Universidad Nacional Autónoma de México

facultad de ingeniería
división de ingeniería eléctrica, ingeniería en computación

Asignatura:

Bases de datos (Clave: 1644)

Presenta:

Proyecto Final
Caso de estudio Mueblería.

Ingeniería en Computación

Integrantes:

Arellanes Conde Esteban
Martínez Jiménez Alejandro
Méndez Galicia Axel Gael
Sánchez Martínez Ximena
Soriano Barrera María Elena

Profesor:

Ing. Fernando Arreola Franco

Horarios: 07:00 a 09:00 hrs.
Lun, Mie, Vie - A305 (PRESENCIAL)
Ciudad Universitaria, CDMX 2020

Capítulo 1

Introducción

1.1. Objetivo

El alumno analizará una serie de requerimientos y propondrá una solución que atienda a los mismos, aplicando los conceptos vistos en el curso.

1.1.1. Descripción del problema

El problema se divide en dos partes:

1.1.2. Parte uno:

Consiste en el diseño de una base de datos. Una mueblería quiere digitalizar su forma de operar para evitar tener sus registros de forma física y tener una vista completa de la información de todas sus sucursales, por lo que se plantea el siguiente requerimiento:

1.1.3. Parte dos:

Una vez diseñada y lista la base de datos, se debe crear un dashboard que permita visualizar:

1. Cantidad de ingresos totales mensuales por sucursal en un año determinado
2. Otra gráfica que considere relevante para usuarios de negocio

1.2. Propuesta de solución

Brevemente nuestra aproximación de la resolución al caso de estudio se encuentra explicada más en detalle en la sección 2.

Índice general

Índice de figuras

Capítulo 2

Desarrollo

2.1. Plan de trabajo:

2.1.1. División de tareas

Tarea	Responsable
Parte 1 - Caso de estudio	(Todos)
Parte 2 - Dashboard	(Ximena)
Presentación	(María)
Exposición - Persona a exponer	(Todos)
Documento LaTeX	(Esteban)

Tabla 2.1: Asignación y división de tareas

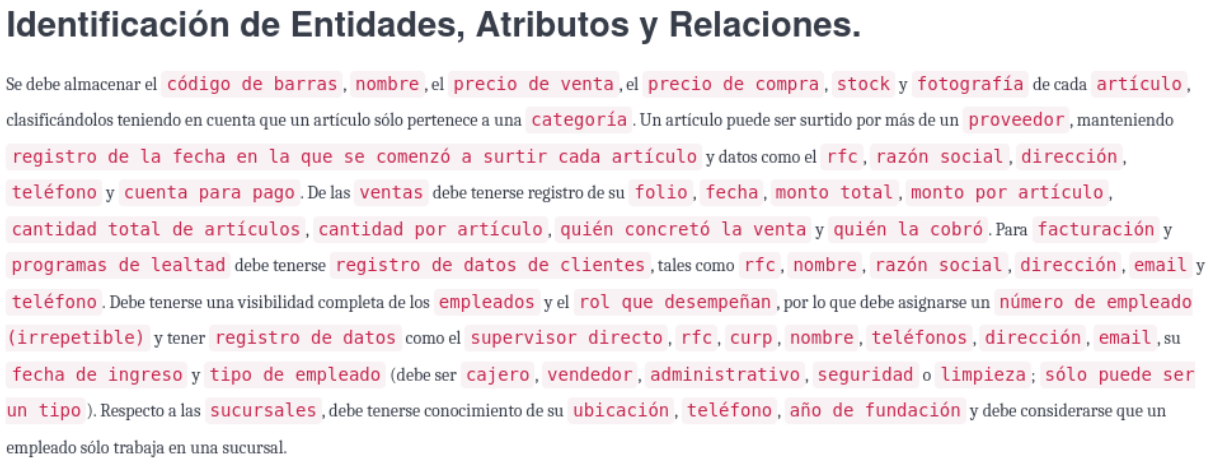
2.2. Diseño:

Descripción de lo realizado en las correspondientes fases de diseño de las bases de datos, agregando los resultados de cada una de ellas.

2.2.1. Análisis de Requerimientos

Caso de estudio: Mueblería - Analisis de Requerimiento

Consiste en el diseño de una base de datos. Una mueblería quiere digitalizar su forma de operar para evitar tener sus registros de forma física y tener una vista completa de la información de todas sus sucursales, por lo que se plantea el siguiente requerimiento:



Identificación de Entidades, Atributos y Relaciones.

Se debe almacenar el `código de barras`, `nombre`, el `precio de venta`, el `precio de compra`, `stock` y `fotografía` de cada `artículo`, clasificándolos teniendo en cuenta que un artículo sólo pertenece a una `categoría`. Un artículo puede ser surtido por más de un `proveedor`, manteniendo `registro de la fecha en la que se comenzó a surtir cada artículo` y datos como el `rfc`, `razón social`, `dirección`, `teléfono` y `cuenta para pago`. De las `ventas` debe tenerse registro de su `folio`, `fecha`, `monto total`, `monto por artículo`, `cantidad total de artículos`, `cantidad por artículo`, `quién concretó la venta` y `quién la cobró`. Para `facturación` y `programas de lealtad` debe tenerse `registro de datos de clientes`, tales como `rfc`, `nombre`, `razón social`, `dirección`, `email` y `teléfono`. Debe tenerse una visibilidad completa de los `empleados` y el `rol que desempeñan`, por lo que debe asignarse un `número de empleado (irrepetible)` y tener `registro de datos` como el `supervisor directo`, `rfc`, `curp`, `nombre`, `teléfonos`, `dirección`, `email`, su `fecha de ingreso` y `tipo de empleado` (debe ser `cajero`, `vendedor`, `administrativo`, `seguridad` o `limpieza`; `sólo puede ser un tipo`). Respecto a las `sucursales`, debe tenerse conocimiento de su `ubicación`, `teléfono`, `año de fundación` y debe considerarse que un empleado sólo trabaja en una sucursal.

Figura 2.1: Análisis de Requerimientos

2.2.2. Entidades y Atributos

- **Artículo:** Atributos:
  - Código de barras
  - nombre
  - precio de venta
  - precio de compra
  - stock
  - fotografía
  - Categoría
- **Proveedor:**
- Hist. Proveedor (Registro de la fecha en la que se comenzó a surtir cada artículo) Atributos:
  - rfc
  - razón social
  - dirección
  - teléfono
  - cuenta para pago
- **Ventas:** Atributos:
  - folio
  - fecha

  - monto total
  - monto por artículo
  - cantidad total de artículos
  - cantidad por artículo
  - quién concreto la venta (empleado_id fk)
  - quién cobró la venta (cliente_id fk)

- **Clientes-Ventas** (Intersección entre ambos conjuntos)
  - facturación
  - programas de lealtad

- **Clientes:** Atributos:
  - rfc
  - nombre
  - razón social
  - dirección
  - email
  - teléfono* (específica sólo 1)

- **Empleados:** Atributos:
  - rol que desempeñan
  - tipo de empleado (cajero, vendedor, administrativo, seguridad, limpieza. Sólo uno*)
  - número de empleado (irrepetible)
  - registro de datos*
  - supervisor directo (Opcional en caso de no tener)
  - rfc
  - curp
  - nombre
  - teléfonos
  - dirección
  - email
  - fecha de ingreso

