

Introducción

La idea de crear la base de datos “Comercio Minorista” surge a partir de la necesidad de Alfredo y Verónica, propietarios de un pequeño negocio, de comenzar a llevar un registro ordenado y sistemático de las ventas de su nuevo emprendimiento.

Cinco años atrás iniciaron su camino como emprendedores con una rotisería que operaba exclusivamente mediante delivery, impulsada por las restricciones de la pandemia. Lograron posicionarse gracias a una estrategia de difusión efectiva en redes sociales, especialmente en plataformas como Instagram y Facebook. A pesar del éxito comercial, decidieron cerrar el negocio tras dos años de actividad, principalmente debido al agotamiento físico y mental que implicaba sostener un emprendimiento con alta demanda y un equipo de trabajo poco estable.

Lejos de rendirse, alquilaron un local con excelente ubicación y abrieron un maxikiosco. Esta vez, el comercio no requería promoción en redes sociales, ya que se encontraba sobre una avenida muy transitada y justo en frente de un colegio de gran tamaño. Con esfuerzo y dedicación, lograron expandirse hasta convertirse en un almacén minorista que ofrecía una variedad de productos: artículos de almacén, lácteos, bebidas, fiambres, frutas, verduras y panificados. Sin embargo, los productos más vendidos eran los chocolates, snacks y golosinas.

En el último año, las ventas cayeron abruptamente, y el incremento en los costos de servicios y alquiler dificultó la reposición de mercadería. Como consecuencia, se vieron obligados a no renovar el contrato de alquiler. No obstante, decidieron continuar emprendiendo, esta vez desde su domicilio, con el objetivo de reducir costos fijos.

Mientras avanzan con los trámites necesarios para habilitar su nuevo comercio, se les remarcó la importancia de llevar un control detallado de las ventas, con el fin de garantizar la sostenibilidad del proyecto y evitar los errores cometidos en emprendimientos anteriores.

En este contexto, la persona que desarrolla el presente proyecto se ofreció voluntariamente para desempeñarse como su Analista de Datos personal, ya que conoce en profundidad sus necesidades, carencias y fortalezas, producto de haber trabajado junto a ellos desde el inicio de sus dos emprendimientos previos. Esta colaboración no solo aporta valor al desarrollo y control del nuevo negocio, sino que también representa una valiosa oportunidad de práctica y crecimiento profesional en el área del análisis de datos.

En este proyecto aún no se cuenta con datos reales para analizar, ya que la base de datos fue concebida como una herramienta que comenzará a utilizarse a partir de la inauguración del nuevo comercio, prevista para mediados del mes de agosto. Por lo tanto, se diseñará y estructurará una base de datos vacía, exclusiva para el negocio, que estará lista para comenzar a registrar información desde el primer día de operaciones.

Sin embargo, dado que este proyecto forma parte del curso “SQL Flex” de la plataforma Coderhouse —en el cual la persona que desarrolla el trabajo se encuentra actualmente inscrita—, se utilizarán datos ficticios generados manualmente. Esta simulación permitirá avanzar con las presentaciones prácticas requeridas por el curso, sin interferir con la futura implementación real en el negocio.

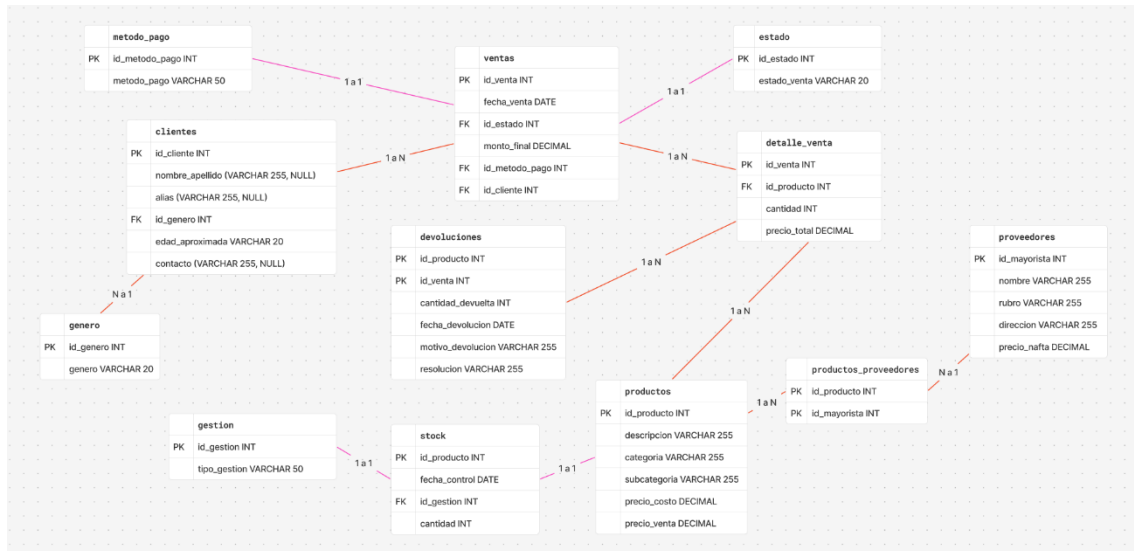
El proyecto se desarrollará en tres etapas bien diferenciadas:

