

Taller 5

Fecha: Octubre de 2023

Sistema de habilidades a evaluar: Usar los conceptos del modelo relacional de las bases de datos, de los lenguajes DDL (*Data Definition Language*), DML (*Data Management Language*) del SQL, y del lenguaje PL/pgSQL (*Procedural Language/PostgreSQL Structured Query Language*) para la gestión de bases de datos en motores de bases de datos.

NOTAS:

- Este taller se debe hacer con carácter evaluativo (punto No 6). Representa en total una calificación de 20%.
- Se entregan ejercicios resueltos como ejemplo para el desarrollo de los demás.

Programación en los motores de bases de datos

La programación en un motor de base de datos permite la capacidad de escribir y ejecutar código o scripts directamente en el mismo motor para poder ejecutar operaciones específicas. Esto implica la creación de **procedimientos almacenados**, **funciones**, **disparadores** (*triggers*) u otros artefactos de programación que se ejecutan en el servidor de la base de datos en respuesta a eventos o solicitudes.

Esta forma de programación permite llevar a cabo tareas como la manipulación y el procesamiento de datos dentro de la base de datos, la automatización de procesos de negocio y la implementación de lógica de aplicación directamente en el backend, lo que puede mejorar el rendimiento y la seguridad de la aplicación. Ejemplos comunes incluyen el uso de SQL-PL (*DB2*), PL/SQL (*Oracle*), T-SQL (*Sql Server*), SQL/PSM (*MySQL*) entre otros lenguajes específicos del motor de base de datos para lograr una funcionalidad específica en la base de datos.

El SQL (*Structured Query Language*) se considera un lenguaje de programación declarativa. La programación declarativa es un paradigma de programación en el que se especifica el "qué" se debe hacer en lugar del "cómo" se debe hacer. En SQL, los usuarios y desarrolladores declaran las operaciones que desean realizar en una base de datos, como consultas, inserciones, actualizaciones o eliminaciones, sin especificar la lógica detallada de cómo se deben llevar a cabo esas operaciones.

Puesta a punto de una base de datos


Para poder realizarla es preciso en primer lugar dominar los siguientes conceptos:

- El **modelo de datos relacional** representa una base de datos como un conjunto de tablas.

Cada **tabla** tiene un número de columnas las cuales tienen nombres únicos. Estas columnas corresponden a los atributos o **Campos**. Las tablas a su vez, presentan

reglones con los valores de los atributos para cada una de las instancias (**Registros**). Veamos un ejemplo con la tabla para almacenar las *Monedas*:

Campos



Registros

| | Id | Moneda | Sigla | Simbolo | Emisor | Imagen |
|---|-----|---------------------|-------|---------|--------|--------|
| ▶ | 2 | Afgani | AFN | NULL | NULL | NULL |
| | 91 | Ariary malgache | MGA | NULL | NULL | NULL |
| | 138 | Baht | THB | NULL | NULL | NULL |
| | 111 | Balboa | PAB | NULL | NULL | NULL |
| | 48 | Birr etíope | ETB | NULL | NULL | NULL |
| | 154 | Bolívar | VEF | NULL | NULL | NULL |
| | 155 | Bolívar soberano | VES7 | NULL | NULL | NULL |
| | 19 | Boliviano | BOB | NULL | NULL | NULL |
| | 54 | Cedi ghanés | GHS | NULL | NULL | NULL |
| | 74 | Chelín keniano | KES | NULL | NULL | NULL |
| | 131 | Chelín somalí | SOS | NULL | NULL | NULL |
| | 146 | Chelín tanzano | TZS | NULL | NULL | NULL |
| | 148 | Chelín ugandés | UGX | NULL | NULL | NULL |
| | 37 | Colón costarrice... | CRC | NULL | NULL | NULL |
| | 135 | Colon Salvador... | SVC | NULL | NULL | NULL |
| | 106 | Córdoba | NIO | NULL | NULL | NULL |

Existe una correspondencia directa entre el concepto de tabla y el concepto matemático de una relación puesto que cada columna de una tabla representa una relación entre un conjunto de valores. De esta manera, una tabla es un conjunto de relaciones, lo cual se denota así:

$$\begin{matrix} n \\ X D_i \\ i = 1 \end{matrix}$$

Donde:

- n es el número de columnas de la tabla
- D_i es el conjunto de valores para la columna i .

A cada línea de una tabla se le conoce también como **tupla**. Para indicar un valor de un atributo en una tupla determinada se utiliza la notación:

$$t_i [\text{atributo}]$$

Donde

- i es el número de la tupla
- *atributo* es el nombre del atributo al cual corresponden al valor

El esquema de una tabla se denotaría así:

```
Nombre tabla (atributo: Dominio, ...)
```

Por ejemplo, para la tabla anterior, su esquema sería :

```
Moneda (Id: entero,  
        Moneda : cadena,  
        Sigla : cadena,  
        Simbolo: cadena,  
        Emisor: cadena,  
        Imagen: bytes)
```

y una variable de tupla de la tabla sería:

```
t4 (Moneda) = 'Balboa'
```

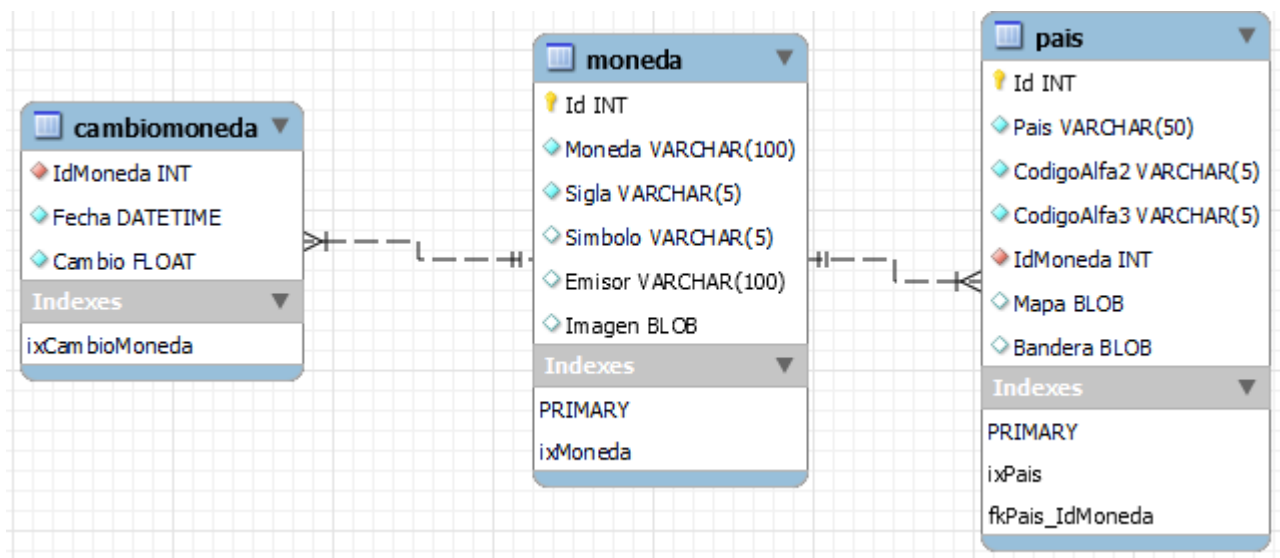
Lo cual quiere decir que el atributo **Moneda** en la tupla **4** equivale a la cadena **'Balboa'**.

