
F. Fin Reserva
FK
Cod. Sede
FK
Cod. Alojamientos
Alojamientos
PK
Cod. Alojamientos
Nombre Alojamiento
Tipo
Dirección
Localidad
Teléfono
Categoria

Vamos a continuar con la parte de los alojamientos. En el ejercicio dije que las sedes se encargan de las reservas de los alojamientos, así que vamos a implementar eso en nuestra BD.

```
#Vamos con la relación entre sedes y alojamientos
CREATE TABLE reservas_alojamientos(
    cod_reserva_aloja CHAR(6) UNIQUE NOT NULL,
    fecha_inicio_reserva DATE NOT NULL,
    fecha_fin_reserva DATE NOT NULL,
    cod_sede INT NOT NULL,
    cod_alojamiento CHAR(6) NOT NULL,
    CONSTRAINT cod_reserva_aloja primary key(cod_reserva_aloja),
    CONSTRAINT sede_reserva foreign key(cod_sede) references sedes(cod_sede),
    CONSTRAINT alojamiento_reserva foreign key(cod_alojamiento) references alojamientos(cod_alojamientos)
);
```

Aquí creo el código para esta tabla intermedia.



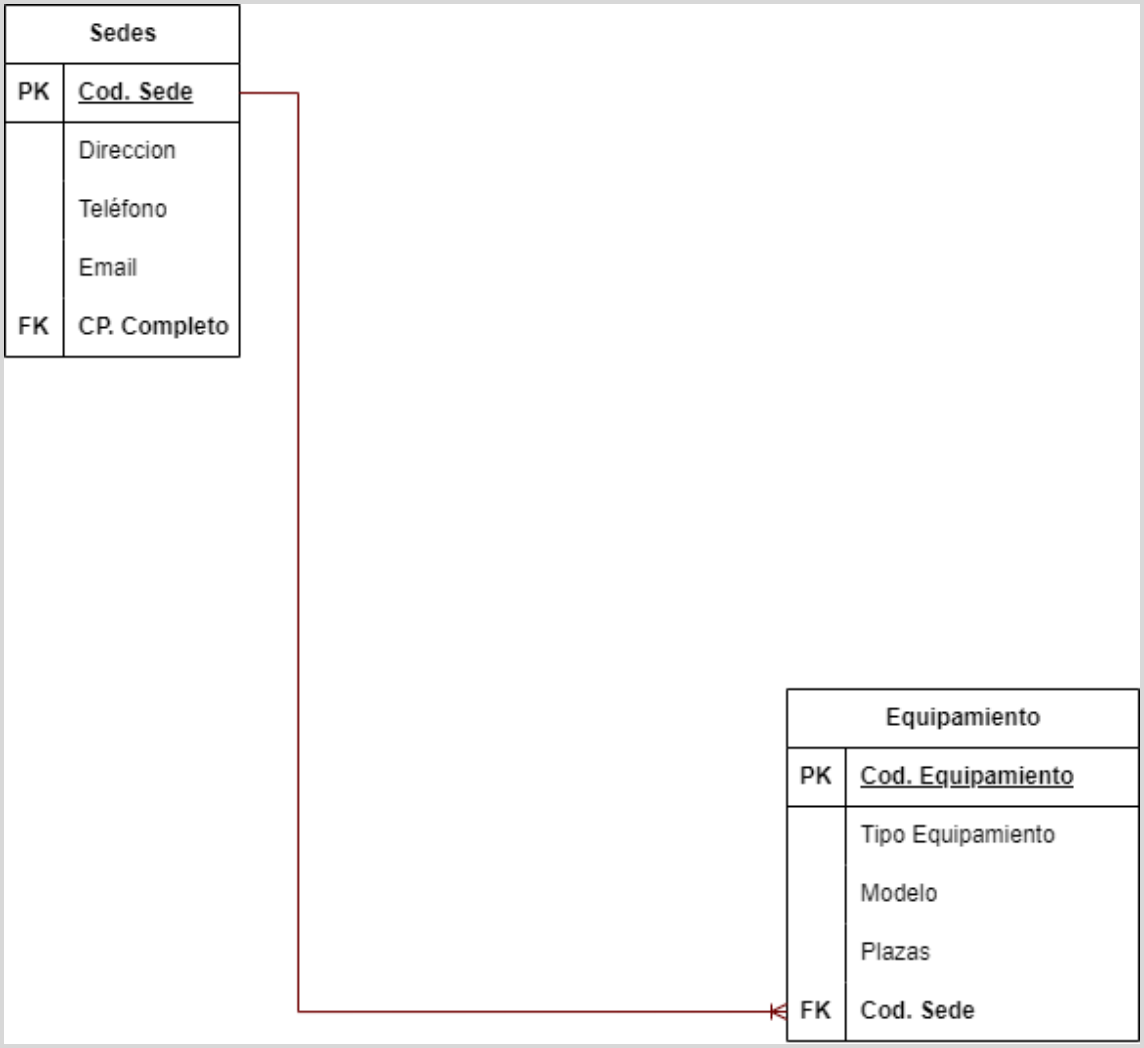

Aquí el resultado de la ingeniería.

## ULTIMAS RELACIONES A RESOLVER

Llegados a este punto, ya casi he resulto todas las relaciones y las he añadido al script, que no quiere decir que las que queden sean más fáciles o ligeras.

Así que vamos con ellas.

Había hablado del equipamiento que podía pertenecer a las sedes, y que luego este equipamiento era revisado por nuestros técnicos, así que vamos a resolver esa parte.




Lo primero es centrarse en la tabla equipamiento. La cual recibe como FK la PK de sedes.

```
#Creo la tabla equipamientos
CREATE TABLE equipamiento(
    cod_equipamiento CHAR(6) UNIQUE NOT NULL,
    tipo_equipamiento VARCHAR(15) NOT NULL,
    modelo VARCHAR(15) NOT NULL,
    plazas SMALLINT,
    cod_sede CHAR(6) NOT NULL,
    CONSTRAINT cod_equipamiento primary key(cod_equipamiento),
    CONSTRAINT sede_propietaria foreign key(cod_sede) references sedes(cod_sede)
);
```

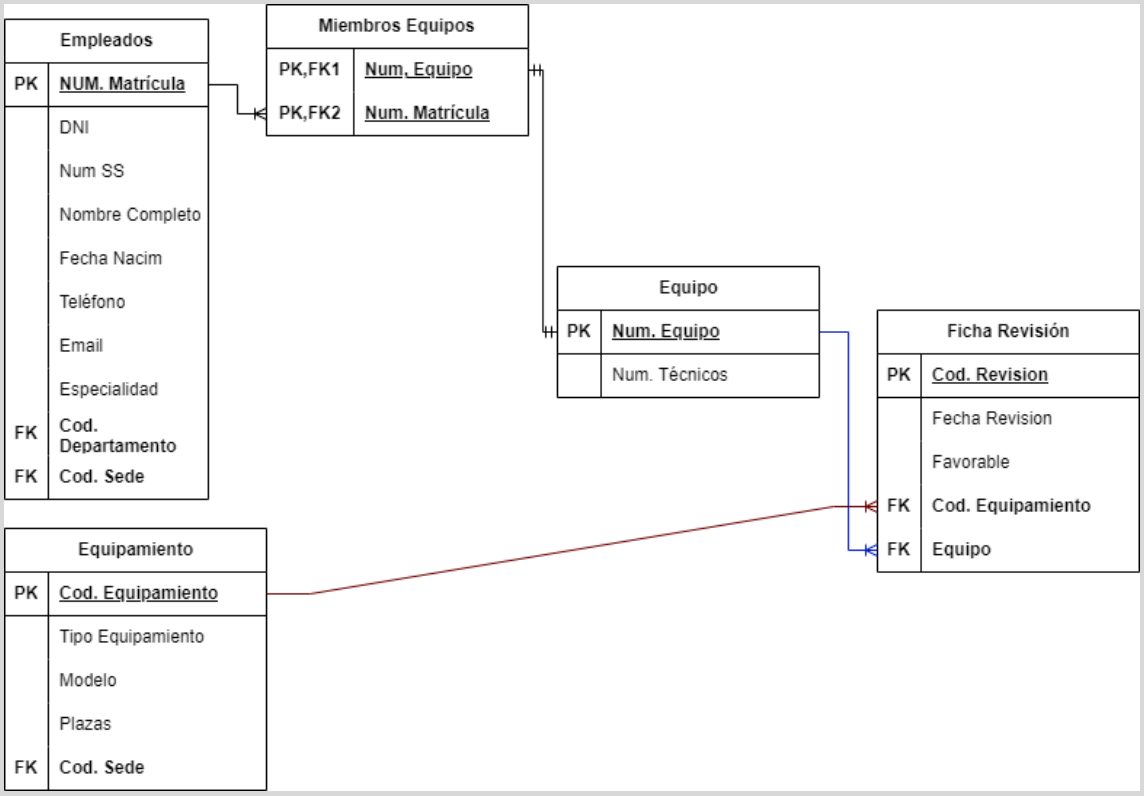
Más adelante me di cuenta que cod_sede tenía que ser INT y lo cambié.

Viernes 20 de diciembre de 2024

En la foto anterior se puede ver la creación de la tabla equipamiento. Como siempre el código, salvo en algunos casos, es un CHAR(6) ya que creo que es mejor para poder tener más variedad de código a crear.

Un par de campos descriptivos que son VARCHAR(15) como tipo de equipamiento y modelo. Plazas es un SMALLINT pero que puede tener un valor nulo ya que quizá ese tipo de equipamiento no se refiera a un vehículo propio.

Añado también el cod_sede como FK.

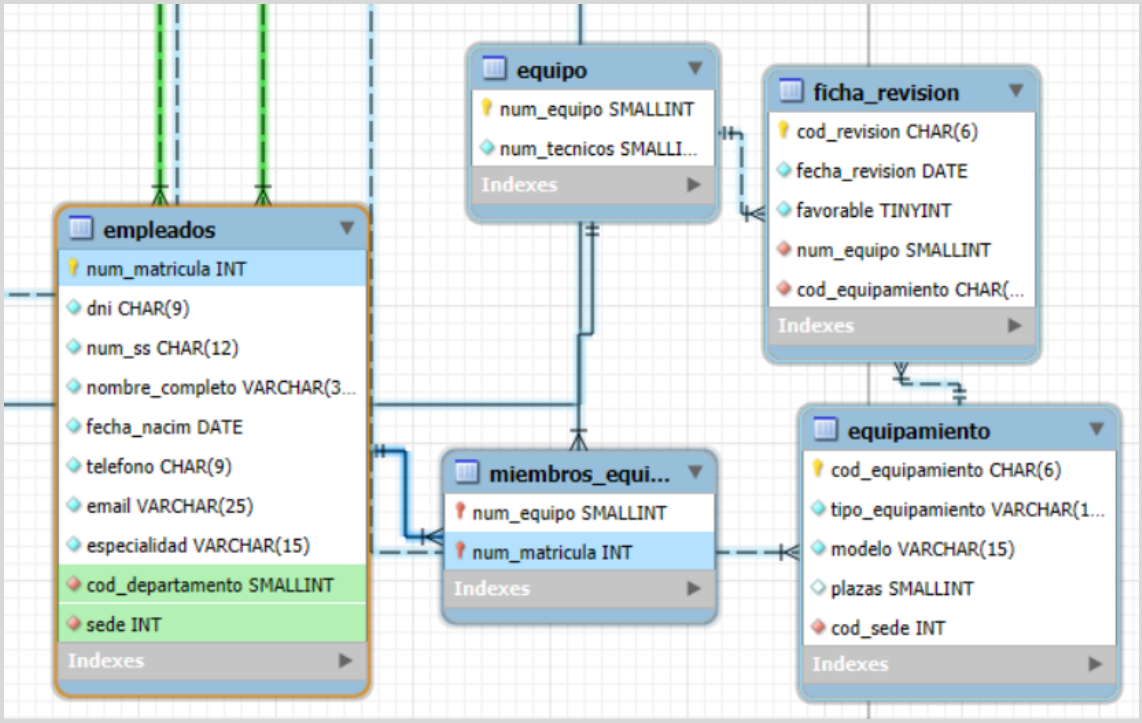



Antes de echar un vistazo a como va quedando el diagrama voy a resolver todo lo relacionado con el equipamiento.

