Part 1: Data Retrieval Queries

A. Simple Queries

1. Write a query to return the Name of every Track in the database.

```
SELECT name
FROM Track
```

2. Write a SQL query that returns album and track information for tracks where the composer information is missing.

```
SELECT Album.Title, Track.Name
FROM Track
INNER JOIN Album ON Track.AlbumId = Album.AlbumId
WHERE Composer IS NULL
```

Title	Name
Balls to the Wall	Balls to the Wall
Warner 25 Anos	Desafinado
Warner 25 Anos	Garota De Ipanema
Warner 25 Anos	Samba De Uma Nota Só (One Note Samba)
Warner 25 Anos	Por Causa De Você
Warner 25 Anos	Ligia
Warner 25 Anos	Fotografia
Warner 25 Anos	Dindi (Dindi)
Warner 25 Anos	Se Todos Fossem Iguais A Você (Instrumental)
Warner 25 Anos	Falando De Amor
Warner 25 Anos	Angela
Warner 25 Anos	Corcovado (Quiet Nights Of Quiet Stars)
Warner 25 Anos	Outra Vez
Warner 25 Anos	O Boto (Bôto)
Warner 25 Anos	Canta, Canta Mais
Alcohol Fueled Brewtality Live! [Disc 1]	Intro/ Low Down
Alcohol Fueled Brewtality Live! [Disc 1]	13 Years Of Grief
Alcohol Fueled Brewtality Live! [Disc 1]	Stronger Than Death
Alcohol Fueled Brewtality Live! [Disc 1]	All For You
Alcohol Fueled Brewtality Live! [Disc 1]	Super Terrorizer
Alcohol Fueled Brewtality Live! [Disc 1]	Phoney Smile Fake Hellos
Alcohol Fueled Brewtality Live! [Disc 1]	Lost My Better Half
Alcohol Fueled Brewtality Live! [Disc 1]	Bored To Tears
Alcohol Fueled Brewtality Live! [Disc 1]	A.N.D.R.O.T.A.Z.
Alcohol Fueled Brewtality Live! [Disc 1]	Born To Booze
Alcohol Fueled Brewtality Live! [Disc 1]	World Of Trouble
Alcohol Fueled Brewtality Live! [Disc 1]	No More Tears
Alcohol Fueled Brewtality Live! [Disc 1]	The Begining... At Last
Alcohol Fueled Brewtality Live! [Disc 2]	Heart Of Gold
Alcohol Fueled Brewtality Live! [Disc 2]	Snowblind

3. Select everything from Invoice orderd by InvoiceDate, newest to oldest

```
SELECT *
FROM Invoice
ORDER BY InvoiceDate DESC
```

4. List all the Track Names for PlayListID = 17

```
SELECT Name
FROM Track
INNER JOIN PlaylistTrack ON Track.TrackID = PlaylistTrack.TrackID
WHERE PlaylistTrack.PlaylistID =17
```

+ Options

←T→				Name
Edit	Inline Edit	Copy	Delete	For Those About To Rock (We Salute You)
Edit	Inline Edit	Copy	Delete	Balls to the Wall
Edit	Inline Edit	Copy	Delete	Fast As a Shark
Edit	Inline Edit	Copy	Delete	Restless and Wild
Edit	Inline Edit	Copy	Delete	Princess of the Dawn
Edit	Inline Edit	Copy	Delete	N.I.B.
Edit	Inline Edit	Copy	Delete	Supernaut
Edit	Inline Edit	Copy	Delete	Wrathchild
Edit	Inline Edit	Copy	Delete	Killers
Edit	Inline Edit	Copy	Delete	Where Eagles Dare
Edit	Inline Edit	Copy	Delete	2 Minutes To Midnight
Edit	Inline Edit	Copy	Delete	Wasted Years
Edit	Inline Edit	Copy	Delete	Run to the Hills
Edit	Inline Edit	Copy	Delete	Enter Sandman
Edit	Inline Edit	Copy	Delete	The Four Horsemen
Edit	Inline Edit	Copy	Delete	Seek & Destroy
Edit	Inline Edit	Copy	Delete	Master Of Puppets
Edit	Inline Edit	Copy	Delete	For Whom The Bell Tolls
Edit	Inline Edit	Copy	Delete	Creeping Death
Edit	Inline Edit	Copy	Delete	Ace Of Spades
Edit	Inline Edit	Copy	Delete	Live To Win
Edit	Inline Edit	Copy	Delete	Looks That Kill
Edit	Inline Edit	Copy	Delete	I Don't Know
Edit	Inline Edit	Copy	Delete	Crazy Train
Edit	Inline Edit	Copy	Delete	Flying High Again
Edit	Inline Edit	Copy	Delete	The Zoo

5. Write a SQL Query that returns the following from the Chinook Database:

Artist Name
Album Title
Track Name
Genre Name
Composer
Unit Price

```
SELECT Artist.Name, Album.Title, Track.Name, Genre.Name, Track.Composer, Track.UnitPrice
FROM Track
INNER JOIN Album ON Album.AlbumId = Track.AlbumId
INNER JOIN Genre ON Genre.GenreId = Track.GenreId
INNER JOIN Artist ON Artist.ArtistId = Album.ArtistId
```

B. Built in Function Queries

1. Write a query will return the total number of invoices grouped by customerid.

```
SELECT CustomerId, COUNT( CustomerID )
FROM Invoice
GROUP BY CustomerId
ORDER BY CustomerId
```