El usuario de un sistema de base de datos puede realizar consultas de estas tablas, inserción de nuevas tupas o actualización de las ya existentes. La expresión de dichas operaciones se realiza mediante unos lenguajes conocidos como **Lenguajes de Consulta Relacional**, como lo es el SQL que se detalla más adelante.

- Actualmente, todos los sistemas de gestión de bases de datos modernos almacenan y tratan la información utilizando el **Modelo relacional**. El nombre relacional procede del hecho de que cada registro de la base de datos contiene información relacionada con un tema y sólo con ese tema. Además, los datos de dos clases de información (tal como países y monedas) pueden ser manipulados como una única entidad basada en los valores de los datos relacionados. Por ejemplo, sería una redundancia almacenar la información sobre el nombre y la sigla de una moneda con todos los países donde operan. De modo que, en un sistema relacional, la información sobre los países contiene algún dato que puede ser utilizado para enlazar cada país con su respectiva moneda. En un **Sistema de Gestión de Base de Datos Relacionales (DBMS)** el sistema trata todos los datos en tablas. Cada **tabla** almacena información sobre un tema (como por ejemplo los autores) y dispone de unas columnas (**campos**) que contienen los diferentes tipos de información sobre ese tema (como por ejemplo el nombre) y de unas filas (**registros**) que describen todos los atributos de una única instancia del tema (por ejemplo, los datos de la moneda "Euro"). Incluso cuando se consulta la base de datos (**consulta**) para buscar información procedente de una o más tablas, el resultado es siempre algo semejante a otra tabla.

La forma como la información de una tabla está relacionada con la información de otra tabla se llamada **Relación**. Por ejemplo, las *monedas* tienen una relación uno a muchos con los *países* ya que una moneda puede ser oficial en muchos países y un país sólo tiene una moneda oficial. Los *cambios de moneda* tienen una relación varios a una con las monedas ya que una moneda puede tener registros de varios cambios (uno por día) y un cambio sólo corresponde a una moneda.

En resumen, el modelo de datos relacional representa una base de datos como un conjunto de tablas y unas relaciones entre las tablas. La siguiente gráfica ilustra el **Diagrama Relacional** obtenido en el DBMS MySQL para la base de datos que se está trabajando en el ejercicio:



Para poder crear dicha base de datos es preciso comprender el siguiente tema.

- El **Lenguaje de Definición de Datos** (en inglés DDL, **Data Definition Language**) es un lenguaje especial que permite la especificación de un esquema de una base de datos por medio de una serie de definiciones.

El resultado de la compilación de las proposiciones en DDL es un conjunto de tablas que se almacenan en un archivo especial conocido como Diccionario de Datos.

Por ejemplo, para la base de datos del ejercicio, las siguientes serían las instrucciones en lenguaje DDL para crear la base de datos en el motor **PostgreSQL**:

```
--Ejecutar primero
CREATE DATABASE Monedas;

--Para las siguientes instrucciones, se debe cambiar la conexión
```

```
-- Crear tabla MONEDA
CREATE TABLE Moneda(
    Id SERIAL PRIMARY KEY,
    Moneda VARCHAR(100) NOT NULL,
    Sigla VARCHAR(5) NOT NULL,
    Simbolo VARCHAR(5) NULL,
    Emisor VARCHAR(100) NULL,
    Imagen bytea NULL
);

-- Crear indice para MONEDA ordenado por MONEDA
CREATE UNIQUE INDEX ixMoneda
    ON Moneda(Moneda);

/* Crear tabla CAMBIOMONEDA */
CREATE TABLE CambioMoneda(
    IdMoneda INTEGER NOT NULL,
    CONSTRAINT fkCambioMoneda_IdMoneda FOREIGN KEY (IdMoneda)
REFERENCES Moneda(Id),
    Fecha DATE NOT NULL,
    Cambio FLOAT NOT NULL
);

-- Crear indice para CAMBIOMONEDA ordenado por MONEDA, FECHA */
CREATE UNIQUE INDEX ixCambioMoneda
    ON CambioMoneda(IdMoneda, Fecha);

--Crear la tabla PAIS
CREATE TABLE Pais(
    Id SERIAL PRIMARY KEY,
    Pais VARCHAR(50) NOT NULL,
   CodigoAlfa2 VARCHAR(5) NOT NULL,
   CodigoAlfa3 VARCHAR(5) NOT NULL,
    IdMoneda INTEGER NOT NULL,
    CONSTRAINT fkPais_IdMoneda FOREIGN KEY (IdMoneda) REFERENCES
Moneda(Id),
    Mapa bytea NULL,
    Bandera bytea NULL
);

--Definir indice unico de PAIS
CREATE UNIQUE INDEX ixPais_Pais
    ON Pais(Pais);
```

En estas instrucciones se pueden destacar:

- La instrucción CREATE DATABASE para crear la base de datos
- La instrucción CREATE TABLE para crear una tabla

- La instrucción CREATE INDEX para crear un índice a una tabla
- El Lenguaje de Manejo de Datos (en inglés DML, **Data Manipulation Language**) permite a los usuarios manejar o tener acceso a los datos que estén organizados por medio del modelo apropiado.

Este lenguaje es un subconjunto del lenguaje **SQL** que se tratará aquí de una manera no exhaustiva. Este lenguaje se introdujo como lenguaje de consulta del **Sistema R**. La sigla SQL significa **Structured Query Language** (*Lenguaje de Consulta Estructurado*).

Las siguientes son las instrucciones básicas que trataremos:

Consulta Básica de Selección

La estructura básica de una expresión de consulta en SQL se compone de tres cláusulas:

- La cláusula **Select** que corresponde a la operación de proyección del álgebra relacional. Sirve para listar los atributos que se desean en el resultado de una consulta.
- La cláusula **From** es una lista de las tablas que se van a examinar durante la ejecución de la expresión.
- La cláusula **Where** corresponde al predicado de selección del álgebra relacional. Se compone de un predicado que incluye atributos de las tablas que aparecen en la cláusula **From**.

Una consulta común en SQL tiene la forma:

```
Select <atributo 1>,<atributo 2>,... ,<atributo n>
from <tabla 1>,<tabla 2>,... ,<tabla n>
Where <predicado>
```

Esta consulta es equivalente a la expresión del álgebra relacional:

$$\prod_{\text{atributo1, atributo2, ..., atributon}} \left(\sigma_{\text{predicado}}(\text{tabla1} \times \text{tabla2} \times \text{tablan}) \right)$$

El SQL forma el producto cartesiano de las tablas que se nombran en la cláusula **From**; realiza una selección del álgebra relacional utilizando el predicado de la cláusula **Where** y proyecta el resultado a los atributos de la cláusula **Select**.

