CANARY ISLANDS DATABASE

Samuel Martín Morales alu0101359526@ull.edu.es
Jorge Domínguez González alu0101330600@ull.edu.es
Cheuk Kelly Ng Pante alu0101364544@ull.edu.es

19 de diciembre de 2023

1. Objetivos del Proyecto

El objetivo del proyecto es el diseño, creación e implementación de una base de datos para gestionar información relacionada con las Islas Canarias. La base de datos debe permitir realizar operaciones CRUD sobre la información almacenada en ella, así como consultas de prueba para demostrar su funcionamiento.

2. Descripción del contexto de la base de datos

La base de datos está destinada para almacenar información sobre las Islas Canarias, como por ejemplo, información sobre las islas, su distribución poblacional, compañías, sitios de interés y animales autóctonos. La gestión de la información de las Islas Canarias es de gran importancia para el turismo, ya que permite a los turistas conocer mejor las islas y su historia, así como los lugares de interés que pueden visitar y los animales o plantas autóctonas que se pueden encontrar en nuestro archipiélago.

2.1. Entidades:

- **Isla:**

  - Atributos: ID (clave primaria), Nombre.

- **Distribución Poblacional:**

  - Atributos: ID (clave primaria), Nombre, Provincia, Capital, Municipio, Poblacion Isla.

- **Compañías:**

  - Atributos: ID (clave primaria), Nombre, Tipo, Sede (relacionada con Islas), Año Fundacion.

- **Comestibles:**

  - Atributos: ID (clave primaria), Nombre, Tipo, Compañia.

- **Productos:**

  - Atributos: ID (clave primaria), Nombre, Islas.

- **Artesania:**

  - Atributos: ID (clave primaria), Nombre, Creador, Tipo.

- **Sitio interes:**

  - Atributos: ID (clave primaria), Nombre, Islas, Nombre Isla, Municipio, Latitud, Longitud,

- **Folklore:**

  - Atributos: ID (clave primaria), Nombre, Lanzamiento, Autor.

- **Seres Vivos:**

  - Atributos: ID (clave primaria), Nombre.

- **Animales autóctonos:**

  - Atributos: ID (clave primaria), Nombre, Nombre Cientifico, Islas, Invasoras, Dieta.

- **Plantas autóctonas:**

    - Atributos: ID (clave primaria), Nombre, Nombre Cientifico, Islas, Invasoras.

- **Nombres Canarios:**

    - Atributos: ID (clave primaria), Nombre, Isla, Genero.

- **Platos:**

    - Atributos: ID (clave primaria), Nombre, Tipo.

- **Ingredientes:**

    - Atributos: ID (clave primaria), Nombre.

Entre estas entidades nos encontramos con los siguientes tipos de entidades:

- **Entidades Fuertes:** Isla, Compañia, Comestibles, Productos, Artesania, Seres Vivos, Nombres Canarios, Platos, Ingredientes.
- **Entidades Débiles:** Distribución Poblacional, Sitio interes, Folklore, Animales autóctonos, Plantas autóctonas.

2.2. Relaciones:

Las relaciones entre las entidades son las siguientes:

- **Isla a Distribución Poblacional**

    - Relacionada por la columna Nombre con la entidad Islas.

- **Compañías a Islas:**

    - Relacionada por la columna Sede con la entidad Islas.

- **Sitio interes a Islas:**

    - Relacionada por la columna Isla con la entidad Islas.

- **Animales autóctonos a Islas:**

    - Relacionada por la columna Islas con la entidad Islas.

2.3. Consideraciones Adicionales:

- La relación entre las entidades Compañías y Islas se establece a través de la columna Sede, indicando la isla donde tienen su sede las compañías.
- La relación entre Sitios Interes y Islas se establece por la columna Isla, indicando en qué isla se encuentra el sitio de interés.
- La relación entre Animales autóctonos e Islas se establece por la columna Islas, indicando las islas a las que están asociados los animales autóctonos.

2.4. Restricciones:

- La relación entre las entidades Compañías y Islas se establece a través de la columna Sede, indicando la isla donde tienen su sede las compañías.
- La relación entre Sitio interes y Islas se establece por la columna Isla, indicando en qué isla se encuentra el sitio de interés.
- La relación entre Animales autóctonos e Islas se establece por la columna Islas, indicando las islas a las que están asociados los animales autóctonos.