CustomerId	COUNT(CustomerID)
1	7
2	7
3	7
4	7
5	7
6	7
7	7
8	7
9	7
10	7
11	7
12	7
13	7
14	7
15	7
16	7
17	7
18	7
19	7
20	7
21	7
22	7
23	7
24	7
25	7
26	7
27	7
28	7
29	7
30	7
31	7
32	7
33	7
34	7
35	7
36	7
37	7
38	7
39	7

2. Write a query which will return total number of tracks per AbumID

```
SELECT Album.AlbumId, Album.Title, COUNT( TrackId )
FROM Track
INNER JOIN Album ON Album.AlbumId = Track.AlbumId
GROUP BY AlbumId
ORDER BY AlbumId
```

+ Options

AlbumId	Title	COUNT(TrackId)
1	For Those About To Rock We Salute You	10
2	Balls to the Wall	1
3	Restless and Wild	3
4	Let There Be Rock	8
5	Big Ones	15
6	Jagged Little Pill	13
7	Facelift	12
8	Warner 25 Anos	14
9	Plays Metallica By Four Cellos	8
10	Audioslave	14
11	Out Of Exile	12
12	BackBeat Soundtrack	12
13	The Best Of Billy Cobham	8
14	Alcohol Fueled Brewtality Live! [Disc 1]	13
15	Alcohol Fueled Brewtality Live! [Disc 2]	5
16	Black Sabbath	7
17	Black Sabbath Vol. 4 (Remaster)	10
18	Body Count	17
19	Chemical Wedding	11
20	The Best Of Buddy Guy - The Millenium Collection	11
21	Prenda Minha	18
22	Sozinho Remix Ao Vivo	3
23	Minha Historia	34
24	Afrociberdelia	23
25	Da Lama Ao Caos	13
26	Acústico MTV [Live]	17
27	Cidade Negra - Hits	14
28	Na Pista	10
29	Axé Bahia 2001	14
30	BBC Sessions [Disc 1] [Live]	14

3. If you bought one copy of every track, how much would you spend in total?

```
SELECT SUM( Track.UnitPrice )
FROM Track
```

+ Options

SUM(Track.UnitPrice)
3680.97

4. What is the average track price in the store?

```
SELECT AVG( Track.UnitPrice )
FROM Track
```

+ Options

Avg(Track.UnitPrice)
1.050805

5. How many distinct genres of music are represented by tracks sold in the store?

```
SELECT GenreId, COUNT( Track.GenreId )
FROM Track
GROUP BY GenreId
```

+ Options

GenreId	COUNT(Track.GenreId)
1	1297
2	130
3	374
4	332
5	12
6	81
7	579
8	58
9	48
10	43
11	15
12	24
13	28
14	61
15	30
16	28
17	35
18	13
19	93
20	26
21	64
22	17
23	40
24	74
25	1

The Tracks in the store cover all 25 genres listed in the genre table

6. How many artists have albums in the store?

```
SELECT COUNT( DISTINCT Album.ArtistId )
FROM Album
```

+ Options

COUNT(DISTINCT Album.ArtistId)
204

7. Create a view (named “cheap_tracks”) for all tracks where UnitPrice < 1

```
CREATE VIEW cheap_tracks AS
SELECT Track.Name
FROM Track
WHERE UnitPrice < 1
```

C. Advanced Queries

1. Find the FirstNames of the employees that are older than their corresponding supervisor. Output should have only one column named "FirstName".

```
SELECT Employee.FirstName
FROM Employee
WHERE EXISTS (

   SELECT *
   FROM Employee AS temp
   WHERE temp.EmployeeId = Employee.ReportsTo
   AND temp.BirthDate > Employee.BirthDate
)
```

+ Options

←T→				FirstName
Edit	Inline Edit	Copy	Delete	Nancy
Edit	Inline Edit	Copy	Delete	Margaret
Edit	Inline Edit	Copy	Delete	Robert
Edit	Inline Edit	Copy	Delete	Laura

2. What is the space, in bytes, occupied by the playlist “Grunge", and how much would it cost? (Assume that the cost of a playlist is the sum of the price of its constituent tracks).

```
SELECT SUM( Track.Bytes ) , SUM( Track.UnitPrice )
FROM Track
INNER JOIN PlaylistTrack ON Track.TrackId = PlaylistTrack.TrackId
WHERE PlaylistId =16
```

+ Options

SUM(Track.Bytes)	SUM(Track.UnitPrice)
130886739	14.85

3. Find audio tracks which have a length longer than the average length of all the audio tracks;

Get Average Length for error checking

```
SELECT AVG( Track.Milliseconds )
FROM Track
```

AVG(Track.Milliseconds)
393599.2121

Actual Query for question

```
SELECT Name, Milliseconds
FROM Track
WHERE Track.Milliseconds > (

    SELECT AVG( Track.Milliseconds )
    FROM Track
)
```

