Chapter 2:

NoSQL Databases
NoSQL Database Systems

Outline

• History

• Concepts
  • ACID
  • BASE
  • CAP

• Data Models
  • Key-Value
  • Document
  • Column-based
  • Graph
60s: IBM developed the Hierarchical Database Model

- Tree-like structure
- Data stored as *records* connected by *links*
- Support only one-to-one and one-to-many relationships

Mid 80’s: Rise of Relational Database Model

- Data stored in a collection of tables (rows and columns)
  → Strict relational scheme
- SQL became standard language (based on relational algebra)
  → Impedance Mismatch!
### History – Impedance Mismatch

Given the LTP scheme from Datenbanksysteme I and an object of type `Supply`:

How to incorporate the data bundled in the object `Supply` into the DB?

<table>
<thead>
<tr>
<th>LNR</th>
<th>Lname</th>
<th>Status</th>
<th>Sitz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNR</th>
<th>Pname</th>
<th>Ort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TNR</th>
<th>Tname</th>
<th>Farbe</th>
<th>Gewicht</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LNR</th>
<th>PNR</th>
<th>TNR</th>
<th>Menge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INSERT INTO L VALUES (Supply.getSupplier().getLNR(), ...);

INSERT INTO P VALUES (Supply.getProject().getPNR(), ...);

...
History – Impedance Mismatch

- Object-oriented encapsulation vs. storing data distributed among several tables
  - Lots of data type maintenance by the programmer

```
INSERT INTO LTP VALUES (...);
```
History

Mid 90’s: Trend of the Object-Relational Database Model

• Data stored as objects (including data and methods)
• Avoidance of object-relational mapping
  → Programmer-friendly
• But still Relational Databases prevailed in the 90’s

Mid 2000’s: Rise of Web 2.0

• Lots of user generated data through web applications
  → Storage systems had to become scaled up
History

Approaches to scale up storage systems

• Two opportunities to solve the rising storage system:
  • Vertical scaling
    Enlarge a single machine
    – Limited in space
    – Expensive
  • Horizontal scaling
    Use many commodity machines and form computer clusters or grids
    – Cluster maintenance
Approaches to scale up storage systems

• Two opportunities to solve the rising storage system:
  • Vertical scaling
    Enlarge a single machine
    – Limited in space
    – Expensive

  • Horizontal scaling
    Use many commodity machines and form *computer clusters or grids*
    – Cluster maintenance
History

Mid 2000’s: Birth of the NoSQL Movement

• Problem of computer clusters:
  Relational databases do not scale well horizontally

→ Big Players like Google or Amazon developed their own storage systems: NoSQL („Not-Only SQL“) databases were born

Today: Age of NoSQL

• Several different NoSQL systems available (>225)
Characteristics of NoSQL Databases

There is no unique definition but some characteristics for NoSQL Databases:

- Horizontal scalability (cluster-friendliness)
- Non-relational
- Distributed
- Schema-less
- Open-source (at least most of the systems)
ACID – The holy grail of RDBMSs:

• **Atomicity**: Transactions happen entirely or not at all. If a transaction fails (partly), the state of the database is unchanged.

• **Consistency**: Any transaction brings the database from one valid state to another and does not break one of the pre-defined rules (like constraints).

• **Isolation**: Concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially.

• **Durability**: Once a transaction has been committed, it will remain so.
BASE – An artificial concept for NoSQL databases:

- **Basically Available**: The system is generally available, but some data might not be available at any time (e.g., due to node failures).

- **Soft State**: The system’s state changes over time. Stale data may expire if not refreshed.

- **Eventual consistency**: The system is consistent from time to time, but not always. Updates are propagated through the system if there is enough time.

→ **BASE** is settled on the opposite site to **ACID** when considering a „consistency-availability spectrum“
Levels of Consistency:

- Eventual Consistency
- Monotonic Read Consistency
  - M.R.C. + R.Y.O.W.
- Immediate Consistency
- Strong Consistency
  - Transactions
- Read-Your-Own-Writes
About the concepts behind NoSQL Databases

Levels of Consistency:

• **Eventual Consistency**: Write operations are not spread across all servers/partitions immediately

• **Monotonononic Read Consistency**: A client who read an object once will never read an older version of this object

• **Read Your Own Writes**: A client who wrote an object will never read an older version of this object

• **Immediate Consistency**: Updates are propagated immediately, but not atomic
Levels of Consistency:

• **Strong consistency**: Updates are propagated immediately + support of atomic operations on single data entities (usually on master nodes)

• **Transactions**: Full support of ACID transaction model
About the concepts behind NoSQL Databases

The two types of consistency:

- **Logical consistency:**
  Data is consistent within itself (Data Integrity)

- **Replication consistency:**
  Data is consistent across multiple replicas (on multiple machines)
Brewer’s CAP Theorem:

**CONSISTENCY**

**AVAILABILITY**

**PARTITION TOLERANCE**

Any networked shared-data system can have at most two of the three desired properties!
DB-Systems allowed by CAP Theorem:

• **CP-Systems**: Fully consistent and partitioned systems renounce availability. Only consistent nodes are available.

• **AP-Systems**: Fully available and partitioned systems renounce consistency. All nodes answer to queries all the time, even if answers are inconsistent.

• **AC-Systems**: Fully available and consistent systems renounce partitioning. Only possible if the system is not distributed.
**CAP Theorem:**

- **C**: All clients always have the same view of the data.
- **A**: Each client can always read and write.
- **P**: The system works well despite physical network partitions.

All clients always have the same view of the data.

Each client can always read and write.

The system works well despite physical network partitions.
CAP Theorem:

C (Consistency)

All clients always have the same view of the data

A (Atomicity)

Each client can always read and write

P (Partition Tolerance)

The system works well despite physical network partitions

ACID

- AC-Systems
  - RDBMSs (MySQL, Postgres, ...)

UCDB

AP-Systems

CP-Systems