

## PRACTICAS DATA 101

La resolución de la práctica se encuentra en este archivo, junto con los archivos .sql todos ellos en el siguiente repositorio:

<https://github.com/jampol71/datawarehouse>

Los archivos que se encuentran en dicho repositorio y que se corresponden con los .sql generados y a los que se hará mención en este documento son los siguientes:

1ANALISIS\_STG\_CLIENTES.sql  
2ANALISIS\_PRODUCTOS.sql  
3ANALISIS\_STG\_FACTURAS.sql  
4ANALISIS\_STG\_CONTACTOS.sql  
5modelo\_clientes.sql  
6modelo\_servicios.sql  
7modelo\_facturas.sql  
8modelo\_contactos.sql

Procedemos a hacer el estudio de cada una de las tablas operacionales para poder realizar el DWH:

### Primera parte

#### Hacer el estudio de STG\_CLIENTES\_CRM

Para hacer el estudio obtendremos los tipos de datos (ilustración 1) mediante la aplicación MySQL Workbench.

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	E:
◆ CUSTOMER_ID	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ FIRST_NAME	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ LAST_NAME	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ IDENTIFIED_DOC	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ GENDER	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ CITY	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ ADDRESS	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ POSTAL_CODE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ STATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ COUNTRY	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ PHONE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ EMAIL	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ BIRTHDAY	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ PROFESION	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ COMPANY	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	

Ilustración 1

Como se puede observar la tabla está compuesta por 15 columnas del tipo varchar. Esto no es adecuado ya que existen campos que no deberían tener ese formato, por ejemplo, POSTAL\_CODE/PHONE deberían ser un INT y BIRTHDAY debería tratarse como una fecha DATETIME. Uno de los perjuicios es que el trabajo de las BBDD utilizando varchar es mucho mayor que por ejemplo mediante la utilización de int.

También observamos que no se cumplen los criterios de normalización, ya que la tabla no dispone de PRIMARY KEY (que en este caso debería ser CUSTOMER\_ID, siendo un campo nullable). La existencia de las PRIMARY KEY nos permiten identificar cada registro de forma única. Por lo que el registro al que asignemos esta propiedad no podrá repetirse. Un ejemplo de creación de PK menos evidente se observará más adelante en la tabla de llamadas, donde, aunque pareciera lo más evidente el número de teléfono no podrá ser la PK debido a que se repite a lo largo de la tabla, y ha sido necesario crear otro registro (ID\_IVR) para tal función.

Las FOREIGN KEY son referencias a registros de otra tabla, para formar entre ambas una relación. Estas claves están relacionadas con las PK de otras tablas. Esto forma la base de los sistemas de bases relationales.

En nuestro caso, la tabla no se encuentra relacionada mediante una FOREIGN KEY con ninguna otra tabla, provocando una isla de información. Por lo que en el nuevo modelo que vamos a desarrollar de Datawarehouse tendremos en cuenta que todas las tablas se encuentren relacionadas de una forma correcta, para aprovechar todas las virtudes de los sistemas de bases de datos relationales.

A nivel de DWH, los datos de las tablas se encuentran en minúscula lo que nos puede provocar problemas al no estar normalizados para trabajar con ellos. Por lo que aplicaremos una serie de transformaciones para facilitar las comparaciones entre los distintos campos (quitar espacios, poner mayúsculas, etc.)

Además, existen muchos datos que se repiten muchas veces a lo largo de la tabla, consumiendo recursos. Estos datos deberíamos sacarlos a otra dimensión y dejar en esta tabla un identificador del mismo que consumirá muchísimos menos recursos.

A nivel de normalización, necesaria para poder trabajar con este tipo de sistemas nos encontramos en las tablas fuente proporcionadas:

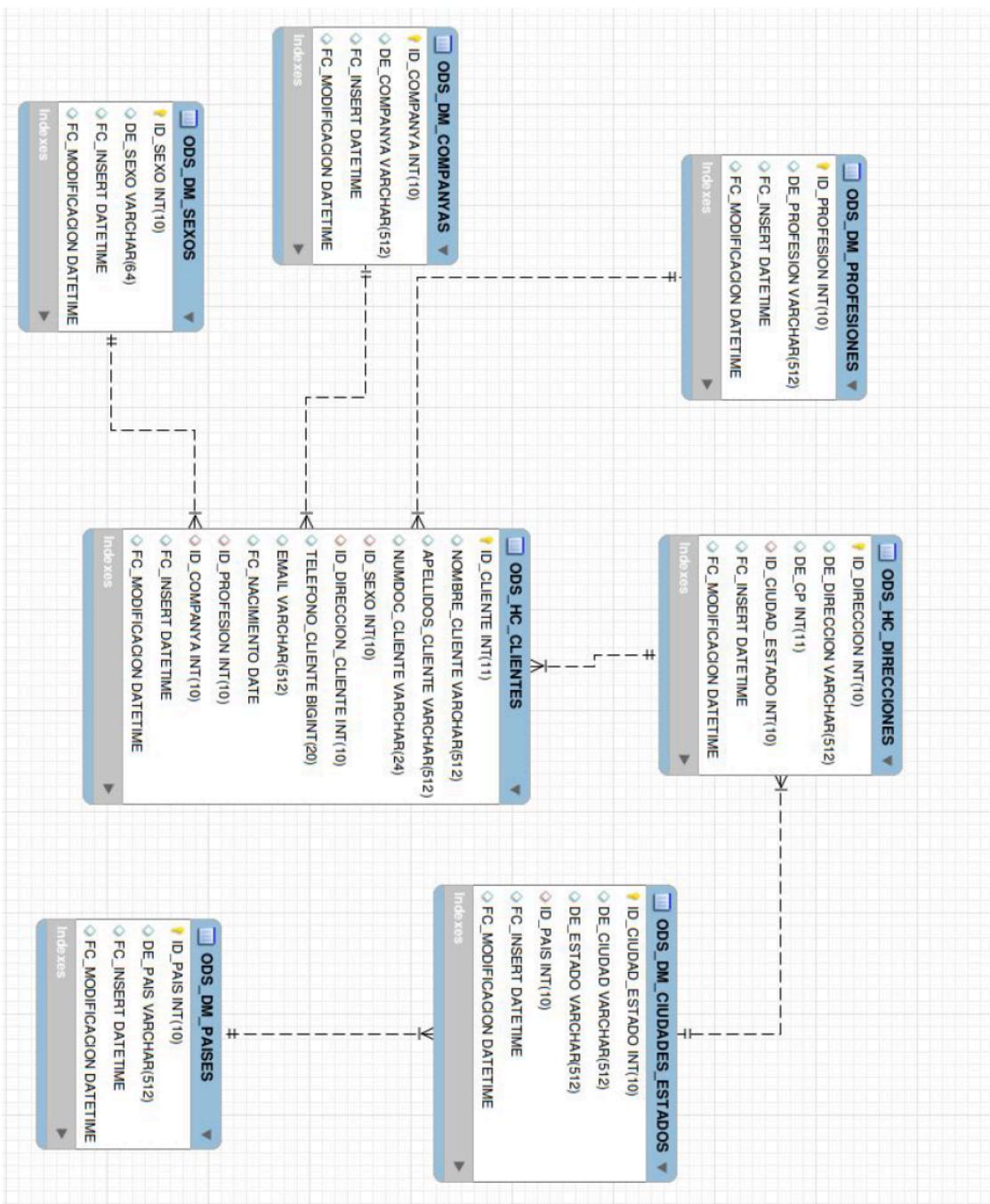
No cumple NF1: La tabla no contiene una clave única. La tabla tiene datos duplicados

