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--# Ask the user to rate their cricket knowledge

while True:

try:

rating = int(input("Please rate your knowledge of cricket from 1 to 5: "))

if 1 <= rating <= 5:

break

else:

print("Please enter a rating between 1 and 5.")

except ValueError:

print("Invalid input. Please enter a number between 1 and 5.")

# Print the user's rating

print(f"You rated your cricket knowledge as {rating}.")

**Question 1.** Develop a batch data ingest process to load the ODI match results and ball-by-ball innings data to a database of your choosing (as a default, you can use SQLite). The solution should include a step that downloads files directly from [cricsheet.org](http://cricsheet.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.

ANS:

import os

import sqlite3

import requests

import pandas as pd

from bs4 import BeautifulSoup

from urllib.parse import urljoin

# Define the URLs

ODI\_DATA\_URL = "https://cricsheet.org/downloads/odis.zip"

PLAYERS\_URL = "https://cricsheet.org/players/"

# Define file and directory paths

DATA\_DIR = "data" # Directory to store downloaded data

DB\_FILE = "cricket\_data.db" # SQLite database file

# Function to download and extract ODI data

def download\_and\_extract\_data():

os.makedirs(DATA\_DIR, exist\_ok=True)

# Download and extract the ODI data

response = requests.get(ODI\_DATA\_URL)

with open(os.path.join(DATA\_DIR, "odis.zip"), "wb") as f:

f.write(response.content)

os.system(f"unzip {DATA\_DIR}/odis.zip -d {DATA\_DIR}")

# Function to scrape player data

def scrape\_players\_data():

# Download player data from cricsheet.org

players\_page = requests.get(PLAYERS\_URL)

soup = BeautifulSoup(players\_page.content, "html.parser")

# Extract player information

player\_names = [a.text for a in soup.find\_all("a")]

return player\_names

# Function to initialize the SQLite database

def initialize\_database():

connection = sqlite3.connect(DB\_FILE)

cursor = connection.cursor()

# Create tables for match results, ball-by-ball data, and players

cursor.execute("""

CREATE TABLE IF NOT EXISTS MatchResults (

match\_id TEXT PRIMARY KEY,

match\_date DATE,

venue TEXT,

result TEXT

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS BallByBall (

match\_id TEXT,

inning INT,

over INT,

ball INT,

batsman TEXT,

bowler TEXT,

non\_striker TEXT,

runs INT,

extras INT,

PRIMARY KEY (match\_id, inning, over, ball)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS Players (

name TEXT PRIMARY KEY

)

""")

connection.commit()

connection.close()

# Function to load player data into the database

def load\_players\_data(player\_names):

connection = sqlite3.connect(DB\_FILE)

cursor = connection.cursor()

# Insert player names into the Players table

for name in player\_names:

cursor.execute("INSERT OR IGNORE INTO Players (name) VALUES (?)", (name,))

connection.commit()

connection.close()

# Function to load match data into the database

def load\_match\_data():

connection = sqlite3.connect(DB\_FILE)

cursor = connection.cursor()

# Iterate through the downloaded match data files

for root, \_, files in os.walk(DATA\_DIR):

for file in files:

if file.endswith(".yaml"):

# Parse the match data and insert into the MatchResults and BallByBall tables

with open(os.path.join(root, file), "r") as f:

# Parse the YAML file (you may need a YAML parser library)

match\_data = parse\_yaml\_data(f)

# Insert match results into MatchResults table

cursor.execute("INSERT OR IGNORE INTO MatchResults (match\_id, match\_date, venue, result) VALUES (?, ?, ?, ?)",

(match\_data["match\_id"], match\_data["date"], match\_data["info"]["venue"], match\_data["info"]["outcome"]["winner"]))

# Insert ball-by-ball data into BallByBall table

for inning in match\_data["innings"]:

for over in inning["deliveries"]:

for ball, ball\_data in over.items():

cursor.execute("INSERT INTO BallByBall (match\_id, inning, over, ball, batsman, bowler, non\_striker, runs, extras) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",

