Assignment submitted by:
Aditya Singh Rajput
Bigdata Batch-1

1. Design a Data Warehouse for IPL Cricket Tournament

Designing a data warehouse for an IPL (Indian Premier League) cricket tournament would involve several steps:

1. Define the business requirements: The data warehouse should be able to store and analyze data related to the IPL tournament, such as match schedules, player statistics, team performance, and ticket sales. The data warehouse should also be able to answer business questions such as which teams and players are performing the best, how ticket sales are trending, and which matches are the most popular.

2. Design the data warehouse schema: This would involve creating a logical and physical data model for the data warehouse, including tables, columns, relationships, and indexes. The schema should include tables for storing information about teams, players, matches, venues, and ticket sales. The schema should also include relationships between tables, such as a relationship between a team and its players or between a match and its venue.

3. Extract, transform, and load (ETL) the data: This would involve extracting data from various sources such as cricket statistics websites, ticket sales databases, and social media platforms. The data would then be transformed to conform to the data warehouse schema and loaded into the data warehouse.

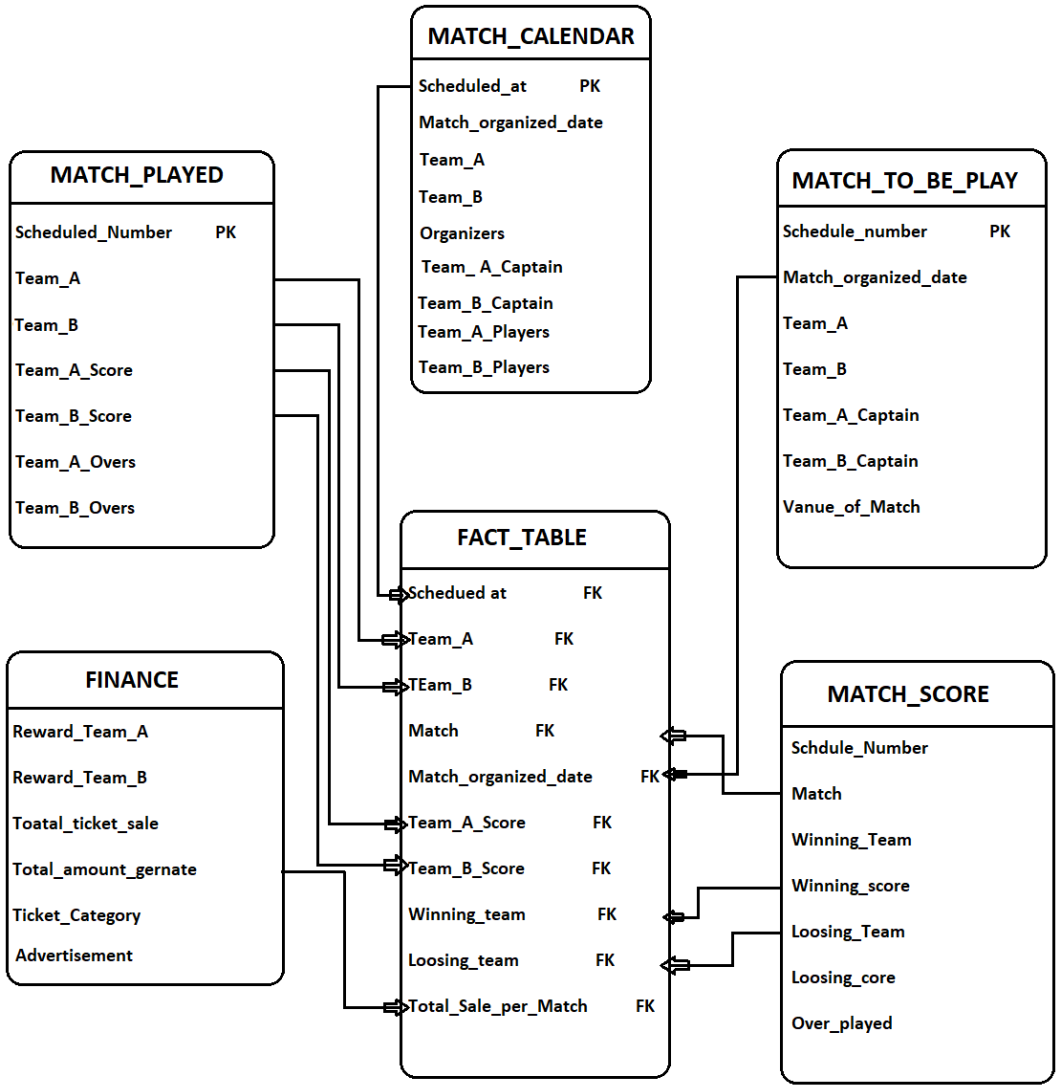
4. Optimize the data warehouse for performance: This would involve indexing the data, partitioning large tables, and implementing measures to improve query performance. This will allow the data warehouse to handle large amounts of data and support complex queries.

