Alexis Ruiz Alcalà 174746
Roger Ventura Castelló 173823

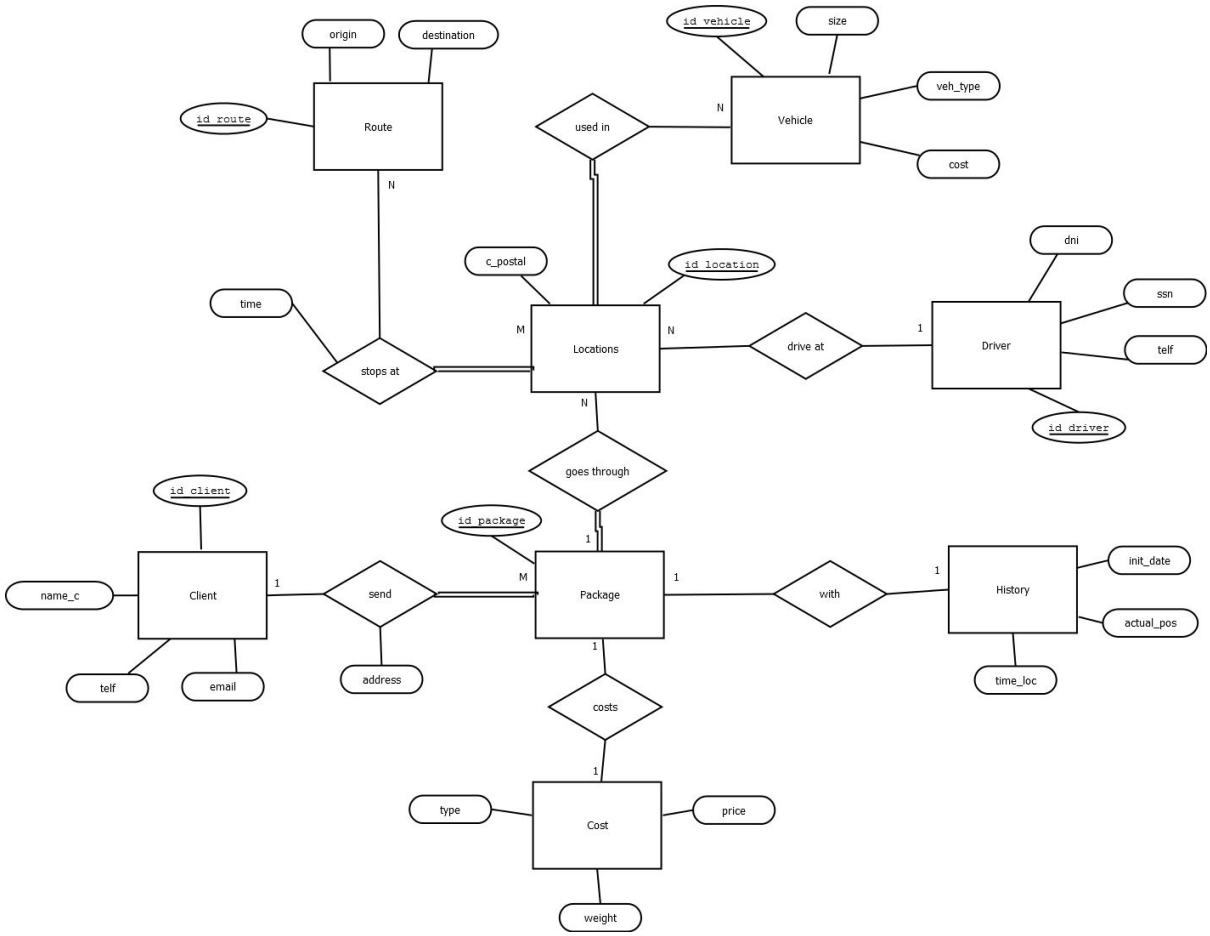
Esmeralda Ruiz
Diseño de Base de Datos

DISEÑO DE BASES DE DATOS

Question 1: Explanation of your database

Nuestra base de datos está orientada a tener controlada la red de una empresa que se dedica al transporte urgente de paquetes. Para esto hemos tenido en cuenta las cuatro partes principales que comentaremos a continuación. Las partes indicadas son las referentes a la relación existente entre los paquetes, los clientes, las localizaciones y el quién y con qué vehículo se encarga de llevar estos paquetes. Por lo tanto, la situación que está describiendo nuestra base de datos es todo el proceso que se lleva a cabo desde que el cliente envía al paquete hasta que llega a su destino.
El proceso del que estamos hablando incluye el que un cliente envíe una paquete el cual tiene un coste y un estado (enviado, recibido, en espera, etc.). El paquete pasa por distintas localizaciones las cuales deben ir registradas. Paralelamente a esto, la compañía tiene asignadas una serie de rutas compuestas por localizaciones y durante las rutas, los conductores o vehículos que transporten la mercancía pueden ir variando. A partir de todo esto ya podemos empezar a diseñar el modelo de Entidad-Relación con el cual elaboraremos la base datos aplicada a la situación dada.

Question 2: Model Entitat-Relació (ER). Explain all the tables created



A continuación, pasaremos a explicar nuestro modelo E-R paso por paso y en preguntas posteriores comprobaremos cómo este tiene algunos errores y cosas a mejorar. En primer lugar, tenemos la tabla *Client* que tiene como clave primaria **id_client** para identificar a este de forma única. A la vez, tenemos que existe una relación binaria entre *Client* y *Package*. Esta relación es 1-N bajo el planteamiento de que un cliente puede enviar N paquetes y un paquete puede ser enviado por un cliente, lo que marca una relación de participación total por parte del paquete. En cuanto a la tabla paquete, se identifica únicamente definiendo como clave primaria **id_package** y **id_client**. Los paquetes tienen un coste, de ahí esa relación 1-1 entre paquete y coste (planteamiento erróneo). Además, cada paquete tiene un histórico ya que queremos saber por dónde pasa, este planteamiento está mal pero lo corregiremos en el diseño final. En cuanto a las tablas en el código, hemos incluido el histórico y el coste como atributos dentro de paquete pensando que no era necesario el crear dos tablas más.

