DATABASES DOCE

MYSQL - RELACIONAL

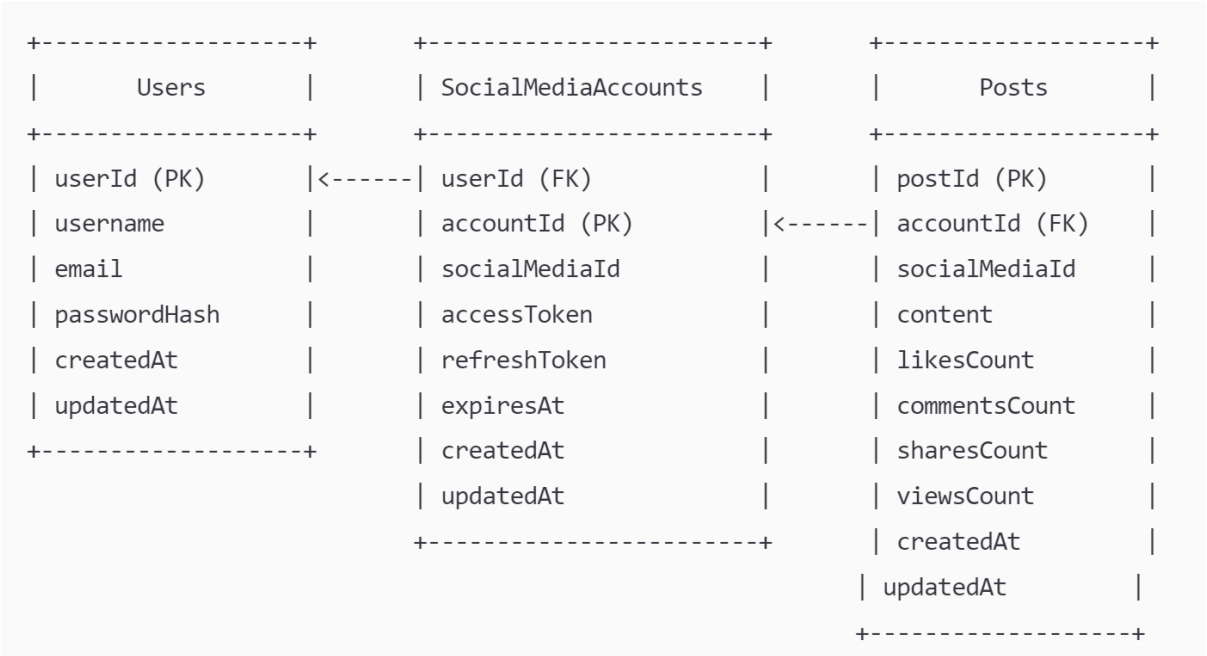
```
CREATE DATABASE DOCE;

USE DOCE;

CREATE TABLE Users (
    userId INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    passwordHash VARCHAR(255) NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);

CREATE TABLE SocialMediaAccounts (
    accountId INT AUTO_INCREMENT PRIMARY KEY,
    userId INT NOT NULL,
    socialMediaId VARCHAR(255) NOT NULL,
    accessToken VARCHAR(255) NOT NULL,
    refreshToken VARCHAR(255),
    expiresAt TIMESTAMP,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (userId) REFERENCES Users(userId)
);

CREATE TABLE Posts (
    postId INT AUTO_INCREMENT PRIMARY KEY,
    accountId INT NOT NULL,
    socialMediaId VARCHAR(255) NOT NULL,
    content TEXT,
    likesCount INT DEFAULT 0,
    commentsCount INT DEFAULT 0,
    sharesCount INT DEFAULT 0,
    viewsCount INT DEFAULT 0,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (accountId) REFERENCES SocialMediaAccounts(accountId)
);
```



Users: Almacena información sobre los usuarios.

SocialMediaAccounts: Almacena información sobre las cuentas de redes sociales asociadas a los usuarios.

Posts: Almacena información sobre las publicaciones realizadas en las cuentas de redes sociales.

