Aerospike

Seznam.cz

PRINCIPAL

PRAGUE MEETUP

# Real-Time Data & AI

## Scale, Speed, and Insights

**Zohar Elkayam**

Principal Solutions Architect

Aerospike

Aerospike

# Databases at the Crossroads of Scale, Real-Time, and AI

Choosing the Right Database for the Right Use Case,
don't leave your database choice to chance!

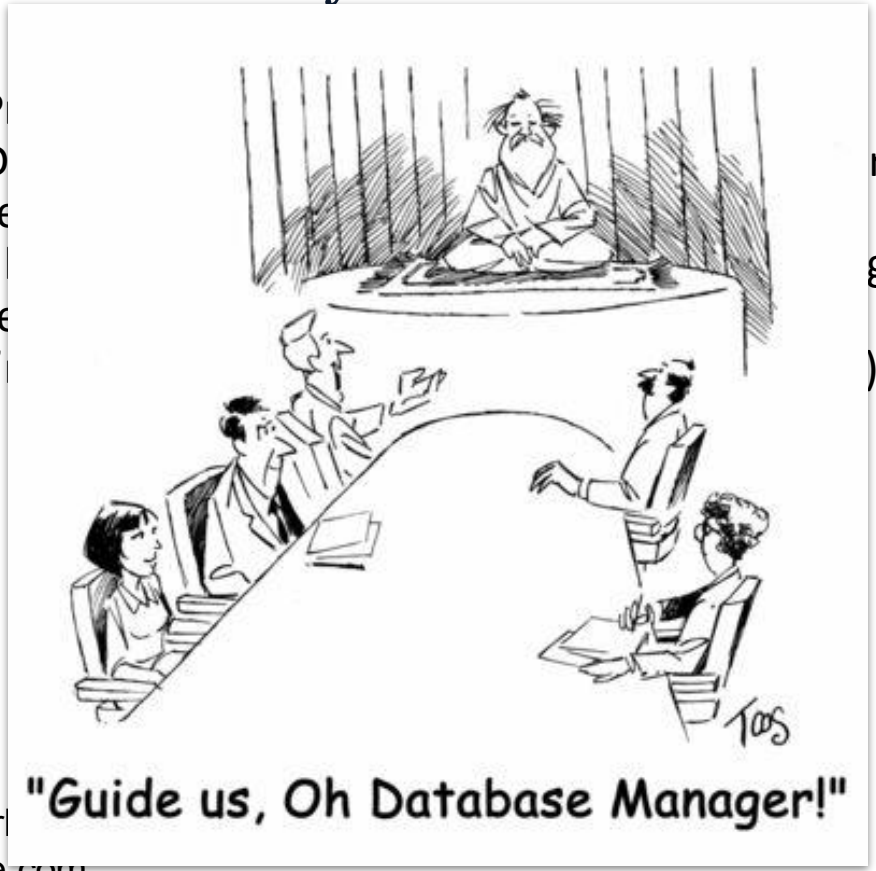
Zohar Elkayam
Principal Solutions Architect
Aerospike

# About Me: Zohar Elkayam

- Principal Solutions Architect at Aerospike.
- Decades of Database Expertise (28+ Years) in relational and non-relational databases.
- I like to talk and write about technology: blogger, public speaker, technology enthusiast.
- I’m driven by Problem-Solving (I love Puzzles!)

# About Me: Zohar Elkayam

- P
- D relational and non-
  re
- I ger, public speaker,
  te
- I’ )




linkedin.com/in/Zoharl
zelkayam@aerospike.com

# Agenda

| | |
|---|---|
| 01 | Databases Basics |
| 02 | Types of Databases |
| 03 | Best Practices |

# 01 Databases Basics

# It's Not Just About 'A' Database Anymore

- One size doesn't fit all
- Wrong choices lead to pain
  - Data inconsistencies and integrity problems.
  - Performance issues and slow response times.
  - Scalability challenges and difficulty handling growth.
  - Increased costs and operational overhead.
  - Security vulnerabilities and data breaches.

# Start with the 'What': Key Use Case Questions

These are some of the CRITICAL questions to ask before choosing a database

- **Use Case:** What is the use case? What problem are you solving?
- **Data Model**: What is the structure of your data?
- **Query Patterns**: How will you be accessing the data?
- **Consistency**: Do you need strong consistency (ACID) or is eventual acceptable?
- **Business impact:** How would the technology affect your business?