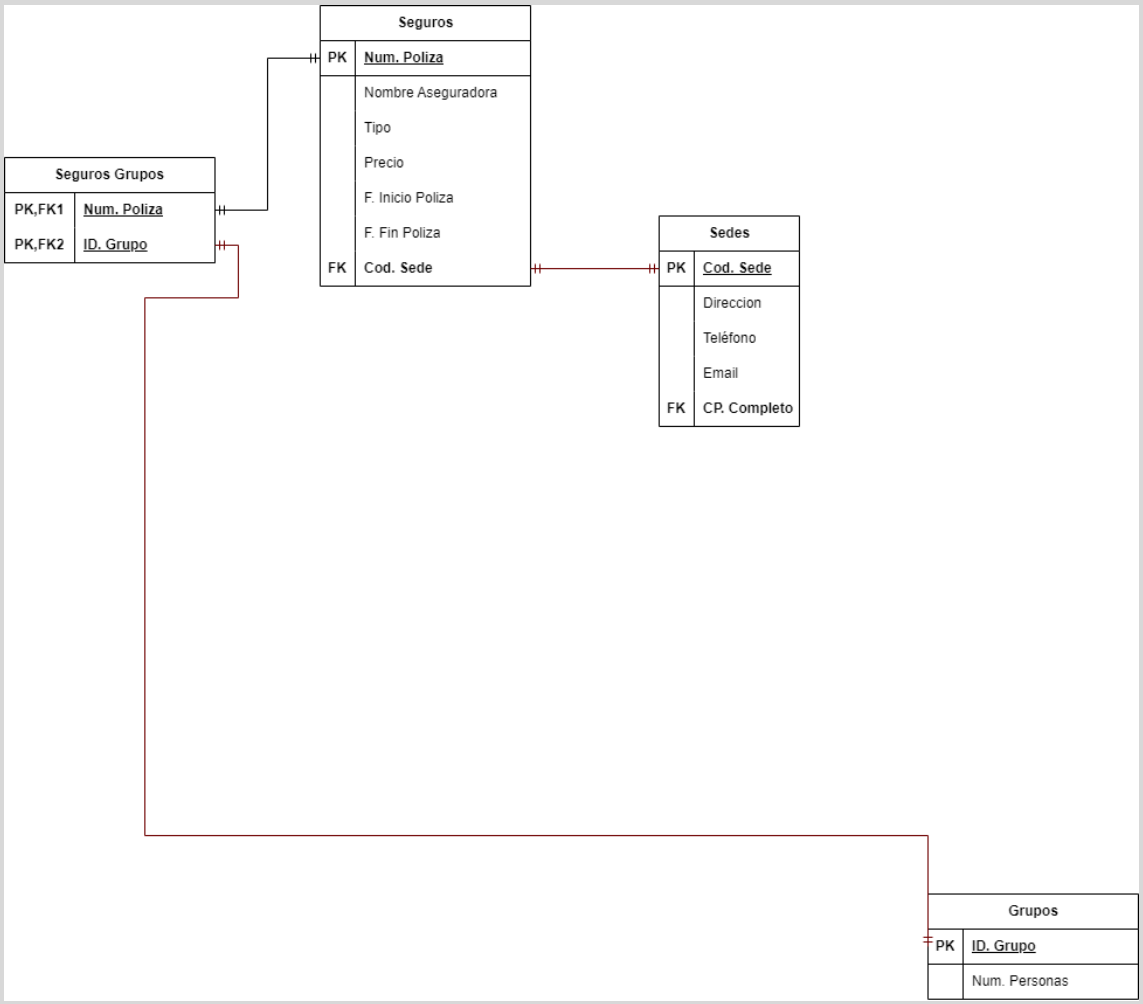
En primera instancia debo de crear la tabla equipo, luego las dos tablas intermedias en cualquier orden.

```
#Creo la tabla equipamientos
CREATE TABLE equipamiento(
    cod_equipamiento CHAR(6) UNIQUE NOT NULL,
    tipo_equipamiento VARCHAR(15) NOT NULL,
    modelo VARCHAR(15) NOT NULL,
    plazas SMALLINT,
    cod_sede INT NOT NULL,
    CONSTRAINT cod_equipamiento primary key(cod_equipamiento),
    CONSTRAINT sede_propietaria foreign key(cod_sede) references sedes(cod_sede)
);

#Rematamos las relaciones que tengan que ver con la tabla equipamiento
CREATE TABLE equipo(
    num_equipo SMALLINT UNIQUE NOT NULL,
    num_tecnicos SMALLINT NOT NULL default 1,
    CONSTRAINT num_equipo primary key(num_equipo)
);

CREATE TABLE miembros_equipo(
    num_equipo SMALLINT NOT NULL,
    num_matricula INT NOT NULL,
    CONSTRAINT equipo primary key(num_equipo, num_matricula),
    CONSTRAINT cod_equipo foreign key(num_equipo) references equipo(num_equipo),
    CONSTRAINT cod_empleado foreign key(num_matricula) references empleados(num_matricula)
);
```

Este es el fragmento de script de esas 3 tablas. Lo hacemos así y ahorramos algo de texto y de tiempo. El script se ejecuta bien, vamos a intentar ver el diagrama.




De nuevo, de la misma forma que en el ejercicio original, se empieza a complicar la visualización, pero espero poder mejorarlo en la parte final del ejercicio. Así es como quedan estas tablas relacionadas después de ejecutar el script.




En el ejercicio también se hablaba de que se contrataban seguros para los grupos y los vehículos, así que vamos con esas partes, que ya son las últimas, y después veremos como se puede organizar el diagrama para que sea mínima mente legible y entendible.

```
CREATE TABLE seguros(
    num_poliza CHAR(6) UNIQUE NOT NULL,
    nombre_aseguradora VARCHAR (20) NOT NULL,
    tipo VARCHAR(12) NOT NULL,
    precio DECIMAL(6,2) NOT NULL,
    f_ini_poliza DATE NOT NULL,
    f_fin_poliza DATE NOT NULL,
    cod_sede INT NOT NULL,
    CONSTRAINT num_poliza primary key(num_poliza),
    CONSTRAINT sede_contrato foreign key(cod_sede) references sedes(cod_sede)
);

#Después de crear la tabla seguros ya puedo crear la otra tabla intermedia para después pasar a la resolucción de vehículos
CREATE TABLE seguro_grupo(
    num_poliza CHAR(6) NOT NULL,
    id_grupo INT NOT NULL,
    CONSTRAINT seguro_grupo primary key(num_poliza, id_grupo),
    CONSTRAINT poliza_grupo foreign key(num_poliza) references seguros(num_poliza),
    CONSTRAINT grupo_asegurado foreign key(id_grupo) references grupos(id_grupo)
);
```

En el script anterior se crean la tabla seguros y la tabla intermedia que la relaciona con grupos.

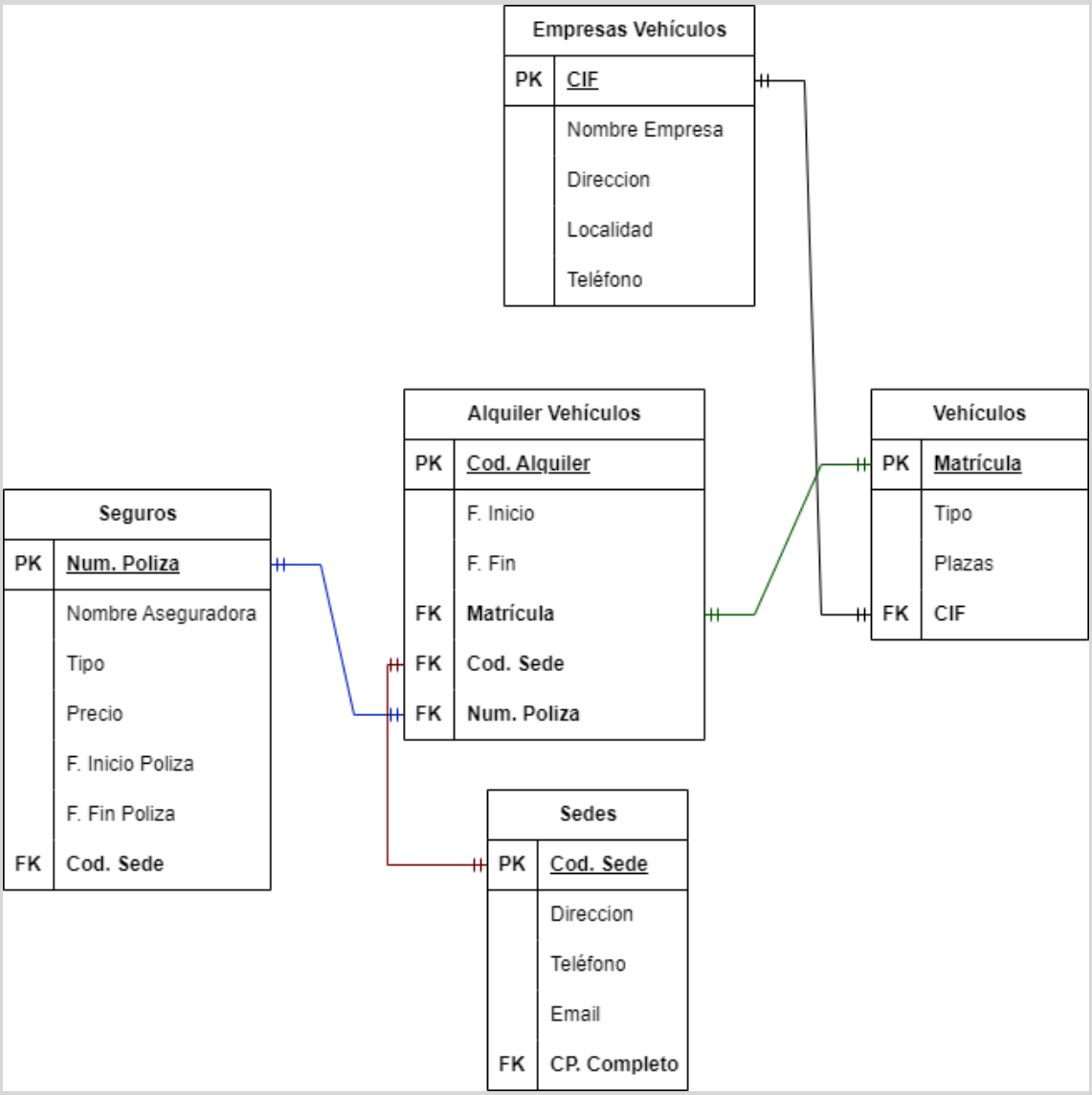
**Incluyo esto aquí, ya que es una corrección. Después de revisar todo, me di cuenta de que podía ahorrarme esta tabla intermedia en la que los seguros se relacionan con los grupos, ya que es más eficiente si relaciono la tabla seguros con la tabla reservas, ya que esa tabla relaciona todo lo necesario con los grupos que contratan la reserva. Dicho esto, procedo a eliminar la tabla intermedia del script y a modificar la tabla reservas.

```
#Como indico en el informe puedo realizar este cambio aquí para optimizar la BD y el diagrama
ALTER TABLE reservas ADD num_poliza CHAR(6) NOT NULL;
ALTER TABLE reservas ADD CONSTRAINT poliza_reserva foreign key(num_poliza) references seguros(num_poliza);
```

Eliminando el código de la tabla intermedia y añadiendo estas dos líneas puedo optimizar todo mucho más, o eso creo yo.

De todo eso me di cuenta al ver la parte de vehículos y su tabla alquileres.

Ahora vamos con la parte de vehículos.

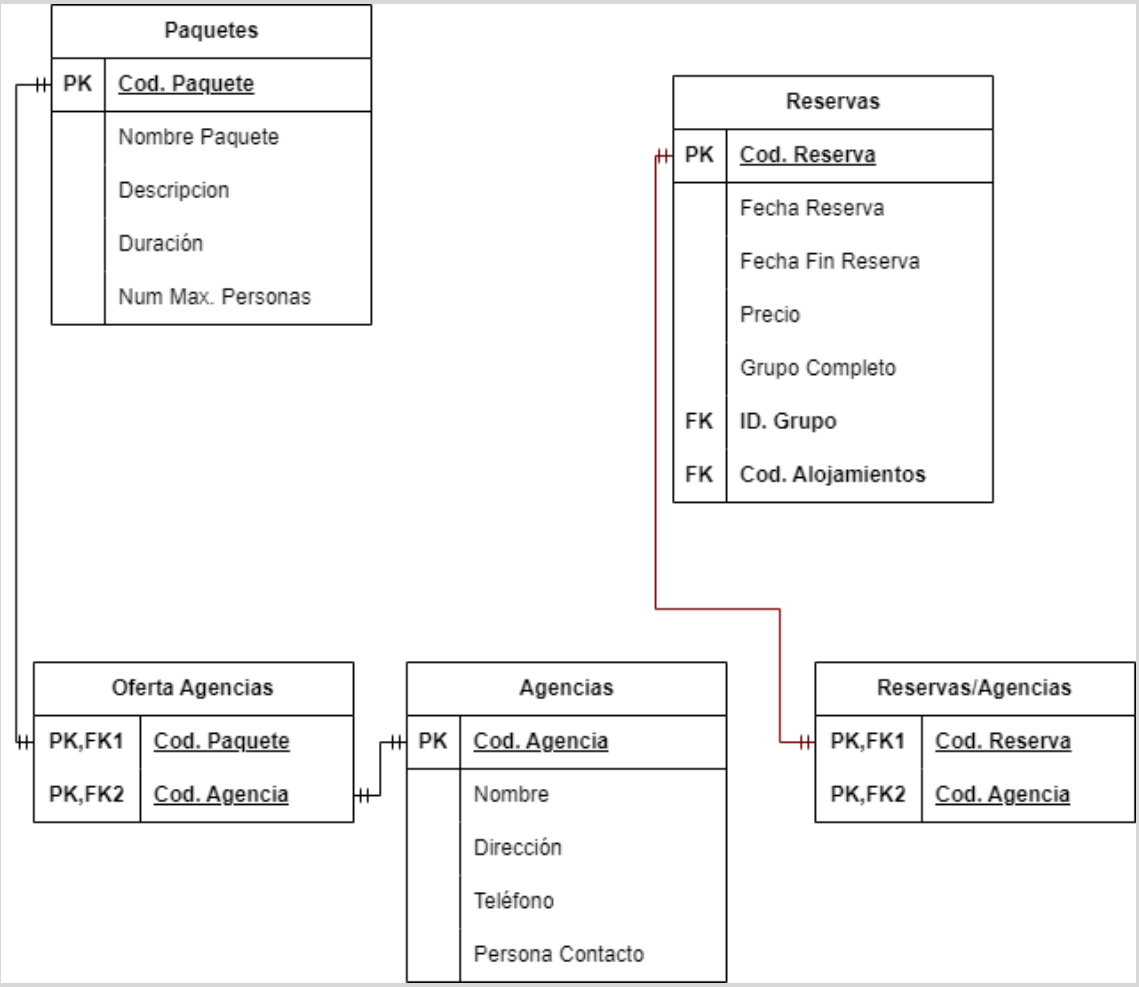



La tabla empresas_vehiculos ya esta creada, así que solo tengo que añadir la tabla vehículos y la intermedia, alquiler vehículos.

```
#Vamos con la parte de vehículos
CREATE TABLE vehiculos(
    matricula CHAR(7) UNIQUE NOT NULL,
    tipo VARCHAR(15) NOT NULL,
    plazas SMALLINT NOT NULL,
    cif CHAR(9) NOT NULL,
    CONSTRAINT matricula primary key(matricula),
    CONSTRAINT identificador_fiscal foreign key(cif) references empresa_vehiculos(cif)
);

CREATE TABLE alquiler_vehiculos(
    cod_alquiler CHAR(6) UNIQUE NOT NULL,
    f_inicio DATE NOT NULL,
    f_fin DATE NOT NULL,
    matricula CHAR(7) NOT NULL,
    cod_sede INT NOT NULL,
    num_poliza CHAR(6) NOT NULL,
    CONSTRAINT cod_alquiler primary key(cod_alquiler),
    CONSTRAINT sede_encargada foreign key(cod_sede) references sedes(cod_sede),
    CONSTRAINT poliza_seguro foreign key(num_poliza) references seguros(num_poliza)
);
```

Aquí están creadas ya la tabla vehículos y la de alquiler_vehiculos. En estas partes me estoy saltando las muestras intermedias del diagrama para aligerar las cosas.

