# Analysis

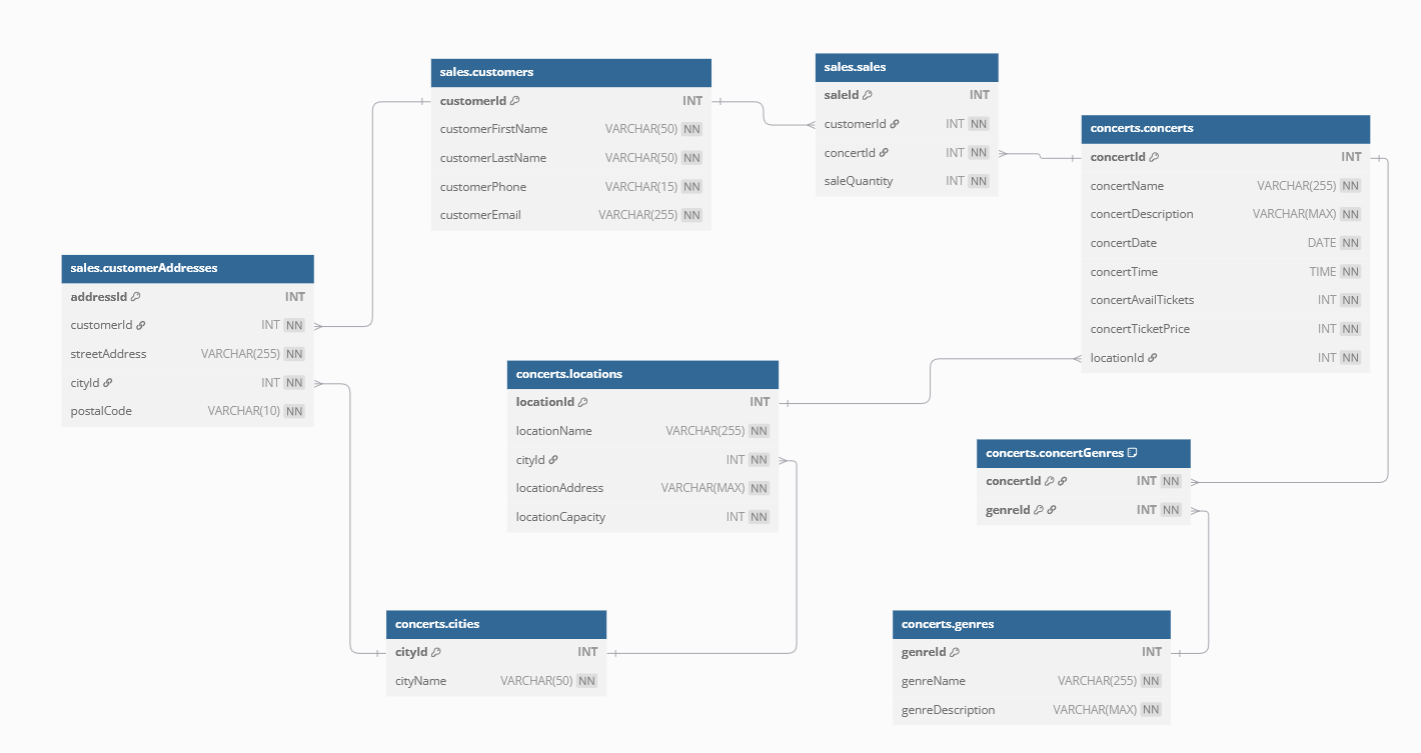
## Relevant Implications:

Future proofing – My database is normalized to the 4th degree, which makes it easy to update and expand on, adding more tables or schemas with minimal disruption to existing data. Many-to-many relationships, such as with tblConcertGenres allows for additional genres to be added as the music scene develops and grows. Other tables, such as tblCities are isolated to allow for easy editing and addition of cities.  
Privacy – My database contains a lot of private information, such as names, email addresses, phone numbers and addresses. Nothing about the database is currently secured, but this could be fixed by encrypting data and adding firewalls, or adding administrative controls, so that the people who are interacting with the database have just enough permissions to do what is necessary for them.

