**Inverted Exam**

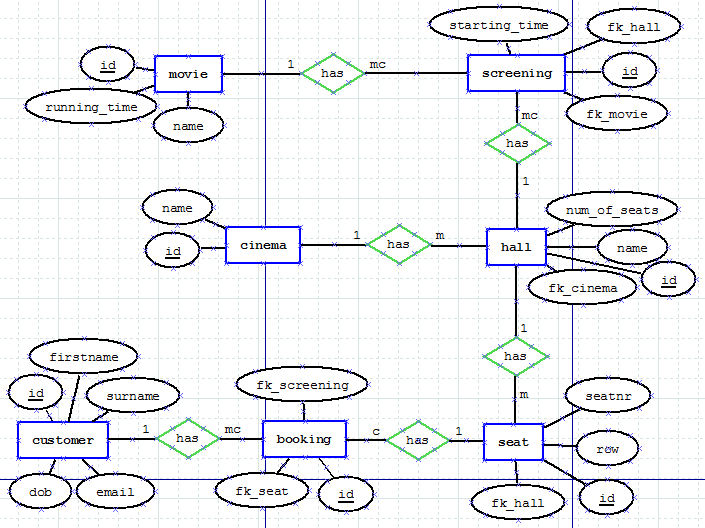
***Angabe***

**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 erhebliche 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 /db des entpackten ZIP-Files befinden sich sämtliche SQL-Dateien mit den Datensätzen für die jeweiligen Tabellen sowie der init.sql Datei, die für das Verständnis der Aufgabenstellung auch hier angeführt ist:~~

|  |  |
| --- | --- |
|  | 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 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?

***Lösung***

1.) Im CustomerDAO können in der Methode reserve() gleich die übergebenen IDs verwendet werden anstatt zuerst ein Seat- und Screening-Objekt zu erstellen.

2.) Des Weiteren können im SeatDAO in der Methode isFree() ebenfalls sofort die übergebenen IDs verwendet werden anstatt zuerst ein Seat- und Screening-Objekt zu erstellen.

3.) Zusätzlich werden unsinnigerweise im SeatDAO in der Methode isFree() alle Seats ausgelesen, danach wird überprüft, ob die IDs mit der übergebenen überein stimmen und wenn ja, wird der Seat in der Variable, die wir in Punkt 2 erstellt haben, gespeichert.

4.)