- **Sucursales:** Atributos:
  - ubicación
  - teléfono (1)
  - año de fundación

2.2.3. Relaciones

- Artículo-Categoría (1:1): “Un artículo sólo pertenece a una categoría"
- Artículo-Proveedor (1:m ó m:m): “Un artículo puede ser surtido por más de un proveedor"
- Artículo-Ventas (1:m ó m:m): “Uno o muchos artículos pueden realizar muchas ventas"
- Clientes-Ventas (1:m ó m:m): “Uno o muchos clientes pueden realizar muchas compras (ventas)"
- Empleado-Sucursal (1:1): “Un empleado sólo trabaja en una sucursal"

2.2.4. Diseño Conceptual

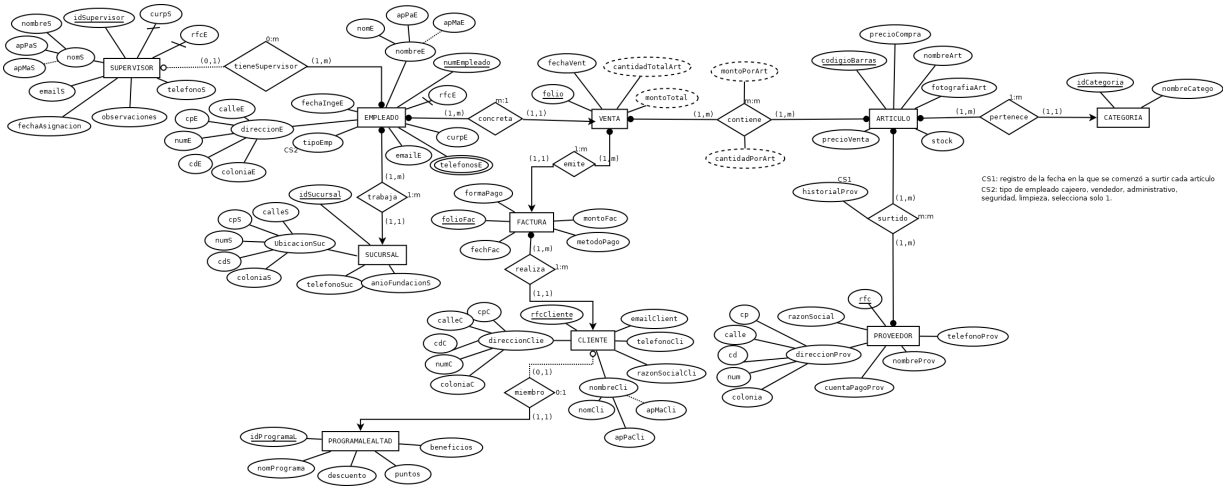


Figura 2.2: Diseño MER

2.2.5. Diseño Relacional

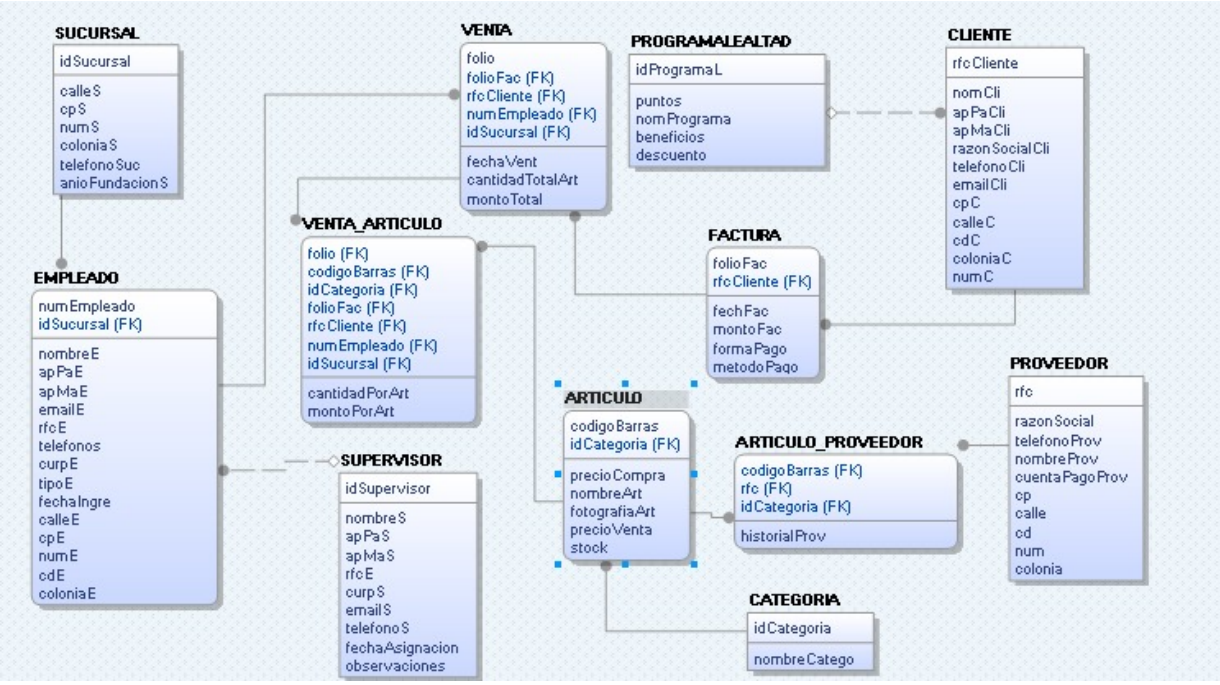


Figura 2.3: Diseño MR

Adicionalmente, deben cumplirse los siguientes puntos:

> Cada que se agregue un artículo a una venta, debe actualizarse los totales (por artículo, venta y cantidad de artículos), así como validar que el artículo esté disponible.

> Crear al menos, un índice, del tipo que se prefiera y donde se prefiera.

> Justificar el porqué de la elección en ambos aspectos.

> Lista de artículos no disponibles o con stock <3. Si el artículo no está disponible, debe aparecer el mensaje ”No disponible”.

> De manera automática se genere una vista que contenga información necesaria para asemejarse a un ticket de venta, incluyendo un folio para facturación (caracteres aleatorios).

> Al iniciar una venta, debe validarse que el vendedor y el cajero, pertenezcan a la misma sucursal, en caso de que no sea así, debe mostrarse un mensaje de error.

> Dado el nombre de un empleado, obtener toda la jerarquía organizacional

Capítulo 3

Implementación:

Descripción de funcionamiento y código de los stored procedures, triggers, funciones, etc. Empleados para cumplir con los requerimientos del problema.

