Entrega 2
RomeroUro

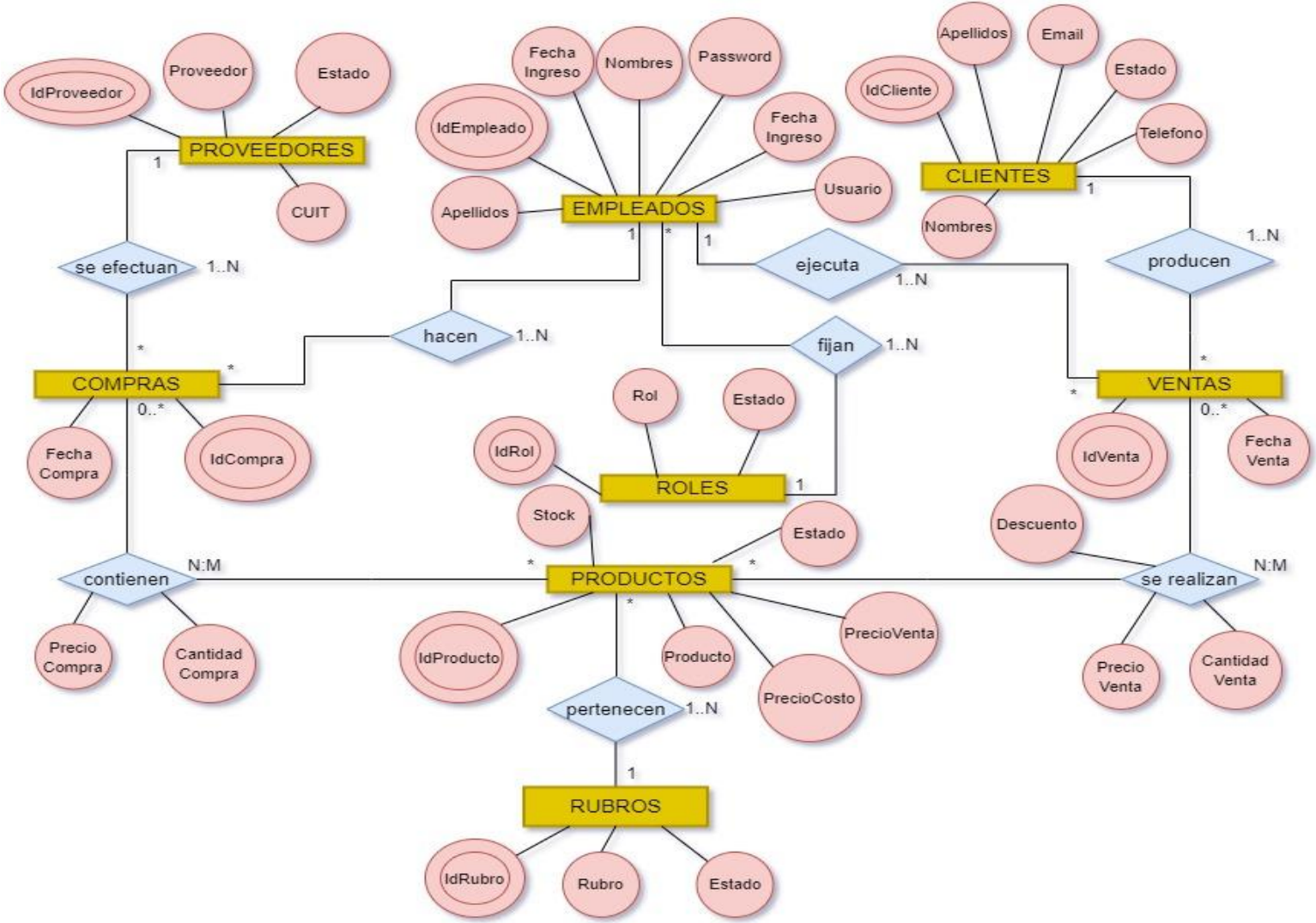
link repositorio: https://github.com/jgromerou/CursoSql

Docs Google:
https://docs.google.com/presentation/d/18Hx17igBV0MVZQlx9tz5lp6NWh7wvpzFr0_rXOoxhKc/edit?usp=sharing

