Informe de Consultas en Proyecto de Base de Datos

Presentado por:

Michael Alejandro Papamija Pantoja

Docente:

Brayan Arcos

Materia:

Desarrollo de Bases de Datos

Instituto Tecnológico del Putumayo

Mocoa – Putumayo

2024

Contenido

1. INTRODUCCION

Este informe se ha elaborado con el objetivo de documentar y analizar de manera sistemática las consultas realizadas en el proyecto de bases de datos relacionales, facilitando así su comprensión y seguimiento. Las consultas se llevaron a cabo utilizando MySQL Workbench, una herramienta esencial para la gestión y manipulación de grandes volúmenes de información. Esta plataforma no solo permite ejecutar consultas SQL, sino que también proporciona un entorno visual que simplifica la interacción con la base de datos.

En el presente informe, se abordan los tipos de consultas que son consideradas fundamentales para el manejo eficiente de la información dentro de la base de datos. Estas consultas son cruciales para extraer datos relevantes y realizar análisis significativos. A través de ejemplos prácticos, se ilustrarán las diferentes operaciones que se pueden llevar a cabo, desde la selección y actualización de registros hasta la creación de nuevas tablas y la definición de relaciones entre ellas.

Además, se explorará la importancia del lenguaje SQL en el contexto de las bases de datos relacionales, destacando cómo este lenguaje permite a los usuarios interactuar con los datos almacenados, optimizando así el rendimiento y la integridad de la información. Este análisis no solo servirá como un recurso educativo para quienes deseen profundizar en el uso de SQL, sino que también proporcionará una referencia útil para el desarrollo futuro del proyecto.

2. Metodología

2.1 Herramientas Utilizadas

Para el desarrollo de este informe se emplearon las siguientes herramientas:

- **MySQL** como sistema de gestión de bases de datos (DBMS).
- **MySQL Workbench** para la creación de la base de datos, definición de tablas y relaciones, y ejecución de consultas SQL.
- **Enlace de GitHub: https://github.com/dev-Alejo24/MySQL-Michael_Papamija-.git**

2.2 Procedimientos

Los procedimientos seguidos para llevar a cabo el análisis fueron los siguientes:

1. Creación del esquema de la base de datos en MySQL Workbench, incluyendo la definición de tablas, claves primarias, foráneas, y las relaciones entre las entidades.
2. Ejecución de consultas SQL en MySQL Workbench para extraer información relevante y validar el correcto funcionamiento de la base de datos.

3. Consultas SQL

Se realizaron diversas consultas SQL para obtener datos relevantes. A continuación, se describen algunas de las consultas más importantes ejecutadas.

3.1. Consultas Complejas

3.1.1 LIKE

```sql
-- LIKE
SELECT name, price, stock
FROM products
WHERE name LIKE '%Arroz%'
   OR name LIKE '%Leche%'
ORDER BY price DESC;
```

Resultado

	name	price	stock
▶	Leche	999.99	10
	Arroz	19.99	100

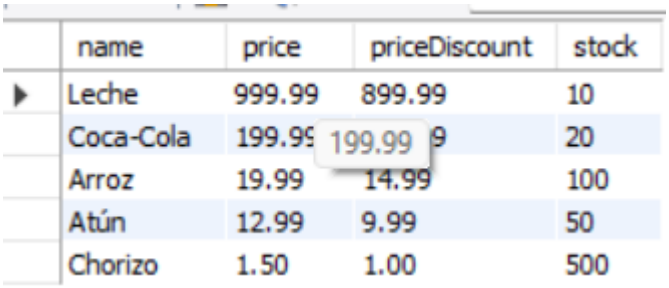
- **SELECT**: Esta cláusula selecciona las columnas name, price y stock de la tabla products.
- **FROM products**: Indica que los datos se extraen de la tabla llamada products.
- **WHERE**: Esta cláusula filtra los resultados según ciertas condiciones:
    - name LIKE '%Arroz%': Busca productos cuyo nombre contenga la palabra "Arroz" en cualquier parte del texto. El símbolo % actúa como un comodín que representa cualquier secuencia de caracteres (incluyendo ninguna).
    - OR name LIKE '%Leche%': También busca productos cuyo nombre contenga la palabra "Leche". La condición OR significa que se seleccionarán productos que cumplan al menos una de estas condiciones.
- **ORDER BY price DESC**: Los resultados se ordenan en orden descendente (DESC) según el precio, es decir, los productos más caros aparecerán primero.

Esta consulta devolverá una lista de productos que contienen "Arroz" o "Leche" en su nombre, mostrando su nombre, precio y stock, ordenados desde el más caro al más barato.

3.1.2. ORDER BY

```
-- ORDER BY DESCUENTO
SELECT `name`, price, priceDiscount, stock
FROM products
WHERE priceDiscount IS NOT NULL
  AND priceDiscount < price
ORDER BY (price - priceDiscount) DESC;
```

RESULTADO:



name	price	priceDiscount	stock
Leche	999.99	899.99	10
Coca-Cola	199.99	199.99	20
Arroz	19.99	14.99	100
Atún	12.99	9.99	50
Chorizo	1.50	1.00	500

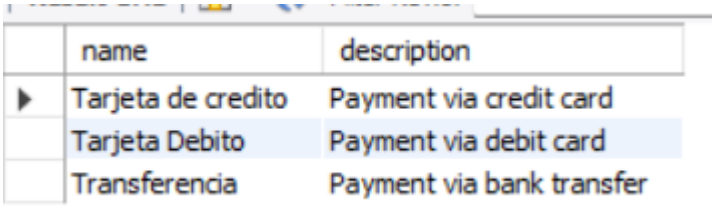
Explicación

- **SELECT**: Esta cláusula selecciona las columnas name, price, priceDiscount y stock de la tabla products.
- **FROM products**: Indica que los datos provienen de la tabla llamada products.
- **WHERE**: Filtra los resultados con las siguientes condiciones:
  - priceDiscount IS NOT NULL: Asegura que solo se incluyan productos que tienen un precio de descuento definido (no nulo).
  - AND priceDiscount < price: Filtra para incluir solo aquellos productos donde el precio de descuento es menor que el precio original, lo que implica que hay un descuento aplicable.
- **ORDER BY (price - priceDiscount) DESC**: Los resultados se ordenan en orden descendente según el valor del descuento (calculado como la diferencia entre el precio original y el precio con descuento). Esto significa que los productos con mayor descuento aparecerán primero.

