Q.1) To design a data warehouse for IPL Cricket Tournament, we first need to identify the key entities involved in the tournament and the data points that need to be collected.

Note: While designing any Data Warehouse, cover the points below.

a. Design Fact & Dimension tables

b. Create meaningful Primary & Foreign keys

c. Try to follow Star/SnowFlake Schema Design

d. Try to write a few SQL queries to generate insightful business metrics (This is the critical

point because you need to understand the Data & Business both.

Entities:

Teams

Players

Matches

Venues

Umpires

Data Points:

Teams: team name, coach name, captain name, players

Players: player name, date of birth, country, batting style, bowling style

Matches: match number, date, venue, team1, team2, winner, toss winner, toss decision, player of the match, umpires

Venues: venue name, city, state, country

Umpires: umpire name, country, matches umpired

With these entities and data points in mind, we can create the following schema for our data warehouse:

Team Dimension Table:

team_id (Primary Key)

team_name

coach_name

captain_name

Player Dimension Table:

player_id (Primary Key)

player_name

date_of_birth

country

batting_style

bowling_style

Match Dimension Table:

match_id (Primary Key)

match_number

match_date

venue_id (Foreign Key)

team1_id (Foreign Key)

team2_id (Foreign Key)

winner_id (Foreign Key)

toss_winner_id (Foreign Key)

toss_decision

player_of_the_match_id (Foreign Key)

umpire1_id (Foreign Key)

umpire2_id (Foreign Key)

Venue Dimension Table:

venue_id (Primary Key)

venue_name

city

state

country

Umpire Dimension Table:

umpire_id (Primary Key)

umpire_name

country

Fact Table:

match_id (Foreign Key)

team_id (Foreign Key)

player_id (Foreign Key)

runs

wickets

overs

extras

total

a. Fact and Dimension Tables:

I have designed a Fact Table that contains the measures or numerical data such as runs, wickets, overs, extras, and total, and connected it with the Dimension Tables that contain the attributes such as team name, player name, venue name, umpire name, etc. The Dimension Tables provide descriptive information about the data in the Fact Table.

b. Meaningful Primary and Foreign Keys:

I have used meaningful Primary and Foreign Keys in the Dimension Tables to ensure data integrity and consistency. For example, in the ***Team Dimension Table***, the *team_id* is the primary key and it is connected to the *team1_id*, *team2_id*, *winner_id*, and *toss_winner_id* in the Match Dimension Table as foreign keys.

c. Star/Snowflake Schema Design:

I have followed the Star Schema Design where the Fact Table is in the center and is connected to the Dimension Tables. This design makes it easy to query the data and provides fast performance. However, the schema design can be further normalized into a Snowflake Schema if required.

d. SQL queries for insightful business metrics:

Some sample SQL queries for insightful business metrics are:

With this schema, we can answer a variety of questions about IPL Cricket Tournament, such as:

Which team won the most matches in a season?

Who scored the most runs in a particular match?

Which team has the most number of players from a particular country?

Which venue hosted the most matches in a season?

How many matches were umpired by a particular umpire?

By analyzing the data in the warehouse, we can gain insights into player and team performance, venue usage, and umpire performance, among other things.

Number of matches won by each team in a season:

```
SELECT T.team_name, COUNT(*) AS wins

FROM Match M

JOIN Team T ON M.winner_id = T.team_id

WHERE M.match_date BETWEEN '2022-01-01' AND '2022-12-31'

GROUP BY T.team_name

ORDER BY wins DESC;
```

Total runs scored by each player in a season:

```
SELECT P.player_name, SUM(F.runs) AS total_runs

FROM FactTable F

JOIN Player P ON F.player_id = P.player_id
```

```
WHERE F.match_date BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY P.player_name
ORDER BY total_runs DESC;
```

Average runs scored in a match by each team in a season:

```
SELECT T.team_name, AVG(F.runs) AS avg_runs
FROM FactTable F
JOIN Match M ON F.match_id = M.match_id
JOIN Team T ON F.team_id = T.team_id
WHERE M.match_date BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY T.team_name
ORDER BY avg_runs DESC;
```

Total number of matches played in each venue in a season:

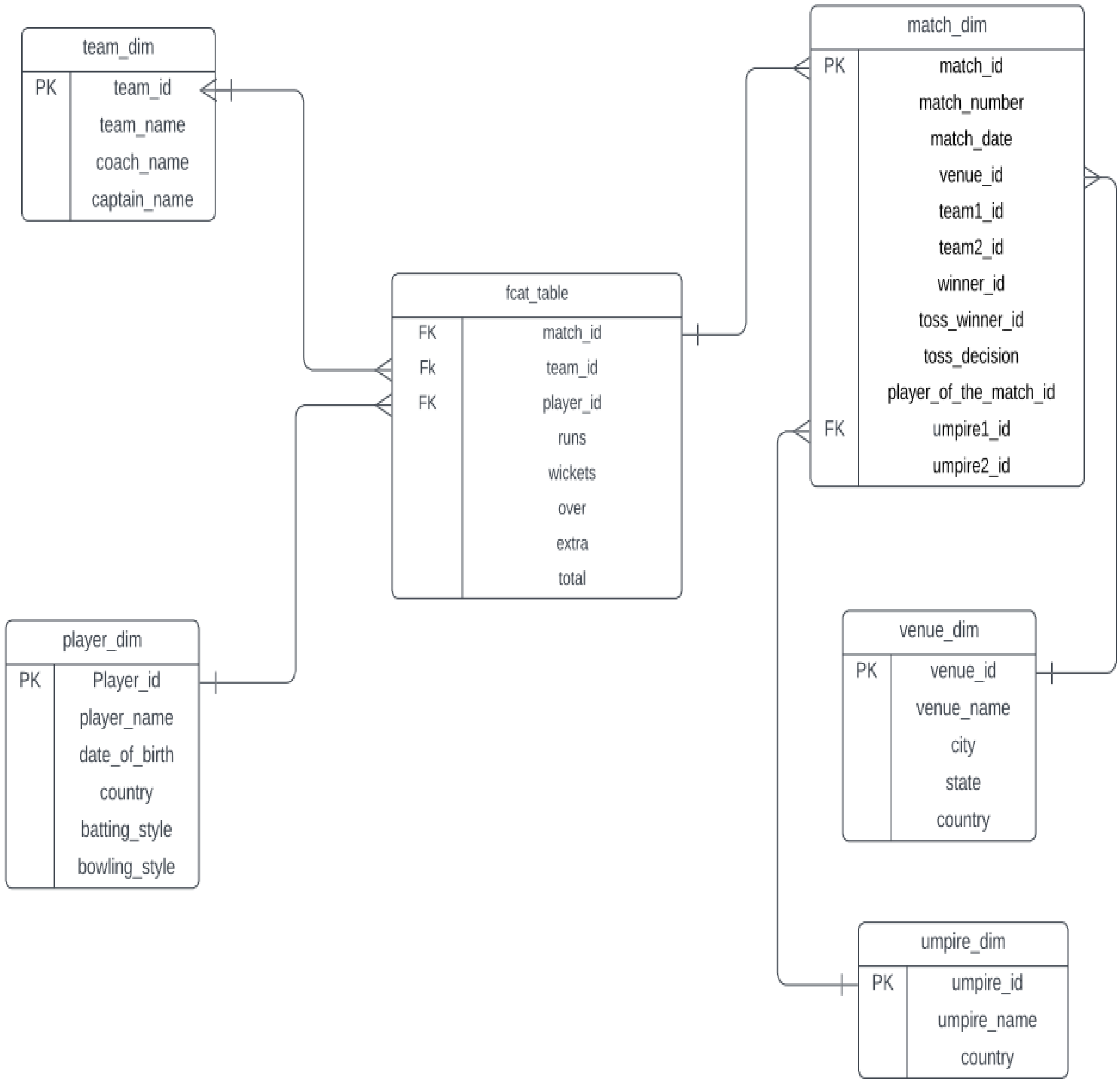
```
SELECT V.venue_name, COUNT(*) AS total_matches
FROM Match M
JOIN Venue V ON M.venue_id = V.venue_id
WHERE M.match_date BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY V.venue_name
ORDER BY total_matches DESC;
```