+ Options

←T→	Name	Milliseconds
Edit Inline Edit Copy Delete	You Oughta Know (Alternate)	491885
Edit Inline Edit Copy Delete	Master Of Puppets	436453
Edit Inline Edit Copy Delete	Snoopy's search-Red baron	456071
Edit Inline Edit Copy Delete	Stratus	582086
Edit Inline Edit Copy Delete	No More Tears	555075
Edit Inline Edit Copy Delete	Snowblind	420022
Edit Inline Edit Copy Delete	Sleeping Village	644571
Edit Inline Edit Copy Delete	Wheels Of Confusion / The Straightener	494524
Edit Inline Edit Copy Delete	Book Of Thel	494393
Edit Inline Edit Copy Delete	Jerusalem	402390
Edit Inline Edit Copy Delete	The Alchemist	509413
Edit Inline Edit Copy Delete	Stone Crazy	433397
Edit Inline Edit Copy Delete	Talkin' 'Bout Women Obviously	589531
Edit Inline Edit Copy Delete	Terra	482429
Edit Inline Edit Copy Delete	Sozinho (Hitmakers Classic Mix)	436636
Edit Inline Edit Copy Delete	Computadores Fazem Arte	404323
Edit Inline Edit Copy Delete	Dazed and Confused	401920
Edit Inline Edit Copy Delete	You Shook Me(2)	619467
Edit Inline Edit Copy Delete	How Many More Times	711836
Edit Inline Edit Copy Delete	Advance Romance	677694
Edit Inline Edit Copy Delete	Loverman	472764
Edit Inline Edit Copy Delete	Mercyful Fate	671712
Edit Inline Edit Copy Delete	Astronomy	397531
Edit Inline Edit Copy Delete	Tuesday's Gone	545750
Edit Inline Edit Copy Delete	River Song	439510
Edit Inline Edit Copy Delete	Voce Nao Entende Nada - Cotidiano	421982
Edit Inline Edit Copy Delete	Terra	401319
Edit Inline Edit Copy Delete	A E O Z	518556
Edit Inline Edit Copy Delete	Burn	453955
Edit Inline Edit Copy Delete	Mistreated	758648
Edit Inline Edit Copy Delete	Smoke On The Water	618031
Edit Inline Edit Copy Delete	You Fool No One	804101
Edit Inline Edit Copy Delete	In My Time Of Dying	666017

4. Which playlists do not contain any tracks for the artists “Black Sabbath" nor “Chico Buarque"?

```
SELECT DISTINCT Playlist.PlaylistId, Playlist.Name
FROM Playlist
INNER JOIN PlaylistTrack ON Playlist.PlaylistID = PlaylistTrack.PlaylistId
INNER JOIN Track ON Track.TrackId = PlaylistTrack.TrackId
INNER JOIN Album ON Album.AlbumId = Track.AlbumId
INNER JOIN Artist ON Artist.ArtistId = Album.ArtistId
WHERE NOT Artist.Name = "Black Sabbath"
```

AND NOT "Chico Buarque"
ORDER BY PlaylistId

←T→	PlaylistId	Name
Edit Inline Edit Copy Delete	1	Music
Edit Inline Edit Copy Delete	3	TV Shows
Edit Inline Edit Copy Delete	5	90's Music
Edit Inline Edit Copy Delete	8	Music
Edit Inline Edit Copy Delete	9	Music Videos
Edit Inline Edit Copy Delete	10	TV Shows
Edit Inline Edit Copy Delete	11	Brazilian Music
Edit Inline Edit Copy Delete	12	Classical
Edit Inline Edit Copy Delete	13	Classical 101 - Deep Cuts
Edit Inline Edit Copy Delete	14	Classical 101 - Next Steps
Edit Inline Edit Copy Delete	15	Classical 101 - The Basics
Edit Inline Edit Copy Delete	16	Grunge
Edit Inline Edit Copy Delete	17	Heavy Metal Classic
Edit Inline Edit Copy Delete	18	On-The-Go 1

Part 2: Entity Relationship, Relational Modelling and NoSQL solution

A) ER Diagram

Assumptions made in the following ER Diagram:

1) There is only a single stage in the theatre. Based on the sentence “box office staff are able to confirm the play being performed that evening” key words being “the play” singular.

2) Checking “which day of the week is any given date” isn’t within the scope of a database and would be handled by the UI. The primary key of the Performance table Performance_Date would be used to either calculate the day of the week or input into an API.

3) Whether a performance is matinee or evening is determined from the Performance_Date DATETIME in the Performance Table. I.e. if time in the DATETIME is before 17:00 on a given day it is matinee and evening if it is after.

4) “When a caller pays with a credit card, it must be recorded whether they were there in person or on the telephone and an authorization code must be obtained and recorded.” Is taken to mean when a “customer” pays with a credit card since a “caller” could mean a person calling but then it would be pointless to record whether they were in person or on the telephone.

   If a customer is in person and pays with cash the foreign keys Card_ID and Authorisation in Booking Invoice remain NULL

