Cloud Computing (INFS3208)

Lecture 6: Databases in Cloud Computing

Lecturer: AsPr Sen Wang

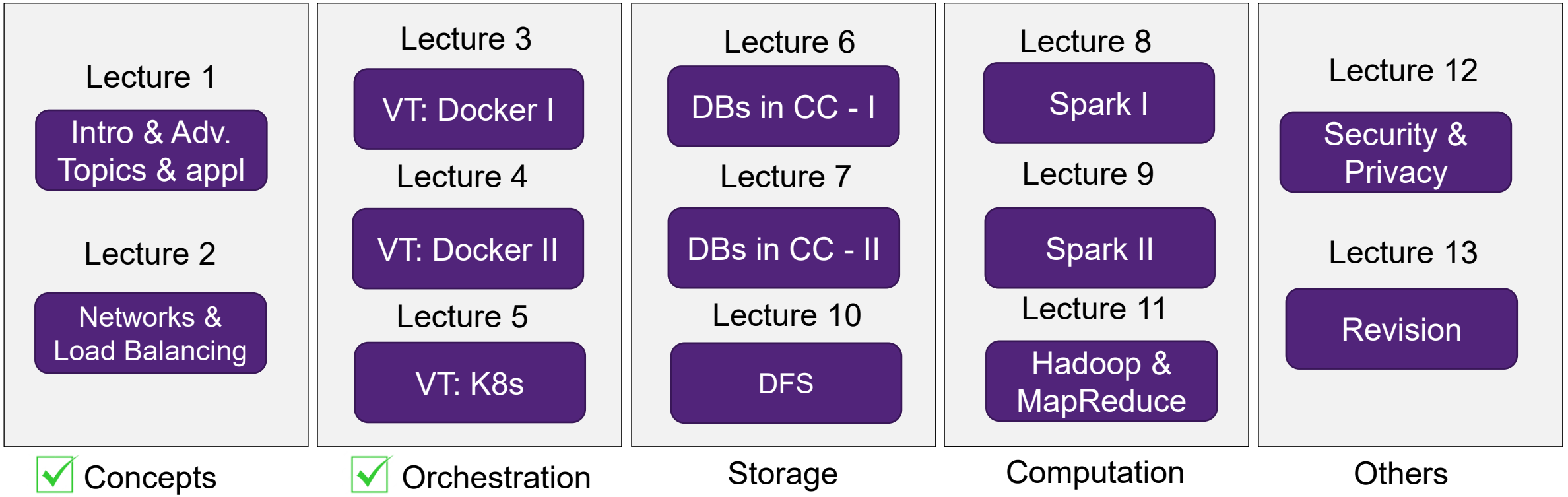
School of Electrical Engineering and Computer Science

Faculty of Engineering, Architecture and Information Technology

The University of Queensland

Course Overview – Lectures

This course includes 13 lectures and 10 tutorial/practical sessions



- Programming Task I is completed and Task II (due on 19 Sept) was released last Friday.
- Individual Project (due on 24 Oct) will be released by this Friday. Discuss it with Tutors.

Outline

- Database Background
- Relational Databases (refresh)
  - Revisit Relational DBs
  - ACID Properties
  - Clustered RDBMs
- Non-relational Databases
  - NoSQL concepts & types
  - Database Partitioning, Hashing in NoSQL, CAP Theorem, BASE
  - MongoDB
  - Cassandra
  - HBase

What is a Database?

"***A set of information held in a computer***"

Oxford English Dictionary

"***One or more large structured sets of persistent data, usually associated with software to update and query the data***"

Free On-Line Dictionary of Computing

"***A collection of data arranged for ease and speed of search and retrieval***"

Dictionary.com

A database is a shared, integrated **computer structure** that stores a collection of the following:

- End-user data—that is, raw facts of interest to the end user
- Metadata, or data about data, through which the end-user data is integrated and managed

1.Coronel, Carlos, and Steven Morris. *Database systems: design, implementation, & management*. Cengage Learning, 2016.

Why Database Systems?

Program data independence

- Easily change the structure of the database without modifying the application program.

Sharing of data, controlled redundancy

- Data can be shared by authorized users.
- Data is recorded in only one place in the database and it is not duplicated.

Concurrent, multi-user access to large volumes of data

- Many authorized users can simultaneously access the same piece of information.
- The remote users can also access the same data.
- Concurrent access to data in the file system leads to incorrect data.

Back-up and recovery for reliability

- Most of the DBMSs provide the 'backup and recovery' sub-systems that automatically create the backup of data and restore data if required.
- In a computer file-based system, the backup and recovery are often inefficient.

Why Database Systems?

Uniform, logical data model for representing data

- The logical data model represents the conceptual data model (including attributes, names, relationships, and other metadata). Logical data model is often developed by UML.

Rich, standard language for querying data

- A number of extended SQL language, such as PLSQL (Oracle), PL/pgSQL (PostgreSQL), Transact-SQL or T-SQL (Microsoft SQL Server)

Effective optimizations for efficient querying

- DBMS queries can be optimized to achieve efficiency

Ensure data integrity within single applications

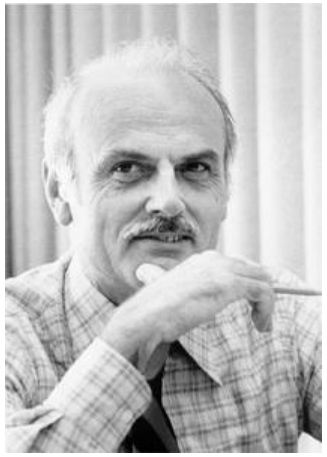
- Integrity constraints or consistency rules can be applied to database so that the correct data can be entered into database.

Relational Database

A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.

In [1], relation algebra for databases was proposed and formed the foundation of modern relational databases.

Five primitive operators in [1] are selection, projection, Cartesian product, set union, and set difference.



SQL
Structured **Q**uery **L**anguage

Turing Award in 1981

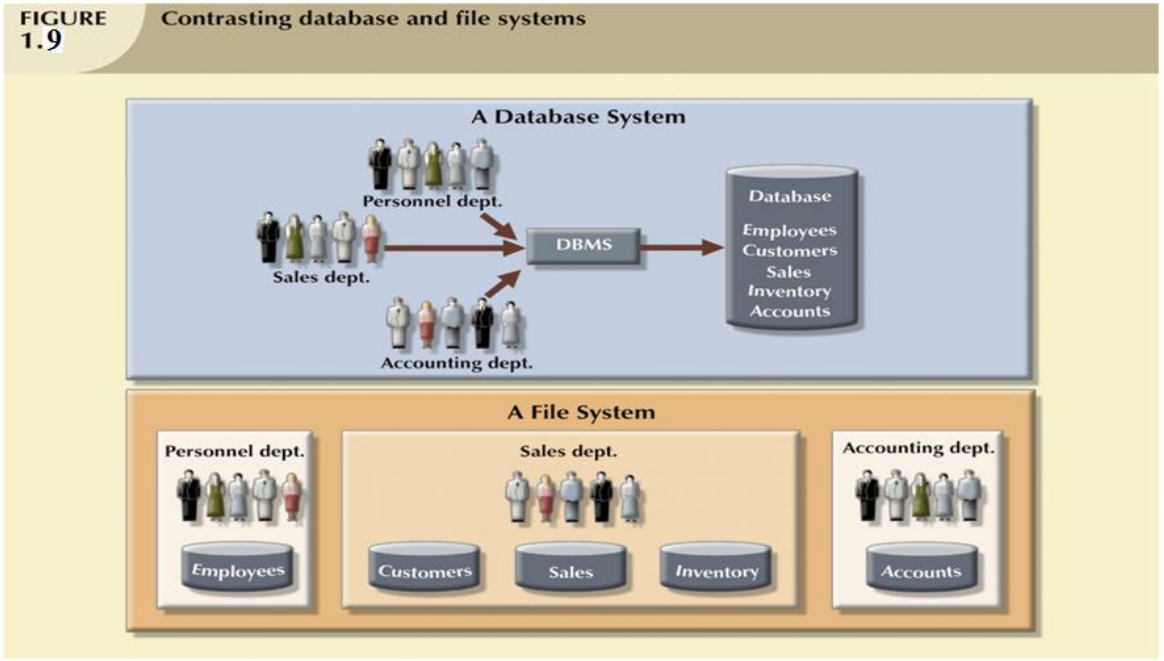
The relational database was invented by E. F. Codd at IBM in 1970.

[1] Codd, E. F. (1970). "Relational Completeness of Data Base Sublanguages"

Key Concepts in Relational Database (1/9)

DBMS

- The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyse the data.
- The DBMS software additionally includes the core facilities provided to administer of the database.
- Often the term "database" is also used to loosely refer to any of the DBMS, the database system or an application associated with the database.



Key Concepts in Relational Database (2/9)

Some important database terms:

Entities / Tables - Entities represent items that we want to store data about (e.g., a student)

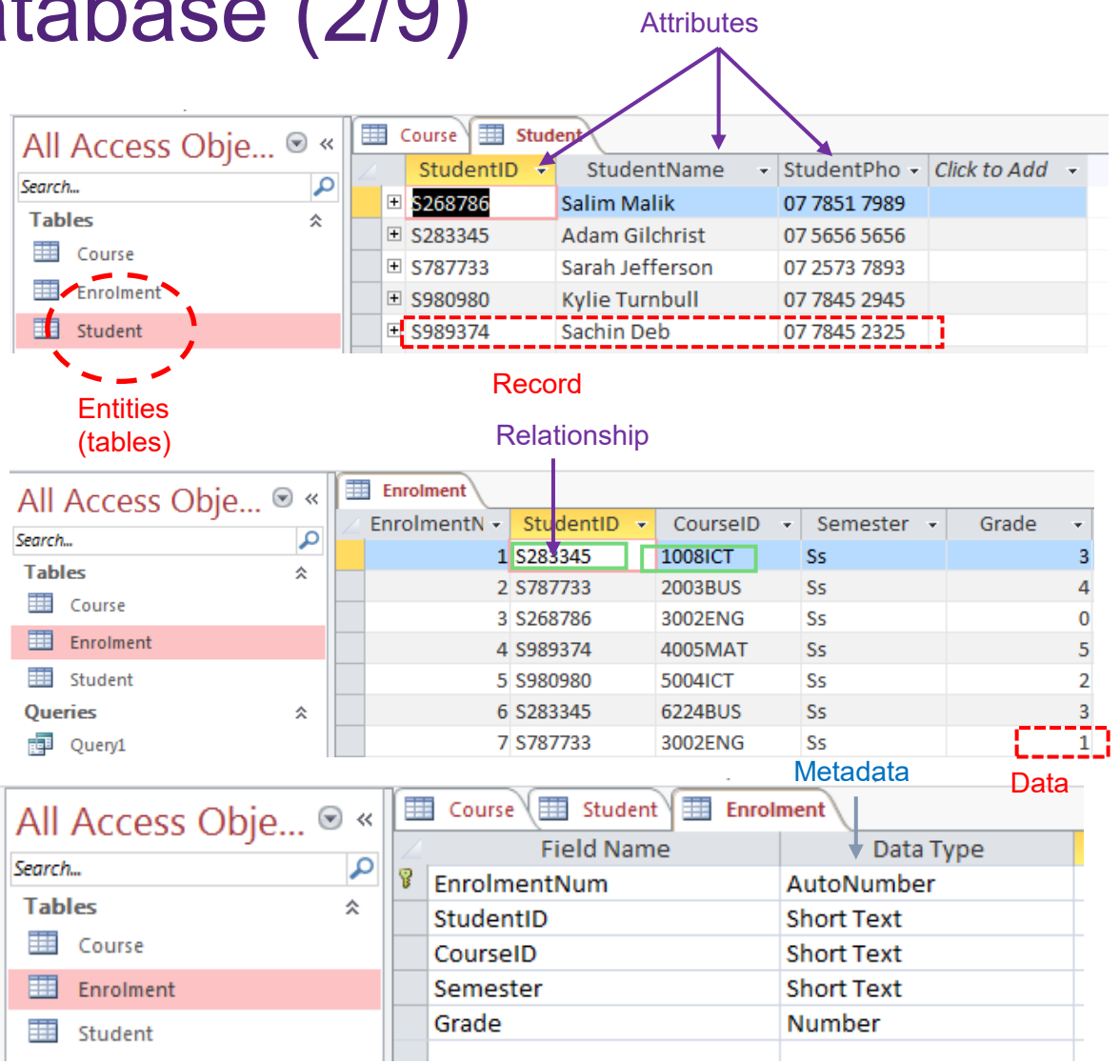
Attributes / Fields / Columns headings - attributes are the pieces of data that we want to store (e.g., the students name)

Relationships - relationships are used to show how entities within the database are related (e.g., a student may be enrolled in a course, so keep student ID in Enrolment table).

