****

**Tecnicatura Universitaria en Programación**

**Base de Datos II**

**Equipo # 29**

**Integrantes:**

**Correa Gimenez, Leandro Sebastian - 23908**

**Jesus Esteban Ballico – 30884**

**Facundo Palomanes – 28799**

Explicación del Sistema

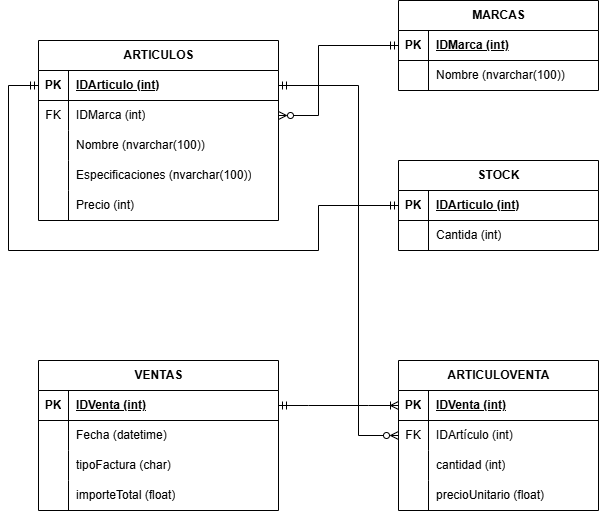
El sistema trata de un punto de venta de carácter general, orientado a pequeños y medianos comercios como kioskos, almacenes y locales de carácter general. El propósito del mismo es facilitar la administración del local, contando con diversas funcionalidades para registrar los movimientos del día a día, y en base a ellas, obtener métricas y reportes para llevar un control del modelo de negocio.

### Funcionalidades principales

* **Gestión del stock:** Permite realizar la carga de artículos al inventario disponible, gestionar su stock disponible del mismo para la venta, asignarle un nombre, marca, categoría, precio.
* **Registro de ventas:** Permite registrar una venta.
* **Reportes y métricas:** A través de las ventas registradas, permite obtener reportes claves para el negocio, como productos más vendidos, ingresos totales por día / semana / mes, reporte de stock actual.
* **Ordenes de compra:** En base al stock que va resultando, permite realizar ordenes de compras de los artículos que tengan un stock por debajo del umbral normal.

El sistema está construido con una base de datos relacional que almacena toda la información necesaria para gestionar estas funcionalidades. Se han implementado tres vistas, dos procedimientos almacenados y dos triggers.

## Diagrama de Entidad Relación



## Objetos de Base de Datos clave en el sistema

### Vista 1

Explicación vista 1

|  |
| --- |
| -- Codigo |

### Vista 2

La vista **VW\_MostrarInformacionVentas** nos devuelve un listado de todos los ítems de todas las ventas realizadas, para luego poder aplicar más filtros sobre el resultado.

La misma nos trae las siguientes columnas:

IDVenta, Fecha, Artículo, Cantidad, Precio Unitario, SubTotal, TipoFactura

La vista simplifica los JOINS, para que podamos filtrar detalles de una venta por su ID, y ver los artículos de esa venta, ver los artículos que se vendieron en un lapso de tiempo, sacar conclusiones de negocio mediante los SubTotales, etc.

|  |
| --- |
| CREATE VIEW VW\_MostrarInformacionVentas AS  SELECT  V.IDVenta, V.Fecha, A.Nombre as 'Artículo', AV.Cantidad, AV.PrecioUnitario, AV.Cantidad \* AV.PrecioUnitario AS 'SubTotal', V.TipoFactura  FROM ArticulosVenta AS AV  INNER JOIN Articulos AS A ON AV.IDArticulo = A.IDArticulo  INNER JOIN Ventas AS V ON AV.IDVenta = V.IDVenta; |

### Vista 3

Aquí se explica el propósito de la Vista 2 del sistema.

|  |
| --- |
| -- codigo |

### Procedimiento Almacenado 1

El procedimiento **SP\_AgregarArticulo** permite dar de alta un artículo nuevo al negocio, y establecerle un posible precio de venta, lo que significa que lo pone disponible para luego cargarle stock, si lo tuviera. Por defecto, establece stock en 0.

