# Conestoga College

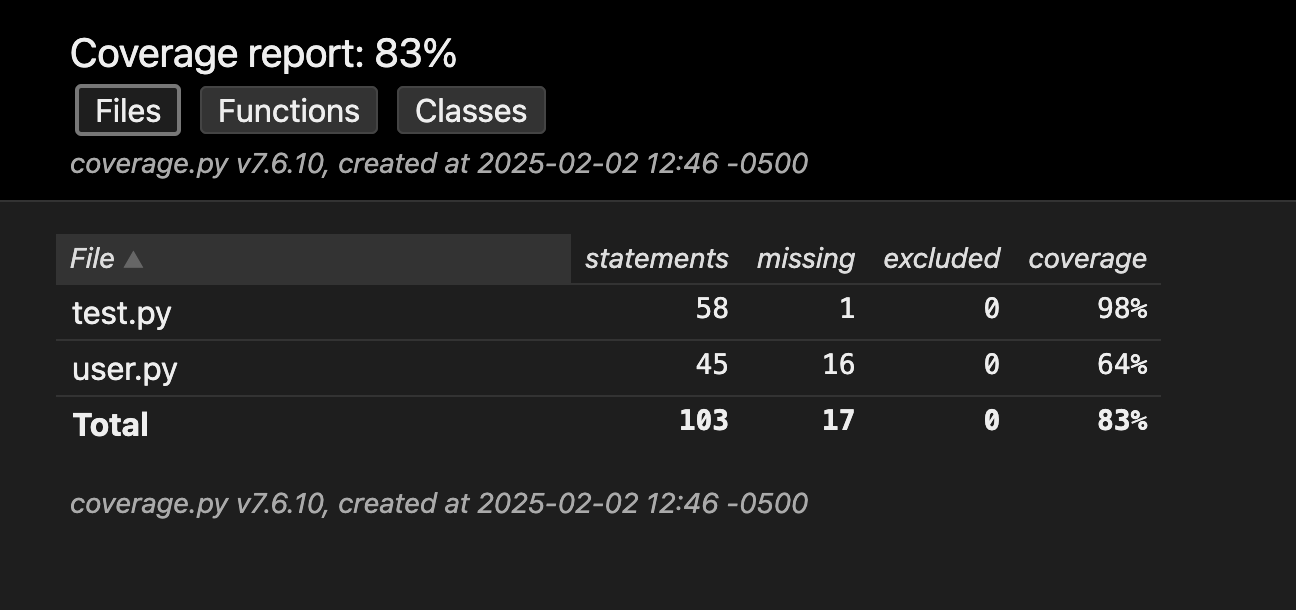
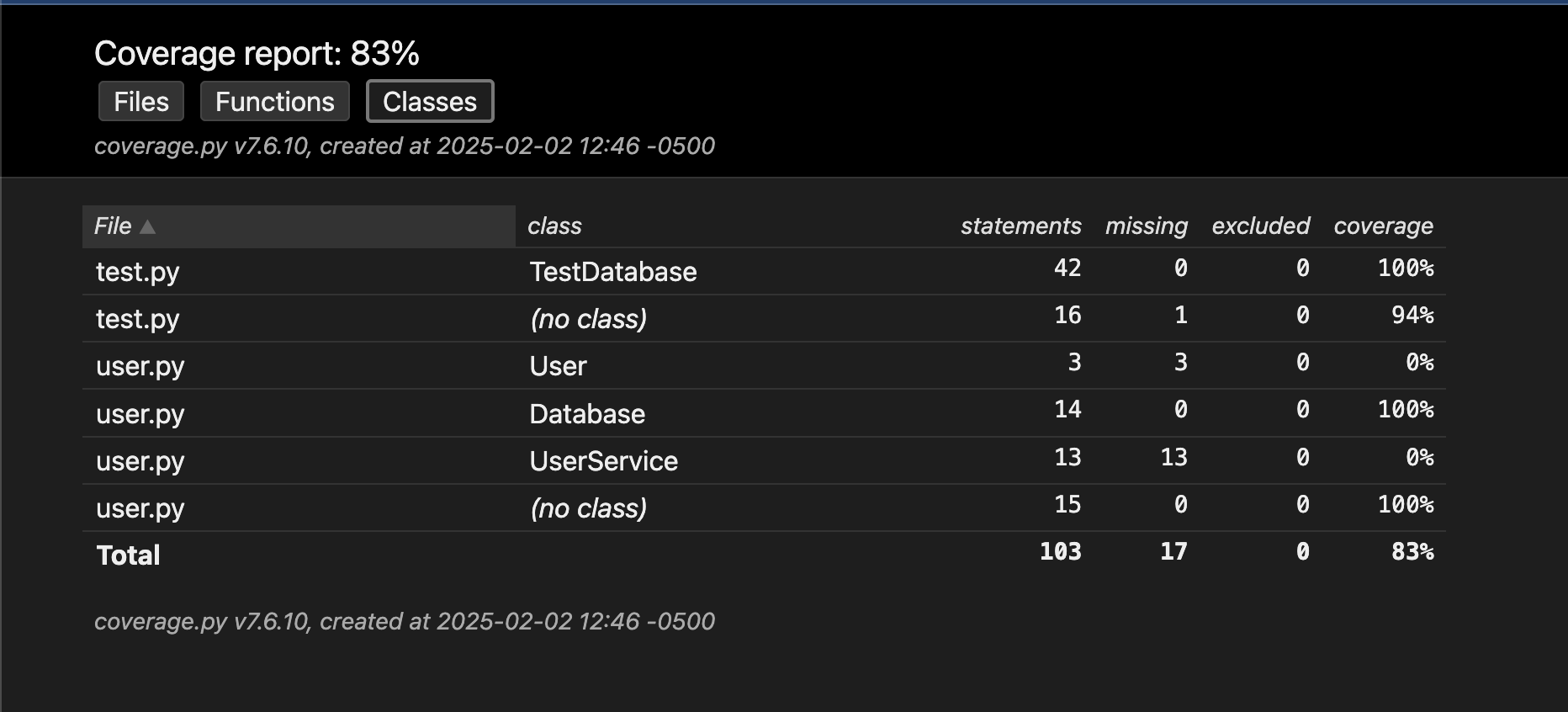
