Informe final del proyecto

Integrantes

- Esteeven Andres Gallegos Calle
- Noemí Amalia Gudiño

1. Introducción y objetivos

El objetivo de este trabajo fue diseñar y desarrollar un Sistema de Gestión y Recomendación de Viajes que integre diferentes tipos de bases de datos, cada una aprovechando sus particularidades técnicas.

El sistema permite:

- Almacenar información de usuarios, destinos turísticos, hoteles, actividades y reservas.
- Gestionar datos temporales como búsquedas recientes y reservas en proceso.
- Analizar relaciones entre usuarios y destinos, así como entre usuarios entre sí, para generar recomendaciones personalizadas.

2. Diseño e integración de las bases de datos

2.1. Esquema general

El sistema se diseñó para que cada tipo de información se guarde en la base que mejor se adapta a sus características. La siguiente tabla resume la decisión de modelado:

Base de datos	Tipo de información almacenada	Justificación
MongoDB	Usuarios, destinos, hoteles, actividades y reservas confirmadas o pagadas.	Permite manejar documentos complejos, ideales para almacenar datos turísticos con diferentes atributos. Facilita consultas agregadas y visualización de estadísticas, su estructura simple sin muchas relaciones facilita su uso sin depender que para mostrar la información completa tenga que acceder a más de una base.

Redis	Reservas pendientes, caches de búsqueda y usuarios conectados en tiempo real.	Su velocidad y soporte para expiración automática lo hacen ideal para datos temporales o volátiles, como reservas en proceso o sesiones de usuario, su uso en caché nos otorga el beneficio que ante bases de datos aún más grandes minimizar el gasto de memoria
Neo4j	Relaciones entre usuarios (AMIGO_DE, FAMILIAR_DE) y entre usuarios y destinos (VISITO)	Su estructura de grafos permite representar relaciones complejas y consultas de recomendación basadas en vínculos sociales.

a) Datos estructurados y permanentes (MongoDB)

- **Colección usuarios**
  Cada documento representa un usuario con los campos:
  { usuario_id, nombre, email, telefono }
- **Colección destinos**
  Representa cada destino turístico con información general:
  { destino_id, ciudad, pais, descripcion }
- **Colección hoteles**
  Contiene los hoteles asociados a distintos destinos, con atributos variables y servicios almacenados como lista:
  { hotel_id, nombre, ciudad, precio, calificacion, servicios }
- **Colección actividades**
  Guarda las actividades turísticas disponibles por destino:
  { actividad_id, nombre, destino, precio, duración }
- **Colección reservas**
  Registra el historial de reservas realizadas, con referencia a usuarios y destinos:
  { reserva_id, usuario_id, destino_id, fecha_reserva, estado, precio_total }
  Las reservas con estado *Confirmada* o *Pagada* se conservan aquí de manera permanente.

Este modelo documental permite una estructura flexible, ya que los documentos no requieren un esquema fijo y pueden ampliarse con nuevos campos sin alterar la base completa.

b) Datos temporales o en memoria (Redis)

Redis se utiliza como base de datos complementaria para información que cambia rápidamente o tiene una vida útil limitada:

- **Reservas pendientes:**
  Cada reserva pendiente se almacena con clave tipo
  reserva:pendiente:<id>
  y contiene un hash con los campos:
  { reserva_id, usuario_id, destino_id, fecha_reserva, estado, precio_total }.
  Se define un tiempo de expiración aleatorio entre 30 y 300 segundos.

- **Usuarios conectados:**
  Cada usuario activo se guarda como
  usuario:conectado:<usuario_id>,
  con sus datos básicos, hora de conexión y expiración automática generada de forma random.

- **Caché de búsqueda**
  Simulación de búsquedas de usuarios en una de las 3 bases principales (destinos, actividades y hoteles) para generar un almacén en redis de data popular según categorias
  cache:busqueda:base:id_dato
  Con sus datos completos y un tiempo de expiración de 1 dia (86400 s)

c) Datos relacionales y de conocimiento (Neo4j)

Neo4j modela las relaciones entre usuarios y destinos mediante nodos y aristas:

- **Nodos:**
  - (Usuario {usuario_id, nombre})
  - (Destino {destino_id, ciudad, pais})
- **Relaciones:**
  - (Usuario)-[:VISITO]->(Destino)
  - (Usuario)-[:AMIGO_DE]->(Usuario)
  - (Usuario)-[:FAMILIAR_DE]->(Usuario)

Este modelo permite realizar consultas complejas de recomendación, cómo identificar destinos visitados por los amigos del usuario, o encontrar patrones de afinidad entre viajeros.

