Databases

(NoSQL / NewSQL / Columnar databases, CAP theorem)

Jana Kostičová, 12.11.2025

Traditional data processing - RDBMSs

- **Logical data structure:** Relational
- **Architecture:** Centralized
- **Scalability:** Vertical
- **Guarantees:** ACID, referential integrity, schema awareness

Vertical scalability Add more power (CPU, RAM) to an existing machine

Horizontal scalability Add more machines

- R & D > 40 years - mature technology
- Historically, networks were slow and less reliable -> databases were designed around a single powerful server to handle many requests
- Ensuring data consistency guarantees is easier with a single node

ACID (RDBMSs)

Atomicity: Either all the changes within the transaction are committed to the database, or none of them are.

Consistency: A transaction brings the database from one consistent state to another, maintaining database invariants

(foreign keys reference valid rows, data types and constraints are respected, ..)

Isolation: Concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially.

Durability: Once a transaction is committed, its changes are permanent and will survive system failures, such as power outages or crashes.

Examples: financial systems, healthcare databases

New requirements

“Big data” - **large** amounts of **heterogeneous** data

Social media data, web logs, sensor data, clickstreams, video streams (e.g. traffic monitoring), …

- A need for distributed architecture / horizontal scaling
  - Vertical scaling in centralized systems hits its limits (small gain at high cost)
- A need to store and process various data formats
  - Relational tables are not longer enough

Analytical processing

- Processing of (often big) data for analytical purposes (e.g. reporting)
- More column-based queries (aggregations)

E-commerce store example:

- *Clickstream from web GUI - see Example*
- *Can be used to provide management with insights into the most clicked products (but there are many other usages)*
- *Clickstreams for global e-commerce platforms have up to **billions of events per day***

Transactional vs analytical processing

Transactional processing (OLTP)

- Task: To process database transactions + to process transactions over “big data”
- Operations: Short, fast transactions
- Historical concepts: Traditional RDBMSs, normalized
- New concepts: NewSQL databases, NoSQL databases

E-commerce store example: managing orders, payments

“Big data” affects both processing types

Analytical processing (OLAP)

- Task: To analyze aggregated data + to analyze “big data”
- Operations: Primarily data reads, including complex, often column-based queries
- Historical concepts: Traditional RDBMSs with “dimensional” data model
- New concepts: Columnar databases, NoSQL databases

E-commerce store example: processing clickstreams / web logs for reporting

Modern data processing

- **Logical data structure:** Relational or other (key-value, document, graph, ..)
- **Architecture:** Centralized or distributed
- **Scalability:** Both vertical and horizontal
- **Guarantees:** ACID / BASE, with or without referential integrity, schema aware / schemaless

Much more options exist, different combinations for different use cases.

! In distributed environments, it is inherently more difficult to ensure data consistency guarantees, such as ACID, referential integrity, and schema-awareness.
(network delays & failures, concurrent processing, …)

See also CAP theorem

New database types

Designed for distributed environment and horizontal scalability

Relational

- Columnar DBs
  - Column-based physical storage, better for analytical queries
  - *Google BigQuery, Amazon Redshift, Apache Druid, Apache Kudu*
- NewSQL DBs
  - Mostly row-based physical storage
  - *Google Spanner, Cockroach DB, …*

Distributed relational databases designed for **analytical processing**

Distributed relational databases designed for **transactional processing**

Non-relational (NoSQL)

- Use non-relational logical data structure (other than tables)
- Can store semi-structured / schemaless data

Key-value stores, document stores, wide-column stores, graph databases

(Mostly) distributed non-relational databases designed for **different use cases**

New database types

columnar DBs
≠
wide-column DBs

Designed for distributed environment and horizontal scalability

Relational

- Columnar DBs
  - Column-based physical storage, better for analytical queries
  - *Google BigQuery, Amazon Redshift, Apache Druid, Apache Kudu*
- NewSQL DBs
  - Mostly row-based physical storage
  - *Google Spanner, Cockroach DB, …*

Distributed relational databases designed for **analytical processing**

Distributed relational databases designed for **transactional processing**

Non-relational (NoSQL)