Vamos, ahora ya sí, con la parte de las agencias para terminar con todo ello.




Bueno, esto ya fue la última parte de mi ejercicio anterior, que quizá no hacía falta para entender y ejecutar el ejercicio, pero ya que lo presenté así hay que terminarlo.

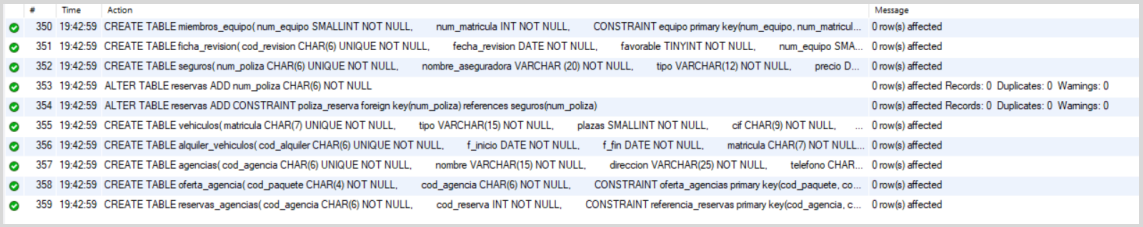
En este caso tengo que crear primero la tabla agencias, después de eso, puedo crear ya las tablas intermedias.

Vamos a ello.

```sql
#Última parte ya, creando la tabla agencias y las auxiliares o intermedias.
CREATE TABLE agencias(
    cod_agencia CHAR(6) UNIQUE NOT NULL,
    nombre VARCHAR(15) NOT NULL,
    direccion VARCHAR(25) NOT NULL,
    telefono CHAR(9) NOT NULL,
    persona_contacto VARCHAR(25) NOT NULL,
    CONSTRAINT cod_agencia primary key(cod_agencia)
);

CREATE TABLE oferta_agencia(
    cod_paquete CHAR(4) NOT NULL,
    cod_agencia CHAR(6) NOT NULL,
    CONSTRAINT oferta_agencias primary key(cod_paquete, cod_agencia),
    CONSTRAINT agencia foreign key(cod_agencia) references agencias(cod_agencia),
    CONSTRAINT paquete_agencia foreign key(cod_paquete) references paquetes(cod_paquete)
);

CREATE TABLE reservas_agencias(
    cod_agencia CHAR(6) NOT NULL,
    cod_reserva INT NOT NULL,
    CONSTRAINT referencia_reservas primary key(cod_agencia, cod_reserva),
    CONSTRAINT agencia_reserva foreign key(cod_agencia) references agencias(cod_agencia),
    CONSTRAINT reserva_externa foreign key(cod_reserva) references reservas(cod_reserva)
);
```

La creación de la tabla agencias es bastante estándar. Luego las otras dos tablas intermedias también, bastante común a lo que ya he hecho en el ejercicio con anterioridad, PK compuesta de dos FK.

Ejecuto el script por última vez (supongo):

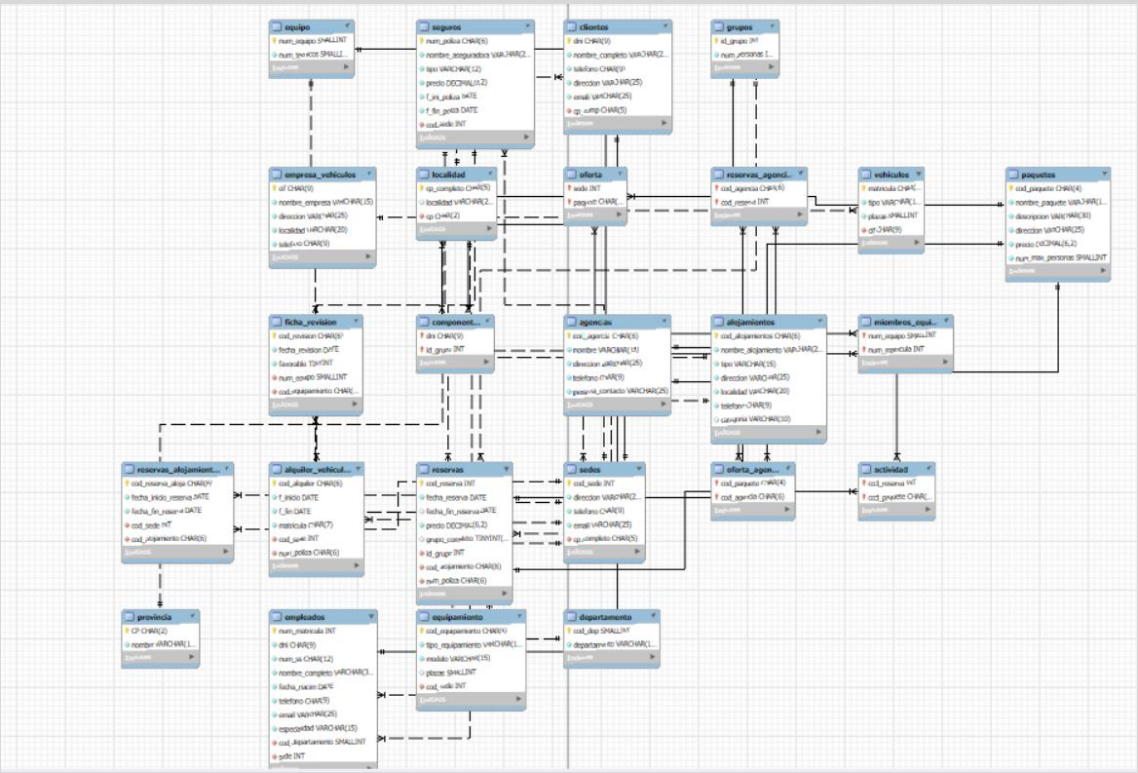


| # | Time | Action | Message |
|---|---|---|---|
| 350 | 19:42:59 | CREATE TABLE miembros_equipo( num_equipo SMALLINT NOT NULL, num_matricula INT NOT NULL, CONSTRAINT equipo primary key(num_equipo, num_matricul... | 0 row(s) affected |
| 351 | 19:42:59 | CREATE TABLE ficha_revision( cod_revision CHAR(6) UNIQUE NOT NULL, fecha_revision DATE NOT NULL, favorable TINYINT NOT NULL, num_equipo SMA... | 0 row(s) affected |
| 352 | 19:42:59 | CREATE TABLE seguros( num_poliza CHAR(6) UNIQUE NOT NULL, nombre_aseguradora VARCHAR (20) NOT NULL, tipo VARCHAR(12) NOT NULL, precio D... | 0 row(s) affected |
| 353 | 19:42:59 | ALTER TABLE reservas ADD num_poliza CHAR(6) NOT NULL | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 354 | 19:42:59 | ALTER TABLE reservas ADD CONSTRAINT poliza_reserva foreign key(num_poliza) references seguros(num_poliza) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 |
| 355 | 19:42:59 | CREATE TABLE vehiculos( matricula CHAR(7) UNIQUE NOT NULL, tipo VARCHAR(15) NOT NULL, plazas SMALLINT NOT NULL, cif CHAR(9) NOT NULL, ... | 0 row(s) affected |
| 356 | 19:42:59 | CREATE TABLE alquiler_vehiculos( cod_alquiler CHAR(6) UNIQUE NOT NULL, f_inicio DATE NOT NULL, f_fin DATE NOT NULL, matricula CHAR(7) NOT NULL... | 0 row(s) affected |
| 357 | 19:42:59 | CREATE TABLE agencias( cod_agencia CHAR(6) UNIQUE NOT NULL, nombre VARCHAR(15) NOT NULL, direccion VARCHAR(25) NOT NULL, telefono CHAR... | 0 row(s) affected |
| 358 | 19:42:59 | CREATE TABLE oferta_agencia( cod_paquete CHAR(4) NOT NULL, cod_agencia CHAR(6) NOT NULL, CONSTRAINT oferta_agencias primary key(cod_paquete, co... | 0 row(s) affected |
| 359 | 19:42:59 | CREATE TABLE reservas_agencias( cod_agencia CHAR(6) NOT NULL, cod_reserva INT NOT NULL, CONSTRAINT referencia_reservas primary key(cod_agencia, c... | 0 row(s) affected |

Todo Ok, salvo que no lo ejecuté por última vez. Al tener tantas tablas he creado nombres de FK repetidos, así que ha tocado modificar un par de cosas y probar hasta que todo queda como tiene que quedar.

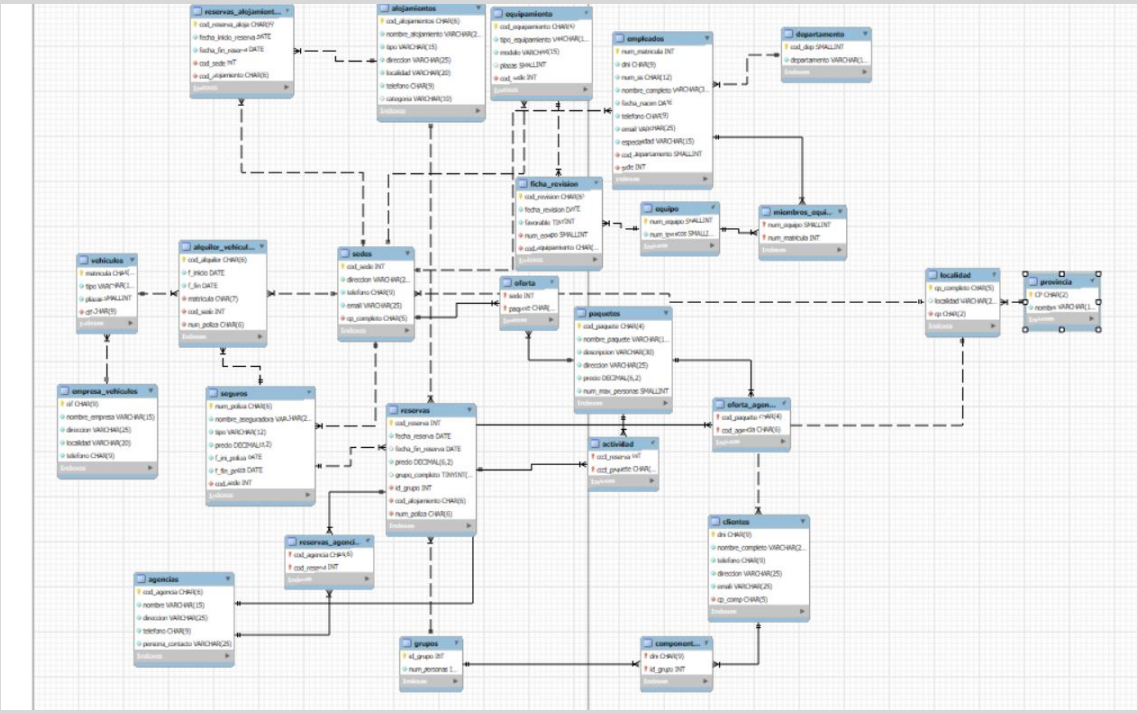
# DIAGRAMA FINAL

Después de generar todo el código necesario vamos a ejecutar la ingeniería inversa y ver cómo queda todo.



