



Universidad Peruana Los Andes
Facultad de Ingeniería
Escuela Profesional de Ingeniería de Sistemas y Computación

Administración de Base de Datos

Enunciado 01:

De acuerdo con la **base de datos** implementada (mínimo 100 registros), utilice los DBMS **Microsoft SQL Server/MySQL**, o un servidor de la nube como Microsoft Azure o Google FireBase. Explique qué problema soluciona su base de datos y responda las siguientes preguntas:

Creamos nuestras tablas en PostgreSQL:

```
CREATE TABLE IF NOT EXISTS Usuarios (  
    Region TEXT PRIMARY KEY,  
    Gerente TEXT NOT NULL  
);  
CREATE TABLE Ordenes (  
    ID_de_fila SERIAL PRIMARY KEY,  
    Prioridad_de_pedido TEXT NOT NULL,  
    Descuento DECIMAL(5, 2) NOT NULL,  
    Precio_unitario DECIMAL(10, 2) NOT NULL,  
    Costo_de_envio DECIMAL(10, 2) NOT NULL,  
    ID_de_cliente INT NOT NULL,  
    Nombre_del_cliente TEXT NOT NULL,  
    Modo_de_envio TEXT NOT NULL,  
    Segmento_de_clientes TEXT NOT NULL,  
    Categoria_de_producto TEXT NOT NULL,  
    Subcategoria_de_producto TEXT NOT NULL,  
    Contenedor_de_producto TEXT,  
    Nombre_del_producto TEXT NOT NULL,  
    Margen_base_del_producto DECIMAL(10, 2),  
    Pais TEXT,  
    Region TEXT REFERENCES Usuarios(Region),  
    Estado_o_Provincia TEXT NOT NULL,  
    Ciudad TEXT NOT NULL,  
    Código_Postal INT NOT NULL,  
    Fecha_del_pedido DATE NOT NULL,  
    Fecha_de_envio DATE NOT NULL,  
    Ganancia DECIMAL(10, 4) NOT NULL,  
    Cantidad_pedida_nueva INT NOT NULL,  
    Ventas DECIMAL(10, 2) NOT NULL,  
    ID_de_pedido INT NOT NULL,  
    CONSTRAINT pedido_fila_unica UNIQUE (ID_de_pedido, ID_de_fila)  
);  
CREATE TABLE Dev (  
    ID_de_pedido INT NOT NULL,  
    ID_de_fila INT NOT NULL,  
    Estado TEXT NOT NULL,  
    CONSTRAINT fk_pedido_fila FOREIGN KEY (ID_de_pedido, ID_de_fila)  
        REFERENCES Ordenes(ID_de_pedido, ID_de_fila)  
);
```

Importamos los datos en formato CSV:

```
SELECT * FROM public.pedidos
ORDER BY id_fila ASC
```

Output Messages Notifications

id_fila [PK] integer	prioridad_pedido character varying (50)	descuento numeric (5,2)	precio_unitario numeric (10,2)	costo_envio numeric (10,2)	id_cliente integer	nombre_cliente character varying (100)	modo_envio character varying (50)	segmento_cliente character varying (50)
64	Medium	0.08	124.49	51.94	553	Kristine Connolly	Delivery Truck	Corporate
87	Critical	0.04	3.08	0.99	3106	Alexander O'Brien	Regular Air	Home Office
88	Critical	0.02	6.48	5.90	3106	Alexander O'Brien	Regular Air	Home Office
89	Critical	0.04	125.99	4.20	3106	Alexander O'Brien	Regular Air	Home Office
106	High	0.01	9.31	3.98	1106	Maxine Collier Grady	Regular Air	Small Business
151	Low	0.06	122.99	19.99	2382	Geoffrey Saunders	Regular Air	Small Business
152	Low	0.08	68.81	60.00	2382	Geoffrey Saunders	Delivery Truck	Small Business
349	Not Specified	0.07	2036.48	14.70	553	Kristine Connolly	Delivery Truck	Corporate
448	Medium	0.10	4.26	1.20	699	Jenny Gold	Regular Air	Consumer
462	Not Specified	0.07	179.99	19.99	471	Ross Simpson	Express Air	Consumer
480	Critical	0.01	3.26	1.86	342	Jacqueline Noble	Regular Air	Corporate
494	Medium	0.10	19.98	4.00	102	Caroline Johnston	Regular Air	Consumer
495	Medium	0.09	2.88	1.49	102	Caroline Johnston	Regular Air	Consumer
522	High	0.07	1.68	1.57	181	Wesley Waller	Regular Air	Corporate