MONGOATLAS - NO RELACIONAL

https://colab.research.google.com/drive/1oUc_r6EMUVogNarfi6XzaGvbMHT3wbJ5?usp=sharing

CRUD

```
from pymongo import MongoClient, errors
from bson.objectid import ObjectId
from datetime import datetime

class dataBaseConnection:
    def __init__(self, uri, database):
        self.uri = uri
        self.database = database

    def connection(self):
        try:
            # Establecer la conexión con la base de datos de MongoAtlas
            connectionDB = MongoClient(self.uri)
            # Seleccionar la base de datos de Mongo
            database = connectionDB[self.database]
            print(f"Conexión exitosa a la base de datos: {self.database}")
            return database
        except errors.ConnectionFailure as e:
            print(f"Error en conexión de base de datos: {e}")
            return None

    def get_collection(self, collection_name):
        db = self.connection()
        if db is not None:
            # Verificar y crear la colección si no existe
            if collection_name not in db.list_collection_names():
                db.create_collection(collection_name)
                print(f"Colección '{collection_name}' creada.")
            return db[collection_name]
        return None

class CrudRepository:
    def __init__(self, collection):
        self.collection = collection

    def createNewItem(self, item):
        try:
            result = self.collection.insert_one(item)
            return str(result.inserted_id)
        except errors.PyMongoError as e:
```

```
            print(f"Error al insertar un nuevo item: {e}")
            return None

    def createNewItems(self, items):
        try:
            result = self.collection.insert_many(items)
            return [str(id) for id in result.inserted_ids]
        except errors.PyMongoError as e:
            print(f"Error al insertar nuevos items: {e}")
            return None

    def readItem(self, query):
        return self.collection.find_one(query)

    def readItems(self, query):
        return self.collection.find(query)

    def updateItem(self, query, update):
        try:
            result = self.collection.update_one(query, {"$set": update})
            return result.modified_count
        except errors.PyMongoError as e:
            print(f"Error al actualizar el item: {e}")
            return None

    def deleteItem(self, query):
        try:
            result = self.collection.delete_one(query)
            return result.deleted_count
        except errors.PyMongoError as e:
            print(f"Error al eliminar el item: {e}")
            return None

class AnalyticsRepository(CrudRepository):
    def __init__(self, collection):
        super().__init__(collection)

    def add_analytics(self, postId, socialMediaId, views, likes, comments, shares, reach,
engagementRate):
        analytics = {
            "postId": ObjectId(postId),
            "socialMediaId": socialMediaId,
            "views": views,
            "likes": likes,
            "comments": comments,
            "shares": shares,
            "reach": reach,
            "engagementRate": engagementRate,
```

```
            "createdAt": datetime.now(),
            "updatedAt": datetime.now()
        }
        return self.createNewItem(analytics)

class CampaignsRepository(CrudRepository):
    def __init__(self, collection):
        super().__init__(collection)

    def add_campaign(self, campaignName, description, startDate, endDate, status):
        campaign = {
            "campaignName": campaignName,
            "description": description,
            "startDate": startDate,
            "endDate": endDate,
            "status": status,
            "createdAt": datetime.now(),
            "updatedAt": datetime.now()
        }
        return self.createNewItem(campaign)

class AudienceRepository(CrudRepository):
    def __init__(self, collection):
        super().__init__(collection)

    def add_audience(self, campaignId, ageRange, gender, location, interests):
        audience = {
            "campaignId": ObjectId(campaignId),
            "ageRange": ageRange,
            "gender": gender,
            "location": location,
            "interests": interests,
