Name: **Ace Vincent Gamitin** Section: **M001**

Program description:

This program, entitled “Movie Manager”, employs all the lessons of Computer Programming 2. It applies CRUD (Create, Read, Update, Delete) through inputting movies with different credentials stored in a three-table SQL database, where it shows one-to-many relationships. This program also provided a Graphical User Interface by implementing PyQt5.

Course reflection:

Throughout this course, I’ve gained a lot of knowledge from every lesson that was covered. I’ve noticed that this course is focused more on analyzing and manipulating data, and I have found it interesting. Although there were several challenges, such as countless errors from the database, more complicated syntax, and trying to manage time for self-learning, I know this is all part of a journey to being a future innovator.

Source Codes

```
1   # creating a database for the program
2   CREATE SCHEMA `movie_manager`;
3   USE `movie_manager`;
4
5   # genre table with genre name and genre ID
6   CREATE TABLE IF NOT EXISTS `genre` (
7       `genre_id` VARCHAR(45) NOT NULL,
8       `genre_name` VARCHAR(45) NOT NULL,
9       PRIMARY KEY (`genre_id`)
10  );
```

```sql
# genre table with genre name and genre ID
CREATE TABLE IF NOT EXISTS `genre` (
    `genre_id` VARCHAR(45) NOT NULL,
    `genre_name` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`genre_id`)
);

# studio table with studio name, studio ID, year founder, and headquarters
CREATE TABLE IF NOT EXISTS `studio` (
    `studio_id` VARCHAR(45) NOT NULL,
    `studio_name` VARCHAR(100) NOT NULL,
    `founded_year` YEAR,
    `headquarters` VARCHAR(100),
    PRIMARY KEY (`studio_id`)
);

# the table that be mainly displayed in the program
CREATE TABLE IF NOT EXISTS `movie` (
    `movie_id` VARCHAR(15) NOT NULL,
    `movie_name` VARCHAR(100) NOT NULL,
    `release_year` YEAR,
    `genre_id` VARCHAR(45) NOT NULL,
    `studio_id` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`movie_id`),
    FOREIGN KEY (`genre_id`) REFERENCES `genre`(`genre_id`),
    FOREIGN KEY (`studio_id`) REFERENCES `studio`(`studio_id`)
);

# inserting some values to the genre table
INSERT INTO `genre` (`genre_id`, `genre_name`) VALUE
('G001', 'Action'),
('G002', 'Drama'),
('G003', 'Sci-Fi'),
('G004', 'Comedy');
```

```sql
# inserting some values to the studio table
INSERT INTO `studio` (`studio_id`, `studio_name`, `founded_year`, `headquarters`) VALUES
('S001', 'Warner Bros.', 1923, 'Burbank'),
('S002', 'Paramount Pictures', 1912, 'Hollywood'),
('S003', 'Marvel Studios', 1993, 'Burbank');

```

```sql
# checking if the tables are now related
SELECT * FROM studio;

SELECT
    m.movie_id,
    m.movie_name,
    m.release_year,
    g.genre_name,
    s.studio_name
FROM
    movie m
JOIN
    genre g ON m.genre_id = g.genre_id
JOIN
    studio s ON m.studio_id = s.studio_id
ORDER BY
    m.movie_id;

```

```python
from PyQt5.QtWidgets import (QMainWindow, QApplication, QMessageBox,
                             QTableWidgetItem, QTableWidget, QLabel, QDialog) # for PyQt5 features
from PyQt5.uic import loadUi # directly loads the ui file
import sys #
from connect_database import ConnectDatabase #separate python file for DB connection
from MovieDetailsDialog import MovieDetailsDialog #separate python file movie details
from AddMovieDialog import AddMovieDetailsDialog #separate python file for inputting new movies
from EditMovieDetailsDialog import EditMovieDetailsDialog #separate python file for inputting movies to edit
```

```python
class main_window(QMainWindow): # the main window
    def __init__(self):
        super(main_window, self).__init__() # initializes the window's function
        loadUi("main.ui", self) # directly loads the main UI without converting it to a py file

        self.db = ConnectDatabase() # connect to MySQL DB

        self.display_all() # displays the current datas on the table

        self.show_all_movies.clicked.connect(self.display_all) # shows main table
        self.show_genres.clicked.connect(self.display_genres) # shows genres table
        self.show_studios.clicked.connect(self.display_studios) # shows studios table

        self.view_btn.clicked.connect(self.get_selected_data) #view button
        self.add_btn.clicked.connect(self.open_add_movie_dialog) # add button
        self.edit_btn.clicked.connect(self.open_edit_movie_dialog) # edit button
        self.delete_btn.clicked.connect(self.delete_movie)  # delete button
```

```python
    # edit a certain movie in the DB
    def open_edit_movie_dialog(self):
        selected_items = self.tableWidget.selectedItems() # select movie through clicking rows

        if not selected_items: # if nothing was selected or empty
            QMessageBox.warning(self, "No Selection", "Please select a movie to edit.")
            return

        # selects the specific cell
        row = selected_items[0].row()
        movie_id_item = self.tableWidget.item(row, 0)

        # handle error for invalid input
        if not movie_id_item:
            QMessageBox.warning(self, "Error", "Could not get movie ID for editing.")
            return

        # movie ID as text
        movie_id = movie_id_item.text()

        try:
            if not self.db or not self.db.cursor: # handles DB connection error
                raise Exception("Database not connected.")

            cursor = self.db.cursor
```

```
53
54              # DB query for fetching details later
55              query = '''
56                  SELECT
57                      m.movie_id,
58                      m.movie_name,
59                      m.release_year,
60                      g.genre_id,  -- Make sure genre_id is selected
61                      g.genre_name,
62                      s.studio_name,
63                      s.studio_id, -- Make sure studio_id is selected
64                      s.founded_year,
65                      s.headquarters
66                  FROM
67                      movie m
68                  JOIN
69                      genre g ON m.genre_id = g.genre_id
70                  JOIN
71                      studio s ON m.studio_id = s.studio_id
72                  WHERE
73                      m.movie_id = %s;
74              '''
75              # executing the query with the inputted primary key
76              cursor.execute(query, (movie_id,))
77              movie_data = cursor.fetchone() # fetching the selected movie
78
79              if not movie_data: # handle invalid selection
80                  QMessageBox.warning(self, "Not Found", "No details found for selected movie to edit.")
81                  return
82
```

```
82
83              # passes the fetched movie_data to the dialog
84              dialog = EditMovieDetailsDialog(self.db, movie_data)
85              if dialog.exec_() == QDialog.Accepted:
86                  self.display_all() # refresh table after successful edit
87
88          except Exception as e: # handles error when loading data from DB
89              QMessageBox.critical(self, "Error", f"Could not load movie data for editing:\n{str(e)}")
90
```

```
91      # add a movie with its details
92      def open_add_movie_dialog(self):
93          dialog = AddMovieDetailsDialog(self.db) # dialog for inputting details
94          if dialog.exec_() == QDialog.Accepted:
95              self.display_all() # refresh the main table after adding a movie
96
```

```python

    # view a movie's details
    def get_selected_data(self):
        selected_items = self.tableWidget.selectedItems() # selecting row by just clicking

        if not selected_items: # handles invalid selections
            QMessageBox.warning(self, "No Selection", "Please select a movie first.")
            return None

        # selects the specific cell
        row = selected_items[0].row()
        movie_id_item = self.tableWidget.item(row, 0)

        # handle error for invalid input
        if not movie_id_item:
            QMessageBox.warning(self, "Error", "Could not get movie ID.")
            return None

        # movie ID as text
        movie_id = movie_id_item.text()

        try:
            if not self.db or not self.db.cursor: # handles DB connection error
                raise Exception("Database not connected.")

```

```python

                # DB query for fetching details later
                cursor = self.db.cursor
                query = '''
                    SELECT
                        m.movie_id,
                        m.movie_name,
                        m.release_year,
                        g.genre_id,
                        g.genre_name,
                        s.studio_name,
                        s.studio_id,
                        s.founded_year,
                        s.headquarters
                    FROM
                        movie m
                    JOIN
                        genre g ON m.genre_id = g.genre_id
                    JOIN
                        studio s ON m.studio_id = s.studio_id
                    WHERE
                        m.movie_id = %s
                    ORDER BY
                        m.movie_id;
                '''

