



PROYECTO FINAL DE LA ASIGNATURA
DE
BASES DE DATOS

06 DE ENERO DE 2022

Integrantes:

*Carranza Ochoa José David,
Casique Corona Luis Enrique,
Sanchez de Santiago Julián,
Salgado Miranda Jorge,
Zárate García Zuriel*

Resumen

En el presente documento se ha de introducir al lector al proceso de diseño de un sistema de base de datos implementado en una mueblería, con la finalidad de realizar un control sobre los productos vendidos, empleados, clientes, proveedores, sucursales y transacciones realizadas por dicho establecimiento. Así mismo se buscará explicar a detalle, las distintas etapas de las que ha constado la creación del producto final, dando a conocer las partes de mayor relevancia y los retos que hemos afrontado hasta concretar el proyecto.

Índice

| | |
|--|-----------|
| 1. Introducción | 3 |
| 2. Plan de trabajo | 4 |
| 3. Diseño | 5 |
| 3.1. Modelo entidad - relación | 5 |
| 3.2. Modelo lógico | 9 |
| 3.3. Normalización | 12 |
| 3.4. Modelo físico | 13 |
| 4. Implementación | 20 |
| 4.1. Creación base de datos | 20 |
| 5. Aplicación Móvil | 27 |
| 6. Conclusiones | 48 |
| 7. Referencias | 51 |

1. Introducción

El fin de este proyecto consiste en el diseño de una base de datos. Una mueblería busca digitalizar su forma de operar para evitar tener sus registros de forma física y tener una vista completa de la información de todas las sucursales, por lo que se plantean ciertos requerimientos.

Sin embargo, es indispensable contar con nociones sólidas realizar un buen sistema informático quien preserve la información; es por ello que este proyecto busca dar una solución óptima a las limitantes que se tengan al momento de guardar la información, considerando como puntos fundamentales la integridad, seguridad y coherencia de los datos.

Así mismo, la solución que se presenta parte desde la recabación de los requerimientos, pasando por el diagrama de modelado principal (Entidad-relación) hasta una representación final expresada en el lenguaje de consultas SQL, donde se consideran los aspectos críticos considerados, mostrando las etapas del diseño de una base de datos para finalmente poder implementarla en una aplicación externa a SQL, la cual es capaz de operar con la base de datos creada para consultar o manipular la información.

Las soluciones expuestas son definidas tras analizar meticulosamente los requisitos solicitados por un cliente, entre ellas se especifican los datos a almacenar así como agentes que intervienen; cada uno de los anteriores fueron mapeados conforme se considera correcto, trabajando con relaciones que interrelacionen a las entidades participantes.

Considerando un caso de uso en entornos donde la rapidez y agilidad para acceder a los datos es fundamental, el entregable consta de una aplicación móvil disponible para dispositivos Android y iOS, esta misma trabaja con los datos almacenados en un servidor de Postgres el cuál se encuentra alojado en la nube desde Heroku.

Durante el desarrollo del presente documento serán mostradas las definiciones de los métodos y explicaciones de cada propuesta de solución individual, mostrando evidencias del funcionamiento de la base de datos como también la conexión desde la interfaz gráfica.

2. Plan de trabajo

Dentro de las diferentes etapas en el desarrollo de la base de datos, cada integrante aportó significativamente al proyecto, desde aportación de ideas hasta la manipulación de los datos; a continuación se muestra un compendio de las actividades realizadas y la participación por integrante.

| Acción | Responsabl | Prioridad | Fecha de inicio | Fecha final |
|---|--|-----------|-----------------|-------------|
| Fase 1: Análisis de requerimientos | | | | |
| Identificar los requerimientos necesarios para la base de datos | Todos | Alta | 04/12/2022 | 05/12/2022 |
| Identificación de entidades y atributos presentes | Todos | Alta | 04/12/2022 | 05/12/2022 |
| Consideración de notas adicionales | Zuriel Zárate | Media | 05/12/2022 | 06/12/2022 |
| Fase 2: Modelo conceptual | | | | |
| Realización individual de MER | Todos | Alta | 06/12/2022 | 10/12/2022 |
| Combinación de propuestas de solución | David Carranza, Julián Sánchez | Media | 10/12/2022 | 12/12/2022 |
| Validación de MER | Jorge Salgado, Luis Casique, Zuriel Zárate | Alta | 13/12/2022 | 13/12/2022 |
| Definición de Software a utilizar e implementación en el mismo | Jorge Salgado | Baja | 13/12/2022 | 13/12/2022 |
| Revalidación de MER | Todos | Alta | 13/12/2022 | 14/12/2022 |
| Fase 3: Modelo lógico V1 | | | | |
| Transformación de entidades a relaciones | Julian Sánchez | Alta | 15/12/2022 | 16/12/2022 |
| Corrección de representación intermedia | David Carranza, Julian Sánchez | Alta | 17/12/2022 | 17/12/2022 |
| Fase 4: Normalización | | | | |
| Aplicación de 1FN, 2FN, 3FN a los datos intermedios | David Carranza | Alta | 18/12/2022 | 20/12/2022 |
| Validación de la normalización | Luis Casique | Media | 23/12/2022 | 23/12/2022 |
| Fase 5: Representación modelo lógico V2 | | | | |
| Definición de Software a utilizar para mapeo final | Jorge Salgado | Media | 23/12/2022 | 23/12/2022 |
| Validación de mapeo en la herramienta seleccionada | Todos | Alta | 24/12/2022 | 24/12/2022 |
| Fase 6: Modelo físico | | | | |
| Creación de tablas e inserción de datos | Jorge Salgado | Alta | 25/12/2022 | 29/12/2022 |
| Creación de triggers que no efectúan la venta | David Carranza, Zuriel Zárate, Luis Enrique, Julián Sánchez | Alta | 25/12/2022 | 05/01/2023 |
| Creación de triggers de procesamiento de venta | David Carranza, Zuriel Zárate, Luis Enrique, Julián Sánchez | Alta | 25/12/2022 | 05/01/2023 |
| Creación de vistas, índices y funciones variadas | Zuriel Zárate, Julián Sánchez | Alta | 28/12/2022 | 04/01/2023 |
| Validación | Todos | Alta | 05/01/2023 | 06/01/2023 |
| Fase 7: Manipulación de la aplicación | | | | |
| Elección del tipo de interfaz a trabajar | Todos | Baja | 04/12/2022 | 04/12/2022 |
| Realización de la interfaz gráfica | Jorge Salgado | Media | 22/12/2022 | 05/01/2023 |

3. Diseño

El proceso de diseño de nuestra base de datos constó de distintas etapas las cuales nos han permitido realizar un análisis meticuloso, con al finalidad de eficientizar el rendimiento de nuestro sistema, así como satisfacer todos los requerimientos establecidos por el cliente. Las etapas de nuestro proceso de diseño han sido:

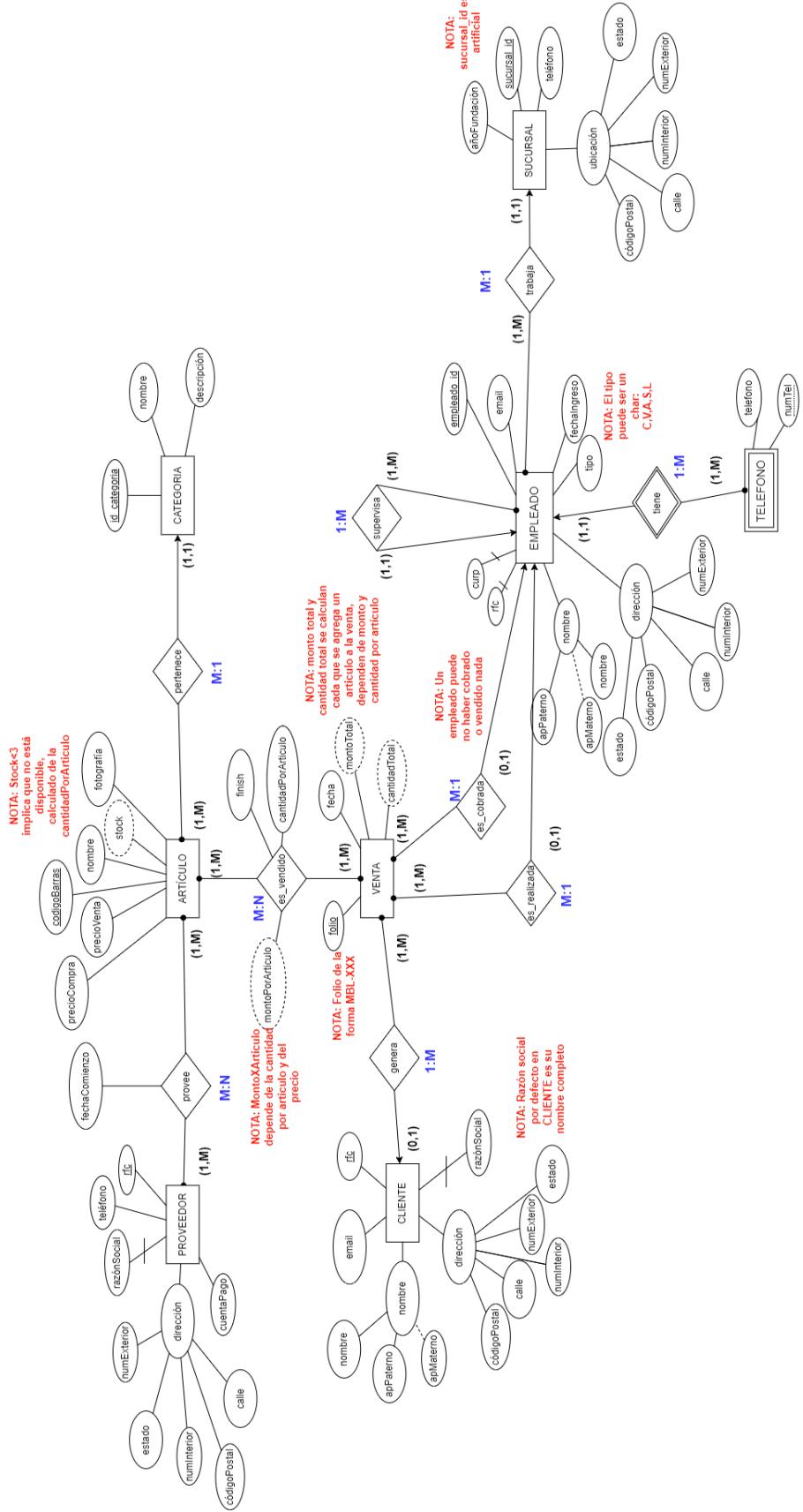
- Análisis de requerimientos
- Modelo conceptual (Modelo entidad - relación)
- Diseño de modelo lógico
- Normalización de los datos (1FN, 2FN, 3FN)
- Diseño físico en SQL
- Exportación a una aplicación para manipular la información

Donde en cada etapa se realizaron pruebas de validación y verificación con los requerimientos solicitados

3.1. Modelo entidad - relación

En esta etapa del proceso de diseño hemos creado y relacionado las distintas entidades que conformarán nuestra base de datos, así como los atributos propios de cada una y las relaciones que las vinculan entre ellas. El diagrama del cual partimos para nuestro modelo intermedio, se muestra más adelante.

