

Sistema de Cinema

Felipe Luís Pinheiro - 18/0052667

João Pedro C.N. Mota - 17/0106144

Pedro Catelli - 17/0112624

Pedro Oliveira - 17/0163768

3 de julho de 2019

Resumo

Neste relatório desenvolvemos os requisitos básicos de um sistema de banco de dados para um modelo de vendas de ingresso de um cinema.

Link para o repositório: https://github.com/flpinheiro/banco_de_dados.

O projeto do programa que usa esse sistema de banco de dados está no repositório : <https://github.com/flpinheiro/UnBCineFlixMVC>

1 Introdução

Requisitos gerais:

- Um cinema pode ter muitas salas, sendo necessário, por tanto, registrar informações a respeito de cada uma, como sua capacidade, ou seja, o numero de assentos disponíveis.
- O cinema apresenta muitos filmes. Um filme tem informações, titulo e duração. Assim, sempre que um filme for ser apresentado, deve-se registrá-lo também.
- Um mesmo filme pode ser apresentado em diferentes salas e em horários diferentes. Cada apresentação em uma determinada sala e horário é chamada sessão. Um filme sendo apresentado em uma sessão tem um conjunto máximo de ingressos, determinado pela capacidade da sala.
- Os clientes do cinema podem comprar ou não ingressos para assistir a uma sessão. O funcionário deve intermediar a compra do ingresso. Um ingresso deve conter informação como o tipo de ingresso (Meio ingresso ou ingresso inteiro). Além disso, um cliente só pode comprar ingressos para sessões ainda não encerradas.

2 Diagrama de Entidade Relacionamento

Na figura 1 mostramos a primeira versão conceitual do sistema do

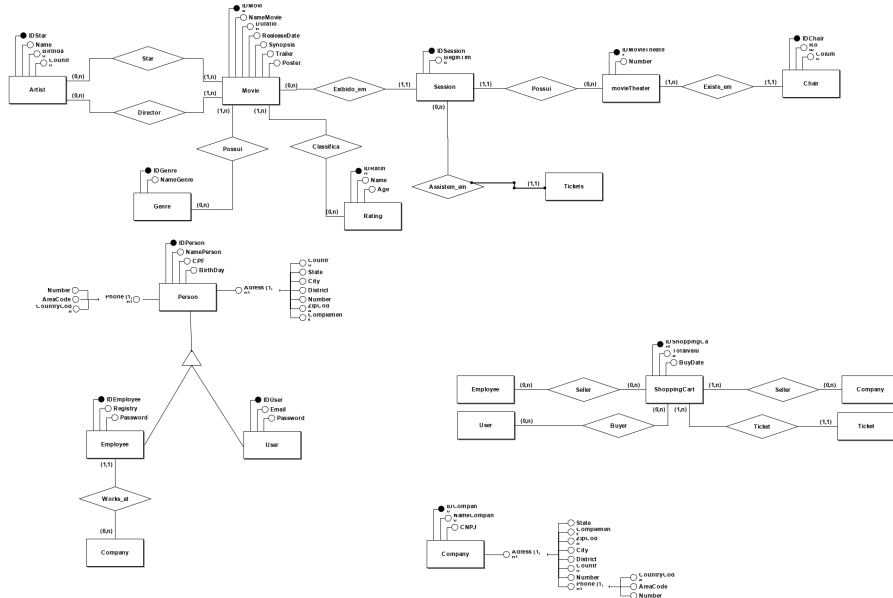


Figura 1: Modelo Entidade Relacionamento

3 Modelo Relacional

Na figura 2 mostramos o modelo relacional utilizado para implementação do programa

4 Consultas

Nesta seção mostramos exemplo de consultas que podem ser realizadas nesse modelo relacional de banco de dados.

```

1 use unbcineflix;
2
3 select * FROM movies, ratings, genremovies, genres where
   ratingid = ratings.id and movies.id = genremovies.
   movieid and genremovies.genreid = genres.id;
4
5 select * from movies, artistmovies, artists where Movies.
   id = artistmovies.MovieId and artistmovies.ArtistId =
   artists.Id;
6
7 select * from movietheaters, addresses, companies where
   addresses.Id = movietheaters.AddressCompanyId and
   addresses.CompanyId = companies.Id and addresses.
   Discriminator = 'AddressCompany';
8
9 select * from session, movietheaters, tickets where
   session.Id = tickets.SessionId and session.
   AddressCompanyId = movietheaters.AddressCompanyId and