- Use non-relational logical data structure (other than tables)
- Can store semi-structured / schemaless data

Key-value stores, document stores, wide-column stores, graph databases

(Mostly) distributed non-relational databases designed for **different use cases**

Logical structure

1. Relational tables
2. Key-value pairs (key value stores)
3. Documents accessed by keys (document stores)
4. Wide-column format (wide-column stores)
5. Graph (graph databases)

Traditional, columnar, newSQL

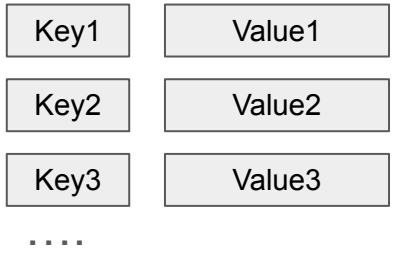
NoSQL databases
(Non-relational databases)

NoSQL databases - common properties

- Non-relational logical structure
- SQL-like or custom query language
- *Mostly* distributed architecture and horizontal scalability
- *Mostly* relaxing data consistency guarantees (ACID, referential integrity, schema awareness)
- *Mostly* intended for transactional processing
- *Often* limited complexity of queries
- *Many* are primarily designed for cloud environment
    - Some of them designed exclusively for specific cloud: Amazon DynamoDB, Google BigTable, ..

Key-value stores (1)

- Unique keys - mostly strings
  - “KA23E”, “product.KA23E”, …
- Values:
  - Various types - string, set, list, JSON object, …
  - “Opaque” to the key-value store
- Only simple queries based on the key



- Most common use cases
  - Fast key-based lookups in distributed environment
  - In-memory key-value stores (Redis, Memcached) are used as distributed caches to store precomputed or intermediate results

Ecommerce store example: Using Redis as distributed cache for storing user sessions:

SESSION:abcd1234 → { "user_id": 42, "role": "admin", "expires": 1697055900 } *AWS*

Redis, Memcached, Amazon DynamoDB, CouchBase

Key value stores (2)

Key	Value
product:KA23E	{"name": "BestBike Junior", "description": "This is the best bicycle for 4-6 years old kids"}
product:XT78S	{"name": "Lego SuperMario", "description": "Luigi's mansion", ...}
product:BG234W	{"name": "SuperColor Washing Powder", "description": "Bright colors!", ...}

```
SET product:KA23E '{"name": "BestBike Junior",...}'
```

- SETNX key value (set if not exists)
- MSET key1 value1 key2 value2 … (set multiple key-value pairs)

```
GET product:KA23E
```

- MGET key1 key2 … (get values for multiple keys)

```
EXISTS product:KA23E
DEL product:KA23E
SET session:123 "active" EX 60  // expirácia - pri cachovaní
```

Example key-based operations in Redis

Document stores

- A document is identified by a unique key
- Document format
  - JSON (BSON), XML, YAML, ..
- Less joins are needed

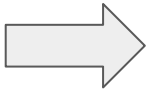
- Most common use cases:
  - Processing hierarchical data and/or semi-structured data in distributed environment

- *MongoDB, CouchDB, Couchbase, …*

```
[
    {
        "code": "KA23E",
        "name": "BestBike Junior",
        "description": "This is the best bicycle
           for 4-6 years old kids",
        "category": "Bicycles",
        "frameSize": "15",
        "wheelSize": "16",
        "recAgeLo": "4",
        "recAgeHi": "7",
        "weight": "4.5"
    },
    {
        "code": "XT78S",
        "name": "Lego SuperMario",
        "description": "Luigi's mansion",
        "category": "Lego sets",
        "piecesCount": "450",
        "recAgeLo": "7"
    },
    {
        "code": "BG234W",
        "name": "SuperColor Washing Powder",
        "description": "Bright colors!",
        "category": "Washing powders",
        "washCyclescount": "30",
        "weight": "3"
    }
]
```