Algo que destaca en nuestro MER es la aparición de una relación de dependencia para los teléfonos del empleado, esta es generada con el fin de ahorrar un paso en la normalización (atributos multivaluados) para posteriormente insertarla directamente como una relación con EMPLEADO. Así mismo, en cuanto a la venta se refiere, quienes cobran y realizan la venta son empleados, los cuales hemos expresado en una relación doble que posteriormente definiremos.



En el MER anterior se logra observar la recopilación de requerimientos solicitados por el cliente. Nuestro cliente es una tienda de muebles que solicita tener toda la información de cada una de las sucursales, esta sera una de nuestras primeras entidades (SUCURSAL). Para cada SUCURSAL se debe conocer la dirección donde se encuentra ubicada, año de fundación y el teléfono para contactar a dicha sucursal. Definitivamente la dirección, al poseer tantos atributos, no es considerada una buena llave primaria. Así que podríamos utilizar el teléfono ya que solo existe uno, pero como este puede cambiar en cualquier momento entonces se decidió crear una llave artificial (idSucursal). En cada SUCURSAL debe existir personal, esta es una nueva entidad (EMPLEADO). En una sucursal existen muchos EMPLEADOS y es necesario conocer el *tipo* de actividad que desempeñan en la sucursal pues estos pueden ser vendedores, supervisores, cajeros, etc. También es necesario conocer la *fechaIngreso* a la empresa, el *nombre* y *dirección* completas del trabajador, aunque el apellido materno es un valor opcional. Adicionalmente se tiene un *email* de contacto, el *curp* y el *rfc*. Como estos 2 últimos atributos son únicos para cada persona también podrían ser llaves primarias sino tuviéramos un *idEmpleado*, así que *curp* y *rfc* se consideran llaves candidatas. Un EMPLEADO debe tener un supervisor que al final también es considerado un empleado por lo que tendrá los mismos atributos antes mencionados así que EMPLEADO se volverá una relación recursiva.

Un EMPLEADO se encarga de realizar una venta mientras que otro EMPLEADO se encarga de finalizar dicha venta cobrando. Para las VENTAS se debe asignar un *folio* que permita ubicar los movimientos y una *fecha* para saber cuando se realizó la venta. Por obvias razones se debe almacenar también la *cantidadTotal* de artículos vendidos así como el *montoTotal* que será calculado dependiendo de los artículos seleccionados.

Para cada ARTICULO se tiene un *codigoBarra* que permite ubicar cada producto, el *nombre* del artículo, una *fotografia*, el *stock*, el *precioCompra* y el *precioVenta*. El atributo *stock* se calcula a partir del número de productos seleccionados en una VENTA.

Cada ARTICULO debe pertenecer a alguna categoría para tener un mejor resumen de los productos. Una CATEGORIA tendrá un *idCategoria*, un *nombre* asociado y la *descripcion* de las características principales de los ARTICULOS pertenecientes a dicha CATEGORIA. Los ARTICULOS son comprados a diversos PROVEEDORES de los que se tiene que conocer el *rfc* que permitirá ubicar a cada proveedor, así como la *dirección* la *cuentaPago* donde se realizaron los

depósitos y la *razonSocial* que tambien podria ser una llave primaria pues es una forma de identificarse ante bancos y entidades, pero ya que puede ser muy larga es preferible utilizar el *rfc*. Un CLIENTE genera puede generar varias compras y los atributos de esta entidad son muy similares a los de un EMPLEADO. Se tiene como llave primaria *rfc* ya que al ser unico para cada persona es un perfecto identificador; un *telefono*, el *nombre* completo, su *direccion* y la *razonSocial* aunque este atributo tiene un valor por default que es simplemente el nombre completo sino se ingresa algun otro valor y este atributo tambien es considerada buena llave aunque, nuevamente por la longitud, simplemente quedara como llave candidata ya que se tiene el *rfc*.

3.2. Modelo lógico

Posterior a la obtención del modelo Entidad Relación, obtuvimos el modelo intermedio, el cual nos permitirá tener una primer iteración del tipo de datos los cuales deberíamos de asignar a cada atributo de nuestras tablas, con la finalidad de obtener el rango y características óptimas para el manejo de cada uno de nuestros valores. El modelo se muestra a continuación en la siguiente tabla.

| Datos Modelo Intermedio | | | |
|-------------------------|------------------|--------------|-------------|
| TABLA | ATRIBUTOS | TIPO DE DATO | RESTRICCIÓN |
| ARTICULO | codigo barras | BIGINT | PK |
| | id categoria | INTEGER | FK |
| | precio venta | NUMERIC(8,2) | - |
| | precio compra | NUMERIC(8,2) | - |
| | nombre | VARCHAR[50] | - |
| | stock | SMALLINT | C |
| | fotografía | TEXT | - |
| CATEGORIA | id categoria | INTEGER | PK |
| | nombre | VARCHAR[50] | - |
| | descripcion | TEXT | - |
| CLIENTE | rfc | CHAR[13] | PK |
| | razon social | VARCHAR[150] | U,N |
| | num exterior | SMALLINT | - |
| | num interior | SMALLINT | - |
| | rfc | VARCHAR[50] | - |
| | estado | VARCHAR[50] | - |
| | codigo postal | BIGINT | - |
| | calle | VARCHAR[50] | - |
| | nombre | VARCHAR[60] | - |
| | apellido paterno | VARCHAR[40] | - |
| | apellido materno | VARCHAR[40] | - |
| | email | VARCHAR[100] | - |

| TABLA | ATRIBUTOS | TIPO DE DATO | RESTRICCIÓN |
|------------|------------------------|--------------|-------------|
| EMPLEADO | <i>id_empleado</i> | INTEGER | PK |
| | curp | CHAR[18] | U |
| | rfc | CHAR[13] | U |
| | nombre | VARCHAR[60] | - |
| | apellido paterno | VARCHAR[40] | - |
| | apellido materno | VARCHAR[40] | - |
| | tipo | CHAR[1] | - |
| | email | VARCHAR[100] | - |
| | numero exterior | SMALLINT | - |
| | numero interior | SMALLINT | - |
| ES VENDIDO | estado | VARCHAR[50] | - |
| | codigo postal | BIGINT | - |
| | calle | VARCHAR[50] | - |
| | empleado id supervisor | INTEGER | FK |
| | <i>id_sucursal</i> | INTEGER | - |
| PROVEEDOR | codigo barras | BIGINT | FK,PK |
| | folio | | FK,PK |
| | NUMERIC(2,8) | N | - |
| | cantidad | SMALLINT | - |
| | finish | BOOLEAN | - |

| TABLA | ATRIBUTOS | TIPO DE DATO | RESTRICCIÓN |
|----------|----------------|--------------|-------------|
| PROVEE | rfc | CHAR[13] | FK,PK |
| | codigo | BIGINT | FK,PK |
| | fecha comienzo | DATE | - |
| SUCURSAL | id sucursal | INTEGER | PK |
| | año fundacion | YEAR (DATE) | - |
| | telefono | BIGINT | - |
| | num exterior | SMALLINT | - |
| | num interior | SMALLINT | - |
| | estado | VARCHAR [50] | - |
| | codigo postal | BIGINT | - |
| TELEFONO | calle | VARCHAR[50] | - |
| | num tel | INTEGER | PK,D |
| VENTA | id empleado | INTEGER | PK,FK |
| | folio | CHAR(7) | PK |
| | rfc | CHAR(13) | FK,N |
| | vendedor | INTEGER | FK |
| | cobrador | INTEGER | FK |
| | fecha | DATE | - |
| | monto total | NUMERIC(8,2) | C,N |
| | cantidad total | SMALLINT | C,N |

3.3. Normalización

El proceso de normalización que hemos realizado, ha tenido como finalidad cumplir con cada una de las condicionales establecidas por las 3 primeras formas normales, de esta manera teniendo en cada tabla una *PRIMARY KEY* que define de manera integra los atributos de cada tupla, así como logramos evitar dependencias funcionales parciales y transitivas. A continuación presentamos los pasos que hemos seguido para realizar la normalización de las tablas donde existe un punto importante a normalizar, omitiendo aquellas donde cumple al instante primera y segunda forma normal (Atributos multivaluados/grupos de repetición y PK compuesta). Las tablas obtenidas tras el proceso de normalización son las siguientes:

1. Proveedor: RFC [PK], Razon social, Telefono, cuenta pago, calle, cp, num Interior, num Exterior
2. Artículo: Codigo Barras [PK], nombre, Precio Venta, Precio Compra, fotografía, stock, id categoria [FK]
3. Provee: (rfc, CodigoBarras) [FK][PK], fechaComienzo
4. Categoria: id categoria, nombre, descripcion
5. Es vendido: (codigoBarras, folio) [FK] [PK], monto, cantidad
6. Venta: Folio [PK], fecha, monto total, cantidad total, empleado id, empleado id2, rfc [FK]
7. Cliente: rfc [PK], email, nombre, apPaterno, apMaterno, calle, cp, numInterior, numExterior
8. Sucursal: sucursal id [PK], año fundacion, telefono, calle, cp, numInterior, numExterior
9. Empleado: empleado id [PK], curp, rfc, tipo, fecha ingreso, email, combre, apPaterno, apMaterno, calle, cp, numInterior, numExterior, sucursal id [FK], empleado id [FK]
10. telefono empleado: num tel [PK], telefono empleado, empleado id [FK]

- En la tabla PROVEE existe una llave compuesta pero cumple por no tener dependencias parciales.
- Tabla ES_VENDIDO tampoco existen dependencias parciales
- Para las tablas SUCURSAL, EMPLEADO, CLIENTE y PROVEEDOR no se cumple con 3FN al tener una transitividad entre el código postal y el estado, por lo que finalmente se separa la transitividad manteniendo como FK el código postal en cada tabla mencionada previamente.