```

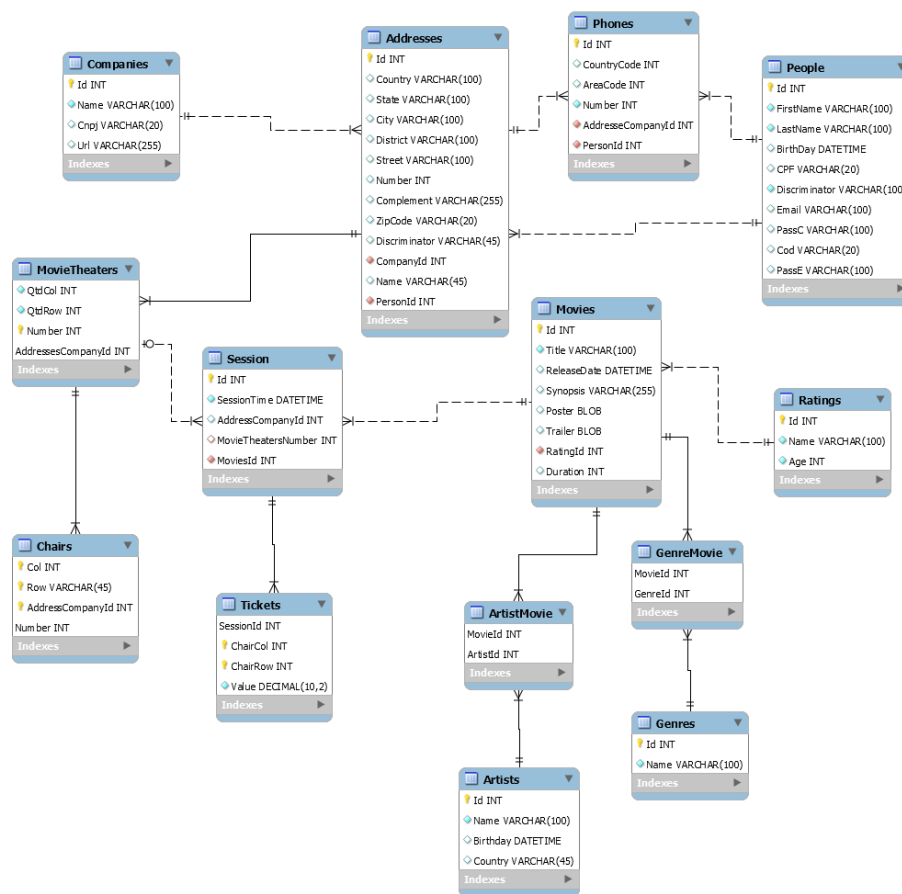


Figura 2: Modelo Relacional

```

    movietheaters.MovieTheaterNumber = session .
    MovieTheaterNumber ;
10
11 select * from people , addresses , phones where people.id =
    addresses . PersonId and people.id = phones . PersonId and
    addresses . Discriminator = 'AddressPerson' ;

```

5 Álgebra Relacional

Nesta seção mostramos as consulta acima realizadas, mas em álgebra relacional.

$$\sigma_{\text{Movies.RatingId} = \text{Ratings.Id} \text{ and } \text{Movies.Id} = \text{GenreMovies.MovieId} \text{ and } \text{genremovies.genreid} = \text{genres.id}} (\text{Movies} \times \text{Ratings} \times \text{GenreMovies} \times \text{Genre})$$

$$\sigma_{\text{Movies.id} = \text{artistmovies.MovieId} \text{ and } \text{artistmovies.ArtistId} = \text{artists.Id}} (\text{movies} \times \text{artistmovies} \times \text{artists})$$

$$\sigma_{\text{addresses.Id} = \text{movietheaters.AddressCompanyId} \text{ and } \text{addresses.CompanyId} = \text{companies.Id} \text{ and } \text{addresses.Discriminator} = \text{'AddressCompany'}} (\text{movietheaters} \times \text{addresses} \times \text{companies})$$

$$\sigma_{\text{session.Id} = \text{tickets.SessionId} \text{ and } \text{session.AddressCompanyId} = \text{movietheaters.AddressCompanyId} \text{ and } \text{movietheaters.MovieTheaterNumber} = \text{session.MovieTheaterNumber}} (\text{session} \times \text{movietheaters} \times \text{tickets})$$

$$\sigma_{\text{people.id} = \text{addresses.PersonId} \text{ and } \text{people.id} = \text{phones.PersonId} \text{ and } \text{addresses.Discriminator} = \text{'AddressPerson'}} (\text{people} \times \text{addresses} \times \text{phones})$$

6 Views

Nesta parte mostramos exemplos da utilização de Views no código do SQL.

```

1 use unbcineflix ;
2
3 drop view addresscompany ;
4
5 drop view AddressPerson ;
6
7 drop view SoldTickets ;

```

	numero sessao	Titulo do filme	sala	dia e hora	numero coluna	numero fileira	valor
▶	1	Rambo	1	2019-06-30 00:00:00	5	1	12.00
	1	Rambo	1	2019-06-30 00:00:00	4	5	10.00

