2.- Datos Agregados en SQL

En esta práctica veremos como gestionar datos agregados con el lenguaje SQL utilizando tres representaciones distintas, primero usando las estructuras proporcionadas por el modelo objeto-relacional y después usando representaciones jerárquicas en XML y JSON.

2.1- Bases de datos objeto-relacionales (tipos compuestos y arrays)

En esta parte de la práctica nos centraremos en el uso de tipos compuestos y arrays en PostgreSQL para gestionar tablas con estructura compleja. Para mantener nuestra base de datos ordenada, vamos a crear un nuevo esquema "agregados" dentro de nuestra base de datos "bdge".

2.1.1 Tipos de dato compuestos

En la siguiente página web de la documentación de PostgreSQL hay información sobre los tipos compuestos:

https://www.postgresql.org/docs/13/rowtypes.html

El sistema de tipos de postgreSQL tiene constructores de tipo ROW, para tipos compuestos, y ARRAY para colecciones. Un campo del resultado de una consulta puede tener un tipo de dato ROW, es decir, un tipo de dato que contiene varias columnas anidadas. Un campo de tipo ARRAY, puede tener también como tipo de dato de cada uno de sus elementos un tipo ROW. Sin embargo, al contrario de lo que ocurren en el estándar de SQL, no se pueden crear directamente estructuras de tabla que tengan columnas de estos tipos ROW si no se crea previamente el tipo con una instrucción CREATE TYPE. En el siguiente trozo de código podemos ver la sintaxis para la creación de literales de tipo ROW.

```
select 45 as id, (24, 'Hola') as elemento_compuesto
```

La siguiente consulta genera una tabla con un tipo de dato ROW sin nombre, pero al intentar crear la tabla que almacene ese resultado el sistema genera un error.

```
select pels.id, pels.titulo,
    case
      when pels.coleccion is null then null
      else (c.id, c.nombre)
    end  as coleccion
from public.peliculas pels left join public.colecciones c  on  pels.coleccion =c.id
order by pels.presupuesto desc
limit 10;

create table peliculas10 as
select pels.id, pels.titulo,
    case
      when pels.coleccion is null then null
      else (c.id, c.nombre)
    end  as coleccion
from public.peliculas pels left join public.colecciones c  on  pels.coleccion =c.id
order by pels.presupuesto desc
limit 10
```

?

Para poder crear nuestra tabla de películas, creamos primero todos los tipos de dato compuestos necesarios.

```
create type coleccion as (
        id int4,
        nombre text
);

create type genero as (
        id int4,
        nombre text
);


CREATE type idioma as (
        id text,
        nombre text
);

CREATE type pais as (
        id text,
        nombre text
);
```

```
CREATE type productora as (
        id int4,
        nombre text
);

CREATE type persona as (
        id int4,
        nombre text
);

CREATE type miembro_reparto as (
        persona persona,
        personaje text
);

CREATE type miembro_personal as (
        persona persona,
        departamento text,
        trabajo text
);
```

Es importante observar la decisión de no incluir el campo "orden" en el tipo "miembro_reparto". Esto se debe a que, al soportar las colecciones con tipos de dato ARRAY, estos ya tienen un orden definido para sus elementos, con lo que este campo ya no será necesario.

2.1.2 Tipos de dato ARRAY

En el manual de PostgreSQL hay información sobre el tipo array y sus funciones y operadores:

https://www.postgresql.org/docs/13/arrays.html

https://www.postgresql.org/docs/13/functions-array.html

Los arrays pueden tener cualquier número de dimensiones y se soportan dos formas de sintaxis. La forma con la palabra clave ARRAY es la usada por el estándar SQL, pero solo se puede usar con arrays unidimensionales. En esta práctica nos centraremos en el uso de arrays unidimensionales. En el siguiente trozo de código tenemos ejemplos de sintaxis de literales de tipo array, tanto usando la palabra clave ARRAY como usando directamente la sintaxis no estándar de PostgreSQL. Hay ejemplos de arrays de dimensiones 1 y 2. La segunda instrucción falla porque un array solo puede tener un tipo de dato para sus elementos, y la segunda falla porque un array no pude tener distintos tamaños de columnas en sus filas (debe de ser una matriz).

```
select cast('{12, 334, 233}' as  integer array), array['hola', 'mundo']

select array[12, 12,'hola']

select array[array[12,6], array[12,12,23]]

select array[array[45,31,45], array[46,13,89]]
```

