UNFV | UNIVERSIDAD NACIONAL FEDERICO VILLAREAL

Facultad de Ingeniería Electrónica e Informática

Escuela profesional: Ingeniería Informática

Implementación de un sistema de Gestión de Base de Datos

INTEGRANTES:

- Palacios Huasupoma, Zummy
- Quiroz Fernández, Sam Bruce Elías
- Ramirez Anampa, Estefani
- Roldan Candiotti, Johan
- Silva Rios, Sheyson Felito

PROFESOR:

Mg. Ivan Crispin Sanchez

Lima, 2024

INDICE

I
PLANEAMIENTO Y ADMINISTRACION DEL PROYECTO

I. PLANEAMIENTO Y ADMINISTRACION DEL PROYECTO

1.1. NEGOCIO

La Farmacia "SaludPrime Farmacia" es una empresa que está ubicada en la Av. San christobal N°612 del distrito de San juan de Lurigancho, Lima, Perú. Es de propiedad del Químico farmacéutico Kevin López Ramos; cuya principal actividad comercial es la venta de medicamentos y de productos de farmacia (productos de aseo personal, cosméticos, entre otros). La actividad comercial que realiza esta Farmacia es principalmente al por menor, por el momento esta actividad se desarrolla manualmente.

1.2. PROBLEMA DEL NEGOCIO

En la actualidad, el procedimiento de control de existencias implica registrar las entradas y salidas de productos en una hoja de cálculo Excel y en un registro físico. Esto significa que, para identificar la escasez de un artículo, solo es posible realizar un conteo a mano. La revisión del inventario se lleva a cabo trimestralmente, lo que resulta en datos desactualizados durante ese intervalo. Por otro lado, la venta de productos se realiza de manera manual, buscando y seleccionando los artículos. Si un producto está agotado, la única manera confiable de confirmarlo es inspeccionando las estanterías. El inventario pierde precisión conforme avanzan las semanas, hasta que finalmente se actualiza.

Además, la supervisión de la fecha de vencimiento de los productos presenta ciertas limitaciones en términos de eficiencia. Esta metodología no logra identificar de manera adecuada los artículos que están cerca de su fecha límite, lo que dificulta la oportunidad para solicitar su reemplazo de manera oportuna. Esto puede generar pérdidas económicas para la farmacia al no gestionar de forma eficaz los productos próximos a caducar.

Los procesos de inventarios y ventas de los productos farmacéuticos de forma manual conllevan los siguientes riesgos:

- Riesgo de expiración de los productos, generando así pérdidas para la farmacia y afectando la rentabilidad de esta.
- Riesgo de una deficiente atención al cliente, generando una demora innecesaria en la atención.
- Riesgo de una deficiente toma de decisiones al no tener un inventario actualizado.

Por lo antes expuesto se plantea desarrollar un sistema web para la gestión de los procesos de inventarios y venta de productos, que permita tener un mejor registro y un control adecuado del stock de productos, reducir el tiempo de atención de los clientes, así como, llevar un control más eficiente de las ventas.

1.3. OBJETIVOS DEL PROYECTO

1.3.1. Objetivo General.

Determinar en qué medida la implementación del Sistema Web de Farmacias influye en la gestión de los procesos de inventarios y ventas de productos de la Farmacia “SaludPrime Farmacia".

1.3.2. Objetivos Específicos.

- Determinar en qué medida la implementación del Sistema Web de Farmacias influye en la gestión del proceso de inventarios de productos de la Farmacia "SaludPrime Farmacia".
- Determinar en qué medida la implementación del Sistema Web de Farmacias influye en la gestión del proceso de ventas de productos de la Farmacia "SaludPrime Farmacia".

1.4. JUSTIFICACION

La informática tiene la capacidad de automatizar diversas tareas administrativas. Los documentos físicos relacionados con el personal pueden ser ingresados sin dificultad en un sistema informático apropiado, lo que permite respaldarlos y mantenerlos actualizados de manera eficiente. La información almacenada en el sistema informático puede ser impresa y accedida en cualquier momento. Esto resulta especialmente beneficioso para el administrador, ya que le proporciona las herramientas necesarias para gestionar eficazmente al personal y calcular el presupuesto requerido mensualmente en base a sus horas de trabajo.

El de un sistema de información dentro de una organización tiene las siguientes ventajas:

- Brinda ventajas competitivas y valor añadido.
- Proporciona a los usuarios más y mejor información en tiempo real.
- Reduce errores, tiempo y recursos redundantes.
- Permite comparar los resultados obtenidos con los objetivos establecidos, para su control y evaluación.
- Posibilita una mejor relación con los clientes.

Debido a los problemas originados por la ineficiente administración de inventarios, la gestión de la fecha de vencimiento de productos y la demora significativa en la búsqueda manual previa a la venta, se ha identificado un riesgo económico sustancial para la farmacia. Con el propósito de abordar esta problemática, se plantea la creación de un Sistema Web para Farmacias que permita llevar a cabo un control efectivo de los inventarios y las ventas.

El Sistema Web para Farmacias se concebirá para mejorar la gestión de los inventarios, proporcionando información precisa, organizada y actualizada. Esto, a su vez, habilitará la toma de decisiones informadas para maximizar las ganancias y reducir las pérdidas en el entorno de la farmacia. En cuanto a las ventas, el sistema contribuirá a brindar un mejor servicio a los clientes al agilizar el proceso de búsqueda de productos.

1.5. BENEFICIOS

Implementar un sólido sistema de base de datos en una farmacia conlleva numerosos beneficios que impactan directamente en la eficiencia, precisión y calidad del servicio que se ofrece. Algunos de estos beneficios son:

- **Gestión de inventario optimizada:** Un sistema de base de datos permite rastrear y administrar con precisión el stock de productos, evitando la escasez o el exceso de inventario. Esto reduce pérdidas y garantiza que los productos estén disponibles cuando se necesiten.
- **Control de caducidad efectivo:** El sistema de base de datos puede alertar automáticamente sobre los productos próximos a vencer, facilitando su reemplazo o descarte a tiempo. Esto minimiza el riesgo de vender productos caducados y reduce pérdidas.
- **Procesos de ventas eficientes**: El acceso rápido a la información en la base de datos agiliza el proceso de búsqueda y venta de productos. Los empleados pueden localizar productos de manera rápida y precisa, mejorando la atención al cliente y reduciendo el tiempo de espera.
- **Toma de decisiones informadas**: Una base de datos bien organizada proporciona datos en tiempo real sobre ventas, tendencias y movimientos de inventario. Esto permite a los administradores tomar decisiones basadas en información actualizada y precisa.
- **Mejora en la atención al cliente:** Al conocer el historial de compras y preferencias de los clientes, la farmacia puede ofrecer recomendaciones personalizadas y mejorar la experiencia de compra.
- **Análisis y reportes detallados:** El sistema de base de datos permite generar informes y análisis detallados sobre ventas, inventario y otros aspectos. Estos informes ayudan a identificar oportunidades de mejora y a planificar estrategias futuras.

- **Seguridad y respaldo de datos:** Una base de datos bien implementada ofrece opciones de respaldo y seguridad de datos, protegiendo la información crítica de la farmacia ante posibles pérdidas o amenazas.

II

ANTECEDENTES

II. ANTECEDENTES

2.1. ANTECEDENTES DE INVESTIGACION

2.1.1. Antecedentes Internacional

Investigando diversas fuentes relacionadas al problema que hemos planteado, se encontraron los siguientes antecedentes **internacionales**, entre otras similares:

- **Parra Medina, Juan Esteban (2020)**: En su tesis titulado "Diseño de un sistema de información para el control de inventario de medicamentos en farmacias colombianas."

  Esta tesis tiene como propósito solucionar las pérdidas monetarias y optimizar el control de la base de datos para todas las farmacias colombianas queriendo dar un mejor manejo de su base de datos a los encargados de cada establecimiento. La finalidad de esta tesis es tener registros de cada medicamento por caducidad o el medicamento se ha agotado, la metodología que trabajaron se desarrolló por pasos cómo una rigurosa investigación de los datos, análisis y diseño y por último una evaluación de validación. (Medina, 2020)

- **Anthony Sebastian Peña Guachimboza (2022)**: En su tesis titulado "Sistema Web aplicando Vue.Js y Laravel para la Gestión de Comercialización de Productos en la Farmacia Farmared's "

  Esta tesis busca facilitar el control de los procesos de ventas y compras de medicamentos utilizando herramientas tecnológicas ya que el control que la farmacia lleva es de manera manual, queriendo brindar unas actualizaciones en el sistema y brindando seguridad en la información de los productos almacenados. Las herramientas tecnológicas que utiliza son el Laravel y el Vue.js que se encargaran del back-end y el front-end, para su sistema de base de datos utilizará MySQL permitiendo hacer consultas y realizar operaciones para cada tabla; como metodología utilizó Extreme Programming (XP) ya que controla las actividades al realizar las historias de usuario. (Guachimboza, 2022)

2.1.2. Antecedentes Nacionales

Investigando diversas fuentes relacionadas al problema que hemos planteado, se encontraron los siguientes antecedentes **nacionales**, entre otras similares:

- **Ruiz Navarro, Maryori Katherine (2019)**: En su tesis titulado " Análisis, diseño e implementación de un sistema de control de inventarios para la farmacia "Danafarma". "

  Esta tesis busca presentar una alternativa de solución a ciertos problemas encontrados como el inadecuado manejo de inventarios y tener una base de datos desactualizada, como tal esta investigación pretende dar beneficio a los usuarios de dicha farmacia. Sabiendo la situación actual de la farmacia, se quiere elaborar un diseño para un sistema de gestión de inventario aplicando como metodología RUP, pasando por procesos de desarrollo como el análisis, implementación y diseño. (Ruiz Navarro, 2019)

- **Melgarejo Rocca, Jose Luis (2018):** En su tesis titulado "Implementación de un sistema de información Web de control de ventas y almacén para la farmacia Bazán – Chimbote "

  La presente tesis se desarrolló para establecer una implementación de un sistema web que ayude al control de ventas e inventario de medicamentos permitiendo el incremento de ventas, ofreciendo el mejor servicio de atención al cliente y disminuyendo el tiempo de búsqueda de cada medicamento. toda información de los medicamentos se conserva mediante una base de datos dinámica en donde se podrá recuperar y actualización de información mediante consultas, como también definir ciertas restricciones de seguridad para evaluar actualizaciones en la base de datos. (Rocca, 2018)

III

MODELO DE NEGOCIACION

III. MODELO DE NEGOCIACION

3.1. DESCRIPCION DE LOS PROCESOS DEL NEGOCIO ESPECIFICOS A MODELAR

3.1.1. Descripción de los Proceso de Compra de Producto(s):

El proceso de compra de producto(s) inicia desde la solicitud de los productos según la disponibilidad, si hay confirmación de stock el usuario realiza la reserva, asimismo el proveedor gestiona la reserva, para posteriormente responder a la reserva y tomando la decisión de confirmar o anular la reserva. *(figura 1)*

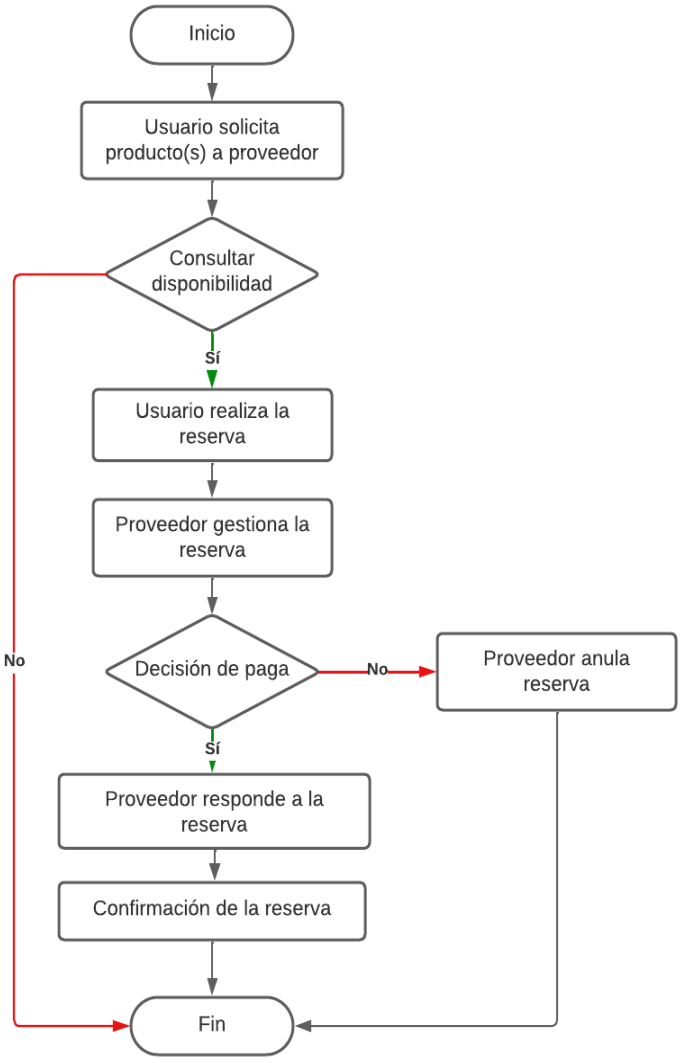


Figura 1: Descripción del Proceso de Compra de Productos

3.1.2. Descripción de los Proceso de Venta de Producto(s):

El siguiente proceso inicia con la consulta del producto al empleado de parte del cliente, el empleado procede a realizar la consulta de stock, si hay confirmación de stock, el empleado gestiona el stock y realiza el cobro del producto. Posteriormente generando la factura de la venta generada. *(figura 2)*