Figura 3: Exemplo de resultado da View SoldTickets

```

8
9 create view AddressCompany as SELECT Country, state, city,
  Street, number, zipcode, name from addresses WHERE
  addresses.Discriminator = 'AddressCompany';
10
11 create view AddressPerson as SELECT Country, state, city,
  Street, number, zipcode from addresses WHERE addresses.
  Discriminator = 'AddressPerson';
12
13 create view SoldTickets as select session.id as 'numero
  sessao', movies.Title as 'Titulo do filme', session.
  MovieTheaterNumber as 'sala', session.SessionTime as '
  dia e hora', ChairCol as 'numero coluna', ChairRow as
  'numero fileira', Value as 'valor' from session,
  movietheaters, tickets, movies where session.Id =
  tickets.SessionId and session.AddressCompanyId =
  movietheaters.AddressCompanyId and movietheaters.
  MovieTheaterNumber = session.MovieTheaterNumber and
  session.MovieId = movies.id;
14
15 select * from addresscompany;
16
17 select * from AddressPerson;
18
19 select * from SoldTickets;

```

Na figura 3 podemos ver um exemplo de resultado mostrado pela viu Sold-Tickets.

7 Script Sql

Nesta seção mostramos o script sql para geração do banco de dados, que foi gerado utilizando o modelo acima e foi gerado automaticamente pelo MySQL.

```

1— MySQL Script generated by MySQL Workbench
2— Thu Jun 27 18:36:45 2019
3— Model: New Model Version: 2.0
4— MySQL Workbench Forward Engineering
5
6SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
7SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
  FOREIGN_KEY_CHECKS=0;
8SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='
  ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,
  NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,
  NO_ENGINE_SUBSTITUTION';
9
10—
11— Schema UnBCineFlix
12—

```

```

13DROP SCHEMA IF EXISTS `UnBCineFlix` ;
14
15— _____
16— Schema UnBCineFlix
17— _____
18CREATE SCHEMA IF NOT EXISTS `UnBCineFlix` DEFAULT
  CHARACTER SET utf8 ;
19USE `UnBCineFlix` ;
20
21— _____
22— Table `UnBCineFlix`.`Addresses`
23— _____
24CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Addresses` (
25  `Id` INT NOT NULL AUTO_INCREMENT,
26  `Country` VARCHAR(100) NULL,
27  `State` VARCHAR(100) NULL,
28  `City` VARCHAR(100) NULL,
29  `District` VARCHAR(100) NULL,
30  `Street` VARCHAR(100) NULL,
31  `Number` INT NULL,
32  `Complement` VARCHAR(255) NULL,
33  `ZipCode` VARCHAR(20) NULL,
34  `Discriminator` VARCHAR(45) NULL,
35  `CompanyId` INT NOT NULL,
36  `Name` VARCHAR(45) NULL,
37  `PersonId` INT NOT NULL,
38  PRIMARY KEY (`Id`),
39  INDEX `fk_Addresses_People1_idx` (`PersonId` ASC)
  VISIBLE,
40  INDEX `fk_Addresses_Companies1_idx` (`CompanyId` ASC)
  VISIBLE,
41  CONSTRAINT `fk_Addresses_People1`
42  FOREIGN KEY (`PersonId`)
43  REFERENCES `UnBCineFlix`.`People` (`Id`)
44  ON DELETE NO ACTION
45  ON UPDATE NO ACTION,
46  CONSTRAINT `fk_Addresses_Companies1`
47  FOREIGN KEY (`CompanyId`)
48  REFERENCES `UnBCineFlix`.`Companies` (`Id`)
49  ON DELETE NO ACTION
50  ON UPDATE NO ACTION)
51ENGINE = InnoDB;
52
53
54— _____
55— Table `UnBCineFlix`.`ArtistMovie`
56— _____
57CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`ArtistMovie` (
58  `MovieId` INT NOT NULL,
59  `ArtistId` INT NOT NULL,
60  PRIMARY KEY (`MovieId`, `ArtistId`),
61  INDEX `fk_Movie_has_Artist_Artist1_idx` (`ArtistId` ASC)
  VISIBLE,
62  INDEX `fk_Movie_has_Artist_Movie1_idx` (`MovieId` ASC)
  VISIBLE,
63  CONSTRAINT `fk_Movie_has_Artist_Movie1`
64  FOREIGN KEY (`MovieId`)
65  REFERENCES `UnBCineFlix`.`Movies` (`Id`)
66  ON DELETE NO ACTION
67  ON UPDATE NO ACTION,
68  CONSTRAINT `fk_Movie_has_Artist_Artist1`
69  FOREIGN KEY (`ArtistId`)