Archivos: .sql,.pdf se encuentran en la carpeta de Entrega2-RomeroUro

DESCRIPCIÓN DE LA TEMÁTICA

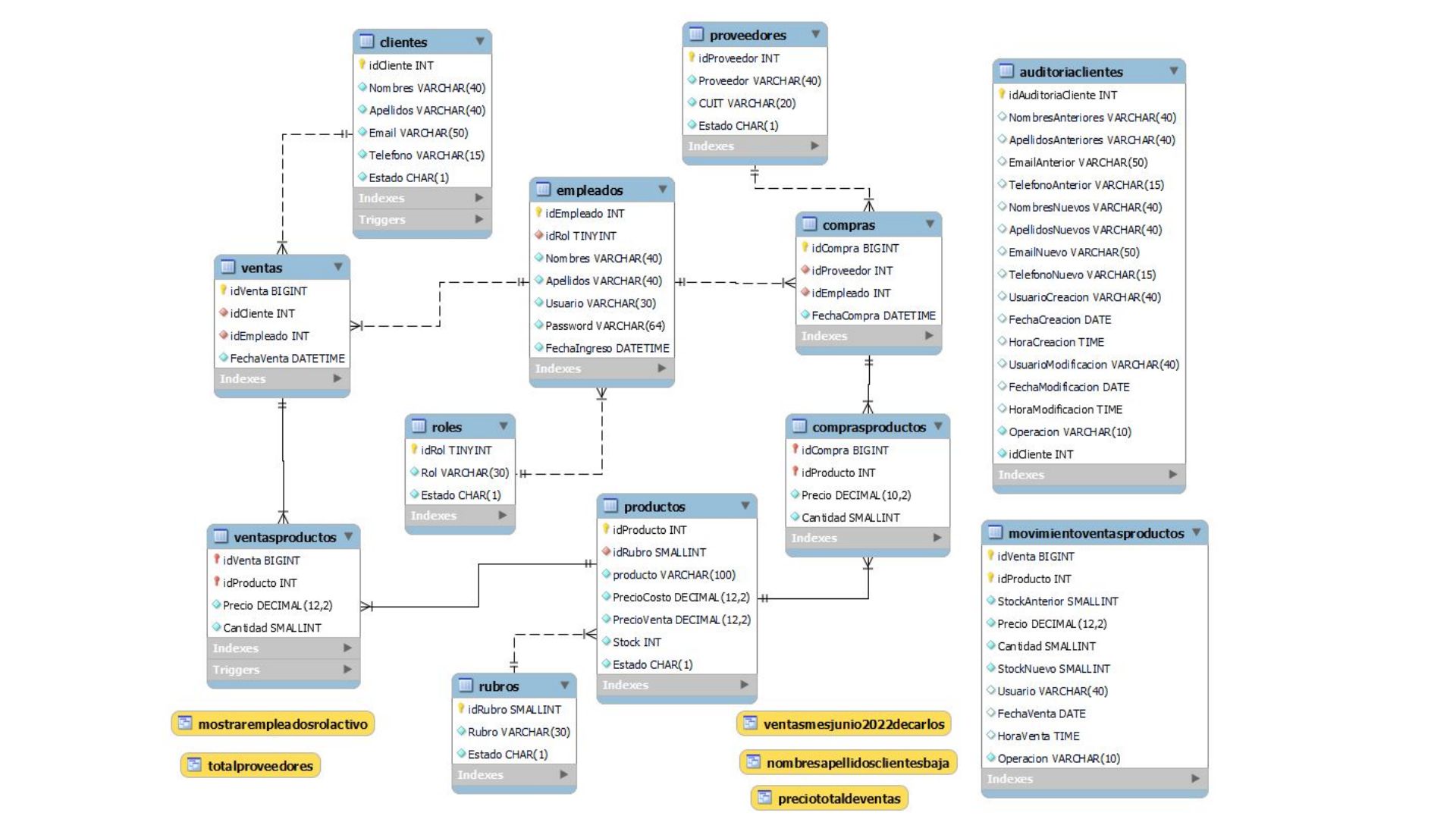
Se trata de un sistema de gestión de compra y venta de productos de Computación, donde sus empleados son quienes se encargan de cargar datos como por ejemplo las compras y ventas de los insumos, además cuenta con un control de stock y cada empleado tiene su propio usuario con un determinado rol(comprador, vendedor, administrador).