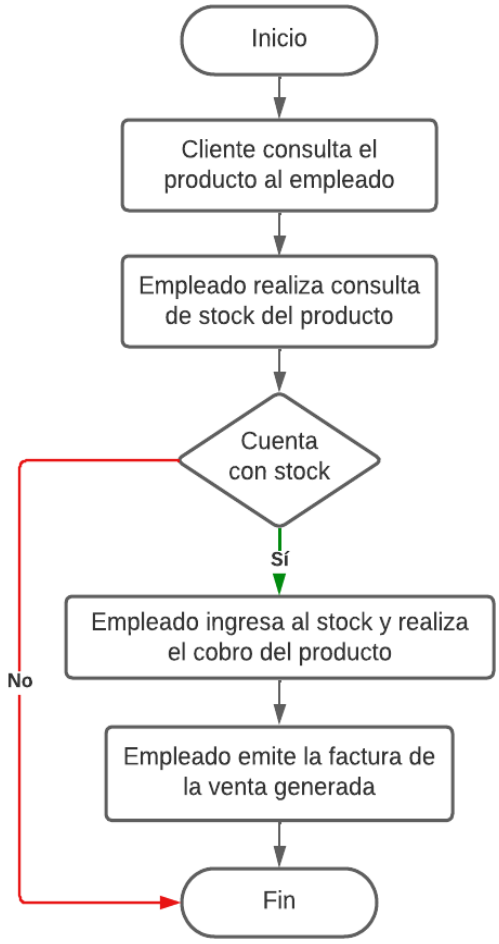
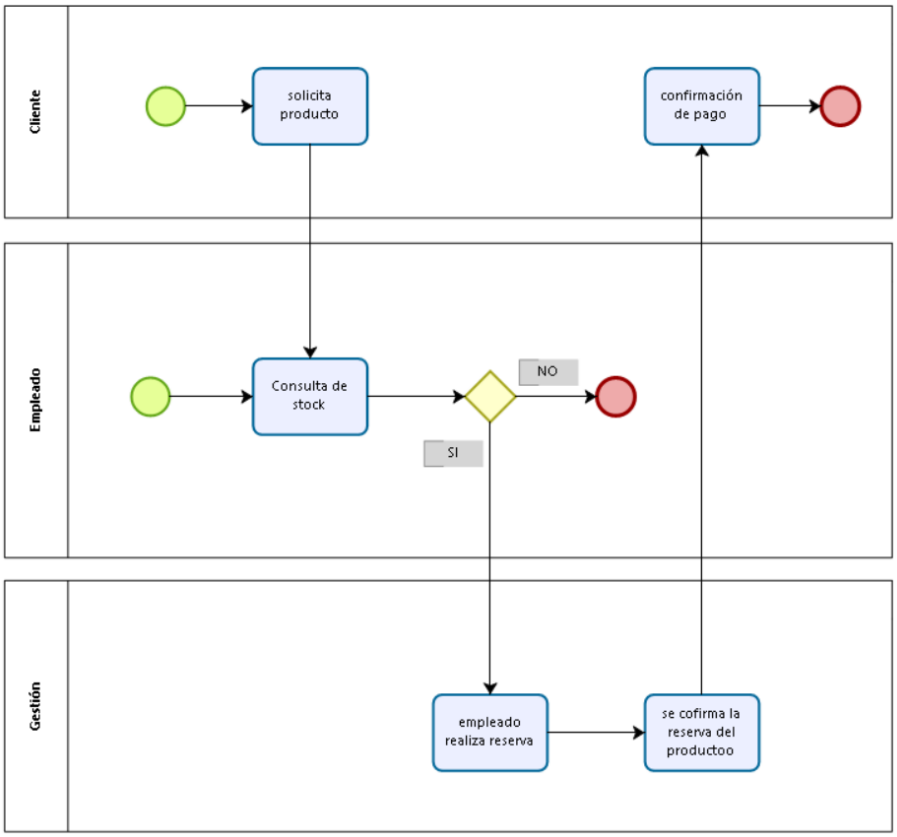
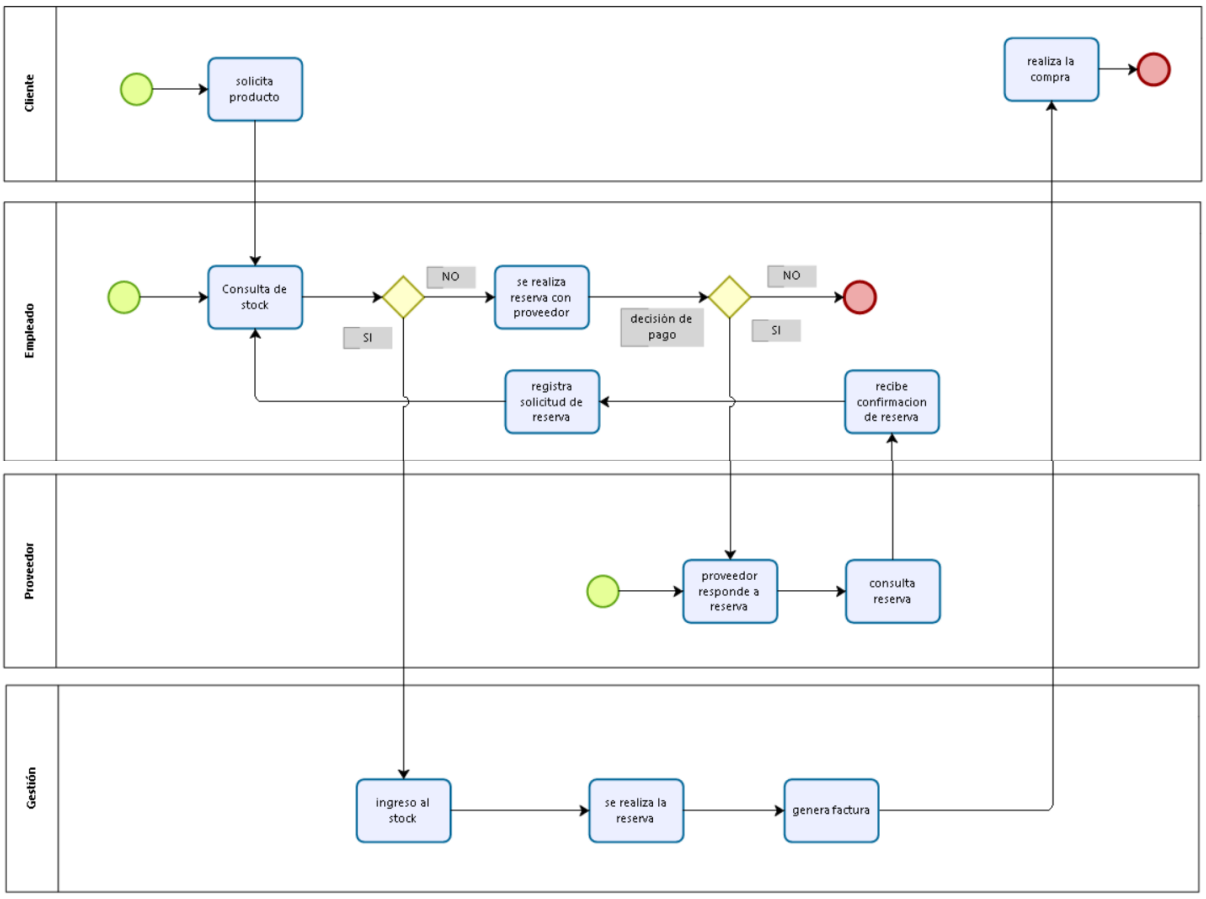


Figura 2: Descripción del Proceso de Venta de Productos

3.2. AS-IS DEL PROCESO DE COMPRA Y VENTA



3.3. TO-BE DEL PROCESO DE COMPRA Y VENTA



IV

REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

IV. ANALISIS DEL REQUERIMIENTO

4.1. REQUERIMIENTOS FUNCINALES

4.1.1. RF1: Gestión de Proveedores

- El sistema debe permitir la creación, actualización y eliminación de registros de proveedores, incluyendo información como nombre, dirección y teléfono.

4.1.2. RF2: Control de Medicamentos

- Debe ser posible agregar, editar y eliminar medicamentos en el sistema, especificando su proveedor, nombre, descripción, precio de venta, stock disponible y fecha de caducidad.
- Se debe implementar un procedimiento almacenado para ajustar automáticamente el stock de un medicamento después de una venta.

4.1.3. RF3: Registro de Clientes

- El sistema debe permitir la administración de registros de clientes, incluyendo su nombre, dirección y número de teléfono.

4.1.4. RF4: Administración de Empleados

- Se debe poder agregar, actualizar y eliminar registros de empleados, con detalles como nombre, cargo, salario y asociación a un usuario.

4.1.5. RF5: Gestión de Ventas

- El sistema debe registrar las ventas realizadas, incluyendo información sobre el cliente, empleado, fecha y monto total de la venta.
- Se debe permitir la cancelación de una compra, lo que implicará la reversión de las actualizaciones en el inventario y los registros asociados.

4.1.6. RF6: Detalles de Ventas

- Se debe poder registrar los detalles de cada venta, incluyendo medicamentos vendidos, cantidades y subtotales.

4.1.7. RF7: Historial de Pagos

- El sistema debe permitir el registro de pagos asociados a cada venta, incluyendo método de pago, monto y fecha.
- Se debe implementar un trigger que actualice automáticamente el monto total de la venta cuando se registre un nuevo pago.

4.1.8. RF8: Autenticación de Usuarios

- Se debe implementar un sistema de autenticación de usuarios con roles diferenciados para empleados y administradores.

4.1.9. RF9: Reportes de Inventario

- El sistema debe generar informes periódicos sobre el estado del inventario, incluyendo detalles sobre medicamentos disponibles y su cantidad.

4.1.10. RF10: Seguimiento de Ventas

- El sistema debe ser capaz de generar reportes de ventas, con información detallada sobre las transacciones realizadas en un período específico.
- Se debe crear una vista que muestre las ventas totales por empleado en un período de tiempo determinado.

4.1.11. RF11: Alertas de Caducidad

- El sistema debe generar automáticamente alertas para los medicamentos que estén a punto de caducar en un plazo definido.
- Se debe enviar una notificación a los empleados encargados para que tomen medidas, como retirar los medicamentos del inventario o aplicar descuentos especiales.

4.1.12. RF12: Historial de Cambios

- El sistema debe registrar y mantener un historial de cambios realizados en los registros críticos, como medicamentos, proveedores y precios.
- Cada cambio debe ser registrado con detalles como fecha, usuario que realizó el cambio y la naturaleza de la modificación.

4.2. REQUERIMIENTOS NO FUNCIONALES

4.2.1. RNF1: Rendimiento y Tiempo de Respuesta

- El sistema debe garantizar un tiempo de respuesta en el rango de milisegundos o segundos, dependiendo de la complejidad y el volumen de datos.
- Las operaciones de consulta de información crítica, como el inventario y los precios de los medicamentos, deben ser especialmente rápidas.

4.2.2. RNF2: Copias de Seguridad (Backups)

- El sistema debe implementar un proceso automatizado de copias de seguridad regulares de la base de datos, incluyendo todos los datos y configuraciones críticas.
- Las copias de seguridad deben almacenarse en ubicaciones seguras y fuera del sitio, como almacenamiento en la nube o dispositivos de respaldo físicos.
- Debe existir la capacidad de realizar restauraciones desde las copias de seguridad en caso de pérdida de datos debido a fallos o errores no previstos.

4.2.3. RNF3: Escalabilidad y Capacidad Futura

- La base de datos debe ser escalable tanto en términos de almacenamiento como de capacidad de procesamiento para acomodar el crecimiento futuro de la farmacia y la expansión de la base de clientes.
- Se debe tener en cuenta la posibilidad de agregar nuevos tipos de datos y funcionalidades sin afectar el rendimiento.

4.2.4. RNF4: Seguridad y Privacidad de Datos

- La base de datos debe implementar medidas sólidas de seguridad, como encriptación de datos en reposo y en tránsito, para proteger la información sensible de clientes y empleados.
- Se deben establecer políticas de acceso y roles para garantizar que solo los usuarios autorizados puedan acceder y modificar los datos.

4.2.5. RNF5: Mantenibilidad

- El diseño de la base de datos debe ser modular y fácil de mantener, permitiendo futuras actualizaciones y cambios en el esquema sin afectar la integridad de los datos.

4.3. REQUERIMIENTOS DE IMPLEMENTACION

4.3.1.1. SOFTWARE:

- **Base de Datos: SQL Server 2019**

  La base de datos se implementará utilizando Microsoft SQL Server 2019 como sistema de gestión de bases de datos relacional. Se aprovecharán las características y mejoras de esta versión para garantizar un rendimiento óptimo y una administración eficiente de los datos.