3.1. Código

A través de la plataforma de github, `https://github.com/FernandoArreolaF/Bases1UNAM`, se creó una carpeta de nuestro equipo donde se anexa lo siguiente correspondiente a código:

- Códigos fuente de MER, MR
- Script de creación de la base de datos y tablas
- Script para el agregado de información
- Script de toda la programación a nivel BD
- Códigos de lo implementado como parte de la etapa de aplicación
- CSV's correspondientes a diccionario de datos, pruebas y dashboards
- Código fuente de nuestra aplicación web

```sql
-- Tabla SUCURSAL
CREATE TABLE SUCURSAL (
    idSucursal SERIAL PRIMARY KEY,
    calleS VARCHAR(100) NOT NULL,
    cpS CHAR(5) NOT NULL,
    numS VARCHAR(10) NOT NULL,
    cdS VARCHAR(100) NOT NULL,
    coloniaS VARCHAR(100) NOT NULL,
    telefonoSuc VARCHAR(20) NOT NULL,
    anioFundacionS INTEGER NOT NULL,
    CONSTRAINT chk_anio_fundacion CHECK (anioFundacionS > 0)
);

--Tabla Supervisor
CREATE TABLE SUPERVISOR (
    idSupervisor INTEGER PRIMARY KEY,
    nombreS VARCHAR(100) NOT NULL,
    apPaS VARCHAR(100) NOT NULL,
    apMaS VARCHAR(100),
    emailS VARCHAR(100) NOT NULL UNIQUE,
    telefonosS VARCHAR(20) NOT NULL,
    rfcS CHAR(13) NOT NULL UNIQUE,
    curpS CHAR(18) NOT NULL UNIQUE,
    fechaAsignacion DATE NOT NULL,
    observaciones TEXT,
    idSucursal INTEGER NOT NULL REFERENCES SUCURSAL(idSucursal)
);

-- Tabla EMPLEADO
CREATE TABLE EMPLEADO (
    numEmpleado SERIAL PRIMARY KEY,
    nombreE VARCHAR(100) NOT NULL,
    apPaE VARCHAR(100) NOT NULL,
    apMaE VARCHAR(100),
    emailE VARCHAR(100) NOT NULL UNIQUE,
    telefonosE VARCHAR(20) NOT NULL,
    rfcE CHAR(13) NOT NULL UNIQUE,
    curpE CHAR(18) NOT NULL UNIQUE,
    tipoEmp VARCHAR(30) NOT NULL CHECK (tipoEmp IN ('cajero', 'vendedor'
    , 'administrador', 'seguridad', 'limpieza')),
    fechIng DATE NOT NULL,
    calleE VARCHAR(100) NOT NULL,
    cpE CHAR(5) NOT NULL,
    numE VARCHAR(10) NOT NULL,
    cdE VARCHAR(100) NOT NULL,
    coloniaE VARCHAR(100) NOT NULL,
    idSucursal INTEGER NOT NULL REFERENCES SUCURSAL(idSucursal),
  idSupervisor INTEGER REFERENCES SUPERVISOR(idSupervisor)
);
```

```
54
55 -- Tabla CLIENTE
56 CREATE TABLE CLIENTE (
57     rfcCliente CHAR(13) PRIMARY KEY,
58     nomCli VARCHAR(100) NOT NULL,
59     apPaCli VARCHAR(100) NOT NULL,
60     apMaCli VARCHAR(100),
61     razonSocialCli VARCHAR(100) DEFAULT 'nombre completo',
62     telefonoCli VARCHAR(20) NOT NULL,
63     emailCli VARCHAR(100) NOT NULL UNIQUE,
64     cpC CHAR(5) NOT NULL,
65     calleC VARCHAR(100) NOT NULL,
66     cdC VARCHAR(100) NOT NULL,
67     coloniaC VARCHAR(100) NOT NULL,
68   numC SERIAL UNIQUE
69     );
70
71 -- Tabla CATEGORIA
72 CREATE TABLE CATEGORIA (
73     idCategoria SERIAL PRIMARY KEY,
74     nombreCatego VARCHAR(50) NOT NULL CHECK (char_length(nombreCatego) >
       0)
75 );
76
77 -- Tabla ARTICULO
78 CREATE TABLE ARTICULO (
79     codigoBarras VARCHAR(50) PRIMARY KEY,
80     precioCompra NUMERIC(10,2) NOT NULL CHECK (precioCompra >= 0),
81     nombreArt VARCHAR(100) NOT NULL,
82     fotografiaArt BYTEA,
83     precioVenta NUMERIC(10,2) NOT NULL CHECK (precioVenta >= 0),
84     stock INTEGER NOT NULL CHECK (stock >= 0),
85     idCategoria INTEGER NOT NULL REFERENCES CATEGORIA(idCategoria)
86 );
87
88 -- Tabla PROVEEDOR
89 CREATE TABLE PROVEEDOR (
90     rfc CHAR(13) PRIMARY KEY,
91     razonSocial VARCHAR(100) NOT NULL,
92     telefonoProv VARCHAR(20) NOT NULL,
93     nombreProv VARCHAR(100) NOT NULL,
94     cuentaPagoProv VARCHAR(50) NOT NULL,
95     cp CHAR(5) NOT NULL,
96     calle VARCHAR(100) NOT NULL,
97     cd VARCHAR(100) NOT NULL,
98     num VARCHAR(10) NOT NULL,
99     colonia VARCHAR(100) NOT NULL
100 );
101
102
103
104
105
106
```

```
107
108 -- Tabla PROVEEDOR_ARTICULO
109 CREATE TABLE PROVEEDOR_ARTICULO (
110     historialProv DATE NOT NULL,
111     rfc CHAR(13) NOT NULL REFERENCES PROVEEDOR(rfc) ON DELETE CASCADE,
112     codigoBarras VARCHAR(50) NOT NULL REFERENCES ARTICULO(codigoBarras)
    ON DELETE CASCADE,
113     PRIMARY KEY (rfc, codigoBarras)
114 );
115
116 CREATE TABLE VENTA (
117     folio SERIAL PRIMARY KEY,
118     fechaVent DATE NOT NULL,
119     cantidadTotalArt INTEGER NOT NULL CHECK (cantidadTotalArt >= 0),
120     montoTotal NUMERIC(10,2) NOT NULL CHECK (montoTotal >= 0),
121     numEmpleadoVendedor INTEGER NOT NULL REFERENCES EMPLEADO(numEmpleado
    ),
122     numEmpleadoCajero INTEGER NOT NULL REFERENCES EMPLEADO(numEmpleado),
123     CONSTRAINT chk_dos_empleados CHECK (numEmpleadoVendedor <>
    numEmpleadoCajero)
124 );
125
126
127 -- Tabla VENTA_DETALLE (VENTA_ARTICULO)
128 CREATE TABLE VENTA_DETALLE (
129     montoPorArt NUMERIC(10,2) NOT NULL CHECK (montoPorArt >= 0),
130     cantidadPorArt INTEGER NOT NULL CHECK (cantidadPorArt > 0),
131     folio INTEGER NOT NULL REFERENCES VENTA(folio),
132     codigoBarras VARCHAR(50) NOT NULL REFERENCES ARTICULO(codigoBarras),
133     PRIMARY KEY (folio, codigoBarras)
134 );
135
136
137 -- Tabla FACTURA
138 CREATE TABLE FACTURA (
139     folioFac INTEGER PRIMARY KEY,
140     fechFac DATE NOT NULL,
141     montoFac NUMERIC(10,2) NOT NULL,
142     formaPago VARCHAR(30) NOT NULL,
143     folio INTEGER NOT NULL REFERENCES VENTA(folio),
144     rfcCliente CHAR(13) REFERENCES CLIENTE(rfcCliente)
145 );
146
147
148 -- Tabla PROGRAMALEALTAD
149 CREATE TABLE PROGRAMALEALTAD (
150     idProgramaL SERIAL PRIMARY KEY,
151     nomPrograma VARCHAR(50) NOT NULL,
152     beneficios TEXT,
153     descuento NUMERIC(5,2) CHECK (descuento >= 0 AND descuento <= 100),
154     puntos INTEGER CHECK (puntos >= 0)
155 );
156
157
```

```
158
159 -------CONSULTAS----------
160 -------CONSULTAS----------
161 -------CONSULTAS----------
162 -------CONSULTAS----------
163 -------CONSULTAS----------
164
165
166 ----CLIENTE
167
168 SELECT * FROM cliente;
169 TRUNCATE TABLE cliente RESTART IDENTITY CASCADE;
170
171 ----supervisores
172
173 SELECT * FROM supervisor;
174 TRUNCATE TABLE supervisor RESTART IDENTITY CASCADE;
175
176 ----EMPLEADO
177
178 SELECT * FROM empleado;
179 TRUNCATE TABLE empleado RESTART IDENTITY CASCADE;
180
181 ----sucursal
182
183 SELECT * FROM sucursal;
184 TRUNCATE TABLE sucursal RESTART IDENTITY CASCADE;
185
186
187 --VENTAS
188 SELECT * FROM venta;
189 TRUNCATE TABLE venta RESTART IDENTITY CASCADE;
190
191 --facturas
192
193 SELECT * FROM factura;
194 TRUNCATE TABLE factura RESTART IDENTITY CASCADE;
195
196 --categoria
197
198 SELECT * FROM categoria;
199 TRUNCATE TABLE categoria RESTART IDENTITY CASCADE;
200
201 ---ARTICULO
202
203 SELECT * FROM articulo;
204 TRUNCATE TABLE articulo RESTART IDENTITY CASCADE;
205
206 --venta detalle
207 SELECT * FROM venta_detalle;
208 TRUNCATE TABLE venta_detalle RESTART IDENTITY CASCADE;
209
210
211
```

```
212
213 --PROVEEDOR
214
215 SELECT * FROM proveedor;
216 TRUNCATE TABLE proveedor RESTART IDENTITY CASCADE;
217
218 --PROVEEDOR articulo
219
220 SELECT * FROM proveedor_articulo;
221 TRUNCATE TABLE proveedor_articulo RESTART IDENTITY CASCADE;
222
223 --PASO 1:--
224 --Se meter n los datos del cliente, --
225 --si ya existe se sobrescribe si no existe se crea un nuevo cliente.--
226
227
228 INSERT INTO CLIENTE (
229     rfcCliente, nomCli, apPaCli, apMaCli, razonSocialCli,
230     telefonoCli, emailCli, cpC, calleC, cdC, coloniaC
231 ) VALUES (
232     'MOOA940615HXX',
233     'Amelia',
234     'Montanez',
235     'Orellana',
236     'Amelia Montanez Orellana',
237     '7089597717',
238     'patriciavarela@gmail.com',
239     '49789',
240     'Calzada Norte Figueroa',
241     'San Asuncion de la Montana',
242     'los altos'
243 )
244 ON CONFLICT (rfcCliente) DO UPDATE
245 SET nomCli = EXCLUDED.nomCli,
246     apPaCli = EXCLUDED.apPaCli,
247     apMaCli = EXCLUDED.apMaCli,
248     razonSocialCli = EXCLUDED.razonSocialCli,
249     telefonoCli = EXCLUDED.telefonoCli,
250     emailCli = EXCLUDED.emailCli,
251     cpC = EXCLUDED.cpC,
252     calleC = EXCLUDED.calleC,
253     cdC = EXCLUDED.cdC,
254     coloniaC = EXCLUDED.coloniaC,
255     numC = EXCLUDED.numC;
256
257
258 --PASO 2:--
259 --Se agrega los datos de la venta como fecha, cantidad total de
       art culos, monto total--
260 --y el numero de empleado tanto del vendedor como del cajero validando
       que no sean el mismo.--
261
262
263
```

```
264
265 INSERT INTO VENTA (
266     fechaVent,
267     cantidadTotalArt,
268     montoTotal,
269     numEmpleadoVendedor,
270     numEmpleadoCajero
271 ) VALUES (
272     '2026-08-11',  -- Formato ISO para la fecha
273     13,
274     26702,
275     13,
276     12
277 );
278
279
280 --PASO 3:--
281 --Se meten los datos para el detalle de la venta, en esta secci n
    encontraremos --
282 --monto por art culo, cantidad por articulo folio y el c digo de
    barras.--
283
284 INSERT INTO VENTA_DETALLE (
285     montoPorArt, cantidadPorArt, folio, codigoBarras
286 ) VALUES
287     (4157.98, 2, 1, '6650300000000'),
288     (1652.00, 1, 1, '1428510000000'),
289   (5565.00, 3, 1, '9344220000000'),
290   (3539.97, 3, 1, '2076980000000'),
291   (9881.97, 3, 1, '7751590000000'),
292   (1611.99, 1, 1, '9894720000000');
293
294
295
296 --PASO 4:
297 --La generaci n de la factura se agregar  la fecha, monto de la
    factura, la ---
298 --forma de pago, el folio de la venta y el RFC del cliente. ---
299
300
301 INSERT INTO FACTURA (
302     folioFac, fechFac, montoFac, formaPago, folio, rfcCliente
303 ) VALUES (
304     1,
305   '2023-08-11',
306   26702.00,
307   'efectivo',
308   1,
309   'MOOA940615HXX'
310 );
```