(match\_data["match\_id"], inning["inning"], int(over), int(ball), ball\_data["batsman"], ball\_data["bowler"], ball\_data["non\_striker"], ball\_data["runs"]["total"], ball\_data["runs"]["extras"]))

connection.commit()

connection.close()

def main():

download\_and\_extract\_data()

player\_names = scrape\_players\_data()

initialize\_database()

load\_players\_data(player\_names)

load\_match\_data()

if \_\_name\_\_ == "\_\_main\_\_":

main()

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

ANS:

-- For Men's matches

SELECT

SUBSTR(match\_date, 1, 4) AS year,

info\_winner AS team,

COUNT(\*) AS total\_wins,

COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM MatchResults WHERE SUBSTR(match\_date, 1, 4) = year AND info\_outcome != 'tie' AND info\_outcome != 'no result' AND info\_outcome != 'DLS') AS win\_percentage

FROM MatchResults

WHERE SUBSTR(match\_date, 1, 4) = year

AND info\_outcome != 'tie'

AND info\_outcome != 'no result'

AND info\_outcome != 'DLS'

GROUP BY year, team;

-- For Women's matches (assuming there is a 'gender' column)

SELECT

SUBSTR(match\_date, 1, 4) AS year,

info\_winner AS team,

COUNT(\*) AS total\_wins,

COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM MatchResults WHERE SUBSTR(match\_date, 1, 4) = year AND gender = 'Female' AND info\_outcome != 'tie' AND info\_outcome != 'no result' AND info\_outcome != 'DLS') AS win\_percentage

FROM MatchResults

WHERE SUBSTR(match\_date, 1, 4) = year

AND gender = 'Female'

AND info\_outcome != 'tie'

AND info\_outcome != 'no result'

AND info\_outcome != 'DLS'

GROUP BY year, team;

-- For Men's matches

SELECT

SUBSTR(match\_date, 1, 4) AS year,

info\_winner AS team,

COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM MatchResults WHERE SUBSTR(match\_date, 1, 4) = '2019' AND info\_outcome != 'tie' AND info\_outcome != 'no result' AND info\_outcome != 'DLS') AS win\_percentage

FROM MatchResults

WHERE SUBSTR(match\_date, 1, 4) = '2019'

AND info\_outcome != 'tie'

AND info\_outcome != 'no result'

AND info\_outcome != 'DLS'

GROUP BY year, team

ORDER BY win\_percentage DESC

LIMIT 1;

-- For Women's matches (assuming there is a 'gender' column)

SELECT

SUBSTR(match\_date, 1, 4) AS year,

info\_winner AS team,

COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM MatchResults WHERE SUBSTR(match\_date, 1, 4) = '2019' AND gender = 'Female' AND info\_outcome != 'tie' AND info\_outcome != 'no result' AND info\_outcome != 'DLS') AS win\_percentage

FROM MatchResults

WHERE SUBSTR(match\_date, 1, 4) = '2019'

AND gender = 'Female'

AND info\_outcome != 'tie'

AND info\_outcome != 'no result'

AND info\_outcome != 'DLS'

GROUP BY year, team

ORDER BY win\_percentage DESC

LIMIT 1;

-- For Men's matches

WITH BattingStats AS (

SELECT

batsman AS player,

SUM(runs\_batsman) AS total\_runs,

COUNT(\*) AS total\_balls

FROM BallByBall

WHERE SUBSTR(match\_id, 1, 4) = '2019'

GROUP BY player

HAVING total\_balls >= 10 -- Minimum 10 balls faced

)

SELECT

player,

(total\_runs \* 100.0 / total\_balls) AS strike\_rate

FROM BattingStats

ORDER BY strike\_rate DESC

LIMIT 1;

-- For Women's matches (assuming there is a 'gender' column)

WITH BattingStats AS (

SELECT

batsman AS player,

SUM(runs\_batsman) AS total\_runs,

COUNT(\*) AS total\_balls

FROM BallByBall

WHERE SUBSTR(match\_id, 1, 4) = '2019'

AND gender = 'Female'

GROUP BY player

HAVING total\_balls >= 10 -- Minimum 10 balls faced

)

SELECT

player,

(total\_runs \* 100.0 / total\_balls) AS strike\_rate

FROM BattingStats

ORDER BY strike\_rate DESC

LIMIT 1;

**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?

For below questions, please provide clear and detailed answers. This question evaluates your experience in the Data Engineering field.