# Design

## Database Storage Design:

### ERD:



### SQL DDL:

## Table Design:

### Concerts Schema

#### tblCities

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Example** |
| cityId | INT (PK) | Unique identifier for the city | 1 |
| cityName | VARCHAR(50) | Name of the city | Auckland |

#### tblLocations

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Example** |
| locationId | INT (PK) | Unique identifier for the location | 1 |
| locationName | VARCHAR(255) | Name of the location | Spark Arena |
| cityId | INT (FK) | References tblCities (cityId) | 1 |
| locationAddress | TEXT | Address of the location | 42-80 Mahuhu Crescent, Parnell |
| locationCapacity | INT | Maximum capacity of the location | 12000 |

#### tblGenres

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Example** |
| genreId | INT (PK) | Unique identifier for the genre | 1 |
| genreName | VARCHAR(255) | Name of the genre | Jazz |
| genreDescription | TEXT | Description of the genre | A diverse genre including traditional, smooth, and fusion jazz. |

#### tblConcerts

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Example** |
| concertId | INT (PK) | Unique identifier for the concert | 1 |
| concertName | VARCHAR(255) | Name of the concert | Russell Peters - Relax World Tour |
| concertDescription | TEXT | Description of the concert | Global stand-up sensation with hilarious new material. |
| concertDate | DATE | Date of the concert | 2025-03-18 |
| concertTime | TIME | Time of the concert | 20:00:00 |
| concertAvailTickets | INT | Number of available tickets | 20:00:00 |
| concertTicketPrice | SMALLMONEY | Price of one ticket | 79.99 |
| locationId | INT (FK) | References tblLocations (locationId) | 1 |

#### tblConcertGenres

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Example** |
| concertId | INT FK | References tblConcerts (concertId) | 1 |
| genreId | INT FK | References tblGenres (genreId) | 1 |

### Sales Schema

#### tblCustomers

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Example** |
| customerId | INT (PK) | Unique identifier for the customer | 1 |
| customerFirstName | VARCHAR(50) | First name of the customer | Andrew |
| customerLastName | VARCHAR(50) | Last name of the customer | King |
| customerPhone | VARCHAR(15) | Contact phone number | +64 292660267 |
| customerEmail | VARCHAR(255) | Email address of the customer | agking541@outlook.com |

#### tblCustomerAddresses

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Example** |
| addressId | INT (PK) | Unique identifier for the address | 1 |
| customerId | INT (FK) | References tblCustomers (customerId) | 1 |
| streetAddress | VARCHAR(255) | Street address of the customer | 40 Parker Avenue, New Lynn |
| cityId | INT (FK) | References tblCities (cityId) | 1 |
| postalCode | VARCHAR(10) | Postal code of the customer's address | 0600 |

#### tblSales

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Exmple** |
| saleId | INT (PK) | Unique identifier for the sale | 1 |
| customerId | INT (FK) | References tblCustomers (customerId) | 1 |
| concertId | INT (FK) | References tblConcerts (concertId) | 1 |
| saleQuantity | INT | Number of tickets sold | 1 |

### SQL DDL for the Tables:

CREATE SCHEMA concerts;

GO

CREATE SCHEMA sales;

GO

CREATE TABLE concerts.tblCities (

cityId INT IDENTITY (1, 1) PRIMARY KEY,

cityName VARCHAR (50) NOT NULL

);

CREATE TABLE concerts.tblLocations (

locationId INT IDENTITY (1, 1) PRIMARY KEY,

locationName VARCHAR (255) NOT NULL,

cityId INT NOT NULL,

locationAddress TEXT NOT NULL,

locationCapacity INT NOT NULL,

FOREIGN KEY (cityId) REFERENCES concerts.tblCities (cityId) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE concerts.tblGenres (

genreId INT IDENTITY (1, 1) PRIMARY KEY,

genreName VARCHAR (255) NOT NULL,

genreDescription TEXT NOT NULL

);

CREATE TABLE concerts.tblConcerts (

concertId INT IDENTITY (1, 1) PRIMARY KEY,

concertName VARCHAR (255) NOT NULL,

concertDescription TEXT NOT NULL,

concertDate DATE NOT NULL,

concertTime TIME NOT NULL,

concertAvailTickets INT NOT NULL,

concertTicketPrice SMALLMONEY NOT NULL,

locationId INT NOT NULL,

FOREIGN KEY (locationId) REFERENCES concerts.tblLocations (locationId) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE concerts.tblConcertGenres (

concertId INT NOT NULL,

genreId INT NOT NULL,

PRIMARY KEY (concertId, genreId),

FOREIGN KEY (concertId) REFERENCES concerts.tblConcerts (concertId) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (genreId) REFERENCES concerts.tblGenres (genreId) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE sales.tblCustomers (

customerId INT IDENTITY (1, 1) PRIMARY KEY,

customerFirstName VARCHAR (50) NOT NULL,

customerLastName VARCHAR (50) NOT NULL,

customerPhone VARCHAR (15) NOT NULL,

customerEmail VARCHAR (255) NOT NULL

);