3.1.3. LIKE, ORDER BY, ASC

```
-- METODO DE PAGO
SELECT `name`, `description`
FROM paymentMethod
WHERE name LIKE 'T%'
   OR description LIKE '%Transferencia%'
ORDER BY name ASC;
```

RESULTADO



	name	description
▶	Tarjeta de credito	Payment via credit card
	Tarjeta Debito	Payment via debit card
	Transferencia	Payment via bank transfer

Explicación

- **SELECT**: Esta cláusula selecciona las columnas name y description de la tabla paymentMethod.
- **FROM paymentMethod**: Indica que los datos provienen de la tabla llamada paymentMethod.
- **WHERE**: Filtra los resultados con las siguientes condiciones:
    - name LIKE 'T%': Busca métodos de pago cuyo nombre comience con la letra "T". El símbolo % indica que puede haber cualquier secuencia de caracteres después de "T".
    - OR description LIKE '%Transferencia%': También incluye métodos de pago cuya descripción contenga la palabra "Transferencia" en cualquier parte del texto.
- **ORDER BY name ASC**: Los resultados se ordenan en orden ascendente (ASC) según el nombre del método de pago.

3.2. Consultas Avanzadas

Las siguientes ocsultas son un poco mas avanzadas ya que se hace uso de funciones pertenecientes a SQL en el filtrado de datos.

3.2.1. NOMBRE COMPLETO

```
-- NOMBRE COMPPLETO
SELECT people.id, CONCAT(UPPER(firstName), ' ', UPPER(SUBSTRING(lastNamePaternal, 1, 1)), '. ', UPPER(lastNamePaternal)) AS
fullName FROM people;
```

RESULTADO

id	fullName
1	JOHN S. SMITH
2	JANE J. JOHNSON
3	MARIA L. LOPEZ
4	PETER B. BROWN
5	LINDA W. WHITE

3.2.2. ROUND

```
-- REDONDEO DE SALARIO
SELECT id, ROUND(salary, 2) AS roundedSalary, ABS(salary - 3000) AS salaryDifference
FROM people
WHERE salary > 2000;
```

RESULTADO

id	roundedSalary	salaryDifference
1	5000	2000
2	3000	0
3	2500	500
4	3200	200
5	4000	1000

Explicación

- **SELECT**: Esta cláusula selecciona el id de la tabla people y crea una nueva columna llamada fullName.

- **CONCAT**: Esta función combina varias cadenas de texto:
    - UPPER(firstName): Convierte el nombre (firstName) a mayúsculas.
    - UPPER(SUBSTRING(lastNamePaternal, 1, 1)): Toma la primera letra del apellido paterno (lastNamePaternal), la convierte a mayúscula y le añade un punto ('.').
    - UPPER(lastNamePaternal): Convierte el apellido paterno completo a mayúsculas.
- **AS fullName**: Renombra la columna resultante como fullName.

3.2.3. SALARY – JOB POSITION

```
-- SALARIO SEGUN PUESTO DE TRABAJO
SELECT idJobPosition, SUM(salary) AS totalSalary, AVG(salary) AS averageSalary
FROM people
GROUP BY idJobPosition;
```

RESULTADO

idJobPosition	totalSalary	averageSalary
1	5000	5000
2	3000	3000
3	2500	2500
4	3200	3200
5	4000	4000

Explicación

- **SELECT**: Esta cláusula selecciona el identificador del puesto de trabajo (idJobPosition), suma los salarios y calcula el promedio de salarios.
- **SUM(salary)**: Calcula la suma total de los salarios para cada puesto de trabajo y lo renombra como totalSalary.
- **AVG(salary)**: Calcula el salario promedio para cada puesto de trabajo y lo renombra como averageSalary.
- **GROUP BY idJobPosition**: Agrupa los resultados por el identificador del puesto de trabajo. Esto significa que se generará una fila por cada puesto de trabajo único en la tabla.

3.2.4. CANTIDAD EMPLEADOS POR PUESTO

```
-- CANT EMPLEADOS POR PUESTO
SELECT jp.name AS position, COUNT(p.id) AS totalEmployees
FROM people p
JOIN jobPosition jp ON p.idJobPosition = jp.id
GROUP BY jp.name;
```

RESULTADO

position	totalEmployees
Gerente	1
Cajero	1
Socio de ventas	1
Control de inventario	1
RR-HH	1

Explicación

- **SELECT**: Esta cláusula selecciona el nombre del puesto (jp.name) y cuenta la cantidad total de empleados (COUNT(p.id)).
- **JOIN jobPosition jp ON p.idJobPosition = jp.id**: Realiza una unión entre las tablas people (alias p) y jobPosition (alias jp) utilizando la relación entre idJobPosition en people y id en jobPosition.
- **COUNT(p.id)**: Cuenta cuántos empleados hay para cada puesto de trabajo.
- **GROUP BY jp.name**: Agrupa los resultados por el nombre del puesto. Esto genera una fila por cada tipo de puesto en la tabla

3.2.5. MAX – MIN PRODUCTS BY CATEGORY

```
-- MAX-MIN PRODUCTS AGRUPADOS POR CATEGORIAS
SELECT idCategory, MAX(price) AS highestPrice, MIN(price) AS lowestPrice
FROM products
GROUP BY idCategory;
```

RESULTADO:

idCategory	highestPrice	lowestPrice
1	999.99	999.99
2	19.99	19.99
3	1.50	1.50
4	12.99	12.99
5	199.99	199.99

Explicación

- **SELECT**: Esta cláusula selecciona el identificador de categoría (idCategory), así como el precio máximo y mínimo.
- **MAX(price)**: Calcula el precio más alto dentro de cada categoría y lo renombra como highestPrice.
- **MIN(price)**: Calcula el precio más bajo dentro de cada categoría y lo renombra como lowestPrice.
- **GROUP BY idCategory**: Agrupa los resultados por categoría. Esto significa que se generará una fila por cada categoría única en la tabla.

3.3. Consultas INNER JOIN

En las siguientes consultas se usa lo aprendido anteriormente, esta vez se implementa o agrega el foltrado de información con INNER JOIN en las consultas SQL.

3.3.1. Lister Product's Category

```
-- Listar productos por categoría
SELECT pc.`name` AS category, p.`name` AS product, p.price
FROM products p
INNER JOIN productsCategories pc ON p.idCategory = pc.id
ORDER BY pc.`name`, p.`name`;
```

RESULTADO

category	product	price
Bebidas	Coca-Cola	199.99
Embutidos	Chorizo	1.50
Enlatados	Atún	12.99
Granos	Arroz	19.99
Lacteos	Leche	999.99

Explicación

- **SELECT**: Selecciona el nombre de la categoría (pc.name), el nombre del producto (p.name) y el precio del producto (p.price).
- **FROM products p**: Indica que la tabla principal es products, usando p como alias.
- **INNER JOIN productsCategories pc ON p.idCategory = pc.id**: Realiza un INNER JOIN con la tabla productsCategories, donde se unen por el campo idCategory en products y el campo id en productsCategories.
- **ORDER BY pc.name, p.name**: Ordena los resultados primero por el nombre de la categoría y luego por el nombre del producto.

