Práctica Data Warehouse

Iván Rubio Moreno

PRIMERA PARTE

Estudio STG_FACTURAS_FCT

Para realizar el estudio de la tabla STG_FACTURAS_FCT realizamos consultas sobre la tabla para obtener el total de valores no vacíos y los valores únicos de cada campo. Para realizar estas consultas he generado el siguiente script:

Script

```
# ESTUDIO DE STG_FACTURAS_FCT

USE STAGE;

SELECT COUNT(*) TOTAL_REGISTROS
, SUM(CASE WHEN BILL_REF_NO IS NOT NULL THEN 1 ELSE 0 END) TOTAL_BILL_REF_NO
, COUNT(DISTINCT CASE WHEN BILL_REF_NO IS NOT NULL AND BILL_REF_NO <> '' THEN
BILL_REF_NO ELSE 0 END) TOTAL_DISTINTOS_BILL_REF_NO

, SUM(CASE WHEN LENGTH(TRIM(CUSTOMER_ID)) <> 0 THEN 1 ELSE 0 END) AS TOTAL_CUSTOMER_ID
, COUNT(DISTINCT CASE WHEN LENGTH(TRIM(CUSTOMER_ID)) <> 0 THEN CUSTOMER_ID ELSE 0 END)
AS TOTAL_DISTINTOS_CUSTOMER_ID

, SUM(CASE WHEN START_DATE IS NOT NULL THEN 1 ELSE 0 END) TOTAL_START_DATE
, COUNT(DISTINCT CASE WHEN START_DATE IS NOT NULL AND START_DATE <> '' THEN START_DATE
ELSE 0 END) TOTAL_DISTINTOS_START_DATE

, SUM(CASE WHEN END_DATE IS NOT NULL THEN 1 ELSE 0 END) TOTAL_END_DATE
, COUNT(DISTINCT CASE WHEN END_DATE IS NOT NULL AND END_DATE <> '' THEN END_DATE ELSE 0
END) TOTAL_DISTINTOS_END_DATE

, SUM(CASE WHEN STATEMENT_DATE IS NOT NULL THEN 1 ELSE 0 END) TOTAL_STATEMENT_DATE
, COUNT(DISTINCT CASE WHEN STATEMENT_DATE IS NOT NULL AND STATEMENT_DATE <> '' THEN
STATEMENT_DATE ELSE 0 END) TOTAL_DISTINTOS_STATEMENT_DATE

, SUM(CASE WHEN PAYMENT_DATE IS NOT NULL THEN 1 ELSE 0 END) TOTAL_PAYMENT_DATE
, COUNT(DISTINCT CASE WHEN PAYMENT_DATE IS NOT NULL AND PAYMENT_DATE <> '' THEN
PAYMENT_DATE ELSE 0 END) TOTAL_DISTINTOS_PAYMENT_DATE

, SUM(CASE WHEN BILL_CYCLE IS NOT NULL THEN 1 ELSE 0 END) TOTAL_BILL_CYCLE
, COUNT(DISTINCT CASE WHEN BILL_CYCLE IS NOT NULL AND BILL_CYCLE <> '' THEN BILL_CYCLE
ELSE 0 END) TOTAL_DISTINTOS_BILL_CYCLE

, SUM(CASE WHEN AMOUNT IS NOT NULL THEN 1 ELSE 0 END) TOTAL_AMOUNT
, COUNT(DISTINCT CASE WHEN AMOUNT IS NOT NULL AND AMOUNT <> '' THEN AMOUNT ELSE 0 END)
TOTAL_DISTINTOS_AMOUNT

, SUM(CASE WHEN BILL_METHOD IS NOT NULL THEN 1 ELSE 0 END) TOTAL_BILL_METHOD
, COUNT(DISTINCT CASE WHEN BILL_METHOD IS NOT NULL AND BILL_METHOD <> '' THEN
BILL_METHOD ELSE 0 END) TOTAL_DISTINTOS_BILL_METHOD

FROM STAGE.STG_FACTURAS_FCT
```

Resultado de la ejecución del script

Campo	Valor
TOTAL_REGISTROS	420000
TOTAL_BILL_REF_NO	420000
TOTAL_DISTINTOS_BILL_REF_NO	420000
TOTAL_CUSTOMER_ID	420000
TOTAL_DISTINTOS_CUSTOMER_ID	20000
TOTAL_START_DATE	420000
TOTAL_DISTINTOS_START_DATE	40
TOTAL_END_DATE	420000
TOTAL_DISTINTOS_END_DATE	20
TOTAL_STATEMENT_DATE	420000
TOTAL_DISTINTOS_STATEMENT_DATE	40
TOTAL_PAYMENT_DATE	420000
TOTAL_DISTINTOS_PAYMENT_DATE	400
TOTAL_BILL_CYCLE	420000
TOTAL_DISTINTOS_BILL_CYCLE	2
TOTAL_AMOUNT	420000
TOTAL_DISTINTOS_AMOUNT	5604
TOTAL_BILL_METHOD	420000
TOTAL_DISTINTOS_BILL_METHOD	3

Conclusiones

Observando la estructura de la tabla, su contenido y con los resultados obtenidos podemos sacar la siguientes conclusiones:

- El campo **BILL_REF_NO** es la clave principal. Todos los registros tienen este campo informado y sus valores son únicos.
- **CUSTOMER_ID** es una foreing que apunta a la tabla de clientes.
- Los campos **START_DATE, END_DATE, STATEMENT_DATE y PAYMENT_DATE** aparentemente son de tipo Date porque aunque en alguno se ve la hora, además de la fecha, pero esta hora es siempre 00:00:00, así que se podría obviar la hora, pero dado que en origen parece que son de tipo DateTime, mantendremos este tipo de dato para evitar futuros problemas en un futuro. Por el lado el contenido,

en los cuatro campos se repiten mucho los valores, así que creo que lo mas conveniente es sacar una dimensión de cada campo.

- **BILL_CYCLE** Como se ve en los resultado solo hay dos valores para los 420.000 registros, por lo que sacaremos este campo a una tabla de dimensión de ciclos de facturación.
- **BILL_METHOD** Al igual que en el dato anterior, a partir de este campo obtenemos una dimensión de métodos de pago.

Estudio STG_CONTACTOS_IVR

Para realizar el estudio de la tabla STG_CONTACTOS_IVR realizamos consultas sobre la tabla para obtener el total de valores no vacíos de cada campo y los valores únicos de cada campo. Para realizar estas consultas he generado el siguiente script:

Script

```
# ESTUDIO DE STG_CONTACTOS_IVR

USE STAGE;

SELECT COUNT(*) TOTAL_REGISTROS
, SUM(CASE WHEN LENGTH(TRIM(ID)) <> 0 THEN 1 ELSE 0 END) TOTAL_ID
, COUNT(DISTINCT CASE WHEN LENGTH(TRIM(ID)) <> 0 AND ID <> '' THEN ID ELSE 0 END)
TOTAL_DISTINTOS_ID

, SUM(CASE WHEN LENGTH(TRIM(PHONE_NUMBER)) <> 0 THEN 1 ELSE 0 END) AS
TOTAL_PHONE_NUMBER
, COUNT(DISTINCT CASE WHEN LENGTH(TRIM(PHONE_NUMBER)) <> 0 THEN PHONE_NUMBER ELSE 0
END) AS TOTAL_DISTINTOS_PHONE_NUMBER

, SUM(CASE WHEN LENGTH(TRIM(START_DATETIME)) <> 0 THEN 1 ELSE 0 END)
TOTAL_START_DATETIME
, COUNT(DISTINCT CASE WHEN LENGTH(TRIM(START_DATETIME)) <> 0 AND START_DATETIME <> ''
THEN START_DATETIME ELSE 0 END) TOTAL_DISTINTOS_START_DATETIME

, SUM(CASE WHEN LENGTH(TRIM(END_DATETIME)) <> 0 THEN 1 ELSE 0 END) TOTAL_END_DATETIME
, COUNT(DISTINCT CASE WHEN LENGTH(TRIM(END_DATETIME)) <> 0 AND END_DATETIME <> '' THEN
END_DATETIME ELSE 0 END) TOTAL_DISTINTOS_END_DATETIME

, SUM(CASE WHEN LENGTH(TRIM(SERVICE)) <> 0 THEN 1 ELSE 0 END) TOTAL_SERVICE
, COUNT(DISTINCT CASE WHEN LENGTH(TRIM(SERVICE)) <> 0 AND SERVICE <> '' THEN SERVICE
ELSE 0 END) TOTAL_DISTINTOS_SERVICE

, SUM(CASE WHEN LENGTH(TRIM(FLG_TRANSFER)) <> 0 THEN 1 ELSE 0 END) TOTAL_FLG_TRANSFER
, COUNT(DISTINCT CASE WHEN LENGTH(TRIM(FLG_TRANSFER)) <> 0 AND FLG_TRANSFER <> '' THEN
FLG_TRANSFER ELSE 0 END) TOTAL_DISTINTOS_FLG_TRANSFER

, SUM(CASE WHEN LENGTH(TRIM(AGENT)) <> 0 THEN 1 ELSE 0 END) TOTAL_AGENT
```

```
 , COUNT(DISTINCT CASE WHEN LENGTH(TRIM(AGENT)) <> 0 AND AGENT <> '' THEN AGENT ELSE 0
END) TOTAL_DISTINTOS_AGENT

FROM STAGE.STG_CONTACTOS_IVR
```