# Continue with the 'How': Technical Requirements

These are more CRITICAL questions to ask before choosing a database

- **Latency Requirements**: How quickly do you need responses?
- **Data Volume and Velocity**: How much data will you accumulate, and at what rate?
- **Scalability Needs**: What are your expected read and write loads? How much will your data grow?
- **Budget**: Do you have specific constraints related to licensing, hosting, or operational costs?

# Databases, In-Memory Databases, and Caches: What's the Difference?

- **Databases**: Provide persistent data storage with ACID properties
  - Advantages: Data durability, data consistency, support for complex queries
  - Disadvantages: Slower access compared to in-memory or cache

# Databases, In-Memory Databases, and Caches: What's the Difference?

- **Databases**: Provide persistent data storage with ACID properties
  - Advantages: Data durability, data consistency, support for complex queries
  - Disadvantages: Slower access compared to in-memory or cache

- **In-memory databases**: Store data in main memory (RAM) for extremely fast access
  - Advantages: Low latency, high throughput
  - Disadvantages: Volatility (data loss if the system crashes), higher cost, size limit

# Databases, In-Memory Databases, and Caches: What's the Difference?

- **Databases**: Provide persistent data storage with ACID properties
  - Advantages: Data durability, data consistency, support for complex queries
  - Disadvantages: Slower access compared to in-memory or cache

- **In-memory databases**: Store data in main memory (RAM) for extremely fast access
  - Advantages: Low latency, high throughput
  - Disadvantages: Volatility (data loss if the system crashes), higher cost, size limit

- **Cache**: A temporary storage area that stores copies of frequently accessed data for faster retrieval
  - Advantages: Reduces database load, improves application response times
  - Disadvantages: Can become outdated (stale data), limited storage capacity

| Feature | Database | In-Memory Database | Cache |
|---|---|---|---|
| **Primary Purpose** | Persistent storage of data | Persistent storage, optimized for speed | Temporary storage for faster data access |
| **Storage Medium** | Disk (HDD or SSD) | Primarily RAM | RAM only |
| **Data Durability** | High | Varied | Low |
| **Data Persistence** | Always | Optional, depending on the config. | Never, by design. |
| **Data Volume** | Designed for large datasets (terabytes+) | Limited by available RAM | Smaller than in-memory databases |
| **Speed** | Slower | Very fast | Fastest |
| **Complexity** | Can be complex | Can be complex, but often simpler than disk-based | Relatively simple (key-value) |
| **Use Cases** | General-purpose data storage, transactional systems | Applications requiring very low latency, real-time analytics | Speeding up access to frequently used data, reducing database load |
| **Examples** | MySQL, PostgreSQL, MongoDB, Aerospike, Cassandra | Aerospike, Redis (with persistence), MemSQL, SAP HANA | Redis, Memcached, Varnish |

# AI is rapidly changing

## Predictive AI

Predict what will happen

→ Inflexible tailor made models for a specific use case

→ Well-defined inputs: structured, consistent

→ Mission critical real time use cases - speed and reliability important

## Generative AI

Generate new artifacts

→ Produce rich, open-ended outputs from a single foundation model (LLM)