5. Implement security and access controls: This would include setting up authentication and authorization for users and applications that need to access the data, and implementing measures to protect the data from unauthorized access or breaches.

6. Test and deploy the data warehouse: This would include testing the data warehouse to ensure that it meets the business requirements and that the data is accurate, and then deploying the data warehouse into a production environment.

7. Monitor and maintain the data warehouse: This would include monitoring the performance of the data warehouse, managing the data and schema, and troubleshooting and resolving any issues that arise.

Star Schema Design for data warehouse schema



```
CREATE TABLE Match_calendar (
  Schduled_Number INT,
  Organized_date DATE,
  Team_A STRING,
  Team_B STRING,
  Team_A_captain STRING,
  Team_B_captain STRING,
  Team_A_Players STRING,
  Team_B_Player STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE match_to_be_play (
  Schdule_number INT,
  Match_organized_date DATE,
  Team_A STRING,
  Team_B STRING,
  Team_B_captain STRING,
  Team_B_captain STRING,
  Match_vanue STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE match_played (
  Schdule_number INT,
  Team_A STRING,
  Team_B STRING,
  Team_A_score STRING,
  Team_B_score STRING,
  Team_A_overs INT,
  Team_B_overs INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';


CREATE TABLE Finance (

  Schdule_number INT,
  Reward_Team_A INT,
```

```
  Reward_Team_B INT,
  Total_ticket_sale INT,
  Ticket_category STRING,
  Total_amount INT,
  Advertisement STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE match_score (
  Schdule_number INT,
  Match STRING,
  Team_B STRING,
  Winning_team  STRING,
  Wineam_score INT,
  Loosing_team STRING,
  Looseteam_score INT,
  Total_over_played INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

FACT TABLE

```
CREATE TABLE Fact_match (
  Schdule_number INT,
  Team_A STRING,
  Organized_date DATE,
  Match STRING,
  Team_B STRING,
  Team_A_score INT,
  Winning_Team STRING,
  Loosing_Team STRING,
  Total_sale_per_Match INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

2. Design a Data Warehouse for Food delivery app like Swiggy, Zomato

Designing a data warehouse for a food delivery app like Swiggy or Zomato would involve the following steps:

1. Identify the data sources: The first step would be to identify the data sources that will be used to populate the data warehouse. These may include data from the app's database, data from external sources such as weather data, and data from third-party providers such as payment processors.

2. Define the data model: The next step would be to define the data model for the data warehouse. This would involve creating the schema for the data warehouse, including the tables and relationships between them. The data model should be designed to support the reporting and analysis needs of the business.

3. Extract, Transform, and Load (ETL): Once the data model is defined, the next step would be to extract the data from the various sources, transform it to fit the data model, and load it into the data warehouse. This process would involve using ETL tools to extract the data, clean and transform it, and load it into the data warehouse.

4. Data Quality and Governance: After loading the data into the data warehouse, it is important to implement data quality and governance processes to ensure that the data is accurate, complete, and consistent. This would involve implementing data validation and cleansing rules, as well as data auditing and monitoring processes.

5. Implement Security Measures: Implementing security measures is also important to protect the data and the system from unauthorized access and breaches. This includes setting up roles and permissions, data encryption, and monitoring for suspicious activities.

6. Reporting and Analysis: Finally, the data warehouse should be set up to support the reporting and analysis needs of the business. This would involve creating data marts and implementing reporting and analysis tools that allow users to access and analyze the data in the data warehouse.