Records / Rows – a logically connected one or more fields, e.g., a student record consisting of name, student number & phone

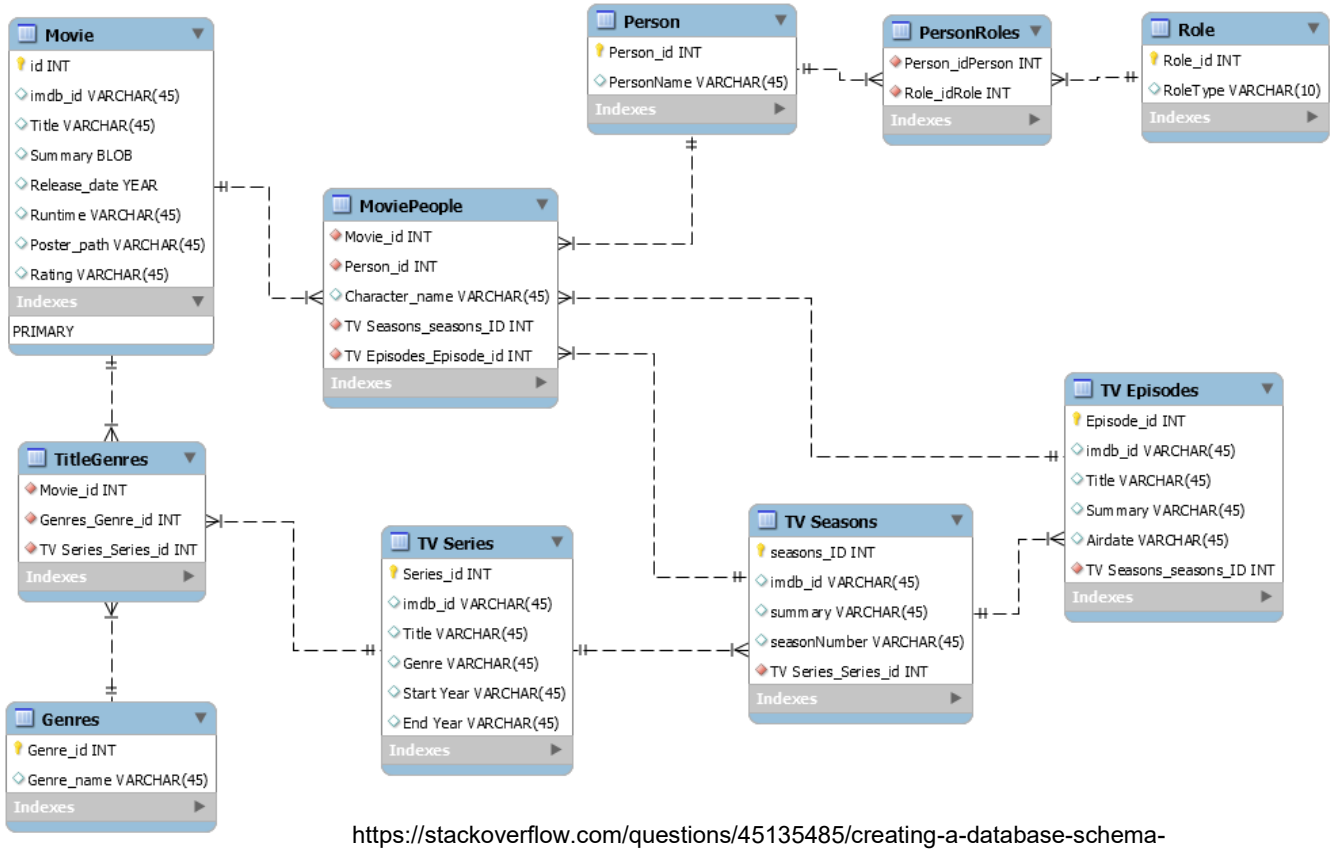
Data / Data item – raw fact, e.g., student grade or phone number.

Metadata – data about data that provides description of data to enable program–data independence, e.g., type of data (number or text)



What is a Database Schema? (3/9)

- The database schema of a database system is its structure described in a formal language supported by the database management system (DBMS).
- The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases).



https://stackoverflow.com/questions/45135485/creating-a-database-schema-from-the-movie-database

https://en.wikipedia.org/wiki/Database_schema

Types of SQL Statements (4/9)

Data **D**efinition **L**anguage (**DDL**) - Defines and modifies a schema e.g. CREATE / DROP / ALTER **table** does not manipulate data

Data **M**anipulation **L**anguage (**DML**) - Language used to retrieve (SELECT), add (INSERT), modify (UPDATE) and DELETE **data**

Data **C**ontrol **L**anguage (**DCL**) statements. Used for providing (GRANT) / withdrawing (REVOKE) **access privileges**

Transaction **C**ontrol **L**anguage (**TCL**) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into **logical transactions**. Example: COMMIT, ROLLBACK, etc.

Key Concepts in Relational Database (5/9)

No duplicated tuples (rows)

- by definition sets do not contain duplicate elements
  - hence tuples (rows) are unique
  - Ensured by entity integrity (primary key)

Tuples are unordered within a relation (table)

- by definition sets are not ordered
  - hence tuples can only be accessed by content

No ordering of attributes within a tuple

- by definition sets are not ordered

Customer

	custno	custname	custadd	credlimit
t1	SMI13	SMITH	Wide Rd, OnePlace, 1111	2000
t2	JON44	JONES	Narrow St, Somewhere 2222	10000
t3	BRO23	BROWN	Here Rd, Lost, 3333	10000

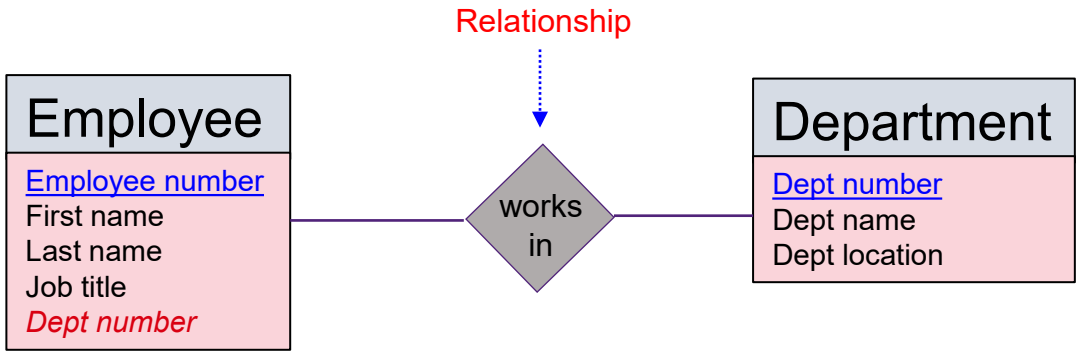
Relation heading

Relation body

Key Concepts in Relational Database (6/9)

Entity-Relation Design (ERD):

- Entity: corresponds to a table in which we store data about a particular thing, e.g., Student
- Attribute: describes characteristics of an entity, e.g. attributes for the Employee entity are employee number, first name, last name, job title etc.
- Relationship: illustrates an association (business rule) between two entities, e.g. a verb



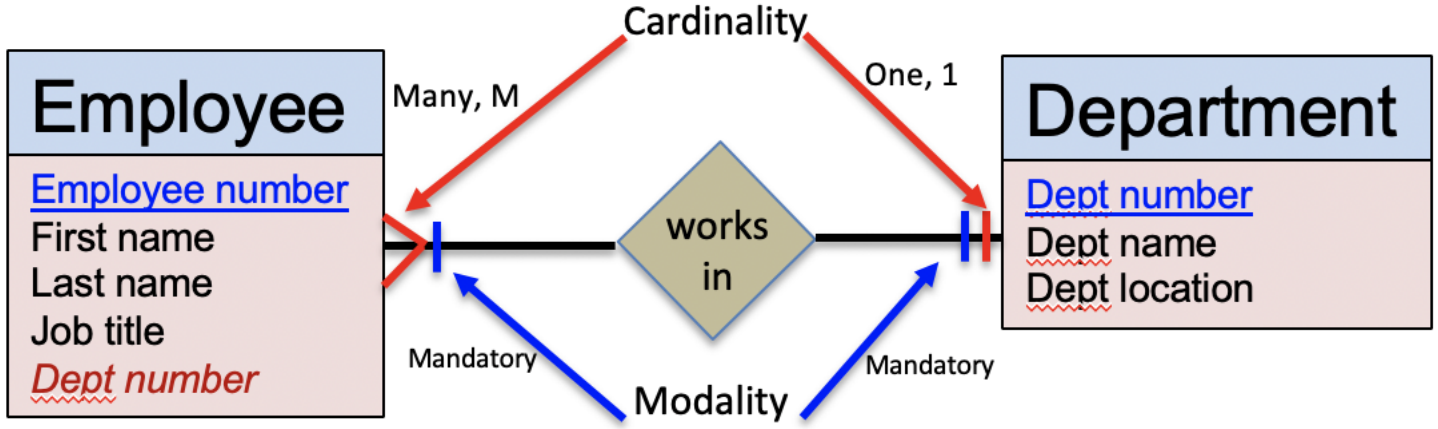
Key Concepts in Relational Database (7/9)

What is an actual business rule between Employee and Department entities?

- An employee works in a department and
- A department can employ many employees.

Cardinality/connectivity specifies maximum number of times an instance of an entity can be related to instances of a related entity (1:1, 1:M, and M:N).

Modality/participation specifies minimum number of times an instance of an entity can be related to instances of a related entity (0 or 1).



Key Concepts in Relational Database (8/9)

NORMALISATION - SIMPLY 'COMMON SENSE'

- Converts a relation into relations of progressively smaller number of attributes and tuples until an optimum level of decomposition is reached - little or no data redundancy exists
- Normalisation is a Relational Database Implementation Model focused approach (it makes extensive use of FK's to connect relations)

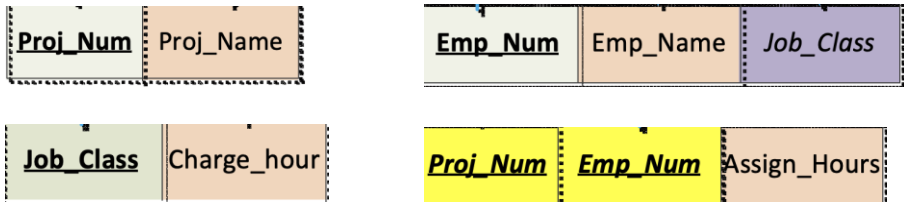
Goals:

- Each table represents a single subject
- No data item will be unnecessarily stored in more than one table, i.e., No data redundancy
- All non-key attributes in a table are dependent on the primary key
- Each table is void of insertion, update, deletion anomalies
- Objective of normalisation is to ensure that all tables are in at least 3NF

FIGURE 6.2 A TABLE IN FIRST NORMAL FORM

Table name: DATA_ORG_1NF **Database name: Ch06_ConstructCo**

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
15	Evergreen	101	John G. News	Database Designer	105.00	19.4
15	Evergreen	105	Alice K. Johnson *	Database Designer	105.00	35.7
15	Evergreen	106	William Smithfield	Programmer	35.75	12.6
15	Evergreen	102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
18	Amber Wave	118	James J. Frommer	General Support	18.36	45.3
18	Amber Wave	104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	48.4
22	Rolling Tide	113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	26.87	22.0
22	Rolling Tide	106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.8
25	Starflight	101	John G. News *	Database Designer	105.00	56.3
25	Starflight	114	Annelise Jones	Applications Designer	48.10	33.1
25	Starflight	108	Ralph B. Washington	Systems Analyst	96.75	23.6
25	Starflight	118	James J. Frommer	General Support	18.36	30.5
25	Starflight	112	Darlene M. Smithson	DSS Analyst	45.95	41.4



No Redundancy!

Key Concepts in Relational Database (9/9)

Transaction (ACID) Properties

- Atomicity - all parts of a transaction be completed successfully otherwise, the transaction is aborted (never partially executed, Done or not done).
- Consistency - concurrent execution of transactions yields consistent results (consistent -> consistent, inconsistent -> roll back -> consistent). Certain predefined rules or integrity constraints (e.g., unique primary keys, foreign key constraints) are maintained **before** and **after** the transaction.
- Isolation - data used during one transaction cannot be used by a second until the first is completed- Multi-user
- Durability – ensures that the result or effect of a committed transaction persists in case of a system failure.

All transaction properties work together to make sure that a database maintains data integrity and consistency for a single-user or a multi-user DBMS.

Examples of ACID (9/9)

Banking Example: transferring AUD btw your **Savings** and **Daily** accounts.

Property	Scenario	Without Property	With Property
Atomicity	Step 1: Savings: - $50 the system crashes Step 2: Daily: + $50	$50 would disappear. It's gone from Savings but never appeared in Daily.	DB knows the transaction didn't complete. It automatically **rolls back** the first step, returning the $50 to your Savings account.
Consistency	"An account balance **cannot be negative**." Savings: $30 - $50.	A balance of -$20 - breaks the bank's rule and puts the data in an invalid, or "inconsistent," state.	DB checks the transaction against its rules (CONSTRAINTS). It sees that the transfer would violate the "no negative balance" rule, so it rejects the entire transaction from the start. Your balances remain unchanged and valid.

Examples of ACID (9/9)

All applications need to be ACID?

Banking Example: transferring AUD btw your **Savings** and **Daily** accounts.