            "createdAt": datetime.now(),
            "updatedAt": datetime.now()
        }
        return self.createNewItem(audience)

class SchedulesRepository(CrudRepository):
    def __init__(self, collection):
        super().__init__(collection)

    def add_schedule(self, accountId, postId, scheduleTime, status):
        schedule = {
            "accountId": ObjectId(accountId),
            "postId": ObjectId(postId),
            "scheduleTime": scheduleTime,
            "status": status,
            "createdAt": datetime.now(),
```

```
            "updatedAt": datetime.now()
        }
        return self.createNewItem(schedule)

if __name__ == '__main__':
    URI =
'mongodb+srv://juanrod:wd7XbiAqhCJrrEBy@cluster0.xwaug.mongodb.net/?retryWrites=tru
e&w=majority&appName=Cluster0'
    DATABASE = 'DOCE'
    USERCOLLECTION = 'Users'
    ACCOUNTCOLLECTION = 'SocialMediaAccounts'
    POSTCOLLECTION = 'Posts'
    ANALYTICSCOLLECTION = 'Analytics'
    CAMPAIGNCOLLECTION = 'Campaigns'
    AUDIENCECOLLECTION = 'Audience'
    SCHEDULECOLLECTION = 'Schedules'

    # Conexión a las colecciones de la base de datos de MongoAtlas
    db_connection = dataBaseConnection(URI, DATABASE)
    usersCollection = CrudRepository(db_connection.get_collection(USERCOLLECTION))
    socialMediaAccountsCollection =
CrudRepository(db_connection.get_collection(ACCOUNTCOLLECTION))
    postsCollection = CrudRepository(db_connection.get_collection(POSTCOLLECTION))
    analyticsCollection =
AnalyticsRepository(db_connection.get_collection(ANALYTICSCOLLECTION))
    campaignsCollection =
CampaignsRepository(db_connection.get_collection(CAMPAIGNCOLLECTION))
    audienceCollection =
AudienceRepository(db_connection.get_collection(AUDIENCECOLLECTION))
    schedulesCollection =
SchedulesRepository(db_connection.get_collection(SCHEDULECOLLECTION))

    # Función para agregar un usuario
    def add_user(username, email, passwordHash):
        user = {
            "username": username,
            "email": email,
            "passwordHash": passwordHash,
            "createdAt": datetime.now(),
            "updatedAt": datetime.now()
        }
        return usersCollection.createNewItem(user)

    # Función para agregar una cuenta de red social
    def add_social_media_account(userId, socialMediaId, accessToken, refreshToken=None,
expiresAt=None):
        account = {
            "userId": ObjectId(userId),
```

```
            "socialMediaId": socialMediaId,
            "accessToken": accessToken,
            "refreshToken": refreshToken,
            "expiresAt": expiresAt,
            "createdAt": datetime.now(),
            "updatedAt": datetime.now()
        }
        return socialMediaAccountsCollection.createNewItem(account)

    # Función para agregar una publicación
    def add_post(accountId, socialMediaId, content, likesCount=0, commentsCount=0,
sharesCount=0, viewsCount=0):
        document = {
            "accountId": ObjectId(accountId),
            "socialMediaId": socialMediaId,
            "content": content,
            "likesCount": likesCount,
            "commentsCount": commentsCount,
            "sharesCount": sharesCount,
            "viewsCount": viewsCount,
            "createdAt": datetime.now(),
            "updatedAt": datetime.now()
        }
        return postsCollection.createNewItem(document)

    # Función para agregar datos analíticos
    def add_analytics(postId, socialMediaId, views, likes, comments, shares, reach,
engagementRate):
        return analyticsCollection.add_analytics(postId, socialMediaId, views, likes, comments,
shares, reach, engagementRate)

    # Función para agregar una campaña
    def add_campaign(campaignName, description, startDate, endDate, status):
        return campaignsCollection.add_campaign(campaignName, description, startDate,
endDate, status)

    # Función para agregar una audiencia
    def add_audience(campaignId, ageRange, gender, location, interests):
        return audienceCollection.add_audience(campaignId, ageRange, gender, location,
interests)

    # Función para agregar un horario
    def add_schedule(accountId, postId, scheduleTime, status):
        return schedulesCollection.add_schedule(accountId, postId, scheduleTime, status)

    # Función para actualizar un usuario
