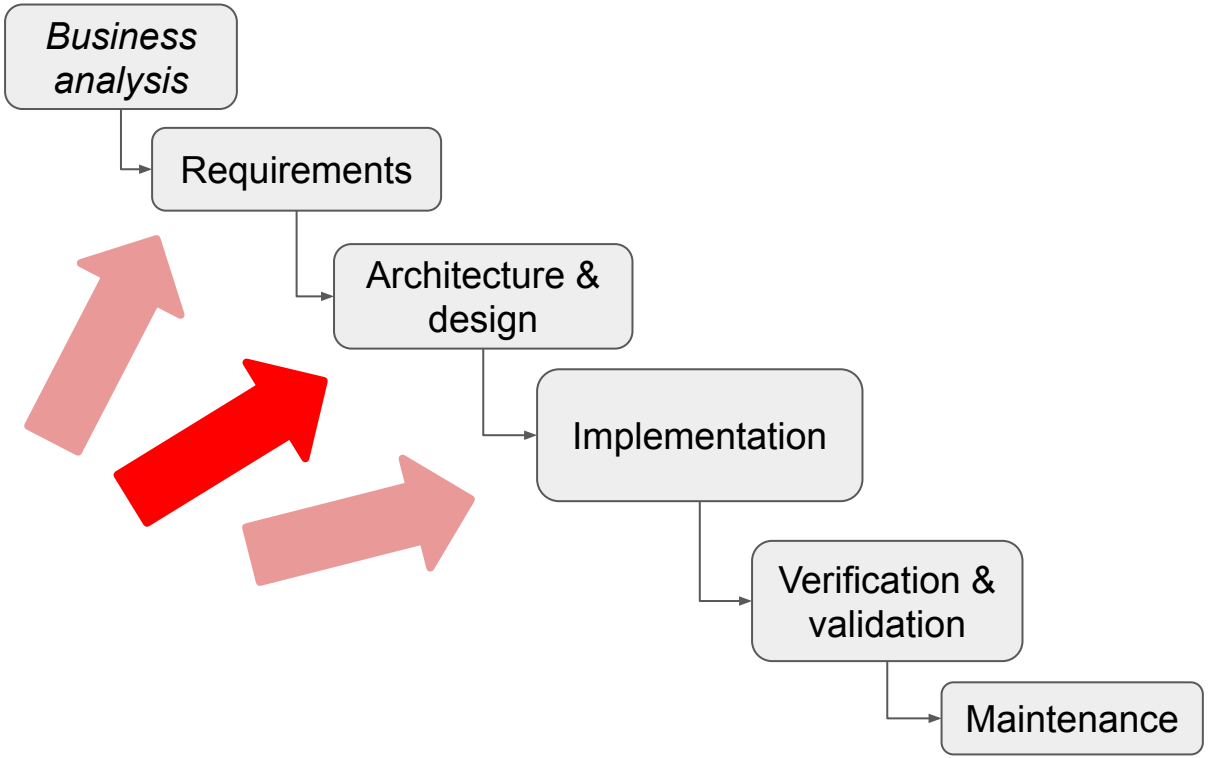
Databases

Jana Kostičová, 23.10.2024

SDLC - Waterfall



Database vs DBMS

Database

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

- Examples: flat or hierarchical file formats (txt, csv, tsv, json, xml, spreadsheets, …), relational tables in RDBMS

DBMS (**D**ata**B**ase **M**anagement **S**ystem)

