VERWALTUNG FÜR KINO

Modul 153 Datenbank Projekt

Daniel Santos Sousa & Alex Kina

BBZW Sursee

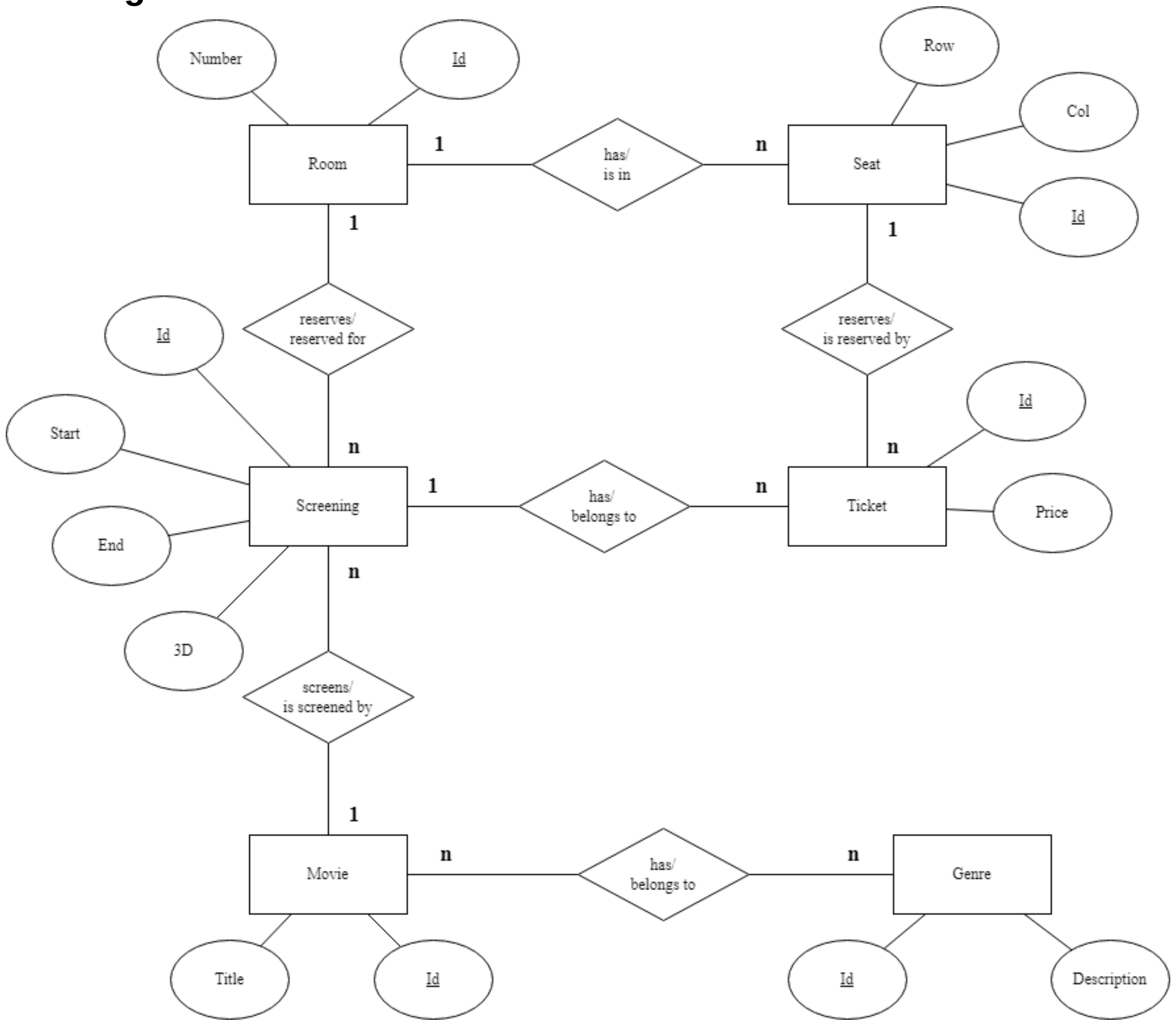
Inhalt

Inhalt

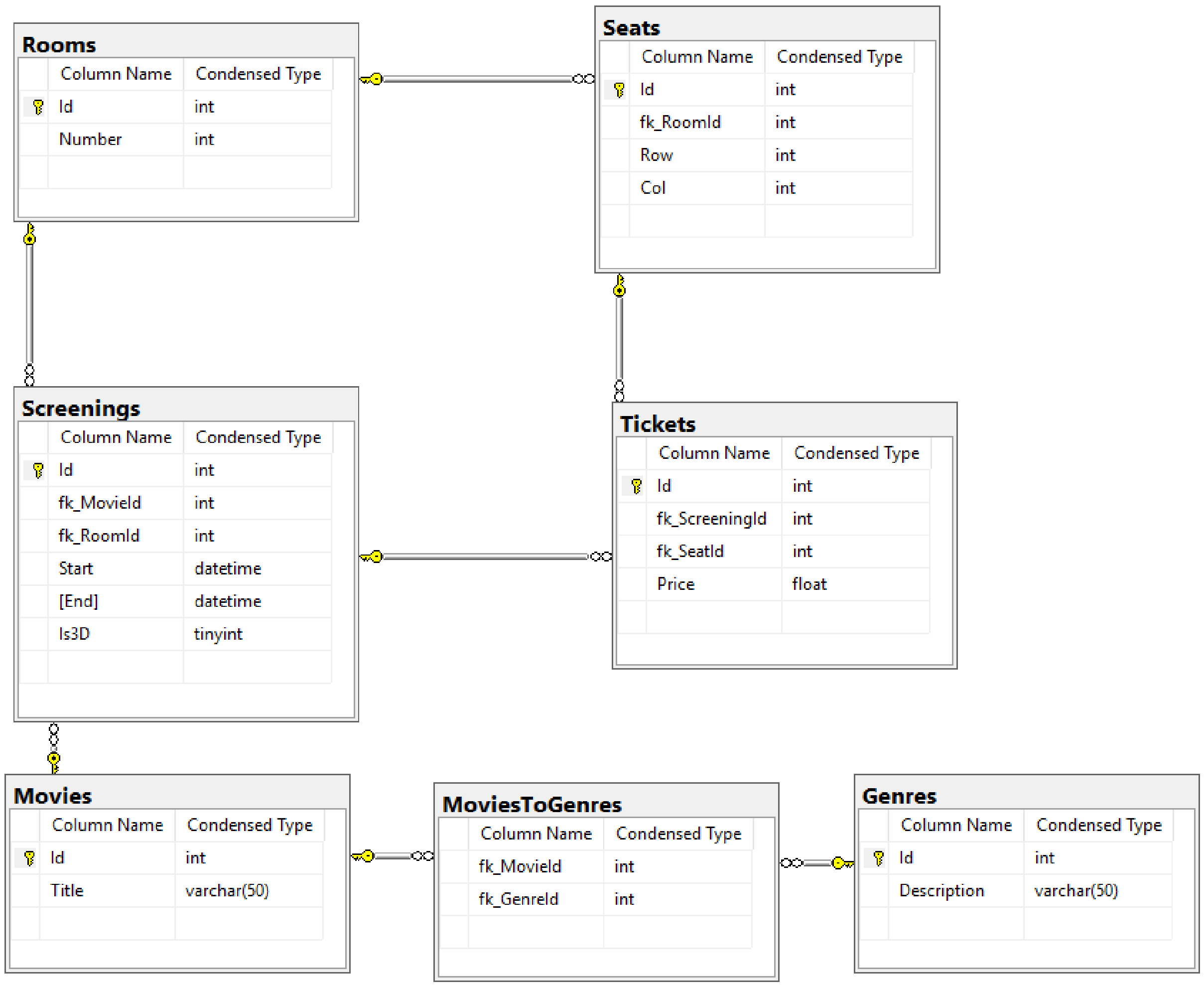
Kurzbeschreibung

Unser Projekt handelt sich um eine Datenbank, mit der man ein Kino verwalten kann.
Mann hat Räume und Filme, welche man in einer Filmvorführung abspielt.

ER-Diagramm



Relationales Model



Rooms

Column Name	Condensed Type
Id	int
Number	int

Seats

Column Name	Condensed Type
Id	int
fk_RoomId	int
Row	int
Col	int

Screenings

Column Name	Condensed Type
Id	int
fk_MovieId	int
fk_RoomId	int
Start	datetime
[End]	datetime
Is3D	tinyint

Tickets

Column Name	Condensed Type
Id	int
fk_ScreeningId	int
fk_SeatId	int
Price	float

Movies

Column Name	Condensed Type
Id	int
Title	varchar(50)

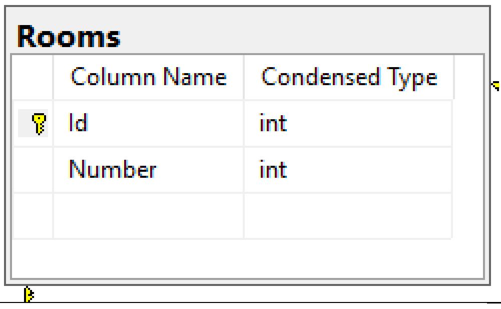
