PROYECTO PARTE 2

AUTORES

SEBASTIAN CORDOBA

SANTIAGO RUEDA PINEDA

CARLOS ERAZO

JUAN MIGUEL ZULUAGA

DOCENTE

LUIS GABRIEL MORENO SANDOVAL

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD INGENIERÍA DE SISTEMAS

BASES DE DATOS

BOGOTA D.C

2023

TABLA DE CONTENIDO

1. INTRODUCCIÓN

El presente proyecto el cual está basado en GBIF (Global Biodiversity Information Facility) consiste en implementar cámaras trampa manteniendo de este modo cautela para así no perturbar el ambiente. Esto, con el fin de capturar a los diferentes y diversos organismos en cada una de sus hábitats para así poder contribuir a mejorar la calidad y la cantidad de los datos de biodiversidad disponibles.

Por otro lado, para la realización del presente proyecto es de vital importancia saber que es GBIF. La organización intergubernamental se define como “El Sistema Global de Información sobre Biodiversidad–GBIF por sus siglas en inglés–es una red internacional e infraestructura de datos financiada por los gobiernos del mundo para dar a cualquiera, en cualquier lugar, acceso abierto a datos sobre todas las formas de vida en la Tierra”.

Es decir, la importancia de la plataforma GBIF radica en que debido a sus características, es quien proporciona los datos sobre la biodiversidad, incluyendo información sobre especies, localizaciones geográficas, registros de ocurrencia y colecciones de especímene posibilitando de esta manera el desarrollo e implementación del proyecto.

2. FORMULACIÓN DEL PROBLEMA

Si bien es cierto existe gran variedad de información respecto a los diferentes organizamos en cada una de sus habitats, se requiere de la constante recopilación de información para que la plataforma GBIF pueda seguir actualizando sus datos y de este modo brindarlos a cualquier persona que los necesite para la educación, para la investigación y para la toma de decisiones que requieran de esta informacion en el ámbito de la conservación de la biodiversidad y la sostenibilidad de los recursos naturales.

3. DESARROLLO

Para el desarrollo e implementación del proyecto se propuso una serie de consultas para así poder interactuar con las relaciones propuestas. Cabe resaltar que para llevar a cabo dicho proceso se requirió de los subconjuntos de SQL, es decir, sus respectivos DDL(s) y DML(s).

El proyecto está compuesto de tres partes principales las cuales son: Creación de relaciones e inserts, consultas y finalmente la respectiva aplicación.

1. **Creacion de relaciones:**

   En este paso de desarrollo se observó que: Existen animales que deben ser captados en lugares específicos donde se repartiran las medios trampa, por ello es necesario crear una relación de Deployment.

   La tabla Deplyment está compuesta por deploymentID de tipo number, donde deploymentID es una llave subrogada (la primary key de la tabla Deplyment), luego está Time de tipo TIMESATMAP para guardar la hora y día donde se tomó (fecha), después está Location de tipo varchar2 para obtener la ubicación, luego está CameraDetails de tipo varchar2 para los detalles que son de la cámara como por ejemplo los diferentes tipos de cámara (sony, canon, nikon). Posteriormente está Featuretype de tipo varchar2 para saber el mecanismo que se utilizó para capturar la foto (trade cámara o cámara trampa), luego está el hábitat de tipo varchar2 que como su nombre dice, es para guardar el hábitat (urban o forest), después se encuentra setupBy de tipo varchar2 que indica la persona que colocó la cámara y finalmente el baitUse de tipo number para saber si se utilizó o no cebo (1 para indicar que si, 0 para indicar que no).

```
-- Crear tabla Deployment
CREATE TABLE Deployment (
  deploymentID NUMBER(10) PRIMARY KEY,
  Time TIMESTAMP,
  Location VARCHAR2(50),
  CameraDetails VARCHAR2(100),
  featuretype VARCHAR2(50),
  habitat VARCHAR2(50),
  setupBy VARCHAR2(50),
  baitUse NUMBER(1,0)
```

En este segundo paso de desarrollo se observó que: Existen animales que son captados por algun medio, por lo que es necesario crear una relación de Media.

Para la relación Media hay una llave foránea la cual es deploymentID, luego está mediaID de tipo number que es la llave subrogada (la primary key de la tabla Media), luego está el sequenceID de tipo number que indica si es foto o video (1 si es foto o 2 si es video), después está mediaType de tipo varchar2, posteriormente está captureMethod de tipo varchar2 que indica si es foto o video, luego está Time de tipo TIMESATMAP para guardar la hora y día donde se tomó (fecha), finalmente está MediaDetails de tipo varchar2 para saber el iso y focal de una cámara.

```
-- Crear tabla Media
CREATE TABLE Media (
  deploymentID NUMBER(10),
  mediaID NUMBER(10),
  sequenceID NUMBER(10),
  mediaType VARCHAR2(50),
  captureMethod VARCHAR2(50),
  Time TIMESTAMP,
  MediaDetails VARCHAR2(100),
  PRIMARY KEY (mediaID),
  CONSTRAINT fk_Deployment FOREIGN KEY (deploymentID) REFERENCES Deployment(deploymentID)
);
```

En el último paso de desarrollo se observó que: Existen animales que son captados por algun medio, por lo que es necesario crear una relación de observación.