2.2. Integración de las bases

El proyecto integró las tres tecnologías de forma coordinada dentro de un mismo entorno de desarrollo en Python.
Las funciones de consulta combinan los datos según sea necesario:

- **Redis + MongoDB:** se integran en la consulta de reservas en proceso, donde Redis guarda temporalmente las reservas pendientes y MongoDB provee la información de los usuarios y destinos asociados.
- **Neo4j + MongoDB:** se combinan en la generación de recomendaciones personalizadas. Neo4j identifica los destinos recomendados a partir de relaciones de amistad y MongoDB permite recuperar los hoteles y servicios disponibles en esos destinos.

Esta integración se realizó mediante conectores específicos (pymongo, redis-py, neo4j.Driver) y consultas parametrizadas en cada motor, garantizando consistencia y eficiencia.

3. Consultas implementadas

A continuación, se muestran capturas de pantallas de las principales consultas desarrolladas.

2.a Usuarios que visitaron Bariloche

```
    with driver.session() as session:
        query = """
        MATCH (u:Usuario)-[:VISITO]->(d:Destino {ciudad: 'Bariloche'})
        RETURN u.nombre AS usuario, u.usuario_id AS id
        ORDER BY u.nombre
        """

        resultado = session.run(query)
        usuarios = []

        for record in resultado:
            usuarios.append({
                "id": record["id"],
                "nombre": record["usuario"]
            })

    # Caso sin resultados
    if not usuarios:
        print("⚠️ No se encontraron usuarios que visitaran Bariloche.")
        return pd.DataFrame(columns=["id", "nombre"])

    return pd.DataFrame(usuarios)

usuarios_bariloche = consulta_a_usuarios_bariloche()

# Tabla
display(usuarios_bariloche)
```

```
================================================
CONSULTA 2.a: Usuarios que visitaron Bariloche
================================================
```

	id	nombre
0	5	Ana Torres
1	2	Juan López
2	14	Lucía Silva
3	1	María Pérez

2.b Recomendación de destinos por amigos:

```
    with driver.session() as session:
        query = """
        MATCH (juan:Usuario {nombre: 'Juan López'})-[:VISITO]->(d:Destino)
        MATCH (juan)-[:AMIGO_DE]->(amigo:Usuario)-[:VISITO]->(d)
        RETURN amigo.nombre AS amigo, d.ciudad AS destino
        ORDER BY amigo.nombre, d.ciudad
        """

        resultado = session.run(query)
        filas = []

        for record in resultado:
            filas.append({
                "amigo": record["amigo"],
                "destino": record["destino"]
            })

    if not filas:
        print("⚠️ No se encontraron amigos de Juan que hayan visitado los mismos destinos.")
        return pd.DataFrame(columns=["amigo", "destino"])

    return pd.DataFrame(filas)


amigos_juan_destinos = consulta_b_amigos_juan_destinos()

display(amigos_juan_destinos)
```

```
============================================================
CONSULTA 2.b: Amigos de Juan que visitaron algún destino que él también visitó
============================================================
```

	amigo	destino
0	María Pérez	Bariloche

2.c Recomendación a un usuario de destinos que él ni sus amigos han visitado

```python
with driver.session() as session:
    query = """
    MATCH (u:Usuario {usuario_id: $uid})
    MATCH (d:Destino)
    WHERE NOT (u)-[:VISITO]->(d)
      AND NOT EXISTS {
          MATCH (u)-[:AMIGO_DE]->(amigo)-[:VISITO]->(d)
      }
    RETURN d.ciudad AS destino, d.pais AS pais
    ORDER BY d.ciudad
    """

    resultado = session.run(query, uid=usuario_id)
    filas = []

    for record in resultado:
        filas.append({
            "destino": record["destino"],
            "pais": record["pais"]
        })

# Si no hay resultados
if not filas:
    print(f"⚠️ {nombre_usuario} y sus amigos ya visitaron todos los destinos disponibles.")
    return pd.DataFrame(columns=["destino", "pais"])

return pd.DataFrame(filas)
```

```
============================================================
CONSULTA 2.c: Sugerir destinos nuevos para María Pérez
============================================================
```

	destino	pais
0	Bogotá	Colombia
1	Cancún	México
2	Cartagena	Colombia
3	El Calafate	Argentina
4	Florianópolis	Brasil
5	Kioto	Japón
6	Londres	Reino Unido
7	Madrid	España
8	Roma	Italia
9	Río de Janeiro	Brasil
10	Tokio	Japón
11	Ushuaia	Argentina