Cada paquete pasa por N paradas con lo cual hemos establecido una relación directa entre *Package* y *Locations* a través de una relación binaria llamada *GoesThrough*. Esta se identifica de forma única teniendo como clave primaria **id_package** e **id_location** con el fin de identificar por que paradas va pasando el paquete. La intención es la correcta pero está mal planteado en la práctica ya que en una localización pueden haber distintas paradas. La tabla *Locations* es donde se establece la relación con conductor, vehículo y a qué rutas pertenece una localización dada. Hemos establecido **id_location** como clave primaria con la intención de hacer única cada ubicación cosa que también es mejorable y que justificaremos el cambio hecho en el diseño final. Cada ruta tiene M *Locations* y cada *Location* pertenece a N rutas de ahí la relación y el poder extraer en qué tiempo está calculado que dada una ruta sepamos a qué hora para en una de las localizaciones. Cada ruta está identificada por un identificador que en tablas se define como un **id_route.**

La tabla correspondiente a los vehículos viene dada por **id_vehicle** como clave primaria y para el conductor tenemos un **id_driver.** En el diseño hemos planteado que un vehículo puede ser usado en M *Locations* y que por cada *Location* pueden haber N *Vehicles*. Establecemos una participación total de *Locations* con *Vehicles* ya que un vehículo debe estar en alguna localización (mal). Por la parte del conductor hemos decidido en el diseño que una location puede tener un conductor, lo que a la postre veremos que está mal ya que obligamos a que cada location solo tenga un conductor.
Para identificar estas relaciones de forma única, tenemos que la tabla used_in, que relaciona *Locations* con *Vehicles*, tiene como clave primaria el **id_vehicle** e **id_loc**. Y driver_at se identifica por el **id_driver** y el **id_loc**.