3.4. Modelo físico

Una vez que hemos finalizado nuestro modelo lógico, hemos de elaborar el código mediante el cual tanto implementaremos nuestra base de datos, así como realizaremos las vistas y procedimientos requeridos por el cliente.

De manera inicial crearemos una base de datos denominada “coders mueblería”, en la cual habremos de plasmar las tablas obtenidas posterior al proceso de normalización realizado. A continuación se muestra el código empleado para crear la base de datos con sus respectivas tablas las cuales incluyen las restricciones pertinentes.

```
-- object: new_database | type: DATABASE --
-- DROP DATABASE IF EXISTS new_database;
CREATE DATABASE codders_muebleria;
-- ddl-end --
\c codders_muebleria

-- object: public."ARTICULO" | type: TABLE --
-- DROP TABLE IF EXISTS public."ARTICULO" CASCADE;
CREATE TABLE public.ARTICULO (
    codigo_barras bigint NOT NULL,
    nombre varchar(50) NOT NULL,
    precio_compra numeric(8,2) NOT NULL,
    precio_venta numeric(8,2) NOT NULL,
    stock smallint NOT NULL,
    fotografia text NOT NULL,
    "id_categoria" integer NOT NULL,
    CONSTRAINT "ARTICULO_pk" PRIMARY KEY (codigo_barras)
);

-- object: public."CATEGORIA" | type: TABLE --
-- DROP TABLE IF EXISTS public."CATEGORIA" CASCADE;
CREATE TABLE public.CATEGORIA (
    id_categoria integer NOT NULL,
    nombre varchar(50) NOT NULL,
    descripcion text NOT NULL,
    CONSTRAINT "CATEGORIA_pk" PRIMARY KEY (id_categoria)
);
```

Creacion base de datos y tabla articulo y categoria

```

-- object: public."PROVEEDOR" | type: TABLE --
-- DROP TABLE IF EXISTS public."PROVEEDOR" CASCADE;
CREATE TABLE public.PROVEEDOR (
    rfc char(13) NOT NULL,
    telefono bigint NOT NULL,
    razon_social varchar(50) UNIQUE NOT NULL,
    cuenta_pago bigint NOT NULL,
    numero_exterior smallint NOT NULL,
    estado varchar(40) NOT NULL,
    numero_interior smallint NOT NULL,
    codigo_postal bigint NOT NULL,
    calle varchar(50) NOT NULL,
    CONSTRAINT "PROVEEDOR_pk" PRIMARY KEY (rfc)
);

-- object: public."VENTA" | type: TABLE --
-- DROP TABLE IF EXISTS public."VENTA" CASCADE;
CREATE TABLE public.VENTA (
    folio char(7) NOT NULL,
    fecha timestamp NOT NULL,
    monto_total numeric(8,2) NULL,
    cantidad_total smallint NULL,
    "rfc_CLIENTE" char(13) NULL,
    "id_empleado" integer NOT NULL,
    "id_empleado1" integer NOT NULL,
    CONSTRAINT "VENTA_pk" PRIMARY KEY (folio)
);

-- object: public."CLIENTE" | type: TABLE --
-- DROP TABLE IF EXISTS public."CLIENTE" CASCADE;
CREATE TABLE public.CLIENTE (
    rfc char(13) NOT NULL,
    email varchar(100) NOT NULL,
    nombre varchar(60) NOT NULL,
    apellido_paterno varchar(40) NOT NULL,
    apellido_materno varchar(40),
    codigo_postal bigint NOT NULL,
    calle varchar(50) NOT NULL,
    numero_interior smallint NOT NULL,
    numero_exterior smallint NOT NULL,
    estado varchar(50) NOT NULL,
    razon_social varchar(150) UNIQUE NULL,
    CONSTRAINT "CLIENTE_pk" PRIMARY KEY (rfc)
);

```

Creacion de tabla proveedor, venta y cliente

```

-- object: public."EMPLEADO" | type: TABLE --
-- DROP TABLE IF EXISTS public."EMPLEADO" CASCADE;
CREATE TABLE public.EMPLEADO(
    id_empleado integer NOT NULL,
    curp char(18) UNIQUE NOT NULL,
    rfc char(13) UNIQUE NOT NULL,
    nombre varchar(60) NOT NULL,
    apellido_paterno varchar(40) NOT NULL,
    apellido_materno varchar(40),
    fecha_ingreso date NOT NULL,
    tipo char(1) NOT NULL,
    email varchar(100) NOT NULL,
    numero_exterior smallint NOT NULL,
    numero_interior smallint NOT NULL,
    estado varchar(50) NOT NULL,
    codigo_postal bigint NOT NULL,
    calle varchar(50) NOT NULL,
    "id_sucursal" integer NOT NULL,
    "id_empleado1" integer NOT NULL,
    CONSTRAINT "EMPLEADO_pk" PRIMARY KEY (id_empleado)
);
-- ddl-end --

-- object: public."TELEFONO" | type: TABLE --
-- DROP TABLE IF EXISTS public."TELEFONO" CASCADE;
CREATE TABLE public.TELEFONO (
    num_tel serial NOT NULL,
    telefono_empleado bigint NOT NULL,
    "id_empleado" integer NOT NULL,
    CONSTRAINT "TELEFONO_pk" PRIMARY KEY (num_tel,id_empleado)
);
-- object: "EMPLEADO_fk" | type: CONSTRAINT --
-- ALTER TABLE public."TELEFONO" DROP CONSTRAINT IF EXISTS "EMPLEADO_fk" CASCADE;
ALTER TABLE public.TELEFONO ADD CONSTRAINT "EMPLEADO_fk" FOREIGN KEY ("id_empleado")
REFERENCES public.EMPLEADO (id_empleado) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

```

Creacion de tabla empleado, telefono y constrain para FK en telefono de empleado

```

-- object: public."SUCURSAL" | type: TABLE --
-- DROP TABLE IF EXISTS public."SUCURSAL" CASCADE;
CREATE TABLE public.SUCURSAL (
    id_sucursal integer NOT NULL,
    anio_fundacion date NOT NULL,
    telefono bigint NOT NULL,
    numero_exterior smallint NOT NULL,
    numero_interior smallint NOT NULL,
    estado varchar(50) NOT NULL,
    codigo_postal bigint NOT NULL,
    calle varchar(50) NOT NULL,
    CONSTRAINT "SUCURSAL_pk" PRIMARY KEY (id_sucursal)
);
-- ddl-end --
ALTER TABLE public.SUCURSAL OWNER TO mdthlconjlitvq;
-- ddl-end --

-- object: public."PROVEE" | type: TABLE --
-- DROP TABLE IF EXISTS public."PROVEE" CASCADE;
CREATE TABLE public.PROVEE (
    "rfc" varchar(13) NOT NULL,
    "codigo_barras" bigint NOT NULL,
    fecha_comienzo date NOT NULL,
    CONSTRAINT "PROVEE_pk" PRIMARY KEY ("rfc","codigo_barras")
);
-- ddl-end --

-- object: public."ES_VENDIDO" | type: TABLE --
-- DROP TABLE IF EXISTS public."ES_VENDIDO" CASCADE;
CREATE TABLE public.ES_VENDIDO (
    "folio" char(7) NOT NULL,
    "codigo_barras" bigint NOT NULL,
    monto numeric(8,2) NULL,
    cantidad smallint NOT NULL,
    finish boolean NOT NULL,
    CONSTRAINT "ES_VENDIDO_pk" PRIMARY KEY ("folio","codigo_barras")
);
-- ddl-end --

create table EMPLEADO_ESTADO(
    codigo_postal bigint NOT NULL,
    estado varchar(40) NOT NULL,
    CONSTRAINT "empleado_cp_pk" PRIMARY KEY (codigo_postal)
);

```

Creacion de tabla sucursal, provee, es vendido y empleado estado

```

create table PROVEEDOR_ESTADO(
    codigo_postal bigint NOT NULL,
    estado varchar(40) NOT NULL,
    CONSTRAINT "proveedor_cp_pk" PRIMARY KEY (codigo_postal)
);

create table CLIENTE_ESTADO(
    codigo_postal bigint NOT NULL,
    estado varchar(40) NOT NULL,
    CONSTRAINT "cliente_cp_pk" PRIMARY KEY (codigo_postal)
);

create table SUCURSAL_ESTADO(
    codigo_postal bigint NOT NULL,
    estado varchar(40) NOT NULL,
    CONSTRAINT "sucursal_cp_pk" PRIMARY KEY (codigo_postal)
);

```

Creacion de tabla proveedor estado, cliente estado y sucursal estado

Cómo se podrá apreciar, las últimas tablas que se han creado incluyen la terminación estado, esto es debido a que para satisfacer la tercera forma normal, hemos de crear tablas adicionales las cuales se conformen por el código postal y el estado de residencia de cada uno de los involucrados, esto al existir dependencia del estado con respecto al código postal al que hagamos alusión. A continuación se muestra el segmento de código en el cual se establece la restricción de llave foránea por parte del código postal en cada una de las tablas que tengan dicha terminación.

```

ALTER TABLE public.EMPLEADO ADD CONSTRAINT "empleado_cp_fk" FOREIGN KEY ("codigo_postal")
REFERENCES public.EMPLEADO_ESTADO (codigo_postal) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE public.PROVEEDOR ADD CONSTRAINT "proveedor_cp_fk" FOREIGN KEY ("codigo_postal")
REFERENCES public.PROVEEDOR_ESTADO (codigo_postal) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE public.CLIENTE ADD CONSTRAINT "cliente_cp_fk" FOREIGN KEY ("codigo_postal")
REFERENCES public.CLIENTE_ESTADO (codigo_postal) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE public.SUCURSAL ADD CONSTRAINT "sucursal_cp_fk" FOREIGN KEY ("codigo_postal")
REFERENCES public.SUCURSAL_ESTADO (codigo_postal) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;

