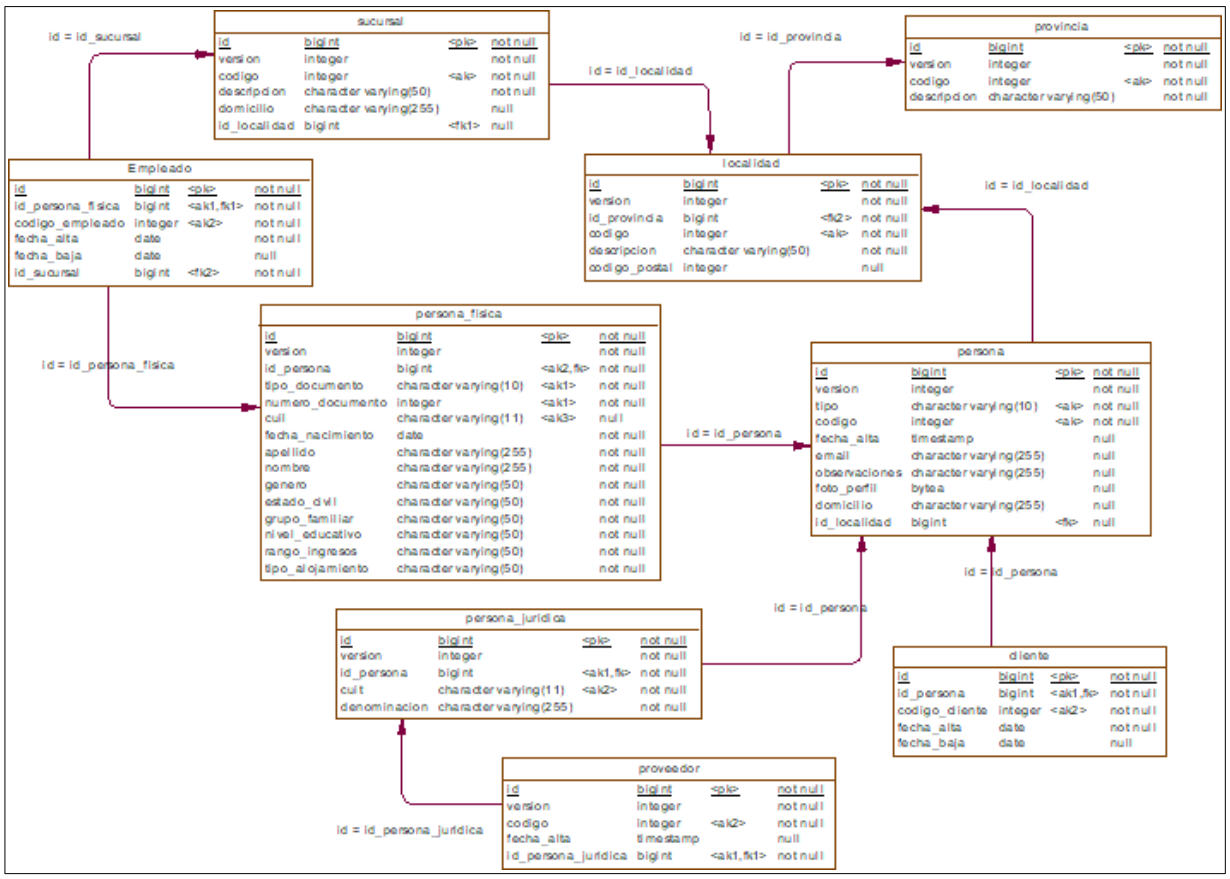
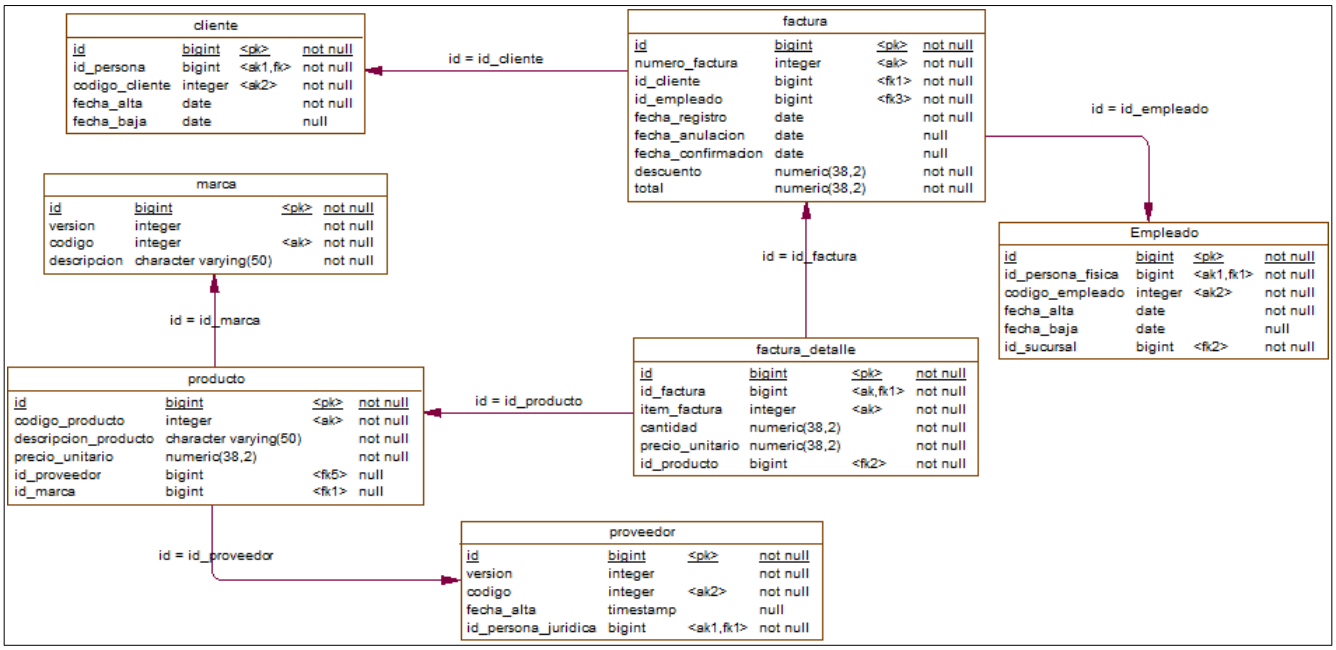
BASES DE DATOS – PARCIAL 2 - 16/11/2024