```

```

70 REFERENCES `UnBCineFlix`.`Artists` (`Id`)
71 ON DELETE NO ACTION
72 ON UPDATE NO ACTION)
73ENGINE = InnoDB;
74
75
76—
77— Table `UnBCineFlix`.`Artists`
78—
79CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Artists` (
80 `Id` INT NOT NULL,
81 `Name` VARCHAR(100) NOT NULL,
82 `Birthday` DATETIME NULL,
83 `Country` VARCHAR(45) NULL,
84 PRIMARY KEY (`Id`))
85ENGINE = InnoDB;
86
87
88—
89— Table `UnBCineFlix`.`Chairs`
90—
91CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Chairs` (
92 `Col` INT NOT NULL,
93 `Row` VARCHAR(45) NOT NULL,
94 `AddressCompanyId` INT NOT NULL,
95 `Number` INT NOT NULL,
96 PRIMARY KEY (`Col`, `Row`, `AddressCompanyId`, `Number`)
97 ,
98 INDEX `fk_Chairs_MovieTheaters1_idx` (`AddressCompanyId`
99 ASC, `Number` ASC) VISIBLE,
100 CONSTRAINT `fk_Chairs_MovieTheaters1`
101 FOREIGN KEY (`Number`)
102 REFERENCES `UnBCineFlix`.`MovieTheaters` (`Number`)
103 ON DELETE NO ACTION
104 ON UPDATE NO ACTION)
105ENGINE = InnoDB;
106
107
108—
109— Table `UnBCineFlix`.`Companies`
110—
111CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Companies` (
112 `Id` INT NOT NULL AUTO INCREMENT,
113 `Name` VARCHAR(100) NOT NULL,
114 `Cnpj` VARCHAR(20) NULL,
115 `Url` VARCHAR(255) NULL,
116 PRIMARY KEY (`Id`))
117ENGINE = InnoDB;
118
119
120—
121— Table `UnBCineFlix`.`GenreMovie`
122—
123CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`GenreMovie` (
124 `MovieId` INT NOT NULL,
125 `GenreId` INT ZEROFILL NOT NULL,
126 PRIMARY KEY (`MovieId`, `GenreId`),
127 INDEX `fk_Movie_has_Genre_Genre1_idx` (`GenreId` ASC)
128 VISIBLE,
129 INDEX `fk_Movie_has_Genre_Movie1_idx` (`MovieId` ASC)
130 VISIBLE,
131 CONSTRAINT `fk_Movie_has_Genre_Movie1`

```

```

128 FOREIGN KEY (`MovieId`)
129 REFERENCES `UnBCineFlix`.`Movies` (`Id`)
130 ON DELETE NO ACTION
131 ON UPDATE NO ACTION,
132 CONSTRAINT `fk_Movie_has_Genre_Genre1`
133 FOREIGN KEY (`GenreId`)
134 REFERENCES `UnBCineFlix`.`Genres` (`Id`)
135 ON DELETE NO ACTION
136 ON UPDATE NO ACTION)
137ENGINE = InnoDB;
138
139
140—
141— Table `UnBCineFlix`.`Genres`
142—
143CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Genres` (
144 `Id` INT ZEROFILL NOT NULL,
145 `Name` VARCHAR(100) NOT NULL,
146 PRIMARY KEY (`Id`))
147ENGINE = InnoDB;
148
149
150—
151— Table `UnBCineFlix`.`MovieTheaters`
152—
153CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`MovieTheaters`
154 (
155 `QtdCol` INT NOT NULL,
156 `QtdRow` INT NOT NULL,
157 `Number` INT NOT NULL,
158 `AddressesCompanyId` INT NOT NULL,
159 PRIMARY KEY (`Number`, `AddressesCompanyId`),
160 INDEX `fk_MovieTheaters_Addresses1_idx` (`
161   AddressesCompanyId` ASC) VISIBLE,
162 CONSTRAINT `fk_MovieTheaters_Addresses1`
163 FOREIGN KEY (`AddressesCompanyId`)
164 REFERENCES `UnBCineFlix`.`Addresses` (`Id`)
165 ON DELETE NO ACTION
166 ON UPDATE NO ACTION)
167ENGINE = InnoDB;
168
169—
170— Table `UnBCineFlix`.`Movies`
171—
172CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Movies` (
173 `Id` INT NOT NULL AUTO_INCREMENT,
174 `Title` VARCHAR(100) NOT NULL,
175 `ReleaseDate` DATETIME NULL,
176 `Synopsis` VARCHAR(255) NULL,
177 `Poster` BLOB NULL,
178 `Trailer` BLOB NULL,
179 `RatingId` INT NOT NULL,
180 `Duration` INT NULL,
181 PRIMARY KEY (`Id`),
182 INDEX `fk_Movie_Rating1_idx` (`RatingId` ASC) VISIBLE,
183 CONSTRAINT `fk_Movie_Rating1`
184 FOREIGN KEY (`RatingId`)
185 REFERENCES `UnBCineFlix`.`Ratings` (`Id`)
186 ON DELETE NO ACTION
187 ON UPDATE NO ACTION)
188ENGINE = InnoDB;