```

Generación del restricciones de PK por parte del código postal con respecto a las tablas con terminación 'Estado'

```

CREATE VIEW vis_articulos_cat_prov AS
    select a.nombre,a.precio_venta,a.fotografia,c.nombre categoria,c.descripcion,p.razon_social empresa FROM articulo a
    LEFT JOIN categoria c ON a.id_categoria=c.id_categoria
    LEFT JOIN provee pe ON a.codigo_barras=pe.codigo_barras
    LEFT JOIN proveedor p ON p.rfc=pe.rfc;

-- object: "Verificadores de valores positivos" | type: CONSTRAINT --
ALTER TABLE public.ARTICULO ADD CONSTRAINT "verifica_Stock" CHECK(stock>=0);
ALTER TABLE public.ARTICULO ADD CONSTRAINT "verifica_PreV" CHECK(precio_venta>=0.00);
ALTER TABLE public.ARTICULO ADD CONSTRAINT "verifica_PreC" CHECK(precio_compra>=0.00);
ALTER TABLE public.ES_VENDIDO ADD CONSTRAINT "verifica_MonArt" CHECK(monto>=0.00);
ALTER TABLE public.ES_VENDIDO ADD CONSTRAINT "verifica_CantArt" CHECK(cantidad>=0);
ALTER TABLE public.VENTA ADD CONSTRAINT "verifica_CantTot" CHECK(cantidad_total>=0);
ALTER TABLE public.VENTA ADD CONSTRAINT "verifica_MontTot" CHECK(monto_total>=0.00);
-- ddl-end --

```

Creación de vista artículos y verificación de valores positivos

```

-- object: "PROVEEDOR_fk" | type: CONSTRAINT --
-- ALTER TABLE public."PROVEE" DROP CONSTRAINT IF EXISTS "PROVEEDOR_fk" CASCADE;
ALTER TABLE public.PROVEE ADD CONSTRAINT "PROVEEDOR_fk" FOREIGN KEY ("rfc")
REFERENCES public.PROVEEDOR (rfc) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "ARTICULO_fk" | type: CONSTRAINT --
-- ALTER TABLE public."PROVEE" DROP CONSTRAINT IF EXISTS "ARTICULO_fk" CASCADE;
ALTER TABLE public.PROVEE ADD CONSTRAINT "ARTICULO_fk" FOREIGN KEY ("codigo_barras")
REFERENCES public.ARTICULO (codigo_barras) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "CATEGORIA_fk" | type: CONSTRAINT --
-- ALTER TABLE public."ARTICULO" DROP CONSTRAINT IF EXISTS "CATEGORIA_fk" CASCADE;
ALTER TABLE public.ARTICULO ADD CONSTRAINT "CATEGORIA_fk" FOREIGN KEY ("id_categoria")
REFERENCES public.CATEGORIA (id_categoria) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "VENTA_fk" | type: CONSTRAINT --
-- ALTER TABLE public."ES_VENDIDO" DROP CONSTRAINT IF EXISTS "VENTA_fk" CASCADE;
ALTER TABLE public.ES_VENDIDO ADD CONSTRAINT "VENTA_fk" FOREIGN KEY ("folio")
REFERENCES public.VENTA (folio) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "ARTICULO_fk" | type: CONSTRAINT --
-- ALTER TABLE public."ES_VENDIDO" DROP CONSTRAINT IF EXISTS "ARTICULO_fk" CASCADE;
ALTER TABLE public.ES_VENDIDO ADD CONSTRAINT "ARTICULO_fk" FOREIGN KEY ("codigo_barras")
REFERENCES public.ARTICULO (codigo_barras) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

```

Generación de restricciones de llaves foráneas

Se generarán restricciones de Cascade para las llaves foráneas en las operaciones de eliminación y actualización.

```
-- object: "CLIENTE_fk" | type: CONSTRAINT --
-- ALTER TABLE public."VENTA" DROP CONSTRAINT IF EXISTS "CLIENTE_fk" CASCADE;
ALTER TABLE public.venta ADD CONSTRAINT "CLIENTE_fk" FOREIGN KEY ("rfc_CLIENTE")
REFERENCES public.CLIENTE (rfc) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "EMPLEADO_fk" | type: CONSTRAINT --
-- ALTER TABLE public."VENTA" DROP CONSTRAINT IF EXISTS "EMPLEADO_fk" CASCADE;
ALTER TABLE public.venta ADD CONSTRAINT "EMPLEADO_fk" FOREIGN KEY ("id_empleado")
REFERENCES public.EMPLEADO (id_empleado) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "EMPLEADO_fk1" | type: CONSTRAINT --
-- ALTER TABLE public."VENTA" DROP CONSTRAINT IF EXISTS "EMPLEADO_fk1" CASCADE;
ALTER TABLE public.venta ADD CONSTRAINT "EMPLEADO_fk1" FOREIGN KEY ("id_empleado1")
REFERENCES public.EMPLEADO (id_empleado) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "SUCURSAL_fk" | type: CONSTRAINT --
-- ALTER TABLE public."EMPLEADO" DROP CONSTRAINT IF EXISTS "SUCURSAL_fk" CASCADE;
ALTER TABLE public.EMPLEADO ADD CONSTRAINT "SUCURSAL_fk" FOREIGN KEY ("id_sucursal")
REFERENCES public.SUCURSAL (id_sucursal) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --

-- object: "EMPLEADO_fk" | type: CONSTRAINT --
-- ALTER TABLE public."EMPLEADO" DROP CONSTRAINT IF EXISTS "EMPLEADO_fk" CASCADE;
ALTER TABLE public.EMPLEADO ADD CONSTRAINT "EMPLEADO_fk" FOREIGN KEY ("id_empleado")
REFERENCES public.EMPLEADO (id_empleado) MATCH FULL
ON DELETE RESTRICT ON UPDATE CASCADE;
-- ddl-end --
```

Generación de restricciones de llaves foráneas

4. Implementación

4.1. Creación base de datos

La base de datos se conformará de un conjunto de tablas, las cuales se han generado de manera previa en el modelo relacional y tras el proceso de normalización. A continuación se muestran las tablas que la constituyen:



Graphics/IM10.png

Generación de restricciones de llaves foráneas

4.1.1. eliminacion2

Este trigger tiene la función eliminar de la tabla artículo los productos que tengan un stock ≤ 3 , dichos artículos serán añadidos a la tabla `low_stock`, en dicha tabla podremos actualizar el valor del stock de cada producto, cuándo este sea superior al valor de la constante

4.1.2. agregacion2

4.1.3. restadorstock

```
--Trigger para validar que en la VENTA los empleados (Vendedor y Cajero) pertenezcan a la misma sucursal
CREATE OR REPLACE function emple_sucur() RETURNS TRIGGER AS $emple_sucur$
DECLARE suc1 integer;
DECLARE suc2 integer;

BEGIN
    SELECT DISTINCT s.idSucursal INTO suc1 FROM VENTA as v1
    LEFT JOIN EMPLEADO AS e ON new.idEmpleado=e.idEmpleado
    LEFT JOIN SUCURSAL AS s ON s.idSucursal=e.idSucursal;

    SELECT DISTINCT s.idSucursal INTO suc2 FROM VENTA as v2
    LEFT JOIN EMPLEADO AS e ON new.idEmpleado=e.idEmpleado
    LEFT JOIN SUCURSAL AS s ON s.idSucursal=e.idSucursal;

    IF  (suc1!=suc2) THEN
        RAISE NOTICE 'Los empleados pertenecen a sucursales diferentes';
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$emple_sucur$ LANGUAGE PLPGSQL;

CREATE TRIGGER emple_sucur BEFORE INSERT OR UPDATE ON venta
FOR EACH ROW EXECUTE FUNCTION emple_sucur();
```

Este trigger determina si los empleados pertenecen a la misma sucursal. En este caso antes de insertarse se actualiza en venta para verificar que los empleados pertenezcan a la misma sucursal, no puede ser después de ser insertados. Se llega a los id de las sucursales y esto se logra por medio de joins, dos joins por medio de venta se conecta a empleado y de empleado a sucursal. Si los ID de las sucursales son iguales entonces pertenecen a la misma sucursal, donde se pueden insertar, caso contrario simplemente retorna un nulo y la inserción no se completa

```

--Trigger que actualiza los valores en stock de los articulos
CREATE OR REPLACE function restador_stock() RETURNS TRIGGER AS $restador_stock$
DECLARE stockArt smallint;
BEGIN
    SELECT stock INTO stockArt FROM articulo where codigo_barras=new.codigo_barras;
    IF(stockArt<=new.cantidad) THEN
        RAISE NOTICE 'La cantidad de articulos en el carrito supera el stock disponible';
        RETURN NULL;
    END IF;

    UPDATE articulo SET stock=(stockArt-new.cantidad) where codigo_barras=new.codigo_barras;

    RETURN NEW;
END;
$restador_stock$ LANGUAGE PLPGSQL;

CREATE TRIGGER restador_stock BEFORE INSERT ON es_vendido
FOR EACH ROW EXECUTE FUNCTION restador_stock();