2.d Recomendación de destinos basados en viajes de amigos

```python
    with driver.session() as session:
        query = """
        MATCH (u:Usuario {usuario_id: $uid})-[:AMIGO_DE]->(amigo)-[:VISITO]->(d:Destino)
        WHERE NOT (u)-[:VISITO]->(d)
        RETURN amigo.nombre AS amigo, d.ciudad AS destino, d.pais AS pais
        ORDER BY amigo.nombre, d.ciudad
        """

        resultado = session.run(query, uid=usuario_id)
        filas = []

        for record in resultado:
            filas.append({
                "amigo": record["amigo"],
                "destino": record["destino"],
                "pais": record["pais"]
            })

    # Si no hay resultados
    if not filas:
        print(f"⚠️ No se encontraron recomendaciones para {nombre_usuario}.")
        return pd.DataFrame(columns=["amigo", "destino", "pais"])

    return pd.DataFrame(filas)

# Ejemplo de ejecución
recomendaciones_juan = consulta_d_recomendar_por_amigos(2, "Juan López")
display(recomendaciones_juan)
```

```
======================================================================
CONSULTA 2.d: Recomendaciones basadas en viajes de amigos para Juan López
======================================================================
```

	amigo	destino	pais
0	María Pérez	París	Francia

2.e Lista de hoteles en los destinos recomendados del punto anterior

```python
    # Nombres de las ciudades recomendadas
    ciudades = destinos_recomendados["destino"].tolist()
    print(f" Buscamos hoteles en: {', '.join(ciudades)}")

    # Buscamps en la colección de MongoDB
    query = {"ciudad": {"$in": ciudades}}
    hoteles = list(db["hoteles"].find(query, {"_id": 0, "nombre": 1, "ciudad": 1, "servicios": 1}))

    if not hoteles:
        print("⚠️  No se encontraron hoteles en los destinos recomendados.")
        return pd.DataFrame(columns=["nombre", "ciudad", "servicios"])

    print(f"\n Se encontraron {len(hoteles)} hoteles:")

    # DataFrame
    return pd.DataFrame(hoteles)

hoteles_recomendados = consulta_e_hoteles_en_destinos(recomendaciones_juan)

display(hoteles_recomendados)
```

Se encontraron 3 hoteles:

	nombre	ciudad	servicios
0	Le Jardin	París	[wifi, desayuno, pileta]
1	Hôtel Lumière	París	[wifi, desayuno]
2	Champs Élysées Inn	París	[wifi, desayuno, spa]

2.f Visualización de las reservas en proceso

```python
reservas = []

for clave in claves:
    datos_raw = r.hgetall(clave)

    datos = {
        (k.decode("utf-8") if isinstance(k, bytes) else k):
        (v.decode("utf-8") if isinstance(v, bytes) else v)
        for k, v in datos_raw.items()
    }

    # Buscamos información en MongoDB
    usuario = db["usuarios"].find_one(
        {"usuario_id": int(datos["usuario_id"])}, {"_id": 0, "nombre": 1}
    )
    destino = db["destinos"].find_one(
        {"destino_id": int(datos["destino_id"])}, {"_id": 0, "ciudad": 1, "pais": 1}
    )

    reservas.append({
        "reserva_id": int(datos["reserva_id"]),
        "usuario": usuario["nombre"] if usuario else f"ID {datos['usuario_id']}",
        "destino": destino["ciudad"] if destino else f"ID {datos['destino_id']}",
        "pais": destino["pais"] if destino else "-",
        "fecha_reserva": datos["fecha_reserva"],
        "estado": datos["estado"],
        "precio_total": float(datos["precio_total"])
    })
```

