



Gestión de Datos

Concesionaria 2C2020

Grupo: Empanada de Mondongo

Alumnos:

- Alexis Herasimiuk
- Nicolás Rickert
- Luis Pulgar
- Martín Mouriño

Fecha de entrega:

- Entrega N°1 - DER - 04/10/2020
- Entrega N°2 - Migración - 26/10/2020
- Entrega N°3 - Business Intelligence - 30/11/2020

Índice

→ Diagrama de Entidad - Relación	2
◆ Cliente	3
◆ Sucursal	4
◆ Tipo Auto	5
◆ Tipo Caja	5
◆ Tipo Transmisión	6
◆ Tipo Motor	6
◆ Auto	7
◆ Fabricante	8
◆ Modelo	8
◆ Autoparte	9
◆ Venta	10
◆ Compra	11
→ Migración	
◆ Name Refactoring	12
◆ Borrado previo	13
◆ Creación de Esquema y Tablas	14
◆ Constraints	15
◆ Foreign Key	16
◆ Funciones	17
◆ Modelo	18
◆ Fabricante	18
◆ Cliente	19
◆ Sucursal	19
◆ Tipo Auto/Caja/Transmisión	20
◆ Auto	20
◆ Autoparte	21
◆ Compra	22
◆ Compra Auto	22
◆ Compra Ítem	23
◆ Factura	25
◆ Factura Auto	25
◆ Factura Ítem	26
◆ Índices	27
◆ Vistas	28
→ Business Intelligence	
◆ DER	33
◆ Borrado Previo	33
◆ Modelo Estrella	33
◆ Fact Tables	33
◆ Migración	35
◆ Dimensiones	37
◆ Tablas de Hechos	41
◆ Cargas de datos	43
◆ Vistas	45

Diagrama de Entidad - Relación

Para ver el diagrama completo en su resolución original [click aquí](#), o bien, el archivo adjunto que acompaña esta entrega. El mismo no se incrustará entero en el documento debido a su gran tamaño horizontal.

En este apartado, se detallarán las decisiones tomadas en las distintas fases de normalización de la Base de datos.

Inicialización de la tabla maestra.

En su estado inicial, se parte de una tabla maestra la cual consta de los siguientes atributos:

CLIENTE_APELLIDO	CLIENTE_NOMBRE	CLIENTE_DIRECCION	CLIENTE_DNI
CLIENTE_FECHA_NAC	CLIENTE_MAIL	COMPRA_FECHA	COMPRA_NRO
SUCURSAL_DIRECCION	SUCURSAL_MAIL	SUCURSAL_TELEFONO	SUCURSAL_CIUADAD
COMPRA_PRECIO	COMPRA_CANT	AUTO_NRO_CHASIS	AUTO_NRO_MOTOR
AUTO_PATENTE	AUTO_FECHA_ALTA	AUTO_CANT_KMS	MODELO_CODIGO
MODELO_NOMBRE	MODELO_POTENCIA	TIPO_TRANSMISION_CODIGO	TIPO_TRANSMISION_DESC
TIPO_CAJA_CODIGO	TIPO_CAJA_DESC	TIPO_MOTOR_CODIGO	TIPO_AUTO_CODIGO
TIPO_AUTO_DESC	FABRICANTE_NOMBRE	AUTO_PARTE_CODIGO	AUTO_PARTE_DESCRIPCION
PRECIO_FACTURADO	CANT_FACTURADA	FACTURA_FECHA	FACTURA_NRO
FAC_CLIENTE_APELLIDO	FAC_CLIENTE_NOMBRE	FAC_CLIENTE_DIRECCION	FAC_CLIENTE_DNI
FAC_CLIENTE_FECHA_NAC	FAC_CLIENTE_MAIL	FAC_SUCURSAL_DIRECCION	FAC_SUCURSAL_MAIL
FAC_SUCURSAL_TELEFONO	FAC_SUCURSAL_CIUADAD		

Proceso de normalización

Tal y como sugiere el [apunte de normalización](#) provisto por la cátedra, se procede a normalizar la base de datos de forma no categorizada, (no se siguen algorítmicamente los pasos de 1FN, 2FN, etc), sino que se realiza en base a cómo van surgiendo las distintas situaciones.

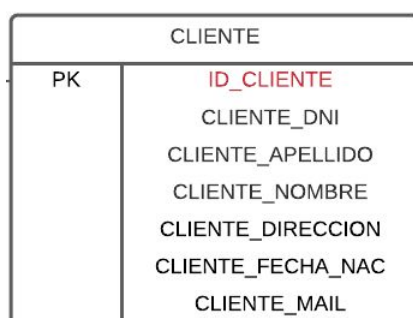
Tabla Cliente

Se detecta una redundancia de campos que refieren a los atributos de un cliente.

CLIENTE_APELLIDO	CLIENTE_NOMBRE
CLIENTE_FECHA_NAC	CLIENTE_MAIL
CLIENTE_DIRECCION	CLIENTE_DNI

FAC_CLIENTE_APELLIDO	FAC_CLIENTE_NOMBRE
FAC_CLIENTE_FECHA_NAC	FAC_CLIENTE_MAIL
FAC_CLIENTE_DIRECCION	FAC_CLIENTE_DNI

Finalmente se define una única tabla CLIENTE:



Todas las tablas que referencian a un CLIENTE deberán tener una Foreign Key **ID_CLIENTE**.


Se decide usar como Primary Key **ID_CLIENTE**, ya que se han detectado incoherencias semánticas en la implementación de la tabla maestra, por ejemplo DNI duplicados, lo que da la pauta que este atributo no puede ser una Primary Key.

```

SELECT CLIENTE_APELLIDO, CLIENTE_NOMBRE, CLIENTE_DNI
FROM gd_esquema.Maestra
WHERE CLIENTE_DNI = 1251711

SELECT FAC_CLIENTE_APELLIDO, FAC_CLIENTE_NOMBRE, FAC_CLIENTE_DNI
FROM gd_esquema.Maestra
WHERE FAC_CLIENTE_DNI = 1251711

```



CLIENTE_APELLIDO	CLIENTE_NOMBRE	CLIENTE_DNI
González	ARCANGEL	1251711

FAC_CLIENTE_APELLIDO	FAC_CLIENTE_NOMBRE	FAC_CLIENTE_DNI
Palma	TERESA	1251711

En las consultas detalladas, se muestra que tanto González Arcangel como Palma Teresa tienen el mismo DNI y puede que esta situación se repita reiteradas veces.

Tabla Sucursal

Se detecta una redundancia de campos que refieren a los atributos de una sucursal.

SUCURSAL_DIRECCION	SUCURSAL_MAIL
SUCURSAL_TELEFONO	SUCURSAL_CIUADAD

FAC_SUCURSAL_DIRECCION	FAC_SUCURSAL_MAIL
FAC_SUCURSAL_TELEFONO	FAC_SUCURSAL_CIUADAD

Finalmente se define una única tabla SUCURSAL:

SUCURSAL	
PK	ID_SUCURSAL SUCURSAL_DIRECCION SUCURSAL_MAIL SUCURSAL_TELEFONO SUCURSAL_CIUADAD

Todas las tablas que referencian a una SUCURSAL deberán tener una Foreign Key **ID_SUCURSAL**.

Se decide usar como Primary Key **ID_SUCURSAL**, ya que no hay campos candidatos en la tabla a identificar unívocamente a una sucursal. Los atributos definidos para una sucursal pueden cambiar, o bien pueden repetirse.

Tabla Tipo Auto

Se detecta una dependencia funcional de campos que definen el tipo de un auto.

$\text{TIPO_AUTO_CODIGO} \rightarrow \text{TIPO_AUTO_DESC}$

Para todo código de tipo de auto existe una única descripción.

Finalmente se define una tabla TIPO_AUTO

TIPO_AUTO	
PK	TIPO_AUTO_CODIGO TIPO_AUTO_DESC

Todas las tablas que referencian al tipo de un auto deberán tener una Foreign Key **TIPO_AUTO_CODIGO**.

Se decide usar como Primary Key **TIPO_AUTO_CODIGO**, ya que es un valor que identifica unívocamente cada uno de los tipos de auto, sin repeticiones.

Tabla Tipo Caja

Se detecta una dependencia funcional de campos que definen el tipo de caja.

$\text{TIPO_CAJA_CODIGO} \rightarrow \text{TIPO_CAJA_DESC}$

Para todo código de caja existe una única descripción.

Finalmente se define una tabla TIPO_CAJA

TIPO_CAJA	
PK	TIPO_CAJA_CODIGO TIPO_CAJA_DESC

Todas las tablas que referencian al tipo de caja de un auto deberán tener una Foreign Key **TIPO_CAJA_CODIGO**.

Se decide usar como Primary Key **TIPO_CAJA_CODIGO**, ya que es un valor que identifica unívocamente cada uno de los tipos de caja, sin repeticiones.

Tabla Tipo Transmisión

Se detecta una dependencia funcional de campos que definen el tipo de transmisión de un auto.

TIPO_TRANSMISIÓN_CODIGO → TIPO_TRANSMISIÓN_DESC

Para todo código de tipo de transmisión existe una única descripción.

