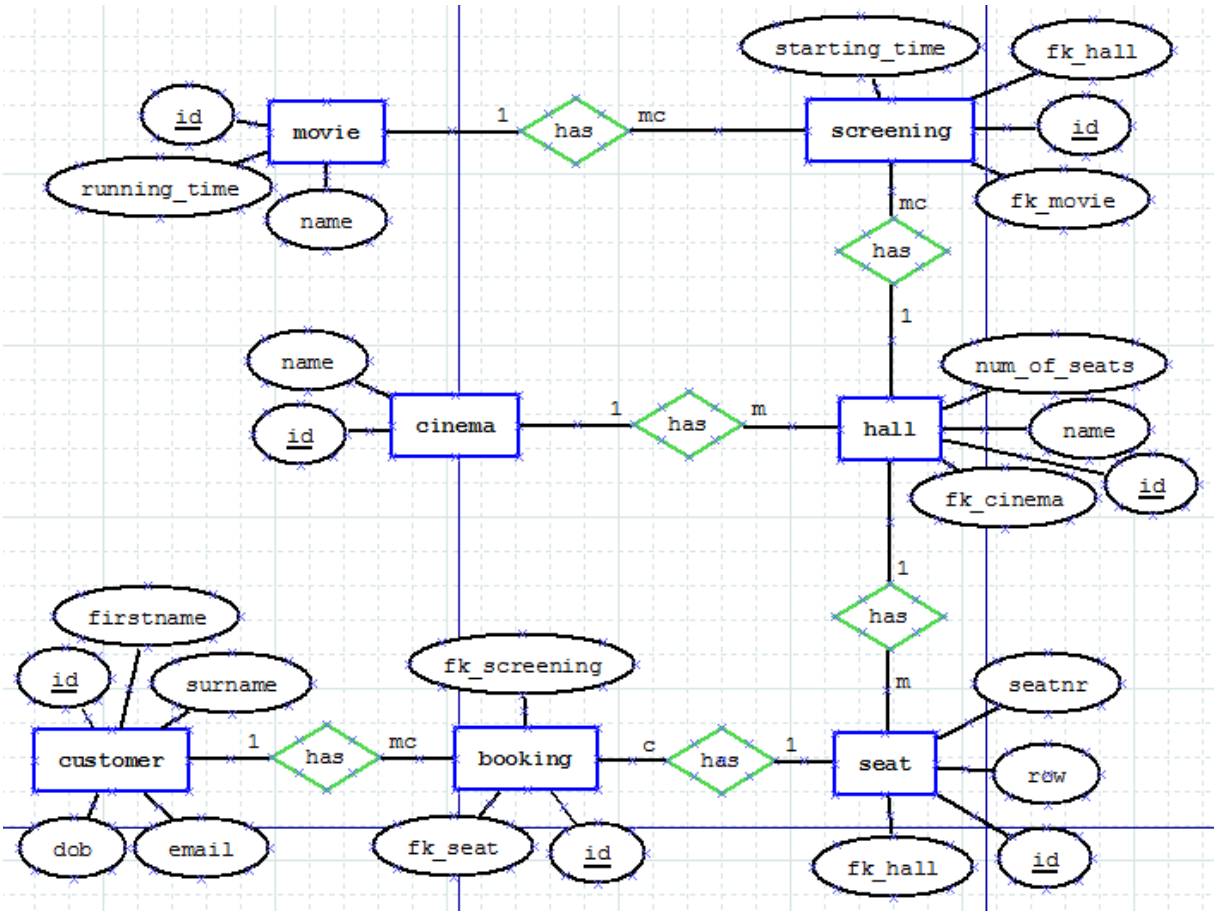
Inverted Exam
Angabe

Hintergrund:

Im Folgenden ist eine Java-Applikation für ein Reservierungssystem eines nicht näher beim Namen genannten, österreichischen Kino-Unternehmens gegeben. Leider hat das Unternehmen unqualifiziertes Personal engagiert, das den hohen Anforderungen und insbesondere den großen Datenmengen dieser Aufgabe nicht gewachsen gewesen war. Infolgedessen schlichen sich Performance-Probleme in das System ein, denen das Management des Kino-Unternehmens nun verzweifelt gegenübersteht. Ihre Aufgabe ist es nun (leider ohne Entlohnung aber dafür mit viel Zeitaufwand) dem Kino-Unternehmen unter die Arme zu greifen und das Reservierungssystem performant zu machen.

Um Ihnen diese Aufgabe zu erleichtern, stellt Ihnen das Kino-Unternehmen selbstverständlich die bisherige Ansammlung an Informationen über das System sowie die Java-Applikation an sich in Form einer ZIP-Datei bereit, die im Folgenden erläutert wird.

