ÍNDICE

Librerías a instalar

- Entorno virtual
    - virtualenv venv
    - source venv/bin/activate
    - code .

Librerías a instalar:

- PyMongo:
    - pip install pymongo

- Mysql Connector:
    - pip install mysql-connector-python

- ZODB:
    - pip install ZODB
    -
- PyMysql:
    - pip install pymysql

- Peewee:
    - pip install peewee pymysql

- Persistent:
    - pip install persistent

- Transaction:
    - pip install transaction

- Cryptography:
    - pip install cryptography

Comandos desde la terminal de mongo

Para entrar en mongo desde la terminal:
- sudo docker start mongo
- mongosh
- mongod

Para entrar en mysql desde la terminal:
- sudo mysql

Para ver los usuarios:
- show users

Para ver las bases de datos:
- show dbs

Usar o crear una base de datos:
- use nombre_base_datos

Mostrar colecciones:
- show collections

Crear una colección:
- db.createCollection("miColeccion")

Ver registros de una colección:
- db.nombre_coleccion.find().pretty()

Borrar una colección:
- db.nombre_coleccion.drop()

Crear un usuario, debe de estar en admin:
- use admin

```
db.createUser({
 user: "usuario",
 pwd: "usuario",
 roles: [{ role: "dbOwner", db: "1dam" }]
})
```

Conexión a la base de datos con Mongodb

```
from pymongo import MongoClient, errors

usuario = "usuario"
clave = "usuario"
base_datos = "1dam"
host = "localhost"
puerto = 27017

try:
    # Intentar conectarse al servidor MongoDB
    client = MongoClient(
        f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}",
        serverSelectionTimeoutMS=5000
    )

    # Seleccionar la base de datos
    db = client[base_datos]

    # Intentar acceder a la base de datos para verificar la conexión
    colecciones = db.list_collection_names()
    print("Conexión exitosa. Colecciones en la base de datos:")
    print(colecciones)

except errors.ServerSelectionTimeoutError as err:
    # Este error ocurre si el servidor no está disponible o no se puede conectar
    print(f"No se pudo conectar a MongoDB: {err}")

except errors.OperationFailure as err:
    # Este error ocurre si las credenciales son incorrectas o no se tienen los permisos
necesarios
    print(f"Fallo en la autenticación o permisos insuficientes: {err}")

except Exception as err:
    # Manejar cualquier otro error inesperado
    print(f"Ocurrió un error inesperado: {err}")

finally:
    # Cerrar la conexión si se estableció correctamente
    if 'client' in locals():
        client.close()
        print("Conexión cerrada.")
```

Insertar datos desde python

```
from pymongo import MongoClient, errors

# Datos de conexión
usuario = "usuario"
clave = "usuario"
host = "localhost"
puerto = 27017

try:
    # Conectar al servidor MongoDB
    client = MongoClient(f"mongodb://{usuario}:{clave}@{host}:{puerto}/",
serverSelectionTimeoutMS=5000)

    # Seleccionar o crear la base de datos
    db = client["1dam"]

    # Crear la colección e insertar datos
    comidas_data = [
        {"nombre": "Pizza", "ingrediente_principal": "Harina", "origen": "Italia",
"tipo_platos": "Plato principal", "tipo_preparacion": "Asado"},
        {"nombre": "Sushi", "ingrediente_principal": "Arroz", "origen": "Japón", "tipo_platos":
"Entrante", "tipo_preparacion": "Crudo"},
        {"nombre": "Hamburguesa", "ingrediente_principal": "Carne", "origen": "Estados Unidos",
"tipo_platos": "Plato principal", "tipo_preparacion": "Frito"},
        {"nombre": "Paella", "ingrediente_principal": "Arroz", "origen": "España",
"tipo_platos": "Plato principal", "tipo_preparacion": "Cocido"},
        {"nombre": "Ceviche", "ingrediente_principal": "Pescado crudo", "origen": "Perú",
"tipo_platos": "Entrante", "tipo_preparacion": "Crudo"}
    ]

# Insertar múltiples documentos en la colección
result = db.Comidas.insert_many(comidas_data)

# Consultar solo dos campos (nombre y origen)
comidas = db.Comidas.find({}, {"_id": 0, "nombre": 1, "origen": 1})

# Proyección para mostrar solo 'nombre' y 'origen'
print("Comidas (solo nombre y origen):")
for comida in comidas: print(comida)

print("Todas las comidas en la colección:")
for comida in comidas: print(comida)

except errors.ServerSelectionTimeoutError as err:
    print(f"No se pudo conectar a MongoDB: {err}")
except errors.OperationFailure as err:
    print(f"Fallo en la autenticación o permisos insuficientes: {err}")
except Exception as err:
    print(f"Ocurrió un error inesperado: {err}")
finally:
    # Cerrar la conexión
    if 'client' in locals():
        client.close()
        print("Conexión cerrada.")
```