1. **Primera etapa:** Análisis del negocio y su contexto, identificando las características, necesidades y desafíos propios del emprendimiento, para asegurar que el diseño de la base de datos sea coherente con la realidad operativa.
2. **Segunda etapa:** Creación de la base de datos en **MySQL Workbench**, donde se definirá la estructura, relaciones entre tablas, y se ejecutarán los scripts SQL necesarios para la creación de entidades, incluyendo la elaboración del diagrama entidad-relación (DER).
3. **Tercera etapa:** Esta fase se desarrollará una vez transcurridos los primeros seis meses de actividad comercial. Se recopilarán y analizarán los datos reales almacenados en la base, identificando insights clave sobre el comportamiento del negocio. A partir de ello, se crearán visualizaciones gráficas que faciliten la interpretación de los resultados y permitan proponer soluciones orientadas a la mejora continua y a la toma de decisiones informadas.

Segunda Etapa

Modelo de negocio y normalización

Luego del profundo análisis desarrollado en la primera etapa, se determina el modelo de negocio:



De izquierda a Derecha:

Tablas clientes y genero

En este tipo de comercio no es habitual que las ventas se registren con información personal detallada de los clientes. Por este motivo, se incorpora una columna denominada **“alias”**, ya que es común identificar a los clientes mediante referencias informales como “vecina” o “hija de Mónica”, aunque también pueden utilizarse nombres y apellidos reales.

Asimismo, los propios dueños del comercio realizarán consumos internos, dado que el negocio será su medio de subsistencia. Estos consumos quedarán registrados bajo el alias **“familia”**, el cual tendrá sentido al vincularse con la tabla *estado*, ya que ese monto no será efectivamente abonado.

Por su parte, la tabla género fue normalizada con el fin de cumplir con las formas normales, dado que se espera una reiteración de valores en dicho campo.

	clientes
PK	id_cliente INT
	nombre_apellido (VARCHAR 255, NULL)
	alias (VARCHAR 255, NULL)
FK	id_genero INT
	edad_aproximada VARCHAR 20
	contacto (VARCHAR 255, NULL)

	genero
PK	id_genero INT
	genero VARCHAR 20

Tablas `productos`, `stock` y `gestión`.

La tabla `productos` contiene toda la información relacionada con los productos ofrecidos en el comercio. Esta se encuentra vinculada con la tabla `stock`, a partir de la cual, mediante la columna `fecha_control`, se podrá realizar un seguimiento de la disponibilidad de cada producto en distintos momentos del tiempo.

A futuro, se prevé la implementación de un *trigger* que permita, mediante un *join*, restar automáticamente la cantidad correspondiente en stock cada vez que se registre una venta.

Por otra parte, la tabla `gestión` fue normalizada con el objetivo de cumplir con las formas normales, ya que se anticipa la reiteración de valores tales como “**ingreso**” y “**egreso**”, que refieren a los movimientos de inventario.

	productos
PK	id_producto INT
	descripcion VARCHAR 255
	categoria VARCHAR 255
	subcategoria VARCHAR 255
	precio_costo DECIMAL
	precio_venta DECIMAL

	stock
PK	id_producto INT
	fecha_control DATE
FK	id_gestion INT
	cantidad INT

	gestion
PK	id_gestion INT
	tipo_gestion VARCHAR 50

Tablas método_pago y estado

Las tablas `método_pago` y `estado` son el resultado de un proceso de normalización aplicado a la tabla `ventas`. Se prevé una alta reiteración de valores en ambas, lo que justifica su separación en tablas independientes.

En el caso de `método_pago`, se registrarán opciones frecuentes como **“efectivo”**, **“transferencia”**, **“tarjeta de débito”** y **“tarjeta de crédito”**.

Respecto a la tabla `estado`, se contemplan situaciones como **“completada”**, **“pendiente de pago”** y **“pago incompleto”**, las cuales permiten reflejar con mayor precisión el estado de cada operación comercial.

	método_pago
PK	id_metodo_pago INT
	metodo_pago VARCHAR 50

	estado
PK	id_estado INT
	estado_venta VARCHAR 20

Tablas ventas y detalle_venta.

La tabla `ventas` contendrá la información general y más relevante de cada operación de compra.

Por su parte, la tabla `detalle_venta` actuará como un desglose específico de cada venta registrada en la tabla principal. Permitirá identificar cada producto adquirido mediante su `id_producto`, indicando además la cantidad comprada y el precio correspondiente en ese momento para cada `id_venta`.

Esta estructura en dos niveles permite una mayor flexibilidad y escalabilidad, facilitando tanto el registro como el análisis detallado de los movimientos comerciales.

	ventas
PK	id_venta INT
	fecha_venta DATE
FK	id_estado INT
	monto_final DECIMAL
FK	id_metodo_pago INT
FK	id_cliente INT

	detalle_venta
PK	id_venta INT
FK	id_producto INT
	cantidad INT
	precio_total DECIMAL

Tabla devoluciones

Aunque se espera que esta tabla no sea utilizada con frecuencia, resulta fundamental para llevar un registro adecuado de los productos devueltos, ya sea por fallas o reclamos por parte de los clientes.

