F21BD Big Data Management

CAP Theorem & NoSQL Databases

Master Exam Preparation Set

Topics Covered:

- Big Data History & Properties (5 V's)
- Distributed Database Systems
- CAP Theorem & PACELC
- NoSQL Database Models
- ACID vs. BASE Principles

Prepared: February 13, 2026

Contents

1 Big Data Fundamentals

1.1 What is Big Data? (ELI5 Explanation)

Simple Definition

Big Data = Data that is **TOO BIG** or **TOO FAST** or **TOO MESSY** for traditional databases to handle easily.
Think of it like trying to store an entire library in your bedroom. A normal bookshelf (traditional database) can't hold millions of books. You need a warehouse system (Big Data system).

1.2 Brief History Timeline (Exam Essentials Only)

Memorize These Key Dates:

Year	Event
1663	John Graunt - First statistical data analysis (bubonic plague study)
1880	Herman Hollerith - Created tabulating machine (reduced US Census processing from 8 years to 3 months)
1989	Tim Berners-Lee - Created World Wide Web (WWW)
2005	**Roger Magoulas coins term "Big Data"** - Business Intelligence software couldn't handle data
2005	**Hadoop created** (based on Nutch + Google MapReduce)
2013	IoT (Internet of Things) generalized

Table 1: Big Data History - Exam Key Dates

Exam Trap

Don't confuse: "Big Data" was coined in **2005**, not when WWW was created (1989).

1.3 Data Scale Units (Must Memorize!)

Data Unit Ladder

Each unit is **1000 times bigger** than the previous one:

$$\begin{aligned}
1\text{ KB (kilobyte)} &= 1000\text{ bytes} = 10^3\text{ bytes}\\
1\text{ MB (megabyte)} &= 1000\text{ KB} = 10^6\text{ bytes}\\
1\text{ GB (gigabyte)} &= 1000\text{ MB} = 10^9\text{ bytes}\\
1\text{ TB (terabyte)} &= 1000\text{ GB} = 10^{12}\text{ bytes}\\
1\text{ PB (petabyte)} &= 1000\text{ TB} = 10^{15}\text{ bytes}\\
1\text{ EB (exabyte)} &= 1000\text{ PB} = 10^{18}\text{ bytes}\\
1\text{ ZB (zettabyte)} &= 1000\text{ EB} = 10^{21}\text{ bytes}\\
1\text{ YB (yottabyte)} &= 1000\text{ ZB} = 10^{24}\text{ bytes}
\end{aligned}$$

Memory Trick: King Henry Died By Drinking Chocolate Milk (KB, MB, GB, TB, PB, EB, ZB, YB)

1.4 The 5 V's of Big Data (Core Exam Concept)

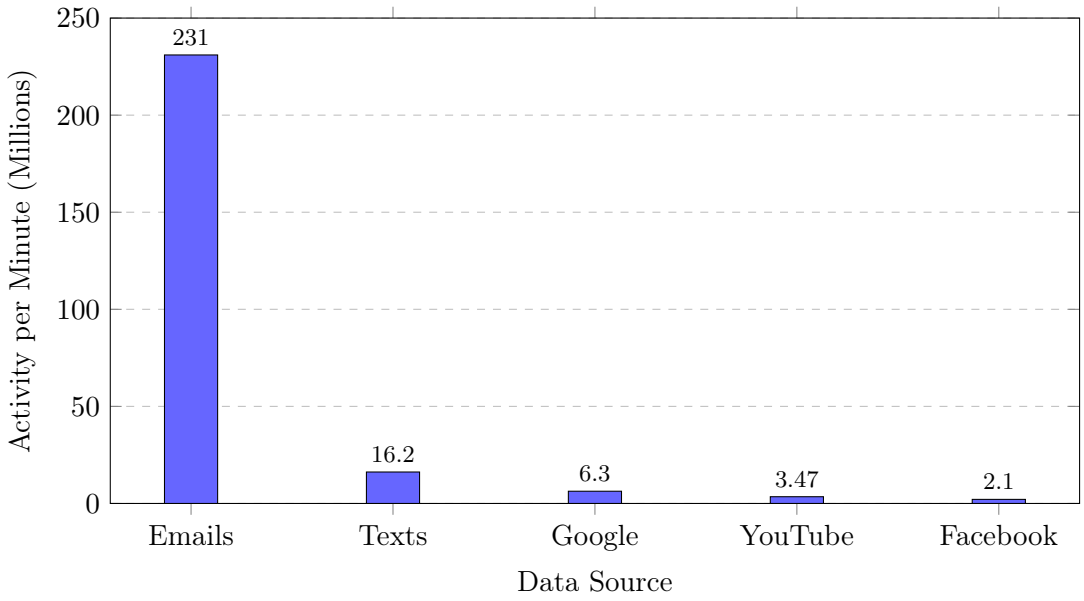
5 V's - You MUST Know These

1. **Volume** = How much data? (Size/Quantity)
   - Example: Facebook stores 300+ petabytes of user data
2. **Velocity** = How fast is data coming in? (Speed)
   - Example: Twitter processes 500+ million tweets per day
3. **Variety** = What types of data? (Structured/Unstructured/Semi-structured)
   - Structured: Excel tables, SQL databases
   - Unstructured: Videos, emails, social media posts
   - Semi-structured: JSON, XML files
4. **Veracity** = Is the data trustworthy? (Quality/Accuracy)
   - Includes noise, inconsistency, completeness, timeliness
   - Example: Sensor data might have errors or missing values
5. **Value** = What insights can we extract? (Meaning)
   - Raw data is useless without analysis
   - Example: Amazon recommendations from purchase history

1.5 Data Production Statistics (Exam Numbers)

Activity (Per Minute)	Count
Google searches	6.3 million
Emails sent	231 million
Texts sent	16.2 million
Netflix streaming	452,000 hours
YouTube videos watched	3.47 million
Facebook active users	2.1 million
TikTok videos watched	625 million

Table 2: Data Generated Every Single Minute (Memorize 2-3 for exam examples)



1.6 IoT Data Generation Examples

IoT Source	Data/Day	Transmitted
Connected Plane	40 TB	0.1%
Connected Factory	1 PB	0.2%
Smart Hospital	5 TB	0.1%
Intelligent Building	275 GB	1%
Smart Car	70 GB	0.1%
Smart Grid	5 GB	1%
Weather Sensors	10 MB	5%

Table 3: IoT Device Data Generation (Exam: Know 2-3 examples)

Critical Exam Point

Notice that most IoT devices generate HUGE amounts of data but only transmit a tiny percentage (0.1%-5%). This is because:

- Bandwidth is expensive
- Most data is not useful
- Only "interesting" events are transmitted

Exam Question Type: "Why don't IoT devices transmit all their data?" Answer: Cost, bandwidth limitations, and most data lacks value.