3. Diseño Conceptual

3.1. Modelo Entidad-Relación

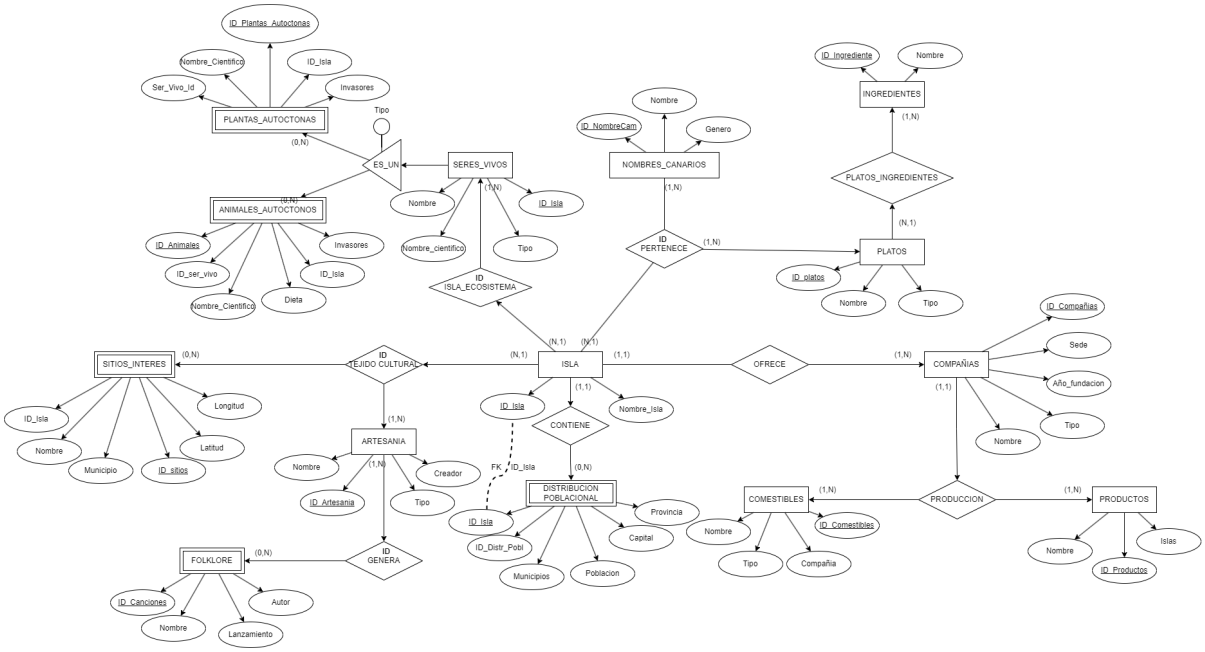

Figura 3.1: Modelo Entidad-Relación

3.2. Modelo Relacional

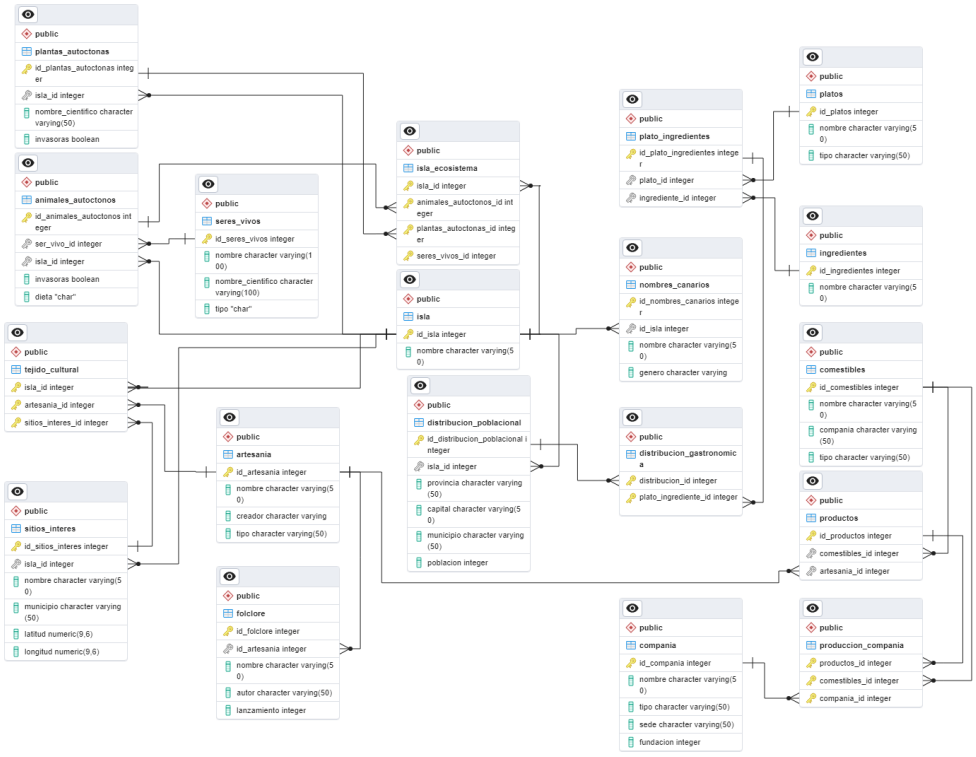

Figura 3.2: Modelo Relacional

3.3. Supuestos Semánticos

Documentación que explique los supuestos semánticos y decisiones de diseño.

4. Scripts SQL

El script *canary_islands.sql* contiene la implementación de la base de datos en PostgreSQL. Para la ejecución del script desde *psql* se debe ejecutar el siguiente comando:

```
$ sudo -u postgres psql
postgres=# \i canary_islands.sql
```

4.1. Creación de la Base de Datos

```
DROP DATABASE IF EXISTS islas_canarias;
CREATE DATABASE islas_canarias with TEMPLATE = template0 ENCODING = 'UTF8';

ALTER DATABASE islas_canarias OWNER TO postgres;

\connect islas_canarias

DROP SCHEMA IF EXISTS public CASCADE;
CREATE SCHEMA public;

ALTER SCHEMA public OWNER TO postgres;

SET default_tablespace = '';

SET default_table_access_method = heap;
```

4.2. Inicialización de las tablas

Al ejecutar el script *canary_islands.sql* se crean las tablas de la base de datos, así como las relaciones entre ellas.