Property	Scenario	Without Property	With Property
Isolation	Daily: $100 Two Ops At the exact same moment: - Transfer $80 to Savings. - Pay $90 to someone.	Balance check might happen ***after*** the transfer has started but ***before*** it's finished. It might read the old value of $100. Thinking you have enough money in Daily account, you make the $90 purchase. Meanwhile, the $80 transfer completes. The database gets confused and the final balance could be wrong. This is called a "race condition."	DB "isolates" the two operations. It will force one to wait until the other is completely finished. For example, the balance check will be forced to wait until the $80 transfer is 100% complete. Only then will it read the new, correct balance of $20 and correctly tell you that you can't afford the $90 purchase.
Durability	Transfer of $50 from Savings to Daily completes successfully. One second later, the entire bank's server loses power and reboots.	When the server **restarts**, the record of your transaction might be gone. The $50 could be back in your Savings account, as if the transfer never happened.	As soon as the transaction was completed, the database wrote the result to a permanent log (like on a hard drive). When the server reboots, it will check this log and ensure all completed transactions are applied. Your $50 will be safe in your Daily account

Clustered RDBMs

- RDBMS is famous for ACID and suitable for transaction workloads, but scalability of RDBMs is an issue.
- To horizontally scale up and support ACID transactions, traditional RDBMs have been extended to maintain high-availability, high-redundancy in a distributed computing environment.
- Products:
  - MySQL NDB Cluster 8.0
  - CockroachDB
  - Amazon Aurora
- Pros: high availability (99.99%+), high throughput and low latency, etc.
- Cons: complex management/deployment/configuration, no foreign key, need huge memory and disk storage.

Pros and Cons of ACID

Pros:

- **Reliability**: Ensures that the database remains in a consistent state, even after unexpected failures.
  - Example: If a power outage occurs during a bank transfer, the system ensures that the money is neither lost from the source account nor duplicated into the destination account.
- **Consistency**: All transactions maintain the database's integrity, ensuring that business rules and constraints are not violated.
  - Example: A library system won't allow lending more books to a user who has reached their borrowing limit.
- **Isolation Guarantees**: Concurrent transactions don't interfere with each other, ensuring that the final result is the same as if the transactions were executed serially.
  - Example: Two customers buying the last piece of an item at the same time will not both succeed in the purchase; the system will handle the transactions in such a way that only one succeeds.
- **Durability**: Once committed, changes made by a transaction are permanent, ensuring that data is not lost.
  - Example: If you've booked a ticket online and received a confirmation, a subsequent system crash will not void your booking.

Pros and Cons of ACID

Question:
Is transaction consistency a MUST for ALL applications?

Cons:

- **Performance Overhead**: Ensuring ACID properties can introduce overheads and thus affecting system performance, especially **in high-transaction environments**.
  - **Example 1**: In e-commerce platforms with **millions of concurrent users**, ensuring isolation might slow down the system due to the need to lock resources.
  - **Example 2**: if a user changes his or her profile picture, consistency ensures that all instances of the old picture across all posts, comments, etc., are **UPDATED IMMEDIATELY**. This might involve multiple reads and writes in a short period of time.
  - Given an application that **needs the high volume of interactions**, the overheads can be magnified:
    - **Database Locks**: Ensuring isolation can lead to database locking, which can slow down operations when many users are trying to access the same data simultaneously.
    - **Database Writes**: The durability property can lead to frequent database writes, affecting write throughput, especially when data is written to multiple locations or backups.
    - **Transaction Management**: Handling atomicity and rolling back transactions in case of failures can introduce additional overhead, especially when transactions involve multiple steps.

Pros and Cons of ACID

Question:
How to achieve ACID in a distributed environment?

Cons:

- **Scalability Concerns:** ACID properties can make it challenging to scale out a system as it grows, especially in distributed database environments.
  - **Example 1**: Distributed RDBMS that need to maintain ACID properties might experience latency as they coordinate between multiple nodes to ensure data consistency.
  - **Example 2**: Ensuring durability means writing data to multiple storage systems, backups, rollback, or even geographically distributed data centers. This can increase the I/O operations and network traffic, impacting the speed and efficiency of write operations.
  - Given an application that **distributes databases for scalability or reliability**, impacts can be:
    - **Network Overhead:** Synchronizing data across distributed systems to maintain ACID properties can significantly increase network overhead, affecting response times.
    - **Storage Overhead:** Durability, especially when combined with distributed storage for redundancy, can require a lot of storage infrastructure, leading to increased costs.
    - **Operational Complexity:** Coordinating transactions across numerous systems, handling rollbacks, managing locks, and ensuring data synchronization can make system management more complex.

Outline

- Database Background
- Relational Databases
  - Revisit Relational DBs
  - ACID Properties
  - Clustered RDBMs
- Non-relational Databases
  - NoSQL concepts & types
  - Database Partitioning, Hashing in NoSQL, CAP Theorem, BASE
  - MongoDB
  - Cassandra
  - HBase

What is NoSQL?

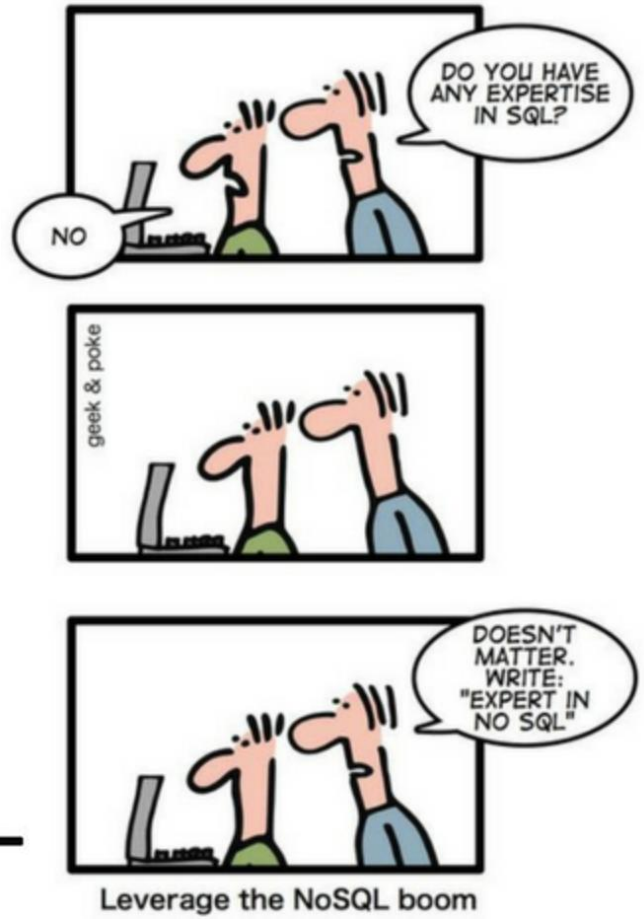
A class of database management systems (DBMS)

NoSQL - "Not only SQL"

- Does not use SQL as querying language
- Distributed, fault-tolerant architecture
- No fixed schema (formally described structure)
- No **joins** (typical in databases operated with SQL)

It's not a replacement for RDBMS but compliments

No Relational Query Languages



http://www.gradpost.ucsb.edu/events/events-article/2019/07/09/default-calendar/academic-job-market-writing-success-series-c.v.s

Why NoSQL?

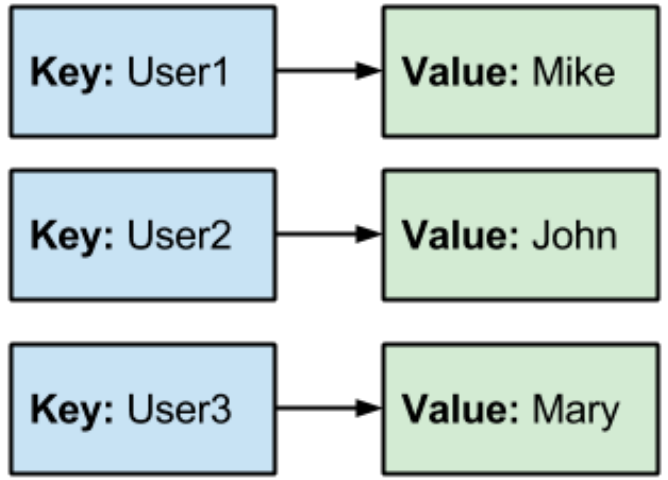
Modern Web applications:

- Support large numbers of concurrent users (tens of thousands, perhaps millions)
  - 527M users online shopping on Taobao on Singles' Day (11.11.22)
- Deliver highly responsive experiences to a globally distributed base of users
  - Need scalability (horizontally scalable)
- Be always available – no downtime
  - Barely see downtime/maintenance for Taobao, YouTube or Amazon
- Handle semi- and unstructured data
  - Twitter tweets / Youtube videos / Instagram photos
- Rapidly adapt to changing requirements with frequent updates and new features
  - Cloud environment & cluster deployment
- The world has gone mobile, but still need data storage.

Types of NoSQL Databases

Key-value stores

- Key is a string, e.g. “s123456”, “user109029”, or “1922887298749 4923”
- Value can be in different types: Double, Int, String, etc.
- Usage cases: Frequent I/O operations in simple data model
  - mobile apps, shopping carts, etc.
- Pros and Cons:
  - Scalable, flexible, high performance for writing
  - Not suitable for structured data and low query performance for conditions
- Representative Products:
  - Redis, Riak, Memcached, etc.

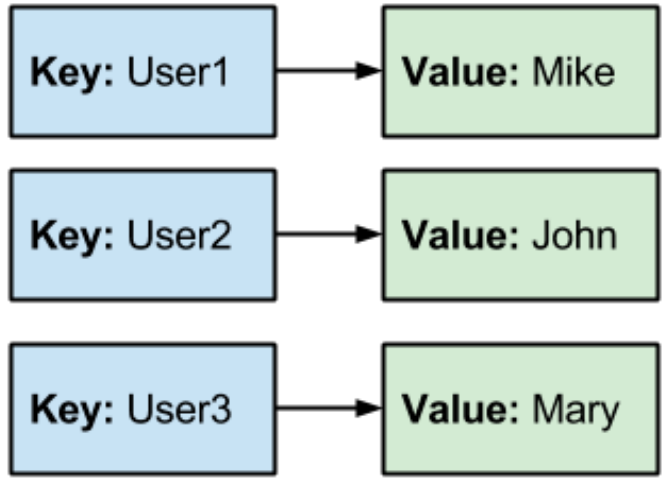


https://www.kdnuggets.com/2016/06/top-nosql-database-engines.html

Types of NoSQL Databases

Well-suited scenarios for Key-value stores

1. **Scalability is Essential**: Key-value stores can distribute data across clusters, making it easier to scale out by adding more machines to the cluster. They can handle huge volumes of data and high-speed read/write operations.
2. **Speed and Low Latency are Required**: Due to their simplicity and in-memory nature (for some key-value stores), these databases can achieve low latency and high throughput.
3. **Applications with Simple Data Structure:**
   a. **Session Management**: Key-value stores can be used for session management where each session ID (the key) maps to session data (the value).
   b. **Caching**: Systems like Redis and Memcached are often used as caching layers in front of more traditional databases to speed up read-heavy applications.
   c. **User Profiles, Preferences, or Configuration Settings**: Each user's unique ID can be used as the key, and their associated details or preferences can be stored as the value.
   d. **Shopping Cart Data**: For e-commerce applications, a user's session ID or user ID can be the key, and the items in their cart can be stored as the value.



Types of NoSQL Databases

Document stores

- Similar to key-value stores, but allows nested values associated with each key;
- Value is a document (a group of key-value pairs)
- Usage cases:
  - Document-oriented data, e.g. Tweets, Facebook posts, Amazon comments/feedback
  - Semi-structured data, e.g. JSON files and XML files.
- Pros and Cons:
  - Flexible data structure, simple, and multi-indexed capability.
  - No unique query language (like calling a function in MongoDB)
- Representative products:
  - MongoDB, CouchDB, etc.

mongoDB

CouchDB relax

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

Types of NoSQL Databases

Well-suited scenarios for Document stores

1. **Hierarchical Data**: Document databases excel at managing data with complex relationships, nested fields, or hierarchical data structures.
2. **Content Management Systems (CMS)**: They can handle the diverse and complex data structures typical of CMS, including metadata, user data, and content.
3. **Catalogs**: For e-commerce applications where product attributes might vary but don't necessarily need to fit into a rigid schema.
4. **Blogging Platforms**: Where each post can have different attributes, such as tags, comments, author information, and more.
5. **User Profiles**: For applications where each user can have a wide variety of associated data, including settings, histories, preferences, and more.
6. **Metadata Storage**: Especially when the metadata is diverse and doesn't fit neatly into a table.

mongoDB

CouchDB
relax

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

Columnar Store

- Data are stored by columns rather than rows.

Row Store v. Column Store

Record #	Name	Address	City	State
0003623	ABC	125 N Way	Cityville	PA
0003626	Newburg	1300 Forest Dr.	Troy	VT
0003647	Flotsam	5 Industrial Pkwy	Springfield	MT
0003705	Jolly	529 S 5th St.	Anywhere	NY

Record #	Name	Address	City	State
0003623	ABC	125 N Way	Cityville	PA
0003626	Newburg	1300 Forest Dr	Troy	VT
0003647	Flotsam	Industrial Pkwy	Springfield	MT
0003705	Jolly	529 S 5th St.	Anywhere	NY

Why Columnar Store?