```

Se ejecuta antes de que se inserte el valor en la tabla es vendido, en este caso el restador realiza una verificación que cada que se inserte un articulo en es vendido va a verificar que la cantidad que se esta insertando al carrito en es vendido no supera a la cantidad presente en stock, si supera entonces manda el mensaje de la cantidad de artículos en el carrito supera el stock disponible y no se realiza la inserción, se retorna un nulo, en el caso que sea menor, se actualiza el valor de stock en la tabla articulo con la cantidad que se esta ingresando, stock menos la cantidad que se especifica en es vendido, donde el código de barras es el mismo y se retorna.

```

create or replace function formato_curp() returns trigger
language plpgsql
as
$$
declare
cadenaCurp char(18);
c char;
begin
cadenaCurp = new.curp;
if char_length(new.curp) is null or char_length(new.curp) != 18 then
raise exception 'curp invalido';
end if;
for i in 1.. 18 loop
c = substring(cadenaCurp, i, 1);
--raise notice 'posición % Caracter % ', i, c;
if (i <= 4 and c !~* '[a-z]')
then
raise EXCEPTION 'caracteres invalidos en las primeras 4 posiciones';
exit;
elsif (i>4 and i <= 10 and c !~* '[0-9]')
then
raise exception 'los caracteres que corresponden a la fecha son incorrectos';
exit ;
elsif (i=11 and c !~* '[h|m]')
then
raise exception 'caracter referente a sexo invalido';
exit ;
elsif (i > 11 and i <= 16 and c !~* '[a-z]')
then
raise exception 'error en entidad federativa o caracteres subsecuentes';
exit ;
elsif (i > 16 and i <= 18 and c !~* '[a-z0-9]')
then
raise exception 'ultimos 2 caracteres invalidos';
exit ;
end if;
end loop;
if (tg_op = 'UPDATE') then
raise notice 'Cambiando curp % a nuevo curp %', old.curp, new.curp;
end if;
return new;
end
$$;

```

función que verifica el formato del curp

Las tablas empleado, cliente y proveedor poseen un atributo rfc. Este atributo es de tipo char y solo acepta 13 caracteres. Esta función que devuelve un trigger verifica que el valor ingresado cumpla con el formato del rfc que se compone de 4 letras al principio que corresponden al apellido paterno, materno y el nombre seguidos de 6 números que indican la fecha de nacimiento. Los últimos 3 caracteres componen la homoclave. Cada que se ingresa o se actualiza un valor en el atributo curp se verifica que se cumplan las condiciones anteriores. Para esto y gracias a la sentencia new disponible en las funciones que devuelven un trigger, se recorre la cadena ingresada mediante un ciclo for extrayendo cada carácter mediante una función substring. Cada carácter se verifica utilizando un if dependiendo de su posición y en caso de que el formato sea incorrecto se

muestra un mensaje corto explicando donde puede haber un error. Si todo el formato es correcto entonces se permite ingresar el valor.

```
create function formato_rfc() returns trigger
    language plpgsql
as
$$
declare
    cadenaRfc char(13);
    c char;
begin
    cadenaRfc = new.rfc;
    if char_length(new.rfc) is null or char_length(new.rfc) != 13 then
        raise exception 'rfc invalido';
    end if;
    for i in 1.. 13 loop
        c = substring(cadenaRfc, i, 1);
        --raise notice 'posicion % Caracter % ', i, c;
        if (i <= 4 and c !~* '[a-z]')
            then
                raise EXCEPTION 'caracteres invalidos en las primeras 4 posiciones';
                exit;
        elsif (i>4 and i <= 10 and c !~* '[0-9]')
            then
                raise exception 'los caracteres que corresponden a la fecha son incorrectos';
                exit ;
        elsif (i > 10 and i <= 13 and c !~* '[a-z0-9]')
            then
                raise exception 'homoclave invalida';
                exit ;
            end if;
        end loop;
    if (tg_op = 'UPDATE') then
        raise notice 'Cambiando rfc % a nuevo rfc %', old.rfc, new.rfc;
    end if;
    return new;
end
$$;
```

función que verifica el formato del rfc

Esta otra función es muy similar a la que verificar el formato del curp. Solo que esta vez la cadena tiene que poseer solo 13 caracteres. De igual forma se utiliza un ciclo for para iterar sobre la cadena con ayuda de la función substring que va obteniendo carácter por carácter y dependiendo la posición se debe tener solo letras o números. Un detalle de ambas funciones es que cuando se utiliza la sentencia update entonces se mostrará el valor antiguo de la columna curp o rfc y después se muestra el valor que sera ingresado. Claro, para esto es necesario que los datos estén correctos y así se permita asignar de forma correcta los valores

```

create trigger verifica_rfcempleado before insert or update on empleado
for each row execute procedure formato_rfc();

create trigger verifica_curgcemplado before insert or update on empleado
for each row execute procedure formato_curg();

create trigger verifica_rfccliente before insert or update on cliente
for each row execute procedure formato_rfc();

```

triggers en tabla que poseen rfc y/o curp

Aquí podemos observar la creación de los triggers para las tablas que llamaran las 2 funciones anteriores para verificar el formato del curp o rfc. Los triggers revisan cada uno de los registros y verifica los datos antes de que se realice una actualización o inserción a las tablas correspondientes.

```

--Trigger para validar que en la VENTA los empleados (Vendedor y Cajero) pertenezcan a la misma sucursal
CREATE OR REPLACE function emple_sucur() RETURNS TRIGGER AS $emple_sucur$
DECLARE suc1 integer;
DECLARE suc2 integer;

BEGIN
    SELECT DISTINCT s.id_sucursal INTO suc1 FROM VENTA as v1
    LEFT JOIN EMPLEADO AS e ON new.id_empleado=e.id_empleado
    LEFT JOIN SUCURSAL AS s ON s.id_sucursal=e.id_sucursal;

    SELECT DISTINCT s.id_sucursal INTO suc2 FROM VENTA as v2
    LEFT JOIN EMPLEADO AS e ON new.id_empleado1=e.id_empleado
    LEFT JOIN SUCURSAL AS s ON s.id_sucursal=e.id_sucursal;

    IF  (suc1!=suc2) THEN
        RAISE NOTICE 'Los empleados pertenecen a sucursales diferentes';
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$emple_sucur$ LANGUAGE PLPGSQL;

CREATE TRIGGER emple_sucur BEFORE INSERT OR UPDATE ON venta
FOR EACH ROW EXECUTE FUNCTION emple_sucur();

```

funcion que verifica que 2 empleados pertenezcan a la misma sucursal

Al efectuar una venta se necesitan 2 empleados de la sucursal, 1 se encarga de hacer la venta y otro se encarga de cobrar para concretar la venta. Por esta razón es necesario asegurarse que estos 2 empleados trabajen en la misma sucursal y la venta se pueda realizar de forma correcta. Se utilizan joins para determinar la sucursal en la cual trabaja el empleado. Esto se realiza 2 veces pues se quiere saber si ambos empleados cumplen con la condición de trabajar en el mismo local. En caso de que la condición no se cumpla entonces la venta no se realizara pues no hay datos suficientes

para crear un nuevo registro en VENTA ya que se regresa un dato vacío o null. Y como esta función afecta las ventas, entonces se crea un trigger en la tabla ventas que cada que cada que se quiera agregar o actualizar un registro de VENTA se verifique que efectivamente los trabajadores involucrados en la venta trabajan en una misma sucursal.

```
--Trigger para agregar por defecto el nombre completo en CLIENTE para razon social
CREATE OR REPLACE function razonS_Default() RETURNS TRIGGER AS $razonS_Default$
BEGIN
    IF  (new.razon_social is NULL) THEN
        update cliente set razon_social=CONCAT(new.nombre,' ',new.apellido_paterno,' ',new.apellido_materno) WHERE rfc=new.rfc;
    END IF;
    RETURN NEW;
END;
$razonS_Default$ LANGUAGE PLPGSQL;

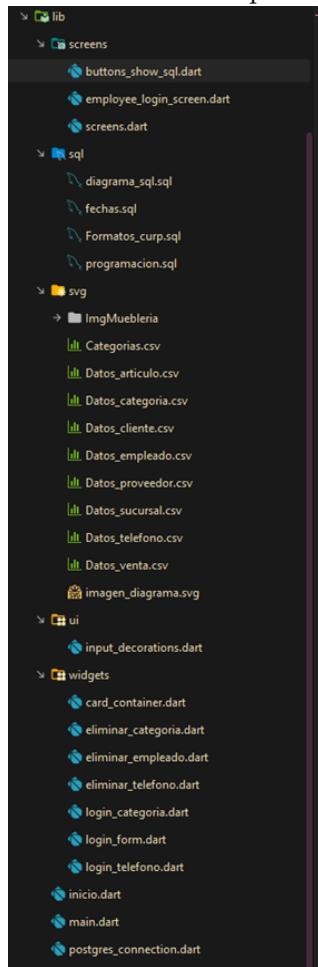
CREATE TRIGGER razonS_Default AFTER INSERT OR UPDATE ON cliente
FOR EACH ROW EXECUTE FUNCTION razonS_Default();
```

funcion que asigna un valor por defecto a la tabla cliente en el atributo razon social

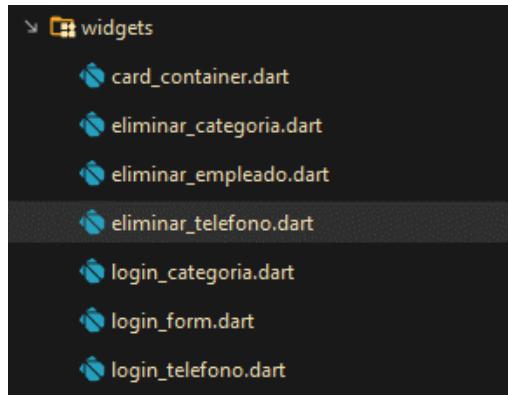
Uno de los requisitos del programa es que el valor por defecto de la columna *razonSocial* de CLIENTE sea el nombre completo. Esta función regresa un trigger así que se puede utilizar la palabra clave *new* y este almacena el registro nuevo antes de que se vea reflejado en la base de datos. Con ayuda de esta palabra se concatenan los datos de los campos *nombre*, *apellidoPaterno* y *apellidoMaterno* esta nueva cadena se asigna al atributo *razonSocial* del nuevo registro o del registro que esta siendo actualizado.

5. Aplicación Móvil

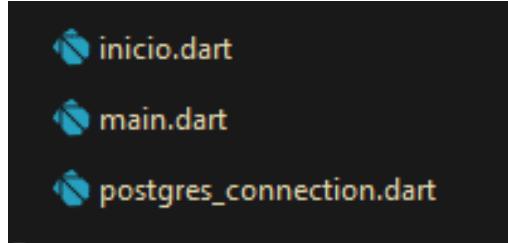
Aquí se muestran el conjunto de directorios con los que esta compuesta la aplicación. Se decidió usar de lenguaje de programación Dart con el Framework Flutter que nos proporciona la utilidad de crear aplicaciones móviles nativas para múltiples entornos.