Si se omite la cláusula **where** el predicado <predicado> es verdadero.

La lista de atributos <atributo 1>,<atributo 2>,... ,<atributo n> puede sustituirse por un asterisco (*) para elegir todos los atributos de las tablas que aparecen en la cláusula **From**.

El resultado de una consulta en SQL es una tabla.

Para la eliminación de duplicados de tuplas, se inserta la palabra reservada **distinct** después de **Select** :

```
Select distinct <atributo 1>,<atributo 2>,... ,<atributo n>  
from <tabla 1>,<tabla 2>,... ,<tabla n>  
where <predicado>
```

El SQL incluye las operaciones **Union** (Unión), **Intersect** (Intersección) y **Minus** (Diferencia) que operan sobre las tablas.

El SQL utiliza los conectivos lógicos **and** (y), **or** (o) y **not** (no)

El siguiente ejemplo ilustra un ejemplo de consulta *Select* que permite obtener la lista de países que tienen la moneda 'Euro':

```
SELECT P.Pais , P.CodigoAlfa2  
FROM Pais P  
JOIN Moneda M ON P.IdMoneda = M.Id  
WHERE M.Moneda='Euro';
```

Obsérvese el uso de la cláusula **Join** que permite especificar la relación que hay entre 2 tablas

Cuyo resultado es el siguiente:



| | Pais | CodigoAlfa2 |
|---|--|-------------|
| ► | Andorra | AD |
| | Austria | AT |
| | Bélgica | BE |
| | Mayotte | YT |
| | Chipre | CY |
| | Dinamarca | DK |
| | Estonia | EE |
| | Finlandia | FI |
| | Åland | AX |
| | Francia | FR |
| | Guayana Francesa | GF |
| | Tierras Australes y Antárticas Francesas | TF |
| | Alemania | DE |
| | Grecia | GR |
| | Guadalupe | GP |
| | Vaticano, Ciudad del | VA |
| | Irlanda | IE |
| | Italia | IT |
| | Letonia | LV |
| | Lituania | LT |
| | Luxemburgo | LU |
| | Malta | MT |
| | Martinica | MQ |
| | Mónaco | MC |
| | Montenegro | ME |
| | Países Bajos | NL |
| | Portugal | PT |
| | Reunión | RE |
| | San Bartolomé | BL |
| | San Martín | MF |

SQL ofrece al usuario cierto control sobre el orden en que se van a mostrar las tuplas. La cláusula **Order by** permite que las tuplas en el resultado aparezcan ordenadas por determinados atributos.

Ejemplo:

Supóngase que la consulta anterior se requiere ordenada por el nombre del pais.

La expresión SQL sería:

```
SELECT P.Pais , P.CodigoAlfa2
FROM Pais P
JOIN Moneda M ON P.IdMoneda = M.Id
WHERE M.Moneda='Euro'
ORDER BY P.Pais;
```


Donde vemos que el resultado varía un poco:

| | Pais | CodigoAlfa2 |
|---|------------------|-------------|
| ► | Åland | AX |
| | Alemania | DE |
| | Andorra | AD |
| | Austria | AT |
| | Bélgica | BE |
| | Chipre | CY |
| | Dinamarca | DK |
| | Eslovaquia | SK |
| | Eslovenia | SI |
| | España | ES |
| | Estonia | EE |
| | Finlandia | FI |
| | Francia | FR |
| | Grecia | GR |
| | Guadalupe | GP |
| | Guayana Francesa | GF |
| | Irlanda | IE |
| | Italia | IT |
| | Letonia | LV |
| | Lituania | LT |
| | Luxemburgo | LU |
| | Malta | MT |
| | Martinica | MQ |
| | Mayotte | YT |
| | Mónaco | MC |
| | Montenegro | ME |
| | Países Bajos | NL |
| | Portugal | PT |
| | Reunión | RE |
| | San Bartolomé | BL |
| | San Marino | SM |

Actualización en SQL

Hasta este momento sólo se ha hecho énfasis en la extracción de información de la base de datos, pero no se ha mencionado ninguna operación que permita añadir, modificar o eliminar información, operaciones conocidas como de actualización.

1. Inserción

Para insertar datos en una tabla debe especificarse la tupla que se va a insertar o una consulta cuyo resultado sea el conjunto de tuplas que se insertará. Esto se denota:

```
Insert into <tabla>  
(<atributo 1>, < atributo 2>, ..., < atributo n>)  
values (<valor 1>, <valor 2>, ..., <valor n>)
```

ó

```
Insert into <tabla>  
Select ..
```

Ejemplo:

Para agregar el registro de la moneda 'Peso Colombiano' se tendría la siguiente instrucción:

```
INSERT INTO Moneda (Id, Sigla, Moneda)  
VALUES(35, 'COP', 'Peso colombiano');
```

2. Modificación

Permite cambiar los valores de los atributos de todas o algunas tuplas de una tabla. Se denota:

```
Update <tabla>  
set <atributo> = <expresion>  
where <predicado>
```

Ejemplo:

Supóngase que en la tabla de la consulta anterior se desea modificar el *Emisor* de la moneda, el cual no había sido insertado (razón por lo cual quedó con valor nulo):

```
UPDATE Moneda  
SET Emisor='Banco de la República'  
WHERE Id=35;
```

3. Eliminación

Funciona de manera similar a una consulta, sólo que el resultado es la desaparición de las tuplas elegidas:

```
Delete from <tabla>  
where <predicado>
```

Es de anotar que este comando, como los anteriores, sólo operan sobre una tabla.