- **Modelado de Datos: Erwin Data Modeler 2020**

  El diseño y modelado de la base de datos se realizará utilizando Erwin Data Modeler 2020. Esta herramienta permitirá crear y visualizar de manera eficiente el esquema de la base de datos, así como establecer relaciones y atributos de manera precisa.

4.3.1.2. HARDWARE:

- **Equipo de Desarrollo:**

  - Se requerirá un computador con un procesador Intel Core i5 de 10th generación o superior para asegurar un rendimiento fluido durante la implementación y desarrollo.
  - La memoria RAM será de al menos 8GB para manejar eficazmente las operaciones y consultas en la base de datos.
  - Se dispondrá de un disco duro de alta capacidad, con al menos 1TB de almacenamiento para alojar la base de datos y otros componentes.

III

DISEÑO

LOGICO

V. DISEÑO LOGICO: MODELO LOGICO

5.1. NORMALIZACION

La normalización es un principio utilizado en la creación de bases de datos, especialmente en las bases de datos relacionales, con el propósito de minimizar la redundancia y mejorar la eficiencia del almacenamiento de información.

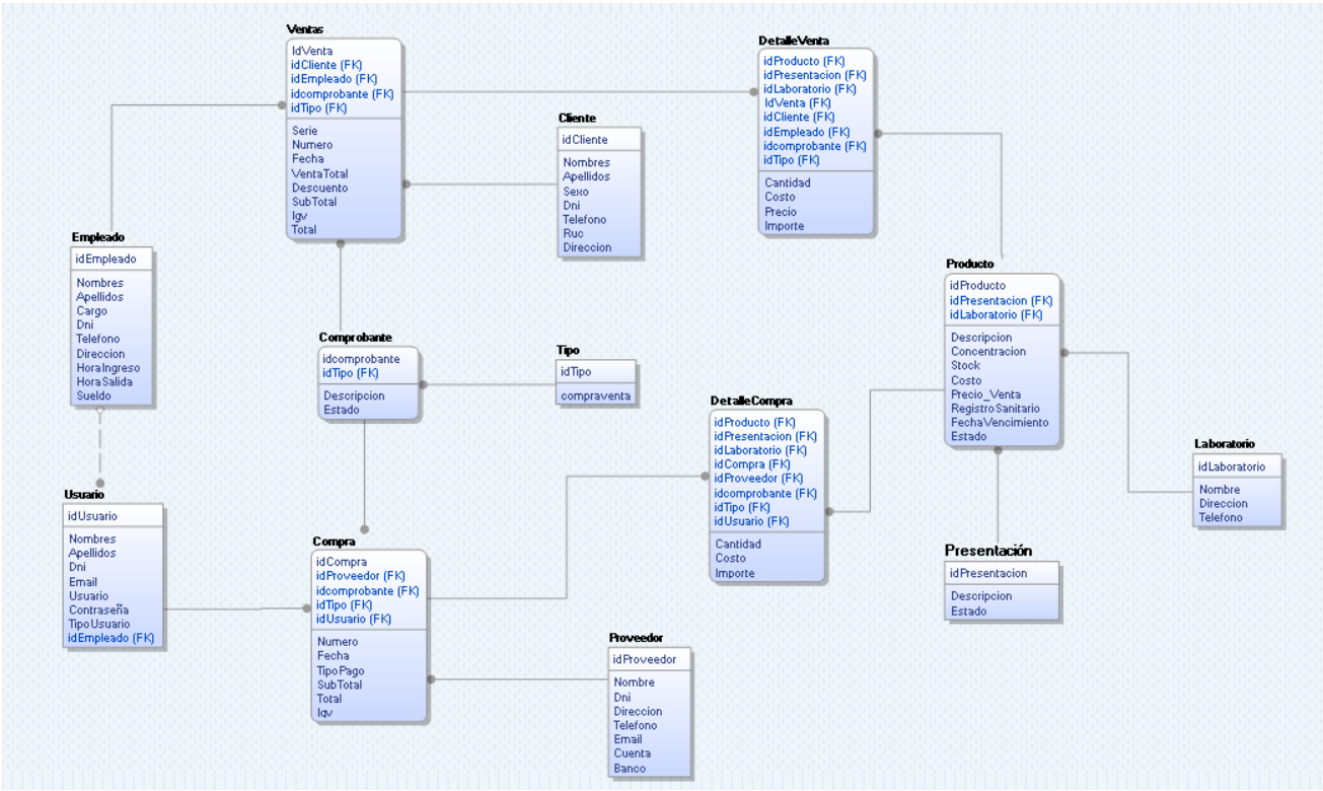
En este tipo de bases de datos, los datos se almacenan en registros dentro de tablas interconectadas mediante claves. Un registro está formado por múltiples campos que contienen valores y están organizados de acuerdo con atributos específicos en las columnas de la tabla.

MODELO LÓGICO BASE DATOS (NORMALIZADO)

Tabla 01	Tabla 02	Tipo de Relación	Tipo de Normalización	Justificación
cliente	ventas	Uno a Muchos	1ª Forma Normal (1FN)	Un cliente puede realizar varias ventas.
usuario	empleado	Uno a Uno	1ª Forma Normal (1FN)	Un usuario puede ser asignado a un empleado (o viceversa).
presentacion	producto	Uno a Muchos	1ª Forma Normal (1FN)	Una presentación puede tener varios productos.
laboratorio	producto	Uno a Muchos	1ª Forma Normal (1FN)	Un laboratorio puede tener varios productos.
producto	detalleventa	Uno a Muchos	1ª Forma Normal (1FN)	Un producto puede estar en varias ventas como detalle.
producto	detallecompra	Uno a Muchos	1ª Forma Normal (1FN)	Un producto puede estar en varias compras como detalle.
proveedor	compra	Uno a Muchos	1ª Forma Normal (1FN)	Un proveedor puede tener varias compras asociadas.
tipo	comprobante	Uno a Muchos	1ª Forma Normal (1FN)	Un tipo puede estar relacionado con varios comprobantes.
empleado	ventas	Uno a Muchos	1ª Forma Normal (1FN)	Un empleado puede realizar varias ventas.
comprobante	ventas	Uno a Muchos	1ª Forma Normal (1FN)	Un comprobante puede estar relacionado con varias ventas.

Tabla 01	Tabla 02	Tipo de Relación	Tipo de Normalización	Justificación
ventas	detalleventa	Uno a Muchos	1ª Forma Normal (1FN)	Una venta puede tener varios detalles de venta.
comprobante	compra	1 a muchos	1NF	Una compra puede tener un comprobante relacionado.
usuario	compra	1 a muchos	1NF	Un usuario puede realizar múltiples compras.
compra	detallecompra	1 a muchos	1NF	Una compra puede incluir múltiples detalles de productos.

5.2. COMPLETAR LAS RELACIONES Y ATRIBUTOS



5.3. DEFINIR INTEGRIDAD DE DOMINIO, DE ENTIDAD Y REFERENCIAL

5.3.1. DEFINICION DOMINIOS

5.3.1.1. CLIENTE

Nombre de Dominio	Significado	Definición del Dominio
idCliente	Identificador único para un estudiante	Entero
Nombre	Nombre completo del cliente	Carácter : Tamaño 50
Apellidos	Apellidos completos del cliente	Carácter : Tamaño 50
Sexo	El sexo del estudiante	Carácter : Tamaño 10
Dni	Identificador único del tipo documento del cliente	Carácter : Tamaño 10
Telefono	Teléfono del cliente	Entero
Ruc	El número de ruc del cliente	
Dirección	Dirección de residencia del cliente	Carácter : Tamaño 40

5.3.1.2. USUARIO

Nombre de Dominio	Significado	Definición del Dominio
idUsuario	Identificador único para un usuario	Entero
Nombre	Nombre completo del usuario	Carácter : Tamaño 50
Apellidos	Apellidos completos del usuario	Carácter : Tamaño 50
Dni	Identificador único del tipo documento del usuario	Carácter : Tamaño 10
Email	Correo electrónico del usuario	Carácter : Tamaño 40
Contraseña	Contraseña que identifica al usuario en el sistema	Entero
TipoUsuario	Cargo del usuario	Carácter : Tamaño 40

5.3.1.3. EMPLEADO

Nombre de Dominio	Significado	Definición del Dominio
idEmpleado	Identificador único para un empleado	Entero
Nombre	Nombre completo del empleado	Carácter : Tamaño 50
Apellidos	Apellidos completos del empleado	Carácter : Tamaño 50
Cargo	El sexo del empleado	Carácter : Tamaño 10
Dni	Identificador único del tipo documento del empleado	Carácter : Tamaño 10
Telefono	Teléfono del empleado	Entero
Dirección	Dirección de residencia del empleado	Carácter : Tamaño 40
HoraIngreso	Hora en la que el empleado ingresa	Hora
HoraSalida	Hora en la que el empleado sale	Hora
Sueldo	Sueldo que recibe un empleado	Decimal: tamaño 7 dígitos, 2 decimales
idUsuario	Identificador único para un usuario	Entero

5.3.1.4. PRESENTACION

Nombre de Dominio	Significado	Definición del Dominio
idPresentacion	Identificador único para una presentación	Entero
Descripcion	Descripción de la presentación del producto	Carácter : Tamaño 50
Estado	Estado que refleja la situación de la presentación	Carácter : Tamaño 50

5.3.1.5. LABORATORIO

Nombre de Dominio	Significado	Definición del Dominio
idLaboratorio	Identificador único para el laboratorio	Entero
Nombre	Nombre completo del laboratorio	Carácter : Tamaño 50
Direccion	Dirección del laboratorio	Carácter : Tamaño 40
Telefono	Teléfono del laboratorio	Entero

5.3.1.6. PRODUCTO

Nombre de Dominio	Significado	Definición del Dominio
idProducto	Identificador único para de un producto	Entero
Descripcion	Descripción del producto	Carácter : Tamaño 50
Concentracion	Concentracion que contiene el producto	Carácter : Tamaño 50
Stock	Stock de los productos	Carácter : Tamaño 10
Costo	Costo por producto	Carácter : Tamaño 10
Precio_Venta	Precio de venta del producto	Decimal: tamaño 7 dígitos, 2 decimales
RegistroSanitario	Registro Sanitario del producto	Carácter : Tamaño 40
FechaVencimiento	Fecha en la vence el producto	Fecha
Estado	Estado que refleja la situación del producto	Carácter : Tamaño 50
idPresentacion	Identificador único para una presentación	Entero
idLaboratorio	Identificador único para el laboratorio	Entero

5.3.1.7. PROVEEDOR

Nombre de Dominio	Significado	Definición del Dominio
idProveedor	Identificador único para un proveedor	Entero
Nombre	Nombre completo del proveedor	Carácter : Tamaño 50
Dni	Identificador único del tipo documento del proveedor	Carácter : Tamaño 10
Telefono	Teléfono del proveedor	Entero
Dirección	Dirección de residencia del proveedor	Carácter : Tamaño 40
Email	Correo electrónico del proveedor	Carácter : Tamaño 40
Cuenta	Número de cuenta del proveedor	Entero
Banco	El nombre del banco en donde se encuentra el proveedor	Carácter : Tamaño 40

5.3.1.8. TIPO

Nombre de Dominio	Significado	Definición del Dominio
idTipo	Identificador único para un tipo de comprobante	Entero
Comprobante	Descripción del comprobante	Carácter : Tamaño 50

5.3.1.9. COMPROBANTE

Nombre de Dominio	Significado	Definición del Dominio
idcomprobante	Identificador único de un comprobante	Entero
Descripcion	Descripción del comprobante	Carácter : Tamaño 50
Estado	Estado que refleja la situación del comprobante	Carácter : Tamaño 50
idTipo	Identificador único para un tipo de comprobante	Entero

5.3.1.10. COMPRA

Nombre de Dominio	Significado	Definición del Dominio
idCompra	Identificador único para la compra	Entero
Numero	Numero de la compra	Carácter : Tamaño 50
Fecha	Fecha en la que se realizó la compra	Carácter : Tamaño 10
TipoPago	El nombre del tipo de pago de la compra	Carácter : Tamaño 30
SubTotal	Ell sub-total de la compra	Decimal: tamaño 7 dígitos, 2 decimales
Total	Valor total que se va a pagar	Decimal: tamaño 7 dígitos, 2 decimales
Igv	Valor del porcentaje a aplicar si el tipo de comprobante es factura	Decimal: tamaño 7 dígitos, 2 decimales
idUsuario	Identificador único para un usuario	Entero
idProveedor	Identificador único para un proveedor	Entero
idcomprobante	Identificador único de un comprobante	Entero

5.3.1.11. DETALLE DE COMPRA

Nombre de Dominio	Significado	Definición del Dominio
idCompra	Identificador único para la compra	Entero
idProducto	Identificador único para de un producto	Entero
Cantidad	Cantidad de producto que se compró	Entero
Costo	El costo de la compra	Decimal: tamaño 7 dígitos, 2 decimales
Importe	Importe de la compra	Decimal: tamaño 7 dígitos, 2 decimales