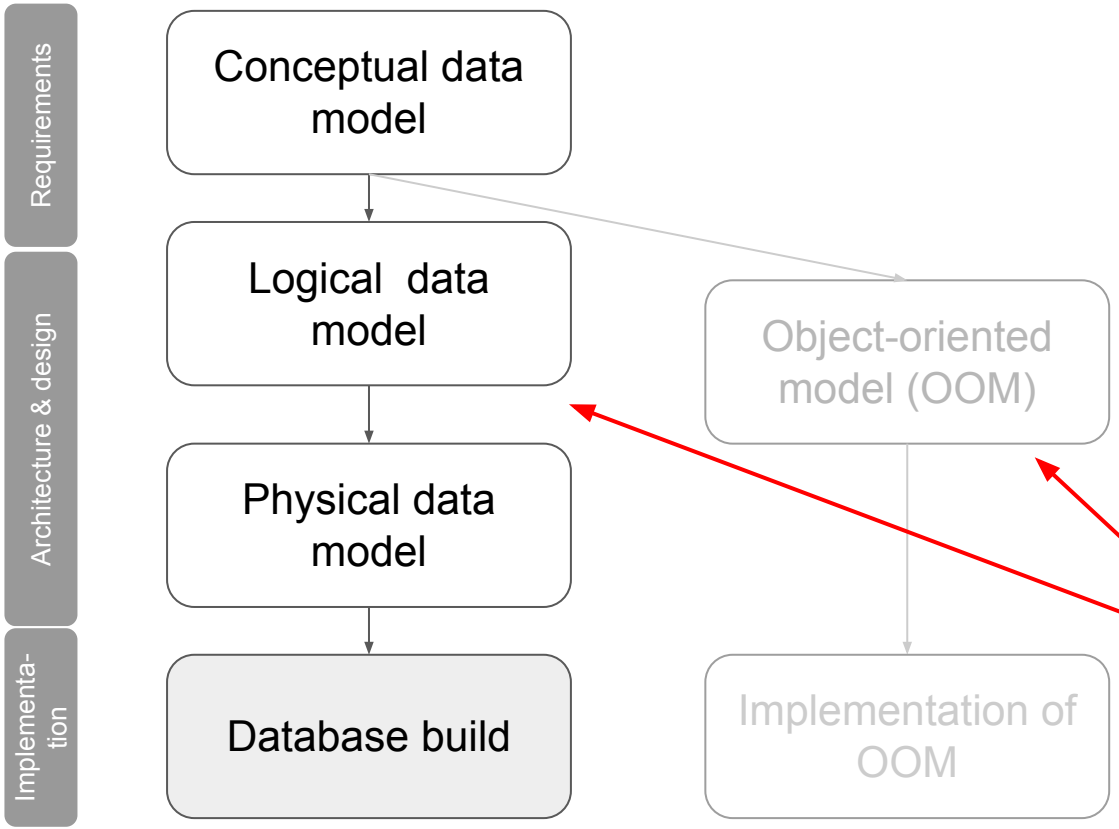
= a software that manages and interacts with the data stored in a database

- Examples: RDBMS = Relational DBMS, NoSQL databases

New “Big data” storage systems:

- NoSQL databases, Columnar databases, NewSQL databases -> DBMSs
- Distributed file systems, Cloud object storage -> databases, but no DBMSs

Database development process



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

Conceptual model is used to derive both more detailed data models as well as O-O models.

There is typically ORM (Object-relational mapping) between O-O model and relational data model.

Conceptual data model

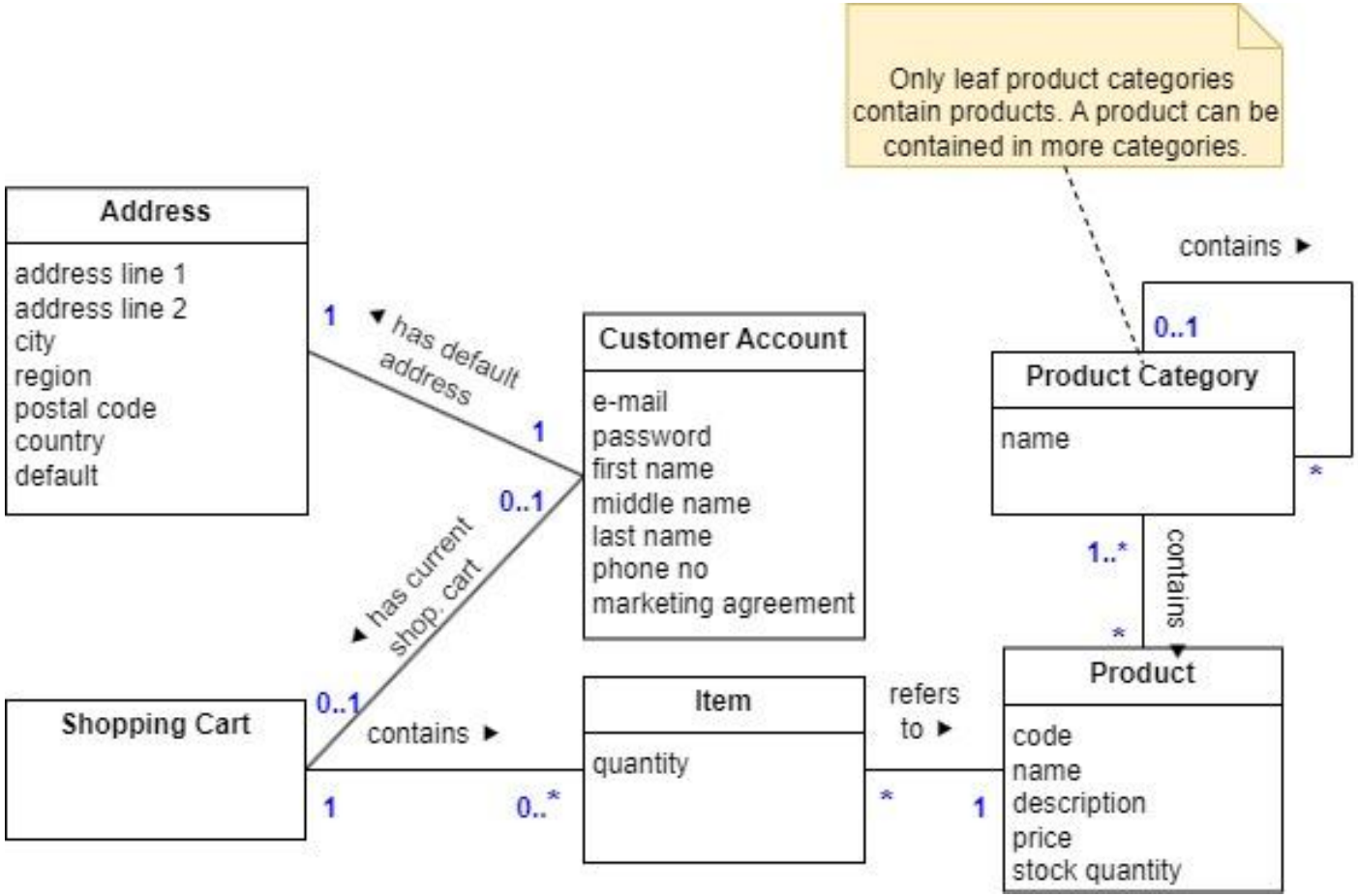
= An abstract and high-level representation of the system that serves to identify the data that will be crucial to a business

