Proyecto Final Administración de Bases de Datos

Alejandro Pérez Álvarez - alu0101215310

Daniel Hernández Fajardo - alu0101320489

Índice

1. Título	3
2. Objetivos del proyecto	3
3. Contexto de la base de datos	3
4. Modelo entidad-relación	4
5. Grafo relacional	6
7. BDD SQL, checks y disparadores	8
8. Consultas de prueba	10
9.API REST	14

1. Título

Space Database

2. Objetivos del proyecto

El objetivo general del proyecto es desarrollar una base de datos capaz de almacenar aquellos elementos que podemos encontrar en nuestro universo, incluyendo sus diferentes atributos y las relaciones entre ambos, permitiendo así un fácil acceso a este tipo de información para que se pueda utilizar de manera educativa para niños y estudiantes.

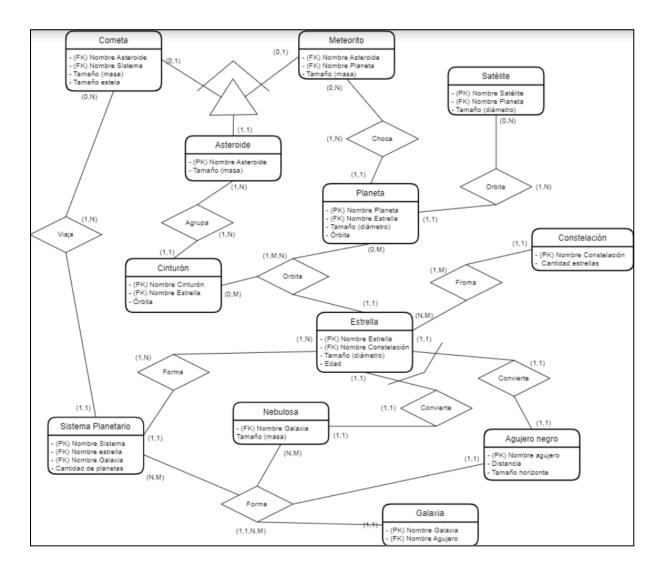
3. Contexto de la base de datos

El concepto elegido para la base de datos ha sido el espacio. Nos hemos decantado por él debido a la gran cantidad de elementos diferentes que podemos encontrar en él, su gran versatilidad, y debido que a pesar de no ser algo de un interés general ni de gran repercusión, supone un tema de gran importancia para aquellas personas que de verdad están interesadas en él.

Respecto a los requisitos del proyecto, el mismo abarca todos los modelos de datos datos en clase: podemos encontrar entidades débiles como lo es la entidad Satélite, ya que depende del su planeta asociado para considerarse un satélite; encontramos relaciones triples como puede ser la relación de orbitar alrededor de una estrella, la cual relaciona a las entidades Planeta, Cinturón y Estrella; contiene tipos ISA, siendo un ejemplo la relación entre las entidades Asteroide, Meteorito y Cometa, ya que los dos últimos son asteroides en sí; consta de diferentes tipos de cardinalidades en las relaciones, tanto 1:1 y 1:N, como en las relaciones con la entidad Estrella, como N:M en las entidades relacionadas con la entidad Galaxia; y por último encontramos también casos de exclusión como

puede ser el de conversión de un estrella, ya que esta misma solo podrá convertirse en un agujero negro o en nebulosa, pero no en ambas a la vez.

4. Modelo entidad-relación



En lo que respecta al diseño del entidad-relación que se ha llevado a cabo podemos apreciar una gran conectividad entre todas las entidades.

En dicho modelo encontramos que el centro neurálgico es la estrella, la cual puede ser orbitada por ninguno o por varios planetas, o por ninguno o varios cinturones de asteroides, mientras que estos solo pueden orbitar alrededor de una estrella.

A su vez, una estrella forma parte de una única constelación, y las constelaciones se componen de varias estrellas.

Los cinturones de asteroides están formados al menos por un asteroide, y este solo puede pertenecer a un cinturón de asteroides. Por otro lado, un asteroide puede convertirse o bien en un cometa, o bien en un meteorito. Estos últimos se estrellan en un único planeta, el cual puede recibir o no varios meteoritos.