```



```

188
189
190— 

---


191— Table `UnBCineFlix`.`People`
192— 

---


193CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`People` (
194 `Id` INT NOT NULL AUTO_INCREMENT,
195 `FirstName` VARCHAR(100) NOT NULL,
196 `LastName` VARCHAR(100) NOT NULL,
197 `BirthDay` DATETIME NULL,
198 `CPF` VARCHAR(20) NULL,
199 `Discriminator` VARCHAR(100) NOT NULL,
200 `Email` VARCHAR(100) NULL,
201 `PassC` VARCHAR(100) NULL,
202 `Cod` VARCHAR(20) NULL,
203 `PassE` VARCHAR(100) NULL,
204 PRIMARY KEY (`Id`))
205ENGINE = InnoDB;
206
207
208— 

---


209— Table `UnBCineFlix`.`Phones`
210— 

---


211CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Phones` (
212 `Id` INT NOT NULL AUTO_INCREMENT,
213 `CountryCode` INT NULL,
214 `AreaCode` INT NULL,
215 `Number` INT NOT NULL,
216 `AdresseCompanyId` INT NOT NULL,
217 `PersonId` INT NOT NULL,
218 PRIMARY KEY (`Id`),
219 INDEX `fk_Phones_Addresses1_idx` (`AdresseCompanyId`
220 ASC) VISIBLE,
221 INDEX `fk_Phones_People1_idx` (`PersonId` ASC) VISIBLE,
222 CONSTRAINT `fk_Phones_Addresses1`
223 FOREIGN KEY (`AdresseCompanyId`)
224 REFERENCES `UnBCineFlix`.`Addresses` (`Id`)
225 ON DELETE NO ACTION
226 ON UPDATE NO ACTION,
227 CONSTRAINT `fk_Phones_People1`
228 FOREIGN KEY (`PersonId`)
229 REFERENCES `UnBCineFlix`.`People` (`Id`)
230 ON DELETE NO ACTION
231 ON UPDATE NO ACTION)
232ENGINE = InnoDB;
233
234— 

---


235— Table `UnBCineFlix`.`Ratings`
236— 

---


237CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Ratings` (
238 `Id` INT NOT NULL AUTO_INCREMENT,
239 `Name` VARCHAR(100) NOT NULL,
240 `Age` INT NOT NULL,
241 PRIMARY KEY (`Id`))
242ENGINE = InnoDB;
243
244
245— 

---


246— Table `UnBCineFlix`.`Session`
247— 

---


248CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Session` (