Contar con esta información permitirá no solo calcular pérdidas económicas, sino también identificar posibles patrones de reclamos, ya sea por tipo de producto, proveedor o comportamiento recurrente de ciertos clientes.

En ese sentido, fue necesario incorporar una **clave compuesta** conformada por id_venta e id_producto, ya que dicha combinación permitirá identificar de manera única cada devolución dentro del sistema.

	devoluciones
PK	id_producto INT
PK	id_venta INT
	cantidad_devuelta INT
	fecha_devolucion DATE
	motivo_devolucion VARCHAR 255
	resolucion VARCHAR 255

Tablas `productos_proveedores` y `proveedores`

La tabla `productos_proveedores` funciona como una tabla de unión o tabla puente, necesaria para resolver la relación de muchos a muchos (N a N) existente entre `productos` y `proveedores`. Esto se debe a que un mismo producto puede ser adquirido en distintos mayoristas, y a su vez, cada mayorista puede proveer diversos productos.

Por su parte, la tabla `proveedores` incluye atributos relevantes como **dirección** y **precio_nafta**, los cuales fueron incorporados con el objetivo de estimar el costo logístico. De esta manera, será posible calcular el gasto que implica retirar pedidos personalmente desde distintos mayoristas, lo que puede impactar directamente en la rentabilidad del negocio.

	productos_proveedores
PK	id_producto INT
PK	id_mayorista INT

	proveedores
PK	id_mayorista INT
	nombre VARCHAR 255
	rubro VARCHAR 255
	direccion VARCHAR 255
	precio_nafta DECIMAL

Creación y carga de la base de datos mediante MySQL Workbench

Como se explicó previamente, esta base de datos todavía no cuenta con datos reales, por lo que se decidió generar los datos manualmente mediante la herramienta de Office, Excel, para que sean lo más leal posible a los datos de un comercio argentino.

A continuación, se comparte el enlace al portfolio personal de la autora del proyecto, el cual contiene una solapa denominada “Proyectos en SQL” desde donde es posible descargar el script en formato SQL. En caso de no poder acceder directamente, copie y pegue el enlace en el navegador.

<https://jacqueline-n-lozano.github.io/portfolio/>

Vistas, funciones, procedimientos almacenados y triggers

Las queries reales utilizadas para los siguientes esquemas, se encuentran disponibles en el siguiente enlace:

<https://jacqueline-n-lozano.github.io/portfolio/mysql.html>

VISTAS

Las vistas serán una ventana hacia los datos que más serán consultados. En ese sentido, se decide crear una vista que muestre las ventas por día y el monto total recaudado:

```
CREATE VIEW vista_ventas_diarias AS
```

```
SELECT fecha_venta, SUM(monto_final) AS total_diario, COUNT(*) AS cantidad_ventas
```

```
FROM ventas
```

```
GROUP BY fecha_venta;
```

The screenshot shows a MySQL database interface. On the left, a sidebar lists various database objects like Tables, Views, Stored Procedures, Functions, and Schemas. The main area displays a SQL query editor with the following code:

```
4
5 -- Ventas por día y monto total
6 CREATE VIEW vista_ventas_diarias AS
```

Below the editor, a 'Result Grid' shows the data for the 'vista_ventas_diarias' view. The table has three columns: 'fecha_venta', 'total_diario', and 'cantidad_ventas'. The data is as follows:

fecha_venta	total_diario	cantidad_ventas
2025-01-10	39333	4
2025-01-11	3150	1
2025-01-15	12488	2
2025-01-20	5327	2
2025-01-25	27370	3
2025-02-01	1414	1
2025-02-02	952	1
2025-02-03	2828	1
2025-02-04	6762	1
2025-02-05	3696	1
2025-02-06	1876	1
2025-02-07	1386	1
2025-02-08	371	1
2025-02-09	21070	1

At the bottom, there is an 'Output' section with a dropdown menu set to 'Action Output'.

Ventas agrupadas por año y mes:

```
CREATE VIEW ventas_por_mes AS
```

```
SELECT
```

```
YEAR(fecha_venta) AS año,
```

```
MONTH(fecha_venta) AS mes,
```

```
COUNT(id_venta) AS cantidad_ventas,
```

