Milestone 1: Requirements Analysis & Conceptual Design

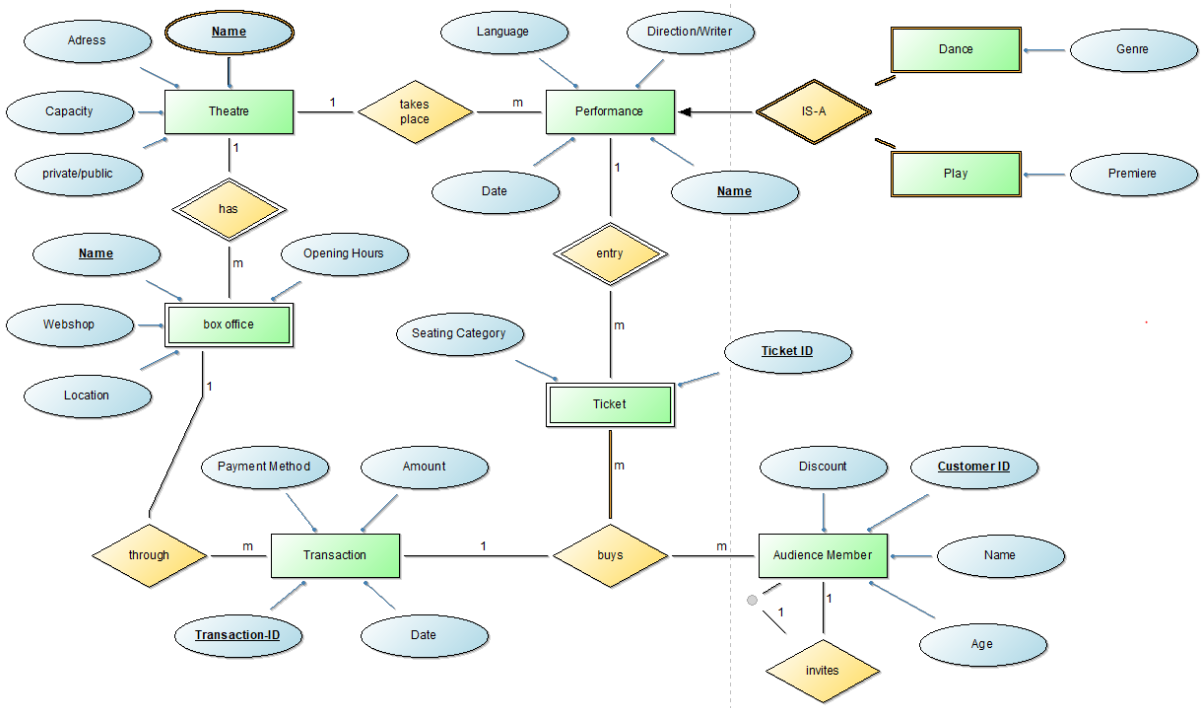
1)a)