Por otra parte, un planeta puede o no tener algunos satélites orbitando a su alrededor, mientras que un satélite sólo puede pertenecer a un planeta.

En lo que respecta a los cometas, estos viajan por un sistema planetario, mientras que este último puede o no tener algunos cometas. De igual forma, un sistema planetario puede estar formado por una o varias estrellas y pertenece a una galaxia.

Las estrellas a su vez, una vez muertas, se convierten o bien en un agujero negro, o bien en una nebulosa las cuales forman parte de una galaxia.

Por último, las galaxias están formadas por un agujero negro en su núcleo, por varias nebulosas a lo largo de la misma, y por varios sistemas planetarios de igual manera.

5. Grafo relacional

constelación (nombre_constelacion, cantidad_estrellas)

estrella(<u>nombre_estrella</u>, <u>nombre_constelación</u>, tamaño, edad)

planeta(nombre_planeta, nombre_estrella,tamaño,orbita, nombre_sistema)

satelite(nombre_satelite, nombre_planeta, tamaño)

agujer_negro(nombre_aqujero, distancia, tamaño_horizonte)

galaxia (nombre_galaxia, nombre_agujero)

nebulosa(<u>nombre_qalaxia</u>, tamaño)

sistema_planetario(nombre_sistema, nombre_estrella, nombre_galaxia, cantidad_planetas)

cinturon(nombre_cinturon, nombre_estrella, orbita)

asteroide(nombre_asteroide, tamaño)

cometa(nombre_asteroide, nombre_sistema, tamaño_estela, tamaño)

meteorito(nombre_asteroide, nombre_planeta, tamaño)

constelación(nombre_constelacion, cantidad_estrellas)

- Como clave primaria encontramos el atributo nombre_contelación.

estrella(<u>nombre_estrella</u>, <u>nombre_constelación</u>, tamaño, edad)

- Como clave primaria encontramos el atributo nombre_estrella.
- Como clave ajena tenemos el atributo nombre_constelación, el cual deberá existir dentro de la tabla constelación o la entrada de la tabla será eliminada debido a la restricción impuesta.

planeta(<u>nombre_planeta</u>, <u>nombre_estrella</u>,tamaño,orbita, nombre_sistema)

- Como clave primaria encontramos el atributo nombre_planeta.
- Como clave ajena tenemos el atributo nombre_estrella, el cual deberá existir dentro de la tabla estrella o la entrada de la tabla será eliminada debido a la restricción impuesta.

satelite(<u>nombre_satelite</u>, <u>nombre_planeta</u>, tamaño)

- Como clave primaria encontramos el atributo nombre_satelite.
- Como clave ajena tenemos el atributo nombre_planeta, el cual deberá existir dentro de la tabla planeta o la entrada de la tabla será eliminada debido a la restricción impuesta.

agujer_negro(nombre_agujero, distancia, tamaño_horizonte)

- Como clave primaria encontramos el atributo nombre_agujero.

galaxia(nombre_galaxia, nombre_agujero)

- Como clave primaria encontramos el atributo nombre_galaxia.
- Como clave ajena tenemos el atributo nombre_agujero, el cual deberá existir dentro de la tabla agujero_negro o la entrada de la tabla será eliminada debido a la restricción impuesta.

nebulosa(<u>nombre_galaxia</u>, tamaño)

- Como clave ajena tenemos el atributo nombre_galaxia, el cual deberá existir dentro de la tabla galaxia o la entrada de la tabla será eliminada debido a la restricción impuesta.

sistema_planetario(<u>nombre_sistema</u>, <u>nombre_estrella</u>, <u>nombre_galaxia</u>, cantidad_planetas)

- Como clave primaria encontramos el atributo nombre_sistema.
- Como claves ajenas tenemos los atributos nombre_estrella y nombre_galaxia, los cuales deberán existir dentro de las tablas nombre_estrella y nombre_galaxia respectivamente o la entrada de la tabla será eliminada debido a la restricción impuesta.

cinturon(<u>nombre_cinturon</u>, <u>nombre_estrella</u>, orbita)

