Eduvos
Your Education. Your Future.

Assignment Cover Sheet

Campus	Bedfordview
Faculty	Information Technology
Module Code	ITAPA2-12
Group	Part-Time After Hours
Lecturer's Name	Nkosivile Mazima
Student Full Name	Ruben Da Silva
Student Number	BE.2022.V6T6C1

Declaration:

I declare that this assessment is my own original work except for source material explicitly acknowledged. I also declare that this assessment or any other of my original work related to it has not been previously, or is not being simultaneously, submitted for this or any other course. I am aware of the AI policy and acknowledge that I have not used any AI technology to generate or manipulate data, other than as permitted by the assessment instructions. I also declare that I am aware of the Institution's policy and regulations on honesty in academic work as set out in the Conditions of Enrolment, and of the disciplinary guidelines applicable to breaches of such policy and regulations.

Lecturer's Comments:

Marks Awarded: %

Signature 	**Date:** 04 June 2025

Table of Contents

Admonitions

To keep my similarity score low, I have pasted screenshots of my code instead in the document. If fellow students have followed the project a large portion of the code will be marked as similar, hence the screenshots.

I have also created a virtual environment for my project as I needed dependencies to be installed such as SQLite, JSON, Tkinter etc.

If the screenshots of my code is not clear, I have the code on my GitHub:

https://github.com/CollectingMangos/Cinema-Sales

Question 1: Server script with SQLite DB

For the first part of my project I decided to tackle the database setup and configuration in its own separate file called “database_config.py”.
In this file I have defined the structure for my database as well as the 2 tables (movies & sales) with relevant attributes and foreign keys.
This file will need to be ran first before the server.py or client.py files.

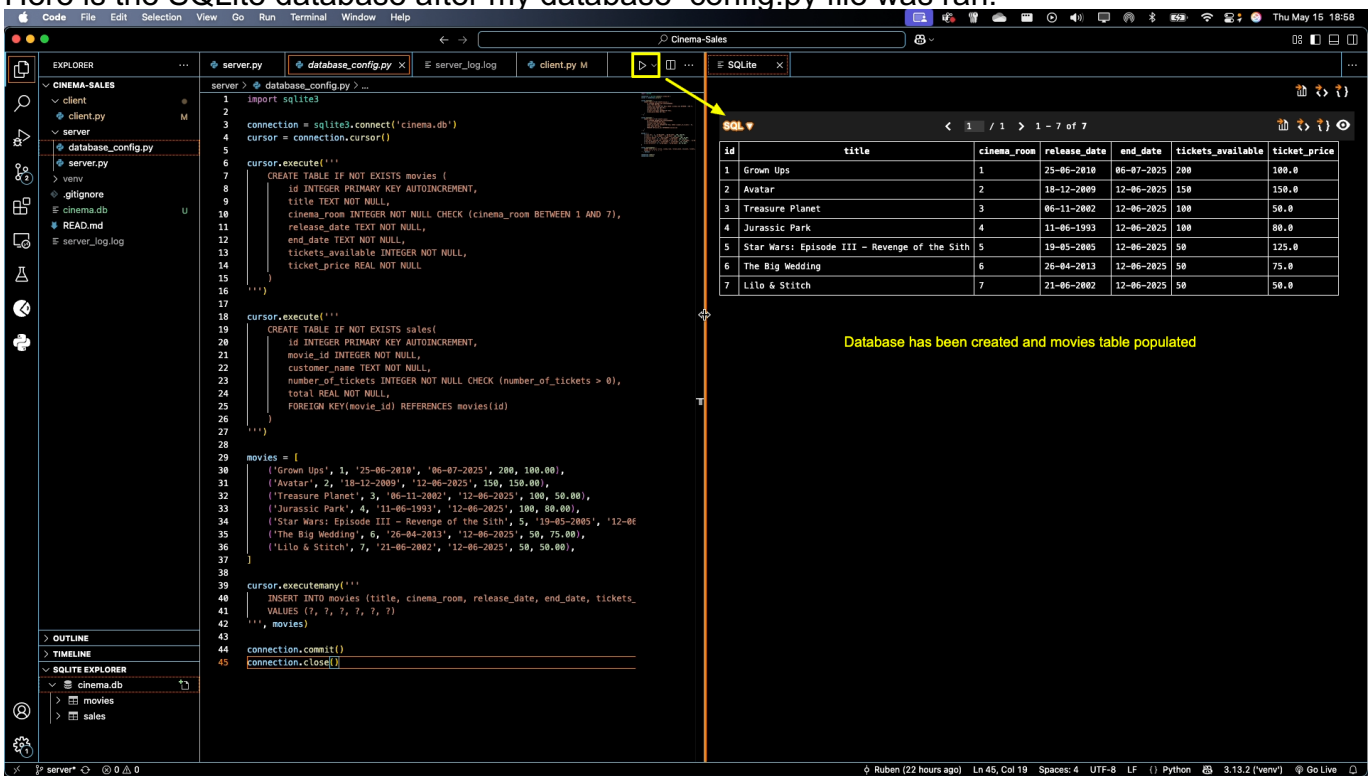
database_config.py:

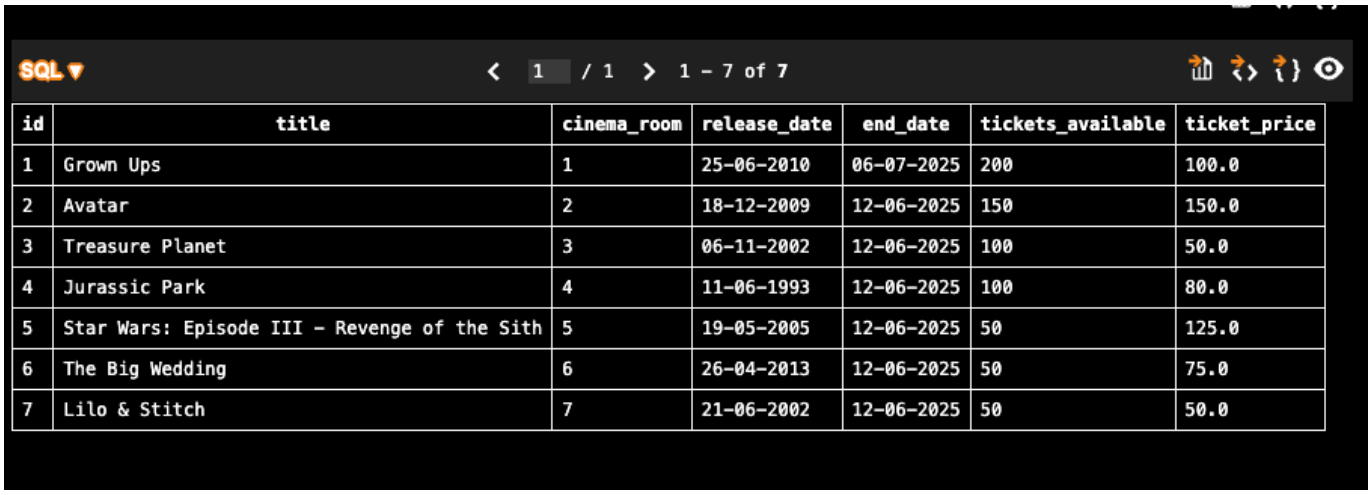
database_config.py

server > database_config.py > ...

```python
import sqlite3

connection = sqlite3.connect('cinema.db')
cursor = connection.cursor()

cursor.execute('''
    CREATE TABLE IF NOT EXISTS movies (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        cinema_room INTEGER NOT NULL CHECK (cinema_room BETWEEN 1 AND 7),
        release_date TEXT NOT NULL,
        end_date TEXT NOT NULL,
        tickets_available INTEGER NOT NULL,
        ticket_price REAL NOT NULL
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS sales(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        movie_id INTEGER NOT NULL,
        customer_name TEXT NOT NULL,
        number_of_tickets INTEGER NOT NULL CHECK (number_of_tickets > 0),
        total REAL NOT NULL,
        FOREIGN KEY(movie_id) REFERENCES movies(id)
    )
''')

movies = [
    ('Grown Ups', 1, '25-06-2010', '06-07-2025', 200, 100.00),
    ('Avatar', 2, '18-12-2009', '12-06-2025', 150, 150.00),
    ('Treasure Planet', 3, '06-11-2002', '12-06-2025', 100, 50.00),
    ('Jurassic Park', 4, '11-06-1993', '12-06-2025', 100, 80.00),
    ('Star Wars: Episode III - Revenge of the Sith', 5, '19-05-2005', '12-06-2025', 50, 125.00),
    ('The Big Wedding', 6, '26-04-2013', '12-06-2025', 50, 75.00),
    ('Lilo & Stitch', 7, '21-06-2002', '12-06-2025', 50, 50.00),
]

cursor.executemany('''
    INSERT INTO movies (title, cinema_room, release_date, end_date, tickets_available, ticket_price)
    VALUES (?, ?, ?, ?, ?, ?)
''', movies)

connection.commit()
connection.close()
```

Here is the SQLite database after my database config.py file was ran:





SQL ▾ 1 / 1 1 - 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	200	100.0
2	Avatar	2	18-12-2009	12-06-2025	150	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III - Revenge of the Sith	5	19-05-2005	12-06-2025	50	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
7	Lilo & Stitch	7	21-06-2002	12-06-2025	50	50.0

For the next part of my project I decided to tackle on the server side of this architecture. This server.py file will manage the connection into the database and allow it to perform particular operations like retrieving the list of movies, adding a new movie, updating details of a movie, deleting a movie, updating the number of tickets for a movie, recording a ticket sale in the sales table and update the number of tickets left for a movie.

