MobyDigital S.A.

Academy

Backend

Matias Arias

Año 2023

Índice

Tercer Evaluación Técnica

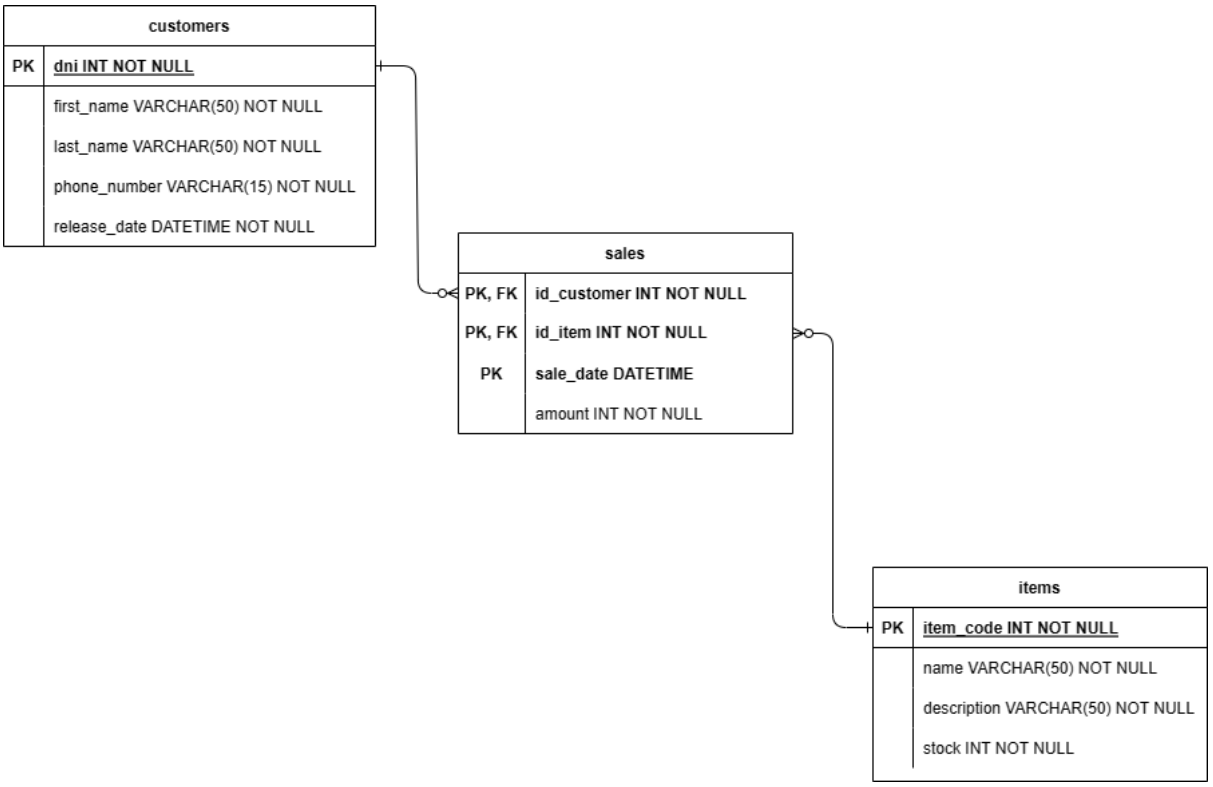
La presente evaluación técnica tiene como objetivo evaluar tus conocimientos sobre el lenguaje SQL. Se tendrán en cuenta todas las queries/consultas entregadas (DML y DDL) así como, el tiempo de resolución y aclaraciones (en caso de ser necesario).

Un comerciante desea modelar una base de datos para llevar el control de sus ventas.