Se trabajará sobre el modelo físico del sistema de “Gestión” que venimos utilizando en las clases. A continuación se presentan los modelos:

Modelo físico de Personas



Modelo físico de Ventas



PRÁCTICA

Ejercicio 1: Actualización de precios

Ajuste de precios de productos de un proveedor específico.

Se requiere aumentar en un 15% el precio_unitario de todos los productos asociados a un proveedor cuyo codigo es dado como parámetro. Escribir una consulta de UPDATE que permita realizar este ajuste en la tabla producto.

Ampliación sugerida:

Codificar un batch en PostgreSQL que realice el ajuste de precios indicado. Declarar una variable para asignar el código del proveedor y agregar manejo de errores para hacer el proceso más seguro y controlado.

```
DO $$
DECLARE
    v_codigo_proveedor INTEGER := 1735;  -- Asignar valor q corresponda
BEGIN
    – Actualización
    UPDATE producto.producto AS p
    SET precio_unitario = p.precio_unitario * 1.15
    FROM persona.proveedor AS pr
    WHERE p.id_proveedor = pr.id
      AND pr.codigo = v_codigo_proveedor;

    RAISE NOTICE 'Precios actualizados para productos del proveedor con código %',
v_codigo_proveedor;

EXCEPTION
    WHEN OTHERS THEN
        -- Mensaje de error
        RAISE WARNING 'Ocurrió un error al actualizar precios para el proveedor con código %: %',
                      codigo_proveedor, SQLERRM;
END $$;

– Util para probar en bd de gestión
– Asignar datos de id_proveedor en producto para probar con datos:
– Los productos se asignan a los dos primeros proveedores que aparecen, en forma arbitraria.
select p.* from persona.proveedor p
join producto.producto prod on (prod.id_proveedor=p.id)
join venta.factura_detalle fd on (fd.id_producto=prod.id)
limit 2
3291
3292
update producto.producto p
set id_proveedor = 3291
where p.id <=300
update producto.producto p
set id_proveedor = 3292
where p.id >300
```