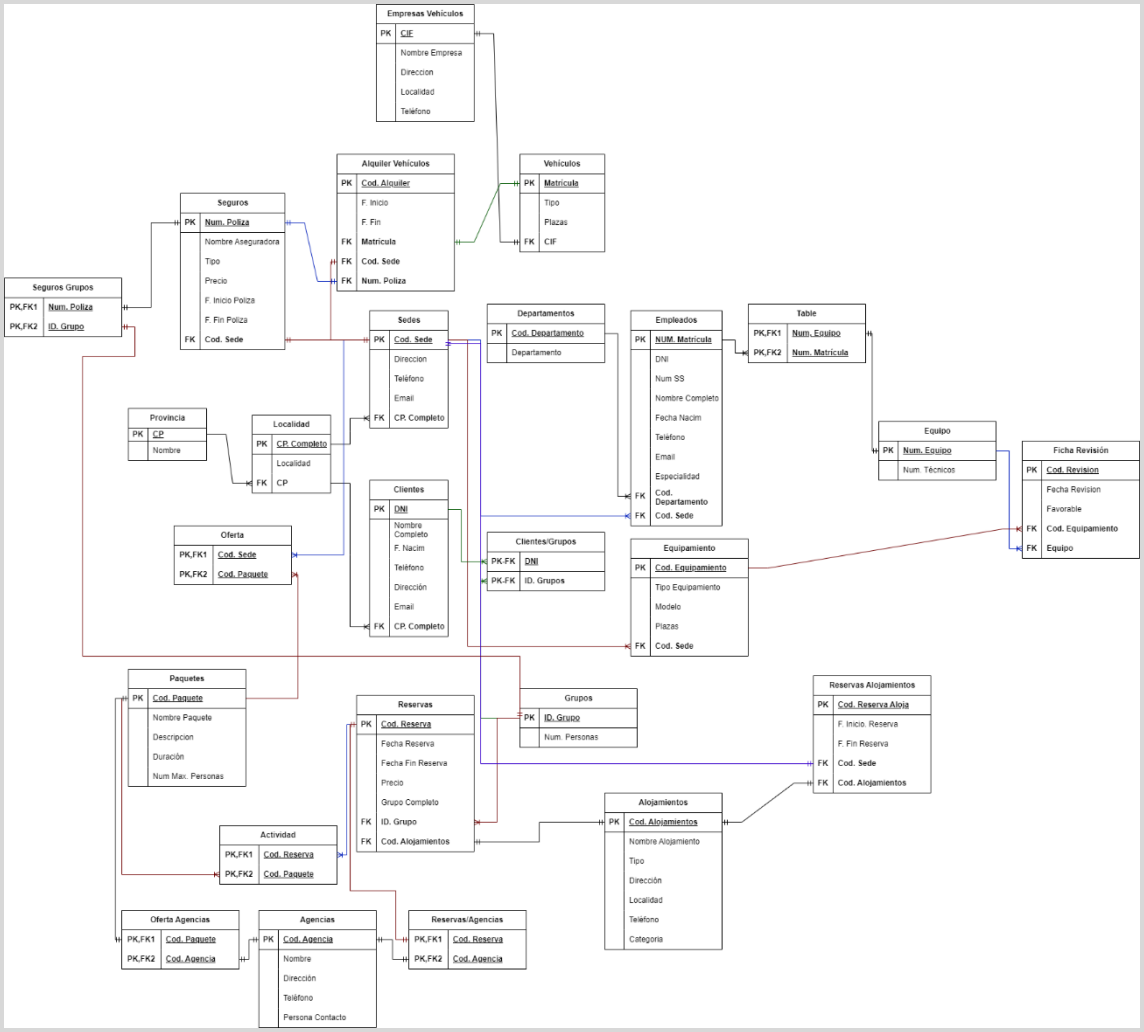
En un principio el programa lo genera así. Voy a intentar ordenarlo un poco para que quede más legible.

Viendo el diagrama y moviendo las tablas me doy cuenta de que no añadí matrícula como FK a la tabla alquileres_vehiculos, cosa que arreglo en este momento mientras voy ordenando el diagrama.



Este sería el diagrama, colocándolo un poco más, aún así, es un diagrama bastante grande, con muchas relaciones entre muchas tablas. A ratos se sigue haciendo difícil de leer.

Es bastante parecido al modelo que diseñé yo, este evidentemente más claro, y refleja lo que yo hice en el anterior ejercicio. Sigue siendo complicado de seguir en algunas partes ya que hay muchas tablas y para mí es difícil conseguir que no se corten demasiadas líneas, y eso que he intentado colocarlo lo mejor posible para que quede claro que líneas van de un sitio a otro.




Como se puede comprobar comparándolo con el mío, el mio es bastante más complicado de ver, aunque en estructura son bastante similares, ya que alguna cosa he ido cambiando al crear el código SQL.

# MODIFICACIONES

No he hecho demasiadas modificaciones en el ejercicio, añadir algún atributo nuevo y poco más.

El cambio más relevante ya lo expliqué con anterioridad al introducir el código SQL. Relacionando seguros con las reservas es más eficiente y práctico que en mi diseño inicial. Todo esto gracias a eliminar una tabla intermedia.

Creo que es la decisión correcta, en principio no me parece que genere problemas, al contrario, creo que los minimiza.

La otra opción que me da vueltas por la cabeza es sobre el tema de las agencias, que es algo de lo que al final me he terminado arrepintiendo. Se generan demasiadas tablas intermedias y no he encontrado la forma de mejorar eso.

Pensé en poner en la tabla reservas un atributo que se refiriese a la sede y otro a la agencia, y que los dos pudiesen tener valores nulos, dando a entender que en la tabla reserva se relaciona todo con la sede o la agencia que lo gestiona. Pero como ninguna de las opciones que pensé me terminaban de convencer, además de que no sabía como resolverlas de forma correcta, si es que la hay, he preferido dejarlo tal cual lo había planteado en un principio, para no terminar complicándolo todo más.

# MI DIAGRAMA RELACIONAL

Me imagino que lo lógico habría sido que esto fuese en primer lugar, pero tomé la decisión que, ya que tenía el diagrama hecho, ir generando la BD acorde a lo que ya tenía e ir modificándolo sobre la marcha.

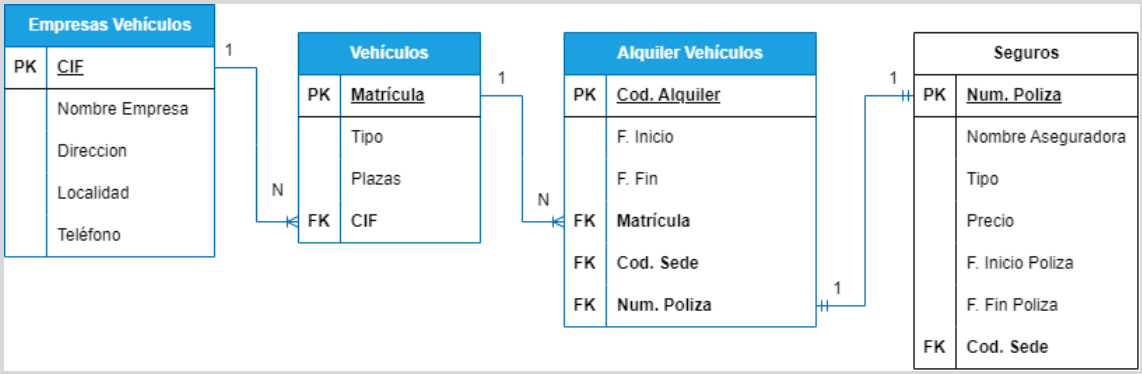
Después de esto iría adaptando el diagrama al trabajo que he realizado, para que quedase todo acorde.

Así que ahora, lo que queda, es ir dejando el diagrama en condiciones, con los tipos de datos en las tablas y las cardinalidades bien puestas, que ya veré que tal me queda.

Lo primero que he hecho es asociar los seguros con las reservas, para que quede como en el ejercicio que he ido realizando. Con eso, elimino la tabla intermedia, y es algo menos a tener en cuenta.

Quizá podría darse el caso de que una reserva no lleve asociada una póliza de seguros, por lo que podría tener un valor de null. Ahora que lo pienso quizá el cambio puede tener sus puntos en contra, pero creo que voy a seguir adelante con ello.

## COMENZANDO CON CAMBIOS Y AJUSTES

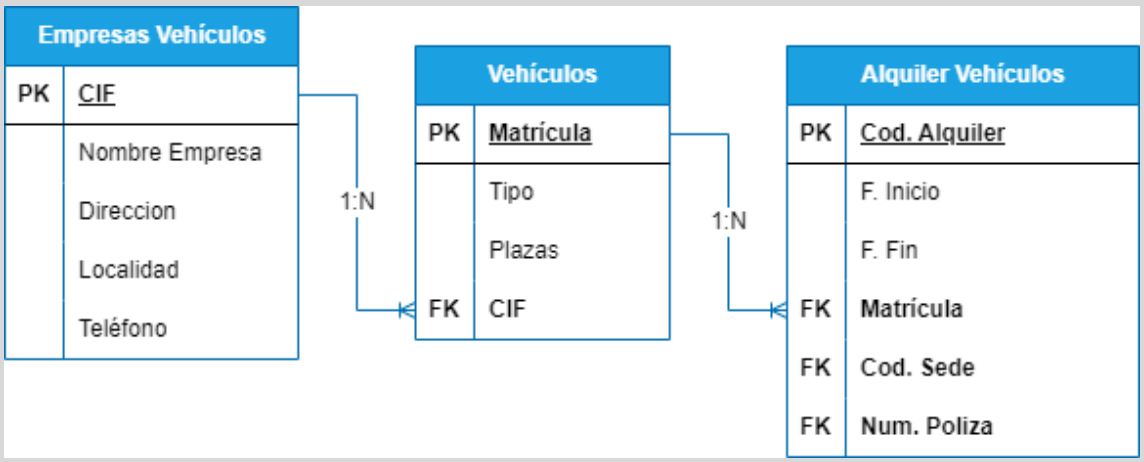



Pare empezar a aclarar el diagrama que ya tenía, antes de añadir los tipos de atributos de cada tabla, lo que hago es colocar cardinalidades e intentar que cada cosa vaya resaltando por bloques.

En este caso, la parte referente a las empresas de vehículos, los vehículos mismo, el alquiler de los mismos y las pólizas de seguros que los cubren. Marco las cardinalidades, espero que estén de forma correcta.

Según he ido avanzando con las modificaciones y la ordenación del diagrama he ido cambiando también la forma en la que anotaba las cardinalidades.

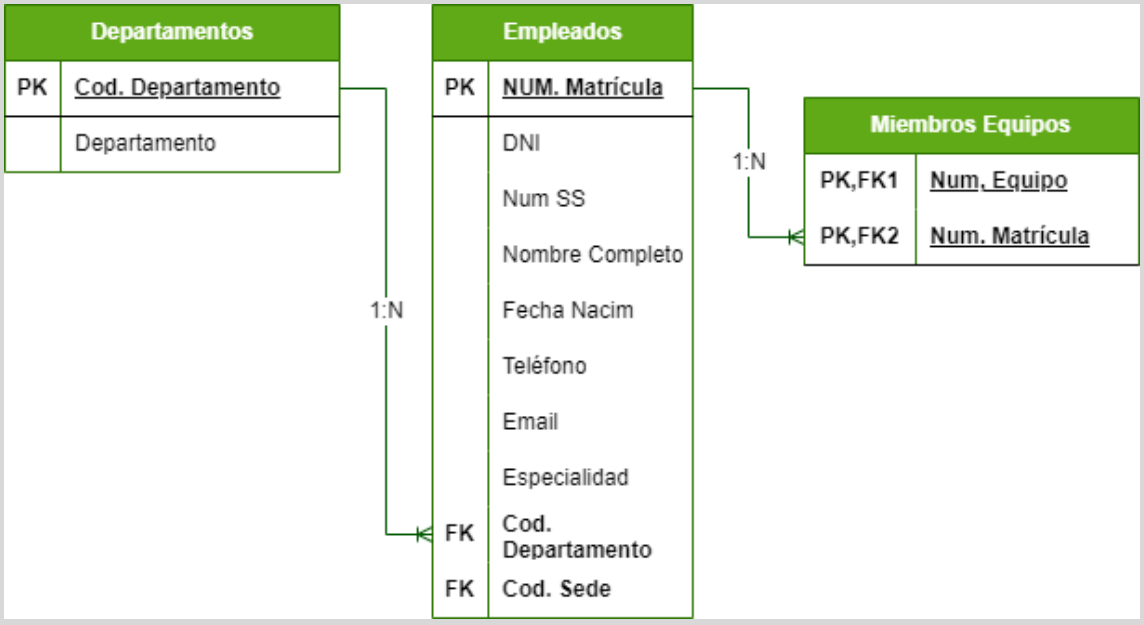
Un ejemplo:

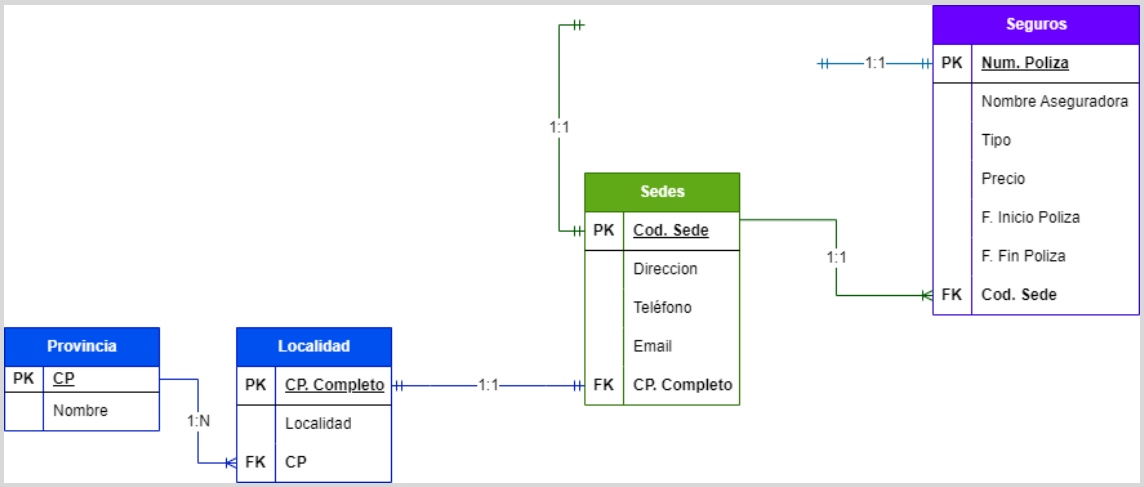


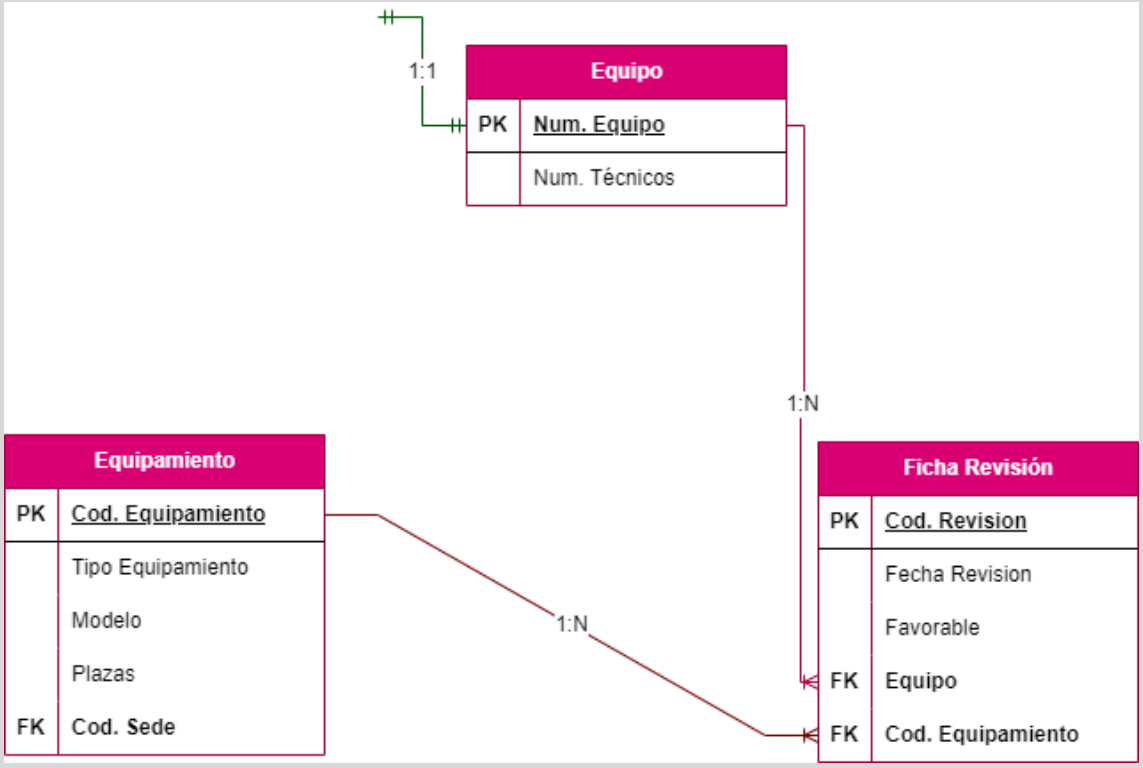

He añadido la cardinalidad a la línea de relación correspondiente, así es más difícil perderse y más fácil tener la cardinalidad bien relacionada con las líneas de relación de cada tabla.

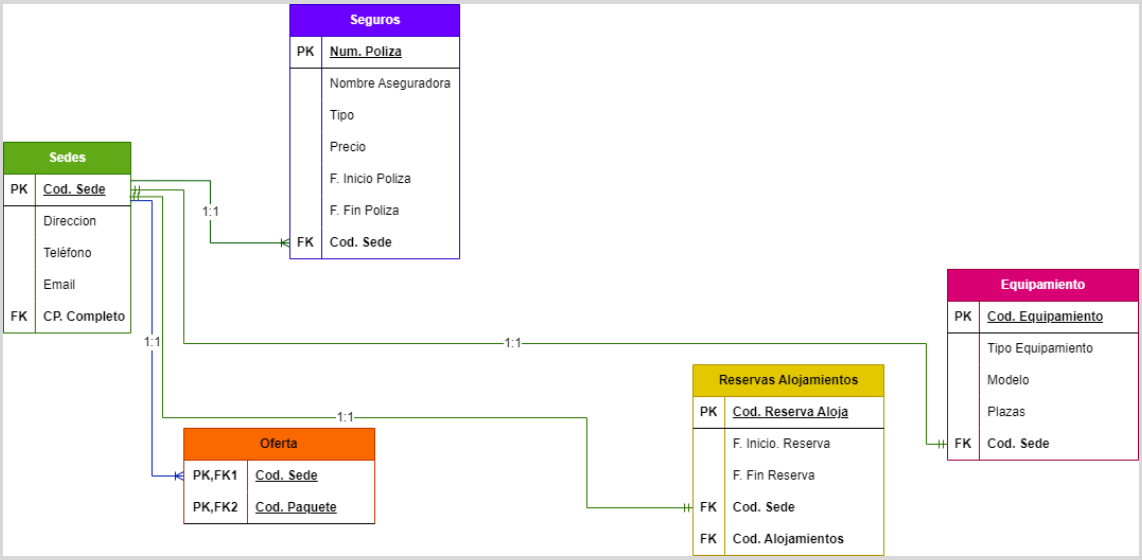
He seguido este método para cada parte del diagrama, y aunque mucho más ordenado y visual, me ha sido imposible evitar algunos cruces de líneas. Ahora bien, aunque el diagrama es ahora más grande, no en tablas sino en el tamaño que ocupa, se ve mucho mejor, incluso en los cortes entre relaciones.

Voy a ir poniendo varias imágenes de las partes, y luego una en conjunto, para que se vea mejor.




Seguros
PK Num. Poliza
Nombre Aseguradora
Tipo
Precio
F. Inicio Poliza
F. Fin Poliza
FK Cod. Sede
1:1
Sedes
PK Cod. Sede
Direccion
Teléfono
Email
FK CP. Completo
1:1
1:1
Provincia
PK CP
Nombre
1:N
Localidad
PK CP. Completo
Localidad
FK CP
1:1

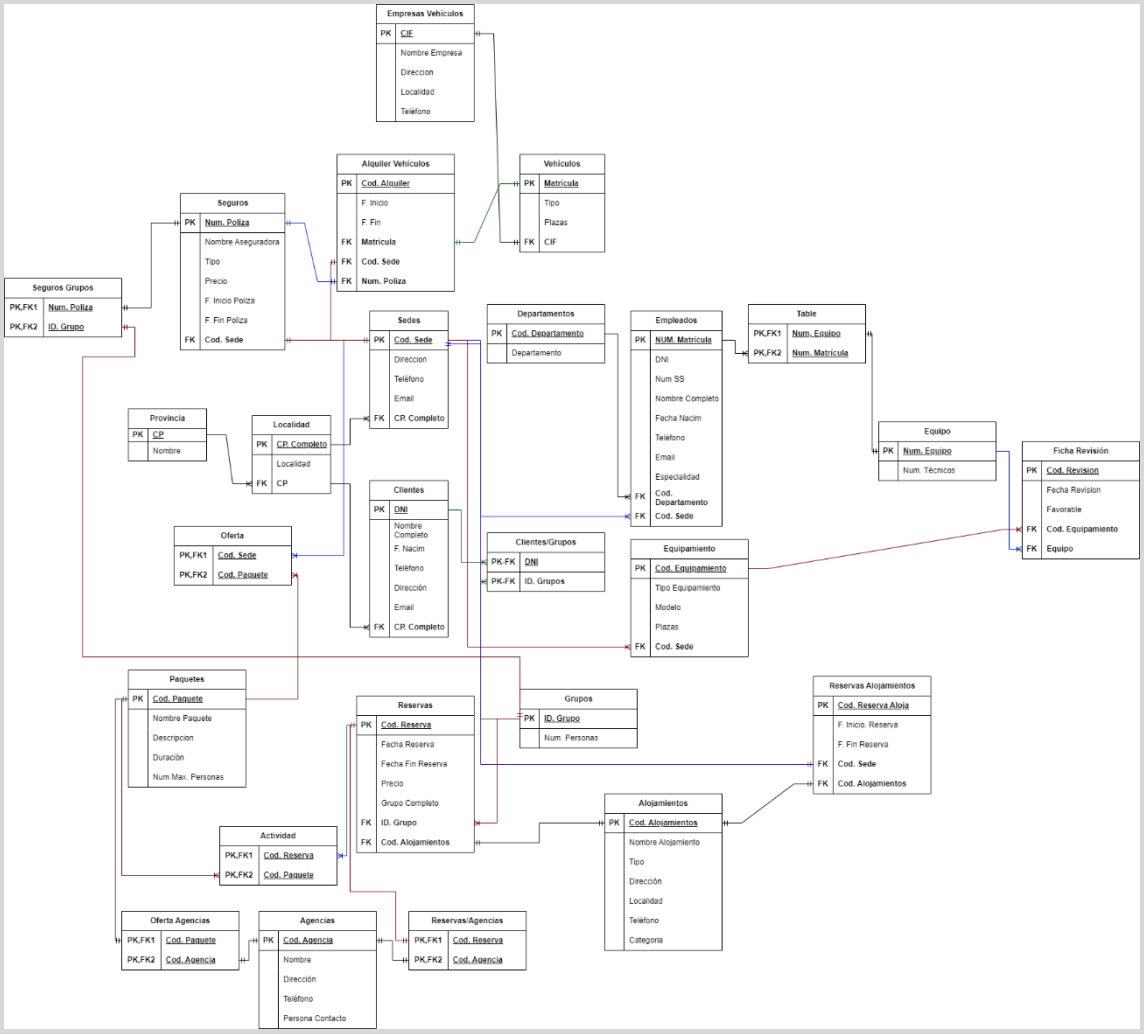

1:1
Equipo
PK Num. Equipo
Num. Técnicos
1:N
Equipamiento
PK Cod. Equipamiento
Tipo Equipamiento
Modelo
Plazas
FK Cod. Sede
1:N
Ficha Revisión
PK Cod. Revision
Fecha Revision
Favorable
FK Equipo
FK Cod. Equipamiento




Con estos ejemplos vamos viendo como va quedando. Todo identificado por colores, cada bloque del diagrama tiene el mismo color para tablas principales e intermedias, la cardinalidad mostrada en cada línea de relación, etc.

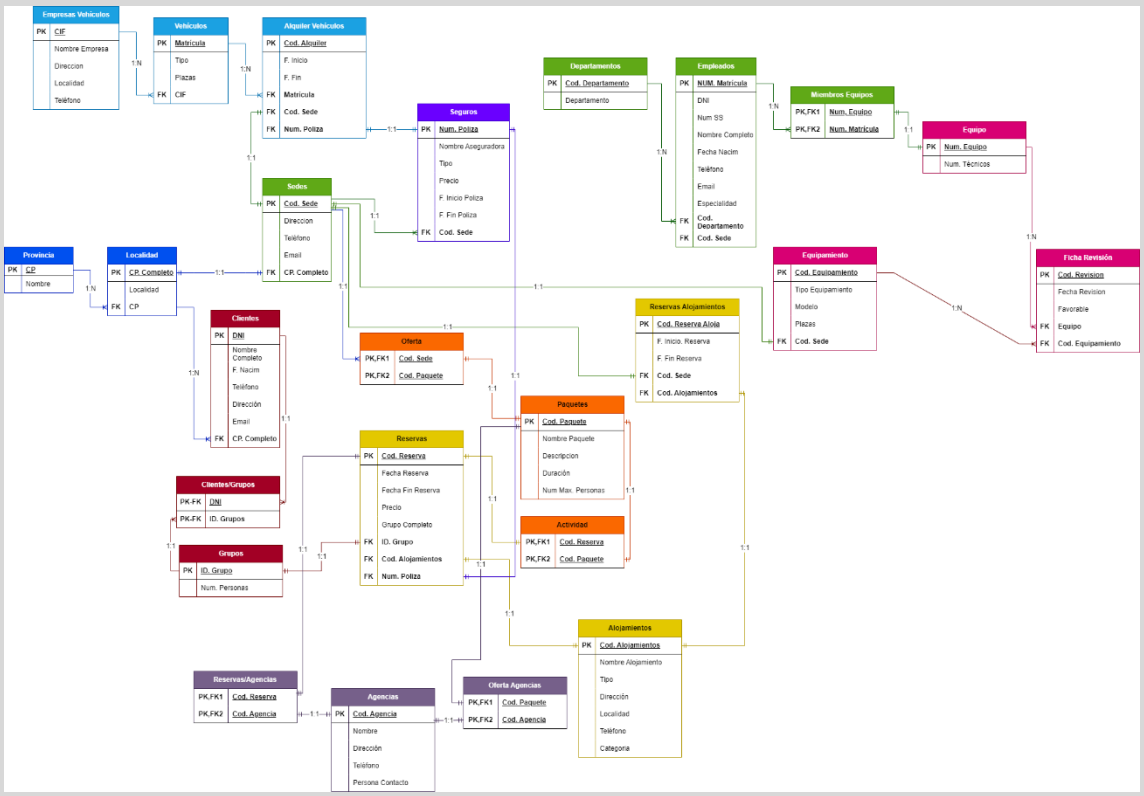
Ahora vamos a mostrar el diagrama completo, que como ya dije, he sido incapaz de evitar que se solapen algunas líneas. Pero, para ser sinceros, al ponerlo junto al diagrama original es como la noche y el día.

## DIAGRAMA VIEJO




Aquí podemos apreciar, en comparación con el actual, el desorden y el caos.

## DIAGRAMA NUEVO




Aunque siguen existiendo zonas que pueden generar algo de confusión, al estar dividida cada sección en colores, con las líneas de su mismo color y las cardinalidades marcadas dentro de cada línea relacional, se puede ver todo mucho más claro.

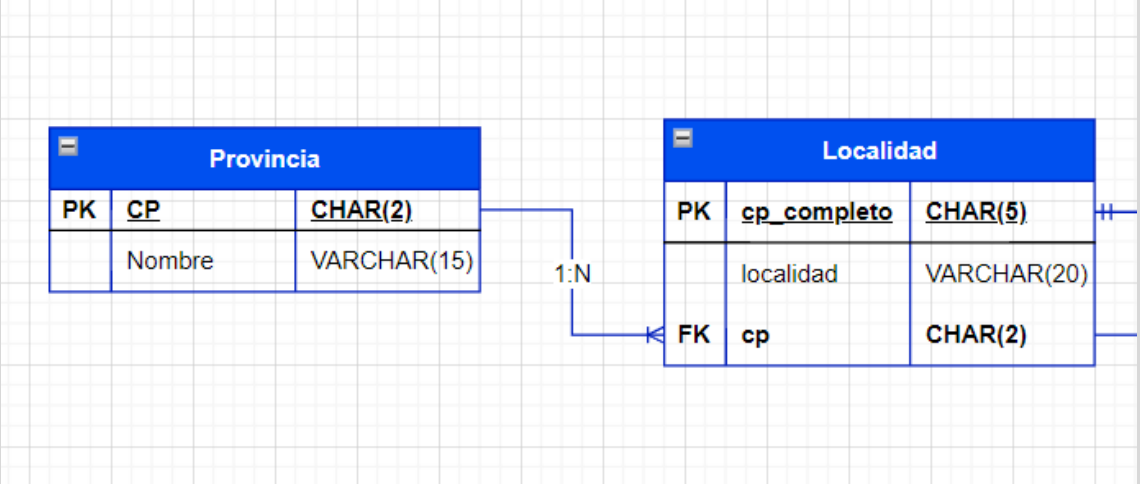
También he cambiado algo la disposición para intentar evitar que se crucen demasiado las líneas.

