Examen de programación Avanzada

Nombre del participante: Elvia Aldana Hernández.

Tecnologías de información Desarrollo 1

Normativa

Lee atentamente las siguientes instrucciones antes de empezar a resolver el examen:

— El examen solo es práctico.

— Duración del examen: 1 Día.

— La organización del examen y la distribución de puntos en los diferentes ejercicios propuestos se resume en la siguiente tabla:

Ejercicios Puntos

1 (SQL) 10

2 (BD) 20

3 (Lógica) 20

3 (Desarrollo) 50

Total 100

Nota:

La calificación aprobatoria será de 70 puntos

1. SQL

1.1) Describe brevemente el funcionamiento del JOIN en el lenguaje de consultas estructurado, los tipos de JOIN's y realiza las siguientes consultas (1.2 y 1.3) tomando como base las tablas dadas.

1.1) Descripción del funcionamiento de JOIN en SQL

En SQL, el JOIN permite combinar filas de dos o más tablas basándose en una condición relacionada, generalmente una clave compartida. Los tipos principales de JOIN son:

- **INNER JOIN**: Devuelve solo las filas que tienen coincidencias en ambas tablas.
- **LEFT JOIN** (o **LEFT OUTER JOIN**): Devuelve todas las filas de la tabla izquierda, y las filas coincidentes de la derecha. Si no hay coincidencia, se completan con NULL.
- **RIGHT JOIN** (o **RIGHT OUTER JOIN**): Devuelve todas las filas de la tabla derecha, y las filas coincidentes de la izquierda. Las filas sin coincidencia se completan con NULL.
- **FULL JOIN** (o **FULL OUTER JOIN**): Devuelve filas cuando hay coincidencias en una de las tablas o en ambas. Las filas sin coincidencia en ambas tablas también se completan con NULL.

Tablas:

Perro - 2

Datos de la tabla miembros:

Papá - 42

Mamá - 45

Hija – 19

Hijo - 16

Datos de la tabla comidamiembros: Enchiladas - Papá

Sopa - Hijo

Ensalada - Mamá

1.2). - Obtener todos los miembros de la familia que tengan una comida asignada (usa cualquier tipo de JOIN).

Usando INNER JOIN para mostrar solo los miembros que tienen una comida asignada.

```
SELECT miembros.Nombre, comidamiembros.Comida
FROM miembros
INNER JOIN comidamiembros ON miembros.Nombre = comidamiembros.Nombre;
```

Resultado esperado

Nombre	Comida
Papá	Enchiladas
Hijo	Sopa
Mamá	Ensalada

1. 3). - Obtener todos los miembros de la familia con su respectiva comida asignada; incluir los miembros que no tienen comida asignada (usa cualquier tipo de JOIN).

Usando `LEFT JOIN` para incluir todos los miembros, y si no tienen una comida asignada, mostrará `NULL` en la columna `Comida`.

```
SELECT miembros.Nombre, comidamiembros.Comida
FROM miembros
LEFT JOIN comidamiembros ON miembros.Nombre = comidamiembros.Nombre;
```

Resultado esperado:

Nombre	Comida
Papá	Enchiladas
Mamá	Ensalada
Hija	NULL
Hijo	Sopa

2. (BD) Crea un diagrama relacional de bases de datos para el siguiente problema:

La empresa necesita para su portal web un sistema de noticias, en el cual se permita publicar varias notas (solo personal interno de la empresa) y a estas se les permita hacer comentarios por cualquier usuario registrado. Es necesario que a cualquier comentario se le pueda dar respuesta (por cualquier otro usuario).

Es importante saber:

Que usuario (interno) subió la nota.

Quien comento la nota (saber si es usuario interno o externo) Fecha y hora del

comentario.

Respuesta del comentario, usuario que respondió, fecha y hora.

NOTA: Actualmente solo se cuenta con la tabla de personal (con los siguientes campos idpersonal, apepaterno, apematerno, nombre, direccion, fechadeingreso; la llave primaria es el idpersonal). Hay que considerar que también se tiene que crear la tabla para el registro de los usuarios y distinguir si son usuarios internos o externos.

