OBLIGATORIO – SQL

Ejercicio 1:

El Doctor Daniel Salinas, Ministro de Salud Pública, se ha enterado que usted se encuentra realizando un curso de análisis de datos con SQL y confía en que será capaz de ayudarlo modelando y diseñando su futura base de datos.

De una entrevista usted obtiene la siguiente información: en esta base de datos desea poder guardar información relativa a hospitales, por ende se requiere almacenar como mínimo la dirección, teléfonos, nombre de cada hospital y tipo, por ejemplo si es especializado.

Cada uno cuenta con varias salas por lo que se desea almacenar su código, nombre y cantidad de camas.

Cada hospital puede tener varias salas, y cada una de ellas pertenece a un solo hospital.

En distintos hospitales puede haber salas con el mismo código, pero esto no pude ocurrir dentro de un mismo hospital.

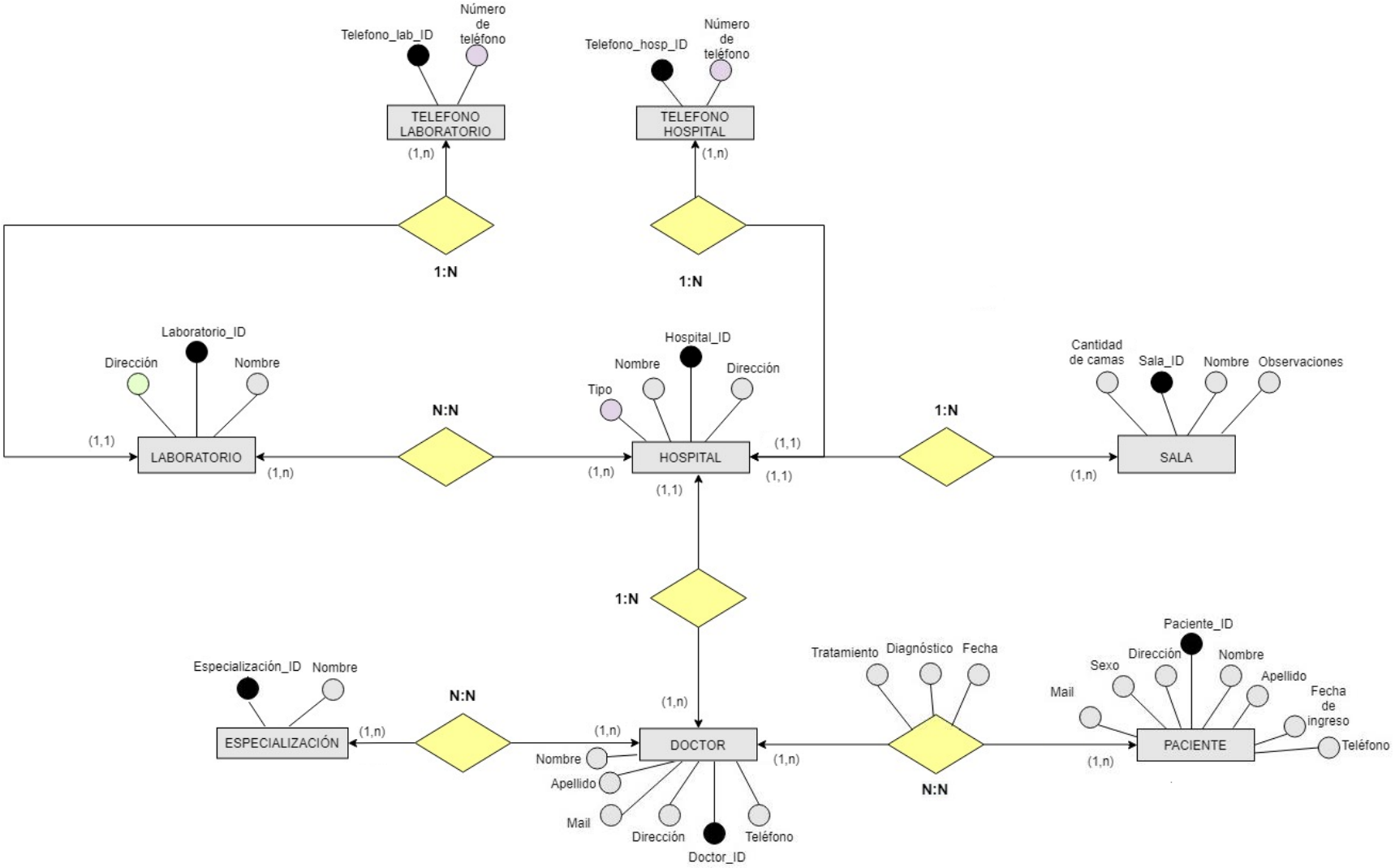
También se desea almacenar la información básica de cada médico y sus especializaciones. Debido a que los hospitales compiten entre sí por un premio de calidad, sus médicos son de dedicación integral.

Es de vital importancia contar con la información de los pacientes, además de los básicos, se desea tener un histórico de los diagnósticos del paciente, y fecha de los mismos, como así también, qué medico fue el responsable de cada diagnóstico.

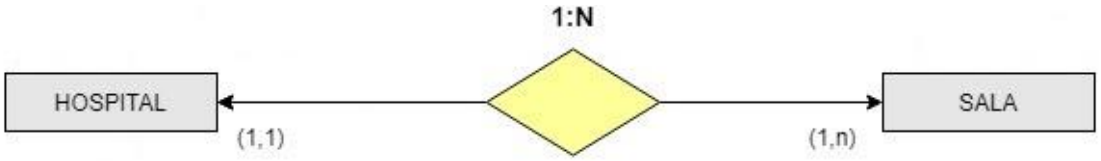
A su vez existen laboratorios que prestan servicios a los hospitales. Se requiere almacenar el nombre, dirección y teléfono de cada laboratorio. Los hospitales pueden trabajar con más de un laboratorio. En una primera instancia no se sabe cuántos teléfonos puede tener cada hospital ycada laboratorio, por lo que se requiere poder almacenar todos.

Se pide lo siguiente:

a) Diseñe el Modelo Entidad Relación que represente el escenario anterior.



b) Especifique las cardinalidades.

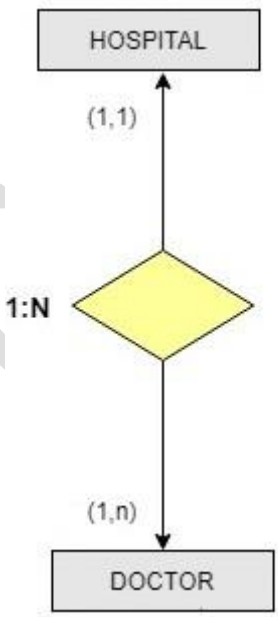


Cada hospital tiene varias salas y una sala pertenece a un solo hospital.

Como la entidad HOSPITAL tiene cardinalidad (1,1), la relación HOSPITAL-SALA no genera tabla, pero la PK de HOSPITAL pasa como FK en la tabla de la entidad SALA.