No cumple NF2: Las columnas no dependen de otras tablas

No cumple NF3: Las columnas no dependen en las otras tablas de PRIMARY KEY

A partir de aquí utilizaremos la información que nos proporciona la tabla para crear un modelo normalizado.

Estudiamos cada uno de los campos por separado, para ver el lugar que ocupará en el modelo:



Una vez esbozado el modelo, veremos qué es lo que vamos a hacer con cada uno de los campos de la tabla:

**CUSTOMER\_ID:** Es el que utilizaremos como ID\_CLIENTE en la tabla ODS\_HC\_CLIENTES

**FIRST\_NAME:** Es el que utilizaremos como NOMBRE\_CLIENTE en la tabla ODS\_HC\_CLIENTES

**LAST\_NAME:** Es el que utilizaremos como APELLIDOS\_CLIENTE en la tabla ODS\_HC\_CLIENTES

**IDENTIFIED\_DOC:** Es el que utilizaremos como NUMDOC\_CLIENTE en la tabla ODS\_HC\_CLIENTES

**GENDER:** Se crea otra tabla de dimensión ODS\_DM\_SEXOS

**CITY:** Se crea otra tabla de dimensión ODS\_DM\_CIUDADES\_ESTADO

**ADDRESS:** Se crea otra tabla de dimensión ODS\_HC\_DIRECCIONES

**POSTAL\_CODE:** Se pone en la tabla de dimensión ODS\_HC\_DIRECCIONES

**STATE:**

**COUNTRY:**

**PHONE:** Es el que utilizaremos como TELEFONO\_CLIENTE en la tabla ODS\_HC\_CLIENTES

**EMAIL:** Es el que utilizaremos como EMAIL en la tabla ODS\_HC\_CLIENTES

**BIRTHDAY:** Es el que utilizaremos como FC\_NACIMIENTO en la tabla ODS\_HC\_CLIENTES

**PROFESION:** La sacamos a la tabla ODS\_DM\_PROFESIONES

**COMPANY:** La sacamos a la tabla ODS\_DM\_COMPANYAS

La tabla ODS\_HC\_DIRECCIONES la rellenaremos de dos sitios, los datos que vienen de clientes y los que vienen de productos. Al comparar las direcciones vemos que estas direcciones no se repiten, por lo que la tabla contendrá la suma de direcciones proporcionadas por las dos tablas.

Además realizaremos una query a la tabla para obtener información adicional de cada uno de los campos. De la query del fichero 1 del repositorio obtenemos la siguiente información:

TOTAL_REGISTROS:	17558
TOTAL_CUSTOMER_ID:	17558
TOTAL_DISTINTOS_CUSTOMER_ID:	17558
TOTAL_FIRST_NAME:	17558
TOTAL_DISTINTOS_FIRST_NAME:	7314
TOTAL_LAST_NAME:	17497
TOTAL_DISTINTOS_LAST_NAME:	14577
TOTAL_IDENTIFIED_DOC:	17497
TOTAL_DISTINTOS_IDENTIFIED_DOC:	17498
TOTAL_GENDER:	17497
TOTAL_DISTINTOS_GENDER:	3
TOTAL_CITY:	17497
TOTAL_DISTINTOS_CITY:	82
TOTAL_ADDRESS:	17497
TOTAL_DISTINTOS_ADDRESS:	17439
TOTAL_POSTAL_CODE:	17497
TOTAL_DISTINTOS_POSTAL_CODE:	274
TOTAL_STATE:	17497
TOTAL_DISTINTOS_STATE:	4
TOTAL_COUNTRY:	17497

TOTAL_DISTINTOS_COUNTRY:	2
TOTAL_PHONE:	17497
TOTAL_DISTINTOS_PHONE:	17498
TOTAL_EMAIL:	17497
TOTAL_DISTINTOS_EMAIL:	17498
TOTAL_BIRTHDAY:	17497
TOTAL_DISTINTOS_BIRTHDAY:	10753
TOTAL_PROFESION:	17497
TOTAL_DISTINTOS_PROFESION:	196
TOTAL_COMPANY:	17451
TOTAL_DISTINTOS_COMPANY:	384

Dudas que nos salen al estudiar la tabla:

- Hay menos números de identidad que clientes tenemos

```
SELECT COUNT(*)
FROM STG_CLIENTES_CRM CLI
WHERE CLI.IDENTIFIED_DOC='';
```

Me sale que tenemos 61 documentos de identidad sin rellenar por lo que está ok.