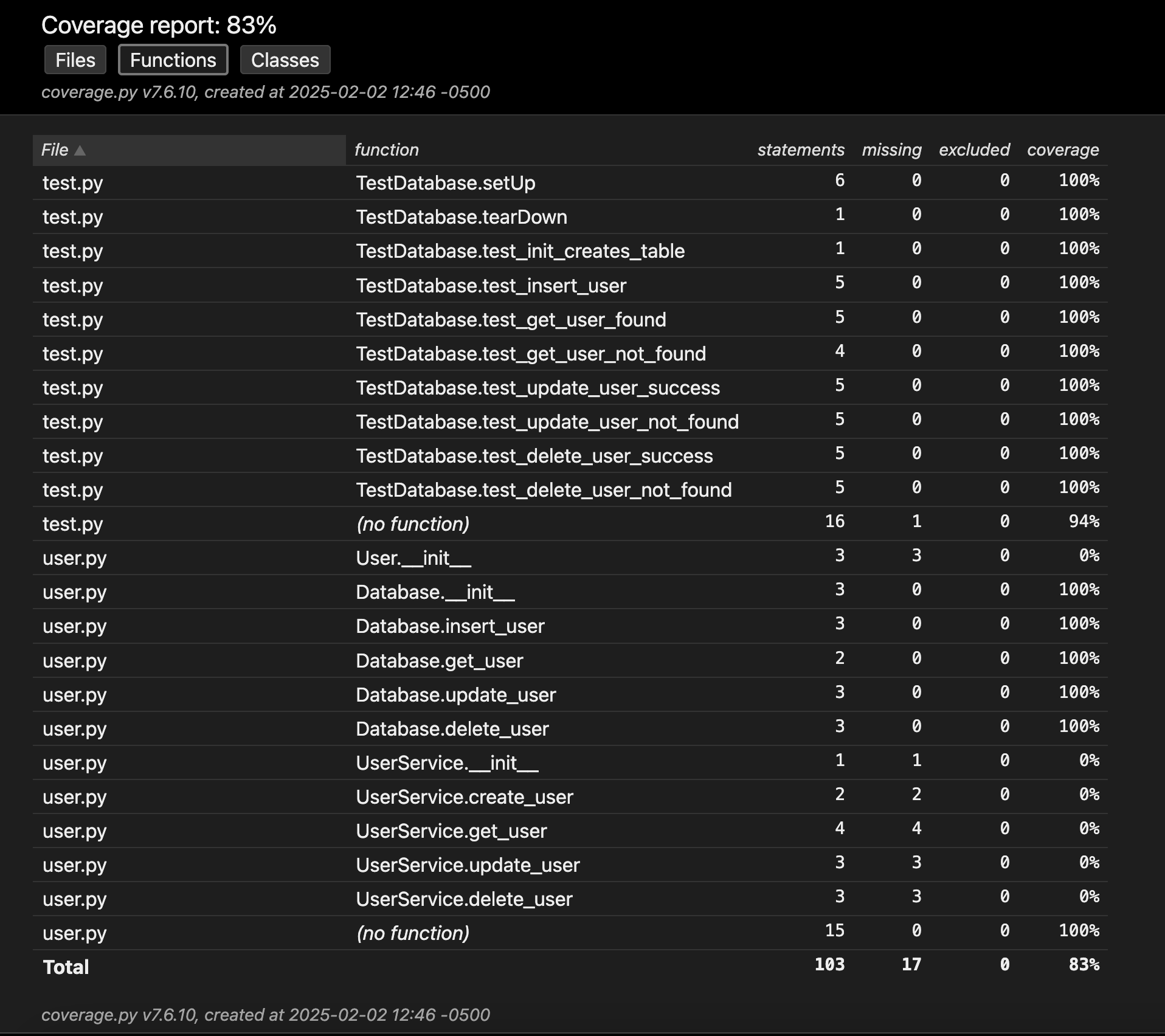
## Assignment 1 Lei Chen-(8945274)