	reserva_id	usuario	destino	pais	fecha_reserva	estado	precio_total
10	3	Gonzalo Ponce	Bariloche	Argentina	2025-03-07	Pendiente	87000.0
18	6	Elena Vázquez	Bariloche	Argentina	2025-03-10	Pendiente	91000.0
14	12	Ramiro Romero	Rosario	Argentina	2025-03-16	Pendiente	107000.0
15	16	Laura Méndez	Bariloche	Argentina	2025-03-20	Pendiente	90000.0
16	19	Romina Cabrera	Cancún	México	2025-03-23	Pendiente	153000.0
12	31	Marina López	Cancún	México	2025-04-04	Pendiente	150000.0
17	34	Esteban Núñez	Madrid	España	2025-04-07	Pendiente	109000.0
5	40	Nicolás Ruiz	Bariloche	Argentina	2025-04-13	Pendiente	98000.0
13	42	Paula Ortiz	Madrid	España	2025-04-15	Pendiente	149000.0
0	55	Martín Díaz	Madrid	España	2025-04-28	Pendiente	108000.0
6	58	Santiago Pérez	Cancún	México	2025-05-01	Pendiente	151000.0
8	66	Lucas Delgado	Bariloche	Argentina	2025-05-09	Pendiente	155000.0
2	69	Milagros Medina	Cancún	México	2025-05-12	Pendiente	100000.0
11	73	Matías Torres	Cancún	México	2025-05-16	Pendiente	155000.0
3	76	Nicolás Ruiz	Madrid	España	2025-05-19	Pendiente	98000.0
7	78	Diego Rivas	Cancún	México	2025-05-21	Pendiente	150000.0
4	83	Sofía Romero	Cancún	México	2025-05-26	Pendiente	89000.0
9	85	Franco Giménez	Bariloche	Argentina	2025-05-28	Pendiente	91000.0

2.g Usuarios conectados actualmente

```python
def obtener_usuarios_conectados():
    activos = []
    ids = r.smembers('usuarios_conectados')

    for usuario in ids:
        clave_hash = f"usuario:conectado:{usuario.decode()}"
        if r.exists(clave_hash):
            datos = r.hgetall(clave_hash)

            usuario_decodificado = {k.decode(): v.decode() for k, v in datos.items()}
            activos.append(usuario_decodificado)
        else:
            r.srem('usuarios_conectados', usuario)

    return activos

def imprimir_usuarios_conectados(usuarios):

    if not usuarios:
        print("\n No hay usuarios conectados actualmente.\n")
        return

    print("\n Usuarios conectados actualmente:")
    print("-" * 60)

    for u in usuarios:
        print(f" ID: {u['usuario_id']}")
        print(f"  Nombre: {u['nombre']}")
        print(f"  Hora de conexión: {u['hora_coneccion']}")
        print("-" * 60)

datos = obtener_usuarios_conectados()
imprimir_usuarios_conectados(datos)
```

```
 Usuarios conectados actualmente:
------------------------------------------------------------
 ID: 42
   Nombre: Elena Vázquez
   Hora de conexión: 2025-10-27 22:28:28
------------------------------------------------------------
 ID: 46
   Nombre: Pamela Benítez
   Hora de conexión: 2025-10-27 22:29:55
------------------------------------------------------------
```

2.h Mostrar destinos con precio inferior a $100.000

```python
destinos_lt = db.destinos.find({ 'precio_promedio': { '$lt': 100000} })

def imprimir_destinos(destinos):

    destinos = list(destinos)

    if not destinos:
        print("\n No se encontraron destinos con precio promedio < 100.000.\n")
        return

    print("\n Destinos con precio promedio menor a 100.000")
    print("-" * 60)

    for d in destinos:
        nombre = f"{d.get('ciudad')}, {d.get('pais')}"
        precio = d.get("precio_promedio")

        if isinstance(precio, (int, float)):
            precio_str = f"${precio:,.0f}".replace(",", ".")
        else:
            precio_str = precio

        print(f" {nombre}")
        print(f"    Precio promedio: {precio_str}")
        print("-" * 60)

imprimir_destinos(destinos_lt)
```

```
Destinos con precio promedio menor a 100.000
--------------------------------------------------
Bariloche, Argentina
   Precio promedio: $90.000
--------------------------------------------------
El Calafate, Argentina
   Precio promedio: $95.000
--------------------------------------------------
```