Resultado de la ejecución del script

Campo	Valor
TOTAL_REGISTROS	202717
TOTAL_ID	202717
TOTAL_DISTINTOS_ID	150000
TOTAL_PHONE_NUMBER	185018
TOTAL_DISTINTOS_PHONE_NUMBER	18226
TOTAL_START_DATETIME	202717
TOTAL_DISTINTOS_START_DATETIME	201098
TOTAL_END_DATETIME	186535
TOTAL_DISTINTOS_END_DATETIME	183678
TOTAL_SERVICE	202502
TOTAL_DISTINTOS_SERVICE	7
TOTAL_FLG_TRANSFER	202717
TOTAL_DISTINTOS_FLG_TRANSFER	2
TOTAL_AGENT	194739
TOTAL_DISTINTOS_AGENT	594

Conclusiones

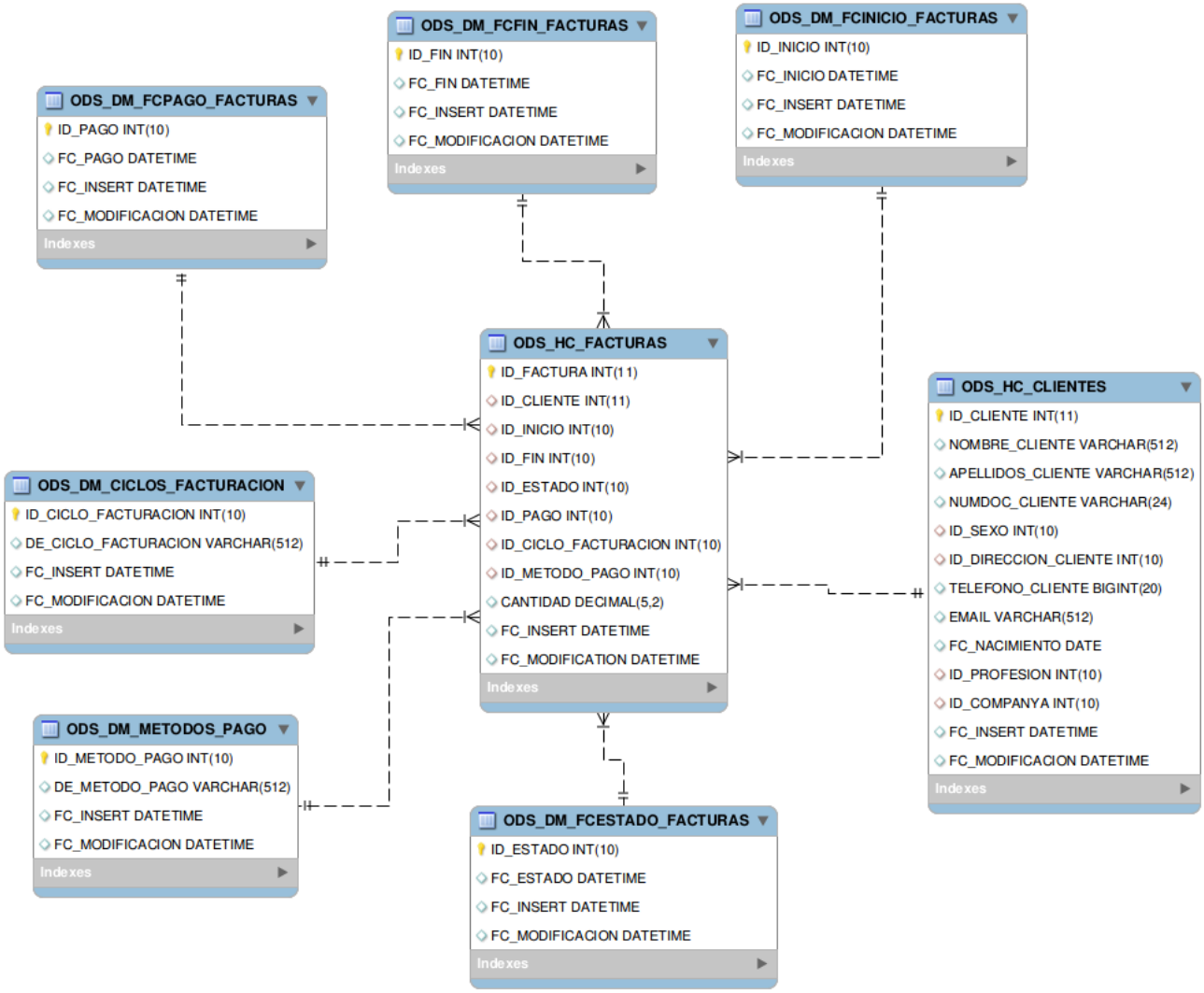
Observando la estructura de la tabla, su contenido y con los resultados obtenidos podemos sacar la siguientes conclusiones:

- El campo **ID**, al contrario de lo que pueda parecer por su denominación, no es la clave primaria de la tabla. No hay ningún registro en el que se encuentre vacío, pero sus valores a veces se repiten, por lo que no sirve para identificar un registro único. Analizando el contenido para comprobar si la PK está formada por mas de un campo, no encuentro ninguna posibilidad viable. Parece que cada ID distinto pertenece a una misma llamada de un cliente. Las veces que se repite el mismo ID son de la misma llamada del cliente y cada registro corresponde a un desvio de llamada a otro empleado de la empresa. Además, el ID se va repitiendo mientras FLG_TRANSFER = False y cuando se repiten los IDs, el valor END_DATETIME del registro de origen es igual al valor START_DATETIME del registro destino de la llamada. Al no poder encontrar un campo clave en el modelo tendremos que crear un campo autonumérico que sea la clave de la tabla.

- **PHONE_NUMBER** Cabría esperar que este campo fuera el número de teléfono de los clientes para poder relacionar esta tabla con la de clientes, pero comprobándolo no coincide ninguno. Esto confirma mi teoría de que los números de teléfono son de empleados de la empresa. Observando el resto de datos no encuentro forma de relacionar esta tabla con la de clientes, así que en caso de incluir una FK a la tabla de clientes, esta tendrá el valor de *DESCONOCIDO* en todos los registros. Además, hay registros en los no hay valor para este campo, por lo que hay casos en los que no quedó registrado, posiblemente por algún error en el origen del dato.
- **START_DATETIME y END_DATETIME** son campos de tipo DateTime que almacenan el inicio y fin de la llamada con cada operador de la empresa.
- El campo **SERVICE** parece que contiene el departamento que atiende la llamada. Tiene 6 posibles valores por lo que que crearemos una dimensión con el valore de este campo.
- Como he comentado antes, **FLG_TRANSFER** indica si la llamada se transfiere a otro agente o si termina con el agente actual y tiene dos posibles valores: **True o False**. Así que este campo lo dejaremos como entero y cuando su valor sea False lo cambiaremos por un 0 y cuando valga True lo cambiaremos por un 1.
- **AGENT** Este campo, por su contenido, parece un nombre de usuario. Había pensado que se podría cruzar con el campo AGENT_CODE de la tabla productos, pero en ese caso es un campo numérico y AGENT es alfanumérico, así que no encuentro manera de relacionarlos. En cualquier caso, haremos una dimensión de este dato ya que parece que puede ser bastante relevante en un futuro para fines estadísticos.

