DWH

UBER

Design Data Model for Uber

First let's understand the important business metrics which we need to derive from the data model:

1. Average ride duration and distance per city.
2. Revenue generated per city.
3. Peak times for rides in different locations.
4. Popular routes and destinations.
5. Average rider rating per driver.
6. Average driver rating per rider.
7. Time taken by a driver from ride acceptance to customer pickup.
8. Rate of ride cancellations by riders and drivers.
9. Impact of weather on ride demand.
10. Rider loyalty metrics such as frequency of use, average spend, and length of customer relationship.

Design Data Model for Uber

Grow Data Skills

Now prepare dimension tables:

Drivers
driver_id (PK)
driver_name
driver_phone_number
driver_email
signup_date
vehicle_type
driver_city_id (FK)

Riders
rider_id (PK)
rider_name
rider_phone_number
rider_email
signup_date
loyalty_status
rider_city_id (FK)

Locations
location_id (PK)
city_id (FK)
location_name
latitude
longitude

Cities
city_id (PK)
city_name
state
country

Dates
date_time (PK)
day
month
year
weekday

Weather
weather_id (PK)
city_id (FK)
date_time (FK)
weather_condition (rain, sunny, snowy)
temperature

Design Data Model for Uber

Now prepare fact table:

Rides
ride_id (PK)
driver_id (FK)
rider_id (FK)
start_location_id (FK)
end_location_id (FK)
ride_date_time (FK)
ride_duration
ride_distance
fare
rating_by_driver
rating_by_rider

RideStatus
id (PK)
ride_id (FK)
status
status_time

Design Data Model for Uber

RideStatus table should be Dimension table or Fact?

The **RideStatus** table contains **ride_id**, and this could introduce some potential confusion or challenges. Generally, dimension tables in a data warehouse schema **shouldn't contain foreign keys from fact tables** (like ride_id from the Rides table) because this could result in them having a one-to-one relationship with the fact table, and could lead to the dimension table effectively **becoming another fact table**.

The RideStatus table, in the proposed schema is tracking the different statuses that a ride could go through in its lifecycle. In this case, RideStatus would be better designed as a **slowly changing dimension** table or as a transaction **fact table**, depending on the business requirements.

If you want to track only the current status of each ride, then RideStatus could be a slowly changing dimension with ride_id as a foreign key and status as an attribute.

Alternatively, if you want to track all status changes for every ride, then RideStatus could be designed as a transaction fact table that captures each status update as a new row, with **ride_id**, **status**, and **timestamp** as attributes. In this case, **ride_id** in RideStatus wouldn't be a foreign key from Rides table, but rather a degenerate dimension.

Ultimately, the decision will depend on your specific use case and requirements. In general, it's important to ensure that dimension tables in a star schema contain attributes that can be used to slice and dice the measures stored in the fact table.

Design Data Model for Uber

Grow Data Skills

SQL Queries to get business insights:

1. Average ride duration and distance per city

```
SELECT c.city_name, AVG(r.ride_duration), AVG(r.ride_distance)
FROM Rides r
JOIN Locations l ON r.start_location_id = l.location_id
JOIN Cities c ON l.city_id = c.city_id
GROUP BY c.city_name;
```

2. Revenue generated per city

```
SELECT c.city_name, SUM(r.fare)
FROM Rides r
JOIN Locations l ON r.start_location_id = l.location_id
JOIN Cities c ON l.city_id = c.city_id
WHERE r.status = 'completed'
GROUP BY c.city_name;
```

3. Peak times for rides in different locations

```
SELECT l.location_name, d.hour, COUNT(*)
FROM Rides r
JOIN Locations l ON r.start_location_id = l.location_id
JOIN Dates d ON r.ride_date_time = d.date_time
GROUP BY l.location_name, d.hour
ORDER BY COUNT(*) DESC;
```

Design Data Model for Uber

Grow Data Skills

SQL Queries to get business insights:

4. Popular routes and destinations

```
SELECT l1.location_name AS start_location, l2.location_name AS end_location, COUNT(*) AS number_of_rides
FROM Rides r
JOIN Locations l1 ON r.start_location_id = l1.location_id
JOIN Locations l2 ON r.end_location_id = l2.location_id
GROUP BY l1.location_name, l2.location_name
ORDER BY number_of_rides DESC;
```

5. Average rider rating per driver

```
SELECT d.driver_name, AVG(r.rating_by_driver)
FROM Rides r
JOIN Drivers d ON r.driver_id = d.driver_id
GROUP BY d.driver_name;
```

6. Average driver rating per rider

```
SELECT ri.rider_name, AVG(r.rating_by_rider)
FROM Rides r
JOIN Riders ri ON r.rider_id = ri.rider_id
GROUP BY ri.rider_name;
```

Design Data Model for Uber

Grow Data Skills

SQL Queries to get business insights:

7. Time taken by a driver from ride acceptance to customer pickup

```
SELECT d.driver_id, d.driver_name, r.ride_id,
      TIMEDIFF(pickup_status.time, acceptance_status.time) AS time_to_pickup
FROM RideStatus acceptance_status
JOIN RideStatus pickup_status ON acceptance_status.ride_id = pickup_status.ride_id
JOIN Rides r ON r.ride_id = acceptance_status.ride_id
JOIN Drivers d ON r.driver_id = d.driver_id
WHERE acceptance_status.status = 'accepted'
AND pickup_status.status = 'started';
```

8. Rate of ride cancellations by riders and drivers

```
SELECT
  (SELECT COUNT(DISTINCT ride_id) FROM RideStatus WHERE status = 'cancelled_by_driver') * 1.0 /
  (SELECT COUNT(DISTINCT ride_id) FROM Rides) as driver_cancellation_rate,

  (SELECT COUNT(DISTINCT ride_id) FROM RideStatus WHERE status = 'cancelled_by_rider') * 1.0 /
  (SELECT COUNT(DISTINCT ride_id) FROM Rides) as rider_cancellation_rate;
```

Design Data Model for Uber

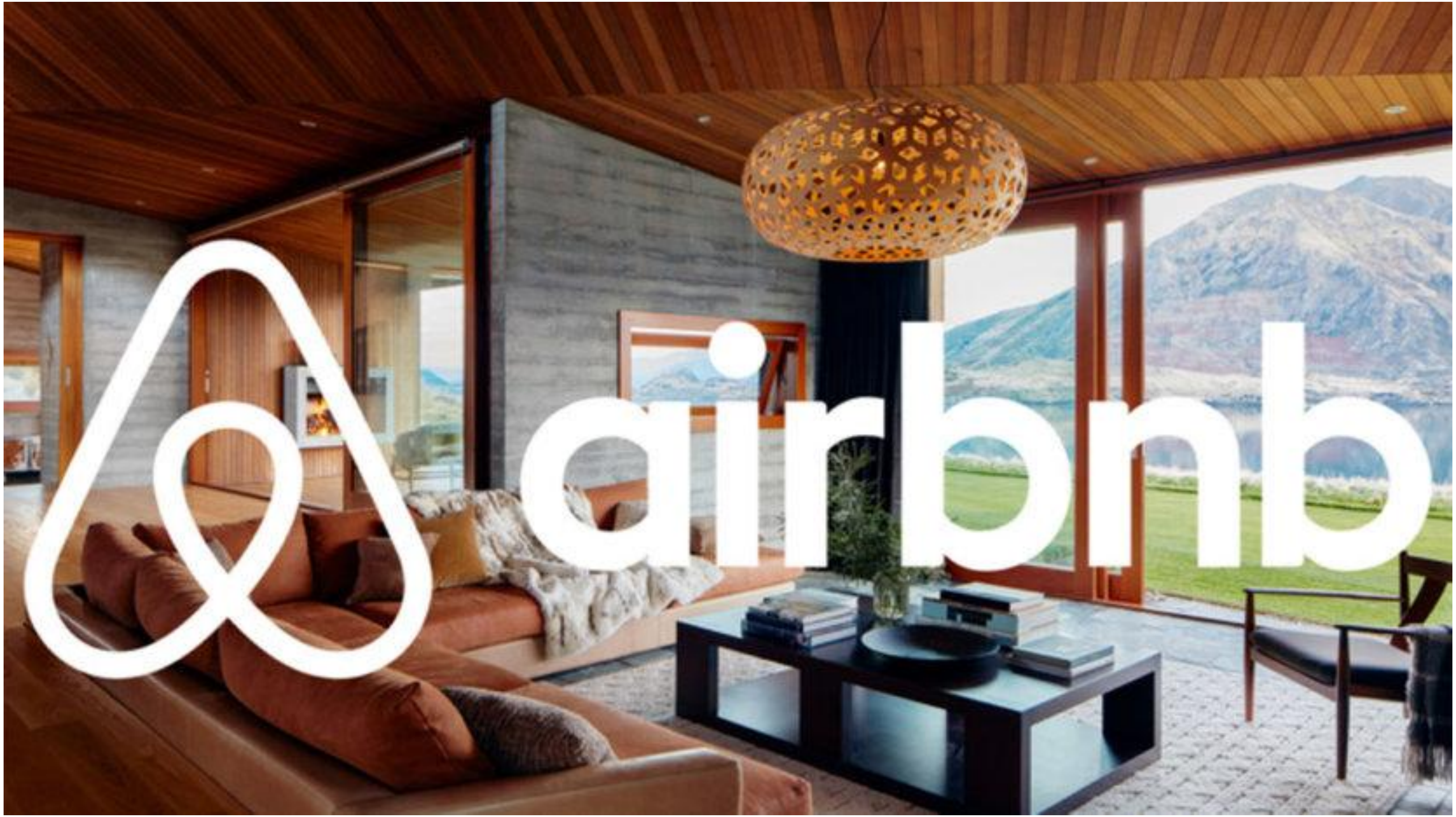
SQL Queries to get business insights:

9. Impact of weather on ride demand

```
SELECT w.weather_condition, COUNT(*) AS number_of_rides
FROM Rides r
JOIN Locations l ON r.start_location_id = l.location_id
JOIN Weather w ON l.city_id = w.city_id
AND DATE_FORMAT(r.ride_date_time, '%Y-%m-%d %H:00:00') = w.date_time
GROUP BY w.weather_condition;
```

10. Rider loyalty metrics

```
SELECT ri.rider_name, COUNT(*), AVG(r.fare), DATEDIFF(MAX(r.ride_date_time), ri.signup_date) as days_since_signup
FROM Rides r
JOIN Riders ri ON r.rider_id = ri.rider_id
GROUP BY ri.rider_name;
```


airbnb

Design Data Model for AirBnb

Grow Data Skills

First let's understand the important business metrics which we need to derive from the data model:

1. Total Revenue Per Host
2. Preferred Payment Method of Guests
3. Daily Revenue
4. Revenue Trends (Monthly)
5. Discrepancies in Amount Paid and Total Price
6. Average Rating of Listings
7. Number of Bookings per Listing
8. Average Response Time of Hosts
9. Total Number of Nights Booked per Listing
10. Most Popular Property Type in Each City

Design Data Model for AirBnb

Grow Data Skills

Now prepare dimension tables:

Listings
listing_id (PK)
host_id (FK)
property_type_id (FK)
city_id (FK)
listing_name
amenities
price_per_night
instant_book_available
is_superhost

Hosts
host_id (PK)
host_name
response_time
response_rate
verification_status

Cities
city_id (PK)
city_name
country_id (FK)

Guests
guest_id (PK)
guest_name
verification_status

Countries
country_id (PK)
country_name

PropertyTypes
property_type_id (PK)
property_type

Time
date_id (PK)
day
month
year
quarter

Design Data Model for AirBnb

Grow Data Skills

Now prepare fact tables:

Bookings
booking_id (PK)
listing_id (FK)
guest_id (FK)
date_id (FK)
number_of_nights
total_price

Reviews
review_id (PK)
guest_id (FK)
listing_id (FK)
date_id (FK)
rating
review_text

Messages
message_id (PK)
guest_id (FK)
host_id (FK)
listing_id (FK)
date_id (FK)
message_text
response_time

Payments
payment_id (PK)
booking_id (FK)
guest_id (FK)
payment_date_id (FK)
payment_method
amount

Design Data Model for AirBnb

Grow Data Skills

Should we consider Messages table as a Fact Table?