- Como clave primaria encontramos el atributo nombre_cinturon.
- Como clave ajena tenemos el atributo nombre_estrella, el cual deberá existir dentro de la tabla estrella o la entrada de la tabla será eliminada debido a la restricción impuesta.

asteroide(nombre_asteroide, tamaño)

- Como clave primaria encontramos el atributo nombre_asteroide.

cometa(<u>nombre_asteroide</u>, <u>nombre_sistema</u>, tamaño_estela, tamaño)

- Como clave primaria encontramos el atributo nombre_asteroide.
- Como clave ajena tenemos el atributo nombre_sistema, el cual deberá existir dentro de la tabla sistema_planetario o la entrada de la tabla será eliminada debido a la restricción impuesta.

meteorito(<u>nombre_asteroide</u>, <u>nombre_planeta</u>, tamaño)

- Como clave primaria encontramos el atributo nombre_asteroide.
- Como clave ajena tenemos el atributo nombre_planeta, el cual deberá existir dentro de la tabla planeta o la entrada de la tabla será eliminada debido a la restricción impuesta.

7. BDD SQL, checks y disparadores

Para la base de datos hemos creado varios ficheros sql, en el primero se realiza la creación de todas las tablas, asignándole los tipos correspondientes a cada valor y las restricciones de claves. Tras ello se tendrá que ejecutar el fichero que contiene los checks junto con una carga de datos de prueba. Y por último existe otro fichero el cual contiene el código de los disparadores.

Se han creado check para comprobar que las claves primarias de las tablas no sean nulas, además de para los atributos de las tablas como tamaño, edad y además, se ha creado checks que comprueben que estos valores no sean menores que 0. A continuación algunos ejemplos:

```
ALTER TABLE constelacion
ADD CONSTRAINT constelacion_not_null
CHECK (nombre_constelacion IS NOT NULL);

ALTER TABLE constelacion
ADD CONSTRAINT constelacion_cantidad_estrellas_negativo
CHECK (cantidad_estrellas > 0);

ALTER TABLE estrella
ADD CONSTRAINT estrella_not_null
CHECK (nombre_estrella IS NOT NULL);

ALTER TABLE estrella
ADD CONSTRAINT estrella_tamaño_negativo
CHECK (tamaño > 0);

ALTER TABLE estrella
ADD CONSTRAINT estrella_edad_negativo
CHECK (edad > 0);
```

En cuanto a los disparadores, se han creado cuatro de ellos. Dos de ellos realizan una actualización de la cantidad de planetas en el caso de un sistema planetario y de estrellas en caso de una constelación en caso de inserción de alguna estrella o planeta.

```
CREATE FUNCTION update constelacion estrellas() RETURNS TRIGGER AS $$
   UPDATE constelacion SET cantidad_estrellas = (SELECT COUNT(*) FROM estrella WHERE nombre_constelacion = NEW.nombre_constelacion)
    WHERE nombre constelacion = NEW.nombre constelacion;
   RETURN NEW;
$$ LANGUAGE plpgsql;
CREATE TRIGGER update_estrellas_constelacion
AFTER INSERT ON estrella
FOR EACH ROW
EXECUTE FUNCTION update_constelacion_estrellas();
CREATE FUNCTION update_sistema_planetario_planetas() RETURNS TRIGGER AS $$
   UPDATE sistema_planetario SET cantidad_planetas = (SELECT COUNT(*) FROM planeta WHERE nombre_sistema = NEW.nombre_sistema)
    WHERE nombre_sistema = NEW.nombre_sistema;
   RETURN NEW;
$$ LANGUAGE plpgsql;
CREATE TRIGGER update_planetas_sistema_planetario
AFTER INSERT ON planeta
FOR EACH ROW
                  undate sistema planetario planetas()
```