## **Code Coverage and Quality Control**

## PROG8840 - Winter 2025 - Section 1

## **Kumar Halder**

2025/02/22

I. The coverage report   
 

# II. Test report

# SQLite Database Code improvements

1. Implemented proper error handling with custom exceptions

class DatabaseError(Exception):

"""Custom exception for database operations"""

Pass

1. Added input validation for age and name

def checkParams(age, name):

if not isinstance(age, int) or age < 0:

raise ValueError("Age must be a positive integer")

if not name.strip():

raise ValueError("Name cannot be empty")

1. Created a context manager for database connections

@contextmanager

def get\_connection(self):

"""Context manager for database connections"""

conn = sqlite3.connect(self.db\_name)

try:

yield conn

except sqlite3.Error as e:

raise DatabaseError(f"Database operation failed: {e}")

finally:

conn.close()

1. Improved user interface with numbered menu

while True:

print("\n=== User Management System ===")

print("1. Get user")

print("2. Create user")

print("3. Update user")

print("4. Delete user")

print("5. Exit")

1. Added docstrings for better documentation
2. Added database constraints (NOT NULL, CHECK)

def \_init\_db(self) -> None:

"""Initialize the database with required tables"""

with self.get\_connection() as conn:

cursor = conn.cursor()

cursor.execute(

'CREATE TABLE IF NOT EXISTS users (user\_id INTEGER PRIMARY KEY, name TEXT NOT NULL, age INTEGER CHECK (age >= 0))'

)

conn.commit()

1. Implemented safe integer input handling

def get\_integer\_input(prompt: str) -> Optional[int]:

"""Safely get integer input from user"""

try:

return int(input(prompt))

except ValueError:

print("Please enter a valid number")

return None

main.py  
```  
from user import Database, UserService, DatabaseError

import sys

from typing import Optional

def get\_integer\_input(prompt: str) -> Optional[int]:

"""Safely get integer input from user"""

try:

return int(input(prompt))

except ValueError:

print("Please enter a valid number")

return None

def main():

try:

db = Database('users.db')

user\_service = UserService(db)

while True:

print("\n=== User Management System ===")

print("1. Get user")

print("2. Create user")

print("3. Update user")

print("4. Delete user")

print("5. Exit")

choice = get\_integer\_input("\nEnter your choice (1-5): ")

if not choice:

continue

if choice == 5:

break

try:

if choice == 1:

if user\_id := get\_integer\_input("Enter user ID: "):

user\_data, status\_code = user\_service.get\_user(user\_id)

print(f"Response: {

user\_data} (Status Code: {status\_code})")

elif choice == 2:

name = input("Enter user name: ").strip()

if age := get\_integer\_input("Enter user age: "):

user\_data, status\_code = user\_service.create\_user(

name, age)

print(f"Response: {

user\_data} (Status Code: {status\_code})")

elif choice == 3:

if user\_id := get\_integer\_input("Enter user ID: "):

name = input("Enter new name: ").strip()

if age := get\_integer\_input("Enter new age: "):

user\_data, status\_code = user\_service.update\_user(

user\_id, name, age)

print(f"Response: {

user\_data} (Status Code: {status\_code})")

elif choice == 4:

if user\_id := get\_integer\_input("Enter user ID: "):

user\_data, status\_code = user\_service.delete\_user(

user\_id)

print(f"Response: {

user\_data} (Status Code: {status\_code})")

except DatabaseError as e:

print("An error occurred while accessing the database" + str(e))

except Exception as e:

print("An unexpected error occurred: " + str(e))

sys.exit(1)

if \_\_name\_\_ == '\_\_main\_\_':

main()  
```  
  
user.py:  
```  
import sqlite3

from typing import Tuple, Optional, Dict, Any

from contextlib import contextmanager

class DatabaseError(Exception):

"""Custom exception for database operations"""

pass

def checkParams(age, name):

if not isinstance(age, int) or age < 0:

raise ValueError("Age must be a positive integer")

if not name.strip():

raise ValueError("Name cannot be empty")

class User:

def \_\_init\_\_(self, user\_id: int, name: str, age: int):

checkParams(age, name)

self.user\_id = user\_id

self.name = name

self.age = age

class Database:

def \_\_init\_\_(self, db\_name: str):

self.db\_name = db\_name

self.\_init\_db()

@contextmanager

def get\_connection(self):

"""Context manager for database connections"""

conn = sqlite3.connect(self.db\_name)

try:

yield conn

except sqlite3.Error as e:

raise DatabaseError(f"Database operation failed: {e}")

finally:

conn.close()

def \_init\_db(self) -> None:

"""Initialize the database with required tables"""

with self.get\_connection() as conn:

cursor = conn.cursor()

cursor.execute(

'CREATE TABLE IF NOT EXISTS users (user\_id INTEGER PRIMARY KEY, name TEXT NOT NULL, age INTEGER CHECK (age >= 0))'

)

conn.commit()

def insert\_user(self, name: str, age: int) -> int:

"""Insert a new user and return their ID"""

checkParams(age, name)

with self.get\_connection() as conn:

cursor = conn.cursor()

cursor.execute(

'INSERT INTO users (name, age) VALUES (?, ?)', (name, age))

conn.commit()

return cursor.lastrowid

def get\_user(self, user\_id: int) -> Optional[Tuple[int, str, int]]:

with self.get\_connection() as conn:

cursor = conn.cursor()

cursor.execute('SELECT \* FROM users WHERE user\_id = ?', (user\_id,))

return cursor.fetchone()

def update\_user(self, user\_id: int, name: str, age: int) -> bool:

checkParams(age, name)

with self.get\_connection() as conn:

cursor = conn.cursor()

cursor.execute(

'UPDATE users SET name = ?, age = ? WHERE user\_id = ?', (name, age, user\_id))

conn.commit()

return cursor.rowcount > 0

def delete\_user(self, user\_id: int) -> bool:

with self.get\_connection() as conn:

cursor = conn.cursor()

cursor.execute('DELETE FROM users WHERE user\_id = ?', (user\_id,))

conn.commit()

return cursor.rowcount > 0

class UserService:

def \_\_init\_\_(self, db: Database):

self.db = db

def create\_user(self, name: str, age: int) -> Tuple[Dict[str, Any], int]:

try:

user\_id = self.db.insert\_user(name, age)

return {"user\_id": user\_id, "name": name, "age": age}, 201

except ValueError as e:

return {"error": str(e)}, 400

except DatabaseError as e:

return {"error": "Internal server error"}, 500

def get\_user(self, user\_id: int) -> Tuple[Dict[str, Any], int]:

user = self.db.get\_user(user\_id)

if user:

return {"user\_id": user[0], "name": user[1], "age": user[2]}, 200

else:

return {"error": "User not found"}, 404

def update\_user(self, user\_id: int, name: str, age: int) -> Tuple[Dict[str, Any], int]:

try:

if self.db.update\_user(user\_id, name, age):

return {"user\_id": user\_id, "name": name, "age": age}, 200

return {"error": "User not found"}, 404

except ValueError as e:

return {"error": str(e)}, 400

except DatabaseError as e:

return {"error": "Internal server error"}, 500

def delete\_user(self, user\_id: int) -> Tuple[Dict[str, Any], int]:

try:

if self.db.delete\_user(user\_id):

return {"message": "User deleted successfully"}, 200

return {"error": "User not found"}, 404

except DatabaseError as e:

return {"error": "Internal server error"}, 500  
```

test.py  
```  
import unittest

from unittest.mock import Mock, patch

from user import Database

class TestDatabase(unittest.TestCase):

def setUp(self):

# Create mock for sqlite3.connect and cursor

self.cursor\_mock = Mock()

self.conn\_mock = Mock()

self.conn\_mock.cursor.return\_value = self.cursor\_mock

self.cursor\_mock.reset\_mock()

self.conn\_mock.reset\_mock()

self.patcher = patch('sqlite3.connect', return\_value=self.conn\_mock)

self.patcher.start()

self.db = Database('test.db')

def reset\_mock(self):

self.cursor\_mock.reset\_mock()

self.conn\_mock.reset\_mock()

def tearDown(self):

self.patcher.stop()

def test\_init\_creates\_table(self):

self.cursor\_mock.execute.assert\_called\_with(

'CREATE TABLE IF NOT EXISTS users (user\_id INTEGER PRIMARY KEY, name TEXT NOT NULL, age INTEGER CHECK (age >= 0))'

)

def test\_insert\_user(self):

self.reset\_mock()

self.cursor\_mock.lastrowid = 1

result = self.db.insert\_user("John", 30)

self.cursor\_mock.execute.assert\_called\_with(

'INSERT INTO users (name, age) VALUES (?, ?)',

("John", 30)

)

self.conn\_mock.commit.assert\_called\_once()

self.assertEqual(result, 1)

def test\_get\_user\_found(self):

self.reset\_mock()

expected\_user = (1, "John", 30)

self.cursor\_mock.fetchone.return\_value = expected\_user

result = self.db.get\_user(1)

self.cursor\_mock.execute.assert\_called\_with(

'SELECT \* FROM users WHERE user\_id = ?',

(1,)

)

self.assertEqual(result, expected\_user)

def test\_get\_user\_not\_found(self):

self.reset\_mock()

self.cursor\_mock.fetchone.return\_value = None

result = self.db.get\_user(1)

self.cursor\_mock.execute.assert\_called\_with(

'SELECT \* FROM users WHERE user\_id = ?',

(1,)

)

self.assertIsNone(result)

def test\_update\_user\_success(self):

self.reset\_mock()

self.cursor\_mock.rowcount = 1

result = self.db.update\_user(1, "John Updated", 31)

self.cursor\_mock.execute.assert\_called\_with(

'UPDATE users SET name = ?, age = ? WHERE user\_id = ?',

("John Updated", 31, 1)

)

self.conn\_mock.commit.assert\_called\_once()

self.assertTrue(result)

def test\_update\_user\_not\_found(self):

self.reset\_mock()

self.cursor\_mock.rowcount = 0

result = self.db.update\_user(1, "John Updated", 31)

self.cursor\_mock.execute.assert\_called\_with(

'UPDATE users SET name = ?, age = ? WHERE user\_id = ?',

("John Updated", 31, 1)

)

self.conn\_mock.commit.assert\_called\_once()

self.assertFalse(result)

def test\_delete\_user\_success(self):

self.reset\_mock()

self.cursor\_mock.rowcount = 1

result = self.db.delete\_user(1)

self.cursor\_mock.execute.assert\_called\_with(

'DELETE FROM users WHERE user\_id = ?',

(1,)

)

self.conn\_mock.commit.assert\_called\_once()

self.assertTrue(result)

def test\_delete\_user\_not\_found(self):

self.reset\_mock()

self.cursor\_mock.rowcount = 0

result = self.db.delete\_user(1)

self.cursor\_mock.execute.assert\_called\_with(

'DELETE FROM users WHERE user\_id = ?',

(1,)

)

self.conn\_mock.commit.assert\_called\_once()

self.assertFalse(result)

def test\_invalid\_age(self):

with self.assertRaises(ValueError):

self.db.insert\_user("John", -1)

def test\_empty\_name(self):

with self.assertRaises(ValueError):

self.db.insert\_user("", 30)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()  
```