- **Tabla Isla:** Es una tabla que representa cada isla del archipiélago canario.

```
CREATE TABLE isla (
    id_isla SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL
);
```

- **Tabla Seres Vivos:** Es una tabla que representa a los seres vivos que habitan en las islas.

```
CREATE TABLE seres_vivos (
    id_seres_vivos SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    nombre_cientifico VARCHAR(100) NOT NULL
);
```

- **Tabla Animales Autoctonos:** Es una tabla que contiene información sobre los animales autóctonos de las islas.

```
CREATE TABLE animales_autoctonos (
    id_animales_autoctonos SERIAL PRIMARY KEY,
    ser_vivo_id INT REFERENCES seres_vivos(id_seres_vivos),
    isla_id INTEGER NOT NULL,
    invasoras BOOLEAN NOT NULL,
    dieta VARCHAR(50) NOT NULL,
    foto VARCHAR(100) NOT NULL,
    CONSTRAINT animales_autoctonos_isla_fkey
        FOREIGN KEY (isla_id)
        REFERENCES isla (id_isla) ON DELETE CASCADE
);
```

- **Tabla Plantas Autoctonas:** Es una tabla que contiene información sobre las plantas autóctonas de las islas.

```
CREATE TABLE plantas_autoctonas (
    id_plantas_autoctonas SERIAL PRIMARY KEY,
    ser_vivo_id INT REFERENCES seres_vivos(id_seres_vivos),
    isla_id INTEGER NOT NULL,
    invasoras BOOLEAN NOT NULL,
    foto VARCHAR(100) NOT NULL,
    CONSTRAINT plantas_autoctonas_isla_fkey
        FOREIGN KEY (isla_id)
        REFERENCES isla (id_isla) ON DELETE CASCADE
);
```

- **Tabla Sitios Interes:** Es una tabla que contiene información sobre los sitios de interés de las islas.

```
CREATE TABLE sitios_interes (
    id_sitios_interes SERIAL PRIMARY KEY,
    isla_id INTEGER NOT NULL,
    nombre VARCHAR(50) NOT NULL,
    municipio VARCHAR(50) NOT NULL,
    latitud DECIMAL(9,6) NOT NULL,
    longitud DECIMAL(9,6) NOT NULL,
    foto VARCHAR(100) NOT NULL,
    CONSTRAINT sitios_interes_isla_fkey
        FOREIGN KEY (isla_id)
        REFERENCES isla  (id_isla) ON DELETE CASCADE
);
```

- **Tabla Distribución Poblacional:** Es una tabla que contiene información sobre la distribución poblacional, como puede ser los municipios que tiene y la poblacion de cada isla.

```
CREATE TABLE distribucion_poblacional (
    id_distribucion_poblacional SERIAL PRIMARY KEY,
    isla_id INTEGER NOT NULL,
    provincia VARCHAR(50) NOT NULL,
    capital VARCHAR(50) NOT NULL,
    municipio VARCHAR(50) NOT NULL,
    poblacion INTEGER NOT NULL,
    CONSTRAINT distribucion_poblacional_isla_fkey
        FOREIGN KEY (isla_id)
        REFERENCES isla (id_isla) ON DELETE CASCADE
);
```

- **Tabla Nombres Canarios:** Es una tabla que contiene nombres canarios.

```
CREATE TABLE nombres_canarios (
    id_nombres_canarios SERIAL PRIMARY KEY,
    id_isla INTEGER NOT NULL,
    nombre VARCHAR(50) NOT NULL,
    CONSTRAINT nombres_canarios_isla_fkey
        FOREIGN KEY (id_isla)
        REFERENCES isla (id_isla) ON DELETE CASCADE
);
```

- **Tabla Platos:** Es una tabla que contiene los platos más típicos de las islas.

```
CREATE TABLE platos (
    id_platos SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    tipo VARCHAR(50) NOT NULL
);
```

- **Tabla Ingredientes:** Es una tabla auxiliar que contiene los ingredientes de los platos.

```
CREATE TABLE ingredientes (
    id_ingredientes SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL
);
```

- **Tabla Comestibles:** Es una tabla que contiene los comestibles, como dulces o bebibas, más típicos de las islas.

```
CREATE TABLE comestibles (
    id_comestibles SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    compania VARCHAR(50) NOT NULL,
    tipo VARCHAR(50) NOT NULL
);
```

- **Tabla Compania:** Es una tabla que contiene información sobre algunas compañías de las islas.

```
CREATE TABLE compania (
    id_compania SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    sede VARCHAR(50) NOT NULL,
    fundacion INTEGER NOT NULL
);
```

- **Tabla Artesania:** Es una tabla que contiene información sobre las artesanías más típicas de las islas.

```
CREATE TABLE artesania (
    id_artesania SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    creador VARCHAR,
    tipo VARCHAR(50) NOT NULL
);
```

- **Tabla Folclore:** Es una tabla que contiene algunas canciones de folclore más típicas de las islas.

```
CREATE TABLE folclore (
    id_folclore SERIAL PRIMARY KEY,
    id_artesania INT REFERENCES artesania(id_artesania),
    nombre VARCHAR(50) NOT NULL,
    tipo VARCHAR(50) NOT NULL
);
```

