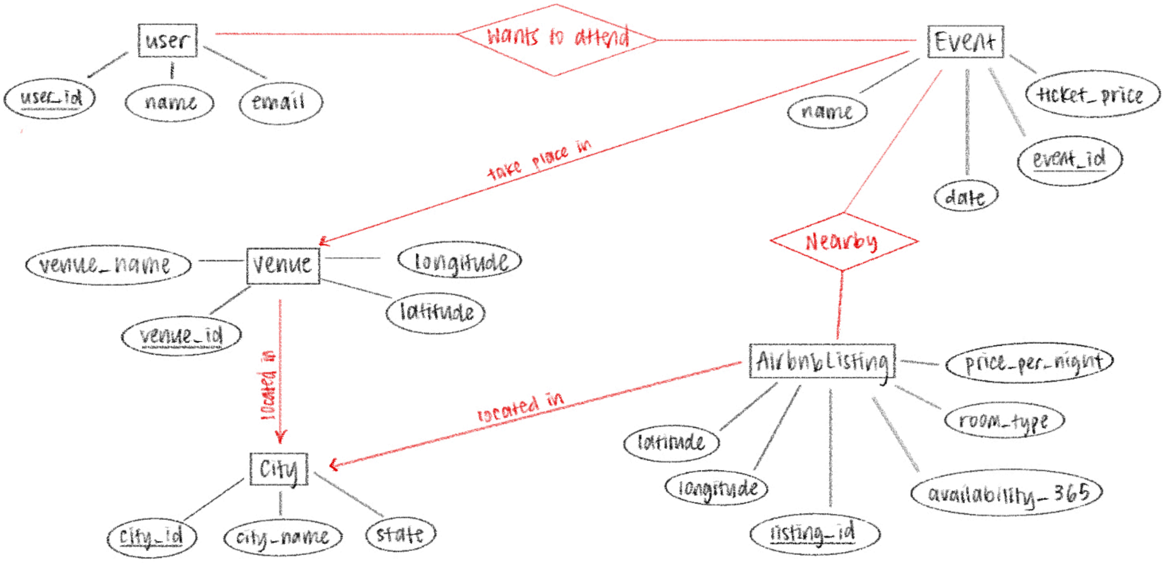
PT1 Stage 2



Entities

1. **User** - represents people using the application
   a. user_id [primary]
   b. name
   c. email
2. **Event** - concerts or live events from Ticketmaster
   a. event_id [primary]
   b. name
   c. date
   d. ticket_price
   e. venue_id [foreign → Venue.venue_id]
3. **Venue** - physical location of events
   a. venue_id [primary]
   b. venue_name
   c. city_id [foreign → City.city_id]
   d. latitude
   e. longitude
4. **City** - unifies Ticketmaster and Airbnb data
   a. city_id [primary]
   b. city_name
   c. state

5. **AirbnbListing** - lodging options from InsideAirbnb
    a. listing_id [primary]
    b. city_id [foreign → City.city_id]
    c. latitude
    d. longitude
    e. price_per_night
    f. availability_365
    g. room_type

Relationships

- User ↔ Event → WantsToAttend (M:N)
    - Bridge entity: WantsToAttend(user_id, event_id)
    - Optional attribute: none
- Event ↔ AirbnbListing → Nearby (M:N)
    - Bridge entity: Nearby(event_id, listing_id)
    - Attributes: total_cost, distance
- Event → Venue (M:1)
    - Each event occurs at one venue
    - Represented by venue_id FK in Event
- Venue → City (M:1)
    - Each venue is located in one city
- AirbnbListing → City (M:1)
    - Each listing is located in one city

Assumptions

1. **Users**
    a. Only a single User entity is maintained
    b. No distinction is made between different types of users
    c. Each user has a unique user_id
2. **Events vs. Venues**
    a. Events are distinct from venues, as each venue can host multiple events over time
    b. Venue information is stored in its own entity, with venue_id referenced in Event as a foreign key
3. **City Entity**
    a. City is separated as an entity to avoid redundancy, because both Venues and AirbnbListings reference it
    b. This ensures consistent storage of city_name and state across the database
4. **Event and Airbnb Relationship**
    a. Modeled as a many-to-many (M:N) relationship with the Nearby bridge table.

b. Includes calculated attributes (distance, total_cost) that depend on both the Event and the AirbnbListing

c. Avoids storing event-specific cost data directly in either the Event or AirbnbListing entities

5. Normalization

a. Each table is in 3NF/BCNF:

i. No partial dependencies - all non-key attributes fully depend on the primary key

ii. No transitive dependencies

b. Derived or calculated attributes (distance, total_cost) are stored in bridge tables rather than in base entities

6. Other Assumptions

a. Each Event occurs at exactly one Venue (M:1)

b. Each Venue belongs to one City (M:1)

c. Each AirbnbListing belongs to one City (M:1)

d. Users can attend multiple events, and events can have multiple attendees (M:N with the WantsToAttend bridge table)

Logical Design Schema

```
User (user_id: INT [PK],
    name: VARCHAR(100),
    email: VARCHAR(100))

WantsToAttend (user_id: INT [FK to User.user_id],
        event_id: INT [FK to Event.event_id],
        PRIMARY KEY(user_id, event_id))

Event (event_id: INT [PK],
     name: VARCHAR(255),
     date: DATE,
     ticket_price: DECIMAL,
     venue_id: INT [FK to Venue.venue_id])

Venue (venue_id: INT [PK],
     venue_name: VARCHAR(255),
     city_id: INT [FK to City.city_id],
     latitude: DECIMAL,
     longitude: DECIMAL)

City (city_id: INT [PK],
   city_name: VARCHAR(100),
   state: VARCHAR(50))

AirbnbListing (listing_id: INT [PK],
          city_id: INT [FK to City.city_id],
          latitude: DECIMAL,
          longitude: DECIMAL,
          price_per_night: DECIMAL,
          availability_365: INT,
          room_type: VARCHAR(50))

Nearby (event_id: INT [FK to Event.event_id],
        listing_id: INT [FK to AirbnbListing.listing_id],
        total_cost: DECIMAL,
        distance: DECIMAL,
        PRIMARY KEY(event_id, listing_id))
```