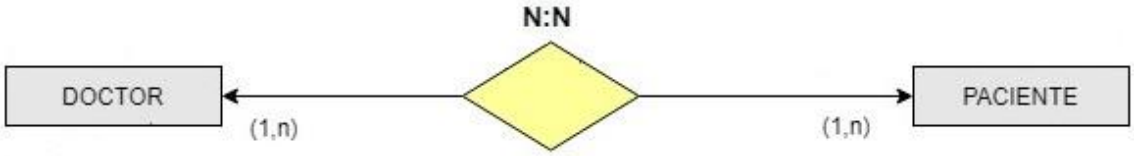
Se utiliza una PK compuesta cuando ningún campo cumple con la condición por sí solo, entonces como en distintos hospitales puede haber salas con el mismo código, pero no dentro de un mismo hospital.

La PK de esta entidad está formada por los campos Sala_ID y Hospital_ID.



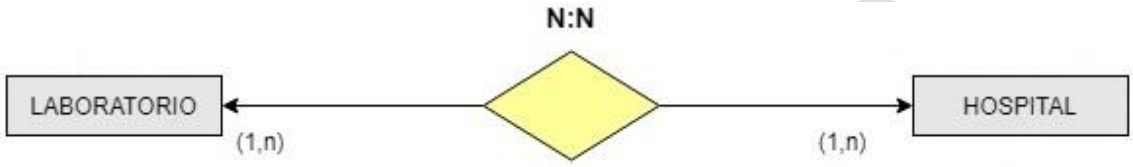
En un hospital trabajan varios doctores pero debido a que los hospitales compiten entre sí por un premio de calidad, sus médicos son de dedicación integral entonces cada doctor trabaja en un solo hospital.

Como la entidad HOSPITAL tiene cardinalidad (1,1), la relación HOSPITAL-DOCTOR no genera tabla pero la PK de HOSPITAL pasa como FK en la tabla de la entidad DOCTOR.



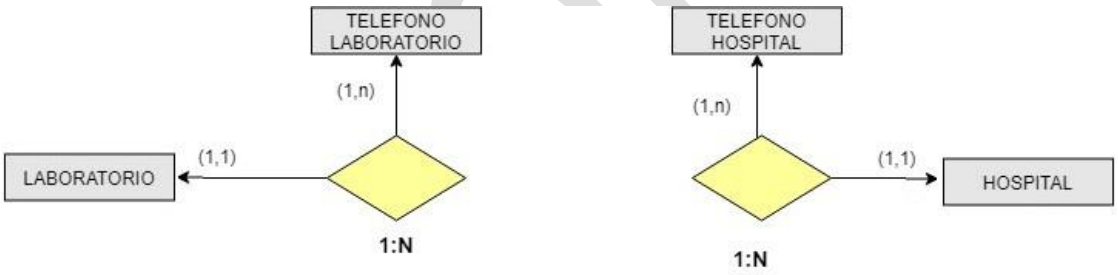
Un doctor revisa a varios pacientes y un paciente es revisado por varios doctores.

En este caso, la relación DOCTOR-PACIENTE genera tabla porque tiene cardinalidad N:N. La relación va a mantener sus atributos y además heredan las PK de ambas tablas.



Un laboratorio abastece a más de un hospital y un hospital puede trabajar con más de un laboratorio.

En este caso, la relación LABORATORIO-HOSPITAL genera tabla porque tiene cardinalidad N:N. La relación hereda las PK de ambas tablas.

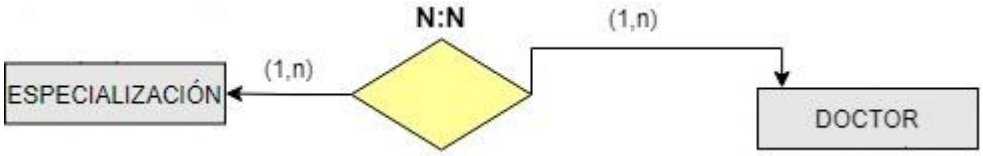


Como no se sabe cuántos teléfonos puede tener cada hospital y cada laboratorio, se requiere poder almacenar todos. Entonces un laboratorio posee varios teléfonos y un teléfono pertenece a un laboratorio. Un hospital posee varios teléfonos y un teléfono pertenece a un hospital.

Hacemos dos tablas de teléfonos, una para laboratorios y otra para hospitales y no una sola tabla de teléfonos de ambos para evitar tener muchos campos con nulos y así optimizar las consultas.

Como las entidades HOSPITAL y LABORATORIO tienen cardinalidad (1,1), la relación TELEFONO LABORATORIO-LABORATORIO y TELEFONO HOSPITAL-HOSPITAL no genera tabla

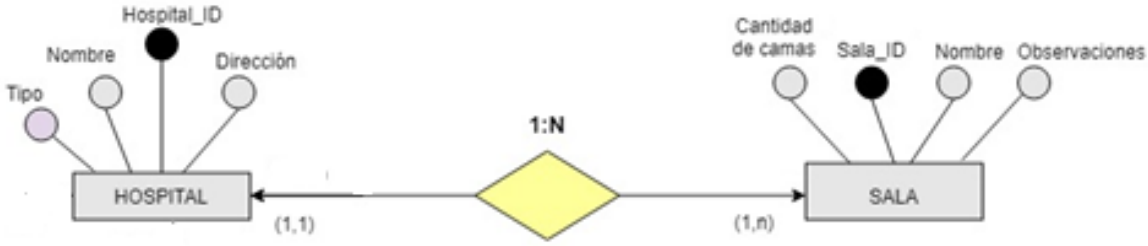
pero las PK de HOSPITAL pasa como FK en la tabla de la entidad TELEFONO HOSPITAL y la PK de LABORATORIO pasa como FK en la tabla de la entidad TELEFONO LABORATORIO.



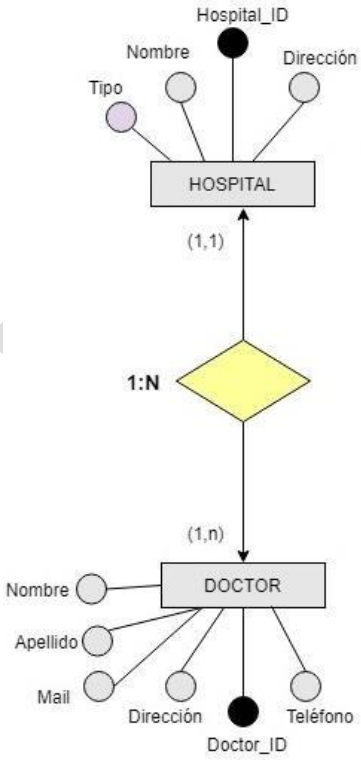
También se desea almacenar la información básica de cada médico y sus especializaciones, entonces cada doctor tiene varias especializaciones y una especialización pertenecen a varios doctores.

En este caso, la relación DOCTOR-ESPECIALIZACION genera tabla porque tiene cardinalidad N:N. La relación hereda las PK de ambas tablas.