Para la relación observation encontramos las llaves foráneas deploymentID y mediaID de tipo number, luego está observationID de tipo number que es la llave subrogada (la primary key de la tabla observationID), luego esta observationType de tipo varchar2 que captura wetland, landscape, animal (es decir, que tipo de observaciones se utilizaron), después el Time que como bien se sabe es la fecha, posteriormente IdentificationDetails de tipo varchar2 que indica si los animales están identificados o no, luego está TaxonDetails de tipo varchar2 que consiste en decir la manera taxonómica de describir los organismos (si empieza por A- se sabe que es animal, si empieza por una P- se sabe que es planta), después está el género de tipo char (M si es hombre, F si es mujer), y finalmente encontramos OrganismDetails de tipo varchar2 que es para saber información extra.

```
-- Crear tabla Observation
CREATE TABLE Observation (
  observationID NUMBER(10),
  mediaID NUMBER(10),
  deploymentID NUMBER(10),
  observationType VARCHAR2(50),
  Time TIMESTAMP,
  IdentificationDetails VARCHAR2(100),
  TaxonDetails VARCHAR2(100),
  Gender CHAR(1),
  OrganismDetails VARCHAR2(100),
  PRIMARY KEY(observationID),
  CONSTRAINT fk_Deployment_Observation FOREIGN KEY (deploymentID) REFERENCES Deployment(deploymentID),
  CONSTRAINT fk_Media_Observation FOREIGN KEY (mediaID) REFERENCES Media(mediaID)
);
```

La tabla Event es fundamental en el contexto de la observación de animales, ya que permite almacenar información detallada sobre los eventos ocurridos. Cada registro en esta tabla representa un evento específico y se identifica de manera única a través del campo eventID. En ella se registra

el tipo de evento, la fecha en que ocurrió, la ubicación donde se llevó a cabo y el tipo de hábitat relacionado.

En cuanto a los atributos, eventID se utiliza como clave primaria y es fundamental para identificar de manera única cada evento. eventType describe el tipo de evento registrado, mientras que eventDate almacena la fecha en que tuvo lugar. locationID guarda el identificador de la ubicación donde ocurrió el evento, y habitatType especifica el tipo de hábitat asociado a dicha ubicación.

```
CREATE TABLE Event (
    eventID VARCHAR(20) PRIMARY KEY,
    eventType VARCHAR(20) CHECK (eventType IN ('animal', 'human', 'vehicle', 'blank', 'unknown', 'unclassified')),
    parentEventID VARCHAR(20),
    eventDate VARCHAR(50) NOT NULL,
    locationID VARCHAR(20)
);
```

La tabla Tramp almacena datos específicos relacionados con la instalación de cámaras trampa. Cada registro en esta tabla tiene un identificador único eventID y registra información sobre la persona responsable de la identificación identifiedBy, el protocolo de muestreo utilizado samplingProtocol, observaciones adicionales de identificación identificationRemarks, los países involucrados country y publishingCountry, y detalles relacionados con el modelo de camara utilizado en la trampza cameraInstallDetails.

```
CREATE TABLE Tramp (
    eventID VARCHAR(20) PRIMARY KEY,
    identifiedBy VARCHAR(20),
    samplingProtocol VARCHAR(20),
    identificationRemarks VARCHAR(20),
    country VARCHAR(50),
    publishingCountry VARCHAR(50)

);
```

La tabla Entity se utiliza para almacenar información detallada sobre diferentes entidades, como animales o plantas. Cada entidad se identifica de manera única mediante el campo entityID y se registra su tipo (entityType), fecha de creación (entityCreated), nombre común (commonName) y nombre científico (scientificName). Estos atributos permiten categorizar y describir de manera precisa cada entidad, lo que facilita su seguimiento, análisis y gestión. La tabla Entity es fundamental para mantener un registro completo y estructurado de diversas entidades, brindando información clave para su identificación y estudio.

```
CREATE TABLE Entity (
    entityID VARCHAR(20) PRIMARY KEY,
    entityType VARCHAR(20),
    entityCreated DATE NOT NULL
);
```

La tabla DigitalEntity se utiliza para almacenar información sobre entidades digitales asociadas a las entidades registradas en la tabla Entity. Cada entidad digital tiene un identificador único (digitalEntityID) y se registra su tipo (digitalEntityType), formato (formato), URL de acceso (accessURI) y el ID de la entidad correspondiente (entityID).

El atributo entityID establece una relación con la tabla Entity, permitiendo vincular cada entidad digital con su entidad correspondiente. Esto facilita la gestión y el acceso a las entidades digitales relacionadas, ya que se puede acceder directamente a través de la URL de acceso.

```
CREATE TABLE DigitalEntity (
    digitalEntityID VARCHAR(20) PRIMARY KEY,
    digitalEntityType VARCHAR(20),
    format VARCHAR(20),
    accessURI VARCHAR(50) NOT NULL
);
```

La tabla Organism se utiliza para almacenar información sobre organismos. Cada organismo tiene un identificador único (organismID) y se registra su alcance (organismScope), sexo (sex) y país (country).

El atributo organismID se utiliza como clave primaria para identificar de manera única cada organismo en la tabla. El atributo organismScope indica el alcance del organismo, como especie o subespecie. El atributo sex registra el sexo del organismo, mientras que el atributo country almacena el país donde se encuentra el organismo

```
CREATE TABLE Organism (
    organismID VARCHAR(20) PRIMARY KEY,
    organismScope VARCHAR(20) CHECK (organismScope IN ('adult', 'subadult', 'juvenile')),
    sex VARCHAR(20)
);
```

La tabla EntityEvent se utiliza para establecer la relación entre entidades y eventos en un contexto determinado. Cada registro en esta tabla vincula un identificador de entidad (entityID) con un identificador de evento (eventID). Se establecen claves externas para mantener la integridad referencial con las tablas Entity y Event.

El atributo entityID es una clave externa que referencia un identificador de entidad en la tabla Entity, mientras que el atributo eventID es una clave externa que referencia un identificador de evento en la tabla Event. Esto permite asociar entidades específicas con eventos relacionados.

```
CREATE TABLE EntityEvent (
    entityID VARCHAR(20),
    eventID VARCHAR(20),
    FOREIGN KEY (entityID) REFERENCES Entity(entityID),
    FOREIGN KEY (eventID) REFERENCES Event(eventID)
);
```

La tabla EntityAssertion se utiliza para almacenar información relacionada con las afirmaciones asociadas a entidades específicas. Cada registro en esta tabla tiene un identificador único (entityAssertionID) y se registra el identificador de la entidad relacionada (entityID). Además, se registran el tipo de afirmación (entityAssertionType), el valor de la afirmación (entityAssertionValue) y la unidad de medida utilizada (entityAssertionUnit).