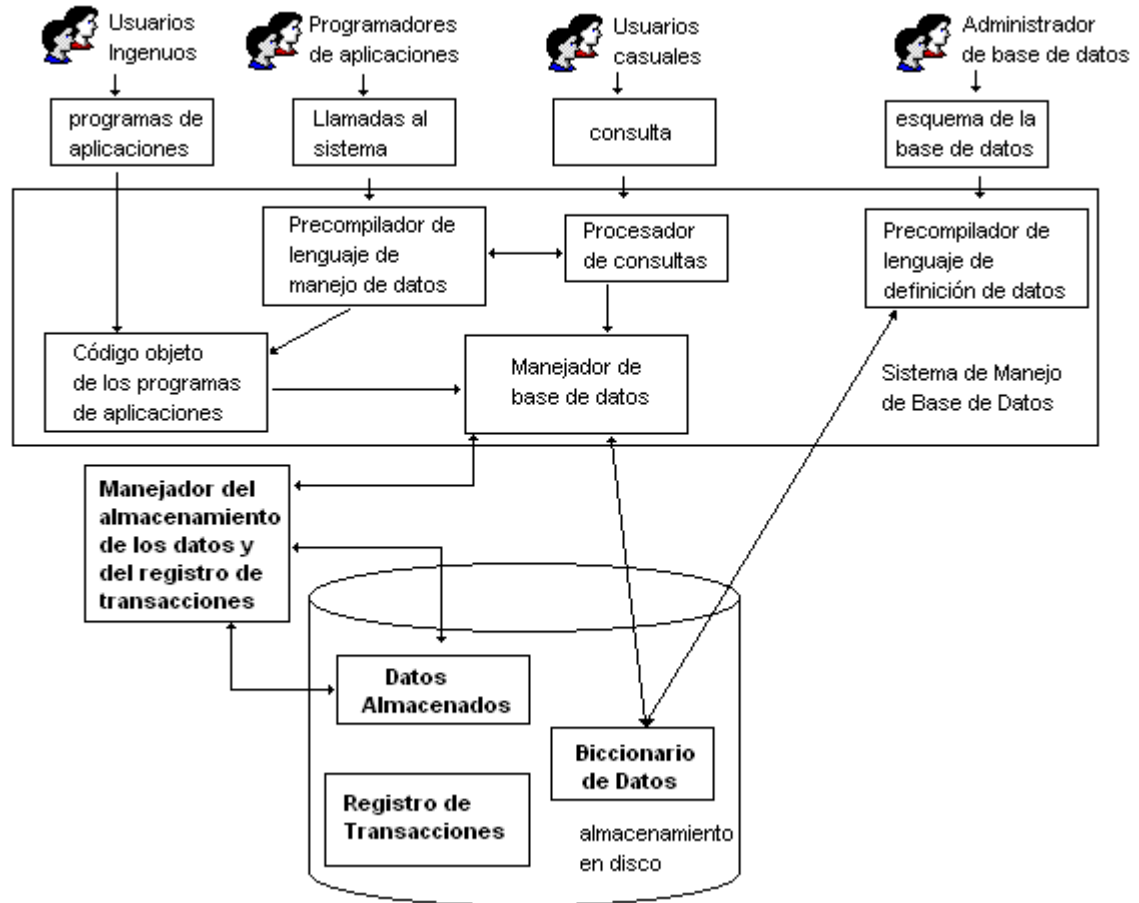
Ejemplo:

Para la tabla de la consulta anterior, eliminar la moneda con *Sigla*='COU'

```
DELETE FROM Moneda  
WHERE Sigla='COU';
```

- El **Manejador de Base de Datos** (En inglés ***DataBase Management System***) es un módulo de programa que constituye la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicaciones y las consultas hechas al sistema. Sus tareas básicas son:
 - Interacción con el manejador de archivos. Es quién se encarga realmente del almacenamiento, recuperación y actualización de los datos en la base de datos.
 - Implantación de la integridad. Realiza las operaciones que limitan la consistencia de los datos.
 - Puesta en práctica de la seguridad. Hace cumplir los requisitos de seguridad para tener acceso a los datos.
 - Respaldo y recuperación. Es responsable de detectar fallas y restaurar la base de datos al estado que existía antes presentarse ésta.
 - Control de Concurrencia. Vela por la consistencia de la información cuando varios usuarios actualizan a la vez la base de datos.

Un sistema de base de datos se divide en una serie de módulos que se encargan de cada una de las tareas del sistema general. El siguiente diagrama describe sus componentes básicos y cómo interactúan los diferentes tipos de usuarios:



La persona que ejerce un control centralizado sobre un DBMS se le conoce como el Administrador de Base de Datos (en inglés DBA, **Data Base Administrator**). Sus funciones principales son:

- Definición de Esquema. Es el creador del esquema original de la base de datos en un DDL.
- Modificación del esquema y de la organización física.
- Administración de la seguridad. Concede autorizaciones para el acceso a los datos a los distintos usuarios de la base de datos.
- Especificación de las limitantes de Consistencia.

¿Qué es PostgreSQL?

PostgreSQL, o simplemente *Postgres*, es un sistema de código abierto de administración de bases de datos del tipo relacional, aunque también es posible ejecutar consultas que sean no relacionales. En este sistema, las consultas relacionales se basan en *SQL*, mientras que las no relacionales hacen uso de *JSON*.

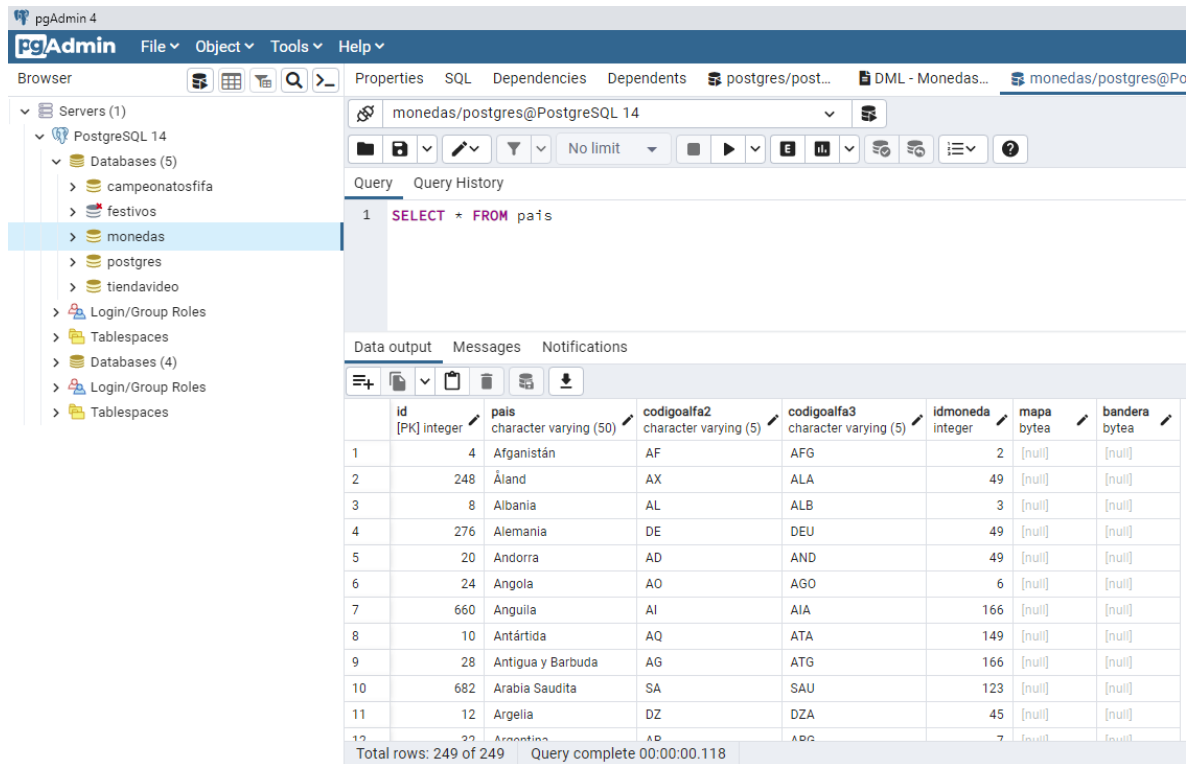
Además de ser un sistema de código abierto es gratuito y multiplataforma, y su desarrollo es llevado adelante por una gran comunidad de colaboradores de todo el mundo que día a día ponen su granito de arena para hacer de este sistema una de las opciones más sólidas a nivel de bases de datos.