Question 3: Code c.sql

Adjunto en la carpeta

Question 4: Code e.sql

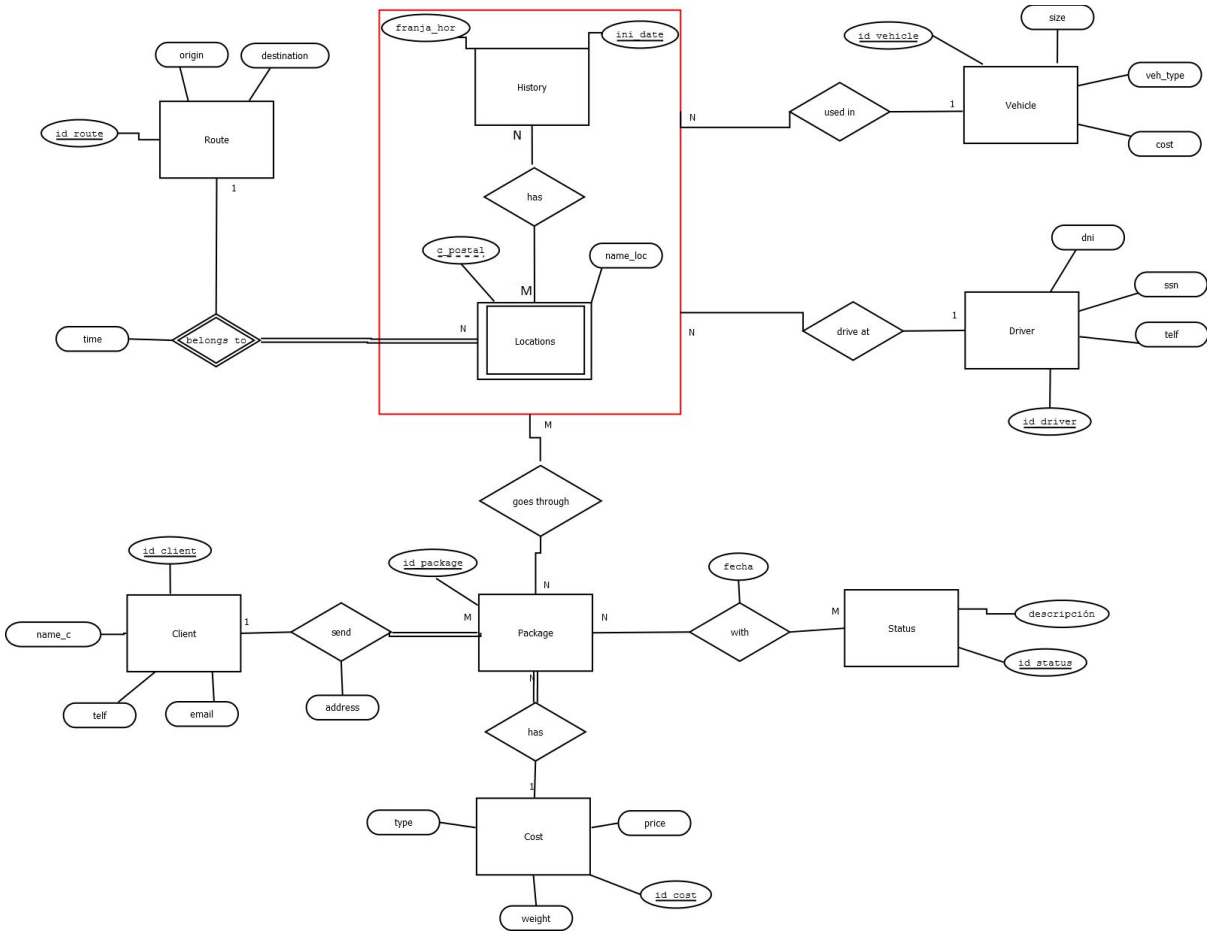
Adjunto en la carpeta

Question 5: Insert 5 tuples at most for each table.
Write the tuples in the report for each table including the insertions suggested in part one: verification design.

```
INSERT INTO Routes (id_route, origin, destination) VALUES
(1,'Madrid', 'Barcelona'),
(2,'Vic', 'Barcelona'),
(3,'Sevilla','Madrid'),
(4,'Barcelona','Granollers'),
(5,'Valencia','Barcelona');

INSERT INTO Locations (c_postal, id_route, name_loc, time_stops) VALUES
(08480,1,'Vallés', '10:30'),
(09234,3,'Toledo','12:15'),
(21032,1,'Reus','16:45'),
(48239,1,'Barcelona','13:23'),
(48239,2,'Barcelona','15:56'),
(48239,4,'Barcelona','10:33'),
(48239,5,'Barcelona','17:13'),
(21032,2,'Reus','17:45'),
(09234,5,'Toledo','08:15');

INSERT INTO Vehicle (id_vehicle, v_type, size, cost) VALUES
(1,'Turisme', 350, 300),
(2,'Moto',250,250),
(3,'Furgoneta',500, 450),
(4,'Camión',1500,600),
(5,'Furgoneta',500,450);

INSERT INTO Driver (id_driver, dni, ssn, telf) VALUES
(1,'934536339G','23526',938432913),
(2,'231938239T','42526',931824382),
(3,'919328312P','31412',621952129),
(4,'231414192F','20103',629304019),
(5,'059382394G','42819',934812394);

INSERT INTO LocHasHist (c_postal, ini_date, franja_hor, id_route, id_vehicle, id_driver) VALUES
(08480,'28/06/2017','Mañana',1,1,1),
(21032,'02/07/2017','Tarde',2,2,2),
(48239,'23/06/2017','Tarde',1,1,2),
(21032,'14/04/2017','Mañana',1,1,3),
(09234,'18/05/2017','Tarde',5,5,5);
```

```
INSERT INTO Clientt (id_client, name_c, telf, email) VALUES
(1,'Apple',923182392,'apple@gmail.com'),
(2,'Google',948322918,'google@hotmail.com'),
(3,'Amazon',934921039,'amazonas@gmail.com'),
(4,'Windows',958432938,'ventanas@gmail.com'),
(5,'Upf',948392839,'upf@upc.edu');

INSERT INTO Cost (id_cost, weight,price, type_p) VALUES
(1,50,800,'Caja'),
(2,2,300,'Sobre'),
(3,600,1300,'Caja grande'),
(4,300,900,'Caja'),
(5,200,300,'Caja');

INSERT INTO PackageSendHas (id_package, id_client ,id_cost, adress) VALUES
(1,1,1,'Pollancre, 18'),
(2,2,2,'Riera, 25'),
(3,3,3,'Moscarola, 147'),
(4,4,4,'Enric Morera, 12'),
(5,5,5,'Tomas Bretón, 17');

INSERT INTO GoesThrough (id_package, c_postal, ini_date, franja_hor, id_route) VALUES
(1,08480,'28/06/2017','Mañana',1),
(2,21032,'02/07/2017','Tarde',2),
(1,48239,'23/06/2017','Tarde',1),
(1,21032,'14/04/2017','Mañana',1),
(5,09234,'18/05/2017','Tarde',5);

INSERT INTO Statuss (id_status, description) VALUES
(1,'In process'),
(2,'enviado'),
(3,'In process'),
(4,'a la espera'),
(5,'recibido');

INSERT INTO With_stat (id_status, id_package, fecha) VALUES
(1,1,'14/06'),
(2,2,'24/12'),
(3,3,'02/11'),
(4,4,'30/08'),
(5,5,'12/02');
```

Question 6: Final model Entitat-Relació (ER)



En esta pregunta pasamos a analizar los cambios realizados para obtener el diseño final y que reglas hemos utilizado para eliminar la redundancia. Existe algún caso como veremos en el que permitimos algo de redundancia la cual está justificada.

En primer lugar, la entidad *Client* no ha sufrido ningún tipo de cambio ni en su tabla ni en sus relaciones. En el resto del diseño hay muchas cosas que se han visto modificadas para darle sentido al diseño y a la base de datos. La relación entre *Package* y *Cost* tenía mal la cardinalidad, debido a que un paquete tiene un coste pero el mismo lo pueden tener N paquetes. Existe una relación fuerte por parte de *Package* ya que este debe tener un coste obligatoriamente. Al tener *Package* una relación de 1-N con *Client* y con *Cost* hemos podido unificar las dos relaciones (send y has) con *Package* en una misma tabla. Cada elemento en la tabla viene identificado de forma única en cada tupla por una clave primaria compuesta del **id_package**, **id_cost** e **id_client**, de tal forma que un paquete pertenece a un cliente, tiene un coste dependiendo del tipo de paquete y un **id_package** para identificar el paquete.

La clave **id_cost** es la clave primaria de *Cost* en la cual tenemos los distintos precios dependiendo del tipo de paquete. Además, algo que teníamos mal era asociar el paquete a

un historial ya que si poníamos una fecha no podíamos poner otro atributo, por lo tanto todos los paquetes con la misma fecha de inicio tendrían el mismo estado lo cual sería redundante y estaría mal. Por ejemplo, si forzamos a que todos tengan **actual_pos** estamos obligando a que tengan el mismo estado lo cual es erróneo.

La solución reside en que en paquete necesitamos el estado pero no el histórico que va asociado en otra parte. Por lo tanto, sustituimos el histórico por *Status* que viene dado por un **id_status** como clave primaria y que a través de su relación con paquete podemos extraer información acerca de en qué fecha estaba en ese estado. *Status* a parte de un identificador contiene una descripción. La relación de que tiene con *Package* es N-M ya que un paquete puede pasar por M estados y un estado puede pertenecer a N paquetes. Esta relación viene dada en una tabla llamada *With_stat* la cual está identificada por el **id_status** y el **id_package**. Hay una relación clara entre los paquetes y por qué zonas pasa cosa que uniremos en unos instantes, pero primero vamos a tratar la otra parte grande dentro de este diseño.

Después de realizar un análisis hemos visto que lo más oportuno es que sea cada Location la que tenga un *Historic*. Para esto es necesario distinguir las diferentes paradas dentro de una misma localización a partir de hacer a *Location* entidad débil y una relación débil con *Route*, de tal forma que para identificar las diferentes paradas dentro de la misma localización haremos que la clave primaria de Locations sea el **c_postal** más el **id_route** que es el identificador de la entidad Owner. Con esto evitamos la redundancia que se podía producir al tener un **id_location** como clave primaria de *Locations*.

En el diseño anterior estábamos diciendo que una localización tenía solo un conductor, y definiendo eso relacionabamos un conductor a una parada y una fecha lo cual está mal. Debido a esto hemos visto que lo más conveniente era hacer una agregación entre *Historic* y *Locations* de tal forma que evitamos redundancia ya que tenemos un conductor concreto asociado a una parada de una localización en una franja horaria concreta. La misma situación se produce con los vehículos.

Traducido a las tablas tenemos una tabla llamada *LocHasHist* que hace referencia a la agregación entre *Historic* y *Location* las cuales hemos unido en una misma tabla. Se identifican de forma unitaria a partir de una clave primaria que está compuesta por **c_postal**, **ini_date**, **franja_hor** e **id_route**. Cada tupla de esta tabla que es una parada en una localización en una franja horaria que tiene su conductor y su vehículo en concreto los cuales son foreign keys que hacen referencia a las tablas vehicle y driver.

Para acabar con la explicación de nuestro diseño pasamos a la unión entre las dos grandes partes de nuestra base de datos. Los paquetes enviados siguen una ruta a partir de una fecha de salida. Además estos paquetes pasan por distintas localizaciones. Con todo esto hemos relacionado *Package* con la agregación entre *Locations* e *Historic* comentado anteriormente. Esto nos permite saber toda la información de nuestro paquete, desde por dónde pasa, a qué horas e incluso quién lo transportaba. La relación se describe en una tabla que contiene como clave primaria **id_package**, **c_postal**, **ini_date**, **franja_hor** e **id_route**.

Question 7: Explain the implemented views and show what they return Explain why the views are used within the real situation of your database and write the tuples returned for one o two examples.

* View that shows all the clients and packages whose current status is 'In process' so that the company can process these packages

Esta view nos devuelve todos los paquetes con su respectivo cliente los cuales están en proceso de envío.

Puede ser útil para el cliente, para saber si su paquete está en proceso o no.

En nuestro caso, nos retorna la siguiente tabla:

id_client	id_package
1	1
3	3

* View that shows for every stop the route and number of packages that we have with the sum of the weights

Esta view nos devuelve de cada parada la ruta, el número de paquetes que pasan por esa parada y la suma de los pesos de estos.

Puede ser útil para que los conductores sepan cuántos paquetes hay y cuánto pesan en total.

En nuestro caso, nos retorna la siguiente tabla:

c_postal	id_route	count(G.id_package)	sum(C.weight)
8480	1	1	50
9234	5	1	200
21032	1	2	52
48239	1	1	50

Question 8: Explain the implemented functions and procedures and show what they return Explain why they are used within the real situation of your database and write the tuples returned for one o two examples.