Finalmente se define una tabla TIPO_TRANSMISIÓN

TIPO_TRANSMISION	
PK	TIPO_TRANSMISION_CODIGO
	TIPO_TRANSMISION_DESC

Todas las tablas que referencian al tipo de transmisión de un auto deberán tener una Foreign Key **TIPO_TRANSMISIÓN_CODIGO**.

Se decide usar como Primary Key **TIPO_TRANSMISIÓN_CODIGO**, ya que es un valor que identifica unívocamente cada uno de los tipos de transmisión, sin repeticiones.

Tabla Tipo Motor

Se detecta una hipotética y predecible dependencia funcional de campos que definen el tipo de motor.

TIPO_MOTOR_CODIGO → TIPO_MOTOR_DESC

Para todo código de motor existiría una única descripción.

En la tabla maestra no existe el atributo TIPO_MOTOR_DESC, pero dada la marcada tendencia, es posible que en un futuro se agregue la descripción del motor al igual que en el resto de las especificaciones (TIPO_AUTO, TIPO_CAJA, y TIPO_TRANSMISIÓN) como [aquí](#) se detalla. Por otra parte, el TIPO_MOTOR_CODIGO, es un código que no permite saber de qué motor se trata (es decir, para quien accede a la base de datos, no identifica cuales son las diferencias entre, por ejemplo, el motor de código 1001 y 1002).

De todas formas, si se hiciese una tabla para el motor, quedaría de un único atributo, lo cual no tendría una funcionalidad útil en el desarrollo del TP. Si en algún momento se quisiese agregar un atributo al motor, efectivamente se creará una tabla en donde **TIPO_MOTOR_CÓDIGO** sea la Primary Key, y en cada tabla que referencie a un tipo de motor, debe contener la respectiva Foreign Key

Tabla Auto

Se detectan dependencias funcionales de atributos que definen un Auto

AUTO_PATENTE → (AUTO_NRO_CHASIS, AUTO_NRO_MOTOR, AUTO_FECHA_ALTA, AUTO_CANT_KMS, TIPO_AUTO_CODIGO, TIPO_MOTOR_CODIGO, TIPO_TRANSMISION_CODIGO, TIPO_CAJA_CODIGO, MODELO_CODIGO, ID_FABRICANTE)

Para una AUTO_PATENTE existe un único número de chasis, número de motor, una fecha de alta, una cantidad de kms determinada, un código de auto, un código de motor, un código de transmisión, un tipo de caja, un modelo y un fabricante

Finalmente se define la tabla AUTO

AUTO	
PK	AUTO_PATENTE
	AUTO_NRO_CHASIS
	AUTO_NRO_MOTOR
	AUTO_FECHA_ALTA
	AUTO_CANT_KMS
FK	TIPO_AUTO_CODIGO
	TIPO_MOTOR_CODIGO
FK	TIPO_TRANSMISION_CODIGO
FK	TIPO_CAJA_CODIGO
FK	MODELO_CODIGO
FK	ID_FABRICANTE

Se decide usar como Primary Key **AUTO_PATENTE**, ya que es un valor que identifica unívocamente cada uno de los autos, sin repeticiones.

TIPO_AUTO_CODIGO es una Foreign Key que permite conocer mediante la tabla TIPO_AUTO, la descripción del tipo de auto, y otros atributos que se definan en el futuro.

TIPO_TRANSMISIÓN_CODIGO es una Foreign Key que permite conocer mediante la tabla TIPO_TRANSMISIÓN, la descripción del tipo de transmisión del auto, y otros atributos que se definan en el futuro.

TIPO_CAJA_CODIGO es una Foreign Key que permite conocer mediante la tabla TIPO_CAJA, la descripción del tipo de caja del auto, y otros atributos que se definan en el futuro.

Tabla Fabricante

Se decide normalizar el fabricante, que actualmente posee su nombre como único atributo; pese a que éste sea único e irrepetible, se decide generar un nuevo atributo que lo identifique de tipo entero, de manera tal que sea más sencillo de indexar en caso de que sea necesario, o a modo prevención ante eventuales cambios de nombre de los distintos fabricantes, o que sea más sencilla la obtención de los nombres de los fabricantes para, por ejemplo, insertarlos como valores posibles del campo de un formulario.

Finalmente se define una tabla FABRICANTE

FABRICANTE	
PK	ID_FABRICANTE FABRICANTE_NOMBRE

Todas las tablas que referencian a un fabricante deberán tener una Foreign Key **ID_FABRICANTE**.

Tabla Modelo

Se detecta una dependencia funcional en los atributos que definen un modelo

$\text{MODELO_CODIGO} \rightarrow \text{MODELO_NOMBRE}, \text{MODELO_POTENCIA}$

Para todo código de modelo existe un único nombre y una potencia determinada.

Finalmente se define una tabla MODELO

MODELO	
PK	MODELO_CODIGO MODELO_NOMBRE MODELO_POTENCIA

Todas las tablas que referencian a un modelo deberán tener una Foreign Key **MODELO_CÓDIGO**.

Se decide usar como Primary Key **MODELO_CODIGO**, ya que es un valor que identifica unívocamente cada uno de los modelos, sin repeticiones.

Tabla Auto parte

Se detecta una dependencia funcional en los atributos que definen una autoparte

$AUTO_PARTE_CODIGO \rightarrow AUTO_PARTE_DESCRIPCION,$
 $AUTO_PARTE_CATEGORIA, ID_FABRICANTE, MODELO_CODIGO$

Para todo código de autoparte existe una única descripción, una categoría, un fabricante y es de un modelo determinado.

Finalmente se define una tabla AUTO_PARTE

AUTO_PARTE	
PK	AUTO_PARTE_CODIGO
	AUTO_PARTE_DESCRIPCION
	AUTO_PARTE_CATEGORIA
FK	ID_FABRICANTE
FK	MODELO_CODIGO

Todas las tablas que referencian a una autoparte deberán tener una Foreign Key **AUTO_PARTE_CODIGO**.

Se decide usar como Primary Key **AUTO_PARTE_CODIGO**, ya que es un valor que identifica unívocamente cada uno de las autopartes, sin repeticiones.

Se agrega el campo AUTO_PARTE_CATEGORIA, el cual no estaba en la tabla maestra, pero se lo referencia en el enunciado.

Venta

Se decide estructurar una venta en forma de Tipo - Subtipo. Aquellos atributos que dependen funcionalmente de FACTURA_NRO y son inherentes a cualquier venta, forman parte de la tabla FACTURA:

FACTURA	
PK	FACTURA_NRO FACTURA_FECHA
FK	ID_CLIENTE
FK	ID_SUCURSAL

Luego, se definen dos tablas hijas que corresponden a la venta de un auto o de una autoparte:

VENTA_AUTO	
PK,FK	FACTURA_NRO
FK	AUTO_PATENTE PRECIO_FACTURADO

VENTA_ITEM	
PK,FK	FACTURA_NRO
PK	NRO_ITEM
FK	AUTO_PARTE_CODIGO CANT_FACTURADA PRECIO_FACTURADO

En donde **FACTURA_NRO** es no solamente Primary Key de cada subtipo de la factura sino también la Foreign Key, responsable de referenciar al supertipo Factura.

Se considera posible vender varias autopartes en una sola venta, mas no así varios autos. Por tal motivo, se agrega una Primary Key **NRO_ITEM** para identificar unívocamente cada ítem de una factura.

Foreign Keys:

FACTURA_NRO permite referenciar la factura de la que se trata la venta.

ID_CLIENTE permite referenciar al cliente que realiza la compra.

ID_SUCURSAL permite referenciar a la sucursal que realiza la venta.

AUTO_PATENTE permite referenciar el auto que interviene en la venta.

AUTO_PARTE_CODIGO permite referenciar la autoparte que interviene en la venta.

Compra

Se decide estructurar una compra en forma de Tipo - Subtipo. Aquellos atributos que dependen funcionalmente de COMPRA_NRO y son inherentes a cualquier compra, forman parte de la tabla COMPRA:

COMPRA	
PK	COMPRA_NRO
	COMPRA_FECHA
FK	ID_SUCURSAL
FK	ID_CLIENTE

Luego, se definen dos tablas hijas que corresponden a la compra de un auto o de una autoparte:

COMPRA_AUTO	
PK,FK	COMPRA_NRO
FK	AUTO_PATENTE
	COMPRA_PRECIO

COMPRA_ITEM	
PK,FK	COMPRA_NRO
PK	NRO_ITEM
FK	AUTO_PARTE_CODIGO
	COMPRA_CANT
	COMPRA_PRECIO

En donde **COMPRA_NRO** es no solamente Primary Key de cada subtipo de la compra sino también la Foreign Key, responsable de referenciar al supertipo COMPRA.

Se considera posible comprar varias autopartes en una sola compra, mas no así varios autos. Por tal motivo, se agrega una Primary Key **NRO_ITEM** para identificar unívocamente cada ítem de una compra.

COMPRA_NRO permite referenciar la compra de la que se trata.

ID_CLIENTE permite referenciar al cliente al que se le realiza la compra.

ID_SUCURSAL permite referenciar a la sucursal que realiza la compra.

AUTO_PATENTE permite referenciar el auto que interviene en la compra.

AUTO_PARTE_CODIGO permite referenciar la autoparte que interviene en la compra.