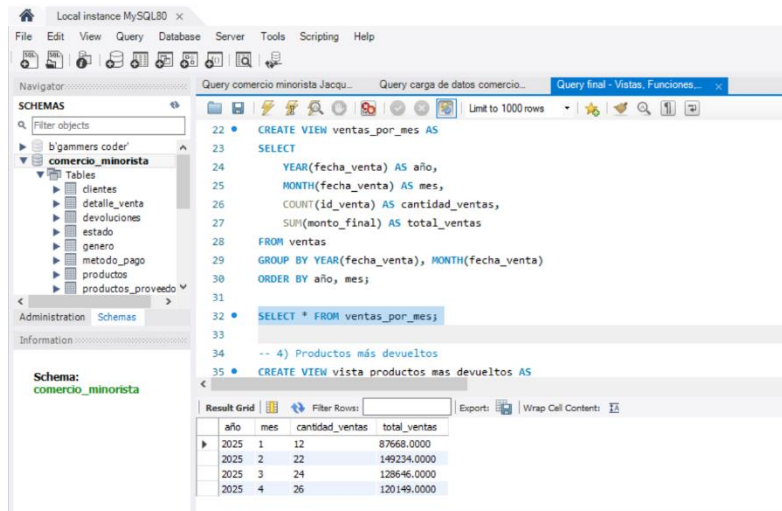
```
SUM(monto_final) AS total_ventas
```

```
FROM ventas
```

```
GROUP BY YEAR(fecha_venta), MONTH(fecha_venta)
```

```
ORDER BY año, mes;
```

```
SELECT * FROM ventas_por_mes;
```



Productos más vendidos:

```
CREATE VIEW vista_top_productos AS
```

```
SELECT p.id_producto, p.descripcion, SUM(dv.cantidad) AS cantidad_vendida,
SUM(dv.precio_total) AS ingresos
```

```
FROM detalle_venta dv
```

```
INNER JOIN productos p ON dv.id_producto=p.id_producto
```

```
GROUP BY p.id_producto, p.descripcion
```

```
ORDER BY cantidad_vendida DESC;
```

io_minorista

```

15 SELECT p.id_producto, p.descripcion, SUM(dv.cantidad) AS car
16 FROM detalle_venta dv
17 INNER JOIN productos p ON dv.id_producto=p.id_producto
18 GROUP BY p.id_producto, p.descripcion

```

Result Grid

id_producto	descripcion	cantidad_venta	ingresos
2	fanta 500cc	7	9604
3	sprite 500cc	7	9604
50	banana 1kg	7	14700
44	zanahoria 1kg	7	9604
1	coca cola 500cc	6	8232
6	seven up 500cc	6	6300
20	leche la serenissima	5	9940
34	pollo - trozado 1kg	5	14700
8	baggio naranja 200cc	5	1855
9	baggio manzana 200cc	5	1855
36	pollo - milanese 1kg	5	26600
37	pollo - picada	5	14700
24	yogurt frutilla treggar...	5	12530
13	manaos naranja 2250cc	5	7070

vista_top_productos 2 x

Output

Action Output

5 21:20:07 SELECT * FROM class LIMIT 0, 1000

Productos más devueltos:

CREATE VIEW vista_productos_mas_devueltos AS

SELECT

p.id_producto,

p.descripcion AS producto,

SUM(d.cantidad_devuelta) AS total_devueltos,

COUNT(DISTINCT d.id_venta) AS cantidad_ventas_con_devolucion

FROM devoluciones d

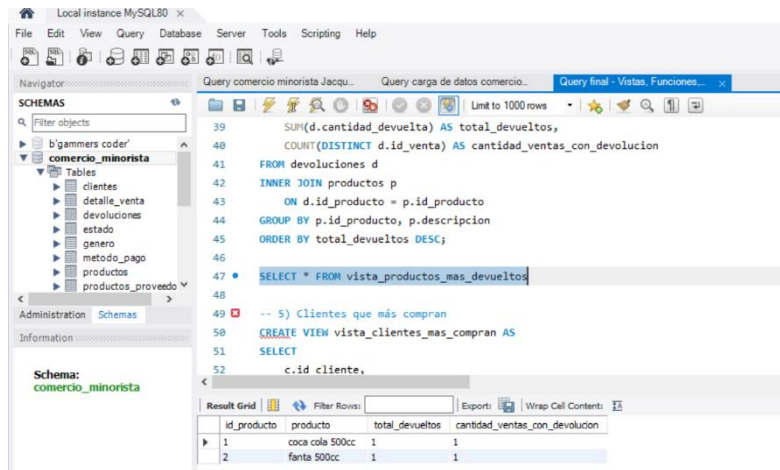
INNER JOIN productos p

ON d.id_producto = p.id_producto

GROUP BY p.id_producto, p.descripcion

ORDER BY total_devueltos DESC;

SELECT * FROM vista_productos_mas_devueltos



Clientes que más compran:

```
CREATE VIEW vista_clientes_mas_compran AS
```

```
SELECT
```

```
c.id_cliente,
```

```
c.nombre_apellido,
```

```
c.alias,
```

```
COUNT(v.id_venta) AS cantidad_compras,
```

```
SUM(v.monto_final) AS monto_total_gastado
```

```
FROM ventas v
```