En el directorio de "screens" se ven las diferentes pantallas que se usaron para elaborar la interfaz gráfica. En el directorio "sql" están todos los comandos sql que necesitaron en la aplicación. En "svg" hay imágenes y los CSV con los que se llenaron las tablas de la base de datos. En el directorio "ui" se encuentra el menú que se uso para poder ingresar a los empleados. En el directorio "widgets" se encuentran todos los widgets que se tuvieron que crear para que la aplicación funcionará correctamente.



El widget cardContainer se uso para poder mostrar el login de una manera mas amigable para el usuario junto con sus validaciones al ingresar los datos, mientras que los eliminar que encargan de crear la interfaz correspondiente a la eliminación de categorias, empleados y teléfonos, mientras que los login se encargan de hacer lo propio con el ingreso de los datos para su subida a la base de datos.



En Postgres connection se ponen todos los valores de conexión con postgres, en inicio y main se puso todo el formato de la aplicación, colores, tipografía, las rutas de como se va a mover el usuario etc. en la siguiente imagen se muestra la cadena de conexión usada en la aplicación, se monto un servidor en Heroku con Postgres, aunque conllevo un gasto, fue el mejor con base en los requerimientos del equipo.

```
var connection:PostgreSQLConnection = PostgreSQLConnection(  
    "ec2-3-230-122-20.compute-1.amazonaws.com",  
    5432,  
    "d47n7tat72jgb5",  
    username: "mdthlconjlitvq",  
    password: "21aafc74c29a1d117e928bbcceabe5eb4ce242b0823e6bd139ddad54622046df",  
    timeoutInSeconds: 300,  
    queryTimeoutInSeconds: 300,  
    allowClearTextPassword: false,  
    useSSL: true,  
);
```

Se usó de 300 milisegundos de timeoutSeconds y de queryTimeoutInSeconds de 300 milisegundos ya que había un problema a la hora de hacer las peticiones con el tiempo de respuesta del servidor y con ello ya se solucionaba.

```
class EliminarCategoria extends StatefulWidget {
  const EliminarCategoria({super.key});

  @override
  State<EliminarCategoria> createState() => _EliminarCategoriaState();
}

class _EliminarCategoriaState extends State<EliminarCategoria> {
  final TextEditingController idCategoriaController = TextEditingController();
  String idCategoria = '111';
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    final newConnection :PostgreSQLConnection = PostgreSQLConnection(...);
    newConnection.open();

    return Scaffold(
      body: Form(
        key: _formKey,
        child: ListView(
          children: [
            Column(
              children: [
                TextFormField(
                  controller: idCategoriaController,
                  autocorrect: false,
                  keyboardType: TextInputType.number,
                  style: const TextStyle(fontSize: 15, color: Colors.black),
                  decoration: InputDecorations.authInputDecoration(
                    hintText: '1234',
                    labelText: 'ID Categoría',
                    prefixIcon: Icons.insert_drive_file_outlined,
                  ),
                  validator: (String? value) {
                    if (value == null || value.isEmpty) {
                      return 'No puede estar vacío';
                    }
                    if (value.contains('.')) || value.contains('-')) {
                      return 'El ID no tiene que tener . ni -';
                    } else if (value.length >= 9) {
                      return 'El ID no puede ser tan largo';
                    }
                    return null;
                  },
                ), // TextFormField
                const SizedBox(
                  height: 20,
                ), // SizedBox
                MaterialButton(
                  onPressed: () {

```

Aquí se puede ver como se hizo el formulario para la implementación de eliminarCategoria, pero se hizo así para los demás, con sus bloques para manejar las excepciones y sus validaciones antes de mandarlo a la base de datos, ya que cada tipo de dato tiene validaciones distintas, e inclusive pasando estas validaciones se hacen otras validaciones en la base de datos antes de ingresar los datos.

```

class CardContainer extends StatelessWidget {
  final Widget child;

  const CardContainer({super.key, required this.child});

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 30),
      child: Column(
        children: [
          const SizedBox(
            height: 20,
          ), // SizedBox
          Container(
            width: double.infinity,
            height: 300,   Miranda, 1/2/2023 2:46 PM
            padding: const EdgeInsets.all(20),
            decoration: _createCardShape(),
            child: child,
          ), // Container
        ],
      ), // Column
    ); // Padding
  }

  BoxDecoration _createCardShape() {
    return BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(25),
      boxShadow: <BoxShadow>[
        BoxShadow(
          color: Colors.black.withOpacity(0.20),
          offset: const Offset(0, 15),
          blurRadius: 15,
        ), // BoxShadow
      ], // <BoxShadow>[]
    ); // BoxDecoration
  }
}

```

Se crearon la parte

del formato para los formularios, con sus distintos textos y formatos específicos.

```

selectImageArticles() async { ... }

selectAllArticles() async { ... }

selectAllCategories() async { ... }

selectAllCustomers() async { ... }

selectAllEmployees() async { ... }

selectAllProviders() async { ... }

selectAllBranches() async { ... }

selectAllTelephones() async { ... }

selectAllSales() async {
  try {
    resultsVentas = await connection.query("Select * from venta");
    debugPrint("seleccion de todas las ventas");

    //Imprime todos los valores de la tabla
    //todo
    for (int i:int = 0; i < resultsVentas.length; i++) {
      for (int j:int = 0; j < resultsVentas[i].length; j++) {
        debugPrint(resultsVentas[i][j].toString());
        //resultsVentas[i][j] = resultsVentas[i][j];
      }
    }
    return resultsVentas;
  } catch (e) {
    if (kDebugMode) {
      print('error');
      print(e.toString());|  Miranda, 1/3/2023 10:50 PM · 5
    }
  }
}

```

Aquí se muestran algunas de las funciones aplicadas para la interacción con la base de datos, ya que se tuvieron que obtener de distintas tablas y distintos valores, algunas requerían hacer inserts,

o deletes para borrar los valores dados un id por el usuario.

```
var connection :PostgreSQLConnection = PostgreSQLConnection(  
    "ec2-3-230-122-20.compute-1.amazonaws.com",  
    5432,  
    "d47n7tat72jgb5",  
    username: "mdthlconjlitvq",  
    password: "21aaafc74c29a1d117e928bbcceabe5eb4ce242b0823e6bd139ddad54622046df",  
    timeoutInSeconds: 300,  
    queryTimeoutInSeconds: 300,  
    allowClearTextPassword: false,  
    useSSL: true,  
);  
  
List<List<dynamic>> resultsArticulos :List<List<dynamic>> = [];  
List<List<dynamic>> resultsCategorias :List<List<dynamic>> = [];  
List<List<dynamic>> resultsClientes :List<List<dynamic>> = [];  
List<List<dynamic>> resultsEmpleados :List<List<dynamic>> = [];  
List<List<dynamic>> resultsProveedores :List<List<dynamic>> = [];  
List<List<dynamic>> resultsSucursales :List<List<dynamic>> = [];  
List<List<dynamic>> resultsTelefono :List<List<dynamic>> = [];  
List<List<dynamic>> resultsVentas :List<List<dynamic>> = [];  
List<List<dynamic>> resultsFotografias :List<List<dynamic>> = [];  
  
class PostgresConnection {  
    Future connect() async {...}  
  
    Future disconnect() async {...}  
  
    selectImageArticles() async {...}  
  
   selectAllArticles() async {...}  
  
   selectAllCategories() async {...}  
  
   selectAllCustomers() async {...}  
  
   selectAllEmployees() async {...}  
  
   selectAllProviders() async {...}  
  
   selectAllBranches() async {...}  
  
   selectAllTelephones() async {...}  
  
   selectAllSales() async {...}  
}
```

Las

funciones retornan una lista de listas dinámicas que son las que se usan para crear los widgets y tener acceso a ellos. Para poder asegurar el correcto funcionamiento de las funciones.

```

class LoginCategoria extends StatefulWidget {
  const LoginCategoria({Key? key}) : super(key: key);

  @override
  State<LoginCategoria> createState() => _LoginCategoriaState();
}

class _LoginCategoriaState extends State<LoginCategoria> {
  final TextEditingController idCategoriaController :TextEditingController = TextEditingController();
  final TextEditingController nombreCategoriaController :TextEditingController =
    TextEditingController();
  final TextEditingController descripcionCategoriaController :TextEditingController =
    TextEditingController();
  String idCategoria :String = '111';
  String nombreCategoria :String = 'nombre';
  String descripcionCategoria :String = 'apellido Paterno';

  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  Miranda, Yesterday · Se agregaron las opciones para agregar empleado, categoria y se mostraron
  @override
  Widget build(BuildContext context) {
    final newConnection :PostgreSQLConnection = PostgreSQLConnection(
      "ec2-3-230-122-20.compute-1.amazonaws.com",
      5432,
      "d47n7tat72jgb5",
      username: "mdthlconjltvq",
      password:
        "21aaafc74c29a1d117e928bbcceabe5eb4ce242b0823e6bd139ddad54622046df",
      timeoutInSeconds: 300,
      queryTimeoutInSeconds: 300,
      allowClearTextPassword: false,
      useSSL: true,
    );
    newConnection.open();

    return Scaffold(
      body: Form(
        key: _formKey,
        child: ListView(
          children: [
            Column(
              children: [
                TextFormField(
                  controller: idCategoriaController,
                  autocorrect: false,
                  keyboardType: TextInputType.number,
                  style: const TextStyle(fontSize: 15, color: Colors.black),
                  decoration: InputDecorations.authInputDecoration(
                    hintText: '1234',
                    labelText: 'ID Categoria',
                    prefixIcon: Icons.insert_drive_file_outlined,
                  ),
                  validator: (String? value) {
                    if (value == null || value.isEmpty) {
                      return 'No puede estar vacio';
                    }
                  }
                )
              ],
            )
          ],
        )
      )
    );
  }
}

