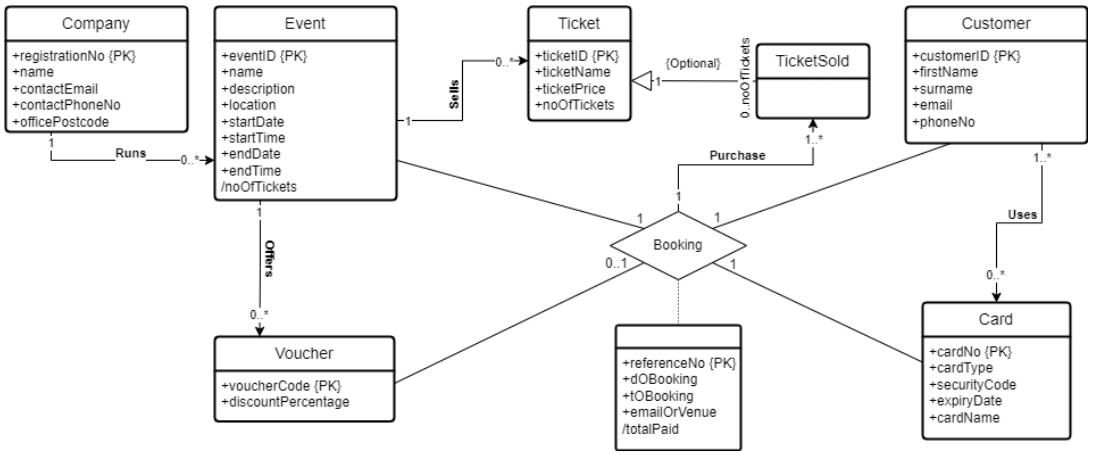
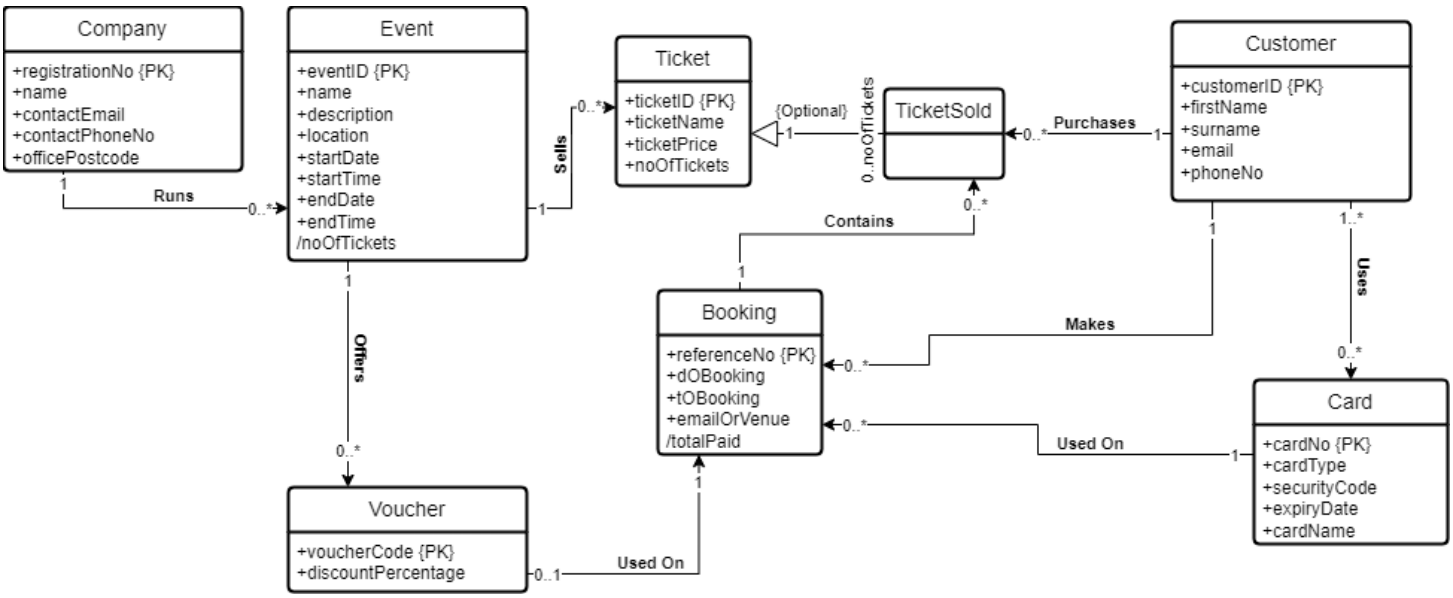
Ticket Design