CRUD con Mongo

Insertar datos, modificar y eliminar usando funciones:

```
from pymongo import MongoClient, errors

# Datos de conexión
usuario = "usuario"
clave = "usuario"
base_datos = "1dam"
host = "localhost"
puerto = 27017

# Función para conectarse a MongoDB
def conectar_mongo(usuario, clave, host, puerto, base_datos):
    try:
        # Intentar conectarse al servidor MongoDB
        client = MongoClient(
            f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}",
            serverSelectionTimeoutMS=5000
        )
        # Seleccionar la base de datos
        db = client[base_datos]
        return client, db
    except errors.ServerSelectionTimeoutError as err:
        print(f"No se pudo conectar a MongoDB: {err}")
    except errors.OperationFailure as err:
        print(f"Fallo en la autenticación o permisos insuficientes: {err}")
    except Exception as err:
        print(f"Ocurrió un error inesperado: {err}")
    return None, None

# Función para añadir nuevas comidas
def anadir_comidas(coleccion_comidas, nuevas_comidas):
    try:
        # Insertar los nuevos documentos
        resultado_insertar = coleccion_comidas.insert_many(nuevas_comidas)
        print("Comidas añadidas con éxito:", resultado_insertar.inserted_ids)
    except Exception as err:
        print(f"Error al añadir comidas: {err}")

# Función para actualizar un documento
def actualizar_comida(coleccion_comidas, consulta_actualizar, nuevo_valor):
    try:
        # Actualizar un campo de un solo documento
        resultado_actualizar = coleccion_comidas.update_one(consulta_actualizar, nuevo_valor)
        print("Documentos actualizados:", resultado_actualizar.modified_count)
    except Exception as err:
        print(f"Error al actualizar comida: {err}")

# Función para eliminar un documento
def eliminar_comida(coleccion_comidas, consulta_eliminar):
    try:
        # Eliminar uno de los documentos
        resultado_eliminar = coleccion_comidas.delete_one(consulta_eliminar)
        print("Documentos eliminados:", resultado_eliminar.deleted_count)
```

```
    except Exception as err:
        print(f"Error al eliminar comida: {err}")

# Función para cerrar la conexión
def cerrar_conexion(client):
    if client:
        client.close()
        print("Conexión cerrada.")

# Función principal que organiza todo el flujo
def main():
    # Conexión a MongoDB
    client, db = conectar_mongo(usuario, clave, host, puerto, base_datos)
    if client is None or db is None:
        return

    coleccion_comidas = db.Comidas

    # Añadir nuevas comidas
    nuevas_comidas = [
        {"nombre": "Lasagna", "ingrediente_principal": "Pasta", "origen": "Italia",
"tipo_platos": "Plato principal", "tipo_preparacion": "Horneado"},
        {"nombre": "Ceviche", "ingrediente_principal": "Pescado crudo", "origen": "Perú",
"tipo_platos": "Entrante", "tipo_preparacion": "Crudo"},
        {"nombre": "Burritos", "ingrediente_principal": "Carne", "origen": "México",
"tipo_platos": "Entrante", "tipo_preparacion": "Frito"}
    ]
    anadir_comidas(coleccion_comidas, nuevas_comidas)

    # Actualizar un campo de un documento
    consulta_actualizar = {"nombre": "Ceviche"}
    nuevo_valor = {"$set": {"tipo_preparacion": "Hervido"}}  # Actualizar tipo_preparacion de
Sushi
    actualizar_comida(coleccion_comidas, consulta_actualizar, nuevo_valor)

    # Eliminar un documento
    consulta_eliminar = {"nombre": "Burritos"}
    eliminar_comida(coleccion_comidas, consulta_eliminar)

    # Cerrar la conexión
    cerrar_conexion(client)

# Ejecutar la función principal
if __name__ == "__main__":
    main()
```

Ideas de ejercicios con Mongo

CSV to Mongo

```
import csv
from pymongo import MongoClient, errors

def insertar_csv_en_mongo(client, base_datos, ruta_csv, coleccion_nombre):
    try:
        # Seleccionar base de datos y colección
        db = client[base_datos]
        coleccion = db[coleccion_nombre]

        # Leer archivo CSV
        with open(ruta_csv, mode='r', encoding='utf-8') as archivo:
            lector_csv = csv.DictReader(archivo)  # Lee el CSV como un diccionario
            datos = [fila for fila in lector_csv]  # Cargar todas las filas como una lista de
diccionarios

            # Insertar datos en la colección
            if datos:
                coleccion.insert_many(datos)  # Inserta múltiples documentos
                print(f"Se insertaron {len(datos)} documentos en la colección
'{coleccion_nombre}'.")
            else:
                print("El archivo CSV está vacío o no tiene datos válidos.")

    except Exception as e:
        print(f"Error al procesar el archivo CSV o insertar datos: {e}")

### Conexión a MongoDB
try:
    # Parámetros de conexión
    usuario = "usuario"
    clave = "usuario"
    host = "localhost"
    puerto = 27017
    base_datos = "prueba"
    ruta_csv = "archivo.csv"  # Ruta del archivo CSV
    coleccion_nombre = "CSV"  # Nombre de la colección en MongoDB

    # Crear cliente MongoDB
    client = MongoClient(
        f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}",
        serverSelectionTimeoutMS=5000
    )

    # Verificar conexión
    db = client[base_datos]
    print("Conexión a MongoDB exitosa.")

    # Llamar a la función para insertar datos
    insertar_csv_en_mongo(client, base_datos, ruta_csv, coleccion_nombre)

except errors.ServerSelectionTimeoutError as err:
```

```
    print("No se pudo conectar a MongoDB. Verifica los datos de conexión.")
    print(f"Error: {err}")

except Exception as e:
    print(f"Error inesperado: {e}")

finally:
    # Cerrar conexión al cliente MongoDB
    client.close()
    print("Conexión a MongoDB cerrada.")
```