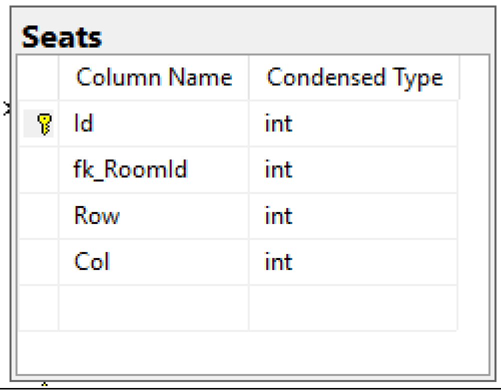
MoviesToGenres

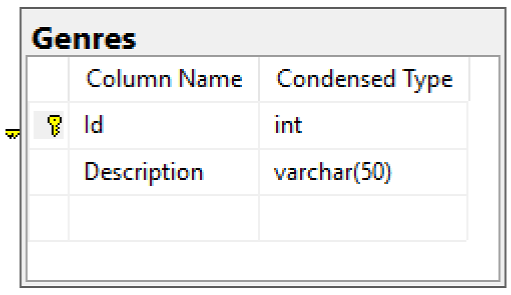
Column Name	Condensed Type
fk_MovieId	int
fk_GenreId	int

Genres

Column Name	Condensed Type
Id	int
Description	varchar(50)

Beschreibung

 **Rooms** — Column Name / Condensed Type: Id int; Number int	**Zimmer** **Number:** Kennzeichen des Zimmers
 **Seats** — Column Name / Condensed Type: Id int; fk_RoomId int; Row int; Col int	**Sitzplätze** **fk_RoomId:** ID vom zugehörigen Raum **Row:** Reihe in der sich der Sitz befindet **Col:** Spalte in der sich der Sitz befindet

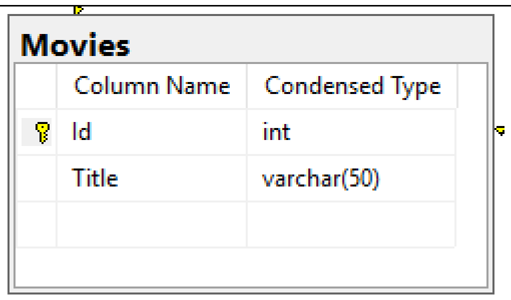


Genres

	Column Name	Condensed Type
🔑	Id	int
	Description	varchar(50)

<u>Genres</u>

Description: Beschreibung der Genre
(Kann ein Satz oder auch ein einzelnes Wort sein. z.B.: «Fantasy» und «Wiedergeboren in einer neuen Welt»)