```

```
120
121             # DB query for fetching details later
122             cursor = self.db.cursor
123             query = '''
124                 SELECT
125                     m.movie_id,
126                     m.movie_name,
127                     m.release_year,
128                     g.genre_id,
129                     g.genre_name,
130                     s.studio_name,
131                     s.studio_id,
132                     s.founded_year,
133                     s.headquarters
134                 FROM
135                     movie m
136                 JOIN
137                     genre g ON m.genre_id = g.genre_id
138                 JOIN
139                     studio s ON m.studio_id = s.studio_id
140                 WHERE
141                     m.movie_id = %s
142                 ORDER BY
143                     m.movie_id;
144             '''
```

```
162     def delete_movie(self):
163         selected_items = self.tableWidget.selectedItems() #Deletes the selected movie from the database after confirmation.
164 
165         if not selected_items: # handles invalid selections
166             QMessageBox.warning(self, "No Selection", "Please select a movie to delete.")
167             return
168 
169         row = selected_items[0].row()
170         movie_id_item = self.tableWidget.item(row, 0)  # get Movie ID from the first column
171 
172         if not movie_id_item: # handles invalid inputs
173             QMessageBox.warning(self, "Error", "Could not get movie ID for deletion.")
174             return
175 
176         # movie ID as text
177         movie_id = movie_id_item.text()
178 
179         # Confirmation dialog
180         confirmation = QMessageBox.question(
181             self,
182             "Confirm Delete",
183             f"Are you sure you want to delete movie with ID {movie_id}?",
184             QMessageBox.Yes | QMessageBox.No,  # use Yes and No buttons
185             QMessageBox.No,  # default button is No
186         )
```

```
187
188         if confirmation == QMessageBox.Yes: # proceeds to deletion
189             try:
190                 if not self.db or not self.db.cursor:
191                     raise Exception("Database not connected.")
192 
193                 cursor = self.db.cursor
194                 query = "DELETE FROM movie WHERE movie_id = %s" # uses primary to delete
195                 cursor.execute(query, (movie_id,))
196                 self.db.con.commit()  # Use self.db.con to commit
197 
198                 QMessageBox.information(self, "Success", "Movie deleted successfully!")
199                 self.display_all()  # refresh the table after deletion
200 
201             except Exception as e:
202                 self.db.con.rollback()  # Use self.db.con to rollback
203                 QMessageBox.critical(self, "Error", f"Could not delete movie:\n{str(e)}")
204
```

```
204 
205     # shows all current movies in the database
206     def display_all(self):
207         try:
208             if not self.db or not self.db.cursor: # handles DB connection error
209                 raise Exception("Database not connected.")
210 
211             # DB query for fetching details later
212             query = '''
213                 SELECT
214                     m.movie_id,
215                     m.movie_name,
216                     m.release_year,
217                     g.genre_name,
218                     s.studio_name
219                 FROM
220                     movie m
221                 JOIN
222                     genre g ON m.genre_id = g.genre_id
223                 JOIN
224                     studio s ON m.studio_id = s.studio_id
225                 ORDER BY
226                     m.movie_id;
227             '''
```

```
            '''
            # executing the query
            self.db.cursor.execute(query)
            results = self.db.cursor.fetchall() # fetching all the datas in the table

            headers = ["ID", "Title", "Year", "Genre", "Studio"] # name of the headers that will be displayed
            self.tableWidget.setColumnCount(len(headers)) # counts columns
            self.tableWidget.setHorizontalHeaderLabels(headers) # assign the headers that will be shown
            self.tableWidget.setRowCount(len(results)) # counts rows

            # builds the table based on the number of columns and rows while also placing their data
            for row_idx, row_data in enumerate(results):
                for col_idx, value in enumerate(row_data):
                    self.tableWidget.setItem(row_idx, col_idx, QTableWidgetItem(str(value)))

            self.tableWidget.resizeColumnsToContents() # resizes the columns based on text length

            # enables row selection
            self.tableWidget.setSelectionBehavior(QTableWidget.SelectRows)
            self.tableWidget.setSelectionMode(QTableWidget.SingleSelection)

        except Exception as e: # handles error when loading data from DB
            QMessageBox.critical(self, "Error", f"Could not load data:\n{str(e)}")

    # display the genre table
    def display_genres(self):
        try:
            if not self.db or not self.db.cursor: # handles database connection failure
                raise Exception("Database not connected.")

            query = 'SELECT * FROM genre' # query for selecting all contents in the table

            self.db.cursor.execute(query) # executing the query
            results = self.db.cursor.fetchall() # fetching the data

            headers = ["Genre ID", "Genre Name"] # for headers to be displayed
            self.tableWidget.setColumnCount(len(headers)) # counts number of columns
            self.tableWidget.setHorizontalHeaderLabels(headers) # assign the headers that will be shown
            self.tableWidget.setRowCount(len(results)) # counts rows

            # builds the table based on the number of columns and rows while also placing their data
            for row_idx, row_data in enumerate(results):
                for col_idx, value in enumerate(row_data):
                    self.tableWidget.setItem(row_idx, col_idx, QTableWidgetItem(str(value)))

            self.tableWidget.resizeColumnsToContents() # resizes the columns based on text length

        except Exception as e: # handles error when loading data from DB
            QMessageBox.critical(self, "Error", f"Could not load data:\n{str(e)}")

    # displays the studios table
```

```
276
277     # displays the studios table
278     def display_studios(self):
279         try:
280             if not self.db or not self.db.cursor: # handles database connection failure
281                 raise Exception("Database not connected.")
282
283             query = 'SELECT * FROM studio' # query for selecting all contents in the table
284
285             self.db.cursor.execute(query) # executing the query
286             results = self.db.cursor.fetchall() # fetching the data
287
288             headers = ["Studio ID", "Studio Name", "Year Founded", "Headquartes"] # for headers to be displayed
289             self.tableWidget.setColumnCount(len(headers)) # counts number of columns
290             self.tableWidget.setHorizontalHeaderLabels(headers) # assign the headers that will be shown
291             self.tableWidget.setRowCount(len(results)) # counts rows
292
293             # builds the table based on the number of columns and rows while also placing their data
294             for row_idx, row_data in enumerate(results):
295                 for col_idx, value in enumerate(row_data):
296                     self.tableWidget.setItem(row_idx, col_idx, QTableWidgetItem(str(value)))
297
298             self.tableWidget.resizeColumnsToContents() # resizes the columns based on text length
299
300         except Exception as e: # handles error when loading data from DB
301             QMessageBox.critical(self, "Error", f"Could not load data:\n{str(e)}")
302
```

```
303 # executes the program in the main window
304 if __name__ == "__main__":
305     app = QApplication(sys.argv)
306     window = main_window()
307     window.show()
308     sys.exit(app.exec_())
309
310
```

```python
import mysql.connector

# for database connection
class ConnectDatabase:
    def __init__(self): # intializes MySQL credentials
        self._user = "root"
        self._password = "CS2025EU"
        self._database = "movie_manager"
        self.con = None
        self.cursor = None
        self.connect()

    def connect(self): # connects to the database with credentials
        try:
            self.con = mysql.connector.connect(
                user=self._user,
                password=self._password,
                database=self._database
            )
            self.cursor = self.con.cursor()
            print("Database connected.")
        except mysql.connector.Error as err:
            print(f"Failed to connect: {err}")
            self.cursor = None

```

EditMovieDetailsDialog.py > ...

```python
from PyQt5.QtWidgets import QDialog, QMessageBox
from PyQt5.uic import loadUi

# for editing details of a movie
class EditMovieDetailsDialog(QDialog):
    def __init__(self, db_connection, movie_data): # receives pre-fetched movie_data
        super(EditMovieDetailsDialog, self).__init__()
        loadUi("AddMovieDialog.ui", self) # reuses the same UI file
        self.db = db_connection
        self.movie_data = movie_data # store the data for populating fields

        self.populate_fields() # call this to fill the fields on dialog open

        self.buttonBox.clicked.connect(self.update_movie)

    def populate_fields(self):
        # unpack the movie_data tuple.
        (movie_id, movie_name, release_year, genre_id, genre_name,
         studio_name, studio_id, founded_year, headquarters) = self.movie_data

        # populate the QLineEdit fields
        self.add_movie_id.setText(str(movie_id))
        self.add_movie_id.setEnabled(False)
        self.add_movie_title.setText(movie_name)
        self.add_movie_year.setText(str(release_year))
        self.add_genre_id.setText(str(genre_id))
        self.add_genre_name.setText(genre_name)
        self.add_studio_id.setText(str(studio_id))
        self.add_studio_name.setText(studio_name)
        self.add_founded_year.setText(str(founded_year))
        self.add_headquarters.setText(headquarters)

```

```python
def update_movie(self):
    # get updated data from the QLineEdit fields
    movie_id = self.add_movie_id.text()
    movie_title = self.add_movie_title.text()
    release_year = self.add_movie_year.text()
    genre_id = self.add_genre_id.text()
    genre_name = self.add_genre_name.text()
    studio_id = self.add_studio_id.text()
    studio_name = self.add_studio_name.text()
    founded_year = self.add_founded_year.text()
    headquarters = self.add_headquarters.text()

    # validates the inputs
    if not all([movie_id, movie_title, release_year, genre_id, genre_name, studio_id, studio_name, founded_year, headquarters]):
        QMessageBox.warning(self, "Missing Data", "Please fill in all fields.")
        return
```

```python
    try:
        # check if genre exists, if not, insert it, otherwise, update it.
        self.db.cursor.execute("SELECT genre_id FROM genre WHERE genre_id = %s", (genre_id,))
        if not self.db.cursor.fetchone():
            self.db.cursor.execute("INSERT INTO genre (genre_id, genre_name) VALUES (%s, %s)", (genre_id, genre_name))
        else:
            self.db.cursor.execute("UPDATE genre SET genre_name = %s WHERE genre_id = %s", (genre_name, genre_id))

        # check if studio exists, if not, insert it, otherwise, update it.
        self.db.cursor.execute("SELECT studio_id FROM studio WHERE studio_id = %s", (studio_id,))
        if not self.db.cursor.fetchone():
            self.db.cursor.execute("INSERT INTO studio (studio_id, studio_name, founded_year, headquarters) VALUES (%s, %s, %s, %s)", (studio_id, studio_name, founded_year, headquarters))
        else:
            self.db.cursor.execute("UPDATE studio SET studio_name = %s, founded_year = %s, headquarters = %s WHERE studio_id = %s", (studio_name, founded_year, headquarters, studio_id))

        # updates movie data
        query = """
            UPDATE movie
            SET movie_name = %s, release_year = %s, genre_id = %s, studio_id = %s
            WHERE movie_id = %s
        """
        self.db.cursor.execute(query, (movie_title, release_year, genre_id, studio_id, movie_id))

        self.db.con.commit() # make the changes

        QMessageBox.information(self, "Success", "Movie updated successfully!")
        self.accept() # close the dialog with Accepted status
    except Exception as e: # handles data update error
        self.db.con.rollback()
        QMessageBox.critical(self, "Error", f"Could not update movie: {str(e)}")
```