- **Tabla Isla Ecosistema:**

```
CREATE TABLE isla_ecosistema (
    isla_id INT REFERENCES isla(id_isla),
    seres_vivos_id INT REFERENCES seres_vivos(id_seres_vivos),
    animales_autoctonos_id INT REFERENCES animales_autoctonos(id_animales_autoctonos),
    plantas_autoctonas_id INT REFERENCES plantas_autoctonas(id_plantas_autoctonas),
    PRIMARY KEY (isla_id, seres_vivos_id, animales_autoctonos_id, plantas_autoctonas_id
    )
);
```

- **Tabla Tejido Cultural:**

```
CREATE TABLE tejido_cultural (
    isla_id INT REFERENCES isla(id_isla),
    artesania_id INT REFERENCES artesania(id_artesania),
    sitios_interes_id INT REFERENCES sitios_interes(id_sitios_interes),
    PRIMARY KEY (isla_id, artesania_id, sitios_interes_id)
);
```

- **Tabla Plato Ingredientes:**

```
CREATE TABLE plato_ingredientes (
    id_plato_ingredientes SERIAL PRIMARY KEY,
    plato_id INT REFERENCES platos(id_platos),
    ingrediente_id INT REFERENCES ingredientes(id_ingredientes)
);
```

- **Tabla Productos:**

```
CREATE TABLE productos (
    id_productos SERIAL PRIMARY KEY,
    comestibles_id INT REFERENCES comestibles(id_comestibles),
    artesania_id INT REFERENCES artesania(id_artesania)
);
```

- **Tabla Produccion Compañia:**

```
CREATE TABLE produccion_compania (
    productos_id INT REFERENCES productos(id_productos),
    comestibles_id INT REFERENCES comestibles(id_comestibles),
    compania_id INT REFERENCES compania(id_compania),
    PRIMARY KEY (productos_id, comestibles_id, compania_id)
);
```

- **Tabla Distribución Gastronomica:**

```
CREATE TABLE distribucion_gastronomica (
  isla_id INT REFERENCES isla(id_isla),
  plato_ingrediente_id INT REFERENCES plato_ingredientes(id_plato_ingredientes),
  PRIMARY KEY (isla_id, plato_ingrediente_id)
);
```

4.3. Inclusión de Datos en las Tablas

Aqui varios ejemplos de como se insertan los datos en las diferentes tablas de la base de datos:

```sql
-- -- Inclusion de datos en la tabla de islas
INSERT INTO isla (nombre) VALUES ('Lanzarote');

-- -- Inclusion de datos en la tabla de distribucion poblacional
INSERT INTO distribucion_poblacional (isla_id, provincia, capital, municipio, poblacion)
VALUES (3, 'Las Palmas', 'Arrecife', 'San Bartolome', 156112);

-- -- Inclusion de datos en la tabla de seres vivos
INSERT INTO seres_vivos(nombre, nombre_cientifico, tipo)
VALUES ('Lagarto Gigante de El Hierro', 'Gallotia simonyi', 'Animal');

-- -- Inclusion de datos en la tabla de animales autoctonos
INSERT INTO animales_autoctonos(ser_vivo_id, isla_id, invasoras, dieta)
VALUES (1, 7, false, 'Insectivoro');

-- -- Inclusion de datos en la tabla de plantas autoctonas
INSERT INTO plantas_autoctonas(ser_vivo_id, isla_id, invasoras) VALUES (21, 2, false);

-- -- Inclusion de datos en la tabla de sitios de interes
INSERT INTO sitios_interes(isla_id, nombre, municipio, latitud, longitud)
VALUES (3, 'Parque Nacional de Timanfaya', 'Yaiza', 29.016667, -13.75);

-- -- Inclusion de datos en la tabla de nombres canarios
INSERT INTO nombres_canarios(id_isla, nombre, genero) VALUES (1, 'Yaiza', 'Mujer');

-- -- Inclusion de datos en la tabla de platos
INSERT INTO platos(nombre, tipo) VALUES ('Papas arrugadas', 'Principal');

-- -- Inclusion de datos en la tabla de ingredientes
INSERT INTO ingredientes(nombre) VALUES ('Carne de cabra');

-- -- Inclusion de datos en la tabla de platos_ingredientes
INSERT INTO plato_ingredientes(plato_id, ingrediente_id) VALUES (1, 1);

-- -- Inclusion de datos en la tabla de comestibles
INSERT INTO comestibles(nombre, compania, tipo) VALUES ('Clipper', 'Ahembo', 'Bebida');

-- -- Inclusion de datos en la tabla de compania
INSERT INTO compania(nombre, tipo, sede, fundacion)
VALUES ('Binter Canarias', 'Aerolinea', 'Gran Canaria', 1989);

-- -- Inclusion de datos en la tabla de artesania
INSERT INTO artesania(isla_id, nombre, creador, tipo)
VALUES (1, 'Cuchillo Canario', 'Desconocido', 'Herramienta');

-- -- Inclusion de datos en la tabla de folclore
INSERT INTO folclore(id_artesania, nombre, autor, lanzamiento)
VALUES (5, 'Cabra Loca', 'Los Gofiones', 2012);

-- -- Inclusion de datos en la tabla de isla_ecosistema
ALTER TABLE isla_ecosistema
ALTER COLUMN plantas_autoctonas_id DROP NOT NULL,
ALTER COLUMN animales_autoctonos_id DROP NOT NULL;

TRUNCATE TABLE isla_ecosistema;