Below I have a screenshot of my **very basic server.py file** which contains some logging but most importantly, has the bare minimum to run the server.

server.py

server > server.py > ...

```python
import socket
import sqlite3
import logging

logging.basicConfig(
    filename='server_log.log',
    level=logging.DEBUG,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

PORT = 5050
SERVER = '127.0.0.1'
ADDRESS = (SERVER, PORT)

logging.info('Server is starting up')
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDRESS)

server.listen()
print(f'[STARTING] Listening on server: {SERVER} & port: {PORT}')
logging.info(f'Server is now listening on {SERVER}:{PORT}')

while True:
    client, address = server.accept()
    print(f'New connection from address: {address}')
    logging.info(f'New connection from {address}')

    request = client.recv(1024).decode()
    print(f'[RECEIVED] {request}')
    logging.debug(f'Received request from {address}:{request}')

    client.send('Message received.'.encode())
    logging.info(f'Sent response to {address}')

    client.close()
    logging.info(f'Connection closed to {address}')
```

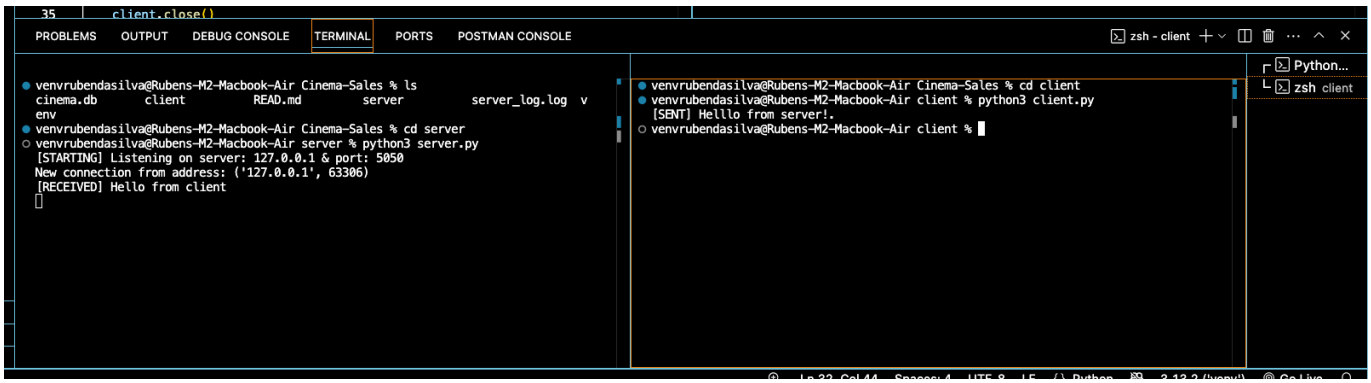
Below is the screenshot for the very basic client.py file which I needed to show that the server can connect and receive a client. **This is not the final file which will be attached later:**

server.py | client.py M

client > client.py > ...

```python
import socket

PORT = 5050
SERVER = '127.0.0.1'
ADDRESS = (SERVER, PORT)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDRESS)

client.send('If this comes through, the test worked'.encode())
print(client.recv(1024).decode())
```

Below are screenshots of the server running as well as a request from my client.py file **to test**:



Now that the basics are up and running my next objective was to create the functions that will handle the requests and execute the relevant SQL queries to the data in the database.

This function is to connect to my database:

```python
def get_db_connection():
    db_path = os.path.abspath(os.path.join(os.path.dirname(__file__), 'cinema.db'))
    logging.debug(f'Database path: {db_path}')
    return sqlite3.connect(db_path)
```

This function is my handler function which is responsible for calling other functions with inputs from the request:

```
def handle_requests(request):
    try:
        operation = request.get('operation')
        if operation == 'get_movies':
            return get_movies()
        elif operation == 'delete_movie':
            return delete_movie(request.get('title'))
        elif operation == 'add_movie':
            return add_movie(
                request.get('title'),
                int(request.get('cinema_room')),
                request.get('release_date'),
                request.get('end_date'),
                int(request.get('tickets_available')),
                float(request.get('ticket_price'))
                )
        elif operation == 'update_movie_details':
            return update_movie_details(
                request.get('title'),
                int(request.get('cinema_room')),
                request.get('release_date'),
                request.get('end_date'),
                float(request.get('ticket_price'))
                )
        elif operation == 'update_tickets_of_movie':
            return update_tickets_of_movie(request.get('title'),int(request.get('tickets_available')))
        elif operation == 'record_ticket_sale':
            return record_ticket_sale(
                request.get('title'),
                request.get('customer_name'),
                int(request.get('number_of_tickets')),
                float(request.get('total'))
            )
        else:
            return {'status': 'failed', 'message': 'Invalid request'}
    except json.JSONDecodeError:
        return {'status': 'failed', 'message': 'Invalid JSON format'}
    except Exception as e:
        return {'status': 'error', 'message': f'Unexpected server error: {e}'}
```

This is the first function to retrieve all the movies:

```
def get_movies():
    try:
        connection = get_db_connection()
        cursor = connection.cursor()
        cursor.execute('SELECT * FROM movies')
        movies = cursor.fetchall()
        return {'status':'success', 'movies': movies}
    except Exception as e:
        logging.error(f'Error fetching movies: {e}')
        return {'status':'error', 'message': 'Failed to get all movies'}
    finally:
        connection.close()
```

This next function is to add a movie to the database:

```python
def add_movie(title, cinema_room, release_date, end_date, tickets_available, ticket_price):
    try:
        title = title.strip()
        connection = get_db_connection()
        cursor = connection.cursor()
        if not (1 <= cinema_room <= 7):
            return {'status': 'error', 'message': 'Cinema room must be between 1 and 7!'}
        if tickets_available < 0:
            return {'status': 'error', 'message': 'Tickets available must be greater than 0!'}
        if ticket_price < 0:
            return {'status': 'error', 'message': 'Ticket price must be greater than 0!'}
        cursor.execute('SELECT title FROM movies WHERE title = ?', (title,))
        if cursor.fetchone():
            return {'status': 'error', 'message': f'Movie: {title} already exists!'}
        cursor.execute('SELECT cinema_room FROM movies WHERE cinema_room = ?', (cinema_room,))
        if cursor.fetchone():
            return {'status': 'error', 'message': f'Cinema room {cinema_room} is already in use!'}
        cursor.execute('''
            INSERT INTO movies (title, cinema_room, release_date, end_date, tickets_available, ticket_price)
            VALUES (?, ?, ?, ?, ?, ?)
        ''', (title, cinema_room, release_date, end_date, tickets_available, ticket_price))
        connection.commit()
        logging.debug(f'Movie added: {title}')
        return {'status': 'success', 'message': f'Movie: {title} has been added successfully'}
    except Exception as e:
        logging.error(f'Error adding the movie: {title}. {e}')
        return {'status': 'error', 'message': f'Failed to add the movie: {title}'}
    finally:
        connection.close()
```

This next function is to update all movie details except for the ‘tickets_available’ since there is a separate function for that:

```python
def update_movie_details(title, cinema_room, release_date, end_date, ticket_price):
    try:
        connection = get_db_connection()
        cursor = connection.cursor()
        cursor.execute('SELECT title FROM movies WHERE title = ?', (title,))
        if not cursor.fetchone():
            logging.debug(f'Movie not found: {title}')
            return {'status': 'error', 'message': f'Movie: {title} not found!'}
        cursor.execute('''
            UPDATE movies
            SET cinema_room = ?, release_date = ?, end_date = ?, ticket_price = ?
            WHERE title = ?
        ''', (cinema_room, release_date, end_date, ticket_price, title))
        connection.commit()
        logging.debug(f'Movie details updated: {title}')
        return {'status': 'success', 'message': f'Movie: {title} has been updated successfully!'}
    except Exception as e:
        logging.error(f'Error updating movie details: {title}. {e}')
        return {'status': 'error', 'message': f'Failed to update movie: {title}'}
    finally:
        connection.close()
```

The next function is to delete a movie from the database:

```
def delete_movie(title):
    try:
        connection = get_db_connection()
        cursor = connection.cursor()
        cursor.execute('SELECT title FROM movies WHERE title = ?', (title,))
        if cursor.fetchone():
            cursor.execute('DELETE FROM movies WHERE title = ?', (title,))
            connection.commit()
            logging.debug(f'Movie deleted: {title}')
            return {'status': 'success', 'message': f'Movie: {title} has been deleted successfully'}
        else:
            logging.debug(f'Movie not found: {title}')
            return {'status': 'error', 'message': f'Movie: {title} not found'}
    except Exception as e:
        logging.error(f'Error deleting movie: {title}. {e}')
        return {'status': 'error', 'message': f'Failed to delete movie: {title}'}
    finally:
        connection.close()
```

Here is the separate function for updating the tickets_available since other functions call this as well:

```
def update_tickets_of_movie(title, tickets_available):
    try:
        title = title.strip()
        tickets_available = int(tickets_available)
        logging.debug(f'Parsed tickets_available: {tickets_available}')
        connection = get_db_connection()
        cursor = connection.cursor()
        cursor.execute('SELECT title FROM movies WHERE title = ?', (title,))
        if cursor.fetchone():
            cursor.execute(
                'UPDATE movies SET tickets_available = ? WHERE title = ?',
                (tickets_available, title)
            )
            connection.commit()
            logging.debug(f'Tickets updated for movie: {title}')
            return {'status': 'success', 'message': f'Tickets updated for movie: {title}'}
        else:
            logging.debug(f'Movie not found: {title}')
            return {'status': 'error', 'message': f'Movie: {title} not found'}
    except Exception as e:
        logging.error(f'Error updating tickets for movie: {title}. {e}')
        return {'status': 'error', 'message': f'Failed to update tickets for movie: {title}'}
    finally:
        connection.close()
```

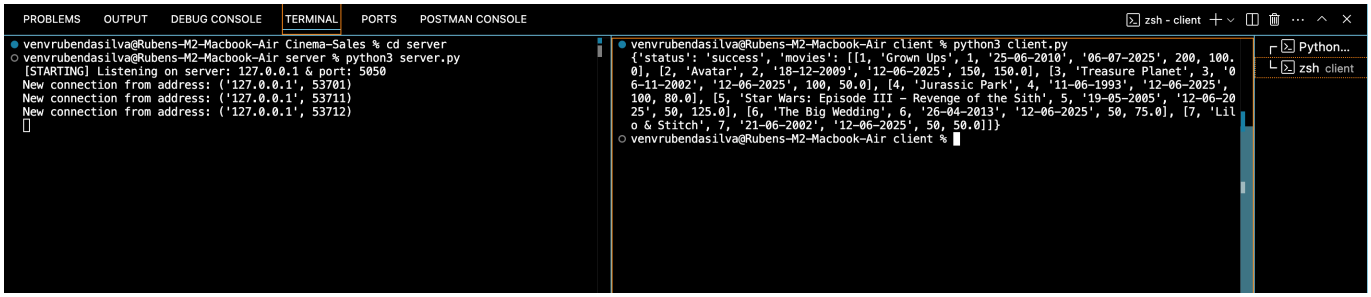
The last function is to record_ticket_sale:

```
def record_ticket_sale(title, customer_name, number_of_tickets, total):
    try:
        title = title.strip()
        customer_name = customer_name.strip()
        number_of_tickets = int(number_of_tickets)
        total = float(total)
        connection = get_db_connection()
        cursor = connection.cursor()
        cursor.execute('SELECT id, tickets_available, ticket_price FROM movies WHERE title = ?', (title,))
        movie = cursor.fetchone()
        if not movie:
            logging.debug(f'Movie not found: {title}')
            return {'status': 'error', 'message': f'Movie: {title} not found!'}
        movie_id, tickets_available, ticket_price = movie
        if tickets_available < number_of_tickets:
            logging.debug(f'Not enough tickets available for movie: {title}')
            return {'status': 'error', 'message': 'Not enough tickets available for this movie!'}
        if total != number_of_tickets * ticket_price:
            logging.debug(f'Total does not match the number of tickets for movie: {title}')
            return {'status': 'error', 'message': 'Total does not match the number of tickets!'}
        cursor.execute('''
            INSERT INTO sales (movie_id, customer_name, number_of_tickets, total)
            VALUES (?, ?, ?, ?)
        ''', (movie_id, customer_name, number_of_tickets, total))
        cursor.execute('''
            UPDATE movies
            SET tickets_available = tickets_available - ?
            WHERE id = ?
        ''', (number_of_tickets, movie_id))
        connection.commit()
```

