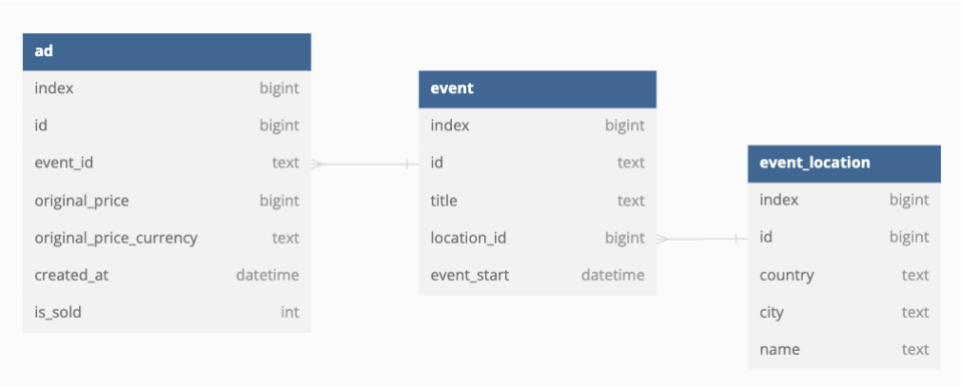
Analytics Engineering Assessment for TicketSwap

1. **Question 1: Looking at these three tables from our database. How would you propose modelling this for analytics purposes?**

Based on the database schema provided, we have three tables that are part of the data model: ad, event, and event_location. For analytics purposes, we would want to model this data in such a way that it is optimized for querying and provides clear insights.



Current Tables:

1. **event**: Contains details about the events.
2. **ad**: Contains information about ticket listings (ads) for events.
3. **event_location**: Contains information about the locations of events

Steps for Modeling

Step 1: Fact Table

Introduce Transaction Fact Table: The transaction table will be the core fact table because it records the events (ticket sales) that are of primary interest for analytics.

- **Attributes:** It should include transaction_id, ad_id (linking to the ad table), buyer_id, seller_id (linking to a potential user table), sale_price, sale_currency, and timestamp of the transaction.

The **ad table** may act as both a fact table and a dimension table, depending on the grain of the analytics.

- **As Fact Table**: Retain listing details, but link it to the transaction table through ad_id.
- **As Dimension Table**: Treat each ad as a dimension of the transaction fact table, providing details about the ticket listed for sale.

Step 2: Dimensional Analysis

- The **ad** table can serve as the dimension table as it records information about the listings and sales of tickets, where each ad is a dimension of the transaction fact table, providing details about the tickets listed for sale.

- The **event** table can be considered a dimension table that describes attributes of events. The **event_location** table, which includes location details, is also a dimension table.

- **Date Dimension Table:** We can a separate date dimension table that contains attributes like year, quarter, month, day, week, etc., which will allow for a more straightforward time-based analysis.

- **User Dimension Table:** We can create a **user** table that contains user-specific attributes such as **user_id**, **name**, **email**, **signup_date**, **location**, and potentially **number_of_transactions**.

Relationships:

1. **Ad to Transaction:**
    - **One-to-One (1-1): Based on the given case data itself, assuming that each ad represents exactly one ticket, then the relationship between ads and transactions simplifies considerably.** Since each ad represents one ticket, each ad can only result in a single transaction when that ticket is sold.

    - **One-to-Many (1-M): If an ad can sell multiple tickets (like a seller offering two tickets in one ad), then it's a 1-M relationship.** Each ad can lead to multiple transactions (each transaction records the sale of one of the multiple tickets).
        - This would lead to including a '**quantity_available'** field in the ad table that indicates how many tickets are available for that ad

2. **Transaction to User**:
    - **Many-to-Many (M-M)**: A single transaction involves one buyer and one seller. However, each user can be involved in multiple transactions, either as a buyer or as a seller. This implies a many-to-many relationship between users and transactions, typically resolved through the transaction table that includes **buyer_id** and **seller_id**.
3. **Ad to Event**:
    - **Many-to-One (M-1)**: Multiple ads can be associated with a single event (e.g., different ticket listings for the same event), but each ad is specific to one event only.
4. **Event to Event Location**:
    - **Many-to-One (M-1):** Typically, an event takes place in one location, and a location can host multiple events over time. Therefore, each record in the event table would be associated with one record in the event_location table, while each event_location could be linked to many records in the event table.