Entidades y Relaciones Propuestas

1. **Tabla personal**: Información de los empleados que pueden publicar notas. (Ya existente)
    - **idpersonal** (PK): Identificador único del personal.
    - **apepaterno**: Apellido paterno del personal.
    - **apematerno**: Apellido materno del personal.
    - **nombre**: Nombre del personal.
    - **direccion**: Dirección del personal.
    - **fechadeingreso**: Fecha de ingreso del personal.
2. **Tabla usuarios**: Información de todos los usuarios (internos y externos).
    - **idusuario** (PK): Identificador único del usuario.
    - **nombre_usuario**: Nombre del usuario (puede ser un nickname).
    - **tipo_usuario**: Tipo de usuario, interno o externo (booleano o enumeración).
3. **Tabla notas**: Información de las notas publicadas en el sistema.
    - **idnota** (PK): Identificador único de la nota.
    - **titulo**: Título de la nota.
    - **contenido**: Contenido de la nota.

- **fecha_publicacion**: Fecha y hora de publicación.
- **idpersonal** (FK): Relación con personal para identificar al empleado que publicó la nota.

4. **Tabla comentarios**: Información de los comentarios realizados en las notas.
   - **idcomentario** (PK): Identificador único del comentario.
   - **contenido**: Contenido del comentario.
   - **fecha_comentario**: Fecha y hora del comentario.
   - **idnota** (FK): Relación con notas para identificar a la nota comentada.
   - **idusuario** (FK): Relación con usuarios para identificar al usuario que hizo el comentario.
5. **Tabla respuestas**: Información de las respuestas a los comentarios.
   - **idrespuesta** (PK): Identificador único de la respuesta.
   - **contenido**: Contenido de la respuesta.
   - **fecha_respuesta**: Fecha y hora de la respuesta.
   - **idcomentario** (FK): Relación con comentarios para identificar al comentario respondido.
   - **idusuario** (FK): Relación con usuarios para identificar al usuario que hizo la respuesta.

Diagrama relacional en texto estructurado:

1. personal
   - idpersonal (PK)
   - apepaterno
   - apematerno
   - nombre
   - direccion
   - fechadeingreso

2. usuarios
   - idusuario (PK)
   - nombre_usuario
   - tipo_usuario (interno o externo)

3. notas
   - idnota (PK)
   - titulo
   - contenido
   - fecha_publicacion
   - idpersonal (FK) -> personal.idpersonal

4. comentarios
   - idcomentario (PK)
   - contenido
   - fecha_comentario
   - idnota (FK) -> notas.idnota
   - idusuario (FK) -> usuarios.idusuario

5. respuestas
   - idrespuesta (PK)
   - contenido
   - fecha_respuesta
   - idcomentario (FK) -> comentarios.idcomentario

- idusuario (FK) -> usuarios.idusuario

Código SQL

```
Table personal {
  idpersonal int [pk, not null] // Primary Key
  apepaterno varchar
  apematerno varchar
  nombre varchar
  direccion varchar
  fechadeingreso date
}

Table usuarios {
  idusuario int [pk, not null] // Primary Key
  nombre_usuario varchar
  tipo_usuario varchar // interno o externo
}

Table notas {
  idnota int [pk, not null] // Primary Key
  titulo varchar
  contenido text
  fecha_publicacion datetime
  idpersonal int [ref: > personal.idpersonal] // Foreign Key referencing personal
}

Table comentarios {
  idcomentario int [pk, not null] // Primary Key
  contenido text
  fecha_comentario datetime
  idnota int [ref: > notas.idnota] // Foreign Key referencing notas
  idusuario int [ref: > usuarios.idusuario] // Foreign Key referencing usuarios
```