2.i Mostrar todos los hoteles de "Jujuy"

```python
hoteles_jujuy = db.hoteles.find({ 'ciudad': 'Jujuy'})

def imprimir_hoteles(hoteles, ciudad='Jujuy'):

    hoteles = list(hoteles)

    if not hoteles:
        print(f"\n No se encontraron hoteles en {ciudad}.\n")
        return

    print(f"\n Hoteles en {ciudad}")
    print("-" * 60)

    for h in hoteles:
        nombre = h.get("nombre")
        precio = h.get("precio")
        calificacion = h.get("calificacion")

        if isinstance(precio, (int, float)):
            precio_str = f"${precio:,.0f}".replace(",", ".")
        else:
            precio_str = precio

        print(f" {nombre}: {calificacion} ★")
        print(f"  Precio promedio: {precio}")
        print("-" * 60)

imprimir_hoteles(hoteles_jujuy)
```

```
Hoteles en Jujuy
------------------------------------------------------------
Altos del Norte: 3 ★
  Precio promedio: 60000
------------------------------------------------------------
Hotel de las Nubes: 5 ★
  Precio promedio: 99000
------------------------------------------------------------
Refugio del Norte: 3 ★
  Precio promedio: 68000
------------------------------------------------------------
```

2.k Mostrar las actividades de "Ushuaia" del tipo "Aventura"

```
actividades_ushuaia = db.actividades.find({
    'ciudad': 'Ushuaia',
    'tipo': 'Aventura'
})

def imprimir_actividades(actividades, ciudad, tipo=None):
    actividades = list(actividades)

    if not actividades:
        print(f"\n No se encontraron actividades de {tipo} en {ciudad}.\n")
        return

    print(f"\n Actividades de {tipo} en {ciudad}")
    print("-" * 40)

    for a in actividades:
        nombre = a.get("nombre")
        print(f"* {nombre}")


imprimir_actividades(actividades_ushuaia, ciudad='Ushuaia', tipo='Aventura')
```

```
 Actividades de Aventura en Ushuaia
----------------------------------------
* Senderismo por el Parque Nacional
```

2.l Mostrar la cantidad de reservas concretadas de cada usuario Mostrar el usuario y la cantidad

```
pipeline = [
    {"$group": {
        "_id": "$usuario_id",
        "total_reservas": { "$sum": 1}
    }},
    {"$lookup": {
        "from": "usuarios",
        "localField": "_id",
        "foreignField": "usuario_id",
        "as": "usuario_info"
    }},
    {"$unwind": "$usuario_info"},
    {"$project": {
        "_id": 0,
        "usuario_id": "$_id",
        "nombre": "$usuario_info.nombre",
        "total_reservas": 1
    }},
    {"$sort": {"usuario_id": 1}}
]
users = db.reservas.aggregate(pipeline)

def imprimir_reservas_por_usuario(reservas):
    print("\n Total de reservas concetradas por usuario")
    print("-" * 60)

    for r in reservas:
        print(f"{r['nombre']} (ID {r['usuario_id']})")
        print(f"   Total de reservas: {r['total_reservas']}")
        print("-" * 60)

imprimir_reservas_por_usuario(users)
```

```
📊 Total de reservas concetradas por usuario
--------------------------------------------------
María Pérez (ID 1)
   Total de reservas: 2
--------------------------------------------------
Juan López (ID 2)
   Total de reservas: 2
--------------------------------------------------
Carla Gómez (ID 3)
   Total de reservas: 2
--------------------------------------------------
Luis Fernández (ID 4)
   Total de reservas: 1
--------------------------------------------------
```

