Distributed Information Management

Daniel Kocher

Salzburg, Summer term 2021

Department of Computer Sciences
University of Salzburg

Part I

Data Management

Literature, Sources, and Credits

Literature:

- Silberschatz et al. *Database System Concepts.* McGraw Hill, Sixth Edition, 2010.
- Wiese. *Advanced Data Management.* De Gruyter Oldenbourg, 2015.

Credits: These slides are partially based on slides of other lectures.

- Nikolaus Augsten, University of Salzburg, Austria (Credits also for valuable discussions and guidance).
- Andrew Pavlo, Carnegie Mellon University (CMU), USA.

Introduction

Motivation

The amount of data is growing rapidly in many different domains.

We do not collect data to simply store it, but we want to

- access it fast at any time and from any place,
- search it (for exact and similar patterns),
- aggregate it (partially),
- join it with other data,
- have multiple users working on the data concurrently,
- make sense out of it.

Motivation

“But I can organize my data without the overhead of a dedicated system!”

Theoretically yes, but this implies:

- Organizing your data in multiple independent (plain) files.
- Other users may have trouble to understand your organization.
- Taking care of all the requirements yourself.
- You drop flexibility and scalability.

What can possibly go wrong?

Example Application

Manage your movies collection in CSV files using Python3.

Movies: (name;year;runtime;genre;)

```
The Godfather;1972;177;Mafia;
The Avengers;2012;143;Sci-Fi;
Batman - The Dark Knight;2008;152;Action;
American Psycho;2000;101;Thriller;
Home Alone;1990;103;Comedy;
```

Actors: (name;birthyear;knownfor;)

```
Marlon Brando;1924;The Godfather;
Al Pacino;1940;The Godfather;
Macaulay Culkin;1980;Home Alone;
Joe Pesci;1943;Home Alone;
Robert Downey Jr.;1965;The Avengers;
Mark Ruffalo;1967;The Avengers;
Christian Bale;1974;Batman - The Dark Knight;
Christian Bale;1974;American Psycho;
Heath Ledger;1979;Batman - The Dark Knight;
Willem Dafoe;1955;American Psycho;
```

Example Application

Query: Find all movies of “Christian Bale”.

```
if __name__ == ”__main__”:
  with open(”actor.csv”, ”r”) as fin:
    for line in fin.readlines():
      parts = line.split(”;”)

      name = parts[0]
      birthyear = int(parts[1])
      knownfor = parts[2]

      if name == ”Christian␣Bale”:
        print(knownfor)
```

Problems in Data Management

Redundancy and Inconsistency:

- Several copies of a datum may exist (possibly stored differently).
- **Redundancy:** Higher storage requirements. How about accessing the data?
  $\Rightarrow$ Multiple file accesses (which is slow).
- **Inconsistency:** What happens if you have to update the data?
  $\Rightarrow$ We must not forget a single copy.
- **Goal:** Minimize redundancy and prevent inconsistency.

Example: Update "The Avengers" to "Marvel's The Avengers".

Example Application

Query: Update “The Avengers” to “Marvel’s The Avengers”.

Movies: (name;year;runtime;genre;)

```
The Godfather;1972;177;Mafia;
Marvel's The Avengers;2012;143;Sci-Fi;
Batman - The Dark Knight;2008;152;Action;
American Psycho;2000;101;Thriller;
Home Alone;1990;103;Comedy;
```

Actors: (name;birthyear;knownfor;)

```
Marlon Brando;1924;The Godfather;
Al Pacino;1940;The Godfather;
Macaulay Culkin;1980;Home Alone;
Joe Pesci;1943;Home Alone;
Robert Downey Jr.;1965;The Avengers;
Mark Ruffalo;1967;The Avengers;
Christian Bale;1974;Batman - The Dark Knight;
Christian Bale;1974;American Psycho;
Heath Ledger;1979;Batman - The Dark Knight;
Willem Dafoe;1955;American Psycho;
```

Problems in Data Management

Data Access and Analysis:

- We want to analyze our data. How to link related data?
  ⇒ Each analysis requires a tailored program.
- **Goal:** Generic analysis and linkage of related data.

Example: List all movies of “Christian Bale” and with its runtime.

Example Application

Query: List all movies of “Christian Bale” and with its runtime.

Movies: (name;year;runtime;genre;)

```
The Godfather;1972;177;Mafia;
Marvel's The Avengers;2012;143;Sci-Fi;
Batman - The Dark Knight;2008;152;Action;
American Psycho;2000;101;Thriller;
Home Alone;1990;103;Comedy;
```

Actors: (name;birthyear;knownfor;)

```
Marlon Brando;1924;The Godfather;
Al Pacino;1940;The Godfather;
Macaulay Culkin;1980;Home Alone;
Joe Pesci;1943;Home Alone;
Robert Downey Jr.;1965;The Avengers;
Mark Ruffalo;1967;The Avengers;
Christian Bale;1974;Batman - The Dark Knight;
Christian Bale;1974;American Psycho;
Heath Ledger;1979;Batman - The Dark Knight;
Willem Dafoe;1955;American Psycho;
```

Problems in Data Management

Data Integrity Issues:

- Updates may violate the integrity of your data.
- How do you ensure data integrity?
  $\Rightarrow$ Each single application must respect all consistency constraints.
- **Goal:** Global definition and monitoring of consistency constraints.

Example: Insert “Christian Bale” as actor in “The Avengers”.

Problems in Data Management

Concurrency Issues:

- Multiple users should be able to access and update the data simultaneously. How do you ensure consistency over all applications that access the data?
- **Anomalies:** Inconsistencies, e.g., lost updates.
- **Efficiency:** If one user locks a file, then the other user must wait.
- **Goal:** Out-of-the-box multi-user operation without anomalies.

Example: Users A inserts “Chris Evans” as actor while user B inserts “Scarlett Johansson” as actress in “The Avengers” simultaneously.

Example Application

Scenario: Users A inserts “Chris Evans” as actor while user B inserts “Scarlett Johansson” as actress in “The Avengers” simultaneously.

Actors: (name;birthyear;knownfor;)

```
Marlon Brando;1924;The Godfather;
Al Pacino;1940;The Godfather;
Macaulay Culkin;1980;Home Alone;
Joe Pesci;1943;Home Alone;
Robert Downey Jr.;1965;The Avengers;
Mark Ruffalo;1967;The Avengers;
Christian Bale;1974;Batman - The Dark Knight;
Christian Bale;1974;American Psycho;
Heath Ledger;1979;Batman - The Dark Knight;
Willem Dafoe;1955;American Psycho;
Scarlett Johansson;1984;The Avengers;
```

Problems in Data Management

Atomicity and Recovery:

- Data must neither be lost nor inconsistent when the system crashes.
- **Atomicity:** Data may be inconsistent if an operation is only applied partially
  $\Longrightarrow$ Execute an operation in an all-or-nothing manner.
- **Recovery:** Backup of data may not reflect the latest state.
- **Goal:** Prevent data loss and inconsistencies by design.

Example: Update all "The Avengers" to "Marvel's The Avengers". Your system crashes in between "Mark Ruffalo" and "Scarlett Johansson".

Example Application

Scenario: Update all "The Avengers" to "Marvel's The Avengers". Your system crashes in between "Mark Ruffalo" and "Scarlett Johansson".

Actors: (name;birthyear;knownfor;)

```
Marlon Brando;1924;The Godfather;
Al Pacino;1940;The Godfather;
Macaulay Culkin;1980;Home Alone;
Joe Pesci;1943;Home Alone;
Robert Downey Jr.;1965;Marvel's The Avengers;
Mark Ruffalo;1967;Marvel's The Avengers;
Christian Bale;1974;Batman - The Dark Knight;
Christian Bale;1974;American Psycho;
Heath Ledger;1979;Batman - The Dark Knight;
Willem Dafoe;1955;American Psycho;
Scarlett Johansson;1984;The Avengers;
```

Problems in Data Management

Other Issues:

- **Efficiency:** Efficient algorithms are required to analyze large amounts of data.
- **General-purpose:** The problems of application developers will partially overlap.
- **Security issues:** Flexible and fine-grained access rights for multiple users.

Database Management Systems

A **database management system** (DBMS) is

(i) a collection of interrelated data, the **database**, and

(ii) a set of programs to access the data.

In other words, you do not have to care about how to store the data (physically), how to analyze it (efficiently), how to (partially) update data, or how to deal with multiple users. A DBMS organizes all this for you.

DBMSs are at the core of many applications.

When Not to Use a DBMS

- The data are too complex to model it.
- Specific requirements like real-time queries or special operations.
- The overhead of a DBMS is too high or unnecessary.
- No or low return on investment (ROI).

One Size Fits All?

General-Purpose DBMS

A DBMS that tries to fit as many application scenarios as possbile with a single system. This implies higher complexity but also a large user base.

Examples:

- PostgreSQL (open source)
- MySQL (open source)
- MonetDB (open source)
- SQLite (open source)
- IBM DB2 (closed source)
- Oracle Database (closed source)
- Microsoft SQL Server (closed source)
- …

General-Purpose DBMS

“But this sounds good, no?”

Problems:

- Unnecessary overhead (e.g., recovery or strong consistency)
- Limited performance
- Application-specific operations are not supported natively
- Limited flexibility

One Size Fits All (OSFA)

"One Size Fits All": An Idea Whose Time Has Come and Gone

Michael Stonebraker[1] and Ugur Cetintemel (2005)

"A one size fits all database doesn't fit anyone"

Werner Vogels[2] (2018)

[1] Database Systems Researcher at the MIT. Won the Turing Award in 2014.

[2] Computer Scientist and CTO at Amazon.

Special-Purpose DBMS

A DBMS that is tailored to fit a specific purpose best, i.e., provide all the functionality that is required while also providing the best performance and flexilibity (with respect to the specific application domain).

Synonyms: Specialized DBMS, purpose-built DBMS.