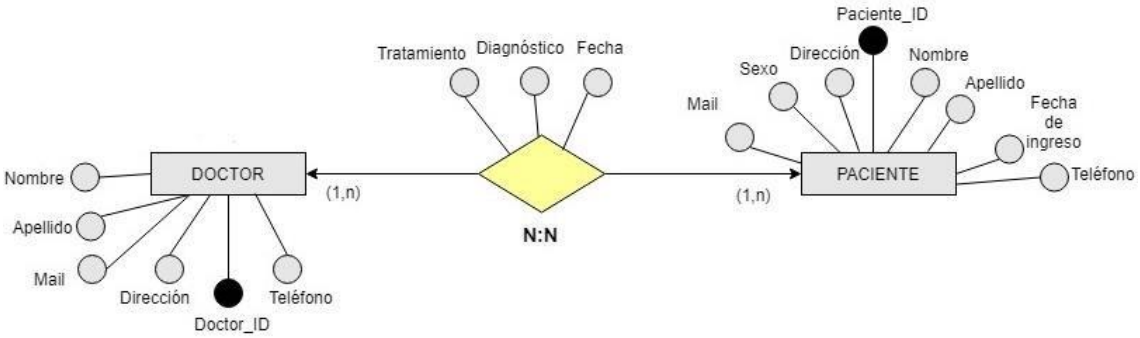
c) Defina al menos cuatro atributos para cada entidad.



El atributo **Observaciones** refiere a si la habitación es compartida o individual. El atributo **Tipo** significa si el hospital es especializado o no.

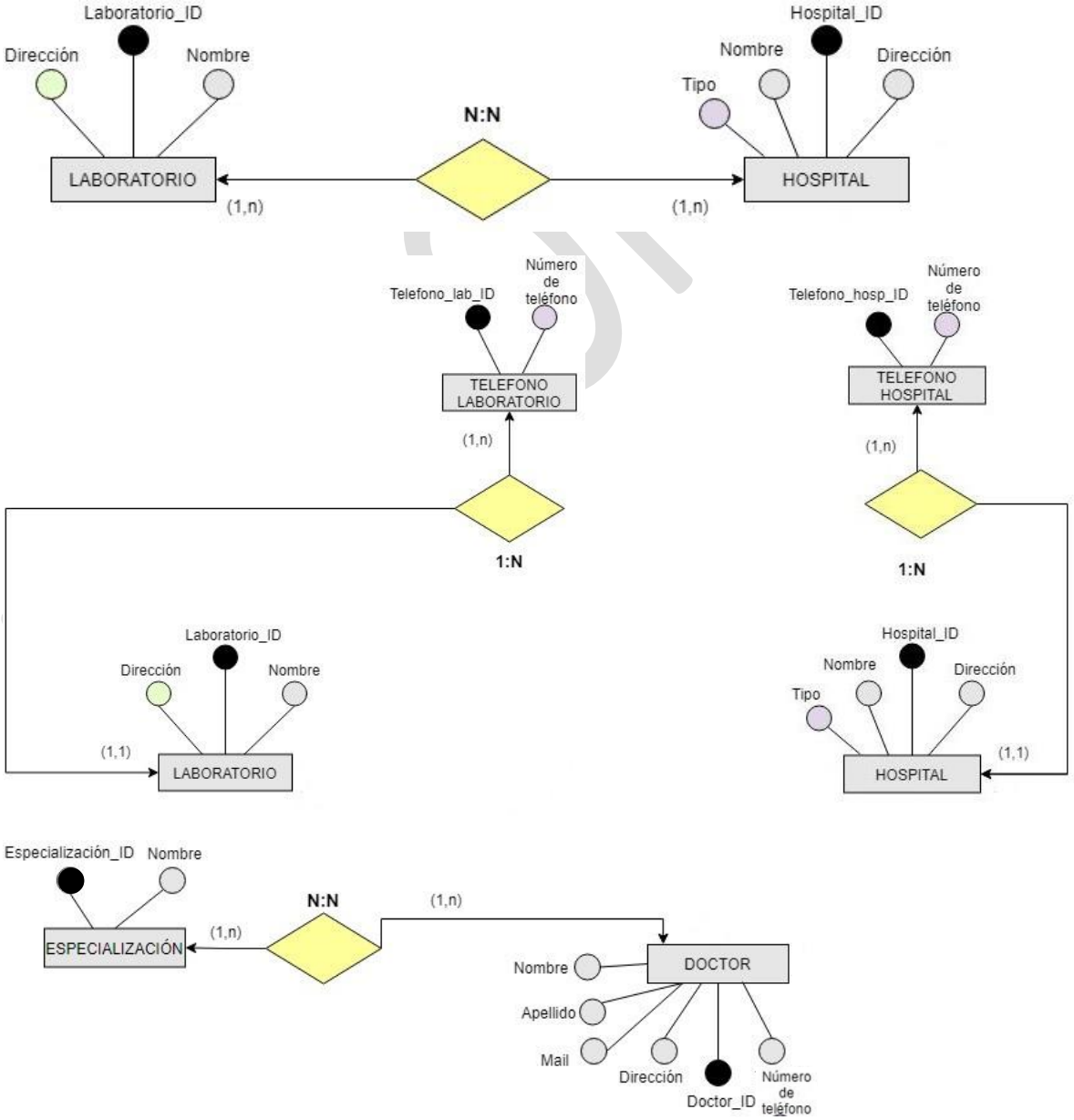


Agregamos el atributo **Mail** a la entidad DOCTOR.