- Hay menos gender que clientes

```
SELECT COUNT(*)
FROM STG_CLIENTES_CRM CLI
WHERE CLI.GENDER='';
```

Me sale que 61 personas no tienen puesto el sexo al que pertenecen.

Lo mismo ocurre con ciudades, direcciones, códigos postales, países, números de teléfono, emails, años de nacimiento y profesiones.

En cambio de 107 clientes no tenemos la empresa donde trabajan.

Cómo se puede observar existen muchos campos que se han quedado **sin rellenar** y esto es un grave error porque no sabemos si es un campo que no es necesario para la empresa o que puede no ser rellenado por la persona que introduce los datos.

Para solucionar este problema, en nuestro modelo todos los campos estarán llenos aunque su valor puede ser 'DESCONOCIDO' indicando que debería estar lleno pero por alguna razón no lo está (y con esta información ya podremos tomar las acciones necesarias dentro de la empresa) o su valor puede ser 'NO APLICA', lo que nos indicaría que es un campo que la empresa no necesita que esté llenado.

Para la creación del nuevo modelo, seguiremos tres pasos. Primero, crearemos las tablas que necesitamos indicando el tipo de dato adecuado en cada uno de los campos, después pondremos las relaciones entre las tablas y por último poblaremos las tablas con datos.

Procedemos a crear las tablas del nuevo modelo de clientes (fichero 5 del repositorio). En este apartado lo más importante es separar aquellas tablas cuyos registros serán más o menos fijos (dimensiones) de aquellas en las que se añadirán registros totalmente distintos (que las llamaremos tablas de hechos).

## a) Hacer el estudio de STG\_PRODUCTOS\_CRM

La tabla tiene las columnas que se definen a continuación:

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
◊ PRODUCT_ID	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ CUSTOMER_ID	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ PRODUCT_NAME	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ ACCESS_POINT	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ CHANNEL	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ AGENT_CODE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ START_DATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ INSTALL_DATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ END_DATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ PRODUCT_CITY	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ PRODUCT_ADDRESS	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ PRODUCT_POSTAL_CODE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ PRODUCT_STATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update
◊ PRODUCT_COUNTRY	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update

Como se puede observar no se encuentra normalizada y tiene los mismos problemas que la tabla anterior al ser todo cadenas y no disponer de PK ni de FK.

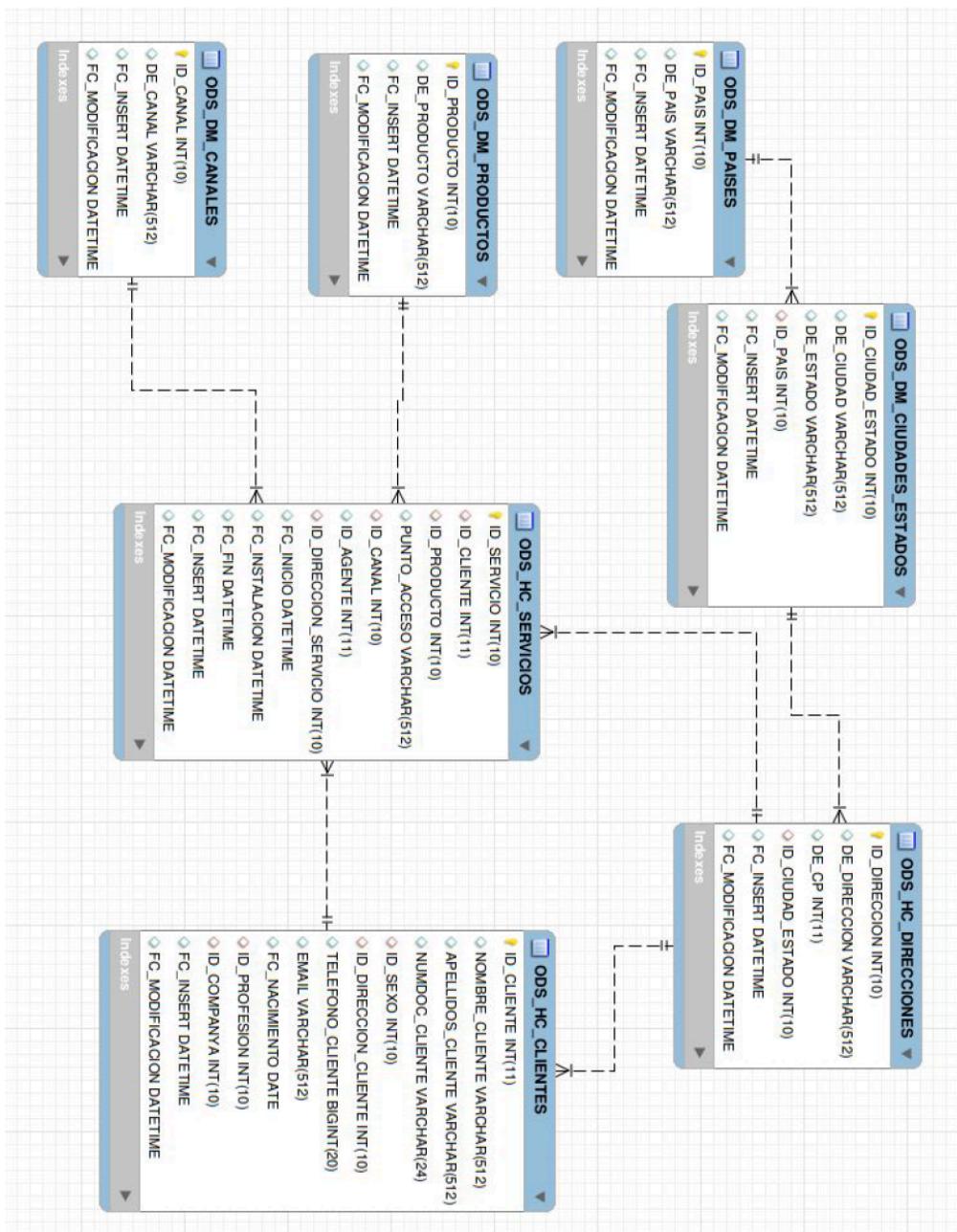
Al ver la tabla de productos, nos damos cuenta de que realmente son SERVICIOS ofrecidos por la empresa de determinados productos. Por lo que crearemos una tabla ODS\_HC\_SERVICIOS que será la principal del modelo.