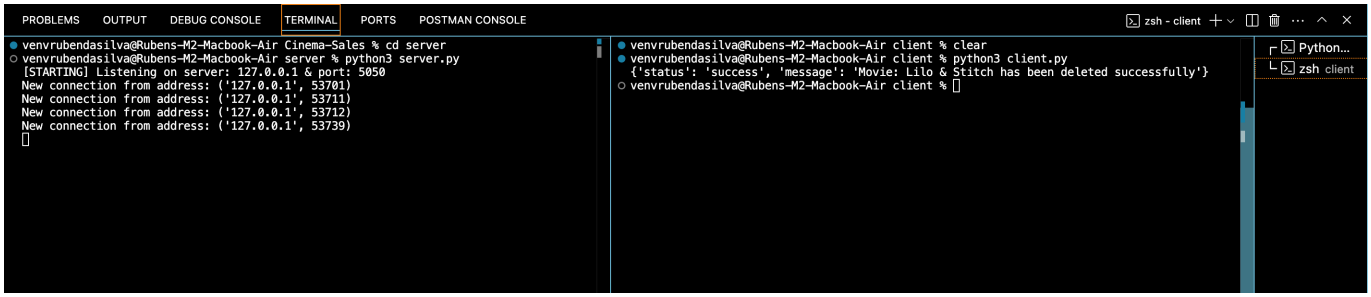
```
        receipt_content = (
            f"--- RECEIPT OF SALE ---\n"
            f"Movie ID: {movie_id}\n"
            f"Title: {title}\n"
            f"Customer: {customer_name}\n"
            f"Tickets Purchased: {number_of_tickets}\n"
            f"Total: R{total:.2f}\n"
        )
        receipts_dir = os.path.join(os.path.dirname(__file__), 'receipts')
        os.makedirs(receipts_dir, exist_ok=True)
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        receipt_filename = f"receipt_{movie_id}_{customer_name.replace(' ', '_')}_{timestamp}.txt"
        receipt_path = os.path.join(receipts_dir, receipt_filename)
        with open(receipt_path, 'w') as file:
            file.write(receipt_content)
        logging.debug(f'Sale recorded and receipt saved: {receipt_filename}')
        return {
            'status': 'success',
            'message': f'Tickets purchased successfully! Receipt saved as {receipt_filename}'
        }
    except Exception as e:
        logging.error(f'Error recording ticket sale for movie: {title}. {e}')
        return {
            'status': 'error',
            'message': f'Failed to record ticket sale for movie: {title}'
        }
    finally:
        connection.close()
```

Here is the proof of concept of the requests working from my client.py file making direct requests:

1. **get_movies():**



2. delete_movie():

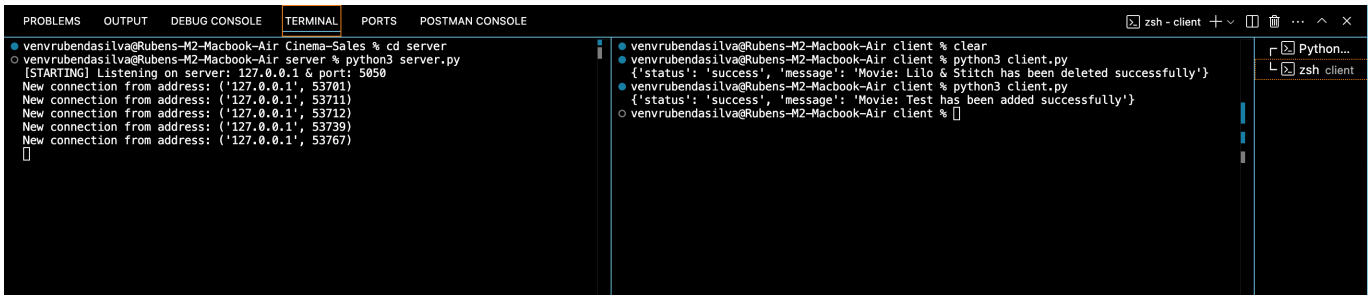


This has now freed up a space in my database as seen below:

SQL ▼ 1 / 1 1 - 6 of 6

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	200	100.0
2	Avatar	2	18-12-2009	12-06-2025	150	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III - Revenge of the Sith	5	19-05-2005	12-06-2025	50	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0

3. add_movie():



This has now updated in the database:

SQL ▼ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	200	100.0
2	Avatar	2	18-12-2009	12-06-2025	150	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	50	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Test	7	20-05-2025	02-06-2025	69	420.0

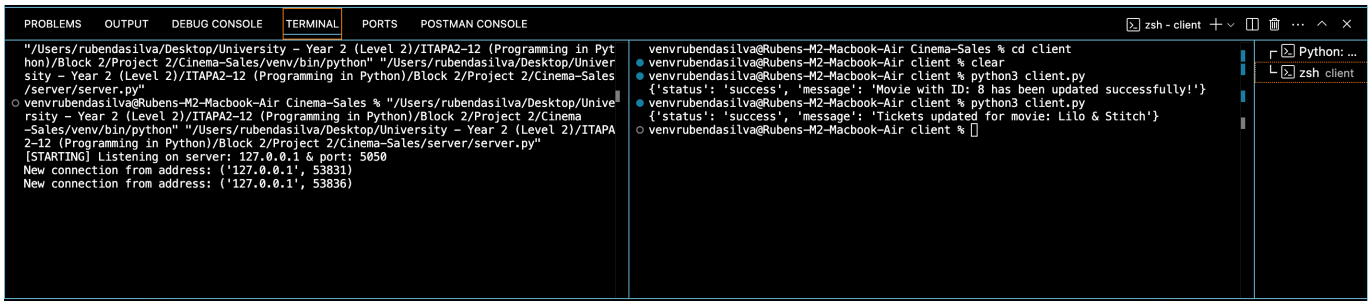
4. update_movie_details():

Updated database:

SQL ▼ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	200	100.0
2	Avatar	2	18-12-2009	12-06-2025	150	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	50	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Lilo & Stitch	7	21-06-2002	02-06-2025	69	50.0

5. update_tickets_of_movie():

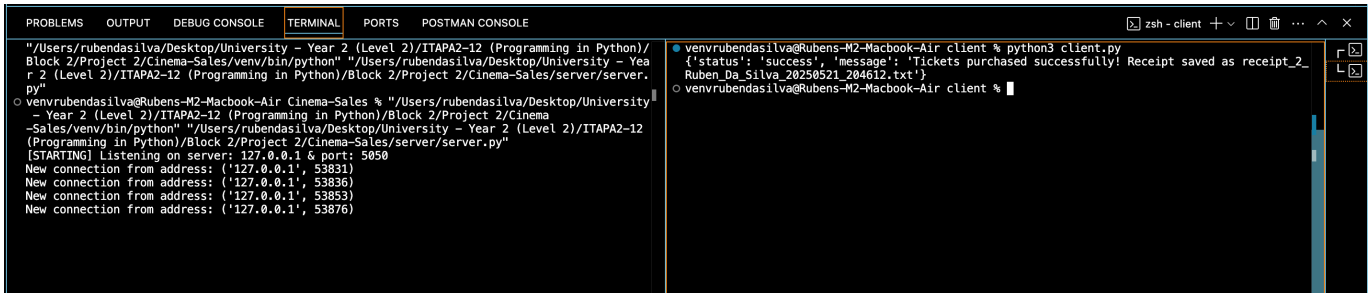


Database:

SQL ▼ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	200	100.0
2	Avatar	2	18-12-2009	12-06-2025	150	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	50	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Lilo & Stitch	7	21-06-2002	02-06-2025	125	50.0

6. record_ticket_sale():

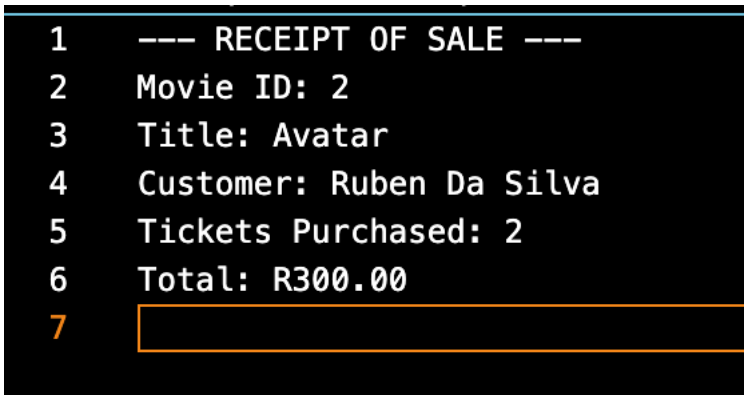


Database has decreased available tickets:

SQL ▼ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	200	100.0
2	Avatar	2	18-12-2009	12-06-2025	146	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	50	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Lilo & Stitch	7	21-06-2002	02-06-2025	125	50.0

Receipt:



Question 2: Client script with Tkinter GUI

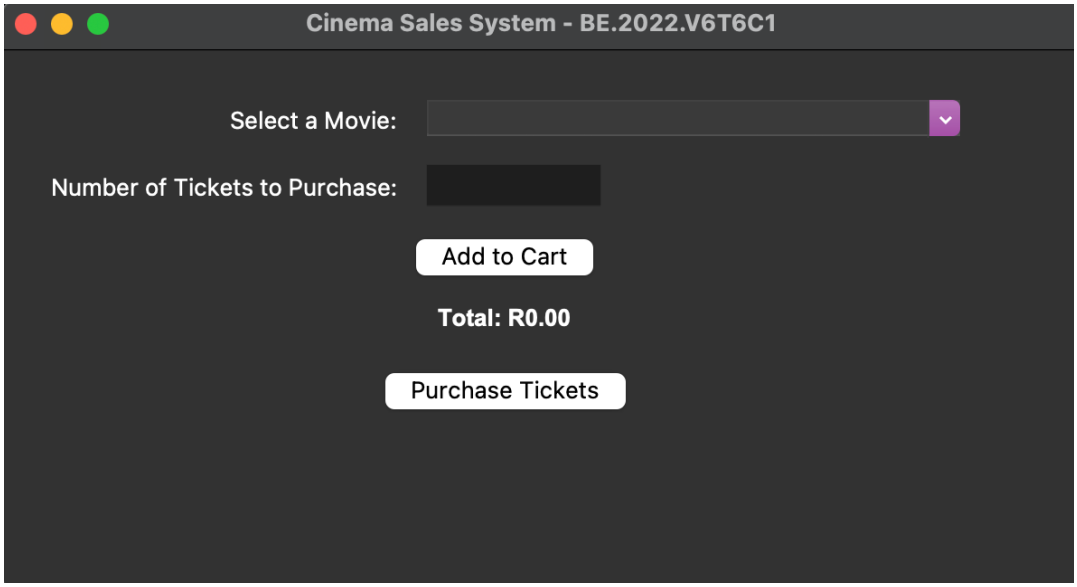
Now that the sever is up and running and can handle requests coming in from my client.py file. I will now tackle the GUI needed to perform these operations/requests.