JSON to Mongo

```
import json
from pymongo import MongoClient, errors

def insertar_json_en_mongo(client, base_datos, ruta_json, coleccion_nombre):
    try:
        # Seleccionar base de datos y colección
        db = client[base_datos]
        coleccion = db[coleccion_nombre]

        # Leer archivo JSON
        with open(ruta_json, mode='r', encoding='utf-8') as archivo:
            datos = json.load(archivo)  # Cargar el archivo JSON como una lista de diccionarios

            # Verificar que los datos sean una lista
            if isinstance(datos, list):
                # Insertar datos en la colección
                if datos:
                    coleccion.insert_many(datos)  # Inserta múltiples documentos
                    print(f"Se insertaron {len(datos)} documentos en la colección
'{coleccion_nombre}'.")
                else:
                    print("El archivo JSON está vacío o no tiene datos válidos.")
            else:
                print("El archivo JSON no contiene una lista de objetos.")

    except Exception as e:
        print(f"Error al procesar el archivo JSON o insertar datos: {e}")

### Conexión a MongoDB
try:
    # Parámetros de conexión
    usuario = "usuario"
    clave = "usuario"
    host = "localhost"
    puerto = 27017
    base_datos = "prueba"
    ruta_json = "archivo.json"  # Ruta del archivo JSON
    coleccion_nombre = "JSON"  # Nombre de la colección en MongoDB

    # Crear cliente MongoDB
    client = MongoClient(
        f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}",
```

```
        serverSelectionTimeoutMS=5000
    )

    # Verificar conexión
    db = client[base_datos]
    print("Conexión a MongoDB exitosa.")

    # Llamar a la función para insertar datos
    insertar_json_en_mongo(client, base_datos, ruta_json, coleccion_nombre)

except errors.ServerSelectionTimeoutError as err:
    print("No se pudo conectar a MongoDB. Verifica los datos de conexión.")
    print(f"Error: {err}")

except Exception as e:
    print(f"Error inesperado: {e}")

finally:
    # Cerrar conexión al cliente MongoDB
    client.close()
    print("Conexión a MongoDB cerrada.")
```

Escribir de Mongo a CSV

```
import csv
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure

# Función para escribir los documentos de MongoDB en un archivo CSV
def escribir_csv(usuario, clave, base_datos, host, puerto, coleccion_nombre,
archivo_csv):
    try:
        # Intentar conectarse al servidor MongoDB
        client = MongoClient(
            f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}",
            serverSelectionTimeoutMS=5000  # Tiempo de espera para la conexión
        )

        # Conexión a la base de datos y colección
        db = client[base_datos]
        coleccion = db[coleccion_nombre]

        # Obtener los documentos de la colección
        documentos = coleccion.find()

        # Escribir los documentos en el archivo CSV
        with open(archivo_csv, mode='w', newline='', encoding='utf-8') as file:
            writer = csv.DictWriter(file, fieldnames=documentos[0].keys())  # Toma
las claves del primer documento como encabezados
            writer.writeheader()  # Escribir los encabezados
            for documento in documentos:
                # Eliminamos el campo '_id' si no quieres incluirlo en el CSV
                documento.pop('_id', None)
                writer.writerow(documento)

        print(f"Los datos se han guardado en {archivo_csv}")

    except ConnectionFailure:
        print("Error: No se pudo conectar al servidor MongoDB.")
    except Exception as e:
        print(f"Ha ocurrido un error: {e}")

# Datos de conexión
usuario = "usuario"
clave = "usuario"
base_datos = "1dam"
host = "localhost"
puerto = 27017
coleccion_nombre = "Comidas"  # Nombre de la colección
archivo_csv = "salida.csv"  # Nombre del archivo CSV

# Llamada a la función
escribir_csv(usuario, clave, base_datos, host, puerto, coleccion_nombre, archivo_csv)
```

Escribir de Mongo a JSON

```
import json
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure

# Función para escribir los documentos de MongoDB en un archivo JSON
def escribir_json(usuario, clave, base_datos, host, puerto, coleccion_nombre,
archivo_json):
    try:
        # Intentar conectarse al servidor MongoDB
        client = MongoClient(
            f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}",
            serverSelectionTimeoutMS=5000  # Tiempo de espera para la conexión
        )

        # Conexión a la base de datos y colección
        db = client[base_datos]
        coleccion = db[coleccion_nombre]

        # Obtener los documentos de la colección
        documentos = coleccion.find()

        # Escribir los documentos en el archivo JSON
        with open(archivo_json, mode='w', encoding='utf-8') as file:
            # Convertimos el cursor de MongoDB en una lista y la escribimos en el
archivo JSON
            json.dump(list(documentos), file, default=str, ensure_ascii=False,
indent=4)

        print(f"Los datos se han guardado en {archivo_json}")

    except ConnectionFailure:
        print("Error: No se pudo conectar al servidor MongoDB.")
    except Exception as e:
        print(f"Ha ocurrido un error: {e}")

# Datos de conexión
usuario = "usuario"
clave = "usuario"
base_datos = "1dam"
host = "localhost"
puerto = 27017
coleccion_nombre = "Comidas"  # Nombre de la colección
archivo_json = "salida.json"  # Nombre del archivo JSON

# Llamada a la función
escribir_json(usuario, clave, base_datos, host, puerto, coleccion_nombre,
archivo_json)
```