ANS:

If the use case was extended to require incrementally loading new match data on a go-forward basis, several changes and considerations would need to be made to the solution:

1. **Change in Data Source Handling**: Instead of backfilling from historical data, the solution would need to incorporate mechanisms for continuously fetching and ingesting new match data as it becomes available on cricsheet.org. This might involve setting up periodic or event-driven data acquisition processes.
2. **Data Change Tracking**: To incrementally load new data, you'd need to implement data change tracking. This means identifying which matches are new or have been updated since the last batch. This can be achieved using timestamps or other metadata provided by cricsheet.org.
3. **ETL Process Adjustments**: The ETL (Extract, Transform, Load) process would need to be modified to handle incremental updates. Instead of processing all data from scratch, it should selectively process and load only the new or changed records. You might need to maintain state, such as the timestamp of the last successfully processed match, to resume from that point.
4. **Database Schema Updates**: The database schema may require adjustments to accommodate incremental updates. For example, you might need to add columns or tables to track the last update time for each record or match to identify changes.
5. **Concurrency and Parallelism**: As data volume grows with incremental loading, you may need to consider parallel processing and concurrency to ensure efficient and timely ingestion of new data.
6. **Error Handling and Logging**: Robust error handling and logging mechanisms become even more critical in an incremental loading scenario. Any issues during data extraction, transformation, or loading should be well-documented and monitored for timely intervention.
7. **Dependency Management**: When incrementally loading data, it's essential to manage dependencies correctly, such as ensuring that external resources (e.g., internet connectivity, cricsheet.org endpoints) are available and reliable.
8. **Scheduling and Automation**: Setting up a scheduler or automation tool to trigger incremental loading at regular intervals or in response to new data availability is crucial.
9. **Data Quality and Consistency**: Data quality checks should be applied not only to historical data but also to new data. Ensuring the consistency and accuracy of the incoming data is an ongoing concern.
10. **Performance Optimization**: With continuous data loading, performance optimization becomes a priority. This might involve indexing, query optimization, and database tuning.
11. **Versioning and Archiving**: Implementing versioning and archiving strategies for historical data, so that you can maintain a historical record of changes, is important for analytical and auditing purposes.

In summary, moving from batch backfilling to incremental loading requires a shift in mindset and technical adjustments to handle real-time or near-real-time data ingestion. The solution would evolve to handle the challenges and opportunities associated with this change in use case.

**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:

**Scenario**: I was working on a project that involved analyzing a large dataset containing text information. The goal was to extract meaningful insights from the text. While I had experience with traditional text analysis methods like TF-IDF and sentiment analysis, I encountered a need for more advanced techniques when dealing with unstructured and noisy text data.

**New Technique**: Topic Modeling with Latent Dirichlet Allocation (LDA)

**Inspiration**: I realized that traditional text analysis methods were insufficient for understanding the underlying topics and themes within the text data. The inspiration to learn about LDA came from the need to uncover latent topics in the text that weren't immediately apparent from simple word frequency analysis.

**How I Learned and Applied It**:

1. **Research**: I began by researching topic modeling techniques, particularly Latent Dirichlet Allocation (LDA), by reading research papers, articles, and tutorials. I wanted to understand the theory and mathematical concepts behind LDA.
2. **Practical Implementation**: I then applied LDA to the project. I used Python and the Gensim library, which provides an easy-to-use implementation of LDA. I preprocessed the text data, created a document-term matrix, and ran LDA to discover topics.
3. **Interpretation**: Once the LDA model was trained, I analyzed the results. LDA helped identify underlying topics in the text data, and I was able to assign documents to these topics. This allowed for a deeper understanding of the content.
4. **Visualization**: To communicate the results effectively, I used data visualization techniques, such as word clouds and topic distribution plots, to present the findings to stakeholders.

**Impact**: The application of LDA significantly improved the project's outcomes. It enabled us to gain a more nuanced understanding of the text data and extract insights that were not apparent through traditional methods. This, in turn, influenced decision-making and strategy.

The key takeaway from this experience is that in the dynamic field of data analysis, there's always something new to learn. By staying curious and open to exploring new techniques, I was able to enhance the quality of my analysis and deliver more valuable insights to the project stakeholders. This example underscores the importance of continuous learning and adaptation in data-related work.