Modelo y datos de facturas

Con las conclusiones obtenidas del estudio de la tabla de facturas he planteado el siguiente modelo:



Para crear el modelo, los índices y las foreingn keys he creado el script MODELO_FACTURAS.sql cuyo contenido es el siguiente:

```
USE ODS;

SET FOREIGN_KEY_CHECKS=0;

DROP TABLE IF EXISTS ODS_DM_FCINICIO_FACTURAS;

CREATE TABLE ODS_DM_FCINICIO_FACTURAS
(ID_INICIO INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
FC_INICIO DATETIME,
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME);

DROP TABLE IF EXISTS ODS_DM_FCFIN_FACTURAS;

CREATE TABLE ODS_DM_FCFIN_FACTURAS
(ID_FIN INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
FC_FIN DATETIME,
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME);
```

```sql
DROP TABLE IF EXISTS ODS_DM_FCESTADO_FACTURAS;

CREATE TABLE ODS_DM_FCESTADO_FACTURAS
(ID_ESTADO INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
FC_ESTADO DATETIME,
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME);

DROP TABLE IF EXISTS ODS_DM_FCPAGO_FACTURAS;

CREATE TABLE ODS_DM_FCPAGO_FACTURAS
(ID_PAGO INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
FC_PAGO DATETIME,
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME);

DROP TABLE IF EXISTS ODS_DM_CICLOS_FACTURACION;

CREATE TABLE ODS_DM_CICLOS_FACTURACION
(ID_CICLO_FACTURACION INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
DE_CICLO_FACTURACION VARCHAR(512),
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME);

DROP TABLE IF EXISTS ODS_DM_METODOS_PAGO;

CREATE TABLE ODS_DM_METODOS_PAGO
(ID_METODO_PAGO INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
DE_METODO_PAGO VARCHAR(512),
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME);


DROP TABLE IF EXISTS ODS_HC_FACTURAS;

CREATE TABLE ODS_HC_FACTURAS
(ID_FACTURA INT NOT NULL PRIMARY KEY,
ID_CLIENTE INT,
ID_INICIO INT UNSIGNED,
ID_FIN INT UNSIGNED,
ID_ESTADO INT UNSIGNED,
ID_PAGO INT UNSIGNED,
ID_CICLO_FACTURACION INT UNSIGNED,
ID_METODO_PAGO INT UNSIGNED,
CANTIDAD DECIMAL(5,2),
FC_INSERT DATETIME,
FC_MODIFICATION DATETIME);


ALTER TABLE ODS_HC_FACTURAS ADD INDEX fk_fac_cli_idx (ID_CLIENTE ASC);
ALTER TABLE ODS_HC_FACTURAS ADD CONSTRAINT fk_fac_cli FOREIGN KEY (ID_CLIENTE)
REFERENCES ODS_HC_CLIENTES (ID_CLIENTE);
```

```
ALTER TABLE ODS_HC_FACTURAS ADD INDEX fk_fac_inicio_idx (ID_INICIO ASC);
ALTER TABLE ODS_HC_FACTURAS ADD CONSTRAINT fk_fac_inicio FOREIGN KEY(ID_INICIO)
   REFERENCES ODS_DM_FCINICIO_FACTURAS(ID_INICIO);

ALTER TABLE ODS_HC_FACTURAS ADD INDEX fk_fac_fin_idx (ID_FIN ASC);
ALTER TABLE ODS_HC_FACTURAS ADD CONSTRAINT fk_fac_fin FOREIGN KEY(ID_FIN)
   REFERENCES ODS_DM_FCFIN_FACTURAS(ID_FIN);

ALTER TABLE ODS_HC_FACTURAS ADD INDEX fk_fac_estado_idx (ID_ESTADO ASC);
ALTER TABLE ODS_HC_FACTURAS ADD CONSTRAINT fk_fac_estado FOREIGN KEY(ID_ESTADO)
   REFERENCES ODS_DM_FCESTADO_FACTURAS(ID_ESTADO);

ALTER TABLE ODS_HC_FACTURAS ADD INDEX fk_fac_pago_idx (ID_PAGO ASC);
ALTER TABLE ODS_HC_FACTURAS ADD CONSTRAINT fk_fac_pago FOREIGN KEY(ID_PAGO)
   REFERENCES ODS_DM_FCPAGO_FACTURAS(ID_PAGO);

ALTER TABLE ODS_HC_FACTURAS ADD INDEX fk_fac_mp_idx (ID_METODO_PAGO ASC);
ALTER TABLE ODS_HC_FACTURAS ADD CONSTRAINT fk_fac_mp FOREIGN KEY
(ID_METODO_PAGO) REFERENCES ODS_DM_METODOS_PAGO (ID_METODO_PAGO);

ALTER TABLE ODS_HC_FACTURAS ADD INDEX fk_fac_cf_idx (ID_CICLO_FACTURACION ASC);
ALTER TABLE ODS_HC_FACTURAS ADD CONSTRAINT fk_fac_cf FOREIGN KEY
(ID_CICLO_FACTURACION) REFERENCES ODS_DM_CICLOS_FACTURACION(ID_CICLO_FACTURACION);
```

Por último, para poblar las nuevas tablas he creado el SCRIPT DATOS_FACTURAS.sql cuyo contenido es:

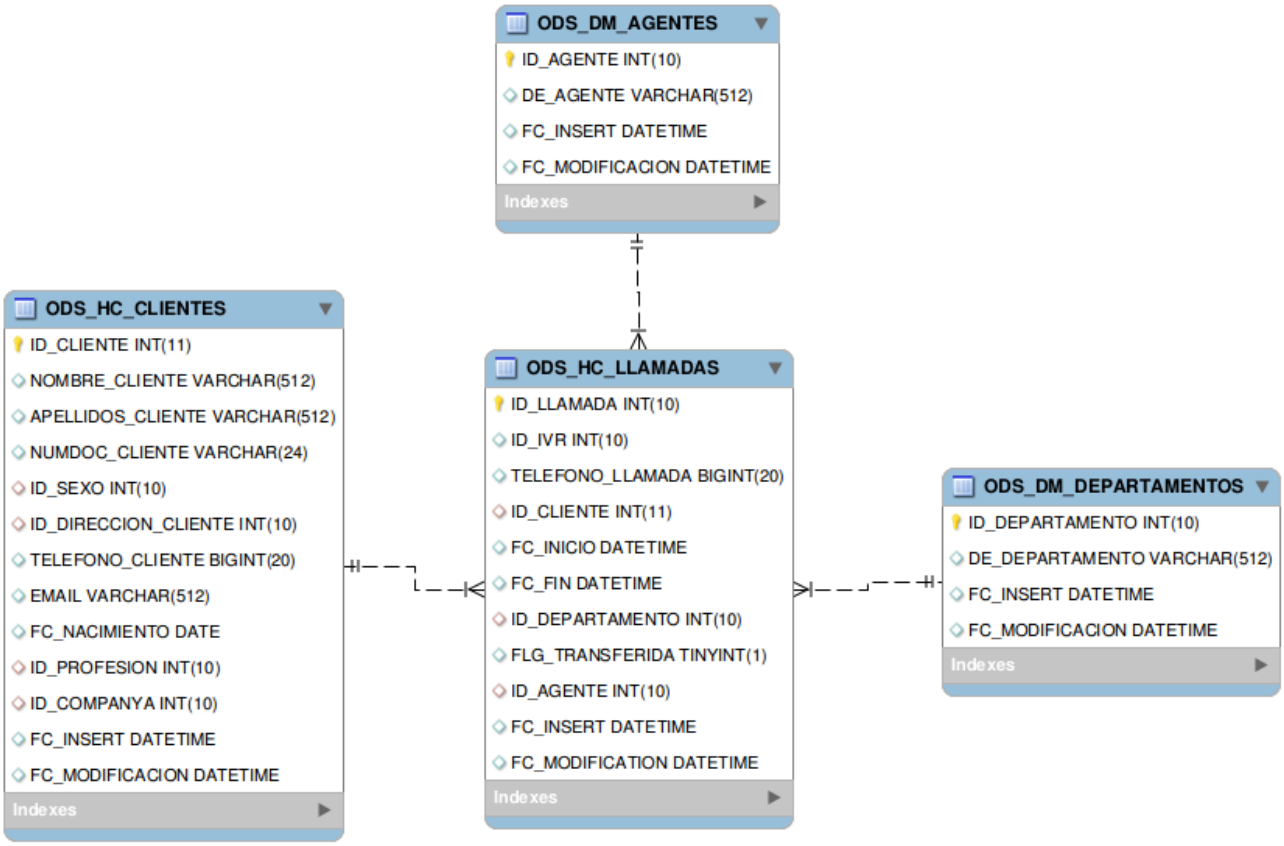
```
USE ODS;

INSERT INTO ODS_DM_FCINICIO_FACTURAS (FC_INICIO, FC_INSERT, FC_MODIFICACION)
SELECT DISTINCT START_DATE, NOW(), NOW()
FROM STAGE.STG_FACTURAS_FCT
WHERE TRIM(START_DATE)<>'';
INSERT INTO ODS_DM_FCINICIO_FACTURAS VALUES (999, STR_TO_DATE('31/12/9999','%d/%m/%Y'),
NOW(),NOW());
INSERT INTO ODS_DM_FCINICIO_FACTURAS VALUES (998, STR_TO_DATE('31/12/9998','%d/%m/%Y'),
NOW(),NOW());
COMMIT;
ANALYZE TABLE ODS_DM_FCINICIO_FACTURAS;

INSERT INTO ODS_DM_FCFIN_FACTURAS (FC_FIN, FC_INSERT, FC_MODIFICACION)
SELECT DISTINCT END_DATE, NOW(), NOW()
FROM STAGE.STG_FACTURAS_FCT
WHERE TRIM(END_DATE)<>'';
INSERT INTO ODS_DM_FCFIN_FACTURAS VALUES (999, STR_TO_DATE('31/12/9999','%d/%m/%Y'),
NOW(),NOW());
INSERT INTO ODS_DM_FCFIN_FACTURAS VALUES (998, STR_TO_DATE('31/12/9998','%d/%m/%Y'),
NOW(),NOW());
COMMIT;
ANALYZE TABLE ODS_DM_FCFIN_FACTURAS;

INSERT INTO ODS_DM_FCESTADO_FACTURAS (FC_ESTADO, FC_INSERT, FC_MODIFICACION)
SELECT DISTINCT STATEMENT_DATE, NOW(), NOW()
```

```
FROM STAGE.STG_FACTURAS_FCT
WHERE TRIM(STATEMENT_DATE)<>'';
INSERT INTO ODS_DM_FCESTADO_FACTURAS VALUES (999, STR_TO_DATE('31/12/9999','%d/%m/%Y'),
NOW(),NOW());
INSERT INTO ODS_DM_FCESTADO_FACTURAS VALUES (998, STR_TO_DATE('31/12/9998','%d/%m/%Y'),
NOW(),NOW());
COMMIT;
ANALYZE TABLE ODS_DM_FCESTADO_FACTURAS;

INSERT INTO ODS_DM_FCPAGO_FACTURAS (FC_PAGO, FC_INSERT, FC_MODIFICACION)
SELECT DISTINCT PAYMENT_DATE, NOW(), NOW()
FROM STAGE.STG_FACTURAS_FCT
WHERE TRIM(PAYMENT_DATE)<>'';
INSERT INTO ODS_DM_FCPAGO_FACTURAS VALUES (999, STR_TO_DATE('31/12/9999','%d/%m/%Y'),
NOW(),NOW());
INSERT INTO ODS_DM_FCPAGO_FACTURAS VALUES (998, STR_TO_DATE('31/12/9998','%d/%m/%Y'),
NOW(),NOW());
COMMIT;
ANALYZE TABLE ODS_DM_FCPAGO_FACTURAS;

INSERT INTO ODS_DM_CICLOS_FACTURACION (DE_CICLO_FACTURACION, FC_INSERT,
FC_MODIFICACION)
SELECT DISTINCT BILL_CYCLE, NOW(), NOW()
FROM STAGE.STG_FACTURAS_FCT
WHERE TRIM(BILL_CYCLE)<>'';
INSERT INTO ODS_DM_CICLOS_FACTURACION VALUES (99, 'DESCONOCIDO', NOW(),NOW());
INSERT INTO ODS_DM_CICLOS_FACTURACION VALUES (98, 'NO APLICA', NOW(),NOW());
COMMIT;
ANALYZE TABLE ODS_DM_CICLOS_FACTURACION;


INSERT INTO ODS_DM_METODOS_PAGO (DE_METODO_PAGO, FC_INSERT, FC_MODIFICACION)
SELECT DISTINCT BILL_METHOD, NOW(), NOW()
FROM STAGE.STG_FACTURAS_FCT
WHERE TRIM(BILL_METHOD)<>'';
INSERT INTO ODS_DM_METODOS_PAGO VALUES (99, 'DESCONOCIDO', NOW(),NOW());
INSERT INTO ODS_DM_METODOS_PAGO VALUES (98, 'NO APLICA', NOW(),NOW());
COMMIT;
ANALYZE TABLE ODS_DM_METODOS_PAGO;


INSERT INTO ODS_HC_CLIENTES VALUES (
    999999999,
    'DESCONOCIDO',
    'DESCONOCIDO',
    '99-999-9999',
    99,
    999999,
    9999999999,
    'DESCONOCIDO',
    STR_TO_DATE('31/12/9999', '%d/%m/%Y'),
    999,
    999,
```

```
    NOW(),
    STR_TO_DATE('31/12/9999', '%d/%m/%Y'));
COMMIT;
ANALYZE TABLE ODS_HC_CLIENTES;

INSERT INTO ODS_HC_FACTURAS (ID_FACTURA, ID_CLIENTE, ID_INICIO, ID_FIN, ID_ESTADO,
ID_PAGO, ID_CICLO_FACTURACION, ID_METODO_PAGO, CANTIDAD, FC_INSERT, FC_MODIFICATION)
SELECT BILL_REF_NO ID_FACTURA,
CASE WHEN TRIM(CLI.ID_CLIENTE)<>'' THEN CLI.ID_CLIENTE ELSE 999999999 END ID_CLIENTE,
CASE WHEN TRIM(FINI.ID_INICIO)<>'' THEN FINI.ID_INICIO ELSE 999 END ID_INICIO,
CASE WHEN TRIM(FFIN.ID_FIN)<>'' THEN FFIN.ID_FIN ELSE 999 END ID_FIN,
CASE WHEN TRIM(FEST.ID_ESTADO)<>'' THEN FEST.ID_ESTADO ELSE 999 END ID_ESTADO,
CASE WHEN TRIM(FPAGO.ID_PAGO)<>'' THEN FPAGO.ID_PAGO ELSE 999 END ID_PAGO,
CASE WHEN TRIM(CFCT.ID_CICLO_FACTURACION)<>'' THEN CFCT.ID_CICLO_FACTURACION ELSE 999
END ID_CICLO_FACTURACION ,
CASE WHEN TRIM(PFCT.ID_METODO_PAGO)<>'' THEN PFCT.ID_METODO_PAGO ELSE 999 END
ID_METODO_PAGO,
CASE WHEN LENGTH(TRIM(AMOUNT))<>0 THEN CAST(AMOUNT AS DECIMAL(5,2)) ELSE '99999.99' END
CANTIDAD,
NOW(),
NOW()
FROM STAGE.STG_FACTURAS_FCT FCT
INNER JOIN
    ODS_DM_FCINICIO_FACTURAS FINI ON CASE WHEN LENGTH(TRIM(FCT.START_DATE))<>0 THEN
STR_TO_DATE(UPPER(TRIM(FCT.START_DATE)),'%Y-%m-%d %H:%i:%s') ELSE
STR_TO_DATE('31/12/9999','%d/%m/%Y') END=FINI.FC_INICIO
INNER JOIN
    ODS_DM_FCFIN_FACTURAS FFIN ON CASE WHEN LENGTH(TRIM(FCT.END_DATE))<>0 THEN
STR_TO_DATE(UPPER(TRIM(FCT.END_DATE)),'%Y-%m-%d %H:%i:%s') ELSE
STR_TO_DATE('31/12/9999','%d/%m/%Y') END=FFIN.FC_FIN
INNER JOIN
    ODS_DM_FCESTADO_FACTURAS FEST ON CASE WHEN LENGTH(TRIM(FCT.STATEMENT_DATE))<>0 THEN
STR_TO_DATE(UPPER(TRIM(FCT.STATEMENT_DATE)),'%Y-%m-%d %H:%i:%s') ELSE
STR_TO_DATE('31/12/9999','%d/%m/%Y') END=FEST.FC_ESTADO
INNER JOIN
    ODS_DM_FCPAGO_FACTURAS FPAGO ON CASE WHEN LENGTH(TRIM(FCT.PAYMENT_DATE))<>0 THEN
STR_TO_DATE(UPPER(TRIM(FCT.PAYMENT_DATE)),'%Y-%m-%d %H:%i:%s') ELSE
STR_TO_DATE('31/12/9999','%d/%m/%Y') END=FPAGO.FC_PAGO
INNER JOIN
    ODS_DM_CICLOS_FACTURACION CFCT ON CASE WHEN LENGTH(TRIM(FCT.BILL_CYCLE))<>0 THEN
UPPER(TRIM(FCT.BILL_CYCLE)) ELSE 'DESCONOCIDO' END=CFCT.DE_CICLO_FACTURACION
INNER JOIN
    ODS_DM_METODOS_PAGO PFCT ON CASE WHEN LENGTH(TRIM(FCT.BILL_METHOD))<>0 THEN
UPPER(TRIM(FCT.BILL_METHOD)) ELSE 'DESCONOCIDO' END=PFCT.DE_METODO_PAGO
LEFT OUTER JOIN
    ODS_HC_CLIENTES CLI ON CASE WHEN LENGTH(TRIM(CUSTOMER_ID))<>0 THEN
TRIM(CUSTOMER_ID) ELSE 999999999 END=CLI.ID_CLIENTE;
COMMIT;
ANALYZE TABLE ODS_HC_FACTURAS;
```