1.7 Correlation vs Causation (Critical Concept)

WARNING: Correlation Causation

Correlation = Two things change together (might be coincidence)
Causation = One thing *causes* the other to change
Formula: Correlation coefficient r ranges from -1 to +1

- $r = 1$: Perfect positive correlation
- $r = 0$: No correlation
- $r = -1$: Perfect negative correlation

Classic Exam Examples from Lecture:

Correlation	r value	Why It's Spurious
Arcade revenue vs CS doctorates	0.985 (98.5%)	Both increased due to tech boom, not causation
Cheese consumption vs Bedsheet deaths	0.947 (94.7%)	Pure coincidence
Pool drownings vs Nicolas Cage films	0.666 (66.6%)	Random chance

Table 4: Spurious Correlations - Exam Warning Examples

Exam Trap Question

"Google Flu Trends had high correlation with CDC data. Does this mean Google searches *cause* flu?"
Answer: NO! People search for flu symptoms *because* they have flu. The flu causes searches, not the other way around. This is correlation used for *prediction*, not causation.

2 Database Technologies

2.1 Relational Databases (RDBMS)

RDBMS Quick Facts

- Invented in the **1970s**
- Based on **relational algebra** (mathematical theory)
- Uses **SQL** (Structured Query Language)
- Examples: Oracle, MySQL, PostgreSQL, MS SQL Server
- Best for: Centralized storage with strict consistency

2.2 ACID Properties (Must Memorize!)

ACID

Atomicity:

- Transaction is "all or nothing"
- If one part fails, entire transaction fails
- Example: Bank transfer - money leaves Account A AND enters Account B, or neither happens

Consistency:

- Database remains in valid state before and after transaction
- All rules (constraints) are followed
- Example: Account balance can't be negative (if that's a rule)

Isolation:

- Multiple transactions don't interfere with each other
- Transactions execute as if they're alone
- Example: Two people withdrawing from same account simultaneously won't cause errors

Durability:

- Once transaction commits, changes are permanent
- Data persists even if system crashes
- Example: Power outage after purchase won't undo your order

Memory Trick: "A Car Is Durable" = Atomicity, Consistency, Isolation, Durability

2.3 Limitations of RDBMS for Big Data

Problem	Explanation
Slow with massive data	Response time increases dramatically with billions of rows
Expensive to scale up	Adding more CPU/RAM to one server is very costly (vertical scaling)
Limited analytical capabilities	Needs special schemas (OLAP) for complex analytics
Hard to distribute	Maintaining ACID across multiple servers is difficult

Table 5: Why RDBMS Struggles with Big Data

3 Distributed Database Systems (DDBMS)

3.1 What is a Distributed Database? (ELI5)

Simple Definition

A **Distributed Database** = Data stored across **multiple computers** (called nodes) connected by a network.
Think of it like a library with multiple buildings. Instead of one giant building with all books, you have several smaller buildings, each holding some books. A computer system helps you find which building has the book you need.

3.2 Two Key Techniques: Fragmentation & Replication

3.2.1 Fragmentation (Partitioning)

Fragmentation Definition

Fragmentation = Splitting a large table into smaller pieces (fragments) stored on different servers.
Why? No single server can hold all the data.
Two Types:

1. **Horizontal Fragmentation** = Split by rows
2. **Vertical Fragmentation** = Split by columns

Worked Example - Horizontal Fragmentation:
Original Table: 1 million customer records

CustomerID	Name	City	Country
1	Alice	London	UK
2	Bob	Paris	France
3	Charlie	Berlin	Germany
...	...	...	...

Table 6: Original Customer Table

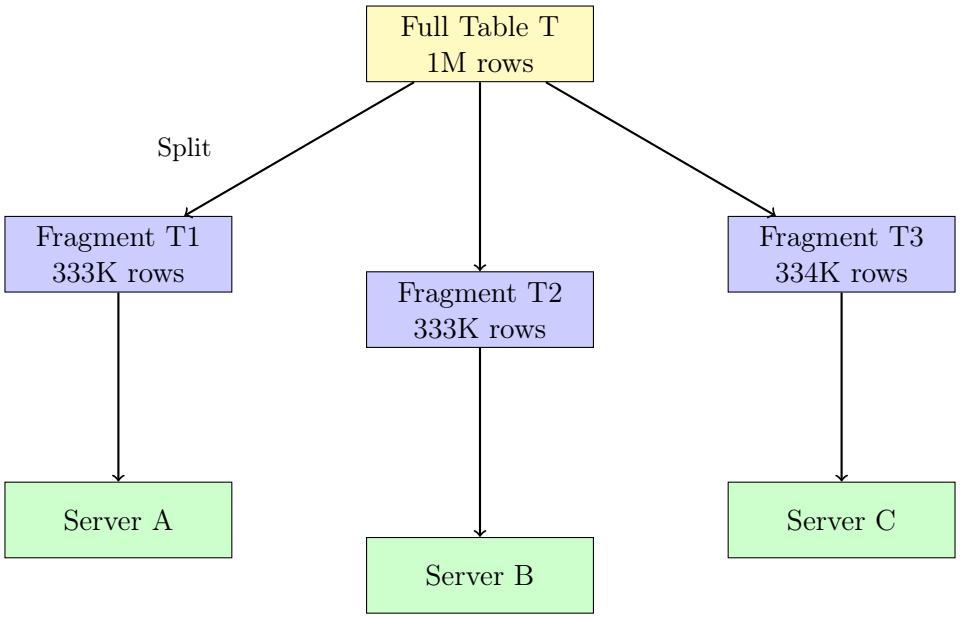
After Horizontal Fragmentation by Country:

- **Fragment 1 (Server A):** All UK customers (rows where Country = 'UK')
- **Fragment 2 (Server B):** All France customers (rows where Country = 'France')

- **Fragment 3 (Server C):** All Germany customers (rows where Country = 'Germany')

Vertical Fragmentation Example:
Split Customer table by columns:

- **Fragment 1 (Server A):** CustomerID, Name (frequently accessed)
- **Fragment 2 (Server B):** CustomerID, Address, Phone (less frequently accessed)



3.2.2 Replication

Replication Definition

Replication = Creating copies of the same data on multiple servers.
Why?

- **Availability:** If one server fails, data still accessible from copies
- **Performance:** Multiple users can read from different copies simultaneously
- **Disaster Recovery:** Backup in case of hardware failure

Example: Fragment T1 has 3 copies:

- Copy1 on Server A (primary)
- Copy2 on Server B (backup)
- Copy3 on Server C (backup)

Exam Question Type

"What's the difference between fragmentation and replication?"
Answer:

- **Fragmentation:** Data is *split* - each piece is different
- **Replication:** Data is *copied* - each piece is the same

You can use BOTH together! Fragment the data, then replicate each fragment.