- Practical use of a column store versus a row store differs little in the relational DBMS world.
- Row-oriented DBs are designed to efficiently return data for a entire row (a record).
  - Example: return a record of user, including name, age, gender, address, etc.
- Row-oriented data models are not efficient for column-wise operations on the whole table
  - Find all records of customer's age between 20 and 30 across a table having 100 million records.
  - Scan the column of age row by row without skipping other fields (name, gender, etc.)
  - Big table will need multiple data block (machine) to fit in all data (more disk operations).
- Columnar store can be more efficiently accessed for some particular operations (e.g. data aggregation)
- Typical Columnar DBs: Clickhouse, MariaDB

RowId	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	80000
002	12	Jones	Mary	50000
003	11	Johnson	Cathy	44000
004	22	Jones	Bob	55000

```
001:10,Smith,Joe,80000;
002:12,Jones,Mary,50000;
003:11,Johnson,Cathy,44000;
004:22,Jones,Bob,55000;
```

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
80000:001,50000:002,44000:003,55000:004;
```

Types of NoSQL Databases

Wide-Column (Column-Family) stores

- uses **tables**, **rows**, and **columns**, but unlike a relational database, the names and format of the columns **can vary** from row to row in the same table.
- A column family is a collection of **rows and columns**, where each row has a unique key and each column has a **name**, **value**, and **timestamp**.
- Each column family can have its own set of columns and can be **optimized for different types of queries**
- Usage cases:
  - realtime Big Data process, e.g. Google search (BigTable), Spotify (user recommendation), Facebook (messages)
  - Data Warehousing, Online Analytical Processing (OLAP), high write throughput handling
- Pros and Cons:
  - High performance in terms of query, scalability, distribution.
  - Incremental data loading and Queries against only a few rows
- Representative products:
  - Apache Cassandra, Apache HBase, Google BigTable, etc.

	Column Family 1		Column Family 2	
	Column 1	Column 2	Column 3	Column 4
Row Key 1	John Smith	Male	40,000	T1: 50 T2: 51 T3: 55
Row Key 2	Sen Wang	Male	50,000	T1: 60 T2: 62

Types of NoSQL Databases

Well-suited scenarios for column-family stores

1. **Time Series Data**: Storing data that is written in a time-ordered sequence, like logs, metrics, and sensor data. Since newer data can be written quickly, and older data can be expired efficiently, column stores are apt for this. Stock price data, IoT and sensor data.
2. **Scalability and Distribution**: When the need arises to distribute data across multiple nodes or even across multiple data centers and geographical regions. Column stores can horizontally scale across a lot of commodity hardware.
3. **Analytics and Business Intelligence**: Since data is stored in columns, aggregations and computations over a single column can be much faster than in row-oriented databases.
4. **Personalization and Recommendation Systems**: They can efficiently look up and adjust user attributes in real-time, useful for generating tailored content or advertisements.

Wide-Column NoSQL vs Columnar Databases

The terms "wide-column" and "column-oriented" are often used interchangeably. However, there are subtle differences between them.

Definition:

- **Wide-column store**: The name "wide-column" comes from the ability to have a virtually unlimited number of columns that can vary for each row. It allows for dynamic column creation.
- **Columnar store**: These are databases that store data table by table column, rather than by table row. They are used in relational database management systems (RDBMS) to manage data warehouses, business intelligence, analytics, etc. The primary goal is to achieve better performance for queries over large volumes of read-only data.

Schema:

- **Wide-column store**: Typically, schema-less within columns. Each row can have a different set of columns, and column names and values can be dynamic.
- **Columnar store**: The schema is defined, but data is stored in a columnar fashion for better I/O, compression, and analytics speed.

Data Model:

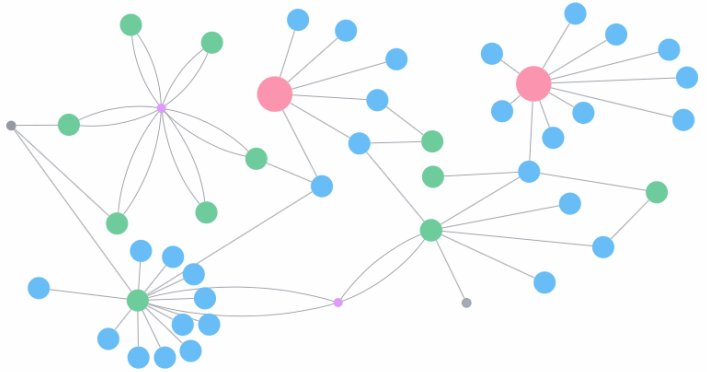
- **Wide-column store**: Uses a distributed model where data is partitioned across many servers. The focus is on horizontal scalability.
- **Columnar store**: While some can be distributed, the focus is more on how data is laid out on disk to optimize for certain types of queries.

In essence, while both wide-column stores and column-oriented databases optimize storage by columns, they have different goals, data models, and ideal use cases.

Types of NoSQL Databases

Graph databases

- Nodes and Edges (relationships) are the bases of graph databases.
  - A node represents an entity, like a user, category, or a piece of data.
  - A relationship represents how two nodes are associated, like friendship, works for, etc.
- The nature of connections in Graph eliminates the (time-consuming) search-match operation found in relational databases.
- Usage cases:
  - Network data, e.g. friendships in Tweets, Facebook, connections in Linkedin
- Pros and Cons:
  - Model complex relationship and support Graph-based algorithm.
  - Complexity (smaller data scale)
- Representative products:
  - Neo4j, OrientDB, InfoGrid, etc.



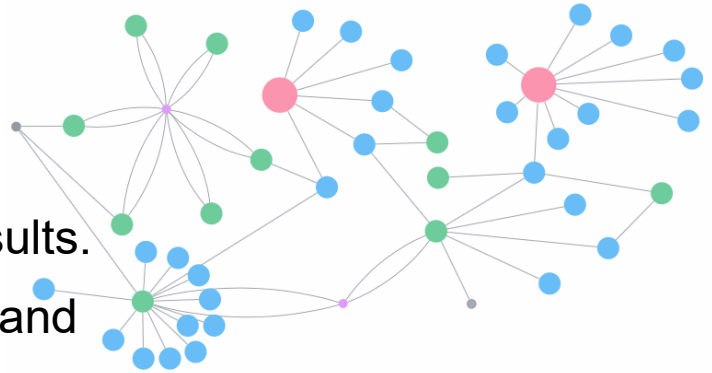
Types of NoSQL Databases

Well-suited scenarios for graph stores

- **Social Networks**: Representing and querying complex relationships between users, such as friends, friends-of-friends, shared interests, and more.
- **Knowledge Graphs**: Connecting related pieces of data across vast datasets, such as linking articles, authors, topics, and references.
- **Logistics and Route Optimization**: Finding the most efficient routes by considering multiple variables and constraints.
- **Bioinformatics and Medical Research**: Modeling and analyzing complex biological systems and pathways.
- **Semantic Search**: Enabling search functionality that understands the relationships between entities and can return more contextually relevant results.
- **Ontologies and Taxonomies**: Mapping complex hierarchies of knowledge and relationships in domains like academia or industry-specific fields.

OrientDB An SAP Company

neo4j



Database Partition

- Database partitioning aims at improvements in
  - Scalability & Performance & Availability
  - Security & Operational flexibility
- A database can be logically divided into multiple partitions.
- Partitions are distinct and independent parts that spread over multiple nodes.
- Popular partition methods in distributed DBMS:
  - Vertical Partitioning
    - Each partition holds a subset of the fields for items in the data store. (SQL vs NoSQL)
    - Frequent fields are placed in one vertical partition (e.g. ProductName)
  - Horizontal Partitioning (or Sharding)
    - Each partition (aka a shard) is a separate data store (a subset of the entire database)

https://en.wikipedia.org/wiki/Partition_(database)
https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning

Vertical Partitioning

- VP aims to reduce the I/O and performance costs.
  - Frequently accessed/coupled items:
    - StuID, Name, GPAs, Program
  - Sensitive data:
    - Gender, D.O.B., Nationality
- VP operates at the entity level within a data store, partially normalizing an entity to break it down from a *wide* item to a set of *narrow* items.
- Ideally suited for wide-column stores
  - E.g. HBase and Cassandra.

StuID	Name	Gender	D.O.B.	Program	GPAs	Nationality
10101	Tom	M	01/09/94	BCS	6.5	AUS
10102	Rocky	M	23/06/95	BIT	6.4	CHN
20101	John	M	12/04/95	MDS	6.6	US
20102	Mary	F	03/02/94	MCB	6.8	AUS
20103	Helen	F	30/10/95	MCS	7	IND

StuID	Name	GPAs	Program
10101	Tom	6.5	BCS
10102	Rocky	6.4	BIT
20101	John	6.6	MDS
20102	Mary	6.8	MCB
20103	Helen	7	MCS

StuID	Gender	D.O.B.	Nationality
10101	M	01/09/94	AUS
10102	M	23/06/95	CHN
20101	M	12/04/95	US
20102	F	03/02/94	AUS
20103	F	30/10/95	IND

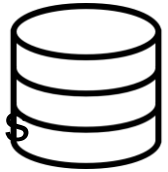
https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning

Horizontal Partitioning - Sharding

- A shard is a horizontal partition of data in a DB.
- Each shard is held on a separate database server instance, to spread the load of a large volume of data.
- Ensure an appropriate sharding key
  - Evenly distribute data across the shards (nodes)
  - Shards don't have to have the same size
  - Balance the number of requests across shards
  - Avoid hot partitions
- Ideally suited for SQL and some NoSQL DBs
  - MySQL NDB Cluster
  - Redis Cluster and MongoDB

StuID	Name	Gender	D.O.B.	Program	GPAs	Nationality
10101	Tom	M	01/09/94	BCS	6.5	AUS
10102	Rocky	M	23/06/95	BIT	6.4	CHN
20101	John	M	12/04/95	MDS	6.6	US
20102	Mary	F	03/02/94	MCB	6.8	AUS
20103	Helen	F	30/10/95	MCS	7	IND

StuID	Name	Gender	D.O.B.	Program	GPAs	Nationality
10101	Tom	M	01/09/94	BCS	6.5	AUS
10102	Rocky	M	23/06/95	BIT	6.4	CHN



StuID	Name	Gender	D.O.B.	Program	GPAs	Nationality
20101	John	M	12/04/95	MDS	6.6	US
20102	Mary	F	03/02/94	MCB	6.8	AUS
20103	Helen	F	30/10/95	MCS	7	IND

https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning

Hashing in NoSQL – What & Why

A mathematical function that takes an input (a **key**) and converts it into a fixed-size number (called a **hash value**).