Migración

Para ver el diagrama completo en su resolución original [click aquí](#), o bien, el archivo adjunto que acompaña esta entrega. El mismo no se incrustará entero en el documento debido a su gran tamaño horizontal.

En este apartado, se detallarán las decisiones tomadas en el desarrollo de la migración de datos en orden tal como se encuentra en el SCRIPT.

Name Refactoring

Previo al desarrollo del script de migración de base de datos, se necesitó refactorizar los nombres que se han visto en el módulo anterior. Dado que la tabla maestra era un conjunto de datos que podrían repetirse en distintas columnas teniendo distintos significados, era entendible que cada columna tuviese en su nombre algún prefijo descriptivo que no es necesario y perjudica la comprensión y la legibilidad en el modelo ya migrado. Para este proceso se decide que todas las tablas y columnas tendrán su nombre sin prefijos ni sufijos, en minúscula y serán palabras separadas por guión bajo para facilitar la legibilidad, unificar convenciones y diferenciar de una manera más sencilla de las palabras reservadas del lenguaje SQL.

Antiguo Nombre	Nuevo Nombre	Antiguo Nombre	Nuevo Nombre
CLIENTE_DNI	dni	TIPO_AUTO_DESC	descripcion
CLIENTE_APELLIDO	apellido	TIPO_CAJA_DESCRIPCION	descripcion
CLIENTE_NOMBRE	nombre	TIPO_TRANSMISION_DESCRIPCION	descripcion
CLIENTE_DIRECCIÓN	dirección	FACTURA_NRO	nro_factura
CLIENTE_FECHA_NAC	fecha_nac	FACTURA_FECHA	fecha
CLIENTE_MAIL	mail	CANT_FACTURADA	cant
SUCURSAL_DIRECCION	direccion	PRECIO_FACTURADO	precio
SUCURSAL_MAIL	mail	AUTO_PATENTE	patente_auto
SUCURSAL_TELEFONO	telefono	AUTO_NRO_CHASIS	nro_chasis
SUCURSAL_CIUDAD	ciudad	AUTO_NRO_MOTOR	nro_motor
AUTO_FECHA_ALTA	fecha_alta	COMPRA_PRECIO	precio
AUTO_CANT_KMS	cant_kms	MODELO_NOMBRE	nombre
COMPRA_NRO	nro_compra	MODELO_POTENCIA	potencia
FABRICANTE_NOMBRE	nombre	AUTO PARTE_DESCRIPCION	descripcion
COMPRA_FECHA	fecha	AUTO PARTE_CODIGO	codigo_autoparte
COMPRA_CANT	cantidad	AUTO PARTE_CATEGORIA	categoría

Borrado previo

Dado que el script fue ejecutado en varias ocasiones durante su desarrollo y que el motor no permite la creación de un objeto hasta que el anterior con el mismo nombre no sea borrado, entonces se decide que al inicio del script de migración se borren absolutamente todos los objetos que podrían, eventualmente, haber sido creados por este mismo proceso. Como esto puede pasar, como también podría ejecutarse una única vez, entonces cada instrucción DROP se encuentra englobada dentro de una instrucción IF, que permite verificar si tal objeto existe y puede ser borrado. De esta manera se confeccionan sentencias con la siguiente forma:

```
IF OBJECT_ID('EMPANADA_DE_MONDONGO.<nombre_del_objeto>') IS NOT  
NULL  
    DROP <TIPO> EMPANADA_DE_MONDONGO.<nombre_del_objeto>;
```

Siendo <nombre_del_objeto> el placeholder del nombre del objeto a borrar, <TIPO> el tipo de objeto (INDEX, PROCEDURE, FUNCTION, TABLE, TRIGGER, VISTA, ETC) y EMPANADA_DE_MONDONGO es el nombre del esquema utilizado.

De la misma manera se borra el esquema:

```
IF EXISTS (SELECT SCHEMA_NAME  
            FROM INFORMATION_SCHEMA.SCHEMATA  
            WHERE SCHEMA_NAME = 'EMPANADA_DE_MONDONGO')  
    DROP SCHEMA EMPANADA_DE_MONDONGO
```

Una vez ejecutado esta parte del script. Todos los objetos creados por el mismo, los cuales serán detallados a continuación, serán borrados, con el objetivo de evitar cualquier error al momento de ejecutar las siguientes instrucciones. Este paso cuesta aproximadamente 4 segundos del tiempo total de ejecución.

Creación del esquema

Dado que el enunciado pide que cada objeto sea creado en el esquema cuyo nombre será el nombre del grupo, cada objeto de los creados a continuación son creados bajo este esquema. Puede ser que a lo largo del documento, para facilitar la lectura y el proceso de documentación se omita el nombre del esquema cuando se nombren los distintos objetos que formarán parte de la migración, aunque todas estas sentencias pueden verse en su versión completa en `script_creacion_inicial.sql`.

Para crear el esquema se utiliza la instrucción:

```
CREATE SCHEMA EMPANADA_DE_MONDONGO;
```

Otra opción sería crear el esquema manualmente, y posteriormente crear un usuario cuyo esquema por defecto sea este, lo que lograría un código más limpio y centrado en lo que realmente importa, sin estar contaminado del nombre del esquema en cada una de las sentencias, repetidas veces. Dado que no se obtuvo respuesta en la [consulta](#) realizada en el grupo de google tal como indica el enunciado antes de la fecha de entrega, se decide tomar el camino que menos problemas puede dar a la hora de ejecutar el script en una base de datos que no se le fue configurado un usuario con default schema.

Creación de tablas

Posteriormente al borrado de todos los objetos que podrían haber quedado residuales de anteriores ejecuciones, se procede a crear las tablas, en orden alfabético en el esquema correspondiente. Los constraints de FOREIGN KEY no son especificados en esta sección puesto que en determinadas ocasiones no es posible crear una tabla con una FK que referencie a una tabla que aún no fue creada. Las tablas son creadas con una sentencia de la siguiente forma:

```
CREATE TABLE <nombre_tabla>(
    <nombre_columna> <TIPO_DATO> PRIMARY KEY,
    <nombre_columna> <TIPO_DATO> <CONSTRAINTS>
);
```

Donde <nombre_columna> son los nombres de columnas que se especifican en la sección de name refactoring, <TIPO DATO> son los tipos de datos, tal cual se encuentran en la tabla maestra (Generalmente NVARCHAR, DECIMAL, DATETIME2) y <CONSTRAINTS> son aquellos constrains que con el grupo se han decidido pertinentes, generalmente NOT NULL o valores DEFAULT, se ampliará más en la sección constraints. Todas las Primary Keys fueron justificadas en la sección anterior, junto con el modelo relacional.

Constraints

En esta sección se detallarán los constraints que se han declarado para cada columna.

NOT NULL:

Se decide que, excepto en ciertas ocasiones, es necesario que todas las columnas sean `NOT NULL`, esto es debido a que no se han encontrado campos nulos en la tabla maestra y considerando el dominio es necesario que esos campos no sean nulos. Por otra parte, este constraint evita errores en la migración, como por ejemplo olvidarse de migrar una columna de una tabla específica. En este caso no se ha definido como `NOT NULL` a la columna `categoría` de la tabla de `autopartes`, ya que esa información no fue provista por la tabla maestra.

DEFAULT:

Aunque la totalidad de campos estaban definidos en la tabla maestra, es decir, no se han encontrado campos nulos. Al momento de insertar nuevos datos, definir un valor por defecto puede ayudar a simplificar los `INSERT`. Por ejemplo se ha definido la fecha actual, como valor por defecto para todos los campos `fecha`, y el valor 1 como cantidad por defecto para todos los campos `cantidad`.

IDENTITY:

El `IDENTITY` permite ir incrementando en un valor definido (generalmente 1), el campo de una tabla, en cada `INSERT` que puede ser individual o masivo. Esto fue utilizado para todas las Primary Key que identifican con un valor numérico un registro de la tabla. Se ha utilizado para los números de factura, de compra, todos los `ID` que se han decidido en la etapa anterior, todos los códigos que identifican la parte de un auto u otras tablas, excepto en aquellas tablas "detail". Por ejemplo, `nro_compra` es `IDENTITY` en la tabla `compra`, pero no en `compra_item`. Dado que la tabla maestra contiene muchos de estos códigos, para la fase de migración se hace uso de dos instrucciones:

```
SET IDENTITY_INSERT <nombre_tabla> ON

-- Inserts de migración --

SET IDENTITY_INSERT <nombre_tabla> OFF
```

Esto permite la inserción de registros con el campo clave que viene de la tabla maestra. Una vez desactivado, continua con el siguiente valor de la secuencia.