El atributo entityID establece una clave externa que referencia un identificador de entidad en la tabla Entity, lo que permite asociar cada afirmación con la entidad correspondiente.

```
CREATE TABLE EntityAssertion (
    entityAssertionID VARCHAR(20) PRIMARY KEY,
    entityID VARCHAR(20),
    entityAssertionType VARCHAR(20),
    entityAssertionValue VARCHAR(20),
    entityAssertionUnit VARCHAR(20),
    FOREIGN KEY (entityID) REFERENCES Entity(entityID)
);
```

La tabla Identification se utiliza para almacenar información relacionada con las identificaciones de entidades. Cada registro en esta tabla tiene un identificador único (identificationID) y se registra el tipo de identificación (identificationType). Además, se almacena la fórmula taxonómica (taxaFormula) y la identificación literal (verbatimIdentification).

El identificationID se utiliza como clave primaria para identificar de manera única cada identificación en la tabla. El identificationType indica el tipo de identificación realizada, como especie, género o familia. La taxaFormula representa la clasificación taxonómica precisa de la entidad identificada, mientras que la verbatimIdentification almacena la identificación literal tal como se registró.

```
CREATE TABLE Identification (
    identificationID VARCHAR(20) PRIMARY KEY,
    identificationType VARCHAR(20),
    taxaFormula VARCHAR(20),
    verbatimIdentification VARCHAR(50)
);
```

La tabla IdentificationEntity se utiliza para establecer la relación entre las identificaciones y las entidades a las que pertenecen. Cada registro en esta tabla vincula un identificador de identificación (identificationID) con un identificador de entidad (entityID). Se establecen claves externas para mantener la integridad referencial con las tablas Identification y Entity.

El atributo identificationID es una clave externa que referencia un identificador de identificación en la tabla Identification, mientras que el atributo entityID es una clave externa que referencia un identificador de entidad en la tabla Entity. Esto permite asociar cada identificación con la entidad correspondiente.

```
CREATE TABLE IdentificationEntity (
    identificationID VARCHAR(20),
    entityID VARCHAR(20),
    FOREIGN KEY (identificationID) REFERENCES Identification(identificationID),
    FOREIGN KEY (entityID) REFERENCES Entity(entityID)
);
```

La tabla Taxon se utiliza para almacenar información sobre las diferentes categorías en la taxonomía biológica. Cada registro en esta tabla tiene un identificador único (taxonID) y se registra el reino al que pertenece la categoría (kingdom) y su nombre científico (scientificName).

El atributo taxonID se utiliza como clave primaria para identificar de manera única cada categoría en la tabla. El campo kingdom indica el reino al que pertenece la categoría, como

Animalia o Plantae. El campo scientificName almacena el nombre científico específico de la categoría según las convenciones de la nomenclatura biológica.

```
CREATE TABLE Taxon (
    taxonID VARCHAR(20) PRIMARY KEY,
    kingdom VARCHAR(20),
    scientificName VARCHAR(50)
);
```

La tabla TaxonIdentification se utiliza para establecer la relación entre los taxones y las identificaciones realizadas en relación con ellos. Cada registro en esta tabla vincula un identificador de taxón (taxonID) con un identificador de identificación (identificationID). Además, se registra el orden taxonómico (taxonOrder) del taxón dentro de la identificación.

El atributo taxonID es una clave externa que referencia un identificador de taxón en la tabla Taxon, mientras que el atributo identificationID es una clave externa que referencia un identificador de identificación en la tabla Identification. Esto permite asociar cada identificación con los taxones correspondientes y mantener la integridad referencial.

```
CREATE TABLE TaxonIdentification (
    taxonID VARCHAR(20),
    identificationID VARCHAR(20),
    taxonOrder INTEGER CHECK (taxonOrder >= 1),
    FOREIGN KEY (taxonID) REFERENCES Taxon(taxonID),
    FOREIGN KEY (identificationID) REFERENCES Identification(identificationID)
);
```

1. **Inserts:**

   A continuación se adjuntan algunos ejemplos de los inserts que sean correspondientes a cada entidad.

En cuanto a la relación Deployment se adjuntan algunos ejemplos de sus respectivos inserts:

```
-- Insertar datos en tabla Deployment
INSERT INTO Deployment (deploymentID, Time, Location, CameraDetails, featuretype, habitat, setupBy, baitUse)
VALUES (1, TO_TIMESTAMP('2022-03-15 10:30:00', 'YYYY-MM-DD HH24:MI:SS'), 'Central Park', 'Nikon 35mm, tripod', 'Wildlife monitoring', 'Forest', 'Jakub Bubnicki', 1);
INSERT INTO Deployment (deploymentID, Time, Location, CameraDetails, featuretype, habitat, setupBy, baitUse)
VALUES (2, TO_TIMESTAMP('2022-03-18 14:45:00', 'YYYY-MM-DD HH24:MI:SS'), 'Yosemite National Park', 'Sony A7III, monopod', 'Landscape photography', 'Mountains', 'Jakub Bubı
INSERT INTO Deployment (deploymentID, time, location, cameraDetails, featuretype, habitat, setupBy, baitUse)
VALUES (3, TO_TIMESTAMP('2022-02-15 08:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Central Park', 'Sony A6400', 'Trail Camera', 'Forest', 'Jakub Bubnicki', 1);
INSERT INTO Deployment (deploymentID, time, location, cameraDetails, featuretype, habitat, setupBy, baitUse)
VALUES (4, TO_TIMESTAMP('2022-03-01 12:30:00', 'YYYY-MM-DD HH24:MI:SS'), 'Yellowstone National Park', 'Canon EOS R', 'Camera Trap', 'Grassland', 'Jakub Bubnicki', 1);
```