JSON shredding

```
[
   {
     "firstName": "Wood",
     "lastName": "Lyons",
     "personalId": 87695,
     "addresses": [
       {
         "street": "Ralph Avenue",
         "number": 895,
         "city": "Bynum",
         "postalCode": 9981,
       },
       {
         "street": "Surf Avenue",
         "number": 386,
         "city": "Fairlee",
         "postalCode": 4470,
       },
       {
         "street": "Hull Street",
         "number": 210,
         "city": "Rockhill",
         "postalCode": 5301
       }
     ]
   }
 ]
```



Persons

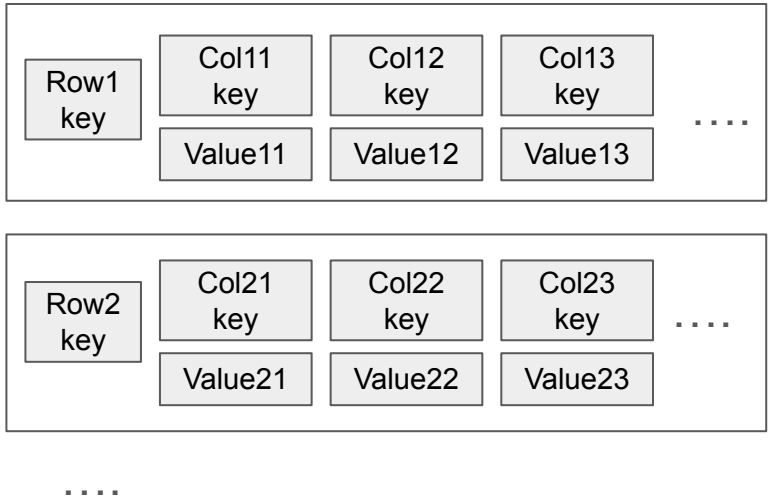
_id	firstName	lastName	personalId
1	Wood	Lyons	87695
…	…	…	…

Addresses

id	street	number	city	postal Code	person Id
1	Ralph Avenue	895	Bynum	9981	1
2	Surf Avenue	386	Fairlee	4470	1
3	Hull Street	210	Rockhill	5301	1
…	…	…	…	…	…

Wide-column stores

- Unique row keys
- Column keys:
  - Not prescribed, each row can have different columns
  - Unique for a specific row key
- Row-based queries
  - Single key / range
- Column-based queries
  - Selective column retrieval
- Use cases:
  - Semi-structured / evolving data, sparse data, time series data (timestamps in row keys)



Another abstractions:

- 2-dimensional array of key-value pairs
- Sparse matrix

Apache Cassandra, Apache HBase, Google BigTable

Graph databases

- Store efficiently (and natively) graph structures
  - Nodes, edges and their properties
- Highly scalable (Neo4j from 2020), ACID is mostly guaranteed
- Use cases:
  - Processing graph data, also in distributed environment
  - Recommendation systems (Netflix, Ebay), social network (Linkedin), ..

- *Neo4j, AllegroGraph, ..*

Centralized database vs distributed database

Centralized database

=> runs and stores data in a single machine

Traditional RDBMSs (some NoSQL databases can be used also in a single machine)

Distributed database

=> runs and stores data across multiple computers (possibly in multiple physical locations)

NoSQL databases, Columnar databases, NewSQL databases

Why to distribute data? - scalability, avoid single point of failure, availability, reliability, response time, ..

Drawbacks: increased operational complexity (network communication), increased learning curve, ...

How to distribute

Two ways to distribute a database (mostly combined):

- **Replication** - storing separate copies of data at two or more nodes
  - Leader(s) - server(s) that receives writes
  - Single leader, Multileader, Leaderless architectures
  - Single leader replication is the most common solution as writes are usually much less frequent than reads
- **Partitioning (sharding)** - data are divided into smaller parts and then stored on separate nodes, i.e., particular partitions (shards) do not share data

  In case of relational tables
  1. Horizontal fragmentation - e.g.split the rows of the table
  2. Vertical fragmentation - e.g.split the columns of the table (primary key in both tables)

Replication vs partitioning

Replication

→ fault tolerance (avoiding single point of failure)
→ load balancing
→ response time (latency)
→ availability

Partitioning (sharding)

→ (horizontal) scalability
→ efficient querying
(only relevant shards are queries, parallel querying, ..)

