I.

1) E
2) B
3) D

II.

a.

```
CREATE DATABASE FilmFestival
GO

USE FilmFestival
GO

--- Producer:Movie relationship is 1:m,
--- i.e. a movie has one producer and a producer can produce multiple movies,
--- that is why we hold a reference to the producer for each movie

--- Cathegory:Movie relationship is 1:m,
--- i.e. a movie has one cathegory and a to a cathegory may belong multiple movies,
--- that is why we hold a reference to the cathegory for each movie

--- Cathegory:Movie relationship is 1:m,
--- i.e. a movie has one cathegory and a to a cathegory may belong multiple movies,
--- that is why we hold a reference to the cathegory for each movie

--- Movie:Schedule relationship is 1:m,
--- i.e. a movie can be scheduled in multiple schedules and a schedule can have one movie
--- that is why we hold a reference to the movie for each schedule

--- Cinema:Schedule relationship is 1:m,
--- i.e. a cinema can be in multiple schedules and a schedule can be scheduled for one
cinema
--- that is why we hold a reference to the cinema for each schedule

--- Schedules table has the tuple (Cinema, Time) as a PK
--- because a cinema can be scheduled only once for a given time


--- The database design is in 3NF because it is in 2NF and
--- there is no non-prime attribute which is transitively dependent on any key in the
relation


CREATE TABLE Cathegories(
    CathegoryID INT PRIMARY KEY IDENTITY(1,1),
       CathegoryName VARCHAR(50),
       CathegoryDescription VARCHAR(500),
)

CREATE TABLE Producers(
    ProducerID INT PRIMARY KEY IDENTITY(1,1),
       ProducerFirstName VARCHAR(50),
```

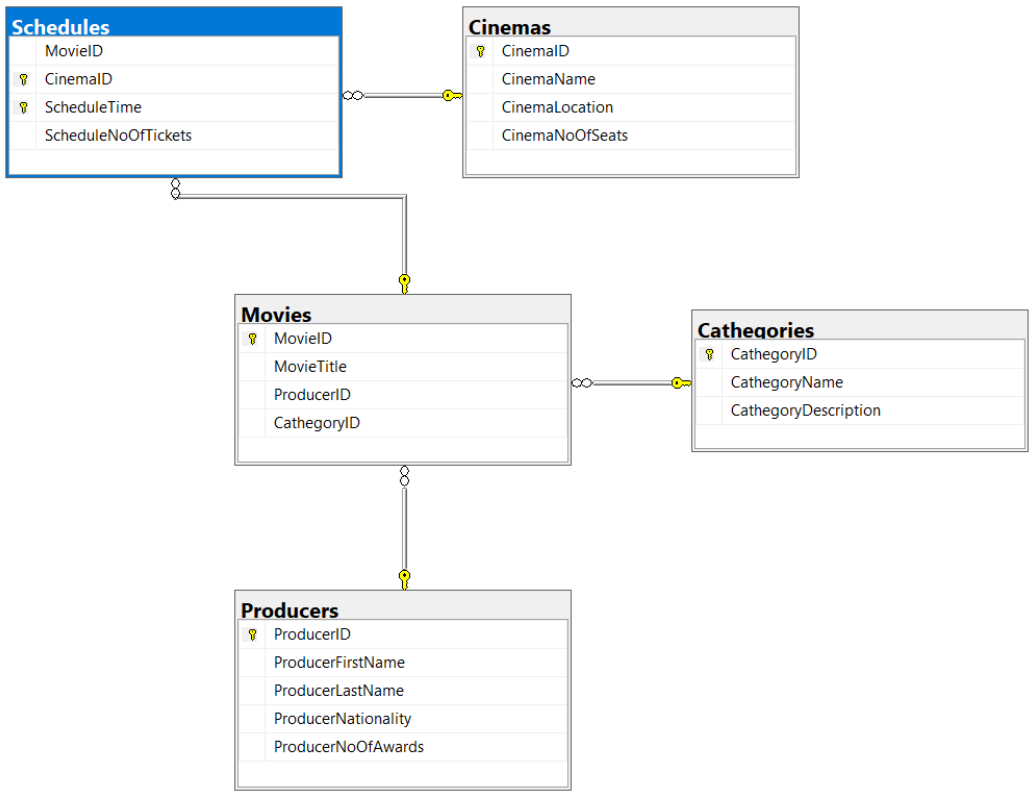
```
       ProducerLastName VARCHAR(50),
       ProducerNationality VARCHAR(50),
       ProducerNoOfAwards SMALLINT,
)

CREATE TABLE Movies(
    MovieID INT PRIMARY KEY IDENTITY(1,1),
       MovieTitle VARCHAR(50),
       ProducerID INT REFERENCES Producers(ProducerID),
       CathegoryID INT REFERENCES Cathegories(CathegoryID)
)

CREATE TABLE Cinemas(
       CinemaID INT PRIMARY KEY IDENTITY(1, 1),
       CinemaName VARCHAR(50),
       CinemaLocation VARCHAR(100),
       CinemaNoOfSeats SMALLINT
)

CREATE TABLE Schedules(
       MovieID INT REFERENCES Movies(MovieID),
       CinemaID INT REFERENCES Cinemas(CinemaID),
       ScheduleTime TIME,
       ScheduleNoOfTickets SMALLINT
       PRIMARY KEY(CinemaID, ScheduleTime)
)

GO
```



b.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace FilmFestivalMasterForm
{
    public partial class Form1 : Form
    {
        SqlConnection connection;
        SqlDataAdapter daProducers, daMovies;
        DataSet ds;
        SqlCommandBuilder cbMovies;
        BindingSource bsProducers, bsMovies;

        private void btnUpdateDB_Click(object sender, EventArgs e)
        {
            daMovies.Update(ds, "Movies");
        }

        public Form1()
        {
            InitializeComponent();
        }

        ~Form1()
        {
            connection.Close();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // instantiate the binding sources
            bsProducers = new BindingSource();
            bsMovies = new BindingSource();

            // bind the data grid views to the corresponding resources
            dgvProducers.DataSource = bsProducers;
            dgvMovies.DataSource = bsMovies;

            // establish the connection to the FilmFestival database
            connection = new SqlConnection(@"Data Source = DESKTOP-JC39FI8\SQLEXPRESS;
Initial Catalog=FilmFestival; Integrated Security=True");

            // instantiate the data set
            ds = new DataSet();

            // instantiate the data adapters and bind them to the data records stored
into the corresponding tables
            daProducers = new SqlDataAdapter("SELECT * FROM Producers", connection);
```

```
            daMovies = new SqlDataAdapter("Select * FROM Movies", connection);

            // instantiate the command builder for the data adapter of the Movies table
            cbMovies = new SqlCommandBuilder(daMovies);

            // fetch the data records from to the source tables into the data adapters
            daProducers.Fill(ds, "Producers");
            daMovies.Fill(ds, "Movies");

            // specify the foreign key relation between the Producers and Movies tables
and add it to the data set
            DataRelation dr = new DataRelation(
                "ProducerMovies",
                ds.Tables["Producers"].Columns["ProducerID"],
                ds.Tables["Movies"].Columns["ProducerID"]
                );
            ds.Relations.Add(dr);


            // set the binding source of the producers to the Producers table
            bsProducers.DataSource = ds;
            bsProducers.DataMember = "Producers";

            // set the binding source of the movies to the binding source of the
producers based on the relation defined above
            bsMovies.DataSource = bsProducers;
            bsMovies.DataMember = "ProducerMovies";
        }
    }
}
```

c.

```
-- TRANSACTION 1

USE FilmFestival
GO


--- READ COMMITTED by default

BEGIN TRAN

UPDATE Movies
SET MovieTitle='T2'
WHERE MovieID=1

COMMIT TRAN


-- TRANSACTION 2

USE FilmFestival
GO
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

BEGIN TRAN

SELECT *
FROM Movies
WHERE MovieID=1

COMMIT TRAN
```

Explanation:

Since T2 runs under READ UNCOMMITTED, it will retrieve the non-updated movie record with id=1 if the T1 didn't get an exclusive lock on it (before entering the update statement).

SOLUTION:

```
-- TRANSACTION 1

USE FilmFestival
GO


--- READ COMMITTED by default

BEGIN TRAN

UPDATE Movies
SET MovieTitle='T2'
WHERE MovieID=1

COMMIT TRAN


-- TRANSACTION 2

USE FilmFestival
GO


SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRAN

SELECT *
FROM Movies
WHERE MovieID=1

COMMIT TRAN
```

Explanation:

Now that the isolation level for T2 is set to READ COMMITTED, it will retrieve the movie record with id=1 after T1 commits.