

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.

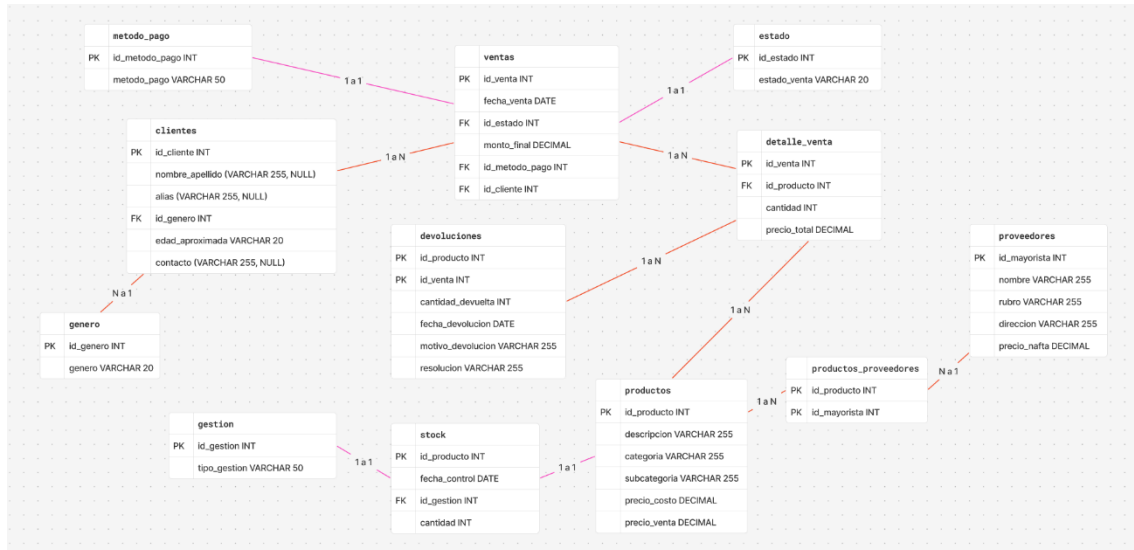
El proyecto se desarrollará en tres etapas bien diferenciadas:

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 - (Primer Entrega Coderhouse)

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.

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.

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.

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.

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.

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.

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 real 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/>

Segunda Entrega Coderhouse

Para poder idear *vistas*, *funciones*, *procedimientos almacenados* y *triggers*, es necesario pensar en el funcionamiento de la base de datos del negocio. Como se aclaró previamente, se trabajará con datos que se actualizarán cada cierta cantidad de tiempo y no con datos en tiempo real.

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

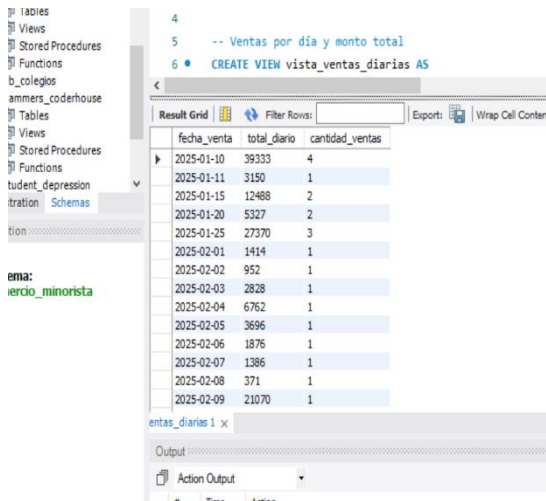
Las vistas serán una ventana hacia los datos que más serán consultados. En ese sentido, se decide crear una vista que muestre las ventas por día y el monto total recaudado:

```
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 including 'Views'. 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'.

También se crea una vista con los productos más vendidos

```
CREATE VIEW vista_top_productos AS
```

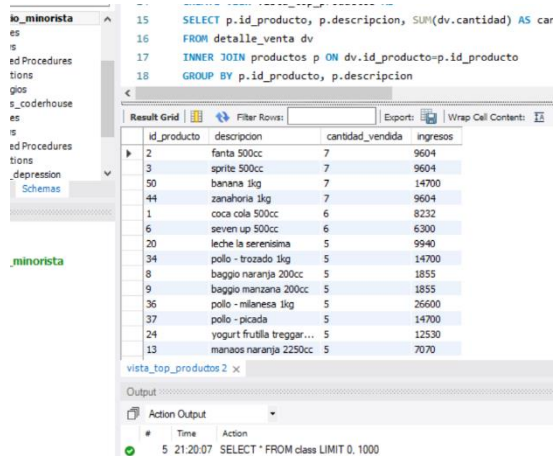
```
SELECT p.id_producto, p.descripcion, SUM(dv.cantidad) AS cantidad_vendida,  
SUM(dv.precio_total) AS ingresos
```

```
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;
```



The screenshot shows a database management tool interface. On the left, there's a sidebar with a tree view containing 'io_minorista', 'es', 'is', 'ed Procedures', 'tions', 'gos', 's_coderhouse', 'as', 'is', 'ed Procedures', 'depression', and 'Schemas'. The main area displays a SQL query in a text editor:

```
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
```

Below the query, there's a 'Result Grid' showing the results of the query. The grid has four columns: 'id_producto', 'descripcion', 'cantidad_vendida', and 'ingresos'. The data is as follows:

id_producto	descripcion	cantidad_vendida	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

At the bottom, there's an 'Output' section with a dropdown menu set to 'Action Output'. It shows a log entry: '5 21:20:07 SELECT * FROM class LIMIT 0, 1000'.

Funciones

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

```
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 ;

The screenshot shows a SQL IDE with a query editor on the left and a results grid on the right. The query in the editor is:

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

The results grid displays the following data:

id_producto	descripcion	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

Stored Procedure

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; -- Esto es porque id_venta no es autoincrement y se deberá calcular
```

```
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
```

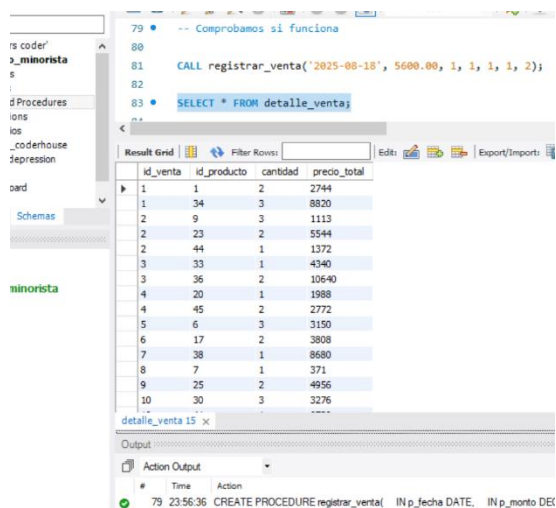
```
WHERE id_producto = p_id_producto;
```

```
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 ;
```



The screenshot shows a database IDE with a SQL editor on the left and a result grid on the right. The SQL editor contains the following code:

```
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;
```

The result grid displays the following data:

	id_venta	id_producto	cantidad	precio_total
1	1	2	2	2744
1	34	3	3	8820
2	9	3	3	1113
2	23	2	2	5544
2	44	1	1	1372
3	33	1	1	4340
3	36	2	2	10640
4	20	1	1	1988
4	45	2	2	2772
5	6	3	3	3190
6	17	2	2	3808
7	38	1	1	8680
8	7	1	1	371
9	25	2	2	4956
10	30	3	3	3276

The bottom of the screenshot shows the 'Output' tab with the following message:

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

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', 100);
115
```

Result Grid

Filter Rows:

Edit: Export/Import: Wrap Cell Content:

	id_producto	fecha_control	unidades_en_stock
▶	1	2025-08-18	98
✱			

stock 16

Output

Action Output

#	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