}

Table respuestas {

idrespuesta int [pk, not null] // Primary Key

contenido text

fecha_respuesta datetime

idcomentario int [ref: > comentarios.idcomentario] // Foreign Key referencing comentarios

idusuario int [ref: > usuarios.idusuario] // Foreign Key referencing usuarios

}

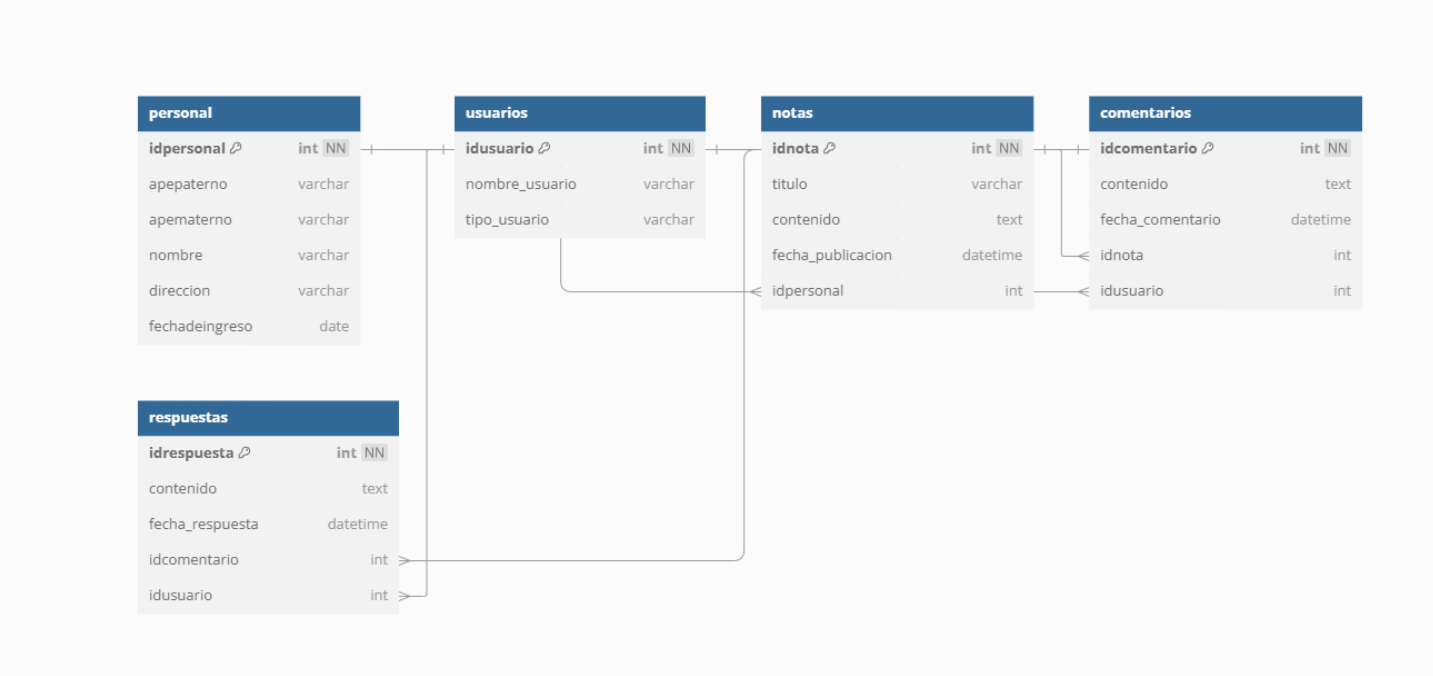
Explicación de la Relación entre Tablas

1. **personal - notas**: Relación de uno a muchos, donde cada miembro del personal puede crear varias notas.
2. **notas - comentarios**: Relación de uno a muchos, donde cada nota puede tener múltiples comentarios.
3. **comentarios - respuestas**: Relación de uno a muchos, donde cada comentario puede tener múltiples respuestas.
4. **usuarios - comentarios/respuestas**: Relación de uno a muchos en ambas tablas. Cada usuario puede comentar en varias notas y responder a varios comentarios.

Explicación del Código

- **[pk, not null]**: Define la columna como clave primaria y que no puede tener valores nulos.
- **[ref: > table.column]**: Crea una clave foránea y establece la relación entre las tablas.
- **varchar** y **text**: Son tipos de datos para cadenas de caracteres. Puedes ajustar el tipo según tus necesidades específicas.

Diagrama Relacional:



3. Desarrolla un SCRIPT en código el algoritmo para obtener el siguiente resultado, dada una secuencia de caracteres, encontrar el número de veces que aparece cada palabra del texto, a lo largo del mismo texto.