- Describes entities and abstract relationships
  - May describe also attributes and cardinalities
- Often derived from domain model
- 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.**

Example



Relationships are explained by textual descriptions.

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 !)

- Dependent on logical data structure
- Still independent from specific DBMS
- Still understood by non-technical people

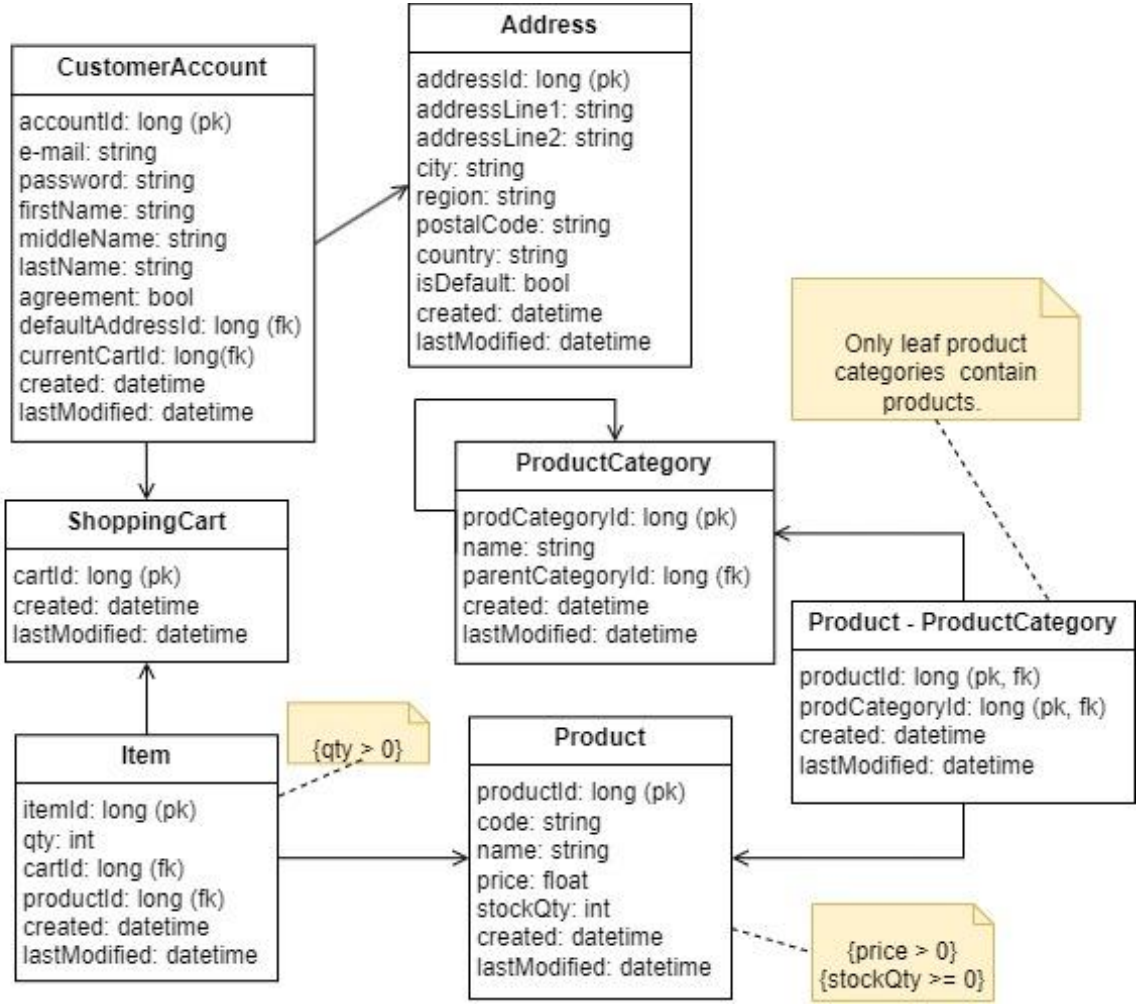
Typically created during **Design & Architecture phase.**

Example

Logical model for relational tables

This model contains explicit foreign keys so it is assumed that non-technical people understand this notation.

