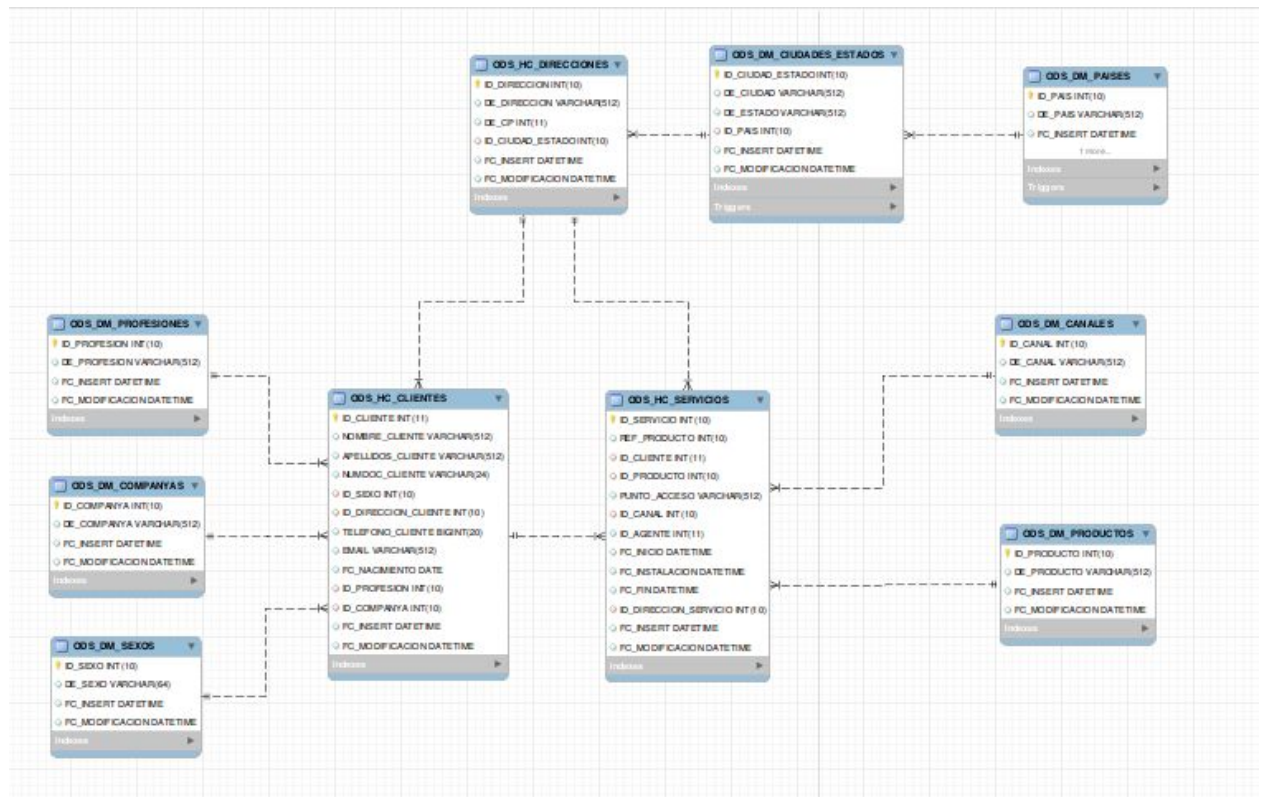


Práctica de DWH



Descripción de la práctica

Sobre el modelo generado en clase, que se puede observar en la parte superior, responder a las preguntas que vienen a continuación.

Índice del documento

| | |
|--|-----------|
| Descripción de la práctica | 1 |
| Estudio de las tablas de STG | 3 |
| Estudio de la tabla STAGE.STG_FACTURAS_FCT. | 3 |
| Estudio de la tabla STG_CONTACTOS_IVR: | 7 |
| Creación de modelos en OSD | 10 |
| Creación del Modelo de FACTURAS: | 10 |
| Creación de la tabla DIM_BILL_CYCLE | 10 |
| Creación de la tabla DIM_BILL_METHOD | 11 |
| Creación de la tabla ODS_FACTURAS: | 13 |
| Relación entre la tabla de Facturas y la de clientes | 16 |
| Creación del modelo de LLAMADAS. | 18 |
| Creación de la tabla de dimensiones DIM_SERVICE | 18 |
| Creación de la tabla de dimensiones DIM_AGENT | 19 |
| Inserción de datos en la tabla de dimensiones DIM_AGENT | 20 |
| Creación de la tabla de hechos: OSD_LLAMADAS | 21 |
| Creación de las Foreign Keys | 22 |
| Inserción de datos en la tabla de hechos | 23 |
| Relación entre la tabla ODS_LLAMADAS y el resto de tablas de hechos. | 25 |
| Diagrama de Entidad-Relación de la Base de datos ODS | 26 |
| Tablas de OSD y sus número de filas | 27 |
| Tabla Ciudades-Estado | 28 |
| Separación del campo DE_DIRECCION en NOMBRE_VIA y NUM_VIA | 28 |
| DATA MANAGEMENT | 30 |
| DATA QUALITY: | 30 |
| MASTER DATA: | 31 |
| DATA MODELING & DESIGN: | 31 |
| Propuesta de mejora de diseño | 32 |
| Reglas para la creación y mantenimiento del DWH | 34 |

■ Primera parte

- Hacer el estudio de:
 - a. STG_FACTURAS_FCT con conclusiones
 - b. STG_CONTACTOS_IVR con conclusiones
- Crear el modelo, las tablas, las FK y poblar los modelos:
 - a. FACTURAS
El diagrama planteado es un modo, pero puedes plantear uno diferente
 - b. LLAMADAS
El diagrama planteado es un modo, pero puedes plantear uno diferente

Estudio de las tablas de STG

Estudio de la tabla STAGE.STG_FACTURAS_FCT.

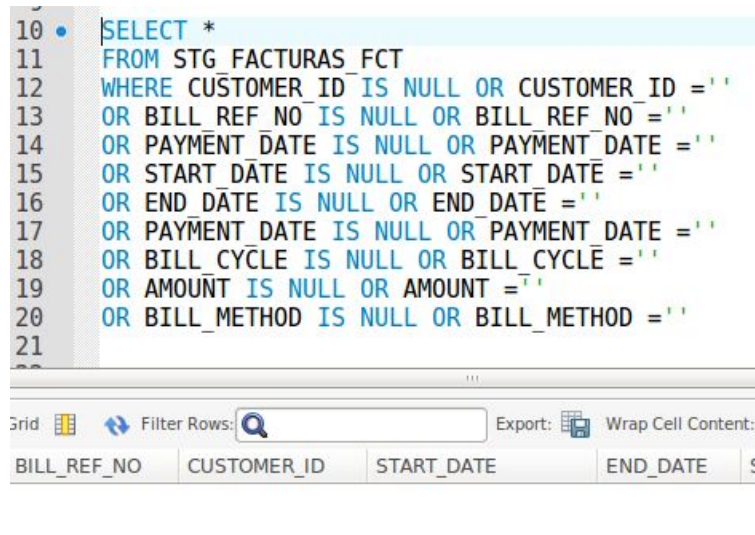
| # | BIN_REF_NO | CUSTOMER_ID | START_DATE | END_DATE | STATEMENT_DATE | PAYMENT_DATE | BILL_CYCLE |
|---|------------|-------------|---------------------|------------|---------------------|---------------------|------------|
| 1 | 572635234 | 10001 | 2016-01-01 00:00:00 | 2016-02-01 | 2016-01-03 00:00:00 | 2016-01-11 00:00:00 | M01 |
| 2 | 572635235 | 10002 | 2016-01-15 00:00:00 | 2016-02-01 | 2016-01-17 00:00:00 | 2016-01-26 00:00:00 | M15 |
| 3 | 572635236 | 10003 | 2016-01-01 00:00:00 | 2016-02-01 | 2016-01-03 00:00:00 | 2016-01-06 00:00:00 | M01 |
| 4 | 572635237 | 10004 | 2016-01-01 00:00:00 | 2016-02-01 | 2016-01-03 00:00:00 | 2016-01-14 00:00:00 | M01 |

Un vistazo rápido sobre esta tabla nos permite comprobar lo siguiente:

Los campos START_DATE, STATEMENT_DATE y PAYMENT_DATE tienen estructura de Timestamp, aunque examinando la tabla de STAGE, son del tipo varchar(512), además no utilizan la parte de tiempo. En cambio el campo END_DATE tiene un formato de fecha, aunque también viene con un tipo de varchar(512). Por lo tanto, será necesaria una unificación de tipos a DATE en todos estos campos, lo que nos facilitará cualquier consulta por rango de fechas.