Con la siguiente instrucción DDL creamos la tabla necesaria para almacenar nuestras películas.

```
CREATE TABLE peliculas (
        id int4 NOT NULL,
        titulo text,
        coleccion coleccion,
        para_adultos bool,
        presupuesto int4,
        idioma_original idioma,
        titulo_original text,
        sinopsis text,
        popularidad float8,
        fecha_emision date,
        ingresos int8,
        duracion float8,
        lema text,
        generos genero ARRAY,
        idiomas_hablados idioma ARRAY,
        paises pais ARRAY,
        personal miembro_personal ARRAY,
        reparto miembro_reparto ARRAY,
        productoras productora ARRAY,
        CONSTRAINT peliculas_pkey PRIMARY KEY (id)
);
```

Vamos ahora a construir las filas de esta nueva tabla, a partir de las tablas de nuestro modelo relacional.

```
insert into peliculas
select pels.id as id,
       pels.titulo as titulo,
       case
          when c.id is null then null
          else cast((c.id,c.nombre) as coleccion)
       end as coleccion,
       pels.para_adultos as para_adultos,
       pels.presupuesto as presupuesto,
       case
          when i1.id is null then null
          else cast((i1.id, i1.nombre) as idioma)
       end as idioma_original,
       pels.titulo_original as titulo_original,
       pels.sinopsis as sinopsis,
       pels.popularidad as popularidad,
       pels.fecha_emision as fecha_emision,
       pels.ingresos as ingresos,
       pels.duracion as duracion,
       pels.lema as lema,
       ARRAY(select cast((g.id, g.nombre) as genero)
        from public.pelicula_genero pg, public.generos g
        where pels.id =pg.pelicula  and pg.genero = g.id) as generos,
       ARRAY(select cast((i2.id, i2.nombre) as idioma)
        from public.pelicula_idioma_hablado pih, public.idiomas i2
        where pels.id=pih.pelicula and pih.idioma=i2.id) as idiomas_hablados,
       ARRAY(select cast((pa.id, pa.nombre) as pais)
        from public.pelicula_pais ppais, public.paises pa
        where pels.id =ppais.pelicula and ppais.pais =pa.id) as paises,
       ARRAY(select cast((cast((per1.id,per1.nombre) as persona), pper.departamento, pper.trabajo) as miembro_personal)
       from public.pelicula_personal pper, public.personas per1
       where pels.id = pper.pelicula and pper.persona = per1.id) as personal,
       ARRAY(select cast((cast((per2.id,per2.nombre) as persona), prep.personaje) as miembro_reparto)
       from public.pelicula_reparto prep, public.personas per2
       where pels.id =prep.pelicula and prep.persona = per2.id
       order by prep.orden) as reparto,
       ARRAY(select cast((prods.id, prods.nombre) as productora)
        from public.pelicula_productora pprod, public.productoras prods
        where pels.id = pprod.pelicula and pprod.productora = prods.id) as productoras
from public.peliculas pels left join public.colecciones c on pels.coleccion = c.id
     left join public.idiomas i1 on pels.idioma_original=i1.id
```

2.1.3 Consulta con tipos compuestos y arrays

Para acceder a los campos de un tipo compuesto usamos la notación ".". Es necesario tener en cuenta que esta misma notación se utiliza para acceder a los campos de una tabla, con lo que puede generar errores de sintaxis inesperados.

1.- Obtener todas las películas de las colecciones que contenga la palabra "Wars". Muestra el título de la película y el nombre de la colección. Ordena el resultado por el nombre de la colección

```
select titulo, (coleccion).nombre as coleccion
from peliculas
where (coleccion).nombre like '%Wars%'
order by (coleccion).nombre
```

Fijarse en la necesidad de usar paréntesis para asegurarse que accedemos primero a la columna de la tabla y después a un campo dentro de esa columna. Si no usamos los paréntesis el sistema cree que estamos intentando acceder al campo nombre de una tabla "coleccion" que no hemos puesto en la cláusula "from". Ahora, como todos nuestros datos están agregados en una única tabla, ya no es necesario realizar la operación de JOIN para acceder a los datos de la colección de la película.