3.3 Database Sharding

Sharding

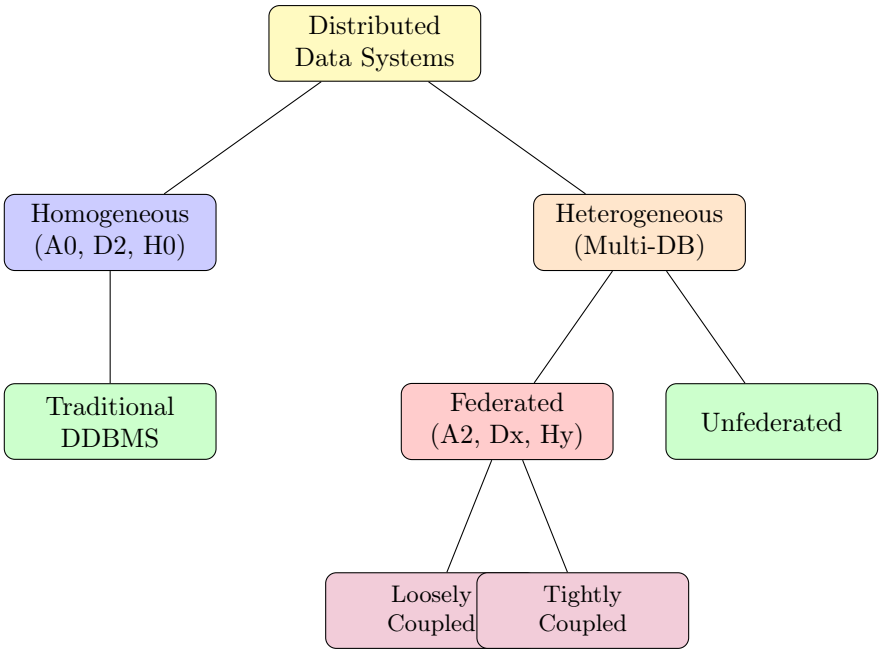
Sharding = Splitting database into "shards" (pieces) across different computers.
Key Rule: Shards MUST be on different servers.
Real Example: Facebook split MySQL database into 4,000 shards in 2011 to handle massive user data.

3.4 Taxonomy of Distributed Systems

Three dimensions to classify distributed databases:

Three Classification Dimensions

1. **Distribution** (How is data shared?)
   - D0: No distribution (single server)
   - D1: Client-server (one server, many clients)
   - D2: Peer-to-peer (multiple equal servers)
2. **Heterogeneity** (Do servers use same technology?)
   - H0: Homogeneous (all servers use same DBMS, e.g., all MySQL)
   - H1: Heterogeneous (servers use different DBMS, e.g., MySQL + Oracle)
3. **Autonomy** (How independently do servers operate?)
   - A0: Tight integration (single coordinator controls all)
   - A1: Semi-autonomous (servers cooperate but have some independence)
   - A2: Independent (servers operate independently, sharing handled by separate software)


Distributed
Data Systems
Homogeneous
(A0, D2, H0)
Heterogeneous
(Multi-DB)
Traditional
DDBMS
Federated
(A2, Dx, Hy)
Unfederated
Loosely
Coupled
Tightly
Coupled

4 CAP Theorem (CRITICAL EXAM TOPIC)

4.1 The Three Properties (Core Definition)

CAP Theorem - The Foundation

In a distributed database system, you can only guarantee **TWO out of THREE** properties:
Consistency:

- All nodes see the same data at the same time
- A read operation returns the most recent write
- Example: After updating your profile picture on Facebook, all friends see the new picture immediately

Availability:

- System answers every request (even if some nodes fail)
- Every request gets a response (but might not be the latest data)
- Example: Website stays online even if one server crashes

Partition Tolerance:

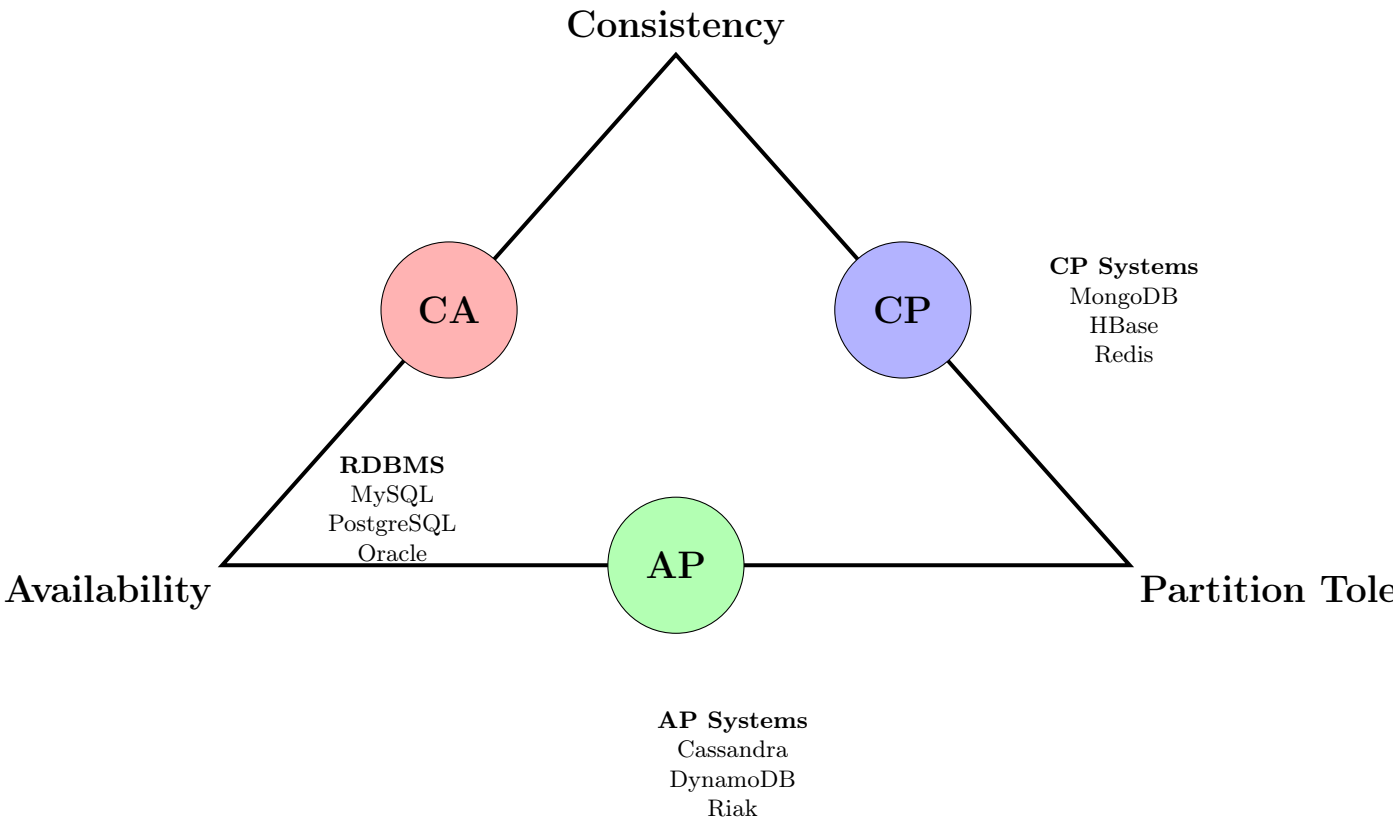
- System continues working even if network fails between nodes
- Can handle message loss between servers
- Example: If cable between two data centers breaks, system still functions