Modelo y datos de llamadas

Con las conclusiones obtenidas del estudio de la tabla de llamadas he planteado el siguiente modelo:



Para crear el modelo, los índices y las foreingn keys he creado el script MODELO_LLAMADAS.sql cuyo contenido es el siguiente:

```
USE ODS;

DROP TABLE IF EXISTS ODS_HC_LLAMADAS;

CREATE TABLE ODS_HC_LLAMADAS (
ID_LLAMADA INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
ID_IVR INT UNSIGNED,
TELEFONO_LLAMADA BIGINT,
ID_CLIENTE INT,
FC_INICIO DATETIME,
FC_FIN DATETIME,
ID_DEPARTAMENTO INT UNSIGNED,
FLG_TRANSFERIDA TINYINT(1),
ID_AGENTE INT UNSIGNED,
FC_INSERT DATETIME,
FC_MODIFICATION DATETIME
);

DROP TABLE IF EXISTS ODS_DM_DEPARTAMENTOS;

CREATE TABLE ODS_DM_DEPARTAMENTOS
```

```
(ID_DEPARTAMENTO INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
DE_DEPARTAMENTO VARCHAR(512),
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME
);

DROP TABLE IF EXISTS ODS_DM_AGENTES;

CREATE TABLE ODS_DM_AGENTES
(ID_AGENTE INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
DE_AGENTE VARCHAR(512),
FC_INSERT DATETIME,
FC_MODIFICACION DATETIME
);

ALTER TABLE ODS_HC_LLAMADAS ADD INDEX fk_lla_dep_idx (ID_DEPARTAMENTO ASC);
ALTER TABLE ODS_HC_LLAMADAS ADD CONSTRAINT fk_lla_dep FOREIGN KEY(ID_DEPARTAMENTO)
   REFERENCES ODS_DM_DEPARTAMENTOS(ID_DEPARTAMENTO);

ALTER TABLE ODS_HC_LLAMADAS ADD INDEX fk_lla_age_idx (ID_AGENTE ASC);
ALTER TABLE ODS_HC_LLAMADAS ADD CONSTRAINT fk_lla_age FOREIGN KEY(ID_AGENTE)
   REFERENCES ODS_DM_AGENTES(ID_AGENTE);

ALTER TABLE ODS_HC_LLAMADAS ADD INDEX fk_lla_cli_idx (ID_CLIENTE ASC);
ALTER TABLE ODS_HC_LLAMADAS ADD CONSTRAINT fk_lla_cli FOREIGN KEY (ID_CLIENTE)
    REFERENCES ODS_HC_CLIENTES (ID_CLIENTE);
```

Por último, para poblar las nuevas tablas he creado el SCRIPT DATOS_LLAMADAS.sql cuyo contenido es:

```
USE ODS;

INSERT INTO ODS_DM_DEPARTAMENTOS (DE_DEPARTAMENTO, FC_INSERT, FC_MODIFICACION)
SELECT DISTINCT UPPER(TRIM(SERVICE)) AS DE_DEPARTAMENTO, NOW(), NOW()
FROM STAGE.STG_CONTACTOS_IVR
WHERE LENGTH(TRIM(SERVICE))<>0
ORDER BY DE_DEPARTAMENTO;

INSERT INTO ODS_DM_DEPARTAMENTOS VALUES (99, 'DESCONOCIDO', NOW(), NOW());
INSERT INTO ODS_DM_DEPARTAMENTOS VALUES (98, 'NO APLICA', NOW(), NOW());

COMMIT;
ANALYZE TABLE ODS_DM_DEPARTAMENTOS;

INSERT INTO ODS_DM_AGENTES (DE_AGENTE, FC_INSERT, FC_MODIFICACION)
SELECT DISTINCT UPPER(TRIM(AGENT)) AS AGENTE, NOW(), NOW()
FROM STAGE.STG_CONTACTOS_IVR
WHERE LENGTH(TRIM(AGENT)) <> 0
ORDER BY AGENTE;

INSERT INTO ODS_DM_AGENTES VALUES (9999, 'DESCONOCIDO', NOW(), NOW());
INSERT INTO ODS_DM_AGENTES VALUES (9998, 'NO APLICA', NOW(), NOW());

COMMIT;
```

```
ANALYZE TABLE ODS_DM_AGENTES;


INSERT INTO ODS_HC_LLAMADAS (ID_IVR, TELEFONO_LLAMADA, ID_CLIENTE, FC_INICIO, FC_FIN,
ID_DEPARTAMENTO, FLG_TRANSFERIDA, ID_AGENTE, FC_INSERT, FC_MODIFICATION)
SELECT
ID AS ID_IVR,
CASE WHEN LENGTH(TRIM(PHONE_NUMBER)) <> 0 THEN TRIM(PHONE_NUMBER) ELSE 9999999999 END
TELEFONO_LLAMADA,
CASE WHEN LENGTH(TRIM(CLI.ID_CLIENTE))<>0 THEN TRIM(CLI.ID_CLIENTE) ELSE 999999999 END
ID_CLIENTE,
CASE WHEN LENGTH(TRIM(START_DATETIME))<>0 THEN STR_TO_DATE(SUBSTRING(START_DATETIME, 1,
19),'%Y-%m-%d %H:%i:%s') ELSE STR_TO_DATE('31/12/9999', '%d/%m/%Y') END
FC_INICIO_LLAMADA,
CASE WHEN LENGTH(TRIM(END_DATETIME))<>0 THEN STR_TO_DATE(SUBSTRING(END_DATETIME, 1,
19),'%Y-%m-%d %H:%i:%s') ELSE STR_TO_DATE('31/12/9999', '%d/%m/%Y') END FC_FIN_LLAMADA,
DEP.ID_DEPARTAMENTO ID_DEPARTAMENTO,
CASE UPPER(TRIM(FLG_TRANSFER)) WHEN 'TRUE' THEN 1 ELSE 0 END FLG_TRANSFERIDA,
AGE.ID_AGENTE ID_AGENTE,
NOW(),
STR_TO_DATE('31/12/9999', '%d/%m/%Y')
FROM
STAGE.STG_CONTACTOS_IVR CONTACTOS
INNER JOIN
    ODS_DM_DEPARTAMENTOS DEP ON CASE WHEN LENGTH(TRIM(SERVICE)) <> 0 THEN
UPPER(TRIM(CONTACTOS.SERVICE)) ELSE 'DESCONOCIDO' END = DEP.DE_DEPARTAMENTO
INNER JOIN
    ODS_DM_AGENTES AGE ON CASE WHEN LENGTH(TRIM(AGENT)) <> 0 THEN
UPPER(TRIM(CONTACTOS.AGENT)) ELSE 'DESCONOCIDO' END = AGE.DE_AGENTE
LEFT OUTER JOIN
    ODS_HC_CLIENTES CLI ON CASE WHEN LENGTH(TRIM(PHONE_NUMBER))<>0 THEN
TRIM(PHONE_NUMBER) ELSE 999999999 END = CLI.TELEFONO_CLIENTE;

COMMIT;
ANALYZE TABLE ODS_HC_LLAMADAS;
```