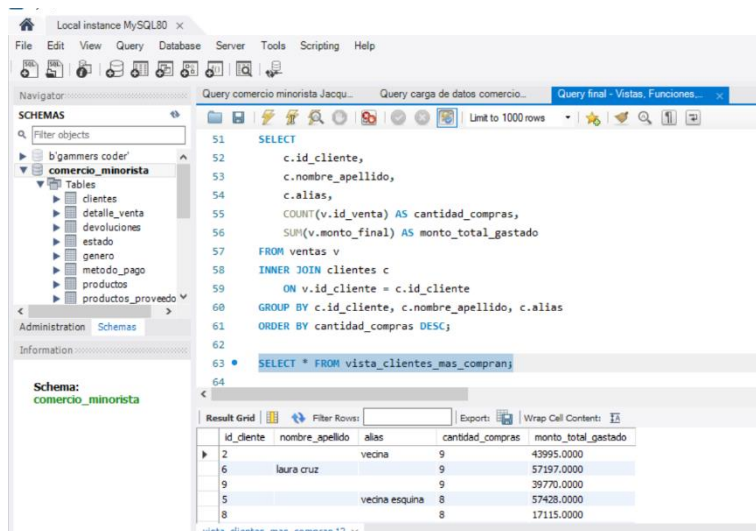
```
INNER JOIN clientes c
```

```
ON v.id_cliente = c.id_cliente
```

```
GROUP BY c.id_cliente, c.nombre_apellido, c.alias
```

```
ORDER BY cantidad_compras DESC;
```

```
SELECT * FROM vista_clientes_mas_compran;
```



FUNCIONES

Se decide crear una función que muestre el rendimiento de cada producto, de esa manera se podrá calcular la ganancia neta que dejará cada producto.

```
DELIMITER //
```

```
CREATE FUNCTION margen_producto(p_id_producto INT)
```

```
RETURNS DECIMAL(10,2)
```

```
DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE v_margen DECIMAL(10,2);
```

```
    SELECT (precio_venta - precio_costo) INTO v_margen
```

```
    FROM productos
```

```
    WHERE id_producto = p_id_producto;
```

```
    RETURN v_margen;
```

```
END//
```

```
DELIMITER ;
```

```

31
32 SELECT (precio_venta - precio_costo) INTO v_margen
33 FROM productos
34 WHERE id_producto = p_id_producto;
35

```

id_producto	description	precio_costo	precio_venta	margen
1	coca cola 500cc	980	1372	392.00
2	fanta 500cc	980	1372	392.00
3	sprite 500cc	980	1372	392.00
4	pepsi 500cc	750	1050	300.00
5	mirinda 500cc	750	1050	300.00
6	seven up 500cc	750	1050	300.00
7	baggio multifruta 200cc	265	371	106.00
8	baggio naranja 200cc	265	371	106.00
9	baggio manzana 200cc	265	371	106.00
10	coca cola 1500cc	2700	3780	1080.00
11	coca cola 2250cc	3100	4340	1240.00
12	manaos cola 2250cc	1010	1414	404.00
13	manaos naranja 2250cc	1010	1414	404.00
14	manaos pomelo 2250cc	1010	1414	404.00

La siguiente se trata de una función que calcula el monto vendido cada cierto período, de esta manera será posible visualizar el monto generado durante fechas que podrían resultar útiles, como por ejemplo: meses, trimestres, cuatrimestres, etc.

```
DELIMITER//
```

```
CREATE FUNCTION calcular_ventas_por_periodo(fecha_inicio DATE, fecha_fin DATE)
```

```
RETURNS DECIMAL(12, 4)
```

```
DETERMINISTIC
```

```
BEGIN
```

```
-- Declara una variable para almacenar el monto total
```

```
DECLARE total_ventas DECIMAL(12, 4);
```

```
-- Calcula la suma del 'monto_final' para todas las ventas entre las fechas de inicio y fin.
```

```
SELECT SUM(monto_final)
```

```
INTO total_ventas
```

```
FROM ventas
```

```
WHERE fecha_venta BETWEEN fecha_inicio AND fecha_fin;
```

```
IF total_ventas IS NULL THEN
```

```
RETURN 0.00;
```

```
ELSE
```

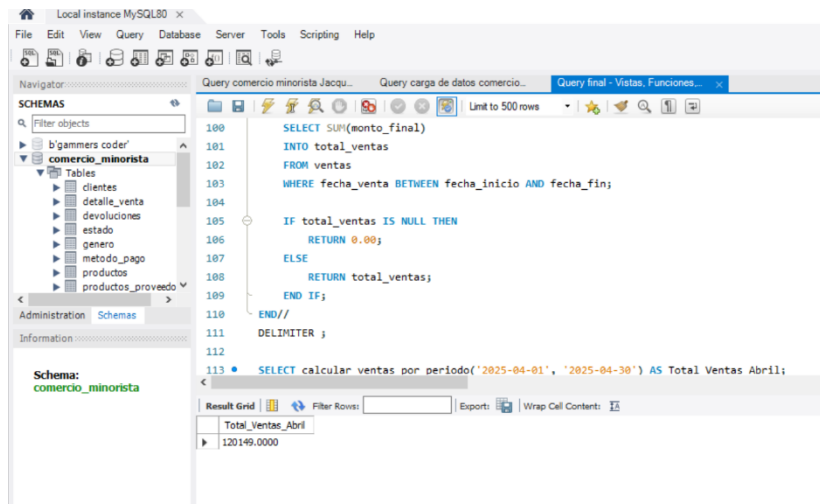
```
RETURN total_ventas;
```

```
END IF;
```

```
END//
```

```
DELIMITER ;
```

```
SELECT calcular_ventas_por_periodo('2025-04-01', '2025-04-30') AS Total_Ventas_Abril;
```