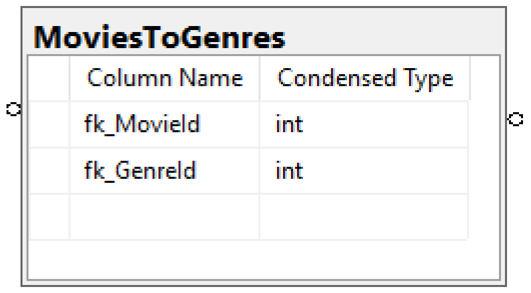


Movies

	Column Name	Condensed Type
🔑	Id	int
	Title	varchar(50)

<u>Filme</u>

Title: Titel des Filmes

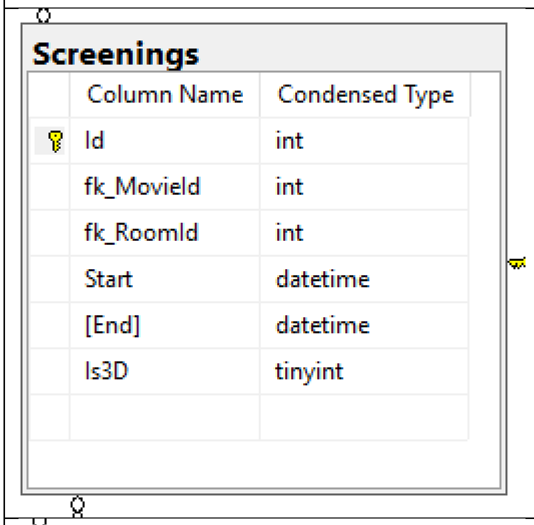


MoviesToGenres

	Column Name	Condensed Type
	fk_MovieId	int
	fk_GenreId	int

<u>Filme zu Genres</u>

fk_MovieId: ID vom Film
fk_GenreId: ID von der zum Film gehörende Genre

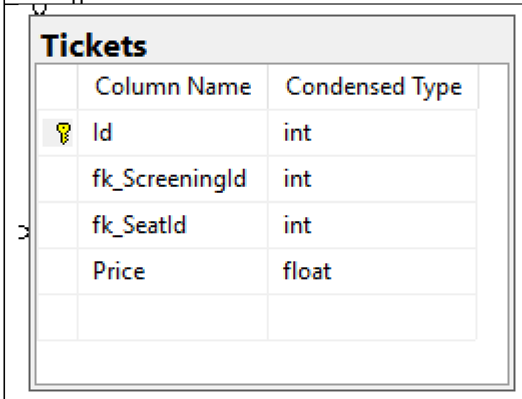


Screenings

	Column Name	Condensed Type
🔑	Id	int
	fk_MovieId	int
	fk_RoomId	int
	Start	datetime
	[End]	datetime
	Is3D	tinyint

<u>Filmvorführungen</u>

fk_MovieId: ID vom abgespielten Film
fk_RoomId: ID vom im abgespielten Raum
Start: Start von der Vorführung
End: Ende der Vorführung
Is3D: Ob es 3D ist (1 = ja, 0 = nein)



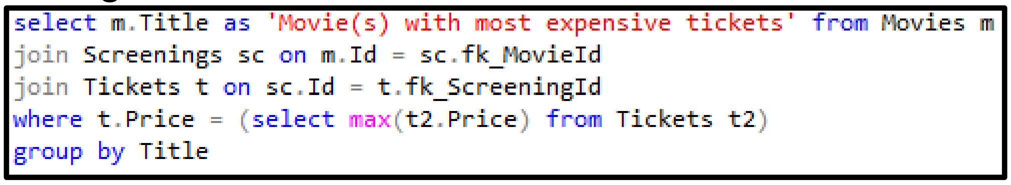
Tickets

	Column Name	Condensed Type
🔑	Id	int
	fk_ScreeningId	int
	fk_SeatId	int
	Price	float

<u>Eintrittskarten</u>

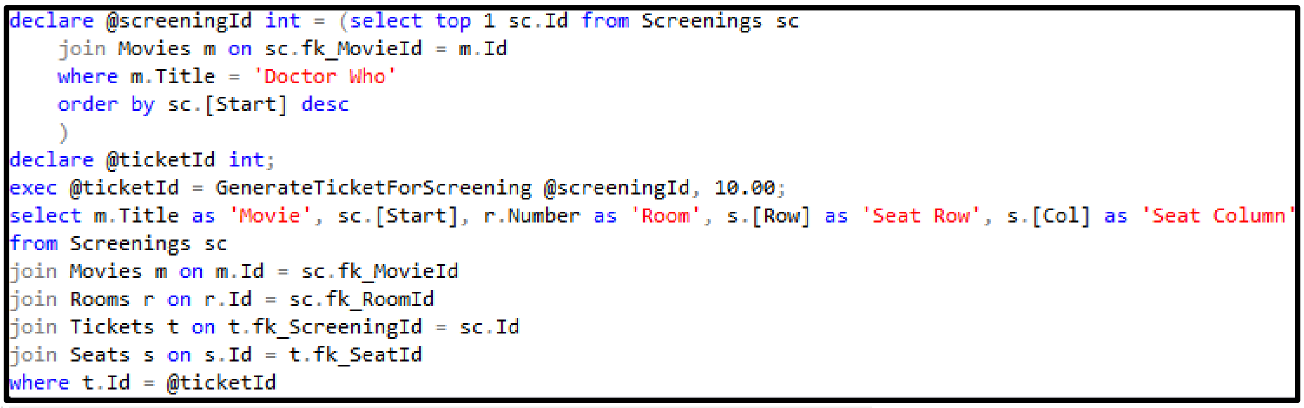
fk_ScreeningId: Vorführung zu der es gehört
fk_SeatId: Sitzplatz der zum Ticket gehört
Price: Preis des Tickets

Abfragen

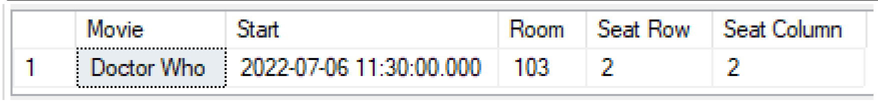


```sql
select m.Title as 'Movie(s) with most expensive tickets' from Movies m
join Screenings sc on m.Id = sc.fk_MovieId
join Tickets t on sc.Id = t.fk_ScreeningId
where t.Price = (select max(t2.Price) from Tickets t2)
group by Title
```

Hier suchen wir den Film aus, der die teuerste Tickets hat, es kann auch mehrere Filme darstellen, falls mehrere gleich Teure Tickets haben.