De la query de análisis de la tabla (fichero 2 del repositorio), sacamos lo siguiente:

TOTAL_REGISTROS:	78495
TOTAL_PRODUCT_ID:	78495
TOTAL_DISTINTOS_PRODUCT_ID:	78495
TOTAL_CUSTOMER_ID:	78495
TOTAL_DISTINTOS_CUSTOMER_ID:	8001
TOTAL_PRODUCT_NAME:	78495
TOTAL_DISTINTOS_PRODUCT_NAME:	6
TOTAL_ACCESS_POINT:	78274
TOTAL_DISTINTOS_ACCESS_POINT:	78275
TOTAL_CHANNEL:	78274
TOTAL_DISTINTOS_CHANNEL:	5
TOTAL_AGENT_CODE:	42630
TOTAL_DISTINTOS_AGENT_CODE:	701
TOTAL_START_DATE:	78495
TOTAL_DISTINTOS_START_DATE:	8035
TOTAL_INSTALL_DATE:	75363
TOTAL_DISTINTOS_INSTALL_DATE:	75360

TOTAL_END_DATE:	46684
TOTAL_DISTINTOS_END_DATE:	46683
TOTAL_PRODUCT_CITY:	78274
TOTAL_DISTINTOS_PRODUCT_CITY:	82
TOTAL_PRODUCT_ADDRESS:	78274
TOTAL_DISTINTOS_PRODUCT_ADDRESS:	77037
TOTAL_PRODUCT_POSTAL_CODE:	78274
TOTAL_DISTINTOS_PRODUCT_POSTAL_CODE:	274
TOTAL_PRODUCT_STATE:	78090
TOTAL_DISTINTOS_PRODUCT_STATE:	4
TOTAL_PRODUCT_COUNTRY:	78274
TOTAL_DISTINTOS_PRODUCT_COUNTRY:	2

El modelo que hemos desarrollado es el siguiente:



Analizaremos campo a campo la tabla:

**PRODUCT\_ID:** Pasará a llamarse ID\_SERVICIO en nuestra tabla principal ODS\_HC\_SERVICIOS

**CUSTOMER\_ID:** Para nosotros ID\_CLIENTE en la tabla ODS\_HC\_SERVICIOS:

La empresa nos indica que comparte CUSTOMER\_ID con STG\_CLIENTES\_CRM, esto nos permitirá relacionar los servicios con el cliente a quién se le ha ofrecido.

En primer lugar miramos si todos los ID\_CLIENTE de la tabla se encuentran en el campo ID\_CLIENTE nuestra tabla principal ODS\_HC\_CLIENTES (que a su vez viene de STG\_CLIENTES\_CRM) . Para hacerlo utilizamos:

```
SELECT distinct (CUSTOMER_ID)
FROM ODS_HC_CLIENTES CLI
RIGHT OUTER JOIN STAGE.STG_PRODUCTOS_CRM PRO
ON PRO.CUSTOMER_ID=CLI.ID_CLIENTE
WHERE CLI.ID_CLIENTE IS NULL;
```

Obteniendo:

#	CUSTOMER_ID
1	16689
2	10000
3	14826

Por lo que hay 3 clientes que se encuentran en la fuente de productos y que no se encuentran en nuestra tabla de clientes.

Ante esta situación, se decide estudiar estos tres casos en particular e iniciar una política para controlar los permisos que tienen cada uno de los empleados en la manipulación de datos (Data Management).

**PRODUCT\_NAME:** Sacaremos una tabla de dimensión ODS\_DM\_PRODUCTOS, donde será DE\_PRODUCTO

**ACCESS\_POINT:** Será PUNTO\_ACCESO en nuestra tabla ODS\_HC\_SERVICIOS

**CHANNEL:** Sacaremos una tabla de dimensión ODS\_DM\_CANALES donde será DE\_CANAL

**AGENT\_CODE:** Se sacarán con la tabla ODS\_DM\_AGENTES\_CC

Hay datos de columnas que se repiten frecuentemente y se podrían sacar, y campos vacíos que deberíamos llenar.

Tenemos algunos campos que diferencian los datos del cliente de los productos instalados con respecto a localización.

Las direcciones que nos proporciona la tabla STG\_PRODUCTOS\_CRM se añadirán a la tabla de direcciones que hemos creado previamente, pero habrá que verificar si ya existen en la misma (esto se puede observar en el archivo 6 del repositorio en la población de la tabla ODS\_HC\_DIRECCIONES).