This process is **extremely fast** and always produces the same output for the same input.

The most common method for distributing data is to use this hash value with the **modulo operator (%)**.

NoSQL databases (e.g., Cassandra, DynamoDB, MongoDB sharding) rely on hashing for:

- **Fast access** – constant-time retrieval.
- **Data distribution** – spreading documents/records evenly across nodes.
- **Load balancing** – prevents hotspots where one server stores too much data.

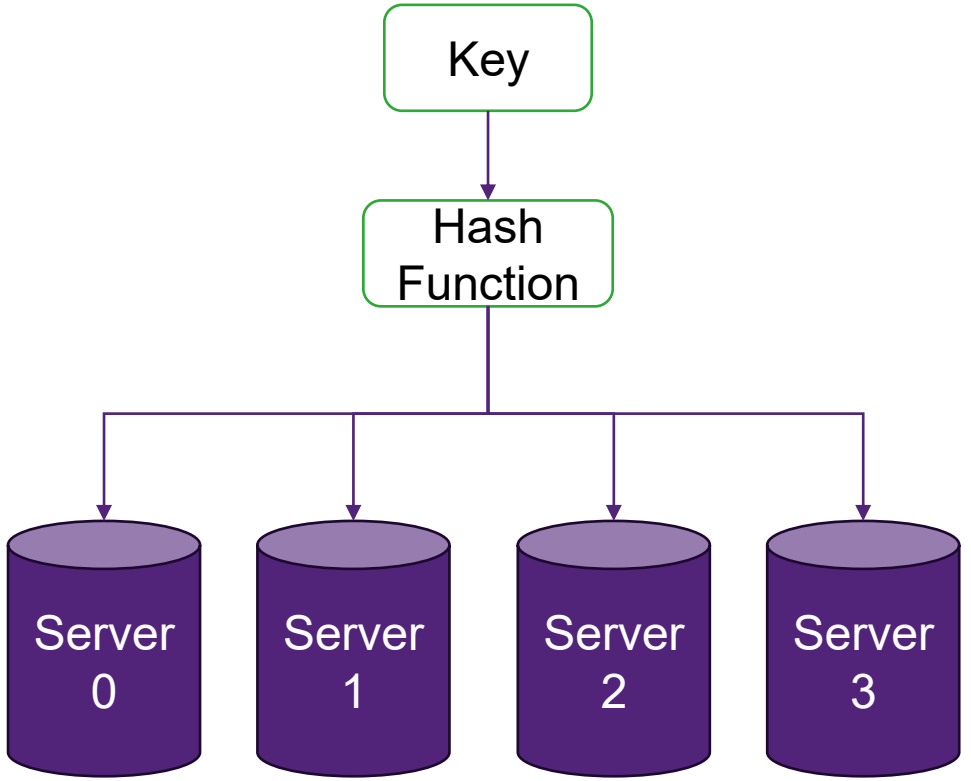
Hashing in NoSQL – An Intuitive Example

A Step-by-Step Example - Imagine you have FOUR database servers and you need to decide where to store a new piece of data.

1. **Input Key:** The unique identifier for your data, for example, "user:1001".
2. **Hash Function:** A hashing algorithm (like CRC16 in Redis or MD5 in MongoDB) processes the key.

   hash("user:1001") → 54321 (This is the hash value)
3. **Modulo Operation:** The hash value is divided by the number of available servers, and the remainder determines the destination.

   54321 % 4 → 1
4. **Data Placement:** The data for "user:1001" is sent to **Server 1**.



Hashing in Redis: The Key to Speed

In a key-value store like Redis, hashing is central to its core architecture and is the primary reason for its high-performance data retrieval.

Primary Uses:

- **Core Data Storage:** Redis uses a hash table as its main data structure to store all key-value pairs. When a key is provided, Redis hashes it to instantly calculate the memory address of its corresponding value. This allows for constant time O(1) lookups, making both read and write operations extremely fast.
- **The "Hash" Data Type:** Redis provides a specific "Hash" data type, which is essentially a collection of field-value pairs stored under a single key. This is a memory-efficient way to represent objects, such as a user profile with multiple attributes.
- **Data Distribution (Clustering):** To scale horizontally, Redis Cluster distributes data across multiple nodes using a concept called "hash slots." Every key is hashed to one of 16,384 slots, and each slot is assigned to a specific node in the cluster.
  - HASH_SLOT = CRC16(key) % 16384

    https://crccalc.com/

Hashing in MongoDB: Even Data Distribution

In a document-oriented database like MongoDB, hashing is not the primary mechanism for storing or retrieving data (that role is filled by B-Tree indexes). Instead, it is used as a specialized tool to solve specific challenges.

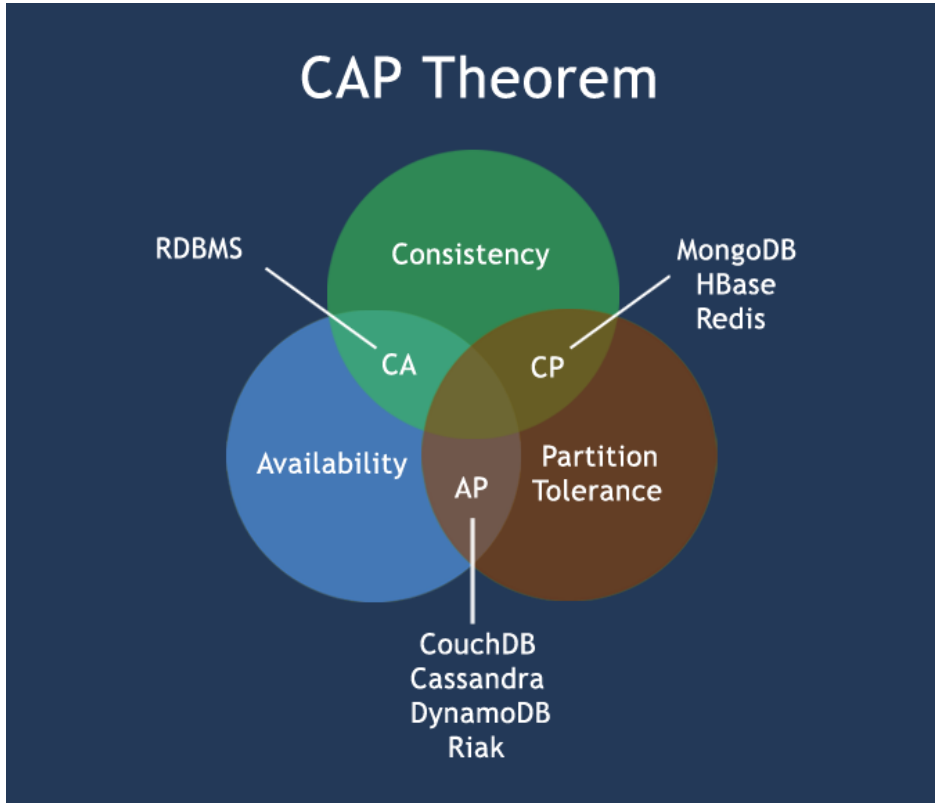
Primary Uses:

- **Even Data Distribution (Hashed Sharding):** MongoDB uses hashed sharding to distribute documents evenly across a cluster of servers. It computes a hash of a field's value (the "shard key") to determine which shard will store the document. This method prevents "**hotspots**" by ensuring that documents with similar but distinct shard key values (like sequential IDs or timestamps) are spread randomly across the cluster.
- **Specialized Indexing:**
  - **Hashed Indexes:** MongoDB allows for the creation of hashed indexes, which store hashes of a field's value. These indexes are primarily used to support hashed sharding.
  - **Geospatial Indexes:** For location-based data, MongoDB uses a **geohashing** algorithm. This technique converts two-dimensional coordinates (latitude and longitude) into a single hash string, which can then be efficiently indexed for fast proximity queries.

CAP Theorem

- The CAP theorem states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:
  - **C**onsistency: Every read receives the most recent write or an error
  - **A**vailability: Every request receives a (non-error) response – without the guarantee that it contains the most recent write
  - **P**artition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Eric Brewer



NoSQL Implementation Options

CA (*Consistency & Availability*)

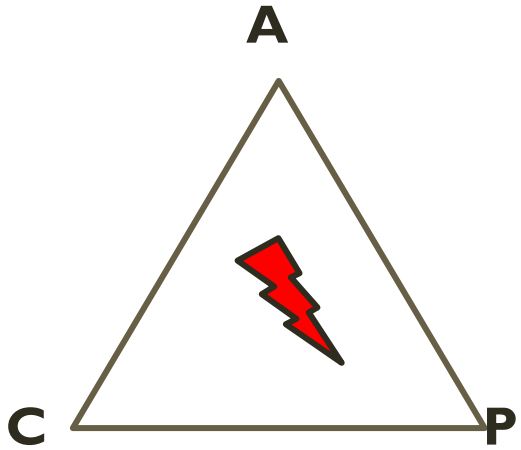
- All clients always have the same view on the data
- Each Client can always read and write
- The system may not tolerance to failure and reconfiguration

AP (*Availability & Partition Tolerance*)

- Each Client can always read and write
- The system works well despite the physical partitions
- Clients may have inconsistent views on the data

CP (*Consistency & Partition Tolerance*)

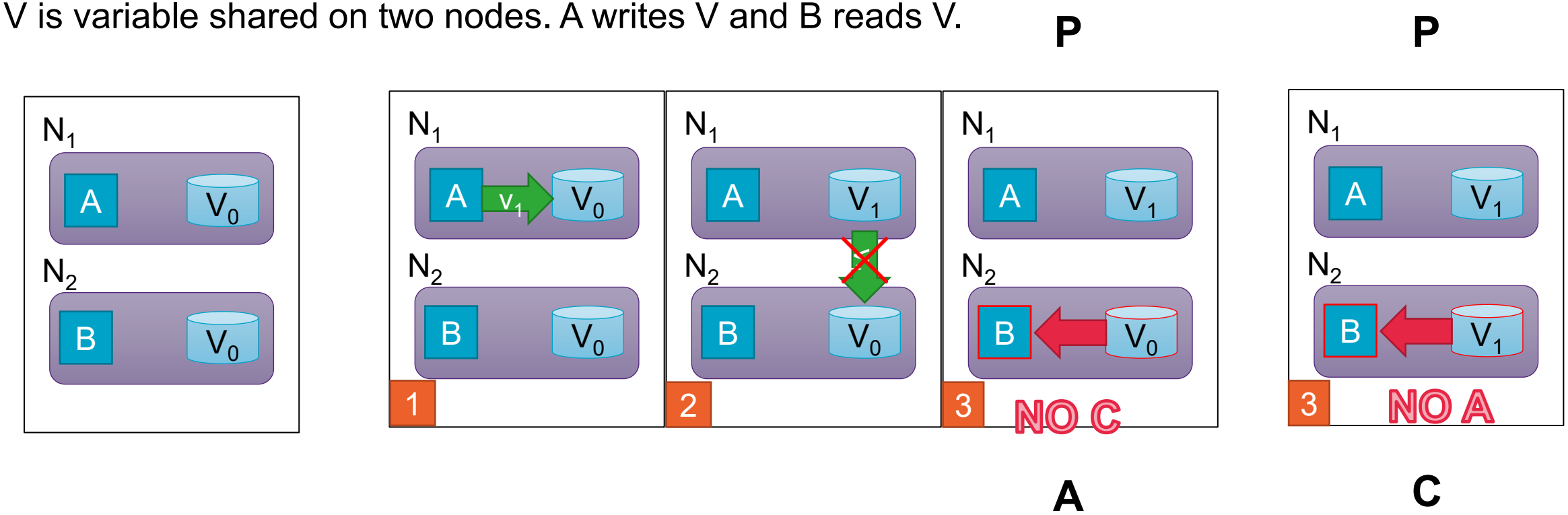
- All clients always have the same view on the data
- The system works well despite the physical partitions
- Clients sometimes may not be able to access data



Example of CAP

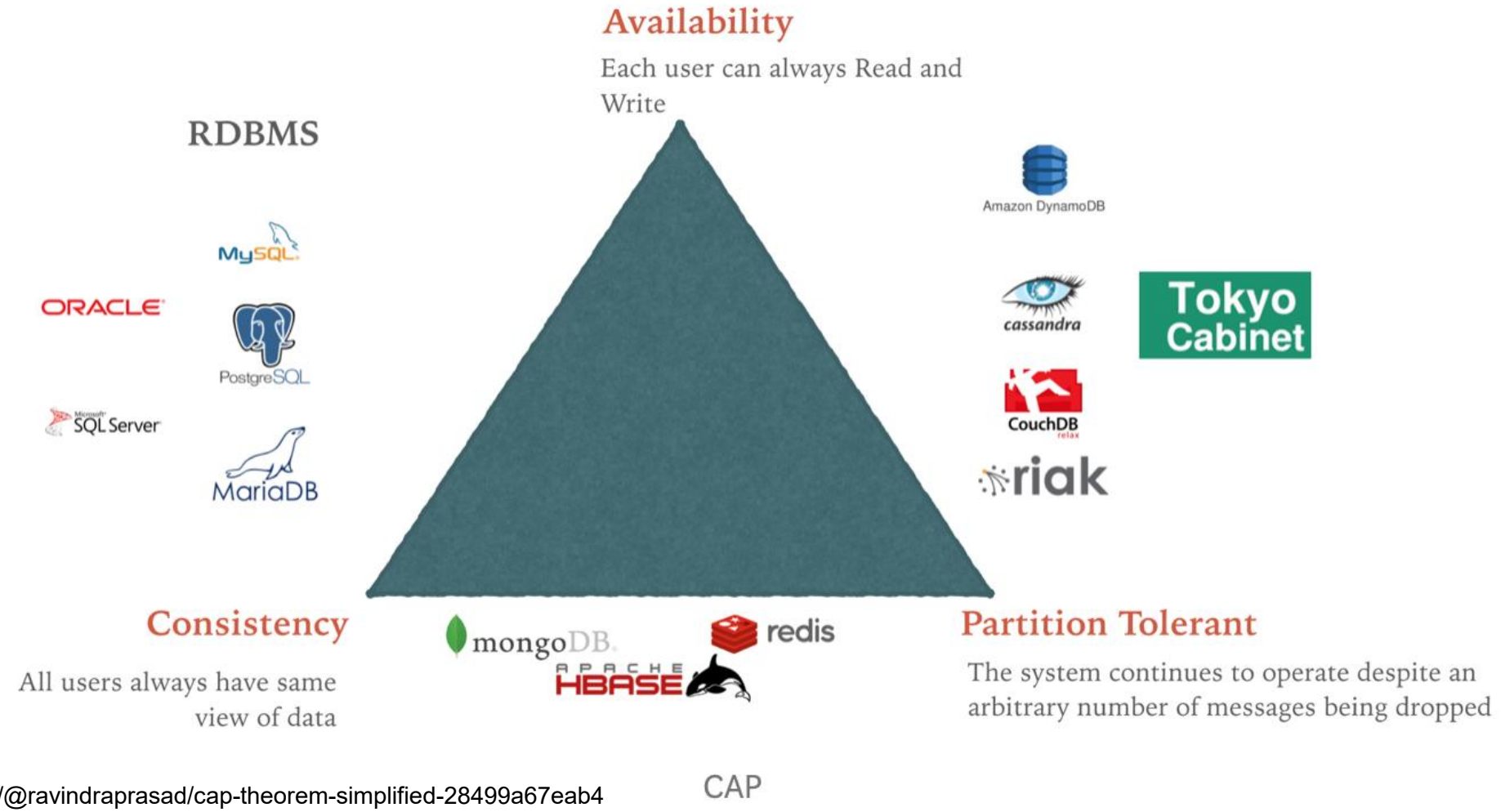
Proof in diagrams

- TWO programs A and B run on TWO nodes N1 and N2, respectively
- V is variable shared on two nodes. A writes V and B reads V.



http://www.julianbrowne.com/article/brewers-cap-theorem

CAP for DBs



https://medium.com/@ravindraprasad/cap-theorem-simplified-28499a67eab4

BASE: ACID Alternative

- Compared to ACID for relational DBs, NoSQL DBs are often associated with BASE:
  - **B**asically **A**vailable, **S**oft-state, **E**ventually consistent
- **B**asically **A**vailable:
  - one distributed system has failure parts but the total system is still working properly.
  - E.g. online shopping on Black Friday
- **S**oft-State: allows delays or outages (short period)
- **E**ventually consistent:
  - Strong consistency: the data should be consistent after every transaction.
  - Rather than requiring consistency after every transaction, it is enough for the distributed database to eventually be in a consistent state.

Question 1:
Is transaction consistency a MUST for ALL applications?

Question 2:
How to achieve ACID in a distributed environment?

Choose the appropriate NoSQL for your apps

Scenario 1: Social Media "Likes" Counter

Requirement: The system must be always available for users to like a post, even if some data centers face issues. An exact, immediate count isn't necessary; it's okay if different users see slightly different like counts for a brief period.

Scenario 2: E-commerce Product Catalog - An online shopping site displays a catalog of products.

Requirement: The product catalog should be available at all times for users to browse, even if there are network issues. It's acceptable if some users don't see the very latest products immediately after they're added.

Scenario 3: E-commerce Transaction System.

Requirement: Every transaction needs to be accurately reflected in the account balance. Even if there's a network partition, it's preferable to deny transactions rather than process them without ensuring consistency.

Scenario 4: Airline Seat Reservation System - An online system for booking airline seats.

Requirement: It's vital to ensure that the same seat isn't booked by multiple passengers. If there's uncertainty due to a network partition, it's better to temporarily halt bookings rather than double-book a seat.

Outline

- Database Background
- Relational Databases
  - Revisit Relational DBs
  - ACID Properties
  - Clustered RDBMs
- Non-relational Databases
  - NoSQL concepts & types
  - Database Partitioning, Hashing in NoSQL, CAP Theorem, BASE
  - MongoDB
  - Cassandra
  - HBase

Examples of NoSQL databases

What is MongoDB?

- A document-oriented NoSQL database:
  - developed by C++ and cross-platform
  - open-source, dfs-based, and horizontally scalable
  - Schema-less: No Data Definition Language (DDL)
  - So you can store hashes with any keys and values that you choose
    - Keys are a basic data type but in reality stored as strings
    - Document Identifiers (_id) will be created for each document, field name reserved by system
  - Uses BSON format: based on JSON (JSON like, B stands for Binary)
- Supports APIs (drivers) in many computer languages (10+)
  - JavaScript, Python, Ruby, Perl, Java, Scala, C#, C++, Haskell, Erlang, etc.

https://en.wikipedia.org/wiki/Hash_function

MongoDB – Document Database

- A record in MongoDB is a document: field and value pairs.
- MongoDB documents are similar to JSON objects.
- The values of fields may include other documents, arrays, and arrays of documents.

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```

The advantages of using documents are:

- Documents (i.e. objects) correspond to native data types in many programming languages.
- Embedded documents and arrays reduce need for expensive joins.

https://en.wikipedia.org/wiki/Hash_function

JSON Format Revisit

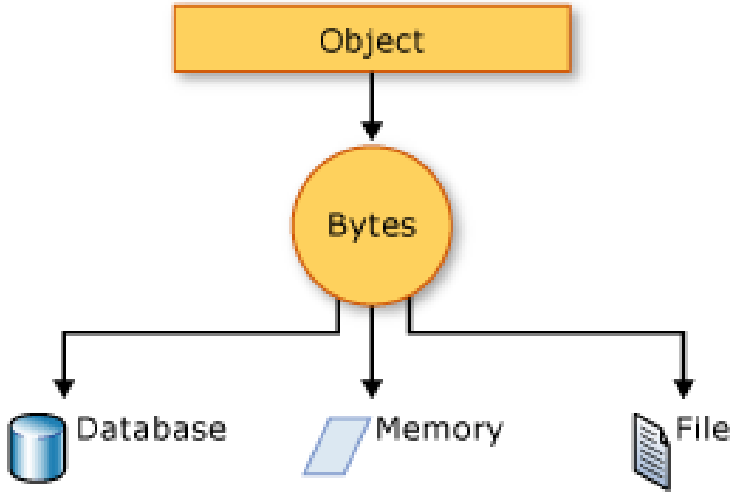
- Data is in *name*/*value* pairs
- A *name/value* pair consists of a field name followed by a colon, followed by a value:
  - **Example**: *name*: "R2-D2"
- Data is separated by commas
  - **Example**: *name*: "R2-D2", race : "Droid"
- Curly braces hold objects
  - **Example**: {*name*: "R2-D2", *race* : "Droid", *affiliation*: "rebels"}
- An array is stored in brackets []
  - **Example**: [{*name*: "R2-D2", *race* : "Droid", *affiliation*: "rebels"}, {*name*: "Yoda", *affiliation*: "rebels"}]

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

https://www.json.org/

BSON format

- BSON is simply binary-encoded serialization of JSON-like documents
- Zero or more *key-value* pairs are stored as a single entity
- Each entry consists of a field name, a data type, and a value
- Support faster encoding and decoding techniques
- Suitable for data storage
- Lightweight, fast and traversable



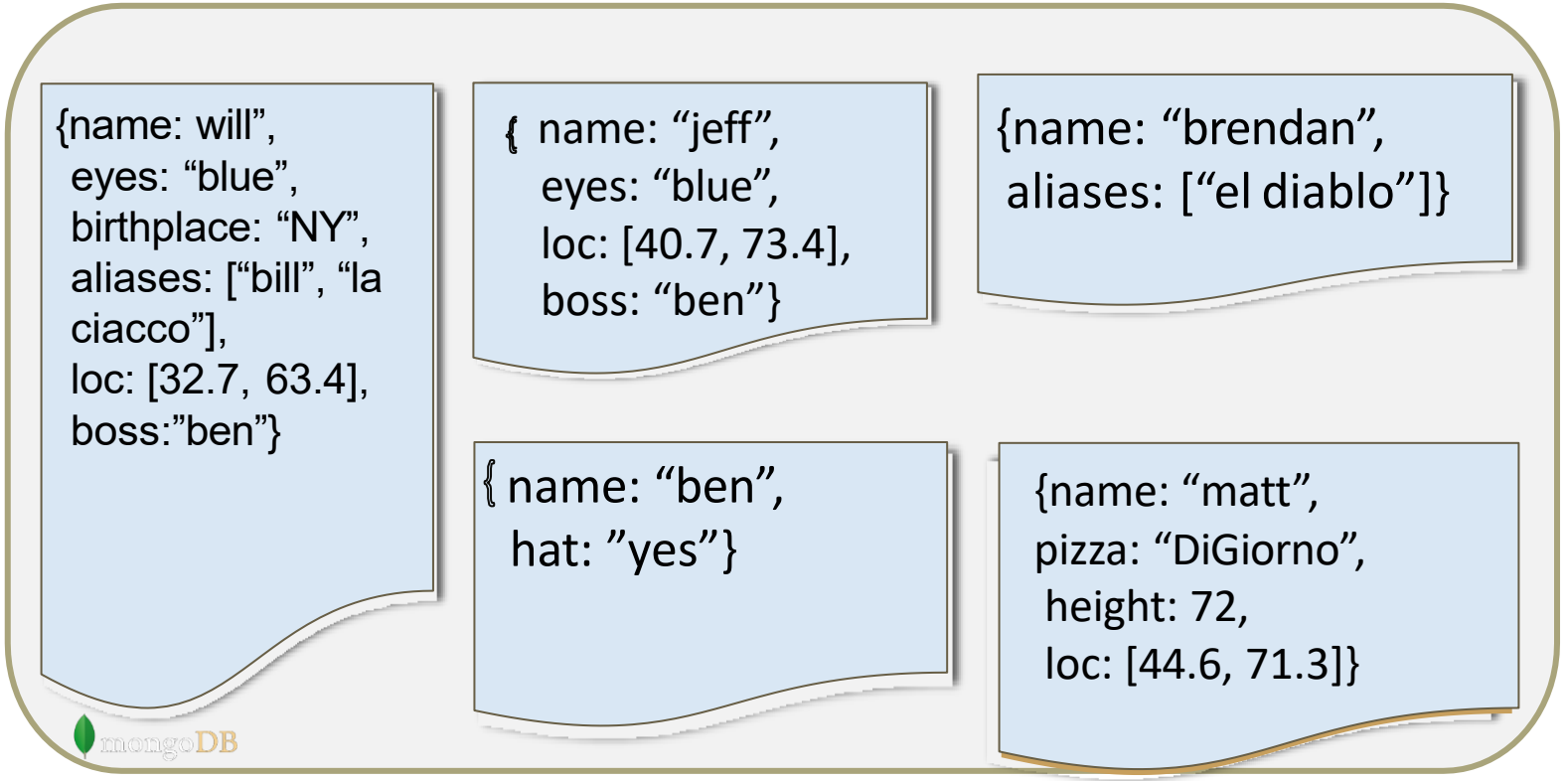

