Base de Datos para Análisis

Caso_1: Implementación de una Base de Datos Relacional en un Entorno Laboral

Optimización del Sistema de Gestión de Ventas en una Tienda de E-commerce

Una tienda de e-commerce, **TechOnline,** ha experimentado un crecimiento significativo en sus ventas, lo que ha generado desafíos en la gestión de datos. Actualmente, la empresa almacena información en hojas de cálculo, lo que dificulta la recuperación de datos y la generación de reportes en tiempo real.
La gerencia ha decidido implementar una base de datos relacional para organizar la información de manera eficiente. Para ello, se han identificado los siguientes requisitos:

- **Almacenamiento estructurado de datos**: Se deben registrar clientes, productos y ventas.
- **Optimización de consultas**: Se debe poder obtener información sobre productos más vendidos, clientes frecuentes y ventas realizadas en un período determinado.
- **Seguridad y permisos**: Solo los gerentes pueden acceder a los reportes de ventas, mientras que los vendedores pueden registrar nuevas transacciones.

El equipo de TI debe diseñar una base de datos relacional que contemple la creación de tablas, relaciones entre ellas y consultas SQL eficientes. Como analista de datos, te han solicitado evaluar la estructura y proponer soluciones basadas en buenas prácticas de bases de datos relacionales.

Preguntas Clave:
1. ¿Qué tablas serían necesarias para modelar la base de datos de TechOnline y qué atributos deberían contener?

Las tablas necesarias serian:

- Clientes: con datos de cliente (nombre, mail, contacto, domicilio etc)
- Productos: información de productos (nombre, precio)
- Ventas: transacciones realizadas
- DestalleVentas: detalles de productos por venta
- Usuarios: empleados con rol de acceso
- Roles: define poles como vendedor, gerente

Atributos por Tabla

- Clientes
  Registra información de quienes compran en la tienda.

Atributo	Tipo	Descripción
cliente_id	INT (PK)	Identificador único
nombre	VARCHAR	Nombre completo

Atributo	Tipo	Descripción
email	VARCHAR	Correo electrónico
telefono	VARCHAR	Contacto telefónico
direccion	TEXT	Domicilio para entregas

- Productos
  Contiene información de cada producto disponible en la tienda.

Atributo	Tipo	Descripción
producto_id	INT (PK)	Identificador único
nombre	VARCHAR	Nombre del producto
precio	DECIMAL	Precio unitario
stock	INT	Unidades disponibles
categoria	VARCHAR	Clasificación (opcional para análisis)
activo	BOOLEAN	Si está disponible para la venta

- Ventas
  Representa cada transacción completada.

Atributo	Tipo	Descripción
venta_id	INT (PK)	Identificador de la venta
cliente_id	INT (FK)	Referencia al cliente
usuario_id	INT (FK)	Empleado que registra la venta
fecha_venta	DATE	Fecha de la transacción
total_venta	DECIMAL	Total facturado

- DetalleVenta
  Permite registrar múltiples productos en una venta (modelo 1:N).

Atributo	Tipo	Descripción
detalle_id	INT (PK)	Identificador del ítem
venta_id	INT (FK)	Referencia a la venta
producto_id	INT (FK)	Referencia al producto
cantidad	INT	Unidades compradas
precio_unitario	DECIMAL	Precio del producto al momento de venta

- Usuarios
  Registra a empleados que interactúan con el sistema.

Atributo	Tipo	Descripción
usuario_id	INT (PK)	Identificador del usuario
nombre	VARCHAR	Nombre completo
email	VARCHAR	Correo institucional

Atributo	Tipo	Descripción
rol_id	INT (FK)	Referencia al tipo de rol
activo	BOOLEAN	Si tiene acceso al sistema

- Roles

  Define permisos de acceso y funciones.

Atributo	Tipo	Descripción
rol_id	INT (PK)	Identificador del rol
nombre_rol	VARCHAR	"Gerente", "Vendedor"
permisos	TEXT	Descripción general de lo que puede hacer

PK = Clave Primaria FK: Clave Foránea

2. ¿Cómo establecerías las relaciones entre las tablas para evitar la duplicidad de datos y mejorar la eficiencia en las consultas?

Para evitar duplicidad y asegurar eficiencia, hay que aplicar principios de normalización, claves foráneas bien definidas, y uso estratégico de índices, separar productos, clientes y empleados en tablas propias para no repetir sus datos en cada venta y DetalleVenta permite modelar ventas con múltiples productos sin duplicar Ventas o Productos.