3.3.2. Productos más caros por categoría

```
-- Mostrar productos más caros por categoría
SELECT pc.`name` AS category, MAX(p.price) AS maxPrice FROM
products p INNER JOIN productsCategories pc ON pc.id = p.idCategory
GROUP BY pc.`name`;
```

RESULTADO

category	maxPrice
Lacteos	999.99
Granos	19.99
Embutidos	1.50
Enlatados	12.99
Bebidas	199.99

Explicación

- **SELECT**: Selecciona el nombre de la categoría (pc.name) y el precio máximo de los productos (MAX(p.price)).
- **FROM products p**: Indica que la tabla principal es products.
- **INNER JOIN productsCategories pc ON pc.id = p.idCategory**: Une las tablas products y productsCategories.
- **GROUP BY pc.name**: Agrupa los resultados por nombre de categoría.

3.3.3. Nombre corto de Rol Empleado

```
-- 2.shift,id,short name y user name de empleado
SELECT p.id, CONCAT(UPPER(SUBSTRING(firstName,1,1)),'.',
 UPPER(lastNamePaternal)) AS shortName, s.`name`, u.`name`
FROM people p
INNER JOIN shift s ON s.id = p.idShift
INNER JOIN users u ON u.id = p.idUser
INNER JOIN usersRoles us ON us.idUser = u.id
INNER JOIN roles r ON r.id = us.idRole
WHERE r.`name` = 'Empleado';
```

RESULTADO

id	shortName	name	name
4	P.BROWN	Fin de semana	Peter Johnson

Explicación

- **SELECT**: Crea un nombre corto combinando la inicial del primer nombre con el apellido paterno y selecciona el salario.

- **FROM people p**: Indica que la tabla principal es people.
- **INNER JOIN users u ON p.idUser = u.id**: Une la tabla people con users.
- **INNER JOIN usersRoles us ON us.idUser = u.id**: Une la tabla users con usersRoles.
- **INNER JOIN roles r ON r.id = us.idRole**: Une la tabla usersRoles con roles.
- **WHERE ... AND r.name = 'Empleado'**: Filtra para incluir solo empleados cuyo salario es mayor que el promedio.

3.3.4. Tipo de documento de cada persona

```
-- tipo de documento de people
SELECT p.id, CONCAT(UPPER(firstName), ' ', UPPER(SUBSTRING(lastNamePaternal, 1, 1)),
 '. ', UPPER(lastNameMaternal)) AS
fullName, dt.`name`
 FROM people p
 INNER JOIN documentType dt ON p.idDocumentType = dt.id;
```

RESULTADO

id	fullName	name
1	JOHN S. DOE	Cedula Ciudadania
2	JANE J. SMITH	Cedula Extranjeria
3	MARIA L. GARCIA	Tarjeta de Identidad
4	PETER B. JOHNSON	Pasaporte
5	LINDA W. BROWN	Prmiso de Residencia

Explicación

- **SELECT**: Selecciona el ID de la persona (p.id), su nombre completo y el tipo de documento.
- **FROM people p**: Indica que la tabla principal es people.
- **INNER JOIN documentType dt ON p.idDocumentType = dt.id**: Une las tablas people y documentType.

Resultado

Devuelve una lista de personas junto con su nombre completo y tipo de documento.

3.3.5. Methods de Pago mas Populares

```
-- Lista de métodos de pago populares
SELECT pm.`name`, COUNT(pd.paymentAmount) AS totalPayments FROM
paymentMethod pm
INNER JOIN paymentDetail pd ON pd.idPaymentMethod = pm.id
GROUP BY pm.`name`
ORDER BY totalPayments DESC;
```

RESULTADO

name	totalPayments
Tarjeta de credito	1
Tarjeta Debito	1
Efectivo	1
Transferencia	1
PayPal	1

Explicación

- **SELECT**: Selecciona el nombre del método de pago y cuenta los pagos realizados.
- **FROM paymentMethod pm**: Indica que la tabla principal es paymentMethod.
- **INNER JOIN paymentDetail pd ON pd.idPaymentMethod = pm.id**: Une las tablas basándose en el método de pago.
- **GROUP BY pm.name**: Agrupa los resultados por método de pago.
- **ORDER BY totalPayments DESC**: Ordena los resultados desde el método más popular al menos popular.

Resultado

Devuelve una lista de métodos de pago junto con la cantidad total de pagos realizados para cada uno.

3.3.6. Empleados con major Salario

```
-- Empleados con salario mayor a la media
SELECT CONCAT(UPPER(SUBSTRING(firstName,1,1)),'.', UPPER(lastNamePaternal)) AS shortName, p.salary
FROM people p INNER JOIN jobPosition jp ON jp.id = p.idJobPosition
WHERE p.salary > (SELECT AVG(salary) FROM people);
```

RESULTADO

shortName	salary
J.SMITH	5000
L.WHITE	4000

Explicación

Esta consulta es similar a una anterior. Selecciona un nombre corto para empleados cuyo salario es mayor al promedio.

Resultado

Devuelve una lista similar a la anterior pero asegurando que solo se incluyan empleados

3.3.7. Proveedores con mas productos

```
-- Lista de proveedores con más productos:
SELECT sp.idSupplier, COUNT(sp.idProduct) AS numProducts, r.`name` AS rol
FROM users u
INNER JOIN usersRoles us ON (us.idUser = u.id)
INNER JOIN roles r ON (r.id = us.idRole)
INNER JOIN supplierProduct sp ON sp.idSupplier = u.id
WHERE r.`name` = 'Proveedor'
GROUP BY sp.idSupplier
ORDER BY numProducts DESC;
```

RESULTADO

idSupplier	numProducts	rol
3	2	Proveedor

Explicación

- **SELECT**: Selecciona el ID del proveedor y cuenta cuántos productos tiene.
- **FROM users u**: Indica que la tabla principal es users.
- **INNER JOIN ... WHERE ... GROUP BY ... ORDER BY ...**: Realiza varias uniones para filtrar solo proveedores y agrupar por ID del proveedor.

Resultado

Devuelve una lista de proveedores junto con la cantidad total de productos que tienen, ordenada desde aquellos con más productos hasta menos.

3.3.8. Empleado con salario alto por posición

```sql
-- Empleados con salarios más altos por posición:
SELECT jp.id, jp.`name` AS pos, p.firstName, p.lastNamePaternal, MAX(p.salary) AS maxSalary
FROM jobPosition jp
INNER JOIN people p ON jp.id = p.idJobPosition
INNER JOIN users u ON p.idUser = u.id
INNER JOIN usersRoles us ON us.idUser = u.id
INNER JOIN roles r ON us.idRole = r.id
WHERE r.`name` = 'Empleado'
GROUP BY jp.id, jp.`name`, p.firstName, p.lastNamePaternal
ORDER BY maxSalary DESC;
```

RESULTADO

id	pos	firstName	lastNamePaternal	maxSalary
4	Control de inventario	Peter	Brown	3200

Explicación

Esta consulta busca obtener los salarios más altos agrupados por puesto:

- Agrupa por puesto y selecciona el salario máximo para cada posición.