CREATE TABLE sales.tblCustomerAddresses (

addressId INT IDENTITY (1, 1) PRIMARY KEY,

customerId INT NOT NULL,

streetAddress VARCHAR (255) NOT NULL,

cityId INT NOT NULL,

postalCode VARCHAR (10) NOT NULL,

FOREIGN KEY (customerId) REFERENCES sales.tblCustomers (customerId) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (cityId) REFERENCES concerts.tblCities (cityId) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE sales.tblSales (

saleId INT IDENTITY (1, 1) PRIMARY KEY,

customerId INT NOT NULL,

concertId INT NOT NULL,

saleQuantity INT NOT NULL,

FOREIGN KEY (customerId) REFERENCES sales.tblCustomers (customerId) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (concertId) REFERENCES concerts.tblConcerts (concertId) ON DELETE CASCADE ON UPDATE CASCADE

);

## Query Design:

### Simple

#### Query Name: gryViewAllCities

The query qryViewAllCities is a query which selects and displays all columns in the tblCities.

**Tables involved:** tblCities

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT cityId AS [City ID],

cityName AS [City Name]

FROM concerts.tblCities;

**Result Output:**

**A black and white rectangular sign with white text

AI-generated content may be incorrect.**

#### Query Name: qryViewAllCustomerAddresses

The query qryViewAllCustomerAddresses is a query which selects and displays all the columns in tblCustomerAddresses.

**Tables involved:** tblCustomerAddresses

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT streetAddress AS [Street Address],

postalCode AS [Postal Code],

cityId AS [City ID]

FROM sales.tblCustomerAddresses;

**Result Output:  
A screenshot of a computer screen

AI-generated content may be incorrect.**

#### Query Name: qryViewAllGenres

The query qryViewAllGenres is a query which selects and displays all the columns in the table tblGenres.

**Tables involved:** tblGenres

**Result:** The query was able to display the records that met the criteria.

**SQL:**

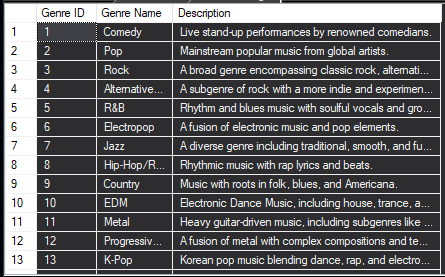
SELECT genreId AS [Genre ID],

genreName AS [Genre Name],

genreDescription AS [Description]

FROM concerts.tblGenres;

**Result Output:**



#### Query Name: qryViewAllVenues

The query qryViewAllVenues is a query which selects and displays the columns locationId, locationName, locationCapacity, locationAddress from tblLocations and cityName from tblCities. It uses Left outer join to get all the concerts, even if the city has been returned as null.

**Tables involved:** tblLocations, tblCities

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT l.locationId AS [Location ID],

l.locationName AS [Venue Name],

l.locationCapacity AS [Venue Capacity],

l.locationAddress AS [Venue Address],

ct.cityName AS [Venue City]

FROM concerts.tblLocations AS l

LEFT OUTER JOIN

concerts.tblCities AS ct

ON l.cityId = ct.cityId;

**Result Output:   
A black and white table with white text

AI-generated content may be incorrect.**

#### Query Name: qryViewConcertsPerDate

The query qryViewConcertsPerDate is a query which selects the names, dates and times of concerts, and displays ones which are on the specified date. It is designed to return data based on the criteria given. The CONVERT function is used to convert the concertTime to hh:mm:ss time as a string, and then get the first 5 chars of the string.

**Tables involved: tblConcerts**

**Result:** The query was able to display the records that met the criteria.

**SQL:**DECLARE @date AS DATE = '2025-06-15';

SELECT concertName AS [Concert Name],

concertDate AS [Date],

CONVERT (VARCHAR (5), concertTime, 108) AS [Time]

FROM concerts.tblConcerts

WHERE concertDate = @date;

**Result Output:  
**

### Advanced

#### Query Name: qryViewAllCustomers

The query qryViewAllCustomers is a query which selects and displays all customers’ IDs, first and last names, phone number, email from tblCustomers, street address and postal code from the table tblCustomerAddresses, and city from tblCitites. Left outer join is used to still display a customer if either the address or city returns null.