Relación	Tipo	Explicación
Cliente → Ventas	1:N	Un cliente puede hacer muchas ventas
Ventas → DetalleVentas	1:N	Una venta tiene múltiples productos asociados
Productos→DetalleVentas	1:N	Cada detalle se vincula a un producto
Usuario → Ventas	1:N	Cada venta es registrada por un empleado
Role → Usuarios	1:N	Cada usuario tiene un rol

3. ¿Qué tipo de consultas SQL se podrían utilizar para obtener el total de ventas en un período específico y los productos más vendidos?

```
SELECT
        SUM(total_venta) AS total_periodo,
        COUNT(*) AS cantidad_ventas
FROM Ventas
WHERE fecha_venta
BETWEEN 'fecha_inicial' AND 'fecha_final';
```

4. ¿Cómo implementarías el sistema de permisos para garantizar que los gerentes puedan acceder a reportes, pero los vendedores solo puedan registrar ventas?

En una base de datos relacional profesional, el control de permisos es clave para garantizar integridad, confidencialidad y facilidad de auditoría

Definir los perfiles funcionales como entidades separadas, asocia cada usuario a su rol mediante FK en la tabla Usuarios.

Tabla de roles

```
CREATE TABLE Roles (
    rol_id INT PRIMARY KEY,
    nombre_rol VARCHAR(50),
    descripcion TEXT );
```

Como ejemplo

```
INSERT INTO Roles VALUES (1, 'Gerente', 'Acceso total a reportes y métricas');
INSERT INTO Roles VALUES (2, 'Vendedor', 'Solo puede registrar ventas');
```

Crea una vista que solo los gerentes puedan consultar lo que permite:

- Evitar que accedan directamente a las tablas base.
- Restringir la lógica agregada a perfiles autorizados.
- Optimizar performance y seguridad.

Utilizando GRANT y REVOKE, puedes controlar acciones permitidas por rol:

- Gerentes: acceso a vistas y reportes agregados
  `GRANT SELECT ON vista_reportes TO gerente`
- Vendedores: acceso sólo a inserciones en ventas
  `GRANT INSERT ON Ventas TO vendedor;`
  `GRANT INSERT ON DetalleVenta TO vendedor`
- Denegar lectura de reportes a vendedores
  `REVOKE SELECT ON vista_reportes FROM vendedor`

Como buenas prácticas adicionales

- Auditoría: registrar qué usuario ejecutó cada acción (usuario_id en tabla Ventas).
- Triggers opcionales: validar que solo roles autorizados modifiquen ciertas tablas.
- Documentar reglas de acceso: ideal para portafolio o manual técnico.

5. ¿Cuáles son los beneficios de utilizar un motor de base de datos relacional en comparación con el almacenamiento en hojas de cálculo?

Al migrar a una base de datos relacional, la empresa podrá:

- Automatizar reporting con vistas gerenciales segmentadas por fecha, producto o cliente.
- Escalar operaciones sin riesgo de corrupción de datos.

- Delegar roles con seguridad y trazabilidad.
- Reducir tiempo en tareas operativas y mejorar la calidad analítica.
- Escalar la información que se relaciona con el negocioo

Los beneficios comparativos son los siguientes:

Beneficio	Base de datos relacional (RDBMS)	Hojas de cálculo
Integridad de datos	Relaciones bien definidas y validación automática	Propensa a errores manuales y duplicados
Consultas complejas	SQL permite filtros, agregaciones y joins robustos	Limitadas a fórmulas simples
Usuarios simultáneos	Manejo eficiente de concurrencia y bloqueos	Conflictos al editar archivos compartidos
Seguridad y permisos	Roles, GRANT/REVOKE y control de acceso por tabla/vista	Poca granularidad
Escalabilidad	Diseñada para millones de registros y múltiples conexiones	Decrece en rendimiento con muchos datos
Búsqueda y filtrado	Índices aceleran búsquedas por campos clave	Manual, lento en grandes archivos
Reporte profesional	Vistas y consultas automatizadas para dashboards	Requiere trabajo manual y macros
Automatización y API	Integración nativa con Python, BI, aplicaciones web	Limitada (VBA, scripts externos)

Caso_2 Análisis y optimización de consultas SQL en un entorno laboral

La empresa DataSoluciones S.A. es una consultora especializada en análisis de datos. Actualmente, está evaluando el desempeño de sus empleados y los ingresos generados en cada departamento. Para esto, cuentan con una base de datos donde se registran los empleados, sus salarios, los departamentos a los que pertenecen y los clientes con los que trabajan.
La empresa necesita obtener información clave sobre su fuerza laboral para tomar decisiones estratégicas. El gerente de recursos humanos ha solicitado las siguientes consultas:

1. Obtener la lista de todos los empleados con su nombre, correo electrónico y departamento.
2. Identificar los empleados cuyo salario es superior a $3,000.
3. Determinar los empleados que trabajan en los departamentos de "Ventas" o "Marketing".
4. Obtener el salario más alto, el más bajo y el salario promedio de la empresa.
5. Listar los clientes que son comunes en las bases de datos de Chile y Perú.

