CSCU9B3 Relational Database Assignment 2024 - Stirling BEaT Case Study

Database Design

Musicians\_Table:

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| musician\_ID | INT (PK, 4) | Unique identifier for each musician |
| name | VARCHAR (30) | Full name of the musician |
| birth\_date | DATE | Date of birth of each musician |
| experience | VARCHAR (20) | Experience level (e.g., "Professional", "Amateur") |

Bands\_Table:

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| band\_ID | INT (PK, 2) | Unique identifier for each band |
| band\_name | VARCHAR (50) | Name of the band |
| genre | VARCHAR (20) | Genre of the band's music (e.g., Jazz, Classical) |
| formation\_year | YEAR | Year the band was formed |
| manager\_phone | VARCHAR (15) | Contact phone number for the band manager |
| manager\_email | VARCHAR (30) | Email address for the bands manager |

Musician\_Band\_Table(Junction Table):

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| musician\_ID | INT (FK, 4) | Unique identifier for each musician |
| band\_ID | INT (FK, 2) | Unique identifier for each band |

Events\_Table:

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| event\_ID | INT (PK, 4) | Unique identifier for each event |
| event\_date | DATE | Date of the event |
| theme | VARCHAR (50) | Theme of the event (e.g., Jazz Night) |
| audience\_size | INT (4) | Number of attendees |
| venue\_ID | INT (FK, 1) | Foreign key referencing venues |

Venue\_Table

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| venue\_ID | INT (PK, 1) | Unique identifier for each venue |
| venue\_name | VARCHAR (20) | Name of the venue |

Band\_Performance\_Table:

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| performance\_ID | INT (PK, 4) | Unique identifier for each performance |
| band\_ID | INT (FK, 4) | Foreign key referencing bands |
| event\_ID | INT (FK, 4) | Foreign Key referencing Events |
| is\_headliner | BOOLEAN | Indicates if band was the headliner |

Instruments\_Table:

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| Instrument\_ID | INT (PK, 2) | Unique identifier for each instrument |
| instrument | VARCHAR (30) | Instrument played by the musician |

Musician\_Instrument\_Table(Junction Table):

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| musician\_ID | INT (FK, 4) | Unique identifier for each musician |
| instrument\_ID | INT (FK, 2) | Unique identifier for each instrument |

Revenue\_Split\_Table:

|  |  |  |
| --- | --- | --- |
| Column | Type | Description |
| revenue\_split\_ID | INT (PK, 4) | Unique identifier for each revenue split entry |
| event\_ID | INT (FK, 1) | ID of the event where revenue was generated |
| performance\_ID | INT (FK, 4) | Unique identifier for each performance |
| band\_ID | INT (FK, 2) | Unique identifier for each band |
| musician\_ID | INT (FK, 4) | Unique identifier for each musician |
| merchandise\_sales | DECIMAL (7, 2) | Total merchandise sales |
| ticket\_revenue | DECIMAL (7, 2) | Total ticket revenue |
| is\_headliner | BOOLEAN | Indicates if band was the headliner |
| band\_revenue\_share | DECIMAL (7, 2) | Total revenue to be split for each band |
| musician\_royalty\_rate | DECIMAL (3, 2) | Royalty rate for each musician |
| musician\_revenue\_earned | DECIMAL (7, 2) | Earnings of each musician for each event |

Entity-Relationship (ER) Diagram

A screenshot of a computer

Description automatically generated

[1] - Was used to make the ER diagram

Justification for Database Design

The database design adheres to 3NF by minimizing redundancy and ensuring data integrity across the tables. Each table is dedicated to a specific entity (such as the Musicians\_Table for storing musicians information) or relationship, with primary keys, which ensure a unique identification. Junction tables, such as Musician\_Band\_Table and Musician\_Instrument\_Table, manage many-to-many relationships, providing flexibility where musicians can belong to multiple bands or play multiple instruments. This design maintains clarity and scalability while preserving the integrity of the data structure.

The database structure is modular and scalable because each table represents a distinct entity or relationship, which makes sure that changes or additions can be made without affecting other parts of the system. An example of this is new revenue types can be added to the Revenue\_Split\_Table by introducing additional columns, or venue details can be expanded by updating the Venue\_Table without altering existing relationships or data integrity. This allows for new requirements to be accommodated as they arise. Overall, this structure enables optimized query performance, making it easier to join tables and retrieve related data efficiently.

SQL Table Creation – [2]

Musicians\_Table:

CREATE TABLE Musicians\_Table (

musician\_ID INT (4) PRIMARY KEY,

name VARCHAR(30),

birth\_date DATE,

experience VARCHAR(20)

);

Bands\_Table:

CREATE TABLE Bands\_Table (

band\_ID INT (2) AUTO\_INCREMENT PRIMARY KEY,

band\_name VARCHAR(50),

genre VARCHAR(20),

formation\_year YEAR,

manager\_phone VARCHAR(15),

manager\_email VARCHAR(30)

);

Musician\_Band\_Table(Junction Table):

CREATE TABLE Musician\_Band\_Table (

musician\_ID INT (4),

band\_ID INT (2),

PRIMARY KEY (musician\_ID, band\_ID),

FOREIGN KEY (musician\_ID) REFERENCES

Musicians\_Table(musician\_ID),

FOREIGN KEY (band\_ID) REFERENCES Bands\_Table(band\_ID)

);

Events\_Table:

CREATE TABLE Events\_Table (

event\_ID INT (4) AUTO\_INCREMENT PRIMARY KEY,

event\_date DATE,

theme VARCHAR(50),

audience\_size INT(3),

venue\_ID INT(1),

FOREIGN KEY (venue\_ID) REFERENCES Venue\_Table(venue\_ID)

);

Venue\_Table:

CREATE TABLE Venue\_Table (

venue\_ID INT (1) AUTO\_INCREMENT PRIMARY KEY,

venue\_name VARCHAR(20)

);

Band\_Performance\_Table:

CREATE TABLE Band\_Performance\_Table (

performance\_ID INT (4) AUTO\_INCREMENT PRIMARY KEY,

band\_ID INT(2),

event\_ID INT(4),

is\_headliner BOOLEAN,

FOREIGN KEY (band\_ID) REFERENCES Bands\_Table(band\_ID),

FOREIGN KEY (event\_ID) REFERENCES Events\_Table(event\_ID)

);

Instruments\_Table:

CREATE TABLE Instruments\_Table (

     instrument\_ID INT (2) AUTO\_INCREMENT PRIMARY KEY,

     instrument\_name VARCHAR(30) UNIQUE

);

Musician\_Instrument\_Table(Junction Table):

CREATE TABLE Musician\_Instrument\_Table (

musician\_ID INT (4),

instrument\_ID INT (2),

PRIMARY KEY (musician\_ID, instrument\_ID),

FOREIGN KEY (musician\_ID) REFERENCES Musicians\_Table(musician\_ID),

FOREIGN KEY (instrument\_ID) REFERENCES Instruments\_Table(Instrument\_ID)

);

Revenue\_Split\_Table:

CREATE TABLE Revenue\_Split\_Table (

revenue\_split\_ID INT (4) AUTO\_INCREMENT PRIMARY KEY,

event\_ID INT (4),

performance\_ID INT (4),

band\_ID INT (2),

musician\_ID INT (4),

merchandise\_sales DECIMAL(7, 2),

ticket\_revenue DECIMAL(7, 2),

is\_headliner BOOLEAN,

band\_total\_revenue DECIMAL(7, 2),

musician\_royalty\_rate DECIMAL(3, 2),

musician\_revenue\_earned DECIMAL(10, 2),

FOREIGN KEY (event\_ID) REFERENCES Events\_Table(event\_ID),

FOREIGN KEY (performance\_ID) REFERENCES Band\_Performance\_Table(performance\_ID),

FOREIGN KEY (band\_ID) REFERENCES Bands\_Table(band\_ID),

FOREIGN KEY (musician\_ID) REFERENCES Musicians\_Table(musician\_ID)

);

Data Import from CSV File – [2]

CREATE TABLE Original\_Raw\_Data (

musician\_ID INT (4),

musician\_name VARCHAR(30),

birth\_date DATE,

instrument VARCHAR(30),

experience VARCHAR(20),

royalty\_rate DECIMAL(3, 2),

band\_name VARCHAR(50),

genre VARCHAR(20),

formation\_year YEAR,

manager\_phone VARCHAR(15),

manager\_email VARCHAR(30),

venue VARCHAR(30),

event\_date DATE,

theme VARCHAR(50),

audience\_size INT(3),

merchandise\_sales DECIMAL(7, 2),

ticket\_revenue DECIMAL(7, 2),

is\_headliner BOOLEAN

);

Data Transfer SQL – [2], [3]

Musicians\_Table:

INSERT INTO Musicians\_Table (musician\_ID, name, birth\_date, experience)

SELECT DISTINCT s.musician\_ID, s.musician\_name, s. birth\_date, s.experience

FROM original\_raw\_data s;

Bands\_Table:

INSERT INTO Bands\_Table (band\_name, genre, formation\_year, manager\_phone, manager\_email)

SELECT DISTINCT s.band\_name, s.genre, s.formation\_year, s.manager\_phone, s.manager\_email

FROM original\_raw\_data s;

Musician\_Band\_Table(Junction Table):

INSERT INTO Musician\_Band\_Table (musician\_ID, band\_ID)

SELECT DISTINCT s.musician\_ID, b.band\_ID

FROM original\_raw\_data s

JOIN Bands\_Table b ON s.band\_name = b.band\_name;

Events\_Table:

INSERT INTO Events\_Table (event\_date, theme, audience\_size, venue\_ID)

SELECT DISTINCT s.Event\_Date, s.Theme, s.Audience\_Size, v.Venue\_ID

FROM original\_raw\_data s

JOIN Venue\_Table v ON s.Venue = v.venue\_name;

Venue\_Table:

INSERT INTO Venue\_Table (venue\_name)

SELECT DISTINCT Venue

FROM original\_raw\_data;

Band\_Performance\_Table:

INSERT INTO Band\_Performance\_Table (band\_ID, event\_ID, is\_headliner)

SELECT b.band\_ID, e.event\_ID, MAX(s.is\_headliner) AS is\_headliner

FROM original\_raw\_data s

JOIN Bands\_Table b ON s.band\_name = b.band\_name

JOIN Events\_Table e ON s.event\_date = e.event\_date

GROUP BY b.band\_ID, e.event\_ID;

Intruments\_Table:

INSERT INTO Instruments\_Table (instrument\_name)

SELECT DISTINCT instrument\_name

FROM Staging\_Instruments\_Table;

Musician\_Instrument\_Table(Junction Table):

INSERT INTO Musician\_Instrument\_Table (musician\_ID, Instrument\_ID)

SELECT DISTINCT m.musician\_ID, i.Instrument\_ID

FROM original\_raw\_data s

JOIN Musicians\_Table m ON m.name = s.musician\_name

JOIN Instruments\_Table i ON i.instrument\_name = s.instrument;

Revenue\_Split\_Table:

INSERT INTO revenue\_split\_table(musician\_ID, band\_ID, event\_ID, performance\_ID, merchandise\_sales, ticket\_revenue, band\_total\_revenue, is\_headliner, musician\_royalty\_rate, musician\_revenue\_earned)

SELECT s.musician\_ID,b.band\_ID,e.event\_ID,bp.performance\_ID,s.merchandise\_sales,s.ticket\_revenue,

CASE

WHEN bp.is\_headliner = TRUE THEN

(0.6 \* (0.8 \* s.ticket\_revenue)) + (0.6 \* s.merchandise\_sales)

ELSE

(0.4 \* (0.8 \* s.ticket\_revenue)) / (

SELECT COUNT(\*)

FROM Band\_Performance\_Table bp2

WHERE bp2.event\_ID = bp.event\_ID AND bp2.is\_headliner = FALSE)

END AS band\_total\_revenue,

bp.is\_headliner,

s.royalty\_rate,

CASE

WHEN bp.is\_headliner = TRUE THEN

((0.6 \* (0.8 \* s.ticket\_revenue)) + (0.6 \* s.merchandise\_sales)) \* s.royalty\_rate

ELSE

((0.4 \* (0.8 \* s.ticket\_revenue)) / (

SELECT COUNT(\*)

FROM Band\_Performance\_Table bp2

WHERE bp2.event\_ID = bp.event\_ID AND bp2.is\_headliner = FALSE)) \* s.royalty\_rate

END AS musician\_revenue\_earned

FROM original\_raw\_data s

JOIN Bands\_Table b ON s.band\_name = b.band\_name

JOIN Events\_Table e ON s.event\_date = e.event\_date

JOIN Band\_Performance\_Table bp ON b.band\_ID = bp.band\_ID AND e.event\_ID = bp.event\_ID;

SQL Queries and Results – [3], [4]

a)

SELECT name AS musician\_name, birth\_date

FROM Musicians\_Table

WHERE birth\_date BETWEEN '1997-01-01' AND '2012-12-31';

|  |  |  |
| --- | --- | --- |
| name | birth\_date |  |
| Johnny Ortiz Jr. | 21/10/2005 |  |
| Peter Schultz | 30/08/1998 |  |

b)

SELECT b.band\_name, COUNT(bp.performance\_ID) AS total\_performances

FROM Bands\_Table b

JOIN Band\_Performance\_Table bp ON b.band\_ID = bp.band\_ID

GROUP BY b.band\_name;

First 5 rows outputted:

|  |  |
| --- | --- |
| band\_name | total\_performances |
| Aurora Woman Echoes | 120 |
| Cassandra Johnson | 123 |
| Crimson Stock Syndicate | 204 |
| David Yu | 159 |
| Electric Government Harmony | 158 |

c)

SELECT b.band\_name, b.genre, b.formation\_year,

SUM(rs.ticket\_revenue) / COUNT(DISTINCT mb.musician\_ID) AS total\_ticket\_revenue,

SUM(rs.merchandise\_sales) / COUNT(DISTINCT mb.musician\_ID) AS total\_merchandise\_sales

FROM Bands\_Table b

JOIN Revenue\_Split\_Table rs ON b.band\_ID = rs.band\_ID

JOIN Musician\_Band\_Table mb ON b.band\_ID = mb.band\_ID

GROUP BY b.band\_name, b.genre, b.formation\_year;

First 5 rows by lowest ticket revenue:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| band\_name | genre | formation\_year | total\_ticket\_revenue | total\_merchandise\_sales |
| Aurora Woman Echoes | Electronic | 2020 | 5545818 | 1607581 |
| Cassandra Johnson | Electronic | 2019 | 1229566 | 312266.8 |
| Crimson Stock Syndicate | Jazz | 2008 | 9914480 | 2488909 |
| David Yu | Funk | 2017 | 1610468 | 396034.6 |
| Electric Government Harmony | Funk | 2020 | 10219114 | 2401527 |

d)

SELECT DISTINCT v.venue\_name

FROM Venue\_Table v

JOIN Events\_Table e ON v.venue\_ID = e.venue\_ID

JOIN Band\_Performance\_Table bp ON e.event\_ID = bp.event\_ID

JOIN Bands\_Table b ON bp.band\_ID = b.band\_ID

WHERE e.event\_date BETWEEN '2023-03-01' AND '2023-03-31'

AND b.band\_name NOT IN ('The Waves', 'Groove Squad')

All venues listed because there is no band named 'The Waves' or 'Groove Squad'

|  |
| --- |
| venue\_name |
| Electric Hall |
| The Pavilion |
| Rhythm Arena |
| Rhythm Hall |
| Grand Lounge |
| Harmony Arena |

e)

SELECT v.venue\_name,

COUNT(e.event\_ID) AS num\_events\_hosted,

SUM(rs.ticket\_revenue) AS total\_ticket\_revenue,

SUM(rs.merchandise\_sales) AS total\_merchandise\_sales,

AVG(e.audience\_size) AS avg\_audience\_size

FROM Venue\_Table v

JOIN Events\_Table e ON v.venue\_ID = e.venue\_ID

JOIN Revenue\_Split\_Table rs ON e.event\_ID = rs.event\_ID

WHERE YEAR(e.event\_date) = YEAR(CURDATE()) - 1

GROUP BY v.venue\_name;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| venue\_name | num\_events\_hosted | total\_ticket\_revenue | total\_merchandise\_sales | avg\_audience\_size |
| The Pavilion | 419 | 3211997 | 1109021 | 253.4368 |
| Grand Lounge | 272 | 1927126 | 739641.3 | 248.2978 |
| Harmony Arena | 402 | 3061477 | 974753.4 | 270.0025 |
| Electric Hall | 260 | 2507213 | 783555.1 | 312.0538 |
| Rhythm Hall | 173 | 2646270 | 415433.4 | 460.6936 |
| Rhythm Arena | 204 | 3674752 | 519366.6 | 527.7157 |

f)

SELECT m.name AS musician\_name, b.band\_name,b.formation\_year, YEAR(m. birth\_date) AS birth\_year

FROM Musicians\_Table m

JOIN Musician\_Band\_Table mb ON m.musician\_ID = mb.musician\_ID

JOIN Bands\_Table b ON mb.band\_ID = b.band\_ID

WHERE b.formation\_year < YEAR(m. birth\_date);

No Output, as it seems no musician was born before the formation of a band, they are a part of.

g)

SELECT v.venue\_name, b.band\_name, MIN(e.event\_date) AS first\_performance\_date

FROM Venue\_Table v

JOIN Events\_Table e ON v.venue\_ID = e.venue\_ID

JOIN Band\_Performance\_Table bp ON e.event\_ID = bp.event\_ID

JOIN Bands\_Table b ON bp.band\_ID = b.band\_ID

WHERE e.theme LIKE '%Jazz%'

AND v.venue\_name = 'The Pavilion'

GROUP BY v.venue\_name, b.band\_name

ORDER BY v.venue\_name, first\_performance\_date;

First 5 rows outputted:

|  |  |  |
| --- | --- | --- |
| venue\_name | band\_name | first\_performance\_date |
| The Pavilion | Crimson Stock Syndicate | 12/11/2019 |
| The Pavilion | Wild Mean Legends | 04/12/2019 |
| The Pavilion | Phantom Especially Riders | 04/12/2019 |
| The Pavilion | Royal Skin Collective | 20/12/2019 |
| The Pavilion | Mystic Series Crew | 20/12/2019 |

h)

SELECT v.venue\_name, e.event\_date, b.band\_name, e.audience\_size

FROM Events\_Table e

JOIN Venue\_Table v ON e.venue\_ID = v.venue\_ID

JOIN Band\_Performance\_Table bp ON e.event\_ID = bp.event\_ID

JOIN Bands\_Table b ON bp.band\_ID = b.band\_ID

WHERE e.audience\_size = (SELECT MAX(audience\_size) FROM Events\_Table)

ORDER BY e.audience\_size, e.event\_date;

|  |  |  |  |
| --- | --- | --- | --- |
| venue\_name | event\_date | band\_name | audience\_size |
| Rhythm Arena | 16/01/2020 | Cassandra Johnson | 919 |
| Rhythm Arena | 16/01/2020 | Aurora Woman Echoes | 919 |

i)

WITH EventRangeRoyalties AS (

SELECT

rs.musician\_ID,

SUM(rs.musician\_revenue\_earned) AS total\_royalties

FROM Revenue\_Split\_Table rs

JOIN Events\_Table e ON rs.event\_ID = e.event\_ID

WHERE e.event\_date BETWEEN '2023-03-01' AND '2023-03-31'

GROUP BY rs.musician\_ID),

MaxRoyalties AS (

SELECT MAX(total\_royalties) AS max\_royalties

FROM EventRangeRoyalties)

SELECT

m.name AS musician\_name,

GROUP\_CONCAT(DISTINCT i.instrument\_name ORDER BY i.instrument\_name SEPARATOR ', ') AS instruments\_played,

mr.total\_royalties

FROM EventRangeRoyalties mr

JOIN MaxRoyalties mx ON mr.total\_royalties = mx.max\_royalties

JOIN Musicians\_Table m ON mr.musician\_ID = m.musician\_ID

JOIN Musician\_Instrument\_Table mi ON m.musician\_ID = mi.musician\_ID

JOIN Instruments\_Table i ON mi.instrument\_ID = i.instrument\_ID

GROUP BY m.musician\_ID, mr.total\_royalties;

|  |  |  |
| --- | --- | --- |
| musician\_name | Instruments\_played | total\_royalties |
| Jason Bowen | Drums, Saxophone | 27333.97 |

j)

SELECT

m.name AS musician\_name,

GROUP\_CONCAT(DISTINCT i.instrument\_name ORDER BY i.instrument\_name SEPARATOR ', ') AS instruments\_played,

GROUP\_CONCAT(DISTINCT DATE\_FORMAT(e.event\_date, '%Y-%m-%d') ORDER BY e.event\_date SEPARATOR ', ') AS performance\_dates

FROM Musicians\_Table m

JOIN Musician\_Band\_Table mb ON m.musician\_ID = mb.musician\_ID

JOIN Musician\_Instrument\_Table mi ON m.musician\_ID = mi.musician\_ID

JOIN Instruments\_Table i ON mi.instrument\_ID = i.instrument\_ID

JOIN Band\_Performance\_Table bp ON mb.band\_ID = bp.band\_ID

JOIN Events\_Table e ON bp.event\_ID = e.event\_ID

WHERE m.name = (SELECT band\_name FROM Bands\_Table b WHERE b.band\_ID = mb.band\_ID)

AND EXISTS (

SELECT 1

FROM Musician\_Band\_Table mb2

JOIN Bands\_Table b2 ON mb2.band\_ID = b2.band\_ID

WHERE mb2.musician\_ID = m.musician\_ID

AND b2.band\_name != m.name

)

GROUP BY m.musician\_ID

ORDER BY m.name;

First 5 dates shown:

|  |  |  |
| --- | --- | --- |
| musician\_name | instruments\_played | performance\_dates |
| David Yu | Trumpet, Violin | 2019-11-22, 2019-12-07, 2019-12-13, 2019-12-18, 2019-12-23... |
| Jeffrey Olson | Harp | 2019-11-18, 2019-11-27, 2019-12-20, 2020-02-05, 2020-02-10… |
| Kristin Lambert | Vocals | 2019-11-15, 2019-11-28, 2019-12-10, 2019-12-31, 2020-01-06… |

References

**[1]** dbdiagram.io, "Database Relationship Diagram Tool." [Online]. Available: <https://dbdiagram.io/d>. [Accessed: Nov. 12, 2024].  
**[2]** W3Schools, "SQL Tutorial." [Online]. Available: <https://www.w3schools.com/sql/default.asp>. [Accessed: Nov. 11, 12, 13, 2024].  
**[3]** SQL Tutorial, "Learn SQL Online." [Online]. Available: <https://www.sqltutorial.org>. [Accessed: Nov. 12, 13, 2024].

**[4]** GeeksforGeeks, "SQL Tutorial." [Online]. Available: <https://www.geeksforgeeks.org/sql-tutorial/>. [Accessed: Nov. 15, 17, 2024].