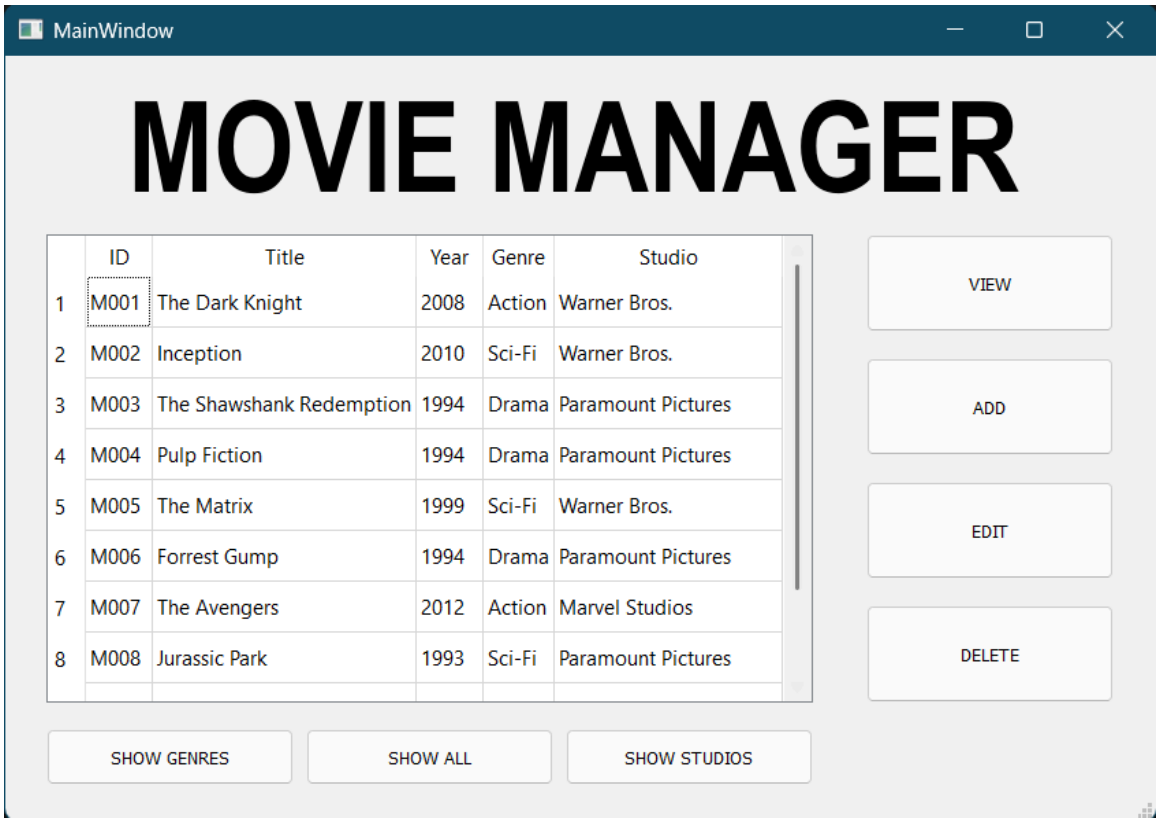
```python
from PyQt5.QtWidgets import QLabel, QDialog, QFormLayout, QPushButton

# for getting the details of a movie
class MovieDetailsDialog(QDialog):
    def __init__(self, movie_data, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Movie Details") # window name
        self.setModal(True) # blocks interaction with other windows in the application
        self.resize(400, 300) # windows size

        layout = QFormLayout() # for arrangement of the data

        # movie Information Section
        layout.addRow(QLabel("<b>Movie Information</b>"))
        layout.addRow("ID:", QLabel(str(movie_data[0])))
        layout.addRow("Title:", QLabel(str(movie_data[1])))
        layout.addRow("Release Year:", QLabel(str(movie_data[2])))

        # genre Information Section
        layout.addRow(QLabel("<b>Genre Information</b>"))
        layout.addRow("Genre ID:", QLabel(str(movie_data[3])))
        layout.addRow("Genre Name:", QLabel(str(movie_data[4])))

        # studio Information Section
        layout.addRow(QLabel("<b>Studio Information</b>"))
        layout.addRow("Studio Name:", QLabel(str(movie_data[5])))
        layout.addRow("Studio ID:", QLabel(str(movie_data[6])))
        layout.addRow("Founded Year:", QLabel(str(movie_data[7])))
        layout.addRow("Headquarters:", QLabel(str(movie_data[8])))

        # add a close button
        close_btn = QPushButton("Close")
        close_btn.clicked.connect(self.close)
        layout.addRow(close_btn)

        self.setLayout(layout)
```

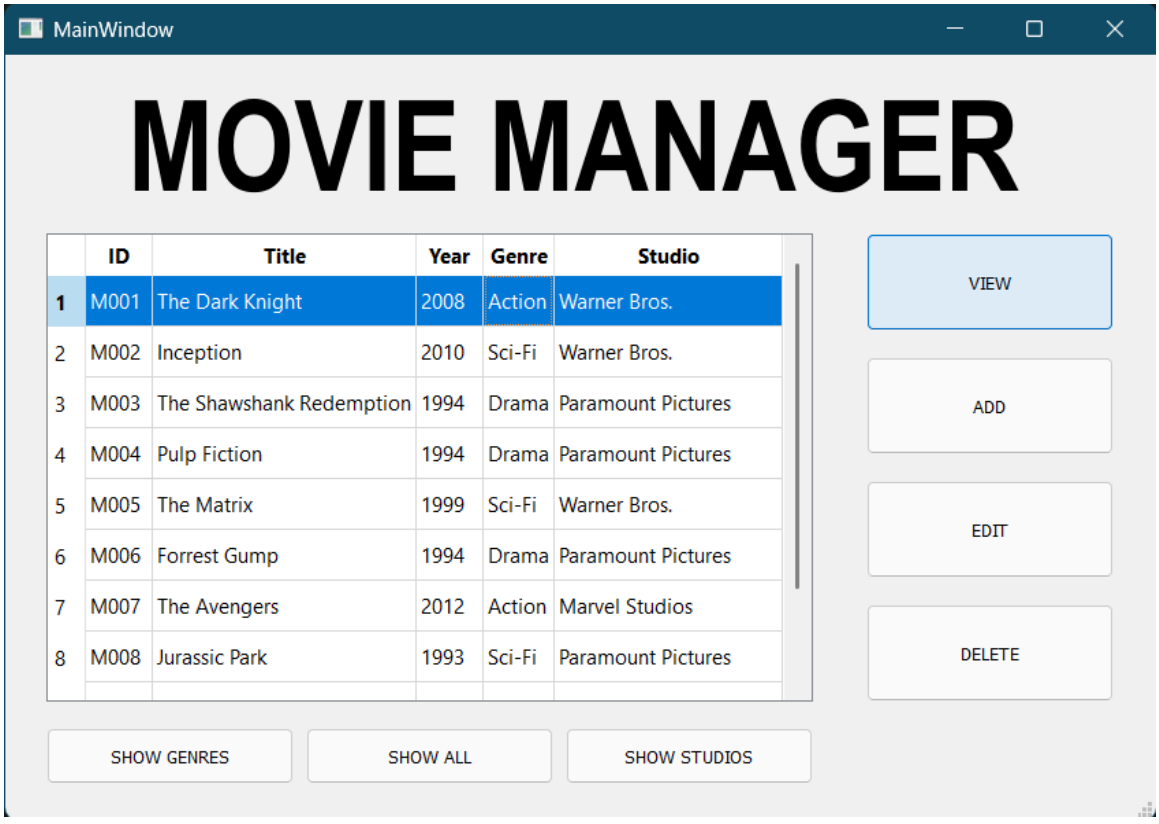
```python
from PyQt5.QtWidgets import QDialog, QMessageBox
from PyQt5.uic import loadUi

# putting new movie with required details
class AddMovieDetailsDialog(QDialog):
    def __init__(self, db_connection):
        super(AddMovieDetailsDialog, self).__init__()
        loadUi("AddMovieDialog.ui", self)
        self.db = db_connection

        self.buttonBox.clicked.connect(self.add_movie) # ok or cancel button box

    def add_movie(self):
        # get data from the QLineEdit fields
        movie_id = self.add_movie_id.text()
        movie_title = self.add_movie_title.text()
        release_year = self.add_movie_year.text()
        genre_id = self.add_genre_id.text()
        genre_name = self.add_genre_name.text()
        studio_id = self.add_studio_id.text()
        studio_name = self.add_studio_name.text()
        founded_year = self.add_founded_year.text()
        headquarters = self.add_headquarters.text()
```

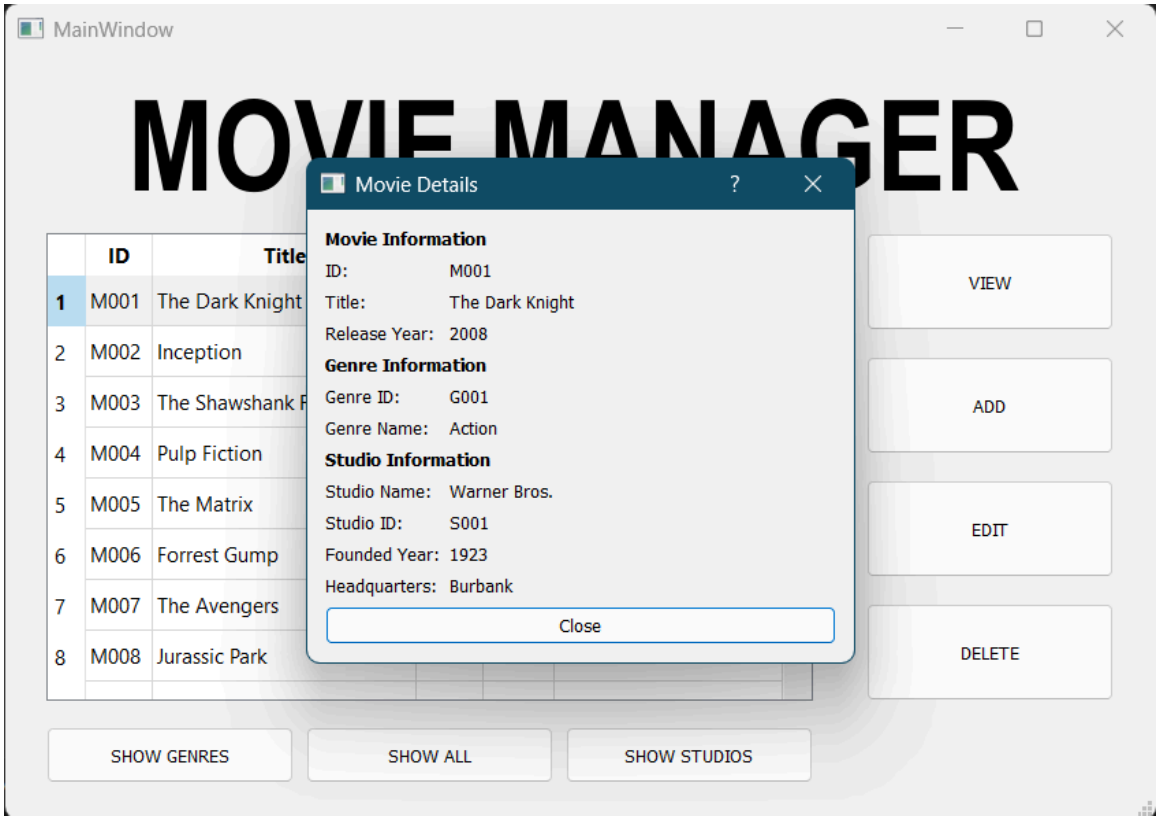
```python
    # validating the inputs
    if not all([movie_id, movie_title, release_year, genre_id, genre_name, studio_id, studio_name, founded_year, headquarters]):
        QMessageBox.warning(self, "Missing Data", "Please fill in all fields.")
        return

    try:
        # check if genre exists, if not, insert it
        self.db.cursor.execute("SELECT genre_id FROM genre WHERE genre_id = %s", (genre_id,))
        if not self.db.cursor.fetchone():
            self.db.cursor.execute("INSERT INTO genre (genre_id, genre_name) VALUES (%s, %s)", (genre_id, genre_name))

        # check if studio exists, if not, insert it
        self.db.cursor.execute("SELECT studio_id FROM studio WHERE studio_id = %s", (studio_id,))
        if not self.db.cursor.fetchone():
            self.db.cursor.execute("INSERT INTO studio (studio_id, studio_name, founded_year, headquarters) VALUES (%s, %s, %s, %s)", (studio_id, studio_name, founded_year, headquarters))
```