If not, it is better to keep the relationship descriptions and cardinalities used in the conceptual model.



Physical data model

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

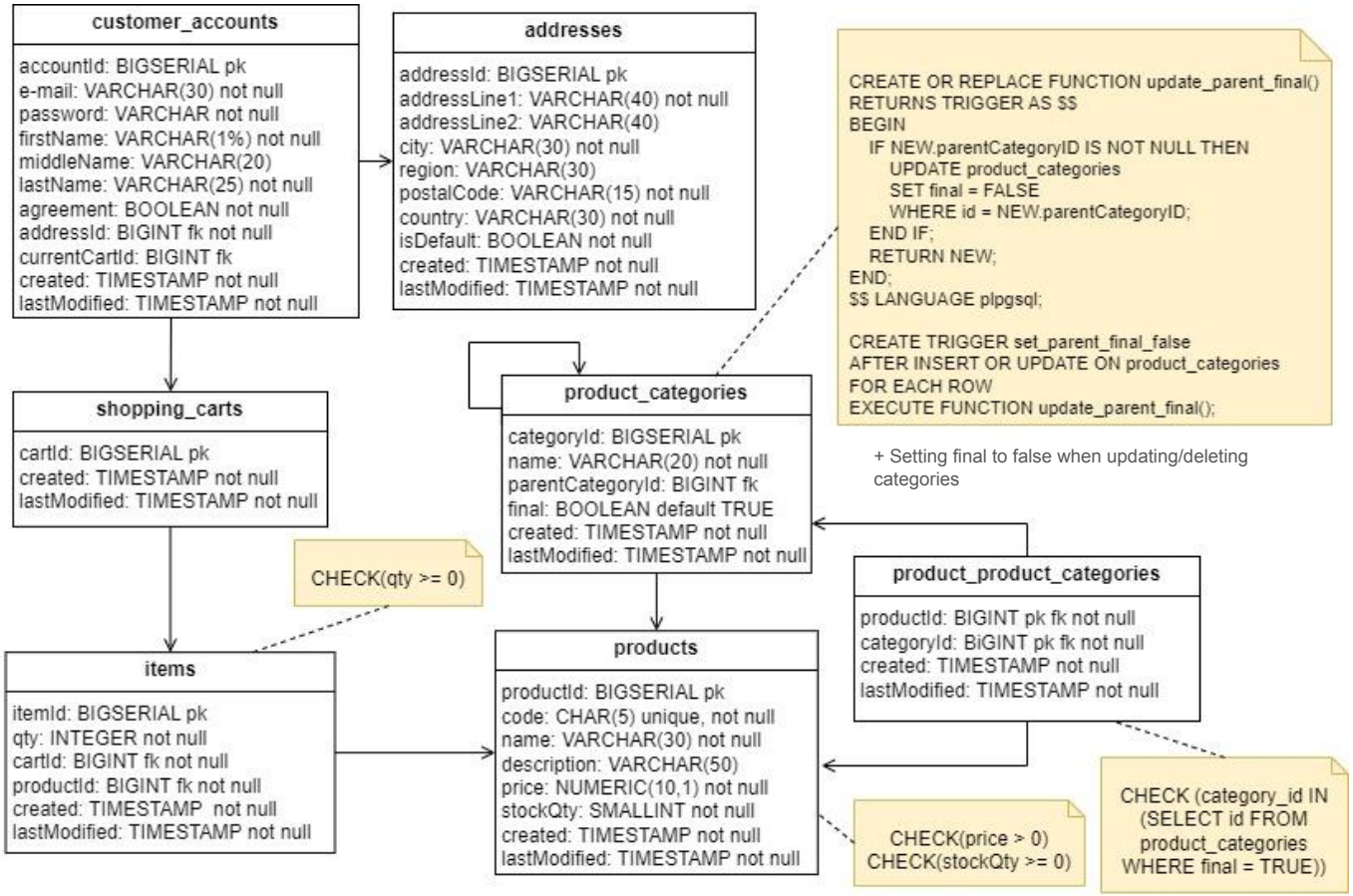
This step starts with associating the model with a Database Management System (DBMS).