**Tables involved:** tblCustomers, tblCustomerAddresses, tblCities

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT cs.customerId AS [Customer ID],

cs.customerFirstName AS [First Name],

cs.customerLastName AS [Last Name],

cs.customerPhone AS [Phone],

cs.customerEmail AS [Email],

ca.streetAddress AS [Street Address],

ct.cityName AS [City],

ca.postalCode AS [Postal Code]

FROM sales.tblCustomers AS cs

LEFT OUTER JOIN

sales.tblCustomerAddresses AS ca

ON cs.customerId = ca.customerId

LEFT OUTER JOIN

concerts.tblCities AS ct

ON ct.cityId = ca.cityId;

**Result Output:  
A screenshot of a computer

AI-generated content may be incorrect.**

#### Query Name: qryViewAllSales

The query qryViewAllSales is a query which selects and displays all sales’ IDs, customers’ first and last names, the concert they bought tickets to, and how many tickets they bought.

**Tables involved:** tblSales, tblCustomers, tblConcerts

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT s.saleId AS [Sale ID],

cust.customerFirstName AS [First Name],

cust.customerLastName AS [Last Name],

c.concertName AS [Concert],

s.saleQuantity AS [Tickets Sold]

FROM sales.tblSales AS s

INNER JOIN

sales.tblCustomers AS cust

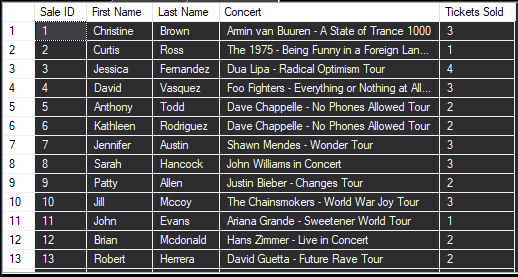
ON s.customerId = cust.customerId

INNER JOIN

concerts.tblConcerts AS c

ON s.concertId = c.concertId

ORDER BY s.saleId;

**Result Output:  
**

#### Query Name: qryViewConcertsInGenre

The query qryViewConcertsInGenre is a query which selects all concerts’ IDs, names, and genres and displays the ones which are in the specified genre. It is designed to return data based on the criteria given.

**Tables involved:** tblConcerts, tblConcertGenres, tblGenres

**Result:** The query was able to display the records that met the criteria.

**SQL:**DECLARE @genre AS VARCHAR (MAX) = 'Rock';

SELECT c.concertId AS [Concert ID],

c.concertName AS [Concert Name],

g.genreName AS [Genre]

FROM concerts.tblConcerts AS c

INNER JOIN

concerts.tblConcertGenres AS cg

ON c.concertId = cg.concertId

INNER JOIN

concerts.tblGenres AS g

ON cg.genreId = g.genreId

WHERE g.genreName = @genre;

**Result Output:  
A screenshot of a concert list

AI-generated content may be incorrect.**

#### Query Name: qryViewConcertsPerCity

The query qryViewConcertsPerCity is a query which selects all concerts’ IDs, names, venue names and the city its in, then displays all the concerts in a specific city. It is designed to return data based on the criteria given. The CAST function converts the cityId, which is stored as an integer, to a string so it can match the data type of the variable.

**Tables involved:** tblConcerts, tblLocations, tblCities

**Result:** The query was able to display the records that met the criteria.

**SQL:**DECLARE @CityName AS VARCHAR (50) = 'City';

SELECT c.concertId AS [Concert ID],

c.concertName AS [Concert Name],

l.locationName AS [Venue Name],

ct.cityName AS [City]

FROM concerts.tblConcerts AS c

INNER JOIN

concerts.tblLocations AS l

ON c.locationId = l.locationId

INNER JOIN

concerts.tblCities AS ct

ON l.cityId = ct.cityId

WHERE ct.cityName = @CityName

OR CAST (ct.cityId AS VARCHAR) = @CityName

ORDER BY l.locationName;

**Result Output:  
**

#### Query Name: qryViewUpcomingConcerts

The query qryViewUpcomingConcerts is a query which selects all the concerts’ names, dates, and venue name, and displays the ones which have a date between the current date to 30 days in the future. The where clause uses the between operator, to get any dates which are between the current date (fetched by the GETDATE function) and 30 days in the future (with the GETDATE and DATEADD functions, adding in days, 30, to the current date).