5.3.1.12. VENTA

Nombre de Dominio	Significado	Definición del Dominio
IdVenta	Identificador único para la venta	Entero
Serie	Serie de la venta	Carácter : Tamaño 50
Numero	Numero de la venta	Carácter : Tamaño 50
Fecha	Fecha en la que se realizó la venta	Carácter : Tamaño 10
VentaTotal	El nombre del tipo de pago de la venta	Carácter : Tamaño 30
Descuento	Descuento que se va a realizar en la venta	Decimal: tamaño 7 dígitos, 2 decimales
SubTotal	Ell sub-total de la venta	Decimal: tamaño 7 dígitos, 2 decimales
Igv	Valor del porcentaje a aplicar si el tipo de comprobante es factura	Decimal: tamaño 7 dígitos, 2 decimales
Total	Valor total que se va a pagar	Decimal: tamaño 7 dígitos, 2 decimales
idCliente	Identificador único para un estudiante	Entero
idEmpleado	Identificador único para un empleado	Entero
idcomprobante	Identificador único de un comprobante	Entero

5.3.1.13. DETALLE DE VENTA

Nombre de Dominio	Significado	Definición del Dominio
IdVenta	Identificador único para la venta	Entero
idProducto	Identificador único para de un producto	Entero
Cantidad	Cantidad que se vendio de cada producto	Entero
Costo	El costo de la venta	Decimal: tamaño 7 dígitos, 2 decimales
Precio	Costo por el producto	Decimal: tamaño 7 dígitos, 2 decimales
Importe	Importe de la compra	Decimal: tamaño 7 dígitos, 2 decimales

5.3.2. DEFINICION INTEGRIDAD DE ENTIDAD Y REFERENCIAL

5.3.2.1. CLIENTE

Atributos	Restricciones
idCliente	NOT NULL , PK
Nombre	NOT NULL
Apellidos	NOT NULL
Sexo	NULL
Dni	NULL
Telefono	NULL
Ruc	NULL
Dirección	NULL

5.3.2.2. USUARIO

Atributos	Restricciones
idUsuario	NOT NULL , PK
Nombre	NOT NULL
Apellidos	NOT NULL
Dni	NULL
Email	NULL
Contraseña	NOT NULL
TipoUsuario	NOT NULL

5.3.2.3. EMPLEADO

Atributos	Restricciones
idEmpleado	NOT NULL , PK
Nombre	NOT NULL
Apellidos	NOT NULL
Cargo	NOT NULL
Dni	NULL
Telefono	NULL
Dirección	NULL
HoraIngreso	NOT NULL
HoraSalida	NOT NULL
Sueldo	NOT NULL
idUsuario	NOT NULL , FK

5.3.2.4. PRESENTACION

Atributos	Restricciones
idPresentacion	NOT NULL , PK
Descripcion	NULL
Estado	NULL

5.3.2.5. LABORATORIO

Atributos	Restricciones
idLaboratorio	NOT NULL , PK
Nombre	NOT NULL
Direccion	NULL
Telefono	NULL

5.3.2.6. PRODUCTO

Atributos	Restricciones
idProducto	NOT NULL , PK
Descripcion	NULL
Concentracion	NOT NULL
Stock	NOT NULL
Costo	NOT NULL
Precio_Venta	NOT NULL
RegistroSanitario	NULL
FechaVencimiento	NOT NULL
Estado	NULL
idPresentacion	NOT NULL , FK
idLaboratorio	NOT NULL , FK

5.3.2.7. PROVEEDOR

Atributos	Restricciones
idProveedor	NOT NULL , PK
Nombre	NOT NULL
Dni	NOT NULL
Telefono	NULL
Dirección	NULL
Email	NOT NULL
Cuenta	NOT NULL
Banco	NOT NULL

5.3.2.8. TIPO

Atributos	Restricciones
idTipo	NOT NULL , PK
Comprobante	NULL

5.3.2.9. COMPROBANTE

Atributos	Restricciones
idcomprobante	NOT NULL , PK
Descripcion	NULL
Estado	NULL
idTipo	NULL

5.3.2.10. COMPRA

Atributos	Restricciones
idCompra	NOT NULL , PK
Numero	NULL
Fecha	NULL
TipoPago	NULL
SubTotal	NULL
Total	NULL
Igv	NULL
idUsuario	NOT NULL , FK
idProveedor	NOT NULL , FK
idcomprobante	NOT NULL , FK

5.3.2.11. DETALLE DE COMPRA

Atributos	Restricciones
idCompra	NOT NULL , FK
idProducto	NOT NULL , FK
Cantidad	NULL
Costo	NULL
Importe	NULL

5.3.2.12. VENTA

Atributos	Restricciones
IdVenta	NOT NULL , PK
Serie	NULL
Numero	NULL
Fecha	NULL
VentaTotal	NULL
Descuento	NULL
SubTotal	NULL
Igv	NULL
Total	NULL
idCliente	NOT NULL , FK
idEmpleado	NOT NULL , FK
idcomprobante	NOT NULL , FK

5.3.2.13. DETALLE DE VENTA

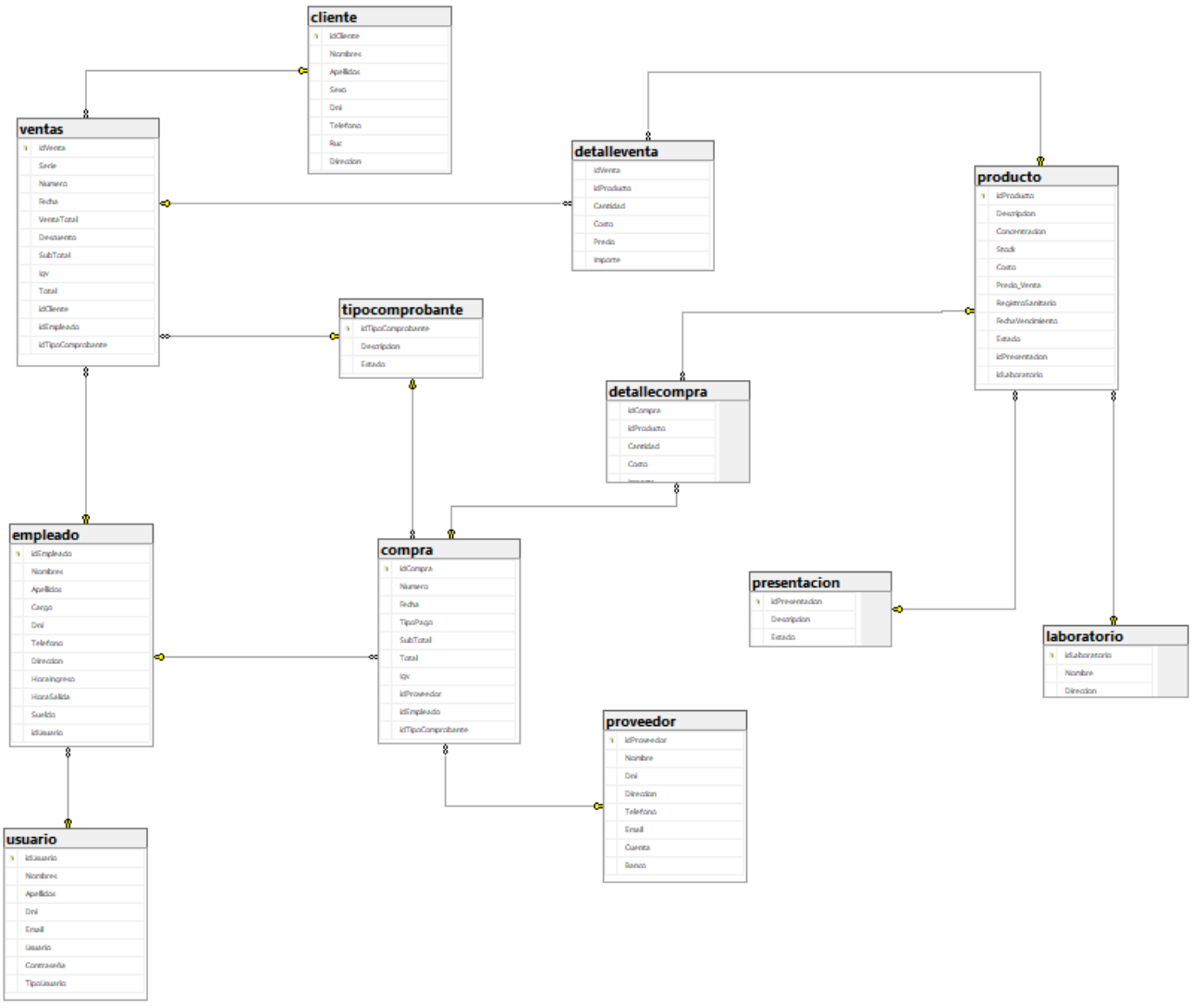
Atributos	Restricciones
IdVenta	NOT NULL , FK
idProducto	NOT NULL , FK
Cantidad	NOT NULL
Costo	NULL
Precio	NULL
Importe	NULL

5.4. LAS RELACIONES

Cliente	1 - N	Ventas
Usuario	1 - 1	Empleado
Presentación	1 - N	Producto
Laboratorio	1 - N	Producto
Producto	1 - N	DetalleVenta
Producto	1 - N	DetalleCompra
Proveedor	1 - N	Compra
Tipo	1 - N	Comprobante
Empleado	1 - N	Ventas
Comprobante	1 - N	Ventas
Ventas	1 - N	DetalleVenta