Number of matches umpired by each umpire in a season:

```
SELECT U.umpire_name, COUNT(*) AS matches_umpired
FROM Match M
JOIN Umpire U ON M.umpire1_id = U.umpire_id OR M.umpire2_id = U.umpire_id
WHERE M.match_date BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY U.umpire_name
ORDER BY matches_umpired DESC;
```

data model for IPL Cricket Tournament

Narayan Haridas Zeermire | May 14, 2023



team_dim	
PK	team_id
	team_name
	coach_name
	captain_name

fcat_table	
FK	match_id
Fk	team_id
FK	player_id
	runs
	wickets
	over
	extra
	total

match_dim	
PK	match_id
	match_number
	match_date
	venue_id
	team1_id
	team2_id
	winner_id
	toss_winner_id
	toss_decision
	player_of_the_match_id
FK	umpire1_id
	umpire2_id

player_dim	
PK	Player_id
	player_name
	date_of_birth
	country
	batting_style
	bowling_style

venue_dim	
PK	venue_id
	venue_name
	city
	state
	country

umpire_dim	
PK	umpire_id
	umpire_name
	country

CREDIT:-

DATA MODELLING TOOL – LUCID CHART

Q.2) Design a Data Warehouse for a Food delivery app like Swiggy, or Zomato?

Determining whether a star schema or snowflake schema is better for a food delivery app like Swiggy or Zomato depends on various factors and considerations. Here are a few points to help you make an informed decision:

Data Complexity and Relationships: Consider the complexity of your data and the relationships between the entities. If your data has a relatively simple structure and does not involve many hierarchies or complex relationships, a ***star schema may be sufficient***.

However, if your data has *intricate hierarchies and relationships* that require more normalization, a snowflake schema might be more appropriate.

Performance and Query Efficiency: Evaluate the performance requirements of your data warehouse. ***Star schema*** typically offers better query performance due to its ***denormalized structure***, as it involves fewer joins. If query speed is crucial and your data volume is not excessively large, a star schema may be a good choice. However, if you anticipate dealing with massive amounts of data and need more flexible query capabilities, the ***snowflake schema's*** normalized structure may be beneficial despite the additional joins.

Data Redundancy and Maintenance: Consider the trade-off between data redundancy and ease of maintenance. ***Star schema*** can result in some data redundancy due *to denormalization*, which simplifies queries but may require updates in multiple places.

Snowflake schema's normalization reduces redundancy, improving data integrity, but can be more complex to maintain and update.

Scalability and Future Growth: Anticipate the future growth and scalability requirements of your data warehouse. ***Snowflake schema's*** normalized structure offers *greater flexibility* and *scalability*. If you expect significant expansion in terms of new dimensions or relationships, the *snowflake schema* may provide a more scalable foundation for your data warehouse.

Reporting and Analytical Needs: Understand the specific reporting and analytical requirements of your food delivery app. Consider the types of queries and analyses you'll perform. If you require simple and straightforward reporting without many complex relationships, a star schema might suffice.

On the other hand, if you anticipate sophisticated analytics involving complex hierarchies and dimensions, the snowflake schema's normalized structure could be advantageous.

Ultimately, there is no one-size-fits-all answer. You should carefully evaluate the nature of your data, performance considerations, maintenance efforts, scalability requirements, and reporting needs to determine whether a star schema or snowflake schema is better suited for your food delivery app like Swiggy or Zomato.

To design a data warehouse for a food delivery app like *Swiggy or Zomato*, we'll follow the given points and create the fact and dimension tables, define primary and foreign keys, and consider a star schema design. Let's proceed with the design:

Fact Table: Order

order_id (primary key)

date_id (foreign key referencing the Date dimension)

customer_id (foreign key referencing the Customer dimension)

restaurant_id (foreign key referencing the Restaurant dimension)

delivery_partner_id (foreign key referencing the Delivery Partner dimension)

order_status

total_amount

payment_method

Dimension Table: Date

date_id (primary key)

date

day

month

year

Dimension Table: Customer

customer_id (primary key)

customer_name

email

phone_number

Dimension Table: Restaurant

restaurant_id (primary key)

restaurant_name

cuisine_type

city

state

Dimension Table: Delivery Partner

delivery_partner_id (primary key)

partner_name

vehicle_type

Now let's write a few SQL queries to generate insightful business metrics:

Total Orders per Day:

```
SELECT date, COUNT(order_id) AS total_orders
FROM Order
JOIN Date ON Order.date_id = Date.date_id
GROUP BY date
ORDER BY date;
```