1) Implemente y explique un Script para crear una **vista** para crear utilizando tres tablas

```
CREATE VIEW Vista_Ordenes_Dev_Usuarios AS
```

```
SELECT
```

```
o.ID_de_fila,
o.Prioridad_de_pedido,
o.Nombre_del_cliente,
o.Region,
o.Estado_o_Provincia,
o.Ciudad,
d.Estado AS Estado_Dev,
u.Gerente
```

```
FROM
```

```
Ordenes o
```

```
JOIN
```

```
Dev d ON o.ID_de_pedido = d.ID_de_pedido AND o.ID_de_fila = d.ID_de_fila
```

```
JOIN
```

```
Usuarios u ON o.Region = u.Region;
```

Output Messages Notifications

id_de_fila integer	prioridad_de_pedido text	nombre_del_cliente text	region text	estado_o_provincia text	ciudad text	estado_dev text	gerente text
20847	High	Bonnie Potter	West	Washington	Anacortes	Returned	William
20228	Not Specified	Ronnie Proctor	West	California	San Gabriel	Returned	William
21776	Critical	Marcus Dunlap	East	New Jersey	Roselle	Returned	Erin

La vista combina los datos de las tres tablas. Se usa JOIN para combinar las tablas Ordenes y Dev, mediante las columnas ID_de_pedido y ID_de_fila, y luego se hace un JOIN con Usuarios mediante la columna Region. Esta vista puede ser utilizada para obtener información combinada sobre las órdenes, el estado de las órdenes en la tabla Dev, y los gerentes de las regiones.

2) Implemente y explique un Script para crear un **procedimiento almacenado** para insertar datos a su base de datos.

Mg. Ing. Raúl Fernández Bejarano

```
CREATE OR REPLACE PROCEDURE insertar_orden(  
    p_prioridad_de_pedido TEXT,  
    p_descuento DECIMAL(5,2),  
    pPrecioUnitario DECIMAL(10,2),  
    p_costo_de_envio DECIMAL(10,2),  
    p_id_de_cliente INT,  
    p_nombre_del_cliente TEXT,  
    p_modo_de_envio TEXT,  
    p_segmento_de_clientes TEXT,  
    p_categoria_de_producto TEXT,  
    p_subcategoria_de_producto TEXT,  
    p_contenedor_de_producto TEXT,  
    p_nombre_del_producto TEXT,  
    p_margen_base_del_producto DECIMAL(10,2),  
    p_pais TEXT,  
    p_region TEXT,  
    p_estado_o_provincia TEXT,  
    p_ciudad TEXT,  
    p_codigo_postal INT,  
    p_fecha_del_pedido DATE,  
    p_fecha_de_envio DATE,  
    p_ganancia DECIMAL(10,4),  
    p_cantidad_pedida_nueva INT,  
    p_ventas DECIMAL(10,2),  
    p_id_de_pedido INT  
)  
LANGUAGE plpgsql  
AS $$
```

```

BEGIN
  INSERT INTO Ordenes (
    Prioridad_de_pedido,
    Descuento,
    Precio_unitario,
    Costo_de_envio,
    ID_de_cliente,
    Nombre_del_cliente,
    Modo_de_envio,
    Segmento_de_clientes,
    Categoria_de_producto,
    Subcategoria_de_producto,
    Contenedor_de_producto,
    Nombre_del_producto,
    Margen_base_del_producto,
    Pais,
    Region,
    Estado_o_Provincia,
    Ciudad,
   Codigo_Postal,
    Fecha_del_pedido,
    Fecha_de_envio,
    Ganancia,
    Cantidad_pedida_nueva,
    Ventas,
    ID_de_pedido
  )
  VALUES (
    p_prioridad_de_pedido,
    p_descuento,
    p_precio_unitario,
    p_costo_de_envio,
    p_id_de_cliente,
    p_nombre_del_cliente,
    p_modo_de_envio,
    p_segmento_de_clientes,
    p_categoria_de_producto,
    p_subcategoria_de_producto,
    p_contenedor_de_producto,
    p_nombre_del_producto,
    p_margen_base_del_producto,
    p_pais,
    p_region,
    p_estado_o_provincia,
    p_ciudad,
    p_codigo_postal,
    p_fecha_del_pedido,
    p_fecha_de_envio,
    p_ganancia,
    p_cantidad_pedida_nueva,
    p_ventas,
    p_id_de_pedido
  )

```

```
CALL insertar_orden(  
    'Alta',  
    15.50,  
    100.00,  
    10.00,  
    123,  
    'Juan Pérez',  
    'Terrestre',  
    'Retail',  
    'Electrónica',  
    'Celulares',  
    'Caja',  
    'iPhone 14',  
    50.00,  
    'México',  
    'West',  
    'CDMX',  
    'Ciudad de México',  
    10000,  
    '2024-11-01',  
    '2024-11-03',  
    200.00,  
    2,  
    300.00,  
    101  
);
```

```
CREATE OR REPLACE PROCEDURE insertar_dev(  
    p_id_de_pedido INT,  
    p_id_de_fila INT,  
    p_estado TEXT  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    INSERT INTO Dev (ID_de_pedido, ID_de_fila, Estado)  
    VALUES (p_id_de_pedido, p_id_de_fila, p_estado);  
END;  
$$;  
CALL insertar_dev(101, 1, 'Pendiente');
```

El procedimiento insertar; orden acepta todos los campos necesarios para insertar una fila en la tabla Ordenes.

Usamos el comando INSERT INTO para agregar un nuevo registro con los valores proporcionados a través de los parámetros.

El procedimiento insertar_dev acepta tres parámetros: p_id_de_pedido, p_id_de_fila y p_estado.

Se utiliza el comando INSERT INTO para agregar una nueva fila a la tabla Dev.

461	1	Alta	15.50	100.00	10.00	123	Juan Pérez	Terrestre	Retail	Electrónica
1953	101	1	Pendiente							

3) Implemente y explique un Script para crear un **procedimiento almacenado** para eliminar datos a su base de datos

```

CREATE OR REPLACE FUNCTION eliminar_orden(p_id_de_pedido INT, p_id_de_fila INT)
RETURNS VOID AS $$
BEGIN
    DELETE FROM Ordenes
    WHERE ID_de_pedido = p_id_de_pedido AND ID_de_fila = p_id_de_fila;
    IF NOT FOUND THEN
        RAISE NOTICE 'No se encontró la orden con ID_de_pedido = % y ID_de_fila = %.', p_id_de_pedido, p_id_de_fila;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Error al eliminar la orden: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
ALTER TABLE Dev
DROP CONSTRAINT fk_pedido_fila,
ADD CONSTRAINT fk_pedido_fila FOREIGN KEY (ID_de_pedido, ID_de_fila)
REFERENCES Ordenes(ID_de_pedido, ID_de_fila)
ON DELETE CASCADE;
SELECT eliminar_orden(101, 1);

```

- **Llamar a la función:** Usa SELECT eliminar_orden (p_id_de_pedido, p_id_de_fila); para eliminar la orden.
- **Verificar dependencias:** Si hay registros relacionados en otras tablas (como Dev), debes eliminarlos primero o configurar eliminaciones en cascada.
- **Manejo de excepciones:** La función te dará detalles sobre si la eliminación fue exitosa o si hubo problemas debido a dependencias u otros errores.

```
SELECT * FROM Ordenes WHERE ID_de_pedido = 101 AND ID_de_fila = 1;
```

Output	Messages	Notifications
SQL		
id_de_fila [PK] integer	prioridad_de_pedido text	descuento numeric (5,2)
precio_unitario numeric (10,2)	costo_de_envio numeric (10,2)	id_de_cliente integer
nombre_del_cliente text	modo_de_envio text	seg text

4) Implemente y explique un Script para crear un **procedimiento almacenado** para actualizar datos a su base de datos

Mg. Ing. Raúl Fernández Bejarano

```

CREATE OR REPLACE PROCEDURE actualizar_orden(
    p_id_de_pedido INT,
    p_id_de_fila INT,
    p_prioridad_de_pedido TEXT,
    p_descuento DECIMAL(5,2),
    p_precio_unitario DECIMAL(10,2),
    p_costo_de_envio DECIMAL(10,2),
    p_id_de_cliente INT,
    p_nombre_del_cliente TEXT,
    p_modos_de_envio TEXT,
    p_segmento_de_clientes TEXT,
    p_categoria_de_producto TEXT,
    p_subcategoria_de_producto TEXT,
    p_contenedor_de_producto TEXT,
    p_nombre_del_producto TEXT,
    p_margen_base_del_producto DECIMAL(10,2),
    p_pais TEXT,
    p_region TEXT,
    p_estado_o_provincia TEXT,
    p_ciudad TEXT,
    p_codigo_postal INT,
    p_fecha_del_pedido DATE,
    p_fecha_de_envio DATE,
    p_ganancia DECIMAL(10,4),
    p_cantidad_pedida_nueva INT,
    p_ventas DECIMAL(10,2)
)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Ordenes
    SET
        Prioridad_de_pedido = p_prioridad_de_pedido,
        Descuento = p_descuento,
        Precio_unitario = p_precio_unitario,
        Costo_de_envio = p_costo_de_envio,
        ID_de_cliente = p_id_de_cliente,
        Nombre_del_cliente = p_nombre_del_cliente,
        Modos_de_envio = p_modos_de_envio,
        Segmento_de_clientes = p_segmento_de_clientes,
        Categoria_de_producto = p_categoria_de_producto,
        Subcategoria_de_producto = p_subcategoria_de_producto,
        Contenedor_de_producto = p_contenedor_de_producto,
        Nombre_del_producto = p_nombre_del_producto,
        Margen_base_del_producto = p_margen_base_del_producto,
        Pais = p_pais,
        Region = p_region,
        Estado_o_Provincia = p_estado_o_provincia,
        Ciudad = p_ciudad,
        Codigo_Postal = p_codigo_postal,
        Fecha_del_pedido = p_fecha_del_pedido,
        Fecha_de_envio = p_fecha_de_envio,
        Ganancia = p_ganancia,
        Cantidad_pedida_nueva = p_cantidad_pedida_nueva,
        Ventas = p_ventas
    WHERE ID_de_pedido = p_id_de_pedido AND ID_de_fila = p_id_de_fila;
    IF NOT FOUND THEN
        RAISE NOTICE 'No se encontró la orden con ID_de_pedido = % y ID_de_fila = %.', p_id_de_pedido, p_id_de_fila;
    END IF;
END;
$$:

```

```
CALL actualizar_orden(
    101,
    2,
    'Alta',
    20.00,
    120.00,
    15.00,
    123,
    'Juan Pérez',
    'Terrestre',
    'Retail',
    'Electrónica',
    'Smartphones',
    'Caja',
    'iPhone 12',
    55.00,
    'Peru',
    'West',
    'CDMX',
    'Lima',
    10000,
    '2024-11-01',
    '2024-11-03',
    210.00,
    3,
    320.00
);
```

- Esta llamada al procedimiento actualizará la orden con ID_de_pedido = 101 y ID_de_fila = 2, cambiando todos los campos de la orden a los valores proporcionados.
- Puedes modificar los valores en la llamada para adaptarlos a lo que necesitas actualizar.

	id_de_fila [PK] integer	prioridad_de_pedido text	descuento numeric (5,2)	precio_unitario numeric (10,2)	costo_de_envio numeric (10,2)	id_de_cliente integer	nombre_del_cliente text	modo_de_envio text	segmento_de_clientes text	categoria_de_producto text	subcategoria text
1	2	Alta	20.00	120.00	15.00	123	Juan Pérez	Terrestre	Retail	Electrónica	Smartphones

- 5) Implemente y explique un Script para crear un **procedimiento almacenado** para realizar cálculos matemáticos de una columna de su base de datos.

```
CREATE OR REPLACE FUNCTION calcular_ganancia()
RETURNS TRIGGER AS $$
BEGIN
    NEW.Ganancia := (NEW.Precio_unitario * NEW.Cantidad_pedida_nueva) - NEW.Costo_de_envio;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_calcular_ganancia
BEFORE INSERT OR UPDATE ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION calcular_ganancia();
```

El **trigger** y la **función** proporcionados realizan el cálculo de la ganancia antes de insertar o actualizar una orden, garantizando que siempre se mantenga actualizado el valor de la columna Ganancia.

Mg. Ing. Raúl Fernández Bejarano


```
SELECT * FROM Ordenes WHERE ID_de_pedido = 101 AND ID_de_fila = 6;
```

Output Messages Notifications

del_producto	pais	region	estado_o_provincia	ciudad	codigo_postal	fecha_del_pedido	fecha_de_envio	ganancia	cantidad_pedido_nueva	ventas	id_de_pedido
50.00	México	West	CDMX	Ciudad de México	10000	2024-11-01	2024-11-03	190.0000	2	300.00	101

- 6) Implemente y explique un Script para crear un **disparador** para ingresar un registro automáticamente en una tabla de su base de datos.

```
CREATE OR REPLACE FUNCTION log_nueva_orden()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Log_Ordenes (ID_de_pedido, Detalle)
    VALUES (NEW.ID_de_pedido, 'Nueva orden insertada: ' || NEW.Nombre_del_cliente);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_log_nueva_orden
AFTER INSERT ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION log_nueva_orden();
```

```
SELECT * FROM Log_Ordenes;
```

Output Messages Notifications

id_de_log	id_de_pedido	fecha_de_log	detalle
[PK] integer	integer	timestamp without time zone	text
1	111	2024-11-24 13:47:11.511406	Nueva orden insertada: Juan Pérez

Al ejecutar el INSERT, el disparador trigger_log_nueva_orden se activará y ejecutará la función log_nueva_orden (), lo que insertará automáticamente un registro en la tabla Log_Ordenes.

- 7) Implemente y explique un Script para crear un **disparador** para elimine un registro automáticamente en una tabla de su base de datos.

```
CREATE OR REPLACE FUNCTION log_eliminar_orden()
RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM Log_Ordenes WHERE ID_de_pedido = OLD.ID_de_pedido;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_log_eliminar_orden
AFTER DELETE ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION log_eliminar_orden();
```

Este script asegura que cada vez que se elimina una orden de Ordenes, el registro correspondiente se elimina automáticamente de Log_Ordenes, manteniendo así la integridad referencial.