De igual manera para la relación Media se adjuntan algunos ejemplos de sus inserts:

```
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(4, 6, 1, 'photo', 'automatic', TO_TIMESTAMP('2022-02-01 12:30:00', 'YYYY-MM-DD HH24:MI:SS'), 'Sunny day, high humidity');
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(4, 7, 1, 'photo', 'automatic', TO_TIMESTAMP('2022-02-01 12:35:00', 'YYYY-MM-DD HH24:MI:SS'), 'Cloudy day, low humidity');
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(4, 8, 2, 'video', 'manual', TO_TIMESTAMP('2022-02-03 14:20:00', 'YYYY-MM-DD HH24:MI:SS'), 'Overcast day, medium humidity');
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(5, 9, 1, 'photo', 'automatic', TO_TIMESTAMP('2022-02-05 09:10:00', 'YYYY-MM-DD HH24:MI:SS'), 'Sunny day, low humidity');
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(5, 10, 2, 'video', 'manual', TO_TIMESTAMP('2022-02-06 16:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Overcast day, high humidity');
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(6, 11, 1, 'photo', 'automatic', TO_TIMESTAMP('2022-02-07 10:30:00', 'YYYY-MM-DD HH24:MI:SS'), 'Sunny day, medium humidity');
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(7, 12, 1, 'photo', 'automatic', TO_TIMESTAMP('2022-02-09 12:20:00', 'YYYY-MM-DD HH24:MI:SS'), 'Sunny day, high humidity');
INSERT INTO Media (deploymentID, mediaID, sequenceID, mediaType, captureMethod, Time, MediaDetails) VALUES
(8, 13, 2, 'video', 'manual', TO_TIMESTAMP('2022-02-11 14:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Overcast day, low humidity');
```

Para la relación Media se adjuntan algunos ejemplos de sus respectivos inserts:

```
-- Insertar datos en tabla Observation
INSERT INTO Observation (observationID, mediaID, deploymentID, observationType, Time, IdentificationDetails, TaxonDetails, OrganismDetails)
VALUES (1, 1, 1, 'Wildlife', TO_TIMESTAMP('2022-03-16 13:15:00', 'YYYY-MM-DD HH24:MI:SS'), 'Brown fur, white spots', 'Ursus americanus', 'Adult male bear');
INSERT INTO Observation (observationID, mediaID, deploymentID, observationType, Time, IdentificationDetails, TaxonDetails, OrganismDetails)
VALUES (2, 2, 2, 'Landscape', TO_TIMESTAMP('2022-03-20 11:00:00', 'YYYY-MM-DD HH24:MI:SS'), NULL, 'Yosemite Valley', 'Granite cliffs, waterfalls');
INSERT INTO Observation (observationID, mediaID, deploymentID, observationType, time, identificationDetails, taxonDetails, Gender ,organismDetails)
VALUES (3, 3, 3, 'Animal', TO_TIMESTAMP('2022-02-17 10:15:00', 'YYYY-MM-DD HH24:MI:SS'), 'Mammal', 'Canidae', 'F','Canis lupus');
INSERT INTO Observation (observationID, mediaID, deploymentID, observationType, time, identificationDetails, taxonDetails, organismDetails)
VALUES (4, 4, 4, 'Plant', TO_TIMESTAMP('2022-03-02 14:25:00', 'YYYY-MM-DD HH24:MI:SS'), 'Flower', 'Rosaceae', 'Rosa canina');
INSERT INTO Observation (observationID, mediaID, deploymentID, observationType, time, identificationDetails, taxonDetails, Gender ,organismDetails)
VALUES (5, 5, 5, 'Animal', TO_TIMESTAMP('2022-03-06 07:30:00', 'YYYY-MM-DD HH24:MI:SS'), 'Bird', 'Accipitridae', 'M' ,'Buteo jamaicensis');

INSERT INTO Observation VALUES (8, 10, 9, 'sighting', TO_TIMESTAMP('2022-02-11 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Canis latrans', 'M','coyote');
INSERT INTO Observation VALUES (9, 11, 9, 'sighting', TO_TIMESTAMP('2022-02-11 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Canis latrans', 'F','coyote');
INSERT INTO Observation VALUES (10, 12, 9, 'sighting', TO_TIMESTAMP('2022-02-11 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Canis latrans', 'M','coyote');
INSERT INTO Observation VALUES (11, 13, 9, 'sighting', TO_TIMESTAMP('2022-02-11 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Canis latrans', 'F','coyote');
INSERT INTO Observation VALUES (12, 6, 4, 'Animal', TO_TIMESTAMP('2022-02-11 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Odocoileus virginianus', 'M','white-taile
INSERT INTO Observation VALUES (13, 7, 5, 'Animal', TO_TIMESTAMP('2022-02-11 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Odocoileus virginianus', 'F','white-taile
INSERT INTO Observation VALUES (14, 8, 11, 'sighting', TO_TIMESTAMP('2022-02-13 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Geothlypis trichas', 'M','common yello
INSERT INTO Observation VALUES (15, 9, 11, 'sighting', TO_TIMESTAMP('2022-02-13 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Geothlypis trichas', 'F','common yello
INSERT INTO Observation VALUES (16, 10, 11, 'sighting', TO_TIMESTAMP('2022-02-13 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Geothlypis trichas', 'M','common yell
INSERT INTO Observation VALUES (17, 11, 11, 'sighting', TO_TIMESTAMP('2022-02-13 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Geothlypis trichas', 'M','common yell
INSERT INTO Observation VALUES (18, 12, 12, 'sighting', TO_TIMESTAMP('2022-02-13 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Rana clamitans', 'F','green frog');
INSERT INTO Observation VALUES (19, 13, 12, 'sighting', TO_TIMESTAMP('2022-02-13 13:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'identified', 'Rana clamitans', 'M','green frog');
```

