Cycle 08 AWS Homework

NoSQL Research & Hands-On

---

1. Four Main Types of NoSQL Databases

A. Key-Value Stores

- **History**: Simplest type, inspired by hash tables; popularized by Amazon's Dynamo (2007).
- **Storage & Retrieval**: Data stored as a **key (unique identifier)** and **value (blob)**. Retrieval by key is O(1).
- **Formats Supported**: JSON, String, Binary, BSON.
- **Examples**: Redis, DynamoDB, Riak.
- **CRUD Example (Redis)**:

```
# Create
SET user:1 "Chetan"
# Read
GET user:1
# Update
SET user:1 "Varma"
# Delete
DEL user:1
```

B. Document Stores

- **History**: Emerged to handle JSON-like data structures; MongoDB first released in 2009.
- **Storage & Retrieval**: Data stored as **documents** (key-value pairs, nested structures).

- **Formats Supported**: JSON, BSON, XML.
- **Examples**: MongoDB, CouchDB, Couchbase.
- **CRUD Example (MongoDB)**:

```
// Create
db.users.insertOne({id: 1, name: "Chetan"})
// Read
db.users.find({id: 1})
// Update
db.users.updateOne({id: 1}, {$set: {name: "Varma"}})
// Delete
db.users.deleteOne({id: 1})
```

C. Column-Family Stores

- **History**: Inspired by Google's Bigtable (2006).
- **Storage & Retrieval**: Data stored in **rows and columns**, but columns grouped into **column families** for fast retrieval.
- **Formats Supported**: Flexible (JSON, plain text, binary).
- **Examples**: Apache Cassandra, HBase, ScyllaDB.
- **CRUD Example (Cassandra CQL)**:

```
-- Create
INSERT INTO users (id, name) VALUES (1, 'Chetan');
-- Read
SELECT * FROM users WHERE id=1;
-- Update
UPDATE users SET name='Varma' WHERE id=1;
-- Delete
DELETE FROM users WHERE id=1;
```

D. Graph Databases

- **History**: Designed for relationship-heavy data; Neo4j launched in 2007.
- **Storage & Retrieval**: Data stored as **nodes** (entities) and **edges** (relationships).
- **Formats Supported**: Property Graph Model (JSON-like), RDF.
- **Examples**: Neo4j, Amazon Neptune.
- **CRUD Example (Neo4j Cypher)**:

```
// Create
CREATE (u:User {id:1, name:'Chetan'})
// Read
MATCH (u:User {id:1}) RETURN u
// Update
MATCH (u:User {id:1}) SET u.name = 'Varma'
// Delete
MATCH (u:User {id:1}) DELETE u
```

---

2. CAP Theorem Analysis

- **Redis (CP – Consistency & Partition Tolerance)**
  - Prioritizes **consistency**: clients always see the latest write.
  - Good for: financial transactions, session stores, caching with strict correctness.
- **Cassandra (AP – Availability & Partition Tolerance)**
  - Prioritizes **availability**: always returns a response even under partition.
  - Eventual consistency model.
  - Good for: IoT, logging, e-commerce with huge scale where slight delays are acceptable.

---

3. DynamoDB Consistency Models

- **Strongly Consistent Read**:
  - Guarantees the read reflects the latest write.
  - Higher latency.
- **Eventually Consistent Read**:
  - Faster response.
  - Might not immediately reflect latest write.

Observation (AWS Free Tier test):

- Write item → Immediate strongly consistent read returns updated value.
- Eventual read may still return old value for a few ms → but much faster.

Trade-off: Consistency vs. Latency.

---

4. Data Modeling Practice

Scenario: Blog Platform (Users, Posts, Comments)

A. Document DB (MongoDB)

```
{
  "user_id": 1,
  "name": "Chetan",
  "posts": [
    {
      "post_id": 101,
      "title": "My First Blog",
      "comments": [
        {"comment_id": 1, "text": "Great post!", "author": "Varma"}
      ]
    }
  ]
}
```

- Embedded documents → fast access in one query.
- Best for apps with flexible schemas.

B. Graph DB (Neo4j)

```
(User)-[:WRITES]→(Post)-[:HAS_COMMENT]→(Comment)
```

- Nodes: User, Post, Comment
- Relationships: WRITES, HAS_COMMENT
- Best for relationship queries like "find all comments by friends of a user".

---

5. Vertical vs. Horizontal Scaling

Case Study: Amazon Prime Day (Amazon itself)

- Moved critical shopping cart service from Oracle (RDS) to DynamoDB.
- **Challenge**: Oracle struggled with **vertical scaling** (bigger servers → limited growth).
- **Solution**: DynamoDB offered **horizontal scaling** (auto-sharding across servers).
- **Benefits**:
  - Millisecond response times at massive scale.
  - Handled **billions of requests/day** with no downtime.
  - Pay-per-use pricing.

---

6. Comparing Database Flavors

Document Databases: MongoDB vs Couchbase

Feature	MongoDB	Couchbase
Data Model	BSON (JSON-like)	JSON Documents

Feature	MongoDB	Couchbase
Query Language	Mongo Query Language (MQL)	N1QL (SQL for JSON)
Strengths	Developer-friendly, flexible	Built-in caching, strong mobile sync
Weaknesses	Sharding complexity	Heavier resource usage
Use Cases	Analytics, Content Mgmt, IoT	Real-time apps, Offline-first mobile