- 8) Implemente y explique un Script para crear un **disparador** para actualice un registro automáticamente en una tabla de su base de datos.

```
CREATE OR REPLACE FUNCTION log_actualizar_orden()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Log_Ordenes
    SET Detalle = 'Orden actualizada: ' || NEW.Nombre_del_cliente || ' (' || NEW.Prioridad_de_pedido || ')',
        Fecha_de_log = CURRENT_TIMESTAMP
    WHERE ID_de_pedido = NEW.ID_de_pedido;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_log_actualizar_orden
AFTER UPDATE ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION log_actualizar_orden();
```

Este script asegura que cada vez que se actualiza una orden en Ordenes, el registro correspondiente se actualiza automáticamente en Log_Ordenes, manteniendo así un historial actualizado de las operaciones.

- 9) Implemente y explique un Script para crear un **disparador** para verificar el control de datos (Ejemplo: que la nota ingresada este entre 0 y 20)

```
CREATE OR REPLACE FUNCTION verificar_datos_orden()
RETURNS TRIGGER AS $$
BEGIN
    -- Verificar que el descuento no supere el 50%
    IF NEW.Descuento > 50 THEN
        RAISE EXCEPTION 'El descuento no puede ser mayor al 50%. Descuento actual: %', NEW.Descuento;
    END IF;

    -- Verificar que el precio unitario no sea negativo
    IF NEW.Precio_unitario < 0 THEN
        RAISE EXCEPTION 'El precio unitario no puede ser negativo. Precio actual: %', NEW.Precio_unitario;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_verificar_datos_orden
BEFORE INSERT OR UPDATE ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION verificar_datos_orden();
```

- **Función del Disparador:** La función `verificar_datos_orden ()` se ejecuta cada vez que se intenta insertar o actualizar una orden en Ordenes. Esta función verifica que el descuento no supere el 50% y que el precio unitario no sea negativo. Si alguna de estas condiciones no se cumple, lanza un error y evita la operación.
- **Disparador:** El disparador `trigger_verificar_datos_orden` se ejecuta automáticamente antes de cada inserción o actualización en Ordenes, llamando a la función `verificar_datos_orden ()` para validar los datos.

- 10) Utilizando Script Crear 03 usuarios con nombres de sus compañeros y uno suyo

Mg. Ing. Raúl Fernández Bejarano

```
-- Crear usuarios de PostgreSQL
CREATE USER Christhian WITH PASSWORD 'password1';
CREATE USER Jeison WITH PASSWORD 'password2';
CREATE USER Pedro WITH PASSWORD 'password3';
CREATE USER Jazztin WITH PASSWORD 'password4';
-- Asignar privilegios a los usuarios
GRANT ALL PRIVILEGES ON DATABASE BDifd TO Christhian;
GRANT ALL PRIVILEGES ON DATABASE BDifd TO Jeison;
GRANT ALL PRIVILEGES ON DATABASE BDifd TO Pedro;
```

- 11) Utilizando un script, copiar la base de datos (creada anteriormente) y compartir en cada uno de los usuarios

```
-- Crear la nueva base de datos a partir de la existente
CREATE DATABASE BDifd_copy WITH TEMPLATE BDifd;
```

```
-- Asignar privilegios a los usuarios
GRANT ALL PRIVILEGES ON DATABASE BDifd_copy TO Christhian;
GRANT ALL PRIVILEGES ON DATABASE BDifd_copy TO Jeison;
GRANT ALL PRIVILEGES ON DATABASE BDifd_copy TO Pedro;
```

- 12) Utilizando un script, generar una copia de seguridad de la base de datos y compartir a cada uno de los usuarios

```
CREATE DATABASE copia_de_seguridad WITH TEMPLATE BDifd;
```

```
-- Asignar privilegios a los usuarios
GRANT ALL PRIVILEGES ON DATABASE copia_de_seguridad TO Christhian;
GRANT ALL PRIVILEGES ON DATABASE copia_de_seguridad TO Jeison;
GRANT ALL PRIVILEGES ON DATABASE copia_de_seguridad TO Pedro;
```

- 13) Utilizando un script, encriptar una de las tablas para que no se puedan ver los datos

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

```
CREATE TABLE IF NOT EXISTS Usuarios (
    Region TEXT PRIMARY KEY,
    Gerente BYTEA NOT NULL
);
```

gerente_encriptado	bytea
[null]	
[null]	
[null]	

- 14) Utilizando un script, aplique la seguridad a nivel de columna, restringiendo el acceso a la columna que contiene la clave primaria de una de las tablas de su base de datos