Pero, de todos modos, hay que tener en cuenta, que todavía tengo que modificar los nombres de los campos para que coincidan con aquellos dados en SQL, ya que SQL prefiere otro tipo de nomenclatura, y hay que añadir los tipos de datos, que eso añade algo más de complejidad al diagrama.

Así que vamos a ello.

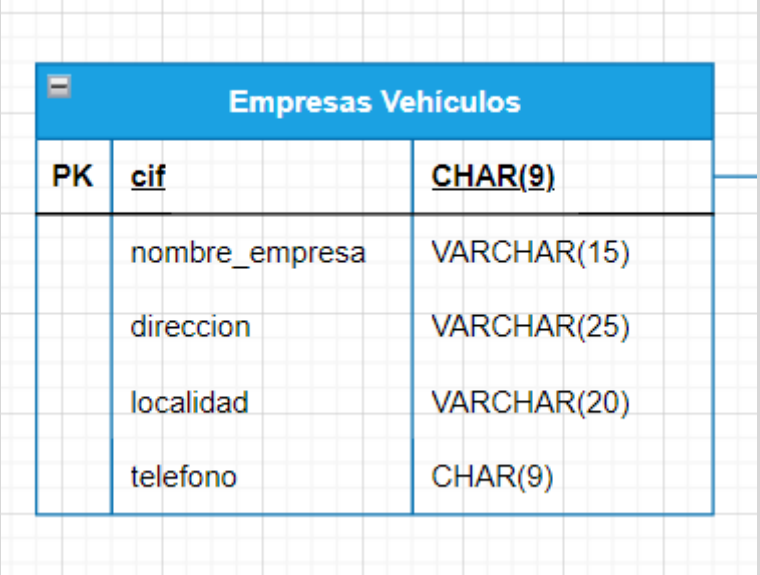
# TERMINANDO DE MODIFICAR EL DIAGRAMA

## MODIFICANDO LOS ATRIBUTOS




Comienzo por la parte de Provincia y Localidad. He cambiado los nombres de los atributos a minúsculas en los casos en los que en sql están de este modo, para que todo este de la misma forma.

En cuanto a los tipos, CP en provincia es CHAR(2) puesto que para identificar una provincia sólo necesito los dos primeros dígitos del código postal, en cp_completo de la localidad es del mismo tipo pero de longitud fija 5 ya que necesito el código completo. En cuanto a localidad será un VARCHAR(20) para poder almacenar el nombre completo de la localidad.


Como aclaración comentar, que voy añadiendo los cambios en el mismo orden que he ido generando el script sql. Ya se que el orden parece aleatorio, pero en principio es porque empecé creando las tablas que no eran dependientes, luego fui añadiendo el resto.

En cuanto a estas dos tablas, decir de nuevo que fui cambiando los nombres para que fuesen iguales en la BD que en el diagrama. Los campos que en estas tablas son descriptivos o sólo texto, como dirección, localidad o departamento, comentar que les asigno a lo largo de todo el ejercicio el tipo VARCHAR, pues su longitud puede ser variable, y a aquellos que se repiten en más de una tabla intento darles siempre la misma longitud (creo que es así, quizá en alguno se me haya pasado y tenga una longitud diferente).

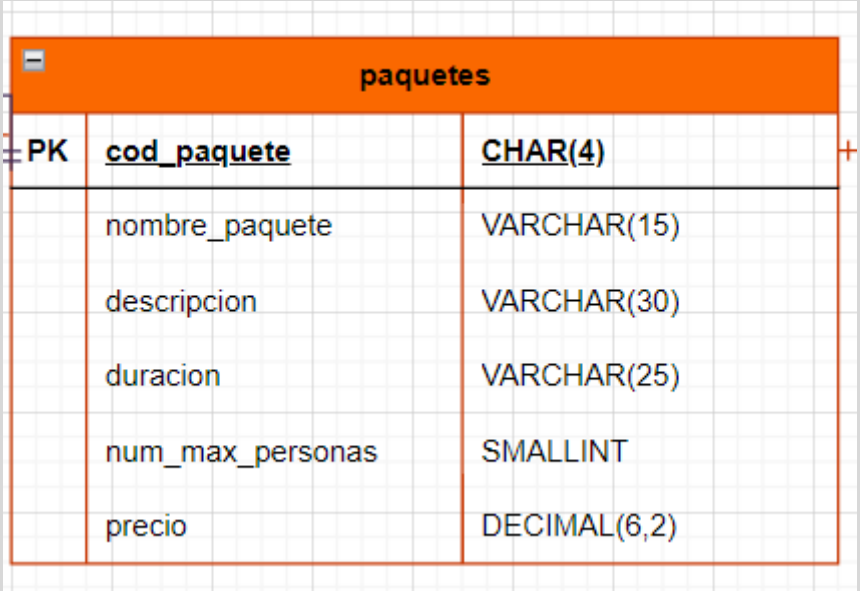
Lo mas reseñable en estas dos es:

cod_dep: un SMALLINT con autoincrement, por si se crean más departamentos.

cif: CHAR(9) pues el cif siempre está formado por una letra y 8 dígitos.

telefono: en todas las tablas que aparece es CHAR(9) pues siempre tiene esa longitud.

Viernes 20 de diciembre de 2024




Comentar también, que intento cambiar los nombres de las tablas para que coincidan con los de la BD.

En este caso, lo más reseñable:

cod_paquete: CHAR(4) para definir el código de un paquete con números y letras me pareció más que suficiente.

Num_max_personas: SMALLINT. Podría haberlo definido como char también, pero consideré, que en caso de que se quisiese hacer un cálculo que involucre a las personas que pueden ejecutar una actividad SMALLINT me vendría muy bien.

Precio: Como me indicaste en clase cuando pregunte, DECIMAL(6,2). Creo que más que suficiente para alojar los precios. De aquí en adelante, el resto de atributos precio que se incluyen en el modelo son todos DECIMAL(6,2).


En esta tabla sigo las mismas directrices que para las anteriores.

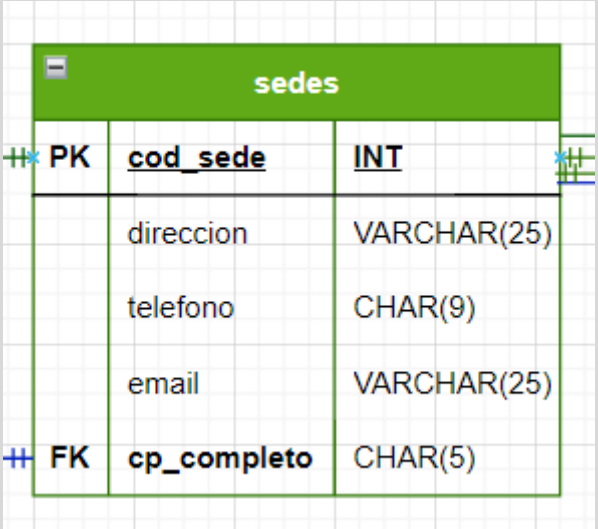
El único atributo que puede cambiar un poco es cod_alojamiento que es un CHAR(6), para identificar a todos los alojamientos que se contratan creo que una combinación alfanumérica de 6 caracteres es lo más idóneo.

El resto sigue el estándar, son campos descriptivos que pueden tener una longitud variable. Teléfono va a ser del mismo tipo que en otras tablas.


Ahora era el tuno de grupos, y ya de paso hice el de la tabla componentes (que relaciona a grupos con clientes).

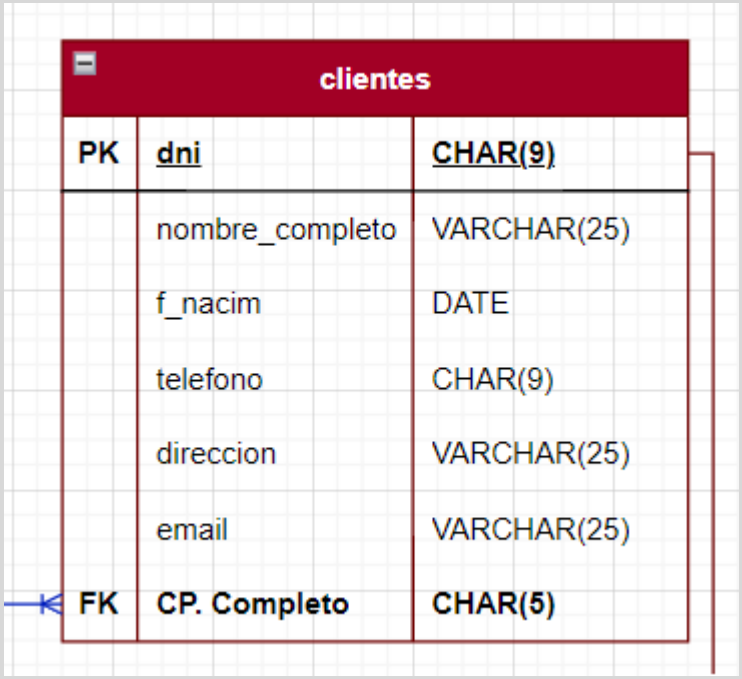
En grupos el id es int, para que pueda ser un campo auto_incremente, y que cada vez que creemos un grupo nuevo se genere el número de forma secuencial y automática. num_personas es int, por si se llegase a dar el caso de que quisiésemos hacer algún cálculo basado en las personas y precio, la media de personas por grupo, etc.

En la tabla componentes id_grupo es int, pues la FK debe de ser del mismo tipo que el atributo al que hace referencia y dni es un CHAR(9) como en la tabla clientes, ya que el dni tiene una longitud fija de 9 caracteres, 8 dígitos y una letra.



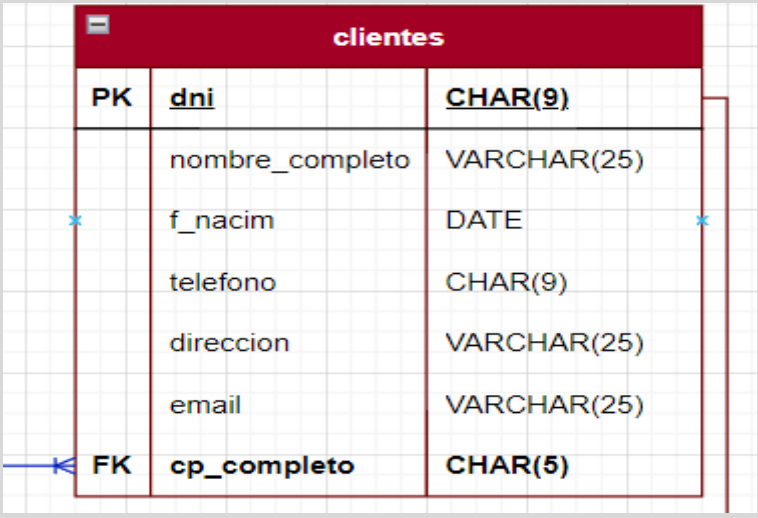

La tabla sedes, que es una de las centrales del ejercicio. Le cambio el nombre para que se ajuste bien al modelo. Todos sus atributos ya han aparecido en otras tablas, por lo que siguen la misma convención. Cp_completo al ser una FK tiene el mismo tipo que en su tabla de origen, la cual ya expliqué anteriormente.

Cod_sede por su parte es del tipo int, para que pueda ser auto_incremente y cada vez que abramos una nueva sede, se incremente de forma automática.



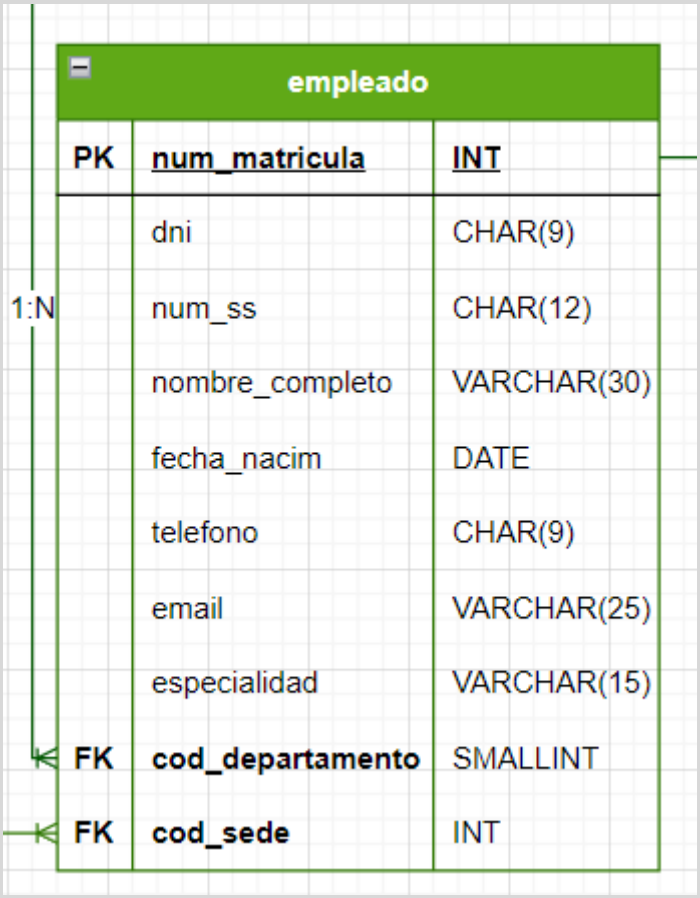

En esta tabla un par de aclaraciones. En el script se me olvidó inserta el campo f_nacim, cosa que hice al completar esta tabla, dándole un tipo de dato DATE, que es el que más se ajusta. El resto de atributos reciben el mismo tipo de dato que ya les he dado en otras tablas y que no creo que tenga que volver a explicar, salvo dni, que aunque ya apareció en otra tabla anterior, recibe el tipo CHAR(9) por su longitud fija.