Foreign Keys

Todas las Foreign Keys detalladas en la fase de normalización de la base de datos, son declaradas posteriormente a la creación de las tablas. Esto es debido a que es posible que eventualmente se quiera hacer referencia a una tabla que aún no fue creada. Por consiguiente se ejecutan alteraciones de tabla previo a la migración de la siguiente forma:

```
ALTER TABLE <nombre_tabla>
ADD
    FOREIGN KEY (<columna_fk>) REFERENCES <nombre_tabla_referenciada>(<columna_pk>),
    FOREIGN KEY (<columna_fk>) REFERENCES <nombre_tabla_referenciada>(<columna_pk>),
```

Funciones Auxiliares

En la fase de normalización, hemos decidido que ciertas tablas tendrán su Primary Key como un ID - IDENTITY, el cual no proviene de la tabla maestra. Por lo tanto, para referenciar clientes, fabricantes, o sucursales, se han definido funciones de la siguiente forma

```
CREATE FUNCTION <nombre_funcion>(<lista_de_params>) RETURNS DECIMAL(18,0) AS
BEGIN

    DECLARE @id_<entidad> DECIMAL(18,0);

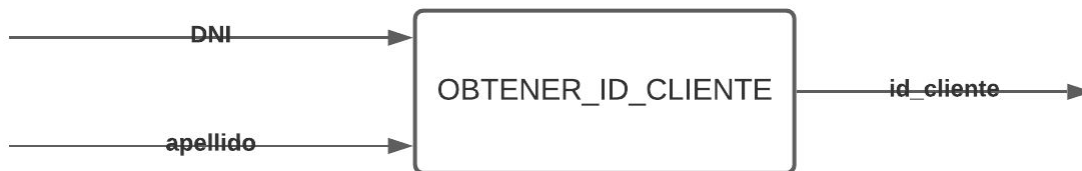
    SELECT @id_<entidad> = id_id_<entidad>
    FROM <nombre_tabla>
    WHERE <lista_de_condiciones>;

    RETURN @id_<entidad>;

END
GO
```

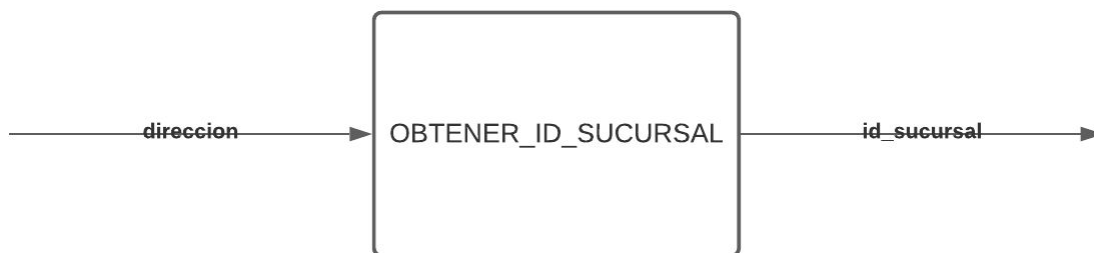
Siendo <entidad> cliente, fabricante, sucursal, <lista_de_params> la lista de parámetros que recibe la función (valores que identifican a la entidad en la tabla maestra), <lista_de_condiciones> las condiciones que especifican que la entidad buscada es la que se solicita mediante parámetros.

Función OBTENER_ID_CLIENTE:



Recibe el DNI del cliente y su apellido, debido a que hay DNI repetidos, por lo tanto se busca en la tabla cliente, el `id_cliente` según dni y apellido del cliente.

Función OBTENER_ID_SUCURSAL:



Función OBTENER_ID_FABRICANTE:



Migración

En este apartado se detalla el procedimiento de la migración de las tablas. Tal como indica el enunciado, cada tabla se migra dentro de una Stored Procedure.

Cada Stored Procedure es de la forma

```
CREATE PROCEDURE migrar_<nombre_tabla> AS
BEGIN

    SET IDENTITY_INSERT <nombre_tabla> ON

    -- Migración

    SET IDENTITY_INSERT <nombre_tabla> OFF

END
```

Solamente se ejecutan las instrucciones de IDENTITY_INSERT en aquellas tablas cuya Primary Key es un IDENTITY. A continuación se detalla el proceso de migración de cada una de las tablas que se realizan dentro de cada procedure.

Migración Modelo

```
INSERT INTO modelo (modelo_codigo, nombre, potencia)
    SELECT DISTINCT modelo_codigo, modelo_nombre, modelo_potencia
    FROM gd_esquema.Maestra
```

Se obtienen los registros sin repetidos de los campos `modelo_codigo`, `modelo_nombre` y `modelo_potencia` de la tabla maestra para ser insertados en la tabla `modelo`.

Migración Fabricante

```
INSERT INTO EMPANADA DE MONDONGO.fabricante (nombre)
    SELECT DISTINCT fabricante_nombre
    FROM gd_esquema.Maestra
    ORDER BY fabricante_nombre
```

Se obtienen los `fabricante_nombre` sin repetidos de la tabla maestra para ser insertados en la tabla `fabricante`.

Migración Cliente

Los clientes en la tabla maestra aparecen tanto desde el lado de compra, como desde el lado de las facturas, tal como se especifica en esta consulta del [google groups](#). Es decir, los datos que componen un cliente, aparecen en 2 ocasiones en la tabla maestra, con los mismos tipos de datos y la misma cantidad de columnas, únicamente con nombres distintos. Esto habilita a hacer una UNIÓN dentro de la misma tabla.

```
INSERT INTO cliente (dni, apellido, nombre, direccion, fecha nac, mail)
  SELECT DISTINCT cliente dni, cliente apellido, cliente nombre, cliente direccion,
    cliente_fecha_nac, cliente_mail FROM gd esquema.Maestra
    WHERE cliente_dni IS NOT NULL

UNION

SELECT DISTINCT fac cliente dni, fac cliente apellido,
  fac cliente nombre fac cliente direccion,
  fac_cliente_fecha_nac, fac_cliente mail FROM gd esquema.Maestra
    WHERE fac_cliente_dni IS NOT NULL
```

De esta manera, se seleccionan todos los clientes que han sido registrados en una factura, todos los clientes que han sido registrados en una compra, y se unen en un único INSERT, que internamente realiza un DISTINCT que permite que el mismo cliente no se repita. Se considera que el registro tiene un cliente, si hay un campo DNI no nulo.

Migración Sucursal

Siguiendo la misma lógica de la migración del cliente, los datos que componen una sucursal, aparecen en 2 ocasiones en la tabla maestra, con los mismos tipos de datos y la misma cantidad de columnas, únicamente con nombres distintos. Esto habilita a hacer una UNIÓN dentro de la misma tabla.

```
INSERT INTO sucursal (direccion, mail, telefono, ciudad)
  SELECT DISTINCT sucursal direccion, sucursal mail, sucursal_telefono,
    sucursal ciudad FROM gd esquema.Maestra
    WHERE sucursal_direccion IS NOT NULL

UNION

SELECT DISTINCT fac sucursal direccion, fac sucursal mail,
  fac sucursal telefono, fac_sucursal_ciudad
  FROM gd esquema.Maestra
  WHERE fac_sucursal_direccion IS NOT NULL
```

De esta manera, se seleccionan todas las sucursales que han sido registradas en una factura, todas las sucursales que han sido registradas en una compra, y se unen en un único INSERT, que internamente realiza un DISTINCT que permite que la misma sucursal no se repita. Se considera que el registro tiene una sucursal, si posee un campo dirección no nulo.

Migración Tipo Auto, Tipo Caja, y Tipo Transmisión

Estas tres tablas poseen una estructura similar (Código y Descripción), con lo cual el proceso de migración es igual cambiando los nombres de las columnas que los componen.

```
INSERT INTO tipo <nombre> (tipo <nombre> codigo, descripcion)
  SELECT DISTINCT tipo <nombre>_codigo, tipo_<nombre>_desc
  FROM gd esquema.Maestra
  WHERE tipo_<nombre>_codigo IS NOT NULL ORDER BY tipo_<nombre>_codigo
```

Donde <nombre> es el placeholder donde va el nombre de la tabla a migrar, pudiendo ser auto, caja o transmisión.

Migración Auto

Siguiendo la misma lógica que las migraciones anteriores, de la tabla maestra se obtienen los campos que componen el registro de un Auto

```
INSERT INTO auto (patente auto, nro chasis, nro motor, fecha alta,
  cant kms, tipo auto codigo, tipo motor codigo,
  tipo transmision codigo, tipo_caja_codigo,
  modelo codigo, id fabricante)
  SELECT DISTINCT auto patente, auto nro chasis, auto nro motor,
    auto fecha alta, auto cant kms, tipo auto codigo,
    tipo motor codigo, tipo transmision_codigo,
    tipo caja codigo, modelo codigo,
    OBTENER ID FABRICANTE(fabricante_nombre)
  FROM gd esquema.Maestra
  WHERE auto_patente IS NOT NULL
```

Obteniendo de la tabla maestra todos los registros que posean una patente no nula e insertándose en la tabla Auto de manera única, sin repetidos.

Dado que la tabla maestra no posee un campo que identifique al fabricante de manera unívoca, se decide utilizar la función auxiliar creada anteriormente para de cada auto, obtener el ID del fabricante y poder hacer referencia a esa tabla.

Migración Autoparte

Siguiendo la misma lógica que la migración de un auto, de la tabla maestra se obtienen los campos que componen el registro de una autoparte.

```
INSERT INTO autoparte (codigo autoparte, descripcion, categoria, id_fabricante,  
                      modelo codigo)  
  SELECT DISTINCT auto parte codigo, auto parte descripcion,  
                  NULL, OBTENER_ID_FABRICANTE(fabricante_nombre),  
                  modelo codigo  
  FROM qd esquema.Maestra  
  WHERE auto_parte_codigo IS NOT NULL
```

Obteniendo de la tabla maestra todos los registros que posean un código de autoparte no nulo e insertándose en la tabla Autoparte de manera única, sin repetidos.