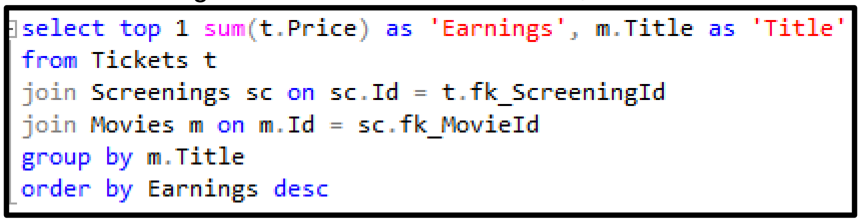


```sql
declare @screeningId int = (select top 1 sc.Id from Screenings sc
    join Movies m on sc.fk_MovieId = m.Id
    where m.Title = 'Doctor Who'
    order by sc.[Start] desc
    )
declare @ticketId int;
exec @ticketId = GenerateTicketForScreening @screeningId, 10.00;
select m.Title as 'Movie', sc.[Start], r.Number as 'Room', s.[Row] as 'Seat Row', s.[Col] as 'Seat Column'
from Screenings sc
join Movies m on m.Id = sc.fk_MovieId
join Rooms r on r.Id = sc.fk_RoomId
join Tickets t on t.fk_ScreeningId = sc.Id
join Seats s on s.Id = t.fk_SeatId
where t.Id = @ticketId
```

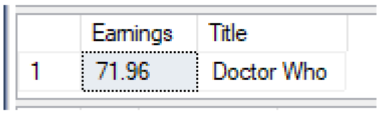


	Movie	Start	Room	Seat Row	Seat Column
1	Doctor Who	2022-07-06 11:30:00.000	103	2	2

Hier suchen wir als erstes die neueste Ausführung vom Film «Doctor Who» dann erstelle ich dafür ein Ticket und gebe die Daten formatiert aus, mit dem Platz zusammen.

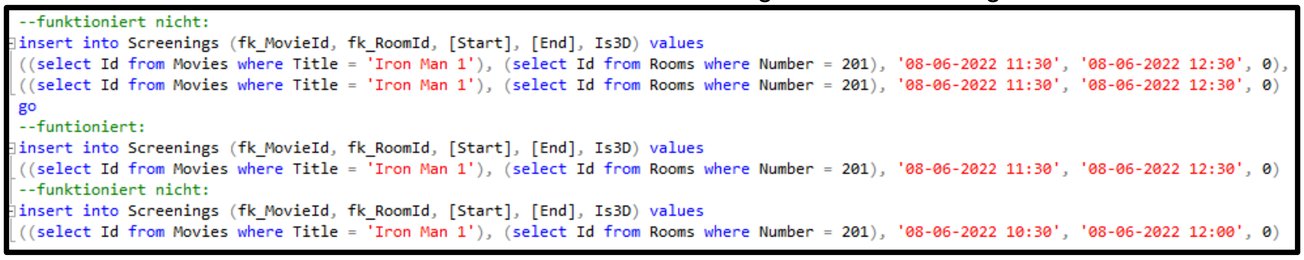


```sql
select top 1 sum(t.Price) as 'Earnings', m.Title as 'Title'
from Tickets t
join Screenings sc on sc.Id = t.fk_ScreeningId
join Movies m on m.Id = sc.fk_MovieId
group by m.Title
order by Earnings desc
```