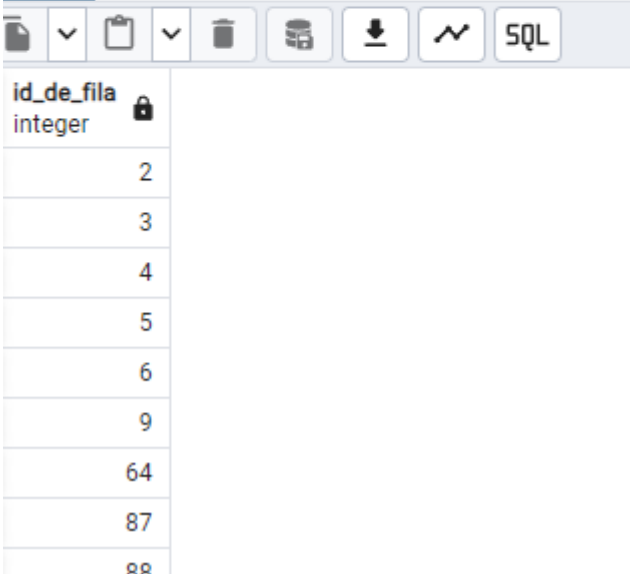
```
CREATE VIEW vista_ID AS
SELECT ID_de_fila FROM Ordenes;

GRANT SELECT ON vista_ID TO Jeison;
REVOKE ALL ON Ordenes FROM Jeison;

SET ROLE Jeison;

SELECT * FROM vista_ID;
```

Output Messages Notifications



id_de_fila integer
2
3
4
5
6
9
64
87

- 15) Utilizando un script, implementé seguridad a nivel de columna restringiendo el acceso a una de las columnas de una tabla.
- 16) Utilizando un script, realice el cifrado transparente de datos (TDE) para una las tablas.
Lo mismo que el numero 15
- 17) Utilizando un script, configure el usuario con el nombre de su compañero para otorgar permisos de SELECT, INSERT, UPDATE y DELETE en la base de datos.

```
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO Christhian;
```

- 18) Utilizando un Scripts realice la validación y filtración de entradas del usuario para evitar caracteres maliciosos (Ejemplo: ', --, ☺)

Mg. Ing. Raúl Fernández Bejarano

```

CREATE OR REPLACE FUNCTION aplicar_validacion_filtrado_ordenes() RETURNS TRIGGER AS $$
BEGIN
    -- Aplicar validación y filtrado al texto
    NEW.Nombre_del_cliente := validar_filtrar_texto(NEW.Nombre_del_cliente);
    NEW.Modos_de_envio := validar_filtrar_texto(NEW.Modos_de_envio);
    NEW.Segmento_de_clientes := validar_filtrar_texto(NEW.Segmento_de_clientes);
    NEW.Categoria_de_producto := validar_filtrar_texto(NEW.Categoria_de_producto);
    NEW.Subcategoria_de_producto := validar_filtrar_texto(NEW.Subcategoria_de_producto);
    NEW.Contenedor_de_producto := validar_filtrar_texto(NEW.Contenedor_de_producto);
    NEW.Nombre_del_producto := validar_filtrar_texto(NEW.Nombre_del_producto);
    NEW.Pais := validar_filtrar_texto(NEW.Pais);
    NEW.Region := validar_filtrar_texto(NEW.Region);
    NEW.Estado_o_Provincia := validar_filtrar_texto(NEW.Estado_o_Provincia);
    NEW.Ciudad := validar_filtrar_texto(NEW.Ciudad);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger para la tabla Ordenes
CREATE TRIGGER trigger_validacion_filtrado_ordenes
BEFORE INSERT OR UPDATE ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION aplicar_validacion_filtrado_ordenes();

-- Función y Trigger para la tabla Usuarios
CREATE OR REPLACE FUNCTION aplicar_validacion_filtrado_usuarios() RETURNS TRIGGER AS $$
BEGIN
    NEW.Gerente := validar_filtrar_texto(NEW.Gerente);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_validacion_filtrado_usuarios
BEFORE INSERT OR UPDATE ON Usuarios
FOR EACH ROW
EXECUTE FUNCTION aplicar_validacion_filtrado_usuarios();

