Mantenimiento de la Persistencia de los Objetos

1) Características, Ventajas y Desventajas de las Bases de Datos Orientadas a Objetos

Introducción:

Las bases de datos orientadas a objetos (OODB) son sistemas diseñados para almacenar y gestionar datos en forma de objetos, similares a los usados en lenguajes de programación orientados a objetos como Java o Python. Al ser orientada a objetos cuenta con la capacidad de la persistencia de datos, esto significa que cuenta con con los conceptos de herencia, encapsulación y polimorfismo. Esto permite trabajar con datos complejos, ideal para la gestión de bases de datos de una biblioteca.

Características:

- Almacenamiento de objetos: Los datos se almacenan como objetos, con atributos y métodos, lo que permite representar entidades complejas (por ejemplo, un objeto “Libro” puede tener vinculado un título, autor, editorial, fecha publicación).

- Soporte para relaciones: Permite la relación entre objetos (por ejemplo, un autor puede estar vinculado a más de un libro en la base de datos).

- Herencia y polimorfismo: Soportan herencia entre clases de objetos y polimorfismo, como en la programación orientada a objetos.

- Datos complejos: Permite colecciones (arrays, listas) y estructuras anidadas (por ejemplo, un objeto JSON en PostgreSQL).

- Integración con lenguajes OO: Se permite la interación con lenguajes como Java, Python o C++.

Ventajas:

- Modelado Natural: Representan datos complejos de una manera mas accesible, excelente para una aplicación como una biblioteca donde los libros tienen múltiples atributos y relaciones.

- Flexibilidad: Soportan estructuras dinámicas, como colecciones, datos anidados, sin esquemas rígidos.

- Rendimiento en Objetos Complejos: Es más eficiente para manejar datos no relacionales.(por ejemplo, lista de géneros de un libro).

Desventajas:

- Curva de aprendizaje: Requiere conocimientos de programación orientada a objetos en lenguajes específicos. Lo que puede aumentar la dificultad al utilizar este método.

- Menor soporte para consultas Ad-Hoc: A diferencia de las bases relacionales, las consultas complejas pueden ser más lentas.
- Falta de estandarización: No existe un estándar universal con SQL para todas las OODB, esto aumenta su dificultad.

- Escalabilidad limitada: Para grandes volúmenes de datos, puede ser menos eficiente que las bases de datos relacionales.

Conclusión:

Las bases de datos orientadas a objetos son una herramienta ideal para la gestión de una biblioteca, donde los datos como libros o autores se editan como objetos. Aunque existe una curva de aprendizaje y su aplicación puede ser difícil, su integración con lenguajes de programación la hace muy útil.

2) Crear Base de datos en PostgreSQL

Utilizaré PG Admin para crear la base de datos en PostgreSQL y también el schema.

```
1
2   CREATE database biblioteca;
3
4   CREATE SCHEMA biblioteca;
```

Primero he creado una tabla de prueba y he insertado datos para verificar que funciona todo correcto.

```
8  CREATE TABLE biblioteca.libros_prueba (
9      id serial primary key,
10     titulo varchar(100)
11 );
12
13 INSERT INTO biblioteca.libros_prueba (titulo) VALUES ('Prueba libro');
14
15 SELECT * FROM biblioteca.libros_prueba;
```

Salida de la búsqueda es correcta:

	id [PK] integer	titulo character varying (100)
1	1	Prueba libro

Luego elimino la tabla de prueba:

	id [PK] integer	titulo character varying (100)
1	1	Prueba libro

3) Creación de tablas y inserción de datos

Primero creo mis propios tipo de datos para utilizarlos como objetos en la tablas.