- 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.**

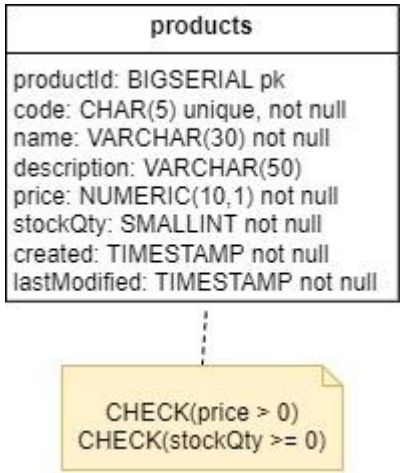
Example

Physical model for PostgreSQL



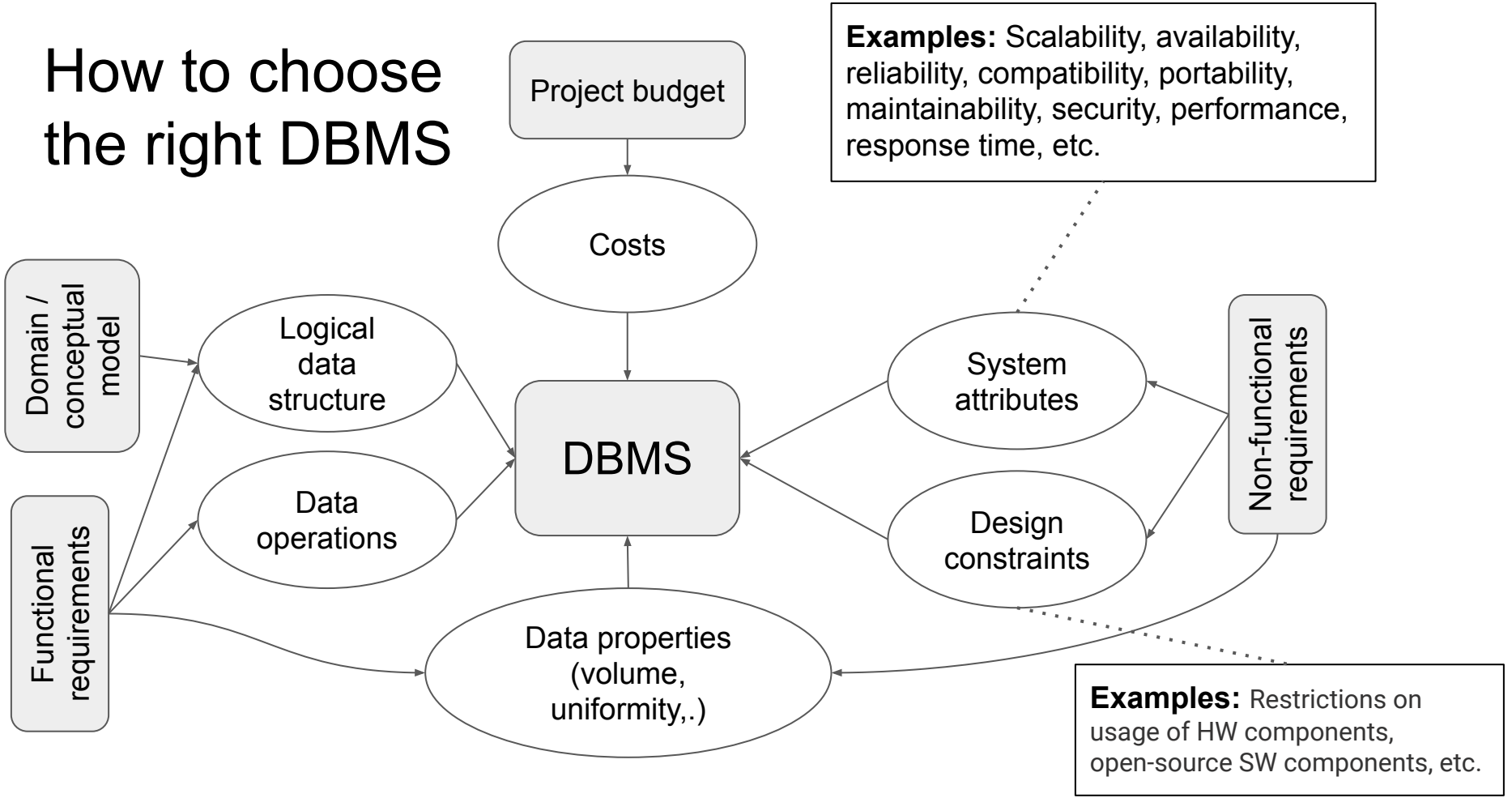
Example

Physical model for PostgreSQL




```
CREATE TABLE products (
  productId BIGSERIAL,
  code CHAR(5) UNIQUE,
  name VARCHAR(15) NOT NULL,
  description VARCHAR(50),
  price  NUMERIC(10,1) NOT NULL,
  stockQty SMALLINT NOT NULL,
  created TIMESTAMP NOT NULL DEFAULT NOW(),
  lastModified TIMESTAMP NOT NULL DEFAULT NOW(),
  PRIMARY KEY(productId),
  CHECK (price > 0),
  CHECK (stockQty >= 0)
);
```