Vamos a buscar valores nulos o blancos en los campos:

```
10 • SELECT *
11 FROM STG_FACTURAS_FCT
12 WHERE CUSTOMER_ID IS NULL OR CUSTOMER_ID = ''
13 OR BILL_REF_NO IS NULL OR BILL_REF_NO = ''
14 OR PAYMENT_DATE IS NULL OR PAYMENT_DATE = ''
15 OR START_DATE IS NULL OR START_DATE = ''
16 OR END_DATE IS NULL OR END_DATE = ''
17 OR PAYMENT_DATE IS NULL OR PAYMENT_DATE = ''
18 OR BILL_CYCLE IS NULL OR BILL_CYCLE = ''
19 OR AMOUNT IS NULL OR AMOUNT = ''
20 OR BILL_METHOD IS NULL OR BILL_METHOD = ''
21
```



The screenshot shows a database query interface. The top part displays a SQL query with line numbers 10 through 21. The query is a SELECT statement with an asterisk, followed by a FROM clause for 'STG_FACTURAS_FCT'. The WHERE clause contains a series of OR conditions checking for NULL or empty string values for various fields: CUSTOMER_ID, BILL_REF_NO, PAYMENT_DATE, START_DATE, END_DATE, and BILL_CYCLE. Below the query, there is a table header with columns: BILL_REF_NO, CUSTOMER_ID, START_DATE, END_DATE, and S.

La consulta no arroja ningún resultado.

Comprobamos la clave primaria de la tabla:

```
SELECT BILL_REF_NO, COUNT(*)
FROM STG_FACTURAS_FCT
GROUP BY BILL_REF_NO
HAVING COUNT(*) > 1
;
```

No devolviendo duplicados, por lo que confirmamos que el campo **BILL_REF_NO** es **PRIMARY KEY** de la tabla.

Visualizamos la estructura de la tabla:

```
Table: STG_FACTURAS_FCT
Columns:
BILL_REF_NO  varchar(512)
CUSTOMER_ID  varchar(512)
START_DATE   varchar(512)
END_DATE     varchar(512)
STATEMENT_DATE varchar(512)
PAYMENT_DATE  varchar(512)
BILL_CYCLE   varchar(512)
AMOUNT       varchar(512)
BILL_METHOD   varchar(512)
```

Observamos que todos los campos vienen como de tipo texto con longitud fija, por lo que tendremos que estudiar su conversión al formato adecuado.

```

31 • SELECT LENGTH(MAX(BILL_REF_NO))
32 FROM STG_FACTURAS_FCT
33

```

| # | LENGTH(MAX(BILL_REF_NO)) |
|---|--------------------------|
| 1 | 9 |

En el campo **BILL_REF_NO** observamos que siempre es un numérico, de hecho el sufijo **_NO** en el nombre del campo nos da una pista que confirma la anterior afirmación.

Examinamos el valor máximo y mínimo que tienen y decidimos hacer una conversión de tipos a decimal(12,0).

Importante a tener en cuenta que no lo podemos poner como INT, ya que la longitud máxima en el caso del unsigned es inferior a la deseada. Podemos usar, pues BIGINT o DECIMAL para la conversión.

El mismo procedimiento empleamos para el campo CUSTOMER_ID:

```

31 • SELECT LENGTH(MAX(CUSTOMER_ID))
32 FROM STG_FACTURAS_FCT
33

```

| # | LENGTH(MAX(CUSTOMER_ID)) |
|---|--------------------------|
| 1 | 5 |

En este caso, con un entero UNSIGNED tendremos suficiente.

Seguimos el análisis con el siguiente campo: **BILL_CYCLE**, viendo que se compone de dos valores:

```

28 SELECT DISTINCT BILL_CYCLE
29 FROM STG_FACTURAS_FCT;
30

```

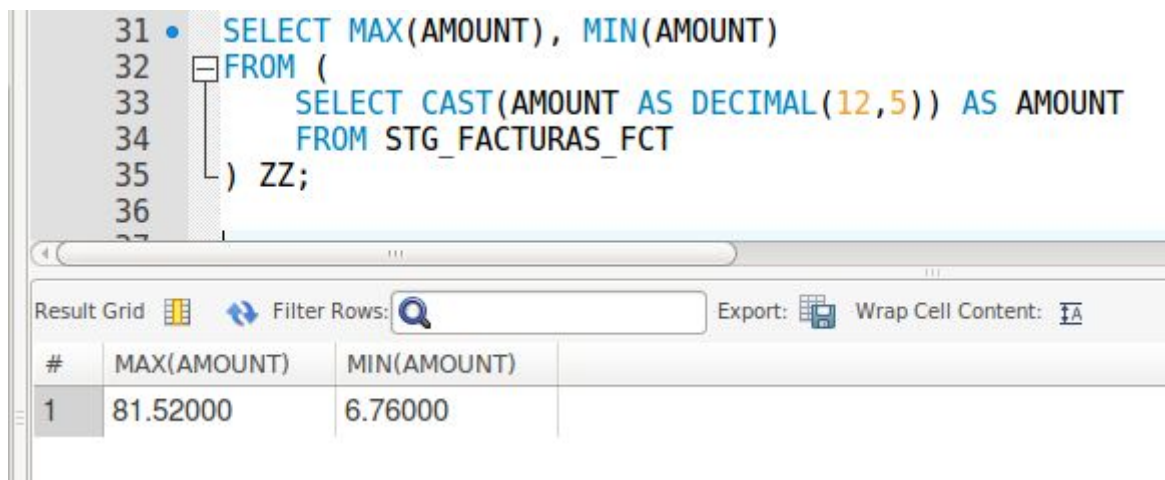
| # | BILL_CYCLE |
|---|------------|
| 1 | M01 |
| 2 | M15 |

Nos podríamos plantear la conveniencia de crear una tabla de dimensiones para almacenar estos valores.

Al ser solamente dos no aportaría mejora de rendimiento, pero las buenas prácticas aconsejan hacerlo.

Examinamos el campo **AMOUNT**:

Aunque el formato de este campo es de texto, examinando un rango de valores vemos que realmente es numérico. Anteriormente hemos verificado que no existen nulos que pudieran dar errores en las operaciones de cálculo. No obstante, antes de hacer la conversión de tipos es importante comprobar el rango de valores que tiene.



```
31 • SELECT MAX(AMOUNT), MIN(AMOUNT)
32 FROM (
33     SELECT CAST(AMOUNT AS DECIMAL(12,5)) AS AMOUNT
34     FROM STG_FACTURAS_FCT
35 ) ZZ;
```

| # | MAX(AMOUNT) | MIN(AMOUNT) |
|---|-------------|-------------|
| 1 | 81.52000 | 6.76000 |

No veo valores negativos, con valores no muy grandes y dos decimales.

Escogemos DECIMAL(6,2) como tipo para el campo.

El último campo BILL_METHOD, se compone exclusivamente de tres valores, sin valores nulos. En este caso es muy recomendable la creación de una tabla de dimensiones con los métodos de pago.

| # | BILL_METHOD |
|---|---------------|
| 1 | Direct debit |
| 2 | Credit Card |
| 3 | Check payment |

Como con el resto de tablas, pasaremos a mayúsculas, eliminando los espacios al inicio y final, e introduciendo los valores de No aplica y desconocido a la lista.