Tu tarea es formular las consultas SQL necesarias para obtener esta información.

Preguntas Clave:
1. ¿Cuál es la consulta SQL para obtener la lista de todos los empleados con su nombre, correo electrónico y departamento?
2. ¿Cómo podrías filtrar los empleados con un salario superior a $3,000 utilizando SQL?
3. ¿Qué sentencia SQL utilizarías para obtener los empleados de los departamentos "Ventas" y "Marketing"?
4. ¿Cómo calcularías el salario más alto, el más bajo y el salario promedio de la empresa con SQL?

DESARROLLO

```
CREATE DATABASE IF NOT EXISTS DataSoluciones;
USE DataSoluciones;

    CREATE TABLE Empleados (
       ID INT AUTO_INCREMENT PRIMARY KEY,
       Nombre VARCHAR(50),
       Correo VARCHAR(100),
       Departamento VARCHAR(50),
       Salario DECIMAL(10,2),
       Pais VARCHAR(50)
    );
    CREATE TABLE ClientesChile (
       NombreCliente VARCHAR(100)
    );
    CREATE TABLE ClientesPeru (
       NombreCliente VARCHAR(100)
```

```
);

INSERT INTO Empleados (Nombre, Correo, Departamento, Salario, Pais)
VALUES
('Andrea Ríos', 'andrea.rios@datasoluciones.com', 'Ventas', 3500.00, 'Chile'),
('Carlos Pérez', 'carlos.perez@datasoluciones.com', 'Marketing', 2800.00, 'Perú'),
('Lucía Torres', 'lucia.torres@datasoluciones.com', 'TI', 4000.00, 'Chile'),
('Manuel Ruiz', 'manuel.ruiz@datasoluciones.com', 'Ventas', 3100.00, 'Perú'),
('Sara Gómez', 'sara.gomez@datasoluciones.com', 'Finanzas', 2900.00, 'Chile'),
('David Herrera', 'david.herrera@datasoluciones.com', 'Marketing', 3200.00, 'Perú'),
('Elena Silva', 'elena.silva@datasoluciones.com', 'TI', 2500.00, 'Perú');

INSERT INTO ClientesChile (NombreCliente) VALUES
('Empresa A'),
('Empresa B'),
('Empresa C'),
('Empresa D'),
('Empresa E');

INSERT INTO ClientesPeru (NombreCliente) VALUES
('Empresa C'),
('Empresa D'),
('Empresa F'),
('Empresa G'),
('Empresa H');

-- Lista de empleados
    SELECT Nombre, Correo, Departamento
    FROM Empleados;

-- Empleados salario superior a 3000
    SELECT Nombre, Salario
    FROM Empleados
    WHERE Salario > 3000;

-- Empleados departamentos de Ventas y Marketing
    SELECT Nombre, Departamento
    FROM Empleados
    WHERE Departamento IN ('Ventas', 'Marketing');

-- Salario más alto, más bajo y salario promedio
    SELECT
      MAX(Salario) AS SalarioMaximo,
      MIN(Salario) AS SalarioMinimo,
      AVG(Salario) AS SalarioPromedio
    FROM Empleados;
```

```
-- Listar clientes Chile y Perú
        SELECT NombreCliente
        FROM ClientesChile
        WHERE NombreCliente IN (
           SELECT NombreCliente FROM ClientesPeru
        );

-- Clientes de Chile
        SELECT NombreCliente
        FROM ClientesChile;

-- Clientes de Perú
        SELECT NombreCliente
        FROM ClientesPeru;

-- Ver todos los clientes juntos indicando de qué país son
        SELECT NombreCliente, 'Chile' AS Pais
        FROM ClientesChile
        UNION -- elimina duplicados por defecto
        SELECT NombreCliente, 'Peru' AS Pais
        FROM ClientesPeru;

-- Ver todos incluso empresas comunes
        SELECT NombreCliente, 'Chile' AS Pais
        FROM ClientesChile
        UNION ALL -- muestra todo, incluyendo clientes repetidos
        SELECT NombreCliente, 'Peru' AS Pais
        FROM ClientesPeru;
```

Preguntas Clave:

1. ¿Cuál es la consulta SQL para obtener la lista de todos los empleados con su nombre, correo electrónico y departamento?

```
SELECT Nombre       as nombre_empleado,
          Correo        as correo_electronico,
       Departamento as nombre_departamento
FROM Empleados;
```

Se usa alias (as) para mejorar la legibilidad en visualización

2. ¿Cómo podrías filtrar los empleados con un salario superior a $3,000 utilizando SQL?

- Se añade una cláusula WHERE a la consulta

```
SELECT Nombre, Salario
FROM Empleados
WHERE Salario > 3000;
```

- Salario > 3000 filtra los registros según lo definido
- Se incluye el campo salario en el SELECT para que sea visible en el resultado.