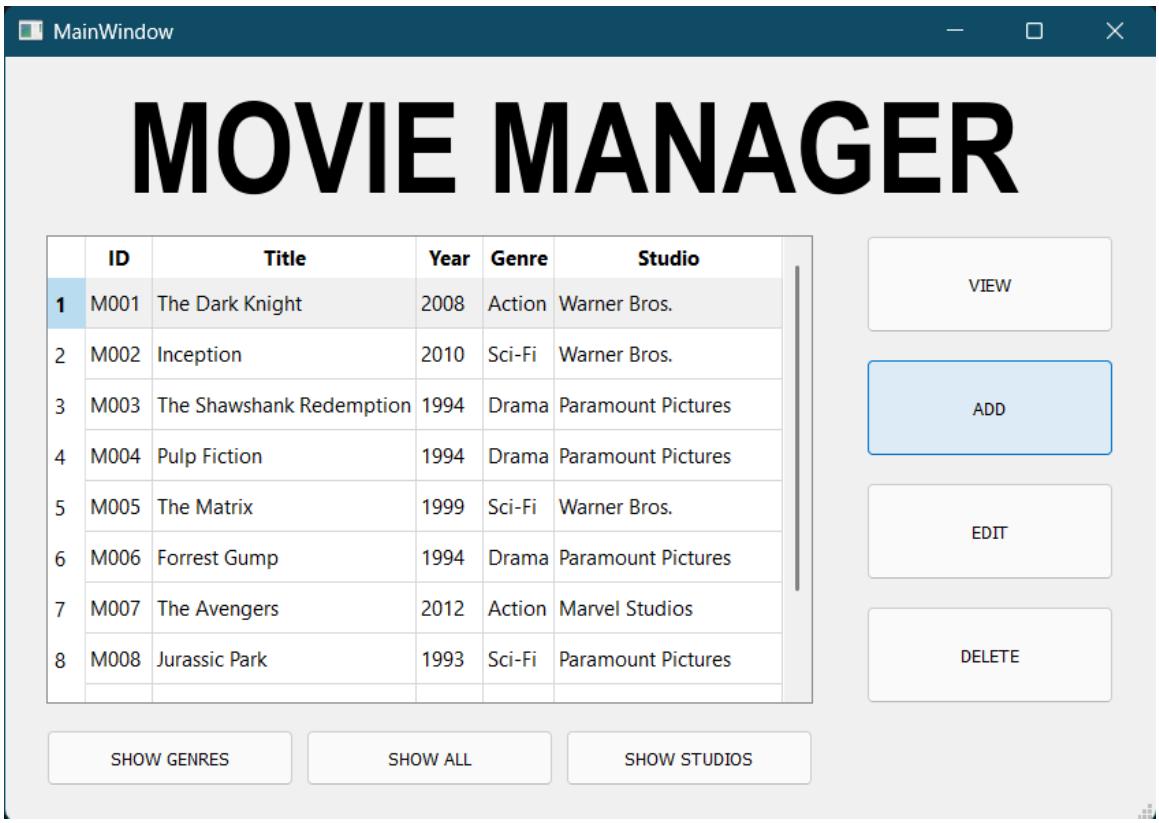
```python
            # insert movie data
            query = """
                INSERT INTO movie (movie_id, movie_name, release_year, genre_id, studio_id)
                VALUES (%s, %s, %s, %s, %s)
            """
            self.db.cursor.execute(query, (movie_id, movie_title, release_year, genre_id, studio_id))

            self.db.con.commit()

            QMessageBox.information(self, "Success", "Movie added successfully!")
            self.accept() # close the dialog if successful
        except Exception as e:

            self.db.con.rollback() # rollback in case of error
            QMessageBox.critical(self, "Error", f"Could not add movie: {str(e)}")
```