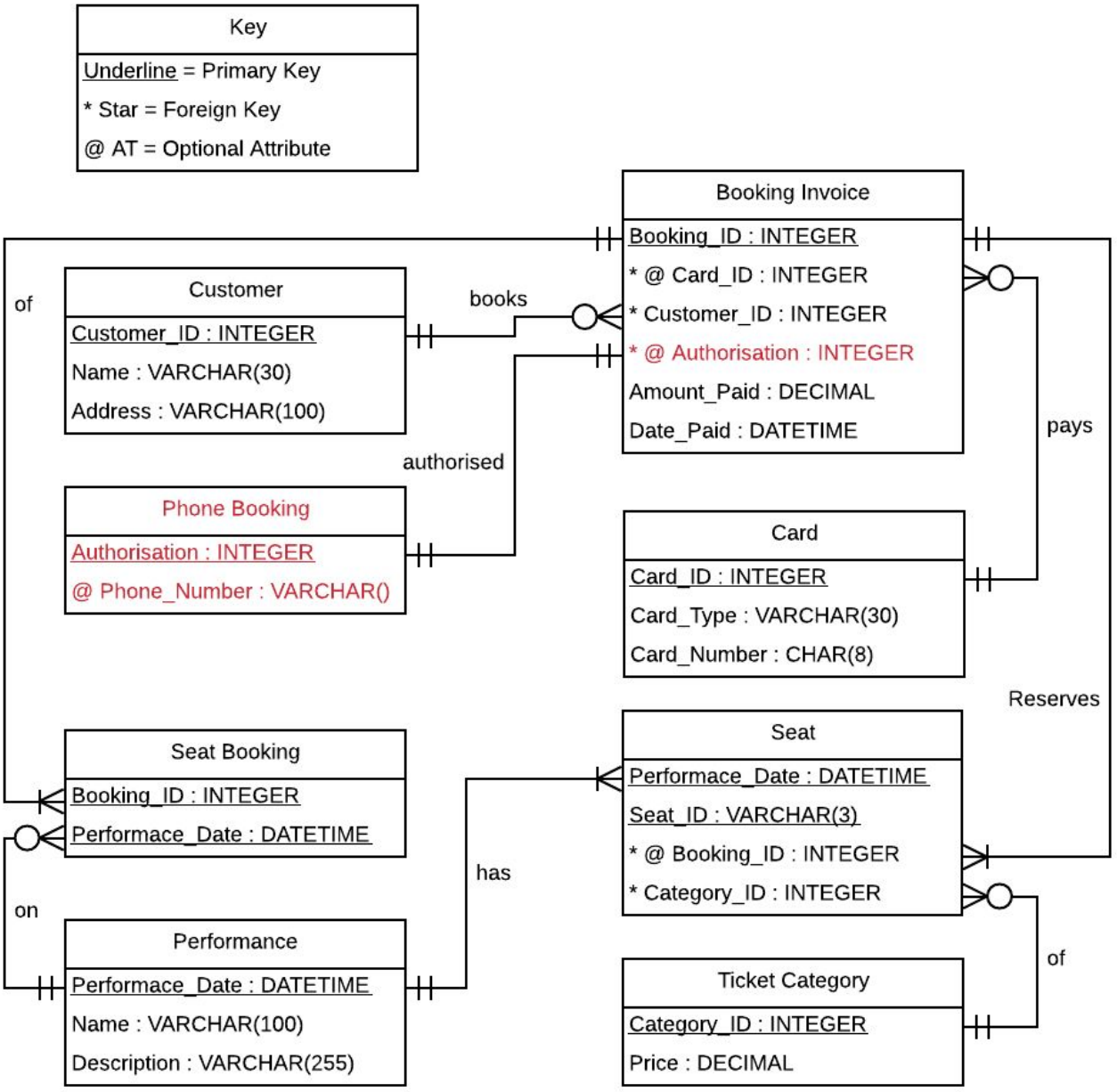
   If a customer is in person and pays with card the foreign key Card_ID, in Booking Invoice, is populated with the Card_ID of the relevant card in the Card Table and Authorisation, in Booking Invoice, remain NULL

   If a customer is on the phone they can not pay in cash so no booking is made

   If a customer is on the phone and pays with card the foreign key Card_ID, in Booking Invoice, is populated with the Card_ID of the relevant card in the Card Table and Authorisation, in Booking Invoice and Phone Booking, is populated with the authorisation obtained. The Phone_Number, in Phone Booking, is optionally recorded to allow contacting the customer.

5) As per Card No in the table for the normalisation section Card_Number in Card Type are assumed to all be 8 digits, possibly beginning with 0. They are thus are stored as CHAR to have a fixed length and maintain the preceding zero.