Special-Purpose DBMS

Temporal Data: A temporal DBMS is optimized to manage and analyze data that references time (i.e., they are timestamped). For example, a time series $x = \langle x_{t_1}, x_{t_2}, \dots, x_{t_n} \rangle$ is often a sequence of n data points that are spaced at strictly increasing times ($t_i < t_{i+1}$ with $i = 1, \dots, n-1$).

Requirements:

- Exact/Approximate matching of (parts of) time series.
- Efficient compression mechanisms.
- Serve specific aspects like valid time or transaction time.
- …

Special-Purpose DBMS

Real-Time Data: A real-time DBMS manages data that is changed continuously. A DBMS that operates in real time answers the queries within a guaranteed time frame (the response time, i.e., it has a deadline).

Requirements:

- Answer every query in a given time frame.
- Query scheduling (or queuing).
- Consistency may not be that important.
- …

Special-Purpose DBMS

Process Mining Data: Process mining engines manage business event logs. An example event log is the sequence of activities if you place an order in some online shop. These systems are required to analyze large amounts of data in real time.

Requirements:

- Optimized, domain-specific language.
- Real-time performance for best user experience.
- …

Special-Purpose DBMS

Multiple specific aspects may need to be combined to serve a novel application scenario. This may also result in a new special-purpose DBMS.

A modern application is not monolithic, i.e., different DBMSs may be used to implement different parts of an application.

One Size Fits All

Towards a One Size Fits All Database Architecture

Jens Dittrich[3] and Alekh Jindal. 2011.

One Size Fits all, Again! The Architecture of the Hybrid OLTP&OLAP Database Management System HyPer

Alfons Kemper[4] and Thomas Neumann[5]. 2011.

[3] Database Systems Researcher at Saarland University.

[4] Database Systems Researcher at the TU Munich co-author of the book *Datenbanksysteme*.

[5] Database Systems Researcher at the TU Munich.

Database Fundamentals

Basic Terminology

Data are facts that are to be stored.

Information is data combined with semantics (meaning).

Knowledge is information combined with an application.

Example Application

What are data, information, and knowledge in our example?

Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
Batman - The Dark Knight	2008	152	Action
American Psycho	2000	101	Thriller
Home Alone	1990	103	Comedy

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
Macaulay Culkin	1980	Home Alone
Joe Pesci	1943	Home Alone
Robert Downey Jr.	1965	The Avengers
Mark Ruffalo	1967	The Avengers
Christian Bale	1974	Batman - The Dark Knight
Christian Bale	1974	American Psycho
Heath Ledger	1979	Batman - The Dark Knight
Willem Dafoe	1955	American Psycho

Basic Terminology

A **database** (DB) is a collection of interrelated data.

Metadata provides us with information about the structure of a database. All the metadata are stored in a **catalog**.

A **database system** (DBS) is also referred to as the combination of a database, the corresponding metadata, and a DBMS (which in this case only provides the set of programs). The terms DBS and DBMS are often used interchangeably.

Example Application

Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
…	…	…	…

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
…	…	…

Catalog:

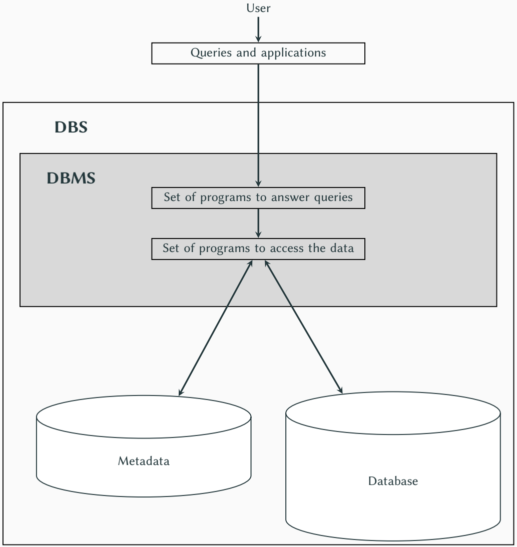
Tables Metadata:

relation	columnCount
Actors	10
Movies	5

Columns Metadata:

columnName	dataType	relation
name	TEXT	Actors
year	INTEGER	Actors
name	TEXT	Movies
…	…	…

Database System


User
Queries and applications
DBS
DBMS
Set of programs to answer queries
Set of programs to access the data
Metadata
Database

More Terminology

A **table** consists of multiple **tuples**, each of which is a sequence of **attributes**. Informally, a tuple (attribute) can be imagined as a row (column) of a table.

A **key** is subset of attributes. A **primary key** is a key of minimum length that uniquely identifies a tuple. A **foreign key** is a reference to a primary key.

Data Modeling

Data-Definition Language (DDL): Specify the structure of your data and the consistency constraints that are enforced.

- **Schema:** Describes the structure of your data and how the data are interrelated, e.g., a movie has 4 columns: name, year, runtime, and genre.
- **Consistency Constraints:** Describe integrity constraints that must be satisfied at any given point in time, e.g., the runtime in minutes is an integer > 0.

Example Application

Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
...	...	...	...

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
...	...	...

Pseudo-DDL (simplified):

```
CREATE TABLE Movies (
  name TEXT KEY,
  year INTEGER,
  runtime INTEGER (> 0),
  genre TEXT
)
```

```
CREATE TABLE Actors (
  name TEXT KEY,
  birthyear INTEGER,
  knownfor TEXT REFERENCES(Movies.name)
)
```

Data Modeling

Data-Manipulation Language (DML): Query and manipulate your data.

- **Query Language:** Allows you to query your data without modifying it, e.g., get all movies of “Christian Bale” or get all actors of “The Avengers”.
- **Manipulation Language:** Allows you to modify your data, e.g., insert a new movie, delete an existing movie, update the movies of a particular actor.

A **query** is a statement that requests some information. Informally, your query “asks” and the database system answers by returning the corresponding information.

Caveat: The term query language often refers to both DML parts.

Example Application

Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
...	...	...	...

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
...	...	...

Pseudo-DML (simplified):

```
SELECT knownfor FROM Actors
WHERE name = 'Christian Bale'

SELECT name FROM Actors
WHERE knownfor = 'The Avengers'
```

```
INSERT INTO Movies
VALUES ('Hulk', 2003, 133, 'Sci-Fi')

UPDATE Actors
SET knownfor = 'American Psycho'
WHERE name = 'Christian Bale'
```

Query Languages

Procedural Languages: Describe what data you want and also how to retrieve those.

Declarative Languages: Describe only what data you are interested in.

Pure Languages: Form the (theoretical) foundation underneath the languages that are used in practice. Examples include relational algebra or tuple calculus.

Our Focus: Declarative languages.

Real-World Analogy

Send someone to the supermarket to get milk.

The SQL Query Language

The **Structured Query Language (SQL)** was developed by IBM and is the de-facto standard language in database systems.

SQL is a declarative query language and includes DDL and DML elements.

The SQL standard (last revision: 2016) comprehensively summarizes all elements.

Inventing new approaches is like *“trying to swim up the Niagara Falls”.*

Michael Stonebraker in Information Age. 2010.

Example Queries in SQL

Data Definition:

```
CREATE TABLE Movies (
  name VARCHAR(50) PRIMARY KEY,
  year INTEGER NOT NULL,
  runtime INTEGER CHECK(runtime > 0),
  genre VARCHAR(50)
);
```

```
CREATE TABLE Actors (
  name VARCHAR(50) PRIMARY KEY,
  birthyear INTEGER,
  knownfor VARCHAR(50) REFERENCES Movies(name)
);
```

- **`VARCHAR(n)`**, **`INTEGER`**: Domain of a single column (data types).
- **`NOT NULL`**, **`CHECK`**: Constraints on a single column.
- **`PRIMARY KEY`**, **`REFERENCES`**: Constraints on an entire table.

Example Queries in SQL

Queries:

```
SELECT knownfor FROM Actors
WHERE name = 'Christian Bale';

SELECT name FROM Actors
WHERE knownfor = 'The Avengers';
```

- **`SELECT`:** Specifies the column to retrieve.
- **`FROM`:** Specifies the tables to consider.
- **`WHERE`:** Specifies the condition(s) the result must satisfy.

Example Queries in SQL

Queries:

```
INSERT INTO Movies(name, year, runtime, genre)
VALUES ('Hulk', 2003, 133, 'Sci-Fi');

UPDATE Actors SET knownfor = 'American Psycho'
WHERE name = 'Christian Bale';
```

- **`INSERT INTO` ... `VALUES`:** Adds new tuple to a table based on the given values.
- **`UPDATE` ... `SET`:** (Partially) changes the values of a tuple.

Data Abstraction

Abstraction: Hide the complexity of the system while providing all the functionality (from people without deep computer science background). Everyone should be able to use a database system.

3 Levels of Data Abstraction (bottom-up):[6]

1. **Physical:** How to store the data (e.g., on hard disk).
2. **Logical:** What data are stored and how are they related.
3. **View:** Specific views on the data (e.g., on a specific part of the data).

[6] Cf. also ANSI/SPARC architecture.

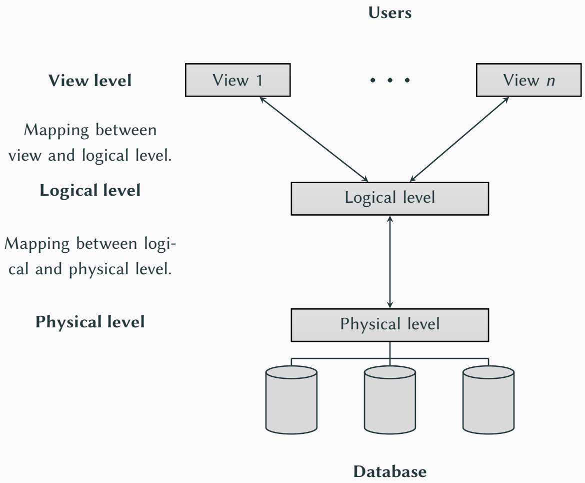
3 Levels of Data Abstraction

The **physical level** defines the data structures that are used to store and access the data physically. Examples are tables or auxiliary structures like indexes.