VI. DISEÑO FISICO: MODELO FISICO

6.1. DISEÑO DE LA BASE DE DATOS FISICA



6.2. IMPLEMENTAR CONSULTAS

```
                                STORED PROCEDURE (22)
--              RF1: Gestión de Proveedores

--01 SP para la creación, actualización y eliminación de registros de proveedores

CREATE PROCEDURE usp_MantenimientoProveedor
       @idProveedor int,
    @Nombres varchar(35),
    @Ruc varchar(11),
       @Dni varchar(8),
    @Direccion varchar(35),
    @Telefono varchar(9),
    @Email varchar(35),
    @Cuenta varchar(35),
    @Banco varchar(35),
       @Operacion int
AS
BEGIN
    IF @Operacion = 1 -- INSERTAR
              BEGIN
                     INSERT INTO proveedor (idProveedor, Nombres, Ruc, Dni, Direccion,
Telefono, Email, Cuenta, Banco)
                     VALUES (@idProveedor, @Nombres, @Ruc, @Dni, @Direccion, @Telefono,
@Email, @Cuenta, @Banco)
              END
       ELSE IF @Operacion = 2 -- ACTUALIZAR
              BEGIN
                     UPDATE proveedor
                     SET Nombres = @Nombres,
                            Ruc = @Ruc,
                            Dni = @Dni,
                            Direccion = @Direccion,
                            Telefono = @Telefono,
                            Email = @Email,
                            Cuenta = @Cuenta,
                     Banco = @Banco
                     WHERE idProveedor = @idProveedor
              END
       ELSE IF @Operacion = 3 -- ELIMINAR
              BEGIN
                     DELETE FROM proveedor
                     WHERE idProveedor = @idProveedor
              END
END
-------------------------------------------------------------------------------

--02 SP para obtener el resumen de compras de un proveedor:

CREATE PROCEDURE sp_ResumenComprasProveedor
    @idProveedor int
AS
BEGIN
    SELECT
        pro.Nombres AS Proveedor,
        c.Numero AS NumeroCompra,
        c.Fecha AS FechaCompra,
        c.Total AS TotalCompra,
        COUNT(dv.idProducto) AS CantidadProductos
    FROM compra c
```

```
    INNER JOIN detallecompra dv ON c.idCompra = dv.idCompra
    INNER JOIN producto p ON dv.idProducto = p.idProducto
       INNER JOIN proveedor pro on pro.idProveedor=c.idProveedor
    WHERE c.idProveedor = @idProveedor
    GROUP BY pro.Nombres, c.Numero, c.Fecha, c.Total
END
-------------------------------------------------------------------------------

--            RF2: Control de Medicamentos

--01 SP para agregar, editar y eliminar medicamentos en el sistema

CREATE PROCEDURE usp_MantenimientoProducto
       @idProducto int,
    @Descripcion varchar(35),
    @Concentracion varchar(30),
       @stock int,
       @Costo money,
    @Precio_Venta money,
    @RegistroSanitario varchar(20),
    @FechaVencimiento date,
       @Estado varchar(10),
       @idPresentacion int,
       @idLaboratorio int,
       @Operacion int
BEGIN
    IF @Operacion = 1 -- INSERTAR
              BEGIN
                     INSERT INTO producto(idProducto, Descripcion, Concentracion, stock,
Costo,  Precio_Venta,  RegistroSanitario,  FechaVencimiento,  Estado,  idPresentacion,
idLaboratorio)
                     VALUES (@idProducto, @Descripcion, @Concentracion, @stock, @Costo,
@Precio_Venta,  @RegistroSanitario,  @FechaVencimiento,  @Estado,  @idPresentacion,
@idLaboratorio)
              END
       ELSE IF @Operacion = 2
              BEGIN
                     UPDATE producto -- ACTUALZAR
                     SET Descripcion = @Descripcion,
                            Concentracion = @Concentracion,
                            stock = @stock,
                            Costo = @Costo,
                            Precio_Venta = @Precio_Venta,
                            RegistroSanitario = @RegistroSanitario,
                            FechaVencimiento = @FechaVencimiento,
                            Estado = @Estado,
                            idPresentacion = @idPresentacion,
                            idLaboratorio = @idLaboratorio
                     WHERE idProducto = @idProducto
              END
       ELSE IF @Operacion = 3 -- ELIMINAR
              BEGIN
                     DELETE FROM producto
                     WHERE idProducto = @idPresentacion
              END
END

-------------------------------------------------------------------------------

--02 SP para obtener el stock bajo de productos:

CREATE PROCEDURE sp_StockBajo
    @MinStock int
AS
```

```
BEGIN
    SELECT
        p.Descripcion AS Producto,
        p.Stock AS StockActual,
        p.Precio_Venta AS PrecioVenta
    FROM producto p
    WHERE p.Stock <= @MinStock
END

-------------------------------------------------------------------------------------03
SP para actualizar Stock luego de realizar una venta


CREATE PROCEDURE sp_ActualizarStockDespuesDeVenta
(
    @idMedicamento int,
    @CantidadVendida int
)
AS
BEGIN
    UPDATE producto
    SET Stock = Stock - @CantidadVendida
    WHERE idProducto = @idMedicamento
END

-------------------------------------------------------------------------------------

-- 04 SP para obtener el producto más vendido en un laboratorio

CREATE PROCEDURE SP_ProductoMasVendidoPorLaboratorioEspecifico
    @LaboratorioId INT
AS
BEGIN
    SELECT TOP 1
        p.Descripcion AS ProductoMasVendido,
        SUM(dv.Cantidad) AS TotalVendido
    FROM producto p
    INNER JOIN detalleventa dv ON p.idProducto = dv.idProducto
    WHERE p.idLaboratorio = @LaboratorioId
    GROUP BY p.Descripcion
    ORDER BY TotalVendido DESC
END

-------------------------------------------------------------------------------------

-- 05 SP: Se debe poder implementar un secion de busqueda de medicamentos para poder
-- visualizar la existencia de los medicamentos en el almacen y de su stock

CREATE PROCEDURE BuscarProducto
    @criterio VARCHAR(30),
    @Prod VARCHAR(20)
AS
BEGIN
    IF @criterio = 'Buscar'
    BEGIN
        SELECT
            p.idProducto,pr.Descripcion                                          AS
presentacion,p.Nombre,p.Concentracion,p.Stock, p.Costo,
                    p.Precio_Venta, p.FechaVencimiento,p.RegistroSanitario,l.Nombre AS
laboratorio,p.Estado
        FROM
            producto p
        INNER JOIN presentacion pr ON p.idPresentacion = pr.idPresentacion
       INNER JOIN laboratorio l ON p.idLaboratorio = l.idLaboratorio
```

```
        WHERE p.Nombre LIKE CONCAT('%',@Prod,'%');
    END
END

-------------------------------------------------------------------------------

--06 Crear un Store Procedure en donde se esten los productos comprados por
por los clientes en un intevalo de fechas*/

create proc usp_cliente_prod_fecha
@fecha1 varchar(10),
@fecha2 varchar(10)
as
 select producto.Nombre,compra.idUsuario,compra.fecha
 from compra
 inner join detallecompra on compra.idCompra=detallecompra.idCompra
 inner join producto detallecomora.idProducto=producto.idProducto
 where Fecha between @fecha1 and @fecha2


-------------------------------------------------------------------------------

--            RF3: Registro de Clientes

--01 SP para la creación, actualización y eliminación de registros de clientes

CREATE PROCEDURE usp_MantenimientoCliente
       @idCliente int,
    @Nombres varchar(35),
    @Apellidos varchar(35),
       @Genero char(1),
       @Dni varchar(8),
    @Telefono varchar(9),
    @Ruc varchar(11),
    @Direccion varchar(50),
       @Operacion int
AS
BEGIN
    IF @Operacion = 1 -- INSERTAR
              BEGIN
                     INSERT INTO cliente(idCliente, Nombres, Apellidos, Genero, Dni,
Telefono, Ruc, Direccion)
                     VALUES (@idCliente, @Nombres, @Apellidos, @Genero, @Dni, @Telefono,
@Ruc, @Direccion)
              END
       ELSE IF @Operacion = 2
              BEGIN
                     UPDATE cliente -- ACTUALZAR
                     SET
                            Nombres = @Nombres,
                            Apellidos = @Apellidos,
                            Genero = @Genero,
                            Dni = @Dni,
                            Telefono = @Telefono,
                            Ruc =@Ruc,
                            Direccion = @Direccion
                     WHERE idCliente = @idCliente
              END
       ELSE IF @Operacion = 3 -- ELIMINAR
              BEGIN
                     DELETE FROM cliente
                     WHERE idCliente = @idCliente
              END
END
```

```
-------------------------------------------------------------------------------
--            RF4: Administración de Empleados

--01 SP para agregar, actualizar y eliminar registros de empleados.

CREATE PROCEDURE usp_MantenimientoEmpleados
       @idEmpleado int,
    @Nombres varchar(35),
    @Apellidos varchar(30),
       @Cargo varchar(30),
       @Dni varchar(8),
    @Telefono varchar(9),
    @Direccion varchar(50),
    @FechaLaboral date,
       @HoraIngreso time,
       @HoraSalida time,
       @Sueldo money,
       @IdUsuario INT,
       @Operacion int
AS
BEGIN
    IF @Operacion = 1 -- INSERTAR
       BEGIN
              INSERT INTO empleado (idEmpleado, Nombres, Apellidos, Cargo, Dni, Telefono,
Direccion, FechaLaboral, HoraIngreso, HoraSalida, Sueldo, idUsuario)
              VALUES  (@idEmpleado,  @Nombres,  @Apellidos,  @Cargo,  @Dni,  @Telefono,
@Direccion, @FechaLaboral, @HoraIngreso, @HoraSalida, @Sueldo, @IdUsuario)
       END
       ELSE IF @Operacion = 2
              BEGIN UPDATE empleado -- ACTUALZAR
                     SET Nombres = @Nombres,
                            Apellidos = @Apellidos,
                            Cargo = @Cargo,
                            Dni = @Dni,
                            Telefono = @Telefono,
                            Direccion = @Direccion,
                            FechaLaboral = @FechaLaboral,
                            HoraIngreso = @HoraIngreso,
                            HoraSalida = @HoraSalida,
                            Sueldo = @Sueldo,
                            idUsuario = @IdUsuario
        WHERE idEmpleado = @IdEmpleado
              END
       ELSE IF @Operacion = 3 -- ELIMINAR
              BEGIN
                     DELETE FROM empleado
                     WHERE idEmpleado = @IdEmpleado
              END
END

--02 SP para el detalle de la venta de un empelado por su id

CREATE PROCEDURE ConsultaDetalleVentaDeEmpleado
@idEmpleado INT,
@idVenta INT
AS
BEGIN
    SELECT dv.IdVenta, dv.idProducto, p.Descripcion AS Producto, dv.Cantidad, dv.Costo,
dv.Precio, dv.Importe
    FROM detalleventa dv
    JOIN producto p ON dv.idProducto=p.idProducto
    JOIN ventas v ON dv.IdVenta=v.IdVenta
    WHERE v.idEmpleado=@idEmpleado AND dv.IdVenta=@idVenta
```

```
END

-------------------------------------------------------------------------------

--03 SP para todas las ventas realizadas de cierto empleado, en un rango de fechas
CREATE PROCEDURE SP_ConsultaVentasPorEmpleado
@idEmpleado INT,
@FechaInicio DATE,
@FechaFin DATE
AS
BEGIN
    SELECT v.IdVenta, v.Serie, v.Numero, v.Fecha, v.VentaTotal, v.Descuento, v.SubTotal,
v.Igv
    FROM ventas v
    WHERE v.idEmpleado=@idEmpleado AND v.Fecha BETWEEN @FechaInicio AND @FechaFin
END

-------------------------------------------------------------------------------

--04 SP para mostrar los empleados con más ventas y la total de sus ventas entre dos
fechas
CREATE PROCEDURE SP_ConsultaEmpleadosConMasVentasEntreFechas
@FechaInicio DATE,
@FechaFin DATE
AS
BEGIN
    SELECT   e.idEmpleado,   e.Nombres,   e.Apellidos,   COUNT(v.IdVenta)   TotalVentas,
SUM(v.VentaTotal) TotalVentasMonto
    FROM empleado e
    LEFT JOIN ventas v ON e.idEmpleado=v.idEmpleado
    WHERE v.Fecha BETWEEN @FechaInicio AND @FechaFin
    GROUP BY e.idEmpleado, e.Nombres, e.Apellidos
    ORDER BY TotalVentas DESC
END

-------------------------------------------------------------------------------

--05 SP para calcular la comisión de un empleado en base a sus ventas:

CREATE PROCEDURE sp_CalcularComisionEmpleado
    @idEmpleado int,
    @PorcentajeComision decimal(5, 2) = 0.05
AS
BEGIN
    DECLARE @TotalVentas decimal(8, 2)
    SELECT @TotalVentas = SUM(v.Total)
    FROM ventas v
    WHERE v.idEmpleado = @idEmpleado
    DECLARE @Comision decimal(8, 2)
    SET @Comision = @TotalVentas * @PorcentajeComision
    SELECT
        e.Nombres + ' ' + e.Apellidos AS Empleado,
        @TotalVentas AS TotalVentas,
        @PorcentajeComision AS PorcentajeComision,
        @Comision AS ComisionCalculada from empleado e
END

-------------------------------------------------------------------------------

/*06.crear un Store Procedure la cantidad de empleados que trabajan en la farmacia
     duarnte el mes y semana*/

CREATE PROC usp_empleado_fecha
    @semana int,
    @mes int
AS
```

```
BEGIN
    SELECT
        COUNT(*) AS TotalEmpleados,
        DATEPART(week, empleado.FechaLaboral) AS Semana,
        DATEPART(MONTH, empleado.FechaLaboral) AS Mes
    FROM
        empleado
    WHERE
        DATEPART(week, empleado.FechaLaboral) = @semana AND
        DATEPART(MONTH, empleado.FechaLaboral) = @mes;
END

-------------------------------------------------------------------------------

--     RF5: Gestión de ventas y compras

--01 SP para aplicar descuento a productos próximos a vencer

CREATE PROCEDURE SP_AplicarDescuentosProductosProximosAVencer
    @DiasRestantes INT,
    @DescuentoPorcentaje DECIMAL(5, 2)
AS
BEGIN
    UPDATE producto
    SET Precio_Venta = Precio_Venta * (1 - @DescuentoPorcentaje / 100)
    WHERE DATEDIFF(DAY, GETDATE(), FechaVencimiento) <= @DiasRestantes
END

-------------------------------------------------------------------------------

--02 SP El sistema debe buscar las compras realizadas, incluyendo información sobre
el proveedor, empleado, fecha y monto total de la compra*/

CREATE PROCEDURE ComprasPorFecha
       @criterio VARCHAR(30),
       @fechaInicio DATE,
       @fechaFin DATE
AS
BEGIN
              IF @criterio = 'Buscar'
                     SELECT     c.idCompra,p.Nombres     AS     proveedor,     c.Fecha,
CONCAT(u.Nombres, ' ' ,u.Apellidos) AS empleado,
                     copr.Descripcion AS tipocomprobante,c.Numero,c.Total
                     FROM compra AS c
                     INNER JOIN proveedor p
                     ON c.idProveedor=p.IdProveedor
                            INNER JOIN usuario u
                            ON c.idUsuario=u.idUsuario
                            INNER JOIN comprobante copr
                            ON c.idcomprobante=copr.idcomprobante
                     WHERE (c.Fecha >=@fechaInicio AND c.Fecha<=@fechaFin)
                     ORDER BY c.idCompra DESC;
       END

-------------------------------------------------------------------------------

--03 crear un store procedure la cantidad de empleados y el promedio de ventas por día

CREATE PROCEDURE sp_PromedioVentasPorDia
    @FechaInicio DATE,
    @FechaFin DATE
AS
BEGIN
    SELECT
        v.Fecha,
```

```
        COUNT(DISTINCT v.idEmpleado) AS CantidadEmpleados,
        AVG(v.VentaTotal) AS PromedioVentas
    FROM Ventas v inner join empleado e on v.idEmpleado=e.idEmpleado
    WHERE v.Fecha BETWEEN @FechaInicio AND @FechaFin
    ORDER BY v.Fecha
END

--04 Store procedure para listar empleados con sus ventas en intervalo
     de fechas en una tabla temporal llamada  ##listar_empleado*/
CREATE PROCEDURE sp_ListarEmpleadosConVentas
    @FechaI DATE,
    @FechaF DATE
AS
BEGIN
    CREATE TABLE ##listar_empleado
    (
        idEmpleado INT,
        Nombre VARCHAR(20),
        FechaVenta DATE,
        MontoVenta DECIMAL(10, 2)
            )
    INSERT INTO ##listar_empleado (EmpleadoID, Nombre, FechaVenta, MontoVenta)
    SELECT e.idEmpleado, e.Nombre, v.Fecha, v.VentaTotal
    FROM empleado e
    INNER JOIN  ventas v ON e.idEmpleado = v.idEmpleado
    WHERE v.Fecha BETWEEN @FechaI AND @FechaF
    SELECT * FROM ##listar_empleado;
END

-------------------------------------------------------------------------------

--     RF6: Detalles de Ventas