Listing 3.1: Tablas y Consultas

```
1
2  ---OBJETIVO 1---
3  --Cada que se agregue un art culo a una venta, debe actualizarse los
       totales
4  ----(por art culo , venta y cantidad de art culos), as  como validar
       que el articulo est  disponible.
5
6  -- Funci n
7  CREATE OR REPLACE FUNCTION actualizar_art()
8  RETURNS TRIGGER AS $$
9  DECLARE
10     stock_actual INT;
11     nuevo_stock INT;
12     precio_unitario NUMERIC(10,2);
13 BEGIN
14     -- Obtener el stock actual
15     SELECT stock, precioVenta INTO stock_actual, precio_unitario
16     FROM ARTICULO
17     WHERE codigoBarras = NEW.codigoBarras;
18
19     -- Validar existencia del art culo
20     IF stock_actual IS NULL THEN
21         RAISE EXCEPTION 'No se encontr  el art culo con c digo de
       barras %.', NEW.codigoBarras;
22     END IF;
23
24     -- Calcular nuevo stock
25     nuevo_stock := stock_actual - NEW.cantidadPorArt;
26
27     -- Validar stock suficiente
28     IF nuevo_stock < 0 THEN
29         RAISE EXCEPTION 'Stock insuficiente para el art culo con
       c digo de barras %.', NEW.codigoBarras;
30     END IF;
31
32     -- Actualizar el stock
33     UPDATE ARTICULO
34     SET stock = nuevo_stock
35     WHERE codigoBarras = NEW.codigoBarras;
36
37     -- Alerta si el nuevo stock es bajo
38     IF nuevo_stock < 3 THEN
39         RAISE NOTICE 'Alerta: El stock del art culo % es menor que 3 (
       stock actual: %).', NEW.codigoBarras, nuevo_stock;
40     END IF;
41
42     -- Calcular el monto por art culo
43     NEW.montoPorArt := NEW.cantidadPorArt * precio_unitario;
44
45
46
47
48
49
```

```
50
51      -- Actualizar el monto total de la venta
52      UPDATE VENTA
53      SET montoTotal = COALESCE(montoTotal, 0) + NEW.montoPorArt,
54          cantidadTotalArt = COALESCE(cantidadTotalArt, 0) + NEW.
    cantidadPorArt
55      WHERE folio = NEW.folio;
56
57      RETURN NEW;
58  END;
59  $$ LANGUAGE plpgsql;
60
61
62  CREATE TRIGGER trg_procesar_venta
63  BEFORE INSERT ON VENTA_DETALLE
64  FOR EACH ROW
65  EXECUTE FUNCTION actualizar_art();
66
67  ---OBJETIVO 1---
68  --Cada que se agregue un art culo a una venta, debe actualizarse los
    totales
69  ----(por art culo, venta y cantidad de art culos), as  como validar
    que el articulo est  disponible.
```