2.- Para cada película de la colección "Star Wars Collection" obtener su título, el nombre y trabajo del primer miembro del personal, y los datos de los cinco primeros miembros del reparto. Ordena el resultado por fecha de emisión.

```
select titulo, (personal[1]).persona.nombre as nombre_personal, (personal[1]).trabajo as trabajo_personal,
       reparto[1:5] as reparto
from peliculas
where (coleccion).nombre = 'Star Wars Collection'
order by fecha_emision
```

El editor solo nos permite ver el primer elemento del array. Podemos transformarlo a tipo "text" para verlos todos.

3.- Obtener un listado de los miembros del personal de la película "The Empire Strikes Back". Ordena el resultado por departamento y trabajo.

```
select (persona).nombre, departamento, trabajo
from peliculas, unnest(personal) per(persona, departamento, trabajo)
where titulo = 'The Empire Strikes Back'
order by departamento, trabajo
```

4.- Obtener un listado del reparto de la película "The Empire Strikes Back", ordenado por el orden en el que aparecen en el array.

```
select orden, (persona).nombre, personaje
from peliculas, unnest (reparto) with ordinality r(persona, personaje, orden)
where titulo = 'The Empire Strikes Back'
order by orden
```

2.2 Gestión de documentos XML con SQL

PostgreSQL proporciona un tipo de datos y un conjunto de funciones, similares a las que define el estándar de SQL, para la gestión de documentos XML. Estas funciones permiten tanto la generación de contenido XML a partir de datos relacionales como el acceso a partes de un documento XML.

Las funciones "xmlparse" y "xmlserialize" permiten el parseo de un documento XML para generar un elemento del tipo de datos "xml" y la obtención del texto de un elemento del tipo de datos "xml".

```
CREATE TABLE docxml (
titulo varchar(500) not null primary key,
doc xml
);

insert into docxml values ('ejemploHTML.xml', xmlparse(DOCUMENT
'<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!--Este es un comentario y no sera tomado en cuenta por el navegador
-->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">
<head>
<title>Titulo de la pagina</title>
</head>
<body>
<p>Primer documento XHTML, es decir un, Hola mundo</p>
<p>En este segundo párrafo voy a incluír un <b>texto en
negrita</b></p>
En este texto libre coloco un <b>salto de linea</b> <br/>
para continuar con algunos <b>caracteres especiales</b>. En concreto:
<br/>
El caracter &amp;amp; se usa para escribir los caracteres especiales.<br/>
Este mismo caracter especial se escribiría &amp;amp;amp.<br/>
Tampoco se pueden usar los <b>signos mayor y menor</b> &amp;lt;(&amp;amp;lt) y
&amp;gt;(&amp;amp;gt);<br/>
Si quieres introducir <b>espacios en blanco</b> &amp;nbsp; &amp;nbsp; tienes
que usar &amp;amp;nbsp
</body>
</html>'));

select xmlserialize(DOCUMENT doc as text)
from docxml
where titulo like '%HTML%';
```