→ Retrieval-augmented generation (RAG) to ground the LLM

→ Challenge: Surface “right” set of documents

## Agentic AI

Planning and doing

→ Goal, not response-driven

→ Use tools and APIs

→ Multi-step

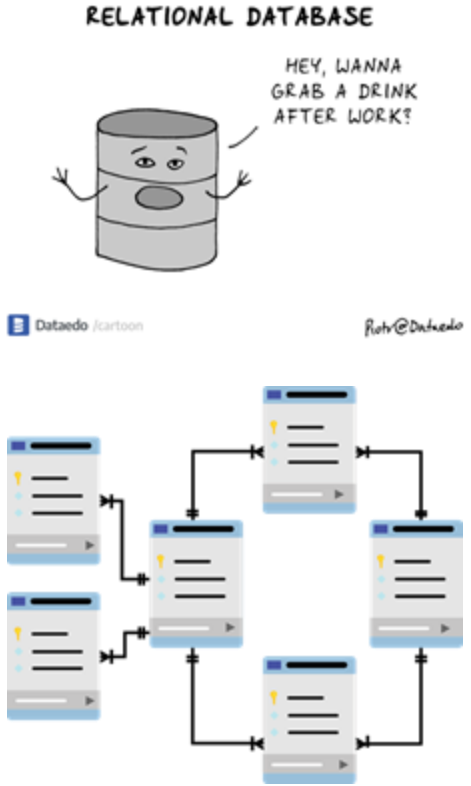
# 02 Database types

A Sea of Options

# Relational Databases: Order in the Data Chaos

- Key Features:
  - **Structured**: data organized in tables with rows and columns.
  - **Referential Integrity**: Ensures and enforce relationships between tables are maintained, preventing inconsistencies using foreign keys.
  - **Transactions**: Allow multiple operations to be grouped together, ensuring that either all operations succeed or none do - ACID properties to guarantee data integrity.
- Limitations: Can struggle with massive scale (especially writes) and highly unstructured data. Vertical scaling can be expensive.
- Common use-cases: ERP systems, CRM, Financial Systems, E-commerce Platforms
- **Examples**: MySQL, Oracle, PostgreSQL, MSSQL Server

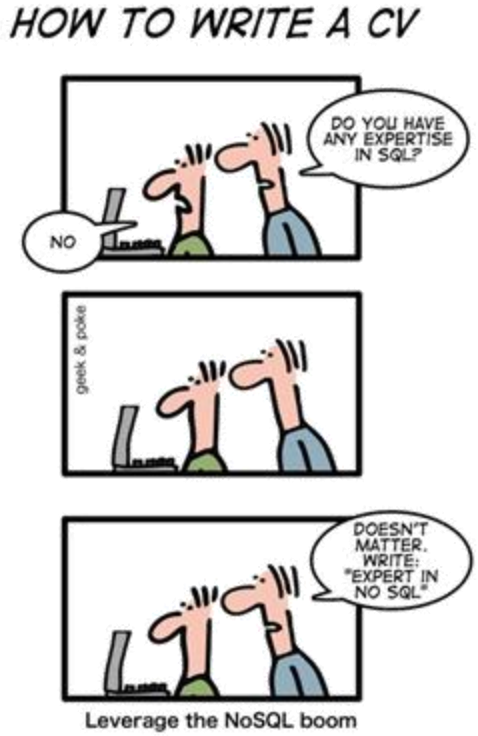


# The Modern Data Challenges

- **Exploding Data Volume & Velocity**: Traditional databases struggle to handle the sheer scale and speed of modern data.
- **Diverse Data Structures**: Applications now require storing and processing unstructured, semi-structured, and polymorphic data.
- **Agile and flexible data model**: The need to quickly adapt to changing schemas.
- **Scalability & Availability**: Businesses demand systems that can scale rapidly and remain highly available, even under heavy load.
- **Cost-Effective Solutions**: Traditional relational database scaling (especially vertical scaling) can be prohibitively expensive.

# Beyond Traditional: The NoSQL Advantage

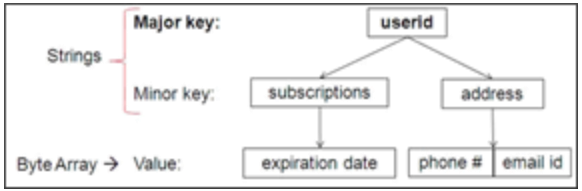
- **Purpose-Built**: Specialized solutions for specific data challenges,not one-size-fits-all.
- **Performance**: Optimized for scale, latency, and throughput.
- **Horizontal Scaling**: Efficiently distribute across commodity hardware.
- **Flexible Consistency**: CAP theorem trade-offs, often relaxing ACID guarantees.
- **Diverse Data Models**: many types - key-value, document, wide-column, and graph and more.
- **Trade-off**: Data Relationships: Joins and referential integrity are generally not supported (managed at the application level).



# NoSQL: Key-Value Stores - Speed and Simplicity

- Best For:
  - Simple data models where you retrieve data by a key.
  - High-velocity, low-latency operations (caching, session management).
  - High-volume reads.
  - Highly partitionable and offer excellent scalability.
- Limitations: Limited querying capabilities (beyond key lookup). Not suitable for complex relationships.

Example use-cases: Caching, Session Management, Shopping Cart data, User Profiles




Solutions:  Aerospike, Amazon DynamoDB, Memcached, Redis

# NoSQL: Document Databases - Flexible Schemas

- Best For:
  - Mostly access by Key however secondary indexes are available.
  - Semi-structured data (e.g., JSON, XML).
  - Applications with evolving schemas.
  - Storing user profiles, product information.
- Limitations: Can be less efficient for complex joins (compared to RDBMS). Consistency models vary.