Resultado esperado: Lista de palabras únicas en el texto, indicando cuantas veces aparecen dentro de la secuencia de caracteres

Dos ejemplos de Texto para aplicar tu algoritmo

"NO ME MIRES QUE MIRAN QUE NOS MIRAMOS. MIREMOS LA MANERA DE NO MIRARNOS.
MIRA: NO NOS MIREMOS. Y CUANDO NO NOS MIREN, NOS MIRAREMOS."

"El hipopótamo Hipo está con hipo. Y su hipopotamito con hipito. ¿Quién les quita el hipo a los hipopótamos Hipo?"

Código Python:

```
from collections import Counter
import re

def contar_palabras(texto):
    # Convertimos a minúsculas para evitar duplicados por mayúsculas
    texto = texto.lower()
    # Usamos regex para extraer solo palabras (omitiendo signos de puntuación)
    palabras = re.findall(r'\b\w+\b', texto)
    # Contamos la frecuencia de cada palabra
    conteo = Counter(palabras)
    return conteo

# Ejemplos de texto
texto1 = "NO ME MIRES QUE MIRAN QUE NOS MIRAMOS. MIREMOS LA MANERA DE NO MIRARNOS. MIRA: NO NOS MIREMOS. Y CUANDO NO NOS MIREN, NOS MIRAREMOS."
texto2 = "El hipopótamo Hipo está con hipo. Y su hipopotamito con hipito. ¿Quién les quita el hipo a los hipopótamos Hipo?"

# Llamadas a la función y muestra de resultados
resultado1 = contar_palabras(texto1)
resultado2 = contar_palabras(texto2)

print("Resultado para el primer texto:")
for palabra, frecuencia in resultado1.items():
    print(f"'{palabra}': {frecuencia}")

print("\nResultado para el segundo texto:")
for palabra, frecuencia in resultado2.items():
    print(f"'{palabra}': {frecuencia}")
```

Explicación del Código

1. **Normalización de Texto**: Convertimos todo el texto a minúsculas para evitar contar palabras duplicadas debido a diferencias en las mayúsculas.
2. **Extracción de Palabras**: Usamos una expresión regular para obtener solo palabras y omitir signos de puntuación.
3. **Conteo de Palabras**: Utilizamos Counter de la biblioteca collections para contar la frecuencia de cada palabra en la lista.

Resultado esperado para el texto 1:

```
'no': 3
'me': 1
'mires': 2
'que': 2
'miran': 1
'nos': 4
'miramos': 1
'miremos': 2
'la': 1
'manera': 1
'mirarnos': 1
'mira': 1
'y': 1
'cuando': 1
'miren': 1
'miraremos': 1
```

Resultado esperado para el texto 2:

```
'el': 1
'hipopótamo': 1
'hipo': 2
'está': 1
'con': 2
'su': 1
'hipopotamito': 1
'hipito': 1
'quién': 1
'les': 1
'quita': 1
'a': 1
'los': 1
'hipopótamos': 1
```

4. (Desarrollo) Crear una solución web para el problema planteado en el punto número 2. El proyecto debe contemplar una parte back (desarrollo de api rest) y otra front (desarrollo de aplicación cliente)

Se desea que el sistema tenga solo el módulo de captura de Noticias, consulta de noticias, comentarios y respuestas.

Puedes utilizar SQL o MySQL instalado de manera local en tu equipo, así como el ambiente necesario para el desarrollo según el lenguaje de programación elegido; lo único requerido es que al finalizar se compartan los siguientes datos:

- Script de Base de Datos
- Acceso a la aplicación
- Envío del código completo
- Subir el proyecto a dockerHub o a algún repositorio de docker y compartir liga (opcional)