Resultado

Devuelve una lista detallada mostrando cada puesto junto al empleado que tiene el salario más alto en ese puesto. En resumen, estas consultas utilizan INNER JOIN para combinar datos relevantes entre múltiples tablas en una base de datos relacional. Cada consulta está diseñada para extraer información específica sobre productos, empleados o métodos de pago basándose en relaciones definidas entre las tablas.

3.4. Consultas RIGTH JOIN

Las consultas **RIGHT JOIN** en SQL son utilizadas para combinar datos de dos o más tablas, priorizando los registros de la tabla de la derecha. Este tipo de unión asegura que

todas las filas de la tabla derecha se incluyan en el resultado, incluso si no hay coincidencias en la tabla izquierda.

3.4.1. Obtener Roles y Usuarios

```sql
-- roles y ususarios
SELECT r.name, u.name
FROM usersRoles ur
RIGHT JOIN roles r ON ur.idRole = r.id
RIGHT JOIN users u ON ur.idUser = u.id;
```

1. **Tablas Involucradas**:
    - usersRoles (alias ur): Esta tabla probablemente relaciona usuarios con roles.
    - roles (alias r): Contiene información sobre los roles disponibles.
    - users (alias u): Contiene información sobre los usuarios.
2. **RIGHT JOIN**:
    - El primer RIGHT JOIN une la tabla roles con usersRoles. Esto significa que se seleccionarán todos los roles, incluso si no hay usuarios asociados a ellos. Si un rol no tiene usuarios, los campos correspondientes de la tabla usersRoles serán nulos.
    - El segundo RIGHT JOIN une la tabla users con el resultado anterior. Esto asegura que se incluyan todos los usuarios, incluso si no tienen un rol asignado.
3. **Resultado**:
    - La consulta devolverá una lista de nombres de roles y nombres de usuarios. Si un rol no tiene usuarios o un usuario no tiene un rol, se mostrarán valores nulos en las columnas correspondientes.

3.4.2. Métodos de pago

```
-- metodos de pago
SELECT pm.name, pd.paymentAmount
FROM paymentMethod pm
RIGHT JOIN paymentDetail pd ON pd.idPaymentMethod = pm.id;
```

1. **Tablas Involucradas**:
    - paymentMethod (alias pm): Contiene información sobre los métodos de pago disponibles.
    - paymentDetail (alias pd): Contiene detalles sobre los pagos realizados, incluyendo el método de pago utilizado.
2. **RIGHT JOIN**:
    - Aquí, el RIGHT JOIN une la tabla paymentDetail con la tabla paymentMethod. Esto significa que se seleccionarán todos los detalles de pago, incluso si no hay un método de pago asociado. Si un detalle de pago no tiene un método correspondiente, los campos de la tabla paymentMethod serán nulos.
3. **Resultado**:
    - La consulta devolverá una lista de nombres de métodos de pago y sus montos correspondientes. Si un detalle de pago no tiene un método asociado, el nombre del método será nulo.

3.5. Consultas LEFT JOIN

Las consultas **LEFT JOIN** en SQL son utilizadas para combinar datos de dos tablas, priorizando los registros de la tabla izquierda. Este tipo de unión asegura que todas las filas de la tabla izquierda se incluyan en el resultado, incluso si no hay coincidencias en la tabla derecha.

3.5.1. Numeros de telefono

```
-- numeros de telefono
SELECT p.firstName, pp.phone
FROM people p
LEFT JOIN peoplePhone pp ON p.id = pp.idPeople;
```

Explicación

- **SELECT**: Selecciona el primer nombre (p.firstName) y el número de teléfono (pp.phone).
- **FROM people p**: Indica que la tabla principal es people.
- **LEFT JOIN peoplePhone pp ON p.id = pp.idPeople**: Realiza un LEFT JOIN con la tabla peoplePhone, donde se unen por el campo idPeople. Esto permite incluir todos los registros de la tabla people, incluso aquellos que no tienen números de teléfono asociados.

3.5.2. Productos con detalles

```
-- productos con detalles
SELECT p.name, pd.detailName
FROM products p
LEFT JOIN productDetails pd ON p.id = pd.idProduct;
```

Explicación

- **SELECT**: Selecciona el nombre del producto (p.name) y el nombre del detalle del producto (pd.detailName).
- **FROM products p**: Indica que la tabla principal es products.
- **LEFT JOIN productDetails pd ON p.id = pd.idProduct**: Realiza un LEFT JOIN con la tabla productDetails, donde se unen por el campo idProduct. Esto asegura que todos los productos se incluyan en los resultados, incluso si no tienen detalles asociados.

Resultado

Devuelve una lista de productos junto con sus detalles. Si un producto no tiene detalles registrados, aparecerá en la lista con el campo correspondiente al detalle como NULL.

3.6. Consultas CROSS JOIN

El **CROSS JOIN** en SQL es una operación que combina todas las filas de una tabla con todas las filas de otra tabla, generando lo que se conoce como un **producto cartesiano**. Esto significa que si la primera tabla tiene n*n* filas y la segunda tabla tiene m*m* filas, el resultado del CROSS JOIN contendrá n×*m* filas.

3.6.1. Metodo de pago y proveedor

```
-- idSippler and paymentMethod
SELECT sp.idSupplier, pm.name AS PaymentMethod
FROM supplierProduct sp
CROSS JOIN paymentMethod pm;
```

Explicación

- **SELECT**: Selecciona el ID del proveedor (sp.idSupplier) y el nombre del método de pago (pm.name).
- **FROM supplierProduct sp**: Indica que la tabla principal es supplierProduct, usando sp como alias.
- **CROSS JOIN paymentMethod pm**: Realiza un CROSS JOIN con la tabla paymentMethod, usando pm como alias. Esto también generará un producto cartesiano entre las dos tablas.

Resultado

El resultado de esta consulta será una lista que muestra todas las combinaciones posibles entre los IDs de los proveedores y los métodos de pago. Si hay, por ejemplo, 4 proveedores y 6 métodos de pago, la consulta devolverá un total de 24 filas (4 x 6), donde cada combinación se muestra en una fila separada.