Estudio de la tabla STG_CONTACTOS_IVR:

| # | ID | PHONE_NUMBER | START_DATETIME | END_DATETIME | SERVICE | FLG_TRANSFER | AGENT |
|---|----|--------------|----------------------------|----------------------------|------------------|--------------|---------------|
| 1 | 0 | 4075503542 | 2017-09-15 11:47:22.980343 | 2017-09-15 11:49:19.980343 | COLLECTIONS | False | jbradley |
| 2 | 1 | 9722617992 | 2017-09-15 11:48:16.980343 | 2017-09-15 11:54:01.980343 | CUSTOMER SERVICE | False | douglaspace |
| 3 | 2 | 4021945400 | 2017-09-15 11:49:05.980343 | 2017-09-15 11:56:44.980343 | SALES | False | |
| 4 | 3 | 2087246547 | 2017-09-15 11:49:40.980343 | 2017-09-15 11:52:07.980343 | SALES | False | scottmurphy |
| 5 | 4 | 9413788352 | 2017-09-15 11:50:38.980343 | 2017-09-15 11:56:26.980343 | CLAIMS | True | nlove |
| 6 | 4 | 8162663799 | 2017-09-15 11:56:26.980343 | 2017-09-15 12:05:51.980343 | SALES | False | christiebrown |

```
Table: STG_CONTACTOS_IVR
Columns:
ID      varchar(512)
PHONE_NUMBER  varchar(512)
START_DATETIME  varchar(512)
END_DATETIME    varchar(512)
SERVICE        varchar(512)
FLG_TRANSFER    varchar(512)
AGENT           varchar(512)
```

Visualizando la estructura de la tabla, vemos que al igual que nos pasaba con la anterior todos los campos vienen en formato de texto, por lo que es necesario realizar una conversión de tipos

Primeramente vamos a buscar valores nulos o en blanco:

```

3 SELECT *
4 FROM STG_CONTACTOS_IVR
5 WHERE ID IS NULL OR ID = ''
6 OR PHONE_NUMBER IS NULL OR PHONE_NUMBER = ''
7 OR START_DATETIME IS NULL OR START_DATETIME = ''
8 OR END_DATETIME IS NULL OR END_DATETIME = ''
9 OR SERVICE IS NULL OR SERVICE = ''
10 OR FLG_TRANSFER IS NULL OR FLG_TRANSFER = ''
11 OR AGENT IS NULL OR AGENT = ''
12 ;

```

| # | ID | PHONE_NUMBER | START_DATETIME | END_DATETIME | SERVICE | FLG_TRANSFER | AGENT |
|---|----|--------------|----------------------------|----------------------------|------------------|--------------|------------|
| 1 | 2 | 4021945400 | 2017-09-15 11:49:05.980343 | 2017-09-15 11:56:44.980343 | SALES | False | |
| 2 | 9 | | 2017-09-15 12:10:01.980343 | | CLAIMS | False | fberry |
| 3 | 16 | 7049848236 | 2017-09-15 11:57:44.980343 | 2017-09-15 12:05:52.980343 | | False | kochdeanna |
| 4 | 24 | 2537327573 | 2017-09-15 12:02:30.980343 | 2017-09-15 12:08:43.980343 | CUSTOMER SERVICE | True | |
| 5 | 26 | | 2017-09-15 12:04:05.980343 | 2017-09-15 12:12:56.980343 | CLAIMS | False | tthompson |
| 6 | 35 | | 2017-09-15 12:10:10.980343 | 2017-09-15 12:13:26.980343 | SALES | False | amanda22 |

Tenemos valores en blanco en los campos **PHONE_NUMBER**, **END_DATETIME**, **SERVICE** y **AGENT**. Es importante tenerlo en cuenta para forzar a valores conocidos.

El primer campo de la tabla, el campo ID es numérico, y se observa por la captura contiene valores duplicados. El nombre del campo no aporta nada, por lo que en la tabla final de ODS lo cambiaremos por otro nombre más identificativo, por ejemplo **CALL_ID**.

Para el campo **PHONE_NUMBER** podríamos pensar en pasarlo a numérico, pero lo dejamos como literal, ya que se puede hacer uso de las extensiones de país (ej. +034). Además si lo pasamos a numérico perderíamos los 0 iniciales, que a veces se utilizan para extensiones.

Los campos **START_DATETIME** y **END_DATETIME** son de tipo DATETIME. Es necesario el cambio de tipo. Por otro lado, también es conveniente cambiar el nombre de las columnas por otro más descriptivo:

START_CALL_DT y END_CALL_DT

Para enriquecer la tabla y mejorar las consultas, añadiremos dos nuevas columnas a la tabla: **START_CALL_DATE** y **END_CALL_DATE**, en donde hemos transformado los datetime en fechas (no utilizo la abreviatura “_D” ya que se puede confundir con Día).

También enriquecemos la tabla con el campo **TIME_MINUTES** que nos da la diferencia entre el inicio y el final de la llamada en minutos.

| # | SERVICE |
|---|------------------|
| 1 | COLLECTIONS |
| 2 | CUSTOMER SERVICE |
| 3 | SALES |
| 4 | CLAIMS |
| 5 | SUPPORT |
| 6 | |
| 7 | FINANCIALS |

STG CONTACTOS IVR 19

Si miramos el campo **SERVICE** y sus valores vemos que puede ser interesante crear una tabla maestra con sus valores y almacenar en la tabla de hechos sólo el Identificador del servicio utilizado. Tendré que tener en cuenta los valores tipo blanco que arrastra.

| # | AGENT | |
|---|---------------|--|
| 1 | jbradley | |
| 2 | douglaspace | |
| 3 | | |
| 4 | scottmurphy | |
| 5 | nlove | |
| 6 | christiebrown | |
| 7 | | |

FLG_TRANSFER es un campo boolean con dos valores.

En este caso, es recomendable pasar este tipo de campo a formato numérico 0-> FALSE, 1->TRUE, que permita un rápido conteo de valores mediante suma directa.

Con el último campo **AGENT**, haremos lo mismo que con SERVICE, creando una tabla maestra de AGENTES y dejando en la tabla de hechos el identificador.

A continuación vamos a buscar las primary keys de la tabla:

```
SELECT ID, START_DATETIME, COUNT(*)
FROM STG_CONTACTOS_IVR
GROUP BY ID, START_DATETIME
HAVING COUNT(*)>1
```

El conjunto que identifica la llamada, junto con la fecha/hora de inicio, marca las PK de la tabla.

El funcionamiento de esta tabla se puede ver en la captura siguiente:

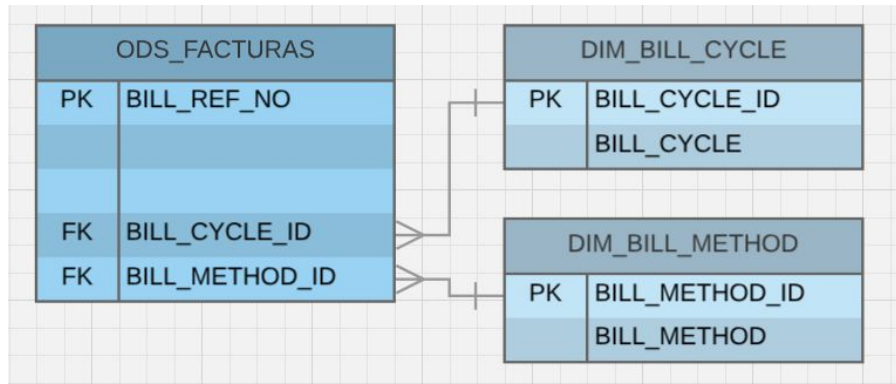
| # | ID | PHONE_NUMBER | START_DATETIME | END_DATETIME | SERVICE | FLG_TRANSFER | AGENT |
|---|--------|--------------|----------------------------|----------------------------|---------|--------------|-------------|
| 1 | 100085 | 8131779555 | 2017-10-28 21:41:24.980343 | 2017-10-28 21:51:19.980343 | CLAIMS | True | josephbrown |
| 2 | 100085 | 5631022375 | 2017-10-28 21:51:19.980343 | 2017-10-28 21:52:48.980343 | CLAIMS | True | hcardenas |
| 3 | 100085 | | 2017-10-28 21:59:25.980343 | 2017-10-28 22:04:40.980343 | CLAIMS | False | trevormay |

Una determinada reclamación (SERVICE=CLAIMS con ID=100085), pasa por diferentes agentes (AGENT), reflejando el campo **FLG_TRANSFER** si ha sido transferida. El comportamiento cronológico cuadra con los campos DATETIME de inicio y final. El campo **PHONE_NUMBER**, parece que indica el número de teléfono entrante, ya que en el primer caso parece un número externo, mientras que el segundo parece un número interno. No obstante, este punto se tendría que verificar con negocio.