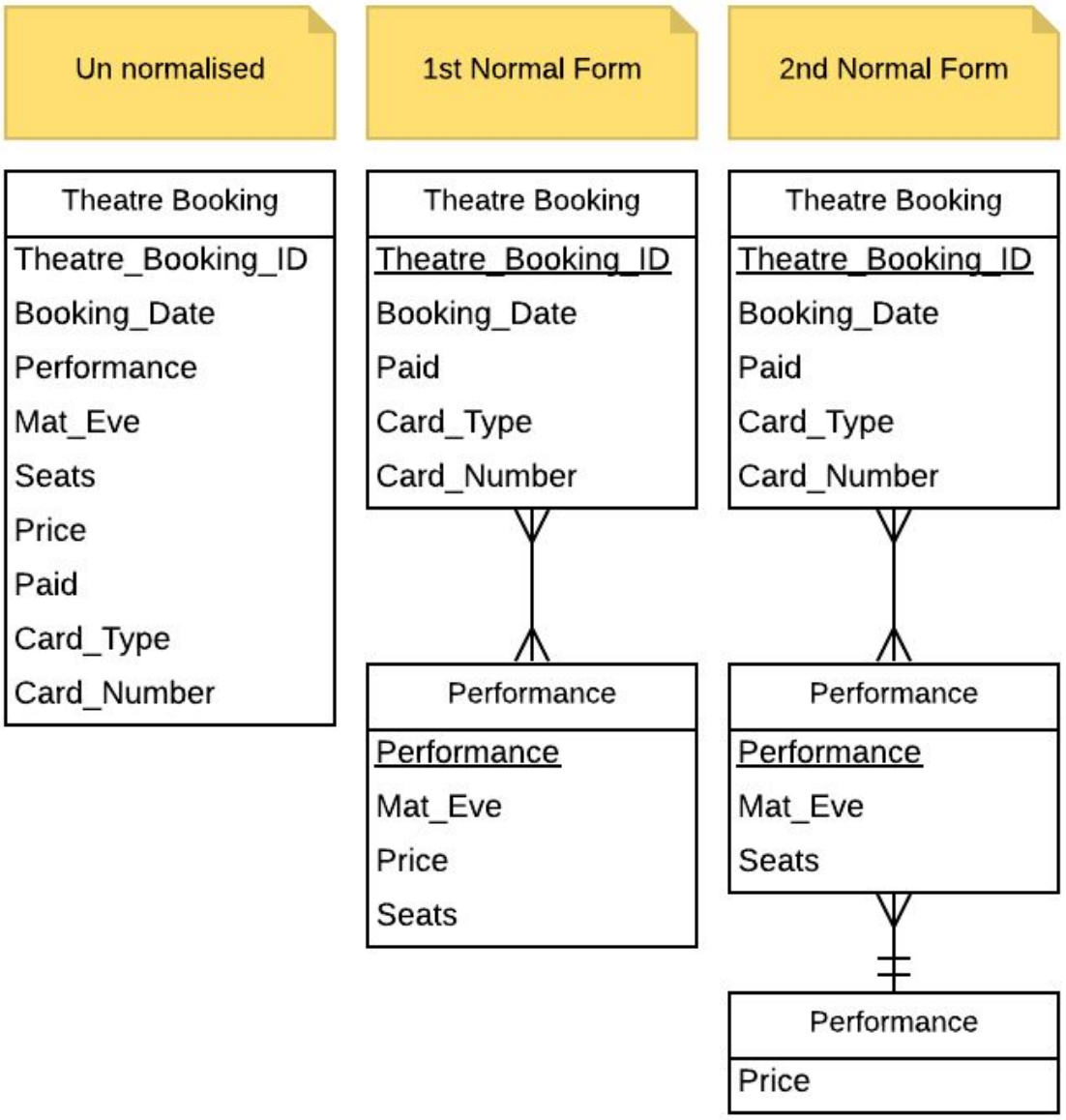
6) A Seat_ID is up to 3 character in a VARCHAR since they are stored as A to Z 0 to 50. The shortest ID is A0 and the longest Z50