CAP Theorem Statement (Memorize This)

Theorem: A distributed system can satisfy **at most TWO** of the three properties (C, A, P) simultaneously.
History:

- 2000: Eric Brewer proposed as conjecture
- 2002: Gilbert and Lynch proved it mathematically (became theorem)

4.2 The Three Combinations (Exam Essential)



4.2.1 CA: Consistency + Availability (No Partition Tolerance)

CA Systems

Examples: Traditional RDBMS (MySQL, PostgreSQL, Oracle)
How it works:

- Data is consistent (all nodes have same data)
- System is available (always responds to requests)
- **BUT:** If network splits (partition), system fails or becomes inconsistent

Techniques Used:

- Replication for high availability
- Two-phase commit protocol for consistency
- Transaction locks

When partition happens:

- Must choose: favor Consistency (CP) or Availability (AP)
- If favor consistency: system refuses to respond (sacrifice availability)
- If favor availability: system responds with potentially wrong data (sacrifice consistency)

Worked Example:

Imagine a bank with 2 servers (Server A and Server B):

1. Both servers have Account Balance = \$1000
2. Network cable between servers breaks (partition!)
3. User tries to withdraw \$500 at ATM connected to Server A
4. At same time, another user tries to withdraw \$700 at ATM connected to Server B

CA Problem:

- Servers can't communicate to ensure consistency
- If both withdrawals succeed → inconsistent! (balance would be -\$200)
- CA system would refuse both transactions (system unavailable) until network fixed

4.2.2 CP: Consistency + Partition Tolerance (Limited Availability)

CP Systems

Examples: MongoDB, HBase, Redis, BigTable
How it works:

- Data is consistent (all nodes agree on same data)
- System handles network partitions
- **BUT:** System may refuse to respond during partition (unavailable)

Techniques Used:

- Quorum protocols (majority voting)
- Paxos algorithm
- Raft consensus algorithm

Trade-off: System waits for nodes to agree before responding → slower response time → limited availability

Quorum Explained (CP Systems)

Quorum = Majority voting to ensure consistency.
Formula:

$$\text{Quorum} = \left\lfloor \frac{N}{2} \right\rfloor + 1$$

Where N = total number of nodes.
Example Calculation:

- If $N = 5$ nodes: Quorum $= \lfloor 5/2 \rfloor + 1 = 2 + 1 = 3$ nodes
- Need 3 out of 5 nodes to agree for write to succeed

Step-by-step:

1. Divide total nodes by 2: $5 \div 2 = 2.5$
2. Round down (floor): $\lfloor 2.5 \rfloor = 2$

3. Add 1: $2 + 1 = 3 \rightarrow$ Need 3 nodes

Worked Example - Quorum in Action:
Database with 5 replicas (Nodes A, B, C, D, E):

1. User updates a record
2. System sends update to all 5 nodes
3. Network partition: Nodes A, B, C in one group; Nodes D, E in another
4. Quorum = 3 nodes needed
5. Group 1 (A, B, C) has 3 nodes → Can process update (maintains consistency)
6. Group 2 (D, E) has only 2 nodes → Refuses update (unavailable to maintain consistency)

4.2.3 AP: Availability + Partition Tolerance (Eventual Consistency)

AP Systems

Examples: Cassandra, DynamoDB, Riak, CouchDB
How it works:

- System always responds (available)
- System handles network partitions
- **BUT:** Data might be temporarily inconsistent (eventual consistency)

Eventual Consistency:

- Nodes may have different data temporarily
- System guarantees: given enough time (and no new updates), all nodes will converge to same data
- "Eventually" = seconds to minutes (depending on system)

Worked Example - Eventual Consistency:
Social media post with 3 replicas (Nodes X, Y, Z):

1. User posts "Hello World!" at 10:00:00
2. Post written to Node X immediately
3. Network is slow (partition)
4. **10:00:01** - Friend A reads from Node X: sees "Hello World!"
5. **10:00:02** - Friend B reads from Node Y: sees nothing (inconsistent!)
6. **10:00:05** - Update reaches Node Y
7. **10:00:05** - Friend B refreshes: now sees "Hello World!"
8. **10:00:10** - All nodes have same data (eventual consistency achieved)

Exam Question Type

"Why does Facebook sometimes show you already saw a notification?"
Answer: Facebook uses AP system (eventual consistency). When you mark notification as read:

- Update written to one server
- Other servers not updated immediately
- You switch to different server → sees old data (notification unread)
- After sync completes (seconds later) → all servers know it's read

4.3 CAP Trade-offs Table

System Type	C	A	P	Use Case
CA				Banking, financial transactions (network failure unacceptable)
CP				Inventory systems, booking systems (must be accurate)
AP				Social media, caching, DNS (uptime critical, slight delays OK)

Table 7: CAP Trade-offs - Which to Choose?

4.4 PACELC Theorem (Extended CAP)

PACELC Theorem

CAP only considers partitions. PACELC extends it:
Statement:

- **IF** Partition (P): trade-off between Availability (A) and Consistency (C)
- **ELSE** (no partition): trade-off between Latency (L) and Consistency (C)

Why? Even without network failure, distributed systems must choose:

- **Low Latency:** Respond fast (might have slightly old data)
- **Consistency:** Ensure all nodes agree (takes longer)

Latency Explained (ELI5):
Latency = Time delay between request and response.
Example:

- You ask Alexa a question
- Latency = time from finish speaking until Alexa starts answering
- Low latency = fast response (milliseconds)
- High latency = slow response (seconds)

System	Partition (PA or PC)	Else (EL or EC)
Dynamo	PA (choose Availability)	EL (choose Latency)
Cassandra	PA (choose Availability)	EL (choose Latency)
Riak	PA (choose Availability)	EL (choose Latency)
MongoDB	PC (choose Consistency)	EC (choose Consistency)
HBase	PC (choose Consistency)	EC (choose Consistency)

Table 8: PACELC Classifications

5 NoSQL Databases

5.1 What is NoSQL? (Complete Definition)

NoSQL Meaning

Originally: "No SQL" (anti-SQL movement)
Now: "Not Only SQL" (gentler definition)
Simple Definition: Databases that DON'T use traditional tables/rows/columns like SQL databases.
History:

- 1998: Carlo Strozzi used "noSQL" for a relational database with no SQL interface
- 2009: NoSQL conference at Rackspace (modern meaning starts here)
- 2009: Driven by web companies needing massive scale (Facebook, Google, Amazon)

5.2 BASE Principles (Alternative to ACID)

BASE

Basic Availability:

- System appears to work most of the time
- Uses replication across servers
- Even if some nodes fail, system still available

Soft State:

- State of system may change over time (even without input)
- Consistency is developer's responsibility, not database's
- Data might be in flux

Eventual Consistency:

- System will become consistent in the future
- Operations can execute without waiting for consistency
- "Eventually" means: given enough time with no new updates, all nodes will have same data

ACID (RDBMS)	BASE (NoSQL)
Strong consistency	Eventual consistency
Isolation	Soft state
Focus on "commit"	Best effort
Pessimistic (assumes problems)	Optimistic (assumes success)
Vertical scaling (scale up)	Horizontal scaling (scale out)

Table 9: ACID vs BASE - Key Differences

5.3 NoSQL Characteristics (Exam Checklist)

NoSQL Key Features

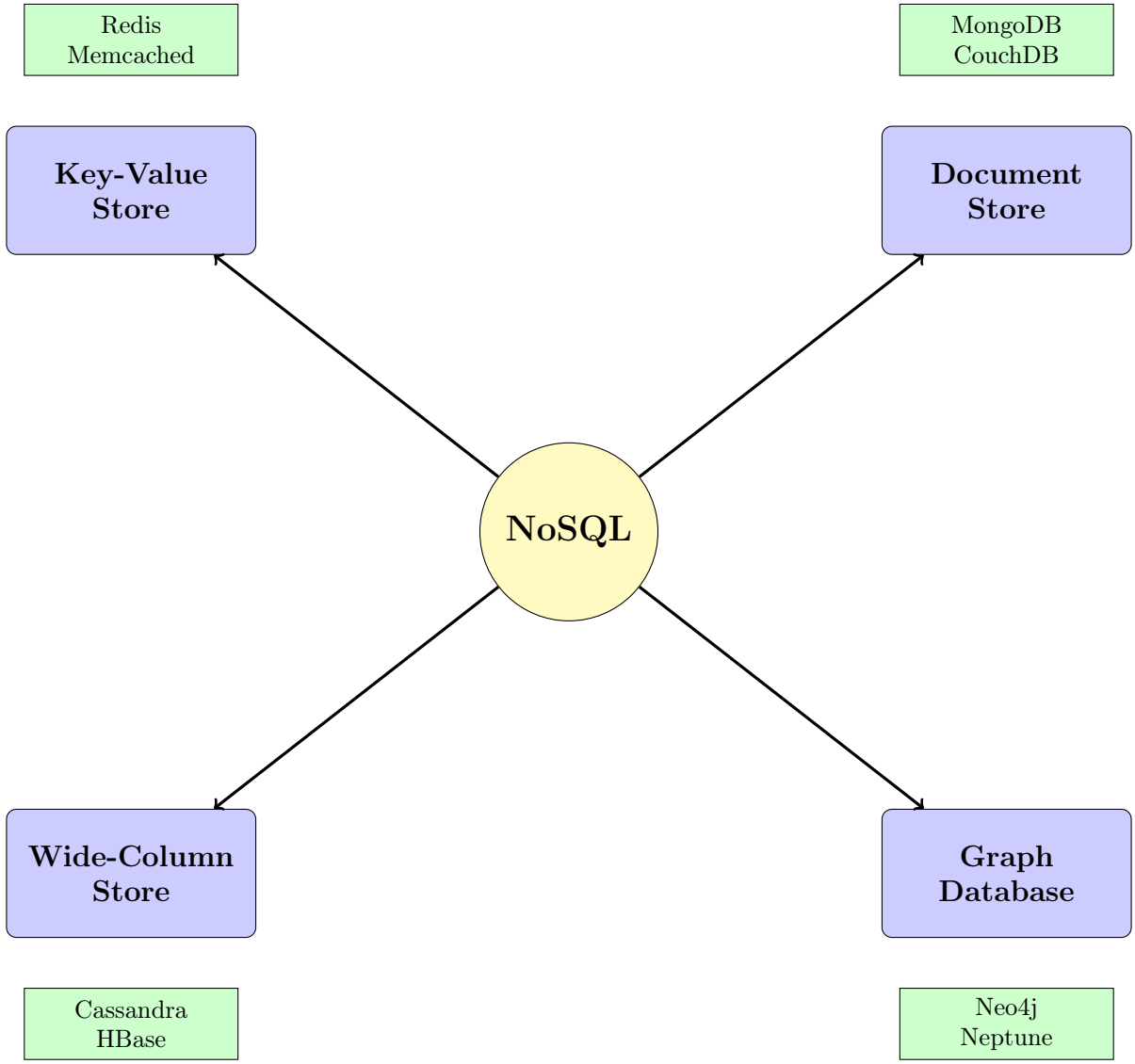
What NoSQL HAS:

1. Flexible schema (no fixed structure)
2. Horizontal scalability (add more servers easily)
3. High availability (replication across clusters)
4. Eventual consistency (BASE principles)
5. Built for distributed systems

What NoSQL LACKS:

1. Fixed schema (data model can evolve)
2. Standard SQL query language (each has own API)
3. ACID transactions (no transaction guarantees)
4. Complex joins (must be done in application code)

5.4 Four NoSQL Data Models



5.4.1 Key-Value Store

Key-Value Store Model

Structure: Data stored as (Key → Value) pairs in hashtable.
Key: Unique identifier (like dictionary key)
Value: Opaque blob (database doesn't know what's inside)
Operations:

- `GET(key)` → retrieve value
- `PUT(key, value)` → store/update value
- `DELETE(key)` → remove entry

Characteristics:

- Extremely fast (O(1) lookups)
- Schema-less

- Usually in-memory (for speed)
- No complex queries

Worked Example - Session Storage:

```
# Storing user session in Redis (Key-Value store)

# Key = session ID
# Value = user data (JSON)

# Store session
key = "session:abc123"
value = {
    "user_id": 42,
    "username": "alice",
    "login_time": "2026-02-13T14:30:00",
    "cart": ["item1", "item2"]
}

redis.set(key, json.dumps(value))  # Store

# Retrieve session
session_data = redis.get(key)  # Get
print(session_data)
# Output: {"user_id": 42, "username": "alice", ...}
```

Real-World Use Cases:

- Session storage (web applications)
- Caching (speed up database queries)
- Real-time leaderboards (gaming)
- Shopping carts (e-commerce)

Examples: Redis, Memcached, Amazon DynamoDB, Riak KV

5.4.2 Document Store

Document Store Model

Structure: Data stored as documents (usually JSON or XML).
Key Difference from Key-Value: Database understands document structure and can query inside it.
Document: Self-contained unit with nested key-value pairs.
Features:

- Documents have unique key (like Key-Value)
- Database can query specific fields inside document
- Indexes on document fields for fast search
- Flexible schema (each document can have different fields)