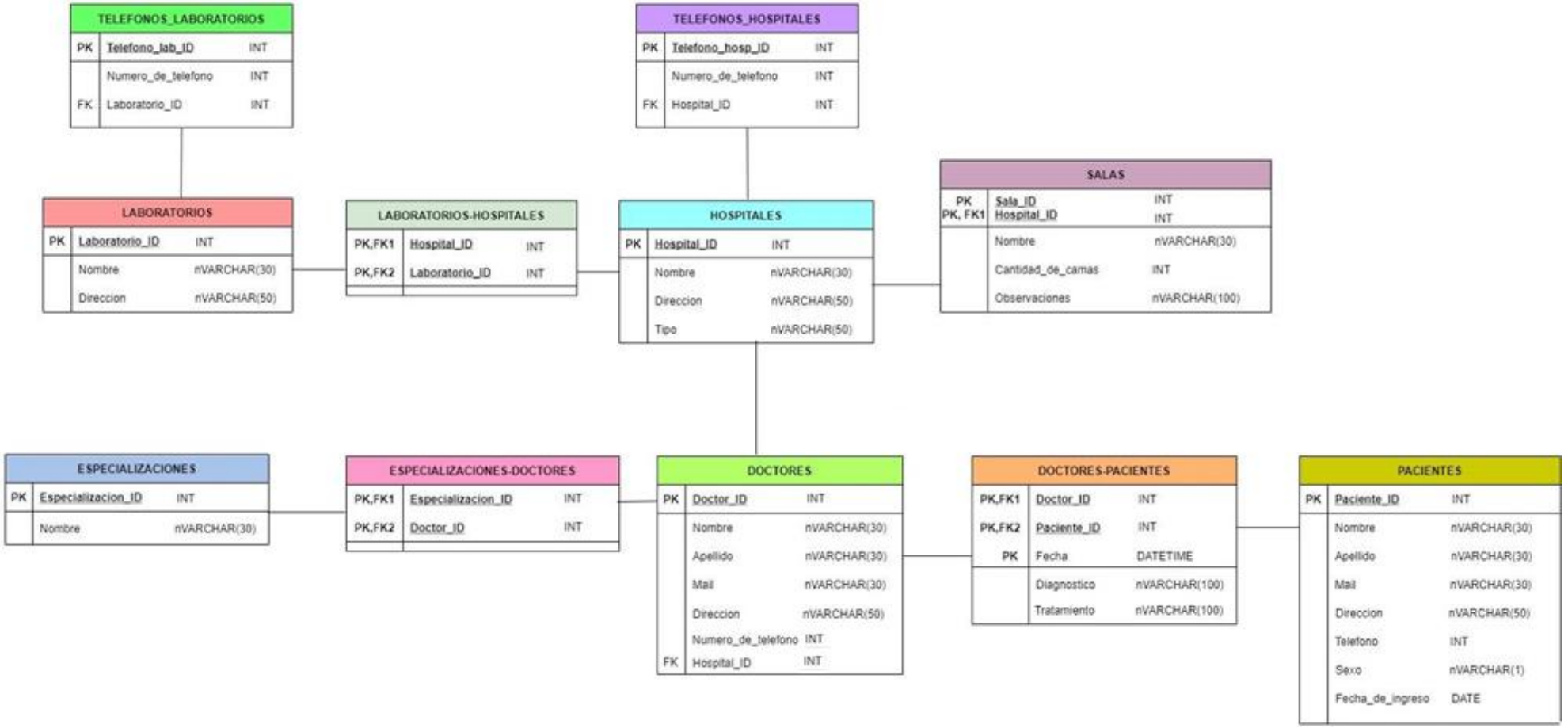


Para conocer más datos del paciente, agregamos los atributos: Mail, Sexo y Fecha de ingreso. Como es importante contar con el histórico de los diagnósticos del paciente y sus fechas, situamos dichos atributos en la relación DOCTOR-PACIENTE, además del atributo Tratamiento.

Los atributos Doctor_ID y Paciente_ID podrían sustituirse por “Cédula” ya que es un campo identificador que no va a repetirse.



d) Presente las tablas con sus campos, claves únicas, foráneas y tipos de datos que se desprende del Modelo Entidad Relación realizado en los puntos a, b y c.



Ejercicio 2:

1) Dada la siguiente consulta:

```
SELECT c.CompanyName, SUM(soh.TotalDue)
AS TotalFROM SalesLT[1].Customer AS c
LEFT JOIN SalesLT.SalesOrderHeader
AS sohON c.CustomerID =
soh.CustomerID
GROUP BY
c.CompanyName
ORDER BY Total
DESC
```

a) Explique conceptualmente qué retorna la misma (información).

La información que se obtiene con esta consulta es los nombres de las compañías de la totalidad de los clientes , hayan o no realizado compras y la suma de todas ellas, ordenados de mayor a menor.

b) Explique con qué intención se utilizó la cláusula LEFT JOIN.

En este caso se utilizó un LEFT JOIN porque queremos mostrar las compañías de todos los clientes existentes en la tabla **Customer**, se encuentren o no en la tabla **SalesOrderHeader**.

La tabla **Customer** muestra la información de los nombres de las compañías en los que los clientes compran y la tabla **SalesOrderHeader** indica la información de las órdenes de compra y además muestra el monto consumido por cada cliente.

Con LEFT JOIN damos prioridad a la tabla de la izquierda y buscamos en la tabla derecha.

Si no existe ninguna coincidencia para alguna de las filas de la tabla de la izquierda, se muestran de igual forma todos los resultados de la primera tabla.

La tabla **Customer** es la tabla LEFT (renombrada con el alias “c”), por lo que todas sus filas se mostrarán en los resultados.

La tabla **SalesOrderHeader** (renombrada con el alias “soh”) es la tabla de la derecha ya que aparece luego del LEFT JOIN, por lo tanto, si se encuentran coincidencias se mostrarán los valores correspondientes, pero sino aparecerá NULL en los resultados.

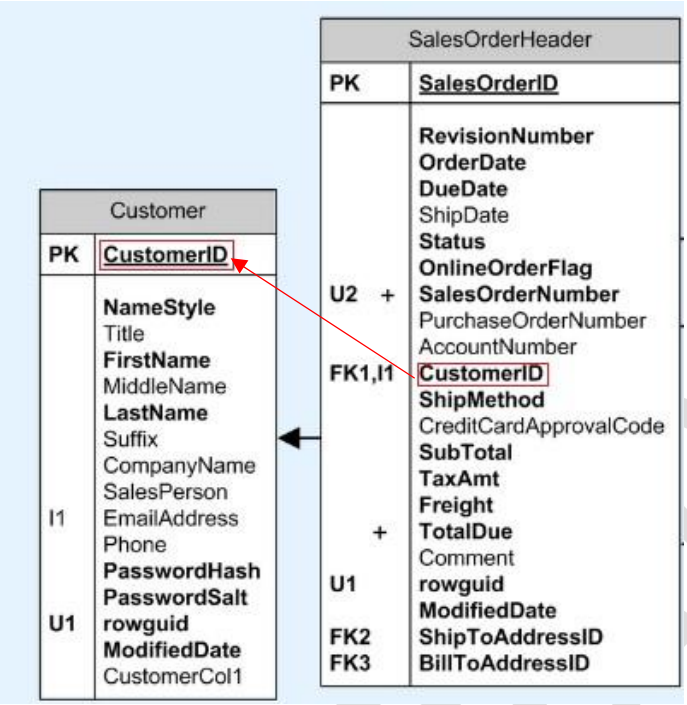
A su vez, el LEFT JOIN trae lo que hay en la intersección de esas dos tablas.

Las dos tablas se combinan especificando el campo en común, en este caso ese campo es la PK de la tabla **Customer** que a su vez es la FK en la tabla **SalesOrderHeader**.

c.CustomerID = soh.CustomerID

Esto nos dice que el campo CustomerID de la tabla Customer sea igual al campo CustomerID de la tabla SalesOrderHeader.