3. ¿Qué sentencia SQL utilizarías para obtener los empleados de los departamentos "Ventas" y "Marketing"?

Para obtener empleados que pertenecen específicamente a los departamentos "Ventas" y "Marketing", se puede usar una cláusula WHERE con IN, que es clara y eficiente

```
SELECT Nombre, Departamento
FROM Empleados
WHERE Departamento IN ('Ventas', 'Marketing');
```

4. ¿Cómo calcularías el salario más alto, el más bajo y el salario promedio de la empresa con SQL?

Para calcular el salario más alto, el más bajo y el promedio en una empresa usando SQL, se puede utilizar funciones agregadas (MAX, MIN, AVG) en una sola consulta.

```
SELECT
   MAX(Salario) AS SalarioMaximo,
   MIN(Salario) AS SalarioMinimo,
   AVG(Salario) AS SalarioPromedio
FROM Empleados;
```

Se usa alias (as) para mejorar la legibilidad en visualización

Caso_3 Análisis y Optimización de Consultas en SQL en un Contexto Empresarial

Optimizando el Análisis de Datos en una Empresa Minorista

La empresa "RetailTech" se especializa en la venta de productos electrónicos y ha implementado un sistema de bases de datos para gestionar clientes, pedidos y productos. Actualmente, enfrentan dificultades para generar reportes eficientes que les permitan analizar las compras de sus clientes y la popularidad de sus productos.
La base de datos está estructurada de la siguiente manera:

- Clientes (id_cliente - PK, nombre, email)
- Pedidos (id_pedido - PK, id_cliente - FK, fecha, total)
- Productos (id_producto - PK, nombre, precio)
- Detalles_Pedido (id_detalle - PK, id_pedido - FK, id_producto - FK, cantidad)

El equipo de RetailTech necesita generar reportes clave para la toma de decisiones y ha solicitado tu ayuda como experto en bases de datos para formular las consultas necesarias.

Preguntas Clave

1. **Relación de Pedidos y Clientes**: Escribe una consulta SQL que muestre el nombre del cliente, la fecha del pedido y el total del pedido.

   Para obtener la relación entre pedidos y clientes, mostrando el nombre del cliente, la fecha del pedido y el total del pedido, se requiere una consulta con JOIN entre las tablas Clientes y Pedidos.

```
SELECT
   c.nombre AS nombre_cliente,
   p.fecha    AS fecha_pedido,
   p.total    AS total_pedido
FROM
   Clientes c
JOIN
   Pedidos p
ON c.id_cliente = p.id_cliente
ORDER BY
   p.fecha DESC;
```

   - Se usa JOIN para vincular Clientes con Pedidos mediante id_cliente.
   - Se seleccionan los campos relevantes para el reporte ejecutivo.
   - Se ordena por fecha descendente para mostrar los pedidos más recientes primero.

2. **Análisis de Ventas:** Obtén una lista de productos vendidos junto con la cantidad total vendida. Ordena el resultado de mayor a menor.

Para realizar un análisis de ventas por producto, se necesita sumar la cantidad vendida de cada producto a partir de la tabla Detalles_Pedido, vinculada con Productos.

```
SELECT
    pr.id_producto,
    pr.nombre            AS nombre_producto,
    SUM(dp.cantidad) AS cantidad_total_vendida
FROM
    Productos pr
JOIN
    Detalles_Pedido dp
ON pr.id_producto = dp.id_producto
GROUP BY
    pr.id_producto, pr.nombre
ORDER BY
    cantidad_total_vendida DESC;
```

- Se usa JOIN para vincular productos con sus ventas.
- SUM(dp.cantidad) agrega todas las unidades vendidas por producto.
- GROUP BY permite agrupar por producto para el análisis.
- ORDER BY DESC muestra primero los productos más vendidos.

3**. Clientes sin Compras**: Genera una consulta que muestre los clientes que no han realizado ningún pedido.

Para identificar clientes sin compras, se necesita buscar aquellos registros en la tabla Clientes que no tienen pedidos asociados en la tabla Pedidos. Esto se puede lograr con una consulta con LEFT JOIN y filtro de NULL, o con una subconsulta NOT IN.

- Versión con JOIN

```
SELECT
    c.id_cliente,
    c.nombre,
    c.email
FROM
    Clientes c
LEFT JOIN
    Pedidos p
ON c.id_cliente = p.id_cliente
WHERE
    p.id_pedido IS NULL;
```

- Versión NOT EXISTS

```
    SELECT
   c.id_cliente, c.nombre,    c.email
FROM
```

```
    Clientes c
WHERE
    NOT EXISTS (
        SELECT 1
        FROM Pedidos p
        WHERE p.id_cliente = c.id_cliente );
```

- LEFT JOIN permite incluir todos los clientes, incluso sin pedidos.
- El filtro WHERE p.id_pedido IS NULL identifica los que no tienen relación.
- La versión con NOT EXISTS es más eficiente en bases grandes, ya que evita agrupaciones innecesarias.