The **logical level** defines the schemata and constraints of the entire database.
Physical structures may be used underneath, but the user does not have to know them.
⇒ **physical data independence**.

The **view level** reduces the complexity by providing only information that is necessary for the respective user. Irrelevant data are not shown.

3 Levels of Data Abstraction



Users

View level

View 1 • • • View *n*

Mapping between view and logical level.

Logical level

Logical level

Mapping between logical and physical level.

Physical level

Physical level

Database

Mappings are used to link the abstraction levels.

Instance vs. Schema

A **schema** describes the overall structure of the data (often using tables) and is typically stable (i.e., rarely modified).

An **instance** is the information that is stored at a particular point in time. The instance may be frequently subject to changes.

Each level has its own schema with the logical schema being the most important one.

A **valid instance** satisfies all structural requirements and consistency constraints.

Instance vs. Schema

Schemata:

Movies:

name	year	runtime	genre

Actors:

name	birthyear	knownfor

Instances:

Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
…	…	…	…

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
…	…	…

Data Independence

Logical Data Independence: The ability to update the logical schema transparently, i.e., no change on the view level is required.

Physical Data Independence: The ability to update the physical schema transparently, i.e., no change on the logical level is required.

Benefits:

- Only the mapping between the levels need to be adapted.
- No change in the application required (it operates on the views).

Assignment 1

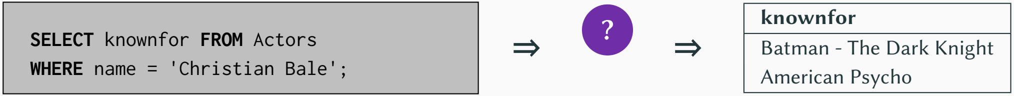
Assignment 1

Summary:

- **Where:** Release and submission via Blackboard.
- **When:** March 22 – April 19, 2021 (resp. late: April 26, 2021).
- **What to do:** (a) Set up a relational database locally (PostgreSQL; data will be provided), (b) get familiar with SQL and learn how to execute queries (most queries will be given), (c) write a small Python3 application that executes the queries, and (d) answer questions regarding the assignment.
- **What to submit:** The Python3 code and the answers to the questions.
- **Grading:** 55% Python3 code, 45% questions (incl. the meeting).

(Declarative) Query Processing

Example SQL Query (+ Result):



```sql
SELECT knownfor FROM Actors
WHERE name = 'Christian Bale';
```

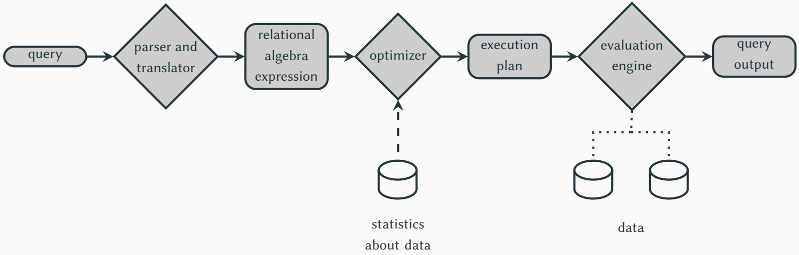
knownfor
Batman - The Dark Knight
American Psycho

Query processing describes the process of extracting data from a database. In other words: What happens in a database when we issue a (declarative) query?

(Declarative) Query Processing

On a high level, **three major components** are used to process a query:

1. **Parser:** Translates the query into an internal representation.
2. **Optimizer:** Choses the most efficient way to evaluate the query.
3. **Evaluation Engine:** Executes the evaluation plan and returns the result.



(Declarative) Query Processing

An evaluation plan typically consists of multiple operation. Optimization is done based on the **estimated costs** of all involved operations.

For a given query, multiple valid **evaluation plans** may exist and must be **compared efficiently** (with respect to their estimated costs).

The estimated costs consider **many different factors** including access to hard disk, time to execute the query on the CPU, or network communication costs.

Data Models

Types of Data Models

3 types of data models that are somewhat related to the 3 levels of data abstraction.

Conceptual data models: High level, i.e., only the schema is reflected but no instances. Related to the view level. Examples include Entity-Relationship (ER) and Unified Modeling Language (UML) models.

Logical data models: Depicts the instances and can be used to implement a database. Related to the logical level. Examples include the relational and the object-based models.

Physical data models: Low level, i.e., as close to the physical storage as possible. Related to the physical level and is typically system-specific.

Types of Data Models – Examples

Logical Model (Relational):

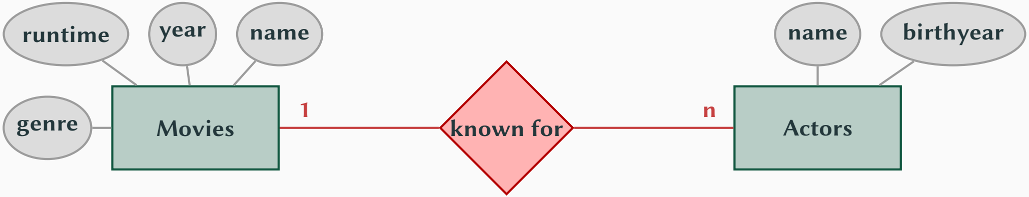
Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
. . .	. . .	. . .	. . .

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
. . .	. . .	. . .

Conceptual Model (ER):



Relational Data Model

Intuitive and **widely used** model. An example of **record-based** models.

A **collection of relations (tables) stores records of data as rows.** A **tuple (row)** is an **entity** of the real world, an **attribute (column)** is a **property** of an entity. The **structure** of a record is **fixed.**

A **relation** has a **name** and a **set of (unique) attributes**. An **attribute** has a **name** and a **predefined domain**, i.e., values originate from a specific domain.

Tables are **filled row-wise** and a **row** represents the **state of an entity.**

Relational Data Model

The set of columns is called **relation schema**, and the set of relation schemata over all tables is called the **database schema**.

Intrarelational constraints: Dependencies inside a single table, Σ_i

Interrelational constraints: Dependencies between different tables, Σ

Relation schema: $R_i = (\{A_{i1}, A_{i2}, \ldots, A_{im}\}, \Sigma_i)$[7].

Database schema: $D = (\{R_1, R_2, \ldots, R_n\}, \Sigma)$.

[7] A_{ij} …Name of the j-th attribute (column) of the i-th relation (table).

Relational Data Model – Example

Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
…	…	…	…

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
…	…	…

Relation schemata:

$$\begin{aligned}\text{Movies} &= (\{\text{name}, \text{year}, \text{runtime}, \text{genre}\}, \{\text{name}, \text{year} \longrightarrow \text{runtime}, \text{genre}\})\\ \text{Actors} &= (\{\text{name}, \text{birthyear}, \text{knownfor}\}, \{\text{name}, \text{birthyear} \longrightarrow \text{knownfor}\})\end{aligned}$$

Database schema:

$$\text{MovieStore} = (\{\text{Movies}, \text{Actors}\}, \{\text{Actors.knownfor} \subseteq \text{Movies.name}\})$$

Relational Data Model

Normalization: Reduce anomalies (which lead to inconsistencies) by distributing attributes among tables and linking them using **foreign key constraints**.

Referential Integrity: Values of foreign keys exist as values in the referenced table, i.e., the referenced table contains at least one tuple that holds the value of the foreign key.

Relational Data Model – Example

Movies:

id	name	year	runtime	genre
1	The Godfather	1972	177	Mafia
2	The Avengers	2012	143	Sci-Fi
3	Batman - The Dark Knight	2008	152	Action
4	American Psycho	2000	101	Thriller
5	Home Alone	1990	103	Comedy

Actors:

id	name	birthyear	knownfor
101	Marlon Brando	1924	The Godfather
102	Al Pacino	1940	The Godfather
103	Macaulay Culkin	1980	Home Alone
104	Joe Pesci	1943	Home Alone
105	Robert Downey Jr.	1965	The Avengers
106	Mark Ruffalo	1967	The Avengers
107	Christian Bale	1974	Batman - The Dark Knight
108	Christian Bale	1974	American Psycho
109	Heath Ledger	1979	Batman - The Dark Knight
110	Willem Dafoe	1955	American Psycho

Relational Data Model – Example

Movies:

id	name	year	runtime	genre
1	The Godfather	1972	177	Mafia
2	The Avengers	2012	143	Sci-Fi
3	Batman - The Dark Knight	2008	152	Action
4	American Psycho	2000	101	Thriller
5	Home Alone	1990	103	Comedy

Actors:

id	name	birthyear
…	…	…
106	Mark Ruffalo	1967
107	Christian Bale	1974
108	Heath Ledger	1979
109	Willem Dafoe	1955

knownFor:

movie-id	actor-id
…	…
2	106
3	108
3	107
4	107
4	109

Relational Data Model

Drawbacks:

- Relations may not be optimal to represent the data.
- Everything is a relation (semantic overloading).
- Homogeneous structure of data.
- Limited flexibility and data types.

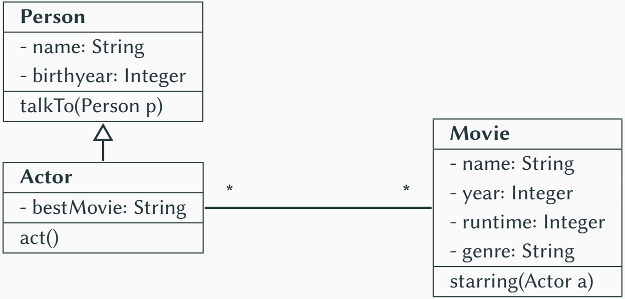
Drawbacks and new challenges gave rise to **non-relational data models.**

Object-Based Data Model

Many programming languages are object-oriented, i.e., based on the concept of objects.

