****

**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

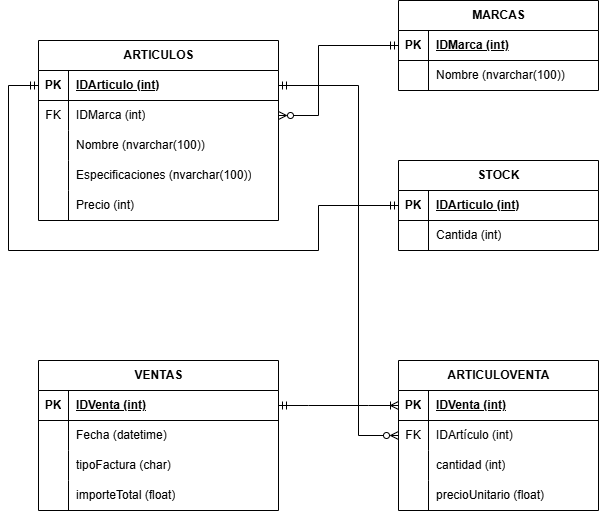
El sistema trata de un punto de venta de carácter general, orientado a pequeños y medianos comercios como kioskos, almacenes y locales de carácter general. El propósito del mismo es facilitar la administración del local, contando con diversas funcionalidades para registrar los movimientos del día a día, 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 cuatro vistas, cinco procedimientos almacenados y dos triggers.

## Diagrama de Entidad Relación



## Objetos de Base de Datos clave en el sistema

### Vista 1

La vista **VW\_MostrarStock** nos devuelve un listado de todos los ítems de la tabla de Stock con el nombre del articulo y la cantidad del mismo.

La misma nos trae las siguientes columnas: Articulo, Cantidad

La vista usa un INNER JOIN para poder mostrar el Nombre del articulo junto con la cantidad filtrando emparejando el IDArticulo de la tabla Articulos y el de Stock obteniendo las propiedades de las 2 tablas.

|  |
| --- |
| CREATE VIEW VW\_MostrarStock AS  SELECT AS Articulos.Nombre AS 'Articulo', Stock.Cantidad AS 'Cantidad'  FROM Articulos  INNER JOIN Stock ON Articulos.IDArticulo = Stock.IDArticulo |

### 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

La vista **VW\_PromedioDeVentas** nos devuelve un solo valor que es el del valor del importe promedio entre todas las ventas.

|  |
| --- |
| CREATE VIEW VW\_PromedioDeVentas AS  SELECT AVG(ImporteTotal) AS PromedioImporteTotal  FROM Ventas; |

### Vista 4

La vista **VW\_MostrarArticulosStockBajo** muestra los artículos que tienen stock bajo.

|  |
| --- |
| CREATE VIEW VW\_MostrarArticulosStockBajo AS  SELECT  Nombre  FROM Articulos  WHERE Eliminado = 1; |

### 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; |

### Procedimiento Almacenado 3

El procedimiento almacenado **sp\_AperturaVenta\_Simple** realiza un registro de venta, para el cual luego se van a cargar los artículos que hacen referencia a esa venta.

El mismo retorna un dato scalar, que hace referencia al ID de venta que se generó, para así poder capturarlo desde la aplicación / sistema, para poder cargar artículos a esa venta.

|  |
| --- |
| CREATE PROCEDURE sp\_AperturaVenta\_Simple (  @TipoFactura CHAR(1),  @ImporteTotal FLOAT  )  AS  BEGIN  SET NOCOUNT ON; -- Evita que el servidor SQL realize una devolución con las filas afectadas, puede mejorar el rendimiento  BEGIN TRY  IF (@TipoFactura != 'A' AND @TipoFactura != 'B') BEGIN  RAISERROR('Tipo de factura inválido', 16, 1);  END  INSERT INTO Ventas (TipoFactura, ImporteTotal)  OUTPUT inserted.IDVenta  VALUES (@TipoFactura, @ImporteTotal);  END TRY  BEGIN CATCH  PRINT ERROR\_MESSAGE();  END CATCH  END |

### Procedimiento Almacenado 4

El procedimiento almacenado **SP\_ModificarStock** realiza una modificación de stock de un artículo. Puede sustraer stock, como adicionar.