3.7. Consultas ALL-ANY

3.7.1. Productos mas caros (ALL)

ALL: Se utiliza para comparar un valor con todos los valores devueltos por una subconsulta. La condición se cumple solo si el valor es mayor (o menor) que todos esos valores.

```sql
-- 1. Encontrar productos más caros
SELECT *
FROM products
WHERE price > ALL (
    SELECT price
    FROM products
    WHERE idCategory = 2
);
```

Explicación

- **SELECT ***: Selecciona todas las columnas de la tabla products.
- **FROM products**: Indica que los datos provienen de la tabla products.
- **WHERE price > ALL (...)**: Filtra los resultados para incluir solo aquellos productos cuyo precio es mayor que todos los precios devueltos por la subconsulta.
    - **Subconsulta**:
        - **SELECT price FROM products WHERE idCategory = 2**: Esta subconsulta selecciona los precios de todos los productos que pertenecen a la categoría con idCategory = 2.

Resultado

La consulta devolverá todos los productos cuyo precio sea mayor que el precio más alto de todos los productos en la categoría especificada (idCategory = 2). Si no hay productos en esa categoría, la subconsulta no devolverá filas, y el resultado será vacío.

3.7.2. Factutas con monto mayor que pago individual (ANY)

ANY: Se utiliza para comparar un valor con al menos uno de los valores devueltos por una subconsulta. La condición se cumple si el valor es mayor (o menor) que al menos uno de esos valores.

```sql
-- 2. Encontrar facturas
SELECT *
FROM invoices
WHERE totalAmount > ANY (
    SELECT paymentAmount
    FROM payment
);
```

Explicación

- **SELECT ***: Selecciona todas las columnas de la tabla invoices.
- **FROM invoices**: Indica que los datos provienen de la tabla invoices.
- **WHERE totalAmount > ANY (...)**: Filtra los resultados para incluir solo aquellas facturas cuyo monto total (totalAmount) es mayor que al menos uno de los montos de pago devueltos por la subconsulta.
    - **Subconsulta**:
        - **SELECT paymentAmount FROM payment**: Esta subconsulta selecciona todos los montos de pago realizados en la tabla payment.

Resultado

La consulta devolverá todas las facturas cuyo monto total sea mayor que al menos uno de los montos de pago registrados en la tabla payment. Si no hay pagos registrados, entonces la subconsulta no devolverá filas, y el resultado será vacío

3.8. Consultas EXISTS

El operador **EXISTS** se utiliza para verificar si una subconsulta devuelve alguna fila. Es particularmente útil cuando se desea comprobar la existencia de registros relacionados sin necesidad de contar o recuperar datos específicos de esas filas. Si la subconsulta encuentra al menos una coincidencia, el resultado es verdadero y se incluyen los registros correspondientes de la consulta principal. Esto puede resultar en consultas más eficientes, especialmente cuando solo se necesita saber si existe alguna relación.

3.8.1. Tiendas con al menos un producto en stock

```sql
-- Encontrar tiendas que tiener
SELECT *
FROM store s
WHERE EXISTS (
    SELECT 1
    FROM storeProducts sp
    WHERE sp.idStore = s.id
    AND sp.stock > 0
);
```

Explicación

- **SELECT ***: Selecciona todas las columnas de la tabla store.
- **FROM store s**: Indica que los datos provienen de la tabla store, usando s como alias.
- **WHERE EXISTS (...)**: Esta cláusula verifica si existe al menos una fila en el resultado de la subconsulta. Si la subconsulta devuelve al menos una fila, la condición se evalúa como verdadera.
    - **Subconsulta**:
        - **SELECT 1 FROM storeProducts sp**: Selecciona un valor constante (1) de la tabla storeProducts, que se utiliza para verificar la existencia de filas.
        - **WHERE sp.idStore = s.id AND sp.stock > 0**: Filtra los resultados de storeProducts para encontrar aquellos productos que pertenecen a la tienda actual (s.id) y que tienen un stock mayor que cero.

Resultado

La consulta devolverá todas las tiendas que tienen al menos un producto en stock. Si no hay productos en stock para una tienda, esa tienda no aparecerá en los resultados.

3.8.2. Facturas con al menos una relación con invoiceDetails

```sql
-- encontrar todas las facturas (
SELECT *
FROM invoices i
WHERE EXISTS (
    SELECT 1
    FROM invoiceDetails id
    WHERE id.idInvoice = i.id
);
```

);

Explicación

- **SELECT ***: Selecciona todas las columnas de la tabla store.
- **FROM store s**: Indica que los datos provienen de la tabla store, usando s como alias.
- **WHERE EXISTS (...)**: Esta cláusula verifica si existe al menos una fila en el resultado de la subconsulta. Si la subconsulta devuelve al menos una fila, la condición se evalúa como verdadera.
    - **Subconsulta**:
        - **SELECT 1 FROM storeProducts sp**: Selecciona un valor constante (1) de la tabla storeProducts, que se utiliza para verificar la existencia de filas.
        - **WHERE sp.idStore = s.id AND sp.stock > 0**: Filtra los resultados de storeProducts para encontrar aquellos productos que pertenecen a la tienda actual (s.id) y que tienen un stock mayor que cero.

Resultado

La consulta devolverá todas las tiendas que tienen al menos un producto en stock. Si no hay productos en stock para una tienda, esa tienda no aparecerá en los resultados.

4. Subconsultas

Las **subconsultas** en bases de datos SQL son consultas anidadas dentro de otra consulta más grande. Se utilizan para realizar operaciones más complejas, donde el resultado de la subconsulta se utiliza como entrada en la consulta principal.

4.1. Subconsultas SELECT

4.1.1. Mostrar cada factura con el total de productos en ella

```sql
-- 1. Mostrar cada factura con el total de productos
SELECT i.*,
COUNT(idProduct) AS numProductosDiferentes
FROM invoices i
LEFT JOIN invoiceDetails id ON i.id = id.idInvoice
GROUP BY i.id;
```

Explicación:

1. **Tablas involucradas**:
   - invoices (**i**): Contiene información sobre las facturas.
   - invoiceDetails (**id**): Relaciona cada factura con los productos que incluye.
2. **Objetivo**: Mostrar cada factura (i.*) junto con el número de productos **diferentes** que incluye.
3. **Desglose**:
   - La consulta utiliza una cláusula LEFT JOIN para combinar las facturas (invoices) con los detalles de las facturas (invoiceDetails) a través de la relación idInvoice.
   - La función agregada COUNT(idProduct) cuenta cuántos productos (de la tabla invoiceDetails) están asociados con cada factura.
   - La cláusula GROUP BY i.id asegura que el conteo se realice para cada factura (i.id).

4.1.2. Aadgds

```sql
-- 2. Mostrar cada producto con su última fecha de venta
SELECT p.*,
    (SELECT MAX(invoiceDate)
     FROM invoices i
     INNER JOIN invoiceDetails id ON i.id = id.idInvoice
     WHERE id.idProduct = p.id) AS ultimaventa
FROM products p;
```

explicación:

1. **Tablas involucradas**:
   - products (**p**): Contiene información sobre los productos.