También tuve que corregir la relación con la tabla localidad, pues en el diagrama la había relacionado mediante el cp_completo (al que ahora que me doy cuenta tengo que cambiar nombre) de forma incorrecta con el campo cp de la tabla localidad.




Arreglado.

Viernes 20 de diciembre de 2024




Varias cosas a comentar aquí:

Arreglo la falta de línea de relación con la tabla sedes, que no la había puesto, con su cardinalidad correspondiente.

Los tipos de datos de las FK son del mismo tipo que la tabla a la que referencian, SMALLINT e INT.

Num_ss es un campo con una longitud fija, de ahí su tipo CHAR, la longitud no estoy seguro de que sea esa, pero creo que lo más importante es dar su tipo correcto.

Num_matricula: tipo INT, para que pueda ser auto_increment y generarse de forma automática al dar de alta a un nuevo empleado.

El resto de atributos, la gran mayoría, ya están explicados en tablas anteriores. Especialidad, que no había aparecido antes, es un campo descriptivo de la especialidad del trabajador, de ahí su tipo, VARCHAR.


Esta tabla bastante sencillita es la de equipos. Sus dos atributos son SMALLINT. La PK para que pueda ser autoincremental, y el num_tecnicos dado que va a almacenar la cantidad de técnicos participantes. También podría ser CHAR(2) por ser un atributo descriptivo más que para hacer operaciones con el.


Esta es la última tabla totalmente independiente del modelo.

En esta todos los atributos son del tipo VARCHAR, pues son descriptivos y pueden tener una longitud variable.

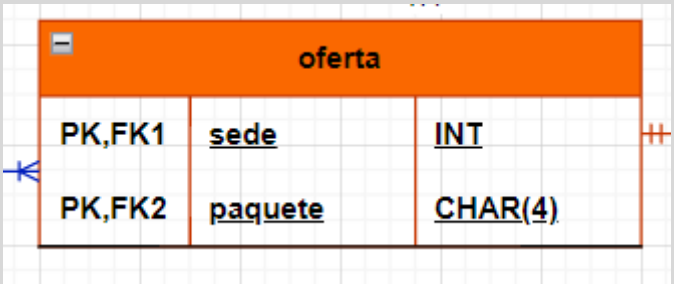
Telefono sigue el mismo patrón que en casos anteriores, con su longitud fija de 9, y será un tipo CHAR.

Cod_agencia en este caso, no la he tenido en cuenta como INT o SMALLINT para que sea auto_increment, primero porque puede darse el caso de que haya más de una agencia en la misma provincia y necesite caracteres alfanuméricos para poder identificarlo así, y segundo, al no ser una entidad que controle nuestra empresa, no necesito específicamente que sea un dato correlativo y consecutivo.

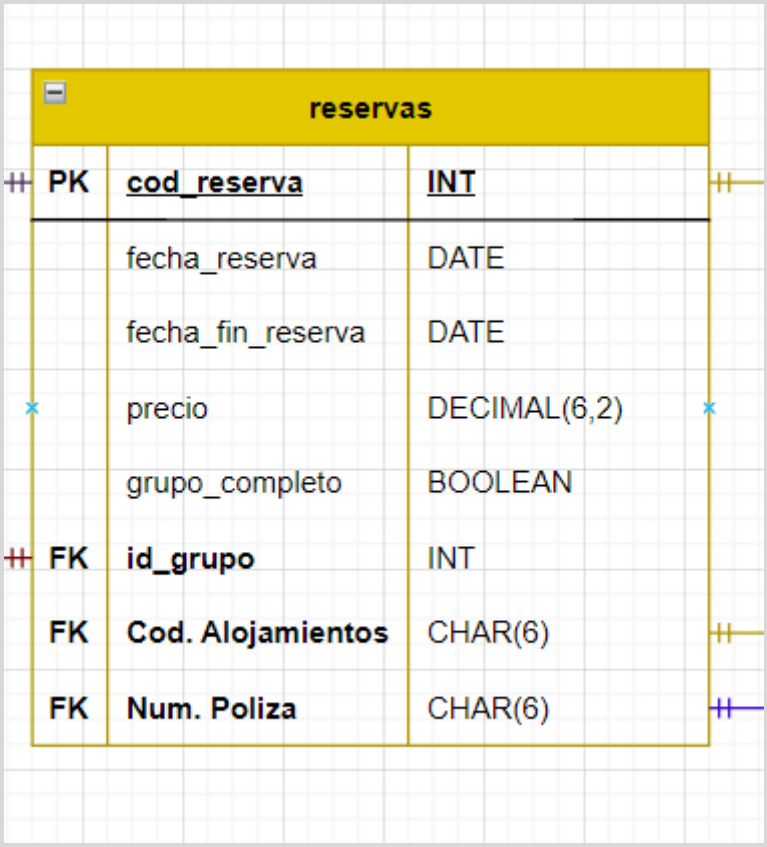
Estas son las consideraciones que he tomado para esta tabla.

## COMPLETANDO EL DIAGRAMA

A partir de ahora, todas las tablas que me quedan por completar son tablas dependientes y que reciben una o varias FK. Por lo tanto como van a existir muchos atributos repetidos y que ya he explicado, iré colocando capturas de cada una de ellas con explicaciones más ligeras y escuetas.




Recibe dos FK, que forman una PK compuesta. Sus tipos coinciden con los de los campos a los que hacen referencia.

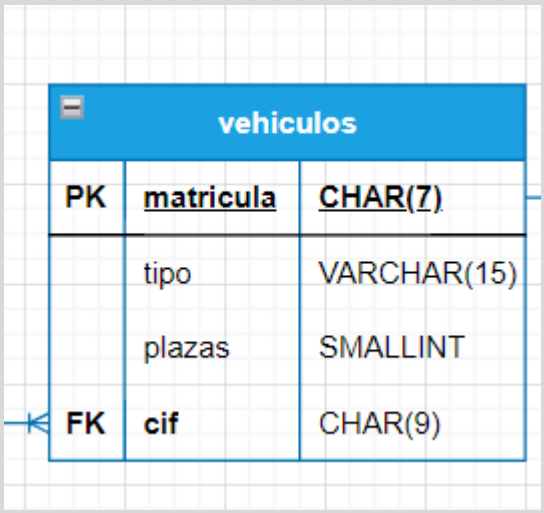



El cod_reserva, un int para que pueda ser auto_increment y secuencial.

Las fechas como tipo DATE.

Precio como en otro ejemplo, un Decimal (6,2).

Las FK como el tipo al que referencian.

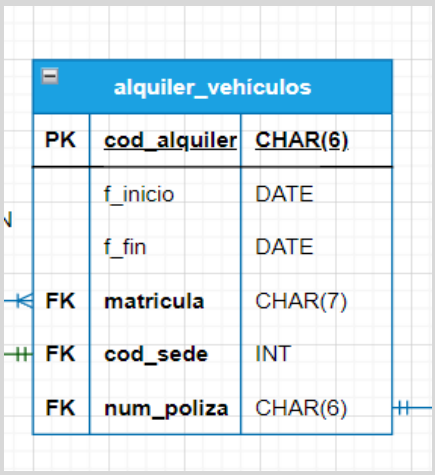



Matricula como CHAR(7) ya que creo que es la longitud completa de la matrícula.

Tipo un varchar, ya que es una descripción breve.

Plazas como SMALLINT, más por la posibilidad de que en algún momento se pueda hacer un cálculo, aunque bien podría ser un CHAR.

Cif, el mismo tipo declarado en la tabla origen.



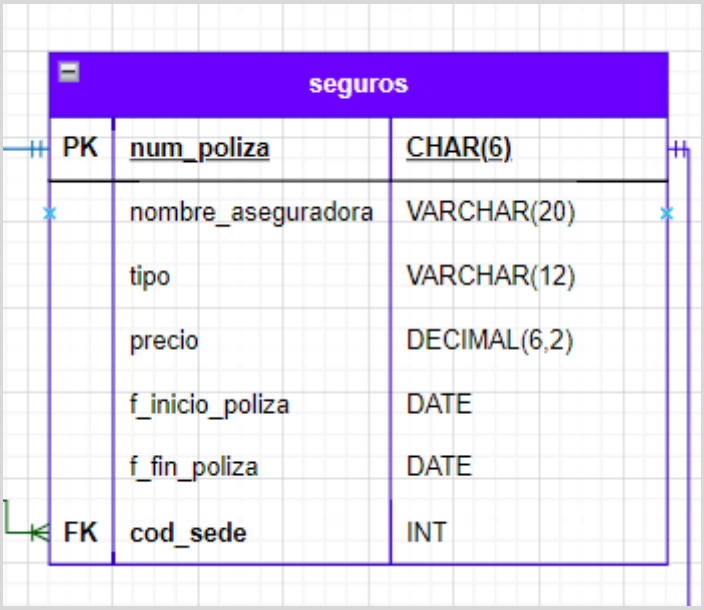

Cod_alquiler como CHAR(6), ya he explicado en otras partes de este ejercicio porque en casos parecido a este elijo un char.

Las fechas de tipo DATE. Nunca DATETIME, ya que no creo que las horas sean relevantes.

Las FK de su tipo original.


Tipo de los atributos según atributos a los que referencia.

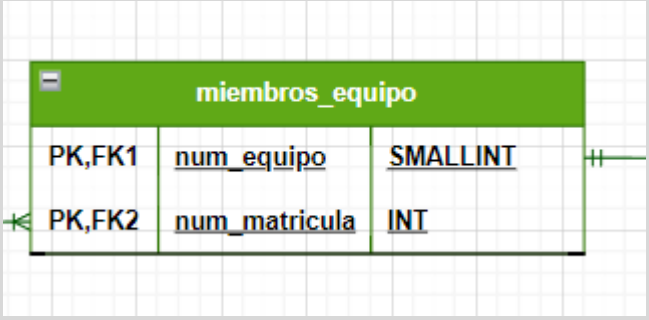

Mismo tipo de tabla que el caso anterior, así que los tipos de datos serán aquellos a los que referencian los atributos de la tabla.



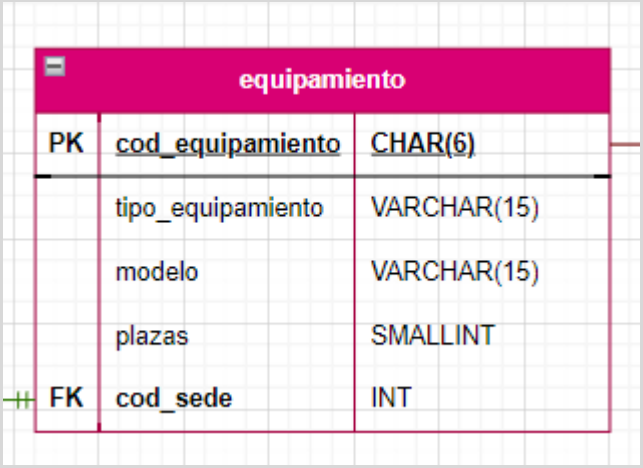

Nada muy distinto a lo visto anteriormente.

Num_poliza como CHAR(6), las fechas tipo DATE, los campos que son descriptivos son VARCHAR, de longitudes estimadas, no se muy bien lo que podrían requerir ese tipo de datos a ciencia cierta, y el precio, como en otros casos DECIMAL(6,2).

Viernes 20 de diciembre de 2024




Otra tabla intermedia cuyos atributos son PK de otras tablas.

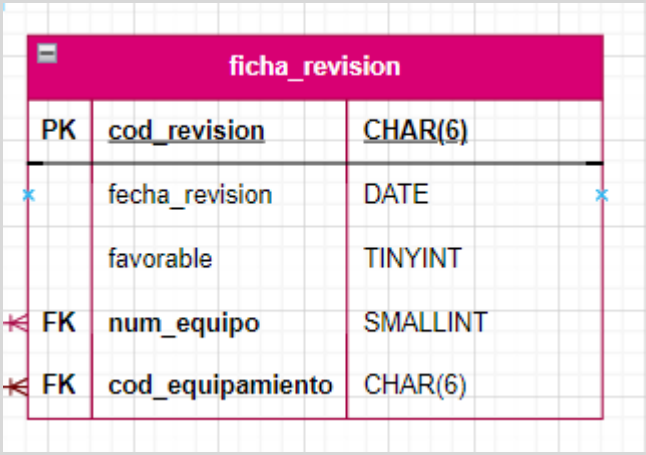



La PK, como en otros casos que creo una PK que no sea auto_increment, quizá necesite algún carácter que no sea numérico para identificarlo mejor.

Los descriptivos, como VARCHAR.

Plazas, como en el caso de los vehículos de alquiler, un SMALLINT, salvo que este puede recibir un valor NULL.

La FK del mismo tipo origen.




En este caso:

Sigo el mismo patrón de tipo de dato para la PK.

La fecha es de tipo DATE.