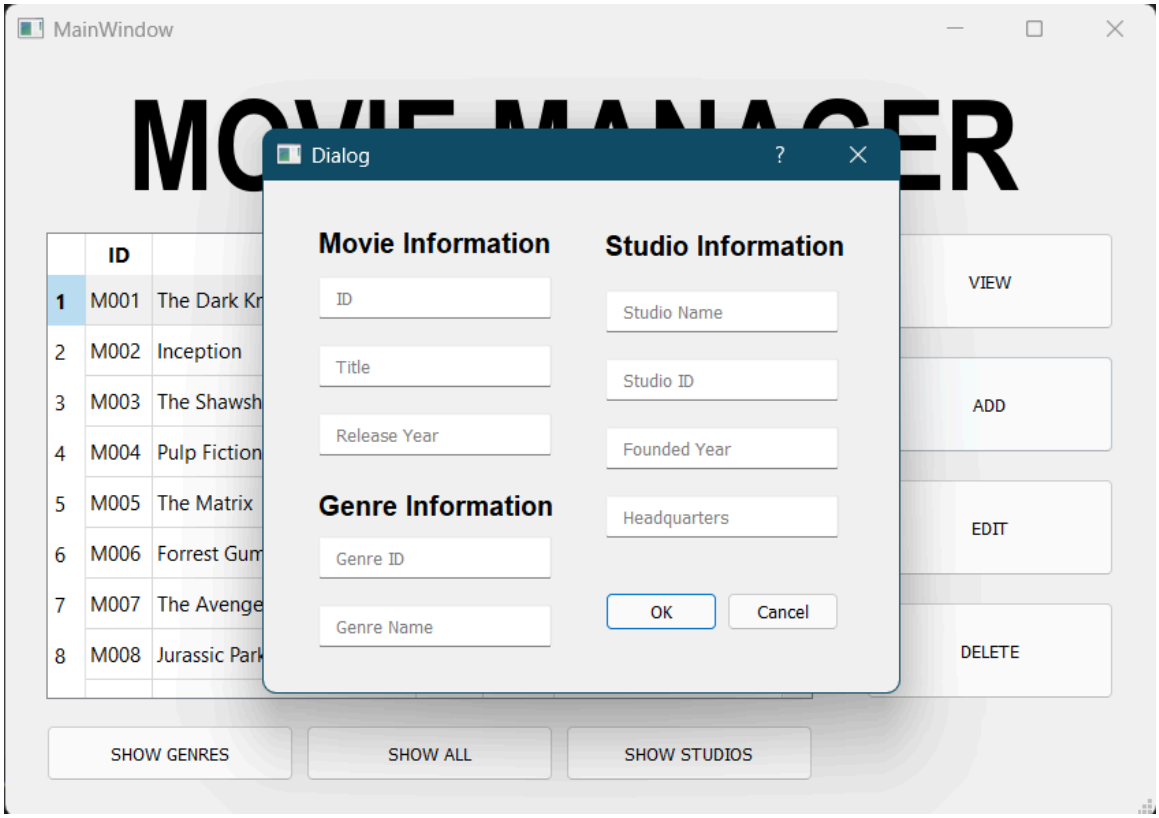
Program Outputs

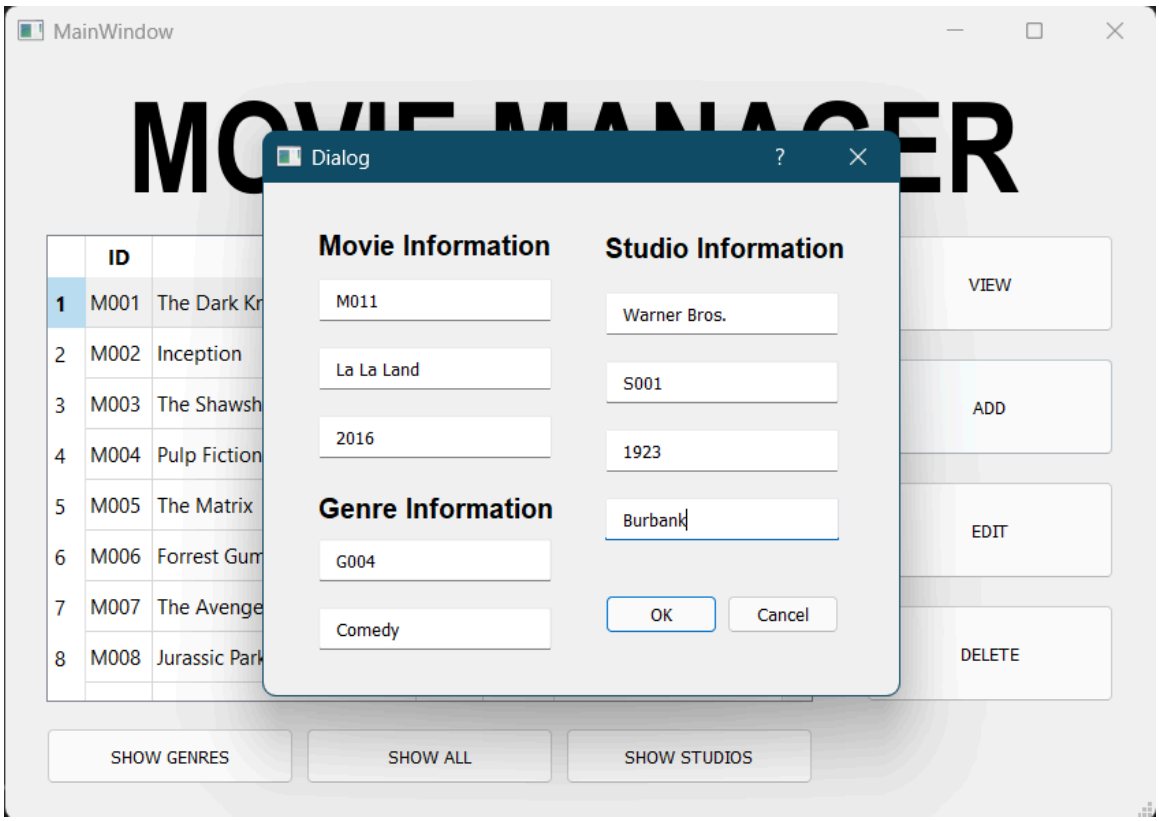


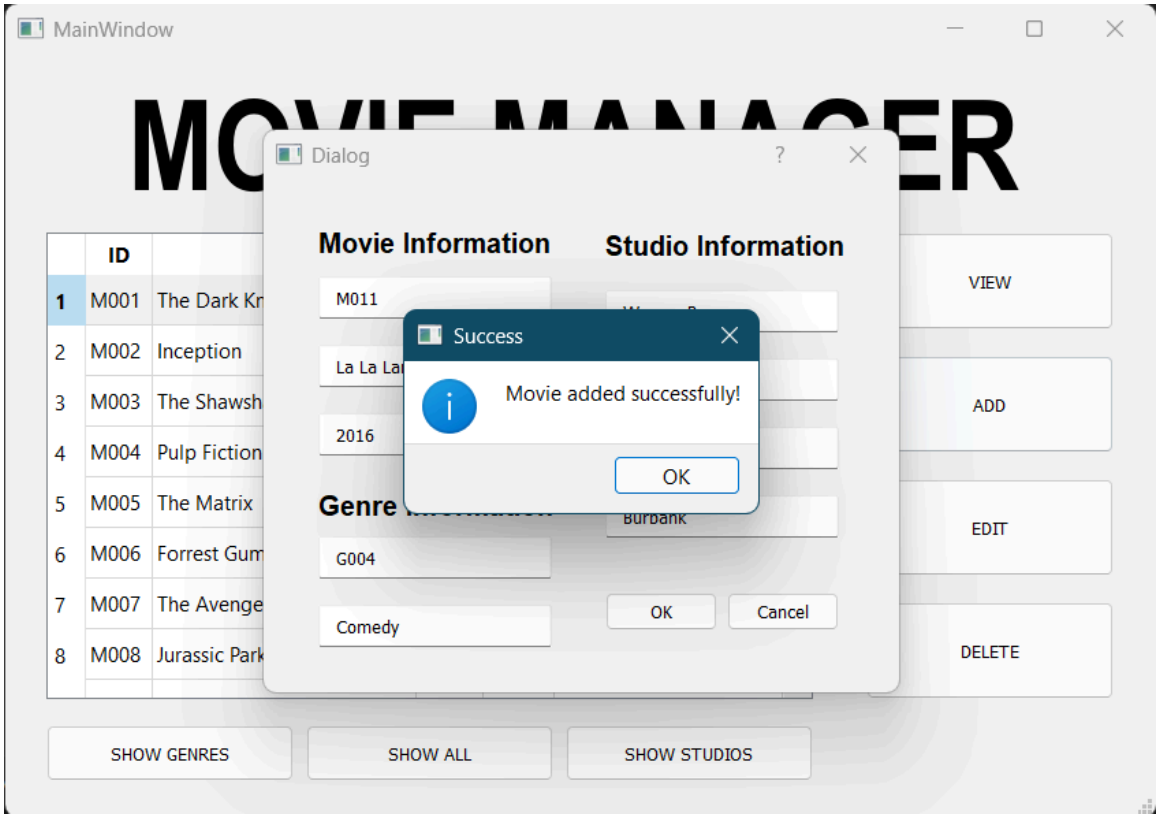


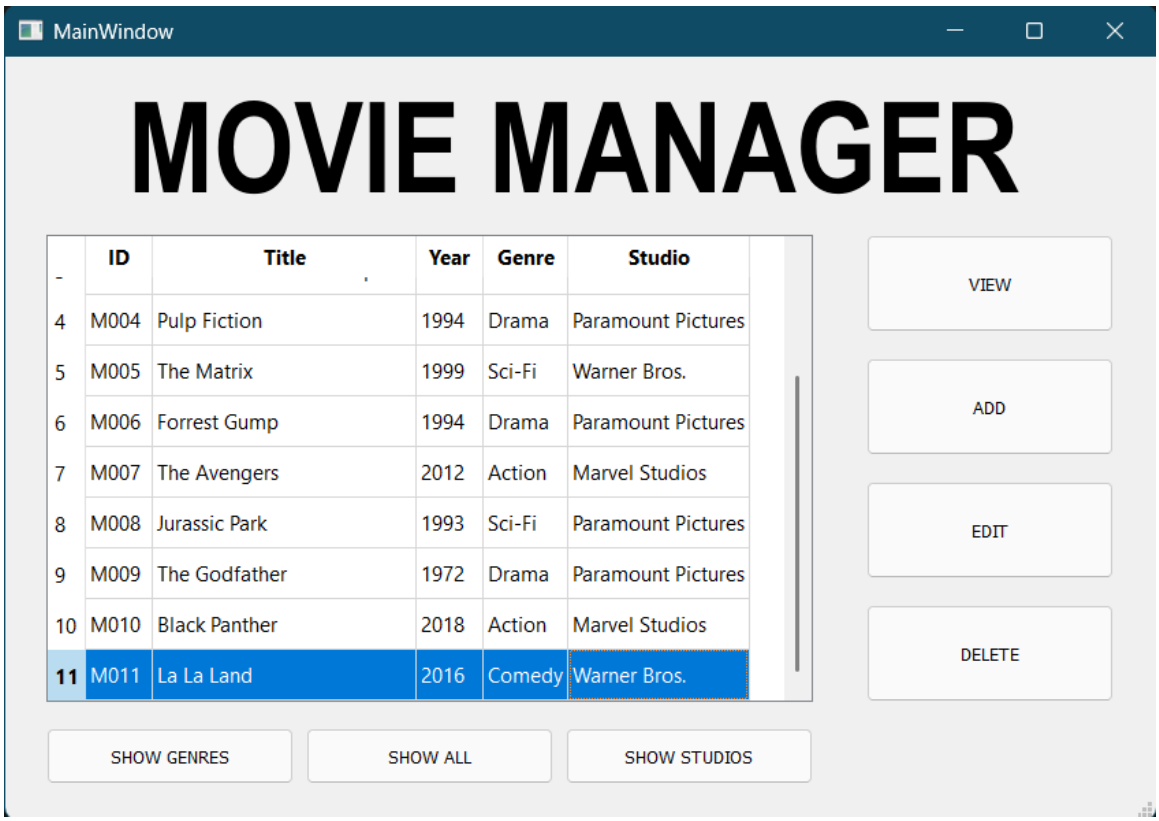

MainWindow
MOVIE MANAGER
Movie Details
Movie Information
ID: M001
Title: The Dark Knight
Release Year: 2008
Genre Information
Genre ID: G001
Genre Name: Action
Studio Information
Studio Name: Warner Bros.
Studio ID: S001
Founded Year: 1923
Headquarters: Burbank
Close
ID
Title
1 M001 The Dark Knight
2 M002 Inception
3 M003 The Shawshank
4 M004 Pulp Fiction
5 M005 The Matrix
6 M006 Forrest Gump
7 M007 The Avengers
8 M008 Jurassic Park
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