Se hace lo mismo para los demás inserts

Entity:

```
-- Insert 1
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (1, 'Type 1', 'Remarks 1');

-- Insert 2
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (2, 'Type 2', 'Remarks 2');

-- Insert 3
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (3, 'Type 3', 'Remarks 3');

-- Insert 4
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (4, 'Type 4', 'Remarks 4');

-- Insert 5
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (5, 'Type 5', 'Remarks 5');

-- Insert 6
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (6, 'Type 6', 'Remarks 6');

-- Insert 7
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (7, 'Type 7', 'Remarks 7');

-- Insert 8
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (8, 'Type 8', 'Remarks 8');

-- Insert 9
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (9, 'Type 9', 'Remarks 9');

-- Insert 10
INSERT INTO Entity (entityID, entityType, entityRemarks)
VALUES (10, 'Type 10', 'Remarks 10');
```

EntityRelationship:

```
-- Insert 1
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (1, 1, 1, 'Type 1', 1, SYSDATE);

-- Insert 2
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (2, 2, 2, 'Type 2', 2, SYSDATE);

-- Insert 3
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (3, 3, 3, 'Type 3', 3, SYSDATE);

-- Insert 4
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (4, 4, 4, 'Type 4', 4, SYSDATE);

-- Insert 5
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (5, 5, 5, 'Type 5', 5, SYSDATE);

-- Insert 6
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (6, 6, 6, 'Type 6', 6, SYSDATE);

-- Insert 7
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (7, 7, 7, 'Type 7', 7, SYSDATE);

-- Insert 8
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (8, 8, 8, 'Type 8', 8, SYSDATE);

-- Insert 9
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (9, 9, 9, 'Type 9', 9, SYSDATE);

-- Insert 10
INSERT INTO EntityRelationship (entityRelationshipID, dependsOnEntityRelationshipID, subjectEnityID, entityRelationShipType, objectEntityID, entityRelationshipDate)
VALUES (10, 10, 10, 'Type 10', 10, SYSDATE);
```

Location:

```
-- Insert 1
INSERT INTO Location (locationID, parentLocationID)
VALUES (1, 1);
```

Protocol:

```
-- Insert 10 rows into Protocol table
INSERT INTO Protocol (protocolID, protocolType)
VALUES (1, 'Type 1');
```

Event table:

```
-- Insert 10 rows into Event table
INSERT INTO Event (locationID, eventID, parentEventID, eventType, protocolID)
VALUES (1, 1, 1, 'Type 1', 1);
```

entity event:

```
-- Insert 10 rows into EntityEvent table
INSERT INTO EntityEvent (EntityID, EventID)
VALUES (1, 1);
```

event:

```
-- Insert 10 rows into Event table
INSERT INTO Event (locationID, eventID, parentEventID, eventType, protocolID)
VALUES (1, 1, 1, 'Type 1', 1);
```

identification:

```
-- Insert 10 rows into Identification table
INSERT INTO Identification (identificationID, identificationTpe, taxoFormula)
VALUES (1, 'Type 1', 'Formula 1');
```

identification entity:

```
-- Insert 10 rows into IdentificationEntity table
INSERT INTO IdentificationEntity (IdentificationID, EntityID)
VALUES (1, 1);
```

Taxon Table

```
-- Insert 10 rows into Taxon table
INSERT INTO Taxon (TaxonID, parentTaxonID)
VALUES (1, 1);
```

2. Consultas:

1. Consultas (vistas):

Vista Punto1: Esta vista proporciona un recuento de los distintos animales observados en cada localización de implementación. Aquí se unen tres tablas: "Deployment", "Media" y "Observation". La vista se agrupa por locationID de la tabla "Deployment", y luego cuenta la cantidad única de observationID de la tabla "Observation". El resultado es una lista de ubicaciones y la cantidad de animales distintos observados en cada una.

Vista Punto2: La vista 'punto2' tiene como objetivo proporcionar información detallada sobre las instalaciones de las cámaras trampa. Une las tablas "Observation", "Deployment" y "Media" para mostrar quién instaló cada trampa (setupBy), qué animales fueron capturados (taxondetails y OrganismDetails), y qué tipo de medios se utilizó (mediatype). Por lo tanto, proporciona un informe completo sobre las trampas, los animales observados y los medios utilizados.

Vista Punto3: La vista 'punto3' se utiliza para verificar información específica sobre las trampas de cámaras. Proporciona información sobre el hábitat donde se instaló la cámara,

el uso del cebo y otros detalles de la cámara. Los datos para esta vista provienen exclusivamente de la tabla "Deployment", pero solo para los registros donde featuretype es 'Camera Trap'.

Vista Punto4: La vista 'punto4' proporciona información sobre cuántos medios se han registrado para cada animal observado. Une las tablas "Observation", "Deployment" y "Media", y cuenta el número de mediaID para cada detalle de organismo único (OrganismDetails). Esto podría ser útil para entender qué animales se capturan con mayor frecuencia en los medios.

Vista Punto5: La vista 'punto5' lista todos los animales que han sido observados en más de una ubicación. Une las tablas "Observation" y "Deployment", agrupa los datos por TaxonDetails y lista todas las ubicaciones donde se ha observado cada taxón. Los taxones que han sido observados en más de una ubicación se incluyen en la vista.

Vista Punto6: La vista 'punto6' muestra la frecuencia de aparición de cada animal en cada ubicación. Une las tablas "Observation", "Deployment" y "Media", y luego agrupa los datos por taxondetails y location. Para cada combinación única de taxón y ubicación, cuenta el número de ubicaciones distintas. El resultado es una lista de animales, ubicaciones y la frecuencia con la que cada animal aparece en cada ubicación.

Vista Punto7: Esta vista se utiliza para determinar qué especies han sido capturadas por todas las cámaras trampa y cuántas veces se observaron. Une las tablas "Observation", "Media" y "Deployment" y agrupa los datos por el nombre científico de la especie. Después de agrupar, la vista filtra los datos para solo incluir especies que se han observado en todas

las cámaras trampa (determinado por el conteo de deploymentID distintos igual a la cantidad total de implementaciones).