```
Bson:
  \x16\x00\x00\x00                  // total document size
  \x02                              // 0x02 = type String
  hello\x00                         // field name
  \x06\x00\x00\x00world\x00         // field value (size of value, value, null
terminator)
  \x00                              // 0x00 = type EOO ('end of object')
```

A document such as {"hello":"world"}

https://en.wikipedia.org/wiki/BSON

Schema Free

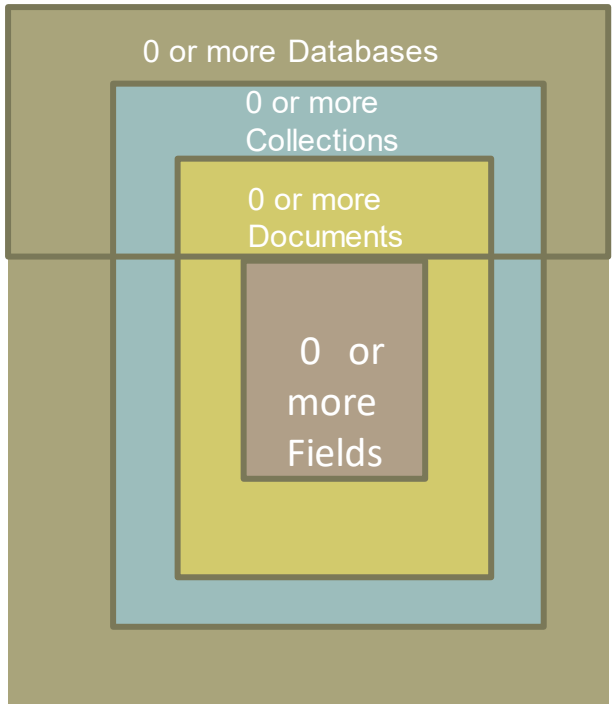
- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data



MongoDB: Hierarchical Objects

- A MongoDB instance may have zero or more 'databases'
- A database may have zero or more 'collections'.
- A collection may have zero or more 'documents'.
- A document may have one or more 'fields'.
- MongoDB 'Indexes' function much like their RDBMS counterparts.

RDBMS	MongoDB
Database	Database
Table, View	Collection
Row	Document (BSON)
Column	Field
Index	Index
Join	Embedded Document
Foreign Key	Reference
Partition	Shard



Note:

- **Collection** is not strict about what it Stores
- **Document** can be Embedded

Key Features of MongoDB

- **High Performance**
  - Support for embedded data models reduces I/O activity on database system.
  - Indexes support faster queries and can include keys from embedded documents and arrays.
- **Rich Query Language**
  - Support CRUD, Data Aggregation, Text Search and Geospatial Queries.
- **High Availability**
  - Support automatic failover and data redundancy.
- **Horizontal Scalability**
- **Multiple Storage Engines**
  - WiredTiger Storage Engine (Default)
  - In-Memory Storage Engine

CRUD operations

- **Create**
  - db.collection.insertOne(<document>)
  - db.collection.insertMany(<document>)
- **Read**
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- **Update**
  - db.collection.update(<query>, <update>, <options>)
- **Delete**
  - db.collection.remove(<query>, <justOne>)

Collection specifies the collection or the ‘table’ to store the document

Create Operations

Create or insert operations add new documents to a collection.

- MongoDB provides the following methods to insert documents into a collection: syntax: db.collection.insertOne() & db.collection.insertMany()
- In MongoDB, insert operations target a single collection.

```
try {
   db.products.insertOne( { item: "card", qty: 15 } );
} catch (e) {
   print (e);
};
```

No "_id"

```
try {
   db.products.insertOne( { _id: 10, item: "box", qty: 20 } );
} catch (e) {
   print (e);
}
```

with "_id"

Create Operations

Create or insert operations add new documents to a collection.

- db.collection.insertMany()

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm"
])
```

- In MongoDB, insert operations target a single collection.
- All write operations in MongoDB are atomic on the level of a single document.