Listing 3.2: Triggers

3.1.1. OBJETIVO 1:

Trigger actualizar_art + trg_procesar_venta

Este trigger se ejecuta antes de insertar algún registro en nuestra tabla venta_detalle, está diseñada que de manera automática control el proceso que ocurre al realizar una venta. Cuando se intenta hacer un insert en esta tabla la función consulta el stock en ese momento y el precio de venta obtenido a partir del código de barras. Se verifica si el articulo existe y si hay suficiente stock para completar la venta.

```
1
2   ---OBJETIVO 2:  INDICE-------
3   ---INDICE-------
4   ---INDICE-------
5   CREATE INDEX idx_venta_fecha ON VENTA (fechaVent);
6
7   SELECT * FROM VENTA WHERE fechaVent BETWEEN '2024-01-01' AND '2024-03-31
    ';
8
9   SELECT * FROM VENTA ORDER BY fechaVent DESC;
10
11  SELECT * FROM VENTA WHERE fechaVent = '2024-04-12';
12  ---INDICE-------
13  ---INDICE-------
```

```
---INDICE-------
```

Listing 3.3: Triggers

3.1.2. OBJETIVO 2: INDICE:

Índice idx_venta_fecha

Este índice que implementamos nos será de mucha ayuda para filtrar o agrupar consultas por fecha, cosas que es muy útil para reportes diarios, semanales, mensuales o anuales que nos permitirán mejorar la eficiencia de nuestra base de datos.

```

----------OBJETIVO 3:  VISTA<3 ----------
----------VISTA<3 ----------
----------VISTA<3 ----------
----------VISTA<3 ----------

CREATE VIEW articulos_no_disponibles AS
SELECT
    codigoBarras,
    nombreArt,
    stock,
    CASE
        WHEN stock = 0 THEN 'No disponible'
        WHEN stock < 3 THEN 'Stock bajo'
        ELSE 'Disponible'
    END AS estado
FROM ARTICULO
WHERE stock < 3;

SELECT * FROM public.articulos_no_disponibles

----------VISTA<3 ----------
----------VISTA<3 ----------
UPDATE articulo
SET stock = 0
WHERE codigoBarras = '1510900000000';
----------VISTA<3 ----------
----------VISTA<3 ----------
```

Listing 3.4: Triggers

3.1.3. OBJETIVO 3: Vista articulos_no_disponibles

La vista fue creada para tener una referencia rápida y clara del estado de los artículos con poco o nulo inventario, mostrando sólo aquellos que tuvieran un stock <3 y con 0 de inventario.

Se creó una vista llamada articulos_no_disponibles que muestra los artículos cuyo stock es menor a 3, clasificándolos según su disponibilidad: si el stock es 0, indica "No disponible"; si es menor a 3 pero mayor a 0, muestra "Stock bajo".

```

--------vista ticket------------
--------vista ticket----
--------vista ticket------------
CREATE OR REPLACE VIEW ticket AS
SELECT
    'MBL-' || LPAD(CAST(v.folio AS TEXT), 3, '0') AS folio_ticket,
    v.fechaVent,
    e1.nombreE || ' ' || e1.apPaE AS vendedor,
    e2.nombreE || ' ' || e2.apPaE AS cajero,
    v.montoTotal,
    COALESCE(f.formaPago, 'NO FACTURADO') AS forma_pago,
    COALESCE(c.nomCli || ' ' || c.apPaCli, 'PUBLICO GENERAL') AS cliente
FROM VENTA v
JOIN EMPLEADO e1 ON v.numEmpleadoVendedor = e1.numEmpleado
JOIN EMPLEADO e2 ON v.numEmpleadoCajero = e2.numEmpleado
LEFT JOIN FACTURA f ON v.folio = f.folio
LEFT JOIN CLIENTE c ON f.rfcCliente = c.rfcCliente;

SELECT * FROM ticket;

SELECT *
FROM ticket
WHERE folio_ticket = 'MBL-001';

--------vista ticket------------
--------vista ticket------------
--------vista ticket------------
```

Listing 3.5: Triggers

3.1.4. OBJETIVO 4: Vista ticket

En este ticket implementado como una vista nos da la información mas relevante de una venta, con el fin de generarlo como comprobante además le damos un formato al folio para que nos lo de como `MBL-XXX`, el vendedor, cajero, el monto total como fue su pago y el nombre del cliente.

```

SELECT *
FROM FACTURA f
LEFT JOIN CLIENTE c ON f.rfcCliente = c.rfcCliente
WHERE f.folio = 1;  -- Usa un folio real de los que est s consultando

DELETE FROM VENTA
WHERE folio = 101;

```

```
-----TRIGGER VALIDAR EMPLEADO------
-----TRIGGER VALIDAR EMPLEADO------
-----TRIGGER VALIDAR EMPLEADO------


CREATE OR REPLACE FUNCTION validar_empleados_misma_sucursal()
RETURNS TRIGGER AS $$
DECLARE
    sucursal_vendedor INTEGER;
    sucursal_cajero INTEGER;
BEGIN
    -- Validar que el vendedor y el cajero no sean la misma persona
    IF NEW.numEmpleadoVendedor = NEW.numEmpleadoCajero THEN
        RAISE EXCEPTION 'El vendedor y el cajero no pueden ser la misma persona.';
    END IF;

    -- Obtener la sucursal del vendedor
    SELECT idSucursal INTO sucursal_vendedor
    FROM EMPLEADO
    WHERE numEmpleado = NEW.numEmpleadoVendedor;

    -- Obtener la sucursal del cajero
    SELECT idSucursal INTO sucursal_cajero
    FROM EMPLEADO
    WHERE numEmpleado = NEW.numEmpleadoCajero;

    -- Verificar existencia de empleados
    IF sucursal_vendedor IS NULL OR sucursal_cajero IS NULL THEN
        RAISE EXCEPTION 'Uno o ambos empleados no existen.';
    END IF;

    -- Verificar que ambos est n en la misma sucursal
    IF sucursal_vendedor <> sucursal_cajero THEN
        RAISE EXCEPTION 'El vendedor y el cajero deben pertenecer a la misma sucursal.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_validar_empleados_sucursal ON VENTA;

CREATE TRIGGER trg_validar_empleados_sucursal
BEFORE INSERT ON VENTA
FOR EACH ROW
EXECUTE FUNCTION validar_empleados_misma_sucursal();

-----TRIGGER VALIDAR EMPLEADO------
-----TRIGGER VALIDAR EMPLEADO------
-----TRIGGER VALIDAR EMPLEADO------
```

