

# Proyecto Final Administración de Bases de Datos

Alejandro Pérez Álvarez - alu0101215310

Daniel Hernández Fajardo - alu0101320489

## **Índice**

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

## **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 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

Female	Male
--------	------



En dicho modelo encontramos que el centro neurálgico es la estrella, la

A su vez, una estrella forma parte de una única constelación, y las

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( <u>nombre_constelacion</u> , cantidad_estrellas)
estrella( <u>nombre_estrella</u> , <u>nombre_constelación</u> , tamaño, edad)
planeta( <u>nombre_planeta</u> , <u>nombre_estrella</u> , tamaño, orbita, nombre_sistema)
satelite( <u>nombre_satelite</u> , <u>nombre_planeta</u> , tamaño)
agujer_negro( <u>nombre_agujero</u> , distancia, tamaño_horizonte)
galaxia( <u>nombre_galaxia</u> , <u>nombre_agujero</u> )
nebulosa( <u>nombre_galaxia</u> , tamaño)
sistema_planetario( <u>nombre_sistema</u> , <u>nombre_estrella</u> , <u>nombre_galaxia</u> , cantidad_planetas)
cinturon( <u>nombre_cinturon</u> , <u>nombre_estrella</u> , orbita)
asteroide( <u>nombre_asteroide</u> , tamaño)
cometa( <u>nombre_asteroide</u> , <u>nombre_sistema</u> , tamaño_estela, tamaño)
meteorito( <u>nombre_asteroide</u> , <u>nombre_planeta</u> , tamaño)

constelación(nombre\_constelacion, cantidad\_estrellas)

- Como clave primaria encontramos el atributo nombre\_constelación.

estrella(nombre\_estrella, nombre\_constelación, 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(nombre\_planeta, nombre\_estrella, 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(nombre\_satelite, nombre\_planeta, 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.

agujero\_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(nombre\_galaxia, 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(nombre\_sistema, nombre\_estrella, nombre\_galaxia, 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(nombre\_cinturon, nombre\_estrella, 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(nombre\_asteroide, nombre\_sistema, 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(nombre\_asteroide, nombre\_planeta, 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 $$
BEGIN
    UPDATE constelacion SET cantidad_estrellas = (SELECT COUNT(*) FROM estrella WHERE nombre_constelacion = NEW.nombre_constelacion)
    WHERE nombre_constelacion = NEW.nombre_constelacion;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_estrellas_constelacion
AFTER INSERT ON estrella
FOR EACH ROW
EXECUTE FUNCTION update_constelacion_estrellas();

--Cada vez que se inserte un planeta en la tabla planeta, se debe actualizar la cantidad de planetas del sistema planetario correspondiente.
CREATE FUNCTION update_sistema_planetario_planetas() RETURNS TRIGGER AS $$
BEGIN
    UPDATE sistema_planetario SET cantidad_planetas = (SELECT COUNT(*) FROM planeta WHERE nombre_sistema = NEW.nombre_sistema)
    WHERE nombre_sistema = NEW.nombre_sistema;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--Cada vez que se inserte un meteorito o cometa, se debe insertar en la tabla asteroide con su nombre y tamaño.
CREATE TRIGGER update_planetas_sistema_planetario
AFTER INSERT ON planeta
FOR EACH ROW
EXECUTE FUNCTION update_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 $$
BEGIN
    IF (TG_OP = 'INSERT') THEN
        IF (NEW.nombre_planeta = 'La Tierra') THEN
            CASE
                WHEN NEW.tamaño <= 3 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 1);
                WHEN NEW.tamaño <= 6 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 2);
                WHEN NEW.tamaño <= 9 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 3);
                WHEN NEW.tamaño <= 12 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 4);
                WHEN NEW.tamaño <= 15 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 5);
                WHEN NEW.tamaño <= 18 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 6);
                WHEN NEW.tamaño <= 25 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 7);
                WHEN NEW.tamaño <= 40 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 8);
                WHEN NEW.tamaño <= 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 9);
                WHEN NEW.tamaño > 60 THEN INSERT INTO meteoritos_tierra (nombre_meteorito, fecha, peligrosidad) VALUES (NEW.nombre_asteroide, NOW(), 10);
            END CASE;
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insert_meteorito_tierra
AFTER INSERT ON meteorito
FOR EACH ROW
EXECUTE FUNCTION insert_meteorito_tierra();

```

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.