```

```sql
INSERT INTO isla_ecosistema(isla_id, seres_vivos_id, animales_autoctonos_id,
    plantas_autoctonas_id)
SELECT isla_id, ser_vivo_id, id_animales_autoctonos, NULL
FROM animales_autoctonos;

INSERT INTO isla_ecosistema(isla_id, seres_vivos_id, animales_autoctonos_id,
    plantas_autoctonas_id)
SELECT isla_id, ser_vivo_id, NULL, id_plantas_autoctonas
FROM plantas_autoctonas;

-- -- Inclusion de datos de la tabla tejido_cultural
ALTER TABLE tejido_cultural
ALTER COLUMN artesania_id DROP NOT NULL,
ALTER COLUMN sitios_interes_id DROP NOT NULL;

TRUNCATE TABLE tejido_cultural;

INSERT INTO tejido_cultural(isla_id, artesania_id, sitios_interes_id)
SELECT isla_id, id_artesania, NULL
FROM artesania;

INSERT INTO tejido_cultural(isla_id, artesania_id, sitios_interes_id)
SELECT isla_id, NULL, id_sitios_interes
FROM sitios_interes;

-- -- Inclusion de datos de la tabla productos
ALTER TABLE productos
ALTER COLUMN comestibles_id DROP NOT NULL,
ALTER COLUMN artesania_id DROP NOT NULL;

INSERT INTO productos(comestibles_id, artesania_id)
SELECT id_comestibles, NULL
FROM comestibles;

INSERT INTO productos(comestibles_id, artesania_id)
SELECT NULL, id_artesania
FROM artesania;

-- -- Inclusion de datos de la tabla produccion_compania
INSERT INTO produccion_compania(comestibles_id, compania_id)
SELECT id_comestibles, id_compania
FROM comestibles INNER JOIN compania ON comestibles.compania = compania.nombre;

-- -- Inclusion de datos de la tabla distribucion gastronomica
INSERT INTO distribucion_gastronomica(isla_id, platos_id)
SELECT isla.id_isla, platos.id_platos
FROM isla, platos;
```

4.4. Implementación de Triggers

- **Trigger 1:**

```
-- -- Trigger para la tabla de isla_ecosistema
-- Si se anade una nueva tupla dentro de la tabla de animales_autoctonos, se anadira una nueva tupla en la tabla de isla_ecosistema
CREATE OR REPLACE FUNCTION insertar_animal_autoctono() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO isla_ecosistema(isla_id, seres_vivos_id, animales_autoctonos_id, plantas_autoctonas_id)
    VALUES (NEW.isla_id, NEW.ser_vivo_id, NEW.id_animales_autoctonos, NULL);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_animal_autoctono
AFTER INSERT ON animales_autoctonos
FOR EACH ROW
EXECUTE PROCEDURE insertar_animal_autoctono();
```

Como se puede observar en el código anterior, este se encarga de asegurar que cada vez que se inserta una nueva fila dentro de la tabla *animales_autoctonos*, se realiza automáticamente una inserción correspondiente en la tabla *isla_ecosistema* con los valores apropiados. Esto permite que cada animal autóctono esté asociado a las islas en las que se encuentra.

- **Trigger 2:**

```
-- Si se anade una nueva tupla dentro de la tabla de plantas_autoctonas, se anadira una nueva tupla en la tabla de isla_ecosistema
CREATE OR REPLACE FUNCTION insertar_planta_autoctona() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO isla_ecosistema(isla_id, seres_vivos_id, animales_autoctonos_id, plantas_autoctonas_id)
    VALUES (NEW.isla_id, NEW.ser_vivo_id, NULL, NEW.id_plantas_autoctonas);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_planta_autoctona
AFTER INSERT ON plantas_autoctonas
FOR EACH ROW
EXECUTE PROCEDURE insertar_planta_autoctona();
```

Este trigger permite asegurarse que cada vez que se inserta una nueva fila en la tabla *animales_autoctonos*, se realiza automáticamente una inserción correspondiente en la tabla *plantas_autoctonas* con los valores apropiados. Esto permite que cada planta autóctona esté asociada a las islas en las que se encuentra y su impacto en el ecosistema.

- **Trigger 3:**

```
-- Si se elimina una tupla dentro de la tabla de animales_autoctonos, se eliminara la tupla correspondiente en la tabla de isla_ecosistema
CREATE OR REPLACE FUNCTION eliminar_animal_autoctono() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM isla_ecosistema
```

```
    WHERE animales_autoctonos_id = OLD.id_animales_autoctonos;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_animal_autoctono
BEFORE DELETE ON animales_autoctonos
FOR EACH ROW
EXECUTE PROCEDURE eliminar_animal_autoctono();
```

De forma resumida, el trigger permite que cada vez que se elimina una fila de la tabla *animales_autoctonos*, se elimina de manera automática la fila correspondiente en la tabla *isla_ecosistema*. Esto garantiza la consistencia de los datos, asegurándose que no haya referencia no válidas dentro de la tabla *isla_ecosistema* después de la eliminación de un animal autóctono.

- **Trigger 4:**

```
-- Si se elimina una tupla dentro de la tabla de plantas_autoctonas, se eliminara la
    tupla correspondiente en la tabla de isla_ecosistema
CREATE OR REPLACE FUNCTION eliminar_planta_autoctona() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM isla_ecosistema
    WHERE plantas_autoctonas_id = OLD.id_plantas_autoctonas;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_planta_autoctona
BEFORE DELETE ON plantas_autoctonas
FOR EACH ROW
EXECUTE PROCEDURE eliminar_planta_autoctona();
```

En resumen, este trigger asegura que cada vez que se elimina una fila de la tabla *platas_autoctonas*, se elimina automáticamente la fila correspondiente en la tabla *isla_ecosistema*. Esto, permite respaldar que no haya referencias no válidas en la tabla *isla_ecosistema* después de la eliminación de una planta autóctona.

