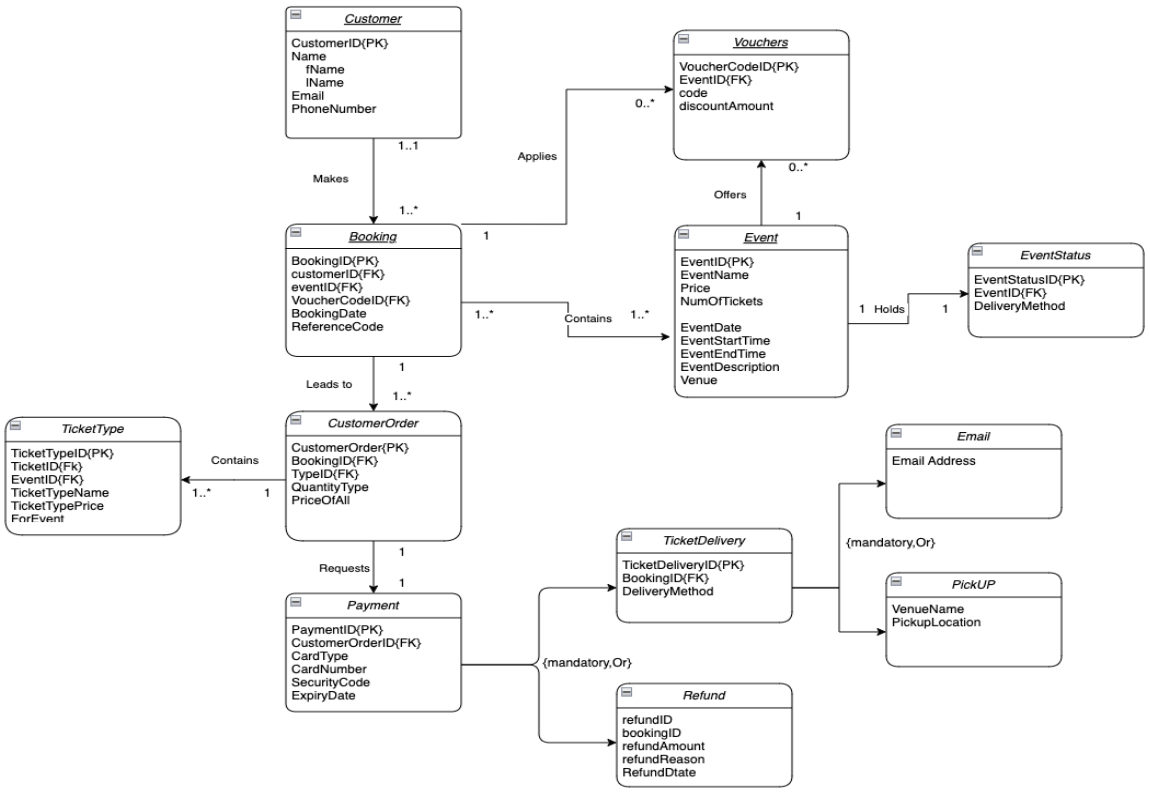
ER diagram: Online Ticket Booking System



Entity Justification

Customer:

The "Customer" entity represents individuals who engage with the online ticket booking system. Each customer is uniquely identified by the "CustomerID," serving as the primary key. The attributes include essential customer information such as first name ("fName"), last name ("lName"), email address ("Email"), and phone number ("PhoneNumber"). These details are crucial for maintaining customer records and facilitating communication.

Customer(fName,lName,Email,PhoneNumber)
Primary key: CustomerID

Booking:

The "Booking" entity captures information about bookings made by customers. The "BookingID" serves as the primary key, uniquely identifying each booking. Attributes such as "BookingDate," "ReferenceCode," and "TotalAmount" store relevant details about the booking. Foreign keys ("CustomerID," "EventID," "VoucherCode") establish relationships with the "Customer" and "Event" entities, enabling a comprehensive representation of booking data.

Booking(BookingDate,ReferenceCode)
Primary Key: BookingID
Foreign key: CustomerID, EventID,VoucherCodeID

CustomerOrder:

The "CustomerOrder" entity provides a detailed breakdown of items booked within a specific booking. The "CustomerOrderID" serves as the primary key, ensuring unique identification. Attributes like "QuantityType" and "PriceOfAll" offer specifics about the booked items. Foreign keys ("BookingID," "TypeID") establish relationships with the "Booking" and "TicketType" entities, contributing to a comprehensive view of booking details.

CustomerOrder(QuantityType,PriceOfAll)
Primary Key: CustomerOrderID
Foreign Key: BookingID,TypeID

Vouchers:

The "Vouchers" entity represents voucher codes associated with events. The "VoucherCodeID" acts as the primary key, uniquely identifying each voucher. Attributes include "Code" and "DiscountAmount," providing information about the voucher code and associated discount. The foreign key ("EventID") establishes a relationship with the "Event" entity, associating vouchers with specific events.

Vouchers(Code,disountAmount)
Primary Key: VoucherCodeID
Foreign Key: EventID

Event:

The "Event" entity represents individual events available for booking. The "EventID" serves as the primary key, uniquely identifying each event. Attributes such as "EventName," "Price," and "NumOfTickets" store key details about the event, including its name, ticket price, and available ticket quantity.

Event(EventName,Price,NumOfTickets,EventDate,EventStartTime,EventEndTime,EventDescription,EventVenue)
PrimaryKey: EventID

EventStatus:

The "EventStatus" entity captures the status or delivery method associated with an event. The "EventStatusID" acts as the primary key. The "DeliveryMethod" attribute provides information about how tickets are delivered. The foreign key ("EventID") establishes a relationship with the "Event" entity, associating the status or delivery method with specific events.