Objects contain data and provide functionality, and interact with each other (e.g., like human beings in the real world).

Example Objects:



Object-Based Data Model

Three Options:

1. **Object-relational data model:** Extends relational data model with object-oriented features.
2. **Object-relational mapping (ORM):** Maps objects to tuples in the relational model and handles the translation.
3. **Object-oriented data model:** Implements an object-based data model natively.

Key-Value Data Model

Prototype of a **schemaless** model where each tuple is a pair of two strings (k, v). k is a **unique key** that is associated with an arbitrary **value** v.

Data are accessed using the key and values are not interpreted by any means.
Example operations:

- `put(k, v)`: Inserts a new pair key-value pairs (k, v)
- `get(k)`: Retrieves the value associated with k.
- `delete(k)`: Deletes the tuple(s) associated with k.

Caveat: In programming often referred to as hash table or dictionary (Python).

Key-Value Data Model

Only keys can be searched, **values cannot be searched.** Combinations of values must be done by the application that accesses the data.

Easy to distribute ⟹ Good for **data-intensive applications.**

Example Key-Value Store:

UserID		Shopping Cart
1002	→	Shoe, Jordans, red, 37 # Computer game “Diablo”, Blizzard # Headset, Razer, Kraken Kitty
1003	→	Wilson American Football, NFL, Replica # Hail Mary, Gloves Receiver, 2.0, Black & White
1004	→	Book “Database System Concepts”, 7th Edition, Silberschatz
…		…

Document-Based Data Model

A document stores data in a semi-structured and nested text format (e.g., XML or JSON).

Each document has a unique identifier, but the value is a document structured in a specific format that is interpretable (as opposed to a key-value store).

JavaScript Object Notation (JSON): Human-readable text format for data structures. A JSON document consists of possibly nested key-value pairs.

Document-Based Data Model

A JSON document consists of a JSON object, which is enclosed by curly braces, { . . . }. Inside, keys and the corresponding values are separated by a colon, and a value can be a JSON object itself.

Example Document Store:

UserID		Shopping Cart

1002	→	{
		“1”: { “type”: “Shoe”, “name”: “Jordans”, “color”: “red”, “size”: 37 },
		“2”: { “type”: “Computer game”, “name”: “Diablo”, “publisher”: “Blizzard” },
		“3”: { “gear”: “Headset”, “producer”: “Razer”, “model”: “Kraken Kitty” }
		}

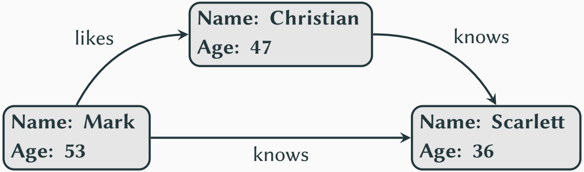
…		…

Graph-Based Data Model

Graphs: Informally, graphs are structures that represent data (as nodes) and their interrelation (as edges in between) by design. Both nodes and edges may carry information. Efficient graph operations are supported natively.

Data and their relationship are distinct naturally (cf. semantic overloading of the relational data model).

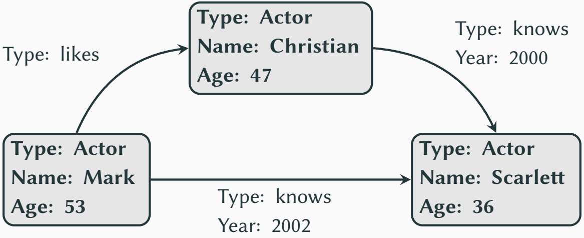
Example Graph:



Graph-Based Data Model

Property Graph Model: Multiple types for nodes/edges (multi-relational), each of which may contain multiple properties (attributes) as name-value pairs (similar to key-value pairs).

Example Property Graph:



Extensible Record Data Model

Also referred to as **wide column data model** and a generalization of Google's BigTable[8] system.

Table cells are represented as (2-dimensional) key-value pairs, with row (unique) and column (repeatedly) being the keys.

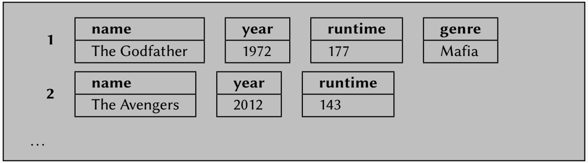
Example: Access table cell `(4:year)`.

id	**name**	**year**	**runtime**	**genre**
1	The Godfather	1972	177	Mafia
2	The Avengers	2012	143	NULL
3	Batman - The Dark Knight	2008	152	Action
4	American Psycho	**2000**	101	Thriller
5	Home Alone	1990	103	Comedy

[8] Chang et al. *Bigtable: A Distributed Storage System for Structured Data.* OSDI, 2006.

Extensible Record Data Model

Column Families: Group columns that are accessed simultaneously.

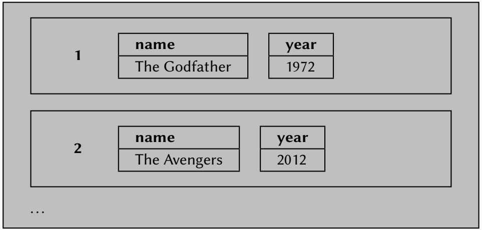


1 name The Godfather year 1972 runtime 177 genre Mafia

2 name The Avengers year 2012 runtime 143

...

CoreInfo:

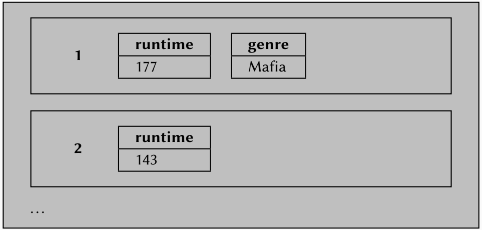


1 name The Godfather year 1972

2 name The Avengers year 2012

...

ExtendedInfo:



1 runtime 177 genre Mafia

2 runtime 143

...

Extensible Record Data Model

Encourages de-normalization (redundancy) for higher performance and provides **more flexibility**, i.e., every row may be composed of different columns.

The query workload defines how the data is modeled $\Rightarrow$ Know your workload.

Array-Based Data Model

Complex structures (like 2-dimensional satellite or *n*-dimensional sensor data) are organized along multiple dimensions.

An array cell contains a tuple of a specific length and a tuple element may contain a value or another array (allowing arbitrary nestings).

Specialized array functionality is natively supported (e.g., joins and aggregations).

Workloads & Challenges

Outline

- New data management **challenges**.
- Basics of **distributed** database systems (also compared to parallel DBS).
- **Consistency guarantees:** ACID vs. BASE.
- The **CAP** Theorem.
- **Workloads:** OLTP vs. OLAP, batch vs. stream processing.

New Data Management Challenges

- Data may be organized in **complex structures** (e.g., graphs).
- Data may be **schemaless** (e.g., not comply to a fixed schema).
- Data may be **sparse** (e.g., values may be non-existent).
- Data or schema may be **constantly changing** (e.g., schema evolution).
- Data may be **distributed over multiple machines.**
    - Access must be **transparent** (i.e., user need not know where data resides).
    - Systems must **scale horizontally** (i.e., new machines come and go).
    - Systems must cope with **large data volumes**.

New Data Management Challenges

NoSQL (Not Only SQL):

- Class of non-relational DBSs.
- Weaker consistency guarantees (e.g., BASE[9])
- Support for schema independence.
- Highly scalable.

NewSQL:

- Class of relational DBSs.
- Provide NoSQL-like scalability (for OLTP[9] workloads).
- Retain consistency guarantees of RDBMS (e.g., ACID[9]).

[9] We will cover this term subsequently.

Parallel Database Systems

Parallel database management systems (PDBMSs) have multiple processors and hard disks that are connected via a fast interconnection.

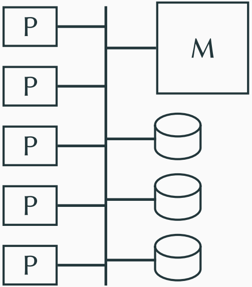
Performance characteristics:

- **Throughput:** Number of tasks (e.g., queries) that can be completed in a given time frame. Example: Queries per seconds.
- **Response time:** Time it takes to complete a single task (e.g., query).

Parallel Database System Architectures

Shared memory DBMSs have many processors and disks that share a common memory (typically via a bus).

- + Efficient communication between processors ($< 1\mu s$).
- - Limited scalability (≤ 64 processors; interconnection to memory becomes the bottleneck).

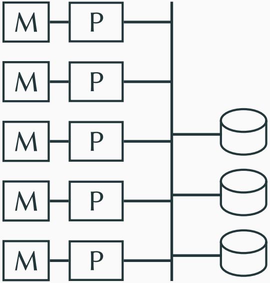


P . . . processor, M . . . memory, ⛁ . . . disks.

Parallel Database System Architectures

Shared disk DBMSs have many processors (with isolated memory) that share all disks (typically via a bus).

- + Scale to a larger number of processors.
- - Communication between processors is slower (ms; bottleneck now at interconnection to disks).

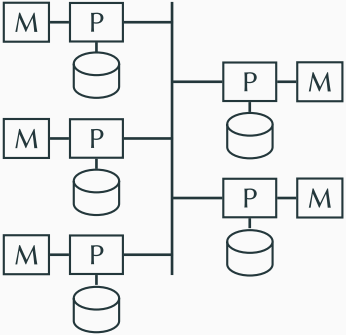


P . . . processor, M . . . memory, ⛁ . . . disks.

Parallel Database System Architectures

Shared nothing DBMSs have many processors with isolated memory and disk(s). The combination of a processor with isolated memory and disk(s) is also referred to as **node**.