How to choose the right DBMS



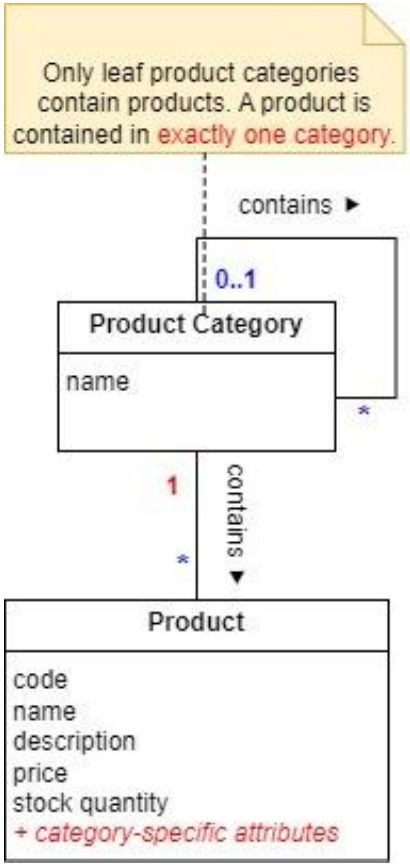
Main aspects to consider

- Logical data structure
  - Relational, key-value, hierarchical documents, graph, …
- Centralized vs distributed database
- On-premise vs cloud database
- Transactional vs analytical processing
- …

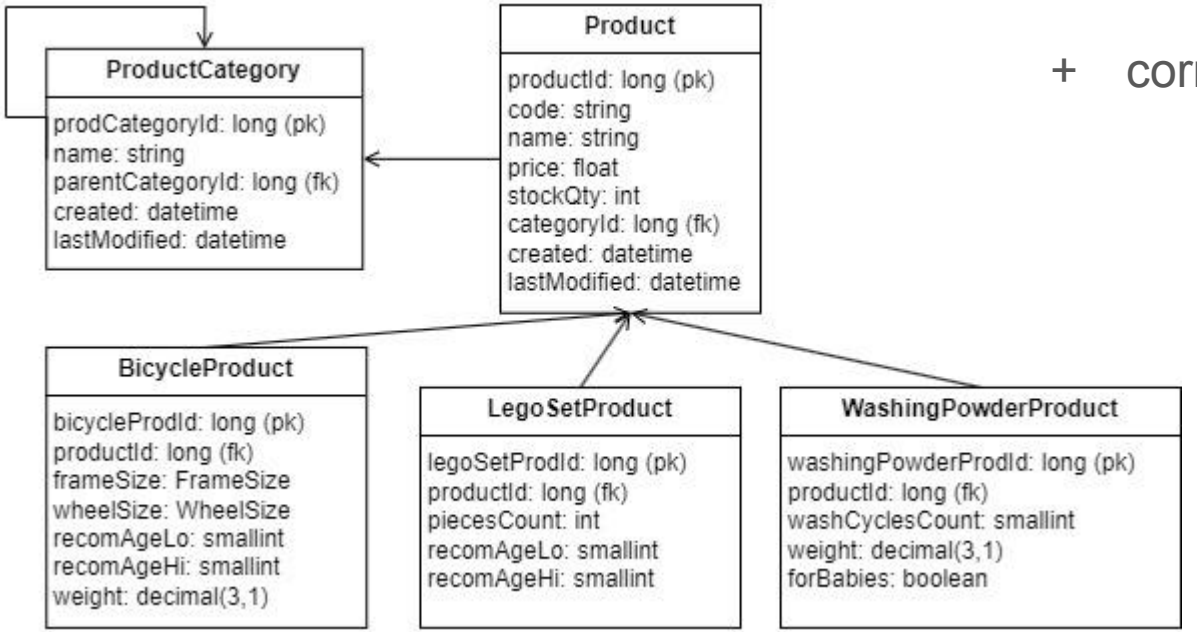
Back to example

Category / attribute	Frame size (enumeration)	Wheel size (enumeration)	Recommended age (range of positive numbers)	Weight (decimal number)	Number of pieces (positive number)	Number of washing cycles (positive number)	For babies (yes/no)
Bikes	X	X	X	X			
Lego sets			X		X		
Washing powders				X		X	X