```

Aquí se

puede ver el login de categoría con sus controller para poder acceder a los datos y poder mandarlos y validarlos en la base de datos y mostrar la actualización en pantalla.

```

class LoginForm extends StatefulWidget {
  const LoginForm({super.key});

  @override
  State<LoginForm> createState() => _LoginFormState();
}

class _LoginFormState extends State<LoginForm> {
  final TextEditingController idEmpleadoController = TextEditingController();
  final TextEditingController nombreController = TextEditingController();
  final TextEditingController apellidoPaternoController = TextEditingController();
  final TextEditingController apellidoMaternoController = TextEditingController();
  final TextEditingController curpController = TextEditingController();
  final TextEditingController rfcController = TextEditingController();
  final TextEditingController fechaIngresoController = TextEditingController();
  final TextEditingController tipoController = TextEditingController();
  final TextEditingController emailController = TextEditingController();
  final TextEditingController numeroExteriorController = TextEditingController();
  final TextEditingController numeroInteriorController = TextEditingController();
  final TextEditingController estadoController = TextEditingController();
  final TextEditingController codigoPostalController = TextEditingController();
  final TextEditingController calleController = TextEditingController();
  final TextEditingController idSucursalController = TextEditingController();
  final TextEditingController idSupervisorController = TextEditingController();

  String idEmpleado String = '111';
  String nombre String = 'nombre';
  String apellidoPaterno String = 'apellido Paterno';
  String apellidoMaterno String = 'apellido Materno';
  String curp String = 'CURP';
  String rfc String = 'RFC';
  String fechaIngreso String = 'fecha Ingreso';
  String tipo String = 'TIPO';
  String email String = 'email';
  String numeroExterior String = 'ext123';
  String numeroInterior String = 'int123';
  String estado String = 'estado';
  String codigoPostal String = 'cp04531';
  String calle String = 'calle';
  String idSucursal String = 'idsuc123';
  String idSupervisor String = 'idSup123';

  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    final newConnection PostgreSQLConnection = PostgreSQLConnection(
      "ec2-3-230-122-20.compute-1.amazonaws.com",
      5432,
      "d47n7tat72jgb5",
      username: "mdthiconjltvq",
      password:
        "21aaafc74c29a1d117e928bbcceabe5eb4ce242b0823e6bd139ddad54622046df",
      timeoutInSeconds: 300,
      queryTimeoutInSeconds: 300,
      allowClearTextPassword: false,
      useSSL: true,
    );
    newConnection.open();

    return Form(
      key: _formKey,
      child: Column(
        children: [
          //ID Empleado
          TextFormField(
            controller: idEmpleadoController,
            autocorrect: false,
            keyboardType: TextInputType.number,
            style: const TextStyle(fontSize: 15, color: Colors.black),
            decoration: InputDecorations.authInputDecoration(
              hintText: '1234',
              labelText: 'ID Empleado',
              prefixIcon: Icons.insert_drive_file_outlined,
            ),
            validator: (String? value) {
              if (value == null || value.isEmpty) {
                return 'No puede estar vacío';
              }
              if (value.contains('.') || value.contains('-')) {
                return 'El ID no tiene que tener . ni -';
              } else if (value.length >= 9) {
                return 'El ID no puede ser tan largo';
              }
            },
          ),
        ],
      ),
    );
  }
}

```

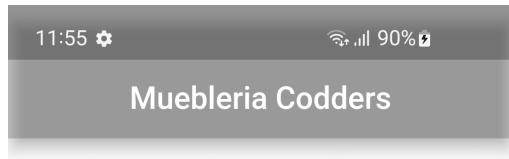
El login más complejo de

elaborar fue el de empleado ya que son demasiadas validaciones y se requerían muchos controller para poder actualizar los datos

```
void main() => runApp(const MyApp());  
  
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      debugShowCheckedModeBanner: false,  
      //theme: ThemeData.dark(),  
      theme: ThemeData.light().copyWith(  
        iconTheme: const IconThemeData(  
          color: Colors.black,  
          opacity: 100,  
          size: 10,  
        ), // IconThemeData  
        hoverColor: Colors.grey[900],  
        primaryColor: Colors.black,  
        scaffoldBackgroundColor: Colors.grey[200],  
        appBarTheme: const AppBarTheme(  
          elevation: 10,  
          color: Colors.black12,  
        ), // AppBarTheme  
        colorScheme: ColorScheme.fromSwatch().copyWith(secondary: Colors.black),  
        focusColor: Colors.grey,  
      ),  
      //initialRoute: 'buttons_show_sql',  
      initialRoute: 'inicio',  
      routes: {  
        'inicio': (context) => const Inicio(),  
        'categoria': (context) => const LoginCategoria(),  
        'telefono': (context) => const LoginTelefono(), // Miranda, Yesterday  
        'login': (context) => const EmployeeLogin(),  
        'eliminar_empleado': (context) => const EliminarEmpleado(),  
        'eliminar_categoria': (context) => const EliminarCategoria(),  
        'eliminar_telefono': (context) => const EliminarTelefono(),  
        'buttons_show_sql': (context) => const ButtonsShowSql(),  
      },  
    ); // MaterialApp
```

En el

archivo de main se crearon los diseños de la app y se establecieron las rutas de interacción con el usuario.



Ir al inicio

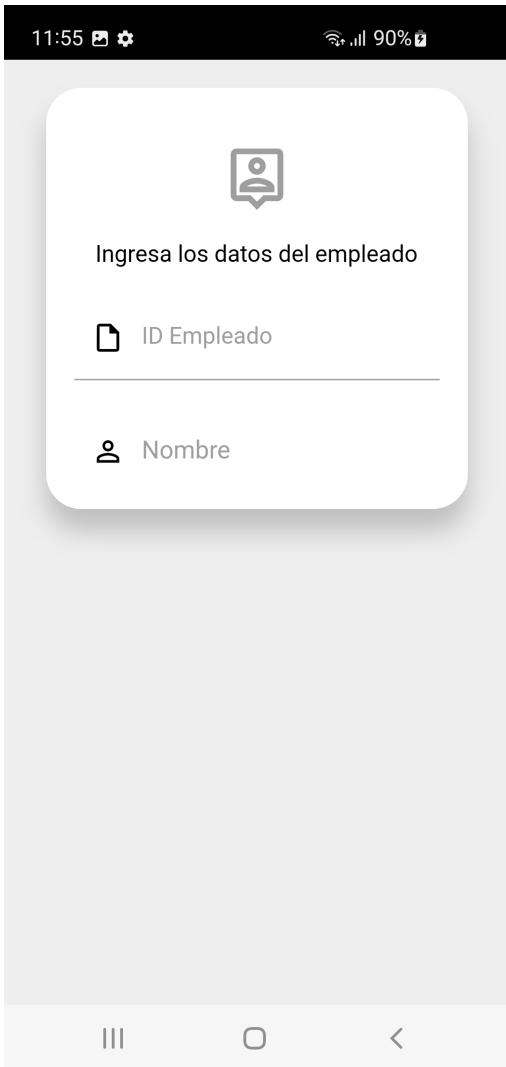


Este es el inicio de la app con un botón que lleva al inicio
donde se encuentras las interacciones con la base de datos.



Aquí se encuentran todas las funcionalidades de la app,

tanto para mostrar valores y borrar los mismos. El botón de Solicitar... lo que hace es que hace la petición a la base de datos para que se muestren en pantalla y las de mostrar/ocultar lo que hacen es que ya teniendo los valores en pantalla poder ocultarlos y mostrarlos sin necesidad de hacer otra petición a la base de datos y evitando así que se sature visualmente la app, ya que son bastantes datos los que se pueden llegar a mostrar en pantalla.

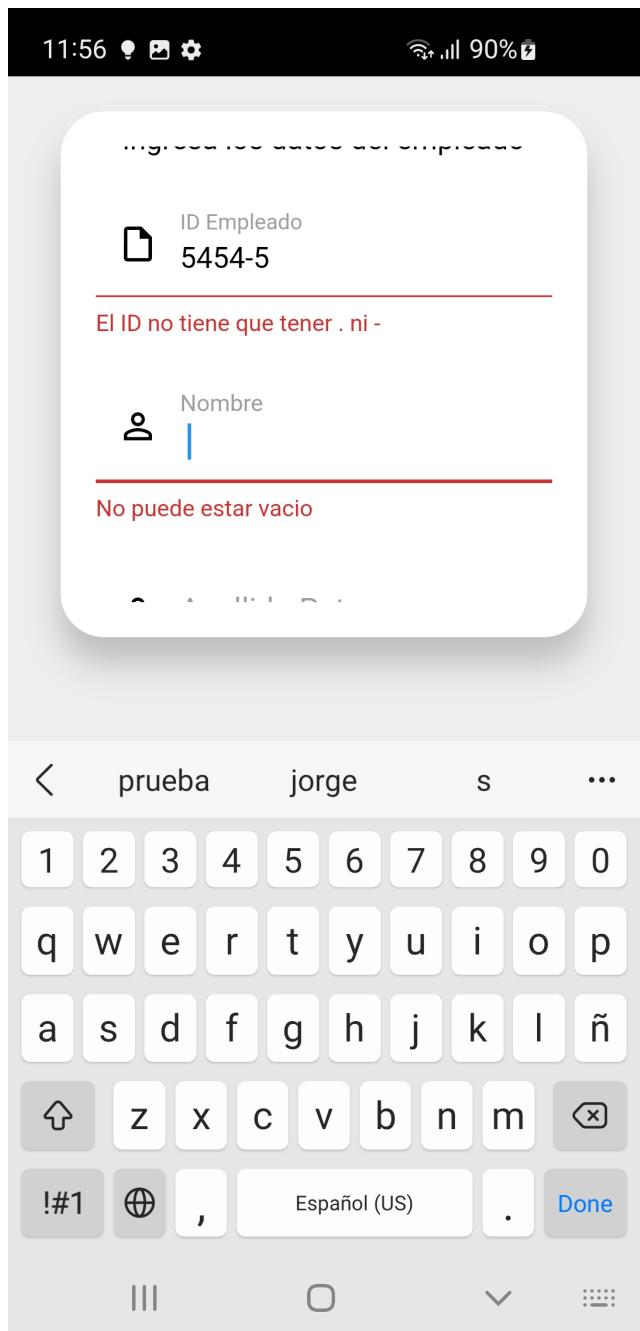


Aquí se muestra la manera en que se ingresan los datos al momento de crear un empleado.



Se hicieron varias validaciones y se muestran en

pantalla como que el curp si es menor a 18 le avisa que esta mal, tanto con el RFC si esta más corto o largo, y en la fecha avisa que tiene que estar en un formato específico.