El mismo tiene todas las validaciones necesarias para evitar inconsistencia en los datos, tales como números negativos, id de artículos que no existan, etc.

|  |
| --- |
| CREATE PROCEDURE SP\_ModificarStock(  @idArticulo INT,  @cantidad INT  )  AS  BEGIN  BEGIN TRY  BEGIN TRANSACTION  -- Validación de idArticulo  DECLARE @stockActual FLOAT; -- Para guardar el stock actual, para luego acumularlo  IF (@idArticulo < 0) BEGIN  RAISERROR('No puedes ingresar un ID de artículo negativo', 16, 1);  END  SELECT @stockActual = Cantidad FROM Stock WHERE IDArticulo = @idArticulo;  IF (@stockActual IS NULL) BEGIN  RAISERROR('No existe este id de artítuclo', 16, 1);  END  IF (@cantidad = 0) BEGIN  UPDATE Stock SET Cantidad = 0 WHERE IDArticulo = @idArticulo;  END  ELSE IF (@cantidad < 0) BEGIN -- Es una modificación de sustracción  -- Validación de stock actual  IF (@stockActual + @cantidad < 0) BEGIN  RAISERROR('No se puede realizar esta modificación porque generaría stock en negativo', 16, 1);  END  END  UPDATE Stock SET Cantidad = @stockActual + @cantidad WHERE IDArticulo = @idArticulo;  COMMIT TRANSACTION;  END TRY  BEGIN CATCH  ROLLBACK TRANSACTION;  PRINT ERROR\_MESSAGE();  END CATCH  END |

### Procedimiento Almacenado 5

El procedimiento **SP\_RegistrarMarca** realiza un alta de una marca al sistema.

|  |
| --- |
| CREATE PROCEDURE SP\_RegistrarMarca(  @nombreMarca nvarchar(100)  )  AS  BEGIN  BEGIN TRY  INSERT INTO Marcas (Nombre) VALUES (@nombreMarca);  END TRY  BEGIN CATCH  PRINT ERROR\_MESSAGE();  END CATCH  END; |

### Trigger 1

El trigger **trg\_AlertaCantidadStockCero** observa la tabla stock en búsqueda de si algún articulo queda con un stock igual a 0, si esto se cumple muestra una alerta diciendo “Atencion, uno o mas productos están actualmente sin stock, así que se les dará una baja lógica del mismo”. Entonces entra en la tabla Articulos y cambia la casilla Eliminado que es de tipo bit not null default 0, poniendo su valor en 1 refiriendose a eliminado. Busca el articulo correcto usando el insertado con la palabra reservada INSERTED refiriéndose al articulo con cantidad cero y buscando su id en la tabla artículos con la propiedad IDArticulo.

|  |
| --- |
| CREATE TRIGGER trg\_AlertaCantidadStockCero  ON Stock  AFTER UPDATE  AS  BEGIN  IF EXISTS (SELECT 1 FROM INSERTED I WHERE I.cantidad = 0)  BEGIN  PRINT ’Atencion, uno o mas productos estan actualmente sin stock, asi que se les dara una baja logica del mismo’  UPDATE Articulos set eliminado = 1 from INSERTED where Articulos.IDArticulo = INSERTED.IDArticulo;  END  END |

### Trigger 2

El trigger **TRG\_ActualizarImporteVenta** es parte de la funcionalidad core del sistema, realiza la actualización del importe de la venta, cada vez que se inserta un nuevo articulo a una venta.

|  |
| --- |
| CREATE TRIGGER trg\_ActualizarImporteVenta  ON ArticulosVenta  AFTER INSERT  AS  BEGIN  SET NOCOUNT ON;  UPDATE V  SET ImporteTotal = (  SELECT SUM(AV.Cantidad \* AV.PrecioUnitario)  FROM ArticulosVenta AV  WHERE AV.IDVenta = V.IDVenta  )  FROM Ventas V  INNER JOIN inserted I ON V.IDVenta = I.IDVenta;  END |

## 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://youtu.be/mqFnW2l1s3E>