UNIVERSITÄT ZU LÜBECK

Information Systems

CS4130-KP06

Prof. Dr. Sylvia Melzer

SoSe2026

Data Formats & Database Architectures

Information Systems

Research Data Reality

- Heterogeneous data
- Texts
- Images
- Metadata
- Annotations
- Semi-structured data
- **Problem: One storage model does not fit all**

Palm leaves with text in Singhalese from Sri Lanka (ca. mid-20th century). Private collection.

From Representation to Storage

- Workflow: DOCX → TEI → JSON → CSV → Database → Viewer
- Research data often pass through **multiple transformation stages** before becoming part of an information system
- Each transformation **changes both the structure and the analytical possibilities** of the data
  - Example: A DOCX document may become TEI/XML through semantic markup and annotation
  - XML data can then be transformed into JSON or CSV for processing and visualization
- Databases provide **persistence, indexing, and structured retrieval** for these transformed datasets
- Viewers and interfaces finally make the **data accessible** for interaction and analysis
- Data workflows therefore connect **representation, storage, querying, and interpretation**

What Is a Database?

- A **database** is not merely a collection of files, but a **structured environment** for persistent data management
- Databases support efficient retrieval, querying, updating, and coordination of information
- They abstract away from physical storage details and provide logical access mechanisms
- Database systems enable multiple users and applications to work with shared data simultaneously
- Unlike simple file systems, databases provide indexing, transactions, and consistency control
- Databases therefore act as mediators between stored data and computational access
- Modern information systems rely on databases as core infrastructural components

What Is a Database?

A database is more than a collection of files.

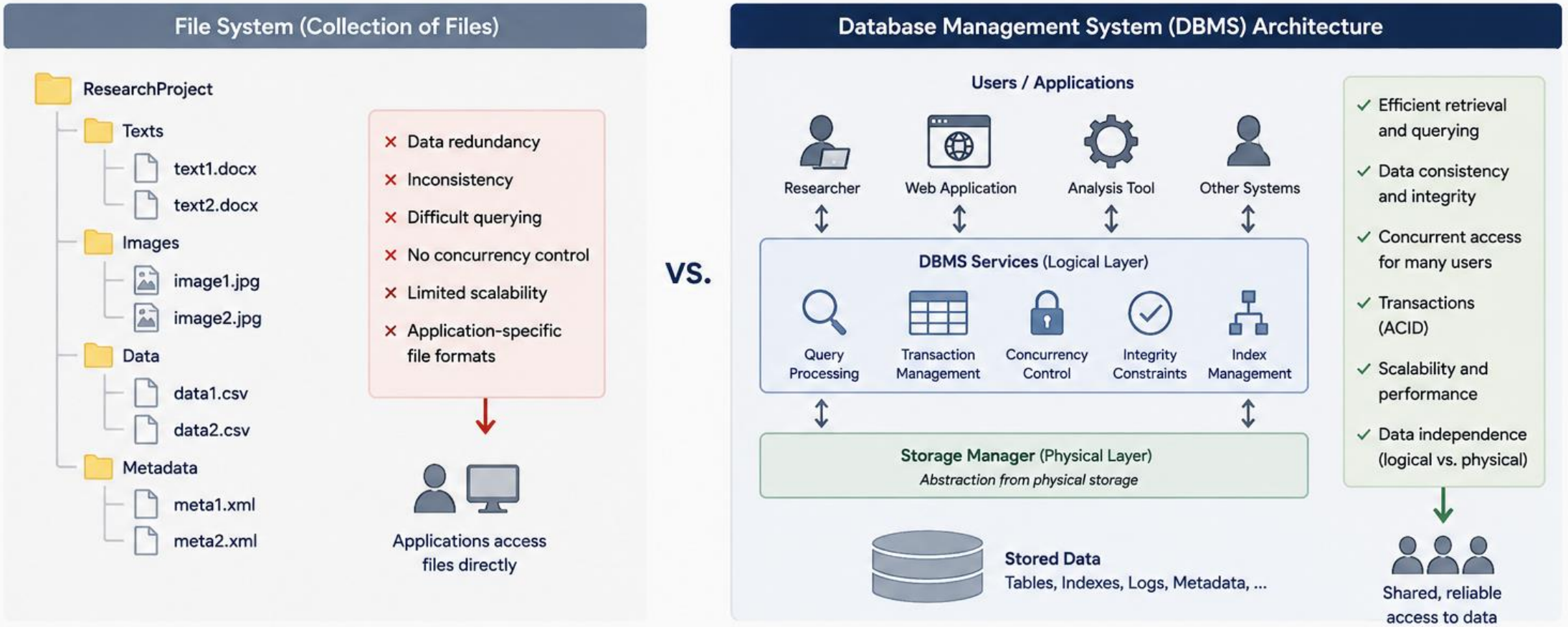


Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

Why Databases?

- File-based systems become **difficult to manage once datasets grow in size, complexity, and number of users**
- **Redundant copies** of files often lead to inconsistencies and **conflicting versions** of the same information
- Simultaneous access by **multiple users** creates **coordination and synchronization problems**
- Large collections of files are difficult to **query efficiently without indexing** and structured access mechanisms
- Simple file systems provide little support for **relationships, constraints, or data integrity**
- Databases emerged as systems for **managing complexity, consistency, and scalability**
- Modern **information systems therefore rely on databases to coordinate persistent and shared knowledge structures**

Why can’t we store everything in one database?

Problems with One Database

- Database systems evolved because previous architectures failed to efficiently handle new forms of data
- Relational DBs struggled with XML
- XML DBs struggled with scalability
- NoSQL emerged from web-scale systems
- Graph DBs emerged from semantic complexity
- A database architecture is used to solve a problem

Why are there so many database systems?

- No single database system can efficiently solve all storage and querying problems
- Different systems emerged to address different computational requirements
- Relational databases prioritize consistency and structured querying
- Document databases support flexible and evolving data structures
- Graph databases focus on highly connected information and semantic relations
- Search engines optimize large-scale text retrieval and ranking
- Database diversity reflects the diversity of data structures and research practices

Data Structure Influence

- Different types of data require different storage strategies
- Structured tables, hierarchical texts, and semantic graphs cannot efficiently be handled in the same way
- Storage systems are designed around assumptions about structure, querying, and scalability
- Research data are often heterogeneous and evolve over time
- Database architectures therefore reflect both technical and epistemic requirements
- The organization of data directly influences retrieval, analysis, and interpretation
- Choosing a storage model is also choosing a way of formalizing knowledge

Evolution of Database Systems

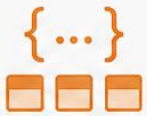
1970s — Relational Databases — **SQL**

Data organized in tables, queried with SQL.

1990s — XML Databases — **XML DB**

Hierarchical and semi-structured data stored as XML.

2000s — NoSQL Databases — **NoSQL**



Flexible schemas for rapidly evolving and large-scale data.

2010s — Graph Databases — **Graph DB**

Data modeled as nodes and edges to capture relationships.

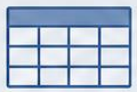
2010s+ — Search Engines — **Search Engines**

Optimized for full-text search and relevance at scale.

Diversity of needs → Diversity of database technologies

One Size Does Not Fit All

Different data → Different requirements → Different systems

Data Characteristics	Structured Tabular Data	Hierarchical Data	Semi-structured / Evolving Data	Highly Connected Data	Unstructured Text / Full Text
					
Example	Census data, bibliographic records	TEI texts, METS packages	Research data, web data, APIs	Persons, places, citations, networks	Articles, books, transcriptions
Best Suited Database Type	SQL (Relational DB)	XML DB (XML Database)	NoSQL (Document / Key-Value)	Graph DB (Graph Database)	Search Engine (Index / Inverted Index)

Key Message: Database systems are designed around different assumptions about data structures, queries, and use cases. Choosing the right system is crucial for efficiency, scalability, and meaningful analysis.

Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

Data Models → Storage Models → Query Models

Data Structure	Typical Storage	Typical Query
CSV	relational	SQL
JSON	document DB	document queries
TEI/XML	XML DB	XPath/XQuery
Linked Data	graph/triple store	SPARQL
Volltext	search engine	inverted index

The same research data can produce completely different computational possibilities depending on how they are stored.

Data Models → Storage Models → Query Models

- Data representation influences how information can be stored and retrieved
- Different structures require different persistence mechanisms and indexing strategies
- Query languages are tightly connected to underlying storage models
- Relational systems support set-oriented querying through SQL
- XML systems enable hierarchical navigation through XPath and XQuery
- Graph systems support pattern matching and semantic traversal using SPARQL
- Query models therefore reflect assumptions about how knowledge is organized

What You Will Learn

- Understand how storage architectures emerge from data structures and research requirements
- Compare relational, hierarchical, document-oriented, and graph-based systems
- Analyze the relationship between storage models and query paradigms
- Reflect on the tradeoff between flexibility, consistency, and scalability
- Explain how databases support reproducibility and interoperability
- Understand why storage systems shape research workflows and analytical possibilities
- Evaluate database architectures as components of research infrastructures

Database Management Systems (DBMS)

- A DBMS coordinates access between users, applications, and persistent data

Function	Purpose
storage	persistent data management
indexing	efficient retrieval
querying	structured access to information
transactions	consistency and reliability
access control	coordinated multi-user access
optimization	efficient execution strategies

ACID Principles

- ACID principles define **reliability guarantees** in transactional database systems
- **Atomicity** ensures that operations are either fully completed or fully rolled back
- **Consistency** guarantees that transactions preserve valid database states and integrity constraints
- **Isolation** prevents concurrent operations from interfering with each other during execution
- **Durability** ensures that committed changes remain persistent even after crashes or failures
- These principles became central for **banking systems, enterprise applications, and critical infrastructures**
- In **research contexts**, transactional consistency also supports reproducibility, traceability, and trustworthy data management

Atomicity

- A banking transfer is not a single action, but a sequence of multiple operations:

1. Withdraw money from Account A
2. Add money to Account B
3. Store the transaction
4. Update account balances

- The problem arises when the system fails during the process

Atomicity: Scenario

- Person A transfers €100 to Person B.
- The system deducts €100 from Account A, but crashes before the money is added to Account B
- Without Atomicity: the money effectively disappears
- With Atomicity: **all operations succeed or none of them are applied**
- The transaction is completely rolled back

Consistency: Scenario

- A university database stores students, courses, and enrollments
- A transaction attempts to register a student for a course that does not exist
- Without Consistency: invalid references and corrupted relationships may appear in the database
- With Consistency: the transaction is rejected because it violates integrity constraints
- A database schema defines structural and logical rules about what counts as valid data
- Consistency ensures that transactions cannot violate these rules

Consistency: Implementation

- A primary key constraint ensures that every record has a unique identifier
  - Two manuscripts cannot have the same ID
- A foreign key constraint ensures that references between tables remain valid
  - An inscription cannot reference a place that does not exist in the Places table
- Uniqueness rules prevent duplicate values where uniqueness is required
  - The same DOI should not appear multiple times
- Data validation rules ensure that values follow expected formats or ranges
  - A year field should contain a valid date and not arbitrary text
- Referential integrity guarantees that relationships between records remain consistent over time
  - If a referenced object is deleted, dependent references must be updated or prevented

Isolation: Scenario

- Two researchers edit the same manuscript record at the same time
- Researcher A changes the manuscript title
- Researcher B changes the dating information
- Without Isolation: one update may overwrite the other, partial updates may become visible, inconsistencies may occur
- With Isolation: each transaction is processed independently and securely

Isolation: Implementation

- The most classical approach is **locking**
- When a transaction accesses data: the database temporarily locks the affected rows, tables, or objects
- Other transactions must: wait, or access different data

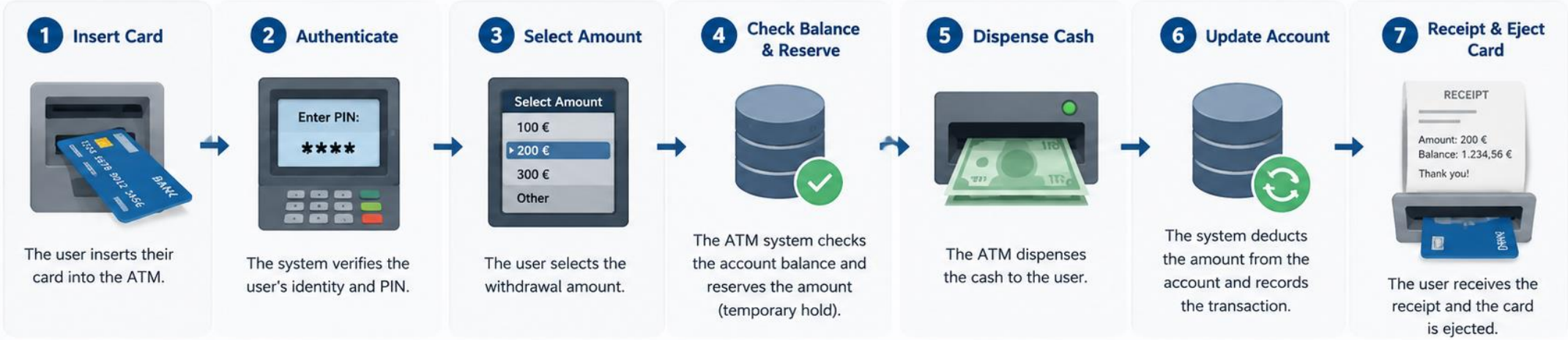
- Researcher A edits Manuscript #102
- The database places a lock on this record
- Researcher B tries to edit the same manuscript
  - the system delays the second operation until the first transaction finishes
  - This prevents conflicting simultaneous updates

Durability: Scenario

- A research infrastructure stores: metadata, annotations, and links between digital objects
- Immediately afterward: the server crashes due to a power failure
- Without Durability: newly stored information may disappear
- With Durability: committed changes survive the failure and remain recoverable
- Database systems use mechanisms such as:
  - transaction logs
  - write-ahead logging
  - checkpoints
  - recovery procedures
- These mechanisms ensure that committed data survive failures

Example: Cash Withdrawal at an ATM

A real-world transaction that illustrates ACID properties



Key Idea: Either all steps of the transaction are completed successfully, or none of them have any lasting effect.
→ This is ensured by the ACID properties.

A Atomicity

The entire transaction is treated as a single unit.

- Either all steps succeed, or none are applied. If the ATM fails after dispensing cash but before updating the account, the system rolls back the transaction.

C Consistency

The transaction brings the database from one valid state to another.

- Account balances, totals and constraints remain valid before and after the transaction.

I Isolation

Transactions are isolated from one another.

- If multiple users withdraw money at the same time, the system ensures they do not interfere with each other.

D Durability

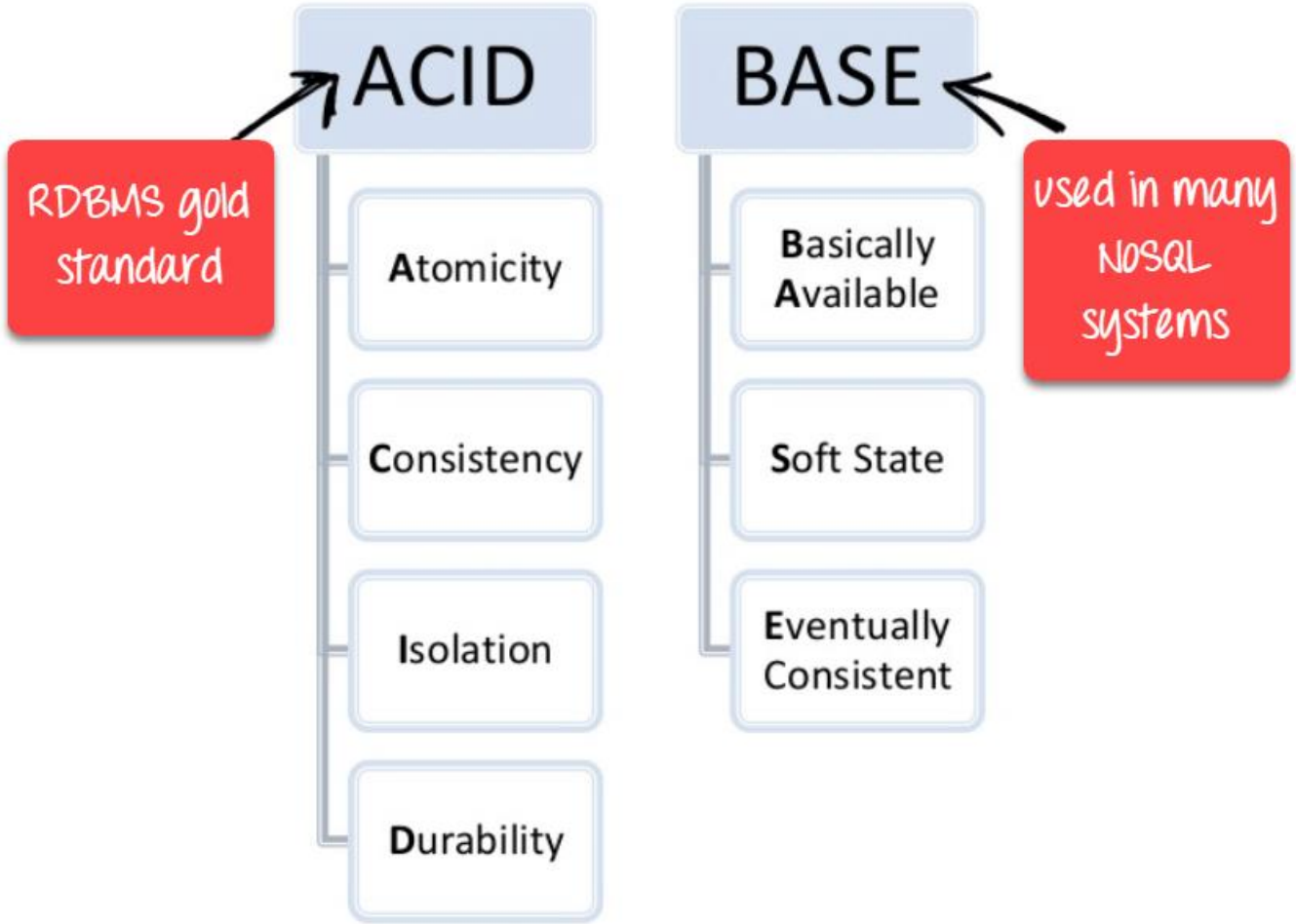
Once the transaction is committed, it remains permanent.

- Even if the system crashes after updating the account, the change is stored permanently (e.g., via logs and backups).

Result: The user receives the cash, the account is updated, and the system remains reliable and consistent.

Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

ACID vs. BASE



https://www.linkedin.com/pulse/acid-vs-base-database-properties-rudra-garnaik/

ACID vs. BASE

- BASE is a model used primarily in:
  - distributed databases
  - NoSQL systems
  - cloud infrastructures
  - large-scale web applications
- It emerged because modern distributed systems often prioritize:
  - availability
  - scalability
  - fault tolerance

  over strict transactional consistency
- Unlike traditional ACID systems, BASE accepts that temporary inconsistencies may occur to keep systems operational and responsive

ACID vs. BASE

- Should systems prioritize strict consistency or high availability and scalability?
- Traditional relational databases mainly follow the ACID model, while many distributed and NoSQL systems adopt principles closer to BASE
- The distinction emerged because large-scale distributed systems face fundamentally different challenges than centralized databases
- Traditional database research focused primarily on: correctness, reliability, consistency, and transactional integrity → This led to the ACID model
- If a systems become global, distributed, web-scale, and highly concurrent, strict ACID guarantees became increasingly difficult and expensive to maintain
- Large distributed infrastructures must continue operating even when servers fail, networks disconnect, or synchronization becomes delayed
- **This creates a fundamental tradeoff: Consistency vs. Availability**

BASE

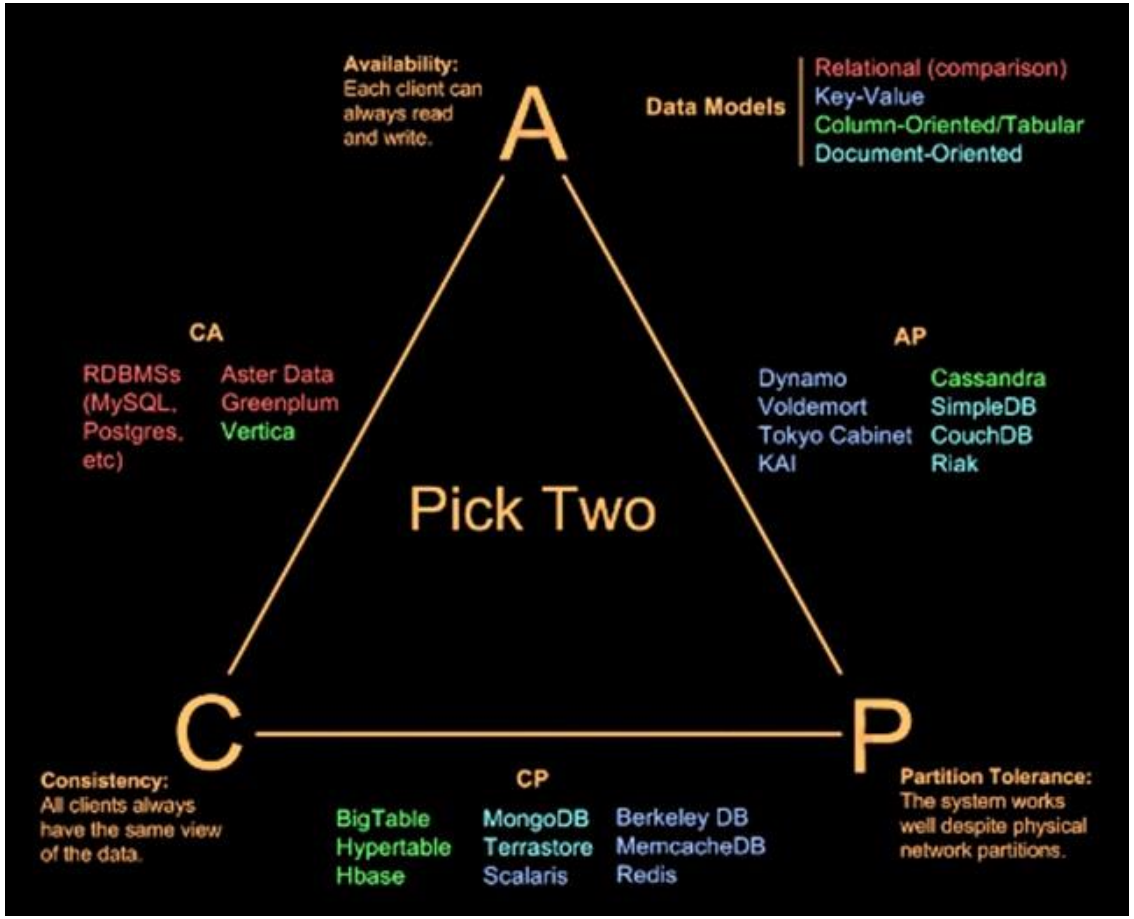
- BASE systems prioritize scalability, fault tolerance, responsiveness, and availability
- Basically Available
  - The system should continue responding, even during failures or network partitions
- Soft State
  - data states may temporarily fluctuate
  - The database does not require immediate synchronization across all nodes
  - The state is therefore “soft” because replicas may temporarily disagree
- Eventual Consistency
  - Instead of guaranteeing immediate consistency, the system will become consistent eventually
  - If no new updates occur all replicas will ultimately converge to the same state

Research Infrastructure Perspective

- Many modern research infrastructures partially adopt BASE-like principles:
  - distributed repositories
  - synchronized mirrors
  - offline-first systems
  - collaborative annotation platforms
  - CouchDB/PouchDB architectures
- These systems accept temporary inconsistency in exchange for:
  - resilience
  - scalability
  - continuous access

CAP Theorem

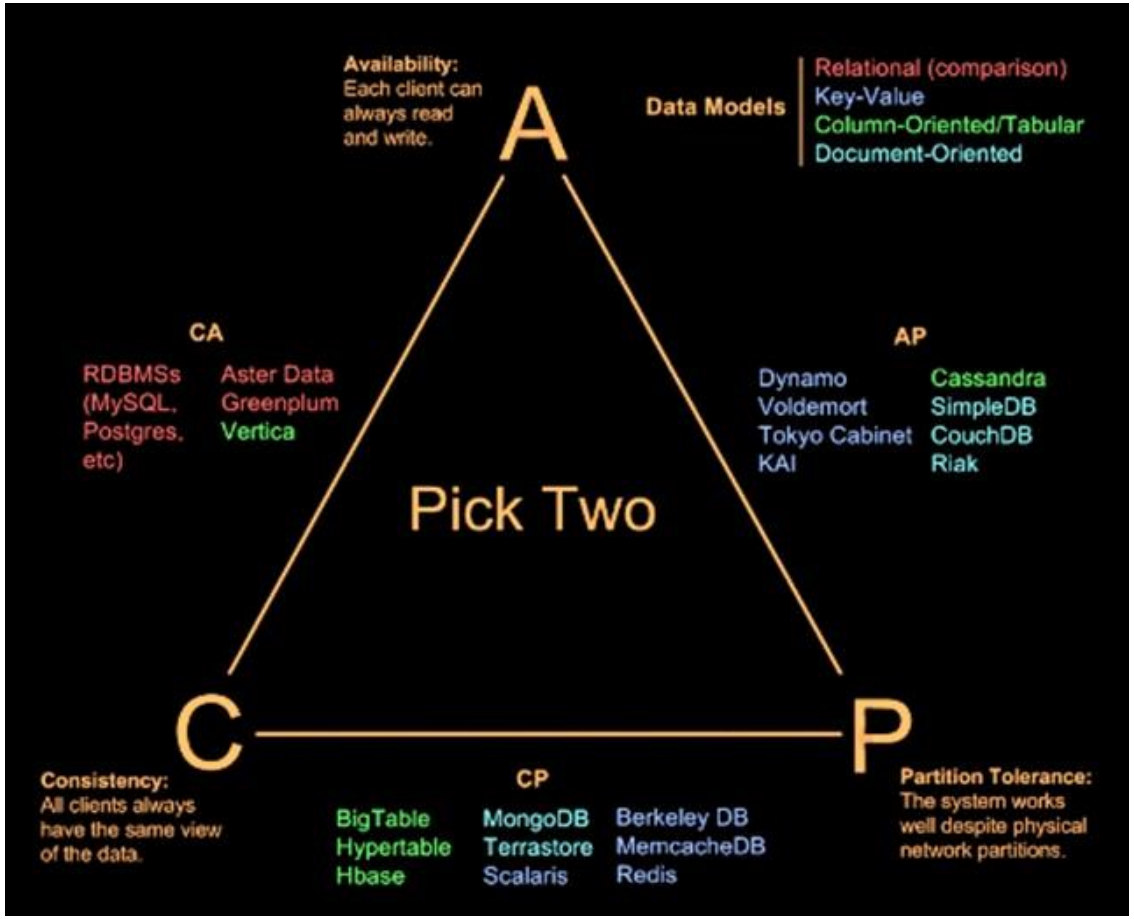
- Brewer's theorem first appeared in 1998
- **Consistency** means that all nodes return the same data at the same time
- **Availability** guarantees that systems continue responding to requests even under failure conditions
- **Partition tolerance** means that systems continue operating despite network interruptions between nodes
- **Theorem:** You can have at most two of these properties (Consistency, Availability, Partitions Tolerance) for any shared-data system



https://3cs1e07ms.wordpress.com/wp-content/uploads/2016/02/cap.jpg

CAP Theorem

- Distributed database systems must balance consistency, availability, and partition tolerance
- Modern architectures therefore prioritize different tradeoffs depending on scalability and reliability requirements
- Distributed research infrastructures often accept eventual consistency to maintain availability and resilience



https://3cs1e07ms.wordpress.com/wp-content/uploads/2016/02/cap.jpg

Schema

- A schema defines the **formal structure and expected organization** of data
- Schemas specify which entities, attributes, relationships, and constraints are considered **valid** within a system
- Database systems use **schemas to coordinate storage, validation, querying, and interoperability**
- Different schema **models reflect different assumptions** about structure, meaning, and interpretation
- Highly rigid schemas prioritize consistency and standardization, while flexible schemas support evolving and heterogeneous data
- Designing a schema therefore also means deciding **how knowledge becomes machine-readable** and computationally accessible
- Examples: XML schema, JSON schema, RDF schema

Schema: Example

The Same Data in Different Schema Models

Schema models shape how we represent, store, and interpret information.

XML Schema

Hierarchical and Nested Structure

```
<manuscript id="m1">
    <title>Kitab al-Hayawan</title>
    <author>
        <name>Al-Jahiz</name>
        <birth>776</birth>
    </author>
    <writtenIn>
        <place id="p1">Baghdad</place>
        <date when="0850"/>
    </writtenIn>
    <language>Arabic</language>
</manuscript>
```

- Data is organized in a tree structure.
- Emphasizes document structure and hierarchy.
- Order of elements can be significant.
- Well-suited for complex, nested documents like TEI.

JSON Schema

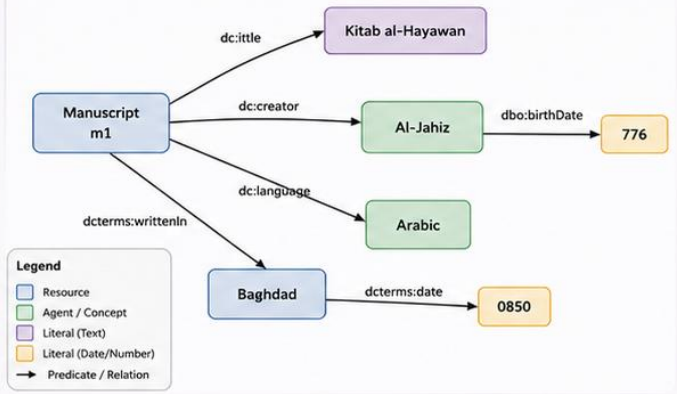
Key–Value and Nested Objects

```
{
  "manuscript": {
    "id": "m1",
    "title": "Kitab al-Hayawan",
    "author": {
      "name": "Al-Jahiz",
      "birth": 776
    },
    "writtenIn": {
      "place": {
        "id": "p1",
        "name": "Baghdad"
      },
      "date": "0850"
    },
    "language": "Arabic"
  }
}
```

- Data is organized as key–value pairs.
- Supports nested objects and arrays.
- Flexible and easy to read and write.
- Common in web APIs and NoSQL databases.

RDF / Graph Schema

Semantic Triples and Relations



- Data is represented as triples: subject – predicate – object.
- Focuses on relationships and semantic connections.
- Highly flexible and extensible.
- Basis for Linked Data and the Semantic Web.

Focus	Document structure	Data interchange	Meaning and relationships
Structure	Tree (hierarchical)	Nested key–value objects	Graph (network of triples)
Strength	Preserves complex nesting and order	Lightweight, human-readable, flexible	Semantic interoperability, extensibility
Typical Use	Digital editions, TEI, configuration files	Web APIs, NoSQL, modern applications	Knowledge graphs, Linked Open Data, ontologies

Key Takeaway: Different schema models encode different assumptions about data, relationships, and meaning.
→ **Choosing a schema is choosing a way of modeling knowledge.**

Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

Schema-on-Write vs Schema-on-Read

- Relational systems typically **validate data before storage** using predefined schemas → This approach is known as **schema-on-write** and prioritizes consistency and integrity
- **NoSQL systems** often allow data to be stored first and **interpreted later during retrieval** → This approach is called **schema-on-read** and supports flexibility and evolving structures
- **Schema-on-write systems are efficient** for stable and predictable datasets
- **Schema-on-read systems** are useful for **heterogeneous, incomplete, or rapidly changing research data**
- The distinction reflects a broader **tradeoff between strict validation and interpretive flexibility**

SQL as Query Language

- SQL (Structured Query Language) is the standard query language for relational database systems
- SQL follows a declarative paradigm in which users specify what data they want rather than how the database should retrieve it
- Relational querying is based on set theory and relational algebra rather than procedural programming logic
- SQL operations such as SELECT, JOIN, and GROUP BY enable filtering, combining, and aggregating structured data
- Query optimizers internally determine efficient execution strategies for complex operations
- SQL is highly effective for structured datasets with stable schemas and clearly defined relationships
- The language therefore reflects the relational assumption that knowledge can be represented through tables and formal relations

Relational Systems

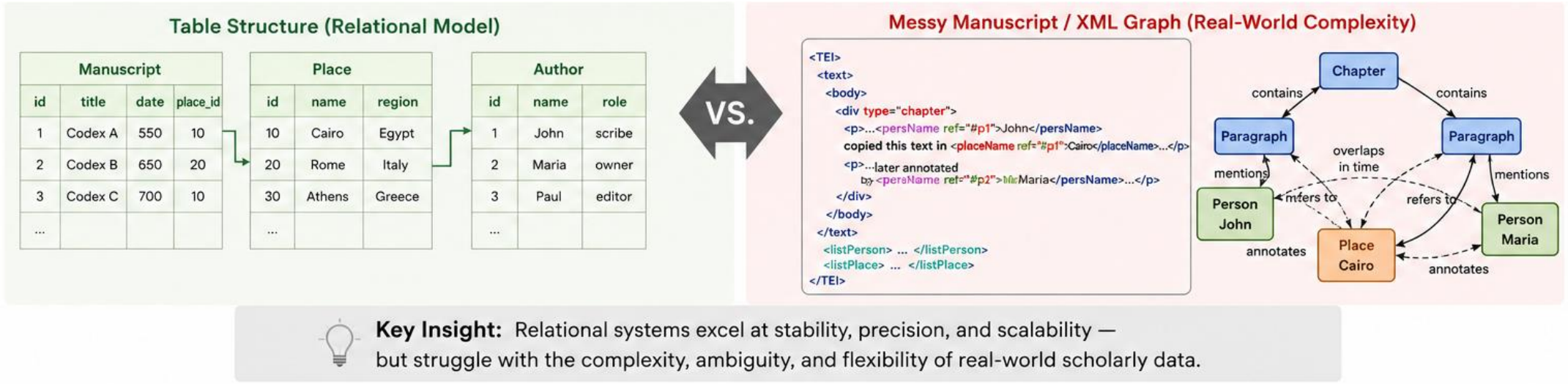
Advantages

- Relational databases provide strong consistency and reliable transaction management
- Mature optimization techniques enable efficient querying across very large structured datasets
- Integrity constraints support data quality and controlled relationships between entities
- Relational systems are highly standardized and widely supported across applications and infrastructures
- Normalized schemas reduce redundancy and improve maintainability over time

Challenges

- Relational systems assume relatively stable and clearly defined structures
- Deeply hierarchical or semi-structured data are often difficult to represent relationally
- Schema evolution can become complex in long-term and exploratory research projects
- Complex JOIN operations may reduce performance in highly connected datasets
- Relational logic often struggles with ambiguity, uncertainty, and overlapping structures common in humanities data

Relational Systems



Key Insight: Relational systems excel at stability, precision, and scalability — but struggle with the complexity, ambiguity, and flexibility of real-world scholarly data.

Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

Hierarchical Data (Recap)

- How can we preserve textual hierarchy?
- Many humanities and archival datasets are inherently hierarchical rather than tabular
- XML-based standards such as TEI, EpiDoc, and METS organize information through nested structures
- Hierarchical data models represent parent-child relationships and ordered document structures
- Textual markup often embeds semantic annotations directly within textual content
- XML therefore preserves structural and contextual relationships that may be difficult to flatten into tables
- Hierarchical organization supports rich representation of texts, metadata, and annotations
- Such models are especially important for digital editions, manuscripts, and archival collections

XML Databases

- Native XML databases are designed specifically for **storing and querying hierarchical XML structures**
- **XML databases preserve document hierarchy** and ordered node relationships directly
- Systems such as BaseX, eXist-db, and MarkLogic support **efficient storage and retrieval of XML collections**
- **XPath** enables navigation through XML trees using structural paths and node relationships
- **XQuery** extends this approach into a full query language for filtering, transforming, and combining XML data
- XML querying therefore follows **navigational tree traversal rather than relational joins**
- XML databases became especially important for digital humanities, scholarly editions, and archival infrastructures

TEI and Storage Problems

- TEI documents often contain deeply nested and highly complex textual structures
- Textual markup may overlap conceptually even when XML requires strict hierarchical ordering
- Mixed content combines text and markup within the same structural context
- Relational databases struggle to preserve nested textual semantics without decomposition
- Flattening TEI structures into tables frequently reduces contextual richness and interpretive flexibility
- Queries across overlapping annotations and embedded structures become computationally difficult
- TEI illustrates the broader tension between scholarly representation and computational storage constraints

TEI / XML (Rich Hierarchical Structure)

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <text>
    <body>
      <div type="chapter" n="1" xml:id="ch1">
        <head>Prologue</head>
        <p xml:id="p1">
          In the beginning <persName ref="#pJohn">John</persName>
          wrote to the <placeName ref="#pRome">church in Rome</placeName>.
        </p>
        <p xml:id="p2">
          He said: <q who="#pJohn">
            <hi rend="italic">Caritas</hi> omnia vincit.
          </q>
          <note type="commentary" resp="#ed1">
            A common <term ref="#t1">motto</term>
            in early Christian manuscripts.
          </note>
        </p>
        <p xml:id="p3">
          <persName ref="#pMaria">Maria</persName>
          responded in <placeName ref="#pAlex">Alexandria</placeName>.
          <add place="above" resp="#ed1">
            (This sentence is part of a later addition.)
          </add>
        </p>
      </div>
    </body>
    <back>
      <listPerson>
        <person xml:id="pJohn"><persName>John</persName></person>
        <person xml:id="pMaria"><persName>Maria</persName></person>
      </listPerson>
      <listPlace>
        <place xml:id="pRome"><placeName>Rome</placeName></place>
        <place xml:id="pAlex"><placeName>Alexandria</placeName></place>
      </listPlace>
```

Loss of Structure

Hierarchy flattened →

Overlapping annotations separated →

Mixed content (split into columns) →

Context & relationships weakened →

Editorial info & nesting lost →

Order and

Relational Model (Flattened Representation)

TABLE: paragraph

p_id	chapter	text
p1	1	In the beginning John wrote to the church in Rome.
p2	1	He said: Caritas omnia vincit. A common motto in early Christian manuscripts.
p3	1	Maria responded in Alexandria. (This sentence is part of a later addition.)

TABLE: person

person_id	name
pJohn	John
pMaria	Maria

TABLE: place

place_id	name
pRome	Rome
pAlex	Alexandria

TABLE: persName (mentions)

mention_id	p_id	person_id	role
1	p1	pJohn	writer
2	p3	pMaria	respondent

TABLE: placeName (mentions)

mention_id	p_id	place_id
1	p1	pRome
2	p3	pAlex

TABLE: quote

quote_id	p_id	who	text
1	p2	pJohn	Caritas omnia vincit.

TABLE: hi

hi_id	quote_id	rend	text
1	1	italic	Caritas

TABLE: note

note_id	p_id	type	resp	text
1	p2	commentary	ed1	A common motto in early Christian manuscripts.

TABLE: term (references)

term_id	note_id	term_ref
1	1	t1 (motto)

TABLE: add (editorial additions)

TABLE: chapter

```xml
    <p xml:id="p2">
      He said: <q who="#pJohn">
        <hi rend="italic">Caritas</hi> omnia vincit.
      </q>
      <note type="commentary" resp="#ed1">
        A common <term ref="#t1">motto</term>
        in early Christian manuscripts.
      </note>
    </p>
    <p xml:id="p3">
      <persName ref="#pMaria">Maria</persName>
      responded in <placeName ref="#pAlex">Alexandria</placeName>.
      <add place="above" resp="#ed1">
        (This sentence is part of a later addition.)
      </add>
    </p>
  </div>
 </body>
 <back>
  <listPerson>
    <person xml:id="pJohn"><persName>John</persName></person>
    <person xml:id="pMaria"><persName>Maria</persName></person>
  </listPerson>
  <listPlace>
    <place xml:id="pRome"><placeName>Rome</placeName></place>
    <place xml:id="pAlex"><placeName>Alexandria</placeName></place>
  </listPlace>
  <listTerm>
    <term xml:id="t1">motto</term>
  </listTerm>
 </back>
</text>
</TEI>
```

Overlapping annotations separated

TABLE: person

person_id	name
pJohn	John
pMaria	Maria

TABLE: place

place_id	name
pRome	Rome
pAlex	Alexandria

Mixed content (split into columns)

TABLE: persName (mentions)

mention_id	p_id	person_id	role
1	p1	pJohn	writer
2	p3	pMaria	respondent

TABLE: placeName (mentions)

mention_id	p_id	place_id
1	p1	pRome
2	p3	pAlex

Context & relationships weakened

TABLE: quote

quote_id	p_id	who	text
1	p2	pJohn	Caritas omnia vincit.

TABLE: hi

hi_id	quote_id	rend	text
1	1	italic	Caritas

Editorial info & nesting lost

TABLE: note

note_id	p_id	type	resp	text
1	p2	commentary	ed1	A common motto in early Christian manuscripts.

TABLE: term (references)

term_id	note_id	term_ref
1	1	t1 (motto)

Order and embedding simplified

TABLE: add (editorial additions)

add_id	p_id	place	resp	text
1	p3	above	ed1	(This sentence is part of a later addition.)

TABLE: chapter

chapter_id	n	head
1	1	Prologue

Key Insight: Translating TEI into relational tables requires flattening, which simplifies storage and querying but inevitably reduces contextual richness, overlapping meaning, and scholarly interpretive flexibility.

Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

Document Databases

- Store information as self-contained documents rather than normalized relational tables
- Systems such as MongoDB, CouchDB, and PouchDB are optimized for flexible and evolving data structures
- Documents may contain nested objects, arrays, and heterogeneous metadata fields
- Supports rapid schema evolution and distributed application development
- Document-oriented systems are particularly useful for research projects with uncertain or changing requirements
- Flexibility may also increase inconsistency and reduce centralized validation control
- Document databases therefore prioritize adaptability and scalability over strict relational normalization

Graph Databases & Semantic Data

- How can we represent complex semantic relationships?
- Many research datasets are fundamentally network-oriented rather than hierarchical or tabular
- Persons, places, manuscripts, citations, and concepts often form highly interconnected knowledge structures
- Relational databases represent relationships indirectly through joins and foreign keys
- Graph databases instead model relationships directly as edges between entities
- This allows efficient traversal of highly connected datasets and complex semantic networks
- Graph structures are especially useful for linked data, knowledge graphs, and semantic infrastructures
- Graph-oriented modeling therefore prioritizes connectivity and relational context over rigid tabular decomposition

Graph Data Model

- Graph databases organize information through nodes, edges, and properties
- Nodes represent entities such as persons, places, manuscripts, or institutions
- Edges represent explicit relationships between entities and may themselves contain properties
- Unlike relational systems, relationships are stored directly rather than reconstructed through joins
- Graph traversal enables efficient exploration of highly connected datasets and semantic pathways
- The graph model is particularly effective for representing networks, citations, and linked knowledge structures
- Graph-oriented querying therefore emphasizes navigation and connection patterns rather than tabular aggregation

RDF and SPARQL

```
SELECT ?manuscript
WHERE {
  ?manuscript :writtenIn :Cairo .
}
```

- RDF (Resource Description Framework) represents information through semantic triples
- Each triple consists of a subject, predicate, and object describing a formal statement
- RDF enables distributed and machine-readable representation of semantic relationships
- Unlike relational databases, RDF focuses on semantic assertions rather than predefined table structures
- SPARQL is the query language used to retrieve and match graph patterns across RDF datasets
- Queries operate through **pattern matching** and graph traversal rather than relational joins
- RDF and SPARQL therefore support interoperability and semantic integration across heterogeneous data sources

Knowledge Graphs

- Knowledge graphs integrate heterogeneous information through semantic relationships and linked entities
- They combine structured data, metadata, identifiers, and semantic assertions into interconnected networks
- Systems such as Wikidata and Europeana connect cultural heritage and research information across institutions
- Linked Open Data enables interoperability through shared vocabularies and globally resolvable identifiers
- Knowledge graphs support semantic querying, contextual enrichment, and distributed integration workflows
- They are increasingly used in digital humanities, recommendation systems, and AI-driven retrieval systems
- Knowledge graphs therefore function as semantic infrastructures for large-scale knowledge organization

Ontologies

- Ontologies define formal conceptual models for representing knowledge domains
- They specify entities, properties, relationships, and semantic constraints in machine-readable form
- Ontologies enable shared understanding across systems, institutions, and research communities
- RDF and OWL ontologies support semantic interoperability and linked data integration
- Ontological modeling separates conceptual meaning from individual database implementations
- Ontologies therefore provide semantic layers above storage architectures and data formats
- Ontology-based systems are central to semantic web technologies and ontology-based data access (OBDA)

Search Systems & Indexing

- Traditional databases are optimized for structured retrieval and transactional consistency
- Search engines are optimized for large-scale text retrieval and relevance ranking
- Full-text systems use inverted indexes rather than relational table scans
- Inverted indexes map terms to document locations and enable efficient keyword retrieval
- Search systems support ranking, stemming, fuzzy matching, and probabilistic relevance scoring
- Unlike SQL systems, search engines often prioritize relevance and retrieval speed over exact consistency
- Search infrastructures therefore enable scalable exploration of large textual collections and digital corpora

Search Systems & Indexing

Search engines are optimized for large-scale text retrieval and relevance ranking using inverted indexes.

Traditional Databases (SQL)

Documents Table

id	title	author	year
1	Hamlet	Shakespeare	1603
2	Othello	Shakespeare	1604
3	Macbeth	Shakespeare	1606
...	...	...	...

- Optimized for structured retrieval and transactional consistency
- Uses relational tables and indexes (B-tree, etc.)
- Exact matches, joins, ACID transactions
- Best for structured, well-defined data

Example Query (SQL)

```
SELECT * FROM Documents WHERE author = 'Shakespeare'
  AND year BETWEEN 1600 AND 1610;
```

Search Engines (Inverted Index)

Documents (Unstructured Text)

Doc 1: Hamlet is a tragedy written by William Shakespeare...
Doc 2: Othello is a play by Shakespeare about jealousy...
Doc 3: Macbeth is a tragedy by William Shakespeare...
...

- Optimized for large-scale text retrieval and relevance ranking
- Uses inverted inblxes (term → postings list)
- Ranking, stemming, fuzzy matching, relevance scoring
- Best for unstructured text, flexible search, and fast retrieval

Example Query (Search)

```
hamlet AND tragedy AND author:shakespeare
```

How an Inverted Index Works

Documents

Doc 1 Hamlet is a tragedy written by Shakespeare.

Doc 2 Othello is a play by Shakespeare about jealousy.

Doc 3 Macbeth is a tragedy by William Shakespeare.

Processing

Tokenize (split into terms)

Normalize (lowercase, remove stopwords, stemming)

Index Terms

Inverted Index (Term Dictionary)

Term	Posting List (docID, positions)
hamlet	(1, 1)
tragedy	(1, 4), (3, 3)
written	(1, 3)
william	(1, 6), (3, 4)
shakespeare	(1, 8), (2, 4), (3, 6)
othello	(2, 1)
jealousy	(2, 6)
macbeth	(3, 1)
...	...

Lookup Example: "tragedy"
→ Return documents 1 and 3 (with positions for ranking/highlighting)

Search Systems Support

- Relevance Ranking (BM25, TF-IDF, etc.)
- Stemming & Lemmatization (root forms of words)
- Fuzzy Matching (tolerate typos): kitteh → kitten
- Faceting & Filtering (refine results)
- Probabilistic Scoring (relevance over exactness)

Databases (SQL)

- ✓ High consistency (ACID)
- ✓ Exact results
- ✓ Better for structured data & transactions

Trade-off

Consistency ⟷ Relevance & Speed

Search Engines

- ✓ High relevance & recall
- ✓ Fast retrieval at scale
- ✓ Better for large, unstructured text collections

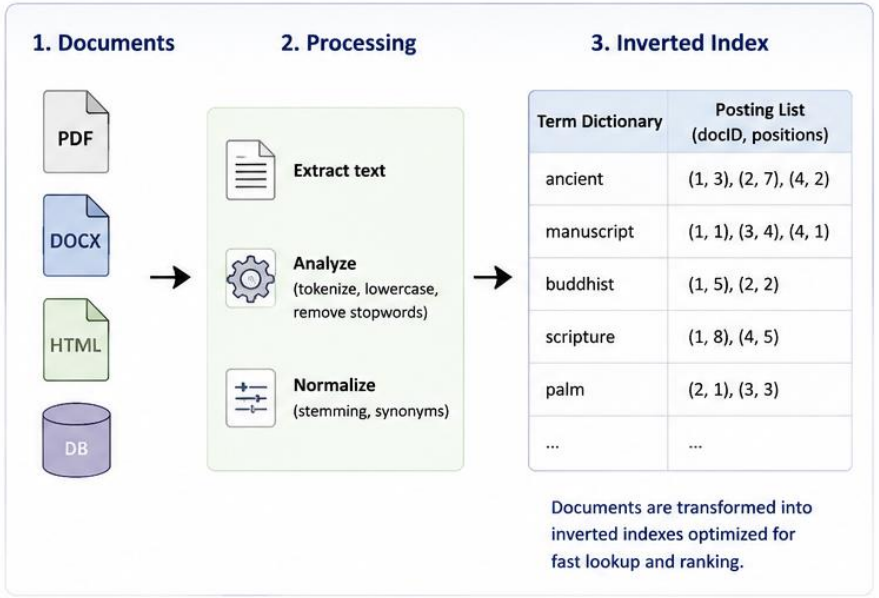
Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

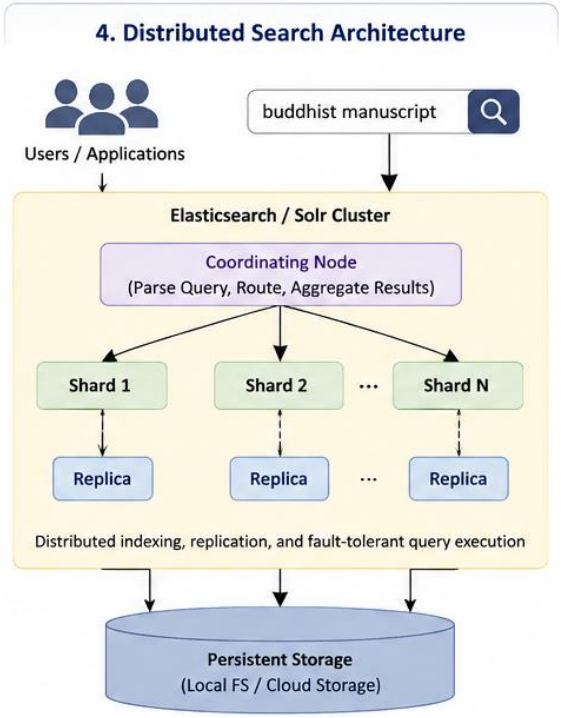
Elasticsearch/Solr

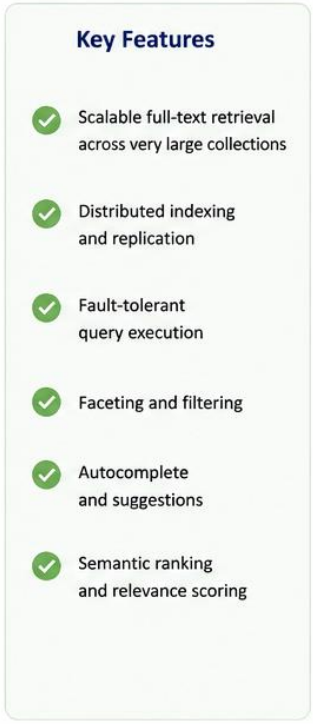
- Elasticsearch and Solr are distributed search platforms built on Apache Lucene indexing technology
- They support scalable full-text retrieval across very large document collections
- Documents are transformed into inverted indexes optimized for rapid keyword lookup and ranking
- Search platforms often provide distributed indexing, replication, and fault-tolerant query execution
- Modern repositories and digital libraries frequently combine databases with dedicated search infrastructures
- Search systems support faceting, filtering, autocomplete, and semantic ranking mechanisms
- Elasticsearch and Solr therefore act as retrieval layers above underlying storage systems

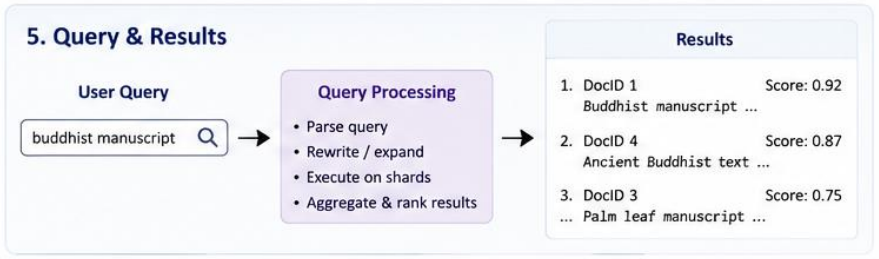
Elasticsearch / Solr

Distributed search platforms for scalable full-text retrieval









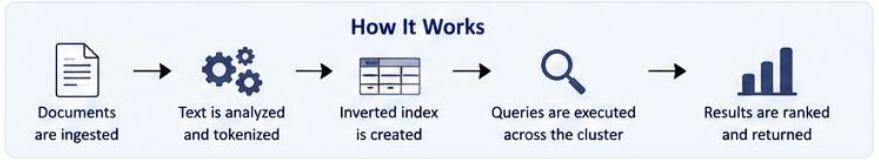


Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

Database Architectures

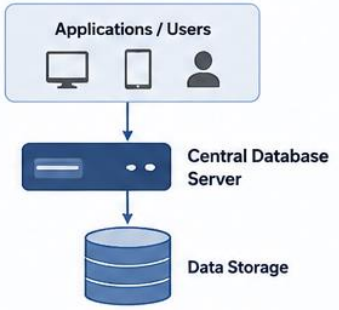
- Database architectures differ in how storage, synchronization, and access are distributed across systems
- Centralized databases store and manage data within a single coordinated infrastructure
- Distributed databases replicate and synchronize information across multiple nodes and locations
- Federated systems integrate independent databases while preserving local autonomy
- Architectural choices influence scalability, resilience, consistency, and interoperability
- Modern research infrastructures often combine centralized, distributed, and federated components simultaneously
- Database architecture therefore becomes a strategic design decision for long-term information systems

Database Architectures

Different architectures define how storage, synchronization, and access are distributed across systems.

1. CENTRALIZED DATABASE

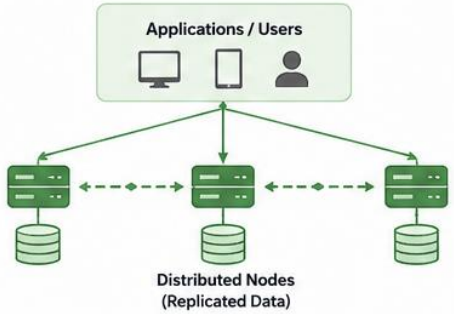
Data is stored and managed within a single coordinated infrastructure.



- Single point of control and management
- Strong consistency and ACID transactions
- Simpler administration and security
- Limited scalability beyond a single system

2. DISTRIBUTED DATABASE

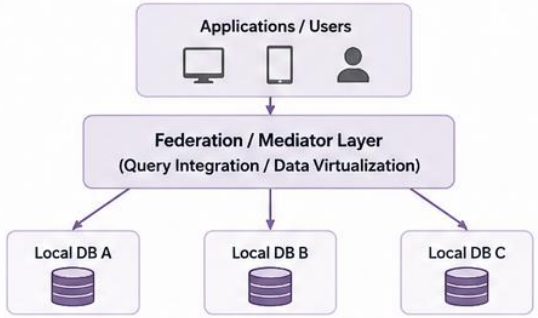
Data is replicated and synchronized across multiple nodes and locations.



- High availability and fault tolerance
- Horizontal scalability and load balancing
- Data replication across locations
- More complex synchronization management

3. FEDERATED SYSTEMS

Independent databases are integrated while preserving local autonomy.



- Integrates heterogeneous, autonomous databases
- Preserves local control and data ownership
- Enables global queries and unified access
- Challenges: schema mapping, consistency, governance

ARCHITECTURAL TRADE-OFFS

	Centralized	Distributed	Federated
Scalability	●●○○○	●●●●●	●●●○○
Resilience	●●○○○	●●●●●	●●●○○
Consistency	●●●●●	●●○○○	●●○○○
Interoperability	●●○○○	●●●○○	●●●●●
Complexity	●○○○○	●●●●○	●●●●○

COMMON USE CASES

Centralized
Banking systems, ERP, enterprise applications

Distributed
Web-scale apps, cloud services, real-time analytics

Federated
Research infrastructures, data integration, multi-institutional information systems

KEY TAKEAWAY

Database architecture shapes how data is stored, synchronized, and accessed.
The right architectural choice depends on scalability needs, consistency requirements, autonomy, and interoperability goals.

Image: Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

Storage Architectures in Research Infrastructures

- Research infrastructures combine storage, retrieval, indexing, and interoperability mechanisms across distributed systems
- Zenodo and InvenioRDM combine databases for metadata, object storage, APIs, and search infrastructures
- Wikidata represents semantic knowledge through graph-based linked data architectures
- Europeana aggregates heterogeneous cultural heritage metadata from many independent institutions
- Research infrastructures therefore require coordination between persistence, synchronization, querying, and semantic integration
- Scalability, sustainability, and interoperability become central architectural requirements
- Storage architectures consequently shape how scholarly knowledge becomes accessible, reusable, and interconnected

Choosing a Database Architecture

- Database architectures must be selected according to data structures, workflows, and research requirements
- Structured tabular datasets often benefit from relational systems and transactional consistency
- Hierarchical and document-oriented data may require XML or document database architectures
- Highly connected datasets and semantic networks are often better represented through graph technologies
- Full-text corpora require search infrastructures optimized for ranking and retrieval performance
- Long-term sustainability, interoperability, and FAIR principles also influence architectural decisions
- Choosing a database architecture therefore means balancing flexibility, scalability, consistency, and computational efficiency

Conclusion

- Different forms of data require different storage, querying, and retrieval architectures
- Database systems are not neutral containers of information, but computational models of knowledge organization
- Relational, hierarchical, document-oriented, graph-based, and search systems each prioritize different assumptions about structure, meaning, and access
- No universal database exists
- Architectural decisions influence interoperability, scalability, sustainability, and epistemic accessibility
- Representation, storage, and querying are deeply connected
- **Choosing a database architecture also means choosing how knowledge can be represented, connected, and explored!**