- **Trigger 5:**

```
-- -- Trigger para la tabla de tejido_cultural
-- Si se anade una nueva tupla dentro de la tabla de artesania, se anadira una nueva
    tupla en la tabla de tejido_cultural
CREATE OR REPLACE FUNCTION insertar_artesania() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO tejido_cultural(isla_id, artesania_id, sitios_interes_id)
    VALUES (NEW.isla_id, NEW.id_artesania, NULL);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_artesania
AFTER INSERT ON artesania
FOR EACH ROW
EXECUTE PROCEDURE insertar_artesania();
```

El trigger permite que cada vez que se inserta una nueva fila en la tabla *artesanía*, se realiza de manera automática una inserción correspondiente en la tabla *tejido_cultural* con los valores apropiados. Esto permite que cada artesanía esté asociada a las islas en las que se encuentra y su contribución al tejido cultural.

- **Trigger 6:**

```
-- Si se anade una nueva tupla dentro de la tabla de sitios_interes, se anadira una
    nueva tupla en la tabla de tejido_cultural
CREATE OR REPLACE FUNCTION insertar_sitio_interes() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO tejido_cultural(isla_id, artesania_id, sitios_interes_id)
    VALUES (NEW.isla_id, NULL, NEW.id_sitios_interes);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_sitio_interes
AFTER INSERT ON sitios_interes
FOR EACH ROW
EXECUTE PROCEDURE insertar_sitio_interes();
```

Este, comprueba que cuando se inserta una nueva fila dentro de la tabla *artesanía*, se realiza una inserción en la tabla *tejido_cultural* . Esto está relacionado con el seguimiento de la artesanía en una isla y su contribución al tejido cultural.

- **Trigger 7:**

```
-- Si se elimina una tupla dentro de la tabla de artesania, se eliminara la tupla
    correspondiente en la tabla de tejido_cultural
CREATE OR REPLACE FUNCTION eliminar_artesania() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM tejido_cultural
    WHERE artesania_id = OLD.id_artesania;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_artesania
BEFORE DELETE ON artesania
FOR EACH ROW
EXECUTE PROCEDURE eliminar_artesania();
```

En síntesis, este trigger permite que cada vez que se elimina una fila de la tabla *artesanía*, se elimina automáticamente la fila correspondiente en la tabla *tejido_cultural*. Esto, permite respaldar que no haya referencias no válidas en la tabla *tejido_cultural* después de la eliminación de una artesanía.

- **Trigger 8:**

```
-- Si se elimina una tupla dentro de la tabla de sitios_interes, se eliminara la tupla
    correspondiente en la tabla de tejido_cultural
CREATE OR REPLACE FUNCTION eliminar_sitio_interes() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM tejido_cultural
    WHERE sitios_interes_id = OLD.id_sitios_interes;
    RETURN OLD;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_sitio_interes
BEFORE DELETE ON sitios_interes
FOR EACH ROW
EXECUTE PROCEDURE eliminar_sitio_interes();
```

De forma resumida, cada vez que se elimina una fila de la tabla *sitios_interes*, se elimina automáticamente la fila correspondiente en la tabla *tejido_cultural*. Esto, permite respaldar que no haya referencias no válidas en la tabla *tejido_cultural* después de la eliminación de un sitio de interés.

- **Trigger 9:**

```
-- -- Trigger para la tabla de productos
-- Si se anade una nueva tupla dentro de la tabla de comestibles, se anadira una nueva
    tupla en la tabla de productos
CREATE OR REPLACE FUNCTION insertar_comestible() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO productos(comestibles_id, artesania_id)
    VALUES (NEW.id_comestibles, NULL);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_comestible
AFTER INSERT ON comestibles
FOR EACH ROW
EXECUTE PROCEDURE insertar_comestible();
```

En resumen , se asegura que cada vez que se inserta una nueva fila en la tabla *comestibles*, se realiza automáticamente una inserción correspondiente en la tabla *productos*.

- **Trigger 10:**

```
-- Si se anade una nueva tupla dentro de la tabla de artesania, se anadira una nueva
    tupla en la tabla de productos
CREATE OR REPLACE FUNCTION insertar_artesania_productos() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO productos(comestibles_id, artesania_id)
    VALUES (NULL, NEW.id_artesania);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_artesania_productos
AFTER INSERT ON artesania
FOR EACH ROW
EXECUTE PROCEDURE insertar_artesania_productos();
```

Resumidamente, cuando se inserta una fila en la tabla *artesanía*, se realiza la inserción en la tabla *productos*.

- **Trigger 11:**

```
-- Si se elimina una tupla dentro de la tabla de comestibles, se eliminara la tupla
    correspondiente en la tabla de productos
```

```
CREATE OR REPLACE FUNCTION eliminar_comestible() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM productos
    WHERE comestibles_id = OLD.id_comestibles;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_comestible
BEFORE DELETE ON comestibles
FOR EACH ROW
EXECUTE PROCEDURE eliminar_comestible();
```

Cuando se elimina una fila en la tabla *comestibles*, se elimina automáticamente la fila correspondiente en la tabla *productos*. Esto, permite respaldar que no haya referencias no válidas en la tabla *productos* después de la eliminación de un comestible.

