An SQL database for concert tour management

IST 659: Database Administration Concepts and Database Management

Team Members (alphabetical): Dan Pacheco, [Other two members omitted for this published version of the paper]

Sept. 12, 2025

Table of Contents

Database Design and Business Purpose

Introduction

The "Superfan" SQL database serves the needs of three distinct stakeholders who need data for multi-city concerts. When you think about concerts, it's tempting to think that it's a piece of cake. A band pulls up in its tour bus, unpacks instruments, gets on the stage to play, and then packs up for the next gig. But the reality is that it is a logistical nightmare for everyone involved.

Artists may belong to more than one band, or open for a band while also traveling for their own tours. They may have different artists and backup singers performing on different dates. And in many cases, there is no tour bus. Artists travel on their own flights from city to city,

often on different dates, stay at different hotels, and find themselves with just an hour to eat dinner very late (if at all) after performing a show.

The concert staff members have multiple skillsets that require them to arrive and leave at different times. They also have their own travel arrangements and hotel schedules. Sometimes, the stagehands aren't even on staff – they are freelancers who operate in a single city and set up and take down stages for different acts.

Finally, there are the tour managers who serve as the unlucky hub for this information as well as having to set up the travel folios for all of these disparate parties. Much of the information is handled with hastily-composed, disorganized emails and PDFs that leave everyone frustrated and confused.

All of this chaos is the perfect use for a SQL database that we call "Superfan." Ironically, Superfan doesn't do anything directly for fans, but it ensures that the concerts they attend happen on time and with the right people in place for every show. And hopefully it also maintains the sanity of the tour manager who is at the center of it all.

Target Audience

The Superfan database is set up to serve the following broad categories of stakeholders:

1. **Tour Managers:** People who assemble and manage all the logistics of a tour for a traveling band and support staff.
2. **Artists and Bands:** Musicians and the bands in which they play. Artists can belong to more than one band.
3. **Concert Staff:** The people who create sets, take them down, run the lights, sound engineers, and so forth.

Data Questions and Answers

Following are some of the questions the Superfan database can answer, broken out by stakeholder:

Artist & Band Management

- Which artists belong to which bands, and what instruments do they play?
- Which artists are scheduled to perform in the next week/month/year?
- Who will be performing at a specific event, and what instruments will they play?

Event Planning & Scheduling

- What is the full schedule of upcoming performances for a given artist/band?
- Which cities and states are most common for upcoming shows?
- What's happening on a particular date (e.g., "What shows are on Sept 10, 2025")?

Logistics & Lodging

- Which hotel is assigned to each band/artist for a given event?
- Are there enough hotel rooms assigned for all staff and artists at an event?
- What's the average lodging cost per night for an event?
- Which hotels are within a certain distance of venues?

Staffing & Operations

- Which staff members are assigned to which bands?
- What jobs are assigned to tour staff, and are there overlaps?
- Which staff are staying in which hotels for which events?

Food & Hospitality

- What restaurants are available near venues for artists/staff?
- Do restaurants cover dietary needs (vegan, gluten-free, etc.) for a specific event?

Database Structure

The database facilitates the storage, retrieval, and manipulation of tour, artist, venue, and concert logistics data. It ensures data integrity through foreign key relationships and enforces business logic through constraints and check conditions. This approach maintains accuracy and supports the complex operational requirements of modern music tour management.

The central entity is the **Event**, which is made of details like date and time, location, ticket price, artists, bands, and a description of the event.

The **Venue** represents the physical locations where events take place and is made up of address, capacity limits, amenities, and security rules. Venues can host multiple events over time, establishing a one-to-many relationship.

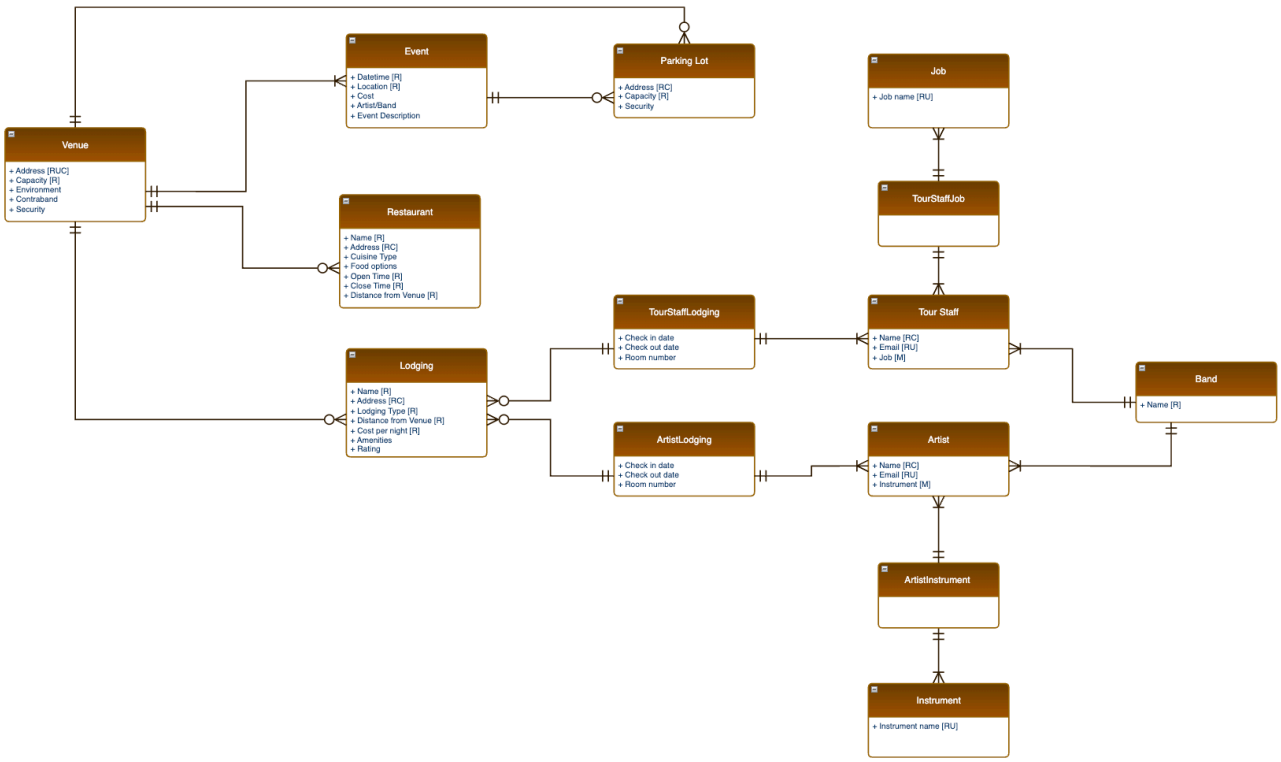
The system also tracks restaurants near venues through the **Restaurant** table. This is made up cuisine types, operating hours, distance from venue, and pricing information.

Parking Lot information helps manage venue accessibility by tracking parking addresses and capacity.

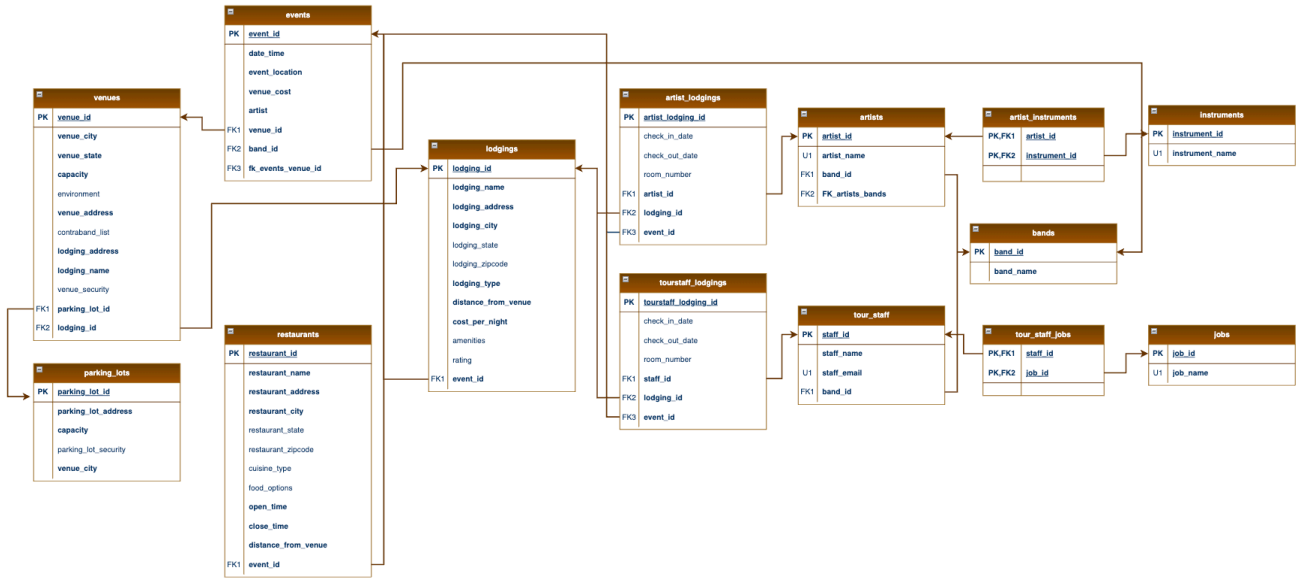
The **Lodging** system manages accommodation needs of everyone involved with a concert. It includes lodging types, costs, amenities, and ratings.

Finally, there are **Artist** and **Band** tables. There are many-to-many relationship between artists and instruments through the **ArtistInstrument** junction table, allowing the system to track which musicians play which instruments.

Conceptual Model



Entity Relationship Diagram (ERD)



Tables

Table: lodging

Column Name	Data Domain
lodging_id	INT, not null
lodging_name	VARCHAR(100), not null
lodging_address	VARCHAR(200), not null
lodging_city	VARCHAR(50), not null
lodging_state	CHAR(2)
lodging_zipcode	VARCHAR(10)
lodging_type	VARCHAR(50), not null
distance_from_venue	DECIMAL(6,2), not null
cost_per_night	DECIMAL(10,2), not null
amenities	TEXT
rating	DECIMAL(2,1)
event_id	INT, not null

Constraints:

- PRIMARY KEY (lodging_id)
- CHECK (cost_per_night >= 0)
- CHECK (rating BETWEEN 1 AND 5)

Table: tourstaff_lodging

Column Name	Data Domain
tourstaff_lodging_id	INT, not null
staff_id	INT, not null
lodging_id	INT, not null
event_id	INT, not null
check_in_date	DATE
check_out_date	DATE
room_number	VARCHAR(20)

Constraints:

- PRIMARY KEY (tourstaff_lodging_id)

Table: artist_lodging

Column Name	Data Domain
artist_lodging_id	INT, not null
staff_id	INT, not null
lodging_id	INT, not null
event_id	INT, not null
check_in_date	DATE
check_out_date	DATE
room_number	VARCHAR(20)

Constraints:

- PRIMARY KEY (artist_lodging_id)

Table: restaurants

Column Name	Data Domain
rentauarnt_id	INT, not null
restaurant_name	VARCHAR(50), not null
restaurant_address	VARCHAR(50), not null
restaurant_city	VARCHAR(50), not null
restaurant_state	VARCHAR(2)
restaurant_zipcode	VARCHAR(10)
cuisine_type	VARCHAR(50)
food_options	VARCHAR(50)
open_time	TIME, not null
close_time	TIME, not null
distance_from_venue	DECIMAL, not null
event_id	INT, not null

Constraints:

- PRIMARY KEY (restaurant_id)

Artists Table:

Purpose: Individual artists and their band affiliations.

Column Name	Data Domain
artist_id	INT, not null
artist_name	VARCHAR(100), not null
band_id	INT

Constraints:

- PRIMARY KEY (artist_id)
- FOREIGN KEY (band_id) REFERENCES bands(band_id)
- UNIQUE (artist_name)

SQL Example:

```
CREATE TABLE artists (
   artist_id INT NOT NULL PRIMARY KEY,
   artist_name VARCHAR(100) NOT NULL,
   band_id INT NULL,
   CONSTRAINT FK_artists_bands FOREIGN KEY (band_id) REFERENCES bands (band_id),
   CONSTRAINT UQ_artists_name UNIQUE (artist_name)
);
```

Key Features: Links individual artists to bands, supports solo artists (NULL band_id), enforces unique artist names. Contains 20 artists including "Aria Vega," "Zephyr Cain," and "Mira Solis."

Table: bands

Purpose: Central registry of musical groups and acts in the tour management system.

Column Name	Data Domain
band_id	INT, not null
band_name	VARCHAR(50), not null

Constraints:

- PRIMARY KEY (band_id)

SQL example:

```
CREATE TABLE bands (
   band_id INT NOT NULL PRIMARY KEY,
   band_name VARCHAR(50) NOT NULL
);
```

Sample data: Contains 20 sample bands including "The Midnight Owls," "Crimson Echoes," "Neon River," and "Crystal Pines."

Table: tour_staff

Column Name	Data Domain
staff_id	INT, not null
staff_name	VARCHAR(25), not null
staff_email	VARCHAR(35), not null
band_id	INT, not null

Constraints:

- PRIMARY KEY (staff_id)

Table: events

Column Name	Data Domain
event_id	INT, not null
venue_id	INT, not null
date_time	DATE, not null
event_location	VARCHAR(25), not null
venue_cost	NUMERIC, not null
artist	VARCHAR(50), not null
band_id	INT, not null
event_brief	VARCHAR(150)

Constraints:

- PRIMARY KEY (event_id)
- FOREIGN KEY (venue_id) REFERENCES venues(venue_id)

Table: parking_lots

Column Name	Data Domain
parking_lot_id	INT, not null
parking_lot_address	VARCHAR(25), not null
capacity	NUMERIC, not null
parking_lot_security	VARCHAR(3)
venue_city	VARCHAR(25), not null

Constraints:

- PRIMARY KEY (parking_lot_id)

Instruments Table

Purpose: A table of all musical instruments and equipment used by artists, tied to their artist IDs.

Column Name	Data Domain
instrument_id	INT, not null
instrument_name	VARCHAR(50), not null

Constraints:

- PRIMARY KEY (instrument_id)
- UNIQUE (instrument_name)

SQL example:

```
CREATE TABLE instruments (
  instrument_id INT NOT NULL PRIMARY KEY,
  instrument_name VARCHAR(50) NOT NULL,
  CONSTRAINT UQ_instruments_name UNIQUE (instrument_name)
);
```

Sample Data: Features futuristic instruments including "Synth Harp," "Solar Drums," "Crystal Guitar," "Echo Flute," and "Bass Engine."

Artist_Instruments Junction Table

Purpose: Many-to-many relationship supporting multi-instrumentalist artists.

Column Name	Data Domain
artist_id	INT, not null
instrument_id	INT, not null

Constraints:

- PRIMARY KEY (artist_id, instrument_id)
- FOREIGN KEY (artist_id) REFERENCES artists(artist_id)
- FOREIGN KEY (instrument_id) REFERENCES instruments(instrument_id)

SQL example:

```
CREATE TABLE artist_instruments (
  artist_id INT NOT NULL,
  instrument_id INT NOT NULL,
  CONSTRAINT PK_artist_instruments PRIMARY KEY (artist_id, instrument_id),
  CONSTRAINT FK_ai_artist FOREIGN KEY (artist_id) REFERENCES artists (artist_id),
  CONSTRAINT FK_ai_instrument FOREIGN KEY (instrument_id) REFERENCES instruments (instrument_id)
);
```

Venues Table

Purpose: Comprehensive venue management with capacity, location, and security details.

venue_id	INT, not null

venue_city	VARCHAR(50), not null
venue_state	VARCHAR(2), not null
capacity	INT, not null
environment	VARCHAR(10)
venue_address	VARCHAR(50), not null
contraband_list	VARCHAR(100)
parking_lot_id	INT, not null
lodging_address	VARCHAR(50), not null
lodging_name	VARCHAR(50), not null
venue_security	VARCHAR(3)
lodging_id	INT, not null

Constraints:

- PRIMARY KEY (venue_id)

SQL example:

```
CREATE TABLE venues (
    venue_id INT NOT NULL PRIMARY KEY,
    venue_city VARCHAR(50) NOT NULL,
    venue_state VARCHAR(2) NOT NULL,
    capacity INT NOT NULL,
    environment VARCHAR(10) NULL,
    venue_address VARCHAR(50) NOT NULL,
    contraband_list VARCHAR(100) NULL,
    parking_lot_id INT NOT NULL,
    lodging_address VARCHAR(50) NOT NULL,
    lodging_name VARCHAR(50) NOT NULL,
    venue_security VARCHAR(3) NULL,
    lodging_id INT NOT NULL
);
```

Sample data: Tracks venue capacity (ranging from 7,545 to 45,833), environment types (Indoor, Outdoor, Mixed), security requirements, and contraband restrictions (e.g., "Drugs, Pets," "Weapons, Alcohol").

6. Events Table

Purpose: Central event scheduling and management with venue and cost tracking.

Structure:

Column Name	Data Domain
event_id	INT, not null
venue_id	INT, not null
date_time	DATETIME2, not null
event_location	VARCHAR(25), not null
venue_cost	DECIMAL(12,2), not null
artist	VARCHAR(50), not null
band_id	INT, not null

Constraints:

- PRIMARY KEY (event_id)
- FOREIGN KEY (venue_id) REFERENCES venues(venue_id)

SQL examples:

```
CREATE TABLE events (
  event_id INT NOT NULL PRIMARY KEY,
  venue_id INT NOT NULL,
  date_time DATETIME2 NOT NULL,
  event_location VARCHAR(25) NOT NULL,
  venue_cost DECIMAL(12, 2) NOT NULL,
  artist VARCHAR(50) NOT NULL,
```

```
  band_id INT NOT NULL,
  CONSTRAINT fk_events_venue_id FOREIGN KEY (venue_id) REFERENCES venues (venue_id)
);
```

Sample data:Manages tour dates from March 2025 through September 2025, tracks venue costs ($49-$200), links events to specific artists and venues.

7. Parking_Lots Table

Purpose: Parking facility management associated with venue locations.

Structure:

Column Name	Data Domain
parking_lot_id	INT, not null
parking_lot_address	VARCHAR(50), not null
capacity	INT, not null
parking_lot_security	VARCHAR(3)
venue_city	VARCHAR(25), not null

Constraints:

- PRIMARY KEY (parking_lot_id)

SQL example:

```
CREATE TABLE parking_lots (
  parking_lot_id INT NOT NULL PRIMARY KEY,
  parking_lot_address VARCHAR(50) NOT NULL,
  capacity INT NOT NULL,
  parking_lot_security VARCHAR(3) NULL,
  venue_city VARCHAR(25) NOT NULL
);
```

Sample data: Parking capacities (84-747 spaces), security provisions, and geographic distribution across tour cities.

8. Lodging Table

Purpose: Accommodation management with quality ratings and proximity tracking.

Structure:

Column Name	Data Domain
lodging_id	INT, not null
lodging_name	VARCHAR(100), not null
lodging_address	VARCHAR(200), not null
lodging_city	VARCHAR(50), not null
lodging_state	CHAR(2)
lodging_zipcode	VARCHAR(10)
lodging_type	VARCHAR(50), not null
distance_from_venue	DECIMAL(6,2), not null
cost_per_night	DECIMAL(10,2), not null
amenities	VARCHAR(MAX)
rating	DECIMAL(2,1)
event_id	INT, not null

Constraints:

- PRIMARY KEY (lodging_id)
- CHECK (cost_per_night >= 0)
- CHECK (rating BETWEEN 1 AND 5)

sql

```
CREATE TABLE lodging (
    lodging_id INT NOT NULL PRIMARY KEY,
    lodging_name VARCHAR(100) NOT NULL,
    lodging_address VARCHAR(200) NOT NULL,
    lodging_city VARCHAR(50) NOT NULL,
    lodging_state CHAR(2) NULL,
    lodging_zipcode VARCHAR(10) NULL,
    lodging_type VARCHAR(50) NOT NULL,
    distance_from_venue DECIMAL(6, 2) NOT NULL,
    cost_per_night DECIMAL(10, 2) NOT NULL CHECK (cost_per_night >= 0),
    amenities VARCHAR(MAX) NULL,
    rating DECIMAL(2, 1) NULL CHECK (rating BETWEEN 1 AND 5),
    event_id INT NOT NULL
);
```

Sample data: Accomondation costs ($120.50-$150.00 per night), distance from venues (0.9-2.8 miles), amenities ("WiFi,Pool,Gym"), and quality ratings (3.8-4.5 stars).

9. Restaurants Table

Purpose: Dining options and food service coordination near venues.

Structure:

Column Name	Data Domain
restaurant_id	INT, not null
restaurant_name	VARCHAR(50), not null
restaurant_address	VARCHAR(50), not null
restaurant_city	VARCHAR(50), not null
restaurant_state	VARCHAR(2)
restaurant_zipcode	VARCHAR(10)
cuisine_type	VARCHAR(50)

Column Name	Data Domain
food_options	VARCHAR(50)
open_time	TIME, not null
close_time	TIME, not null
distance_from_venue	DECIMAL(6,2), not null
event_id	INT, not null

Constraints:

- PRIMARY KEY (restaurant_id)

sql

```
CREATE TABLE restaurants (
    restaurant_id INT NOT NULL PRIMARY KEY,
    restaurant_name VARCHAR(50) NOT NULL,
    restaurant_address VARCHAR(50) NOT NULL,
    restaurant_city VARCHAR(50) NOT NULL,
    restaurant_state VARCHAR(2) NULL,
    restaurant_zipcode VARCHAR(10) NULL,
    cuisine_type VARCHAR(50) NULL,
    food_options VARCHAR(50) NULL,
    open_time TIME NOT NULL,
    close_time TIME NOT NULL,
    distance_from_venue DECIMAL(6, 2) NOT NULL,
    event_id INT NOT NULL
);
```

Sample data: Cuisine types (Diner, BBQ, Italian, Mexican), dietary options (Vegetarian, Vegan, Gluten-Free), operating hours, and venue proximity.

10. Tour_Staff Table

Purpose: Personnel management for tour operations and logistics.

Structure:

Column Name	Data Domain
staff_id	INT, not null
staff_name	VARCHAR(25), not null
staff_email	VARCHAR(35), not null
band_id	INT, not null

Constraints:

- PRIMARY KEY (staff_id)
- FOREIGN KEY (band_id) REFERENCES bands(band_id)

sql

```
CREATE TABLE tour_staff (
  staff_id INT NOT NULL PRIMARY KEY,
  staff_name VARCHAR(25) NOT NULL,
  staff_email VARCHAR(35) NOT NULL,
  band_id INT NOT NULL,
  CONSTRAINT fk_tour_staff_band FOREIGN KEY (band_id) REFERENCES bands (band_id)
);
```

11. Jobs Table

Purpose: Definition of tour roles and responsibilities.

Structure:

Column Name	Data Domain
job_id	INT, not null
job_name	VARCHAR(25)

Constraints:

- PRIMARY KEY (job_id)

sql

```
CREATE TABLE jobs (
  job_id INT NOT NULL PRIMARY KEY,
  job_name VARCHAR(25) NULL
);
```

Available Positions: Stage Setup, Audio Technician, Lighting, Road Manager, Security.

Table: tour_staff_jobs

Column Name	Data Domain
staff_id	INT, not null

Constraints:

- PRIMARY KEY (staff_id)

Table: jobs

Column Name	Data Domain
job_id	INT, not null
job_name	VARCHAR(25)

Constraints:

- PRIMARY KEY (job_id)

Table: venues

Column Name	Data Domain
venue_id	INT, not null
venue_city	VARCHAR(50), not null
venue_state	VARCHAR(2), not null
capacity	NUMERIC, not null
environment	VARCHAR(10)
venue_address	VARCHAR(50), not null
contraband_list	VARCHAR(100)
parking_lot_id	INT, not null
lodging_address	VARCHAR(25), not null
lodging_name	VARCHAR(25), not null
venue_security	VARCHAR(3)
lodgining_id	INT, not null

Constraints:

- PRIMARY KEY (venue_id)

Junction Tables

Artist_Lodging and Tourstaff_Lodging

These tables manage accommodation assignments for artists and staff respectively, tracking check-in/ check-out dates, room assignments, and event associations.

Entity Relationships

1. Band-Artist Relationship (One-to-Many):

Sql example:

```
ALTER TABLE artists ADD CONSTRAINT FK_artists_bands
FOREIGN KEY (band_id) REFERENCES bands (band_id);
```

Artist-Instrument Relationship (Many-to-Many):

Implemented through the `artist_instruments` junction table enabling artists to play multiple instruments.

Event-Venue Relationship (One to Many):

Sql example:

```
ALTER TABLE events ADD CONSTRAINT fk_events_venue_id
FOREIGN KEY (venue_id) REFERENCES venues (venue_id);
```

4. Accommodation Management Relationships: Separate many-to-many relationships for artist and staff lodging assignments with detailed tracking of dates and room numbers.

Real-World Application Scenarios

Scenario 1: Multi-City Tour Planning

A record label plans a 15-city tour featuring multiple bands. The database coordinates venue bookings, manages accommodation for over 50 personnel, and tracks artist schedules across performance dates.

Scenario 2: Emergency Venue Changes

When a venue becomes unavailable, the database can help tour managers find alternative venues with the right capacity, security, and nearby hotel and meal options.

Scenario 3: Budget Optimization

Tour management reduces accommodation costs by 15% through strategic lodging selection while maintaining quality standards using database cost analysis.

Views, Stored Procedures and Indexes

Normalization Function

sql

```sql
CREATE OR ALTER FUNCTION dbo.fn_Normalize(@input NVARCHAR(4000))
RETURNS NVARCHAR(4000)
AS
BEGIN
  RETURN LTRIM(RTRIM(UPPER(ISNULL(@input, ''))))
END
```

Purpose: Standardizes text input for consistent searching and comparison operations.

Band Artist Lookup Function

sql

```sql
CREATE OR ALTER FUNCTION dbo.fn_BandArtists (@band_id INT)
RETURNS TABLE
AS
RETURN
(
  SELECT a.artist_id, a.artist_name, a.band_id
  FROM dbo.artists AS a
  WHERE a.band_id = @band_id
);
```

Purpose: Enables dynamic band roster queries for tour planning and coordination.

Artist-Instrument Capability View

sql

```sql
CREATE OR ALTER VIEW dbo.v_artists_with_instruments
AS
SELECT b.band_id, b.band_name, a.artist_id, a.artist_name,
    i.instrument_id, i.instrument_name
FROM dbo.artists AS a
JOIN dbo.bands AS b ON b.band_id = a.band_id
JOIN dbo.artist_instruments AS ai ON ai.artist_id = a.artist_id
JOIN dbo.instruments AS i ON i.instrument_id = ai.instrument_id;
```

Aggregated Band Roster View

sql

```sql
CREATE OR ALTER VIEW dbo.v_band_roster_with_instruments
AS
SELECT b.band_id, b.band_name, a.artist_id, a.artist_name,
    STRING_AGG(i.instrument_name, ', ') AS instruments_played
FROM dbo.artists AS a
JOIN dbo.bands AS b ON b.band_id = a.band_id
LEFT JOIN dbo.artist_instruments AS ai ON ai.artist_id = a.artist_id
LEFT JOIN dbo.instruments AS i ON i.instrument_id = ai.instrument_id
GROUP BY b.band_id, b.band_name, a.artist_id, a.artist_name;
```

Artist Search Stored Procedure

sql

```sql
CREATE OR ALTER PROCEDURE dbo.p_search_artists
  @q NVARCHAR(200) = NULL,
  @band_id INT = NULL
```

Features: Flexible search supporting partial text matching, exact band filtering, and normalized string comparison.

Advanced Band Roster Procedure

sql

```
CREATE OR ALTER PROCEDURE dbo.p_band_roster_pretty
  @band_ids_csv   NVARCHAR(MAX) = NULL,
  @band_names_csv NVARCHAR(MAX) = NULL,
  @band_id        INT = NULL,
  @band_name      NVARCHAR(200) = NULL
```

Capabilities: Multiple input methods (single ID, single name, CSV lists), dynamic target band selection, formatted output with band headers.

Indexes

1. Event Date Range Index:

sql

```
CREATE NONCLUSTERED INDEX IX_events_date_time_venue
ON events (date_time, venue_id)
INCLUDE (event_id, event_location, artist, band_id, venue_cost);
```

2. Location-Based Event Index:

sql

```
CREATE NONCLUSTERED INDEX IX_events_location_date
ON events (event_location, date_time)
INCLUDE (event_id, venue_id, artist, band_id);
```

3. Lodging Distance Index:

sql

```
CREATE NONCLUSTERED INDEX IX_lodging_event_distance
ON lodging (event_id, distance_from_venue)
INCLUDE (lodging_id, lodging_name, cost_per_night, rating, amenities);
```

4. Restaurant Cuisine Index:

sql

```
CREATE NONCLUSTERED INDEX IX_restaurants_city_cuisine
ON restaurants (restaurant_city, cuisine_type)
INCLUDE (restaurant_id, restaurant_name, distance_from_venue, food_options);
```

Example Use Cases and Queries

1. Band Artist Roster Function Usage

sql

```
SELECT 'Band 3 Artists:' AS Test_Description, *
FROM dbo.fn_BandArtists(3);
```

Results: Returns Mira Solis and Orion Blake from "Neon River" band.

Business Use: Quickly identify all members of a specific band for tour coordination.

2. Tour Schedule and Venue Analysis

sql

```
SELECT TOP 5
    e.event_id, e.date_time, e.event_location, e.artist,
    v.capacity, v.environment, v.venue_security, e.venue_cost
FROM events e
```

```
JOIN venues v ON e.venue_id = v.venue_id
WHERE e.date_time >= '2025-08-01'
ORDER BY e.date_time;
```

Sample Results:

- August 1, 2025: Elara Quinn at New York venue (21,371 capacity, Indoor, $150)
- August 6, 2025: Elara Quinn at New York venue (21,371 capacity, Indoor, $150)
- August 15, 2025: Liam Cross at San Diego venue (7,545 capacity, Outdoor, $76)

Business Use: Revenue forecasting - project ticket sales and pricing based on venue capacity and market size.

3. Lodging Cost Analysis

sql

```
SELECT l.lodging_name, l.lodging_city, l.distance_from_venue,
    l.cost_per_night, l.rating, l.amenities
FROM lodging l
WHERE l.cost_per_night <= 140.00
ORDER BY l.cost_per_night;
```

Business Use: Budget compliance - ensure accommodation expenses stay within approved tour budget limits.

4. Artist Search Functionality

sql

```
EXEC dbo.p_search_artists @q = N'Solis';
```

Results: Returns Mira Solis from "Neon River" band playing Echo Flute

Business Use: Technical setup planning - quickly identify required instruments and equipment before venue arrival.

5. Venue Capacity and Security Analysis

sql

```
SELECT v.venue_city, v.venue_state, v.capacity, v.environment,
    v.venue_security, v.contraband_list
FROM venues v
WHERE v.capacity > 20000
ORDER BY v.capacity DESC;
```

Business Use: Security cost planning - budget appropriate security expenses based on venue size and requirements.

6. Restaurant Coordination

sql

```
SELECT r.restaurant_name, r.restaurant_city, r.cuisine_type,
    r.food_options, r.distance_from_venue,
    FORMAT(r.open_time, 'HH:mm') as opens,
    FORMAT(r.close_time, 'HH:mm') as closes
FROM restaurants r
ORDER BY r.distance_from_venue;
```

Business Use: Meal logistics - coordinate crew dining schedules around restaurant hours and proximity to venues.

7. Artist-Instrument Relationship View

```
SELECT *
FROM dbo.v_artists_with_instruments
ORDER BY band_name, artist_name, instrument_name;
```

Business Use: Shows detailed roster of artists in bands with their specific instruments for technical setup planning.

8. Band Roster with Aggregated Instruments

```
SELECT * FROM dbo.v_band_roster_with_instruments;
```

Business Use: Provides a consolidated view showing each artist and all instruments they play (comma-separated) for quick reference during sound checks.

9. Multi-Parameter Artist Search

```
-- Search by partial artist name
EXEC dbo.p_search_artists @q = N'Cassian';

-- Search by band ID
EXEC dbo.p_search_artists @band_id = 5;

-- Get all artists (no filters)
EXEC dbo.p_search_artists;
```

Business Use: Flexible search functionality for tour managers to find artists by name, band affiliation, or browse all artists.

10. Pretty Band Roster Display

```
-- Multiple bands by name
EXEC dbo.p_band_roster_pretty @band_names_csv = N'Neon River, Static Horizon,
Golden Fractals';

-- Multiple bands by ID
EXEC dbo.p_band_roster_pretty @band_ids_csv = N'2, 5, 9';
```

Business Use: Creates formatted, human-readable band rosters for tour documentation and coordination meetings.

11. Event Overview Dashboard

```
SELECT *
FROM dbo.v_EventOverview
ORDER BY date_time;

-- Filter by city and budget constraints
SELECT *
FROM dbo.v_EventOverview
WHERE venue_city = 'Austin'
  AND (min_lodging_cost_per_night IS NOT NULL AND min_lodging_cost_per_night <=
120)
ORDER BY date_time;
```

Business Use: Comprehensive event planning dashboard showing venue details, band info, restaurant count, and lodging cost analysis for budget planning.

12. Band Lineup with Instruments

```
EXEC dbo.sp_GetBandLineup @band_id = 7;
```

Business Use: Detailed band member listing with all instruments each artist plays, essential for equipment planning and stage setup.

13. Lodging Cost Calculation

```
SELECT
    a.artist_name,
    l.lodging_name,
    al.check_in_date,
    al.check_out_date,
    l.cost_per_night,
    dbo.CalculateLodgingCost(al.check_in_date, al.check_out_date, l.cost_per_night)
AS total_lodging_cost
FROM
    dbo.artist_lodging al
    INNER JOIN dbo.artists a ON al.artist_id = a.artist_id
    INNER JOIN dbo.lodging l ON al.lodging_id = l.lodging_id
WHERE
    al.event_id = 1;
```

Business Use: Calculates total lodging costs per artist for specific events, considering check-in/check-out dates and nightly rates for budget tracking.

14. Date-Based Event Queries (Index Optimized)

```
-- Events in specific month
SELECT TOP 3 event_id, date_time, event_location, artist, venue_cost
FROM events
WHERE date_time >= '2025-08-01' AND date_time < '2025-09-01'
ORDER BY date_time;
```

Business Use: Optimized queries for tour schedule planning and monthly event coordination.

15. Location-Based Event Planning

```
SELECT event_id, event_location, date_time, artist
FROM events
WHERE event_location = 'New York'
ORDER BY date_time;
```

Business Use: City-specific event planning for coordinating multiple shows in the same metropolitan area.

16. Proximity-Based Lodging Selection

```
SELECT l.lodging_name, l.lodging_city, l.distance_from_venue, l.cost_per_night,
l.rating
FROM lodging l
WHERE l.event_id = 1
ORDER BY l.distance_from_venue;
```

Business Use: Find closest accommodations to venues for minimizing travel time and transportation costs.

17. Cuisine-Based Restaurant Planning

```
SELECT restaurant_name, restaurant_city, cuisine_type, food_options,
distance_from_venue
FROM restaurants
WHERE cuisine_type = 'Mexican'
ORDER BY restaurant_city;
```

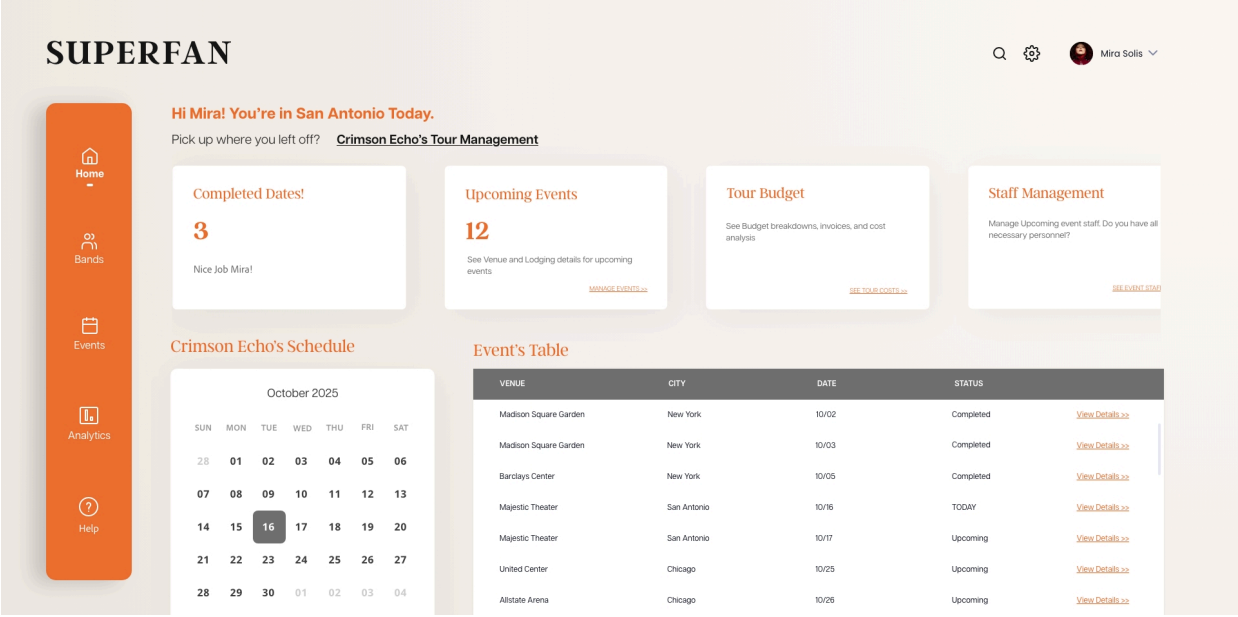
Business Use: Coordinate dining options by cuisine preferences and dietary requirements for tour personnel.

(Paper continues on next page)

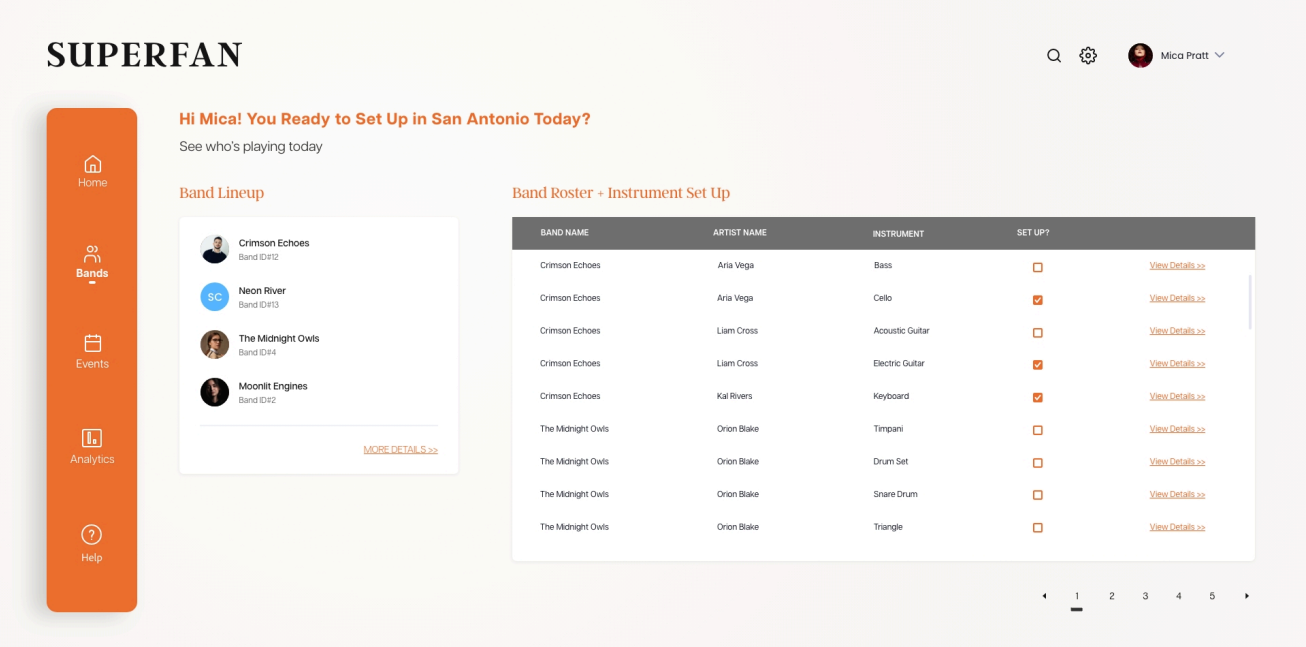
Sample User Interface

Here are some UI elements that could be built to run some of these queries:

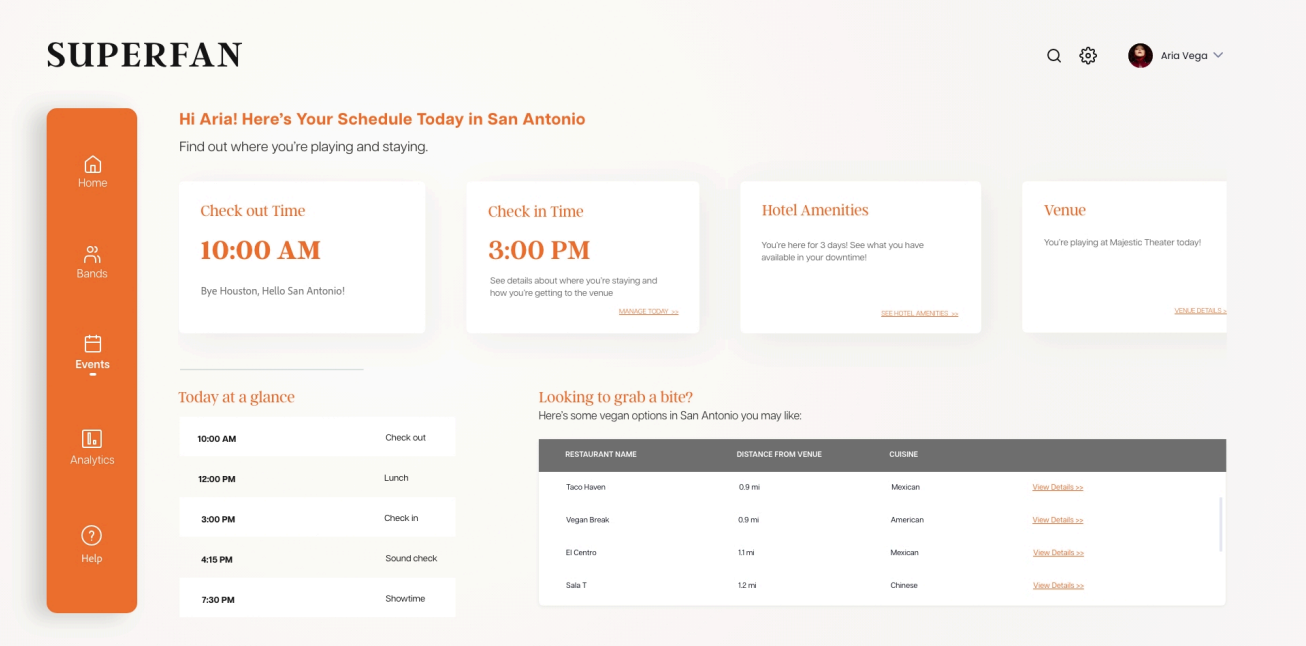
Mockups 1&2 in “Tour Manager View”, Larger dashboard options to include all events and budget and analytic options of tour (Queries 3, 11, 13):



Mockup 3: “Staff View” dashboard option for setting up venue for event (Queries 7, 8, 12):



Mockup 4: “Artist View” dashboard option to see daily schedule, accommodation and venue details and restaurant/food options near venue (Queries 14-17):



Lessons Learned

[Team member 1]: Merging SQL scripts comes with many challenges. I imagine there are proper ways to do it, but I simply jammed them into one file and got it to work. In doing so, obviously I ran into errors where things were mismatching and needed to clean the scripts up. Looking back, it wasn't too bad but at the time doing it could be a bit frustrating because I was starting with the very last error rather than at the first error. That's my lesson learned about merging scripts and errors. As I was fixing the error, starting with the very last one the fixes were either not working or causing more issues. Well, it dawned on me that's because SQL runs from top to bottom so of course I would run into more troubleshooting errors in the reverse. So, once I started correcting errors from the top, I found often all the rest of the errors solved themselves. That's because something at the start of the run flags a bunch of stuff that then errors throughout the rest of the script. As I said before, looking at it presently that's obvious how I could have possibly overlooked that. So, in all I learned good trouble shooting practices, or last least good places to start and ways to save me a tremendous amount of time in future SQL projects.

[Team member 2:] One of my takeaways from the project was how helpful it was to talk through the data modeling process with other people. Although, as we were discussing the project, there were moments where I think we wish we could've just been in person and drawn the whole thing out on a whiteboard which would've made the process go a lot faster. It was interesting to see how our project scope evolved from being perhaps too narrow and not user focused to eventually we had to cut down some original ideas relating to a fan component of the database as it became clear that it didn't fit into what we were actually creating. Personally, coming from a design background, the whole experience beginning to end felt a bit unusual to me, as I think I spent much more time conceptually thinking about the eventual data and how tables would fit together and the actual functionality process that I found it more difficult to visualize and focus on the user. It was still I think ultimately an iterative design process of sorts, however, definitely the opposite of my usual workflow. Creating mockups for example was the last thing I did in the project when basic concepts of user interfaces is something I do very early on in my process to imagine user needs vs. as an afterthought.

Dan Pacheco: The biggest lesson for me has been how challenging it is to have multiple people working on what in the end needs to be a single SQL script because of all the dependencies in the code. If you forget to drop a table or view or really anything higher up, the entire script breaks. When working individually you can plan for that, but in a team setting it becomes a recipe for disaster. We ended up having one team member (Zach) "true up" the final up/down script and also make it transaction safe, and then we each worked on our views and functions independently in our own files. In the working world I imagine there are better environments and best practices for handling this, but we had to figure it out on our own.

Conclusion

Superfan does a great job at handling most of the logistical challenges in concert tour management. We think it's so good that we wonder if this is something that exists in the marketplace yet. If not, maybe someone could make a business out of it. It could be almost like QiuickBooks, but for concert managers.

By centralizing all of the information around artists, venues, hotels, food and even musical instruments and roles into one database, it provides a birds-eye view of all of those relationships and constraints. Its real power is that one person (tour manager) inputs information, but everyone at every level can benefit by seeing the information they need to know at that particular time.

Appendix A - Glossary of Terms

Artist: A musical performer or entertainer who participates in tour events and performances.

Band: A musical group consisting of multiple artists who perform together as a unit.

Capacity: The maximum number of people that can be accommodated in a venue or parking facility.

Database Normalization: The process of organizing data to reduce redundancy and improve data integrity.

Entity Relationship Diagram (ERD): A visual representation of the database structure showing entities, attributes, and relationships.

Event: A scheduled performance or show at a specific venue, date, and time.

Foreign Key: A field that establishes a link between data in two tables and enforces referential integrity.

Instrument: Musical equipment or tools used by artists during performances.

Job: A specific role or position assigned to tour staff members, such as sound engineer or stage manager.

Junction Table: A table that implements many-to-many relationships between other tables in the database.

Lodging: Accommodation facilities such as hotels, motels, or other housing options for artists and staff.

Many-to-Many Relationship: A database relationship where multiple records in one table can be associated with multiple records in another table.

Parking Lot: Designated areas for vehicle parking associated with venue locations.

Primary Key: A unique identifier for each record in a database table.

Referential Integrity: The requirement that foreign key values must correspond to existing primary key values in related tables.

Restaurant: Dining establishments that provide food services for tour personnel and audiences.

Tour: A series of performances by artists at multiple venues over a specified time period.

Tour Staff: Personnel responsible for various aspects of tour operations, including technical, logistical, and administrative roles.

Venue: A location where events and performances take place, such as concert halls, arenas, or outdoor amphitheaters.

Appendix B - Project Meeting Logs

Meetings

Date and topic	Discussed	To-Do's
7/25: 8 p.m. EDT. First group meeting.	First meet and greet. Discussed our backgrounds and skills.	Everyone should come up with ideas for other types of data we could relate to music and concerts. e.g. Hotels, Restaurants, Price for tickets, Price for "swag", other academic data about specific songs (there are data sets out there about that).
7/28 8:30 p.m.	Expanded idea to focus on three stakeholders: concert organizers, artists and fans.	Contribute to Brainstorm doc Zach started to help flesh out needs of stakeholders and eventually the database tables, columns and keys.
8/4 in / after class	Met with Prof. as a group. Assigned tables to ourselves to set up as proposed schema.	Work on tables: Bands and roadies (Dan); Event details (Zach); Places (Nikk)
8/11 in / after class	Combined our database diagrams (conceptual and logical) into one.	Start up/down scripts for our assigned tables.
8/18 in / after class	Fine-tuned database diagrams.	Combine all up/down scripts into one (Zach)
8/25 in / after class	Test up/down script together. Begin working on views, stored prodedures for each of our stakeholder areas.	Test each others' views and procedures on the database from main up/down script.
9/1 in class	Met as a group in second half of class.	Divvied up work. Zach continued with the SQL file, Nikki assigned to complete / clean up database diagrams, Dan volunteered to start the final paper.

9/4 in class	Individually looked at and ran latest SQL file. Provided feedback.	9/9 Final Project Meeting notes Attendees: Zach and Dan attended. Nikki absent. We agreed that we are close to having the final finished. Here's what's left to be done. - Updating your tab of the team work log (All of us) - Writing a paragraph or two of lessons you learned in this final project (All of us) - Making the up/down script transation-safe (Zach) - Combining the question-oriented scripts into one script. (Zach) - Completing final paper, as outlined by Prof. Zink last night (Dan). To do: - Fleshing out the narrative. - Updating all questions to line up with the contents of the questions script. - Superfan product mockups (Nikki) - Executive powerpoint (Nikki)

Dan Pacheco Log

Date	Work Done
16-Jul	Created the https://spotifydatabaseproject.netlify.app/ to be able to download data from spotify and ticketmaster APIs.
23-Jul	Set up a shared folder for the team and initial documents. Downloaded some sample data.
26-Jul	Downloaded music and concert data for Billie Eilish, Coldplay, Kelly Clarkson, Lil Wayne, Paul McCartney, and Shania Twain.
26-Jul	Created a basic import script for downloaded artist and event data. Saved in new SQL folder.
31-Jul	Sent revised pitch to Zink -- received go-ahead.
5-Aug	Created proposed schema for bands and roadies. Decided to put roadie jobs and musician parts in two lookup tables. Created a first SQL script to test this data along with test data from ChatGPT, as suggested by Prof. Zink. SQL for these tables is in the SQL folder as title: superfan_band_and_roadies.sql
12-Aug	Made conceptual diagram for artists and bands.
25-Aug	Started working on an ERD with artists and bands.
26-Aug	Created an up/down script for artists and bands. Passed to Zach and Nikki to add their components.
29-Aug	Started a shell for our paper.
10-Sep	Updated paper with latest logical models, fine tuned the narrrative, updated table descriptions.
12-Sep	Finished final paper after input from team.

Zach Brand Log

Date	Work Done
7/28/2025	Project Brain Storming doc - Dan created a sharepoint folder for us to dump all documents in
8/4 in class	Met with group discussing project layout
8/5/2025	Completed my table conversion and data pull for the event details, venue, and parking lots
8/11/2025 in class	Discussed SQL structure
8/19/2025	Scripted all data tables and began foreign keys but did not complete. Dan and Nikki completed the rest of the database creation. I set up the up/down script layout
8/25/2025 in class	Another touch base in class about project progress
9/1/25 in class	Work divvied up work and prepared for the final touches of the final project
9/2/2025	Fixed some errors on the database script. Successful cleared errors and it ran perfectly. No more errors on the database creation
9/7/2025	Finished my 3 sql query builds to query against the database we created. Did a procedure creation, a view, and a function.
9/8/2025	Final class meeting, discussed the paper writing and touching up the final project
9/10/2025	Completed the merging of everyone's SQL query scripts to query the database, completed my work log. Working to complete my portion of the final paper write up tonight

Nikki Orue Log

8/6	created proposed specs for lodging, venues, restaurants, parking lot/transportation tables
8/14	made conceptual + logical model for lodging, tour staff, restaurants
8/31	added fake data to lodging, restaurant, and staff related tables + my portion of up/down script
9/1	Put together team's conceptual/logical model diagrams
9/11	created sample UI mockups, created power point presentation

Appendix C – Full SQL Code

Please find the full executable SQL from the included scripts. The Up/Down script creates the database. The Questions SQL file implements views and stored procedures that answer some of the business questions referenced earlier. The code is included here, as well as in separate files submitted with this paper.

Paper continues on next page

Up/Down Script

-- Execute this once to create the database:

-- Functions and views for Bands, Artists and Instruments -- Note: Run each function / view / index itself indivually, then the test case for it. -- Dan Pacheco

use superfan; go

-- 1. FUNCTION: BandArtists. Gives you a table that shows which artists are in which bands

create or alter function dbo.fn_BandArtists (@band_id int) returns table as return

(select a.artist_id, a.artist_name, a.band_id from dbo.artists as a where a.band_id = @band_id); go

-- Test BandArtists function to show artists who are in bands based on band number in parens -- Change number to get different results. Note that some list multiple members:

select * from dbo.fn_BandArtists(3);

select * from dbo.fn_BandArtists(7);

select * from dbo.fn_BandArtists(11);

-- 2. VIEW artists_with_instruments: Shows a roster of artists in bands with their instruments

use superfan go

create or alter view dbo.v_artists_with_instruments as select b.band_id, b.band_name, a.artist_id, a.artist_name, i.instrument_id, i.instrument_name from dbo.artists as a

join dbo.bands as b on b.band_id = a.band_id

join dbo.artist_instruments as ai on ai.artist_id = a.artist_id

join dbo.instruments as i on i.instrument_id = ai.instrument_id; go

-- Test the artists_with_instruments view:

select * from dbo.v_artists_with_instruments order by band_name, artist_name, instrument_name

GO -- 3. VIEW v_band_roster_with_instruments. -- Shows band roster with aggregated instruments per artist

CREATE or ALTER view dbo.v_band_roster_with_instruments as select b.band_id, b.band_name, a.artist_id, a.artist_name, STRING_AGG(i.instrument_name, ', ') as instruments_played from dbo.artists as a join dbo.bands as b on b.band_id = a.band_id left join dbo.artist_instruments as ai on ai.artist_id = a.artist_id left join dbo.instruments as i on i.instrument_id = ai.instrument_id group by b.band_id, b.band_name, a.artist_id, a.artist_name; go

-- Test the v_band_roster_with_instruments view

select * from dbo.v_band_roster_with_instruments

go

-- 4. STORED PROCEDURE p_search_artists -- Lets you search for artists by text in the name, or the band -- Uses like to help you find a band even if you only know part of the name.

CREATE OR ALTER FUNCTION dbo.fn_Normalize (@s NVARCHAR(4000)) RETURNS NVARCHAR(4000) AS BEGIN RETURN LOWER(LTRIM(RTRIM(ISNULL(@s, N')))); END GO

create or alter PROCEDURE dbo.p_search_artists @q nvarchar(200) = null, -- free-text for artist/band @band_id int = null -- optional exact band filter as begin set nocount on; declare @qNorm nvarchar(4000) = dbo.fn_Normalize(@q);

```
select
    v.band_id,
    v.band_name,
    v.artist_id,
    v.artist_name,
    v.instruments_played
from dbo.v_band_roster_with_instruments as v
where (@band_id is null or v.band_id = @band_id)
  and (
        @qNorm is null
     or dbo.fn_Normalize(v.artist_name) like '%' + @qNorm + '%'
     or dbo.fn_Normalize(v.band_name)   like '%' + @qNorm + '%'
  )
order by v.band_name, v.artist_name;


end go
```

-- Test the p_search_artists procedure

-- Test csase 1) No filters (returns all rows from the view) exec dbo.p_search_artists;

-- Test case 2: Text search (artist or band). -- Find bands in which band member Cassian Reed, Mira Solis, Dante Vale belong. -- Input any part of their names to see a list. Note that Dante Vale is in two bands. exec dbo.p_search_artists @q = N'Cassian'; exec dbo.p_search_artists @q = N'Solis'; exec dbo.p_search_artists @q = N'Vale';

-- Test case 3: Exact band filter, by number exec dbo.p_search_artists @band_id = 5; exec dbo.p_search_artists @band_id = 3; exec dbo.p_search_artists @band_id = 11;

go

-- 5. PROCEDURE p_band_roster_pretty: A list of bands and the members of each band.

create or alter procedure dbo.p_band_roster_pretty @band_ids_csv nvarchar(max) = null, -- comma separated values e.g. '2,5,9' @band_names_csv nvarchar(max) = null, -- comma separated e.g. 'Queen,Pink Floyd' @band_id int = null, -- single band id @band_name nvarchar(200) = null -- single band name as begin set nocount on;

```
declare @target_bands table (band_id int primary key);

if @band_id is not null
   insert into @target_bands(band_id) VALUES (@band_id);

if @band_name is not null
   insert into @target_bands(band_id)
   select b.band_id
   from dbo.bands as b
   where dbo.fn_Normalize(b.band_name) = dbo.fn_Normalize(@band_name);

if @band_ids_csv is not null
   insert into @target_bands(band_id)
   select distinct TRY_CAST(LTRIM(RTRIM(value)) as int)
   from STRING_SPLIT(@band_ids_csv, ',')
   where TRY_CAST(LTRIM(RTRIM(value)) as int) is not null;

if @band_names_csv is not null
   insert into @target_bands(band_id)
   select distinct b.band_id
   from dbo.bands as b
   join (
      select dbo.fn_Normalize(LTRIM(RTRIM(value))) as name_norm
      from STRING_SPLIT(@band_names_csv, ',')
   ) s on dbo.fn_Normalize(b.band_name) = s.name_norm;

if not exists (select 1 from @target_bands)
   insert into @target_bands(band_id)
   select b.band_id from dbo.bands as b;

;with bands_filtered as (
   select b.band_id, b.band_name
   from dbo.bands as b
   join @target_bands as t on t.band_id = b.band_id
),
member_rows as (
   select
      bf.band_id,
      bf.band_name,
      a.artist_id,
      a.artist_name
```

```
    from bands_filtered as bf
    left join dbo.artists as a
      on a.band_id = bf.band_id
)
select *
from (

    select
        bf.band_id,
        bf.band_name,
        cast(null as int)              as artist_id,
        cast(null as nvarchar(200))    as artist_name,
        0                              as row_type,  -- header first
        cast(N'Band: ' + bf.band_name as nvarchar(400)) as line
    from bands_filtered as bf

    union all

    select
        m.band_id,
        m.band_name,
        m.artist_id,
        m.artist_name,
        1 as row_type,
        case
            when m.artist_name is null
                then cast(N'   - (no members)' as nvarchar(400))
            else cast(N'   - ' + m.artist_name as nvarchar(400))
        end as line
    from member_rows as m
) as R
order by R.band_name, R.row_type, R.artist_name;


end go
```

-- TEST Procedure p_band_roster_pretty: Make a table of all selected bands by name, comma-separated

-- Case 1: List members by band name exec dbo.p_band_roster_pretty @band_names_csv = N'Neon River, Static Horizon, Golden Fractals';

-- Caase 2: List members by band ID exec dbo.p_band_roster_pretty @band_ids_csv = N'2, 5, 9';

-- Some indexes for better performance when traveling city to city -- These should help with the common queries artists and tour managers run

-- Index for finding events by date range (tour scheduling) if exists (select * from sys.indexes where name = 'IX_events_date_time_venue' and object_id = object_id('events')) drop index IX_events_date_time_venue on events;

create nonclustered index IX_events_date_time_venue on events (date_time, venue_id) include (event_id, event_location, artist, band_id, venue_cost);

-- TEST: IX_events_date_time_venue select top 3 event_id, date_time, event_location, artist, venue_cost from events where date_time >= '2025-08-01' and date_time < '2025-09-01' order by date_time;

-- Index for finding events by location and date if exists (select * from sys.indexes where name = 'IX_events_location_date' and object_id = object_id('events')) drop index IX_events_location_date on events;

create nonclustered index IX_events_location_date on events (event_location, date_time) include (event_id, venue_id, artist, band_id);

-- TEST: IX_events_location_date select event_id, event_location, date_time, artist from events where event_location = 'New York' order by date_time;

-- Index for finding lodging by event (artists need to know where they're staying) if exists (select * from sys.indexes where name = 'IX_lodging_event_distance' and object_id = object_id('lodging')) drop index IX_lodging_event_distance on lodging;

create nonclustered index IX_lodging_event_distance on lodging (event_id, distance_from_venue) include (lodging_id, lodging_name, lodging_address, lodging_city, lodging_state, cost_per_night, rating, amenities);

-- TEST: IX_lodging_event_distance select l.lodging_name, l.lodging_city, l.distance_from_venue, l.cost_per_night, l.rating from lodging l where l.event_id = 1 order by l.distance_from_venue;

-- Index for finding restaurants by city and cuisine if exists (select * from sys.indexes where name = 'IX_restaurants_city_cuisine' and object_id = object_id('restaurants')) drop index IX_restaurants_city_cuisine on restaurants;

create nonclustered index IX_restaurants_city_cuisine on restaurants (restaurant_city, cuisine_type) include (restaurant_id, restaurant_name, distance_from_venue, food_options, open_time, close_time);

-- TEST: IX_restaurants_city_cuisine select restaurant_name, restaurant_city, cuisine_type, food_options, distance_from_venue from restaurants where cuisine_type = 'Mexican' order by restaurant_city;

--6 Function -- What is the total lodging cost for each artist for a specified event, considering the number of nights stayed and the cost per night of the lodging?

DROP FUNCTION IF EXISTS dbo.CalculateLodgingCost; GO

CREATE FUNCTION dbo.CalculateLodgingCost (@CheckInDate DATE, @CheckOutDate DATE, @CostPerNight DECIMAL(10,2)) RETURNS DECIMAL(10,2) AS BEGIN DECLARE @TotalCost

DECIMAL(10,2); SET @TotalCost = DATEDIFF(DAY, @CheckInDate, @CheckOutDate) * @CostPerNight; RETURN @TotalCost; END; GO

SELECT a.artist_name, l.lodging_name, al.check_in_date, al.check_out_date, l.cost_per_night, dbo.CalculateLodgingCost(al.check_in_date, al.check_out_date, l.cost_per_night) AS total_lodging_cost FROM dbo.artist_lodging al INNER JOIN dbo.artists a ON al.artist_id = a.artist_id INNER JOIN dbo.lodging l ON al.lodging_id = l.lodging_id WHERE al.event_id = 1;

--7 Stored Procedure --What's the full lineup for a band -- each artist and the instruments they play -- Stored Procedure

USE superfan; GO

IF OBJECT_ID('dbo.sp_GetBandLineup', 'P') IS NOT NULL DROP PROCEDURE dbo.sp_GetBandLineup; GO

CREATE PROCEDURE dbo.sp_GetBandLineup @band_id INT AS BEGIN SET NOCOUNT ON;

SELECT a.artist_id, a.artist_name, STRING_AGG(i.instrument_name, ', ') WITHIN GROUP (ORDER BY i.instrument_name) AS instruments FROM dbo.artists AS a LEFT JOIN dbo.artist_instruments AS ai ON ai.artist_id = a.artist_id LEFT JOIN dbo.instruments AS i ON i.instrument_id = ai.instrument_id

WHERE a.band_id = @band_id GROUP BY a.artist_id, a.artist_name ORDER BY a.artist_name; END; GO

EXEC dbo.sp_GetBandLineup @band_id = 7;

--8 View

--Can we get a one-stop "event Overview" showing each event's date/time, venue location, scheduled band, number of artists in that band, how many restaurants are tied to the event, and cheapest/average lodging cost for that event?

--Event overview view

USE superfan; GO

IF OBJECT_ID('dbo.v_EventOverview', 'V') IS NOT NULL DROP VIEW dbo.v_EventOverview; GO

CREATE VIEW dbo.v_EventOverview AS SELECT e.event_id, e.date_time, e.event_location,

v.venue_city, v.venue_state, b.band_name, (SELECT COUNT() FROM dbo.artists a WHERE a.band_id = e.band_id) AS artist_count, (SELECT COUNT() FROM dbo.restaurants r WHERE r.event_id = e.event_id) AS restaurants_nearby, (SELECT MIN(l.cost_per_night) FROM dbo.lodging l WHERE l.event_id = e.event_id) AS min_lodging_cost_per_night, (SELECT AVG(CAST(l.cost_per_night AS DECIMAL(18,2))) FROM dbo.lodging l WHERE l.event_id = e.event_id) AS avg_lodging_cost_per_night FROM dbo.events AS e LEFT JOIN dbo.bands AS b ON b.band_id = e.band_id LEFT JOIN dbo.venues AS v ON v.venue_id = e.venue_id; GO

SELECT * FROM dbo.v_EventOverview ORDER BY date_time;

```
SELECT * FROM dbo.v_EventOverview WHERE venue_city = 'Austin' AND
(min_lodging_cost_per_night IS NOT NULL AND min_lodging_cost_per_night <= 120) ORDER BY
date_time;
```

Questions/Answers Script

Once the script above is executed, you can run individual functions, views and procedures here:

use master GO

-- 1) See current access mode SELECT name, user_access_desc, state_desc FROM sys.databases WHERE name = N'superfan';

-- 2) Force-remove any sessions and return to MULTI_USER ALTER DATABASE [superfan] SET MULTI_USER WITH ROLLBACK IMMEDIATE;

IF DB_ID(N'superfan') IS NULL BEGIN CREATE DATABASE superfan; END GO USE superfan; GO

GO -- Drop foreign keys IF OBJECT_ID (N'fk_events_venue_id', N'F') IS NOT NULL ALTER TABLE events DROP CONSTRAINT fk_events_venue_id;

IF OBJECT_ID (N'FK_artists_bands', N'F') IS NOT NULL ALTER TABLE artists DROP CONSTRAINT FK_artists_bands;

IF OBJECT_ID (N'fk_artist_lodging_artist', N'F') IS NOT NULL ALTER TABLE artist_lodging DROP CONSTRAINT fk_artist_lodging_artist;

IF OBJECT_ID (N'FK_ai_artist', N'F') IS NOT NULL ALTER TABLE artist_instruments DROP CONSTRAINT FK_ai_artist;

IF OBJECT_ID (N'FK_ai_instrument', N'F') IS NOT NULL ALTER TABLE artist_instruments DROP CONSTRAINT FK_ai_instrument;

-- Drop tables DROP TABLE IF EXISTS artist_instruments;

DROP TABLE IF EXISTS artists;

DROP TABLE IF EXISTS instruments;

DROP TABLE IF EXISTS tourstaff_lodging;

DROP TABLE IF EXISTS artist_lodging;

DROP TABLE IF EXISTS lodging;

```
DROP TABLE IF EXISTS restaurants;

DROP TABLE IF EXISTS tour_staff_jobs;

DROP TABLE IF EXISTS jobs;

DROP TABLE IF EXISTS tour_staff;

DROP TABLE IF EXISTS events;

DROP TABLE IF EXISTS parking_lots;

DROP TABLE IF EXISTS venues;

DROP TABLE IF EXISTS bands;

GO -- Up CREATE TABLE lodging ( lodging_id INT NOT NULL PRIMARY KEY, lodging_name
VARCHAR(100) NOT NULL, lodging_address VARCHAR(200) NOT NULL, lodging_city VARCHAR(50)
NOT NULL, lodging_state CHAR(2) NULL, lodging_zipcode VARCHAR(10) NULL, lodging_type
VARCHAR(50) NOT NULL, distance_from_venue DECIMAL(6, 2) NOT NULL, cost_per_night
DECIMAL(10, 2) NOT NULL CHECK (cost_per_night >= 0), amenities VARCHAR(MAX) NULL, rating
DECIMAL(2, 1) NULL CHECK (rating BETWEEN 1 AND 5), event_id INT NOT NULL );

CREATE TABLE tourstaff_lodging ( tourstaff_lodging_id INT NOT NULL PRIMARY KEY, staff_id INT NOT
NULL, lodging_id INT NOT NULL, event_id INT NOT NULL, check_in_date DATE NULL, check_out_date
DATE NULL, room_number VARCHAR(20) NULL );

CREATE TABLE artist_lodging ( artist_lodging_id INT NOT NULL PRIMARY KEY, artist_id INT NOT NULL,
lodging_id INT NOT NULL, event_id INT NOT NULL, check_in_date DATE NULL, check_out_date DATE
NULL, room_number VARCHAR(20) NULL );

CREATE TABLE restaurants ( restaurant_id INT NOT NULL PRIMARY KEY, restaurant_name
VARCHAR(50) NOT NULL, restaurant_address VARCHAR(50) NOT NULL, restaurant_city VARCHAR(50)
NOT NULL, restaurant_state VARCHAR(2) NULL, restaurant_zipcode VARCHAR(10) NULL, cuisine_type
VARCHAR(50) NULL, food_options VARCHAR(50) NULL, open_time TIME NOT NULL, close_time TIME
NOT NULL, distance_from_venue DECIMAL(6, 2) NOT NULL, event_id INT NOT NULL );

CREATE TABLE bands ( band_id INT NOT NULL PRIMARY KEY, band_name VARCHAR(50) NOT NULL );

CREATE TABLE tour_staff ( staff_id INT NOT NULL PRIMARY KEY, staff_name VARCHAR(25) NOT NULL,
staff_email VARCHAR(35) NOT NULL, band_id INT NOT NULL );
```

CREATE TABLE tour_staff_jobs (staff_id INT NOT NULL PRIMARY KEY);

CREATE TABLE jobs (job_id INT NOT NULL PRIMARY KEY, job_name VARCHAR(25) NULL);

CREATE TABLE venues (venue_id INT NOT NULL PRIMARY KEY, venue_city VARCHAR(50) NOT NULL, venue_state VARCHAR(2) NOT NULL, capacity INT NOT NULL, environment VARCHAR(10) NULL, venue_address VARCHAR(50) NOT NULL, contraband_list VARCHAR(100) NULL, parking_lot_id INT NOT NULL, lodging_address VARCHAR(50) NOT NULL, lodging_name VARCHAR(50) NOT NULL, venue_security VARCHAR(3) NULL, lodging_id INT NOT NULL);

CREATE TABLE events (event_id INT NOT NULL PRIMARY KEY, venue_id INT NOT NULL, date_time DATETIME2 NOT NULL, event_location VARCHAR(25) NOT NULL, venue_cost DECIMAL(12, 2) NOT NULL, artist VARCHAR(50) NOT NULL, band_id INT NOT NULL,);

ALTER TABLE events ADD CONSTRAINT fk_events_venue_id FOREIGN KEY (venue_id) REFERENCES venues (venue_id);

GO CREATE TABLE parking_lots (parking_lot_id INT NOT NULL PRIMARY KEY, parking_lot_address VARCHAR(50) NOT NULL, capacity INT NOT NULL, parking_lot_security VARCHAR(3) NULL, venue_city VARCHAR(25) NOT NULL);

CREATE TABLE artists (artist_id INT NOT NULL PRIMARY KEY, artist_name VARCHAR(100) NOT NULL, band_id INT NULL, CONSTRAINT FK_artists_bands FOREIGN KEY (band_id) REFERENCES bands (band_id));

CREATE TABLE instruments (instrument_id INT NOT NULL PRIMARY KEY, instrument_name VARCHAR(50) NOT NULL);

CREATE TABLE artist_instruments (artist_id INT NOT NULL, instrument_id INT NOT NULL, CONSTRAINT PK_artist_instruments PRIMARY KEY (artist_id, instrument_id), CONSTRAINT FK_ai_artist FOREIGN KEY (artist_id) REFERENCES artists (artist_id), CONSTRAINT FK_ai_instrument FOREIGN KEY (instrument_id) REFERENCES instruments (instrument_id));

-- enforce unique lookups by name ALTER TABLE artists ADD CONSTRAINT UQ_artists_name UNIQUE (artist_name);

ALTER TABLE instruments ADD CONSTRAINT UQ_instruments_name UNIQUE (instrument_name);

-- Bands INSERT INTO bands (band_id, band_name) VALUES (1, 'The Midnight Owls'), (2, 'Crimson Echoes'), (3, 'Neon River'), (4, 'Static Horizon'), (5, 'Velvet Tides'), (6, 'The Solar Nomads'), (7, 'Echo

Lantern'), (8, 'Paper Satellites'), (9, 'Shadow Carousel'), (10, 'The Amber Chords'), (11, 'Broken Compass'), (12, 'Moonlit Engines'), (13, 'The Fireflies'), (14, 'Desert Mirage'), (15, 'Golden Fractals'), (16, 'Iron Petals'), (17, 'The Velvet Storm'), (18, 'Emerald Skies'), (19, 'Storm Harbor'), (20, 'Crystal Pines');

-- Artists linked to bands INSERT INTO artists (artist_id, artist_name, band_id) VALUES (1, 'Aria Vega', 1), (2, 'Liam Cross', 1), (3, 'Nova Lane', 2), (4, 'Kai Rivers', 2), (5, 'Mira Solis', 3), (6, 'Orion Blake', 3), (7, 'Jade Monroe', 4), (8, 'Axel Dusk', 5), (9, 'Elara Quinn', 6), (10, 'Zephyr Cain', 7), (11, 'Nina Frost', 8), (12, 'Dante Vale', 9), (13, 'Lyra Skye', 10), (14, 'Cassian Reed', 11), (15, 'Sierra Vale', 12), (16, 'Rowan Stone', 13), (17, 'Ivy Mars', 14), (18, 'Phoenix Grey', 15), (19, 'Aurora Flint', 16), (20, 'Jax Wilder', 17);

-- Create instruments INSERT INTO instruments (instrument_id, instrument_name) VALUES (1, 'Synth Harp'), (2, 'Solar Drums'), (3, 'Crystal Guitar'), (4, 'Echo Flute'), (5, 'Bass Engine');

-- Assign instruments to artists by name INSERT INTO artist_instruments (artist_id, instrument_id) SELECT a.artist_id, i.instrument_id FROM (VALUES ('Aria Vega', 'Crystal Guitar'), ('Liam Cross', 'Solar Drums'), ('Nova Lane', 'Synth Harp'), ('Kai Rivers', 'Crystal Guitar'), ('Mira Solis', 'Echo Flute'), ('Orion Blake', 'Bass Engine'), ('Jade Monroe', 'Crystal Guitar'), ('Axel Dusk', 'Solar Drums'), ('Elara Quinn', 'Synth Harp'), ('Zephyr Cain', 'Bass Engine'), ('Nina Frost', 'Echo Flute'), ('Dante Vale', 'Solar Drums'), ('Lyra Skye', 'Crystal Guitar'), ('Cassian Reed', 'Synth Harp'), ('Sierra Vale', 'Bass Engine'), ('Rowan Stone', 'Crystal Guitar'), ('Ivy Mars', 'Echo Flute'), ('Phoenix Grey', 'Solar Drums'), ('Aurora Flint', 'Synth Harp'), ('Jax Wilder', 'Bass Engine')) x (artist_name, instrument_name) JOIN artists a ON a.artist_name = x.artist_name JOIN instruments i ON i.instrument_name = x.instrument_name;

-- Venues Table INSERT INTO venues (venue_id, venue_city, venue_state, capacity, environment, venue_address, contraband_list, parking_lot_id, lodging_address, lodging_name, venue_security, lodging_id) VALUES (1, 'New York', 'AZ', 21371, 'Indoor', '492 Joanna Estates', 'Drugs, Pets', 2, '9944 Davis Ridge', 'Davis-Robertson Inn', 'Yes', 1), (2, 'Los Angeles', 'AZ', 40221, 'Mixed', '8667 Ramos Manor', 'Weapons, Alcohol', 3, '87221 Moore Islands Suite 078', 'Carey, Simpson and Jenkins Inn', 'Yes', 2), (3, 'San Diego', 'PA', 7545, 'Outdoor', '093 Todd Green', 'Drugs, Weapons', 6, '700 Hunter Mountains', 'Smith, Anderson and Franco Inn', 'No', 3), (4, 'San Antonio', 'IL', 19322, 'Indoor', '904 Keith River Apt. 789', 'Outside Food, Alcohol', 7, '904 Mary Expressway Suite 561', 'Andrews-Gutierrez Inn', 'Yes', 4), (5, 'San Jose', 'AZ', 11439, 'Indoor', '270 Alvarado Trafficway Suite 286', 'Alcohol, Outside Food', 6, '255 David Vista', 'Jones, Atkins and Daniel Inn', 'No', 5), (6, 'Philadelphia', 'TX', 19113, 'Indoor', '21265 Nancy Extension Suite 568', 'Outside Food, Drugs', 9, '15423 Riley Skyway Apt. 459', 'Hoover-Griffin Inn', 'Yes', 6), (7, 'San Jose', 'AZ', 12473, 'Indoor', '38141 Lee Villages Suite 390', 'Pets, Drugs', 3, '007 Judy Green', 'Novak Inc Inn', 'No', 7), (8, 'Philadelphia', 'PA', 45833, 'Mixed', '386

Samantha Pike Apt. 145', 'Weapons, Drugs', 5, '341 Michelle Mall Apt. 915', 'Hudson-Stewart Inn', 'Yes', 8), (9, 'San Antonio', 'CA', 34981, 'Outdoor', '384 Brian Road', 'Drugs, Outside Food', 10, '94663 Martinez Turnpike Suite 536', 'Haney, Mcgee and Jones Inn', 'No', 9), (10, 'San Jose', 'NY', 15747, 'Indoor', '684 Christina Dam', 'Weapons, Outside Food', 6, '96721 Jeremy Squares', 'Sullivan and Sons Inn', 'Yes', 10);

INSERT INTO events (event_id, venue_id, date_time, event_location, venue_cost, artist, band_id) VALUES (1, 1, '2025-09-10 19:30:00', 'New York', 150.00, 'Aria Vega', 1), (2, 2, '2025-06-11 18:00:00', 'Philadelphia', 60.00, 'Mira Solis', 3), (3, 2, '2025-05-06 20:00:00', 'Philadelphia', 60.00, 'Mira Solis', 3), (4, 3, '2025-07-11 21:00:00', 'San Jose', 75.00, 'Axel Dusk', 5), (5, 4, '2025-04-26 17:00:00', 'Dallas', 49.00, 'Zephyr Cain', 7), (6, 3, '2025-08-15 19:00:00', 'San Jose', 76.00, 'Liam Cross', 1), (7, 1, '2025-08-06 20:30:00', 'New York', 150.00, 'Elara Quinn', 6), (8, 1, '2025-08-01 20:30:00', 'New York', 150.00, 'Elara Quinn', 6), (9, 5, '2025-03-23 18:30:00', 'Phoenix', 200.00, 'Orion Blake', 3), (10, 5, '2025-08-23 19:00:00', 'Phoenix', 200.00, 'Jade Monroe', 4);

INSERT INTO parking_lots (parking_lot_id, parking_lot_address, capacity, parking_lot_security, venue_city) VALUES (1, '49382 Johnson Garden', 381, 'No', 'New York'), (2, '2519 Pacheco Passage', 747, 'Yes', 'Philadelphia'), (3, '825 Smith Rapids Suite 835', 630, 'Yes', 'San Jose'), (4, '145 Kevin Pike', 178, 'No', 'New York'), (5, '41365 Amber Club', 152, 'No', 'New York'), (6, '8357 Estrada Skyway Apt. 758', 484, 'No', 'Philadelphia'), (7, '3816 Nancy Corner Suite 809', 91, 'Yes', 'Philadelphia'), (8, '273 Charles Corner Apt. 013', 368, 'No', 'Phoenix'), (9, '49257 Christian Cove', 84, 'Yes', 'Dallas'), (10, '2777 Melissa Forges', 451, 'Yes', 'San Jose');

--Lodging Table data INSERT INTO lodging (lodging_id, lodging_name, lodging_address, lodging_city, lodging_state, lodging_zipcode, lodging_type, distance_from_venue, cost_per_night, amenities, rating, event_id) VALUES (1, 'Davis-Robertson Inn', '9944 Davis Ridge', 'New York', 'AZ', '10001', 'Hotel', 1.2, 150.00, 'WiFi,Pool,Gym', 4.5, 1), (2, 'Carey, Simpson and Jenkins Inn', '87221 Moore Islands Suite 078', 'Los Angeles', 'AZ', '90001', 'Hotel', 2.8, 120.50, 'WiFi,Free Breakfast,Pet Friendly', 4.2, 2), (3, 'Smith, Anderson and Franco Inn', '700 Hunter Mountains', 'San Diego', 'PA', '92101', 'Hotel', 0.9, 135.75, 'Gym,Garden,Parking', 4.0, 4), (4, 'Andrews-Gutierrez Inn', '904 Mary Expressway Suite 561', 'San Antonio', 'IL', '78201', 'Hotel', 1.4, 130.00, 'Pool,WiFi', 3.8, 5);

--Restaurant Table data INSERT INTO restaurants (restaurant_id, restaurant_name, restaurant_address, restaurant_city, restaurant_state, restaurant_zipcode, cuisine_type, food_options, open_time, close_time, distance_from_venue, event_id) VALUES (1, 'Sunset Diner', '123 Sunset Blvd', 'New York', 'AZ', '10002', 'Diner', 'Vegetarian, Gluten-Free', '06:00:00', '23:00:00', 0.7, 1), (2, 'Blue Moon BBQ', '456 Blue St', 'Los Angeles', 'AZ', '90002', 'Barbecue', 'Vegan', '11:00:00', '02:00:00', 1.2, 2), (3, 'Olive Garden', '789 Olive Rd', 'San Diego', 'PA', '92102', 'Italian', 'Vegetarian', '10:00:00',

'22:00:00', 0.9, 3), (4, 'Taco Haven', '321 Taco Ave', 'San Antonio', 'IL', '78202', 'Mexican', 'Vegan, Gluten-Free', '09:00:00', '00:00:00', 1.1, 4);

--Jobs table data INSERT INTO jobs (job_id, job_name) VALUES (1, 'Stage Setup'), (2, 'Audio Technician'), (3, 'Lighting'), (4, 'Road Manager'), (5, 'Security');

--Tour Staff Table data INSERT INTO tour_staff (staff_id, staff_name, staff_email, band_id) VALUES (1, 'Jackson Miller', 'jackson.miller@example.com', 1), (2, 'Samantha Brooks', 'samantha.brooks@example.com', 2), (3, 'Ethan Wright', 'ethan.wright@example.com', 3), (4, 'Ava Roberts', 'ava.roberts@example.com', 4), (5, 'Mason Clark', 'mason.clark@example.com', 5), (6, 'Olivia Lewis', 'olivia.lewis@example.com', 6), (7, 'Noah Walker', 'noah.walker@example.com', 7), (8, 'Isabella Hall', 'isabella.hall@example.com', 8), (9, 'Lucas Allen', 'lucas.allen@example.com', 9), (10, 'Mia Young', 'mia.young@example.com', 10);

-- Assign staff jobs INSERT INTO tour_staff_jobs (staff_id) VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

-- Assign artist_lodging and tourstaff_lodging sample data INSERT INTO artist_lodging (artist_lodging_id, artist_id, lodging_id, event_id, check_in_date, check_out_date, room_number) VALUES (1, 1, 1, 1, '2025-09-09', '2025-09-11', '201'), (2, 2, 1, 1, '2025-09-09', '2025-09-11', '202'), (3, 3, 2, 2, '2025-06-10', '2025-06-12', '305'), (4, 4, 2, 2, '2025-06-10', '2025-06-12', '306');

INSERT INTO tourstaff_lodging (tourstaff_lodging_id, staff_id, lodging_id, event_id, check_in_date, check_out_date, room_number) VALUES (1, 1, 1, 1, '2025-09-09', '2025-09-11', '101'), (2, 2, 2, 2, '2025-06-10', '2025-06-12', '102'), (3, 3, 3, 4, '2025-07-10', '2025-07-12', '103');

-- Adding missing constraints ALTER TABLE lodging ADD CONSTRAINT fk_lodging_event FOREIGN KEY (event_id) REFERENCES events (event_id);

ALTER TABLE tour_staff ADD CONSTRAINT fk_tour_staff_band FOREIGN KEY (band_id) REFERENCES bands (band_id);

ALTER TABLE artist_lodging ADD CONSTRAINT fk_artist_lodging_artist FOREIGN KEY (artist_id) REFERENCES artists (artist_id);

ALTER TABLE artist_lodging ADD CONSTRAINT fk_artist_lodging_lodging FOREIGN KEY (lodging_id) REFERENCES lodging (lodging_id);

ALTER TABLE artist_lodging ADD CONSTRAINT fk_artist_lodging_event FOREIGN KEY (event_id) REFERENCES events (event_id);

```
ALTER TABLE tourstaff_lodging ADD CONSTRAINT fk_tourstaff_lodging_staff FOREIGN KEY (staff_id)
REFERENCES tour_staff (staff_id);

ALTER TABLE tourstaff_lodging ADD CONSTRAINT fk_tourstaff_lodging_lodging FOREIGN KEY
(lodging_id) REFERENCES lodging (lodging_id);

ALTER TABLE tourstaff_lodging ADD CONSTRAINT fk_tourstaff_lodging_event FOREIGN KEY (event_id)
REFERENCES events (event_id);

-- Show all SELECT * FROM dbo.artist_instruments;

SELECT * FROM dbo.artist_lodging;

SELECT * FROM dbo.artists;

SELECT * FROM dbo.bands;

SELECT * FROM dbo.events;

SELECT * FROM dbo.instruments;

SELECT * FROM dbo.jobs;

SELECT * FROM dbo.lodging;

SELECT * FROM dbo.parking_lots;

SELECT * FROM dbo.restaurants;

SELECT * FROM dbo.tour_staff;

SELECT * FROM dbo.tour_staff_jobs;

SELECT * FROM dbo.tourstaff_lodging;

SELECT * FROM dbo.venues;

-- Count the rows in populated tables SELECT COUNT(*) AS bands FROM bands;

SELECT COUNT(*) AS artists FROM artists;

SELECT COUNT(*) AS instruments FROM instruments;
```

```
SELECT TOP (10) ai.*, a.artist_name, i.instrument_name FROM artist_instruments ai JOIN artists a ON
a.artist_id = ai.artist_id JOIN instruments i ON i.instrument_id = ai.instrument_id ORDER BY
a.artist_name;
```