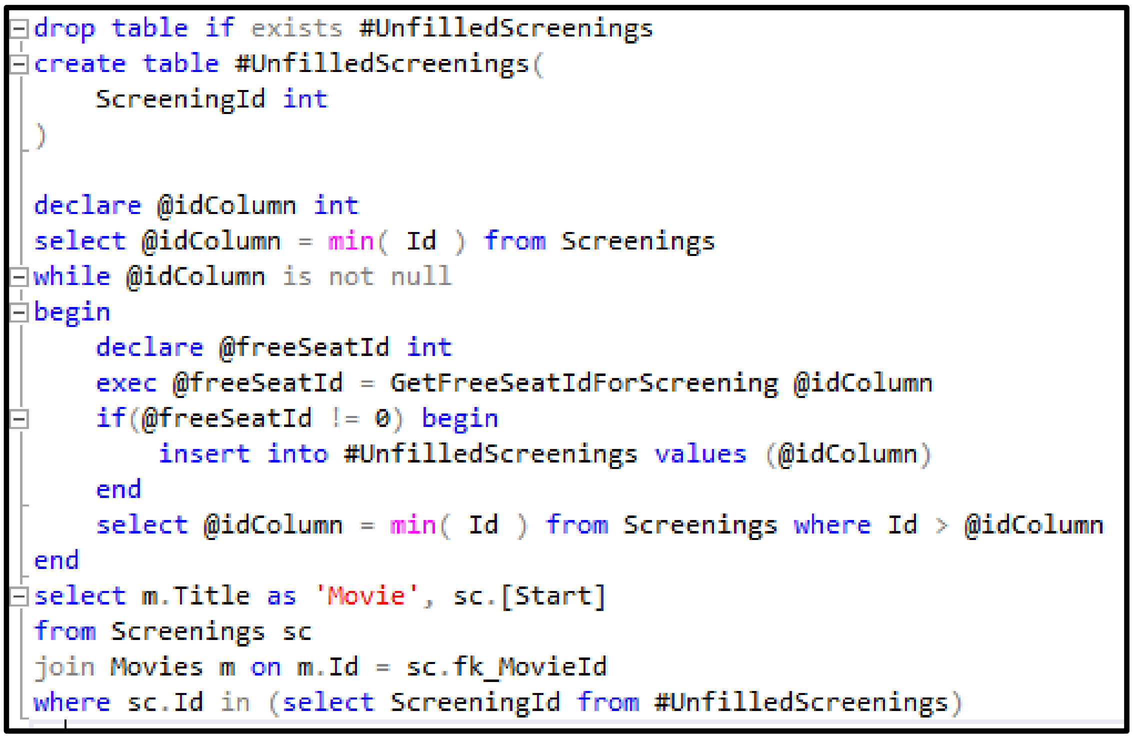
	Earnings	Title
1	71.96	Doctor Who

Hier suche ich den Film aus, welcher am Meisten Umsatz gebracht hat und gebe es dann aus.



```sql
--funktioniert nicht:
insert into Screenings (fk_MovieId, fk_RoomId, [Start], [End], Is3D) values
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201), '08-06-2022 11:30', '08-06-2022 12:30', 0),
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201), '08-06-2022 11:30', '08-06-2022 12:30', 0)
go
--funtioniert:
insert into Screenings (fk_MovieId, fk_RoomId, [Start], [End], Is3D) values
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201), '08-06-2022 11:30', '08-06-2022 12:30', 0)
--funktioniert nicht:
insert into Screenings (fk_MovieId, fk_RoomId, [Start], [End], Is3D) values
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201), '08-06-2022 10:30', '08-06-2022 12:00', 0)
```

Hier fügen wir Vorführungen hinzu, weil am Anfang beide in einem Befehl sind, und die Zeiten ein Konflikt haben wird es abgebrochen und nichts erstellt. Im zweiten teil wird es in zwei Befehle getrennt, deshalb geht der erste und nur das Zweite wird abgebrochen.

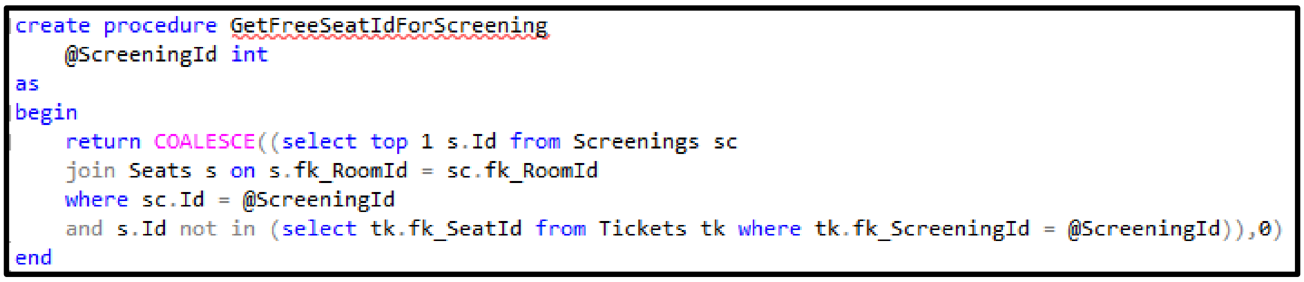


```
drop table if exists #UnfilledScreenings
create table #UnfilledScreenings(
    ScreeningId int
)

declare @idColumn int
select @idColumn = min( Id ) from Screenings
while @idColumn is not null
begin
    declare @freeSeatId int
    exec @freeSeatId = GetFreeSeatIdForScreening @idColumn
    if(@freeSeatId != 0) begin
        insert into #UnfilledScreenings values (@idColumn)
    end
    select @idColumn = min( Id ) from Screenings where Id > @idColumn
end
select m.Title as 'Movie', sc.[Start]
from Screenings sc
join Movies m on m.Id = sc.fk_MovieId
where sc.Id in (select ScreeningId from #UnfilledScreenings)
```

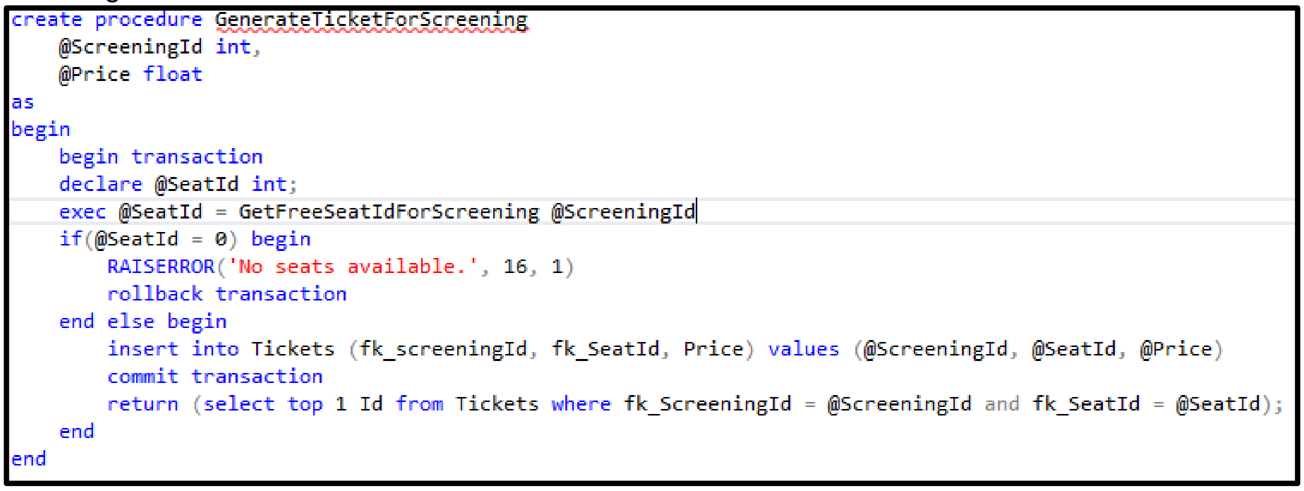
Hier wollten wir bestimmen welche Vorführungen freie Plätze haben, jedoch wollten wir das anhand der stored Procedure machen, deswegen habe ich einen loop erstellt, der alle Zeilen der Vorführungen durchgeht und die Vorführungen, welche noch Plätze haben, habe ich in einer Zwischen Tabelle gespeichert. Am Ende gebe ich es dann noch aus.