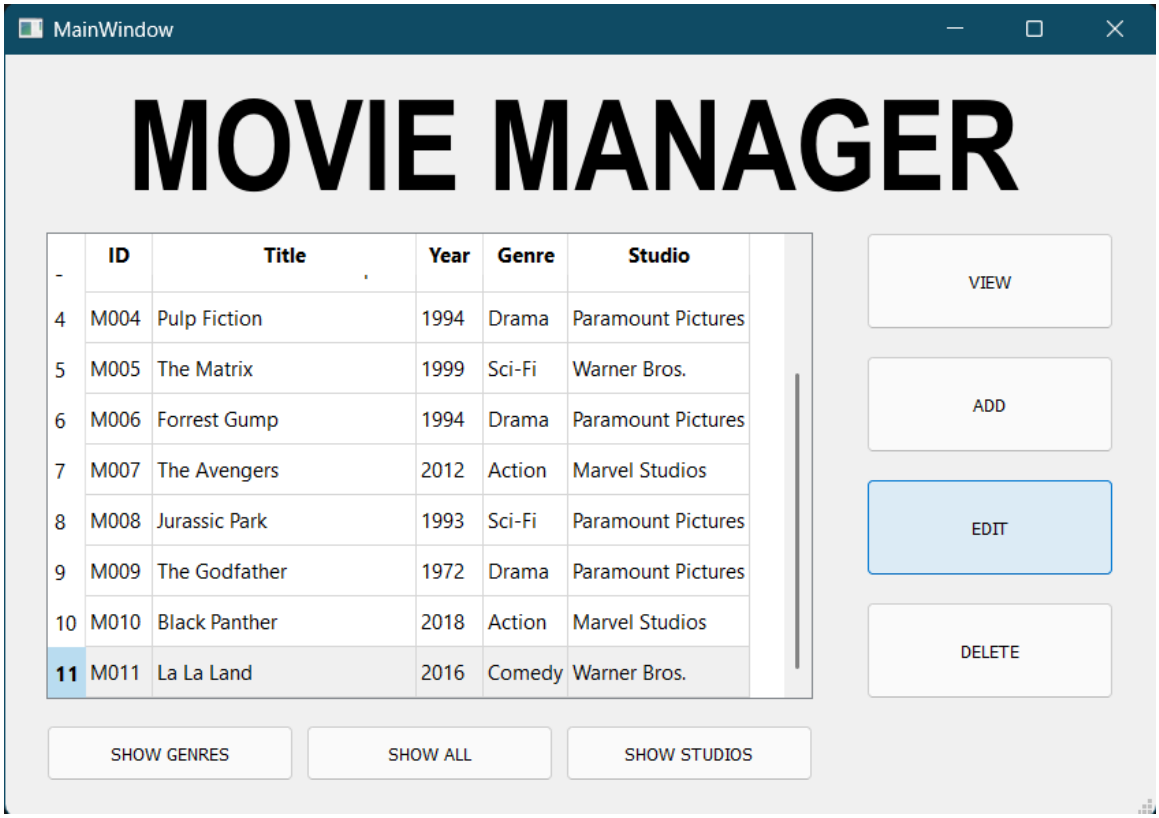

MainWindow
MOVIE MANAGER
ID Title Year Genre Studio
1 M001 The Dark Knight 2008 Action Warner Bros.
2 M002 Inception 2010 Sci-Fi Warner Bros.
3 M003 The Shawshank Redemption 1994 Drama Paramount Pictures
4 M004 Pulp Fiction 1994 Drama Paramount Pictures
5 M005 The Matrix 1999 Sci-Fi Warner Bros.
6 M006 Forrest Gump 1994 Drama Paramount Pictures
7 M007 The Avengers 2012 Action Marvel Studios
8 M008 Jurassic Park 1993 Sci-Fi Paramount Pictures
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

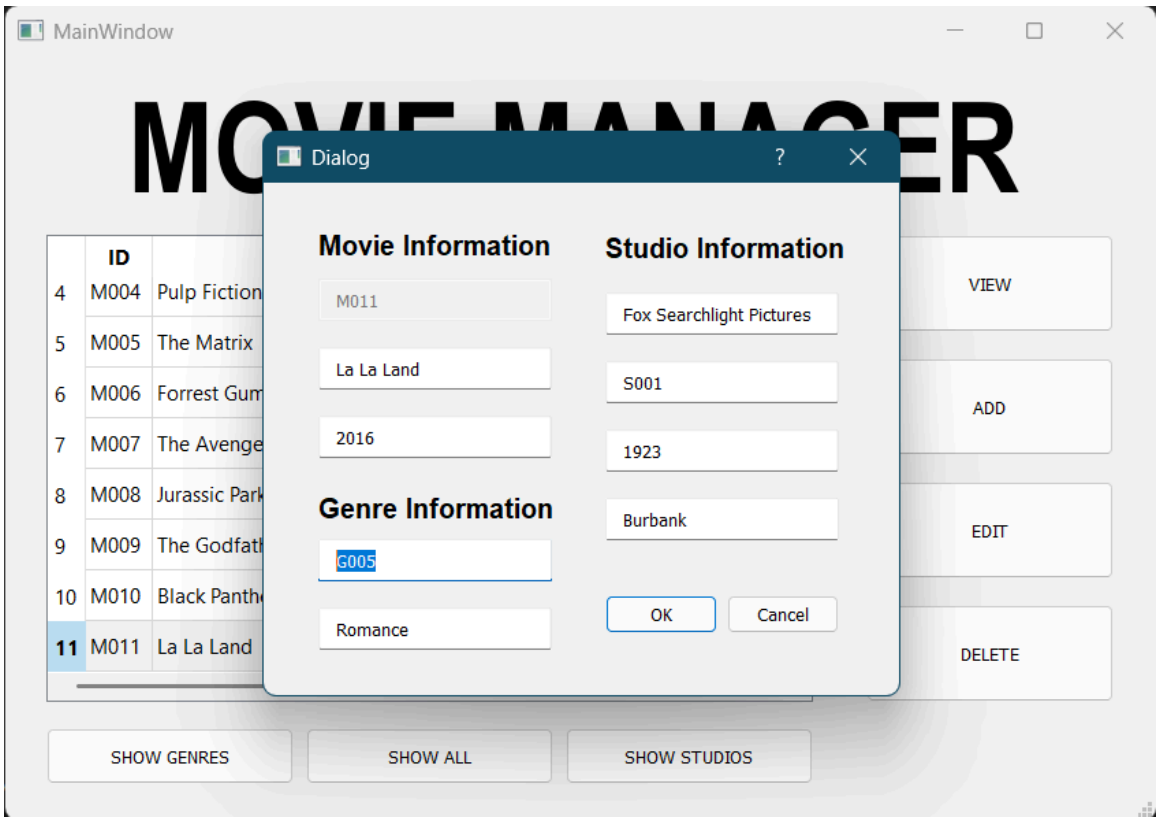

MainWindow
MOVIE MANAGER
Dialog
Movie Information
ID
Title
Release Year
Genre Information
Genre ID
Genre Name
Studio Information
Studio Name
Studio ID
Founded Year
Headquarters
OK
Cancel
ID
1 M001 The Dark Kr
2 M002 Inception
3 M003 The Shawsh
4 M004 Pulp Fiction
5 M005 The Matrix
6 M006 Forrest Gum
7 M007 The Avenge
8 M008 Jurassic Park
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

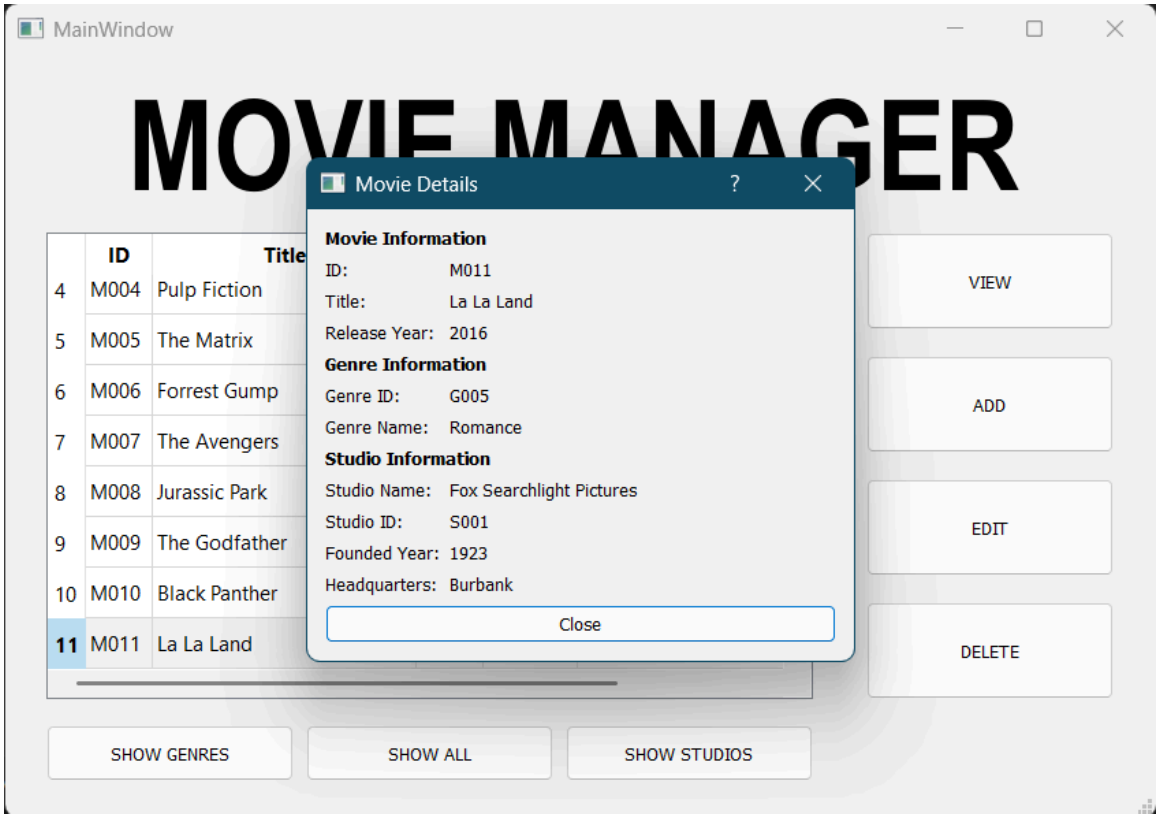

MainWindow
MOVIE MANAGER
Dialog
Movie Information
M011
La La Land
2016
Genre Information
G004
Comedy
Studio Information
Warner Bros.
S001
1923
Burbank
OK
Cancel
ID
1 M001 The Dark Kr
2 M002 Inception
3 M003 The Shawsh
4 M004 Pulp Fiction
5 M005 The Matrix
6 M006 Forrest Gum
7 M007 The Avenge
8 M008 Jurassic Park
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