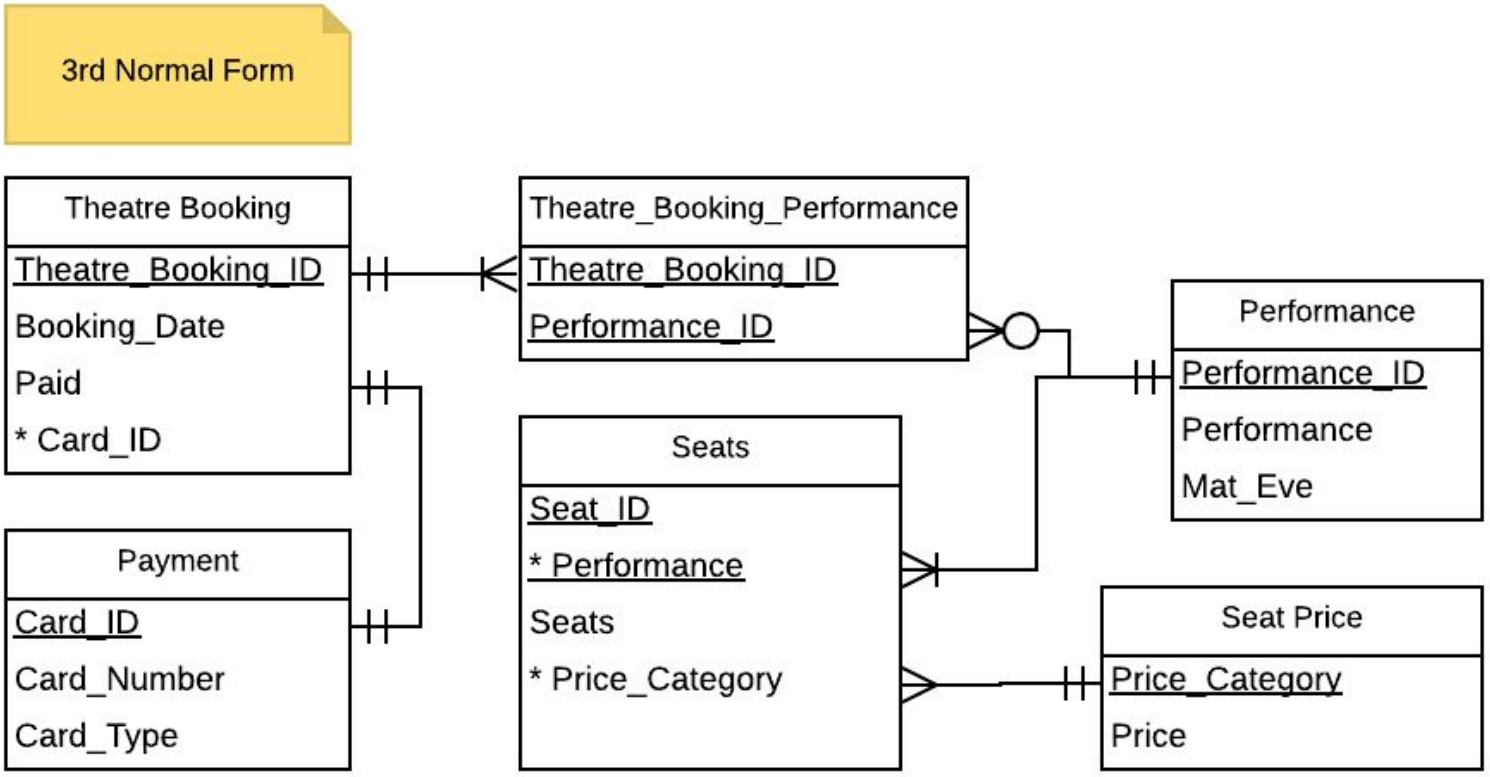

Key
Underline = Primary Key
* Star = Foreign Key
@ AT = Optional Attribute
Booking Invoice
Booking_ID : INTEGER
* @ Card_ID : INTEGER
* Customer_ID : INTEGER
* @ Authorisation : INTEGER
Amount_Paid : DECIMAL
Date_Paid : DATETIME
Customer
Customer_ID : INTEGER
Name : VARCHAR(30)
Address : VARCHAR(100)
books
of
authorised
pays
Phone Booking
Authorisation : INTEGER
@ Phone_Number : VARCHAR()
Card
Card_ID : INTEGER
Card_Type : VARCHAR(30)
Card_Number : CHAR(8)
Reserves
Seat Booking
Booking_ID : INTEGER
Performace_Date : DATETIME
Seat
Performace_Date : DATETIME
Seat_ID : VARCHAR(3)
* @ Booking_ID : INTEGER
* Category_ID : INTEGER
has
on
Performance
Performace_Date : DATETIME
Name : VARCHAR(100)
Description : VARCHAR(255)
of
Ticket Category
Category_ID : INTEGER
Price : DECIMAL

B) Normalisation

Key
Underline = Primary Key
* Star = Foreign Key
@ AT = Optional Attribute

Un-Normalised Data	1st Normal Form	2nd Normal Form	3rd Normal Form
Choose a suitable key and list all data	Separate repeating data into different groups. Groups have keys compound or not (Assuming that two performances can occur on a single day the key must be compound)	Separate into new tables items identified by only part of the key (The price of seats is not related to the performance but the category of the seats themselves)	Separate into new tables items identified by non key items and resolve any many to many relations
Theatre_Booking_ID Booking_Date Performance Mat_Eve Seats Price Paid Card_Type Card_Number	Theatre_Booking_ID Booking_Date Paid Card_Type Card_Number Performance Mat_Eve Price Seats	Theatre_Booking_ID Booking_Date Paid Card_Type Card_Number Performance Mat_Eve Seats Price	Theatre_Booking_ID Booking_Date Paid * Card_ID Card_ID Card_Number Card_Type Performance_ID Theatre_Booking_ID Performance_ID Performance Mat_Eve Seat_ID * Performance seats * Price_Category Price_Category Price


Un normalised
1st Normal Form
2nd Normal Form
Theatre Booking
Theatre_Booking_ID
Booking_Date
Performance
Mat_Eve
Seats
Price
Paid
Card_Type
Card_Number
Theatre Booking
Theatre_Booking_ID
Booking_Date
Paid
Card_Type
Card_Number
Performance
Performance
Mat_Eve
Price
Seats
Theatre Booking
Theatre_Booking_ID
Booking_Date
Paid
Card_Type
Card_Number
Performance
Performance
Mat_Eve
Seats
Performance
Price


3rd Normal Form
Theatre Booking
Theatre_Booking_ID
Booking_Date
Paid
* Card_ID
Theatre_Booking_Performance
Theatre_Booking_ID
Performance_ID
Performance
Performance_ID
Performance
Mat_Eve
Seats
Seat_ID
* Performance
Seats
* Price_Category
Payment
Card_ID
Card_Number
Card_Type
Seat Price
Price_Category
Price

C) Extension to ER Diagram

The extension to allow for telephone booking is highlighted in red in the ER Diagram in section A. The assumption being that details about the booking are entered by the staff member making the call in a similar manner to how detail would be entered if the person was in front of the member of staff. The difference being the detail as highlighted in red are entered.

It is also assumed that the Authorisation code is unique for each booking.

