MT：

1. a) Neo4j is ACID - compliant in the following ways: Atomicity, Consistency, Isolation, Durability

b) In some scenarios, MongoDB can be used as the primary data store to handle most of the read and write operations. Neo4j is used as a secondary database to handle the query and analysis of relational data. To achieve this synergy, the following strategies can be employed: Data Synchronization: Synchronize data from MongoDB to Neo4j on a regular or real-time basis. In this way, Neo4j is able to maintain up-to-date relational data for efficient querying and analysis. Use adapters or middleware: Third-party adapters or middleware can be used to synchronize data and transform queries between MongoDB and Neo4j.

c)

Difference between Native and Non - native Storage in Graph Databases

Native Storage:

Data Structure and Layout: In graph databases with native storage, the storage system is purpose - built for graph - structured data. The data layout is designed to optimize the representation and access of nodes and relationships. For example, it might use a pointer - based structure where each node has direct pointers to its adjacent nodes and relationships. This allows for quick traversal of the graph as the database can directly access the related elements without extensive search operations.

Data Access and Manipulation: Native storage systems have operations and APIs that are tailored to graph - specific tasks. For instance, adding a new relationship between two nodes can be a simple and efficient operation that updates the internal graph structure directly. The storage system understands the graph concepts such as nodes, relationships, and their properties and provides native support for operations like traversing relationships, finding paths, and aggregating data based on the graph structure.

Non - native Storage:

Data Structure and Layout: Graph databases with non - native storage rely on an underlying storage mechanism that is not originally designed for graphs. For example, some graph databases might use a relational database as their storage backend. In such a case, the graph data has to be mapped onto the relational model. Nodes and relationships might be stored as tables, and the graph - specific operations need to be translated into a series of SQL operations. This can lead to a more complex and less intuitive data layout for graph - oriented applications.

Data Access and Manipulation: When using non - native storage, graph - specific operations often involve more overhead. For example, traversing relationships might require multiple JOIN operations in a relational - based storage system. The operations are not as straightforward as in native storage because the underlying storage system does not have native support for graph - specific concepts. As a result, there can be a performance degradation and a more complex programming model when working with non - native storage for graph databases.

Implication of Native Storage on Processing Performance

1.Efficient Graph Traversal

2.Reduced Overhead in Query Processing

3.Scalability for Graph - based Workloads

2. a) Purpose for the First Cypher Script:
The purpose of this Neo4j Cypher script is to create several nodes representing users and a relationship between two of those users.
First, it creates five nodes of the label User with specific username properties. The nodes are for users named Alice, Bob, Charlie, Davina, and Edward.
Then, it creates a relationship of type ALIAS_OF from the node representing Alice to the node representing Bob. This could imply that in the context of the graph's domain, Alice is an alias of Bob.
b) Purpose for the Second Cypher Script:
This Cypher script first locates specific nodes in the graph (the nodes representing Bob, Charlie, Davina, and Edward) and then creates several relationships between the node representing Bob and the other three nodes.
The MATCH clause is used to find the nodes with the given username values for Bob, Charlie, Davina, and Edward.
The CREATE clause then creates three relationships from the node representing Bob to the nodes representing Charlie, Davina, and Edward respectively. The relationship types are EMAILED, CC (which might imply carbon copy in an email context), and BCC (which might imply blind carbon copy in an email context). This could be used to represent communication patterns between these users in an email-like scenario.

3. The Effect of the Subsequent Cypher Script:
MATCH Clause: The MATCH clause is used to find specific patterns in the graph. In this case: It looks for a pattern where there is a node a which is a Person with the name Jim. From this node a (Jim), it follows the [: KNOWS] relationship to another node b. Then from node b, it follows another [: KNOWS] relationship to a third node c. Additionally, it also looks for a direct [: KNOWS] relationship from the node a (Jim) to the node c.
In the context of the previously created graph, when a is Jim, node b could potentially be Jan (since Jim knows Jan and Jan could be the intermediate node to reach another node c). And node c could be Emil (as there are paths from Jim through ian to Emil and also a direct relationship from Jim to Emil).
RETURN Clause: The RETURN clause determines what will be returned as the result of the query. In this script, it returns the nodes b and c. So, based on the possible matches described above, it would return the nodes that match the pattern of being two nodes that are connected to Jim through a chain of KNOWS relationships and also having a direct KNOWS relationship from Jim to one of them.
In summary, the script will return pairs of nodes (node b and node c) that satisfy the specific KNOWS relationship patterns starting from the node Jim in the given graph.

4.CREATE (neo:Company {name: 'Neo'}),
(Ian:Person {name: 'Ian'}),
(job:Job {start: '2011-01-05'}),
(role:Role {name: 'engineer'}),

(neo)-[:EMPLOYER]->(job),
(lan)-[:EMPLOYMENT]->(job),
(job)-[:ROLE]->(role);

AT:

5. // Create a node
CREATE (p1:Person {name: 'Alice'}),
(d1:Department {name: '4FUTURE'}),
(d2:Department {name: 'P0815'});

// Create a relationship
MERGE (p1)-[:MEMBER_OF]->(d1);
MERGE (p1)-[:MEMBER_OF]->(d2);

6. a) Script Purpose：
The purpose of this script is to find the person nodes named “Sally” and “John” in the Neo4j graph database, and find the edge with the relationship “FRIEND_OF” between them. Then it returns the “since” attribute value of this edge, which is the time they became friends, with “friends since” as the alias of the returned result.
b)
i) Query the average rating of Moby Dick:
MATCH (book:Book {title: 'Moby Dick'})
RETURN AVG(book.rating) AS averageRating;
ii) Query the author of Moby Dick:
MATCH (book:Book {title: 'Moby Dick'})
RETURN book.author AS author;
iii) Query Sally's age:
MATCH (sally:Person {name: 'Sally'})
RETURN sally.age AS sallyAge;
iv) Query who read Moby Dick first, Sally or John:
MATCH (sally:Person {name: 'Sally'}), (john:Person {name: 'John'})
MATCH (sally)-[:HAS_READ]->(book:Book {title: 'Moby Dick'})
MATCH (john)-[:HAS_READ]->(book)
RETURN CASE WHEN sally.readDate < john.readDate THEN 'Sally' ELSE 'John' END AS firstReader;