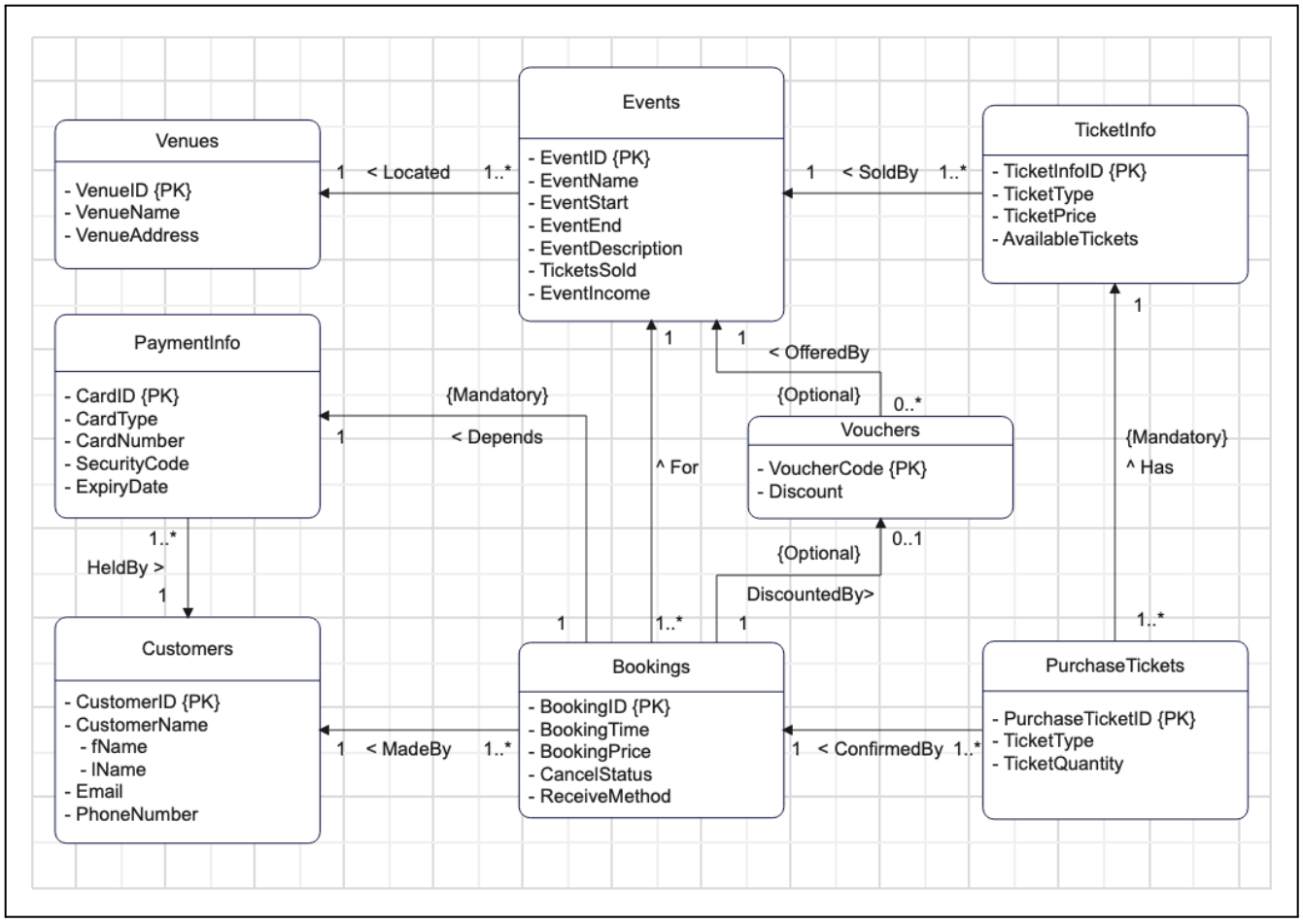
# Conceptual Model Design

## ERD:




# Report

## Entities:

**Venues**

I added a Venues entity to allow for extra details necessary for the ticket buyer and for the database system, as a location in which an event is held is vital information, and often has an impact on the rest of the database. Initially, I was considering simply adding a venue as an attribute for Events, but as I assumed multiple events would be held at the same venue (for example two concerts performed in the same arena), I added Venues as a separate entity. Additionally, it prevents the Events entity from being overcomplex, as it already contains many important attributes. Venues is a simple entity and thus only has three attributes VenueID, VenueName, and VenueAddress. The VenueID attribute is the primary key to uniquely identify each Venue, typically through Events. VenueName and VenueAddress identify the name of the venue and its specific location.

**Events**

The Events entity exists to help link together several other entities. Namely Bookings, Venues, TicketInfo, and Vouchers. The Events entity is composed of the attributes EventID, EventName, EventStart, EventEnd,

EventDescription, TicketsSold, and EventIncome. EventID serves as the primary key to uniquely identify each event. EventName describes the title of the event. EventStart and EventEnd describe the time frame in which the event takes place by datetime. EventDescription simply describes any extra details about the event. TicketsSold calculates the total number of tickets sold for each event. EventIncome calculates the total revenue made from all tickets sold by its type, price, and quantity purchased.