Escribir en JSON y CSV

Se crea un JSON con un diccionario de Comidas , y con el método actualizar_configuracion se cambia para escribir en CSV

```
import json
import csv
import logging
from copy import deepcopy

# Configuración de logging para guardar los mensajes en un archivo
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("log_datos.log"),  # Guardar logs en el archivo
    ]
)

class DataManager:
    def __init__(self, ruta_archivo, tipo_archivo='json'):
        self.ruta_archivo = ruta_archivo
        self.tipo_archivo = tipo_archivo
        self.version = 1
        self.transaccion_activa = False
        self.copia_datos = None
        self.datos = self._leer_archivo() if self._existe_archivo() else []

    def _existe_archivo(self):
        try:
            with open(self.ruta_archivo, 'r'):
                return True
        except FileNotFoundError:
            return False

    def _leer_archivo(self):
        if self.tipo_archivo == 'json':
            with open(self.ruta_archivo, 'r') as archivo:
                return json.load(archivo)
        elif self.tipo_archivo == 'csv':
            datos = []
            with open(self.ruta_archivo, mode='r') as archivo:
                lector = csv.DictReader(archivo)
                for fila in lector:
                    datos.append(fila)
            return datos

    def _guardar_archivo(self):
        if self.tipo_archivo == 'json':
            with open(self.ruta_archivo, 'w') as archivo:
                json.dump(self.datos, archivo, indent=4)
        elif self.tipo_archivo == 'csv' and self.datos:
            with open(self.ruta_archivo, mode='w', newline='') as archivo:
                escritor = csv.DictWriter(archivo, fieldnames=self.datos[0].keys())
                escritor.writeheader()
                escritor.writerows(self.datos)
        logging.info(f"Archivo {self.tipo_archivo.upper()} guardado. Versión actual:
```

```
{self.version}")

    def iniciar_transaccion(self):
        if self.transaccion_activa:
            raise Exception("Ya hay una transacción activa.")
        self.transaccion_activa = True
        self.copia_datos = deepcopy(self.datos)
        logging.info("Transacción iniciada.")

    def confirmar_transaccion(self):
        if not self.transaccion_activa:
            raise Exception("No hay una transacción activa para confirmar.")
        self.version += 1
        self.transaccion_activa = False
        self.copia_datos = None
        self._guardar_archivo()
        logging.info("Transacción confirmada y cambios guardados.")

    def revertir_transaccion(self):
        if not self.transaccion_activa:
            raise Exception("No hay una transacción activa para revertir.")
        self.datos = self.copia_datos
        self.transaccion_activa = False
        self.copia_datos = None
        logging.warning("Transacción revertida. Los cambios no se guardaron.")

    def escribir_dato(self, nuevo_dato):
        if not self.transaccion_activa:
            raise Exception("Debe iniciar una transacción antes de realizar cambios.")
        self.datos.append(nuevo_dato)
        logging.info(f"Dato agregado: {nuevo_dato}")

    def actualizar_configuracion(self, nueva_ruta, nuevo_tipo=None):
        if self.transaccion_activa:
            raise Exception("No se puede cambiar la configuración durante una transacción.")
        self.ruta_archivo = nueva_ruta
        if nuevo_tipo:
            self.tipo_archivo = nuevo_tipo
        logging.info(f"Configuración actualizada. Nueva ruta del archivo:
{self.ruta_archivo}")

# Configuración de logging
logging.basicConfig(level=logging.INFO)

# Instancias de comida
comidas = [
    {"nombre": "Sushi", "tipo": "Japonesa", "precio": 20.0, "ingrediente_principal":
"Pescado", "tipo_plato": "Principal"},
    {"nombre": "Helado", "tipo": "Postre", "precio": 5.0, "ingrediente_principal": "Leche",
"tipo_plato": "Postre"},
    {"nombre": "Ensalada César", "tipo": "Internacional", "precio": 10.0,
"ingrediente_principal": "Lechuga", "tipo_plato": "Entrante"}
]

# Uso del DataManager
# Crear instancia para JSON
data_manager = DataManager('comidas.json', 'json')
```

```
# Iniciar transacción y escribir datos
data_manager.iniciar_transaccion()
for comida in comidas:
    data_manager.escribir_dato(comida)
data_manager.confirmar_transaccion()

# Cambiar configuración para CSV y escribir los mismos datos
data_manager.actualizar_configuracion('comidas.csv', 'csv')

# Iniciar nueva transacción para guardar en CSV
data_manager.iniciar_transaccion()
data_manager.confirmar_transaccion()

print("Datos guardados en JSON y CSV con éxito.")
```