```

1 INSERT INTO asteroide (nombre_asteroide, tamaño) VALUES ('Rita', 18);
2 INSERT INTO meteorito (nombre_asteroide, nombre_planeta, tamaño) VALUES ('Rita', 'La Tierra', 18);
3 SELECT * FROM meteoritos_tierra;
4

```

Data Output Messages Notifications










	nombre_meteorito [PK] character varying (30)	fecha timestamp without time zone	peligrosidad integer
1	Rita	2023-01-12 21:01:05.018303	6

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.

Query Query History

```
1 INSERT INTO estrella(nombre_estrella, nombre_constelacion, tamaño, edad)
2 VALUES ('Hadar', 'Centaurus', 907280, 700203653);
3 SELECT * FROM constelacion;
4
5
6
```

Data Output Messages Notifications



	nombre_constelacion [PK] character varying (30)	cantidad_estrellas bigint
1	Ofiuco	2
2	Centaurus	2

```

1 INSERT INTO planeta(nombre_planeta, nombre_estrella, nombre_sistema, tamaño, orbita)
2 VALUES ('Mercurio', 'Sol', 'Sistema Solar', 12932, 'Circular');
3 SELECT * FROM sistema_planetario;
4 |

```

Data Output Messages Notifications

	nombre_sistema [PK] character varying (30)	nombre_estrella character varying (30)	nombre_galaxia character varying (30)	cantidad_planetas bigint
	Proxima	Proxima Centauri	Via Láctea	4
!	Titawin	Barnard	Triangulum	4
!	Sistema Solar	Sol	Via Láctea	3

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.

```
1 INSERT INTO asteroide (nombre_asteroide, tamaño) VALUES ('Timo', 18);
2 INSERT INTO cometa (nombre_asteroide, nombre_sistema, tamaño_estela, tamaño) VALUES ('Timo', 'Sistema Solar',
3 SELECT * FROM cometa_detectados;
4
```

Data Output Messages Notifications

	nombre_cometa [PK] character varying (30)	fecha timestamp without time zone
1	Timo	2023-01-12 21:45:17.436969

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

```
1 UPDATE sistema_planetario SET cantidad_planetas = -10 WHERE nombre_sistema = 'Proxima';
2
3
4
```

Data Output Messages Notifications

```
ERROR: new row for relation "sistema_planetario" violates check constraint "sistema_planetario_planetas_negativo"
DETAIL: Failing row contains (Proxima, Proxima Centauri, Vía Láctea, -10).
SQL state: 23514
```

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 su vez los planetas que orbitan esas estrellas.

```
1 DELETE FROM constelacion WHERE nombre_constelacion = 'Centaurus';
2 SELECT * FROM planeta;
```

Data Output Messages Notifications

	nombre_planeta [PK] character varying (30)	nombre_estrella character varying (30)	nombre_sistema character varying (30)	tamaño bigint	orbita character varying (30)
1	La Tierra	Sol	Sistema Solar	69340	Circular
2	Jupiter	Sol	Sistema Solar	21440	Circular
3	Mercurio	Sol	Sistema Solar	12932	Circular

## 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

#Ofiuco

*Numero de estrellas: 6*

#Centaurus

*Numero de estrellas: 9*

### Estrellas

#Sol

*Constelación: Ofiuco*

*Radio(km): 696340*

*Edad: 4000000603*

#Proxima Centauri

*Constelación: Centaurus*

*Radio(km): 107280*

*Edad: 4000000653*

#Barnard

*Constelación: Ofiuco*

*Radio(km): 136360*

*Edad: 10000000001*

### Planetas

#La Tierra

*Estrella que orbita: Sol*

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

## Constelaciones, Estrellas y Planetas

---

### Inserciones

[Insertar constelación](#)

[Insertar estrella](#)

[Insertar planeta](#)

### 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 estrella

Nombre de constelación

Radio(km)

Tipo de Orbita

## Eliminar constelación

Nombre

## Actualizar estrella

Nombre

Edad



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.close()
    conn.close()
    return render_template('index.html', planetas=planetas, estrellas=estrellas, constelaciones=constelaciones)
```

```
@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)'
                            'VALUES (%s, %s)',
                            (n_cons, n_estrellas))
            conn.commit()
            cur.close()
            conn.close()

            return render_template('c_constelacion.html')
    except psycopg2.Error as e:
        print(e)
        return jsonify({"error": e})
```

```

@app.route('/d_planeta/', methods=(['GET', 'POST']))
def d_planeta():
    try:
        if request.method == 'POST':
            n_planeta = request.form['n_planeta']

            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute('DELETE FROM planeta WHERE nombre_planeta = %s;', (n_planeta,))
            conn.commit()
            cur.close()
            conn.close()

        return render_template('d_planeta.html')
    except psycopg2.Error as e:
        print(e)
        return jsonify({"error": e})

```