Dos detalles a destacar de PostgreSQL es que posee **data types** (*tipos de datos*) avanzados y permite ejecutar optimizaciones de rendimiento avanzadas, que son características que por lo general solo se ven en sistemas de bases de datos comerciales, como por ejemplo SQL Server de Microsoft u Oracle de la compañía homónima.

Una característica extremadamente importante de *PostgreSQL* es su gran capacidad para el manejo de grandes volúmenes de datos, algo en lo que otros sistemas como *MySQL* aún no hacen tan bien. Las bases de datos de gran tamaño pueden hacer pleno uso del **MVCC** (**Multi-Version Concurrency Control**) de *PostgreSQL*, resultando en un gran rendimiento. MVCC es un método de control que permite realizar tareas de escritura y lectura simultáneamente.

Otro punto muy importante que no se debe dejar de lado es el cumplimiento de *ACID*. ¿Qué es ACID? Estas siglas en inglés refieren a: **atomicity, consistency, isolation y durability**, (atomicidad, consistencia, aislamiento y durabilidad) de las transacciones que se realizan en una base de datos. ¿Y por qué es tan importante? Porque tener soporte completo de *ACID* da la seguridad de que, si se produce una falla durante una transacción, los datos no se perderán ni terminarán donde no deban.

La administración de *PostgreSQL* se vuelve muy sencilla por medio de aplicativos como **PgAdmin**, que básicamente viene a ser un *phpMyAdmin* orientado para *PostgreSQL*. La posibilidad de realizar diversos procedimientos en forma sencilla hace que *PgAdmin* sea ampliamente utilizado, aunque también permite realizar tareas más complejas, así que tanto novatos como usuarios expertos hacen uso de él.

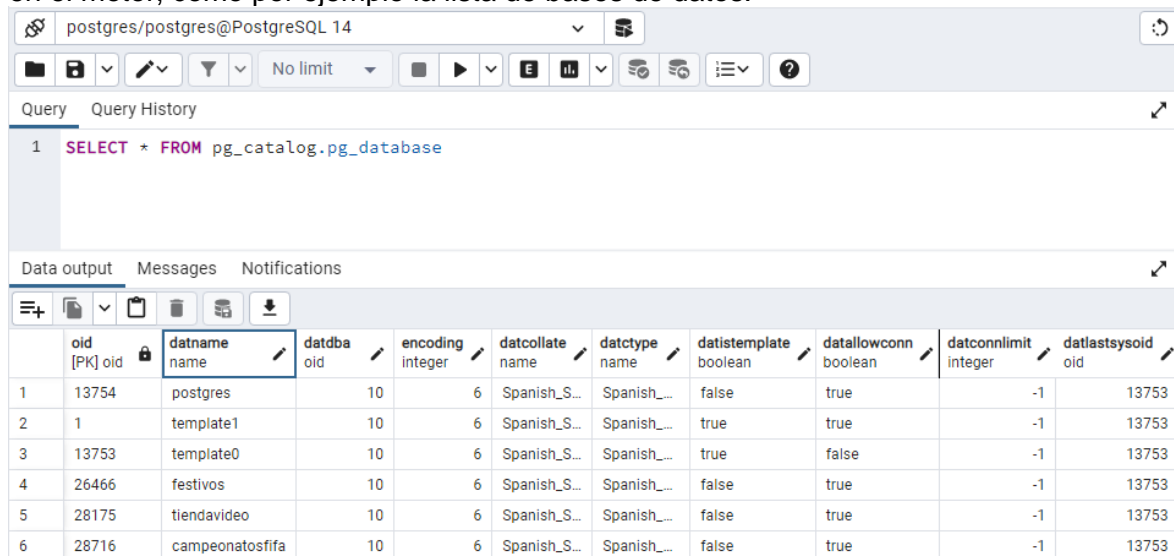


The screenshot shows the pgAdmin 4 interface with a query executed: `SELECT * FROM pais`. The result is displayed in a table with the following columns: `id` (PK Integer), `pais` (character varying (50)), `codigoalfa2` (character varying (5)), `codigoalfa3` (character varying (5)), `idmoneda` (integer), `mapa` (bytea), and `bandera` (bytea). The table contains 12 rows of data representing different countries and their associated codes and flags.

| id | pais | codigoalfa2 | codigoalfa3 | idmoneda | mapa | bandera |
|----|-------------------|-------------|-------------|----------|--------|---------|
| 1 | Afganistán | AF | AFG | 2 | [null] | [null] |
| 2 | Áland | AX | ALA | 49 | [null] | [null] |
| 3 | Albania | AL | ALB | 3 | [null] | [null] |
| 4 | Alemania | DE | DEU | 49 | [null] | [null] |
| 5 | Andorra | AD | AND | 49 | [null] | [null] |
| 6 | Angola | AO | AGO | 6 | [null] | [null] |
| 7 | Anguila | AI | AIA | 166 | [null] | [null] |
| 8 | Antártida | AQ | ATA | 149 | [null] | [null] |
| 9 | Antigua y Barbuda | AG | ATG | 166 | [null] | [null] |
| 10 | Arabia Saudita | SA | SAU | 123 | [null] | [null] |
| 11 | Argelia | DZ | DZA | 45 | [null] | [null] |
| 12 | Argentina | AR | ARG | 7 | [null] | [null] |

Total rows: 249 of 249 Query complete 00:00:00.118

Es posible consultar en la base de datos del sistema, los diferentes objetos que se creen en el motor, como por ejemplo la lista de bases de datos:



The screenshot shows the pgAdmin 4 interface with a query executed: `SELECT * FROM pg_catalog.pg_database`. The result is displayed in a table with the following columns: `oid` (PK oid), `datname` (name), `datdba` (oid), `encoding` (integer), `datcollate` (name), `datctype` (name), `datistemplate` (boolean), `datallowconn` (boolean), `datconnlimit` (integer), and `datlastsysoid` (oid). The table contains 6 rows of data representing different databases in the system.

| oid | datname | datdba | encoding | datcollate | datctype | datistemplate | datallowconn | datconnlimit | datlastsysoid |
|-------|-----------------|--------|----------|--------------|-------------|---------------|--------------|--------------|---------------|
| 13754 | postgres | 10 | 6 | Spanish_S... | Spanish_... | false | true | -1 | 13753 |
| 1 | template1 | 10 | 6 | Spanish_S... | Spanish_... | true | true | -1 | 13753 |
| 13753 | template0 | 10 | 6 | Spanish_S... | Spanish_... | true | false | -1 | 13753 |
| 26466 | festivos | 10 | 6 | Spanish_S... | Spanish_... | false | true | -1 | 13753 |
| 28175 | tiendavideo | 10 | 6 | Spanish_S... | Spanish_... | false | true | -1 | 13753 |
| 28716 | campeonatosfifa | 10 | 6 | Spanish_S... | Spanish_... | false | true | -1 | 13753 |