Read Operations

- Read operations retrieves documents from a collection;
  - i.e. queries a collection for documents.
- MongoDB provides the following methods to read documents from a collection:
  db.collection.find()
  - Find all documents in a collection: `db.bios.find()` => select * from bios; (SQL-like)
  - Query for equality: `db.bios.find( { _id: 5 } )` => select * from bios where _id=5;
  - Query for embedded item:
```
db.bios.find( { "name.last": "Hopper" } )
```
  - Query using operators ($):
```
db.bios.find( { birth: { $gt: new Date('1950-01-01') } } )
```
  - Query for ranges:
```
db.bios.find( { birth: { $gt: new Date('1940-01-01'), $lt: new Date('1960-01-01') } } )
```
  - Query for multiple conditions:
```
db.bios.find( {
   birth: { $gt: new Date('1920-01-01') },
   death: { $exists: false }
} )
```

https://docs.mongodb.org/manual/reference/operator/query/

Read Operations – Query Operators

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value)
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a loginalAND
$not	Inverts the effect of a query expression
$nor	Join query clauses with a logical NOR
$exists	Matches documents that have a specified field

https://docs.mongodb.org/manual/reference/operator/query/

Read Operations

- The find() method returns a cursor to the results and an iteration is required to get the results
- .next() method:

```
var myCursor = db.bios.find( );

var myDocument = myCursor.hasNext() ? myCursor.next() : null;

if (myDocument) {
    var myName = myDocument.name;
    print (tojson(myName));
}
```

- .foreach() method:

```
var myCursor = db.bios.find( );

myCursor.forEach(printjson);
```

- The limit(n) method returns n documents instead of all results:

```
db.bios.find().limit( 5 )
```

- The skip(n) method controls the starting point of the results set:

```
db.bios.find().skip( 5 )
```

https://docs.mongodb.org/manual/reference/operator/query/

Update Operations

- Update operations modify existing documents in a collection.
- MongoDB provides the following methods to update documents of a collection:
  - db.collection.updateOne(), db.collection.updateMany(), and db.collection.replaceOne()
- Conditions with operators can be used to filter the data.

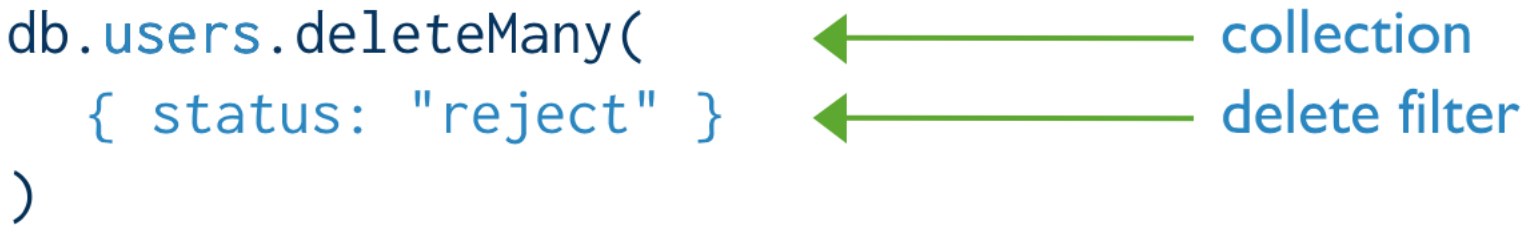
```
db.users.updateMany(                    <-- collection
   { age: { $lt: 18 } },                <-- update filter
   { $set: { status: "reject" } }       <-- update action
)
```

- replaceOne() vs updateOne()
  - replaceOne() allows to replace the entire document with updated field's values
  - updateOne() allows to update the specific field's values.

https://docs.mongodb.org/manual/reference/operator/query/

Delete Operations

- Delete operations remove documents from a collection.
- MongoDB provides the following methods to delete documents of a collection:
  - db.collection.deleteOne()
  - db.collection.deleteMany()
- Similarly, criteria, or filters, that identify the documents to remove can be specified with the same syntax as read operations.



https://docs.mongodb.org/manual/reference/operator/query/

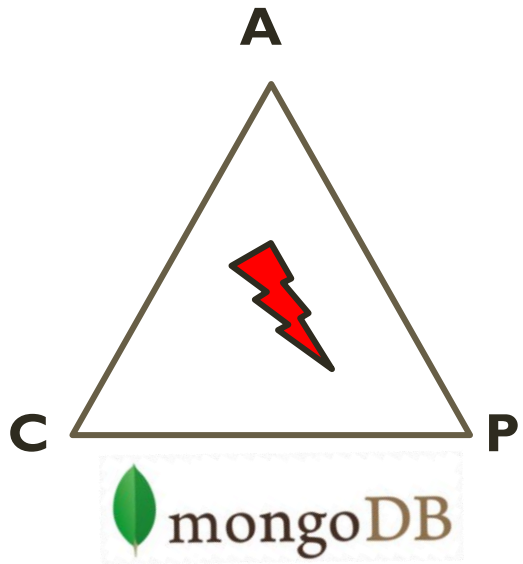
MongoDB: CAP Position

Focus on **Consistency and Partition tolerance** (CP)

Consistency
- all replicas contain the same version of the data

Availability
- system remains operational on failing nodes

Partition tolarence
- multiple entry points
- system remains operational on system split



CAP Theorem: satisfying all three at the same time is impossible

Outline

- Database Background
- Relational Databases
  - Revisit Relational DBs
  - ACID Properties
  - Clustered RDBMs
- Non-relational Databases
  - NoSQL concepts & types
  - Database Partitioning, Hashing in NoSQL, CAP Theorem, BASE
  - MongoDB
  - Cassandra
  - HBase

Example of NoSQL database

What is Cassandra?

Apache Cassandra is a wide-column distributed database:

- Free, open-source, decentralized (P2P), distributed
- Aims to managing very large amounts of data across the world.
- provides highly available service with no single point of failure.
- It is scalable, fault-tolerant, and eventually consistent.
- Created at Facebook, it differs sharply from relational database management systems.
- Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Avinash Lakshman

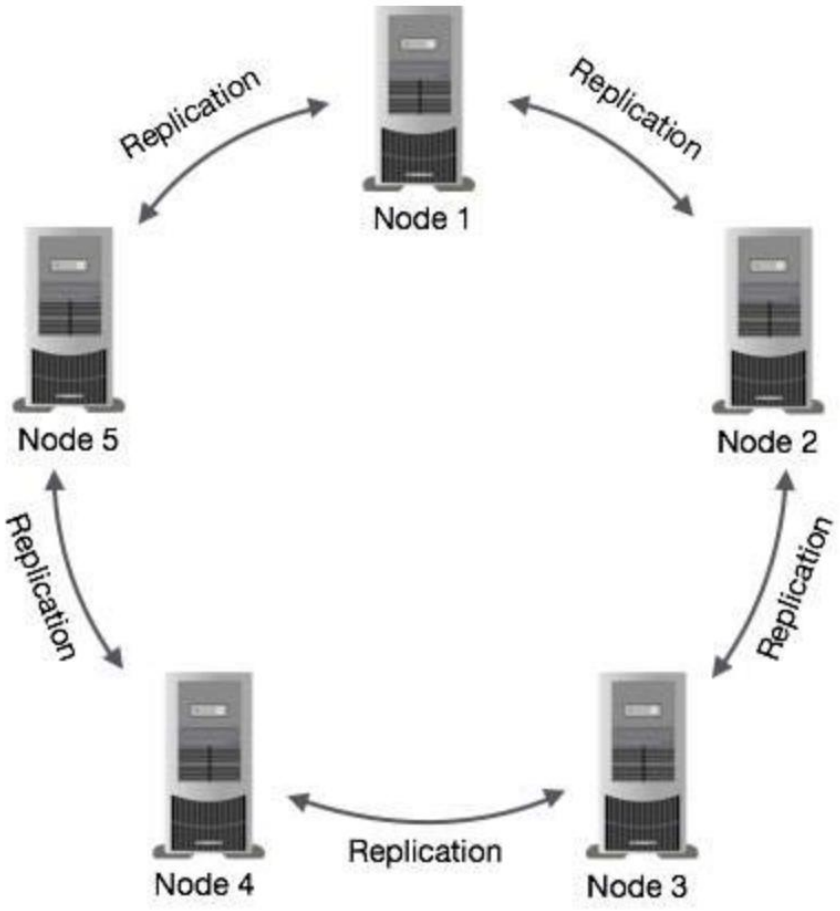
Prashant Malik

Cassandra is the cursed Oracle

https://en.wikipedia.org/wiki/Apache_Cassandra

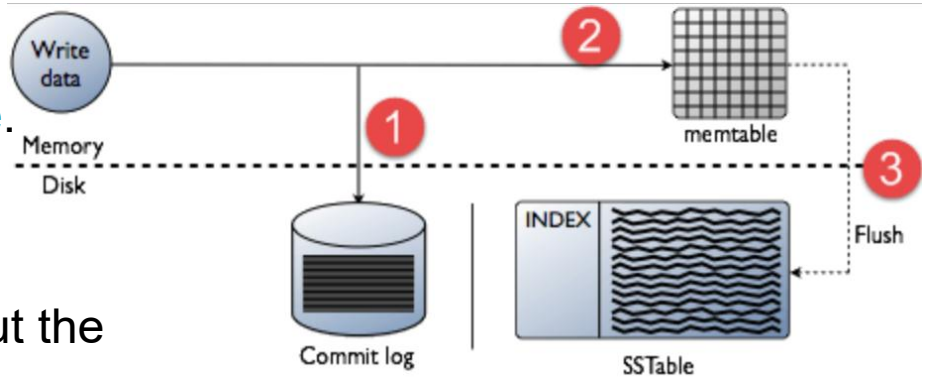
Cassandra Architecture

- The design goal: handle big data workloads across multiple nodes without any single point of failure.
- Cassandra has peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.
  - All the nodes in a cluster play the same role and each node is independent.
  - Each node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
  - When a node goes down, read/write requests can be served from other nodes in the network.
- one or more of the nodes in a cluster act as replicas for a given piece of data.
- Cassandra uses the Gossip Protocol in the background to allow the nodes to communicate with each other and detect any faulty nodes in the cluster.



Read & Write in Cassandra

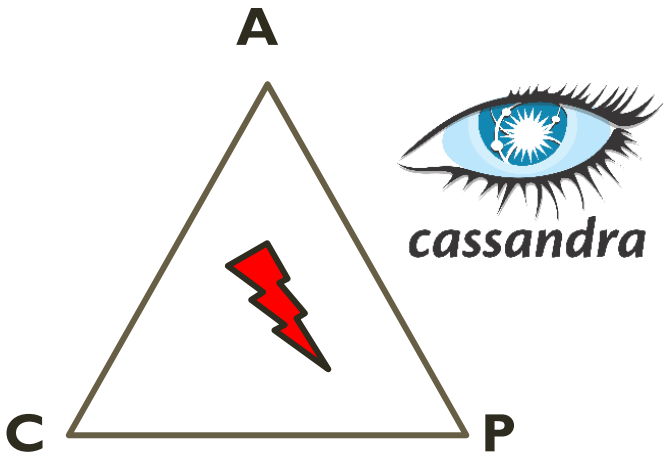
- **Write Operations**
  - Every write activity of nodes is captured by the commit logs written in the nodes in step 1.
  - In step 2, the data will be captured and stored in the mem-table.
  - Whenever the mem-table is full, data will be written into the SStable data file in step 3.
  - All writes are automatically partitioned and replicated throughout the cluster.
  - Cassandra periodically consolidates the SSTables, discarding unnecessary data.
- **Read Operations**
  - Cassandra tries to get values from the mem-table
  - Or finds the appropriate SSTable that holds the required data.



https://en.wikipedia.org/wiki/Apache_Cassandra

Cassandra: a CAP approach

- Cassandra is typically classified as an AP system.
- Availability and Partition Tolerance are generally considered to be more important than consistency in Cassandra.
- Cassandra can be tuned with replication factor and consistency level to also meet C.



CAP Theorem:
satisfying all three at the same time is impossible

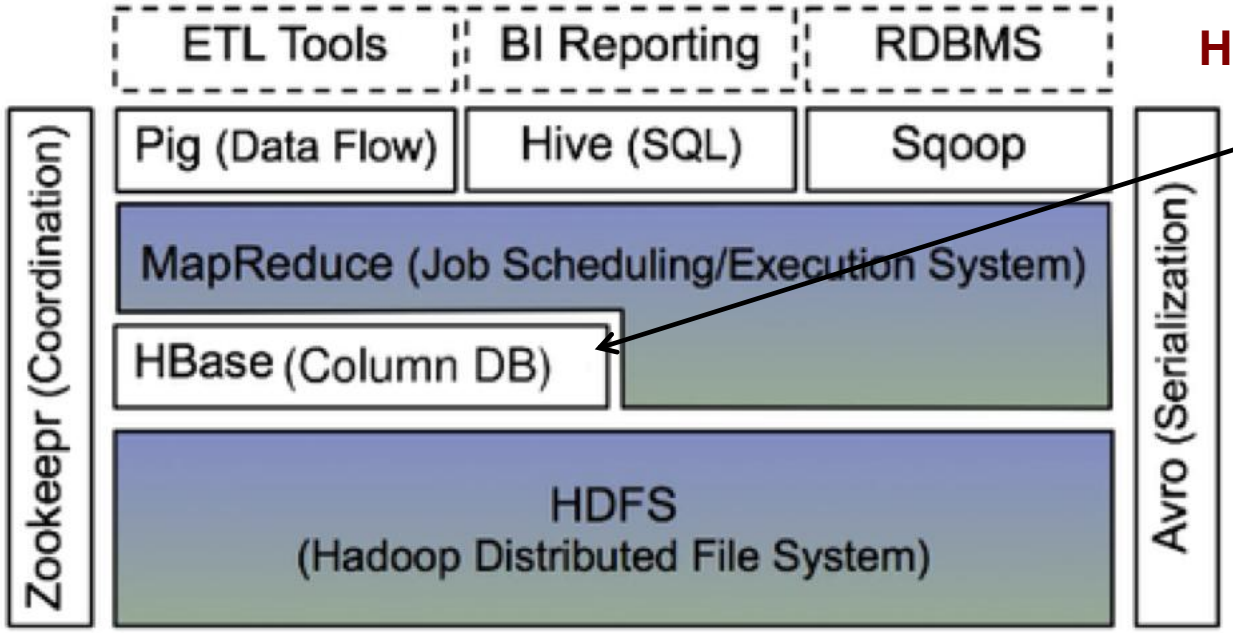
Outline

- Database Background
- Relational Databases
  - Revisit Relational DBs
  - ACID Properties
  - Clustered RDBMs
- Non-relational Databases
  - NoSQL concepts & types
  - Database Partitioning, Hashing in NoSQL, CAP Theorem, BASE
  - MongoDB
  - Cassandra
  - HBase

What is HBase?