Creación de modelos en OSD

Creación del Modelo de FACTURAS:

El modelo entre entidades quedaría de la siguiente manera.



Primero vamos a crear las dos tablas de dimensiones:

Creación de la tabla DIM_BILL_CYCLE

```
/* CREACION DE LAS TABLAS DE DIMENSIONES */

/* DIM_BILL_CYCLE TABLE */

SET FOREIGN_KEY_CHECKS=0;

DROP TABLE IF EXISTS ODS.DIM_BILL_CYCLE;

CREATE TABLE ODS.DIM_BILL_CYCLE (
    BILL_CYCLE_ID INT PRIMARY KEY -- SIN AUTO INCREMENT
    , BILL_CYCLE NVARCHAR(512) NULL
    , DATA_INSERT DATETIME NULL
    , DATA_UPDATE DATETIME NULL
);
```

Vamos a llenar la tabla poniendo en mayúsculas los valores y eliminando los espacios por delante y detrás si los hubiera.

```
INSERT INTO ODS.DIM_BILL_CYCLE (
    SELECT
        @i := @i + 1 AS BILL_CYCLE_ID
    , A.*
    , NOW()
    , NOW()
    FROM (
        SELECT DISTINCT UPPER(TRIM(BILL_CYCLE)) AS BILL_CYCLE
        FROM STAGE.STG_FACTURAS_FCT
    ) A,
    (SELECT @i:=0) AS INIT
);

INSERT INTO ODS.DIM_BILL_CYCLE VALUES(-1,'DESCONOCIDO', NOW(), NOW());
INSERT INTO ODS.DIM_BILL_CYCLE VALUES(-2,'NO APLICA', NOW(), NOW());

ANALYZE TABLE ODS.DIM_BILL_CYCLE;
```

Creación de la tabla DIM_BILL_METHOD

```
SET FOREIGN_KEY_CHECKS=0;

DROP TABLE IF EXISTS ODS.DIM_BILL_METHOD ;

CREATE TABLE ODS.DIM_BILL_METHOD (
    BILL_METHOD_ID INT PRIMARY KEY
    , BILL_METHOD NVARCHAR(512) NULL
    , DATA_INSERT DATETIME NULL
    , DATA_UPDATE DATETIME NULL
);
```

Como en el caso anterior, eliminamos espacios y ponemos en mayúsculas.

```
INSERT INTO ODS.DIM_BILL_METHOD (
    SELECT
        @i := @i +1 AS BILL_METHOD_ID
        , A.*
        , NOW()
        , NOW()
    FROM (
        SELECT DISTINCT UPPER(TRIM(BILL_METHOD))
        FROM STAGE.STG_FACTURAS_FCT
    ) A, (SELECT @i:=0) AS INIT
);

INSERT INTO ODS.DIM_BILL_METHOD VALUES(-1, 'DESCONOCIDO', NOW(), NOW());
INSERT INTO ODS.DIM_BILL_METHOD VALUES(-2, 'NO APLICA', NOW(), NOW());

ANALYZE TABLE ODS.DIM_BILL_METHOD;
```

Creación de la tabla ODS_FACTURAS:

Poniendo en práctica todo lo observado en el punto 1 pasamos a crear el script de creación de la nueva tabla:

```
/* CREACIÓN DE LA TABLA ODS_FACTURAS */

DROP TABLE IF EXISTS ODS_FACTURAS;

CREATE TABLE ODS_FACTURAS (
    BILL_REF_NO DECIMAL(12,0) NOT NULL PRIMARY KEY
    , CUSTOMER_ID INT UNSIGNED NULL
    , START_DATE DATE NULL
    , END_DATE DATE NULL
    , STATEMENT DATE NULL
    , PAYMENT_DATE DATE NULL
    , BILL_CYCLE_ID INT NULL
    , BILL_METHOD_ID INT NULL
    , AMOUNT DECIMAL(6,2) NULL
    , INSERT_DATE timestamp NULL
    , UPDATE_DATE timestamp NULL
);
```

Seguidamente crearemos las relaciones entre la tabla (FK) de hechos y las dos de dimensiones que hemos creado:

```
ALTER TABLE ODS.ODS_FACTURAS
    ADD INDEX idx_ODS_FACTURAS_by_BILL_CYCLE_ID (BILL_CYCLE_ID ASC)
    , ADD CONSTRAINT fk_DIM_BILL_CYCLE
        FOREIGN KEY (BILL_CYCLE_ID)
        REFERENCES ODS.DIM_BILL_CYCLE (BILL_CYCLE_ID)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
;

ALTER TABLE ODS.ODS_FACTURAS
    ADD INDEX idx_ODS_FACTURAS_by_BILL_METHOD_ID (BILL_METHOD_ID
ASC)
    , ADD CONSTRAINT fk_DIM_BILL_METHOD
        FOREIGN KEY (BILL_METHOD_ID)
        REFERENCES ODS.DIM_BILL_METHOD(BILL_METHOD_ID)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
;
```

Realizamos la comprobación de los índices en la tabla de hechos:

```
78 • SHOW INDEX FROM ODS.ODS_FACTURAS;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality |
|--------------|------------|------------------------------------|--------------|----------------|-----------|-------------|
| ODS_FACTURAS | 0 | PRIMARY | 1 | BILL_REF_NO | A | 0 |
| ODS_FACTURAS | 1 | idx_ODS_FACTURAS_by_BILL_CYCLE_ID | 1 | BILL_CYCLE_ID | A | 0 |
| ODS_FACTURAS | 1 | idx_ODS_FACTURAS_by_BILL_METHOD_ID | 1 | BILL_METHOD_ID | A | 0 |

Ahora tenemos que relacionar nuestro modelo en estrella con el resto de tablas.

Hay una relación evidente entre la tabla de FACTURAS y la de CLIENTES.

No nos interesa que un borrado de la tabla madre nos borre el registro de facturas, por lo que lo indicamos cuando creamos el índice (SET NULL).

En cambio, ante una actualización del registro de la tabla madre, es importante actualizar el ID para no perder la relación (Actualización en cascada).