Dado que la tabla maestra no posee un campo que identifique al fabricante de manera unívoca, se decide utilizar la función auxiliar creada anteriormente para de cada auto, obtener el ID del fabricante y poder hacer referencia a esa tabla.

Dado que el enunciado no especifica qué información se guarda en el campo categoría, de momento contendrá un valor nulo por defecto.

Migración Compra

El proceso de migración de la tabla compra, pasó por distintas fases de desarrollo. En un principio, se pretendía seguir con la lógica de las migraciones anteriores. Pero eso implica hacer una doble llamada a funciones en la instrucción SELECT. El problema es que necesitamos que el registro de una compra se almacene una única vez, sin repetidos, y por tal motivo el SELECT va acompañado de un DISTINCT. Las dos llamadas a funciones adicionado a la funcionalidad del DISTINCT, hacían que los registros, siendo una cantidad considerable, se vean obligados a reordenar 3 veces según el plan estimado de ejecución que ofrece SQL Server, esto hace que la migración de esta tabla sea de aproximadamente 20 minutos. Por lo tanto se decide, para evitar que se ordene múltiples veces, insertar los registros de compra sin repetidos en una tabla temporal para luego llamar a las funciones sin necesidad de reordenar. Esto logró que el tiempo de ejecución se reduzca a 8 minutos. Posteriormente, habiéndonos dado cuenta que el mayor costo de esta migración consistía en la obtención de IDS de cliente y sucursal mediante las funciones, se decide crear un índice sobre ambas tablas. De esta manera la ejecución se reduce a unos pocos segundos y se realiza de la siguiente manera:

```
SELECT DISTINCT compra nro, compra_fecha, sucursal_direccion, cliente_dni,
               cliente_apellido
      INTO #temp_compras
      FROM qd esquema.Maestra
      WHERE compra_nro IS NOT NULL

INSERT INTO compra (nro compra, fecha, id sucursal, id cliente)
      SELECT compra nro, compra fecha, OBTENER ID SUCURSAL(sucursal_direccion),
      OBTENER ID CLIENTE(cliente_dni, cliente_apellido)
      FROM #temp_compras

DROP TABLE #temp_compras
```

Migración Compra Auto

Dado que en compra solo se almacenan los datos comunes a la compra de un auto, como a la compra de una autoparte, entonces existen tablas para especificar el tipo de compra con los datos específicos de cada tipo. En este caso la compra de un solo auto y se realiza de manera similar a migraciones anteriores, tomando los campos que componen al registro de la compra de un auto sin tener en cuenta aquellos campos que ya han sido especificados en la tabla compra, excepto la Primary Key que lo referencia, quedando:

```
INSERT INTO EMPANADA DE MONDONGO.compra auto (nro compra, patente_auto, precio)
      SELECT DISTINCT compra nro, auto_patente, compra_precio
      FROM qd esquema.Maestra
      WHERE compra_nro IS NOT NULL AND auto_patente IS NOT NULL;
```

Se considera la compra de un auto, si el número de compra no es nulo y además la patente del auto no es nula.

Migración Compra Ítem

El proceso de migración de la tabla compra ítem, pasó por distintas fases de desarrollo. En un principio, se pretendía obtener todas las compras de un ítem e insertarlas ordenadas por número de compra en un cursor, para luego mediante un contador que se resete cada vez que cambia el número de compra, ir contando el número de ítem que corresponde para ser insertado en la tabla. El problema que tenía esta implementación es que era poco eficiente y se pensó otra solución mediante el uso de un trigger y una función que se detallan a continuación:

Trigger para inserción de Compra de un Ítem.

El desarrollo de este trigger consiste en delegarle a él la selección del número de ítem que corresponde para cada registro de la tabla. Para ello lo que hace es ejecutar la función que se detalla en el próximo punto que retorna el número de ítem que corresponde dado un número de compra. A pesar de tener que realizar un SELECT por cada INSERT, la performance de esta implementación se incrementó. Pasando de una ejecución de más de 10 segundos a unos pocos segundos. Además permite contemplar futuros INSERTS que se pudieran hacer por fuera de la migración.

```
CREATE TRIGGER INSERTANDO_COMPRA_ITEM ON compra_item INSTEAD OF INSERT AS
BEGIN

    DECLARE @nro compra DECIMAL(18,0);
    DECLARE @codigo autoparte DECIMAL(18,0);
    DECLARE @cantidad DECIMAL(18,0);
    DECLARE @precio DECIMAL(18,0);

    DECLARE items CURSOR LOCAL FORWARD ONLY READ ONLY
    FOR SELECT nro_compra, codigo_autoparte, cantidad, precio FROM inserted;

    OPEN items

    FETCH items INTO @nro_compra, @codigo_autoparte, @cantidad, @precio

    WHILE @@FETCH_STATUS = 0
    BEGIN

        INSERT INTO EMPANADA DE MONDONGO.compra item
        (nro compra, nro item, codigo_autoparte, cantidad, precio)
        VALUES (@nro compra,
                EMPANADA DE MONDONGO.OBTENER_NUMERO_ITEM(@nro_compra, 'C'),
                @codigo autoparte,
                @cantidad,
                @precio)

        FETCH items INTO @nro_compra, @codigo_autoparte, @cantidad, @precio

    END

    CLOSE items
    DEALLOCATE items

END
GO
```


Función para obtener número ítem

Tal como se indica en el punto anterior, esta función permite obtener el número de ítem que corresponde a un número determinado de compra, aunque se extendió para el mismo uso que se dará en el proceso de migración del ítem de una factura.

Esta función trae el número máximo de ítem dado el número de compra o de factura y lo incrementa en 1. En caso de ser NULL, se setea en 0.

```
CREATE FUNCTION OBTENER_NUMERO_ITEM(@nro DECIMAL(18,0), @tipo CHAR) RETURNS  
DECIMAL(18,0) AS  
  
BEGIN  
    DECLARE @item_num DECIMAL(18,0)  
  
    IF @tipo = 'C'  
        SELECT @item_num = (COALESCE(MAX(nro item), 0) + 1)  
        FROM compra_item WHERE nro_compra = @nro;  
    ELSE IF @tipo = 'F'  
        SELECT @item_num = (COALESCE(MAX(nro item), 0) + 1)  
        FROM factura_item WHERE nro_factura = @nro;  
  
    RETURN @item_num  
  
END  
GO
```

Finalmente, la migración de compra ítem, teniendo en cuenta el uso del trigger queda de la siguiente forma.

```
INSERT INTO compra_item (nro compra, codigo autoparte, cantidad, precio)  
SELECT compra_nro, auto_parte_codigo, compra_cant, compra_precio  
FROM gd_esquema.Maestra  
WHERE compra_nro IS NOT NULL AND auto_parte_codigo IS NOT NULL;
```

Migración Factura

El proceso de migración de factura concibe la misma lógica que la migración de la tabla compra, se insertan los datos sin repetidos en una tabla temporal, para posteriormente obtener el id de cliente y el id de sucursal e insertarlos en la tabla correspondiente.

```
SELECT DISTINCT factura_nro, factura_fecha, fac_sucursal_direccion, fac_cliente_dni,
               fac_cliente_apellido
    INTO #temp_factura
    FROM qd_esquema.Maestra
    WHERE factura_nro IS NOT NULL

INSERT INTO factura (nro_factura, fecha, id_sucursal, id_cliente)
    SELECT factura_nro, factura_fecha,
           OBTENER ID SUCURSAL(fac_sucursal_direccion),
           OBTENER ID CLIENTE(fac_cliente_dni, fac_cliente_apellido)
    FROM #temp_factura

DROP TABLE #temp_factura
```

Migración Factura Auto

El proceso de migración de factura auto concibe la misma lógica que la migración de la tabla compra auto. Se traen los campos comunes a los registros de factura de un auto sin repetidos y se inserta junto con su correspondiente número de factura, que referencia a la tabla de facturas.

```
INSERT INTO factura_auto (nro_factura, patente_auto, precio_facturado)
    SELECT DISTINCT factura_nro, auto_patente, precio_facturado
    FROM qd_esquema.Maestra
    WHERE factura_nro IS NOT NULL AND auto_patente IS NOT NULL;
```

Migración Factura Item

El proceso de migración de Factura item pasó por las mismas fases de desarrollo de Compra item, puesto que la lógica es muy similar, así que se define de la misma manera un Trigger y se utiliza la función definida en el punto anterior.

Trigger para inserción de Factura de un Item.