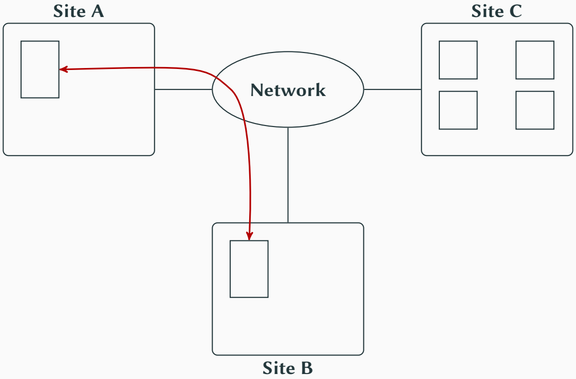
+ Scale to thousands of processors.
- Communication between processors is slow; access to non-local disk data is slow.



P …processor, M …memory, ⛁ …disks.

Distributed Database Systems

Distributed database management systems (DDBMSs) are DBMSs that operate on multiple, geographically separated machines (also called **sites**).



DDBMS vs. Shared-Nothing PDBMS

- Sites within a DDBMS are typically
  - **geograpically separated** (i.e., not a single data center)
    ⇒ lower bandwidth (less throughput), higher latency (higher response time).
  - **separately administered** (i.e., retain some degree of autonomy).
- PDBMS can deal with node failures, whereas **DDBMS** can deal with **failures of entire sites** (e.g., due to natural disasters).
- DDBMS distinguish between **local** and **global transactions.**

Homogeneous vs. Heterogeneous DDBMS

A DDBMS is called **homogeneous** if the nodes share a common global database schema, perform the same tasks (e.g., run the same software), and actively cooperate in processing. Goal: View of a single database.

A DDBMS is called **heterogeneous** if the nodes have different schemata, perform different tasks (e.g., run different software), and may not be aware of other nodes. Goal: Integrate different databases.

Transactions

A **transaction** refers to a **sequence of operations** that accesses and (possibly) updates various data items. A transaction **transitions the database from one consistent state into another consistent state.**

Example: Transfer EUR 500 from bank account A to bank account B.

```
1. BEGIN
2. READ(A)
3. A = A - 500
4. WRITE(A)
5. READ(B)
6. B = B + 500
7. WRITE(B)
8. COMMIT
```

Transactions

A DBMS must deal with **two major issues:**

- **System crash:** Software or hardware failures.
- **Concurrency:** Many different users may work at the same time (i.e., multiple transactions are executed).

A DBS implements particular consistency guarantees. Most relational DBMS manage transactions according to the so-called **ACID properties:** **A**tomicity, **C**onsistency, **I**solation, and **D**urability.

The ACID properties are considered **strong consistency guarantees.**

ACID Properties

Atomicity: Execute **all operations** of a transaction **or none** of them (*“all or nothing”*).

Example: Transfer EUR 500 from bank account A to bank account B.

```
1. BEGIN
2. READ(A)
3. A = A - 500
4. WRITE(A)
5. READ(B)
6. B = B + 500
7. WRITE(B)
8. COMMIT
```

What happens if the system crashes after step 4?

ACID Properties

Consistency: After a transaction, all values in the database are correct (i.e., consistency is preserved).

Example: Transfer EUR 500 from bank account A to bank account B.

```
BEGIN
READ(A)
A = A - 500
WRITE(A)
READ(B)
B = B + 500
WRITE(B)
COMMIT
```

What is a consistency constraint in the above example? A + B is the same before and after the transaction.

ACID Properties

Isolation: A transaction must be unaware of other simultaneously executing transactions (otherwise an inconsistent state may be encountered).

Example: Transfer EUR 500 from bank account A to bank account B.

```
1. BEGIN
2. READ(A)
3. A = A - 500
4. WRITE(A)
5. READ(B)
6. B = B + 500
7. WRITE(B)
8. COMMIT
```

Imagine a second transaction `BEGIN; READ(A); READ(B); print(A + B); COMMIT` in between steps 4 and 5.

ACID Properties

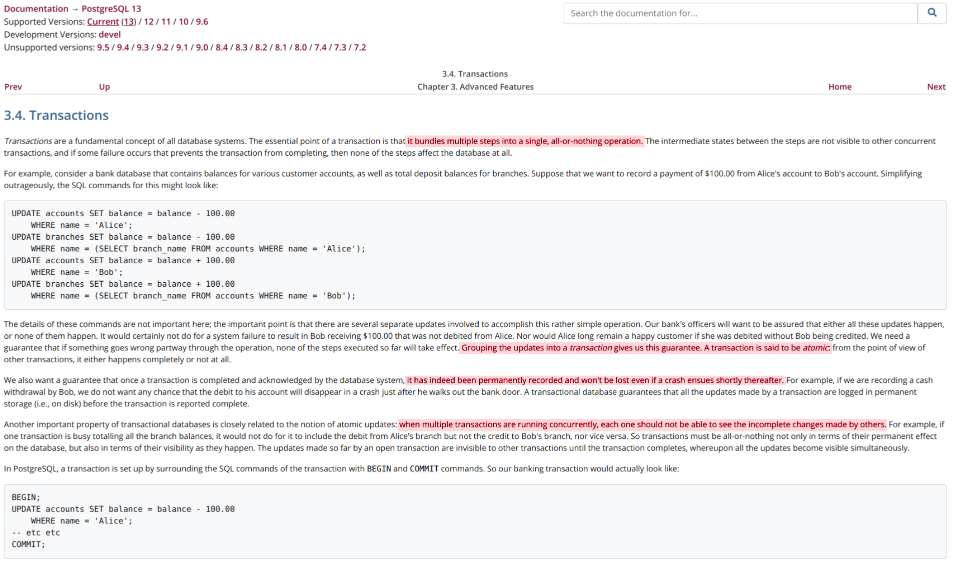
Durability: Successful transaction results persist in the database, even if the system crashes.

Example: Transfer EUR 500 from bank account A to bank account B.

```
1. BEGIN
2. READ(A)
3. A = A - 500
4. WRITE(A)
5. READ(B)
6. B = B + 500
7. WRITE(B)
8. COMMIT
```

If the system crashes and the EUR 500 are lost, nobody is happy.

Transactions in PostgreSQL



Documentation → PostgreSQL 13
Supported Versions: Current (13) / 12 / 11 / 10 / 9.6
Development Versions: devel
Unsupported versions: 9.5 / 9.4 / 9.3 / 9.2 / 9.1 / 9.0 / 8.4 / 8.3 / 8.2 / 8.1 / 8.0 / 7.4 / 7.3 / 7.2

3.4. Transactions
Chapter 3. Advanced Features

3.4. Transactions

Transactions are a fundamental concept of all database systems. The essential point of a transaction is that it bundles multiple steps into a single, all-or-nothing operation. The intermediate states between the steps are not visible to other concurrent transactions, and if some failure occurs that prevents the transaction from completing, then none of the steps affect the database at all.

For example, consider a bank database that contains balances for various customer accounts, as well as total deposit balances for branches. Suppose that we want to record a payment of $100.00 from Alice's account to Bob's account. Simplifying outrageously, the SQL commands for this might look like:

```
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Alice');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Bob');
```

The details of these commands are not important here; the important point is that there are several separate updates involved to accomplish this rather simple operation. Our bank's officers will want to be assured that either all these updates happen, or none of them happen. It would certainly not do for a system failure to result in Bob receiving $100.00 that was not debited from Alice. Nor would Alice long remain a happy customer if she was debited without Bob being credited. We need a guarantee that if something goes wrong partway through the operation, none of the steps executed so far will take effect. Grouping the updates into a *transaction* gives us this guarantee. A transaction is said to be *atomic:* from the point of view of other transactions, it either happens completely or not at all.

We also want a guarantee that once a transaction is completed and acknowledged by the database system, it has indeed been permanently recorded and won't be lost even if a crash ensues shortly thereafter. For example, if we are recording a cash withdrawal by Bob, we do not want any chance that the debit to his account will disappear in a crash just after he walks out the bank door. A transactional database guarantees that all the updates made by a transaction are logged in permanent storage (i.e., on disk) before the transaction is reported complete.

Another important property of transactional databases is closely related to the notion of atomic updates: when multiple transactions are running concurrently, each one should not be able to see the incomplete changes made by others. For example, if one transaction is busy totalling all the branch balances, it would not do for it to include the debit from Alice's branch but not the credit to Bob's branch, nor vice versa. So transactions must be all-or-nothing not only in terms of their permanent effect on the database, but also in terms of their visibility as they happen. The updates made so far by an open transaction are invisible to other transactions until the transaction completes, whereupon all the updates become visible simultaneously.