7. Performance Optimization: Performance optimization is also important, this would include implementing a star schema, using indexes, partitioning and data compression.

8. Continual Maintenance: Continual maintenance is also important to ensure that the data warehouse is running smoothly, that data is accurate and updated, and that the system is secure. This would involve monitoring and troubleshooting

Designing a data warehouse for a food delivery app like Swiggy or Zomato using HiveQL would involve the following steps:

- Create tables to store data: The first step would be to create the tables in Hive to store the data from the various sources. These tables would be based on the data model that you have defined. For example, you could create tables for storing customer data, order data, menu data, and delivery data.

```
CREATE TABLE customers (
  id INT,
  name STRING,
  email STRING,
  phone STRING,
  address STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE orders (
  id INT,
  customer_id INT,
  order_date TIMESTAMP,
  delivery_date TIMESTAMP,
  status STRING,
  total_amount FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE menu (
  id INT,
  name STRING,
  category STRING,
  price FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

CREATE TABLE delivery (
```

```
  id INT,
  order_id INT,
  delivery_person_id INT,
  delivery_date TIMESTAMP,
  status STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Fact table

```
CREATE TABLE order_fact (
  order_id INT,
  customer_id INT,
  restaurant_id INT,
  order_date TIMESTAMP,
  delivery_date TIMESTAMP,
  total_amount FLOAT,
  order_status STRING
)
PARTITIONED BY (year INT, month INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

ALTER TABLE order_fact
ADD FOREIGN KEY (order_id) REFERENCES delivery (order_id)
ALTER TABLE order_fact
ADD FOREIGN KEY (customer_id) REFERENCES orders (customer_id)
ALTER TABLE order_fact
ADD FOREIGN KEY (customer_id) REFERENCES orders (customer_id)
ALTER TABLE order_fact
ADD FOREIGN KEY (total_amount) REFERENCES orders (total_amount)
```

- Load data into the tables: Once the tables are created, the next step would be to load the data into the tables using the LOAD DATA statement. This statement would be used to load data from external sources into the tables.

```
LOAD DATA INPATH '/path/to/customers.csv' INTO TABLE customers;
LOAD DATA INPATH '/path/to/orders.csv' INTO TABLE orders;
LOAD DATA INPATH '/path/to/menu.csv' INTO TABLE menu;
LOAD DATA INPATH '/path/to/delivery.csv' INTO TABLE delivery;
```

- Create views and indexes: To improve the performance of the data warehouse, it's important to create views and indexes on the tables. This can be done by using the CREATE VIEW and CREATE INDEX statements.

```
CREATE VIEW customer_orders AS
SELECT o.id, o.order_date, o.status, c.name, c.email, c.phone
FROM orders o JOIN customers c ON
```

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

Designing and coding a Data Warehouse for a cab ride service like Uber or Lyft using HiveQL would involve several key steps, including creating the necessary tables, loading the data into the tables, and querying the data.

- Data Collection: Collect data from various sources such as ride data from the cab service's API, driver and passenger data from registration and login systems, and payment data from the payment gateway.

- Data Storage: Store the collected data in a distributed file system like HDFS or S3. The data can be stored in various file formats like Parquet, Avro, ORC, etc.

- Data Processing: Use HiveQL to process the data and create the Data Warehouse.

  Create the necessary tables: The first step would be to create the tables that will store the data in the Data Warehouse. These tables would include a table for storing information about the rides (e.g. ride ID, driver ID, rider ID, pickup and drop-off location, etc.), a table for storing information about the drivers (e.g. driver ID, name, location, rating, etc.), and a table for storing information about the riders (e.g. rider ID, name, location, etc.).