Realizo la siguiente query para asegurarme que no se repiten direcciones entre las 2 tablas:

```
SELECT *
FROM STAGE.STG_CLIENTES_CRM CLI
INNER JOIN STAGE.STG_PRODUCTOS_CRM PRO ON CLI.ADDRESS=PRO.PRODUCT_ADDRESS

WHERE UPPER(TRIM(CLIENTE.CITY))=UPPER(TRIM(PRODUCTO.PRODUCT_CITY))
    AND UPPER(TRIM(CLIENTE.STATE))=UPPER(TRIM(PRODUCTO.PRODUCT_STATE))
    AND UPPER(TRIM(CLIENTE.POSTAL_CODE))=UPPER(TRIM(PRODUCTO.PRODUCT_POSTAL_CODE))
    AND UPPER(TRIM(CLIENTE.COUNTRY))=UPPER(TRIM(PRODUCTO.PRODUCT_COUNTRY))
    AND UPPER(TRIM(PRODUCTO.PRODUCT_COUNTRY))='UNITED STATES' OR '';
```

Ésta me devuelve que existen 244 direcciones vacías, pero ninguna dirección que se encuentre completa y esté en las 2 tablas.

#	CUSTOMER_ID	FIRST_NAME	LAST_NAME	IDENTIFIED_DOC	GENDER	CITY	ADDRESS	POSTAL_CODE	STATE	COUNTRY	PHONE	EMAIL
1	14934	Nathaniel										
2	14946	Melli										
3	15004	Axe										
4	15033	Stormy										

El campo profesión también lo sacamos y lo ponemos como **NO APLICA** porque puede ser una persona jubilada.

El campo compañía lo ponemos también como **NO APLICA** por la misma razón que antes.

Es importante señalar que para lograr poner el ID\_DIRECCION en la tabla de ODS\_HC\_SERVICIOS tendremos que hacer dos tablas temporales muy parecidas a las realizadas en clientes que relacionen el ID\_SERVICIO con el ID\_DIRECCION (esto se puede observar en el fichero 6 del repositorio).

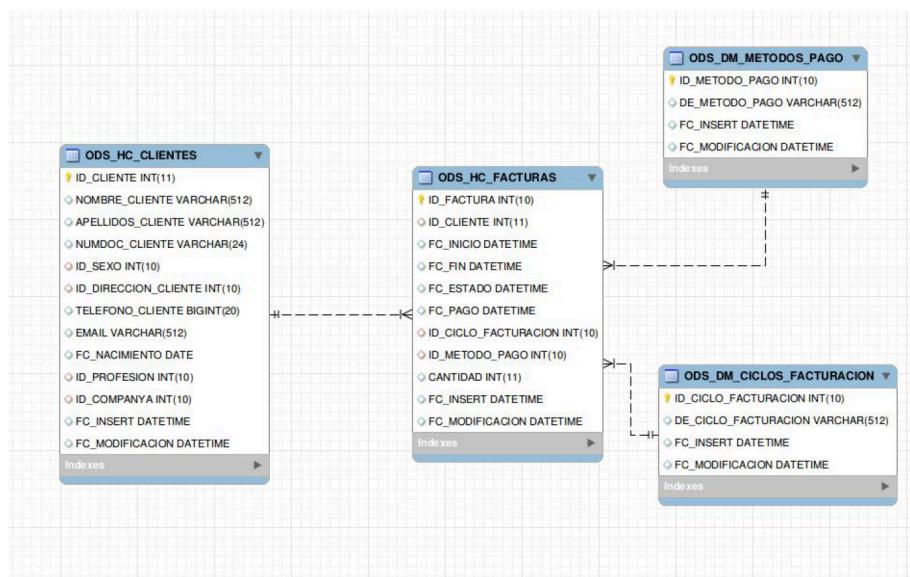
## b) Hacer el estudio de STG\_FACTURAS\_FTC

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
◆ BILL_REF_NO	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ CUSTOMER_ID	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ START_DATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ END_DATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ STATEMENT_DATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ PAYMENT_DATE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ BILL_CYCLE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ AMOUNT	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ BILL_METHOD	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer

Nos encontramos los mismos errores de normalización que en las otras tablas.

Al indicarnos la empresa que comparte CUSTOMER\_ID con STG\_PRODUCTOS y con STG\_CRM,

Pasamos a mostrar el modelo que hemos desarrollado:



### c) Hacer el estudio de STG\_CONTACTOS\_IVR

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
◆ ID	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ PHONE_NUMBER	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ START_DATETIME	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ END_DATETIME	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ SERVICE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ FLG_TRANSFER	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer
◆ AGENT	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,referer

Nos encontramos los mismos errores de normalización que en las otras tablas.

Se intenta mirar si algún número de teléfono de esta tabla cruza con alguno de clientes mediante una query:

```
select *
from ODS.ODS_HC_CLIENTES CLI
INNER JOIN STAGE.STG_CONTACTOS_IVR CONT ON UPPER(TRIM(CONT.PHONE_NUMBER))=UPPER(TRIM(CLIENTE.TELEFONO_CLIENTE));
```

Pero obtenemos que no existe ningún número de teléfono en común entre las dos tablas:

#	ID_CLIENTE	NOMBRE_CLIENTE	APELLIDOS_CLIENTE	NUMDOC_CLIENTE	ID_SEXO	ID_DIRECCION_CLIENTE	TELEFONO_CLIENTE	EMAIL	FC_NACIMIENTO	ID_PF
1	12:48:55	select * #, CASE WHEN length(trim(CLIENTE.ID_CLIENTE))<>0 THEN ...	TH...	0 row(s) returned						1823.142 sec / 0.00...

### No comparte campos con las tablas anteriores

Al intentar buscar a qué cliente pertenecen las llamadas, nos encontramos que con los datos que nos proporciona la tabla es imposible conocer quién las ha realizado.

Por lo que para “solucionar” el problema, crearemos una tabla ODS\_HC\_LLAMADAS que contenga cada una de las llamadas que se han realizado. Debido a que el campo ID\_LLAMADA se repite no podrá ser la PRIMARY KEY, por lo que pondremos otro campo ID\_IVR que será autoincrementable.

El campo ID\_CLIENTE no cruza con ninguno de la tabla de clientes, por lo que se ha generado un cliente ficticio (9999999999) donde irán las llamadas que no se cruzan con

clientes (que en este caso son todas). Para ello, a la hora de poblar la tabla la columna siempre tiene ese valor.