Vista Punto8: La vista 'punto8' proporciona un recuento de los organismos registrados en la base de datos por sexo. Une las tablas "Organism", "Entity" y "EntityEvent", y cuenta la cantidad de cada sexo que aparece en la tabla "Organism". El resultado es un desglose de cuántos organismos de cada sexo se han registrado.

Vista Punto9: La vista 'punto9' clasifica las entidades en la base de datos en "Animal" y "No Animal", basándose en el valor del campo entityType en la tabla "Entity". Esto puede ser útil para obtener una visión general de la cantidad de registros de animales en comparación con otros tipos de entidades.

2. PL/SQL:

Funciones:

El procedimiento almacenado "update_media" tiene como objetivo actualizar el campo "filePath" de la tabla "Media" con el valor proporcionado para un determinado "mediaID". A continuación, se proporciona una explicación paso a paso del procedimiento:

El procedimiento se define utilizando la declaración "CREATE OR REPLACE PROCEDURE" seguida del nombre del procedimiento, en este caso, "update_media". Los parámetros del procedimiento se definen entre paréntesis, en este caso, "p_mediaID" y "p_filePath", seguidos de sus respectivos tipos de datos.

La sección "IS" indica el inicio del cuerpo del procedimiento.

Dentro del cuerpo del procedimiento, se utiliza la instrucción "UPDATE" para modificar los datos en la tabla "Media". Se establece el valor del campo "filePath" con el valor proporcionado en el parámetro "p_filePath" para la fila que coincide con el "mediaID" proporcionado en el parámetro "p_mediaID".

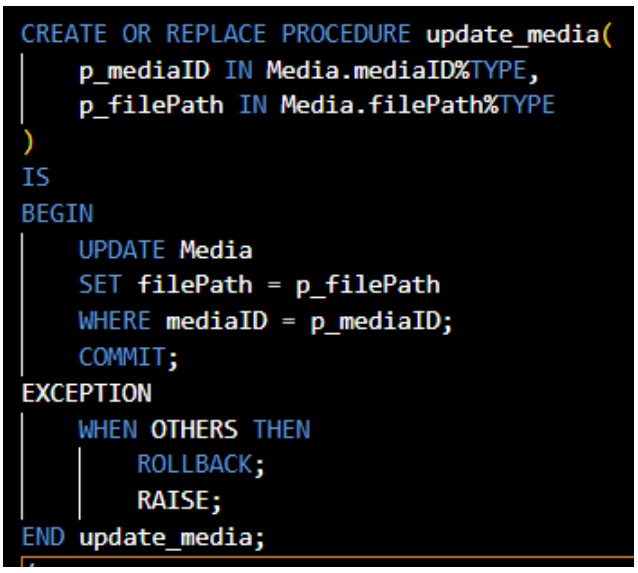
Después de la instrucción "UPDATE", se utiliza la instrucción "COMMIT" para confirmar los cambios realizados en la tabla.

La sección "EXCEPTION" indica que se manejarán las excepciones que puedan ocurrir durante la ejecución del procedimiento.

Dentro del bloque "EXCEPTION", se utiliza la instrucción "ROLLBACK" para deshacer cualquier cambio realizado en caso de que se produzca una excepción.

La instrucción "RAISE" se utiliza para relanzar la excepción y propagarla a un nivel superior en caso de que ocurra un error durante la ejecución del procedimiento.

En resumen, el procedimiento "update_media" permite actualizar el campo "filePath" de la tabla "Media" para un determinado "mediaID" proporcionado como parámetro. Si se ejecuta exitosamente, los cambios se confirman con un "COMMIT". En caso de que se produzca un error, se realiza un "ROLLBACK" para deshacer los cambios y se relanza la excepción para su manejo adecuado.



```
CREATE OR REPLACE PROCEDURE update_media(
    p_mediaID IN Media.mediaID%TYPE,
    p_filePath IN Media.filePath%TYPE
)
IS
BEGIN
    UPDATE Media
    SET filePath = p_filePath
    WHERE mediaID = p_mediaID;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END update_media;
```

```
CREATE OR REPLACE FUNCTION count_animals_by_date(p_date IN DATE)
RETURN NUMBER IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM Observation o
    JOIN Media m ON o.mediaID = m.mediaID
    WHERE TRUNC(TO_DATE(m.timestamp, 'YYYY-MM-DD')) = p_date;

    RETURN v_count;
EXCEPTION
    WHEN OTHERS THEN
        RETURN NULL;
END count_animals_by_date;
```

La función "count_animals_by_date" es una función definida por el usuario en SQL que tiene como objetivo contar la cantidad de animales registrados en la tabla "Observation" para una fecha específica.

La función comienza declarando un parámetro de entrada "p_date" de tipo DATE, que representa la fecha para la cual se desea contar los animales. Luego, se declara una variable local llamada "v_count" de tipo NUMBER, que se utilizará para almacenar el resultado de la consulta COUNT(*), es decir, el número de animales contados.

Dentro del bloque BEGIN, se realiza una consulta utilizando la cláusula SELECT COUNT(*). La consulta se enfoca en las tablas "Observation" y "Media" y utiliza la cláusula JOIN para combinar los registros de ambas tablas. La condición de unión se establece utilizando el ID de los medios (mediaID), asegurando que solo se consideren las observaciones que tienen un medio correspondiente.

Además, se utiliza la función TRUNC junto con TO_DATE para convertir el campo "timestamp" de la tabla "Media" en un formato de fecha con el formato 'YYYY-

MM-DD'. Luego, se compara esta fecha truncada con el parámetro "p_date" para filtrar las observaciones y contar solo aquellas que coinciden con la fecha específica.

El resultado de la consulta COUNT(*) se asigna a la variable "v_count" utilizando la cláusula INTO. Finalmente, la función devuelve el valor de "v_count", representando la cantidad de animales registrados en la fecha proporcionada.