4. **Ventas por Producto:** Escribe una consulta que muestre el nombre de cada producto junto con el total recaudado en ventas.

Para calcular el total recaudado en ventas por producto, se necesita multiplicar la cantidad vendida (Detalles_Pedido) por el precio unitario (Productos) y luego agrupar por producto.

```
SELECT
    pr.id_producto,
    pr.nombre                          AS nombre_producto,
    SUM(dp.cantidad * pr.precio) AS total_recaudado
FROM
    Productos pr
JOIN
    Detalles_Pedido dp
ON pr.id_producto = dp.id_producto
GROUP BY
    pr.id_producto, pr.nombre
ORDER BY
    total_recaudado DESC;
```

5**. Identificación de Pedidos sin Detalles**: Encuentra los pedidos que no tienen productos asociados en la tabla Detalles_Pedido.

Para identificar pedidos sin productos asociados, se debe buscar registros en la tabla Pedidos que no tengan coincidencias en la tabla Detalles_Pedido. Esto se puede lograr con un LEFT JOIN y filtro de NULL, o con NOT EXISTS.

- Versión con JOIN

```
SELECT
    p.id_pedido,
    p.fecha,
    p.total,
    p.id_cliente
FROM
    Pedidos p
LEFT JOIN
    Detalles_Pedido dp
ON p.id_pedido = dp.id_pedido
WHERE
    dp.id_detalle IS NULL;
```

- Versión NOT EXISTS

```
SELECT
    p.id_pedido,
    p.fecha,
    p.total,
    p.id_cliente
FROM
    Pedidos p
WHERE
    NOT EXISTS (
        SELECT 1
        FROM Detalles_Pedido dp
        WHERE dp.id_pedido = p.id_pedido
    );
```

- LEFT JOIN permite incluir todos los pedidos, incluso sin detalles.
- El filtro dp.id_detalle IS NULL identifica los pedidos sin productos.
- NOT EXISTS es más eficiente en bases grandes y evita errores por duplicidad
- Estos pedidos podrían indicar errores de carga, fallas en el sistema o procesos incompletos.
- Se sugiere revisar la lógica de inserción y aplicar validaciones en el sistema para evitar pedidos vacíos.

Caso_4 Análisis de Datos en una Empresa de Retail

OptiRetail es una cadena de tiendas que vende electrónicos y electrodomésticos en varias ciudades. La gerencia ha decidido mejorar su estrategia de ventas mediante el análisis de datos. Para ello, han solicitado a su equipo de analistas que extraigan información clave de la base de datos de ventas.
La base de datos contiene la siguiente información en la tabla ventas:

- id_venta: Identificador de la venta.
- fecha: Fecha en que se realizó la venta.
- sucursal: Nombre de la sucursal donde se realizó la venta.
- vendedor: Nombre del vendedor que realizó la venta.
- producto: Nombre del producto vendido.
- cantidad_vendida: Cantidad de unidades vendidas.
- venta_total: Monto total de la venta en dólares.

Los gerentes necesitan responder las siguientes preguntas para mejorar la eficiencia operativa y las estrategias de ventas.

Preguntas Clave

0. **Tarea Previa, creación BD y tabla**

```
-- Crear base de datos
create database optiretail;

-- Usar la base de datos
use optiretail;

-- Crear tabla ventasventas
create table ventas (
   id_venta int auto_increment primary key,
   fecha date not null,
   sucursal varchar(100) not null,
   vendedor varchar(100) not null,
   producto varchar(100) not null,
   cantidad_vendida int not null,
   venta_total decimal(10,2) not null
);
```

1. **Desempeño de las sucursales:** ¿Cuántas ventas ha realizado cada sucursal? Utiliza la función COUNT() junto con GROUP BY.

Para responder a la pregunta sobre el número de ventas realizadas por cada sucursal, se usa COUNT(id_venta) junto con GROUP BY sucursal

```
SELECT   sucursal,
   COUNT(id_venta) AS cantidad_ventas
FROM   ventas
GROUP BY   sucursal
```

```
ORDER BY   cantidad_ventas DESC;
```

2. **Vendedores destacados:** ¿Cuál es el monto total vendido por cada vendedor? Utiliza SUM() con GROUP BY.

Para identificar a los vendedores destacados según el monto total vendido, se usa SUM(venta_total) junto con GROUP BY vendedor

```
SELECT  vendedor,
   SUM(venta_total) AS total_vendido
FROM  ventas
GROUP BY   vendedor
ORDER BY   total_vendido DESC;
```

3. Productos más vendidos: ¿Cuál es la cantidad total de unidades vendidas por producto? Usa SUM() y agrupa por producto.