- Sparse matrix
- Naive solution: all attributes in “Product” table



Relational logical data model 1

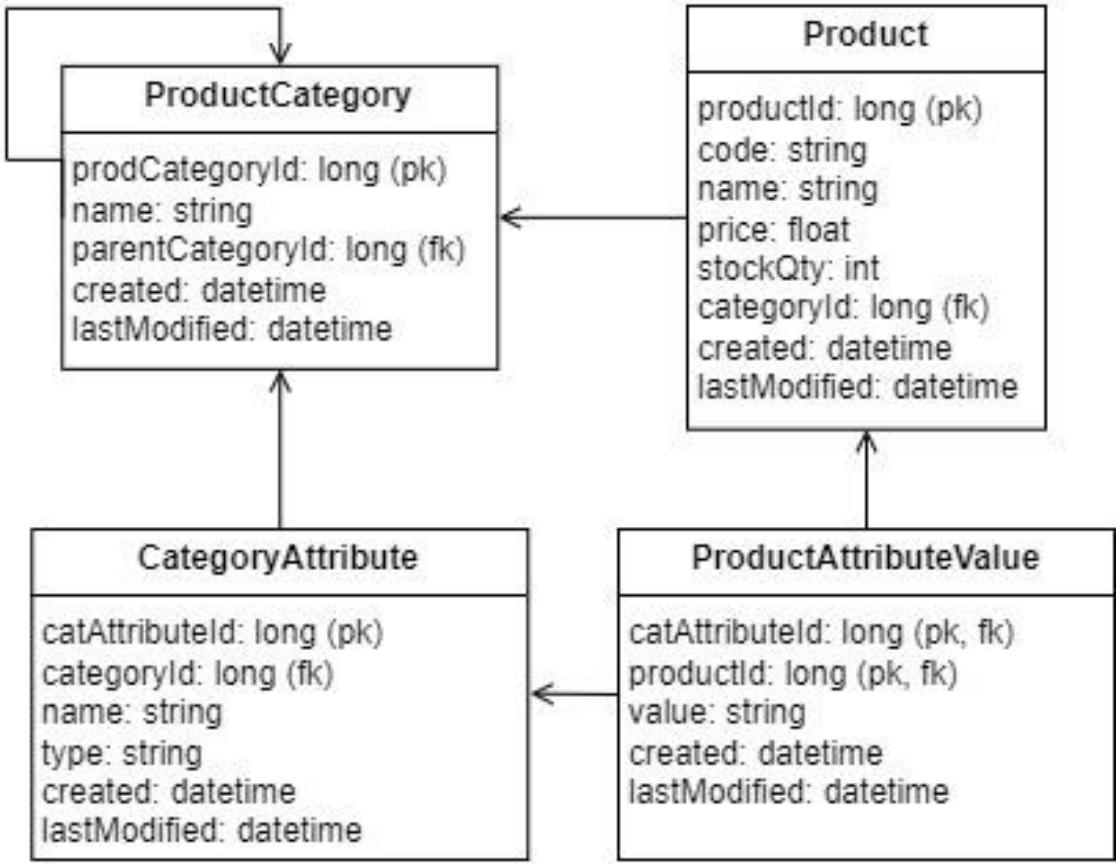


\+ corresponding constraints

Relational logical data model 2

EAV =
entity-attribute-value
model

+ corresponding constraints



NoSQL approaches - examples

- Document store - see demo
- Key-value store:

We talk about **semi-structured data.** Often schema-less approach.

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

- Wide-column store - see demo

NoSQL databases - common properties

- Non-relational logical structure
- SQL-like or custom query language
- Mostly designed for distributed environment and highly scalability
- Mostly relaxing ACID
- Mostly limited:
  - Referential integrity and constraints
  - The complexity of queries
- Often schema-less or with flexible schema
- Many are primarily designed for cloud environment
  - Some of them designed exclusively for cloud: Amazon DynamoDB, Google BigTable, ..

NoSQL:
We should rather talk about **non-relational databases**
= the logical structure is different from relational tables

Logical data structure

1. Relational tables
2. Key-value pairs
3. Documents accessed by keys
4. Wide-column format
5. Graph

} “NoSQL”

- ➢ Logical data structures can be combined
- ➢ RDBMSs’ support for non-relational structures:
  - Often XML / JSON data types are available
  - Key-value pairs, wide-columns and graphs can be stored indirectly

RDBMSs - document support

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

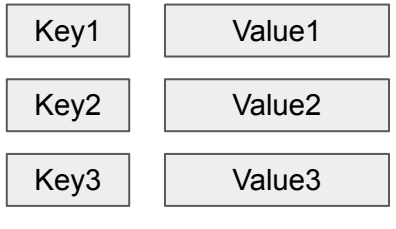
Relational tables

- Traditional RDBMSs
  - Row-based physical storage
  - ACID-compliant
  - Better for row-based queries
  - *PostgreSQL, MySQL, Oracle, Microsoft SQL Server, IBM DB2, Teradata*
- Columnar DBs
  - Columnar physical storage
  - Better for column-based queries, used for analytical purposes
  - *Google BigQuery, Amazon Redshift, Apache Druid, Apache Kudu*
- NewSQL DBs
  - Mostly row-based physical storage - we'll talk about them later
  - *Google Spanner, Cockroach DB, …*

columnar DBs
≠
wide-column DBs

Key-value stores

- Unique keys - mostly strings
  - “KA23E”, “product.KA23E”, …
- Values:
  - Various types - string, set, list, JSON object, …
- Only simple queries based on the key
  - `SET product:KA23E '{"name": "BestBike Junior"}'`
  - `GET product:KA23E, EXISTS product:KA23E, DEL product:KA23E`