Worked Example - User Profile:

```
# MongoDB document example (JSON format)

# Document ID: user123
{
    "_id": "user123",
    "name": "Alice␣Smith",
    "email": "alice@example.com",
    "age": 28,
    "address": {
        "street": "123␣Main␣St",
        "city": "Dubai",
        "country": "UAE"
    },
    "hobbies": ["reading", "coding", "travel"],
    "friends": [
        {"id": "user456", "name": "Bob"},
        {"id": "user789", "name": "Charlie"}
    ]
}

# Query examples in MongoDB:

# 1. Find user by email
db.users.find({"email": "alice@example.com"})

# 2. Find users in Dubai
db.users.find({"address.city": "Dubai"})

# 3. Find users who like coding
db.users.find({"hobbies": "coding"})

# 4. Update age
db.users.update(
    {"_id": "user123"},
    {"$set": {"age": 29}}
)
```

Comparison: Key-Value vs Document

Feature	Key-Value	Document
Structure	Opaque blob (database doesn't look inside)	Structured document (database understands fields)
Queries	Only by key	By key AND by any field inside document
Indexing	Only on key	On key and document fields
Speed	Faster (simpler)	Slightly slower (more features)
Use Case	Caching, sessions	Content management, user profiles

Table 10: Key-Value vs Document Stores

Real-World Use Cases:

- Content management systems (blogs, articles)
- Product catalogs (e-commerce)
- User profiles (social media)

- Configuration files

Examples: MongoDB, CouchDB, Elasticsearch, Amazon DocumentDB

5.4.3 Wide-Column Store

Wide-Column Store Model

Structure: Tables with rows and columns, BUT columns are flexible.
Key Difference from RDBMS: Each row can have different columns.
Also Called: Column-family databases
How it works:

- Data organized by column families (groups of related columns)
- Columns stored separately (vertical sharding)
- Sparse data matrix: not all columns have values
- Efficient for analytical queries on specific columns

Worked Example - User Activity Tracking:

```
Row Key: user123
+----------------+----------------+----------------+
| Column Family  | Column         | Value          |
+----------------+----------------+----------------+
| Profile        | name           | Alice          |
|                | email          | alice@mail.com |
|                | age            | 28             |
+----------------+----------------+----------------+
| Activity       | last_login     | 2026-02-13     |
|                | posts_count    | 42             |
+----------------+----------------+----------------+

Row Key: user456
+----------------+----------------+----------------+
| Column Family  | Column         | Value          |
+----------------+----------------+----------------+
| Profile        | name           | Bob            |
|                | email          | bob@mail.com   |
+----------------+----------------+----------------+
| Activity       | last_login     | 2026-02-12     |
|                | posts_count    | 15             |
|                | comments_count | 100            |
+----------------+----------------+----------------+
```

Notice: User456 has `comments_count` but User123 doesn't. This is OK in wide-column stores!

Cassandra Query Example:

```
-- Create column family
CREATE TABLE users (
    user_id text PRIMARY KEY,
    name text,
    email text,
```

```
    last_login timestamp,
    posts_count int
);

-- Insert data (flexible columns)
INSERT INTO users (user_id, name, email, last_login)
VALUES ('user123', 'Alice', 'alice@mail.com', '2026-02-13');

-- Query by row key
SELECT * FROM users WHERE user_id = 'user123';

-- Query specific columns only
SELECT name, last_login FROM users WHERE user_id = 'user123';
```

Sharding in Wide-Column Stores:

- **Horizontal Sharding:** Rows distributed across servers
  - Example: Users A-M on Server 1, Users N-Z on Server 2
- **Vertical Sharding:** Column families on different servers
  - Example: "Profile" columns on Server 1, "Activity" columns on Server 2

Real-World Use Cases:

- Time-series data (sensor readings, logs)
- Event logging (user actions, clicks)
- Analytical queries (business intelligence)
- Large-scale data warehouses

Examples: Cassandra, HBase, Google BigTable, Amazon Keyspaces

5.4.4 Graph Database

Graph Database Model

Structure: Data stored as graph (nodes + edges).
Nodes: Entities (people, products, locations)
Edges: Relationships between nodes (with properties)
Why? Optimized for queries about relationships.
Key Strength: Traverse connections extremely fast (vs joins in SQL).

Worked Example - Social Network:

Neo4j Query Example (Cypher language):

```
// Create nodes
CREATE (alice:Person {name: "Alice", age: 28})
CREATE (bob:Person {name: "Bob", age: 30})
CREATE (post:Post {content: "Hello World!", date: "2026-02-13"})

// Create relationships
CREATE (alice)-[:FRIENDS {since: 2020}]->(bob)
CREATE (bob)-[:WROTE]->(post)
CREATE (alice)-[:LIKED {timestamp: "2026-02-13T15:00"}]->(post)

// Query: Find Alice's friends
MATCH (alice:Person {name: "Alice"})-[:FRIENDS]->(friend)
RETURN friend.name

// Query: Find friends of friends (2 hops)
MATCH (alice:Person {name: "Alice"})-[:FRIENDS]->()-[:FRIENDS]->(fof)
RETURN fof.name

// Query: Shortest path between Alice and Charlie
MATCH path = shortestPath(
    (alice:Person {name: "Alice"})-[*]-(charlie:Person {name: "Charlie"})
)
RETURN path

// Query: Find posts liked by Alice's friends
MATCH (alice:Person {name: "Alice"})-[:FRIENDS]->(friend)
    -[:LIKED]->(post:Post)
RETURN post.content, friend.name
```

Why Graph Beats SQL for Relationships:

Operation	SQL (Relational)	Graph DB
Find direct friends	1 JOIN	1 traversal (fast)
Friends of friends	Multiple JOINs (slow)	2 traversals (still fast)
Friends of friends of friends	Even more JOINs (very slow)	3 traversals (linear scaling)
Shortest path	Complex recursive query	Built-in algorithm

Table 11: SQL vs Graph for Relationship Queries

Real-World Use Cases:

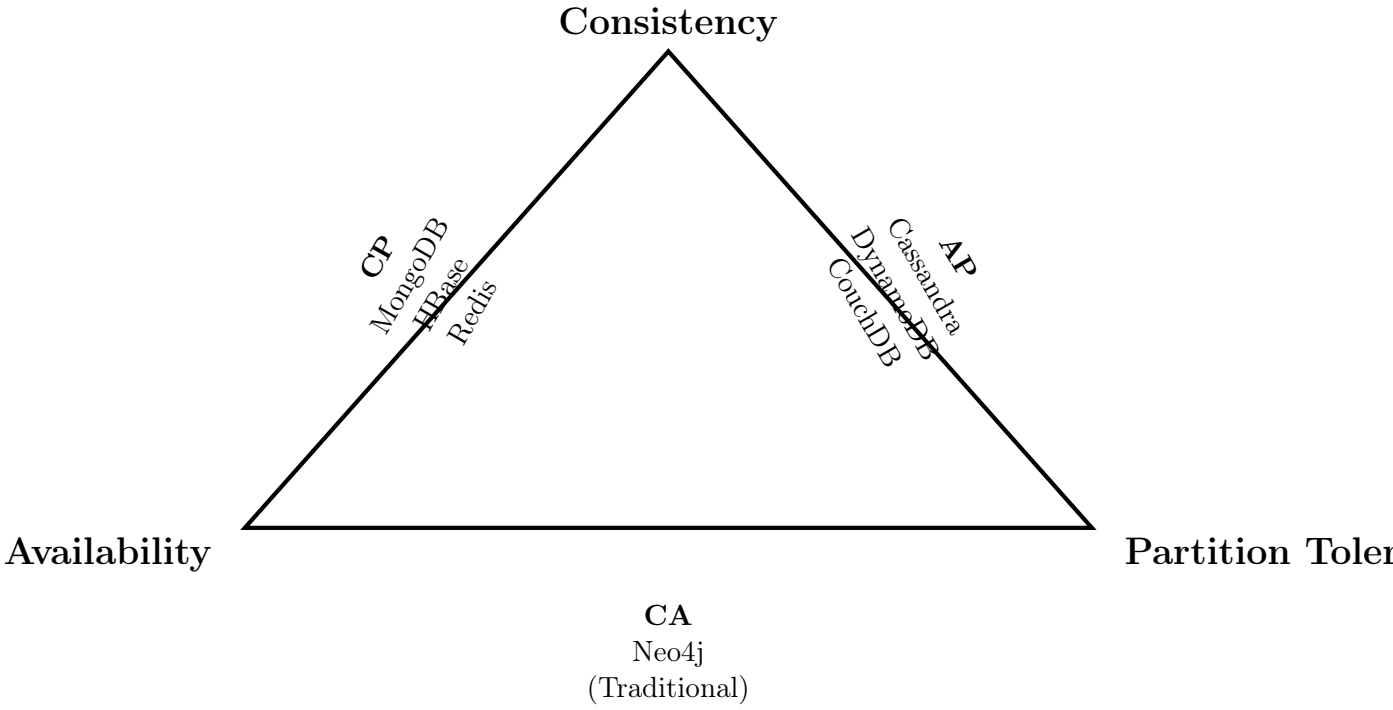
- Social networks (Facebook, LinkedIn)
- Recommendation engines (Amazon, Netflix)
- Fraud detection (banking)
- Network topology (telecom)
- Knowledge graphs (Google Knowledge Graph)

Examples: Neo4j, Amazon Neptune, TigerGraph, ArangoDB

5.5 NoSQL and CAP Classification

NoSQL Type	Database	CAP Model
Key-Value	Redis	CP
	Amazon DynamoDB	AP
Document	MongoDB	CP
	CouchDB	AP
Wide-Column	HBase	CP
	Cassandra	AP
Graph	Neo4j	CA (typically)

Table 12: NoSQL Databases and CAP Properties



5.6 Database Usage Statistics

Rank	Database	Type
1	Oracle	RDBMS
2	MySQL	RDBMS
3	Microsoft SQL Server	RDBMS
4	PostgreSQL	RDBMS
5	MongoDB	NoSQL (Document)
6	Redis	NoSQL (Key-Value)
7	Elasticsearch	NoSQL (Document)
8	IBM DB2	RDBMS
9	SQLite	RDBMS
10	Cassandra	NoSQL (Wide-Column)

Table 13: Top 10 Most Popular Databases (Exam: Know RDBMS still dominates!)

6 Exam Strategy & Quick Reference

6.1 High-Probability Exam Questions

Question Type 1: Definitions

Example: "Define the CAP theorem and explain the three properties."
Answer Template:

- State theorem: "A distributed system can guarantee at most TWO of three properties"
- Define C: "All nodes see same data simultaneously, read returns most recent write"
- Define A: "Every request gets response, even with node failures"
- Define P: "System continues despite network partition/message loss"
- Mention history: Brewer 2000 (conjecture), Gilbert & Lynch 2002 (proof)

Question Type 2: Comparison Tables

Example: "Compare ACID and BASE principles."
Answer Template: Create table with 4-5 dimensions:

- Consistency model (Strong vs Eventual)
- Transaction guarantees (Full vs None)
- Scalability (Vertical vs Horizontal)
- Use cases (Financial vs Social Media)
- Examples (MySQL vs Cassandra)

Question Type 3: Scenario-Based

Example: "You're designing a banking system. Which CAP model should you choose and why?"
Answer Template:

1. Identify requirements: "Banking needs strong consistency (money can't be lost)"
2. Choose model: "CA or CP model"
3. Explain choice: "CP preferred because partitions must be handled, but consistency is critical"
4. Trade-off: "Accept limited availability during network issues rather than inconsistent balances"
5. Technology: "Use RDBMS with replication or MongoDB"

Question Type 4: Calculations

Example: "A system has 7 nodes. What is the minimum quorum?"
Step-by-step:

$$\begin{aligned}\text{Quorum} &= \left\lfloor \frac{N}{2} \right\rfloor + 1 \\ &= \left\lfloor \frac{7}{2} \right\rfloor + 1 \\ &= \lfloor 3.5 \rfloor + 1 \\ &= 3 + 1 \\ &= 4 \text{ nodes}\end{aligned}$$

Answer: Need 4 out of 7 nodes to agree.

6.2 One-Page Cheat Sheet

F21BD Exam Cheat Sheet

5 V's of Big Data:

- Volume (size), Velocity (speed), Variety (types), Veracity (quality), Value (insights)

ACID (RDBMS):

- Atomicity (all-or-nothing), Consistency (valid state), Isolation (no interference), Durability (permanent)

BASE (NoSQL):

- Basic Availability, Soft state, Eventual consistency

CAP Theorem:

- Can only have 2 of 3: Consistency, Availability, Partition Tolerance
- CA = RDBMS (MySQL), CP = MongoDB/HBase, AP = Cassandra/DynamoDB

PACELC Theorem:

- IF Partition: A vs C, ELSE: Latency vs Consistency

4 NoSQL Models:

1. Key-Value: Redis (caching, sessions)
2. Document: MongoDB (user profiles, CMS)
3. Wide-Column: Cassandra (time-series, analytics)
4. Graph: Neo4j (social networks, recommendations)

Fragmentation:

- Horizontal = split by rows, Vertical = split by columns

Quorum Formula:

$$Q = \left\lfloor \frac{N}{2} \right\rfloor + 1$$

Key Dates:

- 2005: "Big Data" coined, Hadoop created
- 2000: CAP conjecture (Brewer)
- 2002: CAP theorem proved (Gilbert & Lynch)

6.3 Common Exam Mistakes to Avoid

1. **Confusing Correlation with Causation**
   - Wrong: "High correlation means one causes the other"
   - Right: "Correlation suggests relationship, but doesn't prove causation"
2. **Saying ALL NoSQL are AP systems**
   - Wrong: "NoSQL databases are AP"
   - Right: "Some NoSQL are AP (Cassandra), others are CP (MongoDB)"