In PostgreSQL, a transaction is set up by surrounding the SQL commands of the transaction with `BEGIN` and `COMMIT` commands. So our banking transaction would actually look like:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
-- etc etc
COMMIT;
```

Intermediate Course Assessment

- Announcement via Blackboard.
- Give anonymous feedback regarding the lecture (up to now).
- Give anonymous feedback regarding assignment 1.

The assessment helps us to improve the course and is very much appreciated!

After-Assignment 1 Meetings

- Available dates/times will be provided via Blackboard (groups).
- Choose a date/time that fits your schedule.
- **One member of your group must enroll** to this date via Blackboard.
- First come first serve.
- Grading will be based on the last submission before the meeting.

Assignment 2

Assignment 2

Summary:

- **Where:** Release and submission via Blackboard.
- **When:** April 19 – May 17, 2021 (resp. late: May 24, 2021).
- **What to do:** (a) Set up a document-based database locally (MongoDB; data will be provided), (b) get familiar with JSON and learn how to execute queries (most queries will be given), (c) write a small Python3 application that executes the queries, and (d) answer questions regarding the assignment.
- **What to submit:** The Python3 code and the answers to the questions.
- **Grading:** 55% Python3 code, 45% questions (incl. the meeting).

Intermediate Course Assessment

- Give anonymous feedback regarding the lecture (up to now).
- Give anonymous feedback regarding assignment 1.

The feedback helps us to improve the course and is very much appreciated!

Transaction Management

Programmers must ensure to properly define the transactions to preserve consistency.

A DBS typically includes a so-called **transaction manager**, which ensures that the transactions comply to the consistency guarantees (e.g., the ACID properties).

A transaction has **committed** if it completes successfully. Otherwise, the transaction is **aborted**. Undoing the changes of an aborted transaction is referred to as **rollback**.

Transaction Management – Atomicity

Before the DBS changes the database, it writes some information (transaction identifier, old/new values) about the changes into a so-called **log file.**

Example: Transfer EUR 500 from bank account A (EUR 800) to account B (EUR 2,000).

Transaction T_1:	Pseudo log file:	Database:
`1. BEGIN`		
`2. READ(A)`		
`3. A = A - 500`		
`4. WRITE(A)`	`T1: OLD(A=800), NEW(A=300)`	`UPDATE(A=300)`
`5. READ(B)`		
`6. B = B + 500`		
`---[ CRASH ]---`		`ROLLBACK T1 on restart`
`7. WRITE(B)`		
`8. COMMIT`		

Transaction Management – Isolation

Strict serial exeuction of concurrent transactions guarantees isolation, but severely limits the performance.

A **concurrency-control scheme** ensures that transactions can execute concurrently (i.e., their operations can be interleaved). Interleaving the operations of a transaction is called **schedule** and may result in a correct database state, or not!

A concurrent schedule is **serializable** if an equivalent serial schedule exists, i.e., the outcome of executing the transactions concurrently is the same as if they would have been executed serially.

Transaction Management – Isolation (Lock-Based)

Example: Transfer EUR 500 from bank account A (EUR 800) to account B (EUR 2,000).

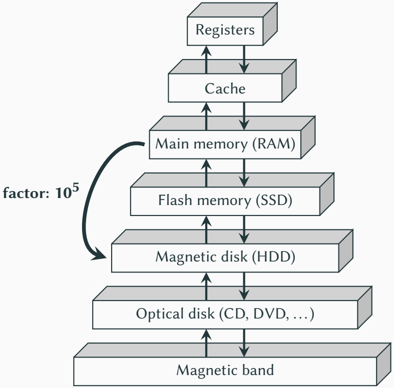
Transaction T_1:	Transaction T_2:	Database:	
1. BEGIN			
2. READ(A)		LOCK(A, T1)	ok
3. A = A - 500			
4. WRITE(A)		UPDATE(A=300)	
	1. BEGIN		
	2. READ(A)	LOCK(A, T2)	conflict
	3. A = A + 1,000		
	4. WRITE(A)		
	5. COMMIT		
5. READ(B)		LOCK(B, T1)	ok
6. B = B + 500			
7. WRITE(B)			
8. COMMIT		UNLOCK(A, T1), UNLOCK(B, T1)	ok

Transaction Management – Durability

Updating the persistent data may be postponed to improve performance, i.e., the data is provisionally updated in main memory (RAM). Data in main memory is volatile, i.e., it is lost on system restart.

Excursion: Nowadays, data can be considerd persistent if it is written to hard disk. DBMSs aim to be efficient/fast, thus they try to **avoid** or **postpone expensive/slow operations** like accesses to hard disk. Multiple data structures are maintained in main memory, which are commly referred to as **buffers.** The content of the buffers is **only written to disk if inevitable.**

Memory Hierarchy



< 1ns	**Brain**
> 1ns	**Room**
≈ 100ns	**City**
$\approx 100\mu s$	
≈ 10ms (10,000,000ns)	**Pluto**
> 1s	

Transaction Management – Durability

Example: Transfer EUR 500 from bank account A (EUR 800) to account B (EUR 2,000).

Transaction T_1:	Pseudo log file:	Database:
`1. BEGIN`		
`2. READ(A)`		
`3. A = A - 500`		
`4. WRITE(A)`	`T1: OLD(A=800), NEW(A=300)`	`postponed UPDATE(A=300)`
`5. READ(B)`		
`6. B = B + 500`		
`7. WRITE(B)`	`T1: OLD(B=2,000), NEW(B=2,500)`	`postponed UPDATE(B=2,500)`
`8. COMMIT`		
		`---[ CRASH ]---`

Distributed Transaction Management

A DDBMS must consider additional aspects:

- **Distributed data storage** (replication and fragmentation).
- **Distributed transactions** (local and global).

Distributed Data Storage

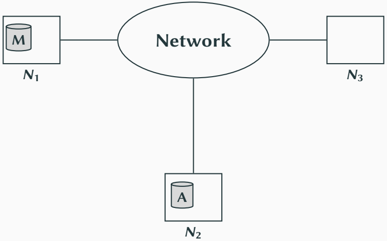
Assume the movies table of our example database and the relational data model.

Movies:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi
...	...	...	...

Actors:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather
...	...	...

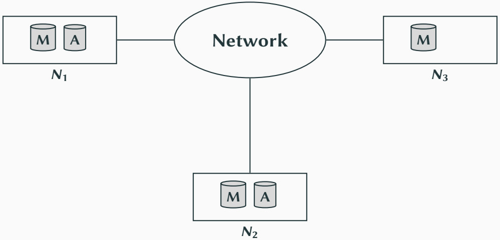


What if N_1 gets disconnected?

Distributed Data Storage – Replication

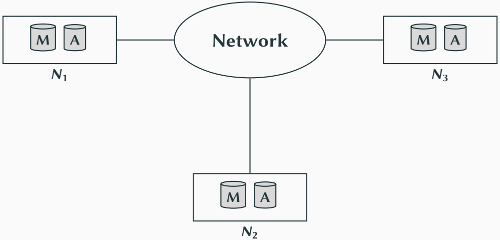
Replication: Data are **replicated** among multiple sites, i.e., a copy (replica) of the same data exists in different sites (intentional redundancy).

Example Replication:



Distributed Data Storage – Replication

Full replication: A copy of a relation is stored at **all sites**.

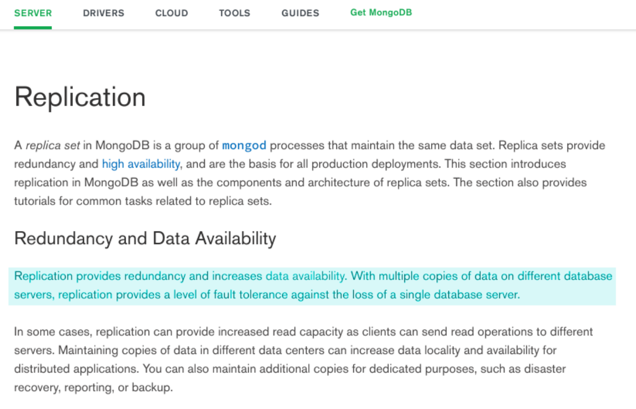


Fully replicated database: Every site contains a copy of the entire database.

Pros: Higher availability and performance, reduced data transfer.

Cons: Increased update costs and higher complexity of concurrency control.

Replication in MongoDB (Document-Based)



Replication

A *replica set* in MongoDB is a group of mongod processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments. This section introduces replication in MongoDB as well as the components and architecture of replica sets. The section also provides tutorials for common tasks related to replica sets.

Redundancy and Data Availability

Replication provides redundancy and increases data availability. With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.

In some cases, replication can provide increased read capacity as clients can send read operations to different servers. Maintaining copies of data in different data centers can increase data locality and availability for distributed applications. You can also maintain additional copies for dedicated purposes, such as disaster recovery, reporting, or backup.

Distributed Data Storage – Fragmentation

Fragmentation: Data are **partitioned into fragements** stored in distinct sites, i.e., a specific part of the data is stored in a site.

Horizontal fragmentation: Relation is split row- or tuple-wise, and each row/tuple resides in a separate site.

Vertical fragmentation: Relation is split into subschemata (based on the columns/attributes), and each subschema resides in a separate site.

Example Horizontal Fragmentation

Movies M1:

name	year	runtime	genre
The Godfather	1972	177	Mafia
The Avengers	2012	143	Sci-Fi

Movies M2:

name	year	runtime	genre
Batman - The Dark Knight	2008	152	Action
American Psycho	2000	101	Thriller

Movies M3:

name	year	runtime	genre
Home Alone	1990	103	Comedy

Actors A1:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather

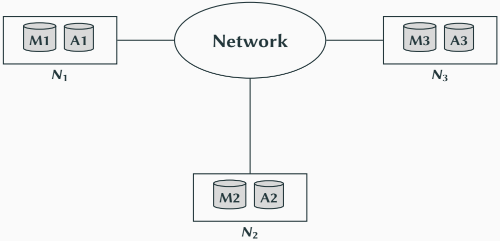
Actors A2:

name	birthyear	knownfor
Macaulay Culkin	1980	Home Alone
Joe Pesci	1943	Home Alone

Actors A3:

name	birthyear	knownfor
Robert Downey Jr.	1965	The Avengers
Mark Ruffalo	1967	The Avengers

Example Horizontal Fragmentation



Actors A1:

name	birthyear	knownfor
Marlon Brando	1924	The Godfather
Al Pacino	1940	The Godfather

Actors A2:

name	birthyear	knownfor
Macaulay Culkin	1980	Home Alone
Joe Pesci	1943	Home Alone

Actors A3:

name	birthyear	knownfor
Robert Downey Jr.	1965	The Avengers
Mark Ruffalo	1967	The Avengers

Distributed Data Storage – Fragmentation

Horizontal and vertical fragmentation can be **mixed.** In any case, the fragmented data must be **reconstructable.**

Pros: Higher performance (on fragments), better locality (reduced data transfer).
Cons: Increased costs on site failure and if data is retrieved from different sites.

Distributed Transactions

Local transactions access/update data at only one (local) site. A local transaction manager enforces the ACID properties.

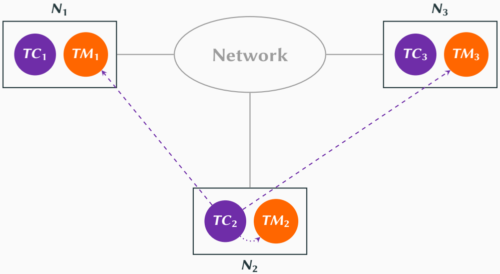
Global transactions access/update data at multiple (local) sites. Local subtransaction are executed at each site. Enforcing the ACID properties is much more complex.

Each site has a **local transaction manager** and a **transaction coordinator.**

Distributed Transactions

The **local transaction manager** ensures that the ACID properties hold for local transactions and maintains the local log files.

The **transaction coordinator** starts transactions that are initiated at a site, distributes the subtransactions to other sites, and ensures that a transaction either executes **at all sites or at none.**



Distributed Transactions – Commit Protocols

Required because we must ensure **atomicity across all sites**, e.g., we are not allowed to commit a transaction at N_1 but abort the same transaction at N_2.

List of Protocols:

- Two-Phase Commit (2PC; used in practice).
- Three-Phase Commit (3PC; solves issues of 2PC but impracticable).
- Persistent Messaging (PM).

Distributed Transactions – 2PC in a Nutshell

Transaction T is initiated at site N_i with transaction coordinator TC_i.

Phase 1: TC_i “asks” other participants to **prepare to commit** (and logs it beforehand). Each **transaction manager** determines if **it can commit**, logs it, and “reports” it to TC_i.

Phase 2: If TC_i receives a **single abort** message, then all participants are **informed to abort *T***. Otherwise, all participants are **notified to commit** T. Prior to that, TC_i logs the decision locally. The involved transaction managers comply to the decision.

Auto-Evaluation

- Structured Query Language (SQL).
- Integritätsbedingungen, valide Instanz.
- Datenabstraktion (physische Datenunabhängigkeit).
- Instanz vs. Schema, Primärschlüssel.
- Operationen in einem Key-Value Store.
- Durchsatz in einem parallelen DBMS (PDBMS).

Short Answers

- Warum deklarative Sprachen und nicht direkt Ausführungsplan?
- Prinzip hinter Datenunabhängigkeit. Warum erstrebenswert?
- Wünschenswertes Szenario für Redundanz. Warum?
- Prinzip hinter Transaktionen bzgl. des DB-Zustandes?
- Serieller vs. serialisierbarer Ablaufplan.
- Rolle des Loggings bzgl. ACID. Vorteil der Logs?

Ad MongoDB Replication in Action

How to **connect** to a MongoDB replica set **transparently?**

Consistency

Different notions of consistency to specify desired properties in a DDBMS. Ideally, all updates appear immediately at all sites in the same order (illusion of a single data copy).

Strong consistency refers to this ideal scenario, but this is often expensive (or even impracticable). **Weak consistency** relaxes the consistency constraints to improve the performance or the availability of a DDBMS.

Availability

High Availability: A DDBMS with extremely low downtime (about 99.99% available).

In large systems, a failure happens frequently (nodes may be down or the network may partition).

Trade consistency to achieve high availability.

Brewer's CAP Theorem

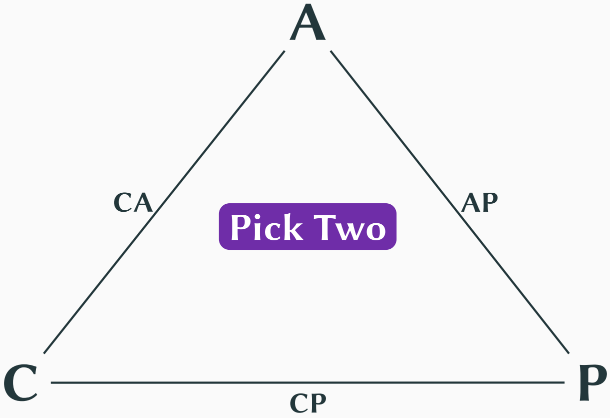
A distributed database system has three properties:

- **C**onsistency: All replicated copies are in the same state.
- **A**vailability: System runs even in case of failures due to replication.
- **P**artition-tolerance: System runs even if the network is partitioned.

Network partition: Network decomposes into multiple parts/subsystems that cannot reach one another.

CAP Theorem: You can have **at most two** of the three **properties.**

Brewer’s CAP Theorem



Brewer’s CAP Theorem

AP Systems: Systems stay available in case of a network partition, but inconsistencies may be introduced. These inconsistencies must be resolved once the network partition is resolved.

CP Systems: Systems maintain consistency in case of a network partition, but may be unavailable for some time.

CA Systems: “Ideal” consistency and availability. No availability and consistency guarantees in case of a network partition.

In Practice: Network partitions cannot be avoided $\Rightarrow$ AP or CP.

The BASE Properties

BASE is an **alternative consistency model** that is **not as strict** as the ACID properties and favors **availability over consistency.**

Basically Available: The system appears to **work most of the time**, i.e., reads/writes should be allowed even if the network partitions, but without consistency guarantee.

Soft State: The **state** of the database **may not be precisely defined** all the time, i.e., replicas do not have to be consistent.

Eventually Consistent: Once the network partitioning is resolved, the states of all replicas converge, i.e., all **replicas become consistent eventually.**

Eventual Consistency

No new updates ⇒ Writes are propagated to all replicas and all replicas converge towards a common state.

Inconsistent replicas must be identified because two replicas may be updated independently (e.g., version-vector scheme).

Inconsistent updates may need to be **merged** (e.g., in the worst case, human interaction is required – comparable to a merge conflict in `git`).

Workloads

With regard to database systems, a **workload** is a set of queries/updates that reflects a **typical usage pattern (load).**

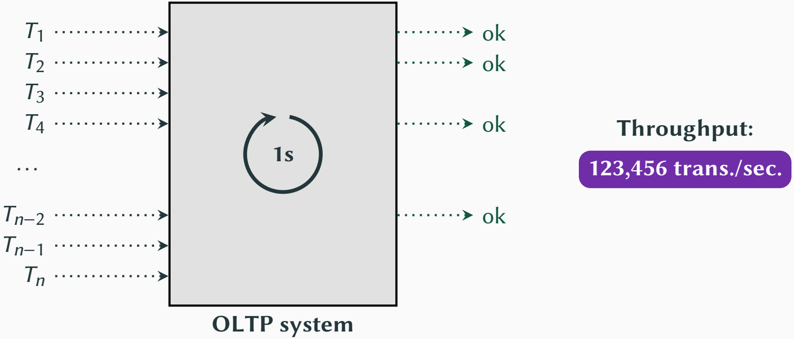
Different database systems perform better/worse on particular workloads.

Transaction Processing Performance Council (TPC): An independent consortium that releases standardized benchmarks for various workloads, the TPC benchmarks [10].

[10] `http://www.tpc.org/`

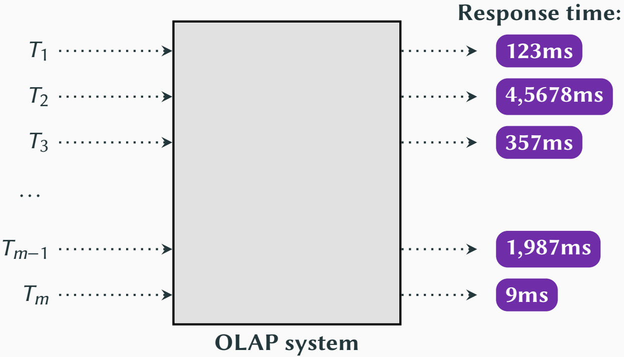
OLTP Workloads

Online Transaction Processing (OLTP): Short-lived read/write transactions with a small footprint (i.e., only a small portion of the data is touched). **Many transactions** must be processed **as fast as possible** (high throughput, low response time). The TPC-C benchmark provides typical OLTP workloads.



OLAP Workloads

Online Analytical Processing (OLAP): Long-running read-only queries that exploratory analyze a large portion of the overall data (to support decision). This often involves complex join operations and the main focus is **low response time.** The TPC-DS benchmark provides typical OLAP workloads. “Data Warehouse”.



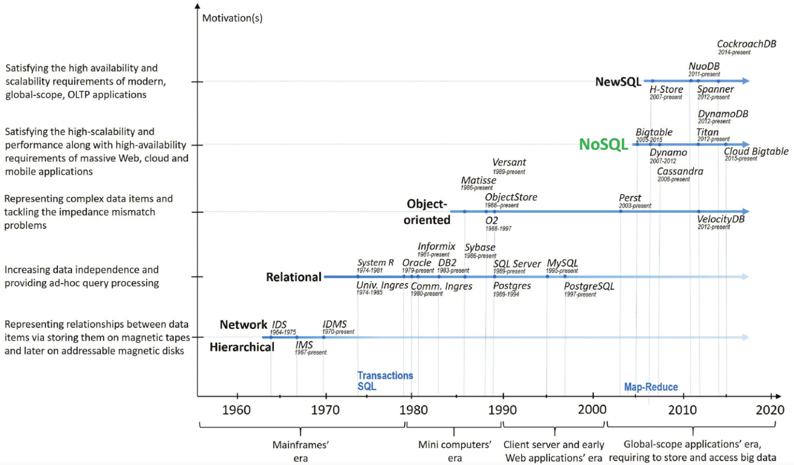
Workloads – Batches vs. Streams

Batch Workloads: A **batch** is a large but **bounded static dataset.** Before data can be processed, all data must be completely available (e.g., on hard disk).

Stream Workloads: A **stream** is an **unbounded evolving dataset.** Data items are processed as they stream into the system one after another, i.e., the data does not have to be completely available.

Systems Potpourri

Continuous Development


Motivation(s)
Satisfying the high availability and scalability requirements of modern, global-scope, OLTP applications