--01   Se debe poder consultar los detalles de cada venta, incluyendo nombre,
--     descripcion del producto total del importe y ganacia
CREATE PROCEDURE VentasPorDetalle
    @criterio VARCHAR(30),
    @fechaIni DATE,
    @fechaFin DATE
AS
BEGIN
    IF @criterio = 'consultar'
    BEGIN
        SELECT
            p.idProducto,p.Nombre,pr.Descripcion,dv.Costo,   dv.Precio,SUM(dv.Cantidad)
AS TotalCantidad,
            SUM(dv.Importe) AS Total_Importe,SUM(dv.Importe - (dv.Costo * dv.Cantidad))
AS GananciaTotal
        FROM
            ventas v
        INNER JOIN detalleventa dv
             ON v.IdVenta = dv.IdVenta
                    INNER JOIN producto p
                    ON dv.idProducto = p.idProducto
                           INNER JOIN presentacion pr
                           ON p.idPresentacion = pr.idPresentacion
        WHERE
            (v.Fecha >= @fechaIni AND v.Fecha <= @fechaFin)
        GROUP BY p.idProducto, p.Nombre,pr.Descripcion,dv.Costo,dv.Precio;
    END
    ELSE
    BEGIN
        PRINT 'criterio invalido usar "consultar" '
    END
```

```
END

-------------------------------------------------------------------------------

--RF9: Reportes de Inventario
--01 SP para obtener un informe de ventas por mes:

CREATE PROCEDURE sp_InformeVentasPorMes
    @Anio int,
    @Mes int
AS
BEGIN
    SELECT
        DATEPART(YEAR, v.Fecha) AS Anio,
        DATEPART(MONTH, v.Fecha) AS Mes,
        COUNT(v.IdVenta) AS NumeroVentas,
        SUM(v.Total) AS TotalVentas
    FROM ventas v
    WHERE DATEPART(YEAR, v.Fecha) = @Anio AND DATEPART(MONTH, v.Fecha) = @Mes
    GROUP BY DATEPART(YEAR, v.Fecha), DATEPART(MONTH, v.Fecha)
END

-------------------------------------------------------------------------------

--RF11: Alertas de caducidad
--01   Advierte de productos pronto a caducar

CREATE PROCEDURE SP_AlertasCaducidadMedicamentos
AS
BEGIN

    DECLARE @PlazoCaducidad INT
    SET @PlazoCaducidad = 30

    DECLARE @FechaLimite DATE
    SET @FechaLimite = DATEADD(DAY, @PlazoCaducidad, GETDATE())

    SELECT p.idProducto, p.Descripcion, p.FechaVencimiento
    FROM producto p
    WHERE p.Estado = 'Disponible' AND p.FechaVencimiento <= @FechaLimite
END

-------------------------------------------------------------------------------

                                    FUNCIONES (15)
--RF1: Gestión de Proveedores
--01 Calcular monto total de compras de un proveedor en un periodo de tiempo
CREATE FUNCTION fn_MontoTotalComprasProveedor
(       @idProveedor int,
    @FechaInicio date,
    @FechaFin date)
RETURNS money
AS
BEGIN
    DECLARE @MontoTotal money
    SELECT @MontoTotal = SUM(c.Total)
    FROM compra c
    WHERE c.idProveedor = @idProveedor
    AND c.Fecha BETWEEN @FechaInicio AND @FechaFin
    RETURN ISNULL(@MontoTotal, 0)
END
-------------------------------------------------------------------------------

--01 función para calcular el sueldo total de un empleado con comisión:
```

```sql
CREATE FUNCTION fn_CalcularSueldoConComision (
    @idEmpleado int
)
RETURNS decimal(10, 2)
AS
BEGIN
    DECLARE @Sueldo float
    DECLARE @PorcentajeComision float= 0.05
    DECLARE @SueldoConComision float

    SELECT @Sueldo = e.Sueldo
    FROM empleado e
    WHERE e.idEmpleado = @idEmpleado

    SET @SueldoConComision = @Sueldo + (@Sueldo * @PorcentajeComision)
    RETURN @SueldoConComision
END
-------------------------------------------------------------------------------

/* RF2: Control de medicamentos
--01 Calcular días restantes que le quedan a un producto antes de su caducidad
CREATE FUNCTION fn_DiasRestantesCaducidad
(@idProducto int)
RETURNS int
AS
BEGIN
    DECLARE @DiasRestantes int

    SELECT @DiasRestantes = DATEDIFF(DAY, GETDATE(), FechaVencimiento)
    FROM producto
    WHERE idProducto = @idProducto

    RETURN ISNULL(@DiasRestantes, 0)
END

-------------------------------------------------------------------------------

-- RF3: Registro de clientes

--01   Cantidad de compras de clientes

CREATE FUNCTION fn_Cantidad_ComprasCliente(
@idCliente int
)
RETURNS int
AS
BEGIN
    DECLARE @TotalCompras int

    SELECT @TotalCompras = COUNT(*)
    FROM ventas
    WHERE idCliente = @idCliente

    RETURN @TotalCompras
END

--RF4: Administración de empleados

--01 RETORNA LA VENTA DE UN EMPLEADO POR SU ID Y EL ID DE LA VENTA
CREATE FUNCTION FN_ObtenerVentaEmpleado
(@idEmpleado INT,@idVenta INT)
RETURNS TABLE
AS
RETURN
```

```
(   SELECT v.IdVenta, v.Serie, v.Numero, v.Fecha, v.VentaTotal, v.Descuento, v.SubTotal,
v.Igv
    FROM ventas v
    WHERE v.idEmpleado=@idEmpleado AND v.IdVenta=@idVenta)

--02 RETORNA EL TOTAL DE LAS VENTAS DE TODOS LOS EMPLEADOS ENTRE CIERTAS FECHAS
CREATE FUNCTION FN_ObtenerTotalVentasPorFechas
(@FechaInicio DATE, @FechaFin DATE)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @TotalVentas DECIMAL(10, 2)
    SELECT @TotalVentas=SUM(VentaTotal)
    FROM ventas
    WHERE Fecha BETWEEN @FechaInicio AND @FechaFin
    RETURN @TotalVentas
END
-------------------------------------------------------------------------------



/* RF5: Gestión de ventas y compras (04)

--01 funcion de la busquedas de productos en el almacen*/

CREATE FUNCTION BuscarProductoFuncion (
    @criterio VARCHAR(30),
    @Prod VARCHAR(20)
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        p.idProducto, pr.Descripcion AS presentacion, p.Nombre, p.Concentracion,
p.Stock, p.Costo,
        p.Precio_Venta, p.FechaVencimiento, p.RegistroSanitario, l.Nombre AS
laboratorio, p.Estado
    FROM
        producto p INNER JOIN presentacion pr
        ON p.idPresentacion = pr.idPresentacion INNER JOIN laboratorio l
        ON p.idLaboratorio = l.idLaboratorio
    WHERE @criterio = 'Buscar' AND p.Nombre LIKE CONCAT('%', @Prod, '%')
)
-------------------------------------------------------------------------------
--02 una funcion de las ventas realizadas por un empleado en un rango de fechas.

CREATE FUNCTION fn_VentasPorEmpleado
(@idEmpledo INT, @FechaInicio DATE, @FechaFin DATE)
RETURNS TABLE
AS
RETURN (
    SELECT Fecha, Total
    FROM Ventas
    WHERE idEmpleado = @idEmpleado AND
       Fecha BETWEEN @FechaInicio AND @FechaFin)
-------------------------------------------------------------------------------
-03. el promedio de ventas diarias de un empleado en un mes específico. */
CREATE FUNCTION fn_PromedioVentasDiarias
(@idEmpleado INT, @Mes INT, @Año INT)
```

```
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @Promedio DECIMAL(10,2)

    SELECT @Promedio = AVG(VentaTotal)
    FROM ventas
    WHERE idEmpleado = @idEmpleado AND MONTH(Fecha) = @Mes AND YEAR(Fecha) = @Año
    GROUP BY DAY(Fecha)
    RETURN @Promedio
END
--04   obtiene el detalle de una compra específica por el ID de compra y producto
CREATE FUNCTION fn_ObtenerDetalleCompra
(@idCompra INT, @idProducto INT)
RETURNS TABLE
AS
RETURN
(   SELECT dc.IdCompra, dc.idProducto, dc.Cantidad, dc.Costo, dc.Importe
    FROM detallecompra dc
    WHERE dc.IdCompra = @idCompra AND dc.idProducto = @idProducto )
-------------------------------------------------------------------------------

--RF6: Detalles de Ventas

--01. una funcion del total de ventas de un producto específico. */
CREATE FUNCTION fn_TotalVentasProducto(@idProducto INT)
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @TotalVentas DECIMAL(10,2)
    SELECT @TotalVentas = SUM(Cantidad * Precio)
    FROM DetallesVenta
    WHERE idProducto = @idProducto
    RETURN @TotalVentas
END

-------------------------------------------------------------------------------

--RF10: Seguimiento de Ventas
--01 función para obtener el promedio de ventas por empleado en un periodo determinado:

CREATE FUNCTION fn_PromedioVentasPorEmpleado (
    @idEmpleado int,
    @FechaInicio date,
    @FechaFin date
)
RETURNS float
AS
BEGIN
    DECLARE @TotalVentas decimal(10, 2);
    DECLARE @Dias int;

    SELECT @TotalVentas = SUM(v.VentaTotal)
    FROM ventas v
    WHERE v.idEmpleado = @idEmpleado
    AND v.Fecha >= @FechaInicio AND v.Fecha <= @FechaFin

    SELECT @Dias = DATEDIFF(day, @FechaInicio, @FechaFin)

    RETURN @TotalVentas / @Dias
END

-------------------------------------------------------------------------------

--02  función que calcula el total de ventas de un empleado en función de su ID :
```

```sql
CREATE FUNCTION fn_TotalVentasEmpleado (@idEmpleado int)
RETURNS float
AS
BEGIN
    DECLARE @TotalVentas float
    SELECT @TotalVentas = SUM(v.VentaTotal)
    FROM ventas v
    WHERE v.idEmpleado = @idEmpleado
    RETURN @TotalVentas
END

--03   Tipo de comprobante con más ventas asociadas

CREATE FUNCTION fn_CantidadVentasPorTipoComprobante(
@idTipoComprobante int
)
RETURNS INT
AS
BEGIN
    DECLARE @NumVentas INT

    SELECT @NumVentas = COUNT(*)
    FROM ventas
    WHERE idcomprobante = @idTipoComprobante;

    RETURN @NumVentas
END

-------------------------------------------------------------------------------

--RF11: Alertas de Caducidad
--01 Se debe enviar una notificación a los empleados encargados para que tomen
medidas */

CREATE FUNCTION AlertasCaducidadMedicamentos()
RETURNS TABLE
AS
RETURN
(SELECT 'Productos con Fecha de Vencimiento Próxima a Vencer' AS Mensaje,
          p.idProducto, p.Nombre, p.FechaVencimiento
    FROM producto p
    WHERE p.Estado = 'Disponible' AND p.FechaVencimiento <= DATEADD(DAY, 30, GETDATE()))
-------------------------------------------------------------------------------

--RF13: Gestión de Laboratorios

--01 Mostrar laboratorio con mayor cantidad de ventas
CREATE FUNCTION FN_LaboratorioMasConsumido()
RETURNS VARCHAR(50)
AS
BEGIN
    DECLARE @Laboratorio VARCHAR(50)
    SELECT TOP 1 @Laboratorio = l.Nombre
    FROM laboratorio l
    INNER JOIN producto p ON l.idLaboratorio = p.idLaboratorio
    INNER JOIN detalleventa dv ON p.idProducto = dv.idProducto
    GROUP BY l.Nombre
    ORDER BY SUM(dv.Cantidad) DESC
    RETURN @Laboratorio
END

-----------------------------------------------------------------
```

```sql
-- VISTAS

--RF1: Gestión de Proveedores
--01 Esta vista permite la información de compras de manera simplificada y seguras.

CREATE VIEW Vista_Compras AS
SELECT c.idCompra, c.Numero, c.Fecha, p.Nombres AS Proveedor, c.SubTotal, c.Total
FROM compra c
INNER JOIN proveedor p ON c.idProveedor = p.idProveedor
-------------------------------------------------------------------------------

--02 vista del historial de compras por proveedor
CREATE VIEW HistorialComprasProveedor AS
SELECT
    p.Nombre AS NombreProveedor,
    c.Fecha,
    prod.Nombre,
    dc.Cantidad,
    prod.Costo,
    (dc.Cantidad * prod.PrecioCompra) AS TotalCompra
FROM
    Proveedores p
INNER JOIN
    Compras c ON p.idProveedor = c.idProveedor
INNER JOIN
    DetalleCompra dc ON c.idCompra = dc.idCompra
INNER JOIN
    Productos prod ON dc.idProducto = prod.idProducto
ORDER BY
    p.Nombre,
    c.FechaCompra
-------------------------------------------------------------------------------

--03 vista del proveedor*/
CREATE VIEW Vista_Proveedor
AS
SELECT idProveedor,Nombre, Ruc AS RUC_EMPRESA ,Dni,Email,Direccion,Telefono
FROM Proveedor
-------------------------------------------------------------------------------

--RF02: Control de medicamentos

-- 3)  Vista para mostrar el resumen de ventas mensuales para cada producto

CREATE VIEW w_VistaVentasMensuales AS
SELECT P.Descripcion AS Producto, DATENAME(MONTH, V.Fecha) AS Mes, YEAR(V.Fecha) AS Año,
            SUM(DV.Cantidad) AS CantidadVendida
FROM producto P
JOIN detalleventa DV ON P.idProducto = DV.idProducto
JOIN ventas V ON DV.IdVenta = V.IdVenta
GROUP BY P.Descripcion, DATENAME
-------------------------------------------------------------------------------

--     RF3: Registro de Clientes

--01  Genere una vista que combine la información del cliente con su historial de
compras,
--mostrando el nombre del cliente, la fecha de compra y el monto total gastado en cada
compra.

CREATE VIEW w_HistorialCompras_Cliente as
SELECT TOP 100 PERCENT
            C.idCliente, C.Dni, C.Nombres + ' ' + C.Apellidos as [Nombres Completos],
```

```
            V.Fecha, SUM(V.TOTAL) AS [Compras Realizadas]
FROM CLIENTE C
JOIN VENTAS V ON V.idCliente = C.idCliente
GROUP BY C.idCliente, C.Dni, C.Nombres, C.Apellidos, V.FECHA
ORDER BY C.idCliente ASC

-------------------------------------------------------------------------------

--RF4: Administración de empleados

--01 VISTAS DE TODAS LAS VENTAS REALIZADAS DE UN EMPLEADO EN UN RANGO DE FECHAS
CREATE VIEW V_VentasPorEmpleadoEntreFechas
AS
SELECT v.IdVenta, v.Serie, v.Numero, v.Fecha, v.VentaTotal, v.Descuento, v.SubTotal,
v.Igv
FROM ventas v
JOIN empleado e ON v.idEmpleado=e.idEmpleado
WHERE v.Fecha BETWEEN @FechaInicio AND @FechaFin

-------------------------------------------------------------------------------

--02 VISTA QUE MUESTRA LOS EMPLEADOS CON MÁS VENTAS Y EL TOTAL EN UN RANGO DE FECHAS
CREATE VIEW V_EmpleadosConMasVentasYTotalEntreFechas
AS
SELECT   e.idEmpleado,   e.Nombres,   e.Apellidos,   COUNT(v.IdVenta)   TotalVentas,
SUM(v.VentaTotal) TotalVentasMonto
FROM empleado e
LEFT JOIN ventas v ON e.idEmpleado=v.idEmpleado
WHERE v.Fecha BETWEEN @FechaInicio AND @FechaFin
GROUP BY e.idEmpleado, e.Nombres, e.Apellidos
ORDER BY TotalVentas DESC

-------------------------------------------------------------------------------

-- RF5: Gestión de ventas y compras (04)

--01   Vista para mostrar la compra de productos

CREATE VIEW w_ProveedorProductosSuministrados AS
SELECT C.idCompra, PD.Descripcion, PR.IdProveedor, PR.Nombres Proveedor, SUM(C.TOTAL)
[Total Compra]
FROM compra C
JOIN proveedor PR ON PR.idProveedor = C.idProveedor
JOIN detallecompra DC ON DC.idCompra = C.idCompra
JOIN producto PD ON PD.idProducto = DC.idProducto
GROUP BY C.idCompra, PR.IdProveedor, PR.Nombres, PD.Descripcion

SELECT * FROM w_ProveedorProductosSuministrados

--02 Te permite acceder a la información de las ventas, incluyendo el nombre completo
del cliente asociado a cada venta
CREATE VIEW Vista_Ventas AS
SELECT v.IdVenta, v.Serie, v.Numero, v.Fecha, c.Nombres + ' ' + c.Apellidos AS Cliente,
v.Total
FROM ventas v
INNER JOIN cliente c ON v.idCliente = c.idCliente

-------------------------------------------------------------------------------

--RF6: Detalles de ventas

--01 Mes más bajo más alto de ventas de cada producto
CREATE VIEW Vista_MesBajoAltoVentasPorProducto
AS
SELECT
```

```
    p.idProducto,
    p.Descripcion AS Producto,
    MIN(FORMAT(v.Fecha, 'MMMM yyyy')) AS MesMasBajo,
    MAX(FORMAT(v.Fecha, 'MMMM yyyy')) AS MesMasAlto
FROM
    producto p
LEFT JOIN detalleventa dv ON p.idProducto = dv.idProducto
LEFT JOIN ventas v ON dv.IdVenta = v.IdVenta
GROUP BY p.idProducto, p.Descripcion
-------------------------------------------------------------------------------

--02 Vista que muestra detalles de ventas con productos
CREATE VIEW Vista_DetallesVentasConProductos
AS
SELECT
    v.IdVenta,
    v.Serie AS SerieVenta,
    v.Numero AS NumeroVenta,
    v.Fecha AS FechaVenta,
    c.Nombres AS NombreCliente,
    c.Apellidos AS ApellidoCliente,
    p.Descripcion AS NombreProducto,
    dv.Cantidad AS CantidadVendida,
    p.Precio_Venta AS PrecioUnitario,
    dv.Importe AS ImporteTotal
FROM
    ventas v
INNER JOIN cliente c ON v.idCliente = c.idCliente
INNER JOIN detalleventa dv ON v.IdVenta = dv.IdVenta
INNER JOIN producto p ON dv.idProducto = p.idProducto

-------------------------------------------------------------------------------

--RF10: Seguimiento de Ventas

--01   Vista para mostrar las ventas totales para cada venta realizada por un empleado
específico.

CREATE VIEW w_VentasTotales_Empleado AS
SELECT TOP 100 PERCENT
             E.idEmpleado, E.Dni, E.Nombres, E.Apellidos, E.Cargo,
             COUNT(V.IdVenta) [Cantidad Ventas], SUM(V.Total)  AS [Ventas Totales]
FROM Empleado E
LEFT JOIN Ventas V ON V.IdEmpleado = E.IdEmpleado
GROUP BY E.idEmpleado, E.Dni, E.Nombres, E.Apellidos, E.Cargo
ORDER BY idEmpleado ASC

-------------------------------------------------------------------------------