- invoices (**i**): Contiene información sobre las facturas.
- invoiceDetails (**id**): Relaciona productos con facturas.

2. **Objetivo**: Mostrar cada producto (p.*) con la fecha más reciente (MAX(invoiceDate)) en la que fue vendido.
3. **Desglose**:
   - Para cada producto (p), se ejecuta una subconsulta en la columna calculada ultimaventa.
   - La subconsulta:
     - Consulta la tabla invoices para encontrar la fecha más reciente (MAX(invoiceDate)) de una factura que incluya ese producto (id.idProduct = p.id).
     - Utiliza un INNER JOIN entre invoices y invoiceDetails para asociar productos con facturas.
   - El resultado de la subconsulta devuelve un único valor: la última fecha de venta para ese producto.
4. **Cómo funciona**:
   - Para cada fila de la tabla products (de la consulta principal), la subconsulta se ejecuta de manera independiente.
   - Si un producto no tiene ventas, el valor de ultimaventa será NULL.

4.2. Subconsultas FROM

4.2.1. Promedio de ventas de cliente con datos personales

```sql
-- 1. Obtener el promedio de ventas por cliente junto con su
SELECT p.*, avgSales.promedioVentas
FROM people p
INNER JOIN (
    SELECT idCustomer, AVG(totalAmount) as promedioVentas
    FROM invoices
    GROUP BY idCustomer
) avgSales ON p.id = avgSales.idCustomer;
```

Explicación:

1. **Tablas involucradas**:
   - people (**p**): Contiene los datos personales de las personas (clientes).
   - invoices: Contiene la información de las facturas, incluyendo los clientes y los montos totales.
2. **Objetivo**: Mostrar los datos personales de cada cliente junto con el promedio de ventas (facturas) que ha generado.
3. **Desglose**:
   - Subconsulta en FROM:
     - Selecciona el identificador del cliente (idCustomer) y calcula el promedio del monto total de las facturas (AVG(totalAmount)).

        - Agrupa los resultados por cliente (GROUP BY idCustomer) para obtener un promedio por cada uno.
        - El resultado es una tabla temporal llamada avgSales que contiene idCustomer y promedioVentas.
    - Consulta principal:
        - Realiza un INNER JOIN entre la tabla people (p) y la tabla temporal avgSales, relacionando p.id con avgSales.idCustomer.
        - Devuelve todos los datos de las personas (p.*) junto con el promedio de ventas.
4. **Resultado**: Una tabla donde cada fila corresponde a un cliente, mostrando su información personal y su promedio de ventas.

4.2.2. Producto mas vendido por categoría

```sql
SELECT pc.`name` as categoria, topProducts.*
FROM productsCategories pc
INNER JOIN (
    SELECT p.idCategory, p.`name`, COUNT(id.idProduct) as vecesVendido
    FROM products p
    INNER JOIN invoiceDetails id ON p.id = id.idProduct
    GROUP BY p.id, p.`name`, p.idCategory
) topProducts ON pc.id = topProducts.idCategory;
```

Explicación:

1. **Tablas involucradas**:
    - productsCategories (**pc**): Contiene los nombres de las categorías de productos.
    - products (**p**): Contiene información sobre los productos, incluyendo su categoría (idCategory).
    - invoiceDetails (**id**): Relaciona productos con facturas para registrar qué productos se han vendido.
2. **Objetivo**: Mostrar los productos más vendidos junto con el nombre de su categoría.
3. **Desglose**:
    - Subconsulta en FROM:
        - Selecciona la categoría del producto (p.idCategory), el nombre del producto (p.name) y el número de veces que fue vendido (COUNT(id.idProduct)).
        - Realiza un JOIN entre products y invoiceDetails para relacionar los productos con las facturas.
        - Agrupa por el ID del producto, su nombre, y su categoría (GROUP BY p.id, p.name, p.idCategory).
        - El resultado es una tabla temporal llamada topProducts que contiene idCategory, name, y vecesVendido.

- Consulta principal:
  - Realiza un INNER JOIN entre productsCategories (pc) y topProducts, relacionando pc.id con topProducts.idCategory.
  - Devuelve el nombre de la categoría (pc.name) y todos los datos de topProducts.

4. **Resultado**: Una tabla donde cada fila muestra una categoría de producto y los productos más vendidos en esa categoría, incluyendo el número de veces vendidos.

4.3. Subconsultas WHERE

4.3.1. Buscar productos que nunca se han vendido

```sql
-- 1. Encontrar productos que nunca se han vendido
SELECT p.*
FROM products p
LEFT JOIN invoiceDetails id ON p.id = id.idProduct
WHERE id.idProduct IS NULL;
```

xplicación:

1. **Tablas involucradas**:
   - products (**p**): Contiene información de los productos.
   - invoiceDetails (**id**): Relaciona productos con facturas, indicando qué productos se han vendido.
2. **Objetivo**: Encontrar los productos que no aparecen en ninguna factura, es decir, productos que nunca han sido vendidos.
3. **Desglose**:
   - Realiza un LEFT JOIN entre products y invoiceDetails para incluir todos los productos, incluso aquellos que no tienen registros en invoiceDetails.
   - La cláusula WHERE id.idProduct IS NULL filtra los productos que no tienen coincidencias en invoiceDetails (los que no se han vendido).
   - Esto ocurre porque, en un LEFT JOIN, cuando no hay coincidencia, las columnas de la tabla derecha (invoiceDetails) serán NULL.
4. **Resultado**: Una tabla que muestra todos los productos que no tienen registros asociados en invoiceDetails, es decir, los que no se han vendido.

4.3.2. Clientes con gastos mayores del promedio

```
-- 2. Encontrar clientes que han gastado más que el p
SELECT p.*
FROM people p
INNER JOIN (
    SELECT idCustomer
    FROM invoices
    GROUP BY idCustomer
    HAVING SUM(totalAmount) > (
        SELECT AVG(totalAmount)
        FROM invoices
    )
) AS avgSpenders ON p.id = avgSpenders.idCustomer;
```

Explicación:

1. **Tablas involucradas**:
   - people (**p**): Contiene información de las personas (clientes).
   - invoices: Contiene información de las facturas, incluyendo el cliente asociado y el monto total.
2. **Objetivo**: Identificar los clientes que han gastado un monto total mayor al promedio de gasto de todos los clientes.
3. **Desglose**:
   - **Subconsulta en HAVING**:
     - Agrupa las facturas por cliente (GROUP BY idCustomer).
     - Calcula la suma total gastada por cada cliente (SUM(totalAmount)).
     - Filtra los clientes cuya suma total sea mayor que el promedio de todos los montos (AVG(totalAmount)), obtenido con una subconsulta.
   - **Consulta principal**:
     - Realiza un INNER JOIN entre people y la subconsulta, relacionando p.id con avgSpenders.idCustomer.
     - Devuelve los datos de los clientes (p.*) que cumplen con la condición.
4. **Resultado**: Una tabla que muestra los datos personales de los clientes que han gastado más que el promedio general.