[1] SalesLT es el schema, el cual describe la estructura de una base de datos en un lenguaje soportado por un sistema de gestión de base de datos.



c) Explique por qué se utilizó la cláusula GROUP BY.

```
GROUP BY c.CompanyName
```

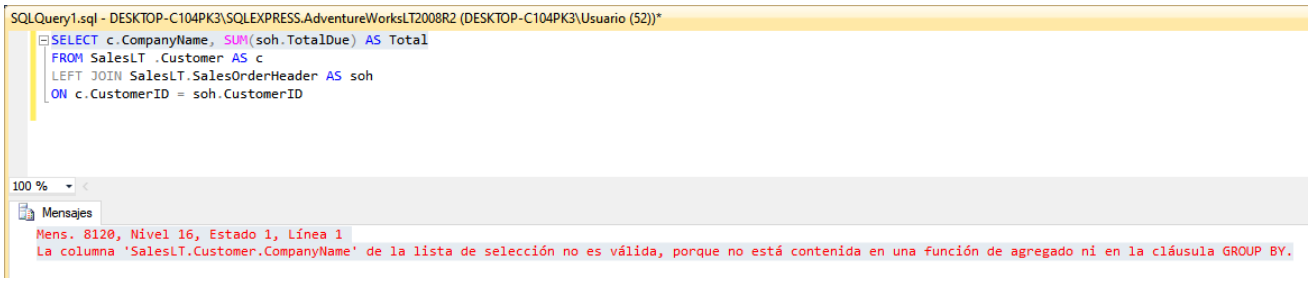
Se agrupa por el campo CompanyName para mostrar las compras totales por compañía. Es decir, va a sumar la cantidad de compras que haya en cada compañía.

Si ejecutamos la consulta siguiente, SQL nos va a mostrar el total de la suma de todas las compras de los clientes en todas las compañías, como un único valor.

```
SELECT SUM(soh.TotalDue) AS Total
FROM SalesLT1.Customer AS c

LEFT JOIN SalesLT.SalesOrderHeader AS soh
ON c.CustomerID = soh.CustomerID
```

Si dejamos la consulta sin el GROUP BY, a SQL le estamos diciendo que traiga el nombre de la compañía,por lo que van a ser varios registros, y por otro lado le decimos que traiga un registro que es la suma de todas las compras de los clientes, por lo cual va a haber un error.

```
SELECT c.CompanyName, SUM(soh.TotalDue)
AS Total

FROM SalesLT1.Customer AS c

LEFT JOIN SalesLT.SalesOrderHeader AS soh
```



Entonces, cada vez que en un SELECT tengamos una función de agregación (una suma en este caso), vamos a tener que usar una cláusula GROUP BY, pero podemos usar un GROUP BY sin tener una funciónde agregación. Se debe agrupar por el campo que no se encuentra en la función de agregación.

Para terminar, se ordenan los valores de los totales en forma descendente para visualizar las compañías de mayor gasto primero. Podemos hacer referencia al alias “Total” porque ORDER BY es lo último que se ejecuta.

```
ORDER BY Total DESC
```

Primero se va a ejecutar la cláusula FROM, luego el ON, luego el GROUP BY, luego el SELECT y por último el ORDER BY.

2) Dada la siguiente consulta:

```
SELECT count(c.CustomerID) AS 'Number of Customers', a.City
FROM SalesLT.Address AS a
INNER JOIN SalesLT.CustomerAddress AS ca
ON a.AddressID=ca.AddressID
INNER JOIN SalesLT.Customer AS c
ON c.CustomerID=ca.CustomerID
WHERE a.StateProvince='Texas'
GROUP BY a.City
HAVING count(c.CustomerID) > 3
ORDER BY 'Number of Customers' DESC
```

a) Explique conceptualmente qué retorna la misma (información).

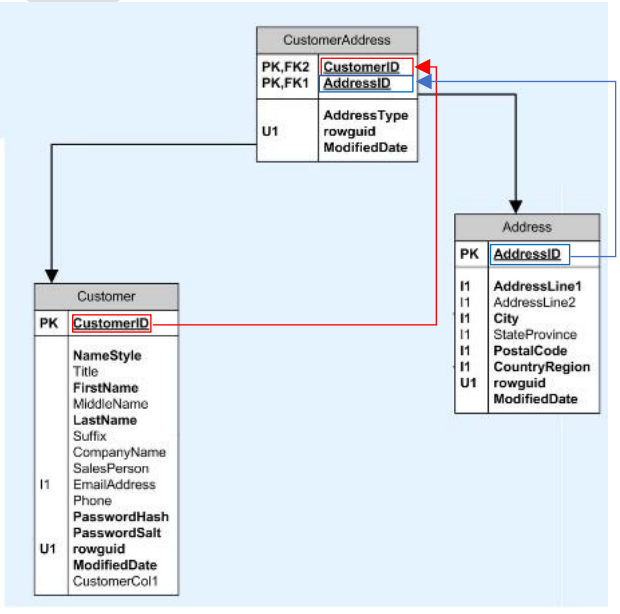
La información que se obtiene con esta consulta es la cantidad de clientes por cada ciudad, en donde elestado sea Texas pero trayendo solamente las ciudades que tienen más de 3 clientes, agrupado por ciudad y ordenados de mayor a menor.

b) Explique con qué intención se utilizaron la cláusula INNER JOIN.

La tabla **Address** (renombrada con el alias "a") muestra todas las direcciones según el estado y ciudad, la tabla **Customer** (renombrada con el alias "c") muestra la información de clientes y la tabla **CustomerAddress** (renombrada con el alias "ca") es la relación entre las dos tablas anteriores; la existencia de esta última tabla se da porque las tablas **Address** y **Customer** tienen cardinalidad N:N, entonces las PK de las mencionadas tablas pasan a ser PK y FK en la relación.

Esta cláusula busca coincidencias entre 2 tablas en función a una columna que tienen en común por lo que sólo la intersección se mostrará en los resultados.

Entre **Address** y **CustomerAddress** se hace **INNER JOIN** para mostrar solo las direcciones de los clientes y luego se vuelve a hacer **INNER JOIN** con **Customer** para mostrar solo cuales son esos clientes.



Las tablas se combinan especificando el campo en común, entre las tablas **Address** y **CustomerAddress** ese campo es la PK de la tabla **Address**, que a su vez es FK en la tabla **CustomerAddress**.

```
a.AddressID = ca.AddressID
```

Esto nos dice que el campo AddressID de la tabla Address sea igual al campo AddressID de la tabla CustomerAddress.

Luego se combina con la tabla **Customer** mediante su PK que es FK en la tabla **CustomerAddress.**

```
c.CustomerID=ca.CustomerID
```

Esto nos dice que el campo CustomerID de la tabla Customer sea igual al campo CustomerID de la tabla CustomerAddress.

c) Explique para qué se utilizó la cláusula HAVING y por qué esto no fue realizado en la cláusula WHERE.

WHERE se aplica sobre registros individuales, es decir registro a registro en la tabla, mientras que con HAVING se puede establecer una condición sobre un grupo de registros, acompañado de la cláusula GROUP BY, cuando se quiere realizar un filtro extra sobre una función de agregación.

```
WHERE a.StateProvince='Texas'
```

Esta sentencia nos indica que se va a filtrar según el campo StateProvince de la tabla **Address** y se van a mostrar solo los registros que digan “Texas”.

```
GROUP BY a.City
```

Se agrupa por el campo City para mostrar las ciudades de Texas y la cantidad de clientes que cada una de ellas tiene.

```
HAVING count(c.CustomerID) > 3
```

Para este segundo filtro no podemos utilizar otro WHERE porque los datos ya fueron procesados en el primero y luego agrupados en el GROUP BY, entonces debemos usar HAVING. Además, como tenemos que filtrar usando la función de agregación, sabemos que indudablemente vamos a necesitar un HAVING. Aquí indicamos que queremos quedarnos solo con las ciudades que tengan más de tres clientes.

