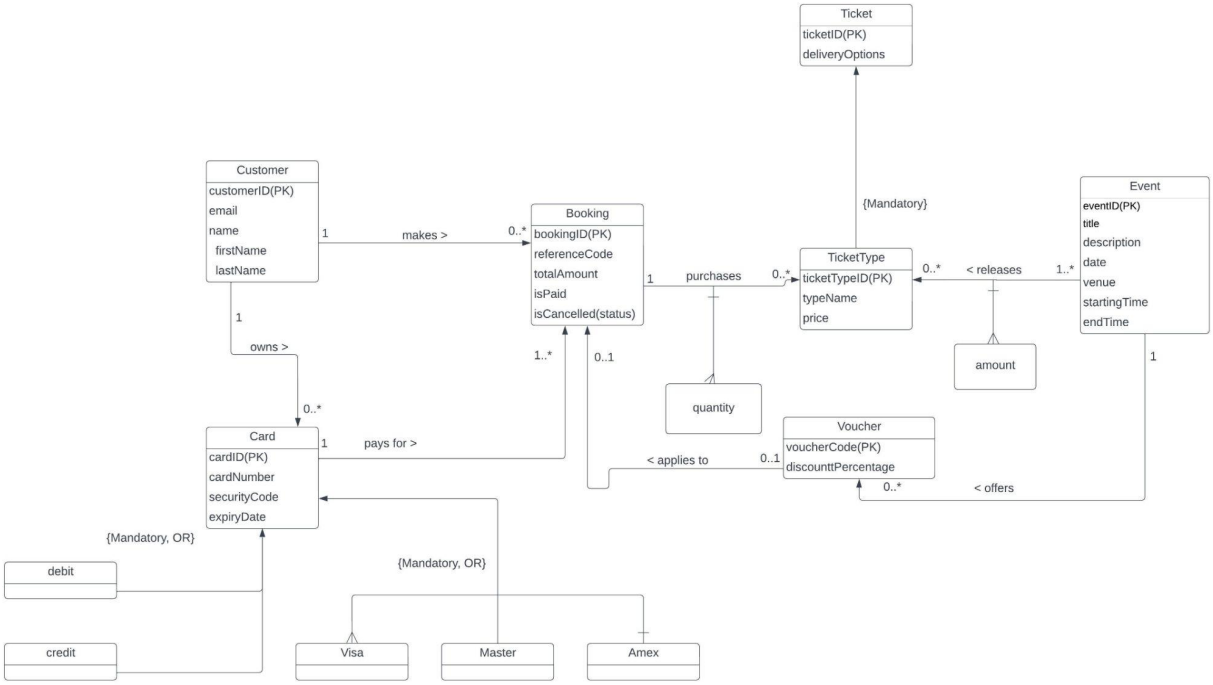
Entity-Relationship Diagram



ENTITY JUSTIFICATION

Commencing with the justification process, I will assess each entity's usage. Additionally, I will explicate the attributes within each entity, highlighting any that require specific justification.

Event:

The Event entity is fundamental as it represents the core elements of the ticketing system – events that customers can book tickets for. It contains the attributes **eventID, title, description, date, venue, startingTime, endTime**. **EventID** is a unique identifier for each event.

TicketType:

The **TicketType** entity is to capture the different types of tickets offered for events. I separated it from the ticket entity as all tickets have to be categorized into a type and they should have different prices accordingly. **ticketType** contains the attributes - **ticketTypeID, typeName, price.** Both **typeName** and **price** are crucial for categorizing tickets as types of tickets are classified by their prices. while **ticketTypeID** is the unique identifier for the entity as types of tickets for different events might have similar names like different child tickets for similar age range and day passes on different days.

Ticket:

The **Ticket** entity tracks individual tickets. A ticket can be any one of the types. This has a mandatory participation as a ticketType as each ticket has to belong to a type and have its relevant price. It contains the attributes **ticketID** and **deliveryOptions**. **ticketID** serves as a primary key for unique identification. I was considering applying specialization for **deliveryOptions**, but given that the options are either pick up in person or send the ticket by email, it is better to get the options joint as an attribute to include it in the **ticket** entity.

Customer:
The **Customer** entity represents individuals using the ticketing system. **Customer** contains the attributes **customerID, email, firstName, lastName.** I included **email** as the customer might choose to pick up the ticket by receiving emails and it makes the customer contactable.

Booking:
The **Booking** entity captures the information related to a customer's reservation. It contains the attributes **bookingID, referenceCode, totalAmount, isPaid, isCancelled. BookingID** ensures a unique identifier. **IsPaid** and **isCancelled** are both a boolean indicator (true/false) indicating if the booking has been paid and canceled respectively. When the time passed the **startingTime** of the event, the isCancelled for the booking will be set to False and could not be changed again.