"distributed architecture" of a database -> both replication and automatic partitioning is built-in

Replication itself is often provided also by traditional RDBMSs.

CAP theorem

We cannot achieve all consistency, availability, and partition tolerance in asynchronous network model.

- **Consistency:** Every read operation returns the most recent write result (= strong consistency)
- **Availability:** Every request receives a response (without guaranteeing it's the most recent data).
- **Partition tolerance:** The system can continue to operate even in the presence of network partitions or communication failures.

Distributed DB must always guarantee partition tolerance. Thus it has do decide between **consistency** and **availability.**

ACID (RDBMSs)

Atomicity: Either all the changes within the transaction are committed to the database, or none of them are.

Consistency: A transaction brings the database from one consistent state to another, maintaining database invariants.

Isolation: Concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially.

Durability: Once a transaction is committed, its changes are permanent and will survive system failures, such as power outages or crashes.

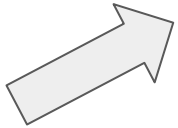
Examples: financial systems, healthcare databases

ACID vs CAP consistency

If a distributed database guarantees ACID consistency, it must also guarantee CAP consistency.

If a distributed database guarantees CAP consistency, it must compromise availability.

ACID Consistency: A transaction brings the database from one consistent state to another, maintaining database invariants.



CAP consistency: Every read operation returns the most recent write result

Compromised availability

Why “new” databases are easier to distribute

- They often compromise consistency
  - Relaxed ACID guarantees & no referential integrity
    - Less communication between network nodes, less locks
    - **BASE approach** (eventual consistency)
  - Schema-less approach
    - Easier partitioning
    - Faster writes (no need to validate data against schema)
- Related data are stored together (document stores)
  - Less joins between documents
  - Less communication between network nodes
- Support for distributed environment by design
  - Both for replication and partitioning

BASE (NoSQL DBs)

Basically Available: The system prioritize high availability, even in case of network partitions or failures.

Soft state: The system can be in a "soft" or intermediate state, which means that data consistency is not guaranteed at all times.

Eventually consistent: Data consistency is achieved over time. There is no requirement for immediate consistency, and different replicas of the data may be out of sync temporarily. However, over time, the data will become consistent through mechanisms like background reconciliation and conflict resolution.

Examples: social media platforms, distributed content delivery networks

	Logical data structure	Architecture	Referential integrity	Schema awareness	ACID vs BASE	Use case
Traditional RDBMSs	Relational (row-oriented) + support for other formats	Centralized + partial support for distributed architectures	Yes	Yes	ACID	Both transactional and analytical
NoSQL databases	Key-value, document, column-wide, graph	Often distributed	No	Weak	Mostly BASE	Mostly transactional
NewSQL databases	Relational (row-oriented)	Distributed	Yes	Yes	ACID	Transactional
Columnar databases	Relational (column-oriented)	Distributed	No or limited	Yes or no	Mostly BASE	Analytical

Distributed architecture of DB supports horizontal scaling
(both replication and partitioning are built-in)

NewSQL - prioritize consistency over availability
NoSQL, Columnar - prioritize availability over strong consistency

Traditional RDBMSs - document & distribution support

	XML	JSON	Distribution
MySQL	Limited	Yes	Limited
PostgreSQL	Limited	Yes	Limited
Microsoft SQL server	Yes	Yes	Limited
Oracle	Yes	Yes	Limited
IBM DB2	Yes	Yes	Yes
SQLite	No	Limited	No

In most of the business applications, a traditional RDBMS is still chosen – mainly because it simply meets the needs of the application.

Resources

- Robert Lukotka: Persistence and Databases
- C. M. Ricardo, Susan D. Urban: Databases Illuminated, 3rd edition, 2015.
- Pramod J. Sadalage and Martin Fowler: NoSQL Distilled, 2012
- Wikipedia: NewSQL
- S.Gilbert, N.Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant webservices
- Cassandra Documentation
- MongoDB Documentation
- Redis Documentation
- What's the Difference Between OLAP and OLTP?