Otro de los disparadores guarda en una tabla cada vez que se inserta un meteorito, asignándole la fecha y un valor de peligrosidad en base al tamaño, si su destino es La Tierra.

```
CREATE FUNCTION insert_meteorito_tierra() RETURNS TRIGGER AS $$
BEGIN

IF (IG_OP = 'INSERT') THEN

IF (NEW.nombre_planeta = 'La Tierra') THEN

CASE

WHEN NEW.tamaño <= 3 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 1);
WHEN NEW.tamaño <= 9 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 2);
WHEN NEW.tamaño <= 9 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 3);
WHEN NEW.tamaño <= 15 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 5);
WHEN NEW.tamaño <= 15 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 5);
WHEN NEW.tamaño <= 25 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 5);
WHEN NEW.tamaño <= 25 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 7);
WHEN NEW.tamaño <= 25 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 7);
WHEN NEW.tamaño <= 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 8);
WHEN NEW.tamaño <= 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 8);
WHEN NEW.tamaño <= 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 8);
WHEN NEW.tamaño <= 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 8);
WHEN NEW.tamaño <= 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 8);
WHEN NEW.tamaño <= 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (N
```

El último disparador se activa cuando se inserta un nuevo cometa y almacena en una tabla su nombre y fecha en la que se ha detectado el cometa.

```
CREATE FUNCTION insert_cometa_detectado() RETURNS TRIGGER AS $$
BEGIN

    IF (TG_OP = 'INSERT') THEN
        INSERT INTO cometa_detectados (nombre_cometa, fecha) VALUES (NEW.nombre_asteroide, NOW());
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insert_cometa_detectado
AFTER INSERT ON cometa
FOR EACH ROW
EXECUTE FUNCTION insert_cometa_detectado();
```

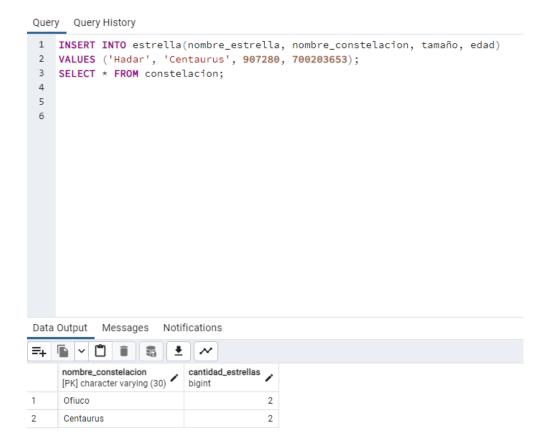
8. Consultas de prueba

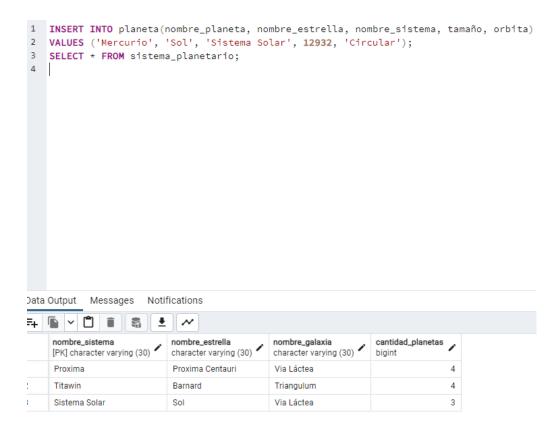
Para probar los disparadores, check y restricciones se ha realizado varias consultas:

Creamos un nuevo asteroide el cual también va a ser un meteorito y al realizar la inserción se activa el disparador y como el planeta de destino es La Tierra se guarda en una tabla junto a la fecha y peligrosidad.



A continuación probamos los disparadores que actualizan el número de estrellas y planetas de las constelaciones y de los sistemas planetarios. En este caso había únicamente una estrella perteneciente a 'Centaurus' y se ha añadido otra. Lo mismo sucede con el 'Sistema Solar', el cual tenía 2 planetas y se le ha añadido uno nuevo.