Pasos para el Desarrollo

1. Configuración de la Base de Datos

Script SQL para la creación de las tablas:

```
CREATE TABLE personal (
  idpersonal INT PRIMARY KEY AUTO_INCREMENT,
  apepaterno VARCHAR(50),
  apematerno VARCHAR(50),
  nombre VARCHAR(50),
  direccion VARCHAR(100),
  fechadeingreso DATE
);

CREATE TABLE usuarios (
  idusuario INT PRIMARY KEY AUTO_INCREMENT,
  nombre_usuario VARCHAR(50),
  tipo_usuario ENUM('interno', 'externo')
);

CREATE TABLE notas (
  idnota INT PRIMARY KEY AUTO_INCREMENT,
  titulo VARCHAR(100),
  contenido TEXT,
  fecha_publicacion DATETIME,
  idpersonal INT,
  FOREIGN KEY (idpersonal) REFERENCES personal(idpersonal)
);

CREATE TABLE comentarios (
  idcomentario INT PRIMARY KEY AUTO_INCREMENT,
  contenido TEXT,
  fecha_comentario DATETIME,
  idnota INT,
  idusuario INT,
  FOREIGN KEY (idnota) REFERENCES notas(idnota),
  FOREIGN KEY (idusuario) REFERENCES usuarios(idusuario)
);

CREATE TABLE respuestas (
  idrespuesta INT PRIMARY KEY AUTO_INCREMENT,
```

```
    contenido TEXT,
     fecha_respuesta DATETIME,
      idcomentario INT,
      idusuario INT,
      FOREIGN KEY (idcomentario) REFERENCES comentarios(idcomentario),
      FOREIGN KEY (idusuario) REFERENCES usuarios(idusuario)
    );
```

2. Backend: API REST con Flask

Estructuras de carpetas del Backend:

```
backend/
├── app.py              # Archivo principal
├── models.py           # Modelos SQLAlchemy
├── routes.py           # Rutas para el API REST
├── config.py           # Configuración de la base de datos
├── requirements.txt    # Dependencias de Python
└── Dockerfile          # Configuración de Docker
```

Configuración básica (config.py):

```
# config.py
SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://usuario:password@localhost/nombre_base_datos'
SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Creación de Modelos (models.py):

```
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()

class Personal(db.Model):
    __tablename__ = 'personal'
    idpersonal = db.Column(db.Integer, primary_key=True)
    apepaterno = db.Column(db.String(50))
    apematerno = db.Column(db.String(50))
    nombre = db.Column(db.String(50))
    direccion = db.Column(db.String(100))
    fechadeingreso = db.Column(db.Date)

class Usuario(db.Model):
    __tablename__ = 'usuarios'
    idusuario = db.Column(db.Integer, primary_key=True)
    nombre_usuario = db.Column(db.String(50))
    tipo_usuario = db.Column(db.Enum('interno', 'externo'))

class Nota(db.Model):
    __tablename__ = 'notas'
    idnota = db.Column(db.Integer, primary_key=True)
    titulo = db.Column(db.String(100))
    contenido = db.Column(db.Text)
    fecha_publicacion = db.Column(db.DateTime)
    idpersonal = db.Column(db.Integer, db.ForeignKey('personal.idpersonal'))

class Comentario(db.Model):
    __tablename__ = 'comentarios'
    idcomentario = db.Column(db.Integer, primary_key=True)
    contenido = db.Column(db.Text)
    fecha_comentario = db.Column(db.DateTime)
    idnota = db.Column(db.Integer, db.ForeignKey('notas.idnota'))
    idusuario = db.Column(db.Integer, db.ForeignKey('usuarios.idusuario'))

class Respuesta(db.Model):
    __tablename__ = 'respuestas'
    idrespuesta = db.Column(db.Integer, primary_key=True)
    contenido = db.Column(db.Text)
    fecha_respuesta = db.Column(db.DateTime)
    idcomentario = db.Column(db.Integer, db.ForeignKey('comentarios.idcomentario'))
    idusuario = db.Column(db.Integer, db.ForeignKey('usuarios.idusuario'))
```

Si se tiene alguna especificación adicional (usuario, contraseña, etc) por favor mencionarla en algún apartado de comentarios dentro de la evaluación.

Rutas para el API (routes.py):

```
from flask import Flask, request, jsonify
from models import db, Personal, Usuario, Nota, Comentario, Respuesta


app = Flask(__name__)
app.config.from_object('config')
db.init_app(app)