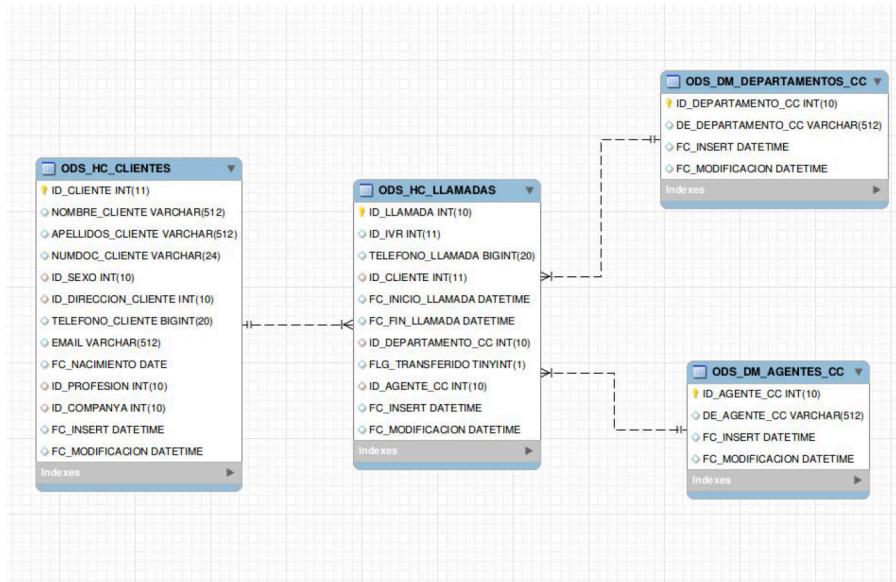
```

INSERT INTO ODS_HC_LLAMADAS(ID_LLAMADA,TELEFONO_LLAMADA,ID_CLIENTE,FC_INICIO_LLAMADA,FC_FIN_LLAMADA,DEPARTAMENTO_CC,FLG_TRANSFERIDO,ID_AGENTE,FC_INSERT
select CONT.ID_LLAMADA
,CASE WHEN length(TRIM(CONT.PHONE_NUMBER))<>0 THEN UPPER(TRIM(CONT.PHONE_NUMBER)) ELSE 999999999 END TELEFONO_LLAMADA
,CASE WHEN length(TRIM(START_DATETIME))<>0 THEN STR_TO_DATE(START_DATETIME, '%Y-%m-%d %T.%f') ELSE str_to_date('9999/12/31 00:00:00', '%Y/%m/%d %T') END FC_
,CASE WHEN length(TRIM(END_DATETIME))<>0 THEN STR_TO_DATE(END_DATETIME, '%Y-%m-%d %T.%f') ELSE str_to_date('9999/12/31 00:00:00', '%Y/%m/%d %T') END FC_
,DEP.ID_DEPARTAMENTO_CC
,CASE WHEN TRIM(FLG_TRANSFER)=TRUE THEN 1 ELSE 0 END FLG_TRANSFERIDO
,AGENTE.ID_AGENTE_CC
,NOW()
,NOW()
from STAGE.STG_CONTACTOS_IVR CONT
INNER JOIN ODS.ODS_DM_DEPARTAMENTOS_CC DEP ON CASE WHEN LENGTH(TRIM(SERVICE))<>0 THEN UPPER(TRIM(SERVICE)) ELSE 9999 END=DEP.DE_DEPARTAMENTO_CC
INNER JOIN ODS.ODS_DM_AGENTES_CC AGENTE ON CASE WHEN LENGTH(TRIM(AGENT))<>0 THEN UPPER(TRIM(AGENT)) ELSE 9999 END=AGENTE.DE_AGENTE
LEFT OUTER JOIN ODS.ODS_HC_CLIENTES CLI ON CASE WHEN LENGTH(TRIM(PHONE_NUMBER))<>0 THEN UPPER(TRIM(PHONE_NUMBER)) ELSE 9999 END=CLI.TELEFONO_CLIENTE;

```

En la tabla ODS\_HC\_LLAMADAS aparecerá siempre el número cliente como 99999999, que es el que hemos escogido como desconocido.

El modelo que se ha desarrollado tiene el siguiente aspecto:



d) Hacer el estudio de STG\_ORDENES\_CRM

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	E>
◆ ID	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ ORDER	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ PHASE	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ AGENT	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ START_DT	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	
◆ END_DT	varchar(512)		YES	latin1	latin1_swedish_ci	select,insert,update,references	

Nos encontramos los mismos errores de normalización que en las otras tablas.