Cada vez que se cree o cambie la estructura de una tabla, es posible verificar su nueva estructura mediante la siguiente consulta:

monedas/postgres@PostgreSQL 14

Query Query History

```

1 SELECT column_name, data_type
2 FROM information_schema.columns
3 WHERE table_name = 'pais'

```

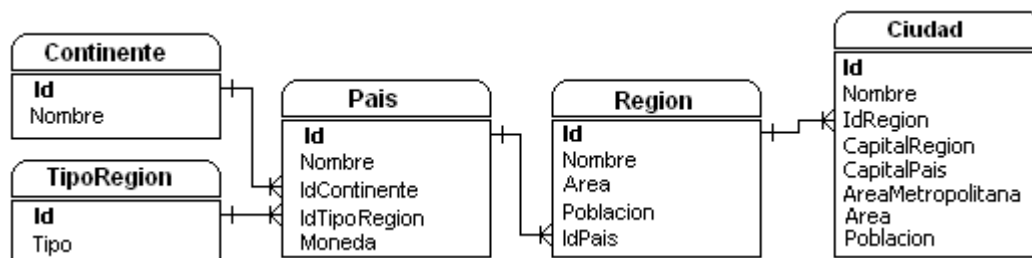
Data output Messages Notifications

| | column_name name | data_type character varying |
|---|---------------------|--------------------------------|
| 1 | idmoneda | integer |
| 2 | mapa | bytea |
| 3 | bandera | bytea |
| 4 | id | integer |
| 5 | codigoalfa3 | character varying |
| 6 | pais | character varying |
| 7 | codigoalfa2 | character varying |

Ejercicio de repaso:

Basado en cada uno de los siguientes modelos relacionales, escribir las instrucciones DDL necesarias para implementar el modelo completo en el motor de base de datos PostgreSQL.

- Base de datos para registro de división política mundial:



R/

| Acción | Instrucción SQL (DDL) |
|----------------------------------|---|
| Crear la tabla Continente | <pre> CREATE TABLE Continente(Id SERIAL PRIMARY KEY, Nombre VARCHAR(50) NOT NULL); </pre> |

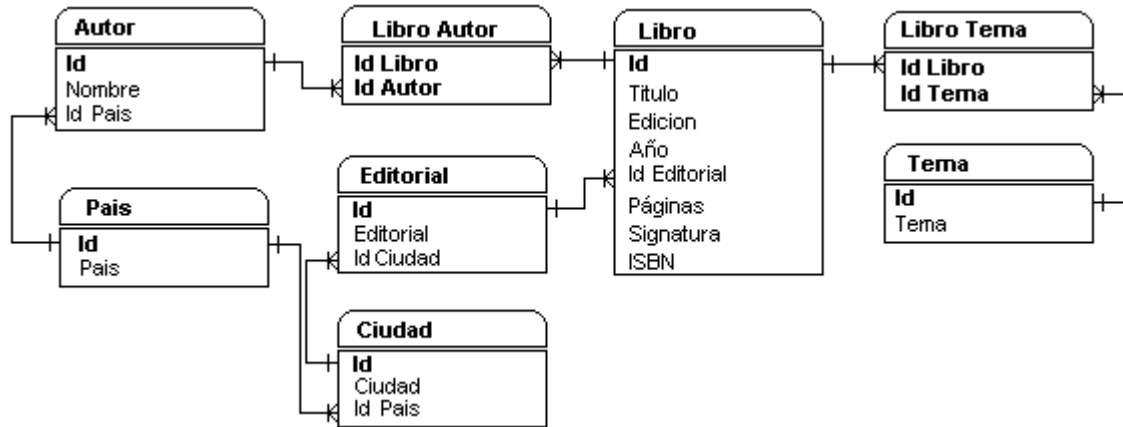


| | |
|--|--|
| Crear índice para el nombre del Continente | <pre>CREATE UNIQUE INDEX ixContinente_Nombre ON Continente(Nombre);</pre> |
| Crear la tabla TipoRegion | <pre>CREATE TABLE TipoRegion(Id SERIAL PRIMARY KEY, Tipo VARCHAR(50) NOT NULL);</pre> |
| Crear índice para el tipo del TipoRegion | <pre>CREATE UNIQUE INDEX ixTipoRegion_Tipo ON TipoRegion(Tipo);</pre> |
| Crear la tabla Pais | <pre>CREATE TABLE Pais(Id SERIAL PRIMARY KEY, Nombre VARCHAR(50) NOT NULL, IdContinente INTEGER NOT NULL, CONSTRAINT fkPais_IdContinente FOREIGN KEY (IdContinente) REFERENCES Continente(Id), IdTipoRegion INTEGER NOT NULL, CONSTRAINT fkPais_IdTipoRegion FOREIGN KEY (IdTipoRegion) REFERENCES TipoRegion(Id), Moneda VARCHAR(30) NULL);</pre> |
| Crear índice para el nombre del Pais | <pre>CREATE UNIQUE INDEX ixPais_Nombre ON Pais(Nombre);</pre> |
| Crear la tabla Region | <pre>CREATE TABLE Region(Id SERIAL PRIMARY KEY, Nombre VARCHAR(50) NOT NULL, IdPais INTEGER NOT NULL, CONSTRAINT fkRegion_IdPais FOREIGN KEY (IdPais) REFERENCES Pais(Id), Area FLOAT NULL, Poblacion INTEGER NULL);</pre> |
| Crear índice único del nombre de la Region por país (el nombre de la región sólo se puede repetir en diferentes países) | <pre>CREATE UNIQUE INDEX ixRegion_IdPais_Nombre ON Region(IdPais,Nombre);</pre> |
| Crear la tabla Ciudad | <pre>CREATE TABLE Ciudad(Id SERIAL PRIMARY KEY, Nombre VARCHAR(50) NOT NULL, IdRegion INTEGER NOT NULL, CONSTRAINT fkCiudad_IdRegion FOREIGN KEY (IdRegion)</pre> |