3. **Claiming you can have all 3 CAP properties**
   - Wrong: "System can be consistent, available, and partition-tolerant"
   - Right: "CAP theorem proves you can only have 2 out of 3"
4. **Thinking Eventual Consistency means "never consistent"**
   - Wrong: "AP systems are always inconsistent"
   - Right: "AP systems are temporarily inconsistent, but converge to consistency given time"
5. **Forgetting PACELC extends CAP**
   - Wrong: "CAP only considers partitions"
   - Right: "PACELC adds: even without partitions, trade-off between Latency and Consistency"

6.4 Exam Answer Templates

For "Explain" Questions:

1. Define the concept (1 sentence)
2. Explain how it works (2-3 sentences)
3. Give a real-world example
4. Mention advantages/disadvantages

For "Compare" Questions:

1. Create a comparison table
2. Highlight key differences (3-5 dimensions)
3. Explain when to use each
4. Give examples of each type

For "Calculate" Questions:

1. Write the formula
2. Show substitution with given values
3. Show each calculation step
4. Box final answer with units

For "Design" Questions:

1. List system requirements
2. Identify constraints (CAP choice)
3. Justify technology choice
4. Explain trade-offs
5. Suggest specific database

6.5 Practice Questions with Model Answers

Practice Question 1

Question: Explain the difference between horizontal and vertical fragmentation in distributed databases. Provide an example for each.
Model Answer:
Horizontal Fragmentation:

- Definition: Splitting a table by rows
- Each fragment contains subset of rows
- All fragments have same columns (schema)
- Example: Customer table split by region
  - Fragment 1: Customers in Europe (rows where region='Europe')
  - Fragment 2: Customers in Asia (rows where region='Asia')
  - Fragment 3: Customers in Americas (rows where region='Americas')

Vertical Fragmentation:

- Definition: Splitting a table by columns
- Each fragment contains subset of columns
- Common column (e.g., ID) repeated in all fragments for joining
- Example: Employee table split by access frequency
  - Fragment 1: EmployeeID, Name, Department (frequently accessed)
  - Fragment 2: EmployeeID, Salary, BankDetails (rarely accessed, sensitive)

Key Difference: Horizontal splits data horizontally (by records), Vertical splits data vertically (by attributes).

Practice Question 2

Question: A distributed system has 9 replica nodes. Using the quorum protocol, how many nodes must agree for a write operation to succeed? Show your calculation.
Model Answer:
Formula:

$$\text{Quorum} = \left\lfloor \frac{N}{2} \right\rfloor + 1$$

Given: $N = 9$ nodes
Step-by-step Calculation:

$$\begin{aligned} \text{Quorum} &= \left\lfloor \frac{9}{2} \right\rfloor + 1 \\ &= \lfloor 4.5 \rfloor + 1 \quad \text{(divide 9 by 2)} \\ &= 4 + 1 \quad \text{(floor of 4.5 is 4)} \\ &= 5 \text{ nodes} \end{aligned}$$

Answer: $\boxed{5 \text{ nodes}}$ must agree for a write operation to succeed.

Explanation: 5 is more than half of 9 (majority). This ensures consistency because any two operations will have at least one node in common ($5 + 5 = 10 > 9$), preventing conflicting concurrent writes.

Practice Question 3

Question: Compare ACID and BASE principles. Create a table highlighting at least 4 differences.
Model Answer:

Practice Question 4

Question: You're designing a social media platform (like Twitter). Which NoSQL data model would you choose and why? What CAP properties would you prioritize?
Model Answer:
NoSQL Model Choice: Wide-Column Store (e.g., Cassandra) or Document Store (e.g., MongoDB)
Justification:

1. **Requirements:**
   - Handle millions of users posting simultaneously (high write throughput)
   - Display timeline/feed quickly (low latency reads)
   - User profiles vary (flexible schema)
   - Must stay online 24/7 (high availability)
2. **Why Wide-Column (Cassandra):**
   - Excellent write performance (append-only)
   - Time-series data (posts ordered by timestamp)
   - Horizontal scaling (add nodes easily)
   - Eventual consistency acceptable (seeing post 1 second late is OK)
3. **CAP Choice: AP (Availability + Partition Tolerance)**
   - Must stay online (availability critical)
   - Eventual consistency acceptable (brief inconsistency in timeline OK)
   - Global distribution (partitions likely)
4. **Trade-offs:**
   - Accept: User might see old data briefly (eventual consistency)
   - Gain: Platform never goes down, scales infinitely

Alternative: Document store (MongoDB) if complex queries on user profiles needed, but would choose CP (consistency over availability during partitions).

7 Final Checklist Before Exam

7.1 Topics to Review Night Before

1. Memorize 5 V's of Big Data (Volume, Velocity, Variety, Veracity, Value)
2. Memorize ACID (Atomicity, Consistency, Isolation, Durability)
3. Memorize BASE (Basic Availability, Soft state, Eventual consistency)
4. Know CAP theorem statement by heart
5. Understand CA vs CP vs AP with examples
6. Know PACELC extension
7. Memorize quorum formula: $Q = \lfloor N/2 \rfloor + 1$
8. Know 4 NoSQL models: Key-Value, Document, Wide-Column, Graph
9. Understand fragmentation (horizontal vs vertical)
10. Understand replication purpose
11. Know correlation causation examples
12. Review data unit ladder (KB, MB, GB, TB, PB, EB, ZB, YB)
13. Memorize key dates: 2005 (Big Data coined, Hadoop), 2000/2002 (CAP)

7.2 Quick Mental Checks

Can you answer these instantly?

- What does CAP stand for? (Consistency, Availability, Partition Tolerance)
- Name 3 CA databases? (MySQL, PostgreSQL, Oracle)
- Name 3 AP databases? (Cassandra, DynamoDB, Riak)
- Name 3 CP databases? (MongoDB, HBase, Redis)
- Quorum for 5 nodes? (3)
- Quorum for 11 nodes? (6)
- What year was "Big Data" coined? (2005)
- ACID is for what type of DB? (Relational/RDBMS)
- BASE is for what type of DB? (NoSQL)

If you hesitated on any, review that section NOW!

7.3 Final Exam Tips

1. **Read questions carefully** - Look for keywords: "explain", "compare", "calculate", "design"
2. **Budget your time** - Don't spend 30 minutes on one question
3. **Show your work** - Especially for calculations (partial credit!)
4. **Use diagrams** - Tables, graphs, flowcharts earn points
5. **Define before explaining** - Always start with clear definition
6. **Use examples** - Real-world examples show understanding
7. **Check CAP logic** - Can't have all 3, make sure answer reflects this
8. **Don't leave blanks** - Partial answer better than no answer
9. **Review answers** - Last 10 minutes, check calculations and definitions
10. **Stay calm** - If stuck, move to next question, come back later

Good Luck on Your Exam!

Remember: Understanding concepts > memorizing facts.
If you understand CAP theorem, ACID vs BASE, and the 4 NoSQL models, you're 80% prepared.