**Tables involved:** tblConcerts, tblLocations

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT concertName AS [Concert Name],

concertDate AS [Date],

locationName AS [Venue Name]

FROM concerts.tblConcerts

INNER JOIN

concerts.tblLocations

ON concerts.tblConcerts.locationId = concerts.tblLocations.locationId

WHERE concertDate BETWEEN GETDATE() AND DATEADD(DAY, 30, GETDATE());

**Result Output:  
A screenshot of a computer

AI-generated content may be incorrect.**

### Complex

#### Query Name: qryMostPopularVenues

The query qryMostPopularVenues is a query which selects and displays the venue name, and the total tickets sold in that venue, then orders it by the total tickets sold. The SUM is an aggregate function which gathers all the data in the specified column and adds it together. You must then group by all other non-aggregate columns to make sure there is only unique values paired to the aggregate values.

**Tables involved:** tblSales, tblConcerts, tblLocations

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT l.locationName AS [Venue Name],

SUM(s.saleQuantity) AS [Total Tickets Sold]

FROM sales.tblSales AS s

INNER JOIN

concerts.tblConcerts AS c

ON s.concertId = c.concertId

INNER JOIN

concerts.tblLocations AS l

ON c.locationId = l.locationId

GROUP BY l.locationName

ORDER BY [Total Tickets Sold] DESC;

**Result Output:  
A screenshot of a computer screen

AI-generated content may be incorrect.**

#### Query Name: qrySearchCustomers

The query qrySearchCustomers is a query which selects all the customers’ IDs, full names, phone numbers, email addresses, and full addresses, then displays the customers which have a name, phone number, email address, or ID which matches the given search. It is designed to return data based on the criteria given. This query uses the CONCAT function to join the customer’s first and last names, seperated by a space to get the customers’ full names. It also uses the same function to concatenate the street address, city name, and postal code, all seperated by a comma and space to give the full, properly formatted address. It also uses LEFT OUTER JOIN so it still displays customers who’s addresses or cities have turned up as NULL. The where statement filters out customers, when they only have one of either their name, phone number, email address, or ID that match the search criteria. It does this by assessing each value, then converting it’s boolean value with the help of a CASE statement to an integer, 1 for true and 0 for false. It then adds these all up, and then if the total adds to one, it displays the customer. This avoids repeats, if their name is exactly the same as their email, or their phone number is the same as their ID.

**Tables involved:** tblCustomers, tblCustomerAddresses, tblCities

**Result:** The query was able to display the records that met the criteria.

**SQL:**DECLARE @search AS VARCHAR (MAX) = NULL;

SELECT cs.customerId AS [Customer ID],

CONCAT(cs.customerFirstName, ' ', cs.customerLastName) AS [Full Name],

cs.customerPhone AS [Phone],

cs.customerEmail AS [Email],

CONCAT(ca.streetAddress, ', ', ct.cityName, ', ', ca.postalCode) AS [Address]

FROM sales.tblCustomers AS cs

LEFT OUTER JOIN

sales.tblCustomerAddresses AS ca

ON cs.customerId = ca.customerId

LEFT OUTER JOIN

concerts.tblCities AS ct

ON ct.cityId = ca.cityId

WHERE (

CASE WHEN CAST(cs.customerId AS VARCHAR) = @search THEN 1 ELSE 0 END +

CASE WHEN cs.customerFirstName = @search

OR cs.customerLastName = @search

OR CONCAT(cs.customerFirstName, ' ', cs.customerLastName) = @search THEN 1 ELSE 0 END +

CASE WHEN cs.customerPhone = @search THEN 1 ELSE 0 END +

CASE WHEN cs.customerEmail = @search THEN 1 ELSE 0 END

) = 1;

**Result Output:  
**

#### Query Name: qryViewAllConcerts

The query qryViewAllConcerts is a query which selects and displays all the concerts’ IDs, names, descriptions, genres, dates and times, available tickets, and the names and addresses of the venues that its being held in. This query uses the STRING\_AGG function to get all the genres that the concert is, and combine them with a comma and a space to avoid repeated rows with different genres only. It then must group by the rest of the non-aggregate columns to make sure there are only unique values paired to the aggregate values. The CONVERT function is used to convert the concertTime to hh:mm:ss time as a string, and then get the first 5 chars of the string. The CONCAT function is used to get the full address, with the street address, then the city concatenated with a comma and space between, for ease of data viewing. It uses LEFT OUTER JOIN, so that if it returns null of the joined values, then it still displays the row.