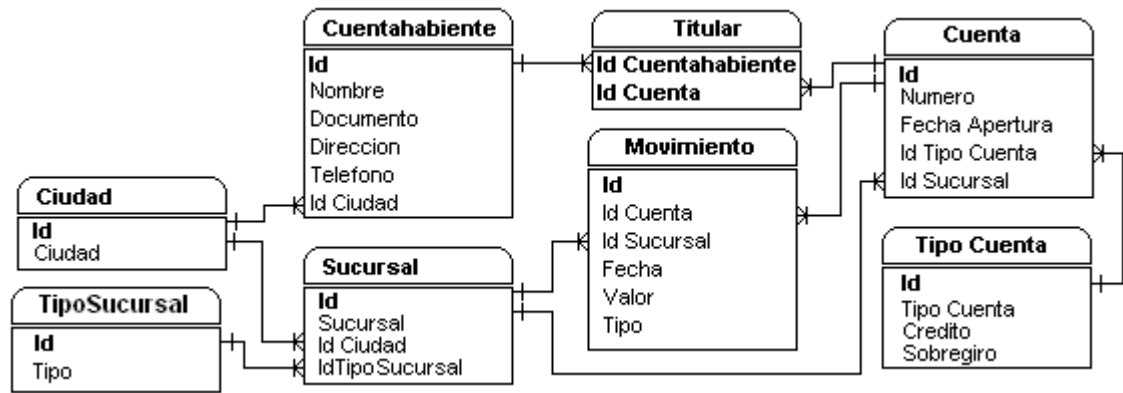
| | |
|--|--|
| | <pre>REFERENCES Region(Id), Area FLOAT NULL, Poblacion INTEGER NULL, CapitalPais BOOLEAN DEFAULT false NOT NULL, CapitalRegion BOOLEAN DEFAULT false NOT NULL, AreaMetropolitana BOOLEAN DEFAULT false NOT NULL);</pre> |
| Crear índice único del nombre de la Ciudad por región (el nombre de la ciudad sólo se puede repetir en diferentes regiones) | <pre>CREATE UNIQUE INDEX ixCiudad_IdRegion_Nombre ON Ciudad(IdRegion,Nombre);</pre> |

2. Base de datos para registro de bibliografía:



Estime los índices que considere necesarios

3. Base de datos para registro de cuentas bancarias:



4. Basado en el modelo relacional de división política mundial, escribir las instrucciones SQL necesarias para realizar las consultas que se enuncian, en el motor de base de datos *PostgreSQL*
- ¿Cuántas divisiones políticas tiene un país y como se le denomina a la división?
 - ¿Cuál es el área y población de los países del continente “América”?
 - ¿Cuál es la capital del país “Argentina”?
 - ¿Dónde está ubicada la ciudad con nombre “Barbosa”?
 - ¿De qué región es capital la ciudad de “Hermosillo”?
 - ¿Cuál es la región de “Brasil” con mayor población?
 - ¿Cuál es la región de “México” con menor área?
 - ¿De qué país es capital “Bogotá”? (El nombre completo de la ciudad es “SANTA FE DE BOGOTA D.C.”)

En las expresiones de consulta, es bueno tener en cuenta la siguiente lista de funciones:

| Tipo | Función | Descripción |
|-------|---------------------------|--|
| Texto | upper(s), lower(s) | Convierte una variable de texto s a mayúscula o minúscula |
| | ltrim(s), rtrim(s) | Quita los espacios en blanco a la izquierda o la derecha del texto s |
| | substr(s,start,l) | Extrae un texto de longitud l desde el texto s a partir de la posición p |
| | position(s2 in s1) | Devuelve la posición de la cadena s2 en la cadena s1 |
| | length(s) | Longitud del texto s |

| | | |
|------------|---|--|
| Fecha | now() | Fecha actual del servidor |
| Numérica | round(x,p) | Redondea el número real x según la precisión p |
| | abs(x) | Valor absoluto de x |
| | random() | Genera un numero aleatorio |
| Conversión | cast (expresion AS tipo_de_dato) | Convierte a tipo_de_dato una expresion |

Y lo siguientes operadores

| Operador | Descripción |
|----------|-------------------------|
| + - / * | Aritmético |
| = | Igualdad |
| <> | Diferente |
| | Concatenación de textos |

R/

- a. Incluyendo todos los países que no tienen regiones registradas:

```
SELECT P.Nombre, COUNT(R.Id), T.Tipo
  FROM Region R RIGHT JOIN Pais P ON R.IdPais = P.ID
 INNER JOIN TipoRegion T ON P.IdTipoRegion = T.Id
 GROUP BY P.Nombre, T.Tipo
 ORDER BY P.Nombre
```

Incluyendo sólo los países que tienen regiones registradas:

```
SELECT P.Nombre, COUNT(R.Id), T.Tipo
  FROM Region R INNER JOIN Pais P ON R.IdPais = P.ID
 INNER JOIN TipoRegion T ON P.IdTipoRegion = T.Id
 GROUP BY P.Nombre, T.Tipo
 ORDER BY P.Nombre
```

- b.

```
SELECT P.Nombre, SUM(R.Area) AS TotalArea,
SUM(R.Poblacion) AS TotalPoblacion
  FROM Region R INNER JOIN Pais P ON R.IdPais = P.ID
 INNER JOIN Continente C ON P.IdContinente = C.Id
 WHERE UPPER(C.Nombre)='AMERICA'
 GROUP BY P.Nombre
 ORDER BY P.Nombre
```

c.

```
SELECT C.Nombre
      FROM Pais P INNER JOIN Region R ON P.Id = R.IdPais
INNER JOIN Ciudad C ON R.Id = C.IdRegion
      WHERE LOWER(P.Nombre)='argentina'
      AND C.CapitalPais=true
```

d.

```
SELECT R.Nombre || ' (' || P.Nombre || ')' AS Ubicacion
      FROM Pais P INNER JOIN Region R ON P.Id = R.IdPais
INNER JOIN Ciudad C ON R.Id = C.IdRegion
      WHERE LOWER(C.Nombre)='barbosa'
      ORDER BY R.Nombre, P.Nombre
```

e.

```
SELECT R.Nombre
      FROM Region R
      JOIN Ciudad C
      ON R.Id=C.IdRegion
      WHERE C.Nombre='Hermosillo' AND
      C.CapitalRegion=true
```

f.

```
SELECT Region.nombre, Region.Poblacion
      FROM Region
      WHERE Poblacion IN
      (SELECT MAX(Region.Poblacion)
      FROM Pais INNER JOIN Region ON
      Pais.Id=Region.IdPais
      WHERE Pais.Nombre='BRASIL')
```

g.

```
SELECT Region.Nombre, Region.Area
      FROM Pais INNER JOIN Region ON Pais.Id=Region.IdPais
      WHERE Pais.Nombre='MEXICO'
      AND Area IN
      (SELECT MIN(Region.Area)
      FROM Pais INNER JOIN Region ON
      Pais.Id=Region.IdPais
      WHERE Pais.Nombre='MEXICO')
```