Para identificar los productos más vendidos por cantidad de unidades, se usa SUM(cantidad_vendida) junto con GROUP BY producto

```
SELECT   producto,
   SUM(cantidad_vendida) AS unidades_vendidas
FROM  ventas
GROUP BY   producto
ORDER BY   unidades_vendidas DESC;
```

4. **Sucursales de alto rendimiento:** ¿Cuáles son las sucursales que han generado más de $50.000 en ventas totales? Utiliza HAVING para filtrar los resultados.

Para identificar las sucursales de alto rendimiento que han generado más de $50.000 en ventas totales, se usa `SUM(venta_total)` con `GROUP BY sucursal` y se filtra con `HAVING`

```
SELECT  sucursal,
   SUM(venta_total) AS ventas_totales
FROM  ventas
GROUP BY   sucursal
HAVING   SUM(venta_total) > 50000
ORDER BY ventas_totales DESC;
```

5. **Análisis del ticket promedio:** ¿Cuál es el valor promedio de venta por sucursal? Usa AVG() para calcularlo y agrúpalo por sucursal.

Para calcular el ticket promedio por sucursal, se usa AVG(venta_total) junto con GROUP BY sucursal . Esto permite evaluar el valor promedio de cada transacción en

cada tienda, una métrica clave para entender el comportamiento de compra y la eficiencia comercial

```
SELECT  sucursal,
   AVG(venta_total) AS ticket_promedio
FROM ventas
GROUP BY  sucursal
ORDER BY ticket_promedio DESC;
```

Caso_5 Aplicación de Consultas Anidadas (SubQueries) en un Contexto Empresarial

La empresa "TechSol" es una compañía de tecnología que maneja grandes volúmenes de datos sobre sus clientes, empleados y ventas. La gerencia desea optimizar sus estrategias de negocio utilizando información clave de la base de datos de la empresa. Actualmente, enfrentan los siguientes desafíos:

1. Identificación de Clientes VIP: La empresa quiere obtener una lista de clientes cuyo gasto total en los últimos seis meses ha sido superior al promedio de todos los clientes en ese mismo periodo.

2. Detección de Empleados Clave: TechSol desea conocer cuáles empleados pertenecen a los mismos departamentos que aquellos con más de 10 años de antigüedad.

3. Actualización de Precios: La empresa planea actualizar el precio de ciertos productos basándose en el promedio de ventas de los últimos tres meses.

4. Clientes Inactivos: La dirección de ventas necesita una lista de clientes que no han realizado compras en el último mes para desarrollar estrategias de reactivación.

Los analistas de datos de TechSol deben diseñar consultas SQL utilizando SubQueries para resolver estos problemas de manera eficiente.

Preguntas Clave:

1. ¿Por qué es recomendable el uso de SubQueries en los problemas planteados en el estudio de caso?

Se recomienda la utilización de SubQueries por los siguientes motivos:

- Modularidad y claridad lógica, permiten descomponer problemas complejos en partes más manejables, cada subconsulta encapsula una lógica específica (por ejemplo, calcular un promedio, filtrar por antigüedad), lo que facilita la lectura, mantenimiento y documentación.
- Comparaciones dinámicas, son ideales para comparar registros individuales con agregados (como promedios o totales), sin necesidad de crear tablas temporales o procedimientos
- Filtrado condicional avanzado, permiten filtrar registros según condiciones que dependen de otros registros, como empleados que comparten departamento con empleado experimentados, lo que se logra con subconsultas en WHERE IN, EXISTS o NOT EXISTS.
- Actualizaciones contextuales, en operaciones UPDATE, las subconsultas correlacionadas permiten ajustar valores según el contexto de cada fila, como actualizar precios según ventas recientes, evita el uso de JOIN en UPDATE, que puede generar duplicaciones si no se controla bien.
- Escalabilidad y parametrización, las subconsultas pueden ser fácilmente parametrizadas (por fechas, productos, segmentos), lo que las hace ideales para scripts reutilizables y automatizados.

2. Escribe una consulta SQL que permita identificar a los clientes VIP en base a su gasto total en los últimos seis meses.

Se requiere comparar el gasto individual con un valor agregado (promedio global),esto se resuelve con una subconsulta escalar en la cláusula HAVING

.

```
SELECT c.id_cliente, c.nombre,
      SUM(v.monto) AS gasto_total
FROM clientes c
JOIN ventas v
ON c.id_cliente = v.id_cliente
WHERE v.fecha >= DATEADD(MONTH, -6, GETDATE())
GROUP BY c.id_cliente, c.nombre
HAVING SUM(v.monto) > (
            SELECT AVG(total_cliente)
            FROM (
               SELECT SUM(v2.monto) AS total_cliente
               FROM ventas v2
               WHERE v2.fecha >= DATEADD(MONTH, -6, GETDATE())
               GROUP BY v2.id_cliente
            ) AS sub
);
```

3. Diseña una consulta SQL para listar los empleados que pertenecen a los mismos departamentos que aquellos con más de 10 años de antigüedad

Se necesita filtrar por departamentos que cumplen una condición sobre otro conjunto de empleados,ideal para una subconsulta en WHERE IN

```
SELECT e.id_empleado, e.nombre, e.departamento
FROM empleados e
WHERE e.departamento IN (
   SELECT DISTINCT departamento
   FROM empleados
   WHERE DATEDIFF(YEAR, fecha_ingreso, GETDATE()) > 10
);
```

4. Propón una consulta SQL para actualizar los precios de productos según el promedio de ventas de los últimos tres meses.

