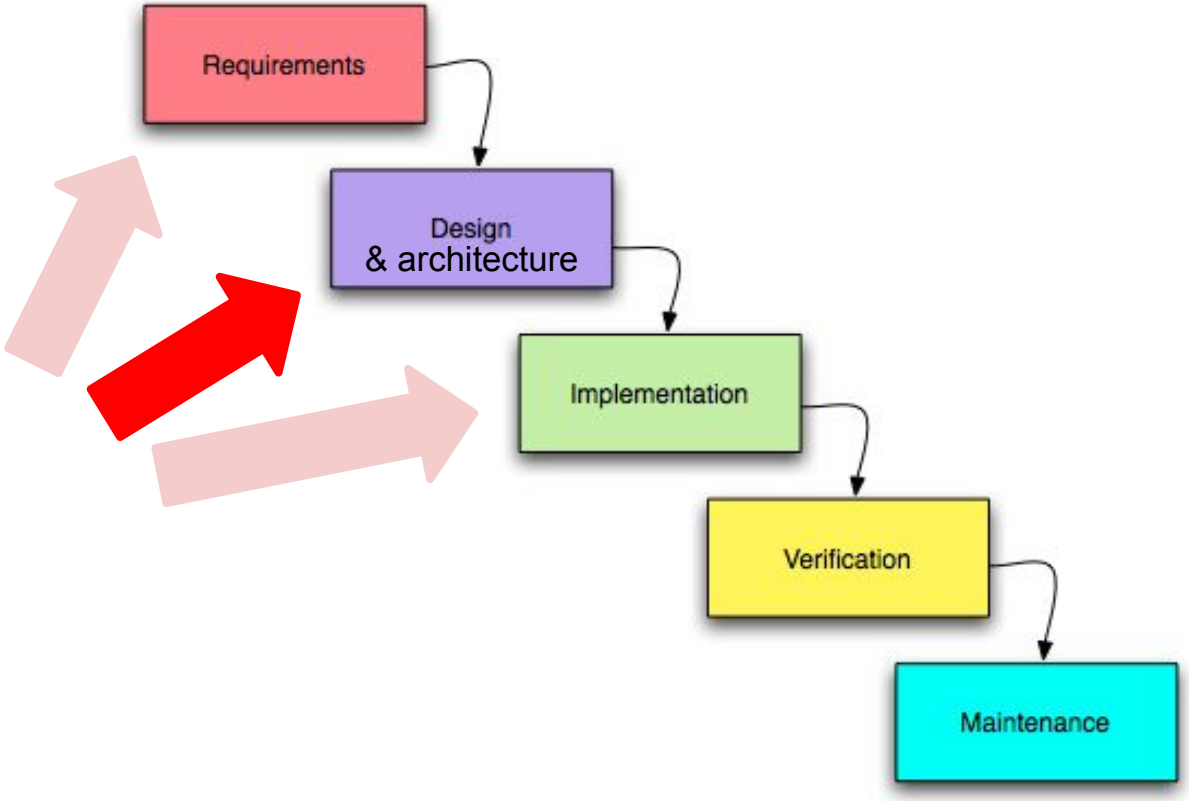
Databases

Jana Kostičová, 25.10.2023

SDLC - Waterfall

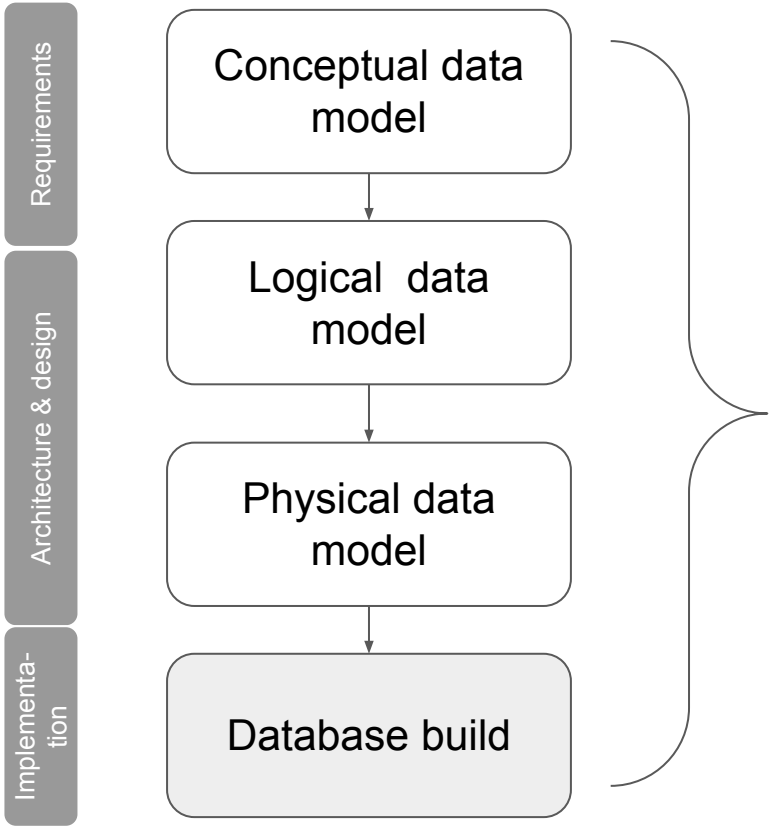


Database

= an organized collection of data, typically stored electronically, allowing easy retrieval, modification, and management of information

- Flat file (txt, csv, tsv, json, xml, spreadsheets, …)
- DBMS = **D**ata**B**ase **M**anagement **S**ystem
  - RDBMS = Relational DBMS

Database development process



Data model = visual representation of data and its relationships within a database

- Traditional approach, mostly used for relational databases
- NoSQL databases - a different approach can be used, often focus on physical design

Conceptual data model

= An abstract and high-level representation of the system that serves to identify the data that will be crucial to a business

- Describes entities and abstract relationships
  - May describe also attributes and cardinalities
- Often supplemented with a glossary
- E-R diagram, UML class diagram or free-form

- Independent from implementation details and specific data storage mechanism
- Easily understood both for technical and non-technical people

Typically created during **Requirements phase.**

Logical data model

= A more detailed view of the data, but still driven by business needs

- Describes entities, attributes, relationships (including cardinalities) and constraints
  - May describe abstract types for attributes and referential integrity (primary keys, foreign keys)
- E-R diagram, UML class diagram
- Data normalization (if it is in accordance with the requirements !)
- Independent from specific DBMS, but mostly created **for relational databases**
- Still understood by non-technical people