En caso de que ocurra alguna excepción durante la ejecución de la función, se utiliza la cláusula EXCEPTION para capturar la excepción y manejarla. En este caso, se devuelve un valor NULL, evitando que la función falle por completo y proporcionando un resultado predeterminado en caso de un error.

```
CREATE OR REPLACE PROCEDURE add_deployment(
    p_deploymentID IN Deployment.deploymentID%TYPE,
    p_start_time IN Deployment.start_time%TYPE,
    p_end_time IN Deployment.end_time%TYPE,
    p_locationID IN Deployment.locationID%TYPE
)
IS
BEGIN
    INSERT INTO Deployment(deploymentID, start_time, end_time, locationID)
    VALUES (p_deploymentID, p_start_time, p_end_time, p_locationID);
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END add_deployment;
/
```

El código proporcionado define un procedimiento almacenado llamado "add_deployment" en SQL. Este procedimiento se encarga de insertar un nuevo registro en la tabla "Deployment" utilizando los valores proporcionados como parámetros de entrada, tales como el ID de despliegue, la hora de inicio, la hora de finalización y la ID de ubicación. Después de la inserción, se realiza una confirmación para asegurar que los cambios sean guardados permanentemente en la base de datos. En caso de que ocurra alguna excepción durante la ejecución del

procedimiento, se realiza un rollback para deshacer los cambios y se propaga la excepción para su posterior manejo.

En resumen, este procedimiento proporciona una forma conveniente y segura de agregar nuevos registros a la tabla "Deployment" de manera controlada y consistente, asegurándose de mantener la integridad de los datos y manejar adecuadamente las posibles excepciones.

3. Indices:

CREATE INDEX idx_observation_time ON Observation (scientificName);

Este índice se crea en la tabla "Observation" y en la columna "scientificName". El propósito de este índice es mejorar la velocidad de búsqueda de observaciones basadas en el nombre científico. Al crear este índice, las consultas que involucran la columna "scientificName" en la tabla "Observation" se ejecutarán más rápido, ya que la base de datos puede buscar directamente en el índice en lugar de tener que recorrer todas las filas de la tabla.

CREATE INDEX IDX_EVENT_TIME ON Event (eventDate);

Este índice se crea en la tabla "Event" y en la columna "eventDate". El objetivo de este índice es mejorar la velocidad de búsqueda de eventos basados en la fecha del evento. Al crear este índice, las consultas que involucran la columna "eventDate"

en la tabla "Event" se ejecutarán más rápidamente, ya que la base de datos puede utilizar el índice para acceder rápidamente a las filas que cumplen con el rango de fechas especificado en la consulta.

CREATE INDEX IDX_ENTITY_TYPE ON Entity (entityType);

Este índice se crea en la tabla "Entity" y en la columna "entityType". El objetivo de este índice es mejorar la velocidad de búsqueda de entidades basadas en el tipo de entidad. Al crear este índice, las consultas que involucran la columna "entityType" en la tabla "Entity" se ejecutarán más rápidamente, ya que la base de datos puede utilizar el índice para encontrar rápidamente las filas que coinciden con el tipo de entidad especificado en la consulta.

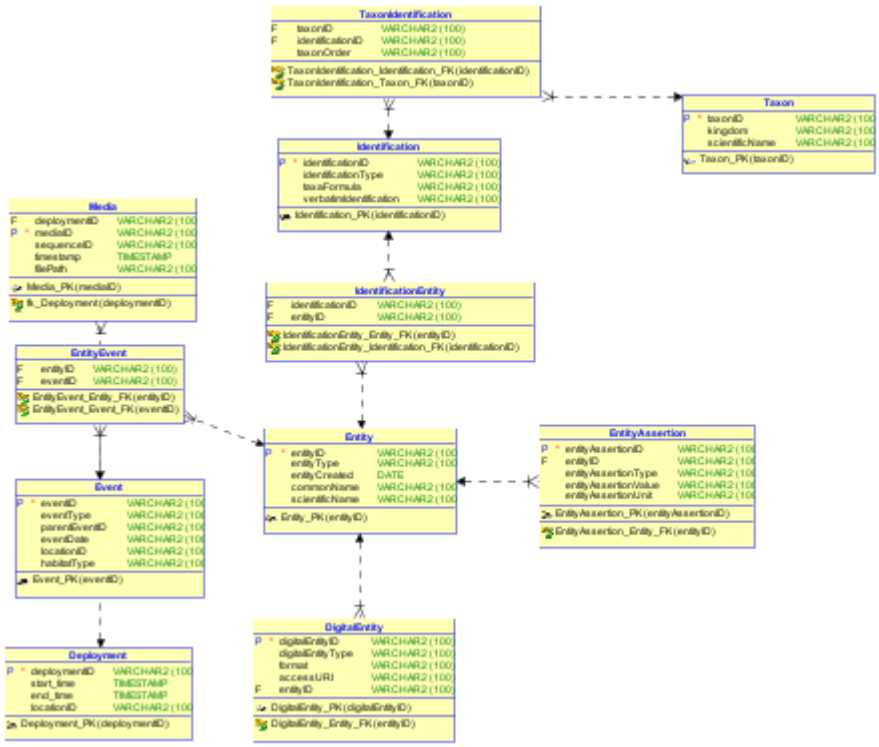
CREATE INDEX idx_identification_type ON Identification (identificationType);

Este índice se crea en la tabla "Identification" y en la columna "identificationType". El propósito de este índice es mejorar la velocidad de búsqueda de identificaciones basadas en el tipo de identificación. Al crear este índice, las consultas que involucran la columna "identificationType" en la tabla "Identification" se ejecutarán más rápidamente, ya que la base de datos puede utilizar el índice para acceder directamente a las filas que coinciden con el tipo de identificación especificado en la consulta.