Segunda parte

Diagrama completo de la base de datos ODS

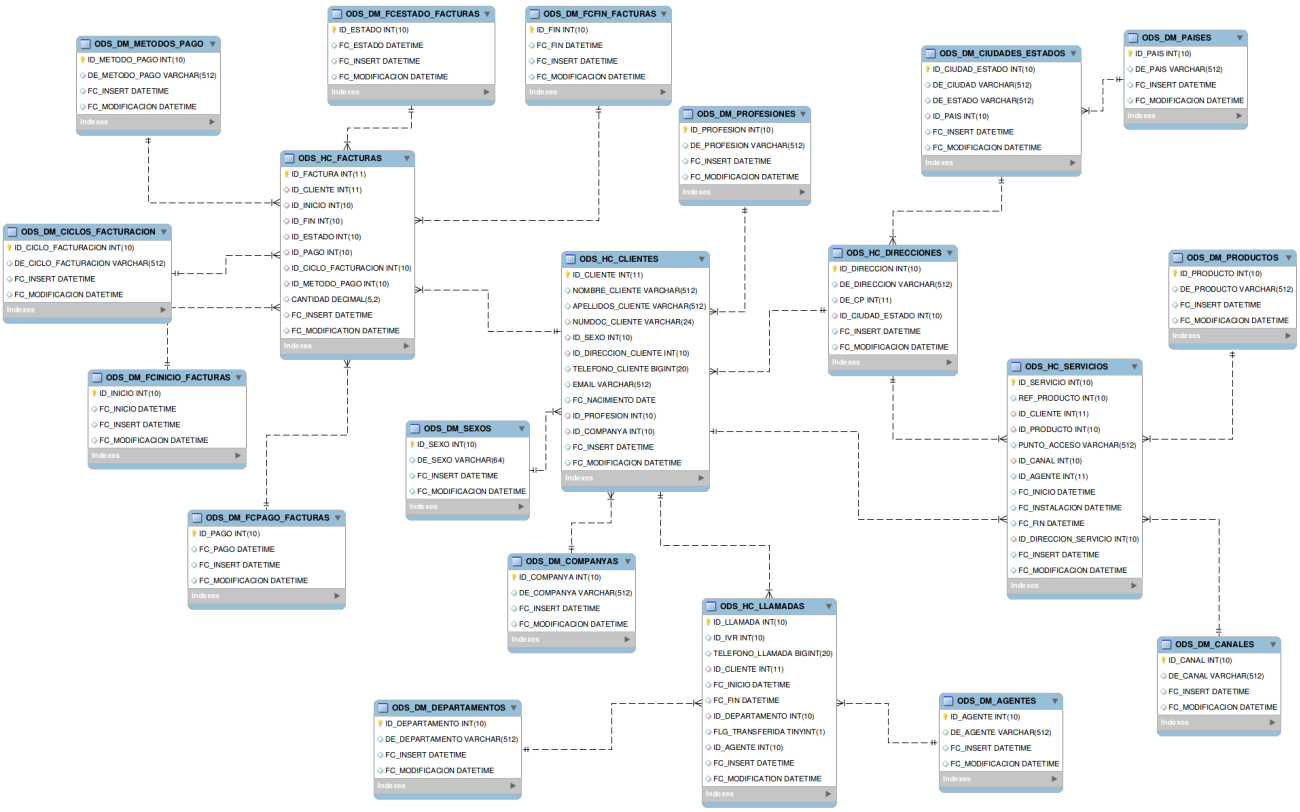

ODS_DM_METODOS_PAGO
ID_METODO_PAGO INT(10)
DE_METODO_PAGO VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_FCESTADO_FACTURAS
ID_ESTADO INT(10)
FC_ESTADO DATETIME
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_FCFIN_FACTURAS
ID_FIN INT(10)
FC_FIN DATETIME
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_PROFESIONES
ID_PROFESION INT(10)
DE_PROFESION VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_CIUDADES_ESTADOS
ID_CIUDAD_ESTADO INT(10)
DE_CIUDAD VARCHAR(512)
DE_ESTADO VARCHAR(512)
ID_PAIS INT(10)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_PAISES
ID_PAIS INT(10)
DE_PAIS VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_HC_FACTURAS
ID_FACTURA INT(11)
ID_CLIENTE INT(11)
ID_INICIO INT(10)
ID_FIN INT(10)
ID_ESTADO INT(10)
ID_PAGO INT(10)
ID_CICLO_FACTURACION INT(10)
ID_METODO_PAGO INT(10)
CANTIDAD DECIMAL(5,2)
FC_INSERT DATETIME
FC_MODIFICATION DATETIME
ODS_DM_CICLOS_FACTURACION
ID_CICLO_FACTURACION INT(10)
DE_CICLO_FACTURACION VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_HC_CLIENTES
ID_CLIENTE INT(11)
NOMBRE_CLIENTE VARCHAR(512)
APELLIDOS_CLIENTE VARCHAR(512)
NUMDOC_CLIENTE VARCHAR(24)
ID_SEXO INT(10)
ID_DIRECCION_CLIENTE INT(10)
TELEFONO_CLIENTE BIGINT(20)
EMAIL VARCHAR(512)
FC_NACIMIENTO DATE
ID_PROFESION INT(10)
ID_COMPANYA INT(10)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_HC_DIRECCIONES
ID_DIRECCION INT(10)
DE_DIRECCION VARCHAR(512)
DE_CP INT(11)
ID_CIUDAD_ESTADO INT(10)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_PRODUCTOS
ID_PRODUCTO INT(10)
DE_PRODUCTO VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_FCINICIO_FACTURAS
ID_INICIO INT(10)
FC_INICIO DATETIME
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_SEXOS
ID_SEXO INT(10)
DE_SEXO VARCHAR(64)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_HC_SERVICIOS
ID_SERVICIO INT(10)
REF_PRODUCTO INT(10)
ID_CLIENTE INT(11)
ID_PRODUCTO INT(10)
PUNTO_ACCESO VARCHAR(512)
ID_CANAL INT(10)
ID_AGENTE INT(11)
FC_INICIO DATETIME
FC_INSTALACION DATETIME
FC_FIN DATETIME
ID_DIRECCION_SERVICIO INT(10)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_FCPAGO_FACTURAS
ID_PAGO INT(10)
FC_PAGO DATETIME
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_COMPANYAS
ID_COMPANYA INT(10)
DE_COMPANYA VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_HC_LLAMADAS
ID_LLAMADA INT(10)
ID_IVR INT(10)
TELEFONO_LLAMADA BIGINT(20)
ID_CLIENTE INT(11)
FC_INICIO DATETIME
FC_FIN DATETIME
ID_DEPARTAMENTO INT(10)
FLG_TRANSFERIDA TINYINT(1)
ID_AGENTE INT(10)
FC_INSERT DATETIME
FC_MODIFICATION DATETIME
ODS_DM_CANALES
ID_CANAL INT(10)
DE_CANAL VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_DEPARTAMENTOS
ID_DEPARTAMENTO INT(10)
DE_DEPARTAMENTO VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME
ODS_DM_AGENTES
ID_AGENTE INT(10)
DE_AGENTE VARCHAR(512)
FC_INSERT DATETIME
FC_MODIFICACION DATETIME

TABLA	FILAS
ODS_DM_AGENTES	595
ODS_DM_CANALES	6
ODS_DM_CICLOS_FACTURACION	4
ODS_DM_CIUDADES_ESTADOS	84
ODS_DM_COMPANYAS	385
ODS_DM_DEPARTAMENTOS	8
ODS_DM_FCESTADO_FACTURAS	42
ODS_DM_FCFIN_FACTURAS	22
ODS_DM_FCINICIO_FACTURAS	42
ODS_DM_FCPAGO_FACTURAS	402
ODS_DM_METODOS_PAGO	5
ODS_DM_PAISES	3
ODS_DM_PRODUCTOS	8
ODS_DM_PROFESIONES	197
ODS_DM_SEXOS	4
ODS_HC_CLIENTES	17,559
ODS_HC_DIRECCIONES	17,499
ODS_HC_FACTURAS	420000
ODS_HC_LLAMADAS	202717
ODS_HC_SERVICIOS	78467

¿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: PAÍS → ESTADOS → CIUDADES → DIRECCIONES

Por que hay nombres de ciudades que se repiten en diferentes estados. Por ejemplo, tenemos la ciudad de Glendale en los estados de Arizona y California. Esto es algo bastante común en Estados Unidos. Por ejemplo, Springfield. La ciudad de la serie de televisión Los Simpsons. Hay 34 estados en Estados Unidos que tienen al menos una comunidad con este nombre.