4.4. Subconsultas IN

Las subconsultas con IN se usan para verificar si un valor en la consulta principal coincide con algún valor devuelto por la subconsulta

4.4.1. Usuarios con rol de Administrador

```sql
-- 1. Encontrar todos los usuarios que ti
SELECT *
FROM users
WHERE id IN (
    SELECT idUser
    FROM usersRoles
    WHERE idRole IN (
        SELECT id
        FROM roles
        WHERE `name` = 'Administrador'
    )
);
```

Explicación:

1. **Tablas involucradas**:
   - users: Contiene información de los usuarios.
   - usersRoles: Relaciona usuarios con roles.
   - roles: Contiene información de los roles.
2. **Objetivo**: Encontrar los usuarios que tienen asignado el rol de "Administrador".
3. **Desglose**:
   - Subconsulta anidada (SELECT id FROM roles):
     - Busca el id del rol cuyo nombre es 'Administrador'.
   - Segunda subconsulta (SELECT idUser FROM usersRoles):
     - Busca los usuarios (idUser) que tienen asignado el idRole obtenido en la subconsulta anterior.
   - Consulta principal:
     - Busca todos los usuarios en users cuyo id esté en los resultados de la segunda subconsulta.
4. **Resultado**: Una lista con todos los usuarios que tienen el rol de administrador.

4.4.2. Productos con stock mayor a 100

```sql
SELECT *
FROM products
WHERE id IN (
    SELECT idProduct
    FROM storeProducts
    WHERE stock > 100
);
```

Explicación:

1. **Tablas involucradas**:
   - products: Contiene información de los productos.
   - storeProducts: Relaciona productos con tiendas e incluye información sobre el stock.
2. **Objetivo**: Encontrar los productos que tienen más de 100 unidades en stock en al menos una tienda.
3. **Desglose**:
   - Subconsulta (SELECT idProduct FROM storeProducts):
     - Busca los productos (idProduct) que tienen un valor de stock mayor a 100 en la tabla storeProducts.
   - Consulta principal:
     - Busca todos los productos en products cuyo id esté en los resultados de la subconsulta.
4. **Resultado**: Una lista de productos que tienen más de 100 unidades en stock en al menos una tienda.

4.5. Subconsultas EXISTS

Las subconsultas con EXISTS se utilizan para verificar si una subconsulta devuelve algún resultado, evaluando la existencia de datos relacionados sin importar los valores específicos.

4.5.1. Al menos 1 producto en stock

```
SELECT *
FROM store s
WHERE EXISTS (
    SELECT 1
    FROM storeProducts sp
    WHERE sp.idStore = s.id
    AND sp.stock > 0
);
```

Explicación:

1. **Tablas involucradas**:
   - store (**s**): Contiene información sobre las tiendas.
   - storeProducts (**sp**): Relaciona las tiendas con los productos, incluyendo el stock disponible.
2. **Objetivo**: Identificar las tiendas que tienen al menos un producto con stock mayor a 0.
3. **Desglose**:
   - Subconsulta en EXISTS:
     - Busca en storeProducts registros donde idStore coincida con el id de la tienda actual (s.id) y el stock sea mayor a 0.
     - Si encuentra al menos un registro que cumpla estas condiciones, devuelve TRUE para esa tienda.
   - Consulta principal:
     - Devuelve las tiendas para las cuales la subconsulta devuelve resultados (EXISTS).
4. **Resultado**: Una tabla que lista las tiendas que tienen al menos un producto con stock disponible.

4.5.2. Facturas con al menos una relación a IncoiceDetails

```
SELECT *
FROM invoices i
WHERE EXISTS (
    SELECT 1
    FROM invoiceDetails id
    WHERE id.idInvoice = i.id
);
```

Explicación:

1. **Tablas involucradas**:
   - invoices (**i**): Contiene información sobre las facturas.
   - invoiceDetails (**id**): Contiene los detalles de las facturas, como los productos vendidos en cada una.
2. **Objetivo**: Identificar las facturas que tienen al menos un registro asociado en invoiceDetails.
3. **Desglose**:
   - Subconsulta en EXISTS:
     - Busca en invoiceDetails registros donde idInvoice coincida con el id de la factura actual (i.id).
     - Si encuentra al menos un detalle asociado, devuelve TRUE para esa factura.
   - Consulta principal:
     - Devuelve las facturas para las cuales la subconsulta devuelve resultados (EXISTS).
4. **Resultado**: Una tabla que lista las facturas que tienen al menos un detalle registrado en la tabla invoiceDetails.

4.6. Subconsultas ALL - ANY

Las subconsultas con ALL y ANY se utilizan para comparar un valor con todos o con al menos uno de los valores devueltos por una subconsulta, respectivamente.

4.6.1. Productos más caros en una categoría (ALL)

```sql
SELECT *
FROM products
WHERE price > ALL (
    SELECT price
    FROM products
    WHERE idCategory = 1
);
```

Explicación:

1. **Tablas involucradas**:
   - products: Contiene información sobre los productos, incluyendo su categoría (idCategory) y su precio (price).

2. **Objetivo**: Encontrar productos cuyo precio sea mayor que el de todos los productos de una categoría específica (en este caso, categoría con idCategory = 1).
3. **Desglose**:
    - Subconsulta:
        - Busca los precios (price) de todos los productos que pertenecen a la categoría 1.
    - Consulta principal:
        - Compara el precio de cada producto (price) en la tabla principal con **todos** los precios devueltos por la subconsulta.
        - Solo se incluyen los productos cuyo precio sea mayor que el precio más alto de la categoría seleccionada.
4. **Cláusula ALL**:
    - La condición price > ALL significa que el precio del producto debe ser mayor que **cada uno** de los precios devueltos por la subconsulta.
    - Si un producto tiene un precio menor o igual a algún precio de la subconsulta, no se incluirá.
5. **Resultado**: Una lista de productos cuyo precio es mayor que el de cualquier producto en la categoría 1.

4.6.2. Facturas con monto de pago mayor individual (ANY)

```
SELECT *
FROM invoices
WHERE totalAmount > ANY (
    SELECT paymentAmount
    FROM payment
);
```

Explicación:

1. **Tablas involucradas**:
    - invoices: Contiene información de las facturas, incluyendo el monto total (totalAmount).
    - payment: Contiene información de los pagos realizados, incluyendo el monto del pago (paymentAmount).
2. **Objetivo**: Encontrar las facturas cuyo monto total sea mayor que al menos un pago individual realizado.
3. **Desglose**:
    - Subconsulta:

        - Obtiene todos los montos de pagos (paymentAmount) registrados en la tabla payment.
    - Consulta principal:
        - Compara el monto total de cada factura (totalAmount) con **cualquier** monto devuelto por la subconsulta.
        - Incluye las facturas cuyo monto sea mayor que al menos un pago individual.
4. **Cláusula ANY**:
    - La condición totalAmount > ANY significa que el monto de la factura debe ser mayor que **al menos uno** de los montos devueltos por la subconsulta.
    - Si hay al menos un pago menor que el monto de una factura, esa factura se incluirá.
5. **Resultado**: Una lista de facturas cuyo monto total es mayor que al menos un pago registrado en la tabla payment.