```

```

249 `Id` INT NOT NULL AUTO_INCREMENT,
250 `SessionTime` DATETIME NOT NULL,
251 `AddressCompanyId` INT NULL,
252 `MovieTheatersNumber` INT NULL,
253 `MoviesId` INT NOT NULL,
254 PRIMARY KEY (`Id`),
255 INDEX `fk_Session_MovieTheaters1_idx` (`AddressCompanyId`
  `ASC`, `MovieTheatersNumber` ASC) VISIBLE,
256 INDEX `fk_Session_Movies1_idx` (`MoviesId` ASC) VISIBLE,
257 CONSTRAINT `fk_Session_MovieTheaters1`
258 FOREIGN KEY (`MovieTheatersNumber`)
259 REFERENCES `UnBCineFlix`.`MovieTheaters` (`Number`)
260 ON DELETE NO ACTION
261 ON UPDATE NO ACTION,
262 CONSTRAINT `fk_Session_Movies1`
263 FOREIGN KEY (`MoviesId`)
264 REFERENCES `UnBCineFlix`.`Movies` (`Id`)
265 ON DELETE NO ACTION
266 ON UPDATE NO ACTION)
267ENGINE = InnoDB;
268
269
270—
271— Table `UnBCineFlix`.`Tickets`
272—
273CREATE TABLE IF NOT EXISTS `UnBCineFlix`.`Tickets` (
274 `SessionId` INT NOT NULL,
275 `ChairCol` INT NOT NULL,
276 `ChairRow` INT NOT NULL,
277 `Value` DECIMAL(10,2) NOT NULL,
278 PRIMARY KEY (`SessionId`, `ChairCol`, `ChairRow`),
279 INDEX `fk_Tickets_Session1_idx` (`SessionId` ASC)
  VISIBLE,
280 CONSTRAINT `fk_Tickets_Session1`
281 FOREIGN KEY (`SessionId`)
282 REFERENCES `UnBCineFlix`.`Session` (`Id`)
283 ON DELETE NO ACTION
284 ON UPDATE NO ACTION)
285ENGINE = InnoDB;
286
287
288SET SQL_MODE=@OLD_SQL_MODE;
289SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
290SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

8 Procedure

Nesta parte mostramos um exemplo simples de uma procedure que pode ser executada neste banco de dados.

9 Camada de Persistência

Para acesso ao banco de Dados foi utilizado o Entity Framework Core versão 2.2.4 e o sistema MySQL como banco de dados de persistência, a seguir

mostramos o código de persistência da aplicação e exemplos do controlador de acesso.

O código a seguir é o código de “Context” do EntityFramework Core o qual foi desenvolvido seguindo os padrão do nomeclatura e de desenvolvimento exigidos pela comunidade, utilizamos esse Framework devido a sua camada de middleware que faz a conversão automática do sistema relacional para a orientação objeto utilizado no programa que foi desenvolvido com C# e ASP.NET Core 2.2 tendo como objetivo final uma aplicação Web que poudesse ser executada por um usuario domestico ou pelos adiministradores do sistema diretamente da empresa, sendo assim uma aplicação completa para uma empresa.

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using UnBCineFlix.Models;

namespace UnBCineFlix.DAL
{
    public class UnBCineFlixContext : DbContext
    {
        public DbSet<Address> Addresses { get; set; }
        public DbSet<AddressCompany> AddressCompanies { get; set; }
        public DbSet<AddressPerson> AddressPeople { get; set; }
        public DbSet<Artist> Artists { get; set; }
        public DbSet<ArtistMovie> ArtistMovies { get; set; }
        public DbSet<Chair> Chairs { get; set; }
        public DbSet<Company> Companies { get; set; }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Employee> Employees { get; set; }
        //errorviemodel
        public DbSet<Genre> Genres { get; set; }
        public DbSet<GenreMovie> GenreMovies { get; set; }
        public DbSet<Movie> Movies { get; set; }
        public DbSet<MovieTheater> MovieTheaters { get; set; }
        public DbSet<Person> People { get; set; }
        public DbSet<Phone> Phones { get; set; }
        public DbSet<Rating> Ratings { get; set; }
        public DbSet<Session> Session { get; set; }
        public DbSet<Ticket> Tickets { get; set; }

        public UnBCineFlixContext()
        {
        }
        public UnBCineFlixContext(DbContextOptions<
            UnBCineFlixContext> option)
            : base(option)
        {
        }
        protected override void OnModelCreating(ModelBuilder
            modelBuilder)
        {
            //Primary Key setup space
            #region pk
            modelBuilder.Entity<Address>().HasKey(a => a.Id);
            modelBuilder.Entity<Person>().HasKey(p => p.Id);
            modelBuilder.Entity<Phone>().HasKey(ph => ph.Id);
            modelBuilder.Entity<Rating>().HasKey(r => r.Id);
            modelBuilder.Entity<Artist>().HasKey(ar => ar.Id);
            modelBuilder.Entity<Movie>().HasKey(m => m.Id);
        }
    }
}
```

```

        modelBuilder.Entity<Company>().HasKey(c => c.Id);
        modelBuilder.Entity<Session>().HasKey(s => s.Id);
        modelBuilder.Entity<ArtistMovie>().HasKey(am => new { am.
MovieId, am.ArtistId });
        modelBuilder.Entity<GenreMovie>().HasKey(gm => new { gm.
GenreId, gm.MovieId });
        modelBuilder.Entity<MovieTheater>().HasKey(mt => new { mt.
AddressCompanyId, mt.MovieTheaterNumber });
        modelBuilder.Entity<Chair>().HasKey(ch => new { ch.
AddressCompanyId, ch.MovieTheaterNumber, ch.Row, ch.Col });
        modelBuilder.Entity<Ticket>().HasKey(t => new { t.
SessionId, t.ChairRow, t.ChairCol });
        #endregion

        //foreign key setup space
        #region fk
        modelBuilder.Entity<AddressPerson>().HasOne(a => a.Person)
        .WithMany(p => p.Addresses).HasForeignKey(a => a.PersonId)
        .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<Phone>().HasOne(ph => ph.Person).
        WithMany(p => p.Phones).HasForeignKey(p => p.PersonId).
        OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<AddressCompany>().HasOne(a => a.
        Company).WithMany(c => c.Addresses).HasForeignKey(ac => ac.
        CompanyId).OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<Phone>().HasOne(ph => ph.
        AddressCompany).WithMany(c => c.Phones).HasForeignKey(p =>
        p.AddressCompanyId).OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<ArtistMovie>().HasOne(am => am.Artist)
        .WithMany(a => a.Movies).HasForeignKey(am => am.ArtistId).
        OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<ArtistMovie>().HasOne(am => am.Movie)
        .WithMany(m => m.Artists).HasForeignKey(am => am.MovieId).
        OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<GenreMovie>().HasOne(gm => gm.Genre).
        WithMany(g => g.GenreMovies).HasForeignKey(gm => gm.GenreId)
        .IsRequired();
        modelBuilder.Entity<GenreMovie>().HasOne(gm => gm.Movie).
        WithMany(m => m.GenreMovies).HasForeignKey(gm => gm.MovieId)
        .IsRequired();

        modelBuilder.Entity<Movie>().HasOne(m => m.Rating).
        WithMany(r => r.Movies).HasForeignKey(m => m.RatingId).
        OnDelete(DeleteBehavior.SetNull);

        modelBuilder.Entity<MovieTheater>().HasOne(mt => mt.
        AddressCompany).WithMany(ac => ac.MovieTheaters).
        HasForeignKey(mt => mt.AddressCompanyId);

        modelBuilder.Entity<Chair>().HasOne(ch => ch.MovieTheater)
        .WithMany(mt => mt.Chairs).HasForeignKey(ch => new { ch.
        AddressCompanyId, ch.MovieTheaterNumber }).IsRequired().
        OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<Session>().HasOne(s => s.MovieTheater)
        .WithMany(mt => mt.Sessions).HasForeignKey(s => new { s.
        AddressCompanyId, s.MovieTheaterNumber });
        modelBuilder.Entity<Session>().HasOne(s => s.Movie).
        WithMany(m => m.Sessions).HasForeignKey(s => s.MovieId);