2.2.1 Generación de contenido XML a partir de datos relacionales

Para crear contenido XML a partir de datos relacionales podemos usar las funciones "xmlelement"(crea un elmento, posiblemente con atributos), "xmlforest" (crea varios elementos a partir de varias expresiones), "xmlagg" (agrega varios elementos), "xmlconcat" (concatena varios trozos de xml).

```
create table peliculasxml as
select pels.id as id,
       pels.titulo as titulo,
       case
          when c.id is null then null
          else xmlelement(name coleccion, xmlattributes(c.id as id), c.nombre)
       end as coleccion,
       pels.para_adultos as para_adultos,
       pels.presupuesto as presupuesto,
       case
          when i1.id is null then null
          else xmlelement(name idioma, xmlattributes(i1.id as id), i1.nombre)
       end as idioma_original,
       pels.titulo_original as titulo_original,
       pels.sinopsis as sinopsis,
       pels.popularidad as popularidad,
       pels.fecha_emision as fecha_emision,
       pels.ingresos as ingresos,
       pels.duracion as duracion,
       pels.lema as lema,
       xmlelement(
        name generos,
        (select xmlagg(xmlelement(name genero, xmlattributes(g.id as id), g.nombre))
        from public.pelicula_genero pg, public.generos g
        where pels.id =pg.pelicula  and pg.genero = g.id)
       ) as generos,
       xmlelement(
        name idiomas,
        (select xmlagg(xmlelement(name idioma, xmlattributes(i2.id as id), i2.nombre))
        from public.pelicula_idioma_hablado pih, public.idiomas i2
        where pels.id=pih.pelicula and pih.idioma=i2.id)
       ) as idiomas_hablados,
       xmlelement(
        name paises,
        (select xmlagg(xmlelement(name pais, xmlattributes(pa.id as id), pa.nombre))
        from public.pelicula_pais ppais, public.paises pa
        where pels.id =ppais.pelicula and ppais.pais =pa.id)
       ) as paises,
       xmlelement(
        name personal,
        (select xmlagg(xmlelement(name trabajador, xmlattributes(per1.id as id, pper.departamento as departamento, pper.trabajo as
trabajo), per1.nombre))
        from public.pelicula_personal pper, public.personas per1
        where pels.id = pper.pelicula and pper.persona = per1.id)
       ) as personal,
       xmlelement(
        name reparto,
        (select xmlagg(xmlelement(name miembroreparto, xmlattributes(per2.id as id, prep.personaje as personaje), per2.nombre) order by
prep.orden)
        from public.pelicula_reparto prep, public.personas per2
         where pels.id =prep.pelicula and prep.persona = per2.id
        )
       ) as reparto,
       xmlelement(
        name productoras,
        (select xmlagg(xmlelement(name productora, xmlattributes(prods.id as id), prods.nombre))
        from public.pelicula_productora pprod, public.productoras prods
        where pels.id = pprod.pelicula and pprod.productora = prods.id)
       ) as productoras
from public.peliculas pels left join public.colecciones c on pels.coleccion = c.id
     left join public.idiomas i1 on pels.idioma_original=i1.id
```

Los agregados que antes almacenábamos con arrays de valores de tipos compuestos, ahora se almacenan en elementos xml con estructura compleja.

2.2.2 Consulta de xml con xpath

5.- Obtener un listado de los miembros del personal de la película "The Empire Strikes Back". Ordena el resultado por departamento y trabajo.

```
select xmlserialize(content (xpath('/trabajador/text()', x))[1] as text) as nombre,
           xmlserialize(content (xpath('/trabajador/@departamento', x))[1] as text) as departamento,
           xmlserialize(content (xpath('/trabajador/@trabajo', x))[1] as text) as trabajo
from peliculasxml pel, unnest(xpath('/personal/trabajador', personal)) as t(x)
where titulo = 'The Empire Strikes Back'
order by departamento, trabajo
```

6.- Obtener un listado del reparto de la película "The Empire Strikes Back", ordenado por el orden en el que aparecen en el xml.

```
select orden,
           xmlserialize(content (xpath('/miembroreparto/text()', x))[1] as text) as nombre,
           xmlserialize(content (xpath('/miembroreparto/@personaje', x))[1] as text) as personaje
from peliculasxml pel, unnest(xpath('/reparto/miembroreparto', reparto)) with ordinality as t(x, orden)
where titulo = 'The Empire Strikes Back'
order by orden
```

2.3 Gestión de documentos JSON con SQL

En las siguientes páginas del manual de PostgreSQL hay información sobre los tipos de datos, funciones y operadores proporcionados para gestionar documentos JSON dentro de las tablas con SQL.

https://www.postgresql.org/docs/13/datatype-json.html

https://www.postgresql.org/docs/13/functions-json.html

Al igual que ocurre con XML, PostgreSQL proporciona un tipo de datos "json" y con conjunto de funciones y operadores que permiten tanto generar documentos JSON a partir de datos relacionales como consultar partes de un documento JSON. Además del tipo "json", también se proporciona un tipo más eficiente "jsonb", que almacena los documentos en formato binario y evita el tener que parsear el texto cada vez que hay que procesarlo y un tipo de datos "jsonpath" para poder hacer consultas que extraen parte de los documentos json (como el xpath de XML). Por falta de tiempo no veremos aquí en detalle toda esta funcionalidad, solo lo suficiente para tener una idea inicial de la funcionalidad proporcionada.