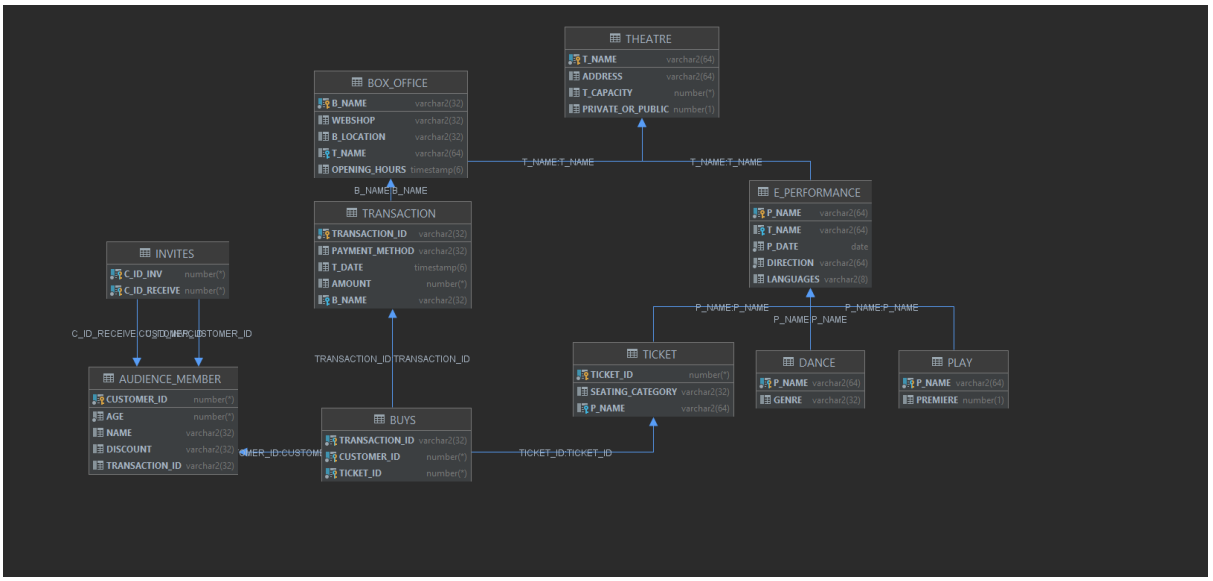
This model corresponds to the idea of theatres as we know them in Vienna. A theatre has a name, which is enough to be an unique identifier, since there is a very limited number of them. Furthermore, it has an adress, a seating capacity and is either privately or state-funded. A theatre has one or more box offices, which have defined opening hours, a name, a location within the theatre and a webshop for purchasing tickets online. Performances take place in a theatre, with language, directors a date and name as attributes. There are specific performances, a dance-performance, with a genre, and a classic play, which can premiere. Audience members can experience these performances by buying tickets, with a seating category and an ID, either through the webshop, or physically at the box office.

An audience member can buy multiple tickets, for oneself, or invite along others. Even if he/she buys multiple tickets, if they are bought at the same time, they are part of one transaction. The transaction is defined by an ID, has a date, an amount and a payment method. Audience members can be identified by their automatically assignend customer ID and have a name and an age, aswell as a discount, which can apply if they have a Membership, or are a student or pensioneer.

1)b)

Initial ER-Diagramm.





Final Diagram rendered by DataGrip based upon actual tables and relations.

Milestone 2: Logical Design

relation + dot (.) + foreign key attribute – references to (◇) – relation + dot (.) + primary key attribute

Theatre (Name, Address, Capacity, private/public)
takes_place.Theatre.Name ◇ Performance.Name
has.Theatre_Name ◇ box_office.Name

box_office(Name, Webshop, Location,Opening_Hours)
has.box_office.Name ◇ Theatre.Name
through.box_office.Name ◇ Transaction.Transaction-ID

Performance (Name, Date,Direction, Language)
is-a.Performance.Name ◇ Dance
is-a.Performance.Name ◇ Play
entry.Performance.Name ◇ Ticket.Ticket-ID
takes_place.Performance.Name ◇ Theatre.Name

Dance (Genre)
is-a.Dance ◇ Performance.Name

Play (Premiere)
is-a.Play ◇ Performance.Name

Ticket (Ticket_ID, Seating_Category)
entry.Ticket.Ticket_ID ◇ Performance.Name
buys.Ticket.Ticket_ID ◇ Audience_Member.Customer_ID
buys.Ticket.Ticket_ID ◇ Transaction.Transaction_ID

Audience_Member (Customer_ID, Age, Name, Discount)
buys.Audience_Member.Customer_ID ◇ Ticket.Ticket_ID
buys.Audience_Member.Customer_ID ◇ Transaction.Transaction_ID
invites.Audience_Member.Customer_ID ◇ .Audience_Member.Customer_ID

Transaction (Transaction_ID, Payment_Method, Date, Amount)
through.Transaction.Transaction_ID ◇ box_office.Name
buys.Transaction.Transaction_ID ◇ Audience_Member.Customer_ID
buys.Transaction.Transaction_ID ◇ Ticket.Ticket_ID

Milestone 4: Implementation

I initially thought about hard coding values for my relations, or generating them from within the java application, but quickly realized that using file imports was a more modular approach. I (hope) the initial workload of configuring the inputs paid off in debugging and changing off table columns.
While I handpicked some of the data, to make sense in this context, I let other entries be generated by https://mockaroo.com/.

Testing of php was done via a docker-hosted instance. I chose bootstrap for the simplicity and way more elegant appearance.