Listing 3.6: Triggers

3.1.5. OBJETIVO 5: Trigger validar_empleados_misma_sucursal

Con esta función nos va a dar una validación automática de los datos a insertar en la tabla de venta para verificar que los dos empleados (vendedor y cajero) sean distintos y trabajen en la misma sucursal. Si al realizar esta verificación nos dice que el vendedor y el cajero son iguales o no son de la misma sucursal nos va a arrojar un error y no permite ingresar datos a la tabla. Con el trigger que se crea se ejecuta una función en automático cada vez que se quiera registrar una nueva venta.

```
1
2
3  ------JERARQUIA--------
4  ------JERARQUIA--------
5  ------JERARQUIA--------
6
7  CREATE OR REPLACE FUNCTION obtener_jerarquia(p_nombre_empleado VARCHAR)
8  RETURNS TABLE (
9      nombre_empleado VARCHAR,
10     puesto_empleado VARCHAR,
11     nombre_supervisor VARCHAR,
12     puesto_supervisor VARCHAR,
13     sucursal_id VARCHAR,
14     direccion_sucursal VARCHAR
15 ) AS $$
16 BEGIN
17     RETURN QUERY
18     SELECT
19         (e.nombreE || ' ' || e.apPaE || COALESCE(' ' || e.apMaE, ''))::
       VARCHAR AS nombre_empleado,
20         e.tipoEmp::VARCHAR AS puesto_empleado,
21         (s.nombreS || ' ' || s.apPaS || COALESCE(' ' || s.apMaS, ''))::
       VARCHAR AS nombre_supervisor,
22         'supervisor'::VARCHAR AS puesto_supervisor,
23         su.cdS::VARCHAR AS sucursal_id,
24         (su.calleS || ', ' || su.coloniaS || ', ' || su.cdS)::VARCHAR AS
       direccion_sucursal
25     FROM EMPLEADO e
26     LEFT JOIN SUPERVISOR s ON e.idSupervisor = s.idSupervisor
27     INNER JOIN SUCURSAL su ON e.idSucursal = su.idSucursal
28     WHERE e.nombreE = p_nombre_empleado;
29 END;
30 $$ LANGUAGE plpgsql;
31
32 ------JERARQUIA--------
33 ------JERARQUIA--------
34 ------JERARQUIA--------
35
36 SELECT * FROM obtener_jerarquia('Camila');
37
38 ------JERARQUIA--------
39 ------JERARQUIA--------
```

```
40 ------JERARQUIA--------
```

Listing 3.7: Triggers

3.1.6. OBJETIVO 6: JERARQUIA

Esta función nos permite hacer una consulta de la estructura jerárquica de un empleado (definido por nosotros por medio de su nombre). Cuando se ejecuta esta función nos va a dar una tabla con los datos sobre este empleado como lo son su nombre, el puesto que tiene, el nombre del supervisor y un identificador de la sucursal en la que trabaja. Se realiza una unión entre las tablas supervisor y sucursal para obtener esta información, una función como esta ayuda identificar de manera más rápida quien supervisa a quien y en que sucursal trabaja esa persona.

3.2. Aplicación:

Descripción de lo que representan cada una de las gráficas que forman parte de su dashboard

3.2.1. Dashboard 1

En este dashboard podemos observar un breve reporte financiero de las ganancias anuales de la empresa con el número de artículos vendidos, precio de compra y venta junto con el artículo más vendido y un mapa de las tres sucursales en México.

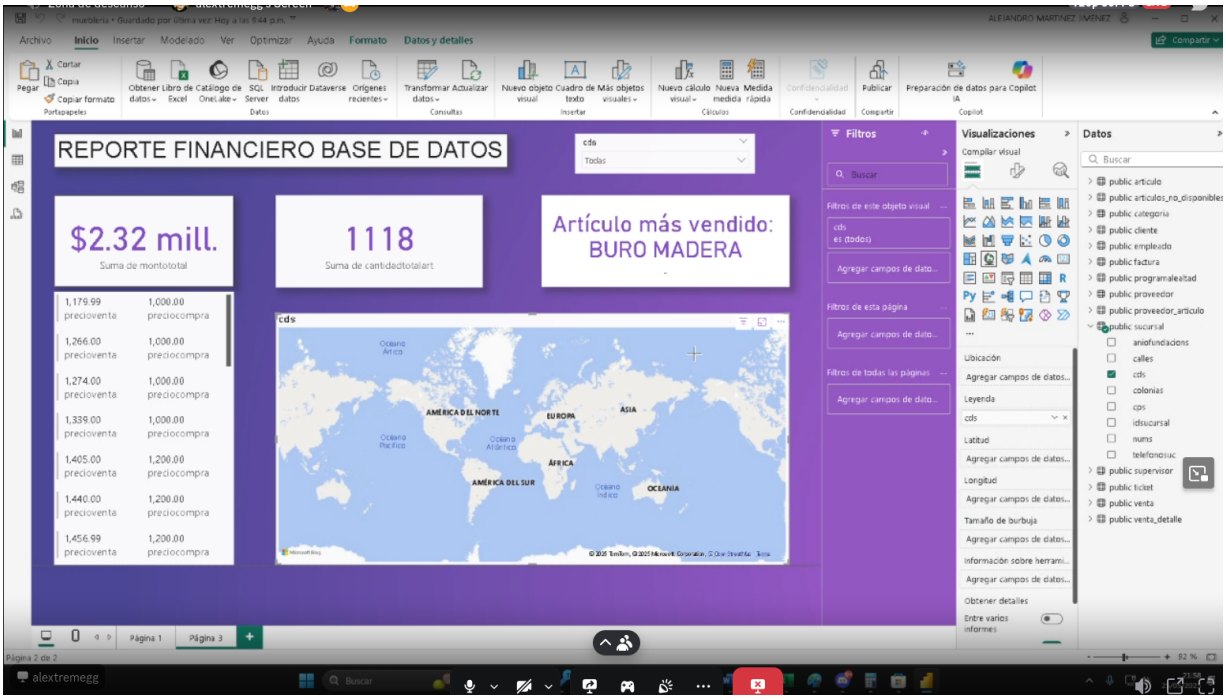


Figura 3.1: Dashboard1.png

3.2.2. Dashboard 2

En este segundo dashboard podemos observar un histograma de la suma del monto total por Año, Trimestre, Mes y Día.