IdProveedor
Proveedor
Estado
PROVEEDORES
CUIT
1
se efectuan
1..N
*
COMPRAS
*
0..*
Fecha Compra
IdCompra
hacen
1..N
Fecha Ingreso
Nombres
Password
IdEmpleado
Fecha Ingreso
Apellidos
EMPLEADOS
Usuario
1
*
1
ejecuta
1..N
fijan
1..N
Rol
Estado
IdRol
ROLES
1
Stock
Estado
*
*
contienen
N:M
PRODUCTOS
IdProducto
Producto
PrecioVenta
PrecioCosto
Precio Compra
Cantidad Compra
pertenecen
1..N
1
RUBROS
IdRubro
Rubro
Estado
Apellidos
Email
IdCliente
Estado
Telefono
CLIENTES
Nombres
1
1..N
producen
*
VENTAS
*
0..*
IdVenta
Fecha Venta
Descuento
se realizan
N:M
Precio Venta
Cantidad Venta

Tablas del sistema de compra venta de productos de Computación - Romero Uro					
Roles: roles de los empleados.					
idRol (PK)	TINYINT	id de rol del empleado			
Rol	VARCHAR(30)	Rol del empleado			
Estado	CHAR(1)	Estado de Rol del empleado ->		A: alta, B: baja	
Empleados: son los únicos que tienen accesos al sistema con un determinado rol					
idEmpleado (PK)	INT	ID del empleado			
idRol (FK)	TINYINT	ID de rol			
Nombres	VARCHAR(40)	Nombre/s del empleado			
Apellidos	VARCHAR(40)	Apellido/s del empleado			
Usuario	VARCHAR(30)	Usuario de ingreso al sistema			
Password	VARCHAR(64)	Password de ingreso al sistema			
FechaIngreso	DATETIME	Fecha de ingreso al sistema			
Rubros: esta tabla contiene los rubros de los productos					
idRubro (PK)	SMALLINT	ID del rubro			
Nombre	VARCHAR(30)	Nombre de rubro			
Estado	CHAR(1)	Estado de rubro	A: alta, B: baja		
Productos: contiene los productos de Computación.					
idProducto (PK)	INT	ID del producto			
idRubro (FK)	SMALLINT	ID del rubro de producto			
Producto	VARCHAR(40)	Nombre del producto			
PrecioCosto	DECIMAL(12,2)	Precio Costo del producto			
PrecioVenta	DECIMAL(12,2)	Precio Venta del Producto			
Stock	INT	Stock			
Estado	CHAR(1)	Estado del producto ------->		A: alta, B: baja	

Proveedores: Esta tabla almacena los proveedores					
idProveedor (PK)	INT	ID del proveedor			
CUIT	VARCHAR(20)	CUIT del proveedor			
Proveedor	VARCHAR(40)	Nombre del proveedor			
Estado	CHAR(1)	Estado del proveedor			
Compras: se registran las compras de los productos de Computación					
idCompra (PK)	BIGINT	ID de compra			
idProveedor (FK)	INT	ID de proveedor			
idEmpleado (FK)	INT	ID de Empleado			
FechaCompra	DATETIME	Fecha de Compra			
ComprasProductos: Relación entre tablas compras y productos					
idCompra (FK)	BIGINT	ID de compra			
idProducto (FK)	INT	ID de producto			
Precio	DECIMAL(12,2)	Precio de Compra de un producto			
Cantidad	SMALLINT	Cantidad de producto/s de una compra			
Clientes: Esta tabla almacena los clientes					
idCliente (PK)	INT	ID del cliente			
Nombres	VARCHAR(40)	Nombre/s de Cliente			
Apellidos	VARCHAR(40)	Apellido/s de Cliente			
Email	VARCHAR(50)	Email del cliente			
Telefono	VARCHAR(15)	Telefono del cliente			
Estado	CHAR(1)	Estado de cliente			
Ventas: se registran las ventas de los productos de Computación					
idVenta (PK)	BIGINT	ID de venta			
idCliente (FK)	INT	ID de cliente			

idEmpleado (FK)	INT	ID de Empleado			
FechaVenta	DATETIME	Fecha de Venta			
VentasProductos: Relación entre tablas ventas y productos					
idVenta (FK)	BIGINT	ID de Venta			
idProducto (FK)	INT	ID de Producto			
Precio	DECIMAL(12,2)	Precio de producto de una venta			
Cantidad	SMALLINT	Cantidad de producto/s de una venta			


