Exploring Non-Relational Databases: MongoDB, DynamoDB, Cassandra, and Neo4j

Part of Module 3

1 Cassandra

Apache Cassandra is an open-source, distributed, column-oriented NoSQL database designed for managing large amounts of data across many commodity servers with no single point of failure.

1.1 Key Features:

- Peer-to-peer architecture — All nodes are equal; no master/slave.
- Highly scalable — Can handle petabytes of data by adding more nodes.
- High availability — Data is replicated; cluster works even if nodes fail.
- Tunable consistency — Clients can choose between speed and consistency (e.g., ONE, QUORUM, ALL).
- Efficient for write-heavy workloads — Optimized write path (commit log + memtable → SSTable).

1.2 How it Works:

- Data Partitioning: Uses consistent hashing to decide which node stores which data based on the partition key.
- Replication: Each row is copied to multiple nodes based on the replication factor.
- Coordinator Node: Any node can accept a client request, acts as coordinator to route requests to the right replicas.
- Gossip Protocol: Nodes constantly exchange state information to stay updated about the cluster.
- Write Path: Data is written to a commit log, then to an in-memory table (memtable), and eventually flushed to disk as immutable SSTables.
- Read Path: Checks memtable and SSTables, merges results, returns the latest data.

1.3 Basic CQL (Cassandra Query Language) Examples

Listing 1: Create Keyspace

```
CREATE KEYSPACE student_data
WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': 2
};
```

Listing 2: Use Keyspace

```
USE student_data;
```

Listing 3: Create Table

```
CREATE TABLE students (
  roll_no INT PRIMARY KEY,
  name TEXT,
  age INT,
  email TEXT
);
```

Listing 4: Insert Data

```
INSERT INTO students (roll_no, name, age, email)
VALUES (101, 'Alice', 20, 'alice@example.com');

INSERT INTO students (roll_no, name, age, email)
VALUES (102, 'Bob', 21, 'bob@example.com');
```

Listing 5: Query Data

```
SELECT * FROM students;

SELECT name, email FROM students WHERE roll_no = 101;
```

1.4 Use case:

- Real-time analytics
- IoT data storage
- Financial transactions
- Recommendation systems
- Messaging platforms

2 Dynamo Db

Amazon DynamoDB is a fully managed, serverless, NoSQL key-value and document database service provided by Amazon Web Services (AWS). It delivers single-digit millisecond performance at any scale, without requiring users to manage servers, patch software, or configure replication.

2.1 Key Features:

- Fully Managed: AWS handles provisioning, scaling, patching, backups, and replication.
- Key–Value and Document Model: Stores data as items (rows) in tables; each item is a collection of attributes (columns).
- High Availability and Scalability: Automatically scales up or down throughput and storage; data is replicated across multiple AZs (availability zones) within a region.
- Performance: Consistent low-latency reads/writes; supports on-demand or provisioned capacity modes.
- Security and Integrations: Integrated with IAM (access control), encryption at rest, streams for event-driven architectures, and triggers via AWS Lambda.

2.2 How it works:

- Tables: Top-level containers for data.
- Items: Each table stores items, analogous to rows.
- Attributes: Each item consists of attributes (name–value pairs).
- Primary Key: Uniquely identifies each item. Either: 1. Partition key (single attribute). 2. Partition + Sort key (composite key).
- Partitioning: Data is distributed across multiple storage nodes based on a hash of the partition key.
- Replication: Synchronous replication across multiple availability zones for fault tolerance.
- Consistency: Supports eventually consistent (default) and strongly consistent reads.
- Access: Typically via AWS SDKs, CLI, or REST APIs.

2.3 Basic DynamoDB CLI Examples:

Listing 6: Create Table

```
aws dynamodb create-table \
  --table-name Users \
  --attribute-definitions AttributeName=UserID,AttributeType=S \
  --key-schema AttributeName=UserID,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

Listing 7: Insert item

```
aws dynamodb put-item \
  --table-name Users \
  --item '{"UserID": {"S": "101"}, "Name": {"S": "Alice"}, "Age": {"N": "20"}}'
```

Listing 8: Query an item