**Example use-cases**: Content Management Systems (CMS), E-commerce Catalogs, User Profiles

Solutions:  Aerospike, Amazon DocumentDB, Couchbase, MongoDB

# NoSQL: Wide-Column Stores - Scalability and High Write Throughput

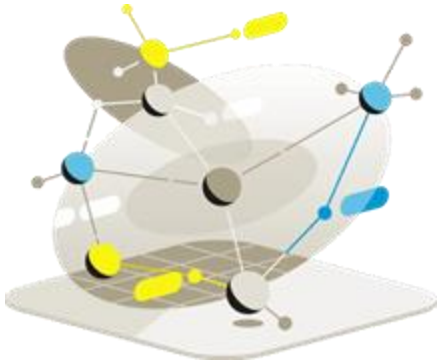
- Best For:
  - Store data in columns grouped into column families.
  - Massive datasets with high write loads.
  - Very good for accumulating records.
  - Applications requiring high availability and fault tolerance.
- Limitations: More complex data modeling. Queries are often less flexible than RDBMS.

**Example use-cases**: IoT Data, Time Series data, Messaging applications, Logging.

Solutions:  Apache Cassandra, HBase, ScyllaDB

# NoSQL: Graph Databases - Relationships Matter

- Best For:
  - Highly interconnected data (social networks, knowledge graphs, fraud detection).
  - Data design based on vertices and edges
  - Can have very complex queries
- Limitations: Can be overkill for simple data models. Less mature tooling and languages.

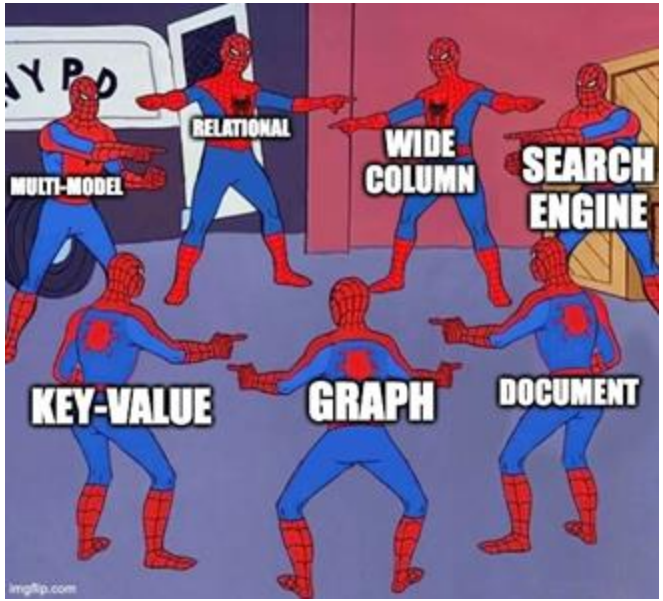


**Example use-cases**: Social Networks, Recommendation Engines, Identity Resolution, Fraud Detection, Knowledge Graphs

Solutions: ArangoDB, JanusGraph, Neo4j

# Beyond the Basics: Specialized Databases

- **Time Series Databases** (e.g., InfluxDB, Prometheus): Optimized for time-stamped data.
- **Search Engines** (e.g., Elasticsearch, Solr): Full-text search, indexing, and analytics.
- **NewSQL Databases** (e.g., CockroachDB, Yugabyte DB, Google Spanner): Aim to combine the scalability of NoSQL with relational guarantees of RDBMS.
- **Dedicated Columnar Databases** (e.g., ClickHouse, Vertica, Snowflake): high-performance analytical workloads on large datasets.



# 03 Best Practices

# No One-Size-Fits-All

Tips for Choosing the Right Database:

- Start with your use case requirements: clearly define the application requirements and the use cases.
- Analyze the data model and characteristics.
- Evaluate the scalability, performance, and cost requirements.
- Consider the security and compliance needs.
- Research and compare different database options - understand the tradeoffs between different database types.
- **There's no "one-size-fits-all" solution.**
- Don't be afraid to experiment and benchmark.

# Conclusion

- We talked about different database factors to consider: Data model, volume, throughput, latency.
- We discussed basic database concepts like ACID, BASE and the Cap Theorem.
- We touched on Relational database and contrasted with NoSQL.
- We explored the different NoSQL solutions and showcased some of the cloud-based solutions.
- We finished with some best practices.

# Final Notes:


Thoroughly analyze your requirements and select the technology that best aligns with your application's **present** and **future** needs. Start planning your database strategy today!

# Let’s make it official

Scan to get the presentations

Aerospike

# Q&A

Aerospike

# Thank You