“Cada día se venden artículos a un cliente. Los datos que son de interés de un cliente son: nombre, apellido, dni (es clave primaria), teléfono y fecha de alta. En el caso de los artículos nos interesa tener un código (clave primaria), nombre, descripción y unidades disponibles. Por último, en cada venta se debe indicar el cliente, el producto y la cantidad vendida y la fecha de la venta. El comerciante considera que todos los datos siempre deben ser cargados”

Si queres ver mi resolución seguí los pasos de: Explicación directorio

Diagrama de Entidad-Relación



Normalización: Primera forma Normal

La normalización es la utilización de ciertas reglas para evitar redundancias e inconsistencias en nuestro diseño de base de datos.
Para verificar si nuestro diseño cumple con la 1FN debemos revisar si cumple con:

- Cada tabla tiene una clave primaria que identifique de forma única le registro y no contiene atributos nulos
- Cada tabla mantiene integridad en sus atributos y todos tienen su debida relación con la tabla
- Todos los atributos son atómicos

A simple vista, para los requerimientos que posee el dominio se puede comprobar que se cumple con la 1FN:

- Customers:
    - La columna dni es la clave primaria y cumple con el requisito de unicidad
    - Cada columna contiene datos atómicos y no redundantes
- Items:
    - La columna item_code es la clave primaria y cumple con el requisito de unicidad
    - Cada columna contiene datos atómicos y no redundantes
- Sales
    - Las columnas “id_customer”,”id_item” y “sale_datetime” forman la clave primaria y cumple con el requisito de unicidad
    - Cada columna contiene datos atómicos y no redundantes.

Aclaración y posibles riesgos:

- El hecho de que la PK de *sales* sea una llave compuesta que incluye el atributo sale_date puede presentar un riesgo o error si se llegan a registrar dos ventas del mismo cliente en el mismo instante. Por el contexto del enunciado y el fin de la evaluación opté por dejarlo de esa manera pero en términos de escalabilidad no estoy seguro si es lo más recomendable.

Inserción de datos DML

Los scripts SQL para insertar los datos se encuentran en el archivo data.sql
Aquí se muestra un ejemplo de cada uno:

```
INSERT INTO items(item_code,name,description,stock)
 VALUES (1,'Yerba','Paquete de 1 kg de yerba',90);

INSERT INTO customers(dni,first_name,last_name,phone_number,registration_date)
 VALUES (33445566,'Juan','Piquete','011-1567899879','2015-05-15');

INSERT INTO sales(id_customer,id_item,sale_datetime,amount)
 VALUES(33445566,1,'2019-01-01 11:30',5);
```

Consultas específicas

Mostrar Nombre y Apellido de los clientes y Nombre del artículo vendido, pero ordenando por la fecha de ventas en forma descendente

```
SELECT
 CONCAT(c.first_name," ",c.last_name) "Cliente",
 i.name "Item",
 s.sale_datetime "Fecha de venta"
FROM sales s
 INNER JOIN customers c ON (s.id_customer = c.dni)
 INNER JOIN items i ON (s.id_item = i.item_code)
ORDER BY s.sale_datetime DESC ;
```

Resultado:

Cliente	Item	Fecha de venta
Marixa Bella	Pan	2019-09-14 19:00:00
Laura Agnay	Queso cremoso	2019-08-14 18:00:00
Andrea Lira	Leche larga vida	2019-07-14 18:30:00
Laura Agnay	Pan	2019-07-13 09:00:00
Esteban Quito	Bolsa de caramelos	2019-06-14 18:05:00
Carlos Calabresa	Yerba	2019-06-14 18:00:00
Esteban Quito	Yerba	2019-05-01 10:30:00
Andrea Lira	Azucar	2019-04-13 09:30:00
Juan Piquete	Leche larga vida	2019-02-14 11:00:00
Carlos Calabresa	Azucar	2019-02-01 12:30:00
Marixa Bella	Pan	2019-01-14 10:00:00
Juan Piquete	Yerba	2019-01-01 11:30:00

Listar los Nombres de los artículos y la suma de las Unidades vendidas. Ordenar alfabéticamente en forma ascendente por el nombre del artículo.

```
SELECT
 i.name "Item",
 SUM(s.amount) "Unidades vendidas"
FROM sales s
 INNER JOIN items i ON (s.id_item = i.item_code)
GROUP BY i.name
ORDER BY i.name ASC;
```

Resultado:

	Item	Unidades vendidas
▶	Azucar	7
	Bolsa de caramelos	6
	Leche larga vida	4
	Pan	5
	Queso cremoso	5
	Yerba	16