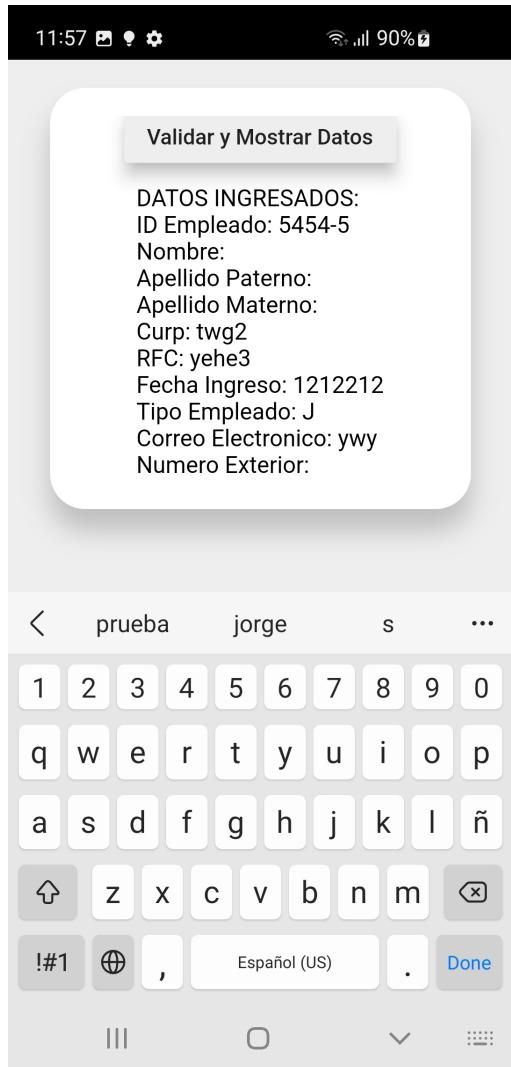


En los ID se avisa que no pueden tener un . o -

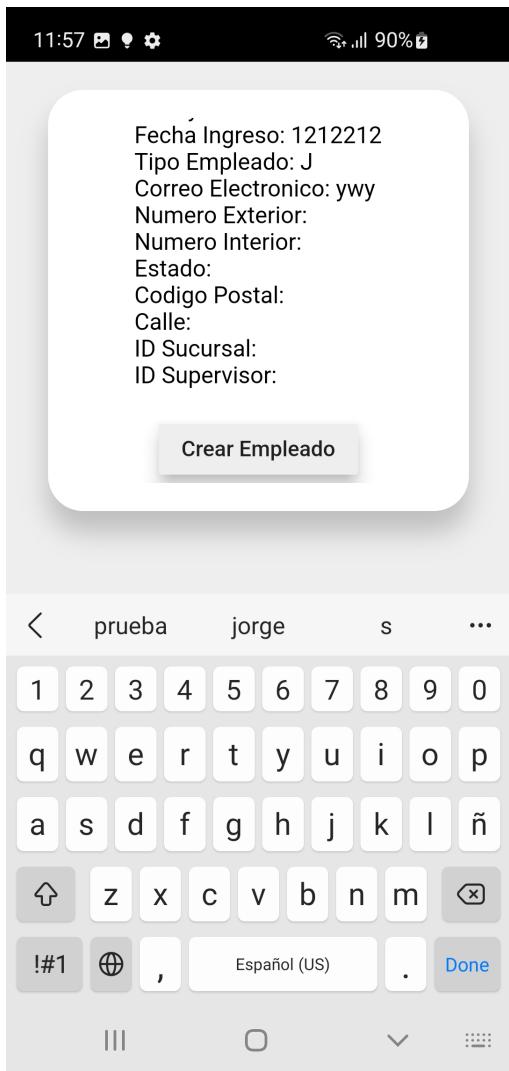
en el texto, ya que tienen que ser números enteros positivos y los atributos, que no son valores que
puedan ser null tienen que tener un valor.



En los correos se valida que contengan el carácter @ ya que si no lo tiene significa que no es un correo electrónico valido. Tanto los numero que no pueden ser null se valida



Ya una vez ingresados los datos se muestran al usuario para que valide si son correctos antes de ingresarlos y si son correctos ya el usuario presiona el botón de “Crear Empleado”.



Presionando el botón se crea el Empleado o categoría o teléfono, dependiendo de lo que sea que se quiera crear y siempre se hacen las validaciones.

11:57 90%

Muebleria Codders



CONJUNTO DE DORMITORIO
\$42100
cristal
muebles de cristal
Maderas SA



COMEDOR RUSTICO
\$21875
madera
muebles de madera

Aquí se muestra como se ven los artículos con su foto,
precio de venta y descripción.



Aquí se muestra como se muestran en pantalla los datos accesados de Artículos, así se muestran los datos de las distintas categorías. Y se actualizan cada que se agregan los valores, solamente se tiene que presionar el botón de Solicitar Info... y se vuelve a hacer la petición a la base de datos para mostrarlos en pantalla



Y para la eliminación de los datos simplemente se tiene que poner el valor del ID de lo que se quiera eliminar, se validan los dato y se elimina si es que existe ese ID.

6. Conclusiones

Carranza Ochoa José David

En el presente proyecto logramos implementar los conceptos de bases de datos aterrizados al modelado de un buen sistema de mueblería, el cual mantiene un soporte de integridad de los datos. Partimos de una recopilación de todo el semestre expresando las distintas etapas de diseño, siendo el modelado de entidad relación el más importante ya que a partir de este se consideran todos los requisitos a emplear, si bien resulta un poco tedioso el efectuar varias validaciones, se considera indispensable el invertirle la mayor cantidad de tiempo.

Avanzando encontramos etapas de mapeo como lo es el modelo intermedio para encontrar su forma lógica, dicho modelo tiene un acercamiento más cercano a SQL, sin embargo aún no se trata de dicho modelo. Cuando se logra llegar al modelo final nos encontramos con el verdadero reto en la programación.

Este proyecto en cuanto a los triggers despertó mi curiosidad y gran apego por resolver problemas, ya que durante el desarrollo de los mismos los errores estaban presentes en todo momento, es por ello que el trabajo colaborativo tomó relevancia para poder resolver los obstáculos, aprendiendo durante la marcha la codificación en base64, uso de disparadores antes y después de una operación sobre una tabla y finalmente, la manipulación de sentencias como INSERT, DELETE, UPDATE junto con la cláusula SELECT.

Salgado Miranda Jorge

Este proyecto tuvo una dificultad adecuada, ya que se pudieron ver temas que no se alcanzaron a tocar en clase, al momento de la implementación gráfica surgieron varios problemas a la hora de la creación de widgets para mostrar los datos en la interfaz gráfica y el hacer validaciones en la misma, se pudieron ver algunas implementaciones del API de postgres con ayuda del lenguaje Dart. Para solventar todos los problemas nos ayudamos en la documentación oficial y en una inteligencia artificial llamada ChatGPT que nos ayuda a generar código y solucionar preguntas de manera adecuada, pero nos ayuda a darnos una idea de como hace

Al momento de extraer las imágenes para mostrarlos en la interfaz se obtuvieron bastantes errores, por como se implementaba la lista de listas dinámicas y el andar haciendo la decodificación de base 64, ya que se guardaban caracteres especiales y se tuvo que hacer una depuración con expresiones

regulares para su correcto funcionamiento. Se cumplieron todos los puntos, aunque una limitante fue el tiempo, ya que son las 11:50 de la noche y seguimos redactando este proyecto para poder entregarlo de manera completa.

Casique Corona Luis Enrique

El objetivo de este proyecto fue cumplido. Tuvimos un acercamiento a un escenario real donde un cliente da ciertos requisitos para crear una base de datos que permita tener un control sobre su inventario. Los conceptos vistos en las clases de teoría proporcionaron una base sólida para generar un primer modelo para la base de datos que, con el paso de los días, fuimos puliendo más y más. Aunque la parte complicada fue programar a nivel de base de datos, aunque la lógica de programación se mantiene y es casi lo mismo que hemos visto en las materias de programación de los primeros semestres, esto fue un nuevo enfoque y costó un poco de trabajo adaptarse a la forma de manejar correctamente las estructuras de control y plantear bien las funciones para que devuelvan el dato que queremos. Luego, otro reto muy grande fue la creación de una app que se conectara a nuestra base de datos, aunque lo más complicado fue el apartado estético y también hubo algunos problemas a la hora de querer obtener imágenes desde la base.

Sánchez de Santiago Julián

El proyecto que se ha realizado en esta ocasión ha sido de suma utilidad para afianzar los conocimientos adquiridos a lo largo del curso, desde la parte conceptual hasta la implementación en un proyecto físico entregable a un cliente. La dificultad del proyecto ha sido adecuada pues nos ha permitido emplear las distintas herramientas y recursos de la programación en base de datos para generar un proyecto funcional que pueda eficientizar el funcionamiento de un proceso existente y el cual pueda brindar al usuario la información de manera íntegra con la finalidad de que pueda emplearla para distintos fines. De manera general, el ha generado un gran reto al tener que recabar herramientas que he empleado anteriormente, así como buscar aprender nuevas que tuvieron que ser empleadas en el mismo.

Zárate García Zuriel

A lo largo de este proyecto, pudimos poner en práctica todos los conceptos aprendidos a lo largo del curso de esta materia. Comenzando por la fase de toma de requerimientos, la comprensión de los mismos, realización de juntas de trabajo para debatirlos y juntas con el profesor para aclarar otros puntos. Posteriormente, la fase de diseño conceptual implementando el modelo Entidad

Relación. Luego, el diseño lógico a través del modelo relacional, el cuál sufrió varios cambios debido a la etapa de normalización. Finalmente, la implementación en la base y la creación de bloques de código para controlar diferentes casos dentro de la base.

Creo que nuestra organización como equipo no fue la mejor, ya que intentamos arrancar con varias de estas etapas al mismo tiempo, provocando cambios en el modelo relacional y entidad relación cuando ya se tenía creada la base y las tablas, cosa que complicó arreglar situaciones particulares que no debían suceder. Sin embargo, me parece que al final se cumplieron los objetivos del proyecto, debido a que pudimos usar la mayoría de conceptos y

7. Referencias

<https://flutter.dev/>

<https://pub.dev/packages/postgres>