Se podría haber hecho una tabla de estados y otra de ciudades, pero hubiéramos necesitado una tabla más para gestionar las relación varios a varios de estas dos tablas.

Separar el campo DE_DIRECCION de la tabla de direcciones en dos campos: NOMBRE_VIA y NUM_VIA

Vemos que las direcciones tienen todas el mismo formato. Primero viene el número y después de nombre de la calle. Por lo tanto, para obtener el número cogemos la cadena de la dirección hasta el primer carácter espacio. Para obtener el número de la vía cogemos la cadena de la dirección a partir del primer espacio.

Con esta información he creado un script que crea los campos NOMBRE_VIA y NUM_VIA a la tabla ODS_HC_DIRECCIONES y a continuación un update para actualizar estos campos con sus valores correspondientes:

```
USE ODS;

ALTER TABLE ODS_HC_DIRECCIONES
ADD NOMBRE_VIA VARCHAR(512) NOT NULL AFTER DE_DIRECCION,
ADD NUM_VIA INT NOT NULL AFTER NOMBRE_VIA;

UPDATE ODS_HC_DIRECCIONES DIR1
INNER JOIN ODS_HC_DIRECCIONES DIR2 ON DIR1.ID_DIRECCION = DIR2.ID_DIRECCION
SET DIR1.NUM_VIA = CASE DIR2.DE_DIRECCION
    WHEN 'NO APLICA' THEN 99998
    WHEN 'DESCONOCIDO' THEN 99999
    ELSE SUBSTRING_INDEX(DIR2.DE_DIRECCION, ' ', 1) END,
DIR1.NOMBRE_VIA = CASE DIR2.DE_DIRECCION
    WHEN 'NO APLICA' THEN 'NO APLICA'
    WHEN 'DESCONOCIDO' THEN 'DESCONOCIDO'
    ELSE SUBSTRING(DIR2.DE_DIRECCION, LENGTH(SUBSTRING_INDEX(DIR2.DE_DIRECCION, ' ',
1)) + 2) END
```

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

Por Data Quality o datos de calidad entendemos que son fiables, actuales, útiles, bien estructurados, etc.

Vamos a analizar los distintos sistemas de los que proviene la información de la práctica:

Facturador

A nivel de estructura la tabla es correcta.

Tiene una clave primaria única, una foreign key a la tabla de clientes, campos de fechas propios de una factura, importe, etc.

Todos los campos tienen valores por lo que no vamos a encontrar valores del tipo "DESCONOCIDO" o "NO APLICA".

El principal problema de esta tabla es la foreign key a la tabla de clientes que no siempre cumple integridad referencial. Analizando la tabla hay 2442 valores en 48840 registros de la tabla de facturas que no tienen correspondencia en la tabla de clientes. Posiblemente sean facturas de clientes que se han dado de baja y ya no aparecen en la tabla de clientes. En cualquier caso, esos registros pueden invalidar en gran medida la funcionalidad del modelo y la capacidad de hacer BI con este Data Warehouse.

IVR

Esta tabla es bastante pobre en términos de calidad. Por el contenido deducimos que en esta tabla se almacenan las llamadas de los clientes y se registran los departamentos de la empresa por los que va pasando la llamada.

El primer problema es que no tiene una clave primaria.

Aunque parecen las llamadas de los clientes, no hay ningún campo que identifique al cliente que interviene en la llamada, lo cual entiendo que es un dato fundamental para poder hacer BI con los datos de esta tabla.

Hay algunos campos, como el teléfono, la fecha de fin o el agente que atiende la llamada que tienen valores vacíos. Estos campos entendemos que siempre tienen que tener valor por la naturaleza del dato. Posiblemente se deban a algún error en el origen de los datos.

Conclusiones

Como conclusión, tenemos un gran problema para relacionar los datos de las facturas y las llamadas con los clientes. Creo que sería lo primero que habría que solucionar de cara a empezar a tener datos de calidad. Principalmente en la tabla de llamadas. Hay otros problemas con datos vacíos, pero tienen una entidad menor respecto a las realizaciones con los clientes.

Master Data

Hemos podido ver varios casos en los que se guardan los mismo datos en diferentes sistemas e incluso en algunos casos con valores distintos.

Por ejemplo, el caso de direcciones, códigos postales, ciudades, estados y países. Sería conveniente tener unas tablas maestras con callejeros, una relación de ciudades y estados y otras de países. El caso de países quizá se podría incluso evitar dado que por ahora el negocio se centra en un único país. En cualquier caso, son datos de dominio publico fáciles de adquirir, por lo que no habría problemas en poblar estas posibles tablas.

Por otro lado, se necesita una tabla maestra de personal donde guardar datos de los empleados de la empresa. En productos se hace referencia a agentes usando un código y en llamadas se hace referencia a agentes por un nombre de usuario o algo similar. Teniendo una tabla maestra con los agentes de la empresa podríamos ver posibles relaciones de estas tablas.

Data Modeling & Design (Tablas en origen)

Como ha quedado claro, sobre todo en la parte de llamadas, no se están aplicando las formas normales en los operacionales. De no haber sido, así nos hubiéramos ahorrado algunos problemas.

Por ejemplo, el campo ID de la tabla de contactos. Con ese nombre parece que se refiere a una PK, pero resulta que tiene valores repetidos, lo que es incompatible con la NF1.

También encontramos que hay FKs que no cumplen las reglas de integridad referencial.

Si estas cosas que parecen básicas se hubieran cumplido, hubiéramos visto de una pasada como se relacionan las tablas y nos habría evitado tener que realizar algunas operaciones para evitar en cierta medida estos problemas.

Entiendo que estos problemas son normales en cualquier organización y que una de las labores del departamento del datos es ir corrigiéndolos.

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

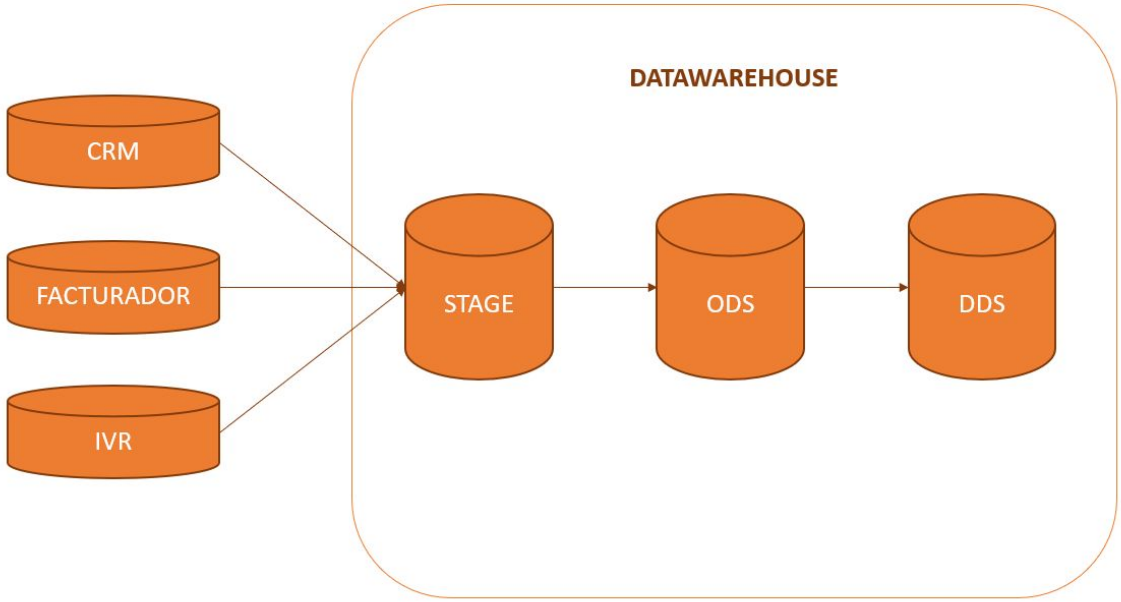
Poco más que añadir a lo explicado anteriormente, aunque siempre hay pequeñas cosas que mejorar. Si se consigue arreglar los problemas detectados en los sistemas de origen, conseguiremos un gran avance.