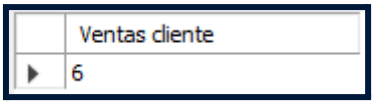
Funciones y procedimientos PL/SQL

Creación de una función

El comerciante ahora solicita determinar la cantidad de ventas realizadas a un cliente dado. Para poder realizar este requerimiento y mantenerlo en el tiempo, te solicitan que crees una función de nombre ventas_cliente que determine la cantidad de ventas realizadas a un cliente en particular. El cliente será un parámetro de la función.

```
DELIMITER $$
CREATE FUNCTION `ventas_cliente` (
dni VARCHAR(9)
)
RETURNS INTEGER
READS SQL DATA
BEGIN
DECLARE customer_sales INTEGER;
SELECT
SUM(s.amount)
FROM sales s
WHERE s.id_customer = dni
INTO customer_sales;
RETURN customer_sales;
END $$
DELIMITER ;
```

Resultado:



	Ventas cliente
▶	6

Creación de un procedimiento almacenado

guardar_datos_cliente

El primer procedimiento se utilizará para insertar/actualizar datos de Clientes. Deberá llamarse guardar_datos_cliente, recibe parámetro análogos a cada columna de la tabla Cliente y no posee un tipo de retorno. Deberá validar la existencia del DNI del cliente y en caso de existir, actualizar los datos, o bien, insertarlos.

```
DELIMITER $$
CREATE PROCEDURE `guardar_datos_cliente`(
dni_p VARCHAR(9),
first_name_p VARCHAR(50),
last_name_p VARCHAR(50),
phone_number_p VARCHAR(15),
registration_date_p DATE
)
BEGIN
  DECLARE listo BOOLEAN DEFAULT FALSE;
  DECLARE saved BOOLEAN DEFAULT FALSE;
  DECLARE dni_cursor integer;
  DECLARE c1 CURSOR FOR SELECT dni FROM customers;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET listo=TRUE;
  OPEN c1;
  c1_loop:LOOP
  FETCH c1 INTO dni_cursor;
   IF dni_cursor=dni_p then
      UPDATE customers SET
first_name=first_name_p,last_name=last_name_p,phone_number=phone_number_
p,registration_date=registration_date_p WHERE dni=dni_p;
      SET saved= TRUE;
    END IF;
  IF listo=TRUE THEN
      IF saved=FALSE THEN
            INSERT INTO
customers(dni,first_name,last_name,phone_number,registration_date)
VALUES
(dni_p,first_name_p,last_name_p,phone_number_p,registration_date_p);
    END IF;
    LEAVE c1_loop;
    END IF;
END LOOP c1_loop;
CLOSE c1;
END $$
DELIMITER ;
```

Resultado previo:

dni	first_name	last_name	phone_number	registration_date
22445566	Laura	Agnay	0342-156378022	2013-03-13
25335999	Marixa	Bella	0342-154348022	2012-02-12
33445566	Juan	Piquete	011-1567899879	2015-05-15
33445588	Andrea	Lira	0342-155398022	2011-01-11
33556677	Esteban	Quito	011-1561233219	2014-04-14
34347788	Carlos	Calabresa	011-1564566549	2016-06-16

Luego se ejecutan las siguientes queries:

```
call guardar_datos_cliente('25335999','Maria','Benitez','0342-154139913','2022-12-18');
call guardar_datos_cliente('43132313','Matias','Arias','0342-154266141','2018-12-09');

select * from customers;
```

```
679 11:50:24 call guardar_datos_cliente('25335999','Maria','Benitez','0342-154139913','2022-12-18') 0 row(s) affected
680 11:50:27 call guardar_datos_cliente('43132313','Matias','Arias','0342-154266141','2018-12-09') 1 row(s) affected
681 11:50:30 select * from customers LIMIT 0, 50000 7 row(s) returned
```

Resultado:

dni	first_name	last_name	phone_number	registration_date
22445566	Laura	Agnay	0342-156378022	2013-03-13
25335999	Maria	Benitez	0342-154139913	2022-12-18
33445566	Juan	Piquete	011-1567899879	2015-05-15
33445588	Andrea	Lira	0342-155398022	2011-01-11
33556677	Esteban	Quito	011-1561233219	2014-04-14
34347788	Carlos	Calabresa	011-1564566549	2016-06-16
43132313	Matias	Arias	0342-154266141	2018-12-09

clientes_que_superan

En este segundo procedimiento que llamaremos clientes_que_superan tendremos que utilizar la función creada antes (ventas_cliente) para obtener las ventas y luego filtrar sólo aquellos clientes que superen la cantidad de ventas indicada, es decir que tendremos un parámetro de entrada (cantidad_ventas) y un cursor de salida (clientes) donde se almacenarán los clientes que cumplan la condición mencionada

Para la resolución de este punto me encontré con el problema de PL/SQL si permite tener un cursor como parámetro de salida pero como en este caso estoy usando MySql como motor de base de datos y este no permite esto. Por lo que no puedo resolver el punto de forma estricta como lo pide la consigna.