A) El primer objeto que creo es tipo_generos para almacenar los diferentes géneros que un libro puede tener.

```
CREATE TYPE biblioteca.tipo_generos AS (
    generos TEXT[]
);
```

B) El segundo objeto es tipo_editorial que permite almacenar nombre de la editorial y el año de publicación del libro.

```
CREATE TYPE biblioteca.tipo_editorial AS (
    nombre varchar(100),
    anio_publicación INT
);
```

C) El tercer objeto que creo es el de tipo_libro para almanecar la información de cada libro en formato objeto.

```
CREATE TYPE biblioteca.tipo_libro AS (
    titulo VARCHAR(100),
    generos biblioteca.tipo_generos,
    disponible BOOLEAN,
    anio_publicacion INT
);
```

D) Luego creo las tablas autores, editoriales y libros.

```
CREATE TABLE biblioteca.autores (
    id_autor SERIAL PRIMARY KEY,
    nombre_autor varchar(100) NOT NULL
);

CREATE TABLE biblioteca.editoriales (
    id_editorial SERIAL PRIMARY KEY,
    info_editorial biblioteca.tipo_editorial NOT NULL
);

CREATE TABLE biblioteca.libros (
    id_libro SERIAL PRIMARY KEY,
    info_libro biblioteca.tipo_libro NOT NULL,
    id_autor INT REFERENCES biblioteca.autores(id_autor),
    id_editorial INT REFERENCES biblioteca.editoriales(id_editorial)
);
```

Cada tabla contiene columnas necesarias para que cumpla su función. Al finalizar la creación de tablas agrego información para comprobar que funcione correctamente.

```
INSERT INTO biblioteca.autores (nombre_autor) VALUES ('J.R.R Tolkien');

-- Insertamos editorial
INSERT INTO biblioteca.editoriales (info_editorial)
VALUES (row('Anaya', 1954)::biblioteca.tipo_editorial);

-- Insertamos Datos del libro
INSERT INTO biblioteca.libros (info_libro)
VALUES(
    ROW('El Señor de los Anillos', ROW(ARRAY['Fantasía', 'Aventura'])::biblioteca.tipo_generos, TRUE
);
```

Decidí agregarle una columna a la tabla autores con el nombre de “titulos” que refleje si un autor tiene mas de 1 libro en la biblioteca.

```
ALTER TABLE biblioteca.autores
ADD COLUMN titulos INT DEFAULT 0;
```

Ahora realizo una consulta a las 3 tablas para verificar que la información de la primera fila insertada esté correcta.

	id_autor integer	nombre_autor character varying (100)	titulos integer	id_libro integer	titulo character varying (100)	id_autor integer	id_editorial integer	generos biblioteca.tipo_generos	disponible boolean	id_editorial integer	info_editorial biblioteca.tipo_editorial
1	1	J.R.R Tolkien	0	2	El Señor de los Anillos	1	1	("{Fantasía,Aventura}")	true	1	(Anaya,1954)

4) Implementación de consultas SQL

1) Consultas de búsqueda.

A) Buscar libro por su Título de manera parcial o exacta utilizando el “ILIKE”.

```
SELECT l.id_libro, (l.info_libro).titulo, a.nombre_autor
FROM biblioteca.libros as l
JOIN biblioteca.autores as a ON l.id_autor = a.id_autor
WHERE (l.info_libro).titulo ILIKE '%señor%';
```

- Explicación: Al utilizar “ILIKE” se puede buscar una palabra clave que es insensible a las minúsculas o mayúsculas. Al buscar %señor% en minúsculas es capaz de encontrarlo y devolverlo.

	id_libro integer	titulo character varying (100)	autor character varying (100)	generos text[]
1	2	El Señor de los Anillos	J.R.R Tolkien	{Fantasía,Aventura}

B) Otro ejemplo de la consulta pero buscando al autor en mayúsculas.

```
SELECT libros.id_libro, libros.titulo, autores.nombre_autor AS autor, (libros.generos).generos as generos
FROM biblioteca.libros
JOIN biblioteca.autores ON libros.id_autor = autores.id_autor
WHERE autores.nombre_autor ILIKE '%TOLKIEN%';
```

C) Buscar por Género.

```
SELECT l.id_libro, (l.info_libro).titulo, a.nombre_autor, ((l.info_libro).generos).generos AS generos
FROM biblioteca.libros l
JOIN biblioteca.autores a ON l.id_autor = a.id_autor
WHERE 'Fantasía' = ANY(((l.info_libro).generos).generos)
ORDER BY l.id_libro ASC;
```

2) Consulta de actualización.