# Ejemplo de ruta para crear una nota
@app.route('/api/notas', methods=['POST'])
def create_nota():
    data = request.json
    nueva_nota = Nota(
        titulo=data['titulo'],
        contenido=data['contenido'],
        fecha_publicacion=data['fecha_publicacion'],
        idpersonal=data['idpersonal']
    )
    db.session.add(nueva_nota)
    db.session.commit()
    return jsonify({"mensaje": "Nota creada con éxito"}), 201


# Ejemplo de ruta para obtener comentarios de una nota
@app.route('/api/notas/<int:idnota>/comentarios', methods=['GET'])
def get_comentarios(idnota):
    comentarios = Comentario.query.filter_by(idnota=idnota).all()
    return jsonify([c.to_dict() for c in comentarios])
```

Dockerfile para el Backend:

```
# Dockerfile

FROM python:3.9

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD ["flask", "run", "--host=0.0.0.0"]
```

3. Frontend: React para Captura y Consulta

Estructura de carpetas del Frontend:

```
frontend/
├── src/
|   ├── App.js           # Componente principal
|   ├── services/        # Funciones para consumir la API
|   ├── components/      # Componentes para Noticias, Comentarios, Respuestas
|   └── index.js         # Archivo de entrada
└── Dockerfile           # Configuración de Docker
```

Ejemplo de Componente para Noticias (src/components/Noticias.js):

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function Noticias() {
  const [noticias, setNoticias] = useState([]);

  useEffect(() => {
    axios.get('/api/notas')
      .then(response => setNoticias(response.data))
      .catch(error => console.error(error));
  }, []);

  return (
    <div>
      <h1>Noticias</h1>
```

```
  {noticias.map(nota => (
          <div key={nota.idnota}>
            <h2>{nota.titulo}</h2>
            <p>{nota.contenido}</p>
          </div>
        ))}
      </div>
   );
}

export default Noticias;
```

Dockerfile para el Frontend:

```
# Dockerfile
FROM node:14
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["npm", "start"]
```

4. Despliegue Opcional en DockerHub

1. **Crear imagen Docker:**

```
docker build -t nombre_usuario/backend .

docker build -t nombre_usuario/frontend .
```

2. **Subir a DockerHub:**

```
docker tag nombre_usuario/backend nombre_usuario/backend:latest

docker tag nombre_usuario/frontend
nombre_usuario/frontend:latest

docker push nombre_usuario/backend:latest

docker push nombre_usuario/frontend:latest
```

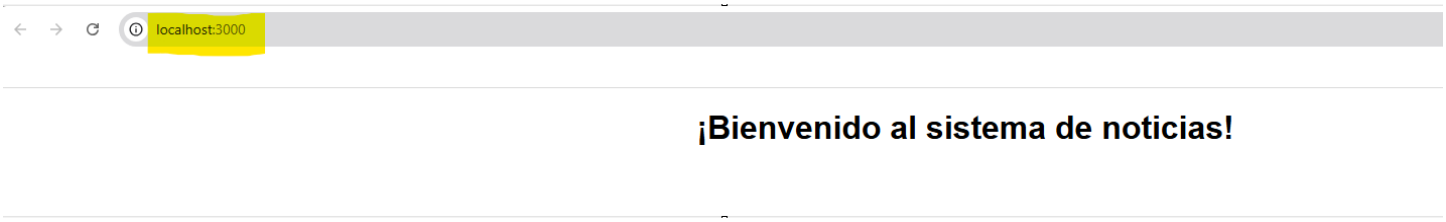
Frontend:

```
  http://localhost:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
Compiled successfully!

You can now view sistema-noticias-frontend in the browser.

  http://localhost:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Backend:

```
   Use a production WSGI server instead.
 * Debug mode: on
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 137-515-813
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Un ejemplo:



Imágenes Docker:

Frontend:

https://hub.docker.com/repository/docker/hanna975/nombre_del_proyecto_frontend

Backend:

https://hub.docker.com/repository/docker/hanna975/nombre_del_proyecto_backend

Proyecto examen UCEM en GitHub:

https://github.com/HannaAlda/ExamenUCEM