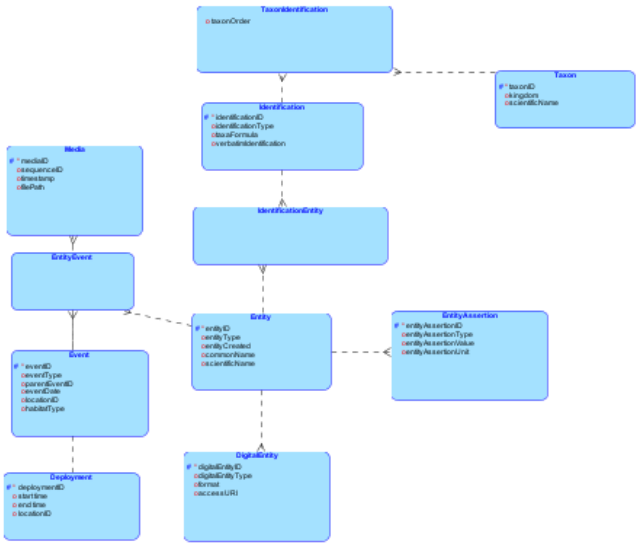
3. Modelos

A continuación, se adjuntan los respectivos modelos los cuales darán una mejor visión de las relaciones para la segunda parte del proyecto**.**

1. Modelo Lógico:



2. Modelo Físico:



4. Aplicación:

El proceso para cargar los datos inicia con el dataset Swedish Malaise Trap Project (Holston & Karlsson, 2023) El cual sigue los datos de las ocurrencias de insectos a lo largo de Suecia. El proceso a seguir consistía en leer los datos y buscar la mayor cantidad de atributos para poder satisfacer el sistema propuesto y para obtener más información hacer uso del API de GBIF. Una vez se pudo capturar toda la información posible, la forma de insertar los archivos consiste en declarar un diccionario con el contenido de la tabla para posteriormente con el uso de cx_oracle ejecutar un INSERT en sql. Mientras tanto el programa informa del total de datos agregados hasta el momento.

```
try:
    event_data = {
        'eventID': row['eventID'],
        'eventType': row['type'],
        'parentEventID': row['parentEventID'],
        'eventDate': row['eventDate'],
        'locationID': row['locationID']
    }
    connection.cursor().execute("INSERT INTO Event VALUES (:eventID, :eventType, :parentEventID, :eventDate, :locationID)", **event_data)
```

```
PS C:\Users\anes_\OneDrive\Documentos\2023-1\bd\Nueva carpeta\Proyecto> ^C
PS C:\Users\anes_\OneDrive\Documentos\2023-1\bd\Nueva carpeta\Proyecto> python .\data_uploading.py
DATOS AGREGADOS: 10/1211
DATOS AGREGADOS: 20/1211
DATOS AGREGADOS: 30/1211
DATOS AGREGADOS: 40/1211
DATOS AGREGADOS: 50/1211
DATOS AGREGADOS: 60/1211
DATOS AGREGADOS: 70/1211
DATOS AGREGADOS: 80/1211
DATOS AGREGADOS: 90/1211
DATOS AGREGADOS: 100/1211
DATOS AGREGADOS: 110/1211
DATOS AGREGADOS: 120/1211
DATOS AGREGADOS: 130/1211
DATOS AGREGADOS: 140/1211
```

Esta solución no se pudo llevar a cabo dado que no se logró recopilar toda la información necesaria para satisfacer todas las tablas del modelo. Por esta razón el equipo tomó la decisión de cargar datos genéricos con el fin de continuar con la operación de la aplicación.

Se usó una plantilla base de HTML donde se puede fijar el título, header y footer de la página. El header supone una navegación entre las tablas existentes (deploymnet, media, observation) y para la sección de “Home” se agrega un botón con el cual se hace posible la navegación entre las consultas propuestas en el proyecto.

```python
@app.route('/deployment')
def deployment():
    cursor.execute("SELECT * FROM Deployment")
    headers = [i[0] for i in cursor.description]
    rows = cursor.fetchall()

    return render_template('base.html', rows=rows, headers=headers)
```

```python
@app.route('/punto1')
def punto1():
    text = "Cantidad de tipo de animales capturado por locationName."
    cursor.execute("SELECT * FROM punto1")
    headers = [i[0] for i in cursor.description]
    rows = cursor.fetchall()

    return render_template('index.html', rows=rows, headers=headers, pag='punto1', opciones=opciones, text=text)
```

La estructura de las páginas que muestran las tablas son muy similares a la Figura 27. donde se hace una consulta a la base de datos para posteriormente cargar las tuplas y enviarlas al render. De igual forma las consultas no varían tanto, pero para este caso se manda un texto adicional para mostrar en pantalla, el estado actual y las posibles páginas en las que se puede navegar posteriormente como se puede ver en los parametros de render_tamplate(), esto es posible gracias al motor de renderizado que utiliza Flask que es llamado Jinja2. Jinja2 se utiliza para separar la lógica de presentación de los datos, lo que significa que el código que manipula los datos se separa de la presentación visual de los mismos.

```
<tr>
    {% for name in headers %}
    <th>{{name}}</th>
    {% endfor %}
</tr>
{% for row in rows %}
<tr>
    {% for data in row %}
    <td>{{data}}</td>
    {% endfor %}
</tr>
{% endfor %}
```

De esta forma, con el uso de Jinja2 es posible interactuar con datos dinámicos que se reciben desde la función, también permite el uso de bloques de contenido dando así paso al uso de ciclos o condicionales aumentando la eficiencia en el desarrollo de la aplicación.

Referencias

Holston, K. and Karlsson, D. (2023) *Swedish malaise trap project (SMTP) collection inventory*, *GBIF*. Available at: https://www.gbif.org/dataset/38c1351d-9cfe-42c0-97da-02d2c8be141c (Accessed: 18 May 2023).

Gbif Rest Api (no date) *GBIF REST API*. Available at: https://www.gbif.org/developer/summary (Accessed: 18 May 2023).