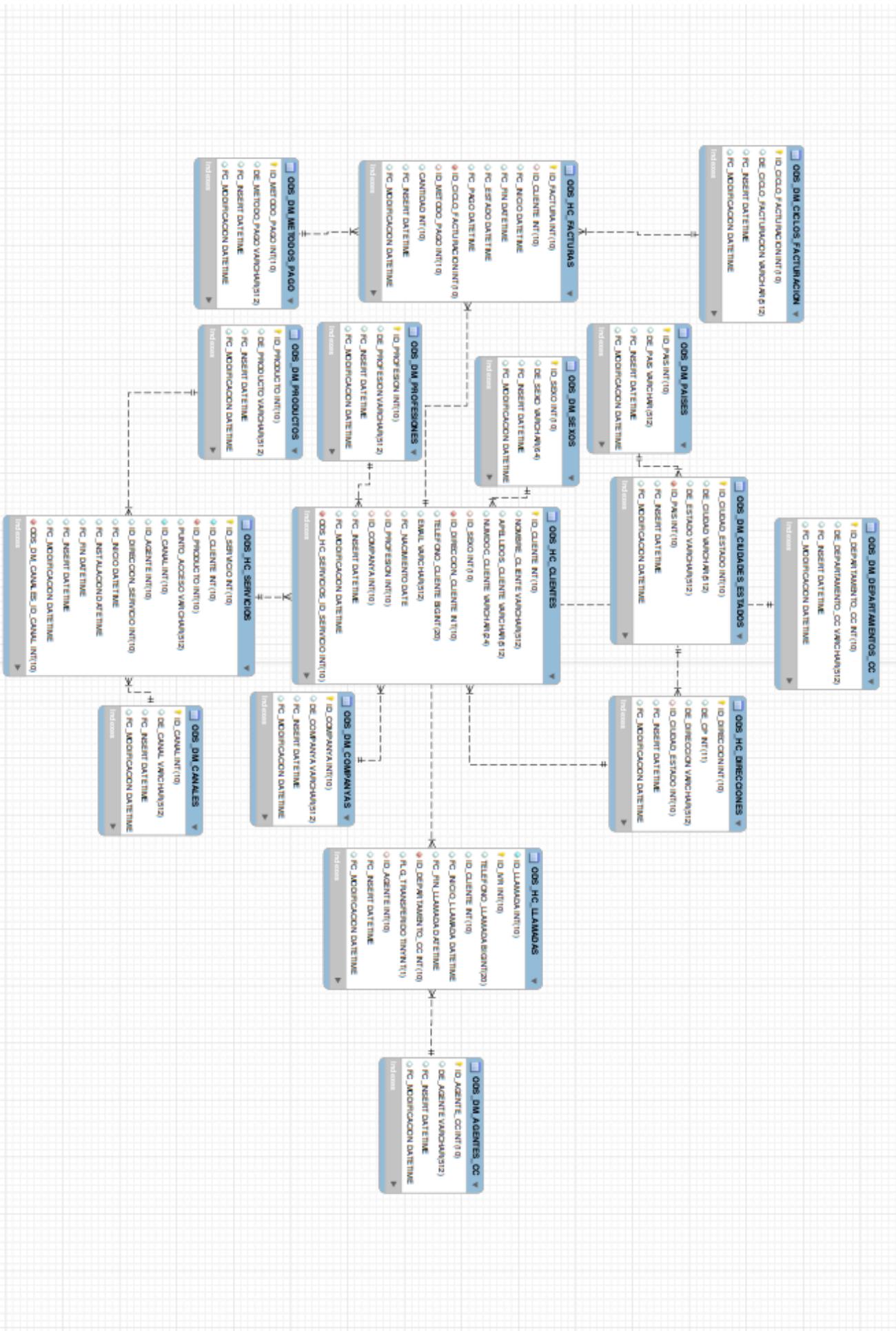
## Crear el modelo, las tablas, las FK y poblar las tablas

El modelo ha sido explicado por partes en el ejercicio anterior. La creación de las FK y la población de las tablas se encuentra en los archivos del repositorio relativos a cada uno de los modelo (5,6,7 y 8).

## Segunda Parte

Adjunta tu diagrama de ODS completo con el número de registros que contiene cada tabla

A continuación, se mostrará el ODS completo para verificar que no se ha quedado ninguna isla de información.



ODS\_DM\_AGENTES\_CC: 593+2  
ODS\_DM\_CANALES: 4+2  
ODS\_DM\_CICLOS\_FACTURACION:2+2  
ODS\_DM\_CIUDADES\_ESTADO:157+2  
ODS\_DM\_COMPANYAS:383+2  
ODS\_DM\_DEPARTAMENTOS\_CC:6+2  
ODS\_DM\_METODOS\_PAGO:3+2  
ODS\_DM\_PAISES:1+2  
ODS\_DM\_PRODUCTOS:6+2  
ODS\_DM\_PROFESIONES:195+2  
ODS\_DM\_SEXOS:2+2

ODS\_HC\_CLIENTES:17558  
ODS\_HC\_DIRECCIONES:95773  
ODS\_HC\_FACTURAS:420000  
ODS\_HC\_LLAMADAS: 202717  
ODS\_HC\_SERVICIOS:78495

Por qué en el modelo de DIRECCIONES dejo en la misma tabla las CIUDADES y los ESTADOS y no los separo en dos tablas distintas para ser más estricta con la jerarquía: PAIS → ESTADOS → CIUDADES → DIRECCIONES

Se deja en la misma tabla ciudades y estados, debido a que hay una ciudad con el mismo nombre que está en 2 estados diferentes lo que nos obliga a juntar las tablas.

#	DE_CIUDAD	DE_ESTADO
1	GLENDALE	CALIFORNIA
2	GLENDALE	ARIZONA

OPCIONAL: ¿Serías capaz de separar el campo DE\_DIRECCION de la tabla de direcciones en dos campos: NOMBRE\_VIA y NUM\_VIA?

## Tercera Parte

La realidad es que si hubiésemos aplicado el “Data Management”, muchas de las acciones que hemos tenido que realizar nos las hubiésemos evitado porque deberían estar controladas de otra forma. Explica qué habrías hecho diferente centrándote en las “patas”:

La gestión de los datos, que engloba el conjunto de todas las disciplinas relacionadas con gestionar los datos como un activo valioso es el Data Management. Su objetivo es tener una visión unificada de todos los datos de la empresa, así como de su entorno de negocio.

- **Data Quality.** Se encarga de definir, controlar y mejorar la calidad de los datos. Es necesario que los datos sean fiables, precisos, consistentes y que proporcionen una visión única



Características subjetivas

Características objetivas

En primer lugar tenemos que cumplir con toda la legislación de protección de datos que exista tanto a nivel europeo como nacional. Si vamos a utilizar ese dato fuera de la UE deberemos enterarnos de las diferentes legislaciones que existen en los países afectados.

Tendremos que tener a una persona encargada que sea capaz de evitar los posibles ataques externos que puedan comprometer los datos que tengamos en nuestra empresa.