CRUD con Mysql

```
import logging
import mysql.connector
from mysql.connector import Error

# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler("databasemanager.log"),  # Logs guardados en un archivo
        logging.StreamHandler(),  # Logs también en consola
    ]
)

class DatabaseManager:
    """Clase para gestionar una base de datos MySQL."""

    def __init__(self, host, user, password, database):
        """Inicializa la conexión a la base de datos."""
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None

    def conectar(self):
        """Establece una conexión con la base de datos."""
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            if self.connection.is_connected():
                logging.info("Conexión exitosa a la base de datos.")
        except Error as e:
            logging.error(f"Error al conectar a la base de datos: {e}")

    def desconectar(self):
        """Cierra la conexión con la base de datos."""
        if self.connection and self.connection.is_connected():
            self.connection.close()
            logging.info("Conexión cerrada.")

    def crear_comida(self, nombre, tipoPlato, origen, precio):
        """Inserta una nueva comida en la base de datos."""
        try:
            cursor = self.connection.cursor()
            query = """
                INSERT INTO Comidas (nombre, tipoPlato, origen, precio)
                VALUES (%s, %s, %s, %s)
            """
            cursor.execute(query, (nombre, tipoPlato, origen, precio))
```

```
            logging.info(f"Comida '{nombre}' insertada exitosamente.")
        except Error as e:
            logging.error(f"Error al insertar la comida '{nombre}': {e}")

    def leer_comidas(self):
        """Recupera todas las comidas de la base de datos."""
        try:
            cursor = self.connection.cursor(dictionary=True)
            cursor.execute("SELECT * FROM Comidas")
            comidas = cursor.fetchall()
            logging.info("Comidas recuperadas:")
            for herramienta in comidas:
                logging.info(herramienta)
            return comidas
        except Error as e:
            logging.error(f"Error al leer las comidas: {e}")
            return None

    def actualizar_comida(self, id, nombre, tipoPlato, origen, precio):
        """Actualiza los datos de una comida en la base de datos."""
        try:
            cursor = self.connection.cursor()
            query = """
                UPDATE Comidas
                SET nombre = %s, tipoPlato = %s, origen = %s, precio = %s
                WHERE id = %s
            """
            cursor.execute(query, (id, nombre, tipoPlato, origen, precio))
            logging.info(f"Comida con ID {id} actualizada exitosamente.")
        except Error as e:
            logging.error(f"Error al actualizar la comida con ID {id}: {e}")

    def eliminar_comida(self, id):
        """Elimina una comida de la base de datos."""
        try:
            cursor = self.connection.cursor()
            query = "DELETE FROM Comidas WHERE id = %s"
            cursor.execute(query, (id,))
            logging.info(f"Comida con ID {id} eliminada exitosamente.")
        except Error as e:
            logging.error(f"Error al eliminar la comida con ID {id}: {e}")

    def iniciar_transaccion(self):
        """Inicia una transacción."""
        try:
            if self.connection.is_connected():
                self.connection.start_transaction()
                logging.info("Transacción iniciada.")
        except Error as e:
            logging.error(f"Error al iniciar la transacción: {e}")

    def confirmar_transaccion(self):
        """Confirma (commit) una transacción."""
        try:
            if self.connection.is_connected():
                self.connection.commit()
                logging.info("Transacción confirmada.")
```

```
        except Error as e:
            logging.error(f"Error al confirmar la transacción: {e}")

    def revertir_transaccion(self):
        """Revierte (rollback) una transacción."""
        try:
            if self.connection.is_connected():
                self.connection.rollback()
                logging.info("Transacción revertida.")
        except Error as e:
            logging.error(f"Error al revertir la transacción: {e}")

# Ejemplo de uso del componente DatabaseManager
if __name__ == "__main__":
    # Instanciar la clase
    db_manager = DatabaseManager("localhost", "usuario", "usuario", "1dam")
    db_manager.conectar()

    # Insertar una nueva comida
    db_manager.crear_comida("Tacos", "Plato Principal", "México", 10.00)
    db_manager.confirmar_transaccion()

    # Leer todas las comidas
    db_manager.leer_comidas()

    # Actualizar una comida
    db_manager.actualizar_comida(1,"Tacos", "Plato Principal", "España", 15.00)
    db_manager.confirmar_transaccion()

    # Eliminar una comida
    db_manager.eliminar_comida(1)
    db_manager.confirmar_transaccion()

    # Manejo de transacciones
    db_manager.iniciar_transaccion()
    db_manager.crear_comida("Pizza", "Plato Principal", "Italia", 20.00)
    db_manager.revertir_transaccion()  # No se guardará la inserción

    # Desconectar
    db_manager.desconectar()
```

