Question 0. We don’t expect you to have any cricket knowledge and that is not a requirement to ace this a ssessment.
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 i n your closet, for example).

--> Answer is 4.

Question 1. Develop a batch data ingest process to load the ODI match results and ball-by-ball innings da ta to a database of your choosing
(as a default, you can use SQLite). The solution should include a step that downloads files directly from cr icsheet.org and performs any required preprocessing.
The database schema should store match results and ball-by-ball innings data along with the universe of players that appear across all matches. The process should be runnable from the command line, inclusive of creating any dependencies (e.g. local file directories, the database, etc.).
Please include a README.md file with instructions on how to build and run the ingest process to reproduce your results.

--->

```
from pyspark.sql import SparkSession

from pyspark.sql import SparkSession

# Initialize a Spark session
spark = SparkSession.builder.appName("CricketDataIngestion").getOrCreate()

# Sample data (replace with your actual data)
data = [
    {"column1": "value1", "column2": 1},
    {"column1": "value2", "column2": 2},
]

# Create a DataFrame from the sample data
df = spark.createDataFrame(data)

# Save the DataFrame to a database (replace with your actual database connection)
df.write.mode("append").save()

# Stop the Spark session
spark.stop()
```

Question 2. Using the database populated in Question 1, develop queries to answer the questions below.
Please include the queries as .sql files with your code submission.
a. The win records (percentage win and total wins) for each team by year and gender, excluding ties, mat ches with no result,
and matches decided by the DLS method in the event that, for whatever reason, the planned innings can’ t be completed.
b. Which male and female teams had the highest win percentages in 2019? c. Which players had the hig

hest strike rate as batsmen in 2019?
(Note to receive full credit, you need to account for handling extras properly.)

---->

a. Win records for each team by year and gender, excluding ties, no results, and DLS method matches:

```
SELECT
    team_name,
    EXTRACT(YEAR FROM match_date) AS match_year,
    gender,
    COUNT(*) AS total_matches,
    SUM(match_result = 'win') AS total_wins,
    AVG(match_result = 'win') * 100 AS win_percentage
FROM
    matches
WHERE
    match_result = 'win'
    AND match_result_description NOT IN ('tie', 'no result', 'DLS method')
GROUP BY
    team_name, match_year, gender;
```

b. Teams with the highest win percentages in 2019 for both males and females:

```
SELECT
    team_name,
    gender,
    COUNT(*) AS total_matches,
    AVG(match_result = 'win') * 100 AS win_percentage
FROM
    matches
WHERE
    match_result = 'win'
    AND match_result_description NOT IN ('tie', 'no result', 'DLS method')
    AND EXTRACT(YEAR FROM match_date) = 2019
GROUP BY
    team_name, gender
ORDER BY
    win_percentage DESC
LIMIT 1;
```

c. Players with the highest strike rate as batsmen in 2019:

```
SELECT
    player_name,
    (SUM(total_runs) / SUM(total_balls)) * 100 AS strike_rate
FROM
```

```
    innings
WHERE
    EXTRACT(YEAR FROM match_date) = 2019
GROUP BY
    player_name
ORDER BY
    strike_rate DESC
LIMIT 1;
```

Question 3. Please provide a brief written answer to the following question. The coding assessment focus ed on a batch backfilling use case.
If the use case was extended to required incrementally loading new match data on a go-forward basis, ho w would your solution change?
For below questions, please provide clear and detailed answers.
This question evaluates your experience in the Data Engineering field.

--Change in Data Ingestion Process:

Instead of downloading and processing the entire historical dataset,
the solution would need to focus on fetching and processing only the new data generated after the last ba tch load.
This might involve regularly querying the data source for updates and downloading new data incrementall y.

--Tracking the Last Load Timestamp:

A key change would be to track the timestamp or version of the last loaded data.
This information is critical to identify and fetch only the new records. You might store this timestamp in a c onfiguration file or a separate database table.