Este trigger tiene la misma funcionalidad que el Trigger detallado para la compra de un ítem, con la única diferencia que esta vez busca por número de factura en la tabla de factura ítem, obteniendo de la misma manera un incremento notable en la performance.

```
CREATE TRIGGER INSERTANDO_FACTURA_ITEM ON factura_item INSTEAD OF INSERT AS
BEGIN

    DECLARE @nro factura DECIMAL(18,0);
    DECLARE @codigo autoparte DECIMAL(18,0);
    DECLARE @cantidad DECIMAL(18,0);
    DECLARE @precio DECIMAL(18,0);

    DECLARE items CURSOR LOCAL FORWARD ONLY READ ONLY
    FOR SELECT nro_factura, codigo_autoparte, cantidad, precio FROM inserted;

    OPEN items

    FETCH items INTO @nro_compra, @codigo_autoparte, @cantidad, @precio

    WHILE @@FETCH_STATUS = 0
    BEGIN

        INSERT INTO factura_item
        (nro factura, nro item, codigo_autoparte, cantidad, precio)
        VALUES (@nro factura,
                OBTENER NUMERO ITEM(@nro_factura, 'F'),
                @codigo autoparte,
                @cantidad,
                @precio)

        FETCH items INTO @nro_compra, @codigo_autoparte, @cantidad, @precio

    END
    CLOSE items
    DEALLOCATE items

END
GO
```

Finalmente, la migración de factura ítem, teniendo en cuenta el uso del trigger queda de la siguiente forma.

```
INSERT INTO factura_item(nro factura, codigo autoparte, cant, precio)
SELECT factura nro, auto_parte_codigo, cant_facturada, precio_facturado
FROM qd esquema.Maestra
WHERE factura_nro IS NOT NULL AND auto_parte_codigo IS NOT NULL
```

Índices

Tal y como se describe en el punto de migración de Compras y de Factura, la performance se ve altamente incrementada con la creación de dos índices sobre la tabla de Sucursal y Cliente, dado que son muchas las búsquedas que realizan las funciones sobre estas tablas, el tiempo de búsqueda se ve reducido gracias al uso de índices. El tiempo de ejecución se redujo varios minutos.

```
CREATE INDEX ix_sucursal ON sucursal (direccion);
```

```
CREATE INDEX ix_cliente ON cliente (dni, apellido);
```

El visor de plan estimado de ejecución de SQL Server propone la creación de índices sobre la tabla maestra para incrementar la velocidad de búsqueda. Se decide como grupo no insertar índices sobre esa tabla ya que no se quiere modificarla en absoluto y estos índices serían sobre distintas columnas, lo que ocuparía un espacio considerable y podría ser contraproducente.

De la misma manera se crean dos índices adicionales sobre las tablas factura_item y compra_item, con el objetivo de acelerar el cálculo del stock de las autopartes. De esta forma, el cálculo del stock total se reduce en 1 minuto aproximadamente.

```
CREATE INDEX ix_compra_item ON compra_item (codigo_autoparte);
```

```
CREATE INDEX ix_factura_item ON factura_item (codigo_autoparte);
```

Vistas

Dado que el enunciado pide crear Views, pero no se aclara para qué ni tampoco fue contestada la [consulta](#) en el grupo antes de la fecha de entrega, entonces se deciden crear Views según casos de uso, seleccionando aquellas queries que podrían ser más complejas, entregando únicamente datos que podrían ser útiles para el usuario, omitiendo códigos, ids y otros datos que no poseen tanto valor informativo. De esta manera se crearon las siguientes views:

Compras de Autos

```
CREATE VIEW view_compra_auto AS
SELECT c.nro compra, ca.patente auto, m.nombre as modelo, ca.precio, c.fecha,
       s.direccion as sucursal, cl.apellido + ', ' + cl.nombre as cliente,
       cl.dni, cl.mail
FROM compra c
JOIN compra auto ca ON c.nro compra = ca.nro compra
JOIN cliente cl ON cl.id cliente = c.id cliente
JOIN auto a ON a.patente auto = ca.patente auto
JOIN modelo m ON m.modelo codigo = a.modelo codigo
JOIN sucursal s ON s.id_sucursal = c.id_sucursal
```

Esta View informa todos los datos correspondientes a la compra de un auto:

- Número de compra
- Patente auto
- Modelo
- Precio
- Fecha
- Sucursal
- Nombre y Apellido del cliente
- Dni
- Mail

Compra de Items

```
CREATE VIEW view compra items AS
SELECT c.nro compra, ci.nro item, ap.descripcion as autoparte, m.nombre as
      modelo, ci.cantidad, ci.precio as total, c.fecha, s.direccion as
      sucursal, cl.apellido + ', ' + cl.nombre as cliente, cl.dni, cl.mail
FROM EMPANADA DE MONDONGO.compra c
JOIN compra item ci ON c.nro compra = ci.nro compra
JOIN cliente cl ON cl.id cliente = c.id cliente
JOIN sucursal s ON s.id sucursal = c.id sucursal
JOIN autoparte ap ON ap.codigo autoparte = ci.codigo_autoparte
JOIN modelo m ON m.modelo_codigo = ap.modelo_codigo
```

Esta view informa todos los datos de las autopartes correspondientes a una compra:

- Número de compra
- Número de Ítem
- Autoparte
- Cantidad
- Precio total
- Fecha
- Sucursal
- Nombre y Apellido del cliente
- DNI
- Mail

Facturación de Autos

```
CREATE VIEW view factura auto AS
SELECT f.nro factura, fa.patente auto, m.nombre as modelo,
      fa.precio, f.fecha, s.direccion as sucursal, cl.apellido + ', ' +
      cl.nombre as cliente, cl.dni, cl.mail
FROM factura f
JOIN factura_auto fa ON f.nro_factura = fa.nro_factura
JOIN cliente cl ON cl.id cliente = f.id cliente
JOIN auto a ON a.patente auto = fa.patente auto
JOIN modelo m ON m.modelo_codigo = a.modelo_codigo
JOIN sucursal s ON s.id_sucursal = f.id_sucursal
```

Esta View informa todos los datos correspondientes a la venta de un auto:

- Número de factura
- Patente auto
- Modelo
- Precio
- Fecha
- Sucursal
- Nombre y Apellido del cliente
- Dni
- Mail

Factura Items

```
CREATE VIEW view factura items AS
SELECT f.nro factura, fi.nro item, ap.descripcion as autoparte,
       m.nombre as modelo, fi.cantidad, fi.precio as total,
       f.fecha, s.direccion as sucursal,
       cl.apellido + ', ' + cl.nombre as cliente, cl.dni, cl.mail
FROM factura f
JOIN factura item fi ON f.nro factura = fi.nro factura
JOIN cliente cl ON cl.id cliente = f.id cliente
JOIN sucursal s ON s.id_sucursal = f.id_sucursal
JOIN autoparte ap ON
    ap.codigo autoparte = fi.codigo autoparte
JOIN modelo m ON m.modelo_codigo = ap.modelo_codigo
```

Esta view informa todos los datos de las autopartes correspondientes a una compra:

- Número de Factura
- Número de Ítem
- Autoparte
- Cantidad
- Precio total
- Fecha
- Sucursal
- Nombre y Apellido del cliente
- DNI
- Mail

Auto

```
CREATE VIEW view auto AS
SELECT f.nombre as fabricante, m.nombre as modelo,
       a.patente auto, a.cant kms, ta.descripcion as tipo_auto,
       a.fecha alta, a.nro chasis, a.nro motor,
       tipo motor codigo as tipo motor, m.potencia,
       tt.descripcion as tipo_transmision, tc.descripcion as tipo_caja
FROM auto a
JOIN tipo auto ta ON ta.tipo auto codigo = a.tipo auto codigo
JOIN tipo transmision tt ON tt.tipo_transmision_codigo =
    a.tipo transmision codigo
JOIN tipo caja tc ON tc.tipo caja codigo = a.tipo caja_codigo
JOIN modelo m ON m.modelo codigo = a.modelo codigo
JOIN fabricante f ON f.id_fabricante = a.id_fabricante
```

Esta view informa todos los datos de un auto específico:

- Fabricante
- Modelo
- Patente
- Cantidad de Kilómetros
- Tipo de auto
- Fecha de alta
- Número de Chasis
- Número de Motor
- Tipo de motor
- Potencia
- Transmisión
- Caja

Autoparte

```
CREATE VIEW autoparte AS
SELECT ap.codigo autoparte, ap.descripcion, ap.categoria,
       f.nombre as fabricante, m.nombre as modelo
FROM autoparte ap
JOIN fabricante f ON f.id fabricante = ap.id fabricante
JOIN modelo m ON m.modelo_codigo = ap.modelo_codigo
```

Esta view informa todos los datos de un autoparte específico:

- Código de autoparte
- Descripción
- Categoría
- Fabricante
- Modelo

Stock

Dado que es muy probable que el stock deba ser calculado con frecuencia, se crearon views que sean capaces de informar el stock de un auto o autoparte.

Autos disponibles:

```
CREATE VIEW view autos en stock AS
SELECT * FROM view auto a
WHERE a.patente_auto NOT IN (SELECT fa.patente_auto factura_auto fa)
```

Autopartes en stock:

```
CREATE VIEW view autopartes en stock AS
SELECT ap.*, STOCK AUTOPARTE(ap.codigo_autoparte) as stock
FROM view_autoparte ap
```

Esta última view necesita saber cuánto se compró de cada autoparte, para luego saber cuánto vendió de cada una, para ello utiliza la siguiente función:

```
CREATE FUNCTION STOCK_AUTOPARTE(@codigo_autoparte DECIMAL(18,0))
RETURNS DECIMAL(18,0) AS

BEGIN

    DECLARE @cantidad comprada DECIMAL(18,0);
    DECLARE @cantidad_vendida DECIMAL(18,0);

    SELECT @cantidad comprada = SUM(ci.cantidad)
    FROM compra_item ci WHERE ci.codigo_autoparte = @codigo_autoparte

    SELECT @cantidad vendida = SUM(fi.cantidad)
    FROM factura_item fi WHERE fi.codigo_autoparte = @codigo_autoparte

    RETURN @cantidad_comprada - @cantidad_vendida

END
```

Para que esta función sea efectiva y performante necesita de la existencia de los índices que se han creado anteriormente.