CRUD ENTRE DOS TABLAS CON PEEWEE

```
import logging
from peewee import Model, CharField, ForeignKeyField, MySQLDatabase

# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler("databasemanager_orm.log"),
        logging.StreamHandler()
    ]
)

# Configuración de la base de datos MySQL
db = MySQLDatabase(
    "1dam",  # Nombre de la base de datos
    user="usuario",  # Usuario de MySQL
    password="usuario",  # Contraseña de MySQL
    host="localhost",  # Host
    port=3306  # Puerto por defecto de MySQL
)

# Modelos de la base de datos
class Proveedor(Model):
    nombre = CharField()
    direccion = CharField()

    class Meta:
        database = db


class Herramienta(Model):
    nombre = CharField()
    tipo = CharField()
    marca = CharField()
    uso = CharField()
    material = CharField()
    proveedor = ForeignKeyField(Proveedor, backref='herramientas')

    class Meta:
        database = db


# Componente DatabaseManagerORM
class DatabaseManagerORM:
    def __init__(self):
        self.db = db

    def conectar(self):
        """Conecta la base de datos y crea las tablas."""
        self.db.connect()
        self.db.create_tables([Proveedor, Herramienta])
        logging.info("Conexión establecida y tablas creadas.")
```

```
    def desconectar(self):
        """Cierra la conexión a la base de datos."""
        if not self.db.is_closed():
            self.db.close()
            logging.info("Conexión cerrada.")

    def iniciar_transaccion(self):
        """Inicia una transacción."""
        self.db.begin()
        logging.info("Transacción iniciada.")

    def confirmar_transaccion(self):
        """Confirma (commit) una transacción."""
        self.db.commit()
        logging.info("Transacción confirmada.")

    def revertir_transaccion(self):
        """Revierte (rollback) una transacción."""
        self.db.rollback()
        logging.info("Transacción revertida.")

    def crear_proveedor(self, nombre, direccion):
        """Inserta un nuevo proveedor."""
        proveedor = Proveedor.create(nombre=nombre, direccion=direccion)
        logging.info(f"Proveedor creado: {proveedor.nombre} - {proveedor.direccion}")
        return proveedor

    def crear_herramienta(self, nombre, tipo, marca, uso, material, proveedor):
        """Inserta una nueva herramienta."""
        herramienta = Herramienta.create(
            nombre=nombre, tipo=tipo, marca=marca, uso=uso, material=material,
proveedor=proveedor
        )
        logging.info(f"Herramienta creada: {herramienta.nombre} - {herramienta.tipo}")
        return herramienta

    def leer_herramientas(self):
        """Lee todas las herramientas."""
        herramientas = Herramienta.select()
        logging.info("Leyendo herramientas:")
        for herramienta in herramientas:
            logging.info(f"{herramienta.nombre} - {herramienta.tipo}
({herramienta.proveedor.nombre})")
        return herramientas


# Instancio DatabaseManagerORM
db_manager = DatabaseManagerORM()

# Me conecto a la base de datos y crear tablas
db_manager.conectar()

# Creo los proveedores
proveedor_a = db_manager.crear_proveedor("Proveedor A", "123-456-789")
proveedor_b = db_manager.crear_proveedor("Proveedor B", "987-654-321")

# Actualizo el contacto de proveedor A
```

```
proveedor_a.direccion = "30276616Z"
proveedor_a.save()
logging.info(f"Proveedor A actualizado: {proveedor_a.nombre} - {proveedor_a.direccion}")

# Elimino al proveedor B
proveedor_b.delete_instance()
logging.info(f"Proveedor B eliminado: {proveedor_b.nombre}")

# Creo las herramientas para el proveedor A
herramienta_martillo = db_manager.crear_herramienta(
    "Martillo", "Manual", "Stanley", "Construcción", "Acero", proveedor_a
)

herramienta_taladro = db_manager.crear_herramienta(
    "Taladro", "Eléctrico", "Stanley", "Perforación", "Plástico", proveedor_a
)

# Consulto las herramientas asociadas al proveedor A
herramientas = db_manager.leer_herramientas()
for herramienta in herramientas:
    if herramienta.proveedor == proveedor_a:
        logging.info(f"Herramienta asociada a Proveedor A: {herramienta.nombre} -
{herramienta.tipo}")

# Actualizo el registro "martillo"
herramienta_martillo.tipo = "Reforzado"
herramienta_martillo.save()
logging.info(f"Herramienta actualizada: {herramienta_martillo.nombre} -
{herramienta_martillo.tipo}")

# Elimino el registro "taladro"
herramienta_taladro.delete_instance()
logging.info(f"Herramienta eliminada: {herramienta_taladro.nombre}")

db_manager.desconectar()
```