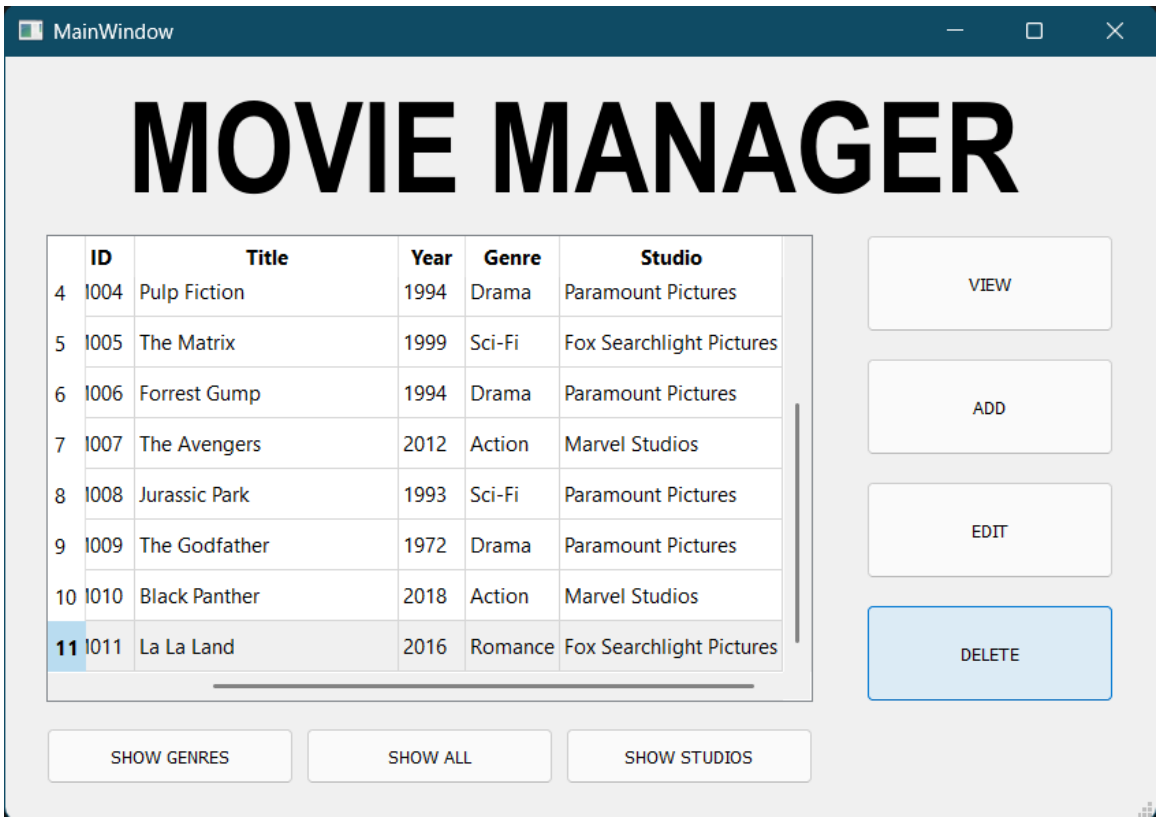

MainWindow
Dialog
Movie Information
Studio Information
M011
2016
G004
Comedy
Burbank
OK
Cancel
Success
Movie added successfully!
OK
ID
1 M001
2 M002 Inception
3 M003
4 M004 Pulp Fiction
5 M005 The Matrix
6 M006
7 M007
8 M008
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

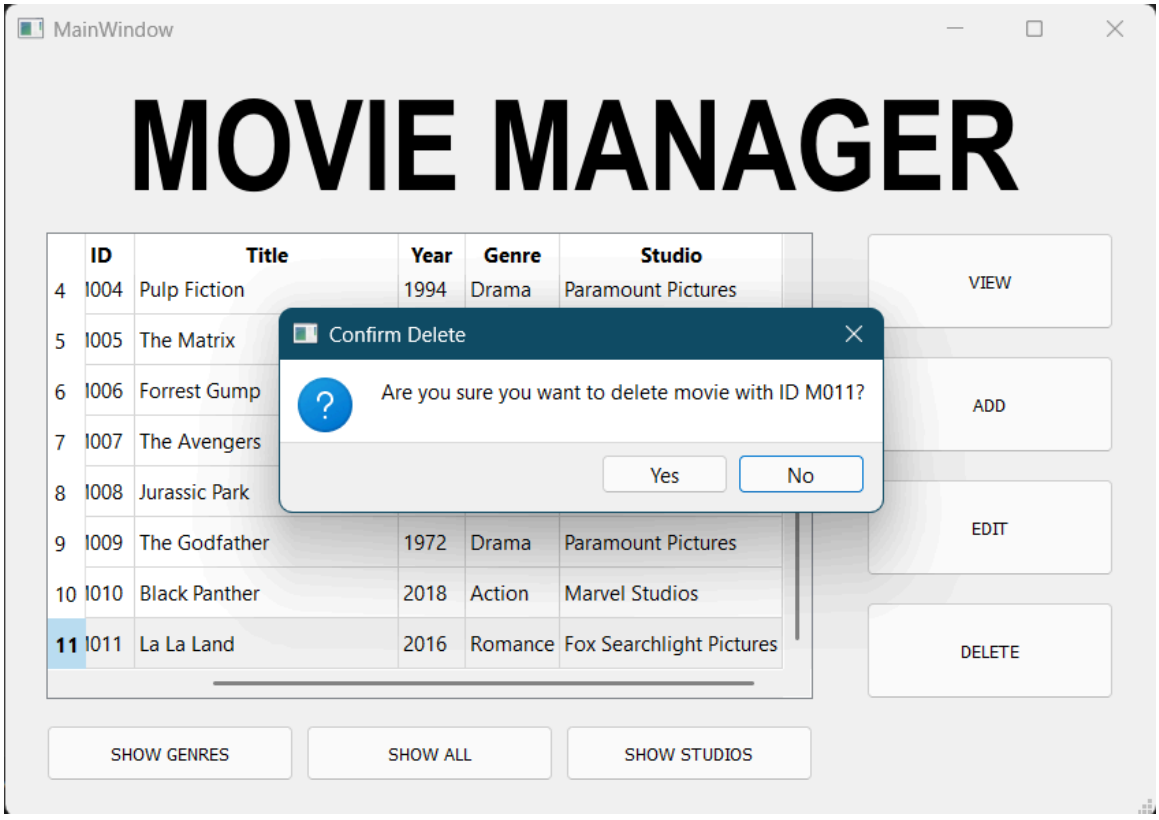

MainWindow
MOVIE MANAGER
ID
Title
Year
Genre
Studio
4 M004 Pulp Fiction 1994 Drama Paramount Pictures
5 M005 The Matrix 1999 Sci-Fi Warner Bros.
6 M006 Forrest Gump 1994 Drama Paramount Pictures
7 M007 The Avengers 2012 Action Marvel Studios
8 M008 Jurassic Park 1993 Sci-Fi Paramount Pictures
9 M009 The Godfather 1972 Drama Paramount Pictures
10 M010 Black Panther 2018 Action Marvel Studios
11 M011 La La Land 2016 Comedy Warner Bros.
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

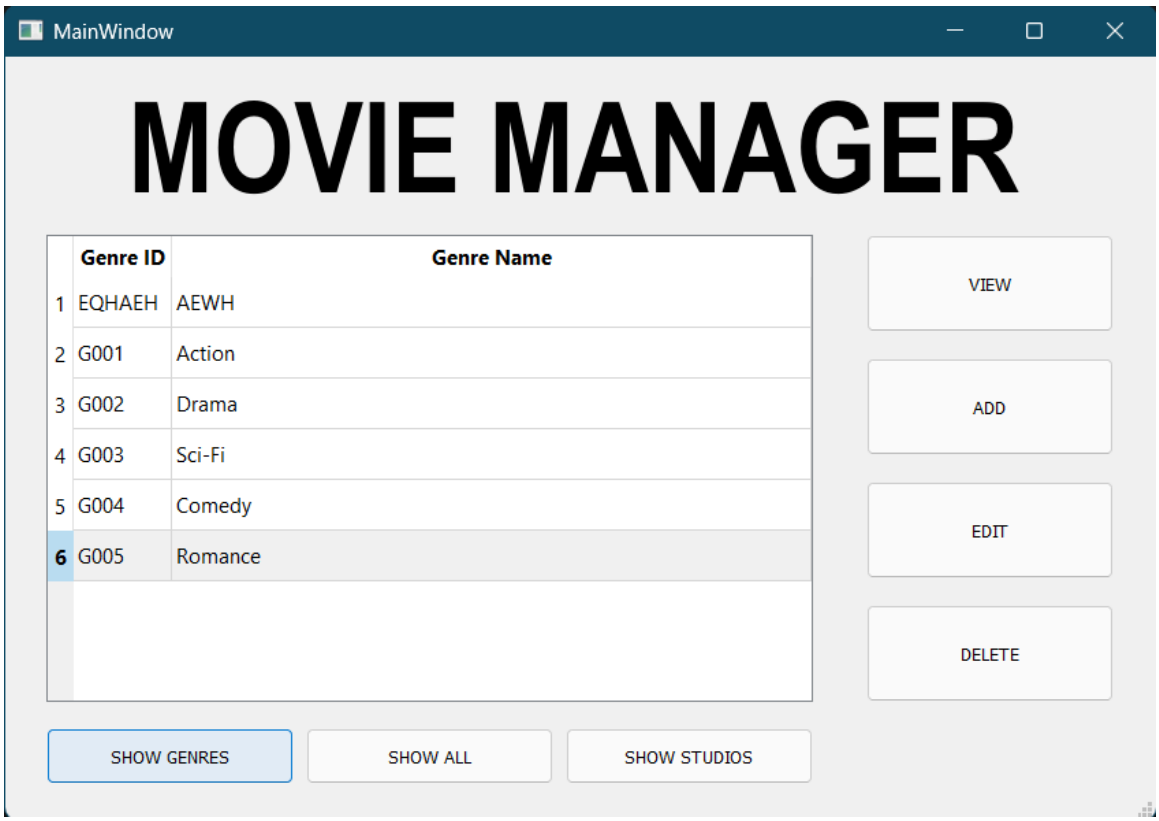

MainWindow
MOVIE MANAGER
ID
Title
Year
Genre
Studio
4 M004 Pulp Fiction 1994 Drama Paramount Pictures
5 M005 The Matrix 1999 Sci-Fi Warner Bros.
6 M006 Forrest Gump 1994 Drama Paramount Pictures
7 M007 The Avengers 2012 Action Marvel Studios
8 M008 Jurassic Park 1993 Sci-Fi Paramount Pictures
9 M009 The Godfather 1972 Drama Paramount Pictures
10 M010 Black Panther 2018 Action Marvel Studios
11 M011 La La Land 2016 Comedy Warner Bros.
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