    def update_user(userId, update):
        query = {"_id": ObjectId(userId)}
```

```
    update["updatedAt"] = datetime.now()
    return usersCollection.updateItem(query, update)

# Función para actualizar una cuenta de red social
def update_social_media_account(accountId, update):
    query = {"_id": ObjectId(accountId)}
    update["updatedAt"] = datetime.now()
    return socialMediaAccountsCollection.updateItem(query, update)

# Función para actualizar una publicación
def update_post(postId, update):
    query = {"_id": ObjectId(postId)}
    update["updatedAt"] = datetime.now()
    return postsCollection.updateItem(query, update)

# Función para eliminar un usuario
def delete_user(userId):
    query = {"_id": ObjectId(userId)}
    return usersCollection.deleteItem(query)

# Función para eliminar una cuenta de red social
def delete_social_media_account(accountId):
    query = {"_id": ObjectId(accountId)}
    return socialMediaAccountsCollection.deleteItem(query)

# Función para eliminar una publicación
def delete_post(postId):
    query = {"_id": ObjectId(postId)}
    return postsCollection.deleteItem(query)

# Ejemplo de uso
if __name__ == "__main__":
    # Agregar un usuario
    new_user_id = add_user(
        username="john_doe",
        email="john.doe@example.com",
        passwordHash="hashed_password"
    )
    print(f"Nuevo usuario agregado con ID: {new_user_id}")

    # Agregar una cuenta de red social
    new_account_id = add_social_media_account(
        userId=new_user_id,
        socialMediaId="instagram",
        accessToken="access_token_here",
        refreshToken="refresh_token_here",
        expiresAt=datetime.now()
    )
```

```
print(f"Nueva cuenta de red social agregada con ID: {new_account_id}")

# Agregar una publicación
new_post_id = add_post(
    accountId=new_account_id,
    socialMediaId="instagram",
    content="¡Hola a todos!",
    likesCount=10,
    commentsCount=5,
    sharesCount=2,
    viewsCount=100
)
print(f"Nueva publicación agregada con ID: {new_post_id}")

# Agregar datos analíticos
new_analytics_id = add_analytics(
    postId=new_post_id,
    socialMediaId="instagram",
    views=100,
    likes=10,
    comments=5,
    shares=2,
    reach=200,
    engagementRate=0.1
)
print(f"Nuevos datos analíticos agregados con ID: {new_analytics_id}")

# Agregar una campaña
new_campaign_id = add_campaign(
    campaignName="Summer Sale",
    description="Promoción de verano",
    startDate=datetime.now(),
    endDate=datetime.now(),
    status="active"
)
print(f"Nueva campaña agregada con ID: {new_campaign_id}")

# Agregar una audiencia
new_audience_id = add_audience(
    campaignId=new_campaign_id,
    ageRange="18-30",
    gender="male",
    location="USA",
    interests=["fashion", "sports"]
)
print(f"Nueva audiencia agregada con ID: {new_audience_id}")

# Agregar un horario
```

```
    new_schedule_id = add_schedule(
        accountId=new_account_id,
        postId=new_post_id,
        scheduleTime=datetime.now(),
        status="scheduled"
    )
    print(f"Nuevo horario agregado con ID: {new_schedule_id}")

    # Actualizar un usuario
    update_result = update_user(new_user_id, {"email":
"john.doe.updated@example.com"})
    print(f"Usuario actualizado: {update_result} documento(s) modificado(s)")

    # Actualizar una cuenta de red social
    update_result = update_social_media_account(new_account_id, {"accessToken":
"new_access_token_here"})
    print(f"Cuenta de red social actualizada: {update_result} documento(s) modificado(s)")

    # Actualizar una publicación
    update_result = update_post(new_post_id, {"content": "¡Hola a todos! (actualizado)"})
    print(f"Publicación actualizada: {update_result} documento(s) modificado(s)")

    # Eliminar una publicación
    delete_result = delete_post(new_post_id)
    print(f"Publicación eliminada: {delete_result} documento(s) eliminado(s)")

    # Eliminar una cuenta de red social
    delete_result = delete_social_media_account(new_account_id)
    print(f"Cuenta de red social eliminada: {delete_result} documento(s) eliminado(s)")

    # Eliminar un usuario
    delete_result = delete_user(new_user_id)
    print(f"Usuario eliminado: {delete_result} documento(s) eliminado(s)")
```