COMPONENTES EN MONGODB

```
import logging
from pymongo import MongoClient
from pymongo.errors import PyMongoError

# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler("databasemanager_documental.log"),  # Logs guardados en un archivo
        logging.StreamHandler(),  # Logs también en consola
    ]
)

class DatabaseManagerDocumental:
    def __init__(self, uri, database_name, collection_name):
        """Inicializa el componente DatabaseManagerDocumental."""
        self.uri = uri
        self.database_name = database_name
        self.collection_name = collection_name
        self.client = None
        self.db = None
        self.collection = None

    def conectar(self):
        """Conectar a la base de datos MongoDB"""
        try:
            self.client = MongoClient(self.uri)
            self.db = self.client[self.database_name]
            self.collection = self.db[self.collection_name]
            logging.info(f"Conectado a MongoDB: {self.database_name}.{self.collection_name}")
        except PyMongoError as e:
            logging.error(f"Error al conectar a MongoDB: {e}")

    def desconectar(self):
        """Cerrar la conexión a MongoDB"""
        if self.client:
            self.client.close()
            logging.info("Conexión a MongoDB cerrada.")

    def crear_documento(self, documento):
        """Insertar un nuevo documento en la colección"""
        try:
            result = self.collection.insert_one(documento)
            logging.info(f"Documento insertado con ID: {result.inserted_id}")
            return result.inserted_id
        except PyMongoError as e:
            logging.error(f"Error al insertar el documento: {e}")

    def leer_documentos(self, filtro={}):
        """Leer documentos de la colección según un filtro"""
        try:
            documentos = list(self.collection.find(filtro))
            logging.info(f"Documentos recuperados: {len(documentos)}")
```

```
            for doc in documentos:
                logging.info(doc)
            return documentos
        except PyMongoError as e:
            logging.error(f"Error al leer los documentos: {e}")
            return []

    def actualizar_documento(self, filtro, actualizacion):
        """Actualizar un documento en la colección"""
        try:
            result = self.collection.update_one(filtro, {"$set": actualizacion})
            if result.modified_count > 0:
                logging.info(f"Documento actualizado: {filtro}")
            else:
                logging.warning(f"No se encontró documento para actualizar: {filtro}")
        except PyMongoError as e:
            logging.error(f"Error al actualizar el documento: {e}")

    def eliminar_documento(self, filtro):
        """Eliminar un documento de la colección"""
        try:
            result = self.collection.delete_one(filtro)
            if result.deleted_count > 0:
                logging.info(f"Documento eliminado: {filtro}")
            else:
                logging.warning(f"No se encontró documento para eliminar: {filtro}")
        except PyMongoError as e:
            logging.error(f"Error al eliminar el documento: {e}")

    def iniciar_transaccion(self):
        """Iniciar una transacción"""
        try:
            self.session = self.client.start_session()
            self.session.start_transaction()
            logging.info("Transacción iniciada.")
        except PyMongoError as e:
            logging.error(f"Error al iniciar la transacción: {e}")

    def confirmar_transaccion(self):
        """Confirmar (commit) una transacción"""
        try:
            if self.session:
                self.session.commit_transaction()
                logging.info("Transacción confirmada.")
        except PyMongoError as e:
            logging.error(f"Error al confirmar la transacción: {e}")

    def revertir_transaccion(self):
        """Revertir (rollback) una transacción"""
        try:
            if self.session:
                self.session.abort_transaction()
                logging.info("Transacción revertida.")
        except PyMongoError as e:
            logging.error(f"Error al revertir la transacción: {e}")

if __name__ == "__main__":
```

```
    # Configurar el componente
    db_manager = DatabaseManagerDocumental(
        uri="mongodb://localhost:27017",
        database_name="1dam",
        collection_name="comidas"
    )

    db_manager.conectar()

    try:
        # Crear documentos dentro de una transacción
        db_manager.iniciar_transaccion()
        db_manager.crear_documento({"nombre": "Paella", "ingrediente_principal": "Arroz",
"origen": "España", "tipo_plato": "Principal"})
        db_manager.crear_documento({"nombre": "Sushi", "ingrediente_principal": "Arroz y
pescado", "origen": "Japón", "tipo_plato": "Principal"})
        db_manager.crear_documento({"nombre": "Moussaka", "ingrediente_principal": "Berenjena y
carne", "origen": "Grecia", "tipo_plato": "Principal"})

        db_manager.confirmar_transaccion()

        # Leer todos los documentos
        db_manager.leer_documentos()

        # Actualizar un documento
        db_manager.iniciar_transaccion()
        db_manager.actualizar_documento({"nombre": "Paella"}, {"origen": "Inglaterra"})
        db_manager.confirmar_transaccion()

        # Eliminar un documento
        db_manager.iniciar_transaccion()
        db_manager.eliminar_documento({"nombre": "Sushi"})
        db_manager.confirmar_transaccion()

    except Exception as e:
        logging.error(f"Error general: {e}")
        db_manager.revertir_transaccion()

    finally:
        db_manager.desconectar()
```