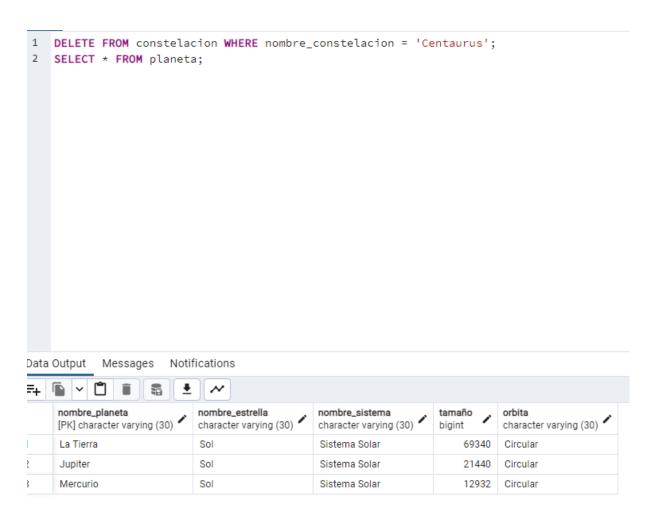
La siguiente prueba es para comprobar el disparador que detecta cometas. Podemos ver que se ha creado un nuevo asteroide el cual es un cometa y al ejecutar se inserta en la tabla cometa_detectado en nombre y la fecha.



A continuación es una prueba de los check en los cuales no se permiten valores negativos.



Por último se comprueban las restricciones de las claves, de forma que al eliminar una constelación, se eliminan todas las estrellas pertenecientes a esta y a si vez los planetas que orbitan esas estrellas.



9.API REST

Para la API REST de nuestra base de datos hemos implementado operaciones sobre unas de las tablas más relevantes de la base de datos. Estas son constelación, estrella y planeta.

Al acceder al despliegue local, nos encontramos con la página principal, en la cual se nos muestra el contenido de dichas tablas. Esto se realiza con un SELECT de cada una de las tablas almacenando los valores de respuesta.

Constelaciones, Estrellas y Planetas Operaciones con la BDD

Constelaciones

#Ofiuco

Numero de estrellas: 6

#Centaurus

Numero de estrellas: 9

Estrellas

#Sol

Constelación: Ofiuca

Radio(km): 696340

Edad: 4000000603

#Proxima Centaur

Constelación: Centaurus

Radio(km): 107280

#Barnard

Radio(km): 136360

Edad: 10000000001

Planetas

#La Tierra

Fetvalla aug arhita: Sal

En cuanto a las demás operaciones, se debe acceder al apartado de operaciones CRUD donde hay enlaces para cada operación de cada tabla.

Constelaciones, Estrell

Inserciones

Insertar constelación

Insertar estrella

<u>Insertar planeta</u>

Extracciones

Eliminar constelación

Eliminar estrella

Eliminar planeta

Modificaciones

Actualizar estrella

Actualizar planeta

A continuación se mostrarán algunas imágenes de las interfaces:

Para poder insertar un planeta, es necesario tener guardado la estrella que orbita

Insertar Planeta

Nombre de planeta Nombre de planeta
Nombre de estrella Nombre de estrella
Nombre de constelación Nombre de sistema
Radio(km) Radio(km)
Tipo de Orbita
Submit

Eliminar constelación

Nombre	Nombre de	constelación	
Submit			

Actualizar estrella

Nombre	Nombre de la estrella	
Edad Ed	dad de la estrella	
Submit]	

Algunas muestras del código de la API donde se realizan las operaciones son los siguientes:

```
@app.route('/')
def index():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute('SELECT * FROM planeta;')
    planetas = cur.fetchall()
    cur.execute('SELECT * FROM estrella;')
    estrellas = cur.fetchall()
    cur.execute('SELECT * FROM constelacion;')
    constelaciones = cur.fetchall()
    cur.execute('SELECT * FROM constelacion;')
    constelaciones = cur.fetchall()
    cur.close()
    conn.close()
    return render_template('index.html', planetas=planetas, estrellas=estrellas, constelaciones=constela
```

```
.
@app.route('/c_constelacion', methods=('GET', 'POST'))
def c_constelacion():
   try:
        if request.method == 'POST':
           n_cons = request.form['n_cons']
           n_estrellas = int(request.form['n_estrellas'])
           conn = get_db_connection()
           cur = conn.cursor()
           if (n_cons != "" and n_estrellas != ""):
                cur.execute('INSERT INTO constelacion (nombre_constelacion, cantidad_estrellas)'
                            (n_cons, n_estrellas))
           conn.commit()
           cur.close()
           conn.close()
       return render_template('c_constelacion.html')
   except psycopg2.Error as e:
       print(e)
        return isonify({"error": e})
```