El mismo valida que no se inserte un precio negativo, y que previamente esté registrada la marca del producto en el sistema.

|  |
| --- |
| CREATE PROCEDURE SP\_registrarArticulo(  @nombreArticulo nvarchar(100),  @idMarca int,  @precioArticulo float  )  AS  BEGIN  BEGIN TRY  BEGIN TRANSACTION    IF (@precioArticulo < 0) BEGIN  RAISERROR('No se puede insertar un precio de artículo negativo', 16, 1)  END  DECLARE @marcaID int  SET @marcaID = 0;  SELECT @marcaID = IDMarca FROM MARCAS WHERE IDMarca = @idMarca;  IF (@marcaID = 0) BEGIN  RAISERROR('Número de ID de marca no existe en la base de datos', 16, 1)  END  INSERT INTO Articulos (Nombre, IDMarca, Precio) VALUES (@nombreArticulo, @idMarca, @precioArticulo);  INSERT INTO Stock VALUES (@@IDENTITY, 0);    COMMIT TRANSACTION;  END TRY  BEGIN CATCH  ROLLBACK TRANSACTION;  PRINT ERROR\_MESSAGE();  END CATCH  END; |

### Procedimiento Almacenado 2

El procedimiento **SP\_RegistrarVentaArticulo** realiza un registro de inserión de un artículo que forma parte de una venta en cuestión. Se le deberá proporcionar el ID de Venta al cual se va a agregar el artículo, el ID del artículo que forma parte de esa venta, y la cantidad que forma parte de la misma.

El procedimiento valida que el ID de venta esté registrado en la base de datos (Por lo que antes, se crea una venta, y luego se cargan los artículos que forman parte de la misma)

El mismo también valida que el ID de artículo proporcionado exista en la base de datos.

También valida que haya stock disponible para realizar dicha venta, además de las evidentes validaciones como números negativos, en ids, y cantidades.

A su vez, realiza una actualización del stock, actualizando dicha tabla.

El procedimiento, guarda en la tabla ‘ArticulosVentas’ el precio unitario del artículo en el momento de ejecución del procedimiento, porque puede que el precio del artículo cambie, sin embargo, el precio del artículo en el momento de la venta, era otro, por eso se guarda este último.

|  |
| --- |
| CREATE PROCEDURE SP\_RegistrarVentaArticulo (  @idVenta INT,  @idArticulo INT,  @cantidad INT  )  AS  BEGIN  BEGIN TRY  BEGIN TRANSACTION    -- Validación de idVenta  IF (@idVenta < 0) BEGIN  RAISERROR('No se puede insertar un id de venta negativo', 16, 1);  END  IF (SELECT count(IDVenta) FROM Ventas WHERE IDVenta = @idVenta) = 0 BEGIN  RAISERROR('No existe este id de venta', 16, 1);  END  -- Validación de idArticulo  DECLARE @precioUnitario FLOAT; -- Para guardar el precio unitario, para después multiplicarlo por la cantidad, y guardar el subtotal en la db  IF (@idArticulo < 0) BEGIN  RAISERROR('No puedes ingresar un ID de artículo negativo', 16, 1);  END  --IF (SELECT count(IDArticulo) FROM Articulos WHERE IDArticulo = @idArticulo) = 0 BEGIN  SELECT @precioUnitario = Precio FROM Articulos WHERE IDArticulo = @idArticulo;  IF (@precioUnitario IS NULL) BEGIN  RAISERROR('No existe este id de artítuclo', 16, 1);  END  -- Validación cantidad  IF (@cantidad < 0) BEGIN  RAISERROR('No puedes insertar una cantidad negativa', 16, 1);  END    -- Validación de stock  DECLARE @stockActual INT;  SELECT @stockActual = Cantidad FROM Stock WHERE IDArticulo = @idArticulo;  IF (@stockActual < @cantidad) BEGIN  RAISERROR('Stock insuficiente para efectuar esta transaccion', 16, 1);  END  -- Actualizar el stock  UPDATE Stock SET Cantidad = @stockActual - @cantidad WHERE IDArticulo = @idArticulo;  -- Registrar la venta del artículo para una venta en específico  INSERT INTO ArticulosVenta VALUES (@idVenta, @idArticulo, @cantidad, @precioUnitario);  COMMIT TRANSACTION;  END TRY  BEGIN CATCH  ROLLBACK TRANSACTION;  PRINT ERROR\_MESSAGE();  END CATCH END; |

### Trigger 1

Explicacion trigger 1

|  |
| --- |
| -- trigger 1 |

### Trigger 2

Aquí se explica el propósito del Trigger 2 de la base de datos.

|  |
| --- |
| -- Trigger 2 |

## Links a los recursos

**Script de creación de base de datos con datos**

<https://github.com/Salchy/TP-integrador-dbii>

**Video demo del sistema (hasta 25 minutos)**

<https://youtube.com/usuario/VideoDemo>