Ejercicio 2: Consulta de ventas por sucursal

Escribir una consulta SQL que muestre el total de ventas por cada sucursal, ordenado de mayor a menor. La consulta debe devolver los siguientes datos:

- código de la sucursal
- nombre o descripción de la sucursal.
- nombre de la provincia y nombre de la localidad en la que se encuentra la sucursal. Si la sucursal no tiene una localidad asignada, debe mostrar "Provincia desconocida" y "Localidad desconocida"
- total de ventas

Las sucursales con mayor facturación ~~mensual~~ deben aparecer primero. Considerar únicamente facturas confirmadas.

La consulta debe implementarse en SQL estándar en una consulta simple, sin el uso de funciones, procedimientos, cursores ni otros elementos avanzados.

```
SELECT s.codigo AS codigo_sucursal,
       s.descripcion AS nombre_sucursal,
       COALESCE(p.descripcion, 'Provincia desconocida') AS provincia,
       COALESCE(l.descripcion, 'Localidad desconocida') AS localidad,
       SUM(f.total) AS total_ventas
FROM venta.factura AS f
JOIN persona.empleado AS e ON f.id_empleado = e.id
JOIN persona.sucursal AS s ON e.id_sucursal = s.id
LEFT JOIN persona.localidad AS l ON s.id_localidad = l.id
LEFT JOIN persona.provincia AS p ON l.id_provincia = p.id
WHERE f.fecha_confirmacion is not null
GROUP BY s.codigo, s.descripcion, p.descripcion, l.descripcion
ORDER BY total_ventas DESC;

También podrían implementar una solución usando CASE o usando UNION:
```

Ejercicio 3: Vista para DW

Crear una vista llamada ***vista_facturacion_dw*** que permita denormalizar, para su uso en un entorno de data warehousing, las tablas factura y factura_detalle.

Cada fila de esta vista debe representar un detalle de la factura como un evento individual, e incluir las siguientes columnas:

- código del cliente
- número de la factura
- año, mes y día de la fecha de facturación
- total facturado
- codigo del producto
- cantidad vendida
- precio unitario del producto
- costo del producto

La vista debe incluir únicamente los datos de ventas realizadas después del año 2021. Considerar únicamente facturas confirmadas. Toda esta información debe estar consolidada en una sola fila por cada producto vendido en una factura.

```
CREATE OR REPLACE VIEW vista_facturacion_dw AS
	SELECT
	    c.codigo AS codigo_cliente,
	    f.numero AS numero_factura,
	    EXTRACT(YEAR FROM f.fecha) AS año,
	    EXTRACT(MONTH FROM f.fecha) AS mes,
	    EXTRACT(DAY FROM f.fecha) AS dia,
	    f.total AS total_facturado,
	    p.codigo AS codigo_producto,
	    fd.cantidad AS cantidad_vendida,
	    fd.precio_unitario AS precio_unitario,
	    p.costo_unitario AS costo_producto
	FROM venta.factura AS f
	JOIN persona.cliente AS c ON f.id_cliente = c.id
	JOIN venta.factura_detalle AS fd ON f.id = fd.id_factura
	JOIN producto.producto AS p ON fd.id_producto = p.id
	WHERE EXTRACT(YEAR FROM f.fecha) > 2021
	       and f.fecha_confirmacion is not null;
```

TEORÍA

1. ¿Se necesitan permisos específicos para que un usuario ejecute un trigger en una base de datos? Justificar la respuesta. Si corresponde, indicar el comando para otorgar estos permisos.