In a typical data warehouse model, fact tables usually contain measurable, numerical, or quantitative data, which can be analyzed or aggregated. In this Airbnb data model, the Messages table is **more of an event or transactional fact table**. These types of fact tables are common in data warehouse designs and track the occurrence of a certain type of business event (in this case, a message being sent).

In this particular scenario, the Messages table may not have typical "**measurable**" data like sales figures or quantities. However, it does store the events of messages being sent between hosts and guests. This can be useful in analytical contexts, like tracking the frequency of communication, understanding patterns of communication, or measuring the time it takes for a host to respond to a guest.

For example, an analyst might count the number of messages to measure how active a host is, or calculate the average response time of a host, which could be considered a measure.

Design Data Model for AirBnb

SQL Queries to get business insights:

1. Total Revenue Per Host

```sql
SELECT h.host_id, h.host_name, SUM(p.amount) AS total_revenue
FROM Payments p
JOIN Bookings b ON p.booking_id = b.booking_id
JOIN Listings l ON b.listing_id = l.listing_id
JOIN Hosts h ON l.host_id = h.host_id
GROUP BY h.host_id, h.host_name;
```

2. Preferred Payment Method of Guests

```sql
SELECT guest_id, payment_method, COUNT(*) as count
FROM Payments
GROUP BY guest_id, payment_method
ORDER BY count DESC;
```

3. Daily Revenue

```sql
SELECT t.date, SUM(p.amount) as daily_revenue
FROM Payments p
JOIN Time t ON p.payment_date_id = t.date_id
GROUP BY t.date;
```

Design Data Model for AirBnb

Grow Data Skills

SQL Queries to get business insights:

3. Revenue Trends (Monthly)

```
SELECT t.month, t.year, SUM(p.amount) as monthly_revenue
FROM Payments p
JOIN Time t ON p.payment_date_id = t.date_id
GROUP BY t.month, t.year
ORDER BY t.year, t.month;
```

4. Discrepancies in Amount Paid and Total Price

```
SELECT b.booking_id, b.total_price, SUM(p.amount) as total_paid
FROM Bookings b
JOIN Payments p ON b.booking_id = p.booking_id
GROUP BY b.booking_id, b.total_price
HAVING b.total_price <> total_paid;
```

5. Average Rating of Listings

```
SELECT l.listing_id, l.listing_name, AVG(r.rating) as average_rating
FROM Reviews r
JOIN Listings l ON r.listing_id = l.listing_id
GROUP BY l.listing_id, l.listing_name;
```

Design Data Model for AirBnb

SQL Queries to get business insights:

7. Number of Bookings per Listing

```
SELECT l.listing_id, l.listing_name, COUNT(b.booking_id) as number_of_bookings
FROM Bookings b
JOIN Listings l ON b.listing_id = l.listing_id
GROUP BY l.listing_id, l.listing_name;
```

8. Average Response Time of Hosts

```
SELECT h.host_id, h.host_name, AVG(m.response_time) as avg_response_time
FROM Messages m
JOIN Hosts h ON m.host_id = h.host_id
GROUP BY h.host_id, h.host_name;
```

9. Total Number of Nights Booked per Listing

```
SELECT l.listing_id, l.listing_name, SUM(b.number_of_nights) as total_nights
FROM Bookings b
JOIN Listings l ON b.listing_id = l.listing_id
GROUP BY l.listing_id, l.listing_name;
```

Design Data Model for AirBnb

SQL Queries to get business insights:

10. Most Popular Property Type in Each City

```
SELECT c.city_name, pt.property_type, COUNT(l.listing_id) as number_of_listings
FROM Listings l
JOIN PropertyTypes pt ON l.property_type_id = pt.property_type_id
JOIN Cities c ON l.city_id = c.city_id
GROUP BY c.city_name, pt.property_type
ORDER BY number_of_listings DESC;
```