```
/* FOREIGN KEY CREATION BETWEEN FACTURAS AND CLIENTES */
```

```
ALTER TABLE ODS.ODS_FACTURAS
  ADD INDEX ix_ODS_FACTURAS_by_CUSTOMER_ID (CUSTOMER_ID ASC)
, ADD CONSTRAINT fk_ODS_CLIENTES
  FOREIGN KEY (CUSTOMER_ID)
    REFERENCES ODS.ODS_HC_CLIENTES (ID_CLIENTE)
  ON DELETE SET NULL
  ON UPDATE CASCADE
;
```

Quedando los índices de esta tabla como siguen:

| # | Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Su |
|---|--------------|------------|------------------------------------|--------------|----------------|-----------|-------------|------|
| 1 | ODS_FACTURAS | 0 | PRIMARY | 1 | BILL_REF_NO | A | 418446 | NULL |
| 2 | ODS_FACTURAS | 1 | idx_ODS_FACTURAS_by_BILL_CYCLE_ID | 1 | BILL_CYCLE_ID | A | 1 | NULL |
| 3 | ODS_FACTURAS | 1 | idx_ODS_FACTURAS_by_BILL_METHOD_ID | 1 | BILL_METHOD_ID | A | 2 | NULL |
| 4 | ODS_FACTURAS | 1 | ix_ODS_FACTURAS_by_CUSTOMER_ID | 1 | CUSTOMER_ID | A | 20230 | NULL |

Ahora vamos a rellenar la tabla con los datos que vienen de STAGE:

```
-- POPULATE THE ODS TABLE
INSERT INTO ODS_FACTURAS (

SELECT
    BILL_REF_NO
    , CUSTOMER_ID
    , START_DATE
    , END_DATE
    , STATEMENT_DATE
    , PAYMENT_DATE
    , BILL_CYCLE_ID
--    , A.BILL_CYCLE
    , BILL_METHOD_ID
--    , A.BILL_METHOD
    , AMOUNT
    , now()
    , now()
FROM (
SELECT
    CAST(BILL_REF_NO AS DECIMAL (12,0) ) AS BILL_REF_NO
    , CAST(CUSTOMER_ID AS UNSIGNED) AS CUSTOMER_ID
    , CAST(START_DATE AS date) AS START_DATE
    , CAST(END_DATE AS DATE) AS END_DATE
    , CAST(STATEMENT_DATE AS DATE) AS STATEMENT_DATE
    , CAST(PAYMENT_DATE AS DATE) AS PAYMENT_DATE
    , BILL_CYCLE
    , BILL_METHOD
    , CAST(AMOUNT AS DECIMAL(6,2)) AS AMOUNT
FROM STAGE.STG_FACTURAS_FCT
)A INNER JOIN ODS.DIM_BILL_CYCLE B
    ON (A.BILL_CYCLE=B.BILL_CYCLE)
    INNER JOIN ODS.DIM_BILL_METHOD C
        ON (A.BILL_METHOD = C.BILL_METHOD)
-- WHERE
--    (BILL_METHOD_ID IS NULL OR
--        BILL_CYCLE_ID IS NULL)
)
;
```

Antes de insertar en la tabla la selección hemos realizado la comprobación para ver que los joins con las nuevas tablas de dimensiones no dejaban ningún nulo. Una vez comprobado, se han comentado y se ha aplicado el insert.

Para comprobar que todo ha correcto, comparamos el número de líneas insertadas respecto del origen:

```
90 • SELECT
91     SUM(NUM_REG_STG_FACTURAS) AS NUM_REG_STG_FACTURAS
92     , SUM(NUM_REG_ODS_FACTURAS) AS NUM_REG_ODS_FACTURAS
93 FROM (
94     SELECT
95         COUNT(*) AS NUM_REG_STG_FACTURAS
96         , 0 AS NUM_REG_ODS_FACTURAS
97     FROM STAGE.STG_FACTURAS_FCT
98     UNION ALL
99     SELECT
100         0 AS NUM_REG_STG_FACTURAS
101         , COUNT(*) AS NUM_REG_ODS_FACTURAS
102     FROM ODS_FACTURAS
103 )ZZ
```

Result Grid

| # | NUM_REG_STG_FACTURAS | NUM_REG_ODS_FACTURAS |
|---|----------------------|----------------------|
| 1 | 420000 | 420000 |

Relación entre la tabla de Facturas y la de clientes

Como siguiente paso se trata de relacionar nuestro diagrama en estrella de Facturas con la tabla de clientes. Con esa relación podremos responder a cuestiones sobre el número de facturas asociado a un determinado cliente (por nombre y apellido) o a estadísticas de facturación por región, por ejemplo.

La siguiente sentencia crea la relación entre las tablas:

```
/* FOREIGN KEY CREATION BETWEEN FACTURAS AND CLIENTES */

ALTER TABLE ODS.ODS_FACTURAS
    ADD INDEX ix_ODS_FACTURAS_by_CUSTOMER_ID (CUSTOMER_ID ASC)
    , ADD CONSTRAINT fk_ODS_CLIENTES
        FOREIGN KEY (CUSTOMER_ID)
            REFERENCES ODS.ODS_HC_CLIENTES (ID_CLIENTE)
        ON DELETE SET NULL
        ON UPDATE CASCADE
;
```

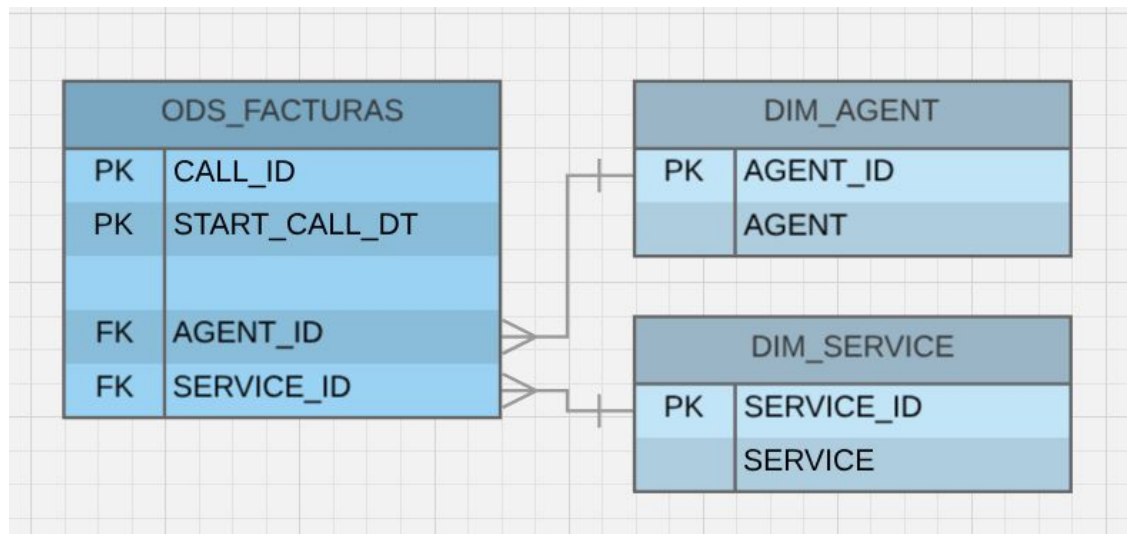
Al relacionar las dos tablas de hecho, hay que tener mucho cuidado al definir la política de borrado y actualización.

Para el borrado, en este caso he optado por poner a null el campo en el caso de borrado, ya que interesa mantener el registro de Factura. Esta acción marca también al registro.

Para la actualización, en cambio, interesa que si cambia el identificador, este cambio se traslade a la tabla de Facturas, para no perder la relación con el cliente. Por eso activamos la actualización en Cascada.

Creación del modelo de LLAMADAS.

Como en el anterior caso, empezamos con la creación de las tablas de dimensiones asociadas: **DIM_AGENT** y **DIM_SERVICE**.



En el diagrama de relaciones anterior podemos ver las tablas a generar y sus relaciones.

Creación de la tabla de dimensiones DIM_SERVICE

```
/* CREATION OF DIM_SERVICE TABLE */

DROP TABLE IF EXISTS ODS.DIM_SERVICE;

CREATE TABLE ODS.DIM_SERVICE (
  SERVICE_ID INT NOT NULL PRIMARY KEY
    , SERVICE NVARCHAR(512) NULL
    , DATETIME_INSERT DATETIME NULL
    , DATETIME_UPDATE DATETIME NULL
);

ANALYZE TABLE ODS.DIM_SERVICE;
```

Inserción de datos en la tabla de dimensiones **DIM_SERVICE**

```
INSERT INTO ODS.DIM_SERVICE (SERVICE_ID, SERVICE,
DATETIME_INSERT, DATETIME_UPDATE) (
SELECT
  @i := @i+1 AS SERVICE_ID
    , TRIM(UPPER(SERVICE))
    , NOW()
    , NOW()
FROM (
  SELECT DISTINCT SERVICE
  FROM STAGE.STG_CONTACTOS_IVR
  WHERE SERVICE !=''
)A, (SELECT @i := 0) INIT
);

INSERT INTO ODS.DIM_SERVICE VALUES(-1, 'DESCONOCIDO', NOW(),
NOW());
INSERT INTO ODS.DIM_SERVICE VALUES(-2, 'NO APLICA', NOW(),
NOW());
```

Creación de la tabla de dimensiones **DIM_AGENT**

```
DROP TABLE IF EXISTS ODS.DIM_AGENT;

CREATE TABLE ODS.DIM_AGENT (
  AGENT_ID INT NOT NULL PRIMARY KEY
    , AGENT NVARCHAR(512) NULL
    , DATETIME_INSERT DATETIME NULL
    , DATETIME_UPDATE DATETIME NULL
);

ANALYZE TABLE ODS.DIM_AGENT;
```

Insertión de datos en la tabla de dimensiones DIM_AGENT

El procedimiento a seguir es el mismo que se ha empleado para insertar datos en el resto de Dimensiones. Normalizamos el campo texto poniéndolo en mayúsculas y eliminando los espacios en blanco delante y detrás, si los hubiera.