```
aws dynamodb get-item \
  --table-name Users \
  --key '{"UserID": {"S": "101"}}'
```

2.4 Use Case:

- Gaming leaderboards.
- Real-time bidding or shopping carts.
- Serverless backends with unpredictable workloads.
- IoT data ingestion.
- Mobile/web application state storage.

3 Mongo DB

MongoDB is an open-source, document-oriented, NoSQL database designed to store and retrieve data in a flexible, JSON-like format. It's widely used for modern web and mobile applications that need high performance, horizontal scalability, and flexible schemas.

3.1 Key Features:

- Document-oriented: Data is stored as BSON (binary JSON) documents, which can have nested structures and arrays.
- Schema-less: No fixed schema; each document in a collection can have different fields.
- Horizontal Scalability: Built-in sharding distributes data across multiple servers.
- High Availability: Replica sets automatically handle failover and provide redundancy.
- Rich Query Language: Supports filtering, aggregation, indexing, and geospatial queries.

3.2 How it works:

- Database: A logical container for collections.
- Collection: A group of related documents (like a table in RDBMS).
- Document: A JSON-like structure with key–value pairs (like a row).
- Replica Sets: Provide redundancy and automatic failover by maintaining multiple copies of the data.
- Sharding: Splits data across shards (clusters of replica sets) to handle massive datasets.
- Indexes: Improve query performance; MongoDB supports single field, compound, text, and geospatial indexes.

3.3 Basic MongoDB Shell Examples:

Listing 9: Create a Database (implicitly)

```
use studentDB
```

Listing 10: Create a Collection (implicitly with first insert)

```
db.students.insertOne({ "_id": 101, "name": "Alice", "age": 20, "email": "alice@example.com" })
```

Listing 11: Insert Multiple Documents

```
db.students.insertMany([
  { "_id": 102, "name": "Bob", "age": 21 },
  { "_id": 103, "name": "Carol", "age": 22 }
])
```

Listing 12: Query Documents

```
db.students.find({ "age": { "$gt": 20 } })
```

Listing 13: Update Document

```
db.students.updateOne(
  { "_id": 101 },
  { "$set": { "email": "alice@newmail.com" } }
)
```

Listing 14: Delete Document

```
db.students.deleteOne({ "_id": 103 })
```

3.4 Use Case:

- Content management systems.
- Catalogs and product data.
- Real-time analytics and personalization.
- Mobile and IoT backends.
- Social networking platforms.

4 Neo4j

Neo4j is a high-performance, open-source, NoSQL graph database. It stores data as nodes and relationships instead of rows or documents. It is optimized for connected data and is widely used for applications that rely on relationships, such as social networks, recommendation engines, and fraud detection.

4.1 Key Features:

- Graph-based data model: Represents data as nodes (entities), relationships (connections), and properties (attributes).
- Cypher query language: A declarative query language designed specifically for working with graphs.
- ACID-compliant: Provides transactional integrity.
- High performance for relationships: Efficient for traversing deep or complex connections.
- Flexible schema: You can add new node types, relationships, or properties without schema migrations.

4.2 How it works:

- Nodes: Represent entities (e.g., Person, Product).
- Relationships: Directed edges that connect nodes (e.g.FRIENDOF, PURCHASED).
- Properties: Key–value pairs that describe nodes and relationships.
- Labels: Classify nodes into types (e.g. :Person, :Movie).
- Indexes and constraints: Speed up lookups and enforce uniqueness.
- ACID Transactions: Ensure data consistency during concurrent operations.

4.3 Basic Cypher Examples:

Listing 15: Create Node

```
CREATE (a:Person {name: 'Alice', age: 30});
CREATE (b:Person {name: 'Bob', age: 28});
```

Listing 16: Create Relationship

```
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'})
CREATE (a)-[:FRIEND_OF]->(b);
```

Listing 17: Query Node

```
MATCH (p:Person) WHERE p.age > 25 RETURN p.name, p.age;
```

Listing 18: Traverse Relationship

```
MATCH (a:Person)-[:FRIEND_OF]->(b:Person)
RETURN a.name AS Person, collect(b.name) AS Friends;
```

Listing 19: Update Property

```
MATCH (p:Person {name: 'Alice'})
SET p.age = 31;
```

Listing 20: Delete a Node

```
MATCH (p:Person {name: 'Bob'}) DETACH DELETE p;
```

4.4 Use Case:

- Social network analysis.
- Recommendation engines.
- Fraud detection.
- Network and IT operations.
- Knowledge graphs and master data management.