clientes
idCliente INT
Nombres VARCHAR(40)
Apellidos VARCHAR(40)
Email VARCHAR(50)
Telefono VARCHAR(15)
Estado CHAR(1)
Indexes
Triggers
proveedores
idProveedor INT
Proveedor VARCHAR(40)
CUIT VARCHAR(20)
Estado CHAR(1)
Indexes
auditoriaclientes
idAuditoriaCliente INT
NombresAnteriores VARCHAR(40)
ApellidosAnteriores VARCHAR(40)
EmailAnterior VARCHAR(50)
TelefonoAnterior VARCHAR(15)
NombresNuevos VARCHAR(40)
ApellidosNuevos VARCHAR(40)
EmailNuevo VARCHAR(50)
TelefonoNuevo VARCHAR(15)
UsuarioCreacion VARCHAR(40)
FechaCreacion DATE
HoraCreacion TIME
UsuarioModificacion VARCHAR(40)
FechaModificacion DATE
HoraModificacion TIME
Operacion VARCHAR(10)
idCliente INT
Indexes
empleados
idEmpleado INT
idRol TINYINT
Nombres VARCHAR(40)
Apellidos VARCHAR(40)
Usuario VARCHAR(30)
Password VARCHAR(64)
FechaIngreso DATETIME
Indexes
compras
idCompra BIGINT
idProveedor INT
idEmpleado INT
FechaCompra DATETIME
Indexes
ventas
idVenta BIGINT
idCliente INT
idEmpleado INT
FechaVenta DATETIME
Indexes
roles
idRol TINYINT
Rol VARCHAR(30)
Estado CHAR(1)
Indexes
comprasproductos
idCompra BIGINT
idProducto INT
Precio DECIMAL(10,2)
Cantidad SMALLINT
Indexes
productos
idProducto INT
idRubro SMALLINT
producto VARCHAR(100)
PrecioCosto DECIMAL(12,2)
PrecioVenta DECIMAL(12,2)
Stock INT
Estado CHAR(1)
Indexes
ventasproductos
idVenta BIGINT
idProducto INT
Precio DECIMAL(12,2)
Cantidad SMALLINT
Indexes
Triggers
movimientoventasproductos
idVenta BIGINT
idProducto INT
StockAnterior SMALLINT
Precio DECIMAL(12,2)
Cantidad SMALLINT
StockNuevo SMALLINT
Usuario VARCHAR(40)
FechaVenta DATE
HoraVenta TIME
Operacion VARCHAR(10)
Indexes
rubros
idRubro SMALLINT
Rubro VARCHAR(30)
Estado CHAR(1)
Indexes
mostrarempleadosrolactivo
totalproveedores
ventasmesjunio2022decarlos
nombresapellidosclientesbaja
preciototaldeventas

Vistas

```
/*1° Vista Total de Proveedores*/
CREATE OR REPLACE VIEW TotalProveedores AS
    (SELECT COUNT(*) AS TotalProveedores FROM
Proveedores);
```

Tabla involucrada: Proveedores.

```
/*2° Vista: Nombres y Apellidos de Clientes dados de Baja */
CREATE OR REPLACE VIEW NombresApellidosClientesBaja AS
    (SELECT DISTINCT c.Nombres,c.Apellidos
FROM Clientes c WHERE c.Estado = 'B');
```

Tabla involucrada: Proveedores.

```
/*3° Vista Mostrar los Empleados con roles activos */
CREATE OR REPLACE VIEW MostrarEmpleadosRolActivo AS
    (SELECT *
    FROM   Empleados
    WHERE  idRol IN (SELECT idRol FROM Roles WHERE
Estado = 'A')
    ORDER BY Empleados.idEmpleado);
```

Tabla involucrada: Roles.

```
/*4° Vista Obtener el total de las ventas por el rango de fechas y por el nombre de Empleados*/
/*Por ejemplo: Ventas del mes de Junio del año 2022 de los empleados llamados Carlos */
CREATE OR REPLACE VIEW VentasMesJunio2022deCarlos AS
        (SELECT
        v.FechaVenta AS FechaVenta,
        v.idVenta AS idVenta,
        v.idEmpleado AS NroEmpleado,
        e.Nombres AS Nombres,
        e.Apellidos AS Apellidos
    FROM
        (Ventas v
        JOIN Empleados e
                    ON ((v.idEmpleado = e.idEmpleado)))
    WHERE
        ((v.FechaVenta BETWEEN '20220601' AND '20220630')
            AND (e.Nombres LIKE '%Carlos%'))
    ORDER BY v.FechaVenta);
```

Tablas involucradas: Ventas, Empleados.

```
/*5° Vista Precio Total por Venta*/
CREATE OR REPLACE VIEW PrecioTotaldeVentas AS
     (SELECT
        v.idVenta AS idVenta,
        SUM(vp.Precio) AS PrecioTotal
     FROM
        (Ventas v
        JOIN VentasProductos vp
              ON (v.idVenta = vp.idVenta))
     GROUP BY v.idVenta);
```

Tablas involucradas: Ventas, VentasProductos.

Funciones

```
/*1° Función: Para saber el Estado de un Proveedor*/
DELIMITER $$
CREATE FUNCTION saberEstadoProveedor(pnombre char(40))
RETURNS CHAR(255)
READS SQL DATA
BEGIN
        DECLARE cantidad int DEFAULT 0;
    SET cantidad = (SELECT count(*) FROM Proveedores WHERE
Proveedor=pnombre AND Estado='A');
    IF(cantidad=1) THEN
                RETURN CONCAT('El proveedor ',pnombre,' está activo');
        ELSE
                RETURN CONCAT('El proveedor ',pnombre,' no está activo');
        END IF;
END $$
```

Tabla involucrada: Proveedores.

```
/*2° Funcion: Para saber cuantas ventas hizo un/a empleado/a por su
nombre */
DELIMITER $$
CREATE FUNCTION saberVentasEmpleadoporsuNombre(pnombre
char(40))
RETURNS CHAR(255)
READS SQL DATA
BEGIN
      DECLARE cantidad int DEFAULT 0;
    SET cantidad = (SELECT count(*) FROM Ventas v
                  INNER JOIN Empleados e
                  ON v.idEmpleado = e.idEmpleado
                  WHERE e.Nombres=pnombre);
            RETURN CONCAT('',pnombre,' realizó ',cantidad,' ventas');
END $$
```

Tabla involucrada: Ventas,Empleados.

Stored Procedure

```
/*1° Stored Procedure: Permite indicar a través de un parámetro el campo de ordenamiento de la tabla Empleados
y mediante un segundo parámetro, si el orden es Ascendente o Descendente.*/
/*ASC o DESC*/
DELIMITER $$
CREATE DEFINER=`root`@`%` PROCEDURE `sp_get_empleados_ordenamiento`(
	IN field CHAR(20),
    IN orden CHAR(4)
)
BEGIN
	IF field <>  '' THEN
		SET @rubro_order = concat('ORDER BY ',field,' ',orden,'');
	ELSE
		SET @rubro_order = '';
	END IF;
    SET @clausula = concat('SELECT * FROM Empleados ',@rubro_order);
	PREPARE runSQL FROM @clausula;
    EXECUTE runSQL;
    DEALLOCATE PREPARE runSQL;
END$$
```

Beneficio: Permite un mejor ordenamiento de la tabla para facilitar la búsqueda de los empleados por orden alfabético.

```
/*2° Stored Procedure: Permite ingresar un nuevo rubro*/
/*Parámetros a ingresar son: Rubro, Estado(A:Alta, B:Baja)*/
DELIMITER $$
CREATE DEFINER=`root`@`%` PROCEDURE `sp_insert_rubro`(
        IN prubro CHAR(30),
    IN pestado CHAR(1)
)
BEGIN
        DECLARE ultimoIDmasuno smallint;
        SET @clausula = concat('SELECT MAX(idRubro) FROM Rubros INTO @ultimoID');
        PREPARE runSQL FROM @clausula;
    EXECUTE runSQL;
    DEALLOCATE PREPARE runSQL;
    set ultimoIDmasuno=@ultimoID+1;
   SET @ftexsql=CONCAT('INSERT INTO Rubros (idRubro,Rubro,Estado) VALUES
(',"",CAST(ultimoIDmasuno AS CHAR),"',",prubro,",'",pestado,"')");
        PREPARE fordsql FROM @ftexsql;
    EXECUTE fordsql;
    DEALLOCATE PREPARE fordsql;
     SELECT 'Se ingresó satisfactoriamente el nuevo rubro de Nombre:',prubro,'con el ID:
',ultimoIDmasuno,";
END$$
```

Tabla involucrada: Rubros

```
/*3° Stored Procedure: Permite Eliminar un rubro por su ID*/
/*Parámetros a ingresar son: ID del rubro*/
DELIMITER $$
CREATE DEFINER=`root`@`%` PROCEDURE `sp_delete_rubro`(
      IN pidrubro CHAR(8)
)
BEGIN
   SET @ftexsql=CONCAT('DELETE FROM Rubros WHERE
idRubro=(',pidrubro,')');
      PREPARE fordsql FROM @ftexsql;
    EXECUTE fordsql;
    DEALLOCATE PREPARE fordsql;
     SELECT 'Se eliminó satisfactoriamente el rubro de ID:',pidrubro,'';
END$$
```