My Entity Relationship Diagram

When initially approaching the problem statement, my Entity Relationship Diagram looked like:



However, to avoid the Chasm Trap between the customer and the ticket, in which it is not possible to see which tickets a customer has purchased, I decided it was best to design the booking relationship to look like the one below instead. This way it is possible to see which customer has purchased which ticket. For this reason, my relational model and the rest of the project is based off of the Entity Relationship Diagram below as opposed to the one above.



My Report

Company

The Company entity seemed logical to design before the Event entity. It represents the companies who run events. Since one company can have either none or many events in the database at once, the relationship with Event is clearly a one-to-many (1:*) binary relationship. The attributes held about the company can be simple such as the company name, contact information, office location, and the company's registration number. The registration number for a company is a unique identifier so it follows that registrationNo should be the primary key for this entity.

Event

I designed the Event entity next and found this to be the most fundamental. It is what the whole system is based around: the events which are put on for customers to attend. I decided that the attributes for the event should be its name, dates, times, location, and description since these were the most important details to describe the event. The noOfTickets attribute represents the number of tickets which may be sold for a particular event. This attribute is derived from the Ticket entity which has its own noOfTickets attribute representing the split of different tickets for each event. For example, an event offers 750 adult tickets and 250 child tickets, thus this event has a total of 1000 tickets. The eventID attribute should obviously be the primary key for this entity and would be automatically incremented since each other attribute could be identical for different events. The one-to-many (1:*) binary relationship between Event and Company will also need a foreign key within this entity, which would reference the registrationNo attribute from Company.

Voucher

The attributes for the Voucher entity are the voucher code and the percentage discount which the voucher offers. The voucher code should be unique and so this is the attribute chosen to be the primary key. Each Event can offer either zero or multiple Vouchers and so this relationship is clearly a one-to-many (1:*) binary relationship. For a booking, either one voucher or no voucher can be used thus the cardinality on the Voucher side should be 0..1 and the cardinality on the Booking side should be 1.

Ticket

I then designed the Ticket entity next since it followed from the relationship this entity has with the Event entity. It represents the ticket types offered for each event. An Event sells Tickets on a one-to-many (1:*) binary relationship - an event can sell either none or many ticket types. The attributes for the Ticket entity then logically follow to be the name of the ticket (for example "Adult"), the price, and the how many tickets can be sold. The noOfTickets attribute represents the number of tickets which can be sold for each specific ticket type for an event. The ticketID is required to be the primary key since it is the only candidate key - it could be an incremental attribute to ensure that each entry has a unique ID. The attribute list will also include the eventID as a foreign key derived from the relationship to Event.

TicketSold

The TicketSold entity uses specialisation/generalisation since a TicketSold is a Ticket attached to a customer after purchase. Thus, it shares all the same attributes of a Ticket which is offered by an Event. This entity is different from the Ticket entity however, since it is not the ticket type but the actual ticket. The only extra attributes which will be required for the TicketSold entity, compared to Ticket, will be derived from the relationships that this entity has with the Customer and Booking entities. The extra attributes would be the foreign keys: customerID and bookingID. The relationship with the Customer entity was added to ensure that the chasm trap, in which it was not possible to see which tickets a customer owned, was avoided. The cardinality of this relationship would clearly be that, for each Ticket, there can be between 0 and `noOfTickets` TicketSold relations (as seen on the diagram above).