Total Revenue per Restaurant:

```
SELECT restaurant_name, SUM(total_amount) AS total_revenue
FROM Order
JOIN Restaurant ON Order.restaurant_id = Restaurant.restaurant_id
GROUP BY restaurant_name
ORDER BY total_revenue DESC;
```

Top 5 Customers by Order Count:

```
SELECT customer_name, COUNT(order_id) AS order_count

FROM Order

JOIN Customer ON Order.customer_id = Customer.customer_id

GROUP BY customer_name

ORDER BY order_count DESC

LIMIT 5;
```

Average Order Value by Payment Method:

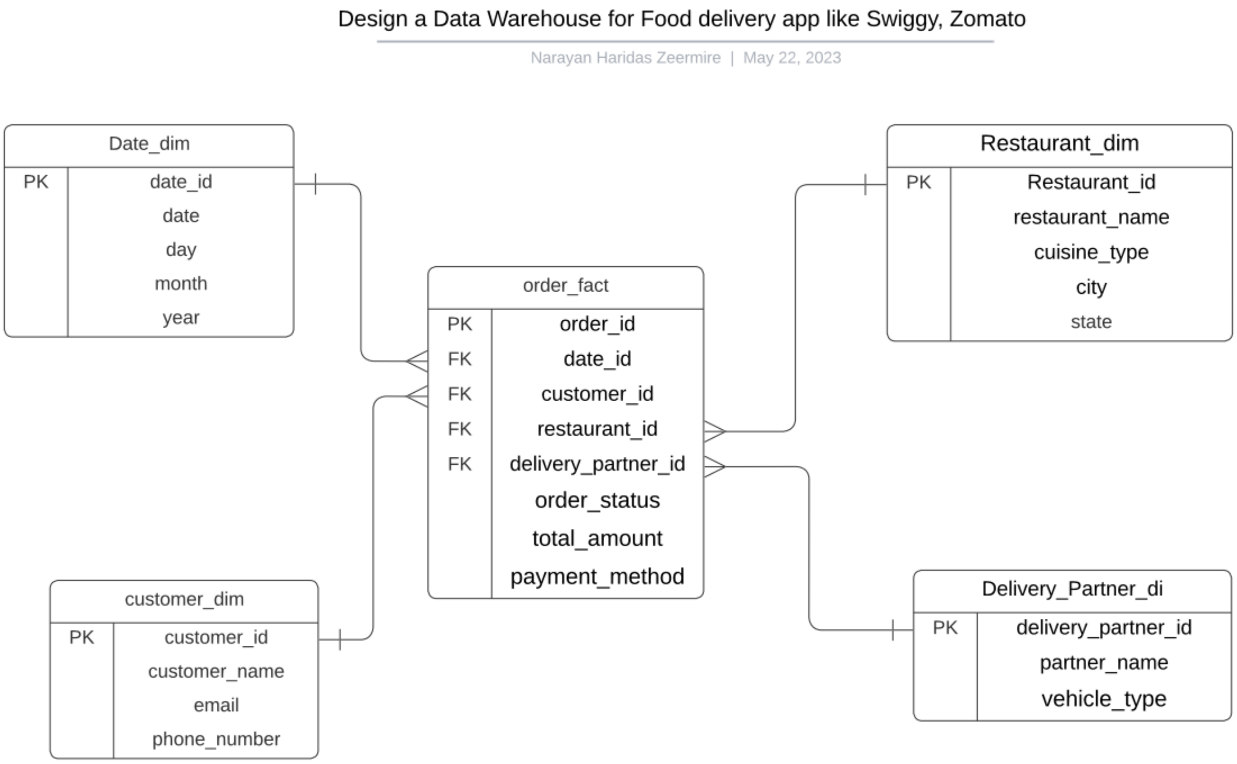
```
SELECT payment_method, AVG(total_amount) AS avg_order_value

FROM Order

GROUP BY payment_method;
```

Orders by Cuisine Type:

```
SELECT cuisine_type, COUNT(order_id) AS order_count

FROM Order

JOIN Restaurant ON Order.restaurant_id = Restaurant.restaurant_id

GROUP BY cuisine_type

ORDER BY order_count DESC;
```

These SQL queries provide insights into business metrics such as daily order count, total revenue per restaurant, top customers by order count, average order value by payment method, and orders by cuisine type. You can further customize and expand these queries based on the specific requirements and metrics you want to analyze in your food delivery app data warehouse.