Deberemos testear con los diferentes departamentos que los diferentes datos se corresponden con los que ellos poseen en sus sistemas.

Además al crear el ODS hemos visto ciertas características que tienen que tener los datos:

- Para un mejor manejo de los mismos todos tendrán que estar en mayúsculas y sin espacios al principio y al final para de esta forma evitar que el sistema no trabaje correctamente con el dato.
  - Cada dato deberá estar en su correspondiente formato o tipo de dato.
  - Las tablas deberán tener la mayor cantidad posible de identificadores relacionados con su correspondiente dimensión.
  - Las tablas deberán estar correctamente relacionadas mediante PK y FK para que no existan islas de información.
  - El dato tiene que estar escrito de la misma forma en todos los sitios. En la práctica por ejemplo hay campos donde en algunos sitios está 'United States' y en otros 'US' provocando problemas a la hora de cruzarlos.
  - Las tablas de origen tienen que tener un mismo patrón a la hora de introducir los datos, ya que nos encontramos tablas que no tienen rellenados datos que son importantes para el negocio.
  - No puede haber campos nulos, tienen que ponerse bien a desconocido o a no aplica según la lógica de negocio.
- 
- **Master Data.** Datos maestros estandarizados en los diferentes sistemas en los que se encuentran.



Como hemos visto anteriormente, existen clientes que se encuentran en la tabla de productos/servicios y que no están en la tabla de clientes, y esto es debido a que no existe un sistema en el que los datos estén estandarizados. Habría que tomar medidas y tener ciertos datos que se encuentren una vez en el sistema sin que exista la posibilidad de que estén en unas tablas y no se hayan pasado a otras.

Se evita la duplicación de los datos dentro de una misma tabla si la fuente de donde se cogen no tiene repetidos y se realizan bien las consultas.

- Data Modeling & Design (me refiero a como están definidas las tablas de origen). Modela y diseña las BBDD, se encarga de su implementación y soporte, de manera que los datos puedan ser utilizados como recursos.

Cada BBDD tiene que tener permiso de acceso sólo por la persona que la vaya a utilizar, evitando que personas que no deban vean datos que no necesitan para su trabajo.



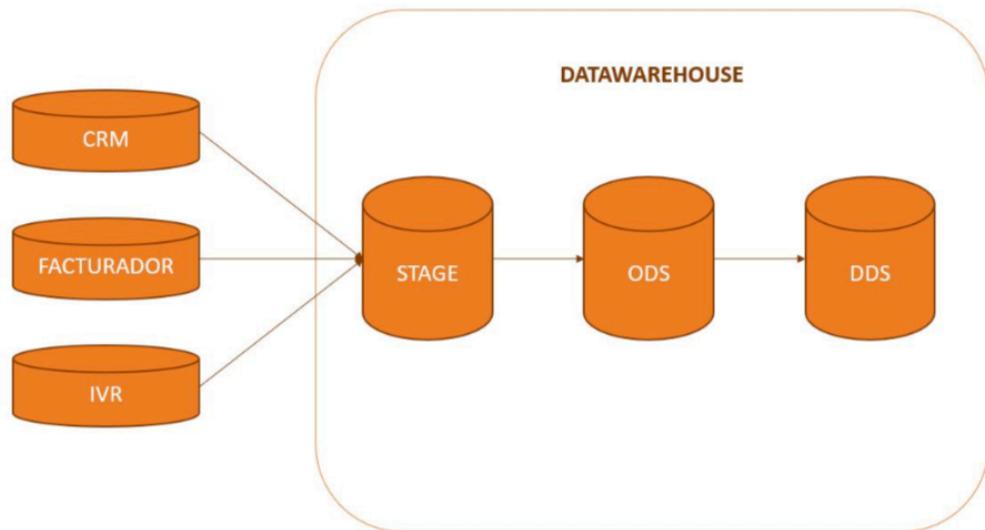
**OPCIONAL:** ¿Aconsejarías algún cambio en los sistemas origen extra teniendo en cuenta el resto de disciplinas del Data Governance?

El Data Governance se hace cargo de tener la trazabilidad de cualquier dato para saber su camino en su flujo de negocio.

**OPCIONAL++:** Utilizando alguna herramienta del mercado o inventándote un modelo en BBDD genera la trazabilidad de la información (Data Governance).

## Cuarta Parte

Después de todo lo visto nuestro ecosistema quedaría así:



¿Lo dejarías así o plantearías otro diseño mejorado? Si es que sí, esbózalo(no te ibas a librar :P)

Desde mi punto de vista, el diseño del modelo cumple de forma satisfactoria con el propósito de un DWH, teniendo copias de las fuentes de datos por si fuera necesario consultar el dato sin manipular y después un lugar donde manipulamos la información para ponerla de la forma que nos interese.

## Quinta Parte

### Escribe tus propias reglas o mandamientos de un DataWarehouse

- No podemos encontrarnos con ningún campo que se encuentre vacío, lo que la empresa tiene que conocer es si determinados campos no han sido rellenados porque no es interesante para el negocio o si determinados campos deberían estar llenos y no lo están.
- Todos los datos se encontrarán en mayúsculas para que no existan problemas de normalización
- Todas las BBDD tienen que estar relacionadas entre sí, para que podamos llegar a cualquier dato que necesitemos
- La información relevante de nuestro DWS deberá estar documentada, para en caso de que necesitemos realizar tareas de mantenimiento conocemos la función de cada una de las partes. Además, una persona que vea por primera vez el sistema podrá hacerse una idea de cómo funciona.
- Algunos datos deberán estar centralizados, para de esta forma no tener duplicidades y asegurarnos que la calidad del mismo es la adecuada (Data Master).
-

## Sexta Parte

¿Nivel de SQL antes y después?

Al empezar estaba más perdido que un muelle en las escaleras de Howgarts.

¿Algún comentario extra que quieras hacer?