CRUD EN ZODB

```
import logging
import transaction
from ZODB import DB, FileStorage
from persistent import Persistent

# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler("databasemanager_object.log"),  # Logs guardados en un archivo
        logging.StreamHandler(),  # Logs también en consola
    ]
)

class Comidas(Persistent):
    def __init__(self, nombre, ingrediente_principal, origen, tipo_plato):
        self.nombre = nombre
        self.ingrediente_principal = ingrediente_principal
        self.origen = origen
        self.tipo_plato = tipo_plato


class DatabaseManagerObject:
    def __init__(self, filepath="1dam.fs"):
        self.filepath = filepath
        self.db = None
        self.connection = None
        self.root = None
        self.transaccion_iniciada = False

    def conectar(self):
        try:
            storage = FileStorage.FileStorage(self.filepath)
            self.db = DB(storage)
            self.connection = self.db.open()
            self.root = self.connection.root()
            if "comidas" not in self.root:
                self.root["comidas"] = {}
            transaction.commit()
            logging.info("Conexión establecida con ZODB.")
        except Exception as e:
            logging.error(f"Error al conectar a ZODB: {e}")

    def desconectar(self):
        try:
            if self.connection:
                self.connection.close()
            if self.db:
                self.db.close()
            logging.info("Conexión a ZODB cerrada.")
        except Exception as e:
            logging.error(f"Error al cerrar la conexión a ZODB: {e}")
```

```
    def iniciar_transaccion(self):
        try:
            transaction.begin()
            self.transaccion_iniciada = True
            logging.info("Transacción iniciada.")
        except Exception as e:
            logging.error(f"Error al iniciar la transacción: {e}")

    def confirmar_transaccion(self):
        if self.transaccion_iniciada:
            try:
                transaction.commit()
                self.transaccion_iniciada = False
                logging.info("Transacción confirmada.")
            except Exception as e:
                logging.error(f"Error al confirmar la transacción: {e}")

    def revertir_transaccion(self):
        if self.transaccion_iniciada:
            try:
                transaction.abort()
                self.transaccion_iniciada = False
                logging.info("Transacción revertida.")
            except Exception as e:
                logging.error(f"Error al revertir la transacción: {e}")

    def crear_comida(self, id, nombre, ingrediente_principal, origen, tipo_plato):
        try:
            if id in self.root["comidas"]:
                raise ValueError(f"Ya existe una herramienta con ID {id}.")
            self.root["comidas"][id] = Comidas(nombre, ingrediente_principal, origen,
tipo_plato)
            logging.info(f"Comida con ID {id} creada exitosamente.")
        except Exception as e:
            logging.error(f"Error al crear la comida con ID {id}: {e}")

    def leer_comidas(self):
        try:
            comidas = self.root["comidas"]
            for id, comida in comidas.items():
                logging.info(
                    f"ID: {id}, Nombre: {comida.nombre}, Ingrediente:
{comida.ingrediente_principal}, "
                    f"Origen: {comida.origen}, Tipo de plato: {comida.tipo_plato}"
                )
            return comidas
        except Exception as e:
            logging.error(f"Error al leer las comidas: {e}")

    def actualizar_comida(self, id, nombre, ingrediente_principal, origen, tipo_plato):
        try:
            comida = self.root["comidas"].get(id)
            if not comida:
                raise ValueError(f"No existe una comida con ID {id}.")
            comida.nombre = nombre
            comida.ingrediente_principal = ingrediente_principal
```

```
            comida.origen = origen
            comida.tipo_plato = tipo_plato
            logging.info(f"Comida con ID {id} actualizada exitosamente.")
        except Exception as e:
            logging.error(f"Error al actualizar la comida con ID {id}: {e}")

    def eliminar_comida(self, id):
        try:
            if id not in self.root["comidas"]:
                raise ValueError(f"No existe una comida con ID {id}.")
            del self.root["comidas"][id]
            logging.info(f"Comida con ID {id} eliminada exitosamente.")
        except Exception as e:
            logging.error(f"Error al eliminar la comida con ID {id}: {e}")


# Ejemplo de uso
if __name__ == "__main__":
    manager = DatabaseManagerObject()
    manager.conectar()
    try:
        # 1. Crear comidas con transacción
        manager.iniciar_transaccion()
        manager.crear_comida(1, "Sushi", "Arroz", "Japón", "Entrante")
        manager.crear_comida(2, "Pizza", "Harina", "Italia", "Principal")
        manager.crear_comida(3, "Gazpacho", "Tomate", "España", "Entrante")

        manager.confirmar_transaccion()

    except Exception as e:
        logging.error(f"Error general: {e}")
        manager.revertir_transaccion()


    # 2. Mostrar comidas
    manager.leer_comidas()

    try:
        # 3. Insertar un dato con ID que ya existe
        manager.iniciar_transaccion()
        manager.crear_comida(1, "Tortilla", "Huevos", "Francia", "Entrante")

    except Exception as e:
        logging.error(f"Error general: {e}")
        manager.revertir_transaccion()

    # 4. Mostrar comidas
    manager.leer_comidas()


    try:
        # 5.Actualizar una comida con transacción. Actualizo ingrediente principal, Origen y
Tipo de plato
        manager.iniciar_transaccion()
        manager.actualizar_comida(1, "Sushi", "Pescado crudo", "China", "Principal")
        manager.confirmar_transaccion()
```

```
except Exception as e:
    logging.error(f"Error general: {e}")
    manager.revertir_transaccion()

# 6. Mostrar comidas
manager.leer_comidas()


try:
    # 7. Eliminar una comida con ID que no exista con transacción
    manager.iniciar_transaccion()
    manager.eliminar_comida(6)

except Exception as e:
    logging.error(f"Error general: {e}")
    manager.revertir_transaccion()

# 8. Mostrar comidas
manager.leer_comidas()

# Desconecto la base de datos
manager.desconectar()
```