```
INSERT ODS.DIM_AGENT (AGENT_ID, AGENT, DATETIME_INSERT,
DATETIME_UPDATE) (
SELECT
    @i := @i +1 AS AGENT_ID
    , A.*
    , NOW()
    , NOW()
FROM (
SELECT DISTINCT TRIM(UPPER(AGENT))
FROM STAGE.STG_CONTACTOS_IVR
WHERE AGENT!=' '
) A, (SELECT @i := 0) INIT
);

INSERT INTO ODS.DIM_AGENT VALUES (-1, 'DESCONOCIDO', NOW(),
NOW());
INSERT INTO ODS.DIM_AGENT VALUES (-2, 'NO APLICA', NOW(),
NOW());
```

Creación de la tabla de hechos: OSD_LLAMADAS

```
/* CREATION OF OSD_LLAMADAS */

DROP TABLE IF EXISTS ODS.ODS_LLAMADAS;

CREATE TABLE ODS.ODS_LLAMADAS (
  CALL_ID INT UNSIGNED NOT NULL
    , PHONE_NUMBER NVARCHAR(15) NULL
    , START_CALL_DT DATETIME NOT NULL
    , END_CALL_DT DATETIME NULL
    , START_CALL_DATE DATE NULL
    , END_CALL_DATE DATE NULL
    , TIME_MINUTES DECIMAL (10,2) NULL
    , SERVICE_ID INT NULL
    , FLG_TRANSFER BIT NULL
    , AGENT_ID INT NULL
    , DATE_INSERT DATETIME NULL
    , DATE_UPDATE DATETIME NULL
    , PRIMARY KEY (CALL_ID, START_CALL_DT)
);

ANALYZE TABLE ODS.ODS_LLAMADAS;
```

Creación de las Foreign Keys

Vamos a crear las relaciones entre la tabla de hechos y las dos de dimensiones para generar el diagrama en estrella.

La estrategia de actualización y de borrado es la que se ha explicado anteriormente, dejando a nulo en caso de borrado y actualizando en cascada para no perder la referencia.

```
/* CREATION OF THE FOREIGN KEYS */

ALTER TABLE ODS.ODS_LLAMADAS
ADD INDEX ix_ODS_LLAMADAS_by_SERVICE_ID (SERVICE_ID ASC)
, ADD INDEX ix_ODS_LLAMADAS_by_AGENT_ID (AGENT_ID ASC)
, ADD CONSTRAINT fk_DIM_SERVICE
    FOREIGN KEY (SERVICE_ID)
    REFERENCES ODS.DIM_SERVICE (SERVICE_ID)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
, ADD CONSTRAINT fk_DIM_AGENT
    FOREIGN KEY (AGENT_ID)
    REFERENCES ODS.DIM_AGENT (AGENT_ID)
    ON DELETE SET NULL
    ON UPDATE CASCADE
;

SHOW INDEX FROM ODS.ODS_LLAMADAS;
SHOW CREATE TABLE ODS.ODS_LLAMADAS;
```

Insertión de datos en la tabla de hechos

```
INSERT INTO ODS.ODS_LLAMADAS (

SELECT
CALL_ID
    , PHONE_NUMBER
    , START_CALL_DT
    , END_CALL_DT
    , START_CALL_DATE
    , END_CALL_DATE
    , TIME_MINUTES
    , SERV.SERVICE_ID
--    , C.SERVICE
    , FLG_TRANSFER
    , AGENT.AGENT_ID
--    , C.AGENT
    , NOW()
    , NOW()
FROM (
SELECT
CALL_ID
    , PHONE_NUMBER
    , START_CALL_DT
    , END_CALL_DT
    , CAST(START_CALL_DT AS DATE) AS START_CALL_DATE
    , CAST(END_CALL_DT AS DATE) AS END_CALL_DATE
    , TIMEDIFF(MINUTE, START_CALL_DT, END_CALL_DT) AS
TIME_MINUTES
    , SERVICE
    , FLG_TRANSFER
    , AGENT
FROM (
SELECT
CALL_ID
    , CASE WHEN PHONE_NUMBER!='' THEN PHONE_NUMBER ELSE 'NO
APLICA' END AS PHONE_NUMBER
    , START_CALL_DT
    , END_CALL_DT
    , CASE WHEN SERVICE!='' THEN SERVICE ELSE 'NO APLICA' END AS
SERVICE
    , FLG_TRANSFER
    , CASE WHEN AGENT!= '' THEN AGENT ELSE 'NO APLICA' END AS
AGENT
FROM (
SELECT
CAST(ID AS SIGNED ) AS CALL_ID
    , CAST(PHONE_NUMBER AS NCHAR(15)) AS PHONE_NUMBER
    , str_to_date(START_DATETIME , '%Y-%m-%d %H:%i:%s.%f') AS
START_CALL_DT
```

```

, str_to_date( CASE WHEN END_DATETIME!='' THEN END_DATETIME
                ELSE '2099-01-01 00:00:00' END , '%Y-%m-%d %H:%i:%s.%f')
AS END_CALL_DT
, UPPER(TRIM(SERVICE)) AS SERVICE
, CASE WHEN UPPER(TRIM(FLG_TRANSFER)) = 'TRUE' THEN 1 ELSE 0
END AS FLG_TRANSFER
, UPPER(TRIM(AGENT)) AS AGENT
FROM STG_CONTACTOS_IVR
) A
) B
) C INNER JOIN ODS.DIM_SERVICE SERV
ON ( C.SERVICE= SERV.SERVICE)
    INNER JOIN ODS.DIM_AGENT AGENT
    ON (C.AGENT=AGENT.AGENT)
);

```

Comprobación de carga