--02 vista para tener un reporte de las ventas anuales por empleado
CREATE VIEW ReporteVentasAnualesPorEmpleado AS
SELECT
    e.idEmpleado,
    e.Nombre AS NombreEmpleado,
    YEAR(v.Fecha) AS Año,
    SUM(d.Cantidad * v.Precio) AS TotalVendido
FROM
    Ventas v
INNER JOIN
    Empleado e ON v.idEmpleado = e.idEmpleado
INNER JOIN
    detalleVenta d ON v.idVenta = d.idVenta
GROUP BY
```

```
    e.idEmpleado,
    e.Nombre,
    YEAR(v.Fecha)
ORDER BY
    Año DESC,
    TotalVendido DESC;

-------------------------------------------------------------------------------

--RF11: Alertas de Caducidad

--Esta vista te devolvería todos los productos activos.
CREATE VIEW Vista_Productos AS
SELECT idProducto, Descripcion, Precio_Venta
FROM producto
WHERE Estado = 'Activo'

-------------------------------------------------------------------------------

--RF13: Gestión de Laboratorios

--01 Vista que muestra información de laboratorios
CREATE VIEW Vista_InformacionLaboratorios
AS
SELECT
    idLaboratorio,
    Nombre AS NombreLaboratorio,
    Direccion AS DireccionLaboratorio,
    Telefono AS TelefonoLaboratorio
FROM laboratorio
-------------------------------------------------------------------------------
                                        TRIGGERS
--PRIMER TRIGGER EN AFTER
--RF2: Control de Medicamentos

--01 Este trigger es actualizar la cantidad de stock de un producto después de que se
inserta
--un nuevo registro en la tabla DetalleCompra.
CREATE TRIGGER tr_AfterInsertDetalleCompra
ON DetalleCompra
FOR INSERT
AS
BEGIN
    DECLARE @idProducto int
    DECLARE @Cantidad int

    SELECT @idProducto = idProducto, @Cantidad = Cantidad
    FROM inserted

    UPDATE producto
    SET Stock = Stock - @Cantidad
    WHERE idProducto = @idProducto
END
-------------------------------------------------------------------------------

--02 El propósito de este trigger es limitar la cantidad de productos que se pueden
--insertar en la tabla detallecompra a un máximo de 100 unidades.
CREATE TRIGGER tr_InsteadOfInsertDetalleCompraLimit
ON detallecompra
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @MaxCantidad int = 100
```

```
    INSERT INTO detallecompra (idCompra, idProducto, Cantidad, Costo, Importe)
    SELECT idCompra, idProducto, Cantidad, Costo, Importe
    FROM inserted
    WHERE Cantidad <= @MaxCantidad

    IF EXISTS (SELECT * FROM inserted WHERE Cantidad > @MaxCantidad)
    BEGIN
        PRINT ('La cantidad de productos excede el límite permitido.')
    END
END


--RF3: Registro de Clientes

--01 Trigger para registrar algun cambio de insercion de clientes */
CREATE TRIGGER tr_ins_cliente
on cliente
for insert
as
   declare @idcliente int
   declare @nombrecliente varchar(20)
   declare @fecha varchar(8)

   set @idcliente=(select idcliente from inserted)
   set @nombre=(select nombrecliente from
                cliente where idcliente=@idcli)
   set @fecha=(select fecha from  inserted)

   insert t_cliente values(@idcli,@nombre,@fecha)
```

```
--RF4: Administración de empleados

--01 TRIGGER DE ACTUALIZACION, PARA CONTROLAR LOS CAMBIOS DEL CAMPO SUELDO DE UN EMPLEADO
--EN UNA TABLA DE REGISTRO CON EL NOMBRE DE REGISTRO_SUELDOS
IF NOT EXISTS (SELECT 1 FROM sys.tables WHERE name='registro_sueldos')
BEGIN
    CREATE TABLE registro_sueldos (
        idRegistro INT IDENTITY(1,1) PRIMARY KEY,
        idEmpleado INT,
        SueldoAnterior DECIMAL(18, 2),
        SueldoNuevo DECIMAL(18, 2),
        FechaModificacion DATETIME
    )
END

CREATE TRIGGER TR_ActualizarSueldo
ON empleados
AFTER UPDATE
AS
BEGIN
    INSERT    INTO    registro_sueldos(idEmpleado,    SueldoAnterior,    SueldoNuevo,
FechaModificacion)
    SELECT
        i.idEmpleado,
        d.Sueldo  SueldoAnterior,
        i.Sueldo  SueldoNuevo,
        GETDATE() AS FechaModificacion
    FROM inserted i
    JOIN deleted d ON i.idEmpleado=d.idEmpleado
END

-------------------------------------------------------------------------------
--02 Trigger que verifica si el salario es mayor o igual al minimo

CREATE TRIGGER trg_ValidarSalarioBeforeUpdate
ON empleado
instead of UPDATE
AS
BEGIN

    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE i.Sueldo < 1250
    )
    BEGIN
        -- Cancelar la actualización
        ROLLBACK TRANSACTION;
    END
END
-------------------------------------------------------------------------------
--03 trigger nuevo empleado
CREATE TRIGGER Trigger_Empleado
ON empleado
AFTER INSERT
AS
BEGIN
    DECLARE @NuevoEmpleadoNombre VARCHAR(50)
    DECLARE @Mensaje VARCHAR(100)

    SELECT @NuevoEmpleadoNombre = Nombres + ' ' + Apellidos
    FROM inserted
```

```
    SET @Mensaje = 'Se ha añadido un nuevo empleado: ' + @NuevoEmpleadoNombre

    PRINT @Mensaje
END
-------------------------------------------------------------------------------
--            RF6: Detalles de Ventas

--01 CREAR TRIGGER PARA CALCULAR EL TOTAL DE LA VENTA
CREATE TRIGGER tr_calcular_subtotal_igv_total
ON ventas
AFTER INSERT
AS
BEGIN
    DECLARE @idVenta INT
    DECLARE @ventaTotal MONEY
    DECLARE @descuento MONEY
    DECLARE @subTotal MONEY
    DECLARE @igv MONEY
    DECLARE @total MONEY

    SELECT @idVenta = IdVenta,
                @ventaTotal = VentaTotal,
           @descuento = Descuento
    FROM inserted

    SET @subTotal = @ventaTotal - @descuento
    SET @igv = @subTotal * 0.18
    PRINT 'IGV INCLUIDO'
       SET @total = @subTotal + @igv
       PRINT 'VENTA EXITOSA'

    UPDATE VENTAS
    SET SubTotal = @subTotal,
        Igv = @igv,
        Total = @total
    WHERE
-------------------------------------------------------------------------------
--RF9: Reportes de Inventario

--01 Trigger que permita reducir el stock de un producto, luego de una venta

CREATE TRIGGER tg_VENTA
ON detalleVenta
for insert
as
declare @idProducto int
declare @stock int
declare @cantidad int

set @cantidad=(select cantidad from inserted)
set @idProducto =(select idProducto from inserted)
set @stock=(select stock from producto where idProducto=@idProducto)
if (@cantidad>0)
   begin
   if (@stock<=2)
   begin
   raiserror ('stock insuficiente')
   --ANULA EL EVENTO DE INSERT
   rollback transaction
   end
   else
```

```
   begin
   --ACTUALIZAMOS LA TABLA PRODUCTO, STOCK Y REDUCIMOS
   update producto set
   stock=stock-@cantidad
   where idProducto=@idProducto
   end
   end
else
 raiserror('la cantidad no es correcta')
 rollback transaction
end
-------------------------------------------------------------------------------
--RF10: Seguimiento de Ventas
-- Trigger que registra ventas en historial de ventas

CREATE TRIGGER trg_RegistrarVentaAfterInsert
ON ventas
AFTER INSERT
AS
BEGIN

    INSERT INTO historial_ventas (idVenta, FechaVenta, MontoTotal)
    SELECT IdVenta, Fecha, Total
    FROM inserted;
END;

-------------------------------------------------------------------------------
--            RF12: Historial de Cambios
--01 Crear un trigger de auditoria para los proveedores.

CREATE TABLE HistorialProveedor_ins (
    IdProveedor INT,
       Nombres varchar(35),
    RUC varchar(11),
       Dni varchar(8),
    FechaCambio DATETIME,
    Usuario VARCHAR(50),
    Motivo varchar(max)
)

CREATE TABLE HistorialProveedor_del (
    IdProveedor INT,
    Nombres varchar(35),
    RUC varchar(11),
       Dni varchar(8),
    FechaCambio DATETIME,
    Usuario VARCHAR(50),
    Motivo varchar(max)
)

-- TRIGGER

CREATE TRIGGER tr_historial_cambios_proveedor
ON proveedor
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @ins INT
    DECLARE @del INT
    DECLARE @Motivo VARCHAR(MAX)

    SET @ins = (SELECT COUNT(*) FROM inserted)
```

```
    SET @del = (SELECT COUNT(*) FROM deleted)

    IF @ins > 0 AND @del = 0
    BEGIN
        SET @Motivo = 'Inserción'
              PRINT @Motivo
    END

       ELSE IF @ins > 0 AND @del > 0
    BEGIN
        SET @Motivo = 'Actualización'
              PRINT @Motivo
    END

       ELSE IF @ins = 0 AND @del > 0
    BEGIN
        SET @Motivo = 'Eliminación'
              PRINT @Motivo
    END

    INSERT INTO HistorialProveedor_ins (IdProveedor, Nombres, RUC, Dni, FechaCambio,
Usuario, Motivo)
    SELECT IdProveedor, Nombres, RUC, Dni, GETDATE(), SYSTEM_USER, @Motivo
    FROM inserted

       INSERT INTO HistorialProveedor_del (IdProveedor, Nombres, RUC, Dni, FechaCambio,
Usuario, Motivo)
    SELECT IdProveedor, Nombres, RUC, Dni, GETDATE(), SYSTEM_USER, @Motivo
    FROM deleted
END
-------------------------------------------------------------------------------
-- CURSORES 5

-- RF3: GESTION DE CLIENTES
--1)   Crear un cursor para recorrer los registros de cliente

DECLARE @idCliente INT
DECLARE @Direccion VARCHAR(50)
DECLARE actualizarDireccion CURSOR FOR
SELECT idCliente, Direccion
FROM cliente

OPEN actualizarDireccion
SET @Direccion = 'Sin dirección'

FETCH NEXT FROM actualizarDireccion INTO @idCliente, @Direccion
WHILE @@FETCH_STATUS = 0
BEGIN
    IF @Direccion IS NULL
    BEGIN
        UPDATE cliente
        SET Direccion = @Direccion
        WHERE idCliente = @idCliente
    END
    FETCH NEXT FROM actualizarDireccion INTO @idCliente, @Direccion
END
CLOSE actualizarDireccion
DEALLOCATE actualizarDireccion
-------------------------------------------------------------------------------
--RF4: Administración de Empleados
--Un cursor para recorrer los registros de la tabla "empleado" e imprimir información
sobre cada empleado.
```

```sql
DECLARE @idEmpleado int
DECLARE @Nombres varchar(35)
DECLARE @Apellidos varchar(35)
DECLARE @Sueldo float

DECLARE cursorEmpleados CURSOR FOR
SELECT idEmpleado, Nombres, Apellidos, Sueldo
FROM empleado

OPEN cursorEmpleados
FETCH NEXT FROM cursorEmpleados INTO @idEmpleado, @Nombres, @Apellidos, @Sueldo

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Empleado: ' + @Nombres + ' ' + @Apellidos
    PRINT 'Sueldo: ' + CAST(@Sueldo AS varchar(10))
    PRINT '------------------------'

    FETCH NEXT FROM cursorEmpleados INTO @idEmpleado, @Nombres, @Apellidos, @Sueldo
END

CLOSE cursorEmpleados
DEALLOCATE cursorEmpleados

-------------------------------------------------------------------------------
-- RF8: Autenticación de Usuarios
--01 un cursor que modificar los gmail de los empleados para una mejor administracion

DECLARE @idus int
DECLARE @dni varchar(9)
DECLARE @nombre varchar(35)

DECLARE cursor_usuario
SCROLL CURSOR
FOR
       SELECT idUsuario,Dni,Nombres
       FROM usuario

       OPEN  cursor_usuario
       FETCH NEXT FROM  cursor_usuario
              INTO @idus,
                           @dni,
                           @nombre
              WHILE @@FETCH_STATUS = 0
              BEGIN
                     UPDATE usuario
                     SET Email = LOWER(SUBSTRING(@nombre,1,4))+@dni+'@gmail.com'
                     WHERE idUsuario = @idus
                     PRINT  ' Cambio ejecutado exitosamente '

                     FETCH NEXT FROM  cursor_usuario
                           INTO @idus,
                                        @dni,
                                        @nombre

              END
       CLOSE cursor_usuario
       DEALLOCATE  cursor_usuario

-------------------------------------------------------------------------------
```

```
--RF9: Reportes de Inventario

--01 Un cursor que emite un listado de productos sin stock
     y que sus precios superen 5 y no exceda de 13.*/
DECLARE @idProducto INT
DECLARE @nombre VARCHAR(50)
DECLARE @stock INT
DECLARE @precio_venta MONEY
DECLARE @costo MONEY
DECLARE @mensaje VARCHAR(100)

DECLARE cursor_producto CURSOR
  FOR
       SELECT P.idProducto,P.nombre,P.stock,P.precio_venta,P.costo
       FROM Producto P
       WHERE (precio_venta BETWEEN 5 AND 13) AND stock IS NULL
OPEN cursor_producto
FETCH NEXT FROM cursor_producto INTO @idProducto, @nombre,@stock,@precio_venta,@costo
WHILE @@FETCH_STATUS=0
BEGIN
  SET @mensaje = cast (@idProducto as varchar(10))+' '+@nombre+' '+cast (@stock as
varchar(10))+' '+
                cast (@precio_venta as varchar(10)+' '+cast (@costo as varchar(10))
  PRINT (@mensaje)
  FETCH NEXT FROM cursor_producto
       INTO  @idProducto, @nombre,@stock,@precio_venta,@costo
END
CLOSE cursor_producto
DEALLOCATE cursor_producto
-------------------------------------------------------------------------------
--RF11: Alertas de Caducidad

--01 Nombre y fecha de caducidad de medicamente pronto a vencer

DECLARE @NombreMedicamento VARCHAR(35)
DECLARE @FechaVencimiento DATE

DECLARE medicamentosCursor CURSOR FOR
SELECT Descripcion, FechaVencimiento
FROM producto
WHERE DATEDIFF(DAY, GETDATE(), FechaVencimiento) <= 30

-- Abrir el cursor
OPEN medicamentosCursor

FETCH NEXT FROM medicamentosCursor INTO @NombreMedicamento, @FechaVencimiento
WHILE @@FETCH_STATUS = 0
BEGIN

    PRINT 'Nombre del Medicamento: ' + @NombreMedicamento
    PRINT 'Fecha de Vencimiento: ' + CONVERT(NVARCHAR(10), @FechaVencimiento, 120)

    FETCH NEXT FROM medicamentosCursor INTO @NombreMedicamento, @FechaVencimiento
END

CLOSE medicamentosCursor
DEALLOCATE medicamentosCursor
```

```
NO FUNCIONAL (17)
--RNF1: Rendimiento y Tiempo de Respuesta
--01 la creación del índice "idx_Descripcion_Producto" en la columna "Descripcion" de
la tabla "producto" tiene como objetivo mejorar la eficiencia de las consultas que buscan
productos por su descripción.
CREATE NONCLUSTERED INDEX idx_Descripcion_Producto
ON producto (Descripcion)


--02 La creación del índice "idx_Fecha_IdEmpleado_Ventas" en las columnas "Fecha" e
"idEmpleado" de la tabla "ventas"
--tiene como objetivo mejorar la eficiencia de las consultas que buscan ventas por estas
dos variables.
CREATE NONCLUSTERED INDEX idx_Fecha_IdEmpleado_Ventas
ON ventas (Fecha, idEmpleado)


--03 Esto significa que la tabla "detallecompra" estará organizada en disco de acuerdo
con el orden del índice,
--lo cual puede afectar el rendimiento de las operaciones de inserción y actualización,
--ya que los datos deben reorganizarse para mantener el orden del índice.
CREATE CLUSTERED INDEX idx_idProducto_DetalleCompra
ON detallecompra (idProducto)

--04 no agrupado para la columna de fecha de vencimiento  */
CREATE NONCLUSTERED INDEX idx_AlertasCaducidadMedicamentos
ON producto (FechaVencimiento);

--05 permitirá que las consultas que buscan empleados por nombre y cargo al mismo
tiempo*/
CREATE INDEX idx_NombreCargo
ON empleado (Nombres, Cargo);

--06 no agrupado para la columna de cantidad,costo, idventa,idProducto  */
CREATE NONCLUSTERED INDEX idx_DetalleVenta
ON detalleventa (cantidad, costo, idProducto, IdVenta);

--07 no agrupado para la columna de Apellidos de Usuario*/
CREATE NONCLUSTERED INDEX idx_Apellidos_Usuario
ON usuario (apellidos);

--08 no agrupado de cantidad y costos para detalleventa*/
CREATE NONCLUSTERED INDEX idx_Cantidad_Costo_DetalleVenta
ON detalleventa (cantidad, costo);

--09 agrupado de la tabla empleados en la columna idEmpleado*/
CREATE CLUSTERED INDEX idx_idEmpleado_Empleado
ON empleado (idEmpleado);

--10 Indice no agrupado
CREATE NONCLUSTERED INDEX IX_Cliente_Nombre
ON cliente (Nombre);

--11 Indice no agrupado
CREATE NONCLUSTERED INDEX IX_Producto_Costo
ON producto (Costo);

--12 Indice agrupado
CREATE CLUSTERED INDEX IX_Proveedor_idProveedor
ON proveedor(idProveedor);

--13 INDICE  CLUSTERED CON LA EL CAMPO SUELDO DE EMPLEADOS
```

```sql
CREATE CLUSTERED INDEX IDX_SUELDO
ON empleados(Sueldo DESC)


--14POR LA FECHA DE INICIO DE ESTAR LABURANDO EN LA EMPRESA
CREATE NONCLUSTERED INDEX IDX_FechaLaboral
ON empleados(FechaLaboral)

--15 Índice agrupado en la tabla producto para la columna idLaboratorio:
CREATE CLUSTERED INDEX idx_idLaboratorio_Producto
ON producto (idLaboratorio)

--16 Índice no agrupado de stock y decripcion para productos:
CREATE NONCLUSTERED INDEX IX_Stock_Descripcion_Producto
ON producto (Stock, Descripcion)


--17 Índice no agrupado para las columnas Nombres y Apellidos de Usuario
CREATE NONCLUSTERED INDEX idx_Nombres_Apellidos_Usuario
ON usuario (Nombres, Apellidos)
```