Para terminar, se ordena la cantidad de clientes en forma descendente para visualizar las ciudades con mayor cantidad de clientes primero. Podemos hacer referencia al alias “Number of Customers” porque ORDER BY es lo último que se ejecuta.

```
ORDER BY 'Number of Customers' DESC
```

Primero se va a ejecutar la cláusula FROM, luego el ON, luego el WHERE, luego el GROUP BY, luego el HAVING, luego el SELECT y por último el ORDER BY.

Ejercicio 3:

Dadas las siguientes consultas resuelva:

- Si son correctas o no, justifique en caso de no serla y proceda a corregirlas.
- Explique conceptualmente qué retorna (qué información).

a)
```
SELECT CustomerID, FirstName, LastName, RevisionNumber
FROM SalesLT.Customer
WHERE CustomerID = (SELECT CustomerID
FROM SalesLT.SalesOrderHeader
WHERE TotalDue > 10000)
```

La consulta no es correcta porque en la tabla Customer no existe una columna que se llame "RevisionNumber", existe en la tabla SalesOrderHeader.

La cláusula WHERE tiene que estar seguida de un IN, porque la subconsulta retorna una lista y no un único valor.

La información que retorna son las identificaciones de los clientes, su nombre y apellido donde los clientes gastaron más de 10000.

```
SELECT c.CustomerID, c.FirstName, c.LastName, soh.RevisionNumber
FROM  SalesLT.Customer as c
JOIN SalesLT.SalesOrderHeader as SOH on c.CustomerID=soh.CustomerID
WHERE  SOH.TotalDue > 10000
```

b)
```
SELECT CustomerID, FirstName, LastName
FROM SalesLT.Customer
WHERE CustomerID IN
(SELECT ca.CustomerID
FROM SalesLT.CustomerAddress ca
INNER JOIN SalesLT.Address a
ON a.AddressID = a.AddressID
WHERE a.StateProvince = 'Quebec')
```

La consulta es incorrecta porque el "ON" está referenciando a.AddressID dos veces, lo correcto sería: ON ca.AddressID = a.AddressID

Retorna las identificaciones de los clientes, su nombre y apellido donde el estado es "Quebec". En este caso, se utiliza una subconsulta.

Entre **Address** y **CustomerAddress** se hace **INNER JOIN** para mostrar solo las direcciones de los clientes y se filtra la tabla según el campo StateProvince para mostrar solo los que digan Quebec. Esto resulta en una lista de identificaciones de los clientes que se van a corresponder con las identificaciones de los clientes de la tabla Customer.

Ejercicio 4:

Se desea obtener un reporte con el total de facturación y cantidad de facturas emitidas para cada modelo de producto (mostrar nombre) ordenado en forma decreciente por cantidad de facturas (mostrar la información de todos los modelos tengan o no factura emitida).

```
SELECT PROM.Name, SUM(SOH.TotalDue) as Total, COUNT(SOH.SalesOrderID)
AS Cantidad
FROM SalesLT.SalesOrderHeader SOH
JOIN SalesLT.SalesOrderDetail SOD ON SOH.SalesOrderID =
SOD.SalesOrderID
JOIN SalesLT.Product PRO ON SOD.ProductID = PRO.ProductID
RIGHT JOIN SalesLT.ProductModel PROM ON PRO.ProductModelID =
PROM.ProductModelID
GROUP BY PROM.Name
ORDER BY Cantidad DESC
```

La tabla **SalesOrderHeader** guarda información sobre una venta realizada y la tabla **SalesOrderDetail** almacena datos sobre un producto que es vendido, proporcionando datos como la cantidad vendida y el precio unitario.

Hacemos un JOIN entre las tablas **SalesOrderHeader** y **SalesOrderDetail** para conseguir las ventas que tienen el detalle sobre los productos. Luego hacemos otro JOIN con la tabla **Product** para ver los productos que tengan detalle.

Por último hacemos un RIGHT JOIN con la tabla **ProductModel** para obtener la información de todos los modelos, tengan o no una factura emitida.

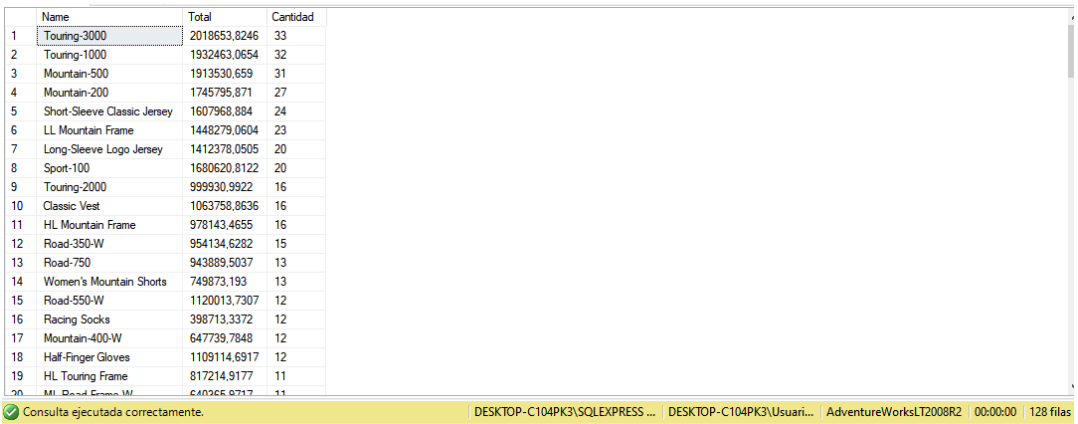
```
GROUP BY PROM.Name
```

Se agrupa por el campo Name de la tabla ProductModel para mostrar el total de facturación según cada modelo y su respectiva cantidad.

```
ORDER BY Cantidad DESC
```

Para terminar, ordenamos en forma decreciente por cantidad de facturas. Podemos hacer referencia al alias “Cantidad” porque ORDER BY es lo último que se ejecuta.

La salida de la consulta se visualiza a continuación:



	Name	Total	Cantidad
1	Touring-3000	2018653,8246	33
2	Touring-1000	1932463,0654	32
3	Mountain-500	1913530,659	31
4	Mountain-200	1745795,871	27
5	Short-Sleeve Classic Jersey	1607968,884	24
6	LL Mountain Frame	1448279,0604	23
7	Long-Sleeve Logo Jersey	1412378,0505	20
8	Sport-100	1680620,8122	20
9	Touring-2000	999930,9922	16
10	Classic Vest	1063758,8636	16
11	HL Mountain Frame	978143,4655	16
12	Road-350-W	954134,6282	15
13	Road-750	943889,5037	13
14	Women's Mountain Shorts	749873,193	13
15	Road-550-W	1120013,7307	12
16	Racing Socks	398713,3372	12
17	Mountain-400-W	647739,7848	12
18	Half-Finger Gloves	1109114,6917	12
19	HL Touring Frame	817214,9177	11

Consulta ejecutada correctamente. | DESKTOP-C104PK3\SQLEXPRESS ... | DESKTOP-C104PK3\Usuari... | AdventureWorksLT2008R2 | 00:00:00 | 128 filas

Se desea obtener un reporte con el nombre de las compañías de los clientes que compraron algún producto de Montaña. Realizarlo sólo con sub-consultas.

```
SELECT CompanyName AS Nombre_compañia
FROM SalesLT.Customer AS C
WHERE C.CustomerID IN
(SELECT SOH.CustomerID
FROM SalesLT.SalesOrderHeader AS SOH
WHERE SOH.SalesOrderID IN
(SELECT SOD.SalesOrderID
FROM SalesLT.SalesOrderDetail AS SOD
WHERE SOD.ProductID IN
(SELECT P.ProductID
FROM SalesLT.Product AS P
WHERE P.ProductModelID IN
(SELECT PM.ProductModelID
FROM  SalesLT.ProductModel  AS  PM
WHERE PM.Name LIKE '%Mountain%'))))
```

La salida de la consulta se visualiza a continuación:

	Nombre_compañia
1	Coalition Bike Company
2	Futuristic Bikes
3	Closest Bicycle Store
4	Many Bikes Store
5	Trailblazing Sports
6	Tachometers and Accessories
7	Metropolitan Bicycle Supply
8	Sports Store
9	Nearby Cycle Shop

Ejercicio 6:

Se desea obtener un reporte con el nombre de la categoría y el nombre del producto de los productos que no han sido ordenados.

```
SELECT PC.Name AS Nombre_categoria, P.Name AS Nombre_producto
FROM SalesLT.Product AS P
JOIN SalesLT.ProductCategory AS PC
ON P.ProductCategoryID = PC.ProductCategoryID
LEFT OUTER JOIN SalesLT.SalesOrderDetail AS SOD
ON P.ProductID = SOD.ProductID
WHERE SOD.ProductID is null
```

Hacemos un JOIN entre las tablas **Product** y **ProductCategory** para conseguir los productos y sus respectivas categorías. Luego hacemos un LEFT OUTER JOIN con la tabla **SalesOrderDetail**, en donde la clave ProductID de la tabla SalesOrderDetail sea nula para obtener los registros que tengan una categoría pero que no fueron ordenados.

La salida de la consulta se visualiza a continuación**:**

	Nombre_categoria	Nombre_producto
1	Road Frames	HL Road Frame - Black, 58
2	Road Frames	HL Road Frame - Red, 58
3	Socks	Mountain Bike Socks, M
4	Socks	Mountain Bike Socks, L
5	Jerseys	Long-Sleeve Logo Jersey, S
6	Road Frames	HL Road Frame - Red, 48
7	Road Frames	HL Road Frame - Red, 52
8	Road Frames	HL Road Frame - Red, 56
9	Road Frames	LL Road Frame - Black, 60
10	Road Frames	LL Road Frame - Black, 62
11	Road Frames	LL Road Frame - Red, 44
12	Road Frames	LL Road Frame - Red, 48
13	Road Frames	LL Road Frame - Red, 52
14	Road Frames	LL Road Frame - Red, 58
15	Road Frames	LL Road Frame - Red, 60
16	Road Frames	LL Road Frame - Red, 62
17	Road Frames	ML Road Frame - Red, 44
18	Road Frames	ML Road Frame - Red, 48
19	Road Frames	ML Road Frame - Red, 52
20	Road Frames	ML Road Frame - Red, 58

Consulta ejecutada correctamente. | DESKTOP-C104PK3\SQLEXPRESS ... | DESKTOP-C104PK3\Usuari... | AdventureWorksLT2008R2 | 00:00:00 | 153 filas

Ejercicio 7:

Se desea obtener un reporte que muestre el nombre, apellido y ciudad de los clientes, que hayan comprado algún producto cuya descripción contenga la palabra “Aluminium” o su categoría contenga la letra “o” en la segunda posición del nombre. El nombre y el apellido deben aparecer en una sola columna (concatenados) separados por un guion, y el apellido debe estar en mayúsculas.

```
SELECT DISTINCT CONCAT(UPPER(C.LastName),' - ', C.FirstName) AS
Nombre_cliente, A.City AS Ciudad
FROM SalesLT.Customer AS C
      JOIN SalesLT.CustomerAddress AS CA
            ON C.CustomerID = CA.CustomerID
      JOIN SalesLT.Address AS A
            ON CA.AddressID = A.AddressID
      JOIN SalesLT.SalesOrderHeader AS SOH
            ON C.CustomerID = SOH.CustomerID
      JOIN SalesLT.SalesOrderDetail AS SOD
            ON SOH.SalesOrderID = SOD.SalesOrderID
      JOIN SalesLT.Product AS P
            ON SOD.ProductID = P.ProductID
      JOIN SalesLT.ProductCategory AS PC
            ON P.ProductCategoryID = PC.ProductCategoryID
      JOIN SalesLT.ProductModel AS PM
            ON P.ProductModelID = PM.ProductModelID
      JOIN SalesLT.ProductModelProductDescription AS PMPD
            ON PM.ProductModelID = PMPD.ProductModelID
      JOIN SalesLT.ProductDescription AS PD
            ON PMPD.ProductDescriptionID = PD.ProductDescriptionID
WHERE PD.Description LIKE '%Aluminium%' OR PC.Name LIKE '_o%'
ORDER BY Nombre_cliente
```

Hacemos un JOIN entre las tablas **Customer**, **CustomerAddress** y **Address** para vincular los clientes con sus direcciones. Luego hacemos un JOIN con las tablas **SalesOrderHeader** y **SalesOrderDetail** para ver las ventas que tienen el detalle sobre los productos.

Después hacemos un JOIN entre **Product** y **ProductCategory** para conseguir la categoría de cada producto y por último hacemos un JOIN entre las tablas **ProductModel**, **ProductModelProductDescription** y **ProductDescription** ya que nos da la descripción de cada modelo de producto.