No se otorgan permisos de ejecución para los triggers a los usuarios, ya que los triggers no se ejecutan directamente por acción de un usuario, sino como consecuencia automática de un evento específico (como una inserción, actualización o eliminación). Debido a esta naturaleza, los triggers se disparan automáticamente cuando ocurre el evento definido, independientemente de quién realice la operación que activa el trigger. Por lo tanto, no existe un comando para otorgar permisos de ejecución sobre triggers, ya que su activación no depende de una invocación explícita por parte del usuario.

2. ¿Qué es un cursor y para qué se utiliza? Describir su utilidad y mencionar las principales sentencias vinculadas al uso de cursores en PostgreSQL o SQL Server.

Un cursor es un objeto de base de datos permite manipular filas de datos de una consulta de forma individual (fila por fila). Son útiles para operaciones complejas en las que cada fila necesita ser procesada secuencialmente, y no pueden abordarse adecuadamente con una simple operación de conjunto.

Definición conceptual: Un cursor es una estructura que representa el contexto de ejecución de una consulta SELECT, manteniendo un puntero al conjunto de filas que se están procesando.

Los cursores son de utilidad en las siguientes situaciones:

- Procesamiento de datos fila por fila: útiles cuando se necesita realizar operaciones secuenciales o lógicas en cada fila de una consulta. Por ejemplo, para ejecutar cálculos complejos o actualizaciones en cada registro de un conjunto de datos.
- Operaciones que no se pueden realizar en conjunto: Cuando el procesamiento en bloque no es adecuado, los cursores permiten aplicar condiciones o transformaciones únicas en cada fila de manera controlada.
- Iteración controlada en entornos críticos: Si una operación en varias filas depende de resultados individuales, los cursores facilitan la lógica y la consistencia.

Las principales sentencias asociadas con los cursores en PostgreSQL y SQL Server incluyen:

DECLARE: Define un cursor asociado a una consulta específica.

OPEN: Abre el cursor, preparando los datos seleccionados para su recorrido.

FETCH: Recupera la siguiente fila del cursor, permitiendo procesar cada fila una por una.

CLOSE: Cierra el cursor y libera los recursos asociados.

DEALLOCATE (solo en SQL Server): Libera completamente el cursor de la memoria.

3.¿Qué es un tablespace en PostgreSQL? Explicar para qué sirve. Crear un tablespace en una carpeta del disco “E:” para guardar los datos, y otro en el disco “F:” para almacenar los índices. Aparte de ambas sentencias de creación, poner un ejemplo de cada caso, para mover una tabla y un índice a su tablespace correspondiente.

Permiten a los DBA definir ubicaciones en el sistema de archivos donde se pueden almacenar los archivos que representan objetos de la base de datos. Una vez creado, se puede hacer referencia a un espacio de tabla por su nombre al crear objetos de base de datos.

Al utilizar tablespaces, un DBA puede controlar la distribución del ESPACIO. Si la partición o el volumen en el que se inicializó el servidor se queda sin espacio y no se puede ampliar, se puede crear un espacio de tabla en una partición diferente y utilizarlo hasta que se pueda reconfigurar el sistema.

También el DBA analiza el patrón de uso de los objetos de la base de datos para optimizar el rendimiento. Por ejemplo, un índice que se utiliza mucho se puede colocar en un disco muy rápido y de alta disponibilidad. Al mismo tiempo, una tabla que almacena datos archivados que rara vez se utilizan o que no son críticos para el rendimiento se puede almacenar en un sistema de disco más lento y menos costoso.

```
CREATE TABLESPACE PGDatos LOCATION 'e:/PG/DatosPostgreSQL'

CREATE TABLESPACE PGIndices LOCATION 'f:/PG/IndicesPostgreSQL'


create table tabla (codigo smallint);

create index idx_tabla on tabla(codigo);


alter table tabla set tablespace PGDatos;

alter index idx_tabla set tablespace PGIndices;
```