2.3.1 Generación de documentos JSON a partir de datos relacionales

```
create table peliculasjson as
select pels.id as id,
       pels.titulo as titulo,
       case
          when c.id is null then null
          else jsonb_build_object('id',c.id, 'nombre',c.nombre)
       end as coleccion,
       pels.para_adultos as para_adultos,
       pels.presupuesto as presupuesto,
       case
          when i1.id is null then null
          else jsonb_build_object('id',i1.id, 'nombre',i1.nombre)
       end as idioma_original,
       pels.titulo_original as titulo_original,
       pels.sinopsis as sinopsis,
       pels.popularidad as popularidad,
       pels.fecha_emision as fecha_emision,
       pels.ingresos as ingresos,
       pels.duracion as duracion,
       pels.lema as lema,
       (select jsonb_agg(jsonb_build_object('id',g.id, 'nombre',g.nombre))
        from public.pelicula_genero pg, public.generos g
        where pels.id =pg.pelicula  and pg.genero = g.id) as generos,
       (select jsonb_agg(jsonb_build_object('id',i2.id, 'nombre',i2.nombre))
        from public.pelicula_idioma_hablado pih, public.idiomas i2
        where pels.id=pih.pelicula and pih.idioma=i2.id) as idiomas_hablados,
        (select jsonb_agg(jsonb_build_object('id',pa.id, 'nombre',pa.nombre))
        from public.pelicula_pais ppais, public.paises pa
        where pels.id =ppais.pelicula and ppais.pais =pa.id) as paises,
        (select jsonb_agg(jsonb_build_object('persona',jsonb_build_object('id',per1.id, 'nombre', per1.nombre),
'departamento',pper.departamento, 'trabajo', pper.trabajo))
        from public.pelicula_personal pper, public.personas per1
        where pels.id = pper.pelicula and pper.persona = per1.id) as personal,
        (select jsonb_agg(jsonb_build_object('persona',jsonb_build_object('id',per2.id, 'nombre', per2.nombre),
'personaje',prep.personaje) order by prep.orden)
        from public.pelicula_reparto prep, public.personas per2
         where pels.id =prep.pelicula and prep.persona = per2.id) as reparto,
        (select jsonb_agg(jsonb_build_object('id',prods.id, 'nombre',prods.nombre))
        from public.pelicula_productora pprod, public.productoras prods
        where pels.id = pprod.pelicula and pprod.productora = prods.id) as productoras
from public.peliculas pels left join public.colecciones c on pels.coleccion = c.id
     left join public.idiomas i1 on pels.idioma_original=i1.id
```

2.3.2 Consulta de datos JSON con SQL

7.- Obtener un listado de los miembros del personal de la película "The Empire Strikes Back". Ordena el resultado por departamento y trabajo.

```
select (a->'persona')->>'nombre' as nombre, a->>'departamento' as departamento, a->>'trabajo' as trabajo
from peliculasjson pel, jsonb_array_elements(personal) as per(a)
where titulo = 'The Empire Strikes Back'
order by departamento, trabajo
```

8.- Obtener un listado del reparto de la película "The Empire Strikes Back", ordenado por el orden en el que aparecen en el xml.

```
select orden, (a->'persona')->>'nombre' as nombre, a->>'personaje' as personaje
from peliculasjson pel, jsonb_array_elements(reparto) with ordinality as reparto(a,orden)
where titulo = 'The Empire Strikes Back'
order by orden
```

Ejercicios

Resuelve cada una de las siguientes consultas utilizando las tres tablas (peliculas, peliculasxml y peliculasjson). Entrega el código SQL generado para cada consulta en un único archivo de texto a través del campus virtual.

E1.- Obtén un listado de películas en las que haya actuado "Penélope Cruz", ordena el listado en orden ascendente por fecha de emisión.

E2.- Obtén un listado de los 10 directores que han generado más beneficios en sus películas. Para cada director obtén el número de películas dirigidas, y el beneficio total.

Última modificación: luns, 13 de setembro de 2021, 15:25