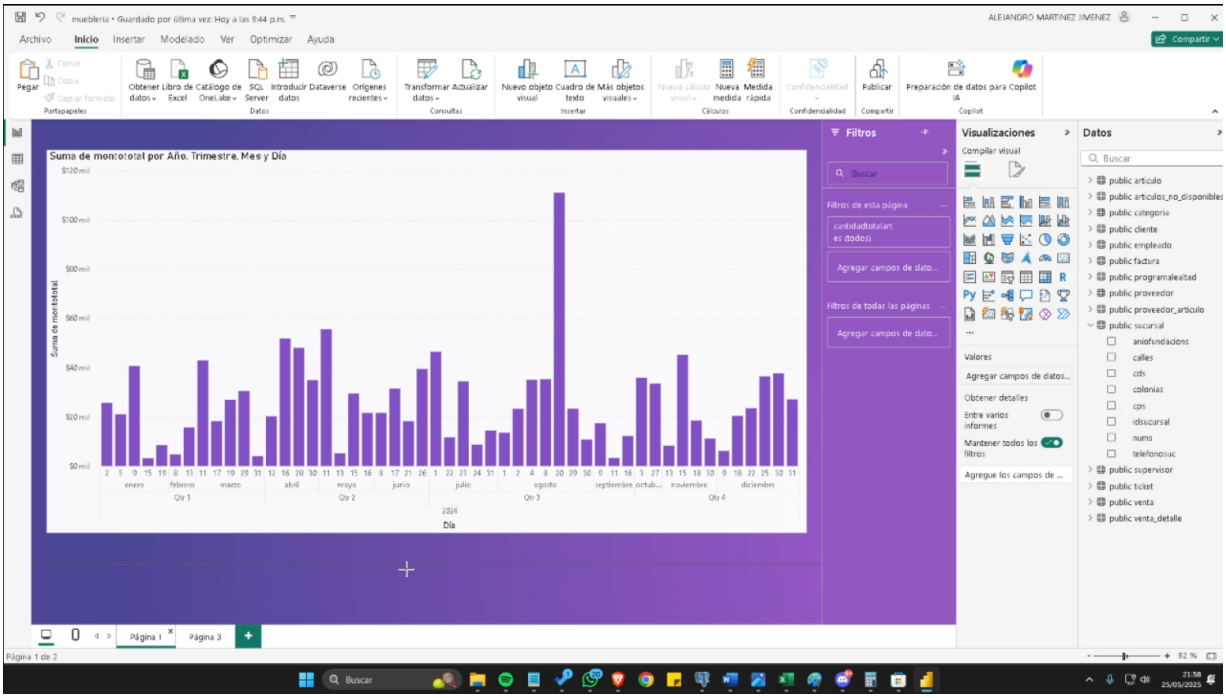


Figura 3.2: Dashboard2.png

3.2.3. Dashboard 3

Por último, este tercer dashboard es un breve croquis de la Ciudad de México con las 20 primeras mueblerías mostradas por google maps obtenida mediante un *scraping web*[1] y procesado en R-Studio.

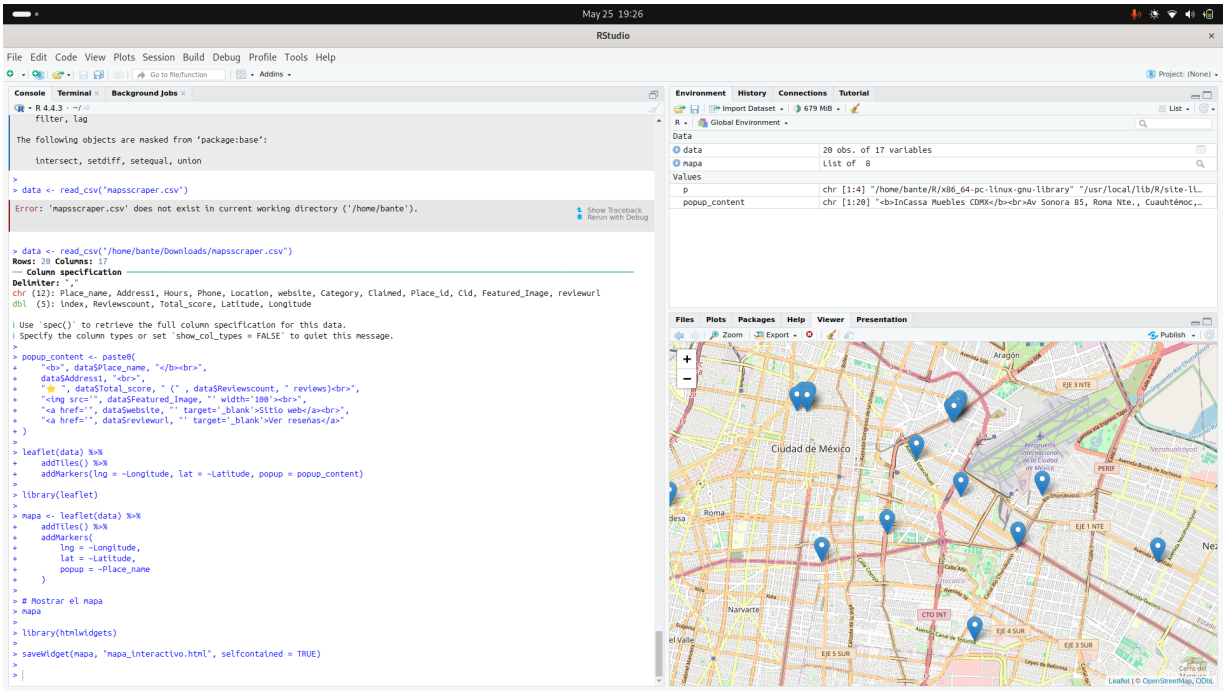


Figura 3.3: Mueblerías CDMX.png

[1]El "web scraping.es una técnica para extraer datos de sitios web de forma automatizada. Un programa o script simula la navegación de un usuario, identifica y extrae información específica del código HTML de la página, y la guarda en un formato estructurado.

Capítulo 4

Conclusiones

4.1. Conclusiones:

Personales, detallando las dificultades, retos, aciertos, etc. que se presentaron en el proyecto.

Arellanes Conde Esteban:

Este proyecto me brindó la oportunidad de desarrollar una base de datos desde sus cimientos, comenzando con el diseño conceptual a través del Modelo Entidad-Relación (MER) y culminando en su implementación física en PostgreSQL. Durante el proceso, llevamos a cabo una transformación cuidadosa del modelo conceptual a un esquema relacional normalizado, aplicando claves primarias, foráneas y restricciones para garantizar la integridad de los datos. Implementamos funciones, triggers e índices utilizando Postgres 12 con PL/pgSQL en pgAdmin y a la par probando en el servidor compartido de la FI, lo que permitió automatizar procesos, validar reglas de negocio y optimizar el desempeño del sistema. También diseñamos vistas que facilitaron la consulta de información relevante, particularmente útil en contextos de análisis de datos con herramientas como Power BI. En conjunto, este trabajo nos permitió integrar teoría y práctica, afianzar nuestras competencias en modelado y programación de bases de datos, y desarrollar una solución sólida, eficiente y adaptable a entornos reales de gestión de información.

Martínez Jiménez Alejandro:

Para este proyecto diseñamos e implementamos una base de datos acerca de una mueblería, a mi parecer se cumplieron en buenas condiciones todos los objetivos planteados como lo son el diseño de un modelo entidad relación (MER) un modelo relacionar (MR) y el respectivo diseño en PostgreSQL. El uso de Pgadmin nos facilitó muchas cosas como la inserción de datos, la eliminación/creación de tablas para hacer pruebas y la visualización de estas para poder tener una mayor claridad con los datos que insertamos. Lo aprendido en clases tanto de teoría como de laboratorio se aplicaron de manera satisfactoria para el

desarrollo de este proyecto y además obtuvimos nuevos conocimiento y habilidades como la familiarización de nuevas herramientas, en este caso power BI para el dashboard de la mueblería.

Méndez Galicia Axel Gael:

La implementación de este proyecto nos permitió desarrollar desde cero una base de datos, iniciando con el diseño conceptual mediante el Modelo Entidad-Relación (MER) y continuando hacia una implementación física en PostgreSQL. A lo largo de este proyecto, fuimos transformando el modelo en un esquema relacional bien estructurado, asegurando la integridad referencial a través de claves primarias, claves foráneas y diversas restricciones. Por consiguiente, implementamos una lógica de negocio utilizando varias funciones, triggers e índices que fueron desarrollados en PL/pgSQL, esto nos permitió automatizar validaciones, mantener consistencia de datos y mejorar el rendimiento de nuestra base. Además, creamos algunas vistas que facilitaron el acceso a información importante, especialmente útil para herramientas de visualización como Power BI. Esta manera de implementar el proyecto nos brindó una experiencia completa, tanto teórica como práctica, así pudimos fortalecer nuestras habilidades en diseño y programación dentro de un entorno más enfocado al mundo laboral. Como resultado, pudimos lograr implementar de manera correcta una base de datos robusta, eficiente y preparada para integrarse con interfaces de más alto nivel.