**TicketInfo**
The TicketInfo entity contains all relevant information about the tickets that are sold by each event. It is connected to the Events entity, as which event correlates to each ticket is part of the information of each ticket. The TicketInfo entity has the attributes TicketInfoID, TicketType, TicketPrice, and AvailableTickets. TicketInfoID serves as the primary key to uniquely identify each ticket's information. TicketType describes what category of ticket it may be, for example adult or child, or any other type specified for an event. TicketPrice describes the price of a ticket and AvailableTickets is the attribute to describe the total number of available tickets of the specified TicketType, the value adjusting based on number of purchased tickets.

**Customers**
The Customers entity is a core entity in my database design as storing the basic information of a customer is important when it comes to each booking as each booking must be made by a customer. Furthermore, the Customers entity will associate each purchased ticket with a particular purchaser through bookings, and allows for contact details for complications or for receiving their ticket/receipt. The Customers entity has the attributes CustomerID, CustomerName, Email, and PhoneNumber. The CustomerID attribute is the primary key to uniquely identify each customer. The CustomerName attribute includes sub attributes fName and lName for first name and last name respectively to make my database easier to search through using queries. Email and PhoneNumber serve as contact information. Additionally, Email is required as the ticket can be sent by email as a receive method.

**Vouchers**
The Vouchers entity exists to link to Bookings Events. It is important to note that Vouchers has an optional constraint with both Bookings and Events as it is not required to use a voucher for a booking  nor for an event to offer any vouchers at all. The Vouchers entity only has two attributes, VoucherCode and Discount. VoucherCode serves as the primary key to uniquely identify each code usage as well as describe the code used in the booking. Discount describes the percentage of discount on the ticket price (e.g. 10% = 0.1). Initially, I was considering using an additional VoucherID attribute to use as the primary key, but found that VoucherCode already does the same.

**PaymentInfo**
The PaymentInfo entity exists to store core card details to link to each booking/purchase in event of a cancellation and also ensure each booking was paid in full by a valid card. The PaymentInfo entity has the attributes CardID, CardType, CardNumber, SecurityCode, and ExpiryDate. The CardID attribute serves as the primary key to uniquely identify each Card and to simplify needing all card details for referencing purposes between entities. CardType describes whether it is a Visa, Mastercard, Amex, etc. Lastly, CardNumber, Security Code, and ExpiryDate, all hold the core information of the card.

**Bookings**
The Bookings entity is the core entity in my database design as it is able to link together many entities for the primary purpose of the system being online ticket booking and is the primary entity to describe each payment

made. The linked other entities are Customers, PaymentInfo, Events, Vouchers, and PurchaseTickets. PaymentInfo is linked with Bookings and not solely Customers as the booking depends on if the purchase was made fully and allows for a way to go and store card details in the case of a cancellation. The Booking entity has five attributes BookingID, BookingTime, BookingPrice, CancelStatus, and ReceiveMethod. Firstly, BookingID is the primary key to have a unique identifier for each booking. BookingTime simply describes the time the booking was made. BookingPrice describes the total sum of each booking (based on the type and quantity of any tickets purchased), which is different from TicketPrice because that purely describes each individual ticket price within its designated ticket information entity, while a booking could have multiple tickets purchased. CancelStatus describes whether or not the customer has chosen to cancel the booking or not. ReceiveMethod describes how the customer wants to receive their tickets (e.g. via email, phone number, or pickup at the event).

**PurchaseTickets**

The PurchaseTickets entity acts as the details of each purchased ticket in the context of the booking. I added this entity rather than keeping all ticket data to a single entity to simplify information based on what it is used for. It is connected to the TicketInfo entity, in order to relate the necessary information about the tickets being purchased, such as its price and available tickets for that ticket type. The PurchaseTickets entity has the attributes PurchaseTicketID, TicketType, and TicketQuantity. PurchaseTicketID serves as the primary key to uniquely identify each ticket(s) that is purchased of a certain type. TicketType describes the same as the TicketInfo entity, that being the type of ticket (e.g. adult, child, etc.) which is why I kept the attribute name the same to keep my database clear. TicketQuantity describes the number of that ticket type the customer has purchased in their booking.

## Relationships:

**Events-Venues (Located)**

The Located relationship links together Events and Venues entities as each Event must refer to a venue in which it is located. The cardinality is a many to one ((1..*) to (1)) relationship as one or many events can be held at the same venue, whether it be the same or different date. This relationship assumes that Events cannot be held at different venues at the same time.

**Bookings-PaymentInfo (Depends)**

The Depends relationship links together Bookings and PaymentInfo entities because since Bookings confirms the total price and is the entity in which payments are made, there must be an associated payment information referred to make the purchase. The cardinality is a one to one ((1..*) to (1)) relationship as each booking will have one set of payment information corresponding to it, though the same card information can be used for multiple bookings, for example if a customer books two different events on separate occasions. This relationship assumes that multiple cards cannot be used for the same booking.