CREACION DE LA BASE DE DATOS Y BACKUP

```
USE master
-- ELIMINAR LA BASE DE DATOS SI EXISTE
IF EXISTS ( SELECT name
            FROM sysdatabases
            WHERE name IN ('farmacia02'))
BEGIN
    DROP DATABASE farmacia02
END
-----------------------------------------------------
-- CREAR DE BASE DE DATOS
CREATE DATABASE farmacia02 -- (SOLO EJECUTAR ESTA L◆NEA)
ON PRIMARY
(
NAME = 'farmacia02_Data',
FILENAME = 'D:₩Base_DatosII₩Proyecto_Final₩Data₩farmacia02.mdf',
SIZE = 120MB,
MAXSIZE = 900MB,
FILEGROWTH = 10%
)

LOG ON
(
NAME = 'farmacia02_Log',
FILENAME = 'D:₩Base_DatosII₩Proyecto_Final₩Data₩farmacia02.ldf',
SIZE = 100MB,
MAXSIZE = 800MB,
FILEGROWTH = 15%
)
USE farmacia02
-----------------------------------------------------------
-- CREAR EL BACKUP DE LA BASE DE DATOS FARMACIA
-----------------------------------------------------------
BACKUP DATABASE farmacia02
TO DISK = 'D:₩Base_DatosII₩Proyecto_Final₩Data₩farmacia02'
-----------------------------------------------------------
-- RESTAURAR EL BACKUP
-----------------------------------------------------------
RESTORE DATABASE farmacia02
FROM DISK = 'D:₩Base_DatosII₩Proyecto_Final₩Data₩farmacia02'
```