- **Trigger 12:**

```
-- Si se elimina una tupla dentro de la tabla de artesania, se eliminara la tupla
    correspondiente en la tabla de productos
CREATE OR REPLACE FUNCTION eliminar_artesania_productos() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM productos
    WHERE artesania_id = OLD.id_artesania;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_artesania_productos
BEFORE DELETE ON artesania
FOR EACH ROW
EXECUTE PROCEDURE eliminar_artesania_productos();
```

Este, se activa antes de eliminar una fila de la tabla *artesanía*. A continuación, elimina la fila correspondientes en la tabla *productos* . Esto, permite respaldar que no haya referencias no válidas en la tabla *productos* después de la eliminación de una artesanía.

- **Trigger 13:**

```
-- -- Trigger para la tabla de produccion_compania
-- Si se anade una nueva tupla dentro de la tabla de comestibles, se anadira una nueva
    tupla en la tabla de produccion_compania
CREATE OR REPLACE FUNCTION insertar_comestible_produccion() RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM produccion_compania WHERE comestibles_id = NEW.
    id_comestibles AND compania_id = (SELECT id_compania FROM compania WHERE nombre =
    NEW.compania)) THEN
        INSERT INTO produccion_compania(comestibles_id, compania_id)
        VALUES (NEW.id_comestibles, (SELECT id_compania FROM compania WHERE nombre =
    NEW.compania));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```
CREATE TRIGGER insertar_comestible_produccion
AFTER INSERT ON comestibles
FOR EACH ROW
EXECUTE PROCEDURE insertar_comestible_produccion();
```

El trigger se activa después de insertar una nueva fila en la tabla *comestibles*. Su función es verificar si ya existe una fila en la tabla *producción_compania*. Si no existe, se inserta una nueva fila en la tabla *producción_compania* . Esto, permite respaldar que no haya referencias no válidas en la tabla *producción_compania* después de la inserción de un comestible.

- **Trigger 14:**

```
-- Si se elimina una tupla dentro de la tabla de comestibles, se eliminara la tupla
    correspondiente en la tabla de produccion_compania
CREATE OR REPLACE FUNCTION eliminar_comestible_produccion() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM produccion_compania
    WHERE comestibles_id = OLD.id_comestibles;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_comestible_produccion
BEFORE DELETE ON comestibles
FOR EACH ROW
EXECUTE PROCEDURE eliminar_comestible_produccion();
```

Este, se activa antes de eliminar una fila de la tabla *comestibles*. La función es eliminar la fila correspondiente en la tabla *producción_compania*. Esto, permite respaldar que no haya referencias no válidas en la tabla *producción_compania* después de la eliminación de un comestible.

- **Trigger 15:**

```
-- -- Trigger para la tabla de distribucion_gastronomica
-- Si se anade una nueva tupla dentro de la tabla de platos, se anadira una nueva tupla
     en la tabla de distribucion_gastronomica
CREATE OR REPLACE FUNCTION insertar_plato_distribucion() RETURNS TRIGGER AS $$
DECLARE
    isla RECORD;
BEGIN
    FOR isla IN SELECT * FROM isla
    LOOP
        INSERT INTO distribucion_gastronomica(isla_id, platos_id)
        VALUES (isla.id_isla, NEW.id_platos);
    END LOOP;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_plato_distribucion
AFTER INSERT ON platos
FOR EACH ROW
EXECUTE PROCEDURE insertar_plato_distribucion();
```

Cuando se inserta una nueva fila en la tabla *platos*. La función se recorre todas las filas de la tabla *isla* e inserta una nueva fila en la tabla *distribución_gastronomica*. Esto, permite respaldar que

no haya referencias no válidas en la tabla *distribución_gastronomica* después de la inserción de un plato.

- **Trigger 16:**

```
1
2  -- Si se elimina una tupla dentro de la tabla de platos, se eliminara la tupla
       correspondiente en la tabla de distribucion_gastronomica
3  CREATE OR REPLACE FUNCTION eliminar_plato_distribucion() RETURNS TRIGGER AS $$
4  BEGIN
5      DELETE FROM distribucion_gastronomica
6      WHERE platos_id = OLD.id_platos;
7      RETURN OLD;
8  END;
9  $$ LANGUAGE plpgsql;
10
11 CREATE TRIGGER eliminar_plato_distribucion
12 BEFORE DELETE ON platos
13 FOR EACH ROW
14 EXECUTE PROCEDURE eliminar_plato_distribucion();
```

En el momento en el que se elimina una fila de la tabla *platos*, se realiza la eliminación de la fila correspondiente en la tabla *distribución_gastronomica*. Esto, permite respaldar que no haya referencias no válidas en la tabla *distribución_gastronomica* después de la eliminación de un plato.