Voucher:
The **Voucher** entity is designed to manage and track voucher-related information. As not every event will have a voucher to apply to it I decided to make it a separate entity, which has an optional participation constraint with the event. It contains the attribute **voucherCode** and **discountPercentage.** I decided not to give the entity an id as each voucher is unique and **voucherCode** itself could be a unique identifier. **DiscountPercentage** allows the system to calculate the discounted amount during the booking process and ensures accurate transactions on the booking.

Card
The **Card** entity manages transaction details of a customer for the bookings. It contains attributes **cardID**, **cardNumber**, **securityCode** and **expiryDate.** Specialization is applied in the types of card**,** making the card to have two mandatory constraints, to show if the card is debit/credit and which company it is issued by. I choose to implement my design in this way as it may become confusing and hard to follow if I make two attributes for cardTypes in the same entity. Thus the **cardType** will include all the card payment options such as Visa debit, Master credit and Amex, and there will be a few combinations.

Relationship Justification
In this part the cardinality of relationships in my design will be covered.

Releases
The **releases** relationship indicates multiple events can be associated with zero to many **TicketType** records, showing that many events can have multiple types of tickets, each represented by a **TicketType**. An event also can release different ticketTypes for different prices for sale. A relationship attribute is depicted in the relationship in the diagram.The relationship attribute **amount** shows how many tickets for the specific ticket type the event is releasing, indicating the maximum number of ticketType the particular event can sell for.

Makes
The relationship is created as the customers will make a booking in the system. Each customer can make unlimited booking thus it is a one-to-many relationship, while the booking is only made by a single customer.

Owns
Once the credit card details are inputted in the system, the system will save the details, so the one-to-many relationship is created. One customer can own multiple cards and save their details in the system.

Pays for
Each booking can only have one associated card payment method, while one card can pay for plenty of bookings so it is a one-to-many relationship.

Purchases
Each booking transaction has the potential to reserve multiple tickets of different types for the chosen event. This accounts for scenarios where customers may purchase tickets for a group or multiple ticket types in a single booking. Every ticket that is part of the reservation process is associated with a specific booking. The relationship contains a relationship attribute **quantity**, indicating how many tickets from a specific type the customer is trying to reserve.

Offers
The **"Offers"** relationship is established to represent the connection between an event and the vouchers it makes available to customers. An event has the flexibility to provide customers with 0 to multiple vouchers. Every voucher is specifically associated with one event as it is unique.

Applies to
The relationship suggests Zero-or-One Voucher for a booking. Not every booking needs to have a voucher. Customers may make bookings without applying for any vouchers. Hence, the minimum cardinality on the Voucher side is set to 0. Each voucher is designed to be applied to a specific booking if they are applied. This ensures a one-to-one relationship, indicating that a voucher can be used for a single booking transaction.

RELATIONAL MODEL

Event (eventID, title, description, date, venue ,startingTime, endTime)
Primary Key: eventID

Ticket (ticketID, deliveryOptions, ticketTypeID)
Primary Key: ticketID
Foreign Key: ticketTypeID references TicketType(ticketTypeID)

TicketType (ticketTypeID, typeName, price):
Primary Key: ticketTypeID

Customer (customerID, email, firstName, lastName, cardID):
Primary Key: customerID
Foreign Key: cardID

Booking (bookingID, referenceCode, totalAmount, isPaid, isCancelled, customerID, cardID, voucherCode):
Primary Key: bookingID
Foreign Key: customerID
Foreign Key: cardID
Foreign Key: voucherCode

Voucher (voucherCode, discountPercentage, EventID):
Primary Key: voucherCode

Foreign Key: eventID

Card (cardID, cardNumber, cardType, securityCode, expiryDate):
Primary Key: cardID

Credit (cardID)
Primary Key: cardID
Foreign Key: cardID references Card(cardID)

Debit (cardID)
Primary Key: cardID
Foreign Key: cardID references Card(cardID)

Visa (cardID)
Primary Key: cardID
Foreign Key: cardID references Card(cardID)

MasterCard (cardID)
Primary Key: cardID
Foreign Key: cardID references Card(cardID)

Amex (cardID)
Primary Key: cardID
Foreign Key: cardID references Card(cardID)

Relationships:
Releases (eventID, ticketTypeID, amount)
Primary Key: eventID, ticketTypeID
Foreign Key: eventID references Event(eventID)
Foreign Key: ticketTypeID references TicketType(ticketTypeID)

Purchases: (bookingID, ticketTypeID, quantity)
Primary Key: bookingID, ticketTypeID
Foreign Key: bookingID references Booking(bookingID)
Foreign Key: ticketTypeID references ticketType(ticketID)