Procedures:

* Given an id of a package, obtain all the stops whose package go through with the corresponding driver and his/her telephone number.

Este procedimiento tiene como entrada un paquete, y nos devuelve una lista con los códigos postales en donde para, que conductor lo lleva y el número del conductor.

Puede ser útil en una situación donde el paquete no llegue a su destino, gracias a este procedimiento podremos saber por qué paradas puede estar o podremos llamar al conductor para saber qué ha pasado.

En nuestro caso llamamos con la entrada de id_packet = 1, y nos retorna esta tabla:

id_package	c_postal	id_driver	telf
1	8480	1	938432913
1	21032	3	621952129
1	48239	2	931824382

* In a specific stop, what packages are being carried in a vehicle.

Este procedimiento tiene como entrada un código postal, y nos devuelve una lista de todos los paquetes que han sido recogidos por algún vehículo en esa parada.

Puede ser útil en una situación donde quieras saber si el paquete ha sido cargado o no en un vehículo en esa parada, se ejecuta este procedimiento y lo sabrás.

En nuestro caso llamamos con la entrada de c_postal = 21032, y nos retorna esta tabla:

c_postal	id_package
21032	2
21032	1

* What packages are being carried in one specific vehicle and in one specific stop

Este procedimiento tiene como entrada un vehículo y una parada de tal manera que nos devuelve el paquete que ha sido recogido en esa parada y por ese vehículo.

Puede ser útil en una situación donde queramos saber los paquetes que ha recogido un vehículo en una misma parada.

En nuestro caso llamamos con la entrada id_vehicle = 1 y c_postale = 08480, y nos retorna esta tabla:

id_vehicle	c_postal	id_package
1	8480	1

Function:

* In one specific stop return what weight is being carried.

Esta función tiene como entrada un código postal y nos devuelve la suma de los pesos de todos los paquetes de esa parada.

Puede ser útil en una situación donde el conductor quiera saber el peso que hay que cargar en esa parada a su vehículo.

En nuestro caso llamamos con la entrada c_postal = 21032, y nos retorna la suma de los pesos de 2 paquetes 50 + 2, se puede apreciar en esta tabla:

WeightOnStop(21032)
52

Question 9: Explain the implemented trigger and show what it returns Explain why they are used within the real situation of your database and write the tuples returned for one o two examples.

* Assign one package to one vehicle checking that there is enough space

En el caso real, este trigger es útil ya que no podemos sobrecargar desde el sistema un vehículo. Por lo tanto, necesitamos controlar que esto no pase mediante un trigger que controle que el paso total de los paquetes no sobrepase la carga máxima del vehículo.

En esta imagen podemos ver cómo una vez creado el trigger, nos añade el primer insert de la tabla PackageSendHas el cual añade un nuevo paquete con el peso de 600, pero se nos salta en el segundo debido a que intentamos añadir el nuevo paquete creado, en una parada en la cual el vehiculo que tendra que cargar el paquete sera una moto que solo tiene una capacidad de 250, por lo tanto no podrá coger el paquete y se nos comunicara con el mensaje “It cannot add more packages, no such space in this vehicle” tal y como se puede observar:

✓	764	13:29:35	CREATE TRIGGER restrictionWeight BEFORE INSERT ON GoesThrough FOR EACH ROW BEGIN DECLARE wei...	0 row(s) affected
✓	765	13:29:35	INSERT INTO PackageSendHas (id_package, id_client ,id_cost, adress) VALUES (6,3,3,'Cirelora, 83'); INSERT INT...	1 row(s) affected
✗	766	13:29:36	INSERT INTO PackageSendHas (id_package, id_client ,id_cost, adress) VALUES (6,3,3,'Cirelora, 83'); INSERT INT...	Error Code: 1644 It cannot add more packages, no such space in this vehicle

Question 11: Code v.sql: Script containing the implemented views

Adjunto en la carpeta

Question 12: Code pf.sql: Script containing the procedures and functions

Adjunto en la carpeta

Question 13: Code del t.sql: Script of triggers

Adjunto en la carpeta