EventStatus(DeliveryMethod)
Primary Key: EventStatusID
Foreign Key: Event ID

Payment:

The "Payment" entity represents payments made for bookings. The "PaymentID" serves as the primary key, ensuring uniqueness. Attributes such as "CardType," "CardNumber," "SecurityCode," and "ExpiryDate" store payment-related details. The foreign key ("CustomerOrderID") establishes a relationship with the "Booking" entity, linking payments to specific bookings.

Payment(CardType,CardNumber,SecurityCode,ExpiryDate)
Primary Key: PaymentID
Foreign Key: CustomerOrderID

TicketDelivery:
The "TicketDelivery" entity represents the chosen delivery method for tickets. The "TicketDeliveryID" serves as the primary key. The "DeliveryMethod" attribute provides information about the selected ticket delivery option. The foreign key ("BookingID") establishes a relationship with the "Booking" entity, associating delivery methods with specific bookings.

TicketDelivery(DeliveryMethod)
Primary Key: TicketDeliveryID
Foreign Key: BookingID

Refund:
The "Refund" entity represents refunds associated with payments. The "RefundID" serves as the primary key, ensuring uniqueness. Attributes such as "RefundAmount," "RefundReason," and "RefundDate" store details about the refund process. The foreign key ("BookingID") establishes a relationship with the "Booking" entity, linking refunds to specific bookings.

Refund(RefundAmount, RefundReason,RefundDate)
Primary Key: RefundID
Foreign Key: BookingID

TicketType:
The "TicketType" entity represents different types of tickets available for events.
The "TypeID" serves as the primary key, uniquely identifying each ticket type. Attributes such as "TicketName" and "EventType" provide information about the nature and category of each ticket. The foreign key ("EventID") establishes a relationship with the "Event" entity, associating ticket types with specific events.

TicketType(TicketTypeName,TicketTypePrice)
Primary Key: TypeID
Foreign Key: EventID

Email:
The "Email" entity likely represents email addresses associated with customers or other entities in the system. Although the specific purpose is not explicitly stated, having a repository for email addresses can facilitate communication and contact.

Email(EmailAddress)

Pickup:
The "Pickup" entity captures information about pickup locations at venues. While the specific purpose is not entirely clear, it may be related to ticket pickup options for customers attending events.

Pickup(VenueName,PickupLocation)

Relationship Justification

Customer to Booking “Makes“ relationship:

A customer needs a relationship with the booking they are creating as it is a main function of the ticket system .Exactly one customer is needed to make a booking, multiple customers cannot make the same booking hence the constraint is 1 on the Customer side.One customer however can make multiple bookings hence it is a one to many relationship.

Booking to Event “Contains” relationship:

The Booking entity needs a relationship with the Event entity as the contents of the booking will need to have either one or multiple events as a part of the booking. One booking must contain at least one event and can even contain multiple in one.As well as this an event can be booked by multiple customers.Also there must be at least one booking in order for it to contain an event. Therefore for this relationship it will be a many to many relationship.

Booking to Voucher “Applies” relationship:

There also needs to be a relationship that specifies what Voucher the customer's booking applies and that is why this was created . There needs to be at least one booking in order for a voucher to be applied , and I made the assumption that a booking can apply multiple vouchers to its booking for the different promotions and different events,also i made it start at zero as some customers may not have a voucher. However a voucher should not be able to be applied to multiple bookings , just like you can get one time vouchers as gifts that are only able to be used by one person. Therefore this relationship is one to many .

Event to Voucher “Offers” relationship:

There should also be a relationship between Event and Vouchers which states the availability of the possible vouchers that can be applied to the customers booking order for specific events. There needs to be exactly one event to multiple vouchers, as one event could have multiple different vouchers that can be applied to it but there shouldn't be multiple events with the same voucher and code etc.In conclusion it should be a one to many relationship.

Booking to CustomerOrder “Leads to “ relationship:

A relationship is needed here because there should be something that summarises the order and has the amount of tickets for each event the customer has booked in a booking. It is very important to have an order of the tickets that they ordered and a calculated price of all this. Multiple bookings can't have the same CustomerOrder as it should be unique to each customer, however one customer should be able to have multiple Customer orders hence it will be one to many.

CustomerOrder to TicketType “Contains” relationship:

CustomerOrder should contain singular or multiple of the ticket entity, and hence there should be a relationship between the two. The relationship should be one to many as one customer order should be able to have at least one Ticket type in order to make the order valid. As well as this a ticket type should have only one customer order as each ticket has a unique id hence multiple customer orders shouldnt have the same ticket with the same ticket id they should each have their own unique ticket id per customer order. That is why I made the relationship one to many.

CustomerOrder to Payment "Requests" relationship:

CustomerOrder to payment "Requests" relationship. This relationship is crucial as there needs to be a way of paying for the customer's order , and in order for simplicity and better readability it should be in a separate class. The relationship should be one to one .

Payment specialisation:

I decided to go with a specialisation for the payment entity and include ticket delivery / refund as a "mandatory,or" type.This is because I assumed as part of the payment the customer will either order their tickets for delivery or process a refund when they can't make it and want there money back

TicketDelivery specialisation:

I made the decision of making the ticket delivery a specialisation because the customer will either (hence the "mandatory,or" decide to pick up the ticket from the venue or instead get it emailed to them to save them the trip.