I started off by getting the basic window setup:

```
window = tkinter.Tk()
window.title("Cinema Sales System - BE.2022.V6T6C1")
window.geometry("600x300")

window.mainloop()
```

Nex up was setting up the dropdown for the movies, the input field for the amount of tickets as well as a button to add to the cart and a button to purchase:

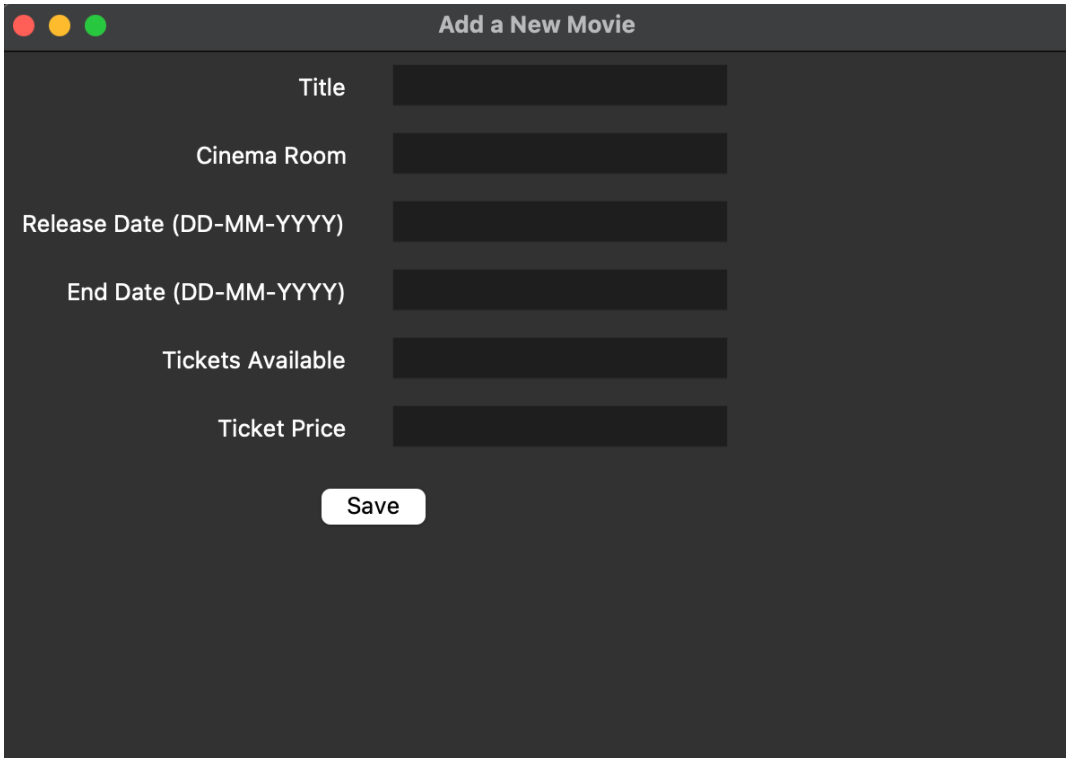
```
window = tkinter.Tk()
window.title("Cinema Sales System - BE.2022.V6T6C1")
window.geometry("600x300")

main_frame = tkinter.Frame(window, padx=20, pady=20)
main_frame.grid(row=0, column=0, sticky="nsew")

tkinter.Label(main_frame, text="Select a Movie:").grid(row=0, column=0, sticky="e", padx=5, pady=5)
movie_variable = tkinter.StringVar()
movie_dropdown = ttk.Combobox(main_frame, textvariable=movie_variable, state="readonly", width=30)
movie_dropdown.grid(row=0, column=1, sticky="w", padx=5, pady=5)

tkinter.Label(main_frame, text="Number of Tickets to Purchase:").grid(row=1, column=0, sticky="e", padx=5, pady=5)
ticket_entry = tkinter.Entry(main_frame, width=10)
ticket_entry.grid(row=1, column=1, sticky="w", padx=5, pady=5)

tkinter.Button(main_frame, text="Add to Cart").grid(row=2, column=0, columnspan=2, pady=5)

total_label = tkinter.Label(main_frame, text="Total: R0.00", font=('Arial', 10, 'bold'))
total_label.grid(row=3, column=0, columnspan=2, pady=5)

tkinter.Button(main_frame, text="Purchase Tickets").grid(row=4, column=0, columnspan=2, pady=10)

window.mainloop()
```



Now that the GUI is basically setup, I need to include buttons to add, update and delete a movie. Ive decided to do this as separate windows.

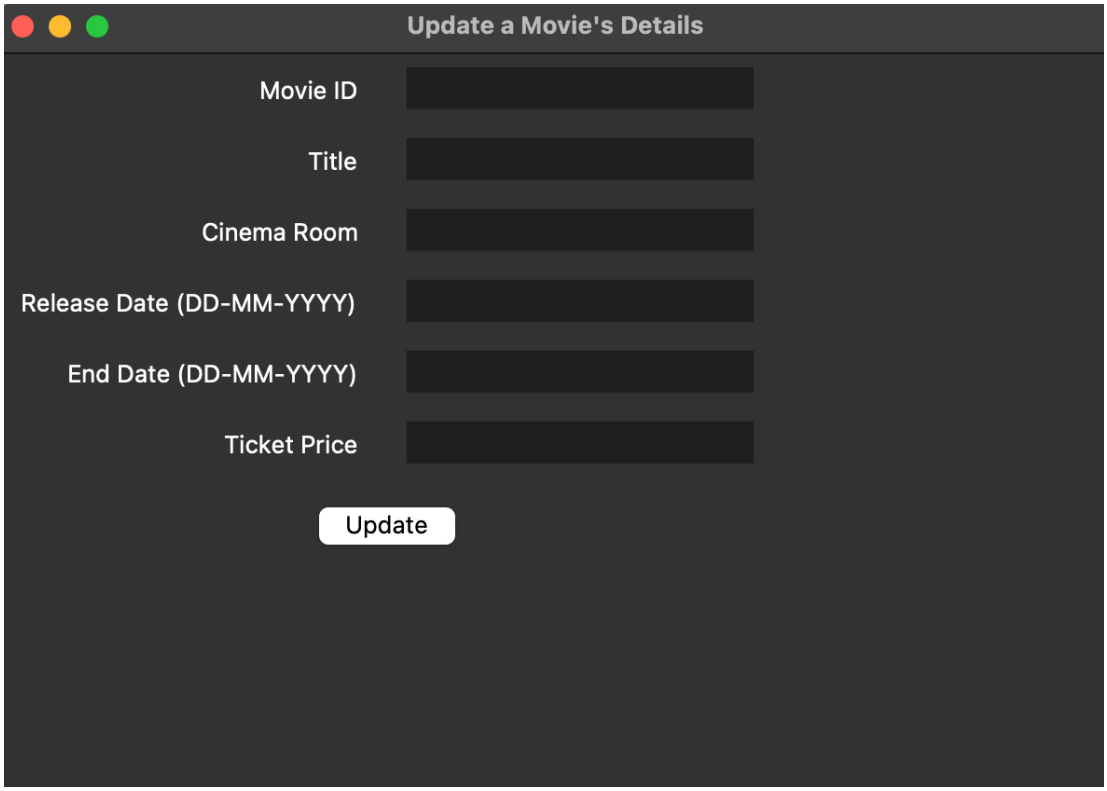
Add a Movie:

```
def open_add_movie_window():
    add_window = tkinter.Toplevel(window)
    add_window.title("Add a New Movie")
    add_window.geometry("600x400")

    fields = {
        "Title": tkinter.Entry(add_window),
        "Cinema Room": tkinter.Entry(add_window),
        "Release Date (DD-MM-YYYY)": tkinter.Entry(add_window),
        "End Date (DD-MM-YYYY)": tkinter.Entry(add_window),
        "Tickets Available": tkinter.Entry(add_window),
        "Ticket Price": tkinter.Entry(add_window),
    }

    for idx, (label, entry) in enumerate(fields.items()):
        tkinter.Label(add_window, text=label).grid(row=idx, column=0, padx=10, pady=5, sticky="e")
        entry.grid(row=idx, column=1, padx=10, pady=5)

    def save_movie():
        request = {
            'operation': 'add_movie',
            'title': fields["Title"].get(),
            'cinema_room': fields["Cinema Room"].get(),
            'release_date': fields["Release Date (DD-MM-YYYY)"].get(),
            'end_date': fields["End Date (DD-MM-YYYY)"].get(),
            'tickets_available': fields["Tickets Available"].get(),
            'ticket_price': fields["Ticket Price"].get(),
        }
        response = send_request(request)
        messagebox.showinfo("Add Movie", response['message'])
        add_window.destroy()

    tkinter.Button(add_window, text="Save", command=save_movie).grid(row=len(fields), column=0, columnspan=2, pady=10)
```