PROCEDIMIENTOS ALMACENADOS

La decisión de crear el siguiente *procedimiento almacenado* se debe a que, constantemente se insertarán datos de nuevas ventas, con este *procedure* la carga de datos será un proceso menos repetitivo y más ágil, reduciendo la posibilidad de errores.

```
DELIMITER //
```

```
CREATE PROCEDURE registrar_venta(
```

```
IN p_fecha DATE,
```

```
IN p_monto DECIMAL(10,2),
```

```
IN p_metodo INT,
```

```
IN p_estado INT,
```

```
IN p_cliente INT,
```

```
    IN p_id_producto INT,

    IN p_cantidad INT

)

BEGIN

    DECLARE v_precio DECIMAL(10,2);

    DECLARE v_new_id INT;


    SELECT IFNULL(MAX(id_venta), 0) + 1 INTO v_new_id FROM ventas;

    INSERT INTO ventas (id_venta, fecha_venta, monto_final, id_metodo_pago, id_estado,
id_cliente)

    VALUES (v_new_id, p_fecha, p_monto, p_metodo, p_estado, p_cliente);


    SELECT precio_venta INTO v_precio

    FROM productos

    WHERE id_producto = p_id_producto;


    INSERT INTO detalle_venta (id_venta, id_producto, cantidad, precio_total)

    VALUES (v_new_id, p_id_producto, p_cantidad, (v_precio * p_cantidad));

END//

DELIMITER ;
```


SQL Editor:

```

79  -- Comprobamos si funciona
80
81  CALL registrar_venta('2025-08-18', 5600.00, 1, 1, 1, 1, 2);
82
83  SELECT * FROM detalle_venta;

```

Result Grid:

	id_venta	id_producto	cantidad	precio_total
1	1	2	2744	
1	34	3	8820	
2	9	3	1113	
2	23	2	5544	
2	44	1	1372	
3	33	1	4340	
3	36	2	10640	
4	20	1	1988	
4	45	2	2772	
5	6	3	3150	
6	17	2	3808	
7	38	1	8680	
8	7	1	371	
9	25	2	4956	
10	30	3	3276	

Output:

```

79 23:56:36 CREATE PROCEDURE registrar_venta( IN p_fecha DATE, IN p_monto DEC

```

El siguiente *stored procedure* gestiona una devolución y restituye el stock correspondiente del producto devuelto siempre y cuando la resolución del conflicto sea catalogada como “Reintegrado al stock”. De esta manera, el stock permanecerá actualizado sin posibilidad de errores u omisiones por olvido.

```
CREATE PROCEDURE registrar_devolucion (
```

```
    IN p_id_producto INT,
```

```
    IN p_id_venta INT,
```

```
    IN p_cantidad_devuelta INT,
```

```
    IN p_fecha DATE,
```

```
    IN p_motivo VARCHAR(255),
```

```
    IN p_resolucion VARCHAR(255)
```

```
)
```

```
BEGIN
```

```
-- Insertar la devolución
```

```
INSERT INTO devoluciones (
```

```
    id_producto, id_venta, cantidad_devuelta, fecha_devolucion, motivo_devolucion,
    resolucion
```

```
)
```

```
VALUES (
```

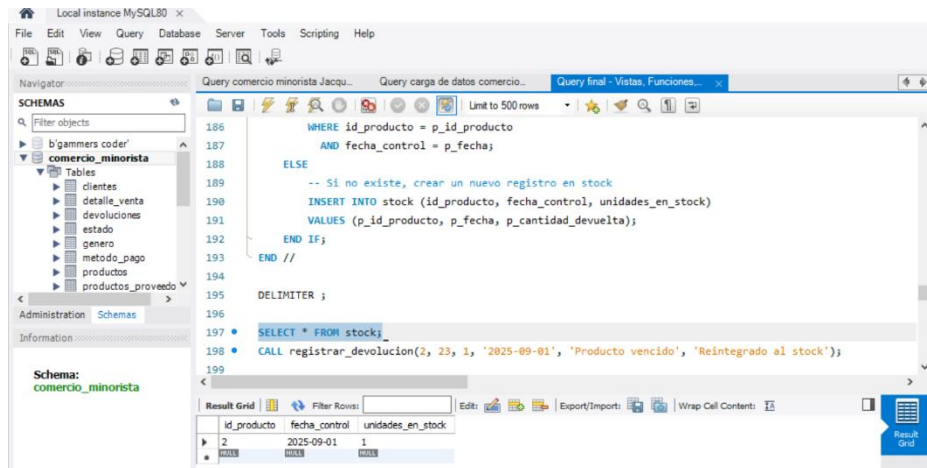
```
    p_id_producto, p_id_venta, p_cantidad_devuelta, p_fecha, p_motivo, p_resolucion
```

```
);
```

```
IF EXISTS (  
    SELECT 1 FROM stock  
    WHERE id_producto = p_id_producto  
    AND fecha_control = p_fecha  
    ) THEN  
    -- Si existe, actualizar sumando la devolución  
    UPDATE stock  
    SET unidades_en_stock = unidades_en_stock + p_cantidad_devuelta  
    WHERE id_producto = p_id_producto  
    AND fecha_control = p_fecha;  
ELSE  
    -- Si no existe, crear un nuevo registro en stock  
    INSERT INTO stock (id_producto, fecha_control, unidades_en_stock)  
    VALUES (p_id_producto, p_fecha, p_cantidad_devuelta);  
END IF;  
END //
```



```
DELIMITER ;  
  
CALL registrar_devolucion(2, 23, 1, '2025-09-01', 'Producto vencido', 'Reintegrado al stock');
```