D) NoSQL Solution

<u>Building the Database</u>
The layout for the database was created with test data then that layout was imported using the following javascript file. After running `use Theatre` in the command line then using cd() to navigate to the file location.

```
> use Theatre
switched to db Theatre
> cd("H:/Adv-Database")
> ls()
[
        "./ER.csv",
        "./README.md",
        "./TestInsertion.js",
        "./Theatre Booking OUTDATED.qsee",
        "./Theatre Booking NoSQL.json",
        "./CreateBooking.js",
        "./showCursorOperation.js",
        "./MyEmployee.js",
        "./.git/",
        "./start_mongobd.bat",
        "./TheatreBookingInsert.js",
        "./RetrieveBooking.js",
        "./ER.sql"
]
> load("TheatreBookingInsert.js")
true
>
```

```
db.Booking_Collection.insert(
{
      _id: 1,
      "Customer" : 1,
      "Amount_Paid" : 240,
      "Date_Paid" : "2019-11-27",
      "Card_Type" : "MASTERCARD",
      "Card_Number" : "84108529",
      "Authorisation" : "",
      "Phone_Number" : "",

      "Performances_Booked" :
      [
            {
                  "id" : 1,
                  "Seats" : "A1,A2"
            },
```

```
            {
                  "id" : 2,
                  "Seats" : "C2-C5"
            }
      ]
})

db.Performance_Collection.insert(
[
      {
            _id: 1,
            "Date" : "2019-12-12 11:00:00",
            "Free_Seats" : "A3-Z50",
            "Name" : "Wet Dog",
            "Description" : "A dog gets wet",
            "Mat_Eve" : "Mat"
      },
      {
            _id: 2,
            "Date" : "2019-12-13 18:00:00",
            "Free_Seats" : "A1-B30,C1,C6-Z50",
            "Name" : "Late Night",
            "Description" : "A person stays up late",
            "Mat_Eve" : "Eve"
      },
      {
            _id: 3,
            "Date" : "2019-12-14 12:00:00",
            "Free_Seats" : "A1-Z50",
            "Name" : "Early Morning",
            "Description" : "A person wakes up early",
            "Mat_Eve" : "Mat"
      }
])

db.Categories_Collection.insert(
[
      {
            _id: 1,
            "Begins" : "A",
            "Ends" : "K",
            "Price" : "40",
            "Number_In_Row" : "30"
      },
      {
            _id: 2,
            "Begins" : "L",
            "Ends" : "S",
            "Price" : "30",
            "Number_In_Row" : "40"
      },
      {
```

```
            _id: 3,
            "Begins" : "T",
            "Ends" : "Z",
            "Price" : "25",
            "Number_In_Row" : "50"
      }
])

db.Customer_Collection.insert(
[
      {
            _id: 1,
            "First_Name" : "Nott",
            "Last_Name" : "Abbot",
            "Address" : "23 not a street"

      },
      {
            _id: 2,
            "First_Name" : "Carne",
            "Last_Name" : "Age",
            "Address" : "n/a"
      },
      {
            _id: 3,
            "First_Name" : "Linton",
            "Last_Name" : "Lash",
            "Address" : "123 Street"
      }


])
```

This adds the following collections to the Theatre database

```
> show collections
Booking_Collection
Categories_Collection
Customer_Collection
Performance_Collection
> db.Booking_Collection.find().pretty()
{
        "_id" : 1,
        "Customer" : 1,
        "Amount_Paid" : 240,
        "Date_Paid" : "2019-11-27",
        "Card_Type" : "MASTERCARD",
        "Card_Number" : "84108529",
        "Authorisation" : "",
        "Phone_Number" : "",
        "Performances_Booked" : [
                {
                        "id" : 1,
                        "Seats" : "A1,A2"
                },
                {
                        "id" : 2,
                        "Seats" : "C2-C5"
                }
        ]
}
> db.Performance_Collection.find().pretty()
{
        "_id" : 1,
        "Date" : "2019-12-12 11:00:00",
        "Free_Seats" : "A3-Z50",
        "Name" : "Wet Dog",
        "Description" : "A dog gets wet",
        "Mat_Eve" : "Mat"
}
{
        "_id" : 2,
        "Date" : "2019-12-13 18:00:00",
        "Free_Seats" : "A1-B30,C1,C6-Z50",
        "Name" : "Late Night",
        "Description" : "A person stays up late",
        "Mat_Eve" : "Eve"
}
{
        "_id" : 3,
        "Date" : "2019-12-14 12:00:00",
        "Free_Seats" : "A1-Z50",
        "Name" : "Early Morning",
        "Description" : "A person wakes up early",
        "Mat_Eve" : "Mat"
}
```

```
> db.Categories_Collection.find().pretty()
{
        "_id" : 1,
        "Begins" : "A",
        "Ends" : "K",
        "Price" : "40",
        "Number_In_Row" : "30"
}
{
        "_id" : 2,
        "Begins" : "L",
        "Ends" : "S",
        "Price" : "30",
        "Number_In_Row" : "40"
}
{
        "_id" : 3,
        "Begins" : "T",
        "Ends" : "Z",
        "Price" : "25",
        "Number_In_Row" : "50"
}
> db.Customer_Collection.find().pretty()
{
        "_id" : 1,
        "First_Name" : "Nott",
        "Last_Name" : "Abbot",
        "Address" : "23 not a street"
}
{
        "_id" : 2,
        "First_Name" : "Carne",
        "Last_Name" : "Age",
        "Address" : "n/a"
}
{
        "_id" : 3,
        "First_Name" : "Linton",
        "Last_Name" : "Lash",
        "Address" : ""
}
```

<u>Creating a Booking</u>

The exact process would be implementation specific but would involve a script that constructs the following:

```
db.Booking_Collection.insert(
{
      _id: 2,
      "Customer" : 4,
      "Amount_Paid" : 400,
      "Date_Paid" : "2019-10-20",
      "Card_Type" : "VISA",
      "Card_Number" : "92219013",
      "Authorisation" : "201",
      "Phone_Number" : "01664012012",

      "Performances_Booked" :
      [
            {
                  "id" : "2019-12-14 12:00:00",
```

```
                "Seats" : "A1,A2,C3-C5"
            }
        ]
})
```

The booking process would involve retrieving details like available seats and their prices to inform the customer and to calculate the total price of the booking..

It would also involve searching the Customer_Collection to determine whether the customer already exists or not to determine whether they should be added to the list. Also to find the highest customer _id to increment it to be used as the _id.