Typically created during **Design & Architecture phase.**

(Relational) physical data model

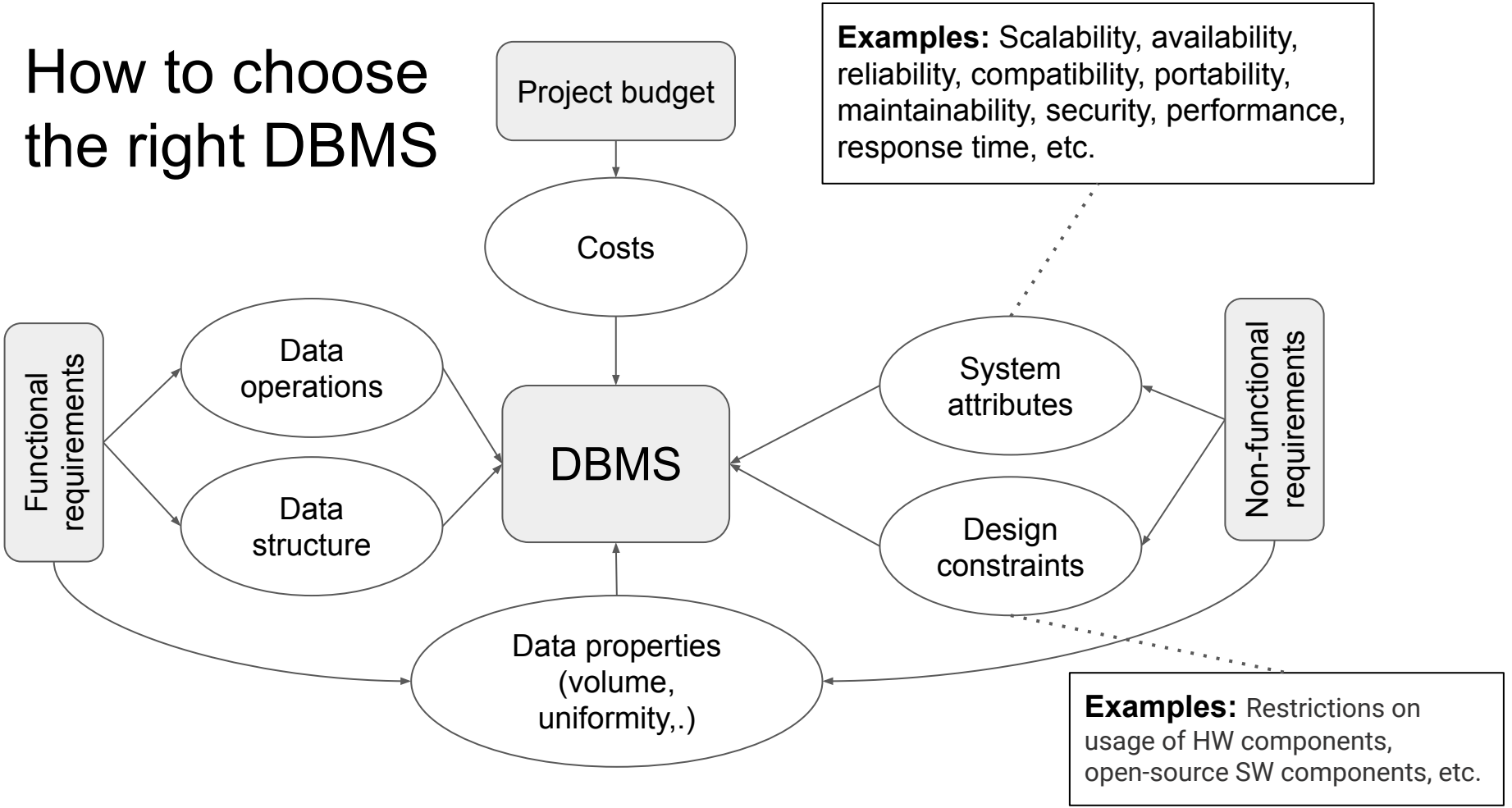
= A database-specific representation of the data
= Actual representation of the database

This step starts associating the model with a Database Management System (DBMS).

- Describes tables, columns and referential integrity
- Describes indexes, triggers, constraints, stored procedures, functions, …
- Uses DBMS specific data types
- Uses DBMS compatible table names and column names
- Validation and optimization (e.g. denormalization)

Typically created during **Design & Architecture phase.**

How to choose the right DBMS



Main aspects to consider

- Logical data structure
- Centralized vs distributed database
- On-premise vs cloud database
- Transactional vs analytical processing
- …

Logical data structure

- Relational tables
  - Stores data as rows in relational tables
- Key-value pairs
  - Stores data as values accessed by keys
- Documents
  - Stores data as documents (JSON, XML, YAML, …) accessed by keys
- Wide-column store
  - Stores data in rows in tables, but columns are not prescribed.
  - May be interpreted as 2-dimensional key-value store
- Graphs
  - Stores data as labeled vertices and edges of a (directed) graph

RDBMS

RDBMS with support for given data structure,

or

“native” database for given data structure
=> **NoSQL databases**

It is often possible to use more than one data structure for our data - also other requirements must be taken into account.