TRIGGERS

El siguiente *trigger* fué ideado desde un principio para poder actualizar la tabla 'stock' a medida que surgiera una nueva venta:

```
DELIMITER //
```

```
CREATE TRIGGER tr_descuento_stock
```

```
AFTER INSERT ON detalle_venta
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    DECLARE v_fecha DATE; -- Variable para guardar la última fecha de control
```

```
    SELECT MAX(fecha_control)
```

```
    INTO v_fecha
```

```
    FROM stock
```

```
    WHERE id_producto = NEW.id_producto;
```

```
    -- Actualizamos
```

```
    UPDATE stock
```

```
    SET unidades_en_stock = unidades_en_stock - NEW.cantidad
```

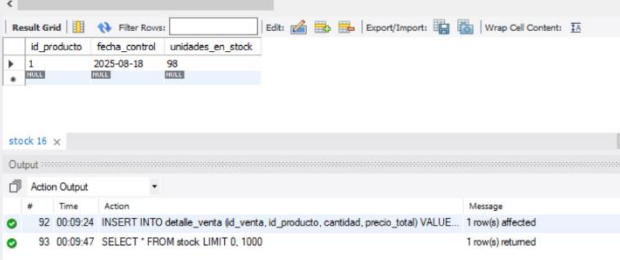
```
WHERE id_producto = NEW.id_producto
```

```
AND fecha_control = v_fecha;
```

```
END//
```

```
DELIMITER ;
```

```
102
103 -- Actualizamos
104 UPDATE stock
105 SET unidades_en_stock = unidades_en_stock - NEW.cantidad
106 WHERE id_producto = NEW.id_producto
107 AND fecha_control = v_fecha;
108 END//
109
110 DELIMITER ;
111
112 -- Dato ficticio para la tabla stock, id_producto 1 se verá modificado con el trigger
113 INSERT INTO stock (id_producto, fecha_control, unidades_en_stock)
114 VALUES (1, '2025-08-18', 98);
115
```



id_producto	fecha_control	unidades_en_stock
1	2025-08-18	98

#	Time	Action	Message
92	00:09:24	INSERT INTO detalle_venta (id_venta, id_producto, cantidad, precio_total) VALUE...	1 row(s) affected
93	00:09:47	SELECT * FROM stock LIMIT 0, 1000	1 row(s) returned

El siguiente *trigger* se encarga de realizar los cálculos necesarios para que, al ingresar una nueva venta y al detallar cada producto comprado, calcule el 'monto_final' de cada 'id_venta' de manera automática. De esta manera se agiliza la carga de datos y se reduce la posibilidad de errores al sumar y multiplicar.

```
DELIMITER //
```

```
CREATE TRIGGER trg_actualizar_monto_venta
```

```
AFTER INSERT ON detalle_venta
```

```
FOR EACH ROW
```

```
BEGIN
```

```
DECLARE precio_prod DECIMAL(12, 2);
```

```
SELECT precio_venta INTO precio_prod
```

```
FROM productos
```

```
WHERE id_producto = NEW.id_producto;
```

UPDATE ventas

SET monto_final = monto_final + (precio_prod * NEW.cantidad)

WHERE id_venta = NEW.id_venta;

END//

DELIMITER ;

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator Query comercio minorista Jacqui... Query carga de datos comercio... Query final - Vistas, Funciones...

SCHEMAS

Filter objects

b'gammers coder

comercio_minorista

Tables

- clientes
- detalle_venta
- devoluciones
- estado
- genero
- metodo_pago
- productos
- productos_proveedor

Administration Schemas

Information

Schema: comercio_minorista

```
255 -- El monto se incrementa con el precio del nuevo producto multiplicado por la cantidad
256 UPDATE ventas
257 SET monto_final = monto_final + (precio_prod * NEW.cantidad)
258 WHERE id_venta = NEW.id_venta;
259 END//
260 DELIMITER ;
261
262 -- Ahora, primero ingreso una venta sin calcular el monto_final, el trigger se activará luego
263 -- y modificaré la tabla detalle_venta
264 INSERT INTO ventas (id_venta, fecha_venta, monto_final, id_metodo_pago, id_estado, id_cliente) VALUES (101
265 INSERT INTO detalle_venta (id_venta, id_producto, cantidad, precio_total) VALUES (101, 1, 1, 50.00);
266
267 SELECT * FROM ventas;
268
```