Satisfying the high-scalability and performance along with high-availability requirements of massive Web, cloud and mobile applications
Representing complex data items and tackling the impedance mismatch problems
Increasing data independence and providing ad-hoc query processing
Representing relationships between data items via storing them on magnetic tapes and later on addressable magnetic disks
CockroachDB 2014-present
NuoDB 2011-present
NewSQL
H-Store 2007-present
Spanner 2012-present
DynamoDB 2012-present
Bigtable 2005-2015
Titan 2012-present
NoSQL
Dynamo 2007-2012
Cloud Bigtable 2015-present
Cassandra 2008-present
Versant 1989-present
Matisse 1986-present
ObjectStore 1988--present
Perst 2003-present
Object-oriented
O2 1988-1997
VelocityDB 2012-present
Informix 1981-present
Sybase 1986-present
System R 1974-1981
Oracle 1979-present
DB2 1983-present
SQL Server 1989-present
MySQL 1995-present
Relational
Univ. Ingres 1974-1985
Comm. Ingres 1980-present
Postgres 1989-1994
PostgreSQL 1997-present
Network
IDS 1964-1975
IDMS 1970-present
Hierarchical
IMS 1967-present
Transactions
SQL
Map-Reduce
1960
1970
1980
1990
2000
2010
2020
Mainframes' era
Mini computers' era
Client server and early Web applications' era
Global-scope applications' era, requiring to store and access big data

Picture taken from Davoudian et al. *A Survey on NoSQL Stores.* ACM Computing Surveys, 2018.

Continuous Development

In-Memory Database Systems: Relies on **main memory (RAM) for storing the data** rather than hard disks (HDDs) or SSDs. Very **low response times (microseconds)** but must deal with durability (logs).

Database as a Service (DBaaS): A cloud-based platform (service) that provides computing infrastructure, data storage and **database functionality in the cloud.** The clients (you) do not have to set up their own database system on their own hardware, but just **access and use a database that runs in the cloud.**

Key-Value Stores – Redis

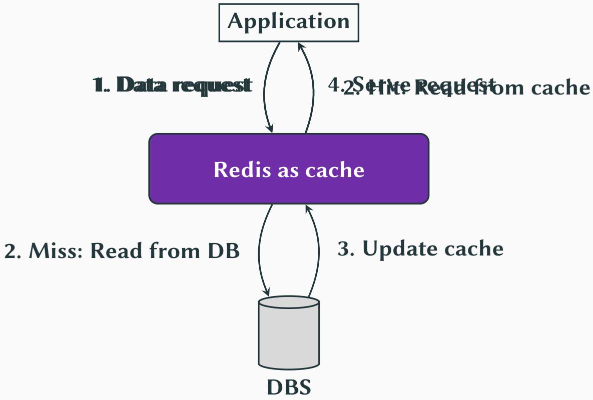
The **RE**mote **DI**ctionary **S**erver [11] is an **in-memory** NoSQL database system based on the **key-value data model** that **supports durability** (i.e., data can be made persistent).

- Redis is a CP system.
- In-memory, i.e., very fast (avg. read/write performance: < 1ms).
- Collection of useful, high-performance data structures (lists, sets, bitmaps, …).
- Replication and persistence support.
- Supports many programming languages incl. Java, Python, C/C++, and JavaScript.

[11] `https://redis.io`

Key-Value Stores – Redis

Use Case: **Cache** between application and database system.



Key-Value Stores – Other Systems

Riak KV [12] is a distributed **persistent** key-value store with support for **advanced data types** (e.g., JSON). It is inspired by Amazon's DynamoDB [13] and provides **high availability** (i.e., is an AP system).

Memcached [14] ("Mem-Cash-Dee") is a distributed **in-memory** key-value store with a performance similar to Redis. However, Memcached supports only the **simple key-value data model** and has **no support for durability.** Not a database system!

[12] https://riak.com/products/riak-kv/

[13] https://aws.amazon.com/dynamodb/

[14] https://memcached.org/

Document Stores – MongoDB

MongoDB [15] is a distributed NoSQL database system based on the document-oriented data model with a focus on high availability and horizontal scalability.

- MongoDB is a CP system.
- Uses JSON as document format (specifically, a binary JSON format called BSON).
- Shared-nothing system architecture.
- JSON-based query language (limited support for joins as we know it).
- Replication and sharding (aka fragmentation) support through replica sets.

[15] https://www.mongodb.com/

Document Stores – Other Systems

Apache CouchDB [16] is a distributed document store with support for **master-master replication** and a conflict resolution protocol. It is an AP system that uses the plain JSON format.

Couchbase [17] is a distributed document store with a built-in in-memory cache (memcached) and an SQL-like query language (N1QL). It uses CouchDB as back end.

[16] https://couchdb.apache.org/

[17] https://www.couchbase.com/

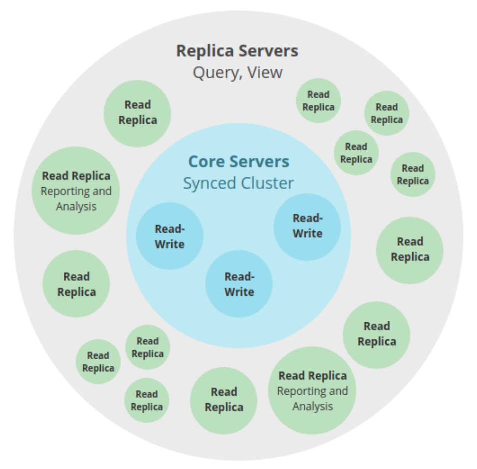
Graph Stores – Neo4J

Neo4j [19] is a NoSQL database system based on the graph data model with ACID guarantees. Neo4j is a **native** graph store.

- Neo4j is a CP system (as of version 3.5).
- Shared-nothing system architecture.
- Set of core servers (serve write+read) and many read replicas.
- Emphasis on read availability (full replication).
- SQL-inspired query language (Cypher) to describe graph patterns (ASCII art).

[19] `https://neo4j.com/`

Graph Stores – Neo4j



https://neo4j.com/docs/operations-manual/current/clustering/introduction/.

Graph Stores – Other Systems

Tigergraph [20] is a distributed native graph store that supports parallel computation and advanced analytics. It has its own query language GSQL.

ArangoDB [21] is a multi-model database system that supports the graph-based model in JSON format.

OrientDB [22] is another multi-model database system that supports the graph-based model. It uses an SQL-like syntax that is extended for graphs.

[20] https://www.tigergraph.com/

[21] https://www.arangodb.com/

[22] https://orientdb.org/

Extensible Record Store – Apache Cassandra

Apache Cassandra [23] is a distributed NoSQL database system based on the extensible record (or wide column) data model. It is a decentralized and "master-less" DDBMS that supports eventual consistency.

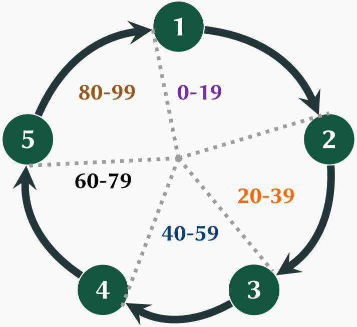
- Apache Cassandra is an AP system (but can be configured as CP system).
- Shared-nothing system architecture.
- Support for nested column families (so-called super column families).
- The Cassandra query language (CQL) resembles SQL.

[23] https://cassandra.apache.org/

Extensible Record Store – Apache Cassandra

Master-less architecture ⇒ **No single point of failure.**

Consistent Hashing (Token Ring)



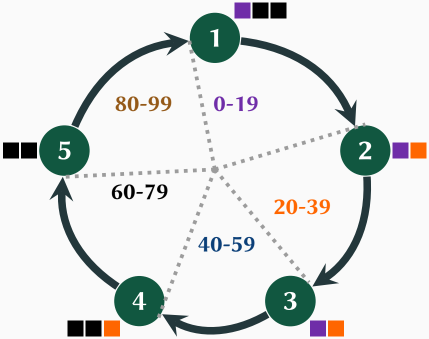
Hash function $h(.)$ determines node.

Data			$h(.)$	Node
Christian	Bale	1974	18	1
Scarlett	Johansson	1984	21	2
Mark	Ruffalo	1967	67	4
Mark	Wahlberg	1971	67	4

Extensible Record Store – Apache Cassandra

Master-less architecture ⟹ **No single point of failure.**

Replication Factor (for example, 3)



Hash function $h(.)$ determines node.

Data			$h(.)$	**Node**
Christian	Bale	1974	18	**1**
Scarlett	Johansson	1984	21	**2**
Mark	Ruffalo	1967	67	**4**
Mark	Wahlberg	1971	67	**4**

Other Systems

- **RDBMS:** PostgreSQL [24], SQLite [25], Oracle [26], Microsoft SQL Server [27], …
- **NewSQL Systems:** CockroachDB [28], VoltDB [29], …
- **In-Memory OTLP & OLAP Systems:** HyPer [30], SAP Hana [31], …
- **Cloud Services:** Amazon DynamoDB [32], Snowflake Cloud Data Warehouse [33], …

[24] https://www.postgresql.org/

[25] https://www.sqlite.org/

[26] https://www.oracle.com/database/

[27] https://www.microsoft.com/en-us/sql-server/sql-server-2019

[28] https://www.cockroachlabs.com/

[29] https://www.voltdb.com/

[30] https://hyper-db.de/

[31] https://www.sap.com/products/hana.html

[32] https://aws.amazon.com/dynamodb/

[33] https://www.snowflake.com/

Additional Material

Use the **database of databases** [34] for a first impression and cross references.

A 37-page **survey on NoSQL database systems** [35]

Books on NoSQL database systems [36] [37]

[34] `https://dbdb.io/`

[35] Davoudian et al. *A Survey on NoSQL Stores.* ACM Computing Surveys, 2018.

[36] Sadalage and Fowler. *NoSQL Distilled – A Brief Guide to the Emerging World of Polyglot Persistence.* Addison-Wesley, 2013.

[37] Redmond and Wilson. *7 Databases in 7 Weeks - A Guide to Modern Databases and the NoSQL Movement.* Pragmatic Bookshelf, 2012.