ESTADÍSTICA

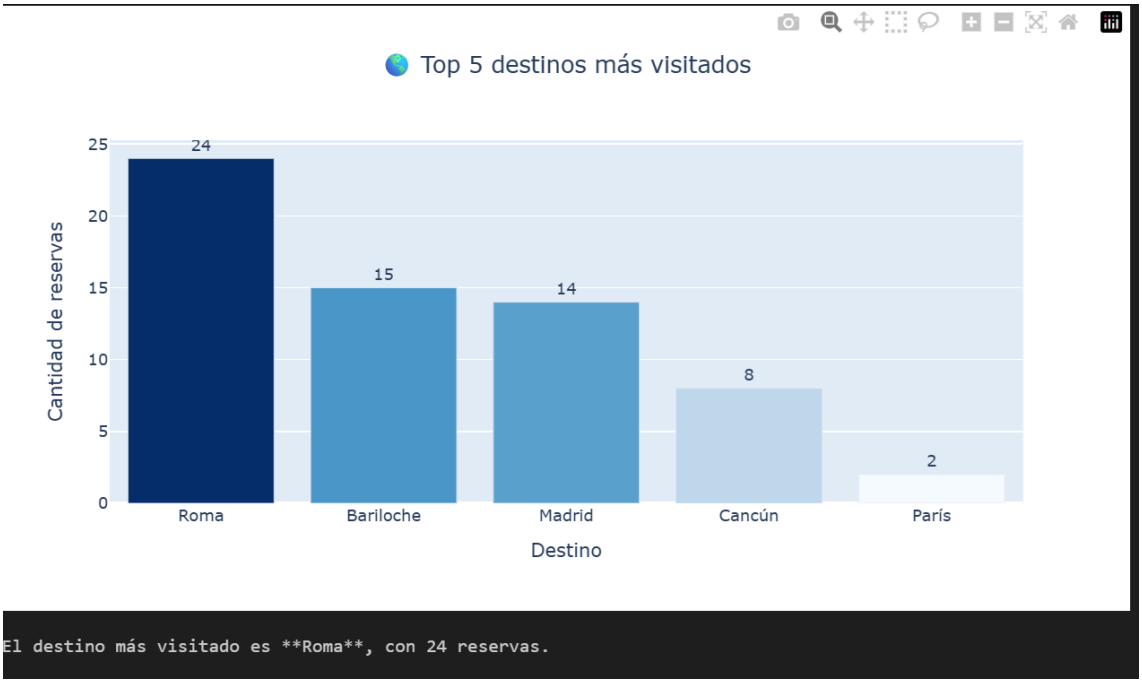
2.m.i Destino más visitado

```python
reservas = list(db.reservas.aggregate([
    {"$match": {"estado": {"$in": ["Confirmada", "Pagada"]}}},
    {"$group": {"_id": "$destino_id", "cantidad": {"$sum": 1}}},
    {"$sort": {"cantidad": -1}},
    {"$limit": 5}
]))

# Mapeo destino_id -> ciudad
destinos_dict = {d["destino_id"]: d["ciudad"] for d in db.destinos.find({}, {"_id": 0, "destino_id": 1, "ciudad": 1})}
ciudades = [destinos_dict.get(a["_id"], str(a["_id"])) for a in reservas]
valores = [a["cantidad"] for a in reservas]

# DataFrame
df = pd.DataFrame({"Destino": ciudades, "Reservas": valores})

# Gráfico de barras (color más fuerte = más reservas)
fig = px.bar(
    df,
    x="Destino",
    y="Reservas",
    color="Reservas",
    color_continuous_scale="Blues",
    title="🌎 Top 5 destinos más visitados",
    text="Reservas"
)
fig.update_traces(textposition="outside")
fig.update_layout(
```




```
El destino más visitado es **Roma**, con 24 reservas.
```

2.m.ii Hotel más barato

```
hoteles = list(db.hoteles.find({}, {"_id": 0, "nombre": 1, "ciudad": 1, "precio": 1}).sort("precio", 1).limit(5))

if not hoteles:
    print("⚠️ No hay datos de hoteles en MongoDB.")
    return pd.DataFrame(columns=["nombre", "ciudad", "precio"])

# Detalle para el gráfico con salto de línea
for h in hoteles:
    h["Hotel_Detalle"] = f"{h['nombre']}<br>{h['ciudad']}"

df = pd.DataFrame(hoteles)

# Ordenamos de menor a mayor precio para que la barra más alta = hotel más caro
df = df.sort_values("precio", ascending=True)

# Gráfico de barras
fig = px.bar(
    df,
    x="Hotel_Detalle",
    y="precio",
    color="precio",  # gradiente según precio
    color_continuous_scale=px.colors.sequential.Blues,
    text="precio",
    title="🏨 Top 5 hoteles más baratos",
    category_orders={"Hotel_Detalle": df["Hotel_Detalle"].tolist()}
)
```

El hotel más barato es **Llanuras Hotel** en La Pampa, con un precio promedio de $58,000

