# Output

## Brief explanation of the design and implementation choices:

**1. Users Table**

* **Columns:**
  + id: A unique identifier for each user, automatically incremented.
  + name: Stores the user's name.
  + email: Stores the user's email, which is unique and cannot be null.
* **Implementation Choices:**
  + The email column is marked as UNIQUE to prevent duplicate registrations and ensure that each user is distinct.

**2. Tickets Table**

* **Columns:**
  + ticket\_id: A unique identifier for each ticket, automatically incremented.
  + name: Describes the name or type of ticket (e.g., event name).
  + price: The price of the ticket.
  + total\_quantity: The total number of tickets available.
  + remaining\_quantity: The number of tickets still available for booking.
  + status: Tracks the availability of the ticket, with possible values being 'AVAILABLE' or 'BOOKED'.
* **Implementation Choices:**
  + The remaining\_quantity is used to manage inventory and ensure that bookings do not exceed available tickets.
  + The status helps differentiate between tickets that are still available and those that have been fully booked.

**3. Bookings Table**

* **Columns:**
  + booking\_id: A unique identifier for each booking, automatically incremented.
  + user\_id: References the user making the booking, ensuring a relationship between the booking and the user.
  + ticket\_id: References the ticket being booked, ensuring a relationship between the booking and the ticket.
  + quantity: The number of tickets being booked.
  + booking\_time: The timestamp when the booking is made.
  + confirmation\_time: The timestamp when the booking is confirmed (can be NULL until confirmation).
  + status: Tracks the state of the booking, with possible values being 'PENDING', 'CONFIRMED', or 'CANCELED'.
* **Implementation Choices:**
  + The status column helps in tracking the lifecycle of a booking, allowing for actions like automatic cancellations or confirmations.
  + The FOREIGN KEY constraints on user\_id and ticket\_id ensure data integrity by linking bookings to valid users and tickets.

**4. Payments Table**

* **Columns:**
  + payment\_id: A unique identifier for each payment, automatically incremented.
  + booking\_id: References the associated booking, ensuring a one-to-one relationship between booking and payment.
  + amount: The amount paid for the booking.
  + payment\_time: The timestamp when the payment was made.
  + payment\_method: The method used for payment, with options like 'CREDIT\_CARD', 'BANK\_TRANSFER', or 'CASH'.
* **Implementation Choices:**
  + The booking\_id is marked as UNIQUE to enforce a one-to-one relationship, ensuring each booking has a single corresponding payment.
  + The payment\_method provides flexibility in handling different payment methods.

**5. Refunds Table**

* **Columns:**
  + refund\_id: A unique identifier for each refund, automatically incremented.
  + booking\_id: References the associated booking, ensuring a one-to-one relationship between booking and refund.
  + refund\_amount: The amount refunded to the user.
  + refund\_time: The timestamp when the refund was processed.
* **Implementation Choices:**
  + Similar to the Payments table, the booking\_id is UNIQUE to ensure each booking has a single corresponding refund, maintaining data integrity.
  + The separation of the Refund table allows for easy tracking and processing of refunds in case of cancellations.

## Edge cases that possibly happen:

Booking Spam:

* Solution: Implement rate limiting on booking actions, CAPTCHA during booking, and automatic cancellation of unconfirmed bookings after a set time.

Double Booking Attempts

* Solution: Use front-end and back-end checks to prevent duplicate submissions, such as disabling the booking button after the first click and checking for existing bookings before creating a new one.

Simultaneous Cancellations and Bookings

* Solution: Use proper database transaction isolation levels to handle concurrent operations, ensuring data consistency.

Session Expiration During Booking

* Solution: Implement clear session management and notify users when their session is about to expire, with an option to extend it.