Tabla involucrada: Rubros

Triggers

```
/*1° Trigger: Permite controlar que la cantidad del producto a vender no supere la cantidad de Stock y lance una señal de error*/
/*Y sino se hace el decremento en Stock*/
/*Guardar nueva venta en la tabla movimientoVentasProductos*/

CREATE TRIGGER BEF_INS_ventasproductos_movimientoVentasProductos
	BEFORE INSERT
	ON VentasProductos
	FOR EACH ROW
	BEGIN
	/*DECLARACION DE VARIABLES*/
	DECLARE _cantidadviejo INTEGER;
	DECLARE _cantidadnueva INTEGER;
	SELECT Stock INTO _cantidadviejo FROM Productos WHERE idProducto = new.idProducto;
	/*SET Y CONSULTAS*/
	SET _cantidadnueva = _cantidadviejo - new.Cantidad;
	IF (_cantidadnueva<0) then
	   SIGNAL SQLSTATE '45000'
	   SET MESSAGE_TEXT = 'No existe esa cantidad de productos en el stock';
	        ELSE
	/*MODIFICAR VALOR DE STOCK DEL PRODUCTO*/
	UPDATE Productos SET Stock=_cantidadviejo - new.Cantidad WHERE idProducto = new.idProducto;
	/*GUARDAR MOVIMIENTO DE LA VENTA DE PRODUCTO*/
	INSERT INTO MovimientoVentasProductos(idVenta,idProducto,StockAnterior,Precio,Cantidad,
	StockNuevo,Usuario,FechaVenta,HoraVenta,Operacion)
	VALUES (NEW.idVenta,NEW.idProducto,_cantidadviejo,NEW.Precio,NEW.Cantidad,
	_cantidadnueva,CURRENT_USER,CURRENT_DATE(),CURRENT_TIME(),'NUEVAVENTA');
	END if;
END$$
```

```
/*2° Trigger: Permite auditar a un nuevo cliente en la*/
/*tabla AuditoriaClientes*/

DROP TRIGGER IF EXISTS AFT_INS_Clientes_AuditoriaClientes;
DELIMITER $$
CREATE TRIGGER AFT_INS_Clientes_AuditoriaClientes
        AFTER INSERT
        ON Clientes
        FOR EACH ROW
        BEGIN
        INSERT INTO
AuditoriaClientes(NombresNuevos,ApellidosNuevos,EmailNuevo,

TelefonoNuevo,UsuarioCreacion,FechaCreacion,HoraCreacion,Operacion,idCliente)
        VALUES
(NEW.Nombres,NEW.Apellidos,NEW.Email,NEW.Telefono,CURRENT_USER,
        CURRENT_DATE(),CURRENT_TIME(),'NUEVO',NEW.idCliente);
END$$
```

```
/*3° Trigger: Permite auditar modificaciones a un cliente en la*/
/*tabla AuditoriaClientes*/

DROP TRIGGER IF EXISTS BEF_UPD_Clientes_AuditoriaClientes;
DELIMITER $$
CREATE TRIGGER BEF_UPD_Clientes_AuditoriaClientes
	BEFORE UPDATE
	ON Clientes
	FOR EACH ROW
	BEGIN
	INSERT INTO AuditoriaClientes(NombresAnteriores,ApellidosAnteriores,
	TelefonoAnterior,EmailAnterior,
	NombresNuevos,ApellidosNuevos,EmailNuevo,
	TelefonoNuevo,UsuarioModificacion,FechaModificacion,HoraModificacion,
	Operacion,idCliente)
	VALUES (OLD.Nombres,OLD.Apellidos,NEW.Telefono,NEW.Email,
	NEW.Nombres,NEW.Apellidos,NEW.Email,NEW.Telefono,CURRENT_USER,
	CURRENT_DATE(),current_time(),'EDITAR',NEW.idCliente);
END$$
```

```
/*4° Trigger: Permite auditar el borrado de un cliente en la*/
/*tabla AuditoriaClientes*/

DROP TRIGGER IF EXISTS AFT_DEL_Clientes_AuditoriaClientes;
DELIMITER $$
CREATE TRIGGER AFT_DEL_Clientes_AuditoriaClientes
        AFTER DELETE
        ON Clientes
        FOR EACH ROW
        BEGIN
        INSERT INTO
AuditoriaClientes(NombresAnteriores,ApellidosAnteriores,
        Operacion,idCliente)
        VALUES (OLD.Nombres,OLD.Apellidos,
        'BORRAR',OLD.idCliente);
END$$
```