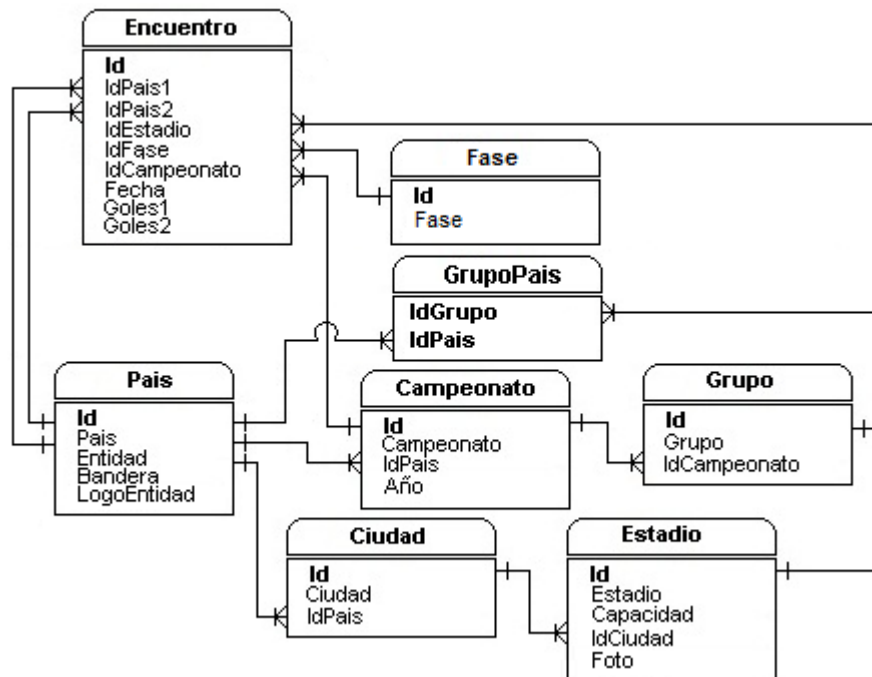
h.

```
SELECT P.Nombre
```

```

FROM Pais P INNER JOIN Region R ON P.Id = R.IdPais
INNER JOIN Ciudad C ON R.Id = C.IdRegion
WHERE POSITION('BOGOTA' IN UPPER(c.Nombre))>0
AND C.CapitalPais=true
  
```

5. Basado en el siguiente modelo relacional para registro de campeonatos de fútbol de selecciones:



escribir las instrucciones SQL necesarias para realizar las consultas que se enuncian, en el motor de base de datos *PostgreSQL*:

- Puntos obtenidos por un equipo en un grupo
- ¿Equipo que más participaciones ha tenido en campeonatos?
- ¿Equipo que más campeonatos ha ganado?
Tenga en cuenta que una de las fases es la FINAL
- Ciudades que tengan más de 1 estadio
- Campeonato en que se haya marcado más goles

R/

- ```

SELECT (SELECT Pais FROM Pais WHERE Id=@IdPais),
 (SELECT Sum(CASE WHEN (IdPais1=@IdPais AND Goles1>Goles2)
OR (IdPais2=@IdPais AND Goles1<Goles2)
THEN 1 ELSE 0 END)
FROM Encuentro INNER JOIN Grupo

```

```

 ON Encuentro.IdCampeonato=Grupo.IdCampeonato
WHERE (IdPais1=@IdPais OR IdPais2=@IdPais)
AND IdFase=1 AND Grupo.Id=@IdGrupo
AND Goles1 IS NOT NULL
AND Goles2 IS NOT NULL)*3+
(SELECT Sum(CASE WHEN (IdPais1=@IdPais AND Goles1=Goles2)
OR (IdPais2=@IdPais AND Goles1=Goles2)
THEN 1 ELSE 0 END)
FROM Encuentro INNER JOIN Grupo
 ON Encuentro.IdCampeonato=Grupo.IdCampeonato
WHERE (IdPais1=@IdPais OR IdPais2=@IdPais)
AND IdFase=1 AND Grupo.Id=@IdGrupo
AND Goles1 IS NOT NULL
AND Goles2 IS NOT NULL) AS Puntos

```

- b. ¿Equipo que más participaciones ha tenido en campeonatos?

```

SELECT TOP 1 Pais, COUNT(IdPais) AS Participaciones
FROM Pais INNER JOIN GrupoPais
 ON Pais.Id=GrupoPais.IdPais
GROUP BY Pais
ORDER BY Participaciones DESC

```

- c. ¿Equipo que más campeonatos ha ganado?

*Tenga en cuenta que una de las fases es la FINAL*

```

SELECT TOP 1 Pais, ((SELECT COUNT(IdPais1)
FROM Encuentro
WHERE IdPais1=P.Id AND Goles1>Goles2
AND IdFase=5
AND Goles1 IS NOT NULL
AND Goles2 IS NOT NULL)+
(SELECT COUNT(IdPais2)
FROM Encuentro
WHERE IdPais2=P.Id AND Goles2>Goles1
AND IdFase=5
AND Goles1 IS NOT NULL
AND Goles2 IS NOT NULL)) AS PG
FROM Pais P
WHERE Id IN (SELECT IdPais1 AS IdPais
FROM Encuentro
WHERE IdFase=5
UNION SELECT IdPais2 AS IdPais
FROM Encuentro
WHERE IdFase=5)
ORDER BY PG DESC

```

Se asume que la fase “*Final*” tiene Id=5

- d. Ciudades que tengan más de 1 estadio

```

SELECT Ciudad
 FROM Ciudad
 WHERE Id IN (SELECT IdCiudad
 FROM Estadio
 GROUP BY IdCiudad
 HAVING Count(*) > 1)

```

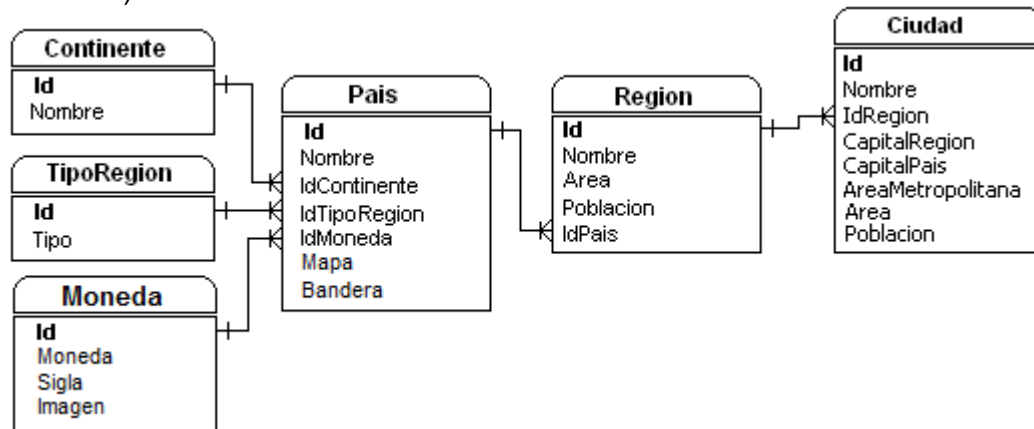
- e. Campeonato en que se haya marcado más goles

```

SELECT TOP 1 Campeonato, SUM(Goles1)+SUM(Goles2) AS Goles
 FROM Encuentro INNER JOIN Campeonato
 ON Encuentro.IdCampeonato=Campeonato.Id
 GROUP BY Campeonato
 ORDER BY Goles DESC

```

6. Basado en el modelo relacional de división política mundial, se desea realizar los siguientes cambios en el modelo relacional inicial (sin pérdida de la información existente):



Se debe escribir por tanto un script que haga lo siguiente:

- Desnormalización en la tabla **Pais** de la información de la **Moneda**
- Agregar campos para imágenes del Mapa y la Bandera del País