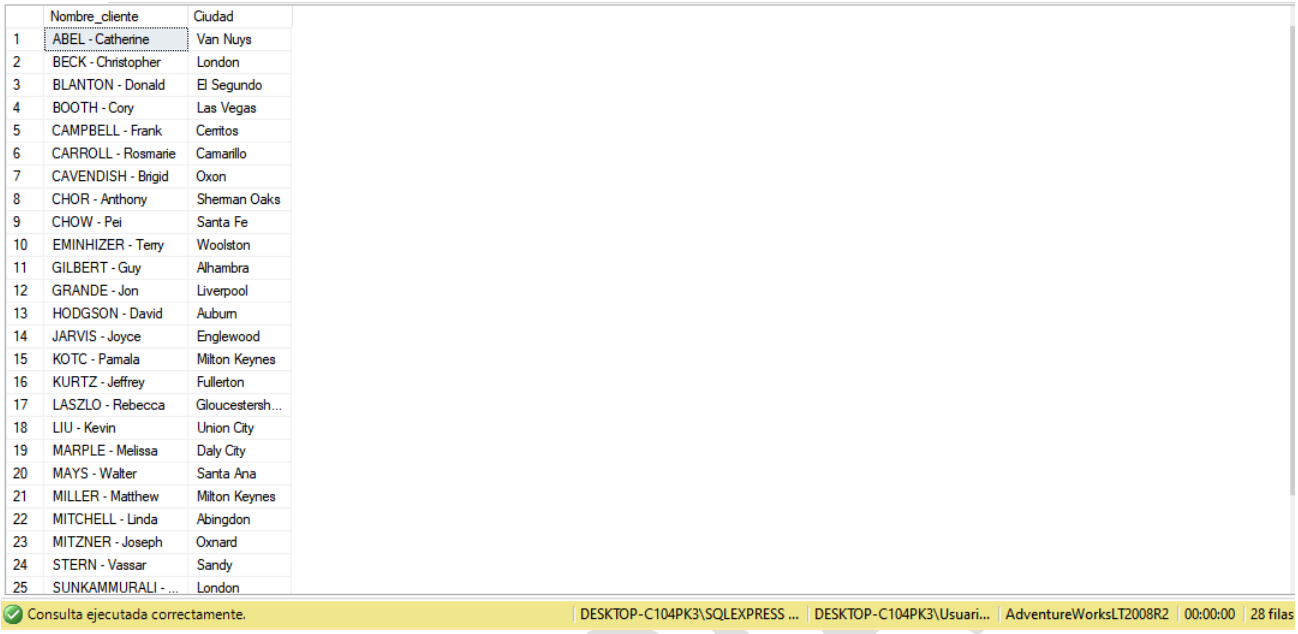
```
WHERE PD.Description LIKE '%Aluminium%' OR PC.Name LIKE '_o%'
```

Esta sentencia nos indica que se va a filtrar según el campo Description de la tabla **ProductDescription** donde se van a mostrar los registros que contengan la palabra “Aluminium” o los registros que contengan la letra “o” en la segunda posición del nombre en la columna Name de la tabla **ProductCategory**.

En este caso se agrega una cláusula DISTINCT para no repetir registros.

Primero se va a ejecutar la cláusula FROM, luego el ON, luego el WHERE, luego el GROUP BY, luego el SELECT, luego el DISTINCT y por último el ORDER BY.

La salida de la consulta se visualiza a continuación:



	Nombre_cliente	Ciudad
1	ABEL - Catherine	Van Nuys
2	BECK - Christopher	London
3	BLANTON - Donald	El Segundo
4	BOOTH - Cory	Las Vegas
5	CAMPBELL - Frank	Cerritos
6	CARROLL - Rosmarie	Camarillo
7	CAVENDISH - Brigid	Oxon
8	CHOR - Anthony	Sherman Oaks
9	CHOW - Pei	Santa Fe
10	EMINHIZER - Terry	Woolston
11	GILBERT - Guy	Alhambra
12	GRANDE - Jon	Liverpool
13	HODGSON - David	Auburn
14	JARVIS - Joyce	Englewood
15	KOTC - Pamala	Milton Keynes
16	KURTZ - Jeffrey	Fullerton
17	LASZLO - Rebecca	Gloucestersh...
18	LIU - Kevin	Union City
19	MARPLE - Melissa	Daly City
20	MAYS - Walter	Santa Ana
21	MILLER - Matthew	Milton Keynes
22	MITCHELL - Linda	Abingdon
23	MITZNER - Joseph	Oxnard
24	STERN - Vassar	Sandy
25	SUNKAMMURALI - ...	London

Consulta ejecutada correctamente. | DESKTOP-C104PK3\SQLEXPRESS ... | DESKTOP-C104PK3\Usuari... | AdventureWorksLT2008R2 | 00:00:00 | 28 filas

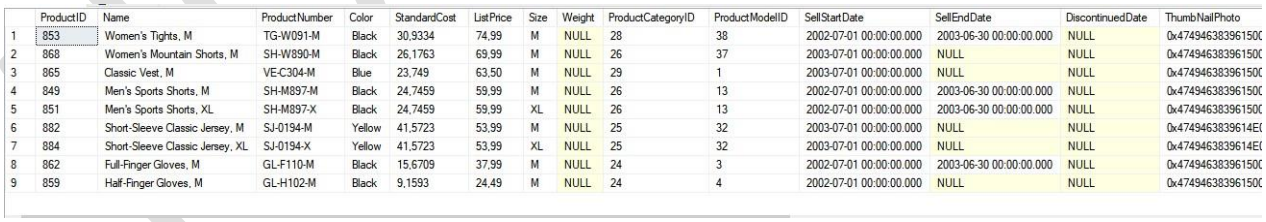
Ejercicio 8

Obtener todos los datos de los productos que tienen precio de lista entre 10 y 500, son exclusivamentede color Rojo, Azul, Amarillo o Negro, y que el tamaño sea M o XL.

Ordénelos por precio de lista de mayor a menor.

```
SELECT *
FROM SalesLT.Product PRO
WHERE PRO.ListPrice BETWEEN 10 AND 500 AND PRO.Color IN ('Red',
'Blue', 'Yellow', 'Black') AND PRO.Size IN ('M', 'XL')
ORDER BY ListPrice DESC
```

La salida de la consulta se visualiza a continuación :



	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductModelID	SellStartDate	SellEndDate	DiscontinuedDate	ThumbNailPhoto
1	853	Women's Tights, M	TG-W091-M	Black	30,9334	74,99	M	NULL	28	38	2002-07-01 00:00:00.000	2003-06-30 00:00:00.000	NULL	0x47494638396150
2	868	Women's Mountain Shorts, M	SH-W890-M	Black	26,1763	69,99	M	NULL	26	37	2003-07-01 00:00:00.000	NULL	NULL	0x47494638396150
3	865	Classic Vest, M	VE-C304-M	Blue	23,749	63,50	M	NULL	29	1	2003-07-01 00:00:00.000	NULL	NULL	0x47494638396150
4	849	Men's Sports Shorts, M	SH-M897-M	Black	24,7459	59,99	M	NULL	26	13	2002-07-01 00:00:00.000	2003-06-30 00:00:00.000	NULL	0x47494638396150
5	851	Men's Sports Shorts, XL	SH-M897-X	Black	24,7459	59,99	XL	NULL	26	13	2002-07-01 00:00:00.000	2003-06-30 00:00:00.000	NULL	0x47494638396150
6	882	Short-Sleeve Classic Jersey, M	SJ-0194-M	Yellow	41,5723	53,99	M	NULL	25	32	2003-07-01 00:00:00.000	NULL	NULL	0x4749463839614E
7	884	Short-Sleeve Classic Jersey, XL	SJ-0194-X	Yellow	41,5723	53,99	XL	NULL	25	32	2003-07-01 00:00:00.000	NULL	NULL	0x4749463839614E
8	862	Full-Finger Gloves, M	GL-F110-M	Black	15,6709	37,99	M	NULL	24	3	2002-07-01 00:00:00.000	2003-06-30 00:00:00.000	NULL	0x47494638396150
9	859	Half-Finger Gloves, M	GL-H102-M	Black	9,1593	24,49	M	NULL	24	4	2002-07-01 00:00:00.000	NULL	NULL	0x47494638396150