Se requiere calcular un promedio por producto y usarlo para modificar registros. Subconsulta correlacionada en UPDATE

```
UPDATE productos
SET precio = (
   SELECT AVG(v.monto)
   FROM ventas v
   WHERE v.id_producto = productos.id_producto
```

```
    AND v.fecha >= DATEADD(MONTH, -3, GETDATE())
)
WHERE id_producto IN (
  SELECT DISTINCT id_producto
  FROM ventas
  WHERE fecha >= DATEADD(MONTH, -3, GETDATE())
);
```

- UODATE indica que se va a modificar el campo precio de la tabla productos.
- Subconsulta correlacionada en SET:
  - Calcula el promedio de monto vendido () para cada producto.
  - La condición v.id_producto = productos.id_producto correlaciona la subconsulta con la fila actual de productos.
  - v.fecha >= DATEADD(MONTH, -3, GETDATE()) filtra las ventas de los últimos 3 meses.
- Filtro en WHERE IN: asegura que solo se actualicen productos que han tenido ventas en los últimos 3 meses y evita modificar productos sin datos recientes, lo que podría generar valores nulos o inconsistentes.

5. Elabora una consulta SQL que permita identificar a los clientes que no han realizado compras en el último mes.

Se requiere excluir clientes con actividad reciente. Subconsulta en NOT IN o NOT EXISTS

```
SELECT id_cliente, nombre
FROM clientes
WHERE id_cliente NOT IN (
  SELECT DISTINCT id_cliente
  FROM ventas
  WHERE fecha >= DATEADD(MONTH, -1, GETDATE())
);
```

Caso_6: Implementación de una Base de Datos en un Entorno Laboral

La empresa TechSolutions, dedicada al desarrollo de software, ha identificado problemas en la gestión de la información de sus empleados y ventas. Actualmente, los registros se manejan en hojas de cálculo, lo que genera errores y dificulta el análisis de datos. Como analista de datos, se te ha asignado la tarea de implementar una base de datos SQL para organizar esta información.
La empresa necesita:

1. Una tabla llamada Empleados con los siguientes campos:
    - ID (entero, clave primaria)
    - Nombre (texto, hasta 50 caracteres)
    - Edad (número entero)
    - FechaIngreso (fecha)
2. Una tabla llamada Ventas que almacene:
    - ID (entero, clave primaria)
    - ClienteID (entero, referencia a un cliente)
    - Monto (decimal con dos decimales)
    - FechaHoraCompra (fecha y hora de compra)
3. Insertar registros en ambas tablas para simular datos reales.
4. Modificar datos cuando sea necesario.
5. Eliminar registros obsoletos de manera segura.

Preguntas Clave:
1. ¿Cuál es la importancia de definir correctamente los tipos de datos en las tablas?
2. ¿Qué precauciones se deben tomar al eliminar registros en una base de datos?
3. Escribe el comando SQL para crear la tabla Empleados según los requerimientos.
4. Escribe un comando SQL para insertar un nuevo empleado llamado "Carlos Pérez", de 30 años, que ingresó el 1 de marzo de 2024.
5. Si la empresa decide modificar el monto de una venta con ID 5 a 1.200.000, ¿qué comando SQL se debe utilizar?

DESARROLLO

Tareas Previas

1.- Creacion de tablas

```
-- Tabla de empleados
CREATE TABLE Empleados (
   ID INT PRIMARY KEY AUTO_INCREMENT,
   Nombre VARCHAR(50),
   Edad INT,
   FechaIngreso DATE
);
```

```
-- Tabla de clientes          (Se agregó para relacionar cliente de la tabla ventas)
CREATE TABLE Clientes (
   ID INT PRIMARY KEY AUTO_INCREMENT,
```

```
    Nombre VARCHAR(50),
    Correo VARCHAR(100)
);

-- Tabla de ventas (con referencia a Clientes)
CREATE TABLE Ventas (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    ClienteID INT,
    Monto DECIMAL(10, 2),
    FechaHoraCompra DATETIME,
    FOREIGN KEY (ClienteID) REFERENCES Clientes(ID)
);
```