```
CREATE TABLE rides (
ride_id INT,
driver_id INT,
rider_id INT,
pickup_location STRING,
dropoff_location STRING,
ride_time TIMESTAMP,
fare FLOAT
)
```

```
CREATE TABLE drivers (
driver_id INT,
name STRING,
location STRING,
rating FLOAT
)

CREATE TABLE riders (
rider_id INT,
name STRING,
location STRING
 )
```

Fact Table

```
CREATE EXTERNAL TABLE ride_data (
ride_id INT,
driver_id INT,
passenger_id INT,
start_time TIMESTAMP,
end_time TIMESTAMP,
pickup_location STRING,
dropoff_location STRING,
ride_distance DOUBLE,
ride_duration INT,
ride_fare DOUBLE
)
STORED AS PARQUET
LOCATION 'hdfs://namenode:port/path/to/ride_data'
```

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

Designing and coding a Data Warehouse for a restaurant table booking app like Dineout using HiveQL would involve several key steps, including data collection, data storage, and data processing. Here is an example of how this can be done:

- Data Collection: Collect data from various sources such as booking data from the app's API, customer data from registration and login systems, menu and pricing information from the restaurants, and payment data from the payment gateway.

- Data Storage: Store the collected data in a distributed file system like HDFS or S3. The data can be stored in various file formats like Parquet, Avro, ORC, etc.
- Data Processing: Use HiveQL to process the data and create the Data Warehouse.

```
CREATE EXTERNAL TABLE booking_data (
booking_id INT,
customer_id INT,
restaurant_id INT,
booking_time TIMESTAMP,
booking_date DATE,
party_size INT,
booking_status STRING,
total_cost DOUBLE
)
STORED AS PARQUET
LOCATION 'hdfs://namenode:port/path/to/booking_data'

CREATE EXTERNAL TABLE customer_data (
customer_id INT,
name STRING,
email STRING,
phone INT,
address STRING
)
STORED AS PARQUET
LOCATION 'hdfs://namenode:port/path/to/customer_data'

CREATE TABLE booking_summary(
restaurant_id INT,
booking_date DATE,
bookings_count INT
)
AS
SELECT restaurant_id, booking_date, COUNT(booking_id)
FROM booking_data
GROUP BY restaurant_id, booking_date
```

FACT TABLE

```
CREATE TABLE Fact_booking
booking_id INT,
customer_id INT,
restaurant_id INT,
booking_time TIMESTAMP,
booking_date DATE,
party_size INT,
booking_status STRING,
total_cost DOUBLE
```

```
)
PARTITIONED BY (booking_year INT, booking_month INT)
STORED AS PARQUET
```

5. Design a Data Warehouse for Covid Vaccination Application

Designing and coding a Data Warehouse for a Covid Vaccination application using HiveQL would involve several key steps, including data collection, data storage, and data processing. Here is an example of how this can be done:

- Data Collection: Collect data from various sources such as vaccination data from the application's API, patient data from registration and login systems, vaccine information, and vaccination center data.

- Data Storage: Store the collected data in a distributed file system like HDFS or S3. The data can be stored in various file formats like Parquet, Avro, ORC, etc.

- Data Processing: Use HiveQL to process the data and create the Data Warehouse.

```
CREATE EXTERNAL TABLE vaccination_data (
patient_id INT,
vaccine_name STRING,
dose INT,
vaccination_date DATE,
vaccination_center_id INT,
vaccinator_name STRING
)
STORED AS PARQUET
LOCATION 'hdfs://namenode:port/path/to/vaccination_data'

CREATE EXTERNAL TABLE patient_data (
patient_id INT,
name STRING,
age INT,
gender STRING,
address STRING,
phone INT
)
STORED AS PARQUET
LOCATION 'hdfs://namenode:port/path/to/patient_data'

CREATE EXTERNAL TABLE vaccine_data (
vaccine_name STRING,
```

```
manufacturer STRING,
dose INT,
dose_interval INT
)
STORED AS PARQUET
LOCATION 'hdfs://namenode:port/path/to/vaccine_data'

CREATE EXTERNAL TABLE vaccination_center_data (
vaccination_center_id INT,
name STRING,
address STRING,
phone INT
)
STORED AS PARQUET
LOCATION 'hdfs://namenode:port/path/to/vaccination_center_data'
```

- Join the data from different sources to create the Data Warehouse:

```
CREATE TABLE vaccination_warehouse AS
SELECT p.patient_id, p.name, vd.vaccine_name, vd.dose, vd.vaccination_date,
vc.name as vaccination_center, vc.address
FROM patient_data p
JOIN vaccination_data vd ON p.patient_id = vd.patient_id
JOIN vaccination_center_data vc ON vd.vaccination_center_id =
vc.vaccination_center_id
```