Colecciones y Datos Almacenados

1. Colección: Users

Descripción: Almacena información sobre los usuarios del sistema.
Campos:

- _id: ObjectId (Identificador único del usuario)
- username: String (Nombre de usuario)
- email: String (Correo electrónico)
- passwordHash: String (Hash de la contraseña)
- createdAt: Date (Fecha de creación del usuario)
- updatedAt: Date (Fecha de última actualización)

2. Colección: SocialMediaAccounts

Descripción: Almacena información sobre las cuentas de redes sociales asociadas a los usuarios.
Campos:

- _id: ObjectId (Identificador único de la cuenta)
- userId: ObjectId (Referencia al usuario al que pertenece la cuenta)
- socialMediaId: String (Identificador de la red social, ej. "Facebook", "Twitter")
- accessToken: String (Token de acceso para la cuenta de redes sociales)
- refreshToken: String (Token de actualización para la cuenta de redes sociales)
- expiresAt: Date (Fecha y hora de expiración del token de acceso)
- createdAt: Date (Fecha de creación de la cuenta)
- updatedAt: Date (Fecha de última actualización)
-

3. Colección: Posts

Descripción: Almacena información sobre las publicaciones realizadas en las cuentas de redes sociales.
Campos:

- _id: ObjectId (Identificador único de la publicación)
- accountId: ObjectId (Referencia a la cuenta de redes sociales a la que pertenece la publicación)
- socialMediaId: String (Identificador de la red social en la que se publicó)
- content: String (Contenido de la publicación)
- likesCount: Number (Número de likes de la publicación)
- commentsCount: Number (Número de comentarios de la publicación)
- sharesCount: Number (Número de veces que se compartió la publicación)
- viewsCount: Number (Número de vistas de la publicación)
- createdAt: Date (Fecha de creación de la publicación)
- updatedAt: Date (Fecha de última actualización)
-

4. Colección: Analytics

Descripción: Almacena datos analíticos sobre las publicaciones y campañas.
Campos:

- _id: ObjectId (Identificador único del registro analítico)
- postId: ObjectId (Referencia a la publicación relacionada)
- socialMediaId: String (Identificador de la red social)
- views: Number (Número de vistas)
- likes: Number (Número de likes)
- comments: Number (Número de comentarios)
- shares: Number (Número de veces que se compartió)
- reach: Number (Alcance de la publicación)
- engagementRate: Number (Tasa de interacción)
- createdAt: Date (Fecha de creación del registro)

- updatedAt: Date (Fecha de última actualización)
-

5. Colección: Campaigns

Descripción: Almacena información sobre las campañas de marketing.
Campos:

- _id: ObjectId (Identificador único de la campaña)
- campaignName: String (Nombre de la campaña)
- description: String (Descripción de la campaña)
- startDate: Date (Fecha de inicio de la campaña)
- endDate: Date (Fecha de finalización de la campaña)
- status: String (Estado de la campaña, ej. "active", "completed")
- createdAt: Date (Fecha de creación de la campaña)
- updatedAt: Date (Fecha de última actualización)
-

6. Colección: Audience

Descripción: Almacena información sobre la audiencia objetivo de las campañas.
Campos:

- _id: ObjectId (Identificador único del registro de audiencia)
- campaignId: ObjectId (Referencia a la campaña relacionada)
- ageRange: String (Rango de edad de la audiencia)
- gender: String (Género de la audiencia)
- location: String (Ubicación de la audiencia)
- interests: Array (Intereses de la audiencia)
- createdAt: Date (Fecha de creación del registro)
- updatedAt: Date (Fecha de última actualización)

7. Colección: Schedules

Descripción: Almacena información sobre los horarios de publicación.
Campos:

- _id: ObjectId (Identificador único del horario)
- accountId: ObjectId (Referencia a la cuenta de redes sociales relacionada)
- postId: ObjectId (Referencia a la publicación relacionada)
- scheduleTime: Date (Fecha y hora programada para la publicación)
- status: String (Estado del horario, ej. "scheduled", "published")
- createdAt: Date (Fecha de creación del horario)
- updatedAt: Date (Fecha de última actualización)