5. **User to Ad**:
    - **One-to-Many (1-M)**: A user can create multiple ads (as a seller of tickets), but each ad is created by one unique user.
6. **User Table Itself**:
    - **One-to-Many (1-M)**: If we consider the buyer and seller roles, a user can be a buyer in many transactions and a seller in many different transactions.

Step 3: Create Primary and Foreign Keys & Indexes

- **Defining Primary Keys:**

    - If the **id** fields are unique for each transaction, ticket, event and location and, it can serve as the primary key.

- **Defining Foreign Keys:**

    - **Transaction to ad**: The transaction table should have an **ad_id** field as a foreign key that references the **id** field of the ad table. This link is essential because each transaction is the result of an ad.
    - **ad to event**: The **event_id** in the **ad** table should be a foreign key that references the primary key of the **event** table.
    - **event to event_location**: The **location_id** in the event table should be a foreign key that references the primary key of the event_location table. This enforces referential integrity between events and their locations.
    - **Transaction to User**: If there are **buyer_id** and **seller_id** fields in the transaction table, they should reference the **user_id** field in the user table. This maintains the relationships between transactions and users.

- **Defining Indexes:**

    - An index on **location_id** in the event table would speed up joins with the event_location table.
    - In the ad table, an index on **event_id** is crucial for joining the **event** table.
    - If there are frequent lookups by **country** or **city** in the event_location table, indexes on these columns would help.
    - Indexes on **buyer_id** and **seller_id** would improve the efficiency of queries related to user activity.
    - Good to mention that indexes consume additional storage space. This needs to be considered, especially in environments where storage cost is a concern.

Step 4: Additional Metrics for Analytical Purposes

- For ad table, we can add **sold_at** and **sale_price** to capture when and how much an ad results in a sale.

    - This suggestion involves adding new records with a new surrogate key (and possible flag column to track the current sale_price) each time a change occurs.

- For ad table, we can calculate and include metrics like **sale_duration** (interval between **created_at** and **sold_at**) and **price_diff** (any change from **original_price** to the sale price).

- We can add **event_popularity_score** to the **event** dimension table as it details an event's characteristics by summarizing related transactions.

  Event Popularity Score = (Total Tickets Sold for Event / Total Tickets Available) * (Total User Set Ticket Alert / Time Until Event Start)

  The first ratio represents the demand, measuring the proportion of tickets sold out of the total available. The second ratio reflects user engagement, such as alert settings, normalized over a time factor.

- Include **ticket_sale_velocity** in the **event** dimension table as well, to describe the rate of ticket sales per event over time.

  Ticket Sale Velocity = Number of Tickets Sold in Time Window/ Duration of Time Window

  Scheduling: The calculation of these metrics would typically be scheduled during off-peak hours if done in batches, or handled by a continuous real-time process, depending on the system's capabilities and requirements.

PS: If ads need to be analyzed in real-time, it would make sense to implement a streaming data pipeline, potentially using AWS Kinesis, to capture real-time changes in ad listings and sales.

PS2: Depending on the database and infrastructure, it's essential to consider whether to store these calculations or compute them on the fly during analysis. Storing them can speed up analysis but may require more storage space and processing during write operations.

Step 5: Automate Data Quality Checks

We need to ensure that the data is consistent and accurate, which is essential for reliable analytics. Hence, we can implement dbt tests for null values, duplicates, and referential integrity to ensure the data remains reliable.