Apologies for not considering the snowflake schema in the previous design. While both the star schema and snowflake schema can be used for designing a data warehouse, let's now incorporate the snowflake schema in the data warehouse design for the food delivery app:



Credit:- *with the help of lucid charts*

Q.3 Design a Data Warehouse for cab ride services like Uber, Lyft

To design a data warehouse for a cab rides service like Uber or Lyft, incorporating the given points, we can follow the star schema design. Here's an example of how the data warehouse could be structured:

Fact Table: Ride

ride_id (primary key)

date_id (foreign key referencing the Date dimension)

driver_id (foreign key referencing the Driver dimension)

rider_id (foreign key referencing the Rider dimension)

vehicle_id (foreign key referencing the Vehicle dimension) pickup_location_id (foreign key referencing the Location dimension)

dropoff_location_id (foreign key referencing the Location dimension)

ride_duration

ride_distance

fare_amount

Dimension Table: Date

date_id (primary key)

date

day_of_week

month

year

Dimension Table: Driver

driver_id (primary key)

driver_name

driver_rating

Dimension Table: Rider

rider_id (primary key)

rider_name

rider_rating

Dimension Table: Vehicle

vehicle_id (primary key)

vehicle_type

vehicle_model

vehicle_year

Dimension Table: Location

location_id (primary key)

location_name

latitude

longitude

Note: Additional dimension tables could be included based on specific requirements, such as a dimension table for payment methods, city, or ride status.

Now, let's write a few SQL queries to generate insightful business metrics:

Calculate the total number of rides per day:

```
SELECT d.date, COUNT(*) AS total_rides

FROM Ride r

JOIN Date d ON r.date_id = d.date_id

GROUP BY d.date;
```

Find the average ride duration by vehicle type:

```
SELECT v.vehicle_type, AVG(r.ride_duration) AS avg_ride_duration

FROM Ride r

JOIN Vehicle v ON r.vehicle_id = v.vehicle_id

GROUP BY v.vehicle_type;
```

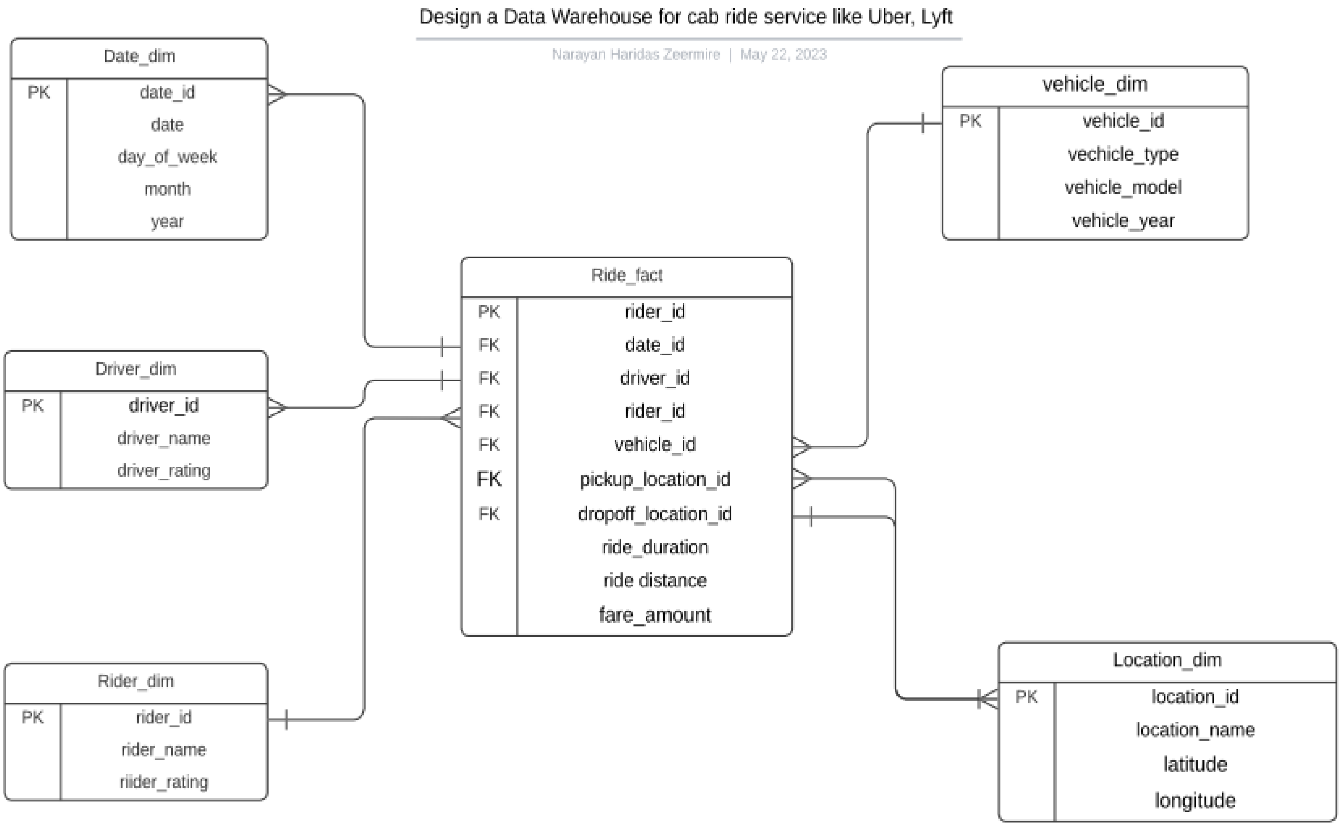
Determine the top 5 drivers with the highest average ratings:

```
SELECT d.driver_name, AVG(rider_rating) AS avg_rating

FROM Ride r

JOIN Driver d ON r.driver_id = d.driver_id

GROUP BY d.driver_name
```