```

- 19) Realice un script que verifiquen que los datos ingresados cumplan con formatos esperados (ej.: números en lugar de texto, longitud máxima).

```

CREATE OR REPLACE FUNCTION validar_datos_ordenes() RETURNS TRIGGER AS $$
BEGIN
    -- Validar números positivos
    IF NEW.Descuento < 0 OR NEW.Precio_unitario < 0 OR NEW.Costo_de_envio < 0 OR NEW.Ganancia < 0 OR NEW.Ventas < 0 THEN
        RAISE EXCEPTION 'Los valores numéricos no pueden ser negativos';
    END IF;

    -- Validar longitud máxima de texto
    IF LENGTH(NEW.Nombre_del_cliente) > 50 THEN
        RAISE EXCEPTION 'El nombre del cliente no puede tener más de 50 caracteres';
    END IF;

    -- Filtrar texto para caracteres maliciosos
    NEW.Prioridad_de_pedido := validar_filtrar_texto(NEW.Prioridad_de_pedido);
    NEW.Nombre_del_cliente := validar_filtrar_texto(NEW.Nombre_del_cliente);
    NEW.Modos_de_envio := validar_filtrar_texto(NEW.Modos_de_envio);
    NEW.Segmento_de_clientes := validar_filtrar_texto(NEW.Segmento_de_clientes);
    NEW.Categoria_de_producto := validar_filtrar_texto(NEW.Categoria_de_producto);
    NEW.Subcategoria_de_producto := validar_filtrar_texto(NEW.Subcategoria_de_producto);
    NEW.Contenedor_de_producto := validar_filtrar_texto(NEW.Contenedor_de_producto);
    NEW.Nombre_del_producto := validar_filtrar_texto(NEW.Nombre_del_producto);
    NEW.Pais := validar_filtrar_texto(NEW.Pais);
    NEW.Estado_o_Provincia := validar_filtrar_texto(NEW.Estado_o_Provincia);
    NEW.Ciudad := validar_filtrar_texto(NEW.Ciudad);
    NEW.Region := validar_filtrar_texto(NEW.Region);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger para la tabla Ordenes
CREATE TRIGGER trigger_validacion_datos_ordenes
BEFORE INSERT OR UPDATE ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION validar_datos_ordenes();

```

- 20) Utilizando un script, configure la auditoría para el seguimiento y registro de acciones en la base de datos

```

CREATE TABLE auditoria (
    id SERIAL PRIMARY KEY,
    tabla TEXT,
    operacion TEXT,
    fecha TIMESTAMPTZ DEFAULT current_timestamp,
    usuario TEXT,
    datos_anteriores JSONB,
    datos_nuevos JSONB
);

-- Paso 2: Crear la Función de Auditoría
CREATE OR REPLACE FUNCTION registrar_auditoria() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO auditoria (tabla, operacion, usuario, datos_nuevos)
        VALUES (TG_TABLE_NAME, TG_OP, current_user, row_to_json(NEW)::jsonb);
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO auditoria (tabla, operacion, usuario, datos_anteriores, datos_nuevos)
        VALUES (TG_TABLE_NAME, TG_OP, current_user, row_to_json(OLD)::jsonb, row_to_json(NEW)::jsonb);
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO auditoria (tabla, operacion, usuario, datos_anteriores)
        VALUES (TG_TABLE_NAME, TG_OP, current_user, row_to_json(OLD)::jsonb);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Paso 3: Crear los Triggers de Auditoría
-- Trigger para la tabla Usuarios
CREATE TRIGGER auditoria_usuarios
AFTER INSERT OR UPDATE OR DELETE ON Usuarios
FOR EACH ROW
EXECUTE FUNCTION registrar_auditoria();

-- Trigger para la tabla Ordenes
CREATE TRIGGER auditoria_ordenes
AFTER INSERT OR UPDATE OR DELETE ON Ordenes
FOR EACH ROW
EXECUTE FUNCTION registrar_auditoria();

-- Trigger para la tabla Dev
CREATE TRIGGER auditoria_dev
AFTER INSERT OR UPDATE OR DELETE ON Dev
FOR EACH ROW
EXECUTE FUNCTION registrar_auditoria();

```

21) Utilizando un script, configure de la memoria y el disco duro

```

ALTER SYSTEM SET shared_buffers = '2GB';
ALTER SYSTEM SET work_mem = '64MB';
ALTER SYSTEM SET maintenance_work_mem = '512MB';
ALTER SYSTEM SET effective_cache_size = '6GB';

```

-- Ajustar los parámetros de disco

```

ALTER SYSTEM SET max_wal_size = '4GB';
ALTER SYSTEM SET min_wal_size = '1GB';
ALTER SYSTEM SET checkpoint_completion_target = 0.7;
ALTER SYSTEM SET random_page_cost = 1.1;

```

-- Aplicar cambios

```

SELECT pg_reload_conf();