RDBMSs - examples of document support

	XML	JSON
MySQL	Limited	Yes
PostgreSQL	Limited	Yes
Microsoft SQL server	Yes	Yes
Oracle	Yes	Yes
IBM DB2	Yes	Yes
SQLite	Limited	Limited

NoSQL databases - examples

Key value store
- Redis, Amazon DynamoDB, Couchbase, …

Document store
- MongoDB, CouchDB, Couchbase, …

Wide-column store
- Apache Cassandra, Apache HBase, …

Graph database
- Neo4j, …

NoSQL is not really a suitable term, rather we should talk about non-relational databases

Centralized database vs distributed database

Centralized database

=> runs and stores data in a single machine

Distributed database

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

Why to distribute data? - scalability, availability, reliability, response time,, …

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

Centralized databases - RDBMSs vs NoSQL

- Traditional RDBMSs
  - Work well in a centralized environment
  - R&D > 40 years
- NoSQL DBs
  - (Most of them) primarily designed for distributed environment, but they can be used also in a single machine
  - Other properties of NoSQL DBs must be taken into account

Distributed databases - How to distribute

There are two very distinct ways to distribute a database (those approaches can be combined):

- **Replication** - storing separate copies at two or more nodes
  - Single leader - one server receives writes.
  - Multileader, Leaderless
- **Fragmentation (partitioning)** - we divide data into smaller parts and then store them on separate nodes
  - Horizontal fragmentation - e.g.split the rows of the table
  - Vertical fragmentation - e.g.split the columns of the table (primary key in both tables)

Single leader replication is the most common solution as writes are usually much less frequent than reads.

Distributed databases - RDBMSs vs NoSQL

- Traditional RDBMSs
  - Difficult to distribute across multiple machines (= to scale horizontally)
- “NoSQL” DBs
  - Work well in distributed environment, they easily scale horizontally
- Distributed SQL DBs
  - Represent a new approach, these DBs are relational but at the same time easy to scale horizontally
  - E.g., Google’s Spanner, CockroachDB

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

Horizontal scalability
Add more machines

Why NoSQL is better at horizontal scaling

- Relaxed ACID guarantees & no integrity constraints
  - Less communication between network nodes, less locks
  - BASE approach (eventual consistency)
- Flexible schema or schema-less approach
  - Easier horizontal fragmentation
  - Faster writes (no need to validate data against schema)
  - Easier to accommodate new data types and changes
- Related data are stored together (document stores)
  - Less joins between documents
  - Less communication between network nodes in case of horizontally fragmented data
- Support for distributed environment by design
  - Both for replication and fragmentation

Kind of “cheating” - the better scaling is achieved by relaxing some of the mechanisms that RDBMSs strictly follows. Drawbacks of this relaxation must be carefully considered.

Not all NoSQL databases provide the same level of support for horizontal scaling!

ACID (RDBMSs)

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

Consistency: Guarantees that a database 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: Ensures that 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

BASE (NoSQL DBs)

Basically Available: The system prioritize high availability, even in the face 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

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.
- **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.**

NoSQL DBs - compromise consistency in favor of availability

On-premise vs cloud database

On-premise DB

- Full control over DB hardware, software, and infrastructure - easier customization of the environment
- Higher level of control over data security
- Some data must be stored on-premises to maintain compliance (e.g., legal)

Cloud DB

- Easier scalability and geographical redundancy
- Lower maintenance costs
- Managed - Database as a service (DaaS, also data as a service)
  - Typically more expensive
  - Suitable if the organization does not have its own staff to develop and maintain the database
- Self-managed database

Transactional vs analytical processing

OLTP: Online transactional processing

- Task: To process database transactions
- Data structure: Relational tables or NoSQL data structures
- Operations: Primarily data writes
- Real-time processing

OLAP: Online analytical processing

- Task: To analyze aggregated data
- Data structure: Relational tables, but multidimensional model is used (star schema, snowflake schema)
- Operations: Primarily data reads, including complex queries
- Batch processing + nowadays also real-time (or close to real-time) processing

Resources

- Robert Lukotka: Persistence and Databases
- S.Gilbert, N.Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant webservices
- What's the Difference Between OLAP and OLTP?