**Tables involved:** tblConcerts, tblConcertGenres, tblGenres, tblLocations, tblCities

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT c.concertId AS [Concert ID],

c.concertName AS [Concert Name],

c.concertDescription AS [Description],

STRING\_AGG(g.genreName, ', ') AS [Genres],

c.concertDate AS [Date],

CONVERT (VARCHAR (5), c.concertTime, 108) AS [Time],

c.concertAvailTickets AS [Available Tickets],

c.concertTicketPrice AS [Ticket Price],

l.locationName AS [Venue Name],

CONCAT(l.locationAddress, ', ', ct.cityName) AS [Venue]

FROM concerts.tblConcerts AS c

LEFT OUTER JOIN

concerts.tblConcertGenres AS cg

ON c.concertId = cg.concertId

LEFT OUTER JOIN

concerts.tblGenres AS g

ON cg.genreId = g.genreId

LEFT OUTER JOIN

concerts.tblLocations AS l

ON c.locationId = l.locationId

LEFT OUTER JOIN

concerts.tblCities AS ct

ON l.cityId = ct.cityId

GROUP BY c.concertId, c.concertName, c.concertDescription,

c.concertDate, c.concertTime, c.concertAvailTickets,

c.concertTicketPrice, l.locationName, l.locationAddress,

ct.cityName;

**Result Output:  
A screen shot of a computer

AI-generated content may be incorrect.**

#### Query Name: qryViewRevenuePerConcert

The query qryViewRevenuePerConcert is a query which selects the all the concerts’ names, and the sum of all their ticket sale prices, then displays them all with the overall sum and total at the top. The COALESCE function is used in conjunction with the ROLLUP sub-clause of GROUP BY which in turn is used in conjunction to the aggregate function SUM. ROLLUP is used to aggregate the grand total of all the revenue. It takes the column specified, which in this case is concerts.tblConcerts.concertName, then displays the total of first all the concertNames’ revenue sums, then displays the total sum of all the concertNames’ revenue sums with a concertName of NULL. The COALESCE function chooses the first non-NULL value, so when ROLLUP displays the overall total with the concertName of NULL, it chooses the next non-NULL value in the COALESCSE function which is set to be '\*\*\*Overall Total\*\*\*'. It then uses ORDER BY, which sorts by the values in the ‘Total Revenue’ column to have the largest at the top, which will be the overall total, then all the ones below from biggest to smallest.

**Tables involved:** tblSales, tblConcerts

**Result:** The query was able to display the records that met the criteria.

**SQL:**SELECT COALESCE(c.concertName, '\*\*\*Overall Total\*\*\*') AS [Concert Name],

SUM(s.saleQuantity \* c.concertTicketPrice) AS [Total Revenue]

FROM sales.tblSales AS s

INNER JOIN concerts.tblConcerts AS c

ON s.concertId = c.concertId

GROUP BY ROLLUP(c.concertName)

ORDER BY [Total Revenue] DESC;

**Result Output:  
A screenshot of a concert list

AI-generated content may be incorrect.**

#### Query Name: qryViewSalesPerConcert

The query qryViewSalesPerConcert is a query which selects all the sales’ IDs, customer names, concert name, sale quantity, ticket price, and the total sale price, then displays the records which concert IDs are the same as the provided ID. It is designed to return data based on the criteria given. This query uses the CONCAT function to join the customer’s first and last names, seperated by a space to get the customers’ full names.

**Tables involved:** tblSales, tblCustomers, tblConcerts

**Result:** The query was able to display the records that met the criteria.

**SQL:**DECLARE @concertId AS INT = 2;

SELECT s.saleId AS [Sale ID],

CONCAT(cs.customerFirstName, ' ', cs.customerLastName) AS [Full Name],

c.concertName AS [Concert Name],

s.saleQuantity AS [Sale Quantity],

c.concertTicketPrice AS [Ticket Price],

(s.saleQuantity \* c.concertTicketPrice) AS [Total Sale Price]

FROM sales.tblSales AS s

INNER JOIN

sales.tblCustomers AS cs

ON s.customerId = cs.customerId

INNER JOIN

concerts.tblConcerts AS c

ON s.concertId = c.concertId

WHERE c.concertId = @concertId

ORDER BY [Sale ID] ASC;

**Result Output:  
A table with text and numbers

AI-generated content may be incorrect.**

## Iterative Improvement:

  
I had to figure out that I need to use a group by statement along with any combining aggregate functions. I fixed it by adding this statement -A screen shot of a computer

AI-generated content may be incorrect.

In the table tblSearchCustomers, I was getting an error where it wouldn’t return the records I wanted when I searched for the customer’s full name, but I fixed it by adding this line – which makes it so it also searches for when the customer’s first name and then last name have been entered, when seperated by a space.