Consultas para actualizar la información ya almacenada en las tablas, por ejemplo título, géneros, editorial o disponibilidad.

Actualización del título del libro utilizando Update y Set

```
UPDATE biblioteca.libros
SET info_libro = ROW(
    'El Hobbit',
    (info_libro).generos,
    (info_libro).disponible,
    (info_libro).anio_publicacion
)::biblioteca.tipo_libro
WHERE id_libro = 2;
```

Actualización de genero.

```
UPDATE biblioteca.libros
SET info_libro = ROW(
    (info_libro).titulo,
    ROW(ARRAY['Fantasía', 'Aventura', 'Épica'])::biblioteca.tipo_generos,
    (info_libro).disponible,
    (info_libro).anio_publicacion
)::biblioteca.tipo_libro
WHERE id_libro = 2;
```

Actualización de la editorial de un libro

```
-- Cambiar la editorial de un libro
UPDATE biblioteca.libros
SET id_editorial = 2
WHERE id_libro = 2;
```

Marcar un libro como no disponible (False)

```
UPDATE biblioteca.libros
SET info_libro = row(
      (info_libro).titulo,
      (info_libro).generos,
      FALSE,
      (info_libro).anio_publicacion
)::biblioteca.tipo_libro
where id_libro = 4;
```

3) Consultas de eliminación.

Con estas consultas podemos eliminar libros, autores o editoriales de la base datos de la biblioteca.

Eliminamos libro por su id

```
DELETE FROM biblioteca.libros
WHERE id_libro = 1;
```

4) Trigger para actualizar titulos automáticamente

Para mantener la columna de títulos actualizada cada vez que se agreguen libros, voy a crear un trigger que cuente cuando se inserte un libro o se elimine.

```
CREATE EXTENSION plpgsql;
CREATE OR REPLACE FUNCTION biblioteca.actualizar_titulos()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE biblioteca.autores a
    SET titulos = (
        SELECT COUNT(l.id_libro)
        FROM biblioteca.libros l
        WHERE l.id_autor = a.id_autor
    );
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Creación de trigger para ejecutar función:

```
CREATE TRIGGER trigger_actualizar_titulos
AFTER INSERT OR UPDATE OF id_autor OR DELETE
ON biblioteca.libros
FOR EACH STATEMENT
EXECUTE FUNCTION biblioteca.actualizar_titulos();
```

Explicación de función y trigger:

- El trigger se activa después de insertar, actualizar o eliminar un libro
- La función actualizar_titulos() recalcula los títulos para cada autor contando sus libros en la tabla libros.
- Con FOR EACH STATEMENT se actualiza todos los contenedores de una vez.

Ahora probamos el tirgger insertando dos libros para diferentes autores:

```
INSERT INTO biblioteca.libros (info_libro, id_autor, id_editorial)
VALUES (
    ROW('Harry Potter y la Cámara Secreta', ROW(ARRAY['Fantasía', 'Juvenil']):
    (SELECT id_autor FROM biblioteca.autores WHERE nombre_autor = 'J.K Rowling
    (SELECT id_editorial FROM biblioteca.editoriales WHERE (info_editorial).no
);
```

```
INSERT INTO biblioteca.libros (info_libro, id_autor, id_editorial)
VALUES (
    ROW('El Silmarillion', ROW(ARRAY['Fantasía', 'Épica'])::biblioteca.tipo
    (SELECT id_autor FROM biblioteca.autores WHERE nombre_autor = 'J.R.R To
    (SELECT id_editorial FROM biblioteca.editoriales WHERE (info_editorial)
);