Prozeduren



```sql
create procedure GetFreeSeatIdForScreening
    @ScreeningId int
as
begin
    return COALESCE((select top 1 s.Id from Screenings sc
    join Seats s on s.fk_RoomId = sc.fk_RoomId
    where sc.Id = @ScreeningId
    and s.Id not in (select tk.fk_SeatId from Tickets tk where tk.fk_ScreeningId = @ScreeningId)),0)
end
```

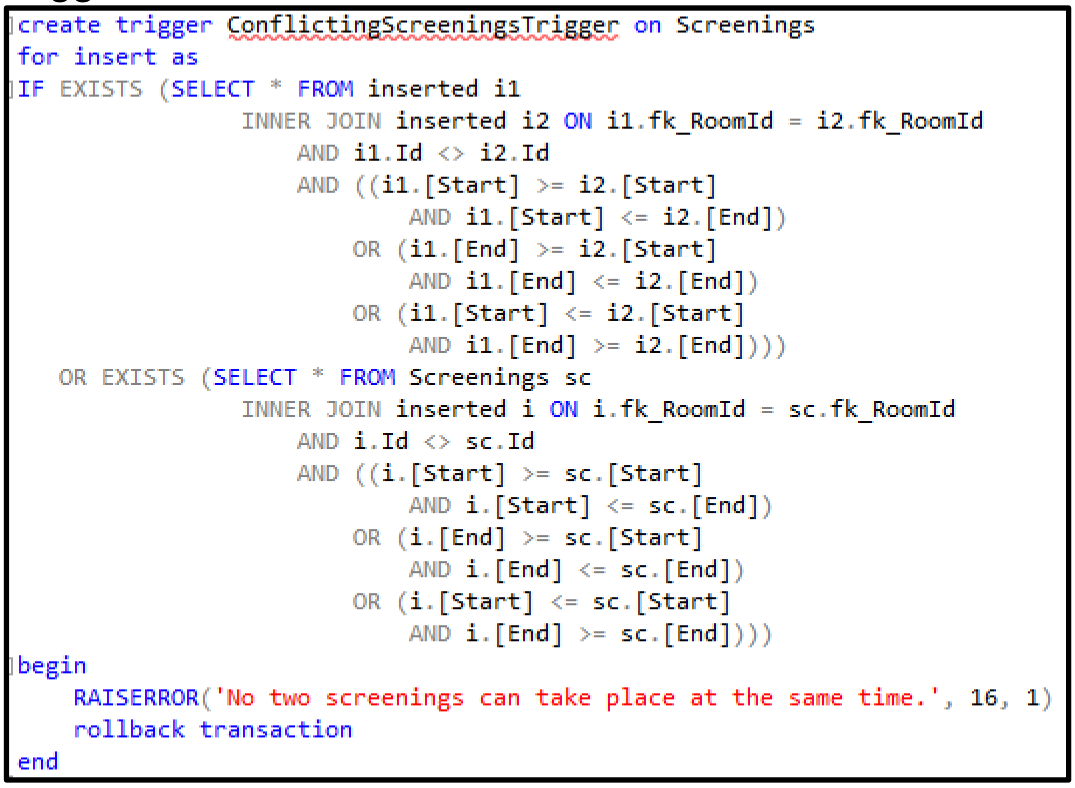
In der ersten Prozedur geben wir die ID von einem freien Platz in einer Vorführung zurück. Parameter ist nur die VorführungsID. Als Rückgabe wird noch gesichert, dass es nicht null, sondern 0 ist, ist eigentlich nicht nötig, weil es automatisch passiert, jedoch kriegt man so keine Warnungen.



```sql
create procedure GenerateTicketForScreening
    @ScreeningId int,
    @Price float
as
begin
    begin transaction
    declare @SeatId int;
    exec @SeatId = GetFreeSeatIdForScreening @ScreeningId
    if(@SeatId = 0) begin
        RAISERROR('No seats available.', 16, 1)
        rollback transaction
    end else begin
        insert into Tickets (fk_screeningId, fk_SeatId, Price) values (@ScreeningId, @SeatId, @Price)
        commit transaction
        return (select top 1 Id from Tickets where fk_ScreeningId = @ScreeningId and fk_SeatId = @SeatId);
    end
end
```

Passend dazu haben wir dann noch eine Prozedur, welche einen Ticket anhand von dem freien Platz generiert, hier nutzt es die vorherige Prozedur, um ein Platz zu finden. Ist kein Platz frei, dann wird es abgebrochen und ein Fehler ausgegeben. Als Parameter ist zusätzlich noch der Preis.

Trigger



```
create trigger ConflictingScreeningsTrigger on Screenings
for insert as
IF EXISTS (SELECT * FROM inserted i1
                INNER JOIN inserted i2 ON i1.fk_RoomId = i2.fk_RoomId
                    AND i1.Id <> i2.Id
                    AND ((i1.[Start] >= i2.[Start]
                            AND i1.[Start] <= i2.[End])
                        OR (i1.[End] >= i2.[Start]
                            AND i1.[End] <= i2.[End])
                        OR (i1.[Start] <= i2.[Start]
                            AND i1.[End] >= i2.[End])))
   OR EXISTS (SELECT * FROM Screenings sc
                INNER JOIN inserted i ON i.fk_RoomId = sc.fk_RoomId
                    AND i.Id <> sc.Id
                    AND ((i.[Start] >= sc.[Start]
                            AND i.[Start] <= sc.[End])
                        OR (i.[End] >= sc.[Start]
                            AND i.[End] <= sc.[End])
                        OR (i.[Start] <= sc.[Start]
                            AND i.[End] >= sc.[End])))
begin
    RAISERROR('No two screenings can take place at the same time.', 16, 1)
    rollback transaction
end
```

Als Trigger haben wir sichergestellt, dass nie zwei Vorführungen zur selben Zeit im selben Raum durchlaufen. Es wird geprüft, ob die andere Vorführung während einer Vorführung endet oder beginnt und auch ob die andere Vorführung diese Umfasst.
Das IF stellt sicher, dass im insert Befehl selbst keine Konflikte gibt, so wie auch Konflikte mit bestehende Vorführungen. Gibt es Konflikte, dann wird das insert abgebrochen und eine Fehlermeldung ausgegeben.