5. Vistas

Las **vistas** son tablas virtuales que representan los resultados de una consulta SQL. Se utilizan para simplificar el acceso a datos complejos y pueden ser creadas, modificadas o eliminadas con las sentencias SQL adecuadas

- Video: https://drive.google.com/file/d/1pxTRK7uNxr_u3RwydDvQmmFSDjtYeh_x/view?usp=sharing

5.1. Pagos realizados por clients

```sql
CREATE OR REPLACE VIEW viewCustomerPayments AS
SELECT
    u.id AS customerId,
    u.`name` AS customerName,
    pd.paymentDate,
    pd.paymentAmount,
    pm.`name` AS paymentMethod
FROM users u
JOIN paymentDetail pd ON u.id = pd.idCustomer
JOIN paymentMethod pm ON pd.idPaymentMethod = pm.id;

SELECT * FROM viewCustomerPayments;
```

5.2. Resumen de facturas emitidas

```sql
-- 3.Resumen de las facturas emitidas
CREATE OR REPLACE VIEW viewSalesInvoice AS
SELECT
    i.id AS invoiceId,
    i.invoiceDate,
    u.`name` AS customerName,
    i.totalAmount
FROM invoices i
JOIN users u ON i.idCustomer = u.id;

SELECT * FROM viewSalesInvoice;
SELECT * FROM viewSalesInvoice WHERE totalAmount < 1000;
```

6. Procedimientos Almacenados

Los **procedimientos almacenados** son conjuntos de instrucciones SQL que se almacenan en el servidor y pueden ser ejecutados repetidamente. Se crean utilizando la sentencia CREATE PROCEDURE y se invocan con CALL.

6.1. Tienda por ID

```
-- 1. Procedimiento para Consultar una Tienda por su ID
DELIMITER //

CREATE PROCEDURE getStoreById (IN storeId INT)
BEGIN
    SELECT * FROM store WHERE id = storeId;
END //

DELIMITER ;

CALL getStoreById(2);
```

6.2. Productos por categoria.

```
-- 2. Procedimiento para Consultar Productos por Categoría
DELIMITER //

CREATE PROCEDURE getProductsCategory (IN categoryId INT)
BEGIN
    SELECT * FROM products WHERE idCategory = categoryId;
END //

DELIMITER ;

CALL getProductsCategory(2)
```

7. Triggers

Los **triggers** son procedimientos que se ejecutan automáticamente en respuesta a eventos específicos en la base de datos, como inserciones, actualizaciones o eliminaciones. Se definen con la sentencia CREATE TRIGGER y pueden ser configurados para activarse antes o después del evento.

7.1. Before Insert payment

```
-- 1. beforeInsert en la tabla payment
DELIMITER //

CREATE TRIGGER beforeInsertPayment
BEFORE INSERT
ON payment
FOR EACH ROW
BEGIN
    IF NEW.paymentAmount IS NULL THEN
        SET NEW.paymentAmount = 0.00;
    END IF;
END //

DELIMITER ;
```

7.2. BeforeUpdate product

```
-- 3. beforeUpdate en la tabla products
DELIMITER //

CREATE TRIGGER beforeUpdateProducts
BEFORE UPDATE
ON products
FOR EACH ROW
BEGIN
    IF OLD.price != NEW.price OR OLD.stock != NEW.stock THEN
        INSERT INTO auditLogs (`user`, `action`, tableName, recordId, oldValue, newValue, actionTime)
        VALUES (USER(), 'BEFORE UPDATE', 'products', OLD.id,
                CONCAT('price: ', OLD.price, ', stock: ', OLD.stock),
                CONCAT('price: ', NEW.price, ', stock: ', NEW.stock), NOW());
    END IF;
END //

DELIMITER ;
```

8. Indices

Los **índices en bases de datos SQL** son estructuras de datos que mejoran la velocidad de las operaciones de búsqueda y recuperación de datos. Funcionan de manera similar a un índice en un libro, permitiendo localizar rápidamente registros específicos sin necesidad de escanear toda la tabla.

8.1. Indices Agrupados

Un **índice agrupado** (clustered index) es una estructura de datos en bases de datos SQL que determina el orden físico en que se almacenan las filas de una tabla.

8.1.1. Users

```
CREATE  INDEX idxUsersId ON users(id);
SELECT * FROM users WHERE id=5;
```

8.1.2. Invoices

```
CREATE  INDEX idxInvoicesId ON invoices(id);
SELECT * FROM invoices WHERE id=1;
```

8.2. Indices No Agrupados

Un **índice no agrupado** (non-clustered index) es una estructura de datos en bases de datos SQL que permite mejorar la velocidad de búsqueda y recuperación de datos sin modificar el orden físico en que se almacenan los registros en la tabla.

8.2.1. Roles por nombre

```
CREATE INDEX idxRolesName ON roles(`name`);
SELECT * FROM roles WHERE `name` = 'Proveedor';
```

8.2.2. Producto por codigo

```
CREATE INDEX idxProductsName ON products(`code`);
SELECT * FROM products WHERE `code` LIKE'A%';
```

8.3. Indices Unicos

Asegura que todos los valores en la columna del índice sean únicos, evitando duplicados y mejorando la integridad de los datos

8.3.1. Email de ususario

```sql
CREATE UNIQUE INDEX idxUsersEmailUnique ON users(email);
SELECT *
FROM users
WHERE email LIKE '%@example.com';
```

8.3.2. Codigo de product

```sql
CREATE UNIQUE INDEX idxProductsCodeUnique ON products(`code`);
SELECT * FROM products WHERE `code` = 'JKL012';
```

8.4. Indices Compuestos

Los **índices compuestos** son estructuras de datos en bases de datos SQL que permiten indexar múltiples columnas de una tabla al mismo tiempo. Esto mejora la eficiencia de las consultas que filtran o clasifican datos basándose en varias columnas.

8.4.1. Facturas por cliente

```sql
CREATE INDEX idx_invoices_date_customer ON invoices(invoiceDate, idCustomer);
SELECT idCustomer, COUNT(*) AS totalInvoices FROM invoices
WHERE invoiceDate >= '2024-10-01' AND invoiceDate <= '2024-10-30'
GROUP BY idCustomer;
```