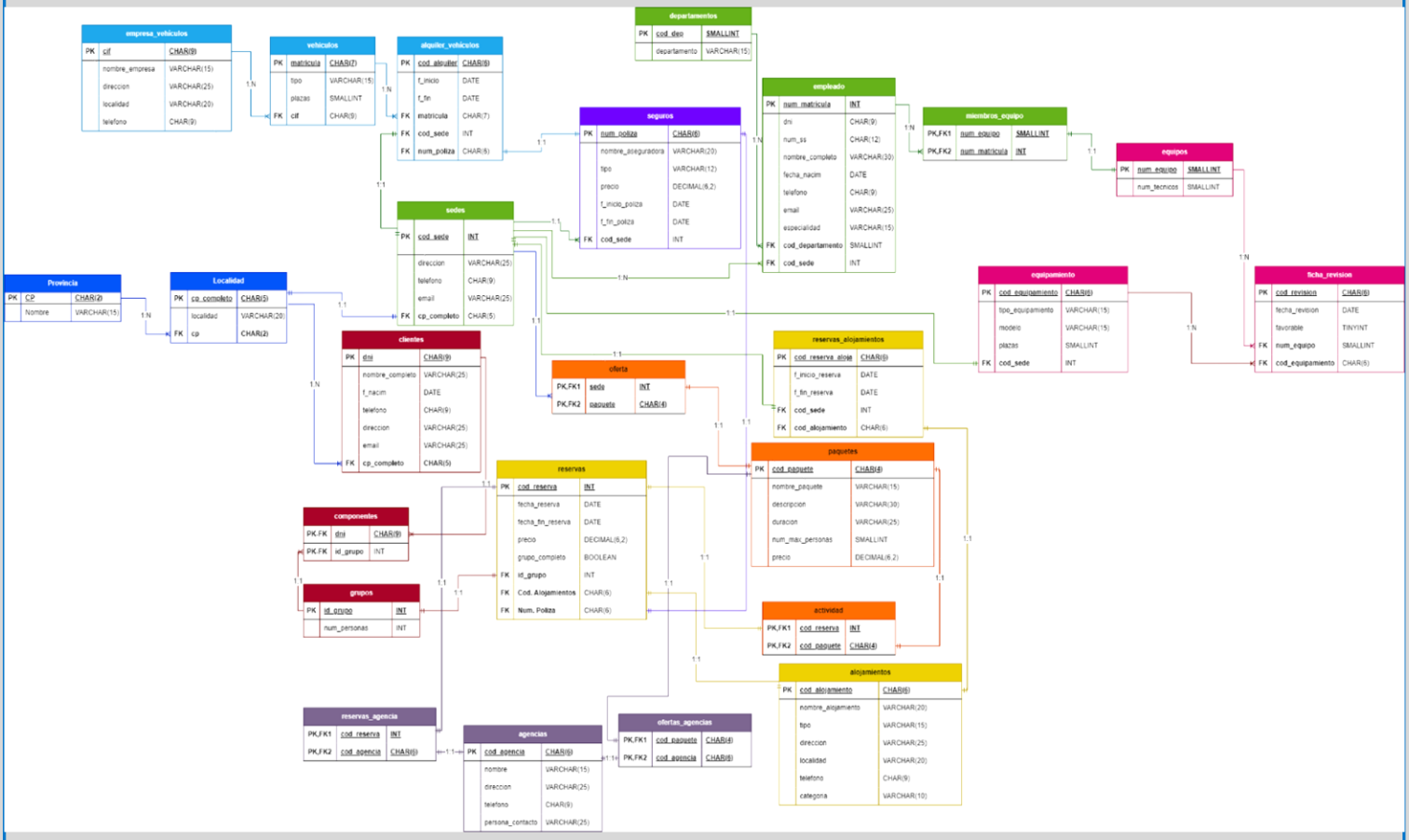
Favorable, que en inicio iba a ser un campo BOOLEAN, el programa me lo creo como TINYINT, no se si será porque en el otro campo BOOLEAN del ejercicio le puse en el script un valor por defecto.

Las FK del tipo original.

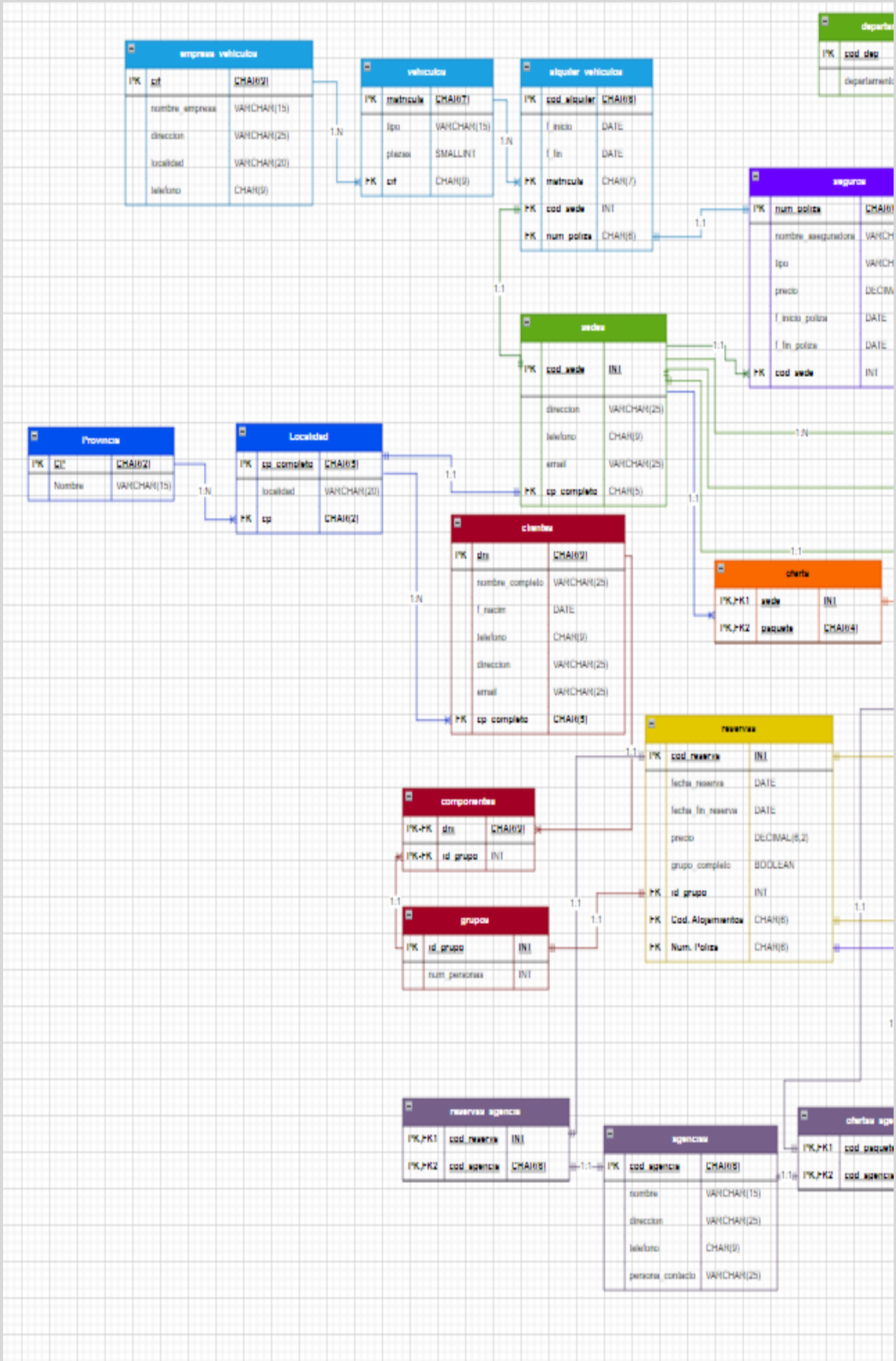

Tabla intermedia que sólo recibe FK como atributos, por lo tanto sus tipos ya están definidos.

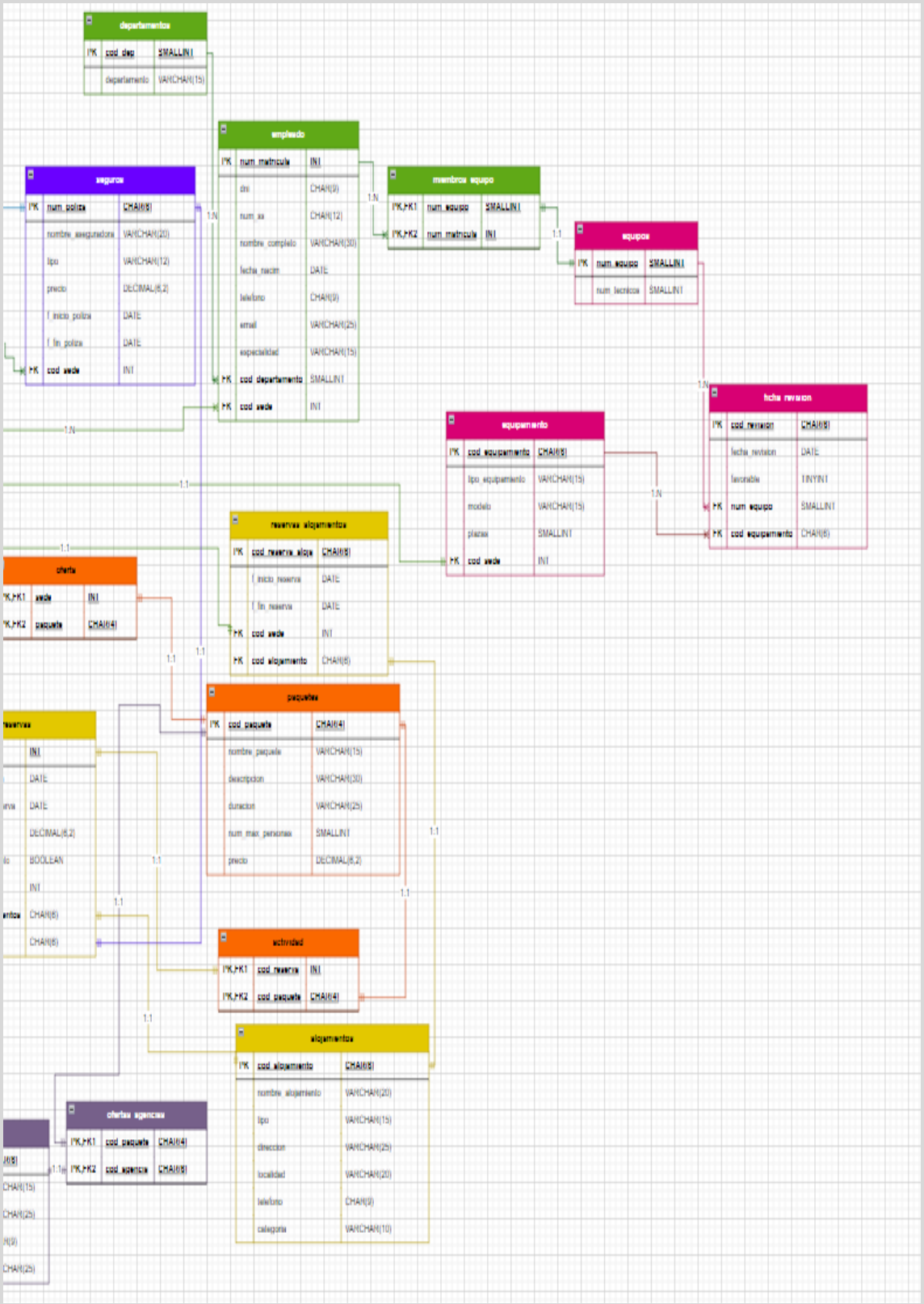

Ya por último. Una PK como las anteriores que he creado. Las fechas de tipo DATE y las FK del tipo original.

# DIAGRAMA FINAL




Espero que la calidad sea la suficiente. Así es como queda el diagrama al final de todo. Creo que es bastante más legible que el anterior, a pesar de llevar incluidas más cosas. Quizá tal y como está la imagen alguna cosa queda un poco borrosa, pero en principio creo que bien. Lo añadiré ahora en dos partes para que se vea mejor.


empresa_vehiculos
cif CHAR(9)
nombre_empresa VARCHAR(15)
direccion VARCHAR(25)
localidad VARCHAR(20)
telefono CHAR(9)
vehiculos
matricula CHAR(7)
tipo VARCHAR(15)
plazas SMALLINT
cif CHAR(9)
alquiler_vehiculos
cod_alquiler CHAR(8)
f_inicio DATE
f_fin DATE
matricula CHAR(7)
cod_aede INT
num_poliza CHAR(8)
seguros
num_poliza CHAR(8)
nombre_aseguradora VARCHAR
tipo VARCHAR
precio DECIMAL
f_inicio_poliza DATE
f_fin_poliza DATE
cod_aede INT
aedes
cod_aede INT
direccion VARCHAR(25)
telefono CHAR(9)
email VARCHAR(25)
cp_completo CHAR(5)
Provincia
CP CHAR(2)
Nombre VARCHAR(15)
Localidad
cp_completo CHAR(5)
localidad VARCHAR(20)
cp CHAR(2)
clientes
dni CHAR(9)
nombre_completo VARCHAR(25)
f_nacim DATE
telefono CHAR(9)
direccion VARCHAR(25)
email VARCHAR(25)
cp_completo CHAR(5)
ofertas
cod_aede INT
paquete CHAR(4)
reservas
cod_reserva INT
fecha_reserva DATE
fecha_fin_reserva DATE
precio DECIMAL(6,2)
grupo_completo BOOLEAN
id_grupo INT
Cod_Alojamiento CHAR(8)
Num_Poliza CHAR(8)
componentes
dni CHAR(9)
id_grupo INT
grupos
id_grupo INT
num_personas INT
reservas_agencias
cod_reserva INT
cod_agencia CHAR(5)
agencias
cod_agencia CHAR(5)
nombre VARCHAR(15)
direccion VARCHAR(25)
telefono CHAR(9)
personas_contacto VARCHAR(25)




Espero que así en dos partes se vea mejor.