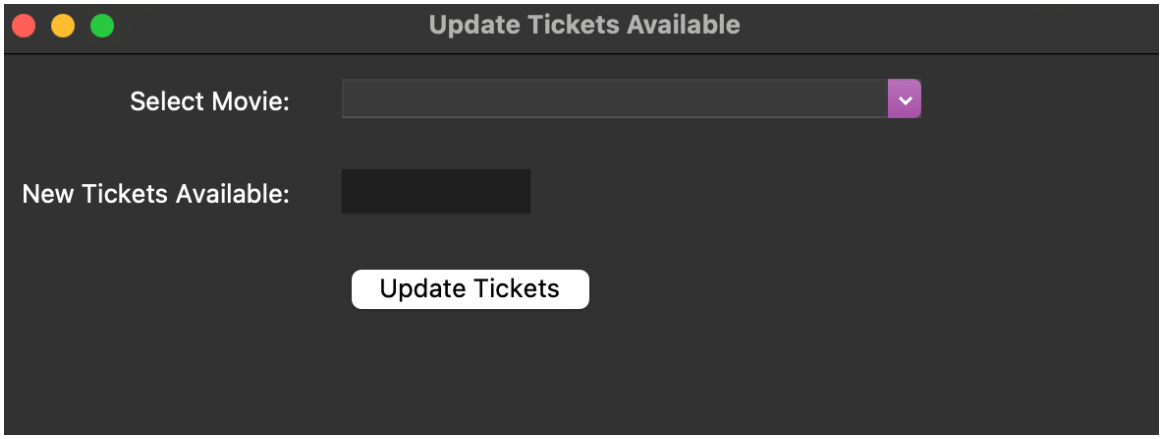
Update a Movie's Details:

```python
def open_update_movie_window():
    update_window= tkinter.Toplevel(window)
    update_window.title("Update a Movie's Details")
    update_window.geometry("600x400")

    fields = {
        "Movie ID": tkinter.Entry(update_window),
        "Title": tkinter.Entry(update_window),
        "Cinema Room": tkinter.Entry(update_window),
        "Release Date (DD-MM-YYYY)": tkinter.Entry(update_window),
        "End Date (DD-MM-YYYY)": tkinter.Entry(update_window),
        "Ticket Price": tkinter.Entry(update_window),
    }
    for idx, (label, entry) in enumerate(fields.items()):
        tkinter.Label(update_window, text=label).grid(row=idx, column=0, padx=10, pady=5, sticky="e")
        entry.grid(row=idx, column=1, padx=10, pady=5)

        def update_movie():
            request = {
                'operation': 'update_movie_details',
                'id': fields["Movie ID"].get(),
                'title': fields["Title"].get(),
                'cinema_room': fields["Cinema Room"].get(),
                'release_date': fields["Release Date (DD-MM-YYYY)"].get(),
                'end_date': fields["End Date (DD-MM-YYYY)"].get(),
                'ticket_price': fields["Ticket Price"].get(),
            }
            response = send_request(request)
            messagebox.showinfo("Update Movie", response['message'])
            update_window.destroy()
    tkinter.Button(update_window, text="Update", command=update_movie).grid(row=len(fields), column=0, columnspan=2, pady=10)
```



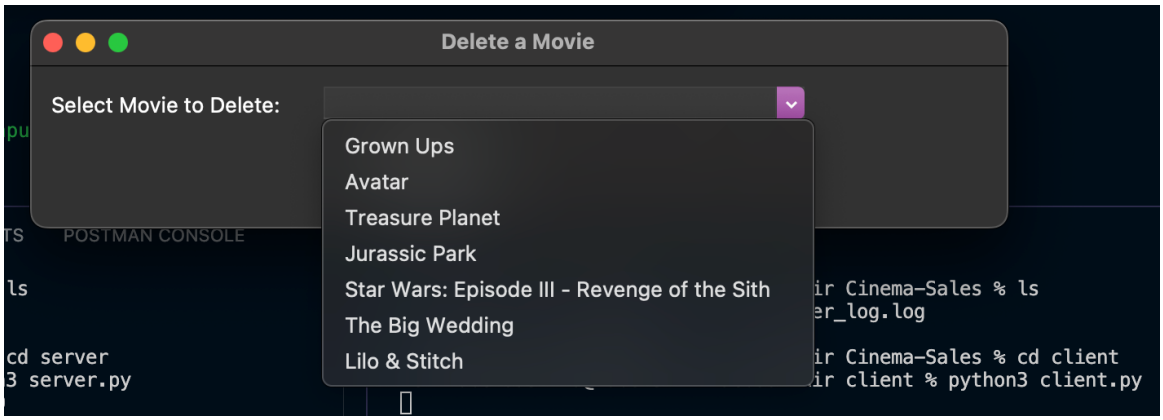
Updating a movie's tickets:

```python
def open_update_tickets_window():
    update_tickets_window = tkinter.Toplevel(window)
    update_tickets_window.title("Update Tickets Available")
    update_tickets_window.geometry("600x200")

    tkinter.Label(update_tickets_window, text="Select Movie:").grid(row=0, column=0, padx=10, pady=10, sticky="e")

    movie_var = tkinter.StringVar()
    movie_dropdown = ttk.Combobox(update_tickets_window, textvariable=movie_var, state="readonly", width=30)
    movie_dropdown.grid(row=0, column=1, padx=10, pady=10)

    response = send_request({'operation': 'get_movies'})
    if response['status'] == 'success':
        movie_titles = [movie[1] for movie in response['movies']]
        movie_dropdown['values'] = movie_titles
    else:
        messagebox.showerror("Error", response['message'])
        update_tickets_window.destroy()
        return

    tkinter.Label(update_tickets_window, text="New Tickets Available:").grid(row=1, column=0, padx=10, pady=10, sticky="e")
    tickets_entry = tkinter.Entry(update_tickets_window, width=10)
    tickets_entry.grid(row=1, column=1, padx=10, pady=10, sticky="w")

    def update_tickets():
        selected_movie = movie_var.get()
        new_tickets = tickets_entry.get()

        if not selected_movie:
            messagebox.showwarning("Missing Input", "Please select a movie.")
            return
        if not new_tickets:
            messagebox.showwarning("Missing Input", "Please enter the new tickets available.")
            return

        try:
            tickets_int = int(new_tickets)
            if tickets_int < 0:
                raise ValueError("Tickets must be a positive number")

            request = {
                'operation': 'update_tickets_of_movie',
                'title': selected_movie,
                'tickets_available': str(tickets_int)
            }
            response = send_request(request)
            messagebox.showinfo("Update Tickets", response['message'])
            update_tickets_window.destroy()
        except ValueError as e:
            messagebox.showerror("Invalid Input", str(e))

    tkinter.Button(update_tickets_window,text="Update Tickets", command=update_tickets).grid(row=2, column=0, columnspan=2, pady=10)
```



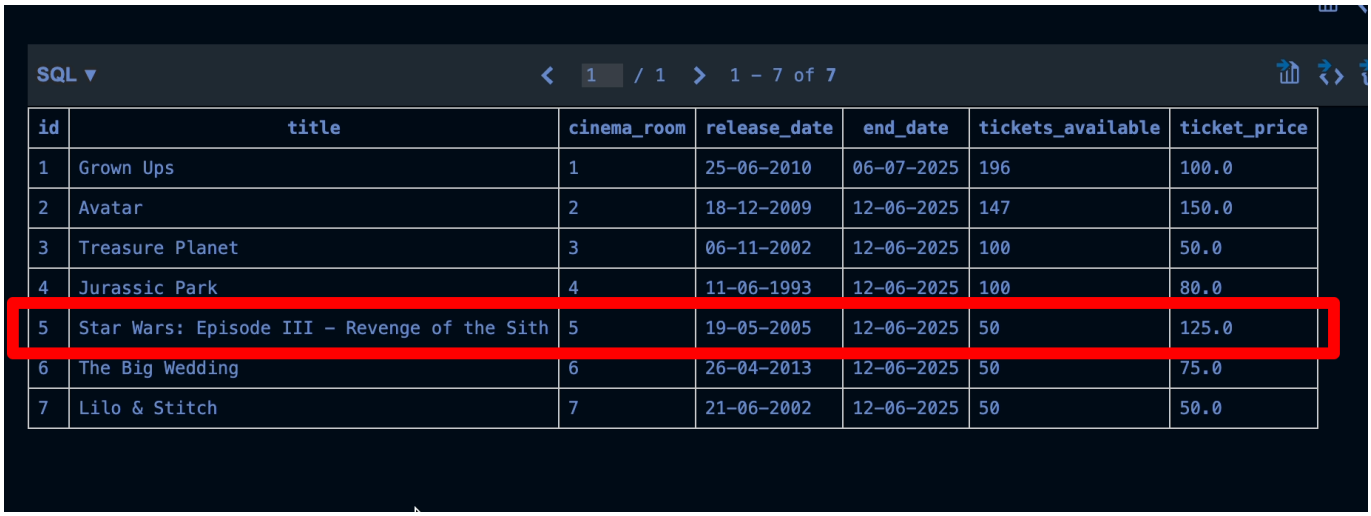
Delete a Movie:

```python
def open_delete_movie_window():
    delete_window = tkinter.Toplevel(window)
    delete_window.title("Delete a Movie")
    delete_window.geometry("600x100")

    tkinter.Label(delete_window, text="Select Movie to Delete:").grid(row=0, column=0, padx=10, pady=10, sticky="e")

    movie_var = tkinter.StringVar()
    movie_dropdown = ttk.Combobox(delete_window, textvariable=movie_var, state="readonly", width=30)
    movie_dropdown.grid(row=0, column=1, padx=10, pady=10)

    response = send_request({'operation': 'get_movies'})
    if response['status'] == 'success':
        movie_titles = [movie[1] for movie in response['movies']]
        movie_dropdown['values'] = movie_titles
    else:
        messagebox.showerror("Error", response['message'])

    def delete_movie():
        title = movie_var.get()
        if not title:
            messagebox.showwarning("Missing Input", "Please select a movie.")
            return

        request = {
            'operation': 'delete_movie',
            'title': title,
        }
        response = send_request(request)
        messagebox.showinfo("Delete Movie", response['message'])
        delete_window.destroy()

    tkinter.Button(delete_window, text="Delete", command=delete_movie).grid(row=1, column=0, columnspan=2, pady=10)
```



Now that the windows are done for each operation, I am now executing each function starting off with Purchasing Tickets for a movie.