```

```

        modelBuilder.Entity<Ticket>().HasOne(t => t.Session).
        WithMany(s => s.Tickets).HasForeignKey(t=> t.SessionId).
        IsRequired();
        #endregion

        //Espaco para propriedades
        #region properties
        modelBuilder.Entity<MovieTheater>().Property<int>("QtdRow")
        .IsRequired();
        modelBuilder.Entity<MovieTheater>().Property<int>("QtdCol")
        .IsRequired();
        #endregion

        //Heranca
        #region heritage
        modelBuilder.Entity<Customer>().HasBaseType<Person>();
        modelBuilder.Entity<Employee>().HasBaseType<Person>();

        modelBuilder.Entity<AddressCompany>(ac => { ac.
        HasBaseType<Address>(); });
        modelBuilder.Entity<AddressPerson>(ac => { ac.HasBaseType
        <Address>(); });
        #endregion

        //Seeding the DataBase
        #region seed
        modelBuilder.Entity<Company>().HasData(
            new Company { Id = 1, Name = "Cine Marx" }
        );

        modelBuilder.Entity<AddressCompany>().HasData(
            new AddressCompany { Id = 1, CompanyId = 1, City = "
        brasilia", District = "Asa Sul", Street = "sql", Number =
        42, Complement = null, Country = "Brasil", State = "DF",
        ZipCode = 7000000, Name = "Brasilia Park" }
        );

        modelBuilder.Entity<MovieTheater>().HasData(
            new MovieTheater { qtdCol:10, qtdRow:10 } {
        MovieTheaterNumber = 1, AddressCompanyId = 1 }
        );

        // inicializa as cadeira da sala->todas.
        for (int i = 0; i < 10; i++)
        {
            for (int j = 0; j < 10; j++)
            {
                var c = new Chair(i, j);
                c.AddressCompanyId = 1;
                c.MovieTheaterNumber = 1;
                modelBuilder.Entity<Chair>().HasData(c);
            }
        }
        modelBuilder.Entity<Customer>().HasData(
            new Customer { Id = 1, FirstName = "Dovakin", LastName
        = "Alcantara", BirthDay = new DateTime(1911, 11, 11), CPF =
        "000.000.000-00", Email = "email@email", PassC = "muito
        louco" },
            new Customer { Id = 2, FirstName = "Machado", LastName
        = "de assis", BirthDay = new DateTime(1911, 11, 11), CPF =
        "333.333.333-33", Email = "email@email", PassC = "muito