Estas vistas no están pensadas para realizar inserts, de hecho no serán posibles a menos que se utilicen triggers sobre ellas. Para realizar inserts sobre las tablas, se prefiere hacerlos directo sobre ellas, dado que tiene menos margen de error el hecho de obtener de un formulario por ejemplo el ID de una sucursal, en lugar de su dirección por cada registro.

Business Intelligence

En este apartado se detalla el procedimiento utilizado para el desarrollo del modelo de Business Intelligence. Dado que son varias tablas grandes, se decide dividir en varios archivos PNG que se encontrarán junto con este documento o bien:

- [Modelo BI Autopartes](#)
- [Modelo BI Autos](#)

Borrado Previo

Igual que en la sección anterior, antes de realizar la migración se realiza un borrado de objetos de base de datos que hayan sido creados por este script. La modalidad es la misma y tiene como objetivo evitar conflictos al ejecutar el script en reiteradas ocasiones. El script no funcionará si anteriormente no se ejecuta script_creación_inicial.sql.

Modelo Estrella

Tal y como se vio en clase, se decide utilizar el modelo estrella para confeccionar el modelo de Business Intelligence.

Fact Tables

Para confeccionar el modelo estrella se han definido las siguientes tablas de hechos:

- bi_compra_autos: Detalla las compras de automóviles.
- bi_venta_autos: Detalla las ventas de automóviles.
- bi_compra_autopartes: Detalla las compras de autopartes.
- bi_venta_autopartes: Detalla las ventas de autopartes.
- bi_stock_autopartes Detalla el stock de autopartes.

Para confeccionar dichas tablas se han tenido en cuenta las siguientes dimensiones:

- bi_compra_autos:
 - tiempo
 - sucursal
 - modelo
 - cliente
 - potencia
 - fabricante
 - tipo auto
 - tipo transmisión
 - tipo caja
 - tipo motor

- bi_venta_autos:
 - tiempo
 - sucursal
 - modelo
 - cliente
 - potencia
 - fabricante
 - tipo auto
 - tipo transmisión
 - tipo caja
 - tipo motor

- bi_compra_autopartes:
 - tiempo
 - sucursal
 - modelo
 - cliente
 - potencia
 - fabricante
 - categoría
 - autoparte

- bi_venta_autopartes:
 - tiempo
 - sucursal
 - modelo
 - cliente
 - potencia
 - fabricante
 - categoría
 - autoparte

- bi_stock_autopartes:
 - tiempo
 - sucursal
 - modelo
 - potencia
 - fabricante
 - categoría
 - autoparte

Se decide separar en distintas tablas de hechos (distintos Data Marts) debido a que, si fuese una sola tabla de hechos, habría que lidiar con claves nulas, así como también atributos nulos, lo cual complica las operaciones sobre las tablas que son necesarias para confeccionar el modelo.

De la misma manera, si bien tiene sentido agregar datos como la edad de los clientes y otras dimensiones en la parte del modelo que se encarga de las compras y ventas, no se ve necesario para, por ejemplo, el cálculo del stock. Podría ser necesario en caso de implementar técnicas de Data Mining, pero se escapa completamente del objetivo del TP y complejiza altamente la confección del modelo.

Migración hacia el modelo de Business Intelligence

Para poder confeccionar el modelo se decide migrar las tablas necesarias que pasarán a ser las dimensiones del modelo.

Tablas migradas:

- Fabricante
- Modelo
- Sucursal
- Tipo Auto
- Tipo Caja
- Tipo Transmisión

Tablas confeccionadas para el modelo de Business Intelligence

Para facilitar la confección del modelo, se han creado las siguientes tablas:

- tiempo
 - id_tiempo
 - mes
 - año
- potencia
 - id_potencia
 - rango
- edad
 - id_edad
 - rango

Proceso de Migración hacia el modelo de Business Intelligence

Para migrar los datos del modelo transaccional hacia el modelo de Business Intelligence, se han utilizado, de la misma manera que la migración de la entrega anterior, Stored Procedures que serán detalladas en el próximo apartado.

A continuación detallaremos aquellas decisiones tomadas acerca de las migraciones.

a) Cliente

- i) No hay forma de calcular el sexo de cada cliente, con lo cual, se agrega la opción 'N' (no especifica) como opción para el sexo.

b) Categoría

- i) No hay información acerca de la categoría en la tabla, con lo cual se completa con 'Desconocido'

c) Cantidad de Cambios

- i) Tal y como se indica en el [post de Google Groups](#), la dimensión cantidad de cambios no debe ser tenida en cuenta

d) Tiempo de stock promedio de autopartes

- i) Tal y como se indica en el [post de Google Groups](#), no debe calcularse el tiempo de stock promedio de autopartes pues no existe trazabilidad alguna.

Dimensión Cliente

De la dimensión cliente el interés está puesto sobre el rango de edad y el sexo, con lo cual se crea la tabla bi_edad:

```
CREATE TABLE EMPANADA DE MONDONGO.bi edad(  
    id edad DECIMAL(18) PRIMARY KEY IDENTITY(1,1),  
    rango NVARCHAR(50) NOT NULL,  
);  
GO
```

Que se carga con la ayuda de la función RANGO_EDAD

```
CREATE FUNCTION EMPANADA_DE_MONDONGO.RANGO_EDAD(@FECHA_NACIMIENTO DATE) RETURNS  
DECIMAL(18) AS BEGIN  
  
    DECLARE @id edad DECIMAL(18);  
    DECLARE @HOY DATE;  
    DECLARE @EDAD INT;  
    SET @HOY = GETDATE();  
    SET @EDAD = (DATEDIFF(DAY, @FECHA_NACIMIENTO, @HOY) / 365)  
  
    IF @EDAD BETWEEN 18 AND 30  
        SELECT @id edad = id edad  
        FROM EMPANADA DE MONDONGO.bi_edad WHERE rango = '18 - 30 años'  
    ELSE IF @EDAD BETWEEN 31 AND 50  
        SELECT @id edad = id edad  
        FROM EMPANADA_DE_MONDONGO.bi_edad WHERE rango = '31 - 50 años'  
    ELSE  
        SELECT @id edad = id edad  
        FROM EMPANADA_DE_MONDONGO.bi_edad WHERE rango = '> 50 años'  
    RETURN @id_edad;  
  
END  
GO
```

En cuanto al SEXO del cliente, éste es desconocido, con lo cual se cargará una Primary Key con una 'N' por defecto (No especifica), pudiendo ser 'M', 'F', 'O' el resto de las opciones posibles para Masculino, Femenino, Otros.

Dimensión Tiempo

De la dimensión tiempo el interés está puesto sobre el mes y el año, con lo cual se crea la tabla bi_tiempo:

```
CREATE TABLE EMPANADA_DE_MONDONGO.bi_tiempo(  
    id_tiempo DECIMAL(18) PRIMARY KEY IDENTITY(1,1),  
    agno DECIMAL(4) NOT NULL,  
    mes DECIMAL(2) NOT NULL  
);  
GO
```

Que se carga con la ayuda del SP bi_cargar_tiempo, que carga todos los posibles pares de año/mes en la tabla, con el objetivo de incrementar la performance de la migración.

```
CREATE PROCEDURE EMPANADA_DE_MONDONGO.bi_cargar_tiempo AS  
BEGIN  
  
    INSERT INTO EMPANADA_DE_MONDONGO.bi_tiempo (agno, mes)  
    SELECT DISTINCT YEAR(fecha), MONTH(fecha)  
    FROM EMPANADA_DE_MONDONGO.compra  
    UNION  
    SELECT DISTINCT YEAR(fecha), MONTH(fecha)  
    FROM EMPANADA_DE_MONDONGO.factura  
  
END  
GO
```

Y con la función ID_TIEMPO:

```
CREATE FUNCTION EMPANADA_DE_MONDONGO.ID_TIEMPO(@fecha DATE) RETURNS DECIMAL(18) AS  
BEGIN  
  
    DECLARE @agno DECIMAL(4), @mes DECIMAL(2), @id_tiempo DECIMAL(18);  
  
    SET @agno = DATEPART(YEAR, @fecha)  
    SET @mes = DATEPART(MONTH, @fecha)  
  
    SELECT @id_tiempo = id_tiempo  
    FROM EMPANADA_DE_MONDONGO.bi_tiempo WHERE agno = @agno AND mes = @mes  
  
    RETURN @id_tiempo  
  
END  
GO
```

Dimensión Potencia

De la dimensión potencia el interés está puesto sobre el rango de potencia, con lo cual se crea la tabla bi_potencia:

```
CREATE TABLE EMPANADA_DE_MONDONGO.bi_potencia(  
    id potencia DECIMAL(18) PRIMARY KEY IDENTITY(1,1),  
    rango NVARCHAR(50) NOT NULL,  
);  
GO
```