Por lo que voy a proponer tres formas de resolverlo:

- Resolución en PL/SQL
- Resolución en MySql con 1 parámetro
- Resolución en MySql con tabla temporal

Resolución en PL/SQL

```
DELIMITER $$
CREATE PROCEDURE `clientes que superan` (
    cantidad_ventas INTEGER,
    clientes    IN OUT SYS_REFCURSOR
)
BEGIN
    OPEN clientes FOR SELECT *  FROM customers
WHERE ventas_cliente(c.dni)>cantidad_ventas;
END$$
DELIMITER ;
```

Resolución en MySql con 1 Parámetro

```
DELIMITER $$
CREATE PROCEDURE `clientes que superan`(
cantidad_ventas INTEGER
)
BEGIN
      SELECT * FROM customers c where
ventas_cliente(c.dni)>cantidad_ventas;
END $$
DELIMITER ;

call clientes_que_superan(5);
```

Resolución en MySql con tabla temporal

```
DELIMITER $$
CREATE PROCEDURE `clientes_que_superan`(
cantidad_ventas INT
)
BEGIN
      DECLARE listo BOOLEAN DEFAULT FALSE;
      DECLARE dni_p VARCHAR(9);
      DECLARE first_name_p VARCHAR(50);
      DECLARE last_name_p VARCHAR(50);
      DECLARE phone_number_p VARCHAR(15);
      DECLARE registration_date_p DATE;
    DECLARE c1 CURSOR FOR SELECT * FROM customers c where
ventas_cliente(c.dni)>cantidad_ventas;
```

```
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET listo=TRUE;
    OPEN c1;
     c1_loop:LOOP
           FETCH c1 INTO
dni_p,first_name_p,last_name_p,phone_number_p,registration_date_p;
        IF listo=TRUE THEN
                 LEAVE c1_loop;
            END IF;
        INSERT INTO
clientes(dni,first_name,last_name,phone_number,registration_date) VALUES
(dni_p,first_name_p,last_name_p,phone_number_p,registration_date_p);
      END LOOP c1_loop;
CLOSE c1;
END $$
DELIMITER ;


DROP TABLE IF EXISTS clientes;
CREATE TEMPORARY TABLE clientes (
 dni VARCHAR(9) NOT NULL,
 first_name VARCHAR(50) NOT NULL,
 last_name VARCHAR(50) NOT NULL,
 phone_number VARCHAR(15) NOT NULL,
 registration_date DATE DEFAULT (CURRENT_DATE) NOT NULL
    );
call clientes_que_superan(5);

select * from clientes;
```

Resultado:

dni	first_name	last_name	phone_number	registration_date
22445566	Laura	Agnay	0342-156378022	2013-03-13
33445566	Juan	Piquete	011-1567899879	2015-05-15
33445588	Andrea	Lira	0342-155398022	2011-01-11
33556677	Esteban	Quito	011-1561233219	2014-04-14
34347788	Carlos	Calabresa	011-1564566549	2016-06-16

Explicación directorio

En el directorio SQL vas a encontrar los siguientes archivos:

- schema.sql : Archivo sql que contiene toda la estructura de la base de datos y la creación de procedimientos almacenados y funciones
- data.sql : Archivo sql que contiene toda la información e inserción de datos de la consigna

- queries.sql: Archivo sql que contiene todas las operaciones para resolver los distintos puntos de la consigna
- tercer_evaluacion_tecnica-DER.png: Imagen PNG que contiene el Diagrama de Entidad-Relación de la base de datos
- capturas/ : Directorio que contiene todos los resultados de las consultas utilizadas para resolver los distintos puntos de la consigna

¿Cómo utilizar los archivos .sql?

- Ejecuta el archivo schema.sql
- Ejecuta el archivo data.sql
- Ejecuta el archivo queries.sql