- Use cases:
  - Fast key-based lookups (many popular key-value stores are in-memory DBs, e.g., Redis)
  - Applications that require scalability

- *Redis, Amazon DynamoDB, CouchBase*

Document stores

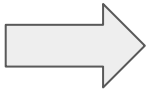
- A document is identified by a unique key
- Document format
  - JSON (BSON), XML, YAML, ..
- Less joins are needed

- Use cases:
  - Hierarchical data, semi-structured data
  - Fast and flexible development of web application
    - MERN stack: MongoDB+Express.js+React+Node.js
  - Applications that require scalability
- *MongoDB, CouchDB, Couchbase, …*

```
[
   {
     "firstName": "Wood",
     "lastName": "Lyons",
     "personalId": 87695,
     "catalog": [
       {
         "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
       }
     ]
   }
 ]
```

JSON to tables

```
[
   {
     "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
…	…	…	…

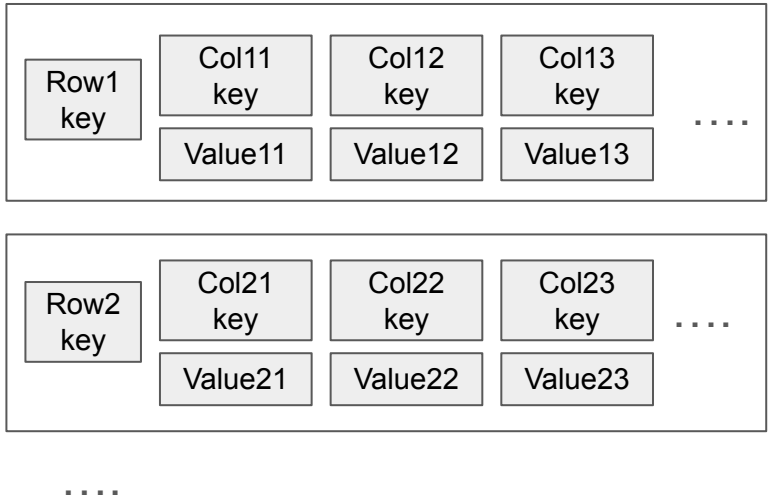
Addresses

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)
  - Applications that require scalability, high-throughput read and write operations

- *Apache Cassandra, Apache HBase, Google BigTable*



Another abstractions:

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

Graph databases

- Store efficiently nodes, edges and their properties
- Often highly scalable
- *Neo4j, ..*

Main aspects to consider

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

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? - avoid single point of failure, scalability, availability, reliability, response time, ..

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

Distributed databases - How to distribute

Two ways to distribute a database (can be combined):

- **Replication** - storing separate copies at two or more nodes
  - Single leader - one server receives writes.
  - Multileader, Leaderless
- **Partitioning (sharding)** - we divide data into smaller parts and then store them on separate nodes
  - In case of relational tables
    - 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
  - Work well in a centralized environment
  - R&D > 40 years
  - Difficult to distribute across multiple machines (= to scale horizontally)

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

Horizontal scalability Add more machines

- “NoSQL” DBs
  - Primarily intended for distributed environment, easy to scale horizontally
  - Many of them can be used also in a single machine

- Columnar DBs, NewSQL DBs
  - Relational databases intended for distributed environment, easy to scale horizontally

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 partitioning

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

New types of relational databases

- Based on relational tables and at the same time easy to scale horizontally

1. **Columnar databases**
   - Columnar physical storage
   - ACID not guaranteed
   - Primarily intended for analytical processing
2. **NewSQL databases**
   - Mostly row-based physical storage
   - ACID guaranteed
     - Availability is compromised in case of network partitions
   - Primarily Intended for transactional processing

- Immature concepts

Main aspects to consider

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

Transactional vs analytical processing

Transactional processing (OLTP)

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

Analytical processing (OLAP)

- Task: To analyze aggregated data + to analyze “big data”
- Operations: Primarily data reads, including complex queries
- Batch processing + real-time processing
- RDBMSs with “dimensional” data model, often denormalized
- New concepts: Columnar databases, NoSQL databases

“Big data” requirements

↳ large volumes of different data:

- Structured / semi-structured / unstructured
- Stored data vs data streams

Other solutions for Big data processing

Distributed file systems + computation frameworks:

- *Hadoop (HDFS) + MapReduce / Spark*

Cloud object storage + computation frameworks:

- *Google Cloud storage + Google Cloud compute engine, Amazon S3 + AWS Lambda / EC2*

→ highly scalable
→ both replication and partitioning are applied
→ efficient parallel processing

Main aspects to consider

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

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

Resources

- Robert Lukotka: Persistence and Databases
- C. M. Ricardo, Susan D. Urban: Databases Illuminated, 3rd edition, 2015.
- 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?