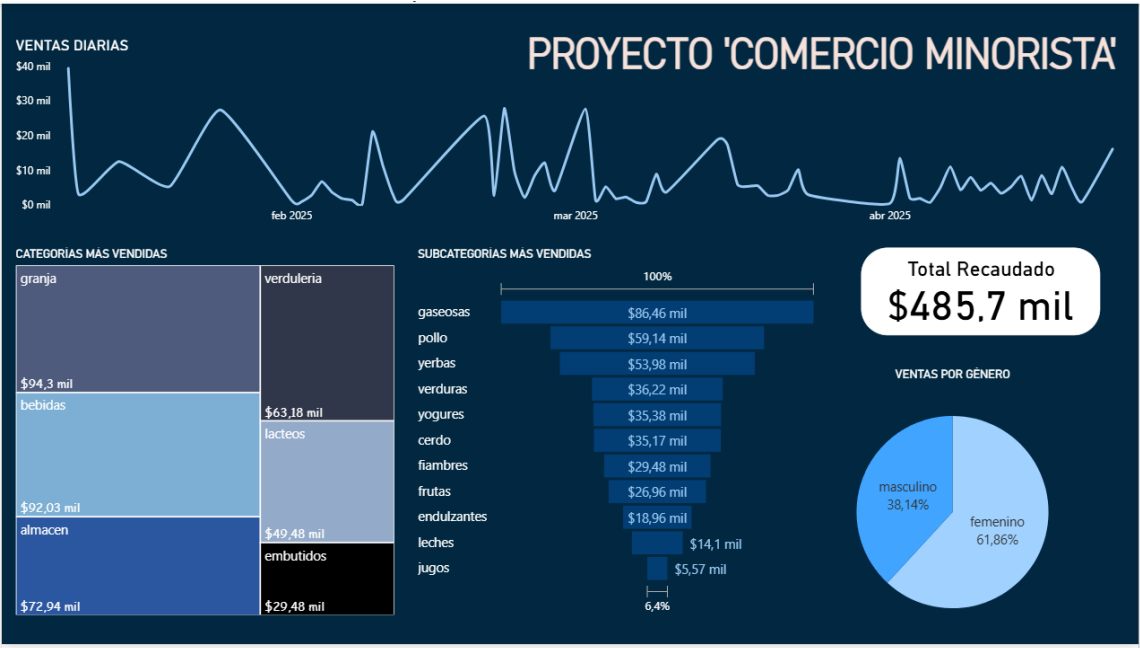
Result Grid

id_venta	fecha_venta	monto_final	id_metodo_pago	id_estado	id_cliente
82	2025-04-23	4116.0000	3	1	8
83	2025-04-23	6000.0000	1	1	9
84	2025-04-23	5999.0000	1	1	6
101	2025-09-01	1372.0000	1	1	1

Tercer Etapa

Para poder contar con *insights* claves y fieles a los datos cargados, se deciden generar gráficos mediante la herramienta de visualización Power BI, este breve gráfico estará disponible para descargar en el portfolio de quien redacta bajo el nombre de “Dashboard del Proyecto ‘Comercio Minorista’”

<https://jacqueline-n-lozano.github.io/portfolio/powerbi.html>



El desarrollo de este proyecto permitió recorrer de manera completa el proceso de construcción y análisis de una base de datos aplicada a un comercio minorista. La base de datos fue diseñada desde cero, considerando las necesidades particulares del negocio, y tras varias modificaciones se alcanzó una estructura robusta y normalizada bajo las tres primeras formas normales. Mediante MySQL Workbench se implementaron las tablas definitivas con sus claves primarias y foráneas, garantizando la integridad referencial. Posteriormente, se incorporaron vistas, funciones, procedimientos almacenados y *triggers*, que aportaron valor agregado y funcionalidad al sistema, siempre contemplando las limitaciones propias de una carga de datos por lote.

A fin de contar con un conjunto de información representativo, se generaron datos manualmente en Excel simulando un escenario realista de un comercio minorista argentino. Con esta base, y a través de Power BI, se elaboraron diferentes visualizaciones que permitieron obtener *insights* clave para la toma de decisiones:

- **Tendencia de ventas:** los registros muestran una leve disminución de las ventas a lo largo del período analizado, lo cual evidencia la necesidad de implementar campañas de marketing y promociones para reactivar la demanda.
- **Segmentación por género:** el análisis revela que las mujeres constituyen la mayoría de los consumidores. Esto sugiere diseñar acciones comerciales que refuercen la fidelidad de este segmento, a la vez que se desarrollen estrategias específicas para atraer al público masculino.
- **Categorías más vendidas:** los rubros de *granja*, *almacén*, *bebidas* y *verdulería* concentran la mayor parte de la facturación. Este patrón indica la importancia de mantener precios competitivos y disponibilidad en estas categorías, ya que representan el núcleo de los ingresos.
- **Subcategorías más relevantes:** productos como gaseosas, pollo y yerbas destacan en volumen de ventas. Estos artículos pueden considerarse estratégicos para campañas de cross-selling o descuentos combinados.

En conclusión, el trabajo integró tanto la parte técnica (modelado, normalización, desarrollo en SQL) como el análisis de negocio (lectura de insights en Power BI), mostrando cómo una base de datos correctamente diseñada y gestionada puede convertirse en una herramienta clave para la toma de decisiones en un comercio minorista.