2.- Insertar registros en ambas tablas para simular datos reales.

```
-- Empleados
INSERT INTO Empleados VALUES ( 'Ana Pérez', 29, '2022-03-10');
INSERT INTO Empleados VALUES ( 'Carlos Soto', 35, '2021-07-22');
INSERT INTO Empleados VALUES ( 'María León', 41, '2019-11-05');

-- Clientes
INSERT INTO Clientes VALUES ( 'Juan Ramírez', 'juan.ramirez@email.com');
INSERT INTO Clientes VALUES ( 'Laura Díaz', 'laura.diaz@email.com');

-- Ventas
INSERT INTO Ventas VALUES (101, 1500.75, '2023-01-15 10:30:00');
INSERT INTO Ventas VALUES (102, 890.40, '2023-02-20 16:45:00');
```

3.- Modificar datos cuando sea necesario

```
-- Actualizar correo de un cliente
UPDATE Clientes
SET Correo = 'juan.r.new@email.com'
WHERE ID = 101;

-- Cambiar fecha de ingreso de un empleado
UPDATE Empleados
SET FechaIngreso = '2022-04-01'
WHERE ID = 1;
```

4.- Eliminar registros obsoletos de manera segura.

```
-- Eliminar venta si no tiene impacto financiero
DELETE FROM Ventas WHERE Monto = 0;
-- Eliminar empleado que ya no trabaja (ejemplo)
DELETE FROM Empleados WHERE ID = 3;
```

Preguntas Claves

1. ¿Cuál es la importancia de definir correctamente los tipos de datos en las tablas?

   Definir correctamente los tipos de datos en una base de datos SQL es fundamental para garantizar su precisión, eficiencia y seguridad.
   - Precisión y validación automática, evita errores como guardar fechas como texto, lo que impide comparaciones o cálculos temporales, por ejemplo, FechaIngreso DATE permite filtrar empleados por antigüedad sin transformaciones extra.
   - Optimización del rendimiento, SQL usa índices y estructuras internas según el tipo de dato, por ejemplo, INT para IDs permite búsquedas rápidas y operaciones lógicas eficientes.
   - Seguridad e integridad referencial, tipos mal definidos pueden permitir registros inválidos, como correos en campos numéricos, FOREIGN KEY (ClienteID) REFERENCES Clientes (ID) solo funciona si los tipos son compatibles.
   - Facilidad de mantenimiento y escalabilidad, datos limpios permiten integrar automatizaciones,visualizaciones y auditorías sin necesidad de scripts de limpieza.

2. ¿Qué precauciones se deben tomar al eliminar registros en una base de datos?

   Eliminar registros en una base de datos debe realizarse con cuidado ya que puede provocar pérdida de información crítica, errores en reportes donde la trazabilidad y seguridad son claves.
   Por lo tanto, debe tomarse las siguientes precauciones:
   - Verifica dependencias y relaciones, antes de hacer un DELETE, asegurarse de que el registro no esté referenciado por otras tablas (vía claves foráneas) usando comandos como SELECT * FROM TablaRelacionado WHERE FK = ID para verificar.
   - Documenta y justifica la eliminación, mantener un registro del motivo, fecha y usuario responsable. esto facilita auditorías y trazabilidad y guardar en una tabla de respaldo los registros eliminados.
   - Requerir autorización o doble validación, implementando controles para que la eliminación no sea accidental y en base con roles y segmentar permisos específicos
   - Evitar eliminar por condiciones ambiguas, usar filtros precisos, por ejemplo : WHERE Fecha < '2023-01-01' AND Estado = 'Inactivo' evita eliminar activos por error y siempre pruebar antes con SELECT para confirmar el subconjunto que será eliminado.
   - Considerar una eliminación lógica en vez de física, a veces es preferible añadir una columna como Eliminado BOOLEAN y marca registros sin borrarlos físicamente.
   - En operaciones masivas, hacer respaldo antes, BACKUP o copia en tabla temporal, especialmente útil si el sistema no tiene rollback automático o si no se usa BEGIN TRANSACTION.
3. Escribe el comando SQL para crear la tabla Empleados según los requerimientos.

```
-- Tabla de empleados
        CREATE TABLE Empleados (
           ID INT PRIMARY KEY AUTO_INCREMENT,
           Nombre VARCHAR(50),
           Edad INT,
           FechaIngreso DATE
        );
```

4. Escribe un comando SQL para insertar un nuevo empleado llamado "Carlos Pérez", de 30 años, que ingresó el 1 de marzo de 2024.

```
INSERT INTO Empleados VALUES (' Carlos Pérez ', 30, '2024-03-01');
```

5. Si la empresa decide modificar el monto de una venta con ID 5 a 1.200.000, ¿qué comando SQL se debe utilizar?

```
UPDATE Ventas SET Monto = 1200000.00 WHERE ID = 5;
```