Anhang

Create.sql teil 1

```
use master;
drop database if exists CinemaManagement;
GO
create database CinemaManagement;
GO
use CinemaManagement;

create table Genres(
	Id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	[Description] varchar(50)
);

create table Movies(
	Id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	Title varchar(50)
);

create table MoviesToGenres(
	fk_MovieId int FOREIGN KEY REFERENCES Movies(Id),
	fk_GenreId int FOREIGN KEY REFERENCES Genres(Id)
)

create table Rooms(
	Id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	Number int
);

create table Screenings(
	Id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	fk_MovieId int FOREIGN KEY REFERENCES Movies(Id),
	fk_RoomId int FOREIGN KEY REFERENCES Rooms(Id),
	[Start] datetime,
	[End] datetime,
	Is3D tinyint
);

create table Seats(
	Id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	fk_RoomId int FOREIGN KEY REFERENCES Rooms(Id),
	[Row] int,
	[Col] int
);

create table Tickets(
	Id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	fk_ScreeningId int FOREIGN KEY REFERENCES Screenings(Id),
	fk_SeatId int FOREIGN KEY REFERENCES Seats(Id),
	Price float
);
```

Create.sql teil 2

```
GO
create procedure GetFreeSeatIdForScreening
    @ScreeningId int
as
begin
       return COALESCE((select top 1 s.Id from Screenings sc
       join Seats s on s.fk_RoomId = sc.fk_RoomId
       where sc.Id = @ScreeningId
       and s.Id not in (select tk.fk_SeatId from Tickets tk where tk.fk_ScreeningId =
@ScreeningId)),0)
end

GO
create procedure GenerateTicketForScreening
    @ScreeningId int,
       @Price float
as
begin
       begin transaction
       declare @SeatId int;
       exec @SeatId = GetFreeSeatIdForScreening @ScreeningId
       if(@SeatId = 0) begin
              RAISERROR('No seats available.', 16, 1)
              rollback transaction
       end else begin
              insert into Tickets (fk_screeningId, fk_SeatId, Price) values (@ScreeningId, @SeatId,
@Price)
              commit transaction
              return (select top 1 Id from Tickets where fk_ScreeningId = @ScreeningId and
fk_SeatId = @SeatId);
       end
end

GO
create trigger ConflictingScreeningsTrigger on Screenings
for insert as
IF EXISTS (SELECT * FROM inserted i1
                           INNER JOIN inserted i2 ON i1.fk_RoomId = i2.fk_RoomId
                                  AND i1.Id <> i2.Id
                    AND ((i1.[Start] >= i2.[Start]
                                         AND i1.[Start] <= i2.[End])
                                  OR (i1.[End] >= i2.[Start]
                                         AND i1.[End] <= i2.[End])
                                  OR (i1.[Start] <= i2.[Start]
                                         AND i1.[End] >= i2.[End])))
   OR EXISTS (SELECT * FROM Screenings sc
                           INNER JOIN inserted i ON i.fk_RoomId = sc.fk_RoomId
                                  AND i.Id <> sc.Id
                    AND ((i.[Start] >= sc.[Start]
                                         AND i.[Start] <= sc.[End])
                                  OR (i.[End] >= sc.[Start]
                                         AND i.[End] <= sc.[End])
                                  OR (i.[Start] <= sc.[Start]
                                         AND i.[End] >= sc.[End])))
begin
       RAISERROR('No two screenings can take place at the same time.', 16, 1)
       rollback transaction
end
```

Insert.sql teil 1

```sql
use CinemaManagement;
go
insert into Genres ([Description]) values ('Action') , ('Comedy'), ('Romance'), ('Fantasy'),
('Sci-Fi');
go
insert into Movies (Title) values ('Iron Man 1') , ('Avengers'), ('Star Wars'), ('Doctor Who'),
('I am a spider so what?'), ('I Was the Seventh Prince When I Was Reincarnated');
go
insert into MoviesToGenres (fk_GenreId, fk_MovieId) values
	((select Id from Genres where [Description] = 'Action'), (select Id from Movies where Title
= 'Iron Man 1')),
	((select Id from Genres where [Description] = 'Action'), (select Id from Movies where Title
= 'Avengers')),
	((select Id from Genres where [Description] = 'Comedy'), (select Id from Movies where Title
= 'Star Wars')),
	((select Id from Genres where [Description] = 'Sci-Fi'), (select Id from Movies where Title
= 'Star Wars')),
	((select Id from Genres where [Description] = 'Action'), (select Id from Movies where Title
= 'Doctor Who')),
	((select Id from Genres where [Description] = 'Sci-Fi'), (select Id from Movies where Title
= 'Doctor Who')),
	((select Id from Genres where [Description] = 'Fantasy'), (select Id from Movies where
Title = 'I am a spider so what?')),
	((select Id from Genres where [Description] = 'Romance'), (select Id from Movies where
Title = 'I Was the Seventh Prince When I Was Reincarnated')),
	((select Id from Genres where [Description] = 'Fantasy'), (select Id from Movies where
Title = 'I Was the Seventh Prince When I Was Reincarnated'));
go
insert into Rooms (Number) values (101), (102), (103), (104), (201), (202), (203), (204);
go

declare @room int = 101;
while @room < 205
begin
insert into Seats (fk_RoomId, Col, [Row]) values
	((select Id from Rooms where Number = @room), 1, 1),
	((select Id from Rooms where Number = @room), 2, 1),
	((select Id from Rooms where Number = @room), 3, 1),
	((select Id from Rooms where Number = @room), 1, 2),
	((select Id from Rooms where Number = @room), 2, 2),
	((select Id from Rooms where Number = @room), 3, 2)

if(@room <> 104)
	set @room = @room + 1
else
	set @room = 201
end
go

insert into Screenings (fk_MovieId, fk_RoomId, [Start], [End], Is3D) values
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 101),
'08-06-2022 11:30', '08-06-2022 12:30', 0),
((select Id from Movies where Title = 'Avengers'), (select Id from Rooms where Number = 101), '08-
06-2022 13:45', '08-06-2022 15:00', 0),
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 102),
'08-06-2022 11:30', '08-06-2022 12:30', 1),
((select Id from Movies where Title = 'Doctor Who'), (select Id from Rooms where Number = 103),
'07-06-2022 11:30', '07-06-2022 12:30', 0),
((select Id from Movies where Title = 'I am a spider so what?'), (select Id from Rooms where
Number = 103), '08-06-2022 13:30', '08-06-2022 15:30', 0),
((select Id from Movies where Title = 'Star Wars'), (select Id from Rooms where Number = 102),
'08-06-2022 12:45', '08-06-2022 14:00', 1),
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 103),
'08-06-2022 09:30', '08-06-2022 10:30', 0),
((select Id from Movies where Title = 'I Was the Seventh Prince When I Was Reincarnated'), (select
Id from Rooms where Number = 104), '08-06-2022 11:30', '08-06-2022 12:30', 0),
((select Id from Movies where Title = 'Star Wars'), (select Id from Rooms where Number = 101),
'08-06-2022 10:30', '08-06-2022 11:15', 0)
```