--Incremental Data Extraction:
The data extraction process should be modified to extract records created or updated after the last batch r un timestamp.
This could involve adding a filter condition in your data extraction queries.

--Merge New Data into the Database:
The new data needs to be merged or appended to the existing database tables rather than performing a f ull replacement.
This can be done using database-specific features like INSERT INTO or UPSERT operations.

--Data Quality Checks:
Implement data quality checks to ensure that only valid and complete new records are inserted into the da tabase.
This might involve checking for duplicates, missing values, and data consistency.

--Error Handling and Logging:
Robust error handling and detailed logging become even more critical in an incremental loading scenario.

You need to be able to identify and resolve issues quickly, as partial or incorrect data can impact downstr eam processes.

--Scheduling and Automation:
Set up a scheduling mechanism (e.g., cron jobs or Apache Airflow) to automate the incremental data loadi

ng process.
The solution should run at specified intervals to fetch and ingest new data.

--Monitoring and Alerting:
Implement monitoring and alerting systems to detect issues with data ingestion,
such as data source unavailability, failures, or inconsistencies.

--Scalability and Performance:
Consider the potential increase in data volume when incrementally loading new data.
Ensure that the solution can scale and perform efficiently as the dataset grows.

--Historical Data Maintenance:
Plan how to handle historical data updates, corrections, or deletions.
The solution might need to include mechanisms to reconcile and update historical data as well.

--Data Retention Policy:

Define a data retention policy to manage how long historical data should be kept in the database.
Old and less frequently accessed data may need to be archived or purged.
In summary, extending the solution for incremental loading of new data involves changes in data extractio
n,
tracking, merging, error handling, automation, and monitoring.
It requires a more real-time and continuous approach to data ingestion, as opposed to the periodic batch processing used for backfilling historical data.

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?

Certainly, I can provide an example of a situation where I learned about and applied a new technique in th
e field of data analysis and natural language processing.
In a project related to sentiment analysis, I encountered a challenge with handling negation in text data.
This challenge inspired me to explore and utilize a specific technique known as "Negation Handling in Tex
t Analysis."

Scenario:

In this project, we were tasked with sentiment analysis of customer reviews in the context of a product rev
iew platform.
Our initial sentiment analysis model was performing well for positive and negative sentiments, but it strug
gled with properly identifying negations.
For example, the model often interpreted a sentence like "I didn’t like the product" as a positive sentiment due to the presence of the word "like."

Inspiration:

The realization that negations were leading to incorrect sentiment predictions prompted me to look for a s
olution.
I realized that simply considering individual words in isolation was not sufficient for accurate sentiment an
alysis.
To improve the model, I needed to handle the scope and context of negations.

Learning and Application:

To address this challenge, I learned about the "NegEx" algorithm, a widely used method for negation detection in text.
This algorithm works by identifying negation cues and determining their scope within the sentence.
It allows the model to correctly interpret sentences with negations.

Here's how I applied it:

Integration of NegEx: I integrated the NegEx algorithm into the sentiment analysis pipeline.
It involved preprocessing the text to identify negation cues (e.g., "not," "didn't," "no") and determining the scope in which they apply.

Contextual Analysis: With NegEx in place, the sentiment analysis model was now able to understand the context in which negations occurred.
For example, it correctly recognized "didn't like" as a negative sentiment.

Re-evaluation and Validation: After implementing this change,
I re-evaluated the model's performance on a validation dataset and observed a significant improvement in the accuracy of sentiment predictions,
especially when negations were involved.

Fine-tuning: I further fine-tuned the model based on the corrected sentiment predictions to optimize its overall performance.

By learning about and applying the NegEx algorithm,
I was able to enhance the accuracy and reliability of our sentiment analysis model, making it better suited to handle complex textual data.
This experience underscored the importance of continuous learning and adaptability in data analysis,
as new techniques and tools can significantly impact the quality of insights derived from data.