INSERT INTO biblioteca.libros (info_libro, id_autor, id_editorial)
VALUES (
    ROW('Los Hijos de Húrin', ROW(ARRAY['Fantasía', 'Épica'])::biblioteca.t
    (SELECT id_autor FROM biblioteca.autores WHERE nombre_autor = 'J.R.R To
    (SELECT id_editorial FROM biblioteca.editoriales WHERE (info_editorial)
);
```

Haciendo una rápida búsqueda se puede confirmar que el trigger está funcionando y devuelve el resultado esperado.

	id_autor [PK] integer	nombre_autor character varying (100)	titulos integer
1	1	J.R.R Tolkien	3
2	2	J.K. Rowling	2
3	3	Gabriel García Márquez	1
4	4	Jane Austen	1

5) Definiendo tipo de dato objeto y colección.

1) Colecciones y objetos creados:

```
CREATE TYPE biblioteca.tipo_generos AS (
    generos TEXT[]
);

CREATE TYPE biblioteca.tipo_editorial AS (
    nombre varchar(100),
    anio_publicación INT
);

-- Creación de objeto tipo libro
CREATE TYPE biblioteca.tipo_libro AS (
    titulo VARCHAR(100),
    generos biblioteca.tipo_generos,
    disponible BOOLEAN,
    anio_publicacion INT
);
```

A) tipo_generos(Colección)

- Propósito: Almacena una lista de géneros para un libro como un array de texto. Ejemplo [“Fantasía”, “Juvenil”], de esta manera permite almacenar múltiples géneros por libro.

- Uso en la base de datos: Integrado en **tipo_libro** para realizar búsquedas por genero.

```
SELECT (info_libro).titulo
FROM biblioteca.libros
WHERE 'Fantasía' = ANY(((info_libro).generos).generos);
```

Ventaja: Simplifica la gestión de genero sin requerir una tabla extra, alineándose con las colecciones en bases de datos orientadas a objetos.

Resultado búsqueda:

titulo character varying (100)
Harry Potter y la Piedra Filosofal
El Señor de los Anillos
Harry Potter y la Cámara Secre...
El Silmarillion

B) tipo_editorial(objeto):

- Propósito: Representa la información de una editorial como un objeto con nombre y anio_publicacion por ejemplo, {nombre: 'Bloomsbury', anio_publicacion: 1997}).

- Uso en base de datos: Se utiliza en la tabla editoriales para almacenar los datos estructurados para poder consultarlo.

```
SELECT (info_editorial).nombre
FROM biblioteca.editoriales
WHERE (info_editorial).anio_publicacion = 1997;
```

Ventaja: Estructura los datos como un objeto, facilitando consultas.

Resultado búsqueda:

titulo character varying (100)
El Señor de los Anillos
Harry Potter y la Cámara Secre...
El Silmarillion
Los Hijos de Húrin

C) tipo_libro(objeto):

- Propósito: Encapsula los atributos de un libro (titulo, generos, disponible, anio_publicacion) como un objeto.

- Uso en base de datos: Lo utilizo en la tabla libros(columna info_libro) para almacenar datos del libro en una sola estructura.

```
SELECT (info_libro).titulo, ((info_libro).generos).generos
FROM biblioteca.libros
WHERE (info_libro).disponible = TRUE;
```

- Ventaja: Refuerza el enfoque orientado a objetos, centralizando los atributos del libro y permitiendo consultas estructuradas.

Resultado búsqueda:

titulo character varying (100)	generos text[]
Harry Potter y la Piedra Filosofal	{Fantasía,Juvenil}
Orgullo y Prejuicio	{Romance,Clásico}
1984	{Distopía,"Ciencia Ficción"}
Asesinato en el Orient Express	{Misterio,Crimen}