- **Trigger 17:**

```
1  -- -- Trigger para la tabla de seres_vivos
2  -- Si se anade una nueva tupla dentro de la tabla de seres_vivos, se anadira una nueva
       tupla en la tabla de animales_autoctonos
3  CREATE OR REPLACE FUNCTION insertar_seres_vivos() RETURNS TRIGGER AS $$
4  BEGIN
5      IF NEW.tipo = 'Animal' THEN
6          INSERT INTO animales_autoctonos(ser_vivo_id, isla_id)
7          VALUES (NEW.id_seres_vivos, 1);
8      ELSE
9          INSERT INTO plantas_autoctonas(ser_vivo_id, isla_id)
10         VALUES (NEW.id_seres_vivos, 1);
11     END IF;
12     RETURN NEW;
13 END;
14 $$ LANGUAGE plpgsql;
15
16 CREATE TRIGGER insertar_seres_vivos
17 AFTER INSERT ON seres_vivos
18 FOR EACH ROW
19 EXECUTE PROCEDURE insertar_seres_vivos();
```

Este trigger se activa cuando se inserta una nueva fila en la tabla *seres_vivos*. La función es verificar el tipo de ser vivo que se acaba de agregar (animal o planta), y, en base a dicha condición se inserta una nueva fila dentro de la tabla *animales_autoctonos* o *plantas_autoctonas*.

- **Trigger 18:**

```
1  -- Si se elimina una tupla dentro de la tabla de seres_vivos, se eliminara la tupla
       correspondiente en la tabla de animales_autoctonos y platas_autoctonas
```

```
CREATE OR REPLACE FUNCTION eliminar_seres_vivos() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM animales_autoctonos
    WHERE ser_vivo_id = OLD.id_seres_vivos;
    DELETE FROM plantas_autoctonas
    WHERE ser_vivo_id = OLD.id_seres_vivos;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_seres_vivos
BEFORE DELETE ON seres_vivos
FOR EACH ROW
EXECUTE PROCEDURE eliminar_seres_vivos();
```

Tras la eliminación de una fila dentro de la tabla *seres_vivos* , se activa una función que permite eliminar las filas correspondientes en las tablas *animales_autoctonos* o *plantas_autoctonas*. Esto, permite mantener la integridad de los datos.

- **Trigger 19:**

```
-- -- Trigger para la tabla de seres_vivos
-- -- Comprobacion de que el nombre del ser vivo no se repita
CREATE OR REPLACE FUNCTION comprobar_nombre_seres_vivos() RETURNS TRIGGER AS $$
DECLARE
    nombre_seres_vivos RECORD;
BEGIN
    FOR nombre_seres_vivos IN SELECT * FROM seres_vivos
    LOOP
        IF nombre_seres_vivos.nombre = NEW.nombre THEN
            RAISE EXCEPTION 'El nombre del ser vivo ya existe';
        END IF;
    END LOOP;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER comprobar_nombre_seres_vivos
BEFORE INSERT OR UPDATE ON seres_vivos
FOR EACH ROW
EXECUTE PROCEDURE comprobar_nombre_seres_vivos();
```

El trigger se activa antes de insertar o actualizar una fila de la tabla *seres_vivos*. La función se encarga de verificar que el nombre del ser vivo que se está intentando insertar o actualizar no se encuentre presente dentro de la tabla. Si el nombre se encuentra duplicado se lanza una excepción por parte de la función con el mensaje `El nombre del ser vivo ya existe`.

5. Consultas de Ejemplo

5.1. Consultas SQL

Ejemplos de consultas que demuestren el funcionamiento de la base de datos.

6. Implementación de API con Flask

6.1. API REST

Desarrollo de una API mediante Flask para realizar operaciones CRUD.

7. Entrega

7.1. Repositorio en GitHub

Enlace al Repositorio: `https://github.com/feichay10/Proyecto-Final-ADBD`

7.2. Imágenes Adjuntas

Modelo Entidad-Relación, Grafo Relacional y capturas de consultas y operaciones en las tablas.

8. Ejecución del Proyecto

En cuanto a la correcta ejecución del proyecto, en primer lugar, se debe de realizar la carga de la base de datos dentro de *PostgreSQL*. Para ello, se realiza la ejecución de los siguiente comandos, los cuales permiten ejecutar el script *canary_islands.sql* que contiene la implementación de la base de datos, junto con sus respectivas tablas, relaciones, triggers y datos.

```
$ sudo -i -u postgres            # Acceder a la cuenta de postgres
$ psql                           # Acceder a la consola de PostgreSQL
postgres=# \i canary_islands.sql    # Ejecutar el script
```

> **Nota:** Para la ejecución del script, se debe de tener en cuenta que el archivo *canary_islands.sql* debe de estar en la misma carpeta que la consola de *PostgreSQL*. O en su defecto, se puede hacer uso de la ruta absoluta del archivo.

Una vez se tiene cargada la base de datos dentro de PostgreSQL, se procede a la ejecución de la API, la cuál permite realizar una serie de operaciones con la propia base de datos de forma remota. Para ello, se debe de realizar la modificación del script de la API, de tal manera que se puedan establecer los parámetros de conexión con la base de datos, de manera correcta, siendo adaptados a la configuración de la propia base de datos local de cada usuario.

```
# Parámetros de conexión con la base de datos
def get_db_connection():
conn = psycopg2.connect(host='localhost',
database="islas_canarias",
user="postgres",
password="[clave]")
return conn
```

Tras la modificación de dichos parámetros dentro del script, se procede a la ejecución de la API, la cuál se realiza mediante el siguiente comando:

```
$ flask --app app.py run --host 0.0.0.0 --port 8080   # Ejecución de la API
```

9. Bibliografía

Bibliografía

[1] https://www.canaryislands.org/