Si que aconsejaría que usaran los tipos de datos que para algo están. Todos los campos son varchar de 512 caracteres. No estaría mal que usaran cuando corresponda tipos INT, DATE, DATETIME, etc. Y los que realmente sean de tipo carácter, en muchos casos seguramente se pueda reducir el espacio de los campos y bajar de los 512 caracteres.

Ya puestos podrían poner como no nulos los campos obligatorios y establecer valores por defecto.

Cuarta parte

Después de todo lo visto nuestro ecosistema quedaría así:

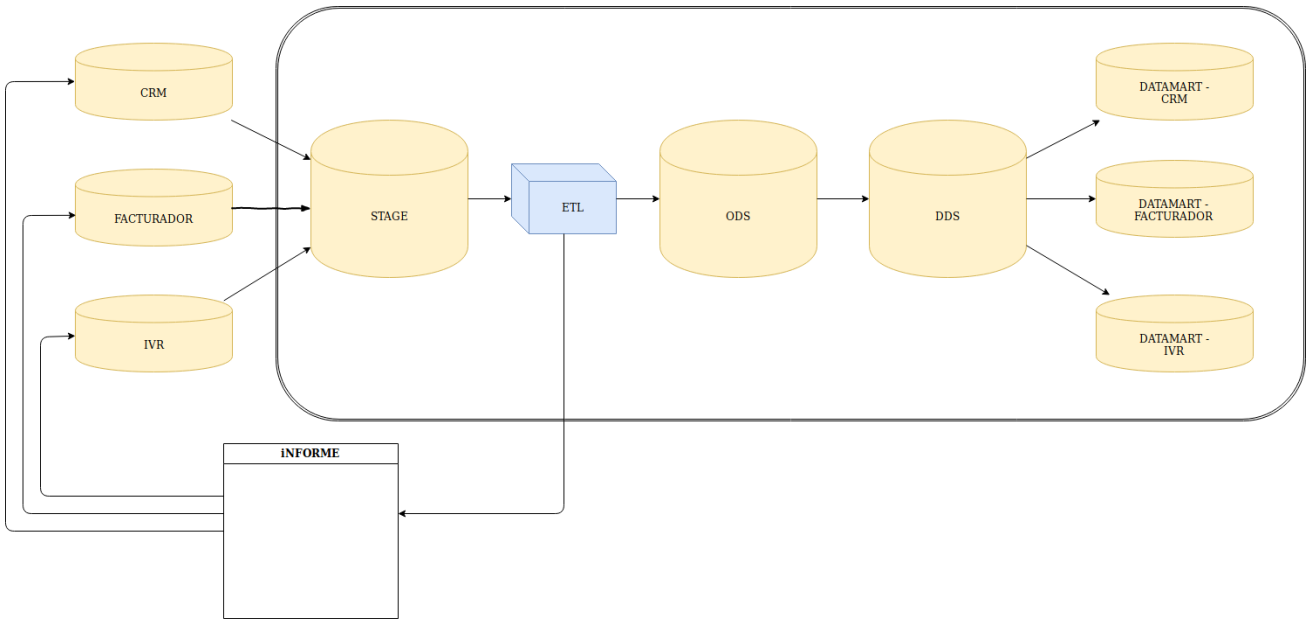


¿Lo dejarías así o plantearías otro diseño mejorado? Si es que sí, esbózalo(no te ibas a librar :P)

Mis cambios en el ecosistema

Desde mi punto de vista, optaría por un modelo Top-Down y añadiría un Data Mart para cada departamento del que parten los datos. Así, cada departamento podría acceder a su propio Data Mart y hacer consultas o obtener informes desde su propio Data Mart. De esta manera, actuando en su propio Data Mart, no tienen que estar accediendo a nuestras bases de datos.

Para la práctica hemos realizado todas las consultas a mano, pero a lo mejor se podría haber un software de ETL para realizar la carga de datos en ODS y DDS. Además, dadas la posibilidades de automatización de estos tipos de software, podríamos generar un informes con los resultados de la ejecución de la carga de datos. Uno de estos informes podría contener estadísticas globales de los datos guardados y de los problemas o errores que se puedan encontrar. Otro informe puede ser para los sistemas de origen informándoles de problemas de los datos como valores nulos, FKs huérfanas, etc.



Quinta parte

Escribe tus propias reglas o mandamientos de un DataWarehouse

Creo que hubiera estado bastante bien poder llegar más lejos y haber podido montar la parte de DDS, algo de explotación de los datos e incluso llegar a los Data Marts como propongo en el punto anterior, pero entiendo que el desarrollo de la práctica hubiera sido demasiado extenso y algunas partes de ese desarrollo llegaremos próximamente en el Bootcamp.

Lo comento principalmente porque en la parte de facturas he optado por un modelo en el que he sacado dimensiones prácticamente de todos los campos y creo que me falta información para saber si he tomado una buena decisión o no. Al final, la experiencia es la que te ayuda a tomar estas decisiones, así que en mi caso voy a ciegas.

En la carga de datos, al tener más tablas, se hace más pesada la ejecución de los acripts ya que tiene que realizar más cruces de tablas. Si con eso conseguimos que a la hora de consultar los datos en el DDS o en los Data Marts las consultas sean más ágiles a lo mejor merece la pena penalizar la carga de datos y favorecer las consultas.

Al incrementarse los tiempos de carga por el esquema que he planteado, estaría bien saber el volumen de datos que vamos a recibir diariamente. Al ser facturas de un empresa de telecomunicaciones y viendo el contenido de los datos, entiendo que se emiten facturas dos veces al mes y unas 10500 facturas cada ciclo. Así que, después de la carga inicial, las cargas increméntales no deberían ser muy pesadas, aunque no estoy teniendo en cuenta el posible incremento de clientes y por lo tanto de facturas emitidas.

Dicho todo esto, en función de los resultados obtenidos, una regla estaría dedicada a las dimensiones. Si el camino por que el que he optado es el correcto, la recomendación general sería sacar dimensiones de todos lo campos que se crea conveniente en función del número de distintos valores que tenga en la tabla. Por el contrario, si la decisión fuera errónea, la regla sería sacar dimensiones cuando se crea necesario pero no hacerlo a lo loco y comprobar antes el impacto que pueda tener en la carga de datos.

Otra regla, creo fundamental, es estudiar bien los datos de origen. Obtener el número de valores distintos de cada campo para saber si tenemos que sacar una dimensión de un un campo está bien, pero también hay que ver, por ejemplo, si un campo es una foreign key a un tabla de hechos si todos los valores tienen correspondencia para ver la calidad de los datos de origen. Tenemos como ejemplo la tabla de CIUDADES-ESTADOS. Posiblemente, si no hubiéramos revisado el contenido, no hubiéramos visto que hay nombres ciudades que se repiten en distintos estados y seguramente el modelo planteado podría haber sido incorrecto. También tenemos el ejemplo de países. Aunque solo tenemos un país en los datos, en una tabla de origen venía informado con la iniciales:"US". Y en otra como "United states". Es el mismo país con dos denominaciones distintas. Por lo tanto, la conclusión es que hay que estudiar los valores de los campos clave para poder actuar en consecuencia y modelar o tratar los datos de la mejor manera.

Posiblemente hay que tener más cosas en cuenta, pero estas dos son las que veo más importantes para crear un buen modelo y crear las cargas de datos adecuadas. Seguramente de cualquier proyecto real de grandes dimensiones empezaríamos a sacar algunas reglas o mandamientos mas, pero como aproximación creo que ha estado bien el ejercicio.