2.m.iii Actividad más popular

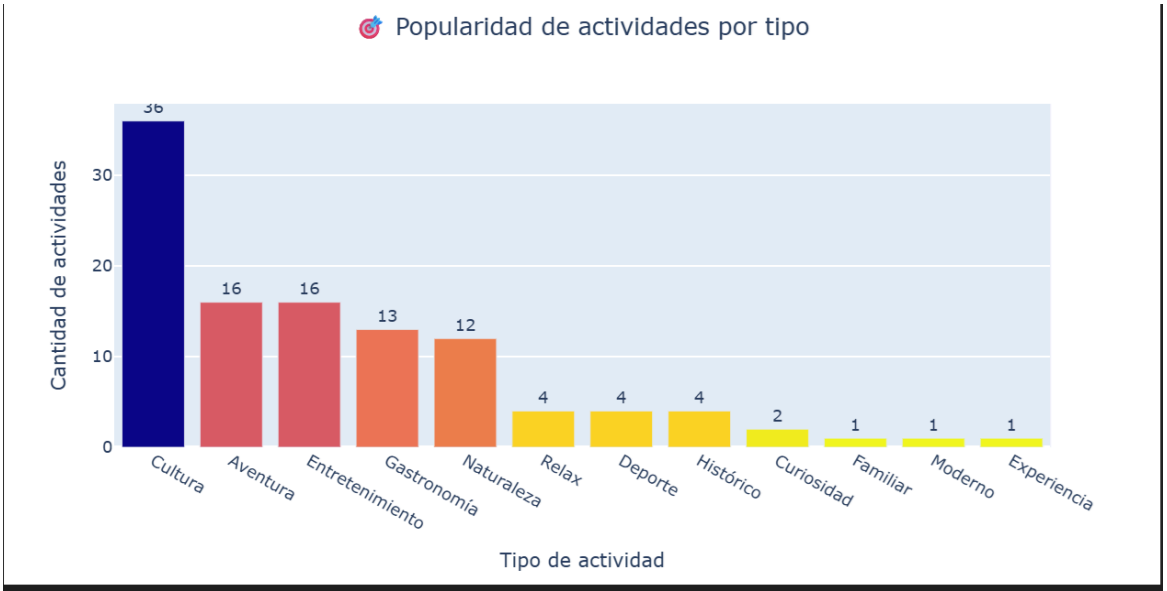
```
actividades = list(db.actividades.aggregate([
    {"$group": {"_id": "$tipo", "cantidad": {"$sum": 1}}},
    {"$sort": {"cantidad": -1}},
    {"$limit": 1}
]))

if not actividades:
    print("⚠️ No hay datos de actividades en MongoDB.")
    return pd.DataFrame(columns=["Tipo", "Cantidad"])

tipo_mas_popular = actividades[0]["_id"]
cantidad = actividades[0]["cantidad"]

# DataFrame con conteo de todos los tipos
df = pd.DataFrame(list(db.actividades.aggregate([
    {"$group": {"_id": "$tipo", "cantidad": {"$sum": 1}}},
    {"$sort": {"cantidad": -1}}
])))
df.rename(columns={"_id": "Tipo"}, inplace=True)

# Gráfico
fig = px.bar(
    df,
    x="Tipo",
    y="cantidad",
    color="cantidad",
    color_continuous_scale=px.colors.sequential.Plasma_r,
```




```
La actividad más popular es del tipo **Cultura** con 36 actividades.
```

MODIFICACIÓN DE DATOS

1. **Incrementar el precio de las actividades de Tucuman un 5%**

```
tucuman = db.actividades.find({ "ciudad": "Tucumán"})
print('Precios Originales en Tucumán')
for t in tucuman:
    print(f"{t['nombre']}: {t['precio']} $")

# Aumentar precio un 5%
db.actividades.update_many(
  { 'ciudad': 'Tucumán'},
  { '$mul': { 'precio': 1.05 } }
 )

tucuman = db.actividades.find({ "ciudad": "Tucumán"})

print('Nuevos precios en Tucumán')
for t in tucuman:
    print(f"{t['nombre']}: {t['precio']} $")
```

```
Precios Originales en Tucumán
Caminata por los Valles Calchaquíes: 42000.0 $
Visita a la Casa Histórica: 36750.0 $
Nuevos precios en Tucumán
Caminata por los Valles Calchaquíes: 44100.0 $
Visita a la Casa Histórica: 38587.5 $
```

2. **Agregar al hotel id=1 el servicio de SPA**

```
db.hoteles.update_one(
    { "hotel_id": 1 },
    { "$addToSet": { "servicios": { "$each": ["Spa"] } } }
)

info_hotel = db.hoteles.find_one({ "hotel_id": 1 })

def impresion_informacion_hotel(info):
    print('-'*25)
    print(f"Servicios del Hotel: {info['nombre']}")
    i = 0
    for s in info['servicios']:
        i = i + 1
        print(f"{i}: {s}")

impresion_informacion_hotel(info_hotel)
```

```
-------------------------
Servicios del Hotel: Hotel Sol
1: wifi
2: pileta
3: desayuno
4: Spa
```