```

22) Utilizando un script, genere una copia de seguridad de la base de datos

Lo mismo que el numero 12

23) Realice un script para programar backups automatizados de su base de datos

Mg. Ing. Raúl Fernández Bejarano

```

CREATE OR REPLACE FUNCTION backup_database()
RETURNS VOID AS $$
DECLARE
    backup_cmd TEXT;
BEGIN
    backup_cmd := 'pg_dump -U ' || current_user || ' -F c -b -v -f C:\Users\cjazz\Documents\2024-II\Base de datos II\Respaldo' || TO_CHAR(NOW(), 'YYYYMMDDHH24MISS') || '.dump BD1fd';
    PERFORM dblink_exec('dbname=mydb', 'SELECT shell('' || backup_cmd || '')');
END;
$$ LANGUAGE plpgsql;

-- Crear la Función para Ejecutar el Backup si Corresponde
CREATE OR REPLACE FUNCTION ejecutar_backup_si_corresponde()
RETURNS TRIGGER AS $$
BEGIN
    IF (current_timestamp::time >= '08:00:00'::time AND current_timestamp::time < '08:01:00'::time) AND
        (EXTRACT(DOW FROM current_timestamp) = 1) THEN -- Lunes a las 8 AM
        PERFORM backup_database();
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Crear el Trigger Asociado con la Tabla Usuarios
CREATE TRIGGER trigger_backup
AFTER INSERT OR UPDATE ON Usuarios
FOR EACH ROW
EXECUTE FUNCTION ejecutar_backup_si_corresponde();

```

24) Utilizando un script, genere la restauración de la base de datos

```

-- Restaurar los datos desde un archivo CSV
COPY Usuarios (Region, Gerente)
FROM 'C:\Users\cjazz\Documents\2024-II\Base de datos II\3 tablas\Usuarios.csv'
DELIMITER ','
CSV HEADER;

```

25) Utilizando un script, cree un espejo de la base de datos

Lo mismo que el 11.

```

-- Crear la nueva base de datos a partir de la existente
CREATE DATABASE BD1fd_copy WITH TEMPLATE BD1fd;

-- Asignar privilegios a los usuarios
GRANT ALL PRIVILEGES ON DATABASE BD1fd_copy TO Christian;
GRANT ALL PRIVILEGES ON DATABASE BD1fd_copy TO Jeison;
GRANT ALL PRIVILEGES ON DATABASE BD1fd_copy TO Pedro;

```

26) Utilizando un script, para enviar datos a la base de datos espejo creada

Lo mismo que el 11.

```

-- Crear la nueva base de datos a partir de la existente
CREATE DATABASE BD1fd_copy WITH TEMPLATE BD1fd;

-- Asignar privilegios a los usuarios
GRANT ALL PRIVILEGES ON DATABASE BD1fd_copy TO Christian;
GRANT ALL PRIVILEGES ON DATABASE BD1fd_copy TO Jeison;
GRANT ALL PRIVILEGES ON DATABASE BD1fd_copy TO Pedro;

```

27) Utilizando un script, de permiso a un usuario por un determinado tiempo


```

CREATE OR REPLACE FUNCTION otorgar_permisos_temporales(
    p_usuario TEXT,
    p_tabla TEXT,
    p_duracion INTERVAL
) RETURNS VOID AS $$
DECLARE
    v_comando_otorgar TEXT;
    v_comando_revocar TEXT;
    v_start_time TIMESTAMPTZ := NOW();
BEGIN
    -- Otorgar permisos
    v_comando_otorgar := 'GRANT SELECT, INSERT, UPDATE, DELETE ON ' || p_tabla || ' TO ' || p_usuario;
    EXECUTE v_comando_otorgar;

    -- Esperar el tiempo especificado
    PERFORM pg_sleep(EXTRACT(EPOCH FROM p_duracion)::INT);

    -- Revocar permisos
    v_comando_revocar := 'REVOKE SELECT, INSERT, UPDATE, DELETE ON ' || p_tabla || ' FROM ' || p_usuario;
    EXECUTE v_comando_revocar;

    -- Registrar la acción
    RAISE NOTICE 'Permisos otorgados a % para la tabla % a las %, y revocados a las %', p_usuario, p_tabla, v_start_time, NOW();
END;
$$ LANGUAGE plpgsql;

-- Ejecutar la Función para Otorgar Permisos Temporales
SELECT otorgar_permisos_temporales('Christhian', 'Usuarios', INTERVAL '1 hour');
SELECT otorgar_permisos_temporales('Christhian', 'Ordenes', INTERVAL '1 hour');
SELECT otorgar_permisos_temporales('Christhian', 'Dev', INTERVAL '1 hour');

```

28) Utilizando un script, realice la replicación de bases de datos

29) Explique que es Always On Availability Groups

Always On Availability Groups es una funcionalidad de alta disponibilidad y recuperación ante desastres en SQL Server permite replicar bases de datos entre múltiples servidores, asegurando continuidad operativa mediante failover automático o manual.

30) Explique que es Log Shipping

Log Shipping es una solución de respaldo y recuperación en SQL Server que permite copiar y restaurar periódicamente registros de transacciones (transaction logs) de una base de datos primaria a una o más bases de datos secundarias proporciona una forma sencilla de recuperación ante desastres, pero no admite failover automático.