Customer

The Customer entity holds information about the people purchasing tickets for events. The attributes required for this entity are the customer's name and contact details. I decided it was best to design the name to be split down into first name and surname as this makes both sorting and searching the database easier. For the contact details, I added phone number and email address. These attributes should include formatting checks (for example, making sure that the email address contains an @ symbol), using regular expression, to ensure that no errors occur during data insertions. The Customer entity has relationships with Card, TicketSold, and Booking. The relationship with TicketSold will be one-to-many (1:*) since a customer can purchase as many tickets as they like. The relationship with Booking is also one-to-many (1:*) since a customer can have multiple bookings for their tickets they have purchased. The relationship with Card was a slightly trickier one to think about since a one-to-many relationship would allow a customer to use multiple cards to purchase tickets but would not allow multiple customers to use the same card. The latter can sometimes occur though (for example, two people with a joint bank card) and so it became obvious that a many-to-many (*:*) relationship was more suitable. The cardinality on the Customer side of the relationship should be 1..* since the Customer has mandatory participation, and the Card side should be 0..* since the Card entity has optional participation.

Card

This entity obviously represents the payment cards that Customers use to purchase tickets. The attributes required for this entity are the card number, type, security code, expiration date, and the nickname the customer wishes to assign to the card. I chose the card number as the primary key for this entity as this is a unique number (each debit/credit card has a different card number). In my relational model, the many-to-many (*:*) relationship with the Customer entity would require an extra relation (for example, a CardLink table) else the primary key constraint - uniqueness - would be breached. Card also has a relationship with Booking since it is useful to store which card is used for each booking. This relationship is a one-to-many (1:*) relationship since only one card can be used per booking, but many bookings can be made with the same card.

Booking

The booking entity was certainly the most complex to design. It has relationships with several other entities as well as its individual attribute design. The attributes I chose were the date and time that the booking was made, and whether the tickets are to be picked up at the venue or emailed. The date and time attributes can utilise the SQL functions *current_date()* and *current_time()*. Since none of the aforementioned attributes are unique to a booking, an incremental booking reference number is added to the attribute list. The relationships between Booking and the Voucher, TicketSold, Customer, and Card entities are justified in their own respective explanations. For each of these relationships except from TicketSold, the Booking entity would also include the foreign keys referencing the primary keys for Customer, Card, and Voucher.

My Relational Model

Company (registrationNo, name, contactEmail, contactPhoneNo, officePostcode)
Primary Key: registrationNo

Event (eventID, name, description, location, startDate, startTime, endDate, endTime, companyRegNo)
Primary Key: eventID
Foreign Key: companyRegNo references Company(registrationNo)

Voucher (voucherCode, discountPercentage, eventID)
Primary Key: voucherCode
Foreign Key: eventID references Event(eventID)

Customer (customerID, firstname, surname, email, phoneNo)
Primary Key: customerID
Alternate Key: email, phoneNo

CardLink (customerID, cardNo)
Primary Key: customerID, cardNo
Foreign Key: customerID references Customer(customerID), cardNo references Card(cardNo)

Card (cardNo, cardType, securityCode, expiryDate, cardName)
Primary Key: cardNo

Booking (bookingRef, dOBooking, tOBooking, emailOrVenue, customerID, voucherCode, eventID, totalPaid, cardNo)
Primary Key: bookingRef
Foreign Key: customerID references Customer(customerID), voucherCode references Voucher(voucherCode), eventID references Event(eventID), cardNo references Card(cardNo)

Ticket (ticketID, ticketName, ticketPrice, eventID, noOfTickets)
Primary Key: ticketID
Foreign Key: eventID references Event(eventID)

TicketSold (ticketID, customerID, bookingRef)
Foreign Key: customerID references Customer(customerID), bookingRef references Booking(bookingRef)