ORDER BY avg_rating DESC

LIMIT 5;

These SQL queries provide examples of how you can generate business metrics by joining fact and dimension tables in the data warehouse. You can further expand on these queries based on the specific metrics and insights required for your analysis.



Design a Data Warehouse for cab ride service like Uber, Lyft

Narayan Haridas Zeermire | May 22, 2023

Q.4) Design a Data Warehouse for Restaurent table booking app like Dineout

Fact Table: Booking

booking_id (primary key)

date_id (foreign key referencing the Date dimension)

customer_id (foreign key referencing the Customer dimension)

restaurant_id (foreign key referencing the Restaurant dimension)

table_id (foreign key referencing the Table dimension)

booking_status

booking_start_time

booking_end_time

party_size

Dimension Table: Date

date_id (primary key)

date

day_of_week

month

year

Dimension Table: Customer

cusomer_id (primary key)

customer_name

customer_email

customer_phone

Dimension Table: Restaurant

restaurant_id (primary key)

restaurant_name

restaurant_location

Dimension Table: Table

table_id (primary key)

table_name

table_capacity

restaurant_id (foreign key referencing the Restaurant dimension)

Note: Additional dimension tables could be included based on specific requirements, such as a dimension table for cuisines, discounts, or booking status.

Now, let's write a few SQL queries to generate insightful business metrics:

Calculate the total number of bookings per day:

```
SELECT d.date, COUNT(*) AS total_bookings

FROM Booking b

JOIN Date d ON b.date_id = d.date_id

GROUP BY d.date;
```