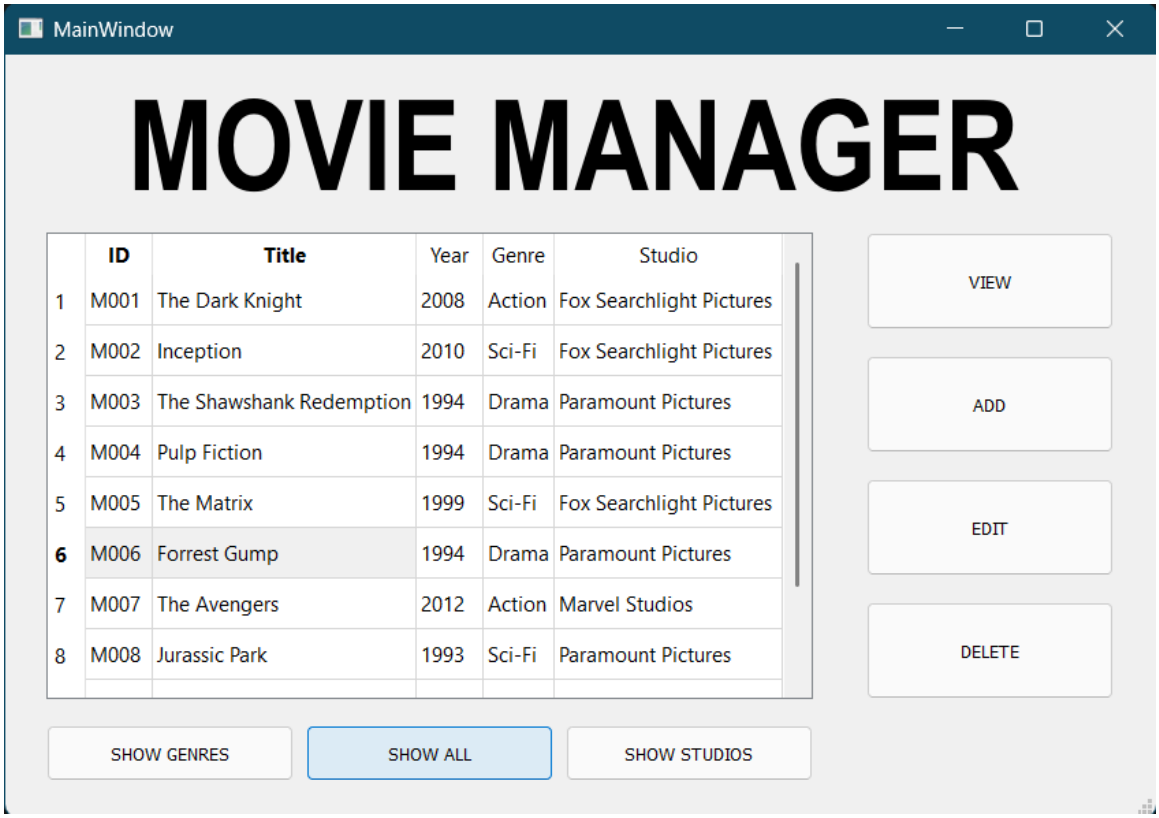

MainWindow
Dialog
Movie Information
M011
La La Land
2016
Genre Information
G005
Romance
Studio Information
Fox Searchlight Pictures
S001
1923
Burbank
OK
Cancel
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS

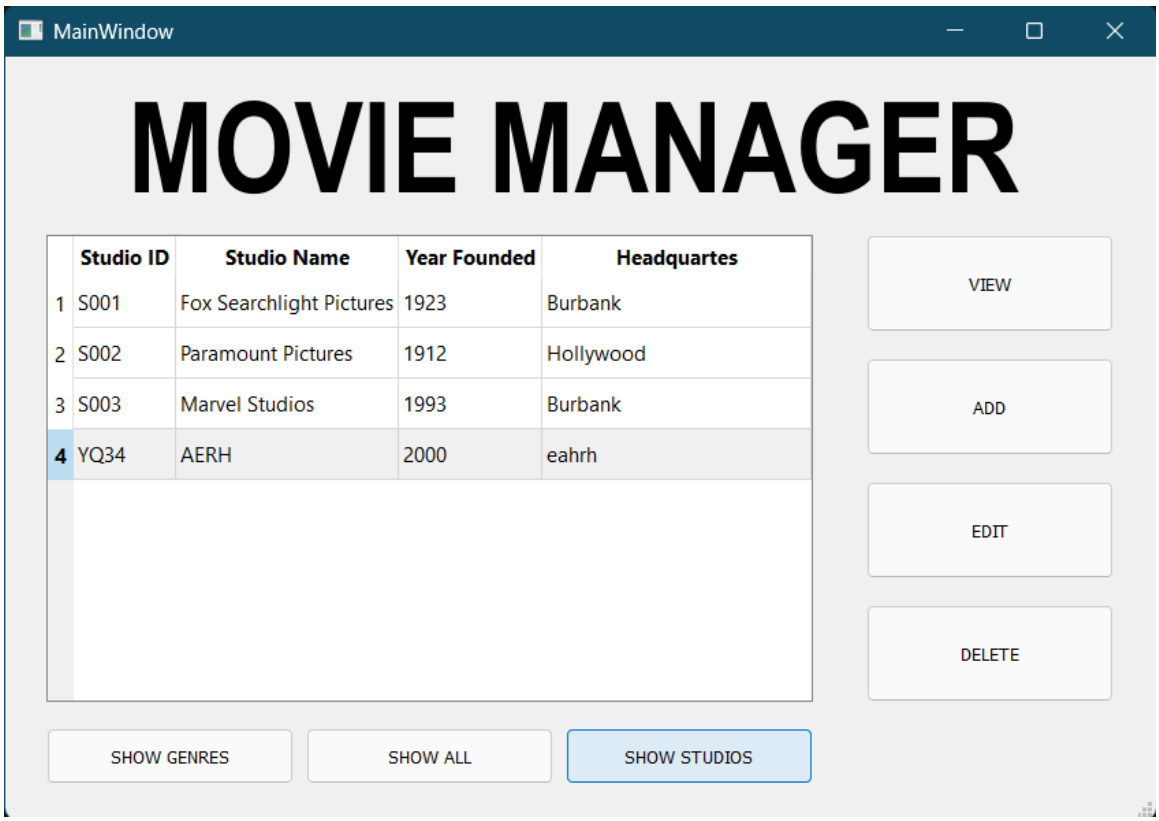

MainWindow
MOVIE MANAGER
Movie Details
Movie Information
ID: M011
Title: La La Land
Release Year: 2016
Genre Information
Genre ID: G005
Genre Name: Romance
Studio Information
Studio Name: Fox Searchlight Pictures
Studio ID: S001
Founded Year: 1923
Headquarters: Burbank
Close
ID
Title
4 M004 Pulp Fiction
5 M005 The Matrix
6 M006 Forrest Gump
7 M007 The Avengers
8 M008 Jurassic Park
9 M009 The Godfather
10 M010 Black Panther
11 M011 La La Land
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS


MainWindow
MOVIE MANAGER
ID Title Year Genre Studio
4 I004 Pulp Fiction 1994 Drama Paramount Pictures
5 I005 The Matrix 1999 Sci-Fi Fox Searchlight Pictures
6 I006 Forrest Gump 1994 Drama Paramount Pictures
7 I007 The Avengers 2012 Action Marvel Studios
8 I008 Jurassic Park 1993 Sci-Fi Paramount Pictures
9 I009 The Godfather 1972 Drama Paramount Pictures
10 I010 Black Panther 2018 Action Marvel Studios
11 I011 La La Land 2016 Romance Fox Searchlight Pictures
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS


MainWindow
MOVIE MANAGER
ID
Title
Year
Genre
Studio
4 I004 Pulp Fiction 1994 Drama Paramount Pictures
5 I005 The Matrix
6 I006 Forrest Gump
7 I007 The Avengers
8 I008 Jurassic Park
9 I009 The Godfather 1972 Drama Paramount Pictures
10 I010 Black Panther 2018 Action Marvel Studios
11 I011 La La Land 2016 Romance Fox Searchlight Pictures
Confirm Delete
Are you sure you want to delete movie with ID M011?
Yes
No
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS


MainWindow
MOVIE MANAGER
Genre ID
Genre Name
1 EQHAEH AEWH
2 G001 Action
3 G002 Drama
4 G003 Sci-Fi
5 G004 Comedy
6 G005 Romance
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS


MainWindow
MOVIE MANAGER
ID
Title
Year
Genre
Studio
1 M001 The Dark Knight 2008 Action Fox Searchlight Pictures
2 M002 Inception 2010 Sci-Fi Fox Searchlight Pictures
3 M003 The Shawshank Redemption 1994 Drama Paramount Pictures
4 M004 Pulp Fiction 1994 Drama Paramount Pictures
5 M005 The Matrix 1999 Sci-Fi Fox Searchlight Pictures
6 M006 Forrest Gump 1994 Drama Paramount Pictures
7 M007 The Avengers 2012 Action Marvel Studios
8 M008 Jurassic Park 1993 Sci-Fi Paramount Pictures
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS


MainWindow
MOVIE MANAGER
Studio ID
Studio Name
Year Founded
Headquartes
1 S001 Fox Searchlight Pictures 1923 Burbank
2 S002 Paramount Pictures 1912 Hollywood
3 S003 Marvel Studios 1993 Burbank
4 YQ34 AERH 2000 eahrh
VIEW
ADD
EDIT
DELETE
SHOW GENRES
SHOW ALL
SHOW STUDIOS