Que se carga con la ayuda del SP bi_cargar_potencia, que carga todos los posibles rangos de potencia en la tabla, con el objetivo de incrementar la performance de la migración.

```
CREATE PROCEDURE EMPANADA_DE_MONDONGO.bi_cargar_potencia AS  
BEGIN  
  
    INSERT INTO EMPANADA_DE_MONDONGO.bi_potencia (rango)  
    VALUES ('50-150cv'), ('151-300cv'), ('> 300cv');  
  
END  
GO
```

Y la función RANGO_POTENCIA:

```
CREATE FUNCTION EMPANADA_DE_MONDONGO.RANGO_POTENCIA(@POTENCIA DECIMAL(18)) RETURNS  
DECIMAL(18) AS  
BEGIN  
  
    DECLARE @id_potencia DECIMAL(18);  
  
    IF @POTENCIA BETWEEN 50 AND 150  
        SELECT @id_potencia = id_potencia  
        FROM EMPANADA_DE_MONDONGO.bi_potencia WHERE rango = '50-150cv'  
  
    ELSE IF @POTENCIA BETWEEN 151 AND 300  
        SELECT @id_potencia = id_potencia  
        FROM EMPANADA_DE_MONDONGO.bi_potencia WHERE rango = '151-300cv'  
  
    ELSE  
        SELECT @id_potencia = id_potencia  
        FROM EMPANADA_DE_MONDONGO.bi_potencia WHERE rango = '> 300cv'  
  
    RETURN @id_potencia;  
  
END  
GO
```


Otras dimensiones

El resto de las dimensiones se migran de la misma forma que se han migrado en la sección anterior, con sus campos correspondientes exceptuando:

- a) Modelo:
 - i) Se remueve 'modelo_potencia' de entre sus atributos, pasa a ser una nueva dimensión
- b) Autoparte:
 - i) Se remueve 'categoría' de entre sus atributos, pasa a ser una nueva dimensión
 - ii) Se remueve 'id_fabricante' de entre sus atributos, pasa a ser una nueva dimensión
 - iii) Se remueve 'modelo_codigo' de entre sus atributos, pasa a ser una nueva dimensión

Tablas de Hechos

Se han creado las siguientes tablas de hechos tal y como se mencionó anteriormente:

```
CREATE TABLE EMPANADA_DE_MONDONGO.bi_compra_autos(  
  
    id tiempo DECIMAL(18),  
    id sucursal DECIMAL(18),  
    modelo codigo DECIMAL(18),  
    id edad DECIMAL(18),  
    sexo CHAR(1) CHECK (sexo IN ('F', 'M', 'O', 'N')),  
    id potencia DECIMAL(18),  
    id fabricante DECIMAL(18),  
    tipo auto codigo DECIMAL(18),  
    tipo transmision codigo DECIMAL(18),  
    tipo caja codigo DECIMAL(18),  
    tipo motor DECIMAL(18),  
    cantidad cambios DECIMAL(18),  
    cantidad comprada DECIMAL(18),  
    total comprado DECIMAL(18,2),  
    PRIMARY KEY (id tiempo, id sucursal, modelo codigo,  
    id edad, sexo, id potencia, id fabricante, tipo auto codigo,  
    tipo_transmision_codigo, tipo_caja_codigo, tipo_motor, cantidad_cambios)  
);  
GO
```

```
CREATE TABLE EMPANADA_DE_MONDONGO.bi_venta_autos(  
  
    id tiempo DECIMAL(18),  
    id sucursal DECIMAL(18),  
    modelo codigo DECIMAL(18),  
    id edad DECIMAL(18),  
    sexo CHAR(1) CHECK (sexo IN ('F', 'M', 'O', 'N')),  
    id potencia DECIMAL(18),  
    id fabricante DECIMAL(18),  
    tipo auto codigo DECIMAL(18),  
    tipo transmision codigo DECIMAL(18),  
    tipo caja codigo DECIMAL(18),  
    tipo motor DECIMAL(18),  
    cantidad cambios DECIMAL(18),  
    cantidad vendida DECIMAL(18),  
    total vendido DECIMAL(18,2),  
    PRIMARY KEY (id tiempo, id sucursal, modelo codigo,  
    id edad, sexo, id potencia, id fabricante, tipo auto codigo,  
    tipo_transmision_codigo, tipo_caja_codigo, tipo_motor, cantidad_cambios)  
);  
GO
```

```

CREATE TABLE EMPANADA_DE_MONDONGO.bi_compra_autopartes(

    id tiempo DECIMAL(18),
    id sucursal DECIMAL(18),
    modelo codigo DECIMAL(18),
    id edad DECIMAL(18),
    sexo CHAR(1) CHECK (sexo IN ('F', 'M', 'O', 'N')),
    id potencia DECIMAL(18),
    id fabricante DECIMAL(18),
    codigo autoparte DECIMAL(18),
    categoria VARCHAR(50),
    cantidad comprada DECIMAL(18),
    total comprado DECIMAL(18,2),
    PRIMARY KEY (id tiempo, id sucursal, modelo codigo,
    id_edad, sexo, id_potencia, id_fabricante, categoria, codigo_autoparte)
);
GO

CREATE TABLE EMPANADA_DE_MONDONGO.bi_venta_autopartes(

    id tiempo DECIMAL(18),
    id sucursal DECIMAL(18),
    modelo codigo DECIMAL(18),
    id edad DECIMAL(18),
    sexo CHAR(1) CHECK (sexo IN ('F', 'M', 'O', 'N')),
    id potencia DECIMAL(18),
    id fabricante DECIMAL(18),
    codigo autoparte DECIMAL(18),
    categoria VARCHAR(50),
    cantidad vendida DECIMAL(18),
    total vendido DECIMAL(18,2),
    PRIMARY KEY (id tiempo, id sucursal, modelo codigo,
    id_edad, sexo, id_potencia, id_fabricante, categoria, codigo_autoparte)
);
GO

CREATE TABLE EMPANADA_DE_MONDONGO.bi_stock_autopartes(

    id tiempo DECIMAL(18),
    id sucursal DECIMAL(18),
    modelo codigo DECIMAL(18),
    id potencia DECIMAL(18),
    id fabricante DECIMAL(18),
    codigo autoparte DECIMAL(18),
    categoria VARCHAR(50),
    cantidad vendida DECIMAL(18),
    total vendido DECIMAL(18,2),
    PRIMARY KEY (id tiempo, id sucursal, modelo codigo,
    id_edad, sexo, id_potencia, id_fabricante, categoria, codigo_autoparte)
);
GO

```

Adicionalmente se han creado todas las Foreign Keys necesarias para las tablas de hechos hacia las dimensiones. Utilizando la misma modalidad que en la sección anterior.

Carga de Datos

Luego de migrar todas las tablas que se mencionaron anteriormente, prosigue la carga de las tablas de hechos con Stored Procedures que son demasiado extensas para plasmarlas en el documento.

Carga de Compras de Autos

- Se cargan todas las compras de autos con sus respectivas dimensiones en la tabla de hechos, obteniendo los datos de las tablas:
 - compra_auto
 - compra
 - cliente
 - auto
 - modelo
- Con los atributos:
 - cantidad_comprada
 - total_comprado

Carga de Ventas de Autos

- Se cargan todas las ventas de autos con sus respectivas dimensiones en la tabla de hechos, obteniendo los datos de las tablas:
 - factura
 - factura_auto
 - cliente
 - auto
 - modelo
- Con los atributos:
 - cantidad_vendida
 - total_comprado

Carga de Compras de Autopartes

- Se cargan todas las compras de autopartes con sus respectivas dimensiones en la tabla de hechos, obteniendo los datos de las tablas:
 - compra
 - compra_item
 - cliente
 - autoparte
 - modelo
- Con los atributos:
 - cantidad_comprada
 - total_comprado

Carga de Ventas de Autopartes

- Se cargan todas las compras de autopartes con sus respectivas dimensiones en la tabla de hechos, obteniendo los datos de las tablas:
 - factura
 - factura_item
 - cliente
 - autoparte
 - modelo
- Con los atributos:
 - cantidad_vendida
 - total_comprado

Carga de Stock de Autopartes

- Se carga el stock de cada autoparte según las dimensiones correspondientes, obteniendo los datos de las tablas:
 - bi_venta_autopartes
 - bi_compra_autopartes

Creación de Vistas

Por último se han creado las vistas necesarias para recabar datos del modelo de inteligencia de negocios basada en las tablas de hechos creadas anteriormente. Su definición es suficientemente extensa como para no incluirlas en este documento.

Se han creado las vistas:

- v_ganancias_x_sucursal_mes_autos: Muestra las ganancias mensuales por ventas de autos por sucursal
- v_promedios_compra_venta_autos: Muestra el precio promedio tanto de compras como de ventas de cada modelo de auto.
- v_promedios_compra_venta_autopartes: Muestra el precio promedio tanto de compras como de ventas de cada autoparte
- v_ganancias_x_sucursal_mes_autopartes: Muestra las ganancias mensuales por ventas de autopartes por sucursal
- v_maxima_cant_stock_x_sucursal: Muestra la máxima cantidad de stock de autopartes que hubo en cada sucursal. Stock negativo implica que vendió autopartes que no han sido compradas por esa sucursal.