```

SELECT
SUM(NUM_REG_STG_CONTACTOS) AS NUM_REG_STG_CONTACTOS
, SUM(NUM_REG_ODS_LLAMADAS) AS NUM_REG_ODS_LLAMADAS
FROM (
SELECT
COUNT(*) AS NUM_REG_STG_CONTACTOS
, 0 AS NUM_REG_ODS_LLAMADAS
FROM STAGE.STG_CONTACTOS_IVR
UNION ALL
SELECT
0 AS NUM_REG_STG_CONTACTOS
, COUNT(*) AS NUM_REG_ODS_LLAMADAS
FROM ODS.ODS_LLAMADAS
)ZZ
;

```

| # | NUM_REG_STG_CONTACTOS | NUM_REG_ODS_LLAMADAS |
|---|-----------------------|----------------------|
| 1 | 202717 | 202717 |

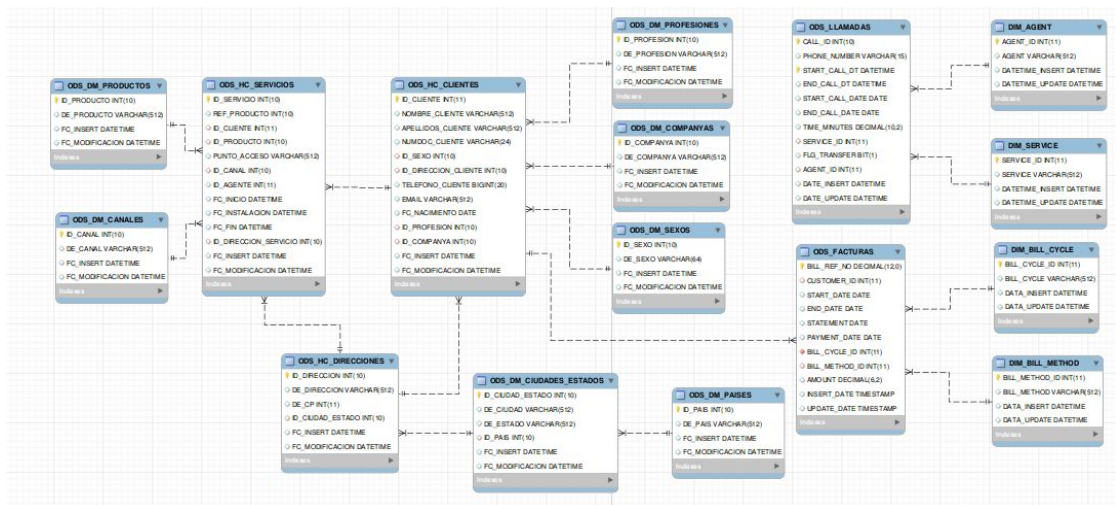
Relación entre la tabla ODS_LLAMADAS y el resto de tablas de hechos.

Se ha comprobado la existencia de alguna relación posible, pero no se ha identificado. Como punto de mejora, se tendría que identificar hablando con negocio la manera de poder identificar a los clientes entrantes. Si son reclamaciones, son clientes ya de la compañía y si son ventas, pueden ser de la compañía o nuevos. En el segundo caso, para realizar la venta se tendrá que entrar sus datos por lo que hay que identificar el lugar en donde se guardan.

Segunda parte

- Adjunta tu diagrama de ODS completo con el número de registros que contiene cada tabla
- ¿Por qué en el modelo de DIRECCIONES deje en la misma tabla las CIUDADES y los ESTADOS y no los separe en dos tablas distintas para ser más estricta con la jerarquía:
PAIS → ESTADOS → CIUDADES → DIRECCIONES
- Separar el campo DE_DIRECCION de la tabla de direcciones en dos campos: NOMBRE_VIA y NUM_VIA

Diagrama de Entidad-Relación de la Base de datos ODS



Tablas de OSD y sus número de filas

| Table Type | Table Name | Row Number |
|------------|-------------------------|------------|
| Dimensión | DIM_AGENT | 595 |
| Dimensión | DIM_BILL_CYCLE | 3 |
| Dimensión | DIM_BILL_METHOD | 4 |
| Dimensión | DIM_SERVICE | 8 |
| Dimensión | ODS_DM_CANALES | 6 |
| Dimensión | ODS_DM_CIUDADES_ESTADOS | 234 |
| Dimensión | ODS_DM_COMPANYAS | 385 |
| Dimensión | ODS_DM_PAISES | 3 |
| Dimensión | ODS_DM_PRODUCTOS | 8 |
| Dimensión | ODS_DM_PROFESIONES | 197 |
| Dimensión | ODS_DM_SEXOS | 4 |
| Hechos | ODS_FACTURAS | 420000 |
| Hechos | ODS_HC_CLIENTES | 17561 |
| Hechos | ODS_HC_DIRECCIONES | 113663 |
| Hechos | ODS_HC_SERVICIOS | 78495 |
| Hechos | ODS_LLAMADAS | 202717 |

Tabla Ciudades-Estado

La tabla ciudad-Estado se podría haber construido como dos tablas por separado, con la precaución de introducir un identificador de estado dentro de la tabla ciudad. Al introducir las dos dimensiones en una misma tabla se realiza lo que se conoce como un aplanamiento de jerarquía. Tiene la ventaja de que

Separación del campo DE_DIRECCION en NOMBRE_VIA y NUM_VIA

El campo DE_DIRECCION se extrae de la tabla
STAGE.STG_CLIENTES_CRM.ADDRESS:

| DOC | GENDER | CITY | ADDRESS | POSTAL_CODE | STATE | COUNTRY |
|-----|--------|-------------|------------------------|-------------|------------|---------|
| 5 | Male | Las Vegas | 68 Kennedy Pass | 89130 | Nevada | US |
| 3 | Female | Reno | 8014 Esch Alley | 89510 | Nevada | US |
| 4 | Male | Las Vegas | 68 Delaware Drive | 89193 | Nevada | US |
| 4 | Female | Sacramento | 5393 Stang Plaza | 94297 | California | US |
| 2 | Male | Los Angeles | 97744 Montana Park | 90035 | California | US |
| 3 | Female | Phoenix | 7070 Meadow Vale Trail | 85062 | Arizona | US |

Vemos que su estructura sigue el siguiente patrón: NUM + Espacio + DIRECCION.

Utilizando una combinación de SUBSTRING, SUBSTRING_INDEX y LENGTH obtenemos el resultado.

| # | ADDRESS | substring_index(ADDRESS, ' ', 1) | substring(ADDRESS FROM length(substring_index(ADDRESS, ' ', 1)) + 1) |
|---|------------------------|----------------------------------|--|
| 1 | 68 Kennedy Pass | 68 | Kennedy Pass |
| 2 | 8014 Esch Alley | 8014 | Esch Alley |
| 3 | 68 Delaware Drive | 68 | Delaware Drive |
| 4 | 5393 Stang Plaza | 5393 | Stang Plaza |
| 5 | 97744 Montana Park | 97744 | Montana Park |
| 6 | 7070 Meadow Vale Trail | 7070 | Meadow Vale Trail |
| 7 | 77353 Ridgeway Parkway | 77353 | Ridgeway Parkway |
| 8 | 5 Lien Avenue | 5 | Lien Avenue |

La query que extrae la siguiente tabla es la siguiente:

```
SELECT
    ADDRESS,
    substring_index(ADDRESS, ' ', 1),
    substring(ADDRESS FROM length((substring_index(ADDRESS, ' ',
1)))+1)
FROM STAGE.STG_CLIENTES_CRM
;
```

■ 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”:

- Data Quality

Nota: Además de las obvias que nos han salido al crear ODS que hay que describirlas, ¿se te ocurre alguna otra normalización?

- Master Data

- Data Modeling & Design (me refiero a cómo están definidas las tablas en origen)

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

DATA MANAGEMENT

DATA QUALITY:

Un asunto importante para mejorar la Calidad del dato es intentar no entrar valores duplicados. Por tanto es importante establecer controles que alerten si un cliente ya se encuentra dado de alta.

Otra medida es establecer un patrón para los teléfonos para tenerlos normalizados.

Es muy interesante y se está realizando cada vez más, la comprobación de la dirección del maestro de direcciones con un callejero (ej. el de google) . Esta medida enriquece la Base de datos completando la dirección en caso de que falte algún dato (ej. el código postal) y detecta errores de dirección que puede dar lugar a que no se reciba el correo postal. Una vez éstas direcciones se encuentran localizadas, es muy fácil geo-posicionarlas. Esto permite mejorar el reporting visual sobre mapas y la segmentación de clientes.

MASTER DATA:

La clave del Master Data Management es tener unos datos maestros únicos y confiables. De esta manera se evitan problemas relativo a datos de contacto de clientes, como son direcciones o teléfonos.

El mayor problema que veo en nuestro DWH se encuentra en el origen de estos datos maestros, ya que son extraídos de tablas de hecho. Los datos maestros por definición son de lenta actualización y permanecen estables por tiempo.

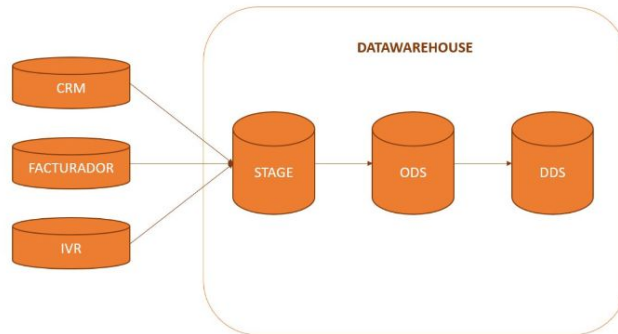
Mi propuesta es extraer los dato maestros de tablas fijas o ficheros csv.

DATA MODELING & DESIGN:

A la hora de descargar los datos al STAGE se han establecido como tipo fijo (VARCHAR) en algunas tablas. He comprobado que en origen vienen igual, pero sería conveniente clarificar la estructura final. Durante el modelado se ha fijado una longitud en función de los datos de entrada, pero ésto no tiene en cuenta el crecimiento futuro.

Cuarta parte

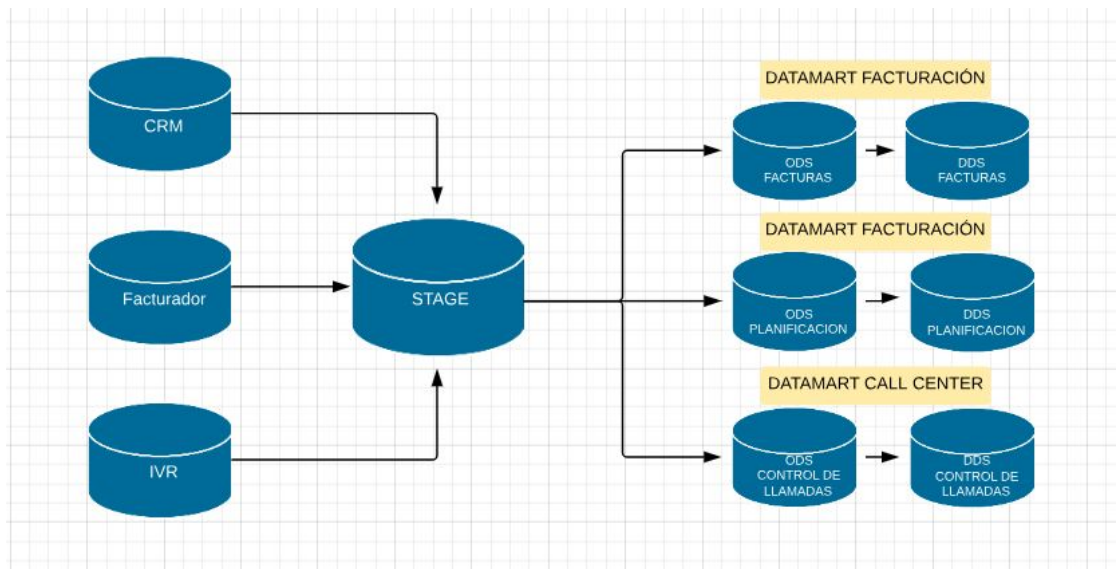
Después de todo lo visto nuestro ecosistema quedaría así:



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

Propuesta de mejora de diseño

Dado que el fin último del DWH está en servir como plataforma de explotación y análisis, me gusta orientar el diseño a tal fin. Por ello adoptaría una estructura de Datamart como salida, preparando y separando los datos para cada finalidad con una estrategia de **“Buttom to up”**.



En nuestro caso, un DataMart de Facturación, otro para el control de llamadas y el último para planificación de servicios. Cada DataMart contiene su propio ODS normalizado con el criterio de SCD que sea necesario.

La desventaja de este modelo es que duplica información, pero si las tablas se optimizan llevando sólo la información necesaria para su explotación, el rendimiento no queda penalizado y más teniendo en cuenta el alto rendimiento de los SGBD actuales.

Por otro lado este sistema tiene la ventaja de que es más adaptable, ya que cualquier modificación solicitada por un área de negocio, no afecta al resto, y también es más robusto, ya que una incidencia en una de las cadenas sólo afecta al Datamart del que depende la carga.

A nivel físico también tenemos la ventaja de que la distribución de las bases de datos y su crecimiento sigue el orden lógico. Con este modelo, un crecimiento físico de una de las bases de datos, nos permite separarlo del resto, no comprometiendo la integridad del sistema.

En cuanto a la seguridad, la replicación y redundancia también se ven afectados positivamente, ya que no todos los DataMart tendrán la misma criticidad ni serán actualizados al mismo tiempo, unos tendrán actualizaciones diarias, mientras que otros serán mensuales. A nivel de backup se pueden planificar acorde a esta actualización.

■ Quinta parte

Escribe tus propias reglas o mandamientos de un DataWarehouse

Reglas para la creación y mantenimiento del DWH

- Todas las tablas de hechos tienen que tener una tabla de dimensión de tiempos asociada. En nuestro caso, por ejemplo, faltaría asociar esta tabla a cada una de las tablas de hecho. Por ejemplo, en la de Facturas a la fecha de emisión de factura, y en la de servicios.
- Si se puede, generar tabla de hechos dependientes del tiempo. Por ejemplo, la tabla ODS_HC_SERVICIOS es una tabla con fecha de inicio y fin. Generando esta tabla dependiente del tiempo, por ejemplo mensual, es fácil de responder a la pregunta sobre los servicios vigentes por mes. Una tabla dependiente del tiempo debe de estar marcada. Como regla, suelo emplear un sufijo. En este caso quedaría como ODS_HC_SERVICIOS_M.
- Cada datamart, y por tanto sus tablas de hechos, deben de contar con su propia tabla temporal, y todas las tablas de hechos dependientes del tiempo se montan a partir de esa tabla. De esa manera, controlando la tabla temporal, se controla la carga. Abriendo el rango de la tabla de tiempos, por ej. de 3 a 5 años, cargamos todas las tablas dependientes durante ese periodo.
- Salvo excepciones, por ejemplo “badget”, no visualizar datos a futuro. En este caso, la tabla de tiempo también controla la parte final del rango. En el caso de previsiones es mejor hacer un cambio de escala temporal y traerlas a presente como dato asociado. Por ejemplo, previsión de Enero del año+1 está asociado a Enero del año actual.
- La granularidad tiene que ser la misma para todos. En este sentido son muy importante mantener unas jerarquías empresariales comunes. En caso de no ser posible, unificar a partir de cierto nivel, por ejemplo puede ser diferente a nivel más bajo, como Centro de Trabajo, Sucursal, o incluso Centro de coste, pero se pueden unificar a nivel de Área y Empresa.

-
- Resolver los loop en las relaciones entre tablas. Por ejemplo, la tabla de direcciones tienen relación 1 a varios con la de Servicios y la de clientes. Un sistema de reporting que intentara realizar una query sobre alguna dimensión podría entregar resultados no deseados. En este caso, se puede mantener la relación, si el sistema de reporting permite el uso de **contextos** o **alias** que resuelvan el bucle.
 - Nunca dejar una relación entre tablas de N:N. En caso de darse, utilizar una tabla intermedia que rompa esa relación.
 - Llevar las descripciones a tablas de dimensiones evitando que esté en la tabla de hechos.
 - En cuanto a la convención de los nombres, para la tabla de hechos mantengo la de OSD, ya que me gusta que el prefijo sea el mismo que el de la base de datos. Para las tablas de dimensiones suelo utilizar el prefijo DIM. En este modelo no tenemos claro la granularidad de la tabla, pero suelo utilizar un sufijo que me la indica (_M para mes, _D para diaria, _A para anual y un especial _MAN para aquellas que provienen de fuentes de datos manuales (txt, excel, csv,...))
 - Por último, reevalúa constantemente el rendimiento del sistema estableciendo KPIs que midan y evalúen la calidad del dato, así como auditoría para comprobar los puntos más sensible. Un DWH debe de reflejar la naturaleza empresarial. Una empresa activa obliga a hacer reingeniería y cambios más a menudo.