- Inspired by Google's proprietary distributed BigTable
- Open-source Apache project
- Runs on top of HDFS
- Written in Java
- NoSQL (Not Only SQL) Distributed Database
- Consistent and Partition tolerant
- Runs on commodity hardware
- Large Database (terabytes to petabytes).
- Many companies are using HBase
  - Facebook, Twitter, Adobe, Mozilla, Yahoo!, Trend Micro, and StumbleUpon

HBase – Architecture



HBase is built on top of HDFS

HBase files are internally stored in HDFS

Wide-col NoSQL	**BigTable**	**HBase**
DFS	GFS	HDFS
Computation Model	MapReduce	Hadoop MapReduce

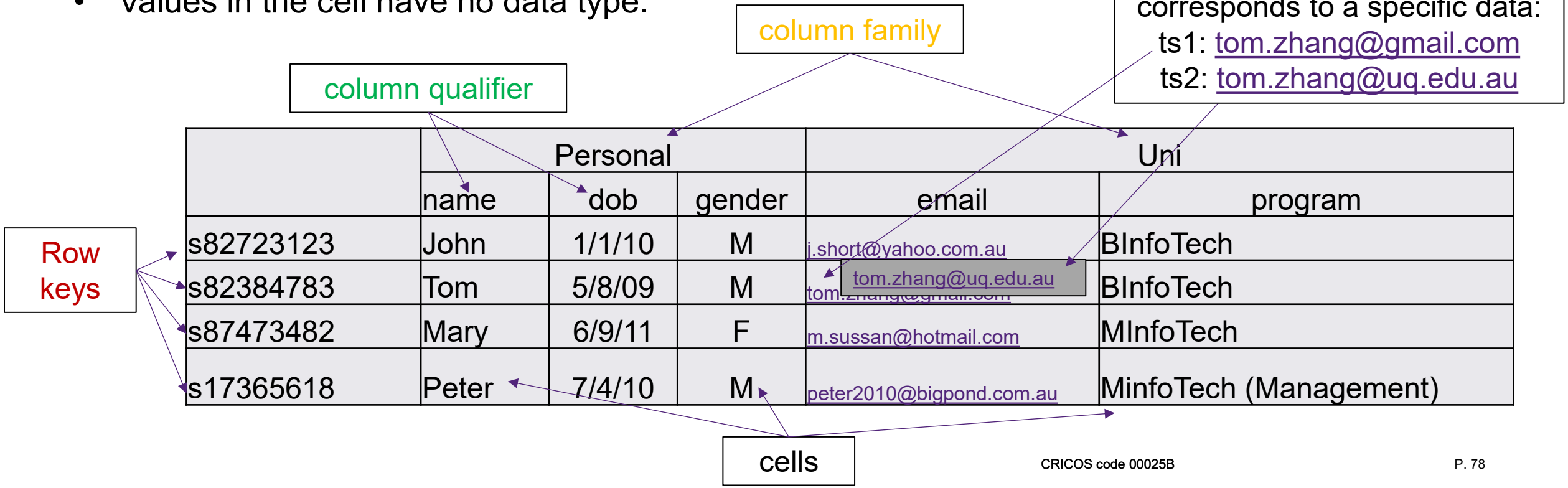
HBase vs HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes
- HDFS is a DFS while HBase is a wide-column NoSQL distributed databases.
  - HDFS is good for batch processing (scans over big files)
    - Not good for record lookup (random access)
    - Not good for incremental addition of small batches
    - Not good for updates
  - HBase is designed to efficiently query data records
    - Fast record lookup (online process)
    - Support for record-level insertion
    - Support for updates (not in place)
- If your application has neither random reads or writes, stick to HDFS

HBase – Data Model

- HBase is a sparse, multi-dimensional, sorted map.
- There are four components in an index to locate the data's value in the map:
- Row key, column family, column qualifier, and time stamp
- Values in the cell have no data type.

There are two time stamps for this cell, ts1 and ts2. Each ts corresponds to a specific data:
ts1: tom.zhang@gmail.com
ts2: tom.zhang@uq.edu.au



	Personal			Uni	
	name	dob	gender	email	program
s82723123	John	1/1/10	M	j.short@yahoo.com.au	BInfoTech
s82384783	Tom	5/8/09	M	tom.zhang@uq.edu.au	BInfoTech
s87473482	Mary	6/9/11	F	m.sussan@hotmail.com	MInfoTech
s17365618	Peter	7/4/10	M	peter2010@bigpond.com.au	MinfoTech (Management)

HBase – Data Model

- In HBase, each row has a row key that can be sorted and arbitrary number of columns.
- In the horizontal direction, a table is compromised by column families, each of which can have arbitrary number of columns.
- Column family can be dynamically extended by adding a new column family or a column
- HBase will keep the historical records when updating with the new value:
  - tom.zhang@uq.edu.au is new and the previous tom.zhang@gmail.com is be kept.

	Personal			Uni		
	name	dob	gender	email	program	GPA
s82723123	John	1/1/10	M	j.short@yahoo.com.au	BInfoTech	4.75
s82384783	Tom	5/8/09	M	tom.zhang@uq.edu.au	BInfoTech	5.25
s87473482	Mary	6/9/11	F	m.sussan@hotmail.com	MInfoTech	6
s17365618	Peter	7/4/10	M	peter2010@bigpond.com.au	MinfoTech (Management)	6.5

HBase – Data Model

Key	Value
[”s82384783”, “Uni”, “emai”,”ts1”]	“tom.zhang@gmail.com”
[”s82384783”, “Uni”, “emai”,”ts2”]	“tom.zhang@uq.edu.au”

Row keys

column family

column qualifier

Timestamp

	Personal			Uni		
	name	dob	gender	email	program	GPA
s82723123	John	1/1/10	M	j.short@	BInfoTech	4.75
s82384783	Tom	5/8/09	M	tom.zha	ch	5.25
s87473482	Mary	6/9/11	F	m.sussan@hotmail.com	MInfoTech	6
s17365618	Peter	7/4/10	M	peter2010@bigpond.com.au	MinfoTech (Management)	6.5

tom.zhang@uq.edu.au

```
{
    "s82384783":
    {
        "Personal":
        {
            "name":
            {
                "Timestamp1": "Tom"
            }
            "dob":
            {
                "Timestamp1": "5/8/09"
            }
            "gender":
            {
                "Timestamp1": "M"
            }

        }
        "Uni":
        {
            "email":
            {
                "Timestamp1": "tom.zhang@gmai.com"
                "Timestamp2": "tom.zhang@uq.edu.au"
            }
            ...
        }
    }
}
```

HBase – Conceptual View

The table stores the crawled data from a website:

Row Key	**TS**	**Column Family** contents	**Column Family** anchor
”au.net.abc.www"	t5		anchor:abcsi.com=”ABC”
	t4		anchor:my.look.ca=”ABC.net.au"
	t3	contents:html="<html>..."	
	t2	contents:html="<html>..."	
	t1	contents:html="<html>..."	

HBase – Physical View

Row Key	TS	Column Family contents
"au.net.abc.www"	t3	contents:html="<html>..."
	t2	contents:html="<html>..."
	t1	contents:html="<html>..."

Row Key	TS	Column Family anchor
"au.net.abc.www"	t5	anchor:abcsi.com=”ABC”
	t4	anchor:my.look.ca=”ABC.net.au"

HBase – Shell (commands)

Command	Description
list	Shows list of tables
create ‘users’, ‘info’	Creates users table with a single column family name info.
put ‘users’, ‘row1’, ‘info:fn’, ‘John’	Inserts data into users table and column family info.
get ‘users’, ‘row1’	Retrieve a row for a given row key
scan ‘users’	Iterate through table users
disable ‘users’ drop ‘users’	Delete a table (requires disabling table)

CRUD explained

CREATE	=	PUT
READ	=	GET
UPDATE	=	PUT
DELETE	=	DELETE

HBase – Java API (examples)

Command	Description
Get	Get get = new Get(String.valueOf(uid).getBytes()); Result[] results = table.get(gets);
Put	Put p = new Put(Bytes.toBytes(""+user.getUid())); p.add(Bytes.toBytes("info"), Bytes.toBytes("fn"), Bytes.toBytes(user.getFirstName())); p.add(Bytes.toBytes("info"), Bytes.toBytes("ln"), Bytes.toBytes(user.getLastName())); table.put(p);
Delete (column, column family)	Delete d = new Delete(Bytes.toBytes(“”+user.getUid())); d.deleteColumn(Bytes.toBytes("info"), Bytes.toBytes("fn"), Bytes.toBytes(user.getFirstName()), timestapmp1);
Batch Operations	List of Get, Put or Delete operations
Scan	Iterate over a table. Prefer Range / Filtered scan. Expensive operation.

Comparison: Cassandra, HBase, MongoDB, and Redis

	Cassandra	HBase	MongoDB	Redis
CAP	AP	CP	CP	CP
Network	Masterless (P2P)	Master /Slave	Master/Slave	Master/Slave
Data Store	Wide-column	Wide-column	Document	Key-value
Architecture	Peer-to-peer	Hierarchical	Nested (Hierarchical)	
Data Lake	Too Complex	Best Choice	OK	not typically used as a data lake.
IoT or Web	Best Choice	Lack of Record-Level Indexing	Best Choice	suitable for real-time analytics for IoT and web.
Text Data	Good	Good	Good	Good
Schema	Yes (can replace RDB)	No	No	No (schema-less)
AWS	Amazon offers "Amazon Keyspaces	Amazon EMR (Elastic MapReduce)	Amazon DocumentDB	Amazon ElastiCache
GCP	No native support, but can be deployed on GCP	Google's Bigtable	GCP **does not** have a native MongoDB, but "MongoDB Atlas", can be deployed on GCP.	Cloud Memorystore
Azure	Azure Cosmos DB" provides a Cassandra API	Azure HDInsight	Azure Cosmos DB provides MongoDB API	Azure Cache

Reading List

1. Database Wiki, https://en.wikipedia.org/wiki/Database
2. NoSQL: A Database for Cloud Computing
   https://pdfs.semanticscholar.org/f5ba/838ce31f14d83705ba9cc24ba1297ed1808d.pdf
3. SQL Databases v. NoSQL Databases http://delivery.acm.org/10.1145/1730000/1721659/p10-stonebraker.pdf?ip=130.102.82.27&id=1721659&acc=ACTIVE%20SERVICE&key=65D80644F295BC0D.5A1472836B5B8FB5.4D4702B0C3E38B35.4D4702B0C3E38B35&__acm__=1527074155_3df851f19f656db4a19a2d998ed62938
4. A performance comparison of SQL and NoSQL databases
   https://ieeexplore.ieee.org/abstract/document/6625441/
5. MongoDB Tutorial for Beginners
   https://www.guru99.com/mongodb-tutorials.html
6. Cassandra Tutorial for Beginners
   https://teddyma.gitbooks.io/learncassandra/content/index.html

References

1.Coronel, Carlos, and Steven Morris. *Database systems: design, implementation, & management*. Cengage Learning, 2016.

2.Why NoSQL? https://www.couchbase.com/resources/why-nosql

3.Introduction to MongoDB. https://docs.mongodb.com/manual/introduction/

4.https://www.tutorialspoint.com/cassandra/cassandra_architecture.htm

5.Stonebraker, Michael. "SQL databases v. NoSQL databases." Communications of the ACM 53, no. 4 (2010): 10-11.

6.Khan, Heena. "NoSQL: A database for cloud computing." International Journal of Computer Science and Network 3, no. 6 (2014): 498-501.

7.Li, Yishan, and Sathiamoorthy Manoharan. "A performance comparison of SQL and NoSQL databases." In Communications, computers and signal processing (PACRIM), 2013 IEEE pacific rim conference on, pp. 15-19. IEEE, 2013.

8.https://www.w3.org/TR/rdf-sparql-query/