```

```

louco 2" }
);

modelBuilder.Entity<Employee>().HasData(
    new Employee { Id = 3, FirstName = "Dovakin", LastName
= "Alcantara", BirthDay = new DateTime(1911, 11, 11), CPF =
"000.000.000-00", Cod = 123456, PassE = "12" }
);

modelBuilder.Entity<AddressPerson>().HasData(
    new AddressPerson { Id = 3, City = "brasilia", District
= "Asa Sul", Street = "sql", Number = 42, Complement =
null, Country = "Brasil", State = "DF", ZipCode = 7000000,
PersonId = 1 },
    new AddressPerson { Id = 2, City = "brasilia", District
= "Asa norte", Street = "Campus Darcy Ribeiro", Number =
0, Complement = "ICC Norte", Country = "Brasil", State = "
DF", ZipCode = 70000000, PersonId = 2 }
);

modelBuilder.Entity<Phone>().HasData(
    new Phone { Id = 1, CountryCode = 55, AreaCode = 61,
Number = 55551234, PersonId = 1 },
    new Phone { Id = 2, CountryCode = 55, AreaCode = 61,
Number = 999954321, AddressCompanyId = 1 },
    new Phone { Id = 3, CountryCode = 55, AreaCode = 61,
Number = 999912345, PersonId = 2 }
);

modelBuilder.Entity<Rating>().HasData(
    new Rating { Id = 1, Name = "Livre", Age = 0 },
    new Rating { Id = 2, Name = "NR 10", Age = 10 },
    new Rating { Id = 3, Name = "NR 12", Age = 12 },
    new Rating { Id = 4, Name = "NR 14", Age = 14 },
    new Rating { Id = 5, Name = "NR 16", Age = 16 },
    new Rating { Id = 6, Name = "NR 18", Age = 18 }
);

modelBuilder.Entity<Artist>().HasData(
    new Artist { Id = 1, Name = "Silvester Stallone",
Country = "USA", BirthDay = new DateTime(1946, 6, 6) },
    new Artist { Id = 2, Name = "Arnold Schwarzenegger",
Country = "Autria", BirthDay = new DateTime(1947, 6, 30) }
);

modelBuilder.Entity<Movie>().HasData(
    new Movie { Id = 1, Title = "Rambo 3", Duration = 180,
ReleaseDate = new DateTime(2000, 12, 25), RatingId = 6 },
    new Movie { Id = 2, Title = "Rambo 2", Duration = 200,
ReleaseDate = new DateTime(1990, 12, 25), RatingId = 6 },
    new Movie { Id = 3, Title = "Rambo ", Duration = 160,
ReleaseDate = new DateTime(1985, 12, 25) }
);

modelBuilder.Entity<ArtistMovie>().HasData(
    new ArtistMovie { MovieId = 1, ArtistId = 1 },
    new ArtistMovie { MovieId = 2, ArtistId = 1 },
    new ArtistMovie { MovieId = 3, ArtistId = 1 },
    new ArtistMovie { MovieId = 1, ArtistId = 2 }
);

modelBuilder.Entity<Genre>().HasData(
    new Genre { Id = 1, Name = "Action" },

```

```

        new Genre { Id = 2, Name = "comedy" }
    );
    modelBuilder.Entity<GenreMovie>().HasData(
        new GenreMovie { MovieId = 1, GenreId = 1 },
        new GenreMovie { MovieId = 2, GenreId = 1 },
        new GenreMovie { MovieId = 3, GenreId = 1 }
    );
    modelBuilder.Entity<Session>().HasData(
        new Session { AddressCompanyId = 1, SessionTime =
DateTime.Today.AddDays(3), MovieId = 3, MovieTheaterNumber
= 1, Id = 1 }
    );

    modelBuilder.Entity<Ticket>().HasData(
        new Ticket { SessionId = 1, ChairCol = 4, ChairRow = 5,
Value = 10 }
    );
    #endregion
}
protected override void OnConfiguring(
DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseMySQL("Server=localhost;DataBase=
unbcineflix;Uid=root;Pwd=@VTQpZGC8*qkj\\$uu");
    }
}
}
}

```

A seguir mostramos alguns exemplos de código de acesso ao banco de dados leitura e escrita usando o Entity Framework e explicamos como ele funciona.

```

var session = await _context.Session
    .Include(s => s.Tickets)
    .Include(s => s.Movie)
    .Include(s => s.MovieTheater)
    .ThenInclude(mt => mt.Chairs)
    .Include(s => s.MovieTheater)
    .ThenInclude(mt => mt.AddressCompany)
    .ThenInclude(ac => ac.Company)
    .FirstOrDefaultAsync(m => m.Id == id);

```

Acima mostramos o processo de leitura de uma Session no Banco de Dados, no qual é realizado um Join com os objetos/tabelas Tickets, Movie, MovieTheater, Chairs, AddressCompany, Company, pois nesse caso em especial queríamos mostrar que uma determinada sessão *i* seria exibida em um determinado dia, em um determinado local, por uma determinada empresa, além de precisarmos saber quais cadeiras existem dentro da sala na qual a sessão será exibida e quais ingressos já foram vendidos.

```

var ticket = await _context.Tickets
    .FirstOrDefaultAsync(t=>
        (t.SessionId == sessionId &&
         t.ChairRow == chairRow &&
         t.ChairCol == chairCol));

```

Neste caso é uma busca bem mais simples, simplesmente queremos saber se o Ticket de uma dada Session, com uma determinada cadeira coluna (ChairCol) e Fileira (ChairRow) existe, ou seja, foi vendido.

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Name,BirthDay,
,Country")] Artist artist)
{
    if (ModelState.IsValid)
    {
        _context.Add(artist);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(artist);
}

```

Acima mostramos o método completo da camada de persistência, controlador, que é usado para adicionar um novo objeto artista dentro do banco de dados relacional, pela simplicidade proporcionada pelo framework utilizado acreditamos ser desnecessário separar a camada de persistência do controlador, apesar que seria especialmente útil se desejarmos

10 Avaliação das Formas Normais