Sánchez Martínez Ximena:

Este proyecto nos mostró una gran oportunidad para aplicar y reforzar los conocimientos adquiridos en el diseño, implementación y gestión de bases de datos, tomando como caso de estudio una mueblería. A lo largo del desarrollo, logramos cumplir con los objetivos planteados, desde la creación del Modelo Entidad-Relación (MER) y el Modelo Relacional (MR) hasta su implementación práctica en PostgreSQL, utilizando herramientas como PgAdmin para agilizar procesos como la manipulación de datos y el manejo de las estructuras. Pudimos reforzar tanto conceptos teóricos como prácticos que adquirimos en el curso, así como también nos permitió adquirir nuevas habilidades, como el uso de Power BI, lo cual para mí fue la primera vez que lo utilizaba. Power BI nos ayudó para la visualización de datos, lo que enriqueció nuestra perspectiva sobre cómo integrar bases de datos con herramientas analíticas. Además, destacamos la importancia del trabajo colaborativo, ya que la división de tareas, la discusión de ideas y la resolución conjunta de problemas fueron clave para lograr una solución robusta, funcional y accesible. Por último, el proyecto no solo cumplió con los requerimientos técnicos, sino que también nos demostró la importancia de las bases de datos en la organización eficiente de la información, la seguridad y la escalabilidad de los sistemas. El resultado final de nuestra implementación fue una base de datos bien estructurada, con integridad referencial, automatización de procesos mediante triggers y funciones, y capacidad de integración con interfaces externas.

Soriano Barrera María Elena:

Con la realización de esta práctica podemos tener una organización más eficiente de la información, garantizando la seguridad y la rapidez de las consultas. En este proyecto el desarrollo de una base de datos de una mueblería implicó analizar de manera profunda para ver las información requerida enfocándonos en distintas entidades para aprender y comprender la importancia de cada individuo dentro de un sistema. Utilizando Postgres y aplicando conocimientos durante el curso pudimos desarrollar de manera optima todos los requerimientos dados por el profesor. Implementamos materiales y herramientas gráficas y conceptual para el modelo entidad relación, modelo relacional y un diccionario de datos.

Capítulo 5

Referencias y Bibliografía

A continuación algunas de las siguientes referencias utilizadas respecto a los sistemas de bases de datos utilizados en este documento: [PostgreSQL Global Development Group], [1], [2], [3], [4], [5], [6].

Bibliografía

[1] Beaulieu, A. (2020). *Learning SQL: Generate, Manipulate, and Retrieve Data*. O'Reilly, 3rd edition.

[2] Casteel, J. (2015). *Oracle 12c: SQL*. Cengage Learning.

[3] Elmasri, R. and Navathe, S. B. (2017). *Fundamentals of Database Systems (Global Edition)*. Pearson.

[4] IEEE (1984). Guide to software requirements specifications, ieee std 830-1984. `https://chumpolm.files.wordpress.com/2018/09/ieee-std-830-1984.pdf`. Accessed: 2025-05-24.

[5] Kriegel, A. and Trukhnov, B. (2008). *SQL Bible*. Wiley, 2nd edition.

[6] Martínez, A. D. M., Piattini, M., and Esperanza, M. (2000). *Diseño de bases de datos relacionales*. Alfaomega, México.

[PostgreSQL Global Development Group] PostgreSQL Global Development Group. Postgresql documentation. `https://www.postgresql.org/docs/`. Accessed: 2025-05-24.

Notación

Este capítulo resume la notación matemática más común en LaTeX, organizada por categorías.

5.1. Letras griegas y hebreas

α `\alpha` β `\beta` γ `\gamma` δ `\delta`
ϵ `\epsilon` ε `\varepsilon` ζ `\zeta` η `\eta`
θ `\theta` ϑ `\vartheta` ι `\iota` κ `\kappa`
λ `\lambda` μ `\mu` ν `\nu` ξ `\xi`
π `\pi` ρ `\rho` σ `\sigma` τ `\tau`
υ `\upsilon` ϕ `\phi` φ `\varphi` χ `\chi`
ψ `\psi` ω `\omega` Γ `\Gamma` Δ `\Delta`
Θ `\Theta` Λ `\Lambda` Ξ `\Xi` Π `\Pi`
Σ `\Sigma` Υ `\Upsilon` Φ `\Phi` Ψ `\Psi`
Ω `\Omega`

5.2. Constructos matemáticos

- Fracción: $\frac{a+b}{c}$ `\frac{a+b}{c}`
- Raíz cuadrada: $\sqrt{x}$ `\sqrt{x}`, raíz n-ésima: $\sqrt[n]{x}$ `\sqrt[n]{x}`
- Envolventes: $\overline{abc}$ `\overline{abc}`, $\underline{abc}$ `\underline{abc}`
- Derivada: f' `f'`
- Agrupadores: $\overbrace{abc}^{texto}$, $\underbrace{abc}_{texto}$

5.3. Delimitadores y agrupadores

- Llaves: `\left\{ ... \right\}`
- Paréntesis automáticos: `\left( ... \right)`
- Absoluto: `\left| ... \right|`
- Normas: `\left\| ... \right\|`

5.4. Operadores grandes

$\sum$ `\sum` $\prod$ `\prod` $\int$ `\int` $\coprod$ `\coprod`
$\bigcup$ `\bigcup` $\bigcap$ `\bigcap` $\bigsqcup$ `\bigsqcup`
$\bigvee$ `\bigvee` $\bigwedge$ `\bigwedge`

5.5. Funciones estándar

`\sin`, `\cos`, `\tan`, `\log`, `\ln`, `\exp`, `\lim`, `\min`, `\max`, `\gcd`, etc.
Ejemplo: $\cos(x)$ `\cos(x)`

5.6. Relaciones y operadores binarios

Ejemplos:

$=$ `=` $\neq$ `\neq` $\approx$ `\approx` $\equiv$ `\equiv`
$\leq$ `\leq` $\geq$ `\geq` $\subset$ `\subset` $\supseteq$ `\supseteq`
$\in$ `\in` $\notin$ `\notin` $\cup$ `\cup` $\cap$ `\cap`

5.7. Flechas y mapas

$\rightarrow$ `\rightarrow` $\Leftarrow$ `\Leftarrow` $\mapsto$ `\mapsto`
$\longrightarrow$ `\longrightarrow` $\leftrightarrow$ `\leftrightarrow`
$\uparrow$ `\uparrow` $\downarrow$ `\downarrow` $\Updownarrow$ `\Updownarrow`

5.8. Símbolos misceláneos

∞ `\infty` $\forall$ `\forall` $\exists$ `\exists` ∇ `\nabla`
$\emptyset$ `\emptyset` ∂ `\partial` $\angle$ `\angle`

5.9. Estilos de letra matemática

- $\mathcal{A}$ `\mathcal{A}`
- $\mathbb{R}$ `\mathbb{R}`
- $\mathfrak{g}$ `\mathfrak{g}`
- X `\mathsf{X}`
- $\mathbf{v}$ `\mathbf{v}`

5.10. Tamaños de fuente en modo matemático

- `\displaystyle` para ecuaciones en display: $\displaystyle\int x\,dx$
- `\textstyle` (uso dentro de texto): $\textstyle\int x\,dx$
- `\scriptstyle` y `\scriptscriptstyle` para subíndices