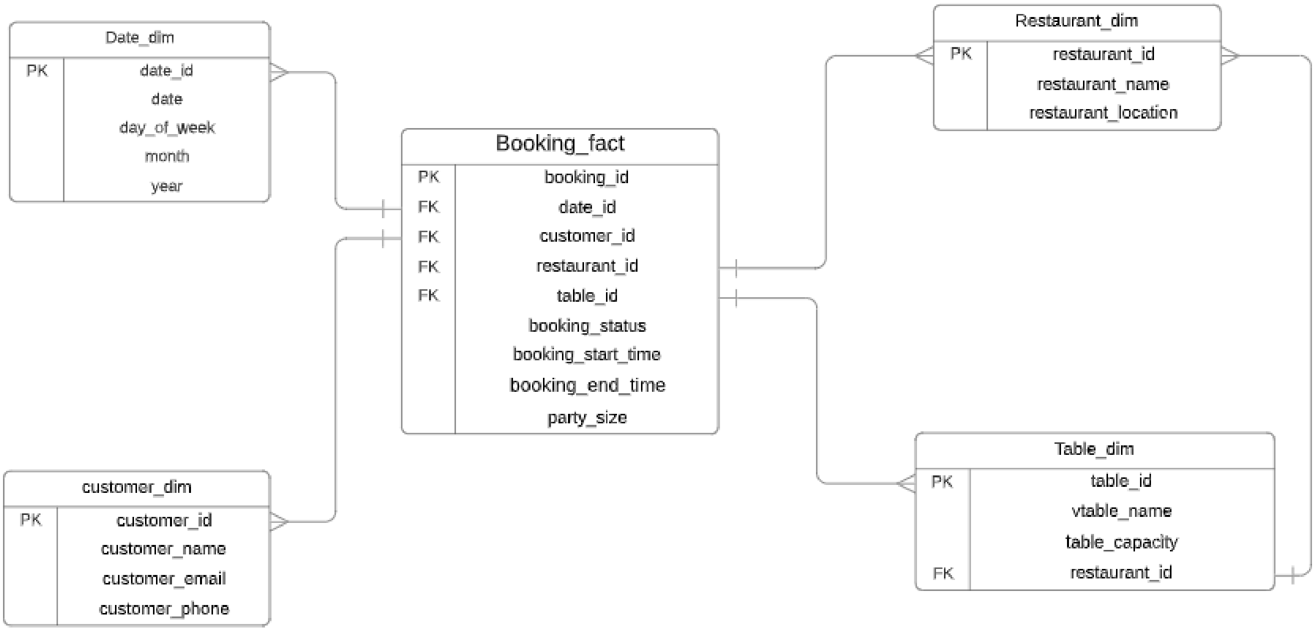
Find the average party size for each restaurant:

```
SELECT r.restaurant_name, AVG(b.party_size) AS avg_party_size

FROM Booking b

JOIN Restaurant r ON b.restaurant_id = r.restaurant_id

GROUP BY r.restaurant_name;
```

Determine the top 5 customers with the highest number of bookings:

```
SELECT c.customer_name, COUNT(*) AS total_bookings

FROM Booking b

JOIN Customer c ON b.customer_id = c.customer_id

GROUP BY c.customer_name

ORDER BY total_bookings DESC

LIMIT 5;
```

These SQL queries demonstrate how you can generate business metrics by joining fact and dimension tables in the data warehouse. You can further customize and expand on these queries based on the specific metrics and insights required for your analysis.



Design a Data Warehouse for Restaurent table booking app like Dineout

Narayan Haridas Zeermire | May 22, 2023

Q.5. Design a Data Warehouse for Covid Vaccination Application

To design a data warehouse for a COVID vaccination application, considering the given points, we can follow the star schema design. Here's an example of how the data warehouse could be structured:

Fact Table: Vaccination

vaccination_id (primary key)

date_id (foreign key referencing the Date dimension)

center_id (foreign key referencing the Vaccination Center dimension)

patient_id (foreign key referencing the Patient dimension)

vaccine_id (foreign key referencing the Vaccine dimension)

dose_number

vaccination_status

vaccination_date

Dimension Table: Date

date_id (primary key)

date

day_of_week

month

year

Dimension Table: Vaccination Center

center_id (primary key)

center_name

center_location

center_capacity

Dimension Table: Patient

patient_id (primary key)

patient_name

patient_age

patient_gender

Dimension Table: Vaccine

vaccine_id (primary key)

vaccine_name

manufacturer

dosage_interval

Note: Additional dimension tables could be included based on specific requirements, such as a dimension table for healthcare providers, vaccination status, or adverse events.

Now, let's write a few SQL queries to generate insightful business metrics:

Calculate the total number of vaccinations administered per day:

```
SELECT d.date, COUNT(*) AS total_vaccinations

FROM Vaccination v

JOIN Date d ON v.date_id = d.date_id

GROUP BY d.date;
```

Find the top 5 vaccination centers with the highest vaccination rates:

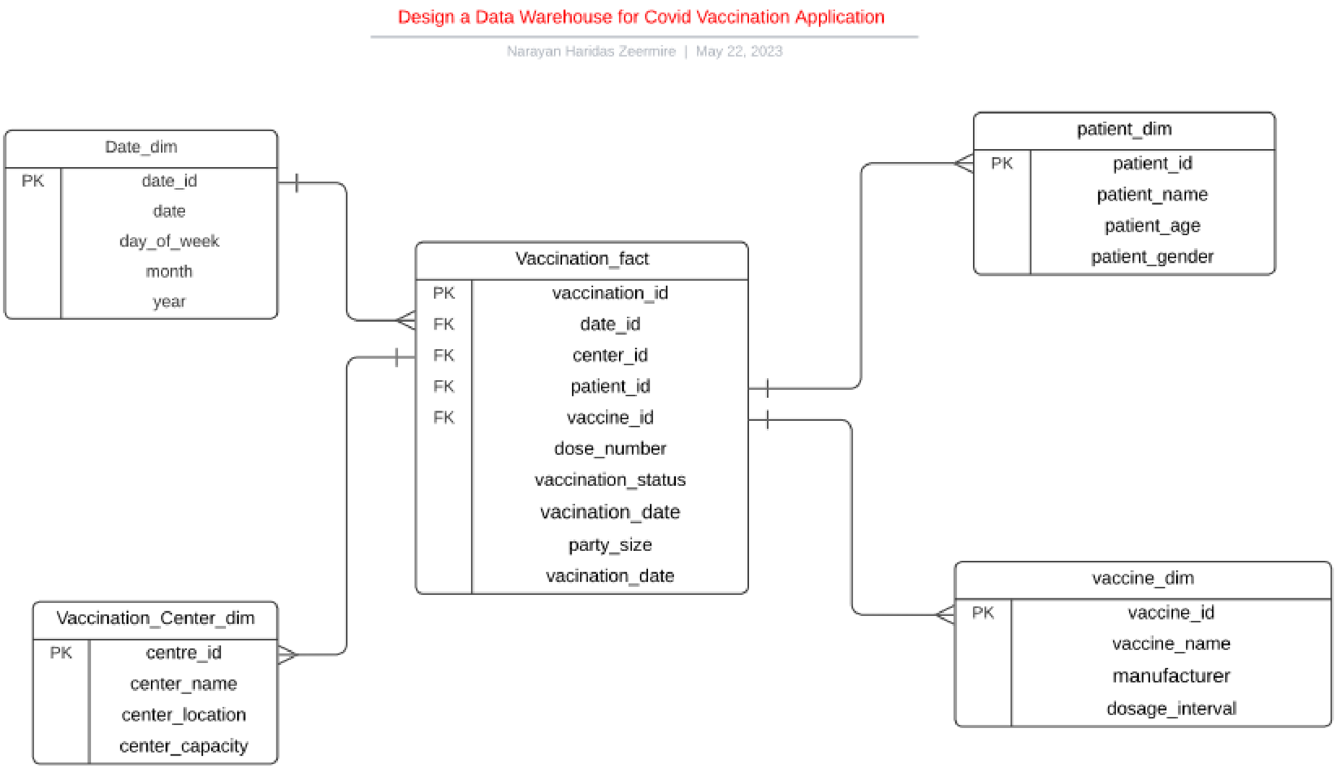
```
SELECT c.center_name, COUNT(*) AS total_vaccinations

FROM Vaccination v

JOIN VaccinationCenter c ON v.center_id = c.center_id

GROUP BY c.center_name

ORDER BY total_vaccinations DESC
```

```
LIMIT 5;
```

Determine the percentage of fully vaccinated patients:

```
SELECT COUNT(*) AS fully_vaccinated_count, (COUNT(*) / (SELECT COUNT(*) FROM Patient)) * 100
AS percentage

FROM Vaccination v

JOIN Patient p ON v.patient_id = p.patient_id

WHERE v.vaccination_status = 'Fully Vaccinated';
```

These SQL queries provide examples of how you can generate business metrics by joining fact and dimension tables in the data warehouse. You can further customize and expand on these queries based on the specific metrics and insights required for your analysis.


Design a Data Warehouse for Covid Vaccination Application
Narayan Haridas Zeermire | May 22, 2023
Date_dim
PK
date_id
date
day_of_week
month
year
patient_dim
PK
patient_id
patient_name
patient_age
patient_gender
Vaccination_fact
PK
vaccination_id
FK
date_id
FK
center_id
FK
patient_id
FK
vaccine_id
dose_number
vaccination_status
vacination_date
party_size
vacination_date
vaccine_dim
PK
vaccine_id
vaccine_name
manufacturer
dosage_interval
Vaccination_Center_dim
PK
centre_id
center_name
center_location
center_capacity