Datenmodell



Wie im oben angeführten ER-Diagramm zu erkennen, besteht das Back-End des Programms aus folgenden Tabellen:

- customer

- booking
- cinema
- seat
- hall
- movie
- screening

Im Ordner /scr des entpackten Projektordners befinden sich im init.sql sämtliche SQL-Testdaten bzw. auch die kreierte Datenstruktur. Die Create-Statements der init.sql Datei sind für das Verständnis der Aufgabenstellung auch hier grob angeführt:

```
CREATE TABLE cinema (
    id BIGINT PRIMARY KEY,
    name VARCHAR(100)
);

CREATE TABLE hall (
    id BIGINT PRIMARY KEY,
    name VARCHAR(2),
    num_of_seats BIGINT,
    fk_cinema BIGINT REFERENCES cinema(id)
);

CREATE TABLE seat (
    id BIGINT PRIMARY KEY,
    seatnr BIGINT,
    row BIGINT,
    fk_hall BIGINT REFERENCES hall(id),
);

CREATE TABLE customer(
    id BIGINT PRIMARY KEY,
    firstname VARCHAR(50),
    surname VARCHAR(50),
    email VARCHAR(50),
    dob DATE
);

CREATE TABLE movie (
    id BIGINT PRIMARY KEY,
    name VARCHAR(100),
    running_time BIGINT
);

CREATE TABLE screening (
    id BIGINT PRIMARY KEY,
    starting_time TIMESTAMP,
    fk_movie BIGINT REFERENCES movie(id),
    fk_hall BIGINT REFERENCES hall(id)
);

CREATE TABLE booking (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    fk_seat BIGINT REFERENCES seat(id),
    fk_screening BIGINT REFERENCES screening(id)
);
```

Jeder **customer** kann keinen oder mehrere bookings machen.

Ein **booking** hat wiederum jeweils nur einen customer.
Anhand eines bookings kann ein customer einen seat reservieren, wobei jeder seat zu einem oder keinem booking gehört. Wenn ein seat besetzt wurde, bekommt das booking anhand eines Fremdschlüssels fk_seat die dazugehörige id des seats.

Ein **seat** besitzt zu jedem Zeitpunkt auch einen Fremdschlüssel fk_hall zur dazugehörigen hall.

Eine **hall** hat mehrere seats, aber ein seat gehört nur zu einer hall. Des Weiteren gehört eine hall immer zu einem cinema und hat dementsprechend auch einen Fremdschlüssel fk_cinema.

Ein **cinema** wiederum kann natürlich mehrere halls besitzen.

Ein **screening** hat einen Fremdschlüssel fk_hall, da ein screening immer zu einer hall gehört, eine hall wiederum aber mehrere screenings haben kann. Außerdem besitzt ein screening auch einen Fremdschlüssel fk_movie, da ein screening nur einem movie entsprechen kann.

Ein **movie** wiederum kann aber natürlich mehrere screenings haben, da ein Film ja mehr als einmal vorgeführt wird.

Ein **booking** besitzt neben dem fk_seat auch einen Fremdschlüssel fk_screening, der zur id des screenings referenziert. Da – wie bereits angemerkt – sowohl ein screening als auch ein seat jeweils einen Fremdschlüssel fk_hall besitzen, wird bei einer eingehenden Reservierung mitunter anhand der unten genannten View überprüft, ob diese übereinstimmen.
→ **Sprich:** Läuft die gewünschte Filmvorführung auch in dem Saal, in dem auch der Sitzplatz reserviert wurde und ist dieser Sitzplatz noch frei bzw. gibt es bereits eine Reservierung für den angestrebten Sitzplatz und der angestrebten Vorführung oder nicht?

Nicht außer Acht zu lassen, ist außerdem die View **seatsPerScreening**, die am Ende des init.sql erstellt wird und in welcher die Tabellen seat und screening durch die gemeinsamen hall ids gejoint werden:

```
-- ********************************************************************************************************
-- Create view
-- ********************************************************************************************************

CREATE VIEW seatsPerScreening
    AS SELECT se.id AS seat, sc.id AS screening
    FROM seat se
    INNER JOIN screening sc
    ON se.fk_hall = sc.fk_hall;
```

Aufgabe:

Unter /test befindet sich die Datei „DatabaseTests.java“. Anhand dieser können die jeweiligen Tests der Applikation durchgeführt werden. Im Konsolen-Output wird dabei stets die benötigte Zeit in Millisekunden ausgegeben.

Ziel der Aufgabe ist es, bei einem Gesamtdurchlauf rund 20ms zu erreichen und zu dokumentieren, durch welche Änderungen das umgesetzt wurde bzw. was die dahinterliegenden Probleme waren, die die Performance beeinträchtigt haben.
Tipp: Wichtig sind vor allem jene Klassen, die für eine Reservierung benötigt werden (CustomerDAO, SeatDAO, CheckSeatInHall).

Angestrebte Zeit: ~20ms

Gemessene Zeit am Ende: _____

Dokumentation der gefundenen Performance Probleme & deren Lösungsumsetzung: