Database Systems

Introduction to Database Systems

Prof. Dr. Agnès Voisard   Muhammed-Ugur Karagülle

Freie Universität Berlin,
Institute of Computer Science, Databases and Information Systems Group

Fraunhofer FOKUS

2025

Books

- R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems, 7th edition*, Benjamin/Cummings, ISBN 9781292097626, 2016
- J.D. Ullman, J. Widom, and H. Garcia-Molina, *Database Systems: Pearson New International Edition*, Pearson Education Limited, ISBN 9781292024479, 2013
- A. Silberschatz, H.F. Korth, and S. Sudarshan, *Database System Concepts Fifth Edition*, McGraw-Hill, ISBN 0072958863, 2005
- A. Kemper and A. Eickler, *Datenbanksysteme: eine Einführung*, de Gruyter Oldenbourg, ISBN 9783110443752, 2015

Notes

Outline

Imagine real-world scenarios:

- University Administration:
  - "Find all available lecture halls Friday at 10 am"
  - "List all students enrolled in the Database course"
  - "Calculate the average number of courses for the students"

Why do we need specialized database systems?

Efficient and reliable management of complex data operations

Limitations of Traditional File Systems

Traditional file-based systems quickly run into problems:

- Data redundancy and inconsistency
- Difficulty enforcing data integrity constraints
- Lack of concurrent access handling
- Security risks and lack of standardized access control
- Difficulty querying complex datasets efficiently

Result: Chaos, inefficiency, errors, and high maintenance costs

Databases solve these issues by providing:

- **Structured Data Representation:** Clear schema definition and metadata
- **Efficient Data Access**: Fast querying and indexing
- **Integrity and Consistency**: Reliable transactions and fault tolerance
- **Secure Access**: Controlled permissions and data protection
- **Concurrent Usage**: Safe multi-user interaction without data corruption

Basic Definitions

Database (DB) Collection of related data organized according to a schema

Database Management System (DBMS) Collection of software programs for defining, constructing, and manipulating a database

Database System (DBS) Combination of DB and DBMS software

Why separate database from software (DBMS)?

Clear separation allows flexibility, modularity, and efficient data management

Goals and Functions of a DBMS

A DBMS supports three core functions:

- **Defining** databases (schema specification)
  - What data will be stored?
  - What types, constraints, and structures for the data?
- **Constructing databases:** entering values, physically storing data on disk
- **Manipulating databases:**
  - Querying data to retrieve specific information
  - Modifying data (inserting, updating, deleting information)

Who Interacts with Databases?

Database systems involve different types of users:

- **End users:** Query data via applications or interfaces
- **Database Administrators (DBA):** Manage, secure, and optimize databases
- **Database designers:** Design logical structure (schema)
- **System analysts:** Identify user requirements and system specifications
- **Application programmers:** Develop software using database APIs

Practical Example

University database: Students (end users), Admin staff (DB administrators), Developers (DB designers and programmers)

Database vs. Schema

Two central concepts:

- **Database Schema:**
  - Structural description of the database
  - Defines data types, structures, relationships (static)
  - Metadata stored separately (self-describing)
- **Database:**
  - Dynamic collection of data
  - Changes frequently (inserts, updates, deletes)
  - Represents real-world entities

- Database contains data **and metadata**:
  - **Metadata:** Description of data structures, schemas, constraints
  - Stored in a **System Catalog** managed by DBMS
- **Self-describing databases**:
  - DBS stores information about structure within itself
  - Allows software and users to interact without knowing physical details

Key Advantage: Modularity

DBMS software is generic, not tied to specific applications or data structures

Databases must address several key challenges:

- **Scalability:** Handle large data volumes efficiently
- **Security & Compliance:** Prohibit non-authorize access to data
- **Performance & Efficiency:** Optimize queries and processing speed
- **Reliability & Fault Tolerance:** Ensure continuous availability
- **Concurrency & Transactions:** Support multi-user access without conflicts

Why does this matter?

Every database system — SQL, NoSQL, cloud — must address these fundamental challenges

Why is database security essential?

- Prevent **unauthorized access** with authentication & encryption
- Ensure **regulatory compliance** (GDPR, HIPAA, financial standards)
- Maintain **audit logs** to track access & modifications

Without proper security...

Data breaches lead to legal and financial damage

What happens when databases grow?

- Small databases: Run on a single server
- Large-scale systems: May require **distributed architectures**
- Two common approaches:
  - **Vertical Scaling (Scale Up)**: Add more CPU/RAM to a single machine
  - **Horizontal Scaling (Scale Out)**: Distribute data across multiple servers

How do databases remain fast as they grow?

- **Indices** speed up queries
- **Query optimization** reduces processing time
- **Caching mechanisms** store frequent results

Databases must handle failures safely:

- **ACID Transactions (Atomicity, Consistency, Isolation, Durability)**
- **Backup & Recovery Strategies**
- **Replication & Redundancy** to prevent data loss

Definition

A **Data Model** defines:

- **Data Structures:** How data is organized (tables, graphs, trees)
- **Operations:** Allowed manipulations (queries, insertions, updates)
- **Constraints:** Rules ensuring integrity and correctness

Data Model is used for:

- defining the schema (Data Definition Language: DDL)
- accessing and updating the DB (Data Manipulation Language: DML)

How does a data model translate into a schema?

Database Schema (static) Defines the structure of data:

- Table definitions, data types, relationships, constraints
- Remains (relatively) stable over time

Database Instance (dynamic) Actual data:

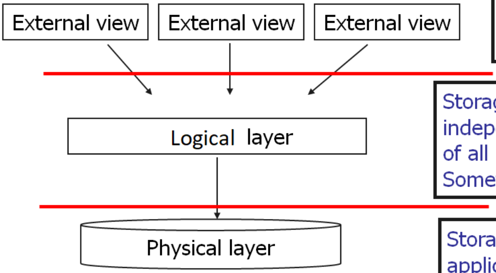
- Frequently changing, reflecting real-world operations
- Current snapshot of database state

Three-Schema Architecture (ANSI/SPARC)

Database systems introduce three distinct abstraction levels:

- **External (Logical) Level:**
  - Views tailored to specific user groups
- **Conceptual Level:**
  - Complete logical description (ER-model, tables)
  - Independent from physical implementation
- **Internal (Physical) Level:**
  - Actual storage structures (index, B-Trees, hashing)

Three-Schema Architecture (ANSI/SPARC) (cont'd)



Different applications have different views of storage and data model

Storage and data model independent definition of all data in the database. Sometimes called *logical layer*

Storage model hidden from applications

Why three levels of abstraction?

To achieve logical and physical data independence

Database systems provide levels of abstraction to:

- Shield application programs from changes in data storage
- Allow schema modifications without affecting applications

Definition: Data Independence

The capability to change the database schema at one level without affecting the schema at a higher level

Types of Data Independence

Two levels of data independence are crucial:

Physical Data Independence: Changes in the physical schema (e.g., indexing methods, storage devices) do not affect the logical structure or applications

Logical Data Independence: Changes to the **logical schema** (e.g., table structure) have minimal or no impact on existing applications

Goal: Reduce maintenance overhead and allow flexibility in DB evolution

Introduced by Edgar F. Codd (1970)

- Data structured as tables (relations)
- Separation of schema (metadata) and data
- Uses simple, powerful query languages (SQL)

Example: University database table

Course	Lecturer	Room
Database Systems	Prof. Voisard	T9-Gr.HS
Linear Algebra	Dr. Willert	T9-SR005
Datastructures and Algorithms	Prof. Mulzer	T9-Gr.HS

The dominating Relational DBMS

- Oracle Database
- Postgres (open-source!)
- MySQL (open-source!)
- Microsoft SQL Server
- IBM Db2
- personal, low cost desktop DBS: MSAccess

This list is not complete as the landscape is quite dynamic

Relational data model is still the industry standard..
.. but other data models exist, which address specific needs

Why NoSQL?

- Traditional relational databases struggle with modern scalability needs
- NoSQL databases emerged to handle large-scale, high-velocity, and semi-structured data
- Designed for flexible, distributed, and high-performance applications

Key Features of NoSQL

- Schema-less data storage
- Horizontal scalability
- More features dependent on model type

Types of NoSQL Databases

Popular NoSQL categories:

- **Key-Value Stores:** Redis, Amazon DynamoDB
- **Document Databases:** MongoDB, Couchbase
- **Column-family Stores:** Apache Cassandra, Apache HBase
- **Graph Databases:** Neo4j, Amazon Neptune

Managed database services in the cloud:

- Provides fully managed database solutions without infrastructure maintenance
- **Examples:** AWS RDS, Azure SQL Database, Google Cloud Firestore
- **Benefits:** Scalability, cost-effectiveness, automatic backups, and high availability

Realistic Example

A startup migrated its SQL database to a Cloud Database. This transition eliminated infrastructure costs and overhead while gaining automated scaling and high availability.

Processing real-time data streams:

- Used for handling **continuous, high-speed data flows**.
- **Examples:** Apache Kafka (distributed event streaming), Apache Flink (real-time analytics).
- **Benefits:** Enables **low-latency processing, event-driven architectures, and real-time decision-making**.

Realistic Example

LinkedIn uses Apache Kafka to aggregate log information from different services and systems to conduct real-time analysis of service health and performance

Decentralized data processing on the edge:

- Used for **storing and processing data closer to the source** (IoT, mobile, autonomous systems)
- **Examples:** SQLite (lightweight embedded DB), FaunaDB (distributed serverless DB)
- **Benefits: Reduces latency, works offline, and minimizes network dependence**

Realistic Example

An autonomous vehicle fleet uses an **Edge Database** to process sensor data locally This enables **real-time navigation decisions** without relying on cloud connectivity, ensuring **faster reactions to road conditions**

Operational requirement

- The DBS should never do anything that destroys the **consistency of database** and modeled reality (called **integrity**)
- **Main technical issue**: Execution of operations must **guarantee correctness properties**

Example

Transfer $ 100 from one account a1 to another one a2.

1. Reading the value x of account a1
2. Decrease the value x by $ 100
3. Write the new value of x to the account a1
4. Read the value y of account a2
5. Increase the value of y by $ 100
6. Write the new value of y to the account a2

Challenges (cont'd)

Operational requirement

- No **interference of operations** of different users

Example

Reservation system: Two independent users want to reserve the same seat on a plane

Fail-safe operation

- **System failure** should **not corrupt database** state

Example

System crash when writing new account balance on disk.
DB must not be corrupted

Technical Challenges

Efficiency

- Hundreds of clients active on the same DB
- Hundreds or thousands operations / sec
- Response time requirement in interactive environment: < 3 sec

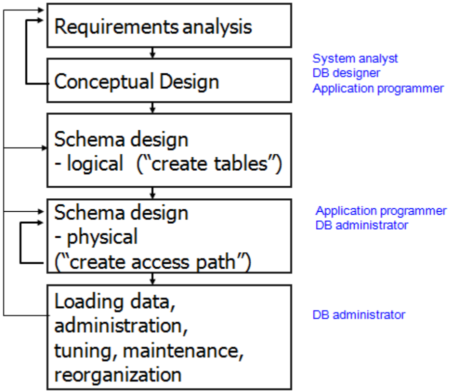
Data security

- Access by unauthorized users might be a disaster

Synchronisation of independent DB-users

- How to avoid conflicting read / write access?
  $\Rightarrow$ concurrent programming
- But DB have many resources:
  Each record is a resource – there may be millions[1] of them!
  $\Rightarrow$ Synchronization of thousands of concurrent operations?

[1] Wal-Mart: 200 mio transaction / week = 300 TA/sec (24/7)



Compare

- Life cycle of hardware is about 3 years
- Life cycle of software is about 5 years
- Life cycle of data is about 30 years

Summary

- Database $\neq$ Database System
- Database
  - **Data**
  - **Metadata** (Schema)
- Data model
- Relational Data Model (RDM) / SQL
- NoSQL Data Models
- Technical Requirements
  - Concurrency
  - Fault-tolerance
  - Integrity
  - Efficiency
- New Trends Challenges
- Lifecycle

Questions?

What will come next?

1. Welcome to Database Systems
2. Introduction to Database Systems
3. Entity Relationship Design Diagram (ERM)
4. Relational Model
5. Relational Algebra
6. Structured Query Language (SQL)
7. Relational Database Design - Functional Dependencies
8. Relational Database Design - Normalization
9. Online Analytical Processing + Embedded SQL
10. ~~Data Mining~~
11. Physical Representation - Storage and File Structure
12. Physical Representation - Indexing and Hashing
13. Transactions
14. Concurrency Control Techniques
15. Recovery Techniques
16. Query Processing and Optimization

- Business Data Processing as the driving force for DBS development
- about 1965 file system approach to data management leads to chaos
- **What are the right abstractions?** $\Rightarrow$ **Data model**
- 1970: Tables!
- 1973: Research prototypes for Relational DBS, Transactions
- 1980: RDBMS everywhere, Distributed DBS

Brief History of Databases (cont'd)

- 1990: Object orientation ⇒ OO data model and OODBMS ⇒ Object-Relational systems
- 1995: Wide scale distribution, **WEB**
- 1997: Semistructured data, Image DB, …, XML / DB
- 2000++ Mobility and DBMS
- 2005++ Unstructured Data – e.g., text. Querying text?
- Automated **Object-relational mapping**:
  Only objects in the program, relations are not the main focus