3. Eliminar el destino que desee

```
destino_a_eliminar = input("Ingrese un destino a retirar de la base de datos: ")
destino_a_eliminar = destino_a_eliminar.capitalize()

Inicial = db["destinos"].find_one( {"ciudad": destino_a_eliminar})

if not Inicial:
    print(f"❌ El destino '{destino_a_eliminar}' no existe.")
else:
    destino_id = Inicial["destino_id"]
    print("Datos iniciales:", Inicial)

    with driver.session() as session:
        existe = session.run(
            "MATCH (d:Destino {destino_id: $did}) RETURN d",
            did=destino_id
        ).single()

        if existe:
            session.run("""
                MATCH (d:Destino {destino_id: $did})
                DETACH DELETE d;
            """, did=destino_id)

    # Eliminar hoteles y destino en MongoDB
    resultado_hoteles = db.hoteles.delete_many({'ciudad': destino_a_eliminar})
    resultado_destino = db.destinos.delete_many({'ciudad': destino_a_eliminar})

    # Verificar
    final = db["destinos"].find_one({"ciudad": destino_a_eliminar})
    print(f"Datos finales: {final}")
    if not final:
        print(f"Destino {destino_a_eliminar} retirado")
```

```
Ingrese un destino a retirar de la base de datos:  Miami
Datos iniciales: {'_id': ObjectId('6901614614e125fea54f72c7'), 'destino_id': 22, 'ciudad':
'Miami', 'pais': 'Estados Unidos', 'tipo': 'Playa', 'precio_promedio': 150000}
Datos finales: None
Destino Miami retirado
```

4. **Eliminar un usuario que desee**

```
usuario_a_eliminar = input("Ingrese un usuario a retirar de la base de datos: ")

with driver.session() as session:

    existe = session.run(
        "MATCH(u:Usuario {nombre: $nombre}) RETURN u",
        nombre= usuario_a_eliminar
    ).single()

    if existe:
        session.run(
            """
            MATCH (u:Usuario {nombre: $nombre})
            DETACH DELETE u;
            """,
            nombre=usuario_a_eliminar
        )
        print(f"Relaciones de Usuario '{usuario_a_eliminar}' eliminadas")

resultado_usuario = db.usuarios.delete_one({'nombre': usuario_a_eliminar})
print(f"   Registros eliminados: {resultado_usuario.deleted_count}")
```

```
Ingrese un usuario a retirar de la base de datos:  Valentina Castro
Relaciones de Usuario 'Valentina Castro' eliminadas
   Registros eliminados: 1
```

5. **Eliminar las relaciones Amigo _de para un usuario que quiera**

```
usuario_nombre = input("Ingrese el nombre del usuario para eliminar relaciones AMIGO_DE: ")

with driver.session() as session:
    # Verificar si el usuario existe
    existe = session.run(
        "MATCH (u:Usuario {nombre: $nombre}) RETURN u",
        nombre=usuario_nombre
    ).single()

    if existe:
        # Eliminar relaciones AMIGO_DE en cualquier dirección
        resultado = session.run(
            """
            MATCH (u:Usuario {nombre: $nombre})-[r:AMIGO_DE]-(otro:Usuario)
            DELETE r
            RETURN count(r) AS eliminadas
            """,
            nombre=usuario_nombre
        ).single()

        cantidad = resultado["eliminadas"]

        if cantidad > 0:
            print(f"✅ Se eliminaron {cantidad} relaciones AMIGO_DE del usuario '{usuario_nc
        else:
            print(f"ℹ️ El usuario '{usuario_nombre}' no tenía relaciones AMIGO_DE.")
    else:
        print(f"El usuario '{usuario_nombre}' no posee relaciones o no se encuentra en la ba
```

```
Ingrese el nombre del usuario para eliminar relaciones AMIGO_DE:  Juan López
✅ Se eliminaron 2 relaciones AMIGO_DE del usuario 'Juan López'.
```

4. Conclusión

El proyecto permitió alcanzar los objetivos propuestos: crear una estructura distribuida, integrar distintos tipos de consultas y generar visualizaciones claras de los datos. Este trabajo ayudó a afianzar los contenidos vistos en la materia, a ponerlos en práctica en un caso concreto y a experimentar con un escenario cercano a los sistemas reales de gestión turística y recomendación.