**TicketInfo-Events (SoldBy)**

The SoldBy relationship links together TicketInfo and Events entities as tickets are sold by the event and the ticket information holds all key information for tickets. The cardinality is a many to one ((1..*) to (1)) relationship as one event will only ever correspond to one or many tickets with an assumption that an event will never not sell tickets and that tickets can be multipurpose, as in used for two separate events.

**PaymentInfo-Customers (HeldBy)**

The HeldBy relationship links together PaymentInfo and Customers entities as each card must be linked with a customer using it. This assumes that the customer is the cardholder. The cardinality is many to one ((1..*) to (1)) relationship because a customer could use one or multiple payment methods (e.g. different credit cards for separate bookings). This relationship makes the assumption that separate customers would not be sharing card information.

**PurchaseTickets-Bookings (ConfirmedBy)**
The ConfirmedBy relationship links PurchaseTickets and Booking entities as each booking must have associated tickets that were purchased to calculate the total price and also have a reference for sending the tickets (ReceiveMethod) such as by email (via Customers). The cardinality is a many to one ((1..*) to (1)) relationship as one or multiple purchased tickets can only correspond to a single booking. A customer cannot book the same ticket through multiple bookings.

**Bookings-Customers (MadeBy)**
The MadeBy relationship links Bookings and Customer entities as each booking must have an associated customer who makes the booking. The cardinality is a many to one ((1..*) to (1)) relationship as one or many bookings can be made by each customer. This makes the assumption that bookings will never be shared among multiple customers.

**Bookings-Vouchers (DiscountedBy) {Optional}**
The DiscountedBy relationship links Bookings and Vouchers entities as each booking can include the use of a voucher code to discount the BookingPrice. This relationship has an optional tag as each booking does not require the use of a voucher to be completed but each customer has the option of using a voucher. The cardinality is an optional one to one ((1) to (0..1)) relationship as zero or one voucher codes can only be used, if one to discount the price of one booking at a time. This relationship makes the assumption that multiple vouchers cannot be applied to the same booking.

**Vouchers-Events (OfferedBy) {Optional}**
The OfferedBy relationship exists as an event can offer vouchers to help boost sales of tickets and promote the event, thus making the vouchers for that event. This relationship has an optional tag as each event is assumed to not be required to offer vouchers. The cardinality is an optional many to one ((0..*) to (1)) relationship as many or zero voucher codes can only be offered for a singular event. This relationship makes the assumption that multiple events cannot offer the same voucher codes.

**Bookings-Events (For)**
The For relationship links Bookings and Events entities as each booking must correspond to an event The cardinality is a many to one ((1..*) to (1)) relationship as there can be many bookings for the same event but each booking cannot be for more than one event. This makes the assumption that bookings cannot book multiple events at once.

**PurchaseTickets-TicketInfo (Has) {Mandatory}**
The Has relationship links PurchaseTickets and TicketInfo entities as each purchased ticket must have a corresponding ticket information reference that details ticket pricing and ticket availability by type. The cardinality is (1..*) to (1) as many purchased tickets can correspond to the same ticket information, given that the TicketType matches.

# Logical Model Design

**Venues** (VenueID, VenueName, VenueAddress)
**Primary Key:** VenueID

**Events** (EventID, EventName, EventStart, EventEnd, EventDescription, TicketsSold, EventIncome, VenueID)
**Primary Key:** EventID
**Foreign Key:** VenueID references Venues(VenueID)

**TicketInfo** (TicketInfoID, TicketType, TicketPrice, AvailableTickets, EventID)
**Primary Key:** TicketInfoID
**Foreign Key:** EventID references Events(EventID)

**Customers** (CustomerID, fName, lName, Email, PhoneNumber)
**Primary Key:** CustomerID

**Vouchers** (VoucherCode, Discount, EventID)
**Primary Key:** VoucherCode
**Foreign Key:** EventID references Events(EventID)

**PaymentInfo** (CardID, CardType, CardNumber, SecurityCode, ExpiryDate, CustomerID)
**Primary Key:** CardID
**Foreign Key:** CustomerID references Customers(CustomerID)

**Bookings** (BookingID, BookingTime, BookingPrice, CancelStatus, ReceiveMethod, CustomerID, CardID, EventID, VoucherCode)
**Primary Key:** BookingID
**Foreign Key:** CustomerID references Customers(CustomerID)
**Foreign Key:** CardID references PaymentInfo(CardID)
**Foreign Key:** EventID references Events(EventID)
**Foreign Key:** VoucherID references Vouchers(VoucherID)

**PurchaseTickets** (PurchaseTicketID, TicketType, TicketQuantity, TicketInfoID, BookingID)
**Primary Key:** PurchaseTicketID
**Foreign Key:** TicketInfoID references TicketInfo(TicketInfoID)
**Foreign Key:** BookingID references Bookings(BookingID)