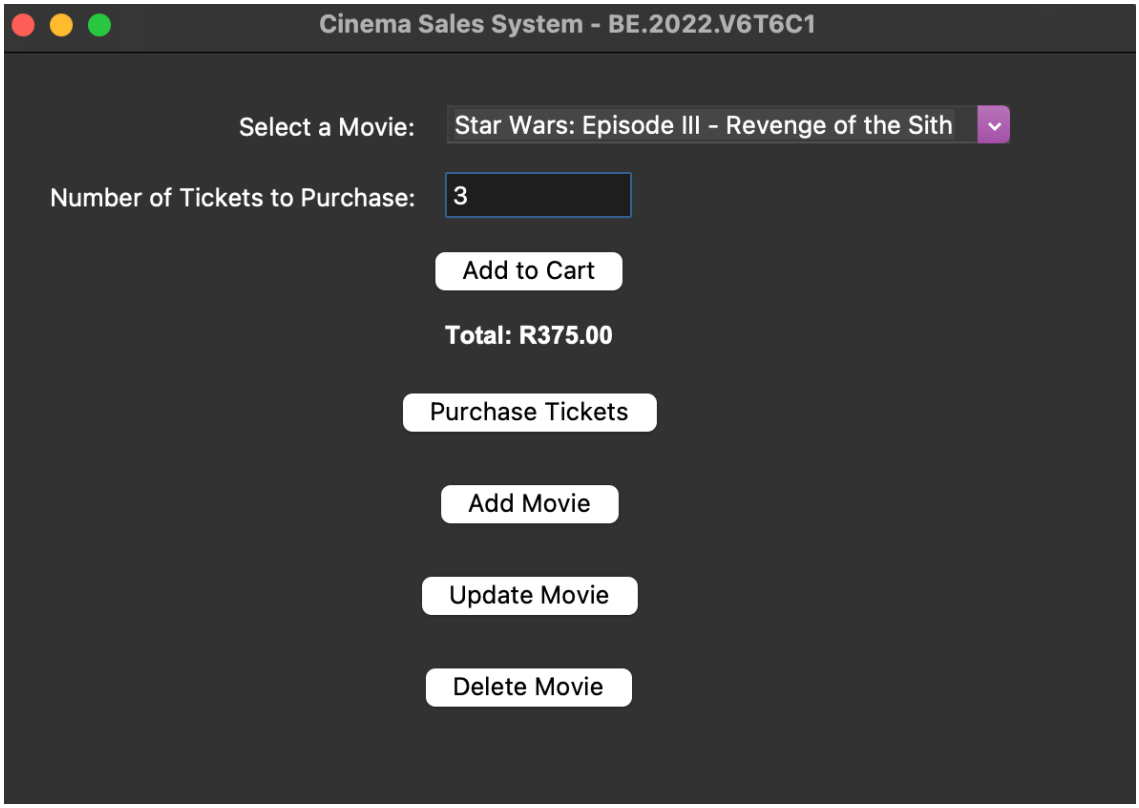
As we can see for Star Wars: Episode III – Revenge of the Sith, there are 50 tickets available.



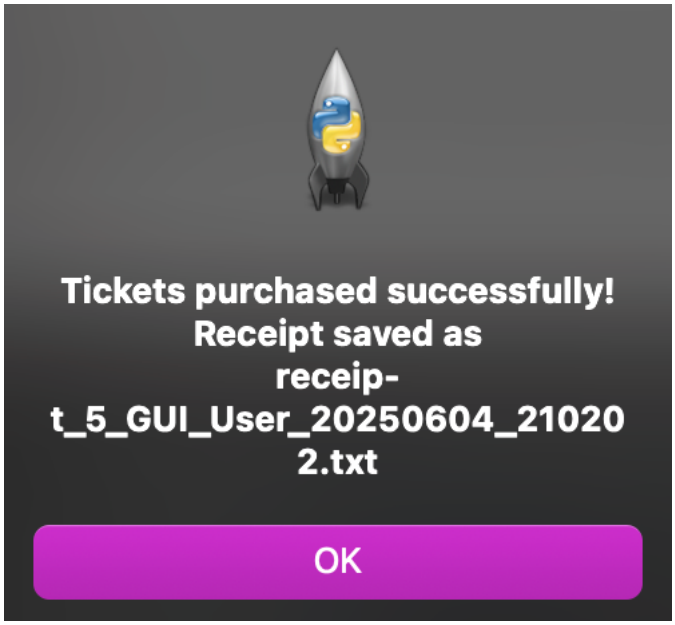
SQL ▾ 1 / 1 1 – 7 of 7

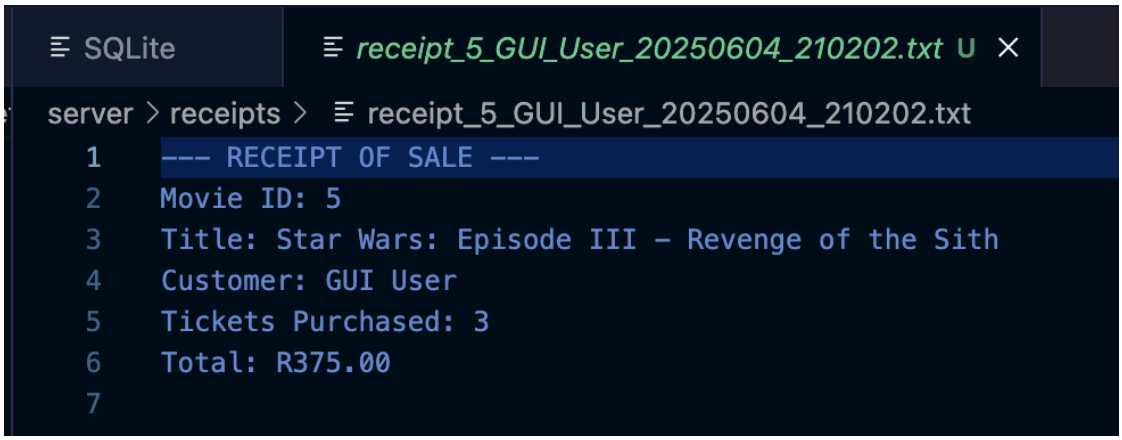
id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	50	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
7	Lilo & Stitch	7	21-06-2002	12-06-2025	50	50.0

For this test I will be buying 3 tickets @ R125.00 each making my total R375.00.

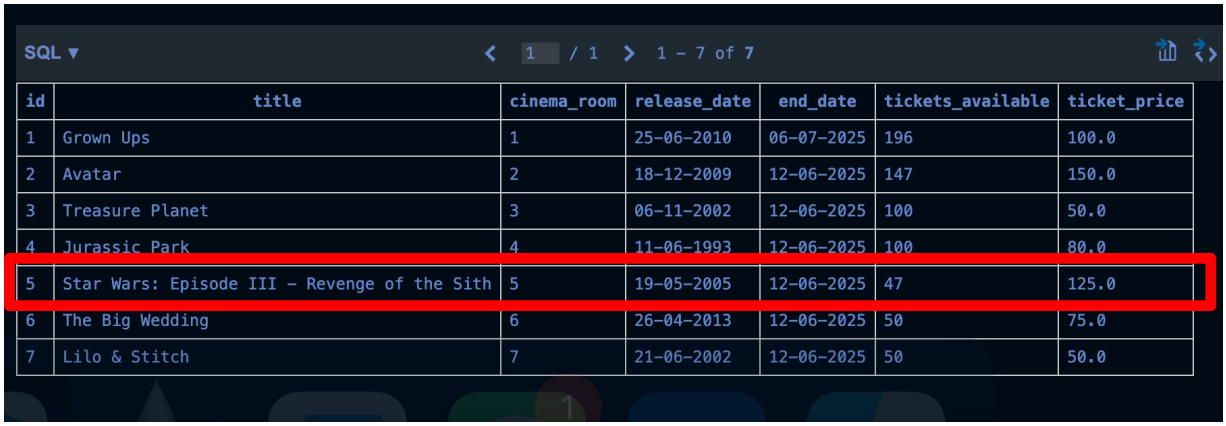


Receipt of purchase:





Ticket count after purchase for Star Wars: Episode III – Revenge of the Sith:



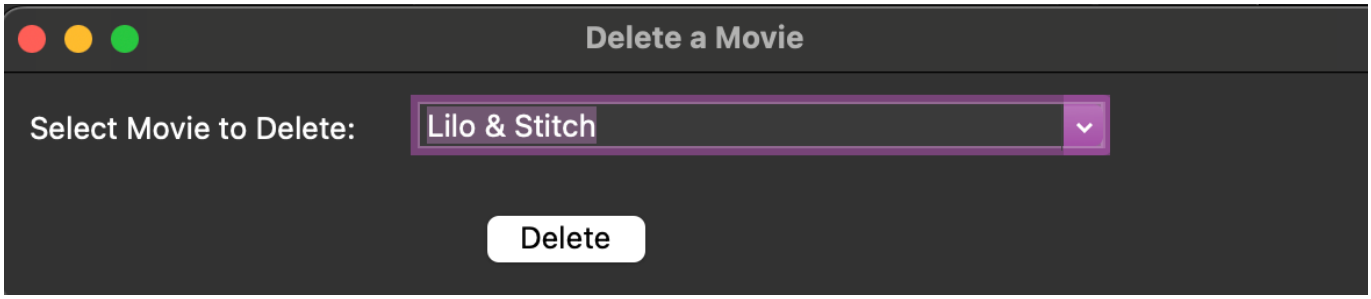
id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
7	Lilo & Stitch	7	21-06-2002	12-06-2025	50	50.0

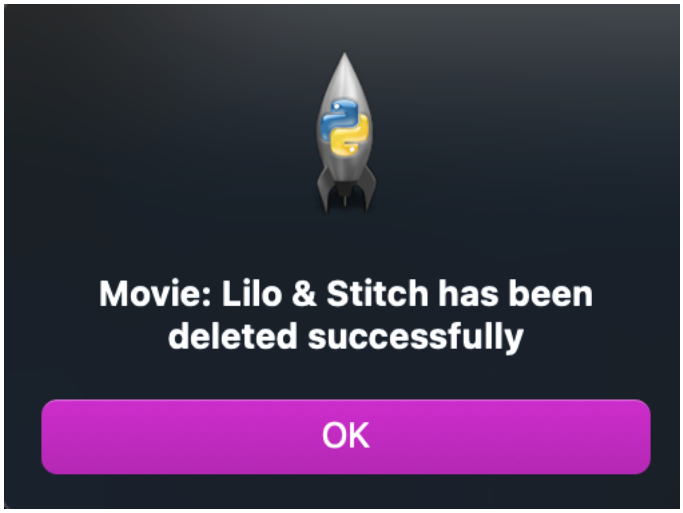
I will now delete a movie “Lilo & Stitch”, here is the current DB:

SQL ▾ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
7	Lilo & Stitch	7	21-06-2002	12-06-2025	50	50.0

This is me deleting:





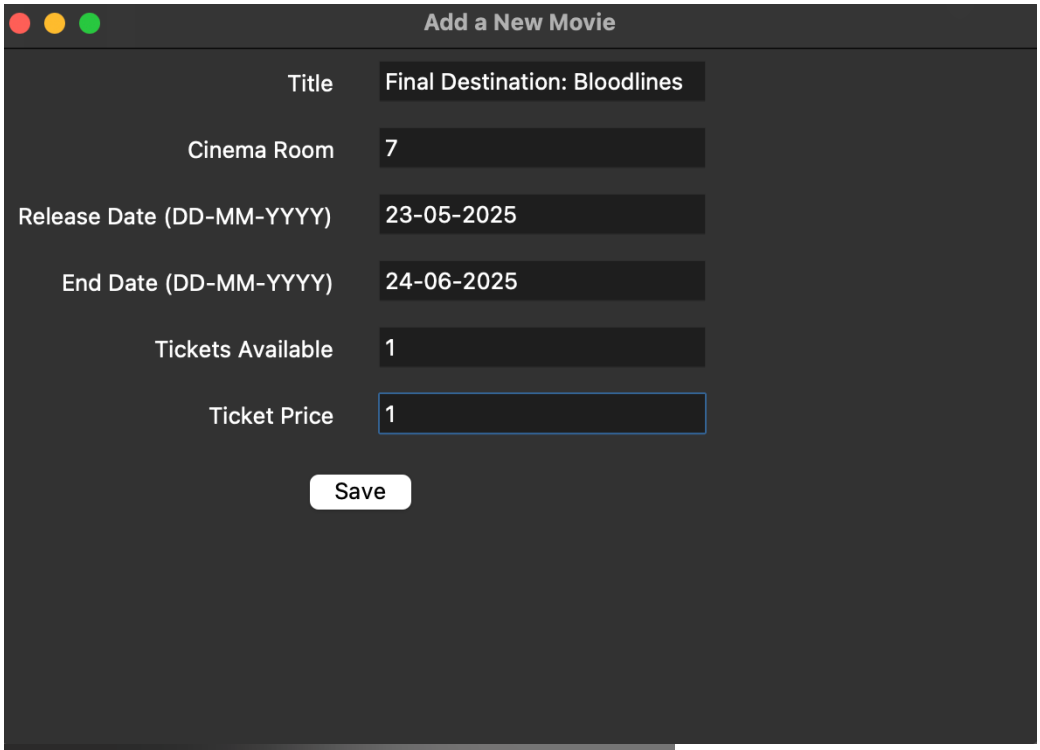
This is the DB after my deletion:

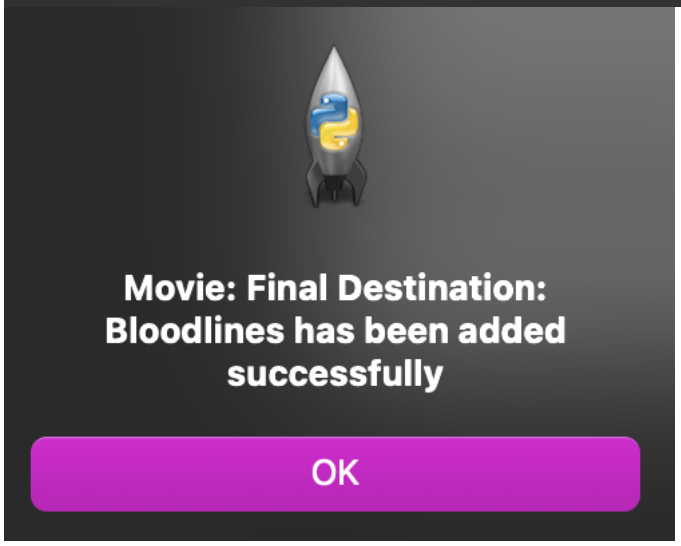
SQL ▾ 1 / 1 1 – 6 of 6

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0

I will now be adding a new movie “Final Destination: Bloodlines”

Here is me adding the movie:





This is the DB with my new movie:

SQL ▾ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Final Destination: Bloodlines	7	23-05-2025	24-06-2025	1	1.0

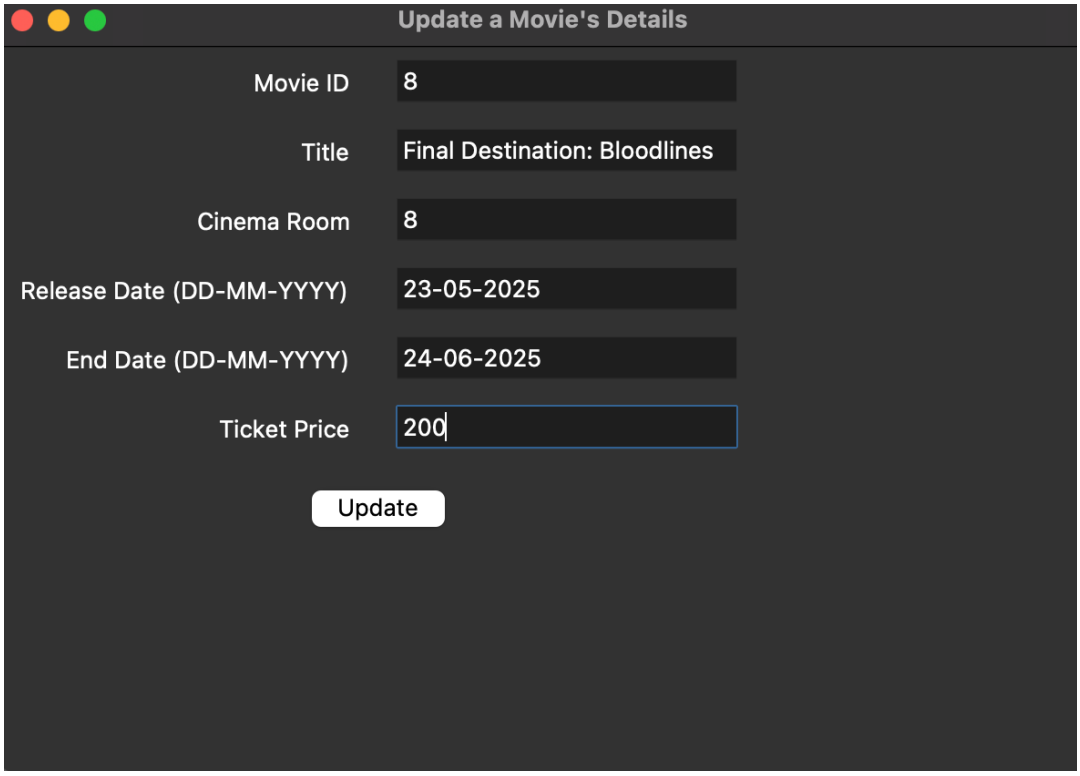
I purposely left the tickets available and the ticket price as 1 respectively so that I could update it for my next function.

I will now update the new movie as seen in the DB:

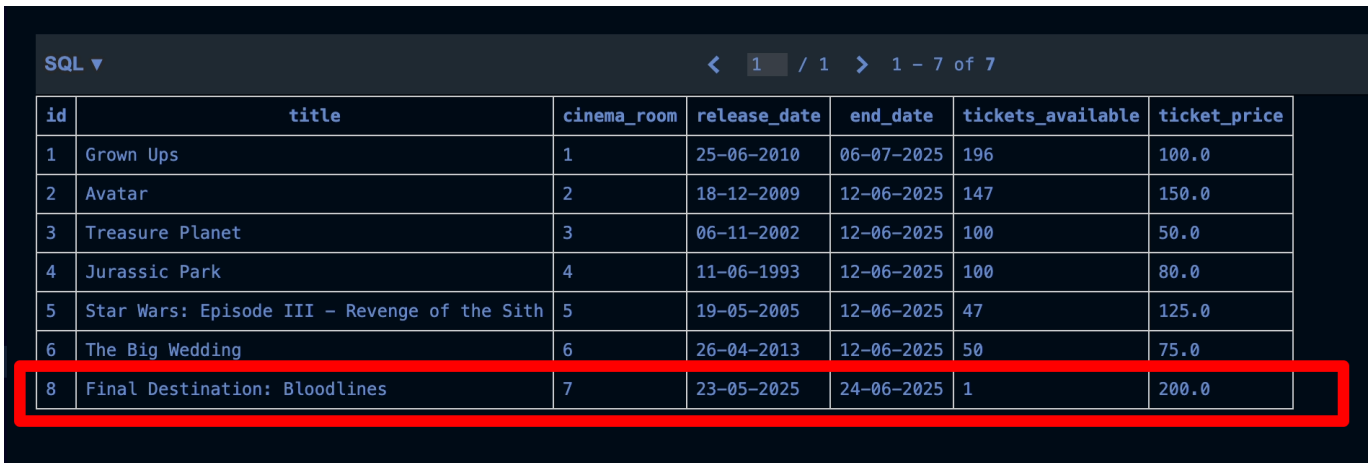
SQL ▾ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Final Destination: Bloodlines	7	23-05-2025	24-06-2025	1	1.0

Making the updates:



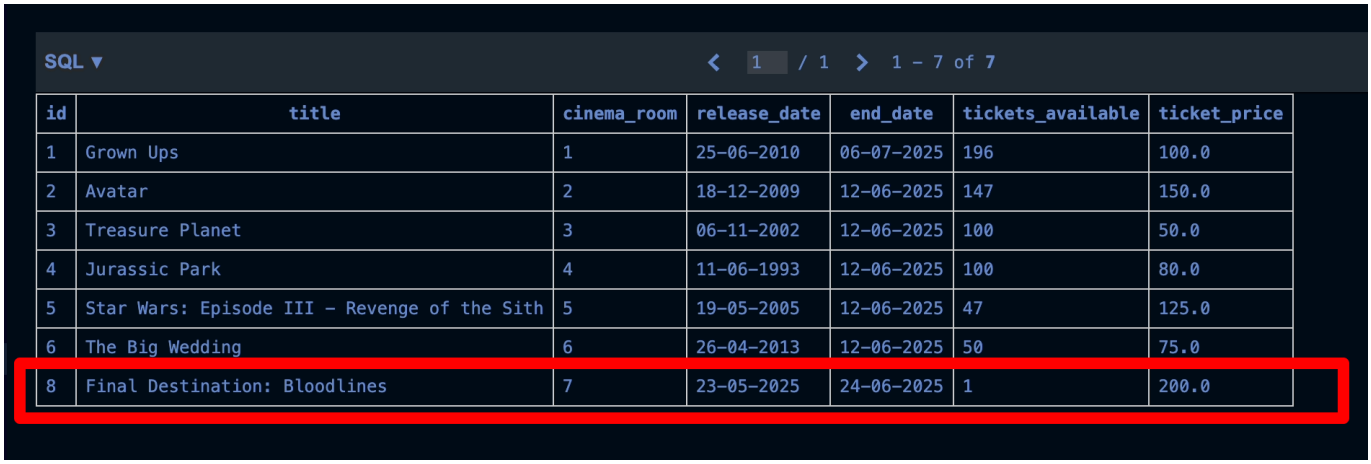
Record after in the DB:



SQL ▾ 1 / 1 1 – 7 of 7

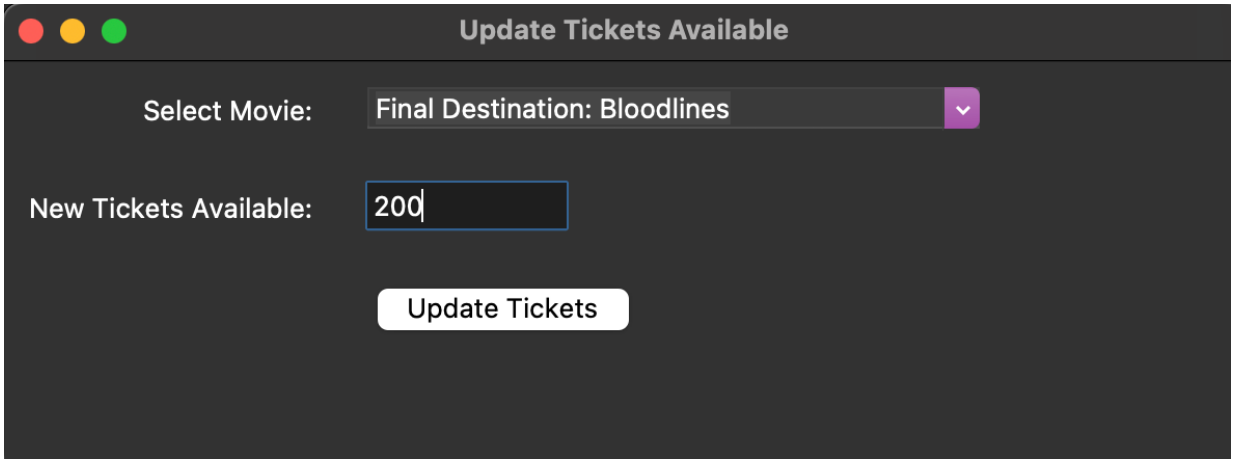
id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Final Destination: Bloodlines	7	23-05-2025	24-06-2025	1	200.0

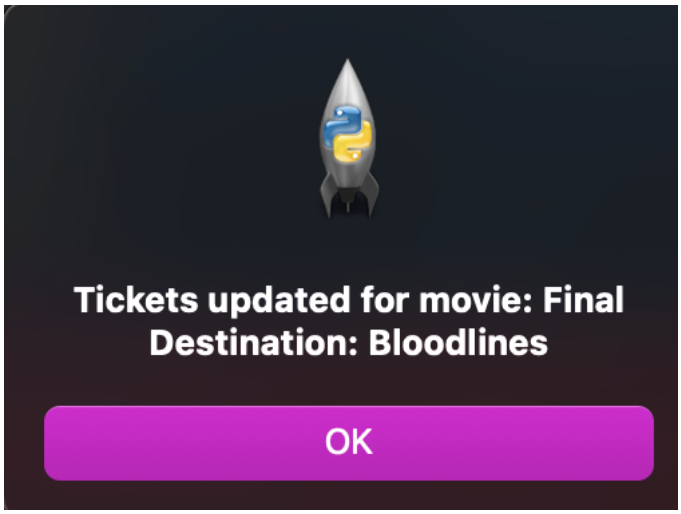
For my last function I will be updating the number of tickets available. As we can see in the DB there is only 1 ticket available:



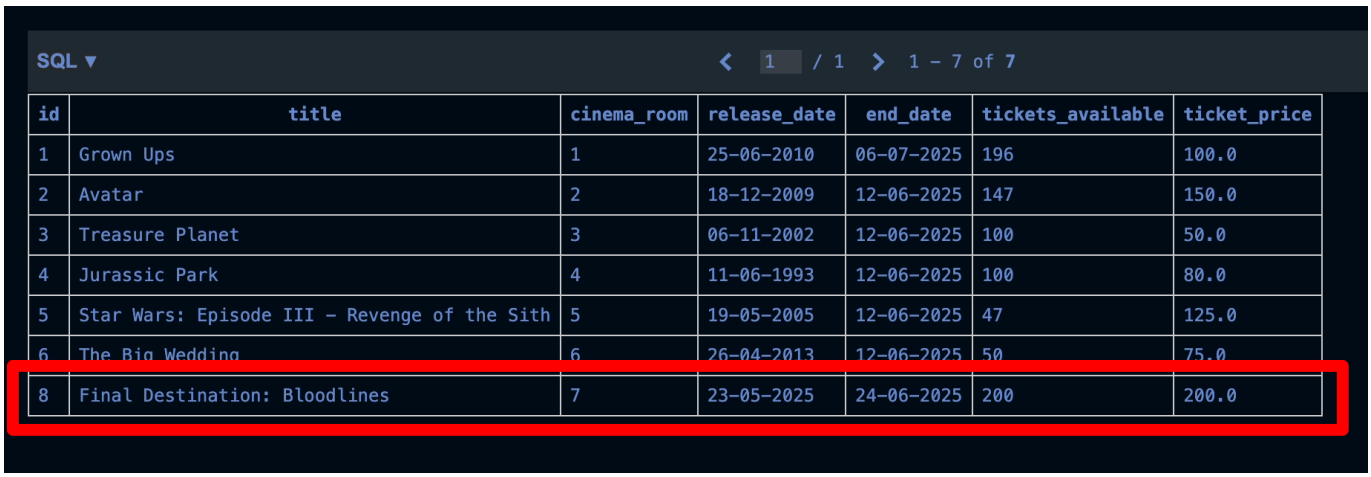
SQL ▾ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Final Destination: Bloodlines	7	23-05-2025	24-06-2025	1	200.0





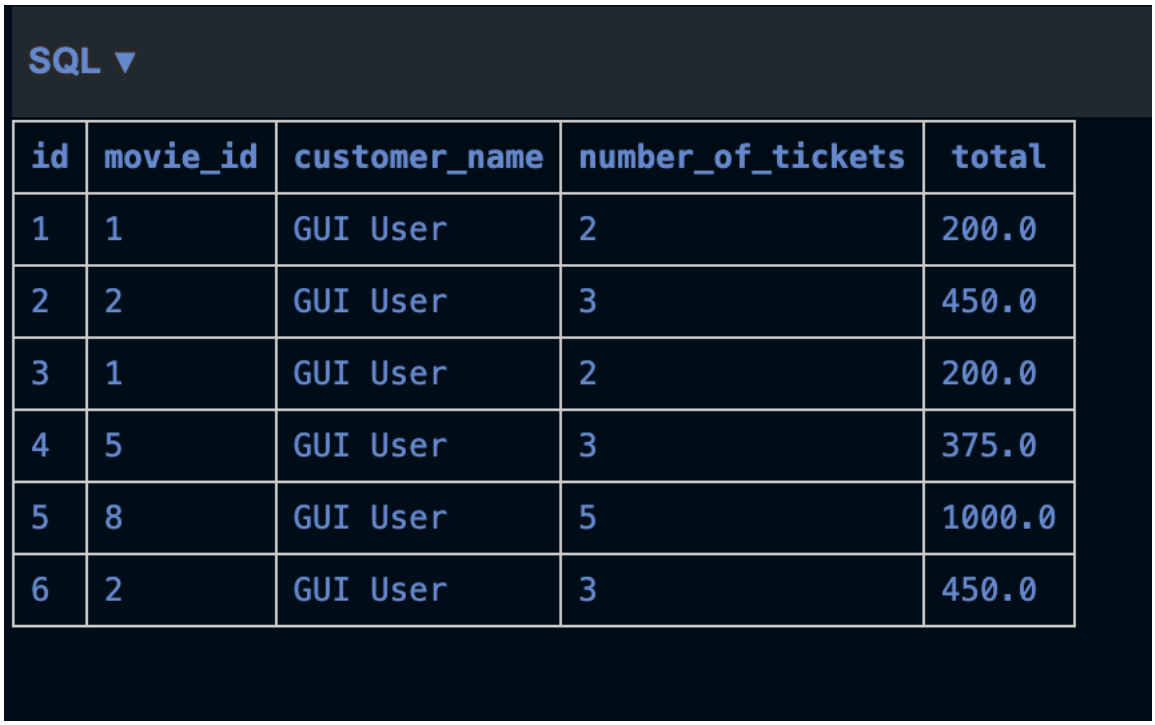
DB after the update:



SQL ▾ 1 / 1 1 – 7 of 7

id	title	cinema_room	release_date	end_date	tickets_available	ticket_price
1	Grown Ups	1	25-06-2010	06-07-2025	196	100.0
2	Avatar	2	18-12-2009	12-06-2025	147	150.0
3	Treasure Planet	3	06-11-2002	12-06-2025	100	50.0
4	Jurassic Park	4	11-06-1993	12-06-2025	100	80.0
5	Star Wars: Episode III – Revenge of the Sith	5	19-05-2005	12-06-2025	47	125.0
6	The Big Wedding	6	26-04-2013	12-06-2025	50	75.0
8	Final Destination: Bloodlines	7	23-05-2025	24-06-2025	200	200.0

Sales table after some sales:



SQL ▾

id	movie_id	customer_name	number_of_tickets	total
1	1	GUI User	2	200.0
2	2	GUI User	3	450.0
3	1	GUI User	2	200.0
4	5	GUI User	3	375.0
5	8	GUI User	5	1000.0
6	2	GUI User	3	450.0

References

- *Grammarly*. Available at: https://app.grammarly.com/ (Accessed: 24 May 2025).
- Amos, D. (2024) *Object-oriented programming (OOP) in Python*, *Real Python*. Available at: https://realpython.com/python3-object-oriented-programming/ (Accessed: 24 May 2025).
- *An easy way to master sqlite fast* (2024) *SQLite Tutorial*. Available at: https://www.sqlitetutorial.net/ (Accessed: 25 February 2025).
- Foster, J. (2020) *Python for beginners: Learn the fundamentals of Computer Programming*. Elluminet Press.
- Ramos, L.P. (2024) *Python classes: The power of object-oriented programming*, *Real Python*. Available at: https://realpython.com/python-classes/ (Accessed: 25 May 2025).
- *SQLite insert into* (no date) *w3resource*. Available at: https://www.w3resource.com/sqlite/sqlite-insert-into.php (Accessed: 31 May 2025).
- Vanderheyden, T. (2022) *Object-oriented programming in Python (oop): Tutorial*, *DataCamp*. Available at: https://www.datacamp.com/tutorial/python-oop-tutorial (Accessed: 31 May 2025).
- *W3schools.com* (no date) *W3Schools Online Web Tutorials*. Available at: https://www.w3schools.com/python/gloss_python_object_methods.asp (Accessed: 1 June 2025).
- GeeksforGeeks, G. (2024) *Python tkinter*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/python-gui-tkinter/ (Accessed: 03 June 2025).
- Schafer, C. (2024) *Python Tkinter Tutorial (Part 1): Getting Started, Elements, Layouts, and Events*, *YouTube*. Available at: https://www.youtube.com/watch?v=epDKamC-V-8&pp=0gcJCdgAo7VqN5tD (Accessed: 03 June 2025).