In this case the customer is new so they are added to the Customer_Collection

```
db.Customer_Collection.insert(
{
      _id: 4,
      "First_Name" : "Clinton",
      "Last_Name" : "East",
      "Address" : "West"
})
```

Retrieving a Booking

To find a single booking the following command can be used

```
db.Booking_Collection.find({_id : 1}).pretty()
```

```
> db.Booking_Collection.find({_id : 1}).pretty()
{
        "_id" : 1,
        "Customer" : 1,
        "Amount_Paid" : 240,
        "Date_Paid" : "2019-11-27",
        "Card_Type" : "MASTERCARD",
        "Card_Number" : "84108529",
        "Authorisation" : "",
        "Phone_Number" : "",
        "Performances_Booked" : [
                {
                        "id" : 1,
                        "Seats" : "A1,A2"
                },
                {
                        "id" : 2,
                        "Seats" : "C2-C5"
                }
        ]
}
```

To retrieve all bookings made by a certain customer:

```
db.Booking_Collection.find({"Customer" : 4}).pretty()
```

```
> db.Booking_Collection.find({"Customer" : 4}).pretty()
{
        "_id" : 2,
        "Customer" : 4,
        "Amount_Paid" : 400,
        "Date_Paid" : "2019-10-20",
        "Card_Type" : "VISA",
        "Card_Number" : "92219013",
        "Authorisation" : "201",
        "Phone_Number" : "01664012012",
        "Performances_Booked" : [
                {
                        "id" : 3,
                        "Seats" : "A1,A2,C3-C5"
                }
        ]
}
{
        "_id" : 3,
        "Customer" : 4,
        "Amount_Paid" : 400,
        "Date_Paid" : "TBD",
        "Card_Type" : "TBD",
        "Card_Number" : "TBD",
        "Authorisation" : "TBD",
        "Phone_Number" : "TBD",
        "Performances_Booked" : [
                {
                        "id" : 3,
                        "Seats" : "A1,A2,C3-C5"
                }
        ]
}
{
        "_id" : 4,
        "Customer" : 4,
        "Amount_Paid" : 400,
        "Date_Paid" : "TBD",
        "Card_Type" : "TBD",
        "Card_Number" : "TBD",
        "Authorisation" : "TBD",
        "Phone_Number" : "TBD",
        "Performances_Booked" : [
                {
                        "id" : 2,
                        "Seats" : "A1,A2,C3-C5"
                }
        ]
}
```

To retrieve all bookings for a certain performance

```
db.Booking_Collection.find({"Performances_Booked.id" : 2}).pretty()
```

```
> db.Booking_Collection.find({"Performances_Booked.id" : 2}).pretty()
{
        "_id" : 1,
        "Customer" : 1,
        "Amount_Paid" : 240,
        "Date_Paid" : "2019-11-27",
        "Card_Type" : "MASTERCARD",
        "Card_Number" : "84108529",
        "Authorisation" : "",
        "Phone_Number" : "",
        "Performances_Booked" : [
                {
                        "id" : 1,
                        "Seats" : "A1,A2"
                },
                {
                        "id" : 2,
                        "Seats" : "C2-C5"
                }
        ]
}
{
        "_id" : 4,
        "Customer" : 4,
        "Amount_Paid" : 400,
        "Date_Paid" : "TBD",
        "Card_Type" : "TBD",
        "Card_Number" : "TBD",
        "Authorisation" : "TBD",
        "Phone_Number" : "TBD",
        "Performances_Booked" : [
                {
                        "id" : 2,
                        "Seats" : "A1,A2,C3-C5"
                }
        ]
}
```

To retrieve customer, performance, and/or pricing details, say to be shown on an invoice, would again be implementation specific but would involve retrieving the relevant booking in Booking_Collection then using the id's to retrieve the corresponding items from Customer_Collection, Performance_Collection, and Categories_Collection

E) Compare and Contrast Reflection

Creating the relational solution before the non-relational one meant that determining important information for the situation was already completed and the largest work was determining how to order that information into a form that fully utilised the benefits of a non-relational database. This involved deciding which sections of the data belong in their own separate collections and which could simply be embedded documents.

In the non-relational solution the elements relating to payment card and authorisation of card payments over phone bookings are embedded documents because the degree of duplicated data is minimal and by doing so it reduces the need for the complex application code that is needed to resolve the relational queries.

The relational solution keeps data duplication to a minimum since it is fully normalised however in doing so it complicates the queries that are required to access that data.

Given that there are one thousand seats in the theatre, (11 * 30) + (8 * 40) + (7 * 50) = 1000, the relational solution introduces another thousand rows in the seats table for every performance. This allows for the categories of seats to be assigned and their relation to performances and bookings to be resolved. However this is likely to be a cause of performance issued when querying the table. In contrast the non-relational solution stores which seats are free and which are booked in separate items in the database as a string of comma separated values where single seats are stored as their assigned letter and number and a range of seats without any gaps is stored by the first and last values. For example a customer wants to book seats 1 , 3,4,5,6 , 29,30 in row A and seats 1,2,3 in row B.
That would be stored as A1 , A3-A6 , A29-30, B1-B3

While doing so compresses a thousand rows into two values it adds an extra level of complexity in regards to retrieving and utilising the data in the application layer. It also means that all bookings must access the same value to update which seats are booked which could cause issues if a queue is formed leading to a slow response to the application making the request. This would not occur in the relational solution since each seat on a performance can be accessed separately.

In conclusion I am unsure if my solutions fully demonstrate the advantages and disadvantages of either method however the work involved in solving the same problem with different methods did help to contrast the differences and similarities between the two.