Insert.sql teil 2

```sql
go

declare @screeningsId int
set @screeningsId = (select Id from Screenings
       where fk_MovieId = (select Id from Movies where Title = 'Iron Man 1')
       and fk_RoomId = (select Id from Rooms where Number = 101)
       and [Start] = '08-06-2022 11:30')
exec GenerateTicketForScreening @ScreeningsId,  10.50;
exec GenerateTicketForScreening @ScreeningsId,  10.50;
exec GenerateTicketForScreening @ScreeningsId,  10.50;
exec GenerateTicketForScreening @ScreeningsId,  10.50;
exec GenerateTicketForScreening @ScreeningsId,  10.50;
exec GenerateTicketForScreening @ScreeningsId,  10.50;

set @screeningsId = (select Id from Screenings
       where fk_MovieId = (select Id from Movies where Title = 'Doctor Who')
       and fk_RoomId = (select Id from Rooms where Number = 103)
       and [Start] = '07-06-2022 11:30')
exec GenerateTicketForScreening @ScreeningsId,  15.49;
exec GenerateTicketForScreening @ScreeningsId,  15.49;
exec GenerateTicketForScreening @ScreeningsId,  15.49;
exec GenerateTicketForScreening @ScreeningsId,  15.49;

set @screeningsId = (select Id from Screenings
       where fk_MovieId = (select Id from Movies where Title = 'Avengers')
       and fk_RoomId = (select Id from Rooms where Number = 101)
       and [Start] = '08-06-2022 13:45')
exec GenerateTicketForScreening @ScreeningsId,  15.49;
exec GenerateTicketForScreening @ScreeningsId,  15.49;
go
use master;
```

Query.sql Teil 1

```sql
use CinemaManagement;
go
select m.Title as 'Movie(s) with most expensive tickets' from Movies m
join Screenings sc on m.Id = sc.fk_MovieId
join Tickets t on sc.Id = t.fk_ScreeningId
where t.Price = (select max(t2.Price) from Tickets t2)
group by Title

go
declare @screeningId int = (select top 1 sc.Id from Screenings sc
       join Movies m on sc.fk_MovieId = m.Id
       where m.Title = 'Doctor Who'
       order by sc.[Start] desc
       )
declare @ticketId int;
exec @ticketId = GenerateTicketForScreening @screeningId, 10.00;
select m.Title as 'Movie', sc.[Start], r.Number as 'Room', s.[Row] as 'Seat Row', s.[Col] as 'Seat
Column'
from Screenings sc
join Movies m on m.Id = sc.fk_MovieId
join Rooms r on r.Id = sc.fk_RoomId
join Tickets t on t.fk_ScreeningId = sc.Id
join Seats s on s.Id = t.fk_SeatId
where t.Id = @ticketId

go
select top 1 sum(t.Price) as 'Earnings', m.Title as 'Title'
from Tickets t
join Screenings sc on sc.Id = t.fk_ScreeningId
join Movies m on m.Id = sc.fk_MovieId
group by m.Title
order by Earnings desc
```

Query.sql Teil 2

```
go

--funktioniert nicht:
insert into Screenings (fk_MovieId, fk_RoomId, [Start], [End], Is3D) values
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201),
'08-06-2022 11:30', '08-06-2022 12:30', 0),
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201),
'08-06-2022 11:30', '08-06-2022 12:30', 0)
go
--funtioniert:
insert into Screenings (fk_MovieId, fk_RoomId, [Start], [End], Is3D) values
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201),
'08-06-2022 11:30', '08-06-2022 12:30', 0)
--funktioniert nicht:
insert into Screenings (fk_MovieId, fk_RoomId, [Start], [End], Is3D) values
((select Id from Movies where Title = 'Iron Man 1'), (select Id from Rooms where Number = 201),
'08-06-2022 10:30', '08-06-2022 12:00', 0)
go

drop table if exists #UnfilledScreenings
create table #UnfilledScreenings(
       ScreeningId int
)

declare @idColumn int
select @idColumn = min( Id ) from Screenings
while @idColumn is not null
begin
    declare @freeSeatId int
       exec @freeSeatId = GetFreeSeatIdForScreening @idColumn
       if(@freeSeatId != 0) begin
              insert into #UnfilledScreenings values (@idColumn)
       end
    select @idColumn = min( Id ) from Screenings where Id > @idColumn
end
select m.Title as 'Movie', sc.[Start]
from Screenings sc
join Movies m on m.Id = sc.fk_MovieId
where sc.Id in (select ScreeningId from #UnfilledScreenings)
go
use master;
```