Question 0. We don't expect you to have any cricket knowledge and that is not a requirement to ace this assessment. But we understand that familiarity with cricket may vary from one candidate to the next so we would like to know how you would rate your knowledge of cricket from 1 to 5, where 1 is basically no knowledge (like you had never seen or read anything about the sport until the days before this assessment) and 5 is highly knowledgeable (you watch matches regularly and have a jersey for the Rajasthan Royals in your closet, for example).

ANS : 3 (Average)

Question 3. Please provide a brief written answer to the following question. The coding assessment focused on a batch backfilling use case. If the use case was extended to required incrementally loading new match data on a go-forward basis, how would your solution change?

ANS :

If the use case was extended to require incrementally loading new match data on a go-forward basis, the solution would need to be adapted to handle incremental data updates efficiently. Here are the key changes that would be made to the solution:

1. Data Tracking:

Implement a mechanism to keep track of the last processed match or the timestamp of the last update. This ensures that only new matches (those occurring after the last processed match) are fetched and processed.

2. Incremental Data Extraction:

Modify the data ingestion process to fetch and process only the new match data. This can be achieved by querying the data source for records created or modified after the last update timestamp.

3. Database Update:

Instead of recreating the entire database, implement logic to insert or update records in the existing database. This is usually done using SQL statements like `INSERT INTO` for new records and `UPDATE` for existing records based on unique identifiers.

4. Error Handling and Retry Mechanism:

Implement error handling and retry mechanisms to handle network issues or temporary unavailability of the data source. If a data retrieval or insertion fails, the system should be able to retry the operation after a certain interval.

5. Data Integrity and Consistency:

Ensure data integrity and consistency during incremental updates. This might involve handling data conflicts, ensuring referential integrity, and maintaining consistency across related tables in the database.

6. Scheduled Jobs or Triggers:

Set up scheduled jobs or triggers to automate the incremental data loading process. These jobs can run at specified intervals (e.g., daily, hourly) to fetch and process new data without manual intervention.

7. Monitoring and Logging:

Implement logging and monitoring mechanisms to track the status of data updates. Logs can be helpful for diagnosing issues, and monitoring tools can provide insights into the performance and health of the data loading process.

8. Versioning and Schema Evolution:

Plan for versioning and schema evolution of the database. As new data fields or structures are introduced in the source data, the database schema should be able to evolve without disrupting the existing functionality.

By incorporating these changes, the solution can efficiently handle incremental data loading, ensuring that only new match data is processed and integrated into the existing database while maintaining data integrity and reliability.

Question 4. Can you provide an example of when, during a project or analysis, you learned about (or created) a new technique, method, or tool that you hadn't known about previously? What inspired you to learn about this and how were you able to apply it?

ANS :

During the cricket project, I encountered a performance bottleneck while processing a large volume of ball-by-ball data. The challenge was to optimize the data processing pipeline to handle the massive dataset more efficiently, ensuring faster processing times and reduced resource consumption.
To address this challenge, I researched and implemented parallel processing techniques. While I had a basic understanding of parallel processing, this project pushed me to explore Apache Spark's parallelism capabilities in depth. I learned about partitioning strategies, data shuffling optimizations, and how to leverage Spark's cluster computing power effectively.
Inspired by the need for efficiency, I fine-tuned the Spark jobs, optimizing the number of partitions and strategically caching intermediate data. Additionally, I explored broadcast variables to efficiently distribute smaller reference datasets across the cluster, reducing unnecessary data shuffling.
The application of these parallel processing techniques significantly improved the processing speed and resource utilization of our data pipeline. It not only solved the immediate problem but also enhanced my understanding of distributed computing concepts in real-world scenarios.
This experience motivated me to further explore Apache Spark's advanced features, leading me to experiment with dynamic allocation and cluster management strategies. Although these techniques were not entirely new, their practical application in optimizing large-scale data processing was an enriching learning experience.