Relational Design & Normalization Report

1. Conceptual Design to Relational Schema

The conceptual design (E-R Model) was converted into a relational schema by creating tables for each entity and establishing relationships using foreign keys.

Entities Mapped to Tables:

- **Organizer** $\rightarrow$ `organizers`
- **Venue** $\rightarrow$ `venues`
- **Event** $\rightarrow$ `events`
- **TicketType** $\rightarrow$ `ticket_types`
- **Attendee** $\rightarrow$ `attendees`
- **Ticket** $\rightarrow$ `tickets`
- **Payment** $\rightarrow$ `payments`
- **PromoCode** $\rightarrow$ `promo_codes`

Relationships Mapped:

- **M:N Relationship**: The relationship between `payments` and `tickets` is Many-to-Many (one payment can cover multiple tickets, and theoretically a split payment could cover one ticket, though here it's modeled as a bundle). This was resolved using a junction table `payment_tickets` .
- **1:N Relationships**: Handled by placing the Primary Key of the "1" side as a Foreign Key in the "N" side (e.g., `venue_id` in `events` table).

---

2. Normalization Process

The database schema has been normalized to **Third Normal Form (3NF)** and **Boyce-Codd Normal Form (BCNF)** to reduce redundancy and improve data integrity.

Step 1: First Normal Form (1NF)

Goal: Ensure atomicity (no repeating groups, atomic values).

- **Action**: All attributes were designed to hold atomic values.
  - *Example*: Instead of storing a list of ticket prices in the `events` table (e.g., "VIP: $100, GA: $50"), we created a separate `ticket_types` table where each row represents a single ticket type with a single price.
  - *Example*: `attendees` name is split into `first_name` and `last_name` for better atomicity.
  - *Example*: Addresses in `venues` are split into `address` , `city` , and `state` .

Step 2: Second Normal Form (2NF)

Goal: Eliminate partial dependencies (all non-key attributes must depend on the *entire* primary key).

- **Action**: This primarily applies to tables with composite primary keys.
  - *Analysis of `payment_tickets`* : The PK is `(payment_id, ticket_id)` . This is a pure junction table with no additional non-key attributes, so it automatically satisfies 2NF.
  - *Analysis of other tables*: All other tables (`events` , `venues` , etc.) have a single-column surrogate Primary Key (`id`), so they automatically satisfy 2NF because there are no

composite keys to create partial dependencies.

Step 3: Third Normal Form (3NF)

Goal: Eliminate transitive dependencies (non-key attributes should not depend on other non-key attributes).

- **Action**: We ensured that non-key columns depend *only* on the Primary Key.
    - **Refactoring `events` table**:
        - *Problem*: If we stored `venue_name` and `venue_address` directly in the `events` table, they would depend on `venue_id` (a non-key attribute in `events`), not just the `event_id`.
        - *Solution*: We moved venue details to a separate `venues` table and referenced it via `venue_id`. Now, `events.venue_id` determines the venue, and `venues.id` determines the address. No transitive dependency exists in `events`.
    - **Refactoring `tickets` table**:
        - *Problem*: If we stored `ticket_price` and `ticket_perks` in the `tickets` table, they would depend on the type of ticket, not the specific ticket instance.
        - *Solution*: We created `ticket_types` to store `price`, `description`, and `perks`. The `tickets` table only references `ticket_type_id`.
    - **Refactoring `payments` table**:
        - *Problem*: Storing `promo_code_discount` directly in `payments` would create redundancy if the discount rules changed.
        - *Solution*: We reference `promo_code_id`. The `payments` table stores the final `amount` (which is historical data and thus distinct from the current promo rule), but the logic for the discount is derived from the `promo_codes` table.

Boyce-Codd Normal Form (BCNF)

Goal: Every determinant must be a candidate key.

- **Analysis**:
    - In our schema, all tables use surrogate Primary Keys (`id`).
    - All functional dependencies are of the form `id -> attributes`.
    - Since `id` is a candidate key (and the primary key), all tables satisfy BCNF.
    - *Exception Check*: `ticket_types` has a unique constraint on `(event_id, type_name)`. This means `(event_id, type_name)` determines `price`, `quantity`, etc. Since `(event_id, type_name)` is a candidate key (enforced by UNIQUE constraint), BCNF is preserved.

---

3. Schema Summary

Table	Normal Form	Justification
`organizers`	3NF/BCNF	PK `id` determines all fields. No transitive dependencies.
`venues`	3NF/BCNF	PK `id` determines location/capacity.

`events`	3NF/BCNF	PK `id` determines event details. Venue/Organizer info moved to separate tables.
`ticket_types`	3NF/BCNF	PK `id` determines price/perks. Specific to an event.
`attendees`	3NF/BCNF	PK `id` determines user info.
`tickets`	3NF/BCNF	PK `id` determines specific seat/status. Relies on `ticket_types` for pricing info.
`payments`	3NF/BCNF	PK `id` determines transaction details.
`promo_codes`	3NF/BCNF	PK `id` determines discount rules.

4. Integrity Constraints

To ensure the relational design remains valid, the following constraints were implemented in SQL:

1. **Entity Integrity**: All tables have a `PRIMARY KEY`.
2. **Referential Integrity**: `FOREIGN KEY` constraints with `ON DELETE/UPDATE` rules ensure no orphaned records (e.g., you cannot delete an Event if Tickets exist for it, unless cascading is explicitly allowed).
3. **Domain Integrity**: `ENUM` types for status/categories and `CHECK` constraints (e.g., `capacity > 0`, `end_time > start_time`) ensure valid data values.