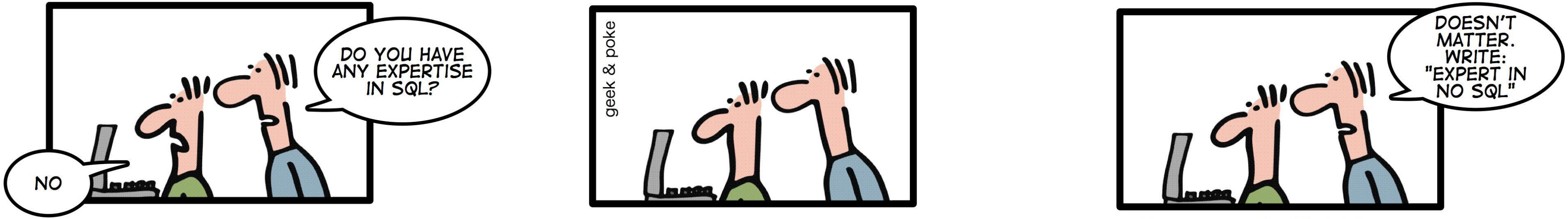
WILD CODE SCHOOL

Persistence Shootout

SQL vs. NoSQL vs. NewSQL - when to use which persistence?



- Relational DBMS and SQL
- RDBMS Spring Data Demo
- SQL and NoSQL, what’s the difference?
- A little theory: CAP, ACID and BASE
- Use Cases for Database Types

WILD CODE SCHOOL

Relational Database Management Systems (RDBMS)

Relational Database Management Systems (RDBMS)

MySQL®

MariaDB

ORACLE

PostgreSQL

SQLite

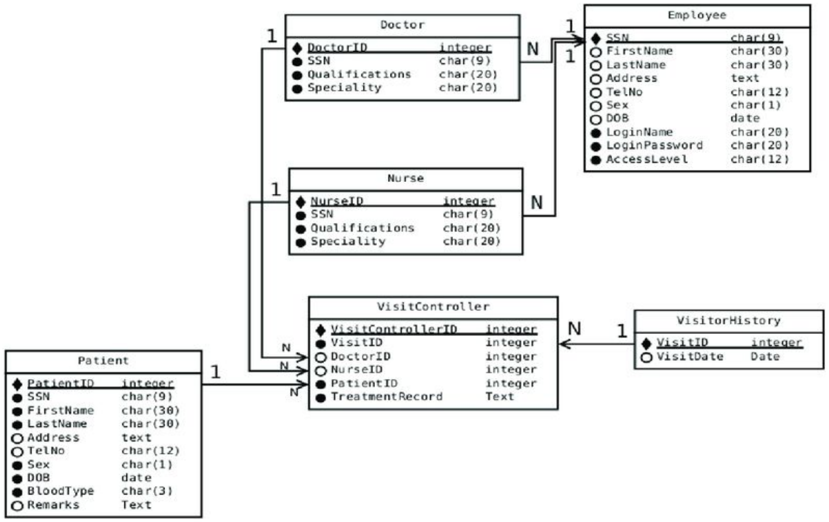
Microsoft® SQL Server®

IBM DB2

Introduction RDBMS (30k ft.)

Relational Databases are used to

- **Store** data permanently
- **Organize** data logically through **relationships**
- **Access and manage** data through **relationships** with **SQL** language



Almost like a Spreadsheet

Data is stored in **2D tables** whose intersections are called **fields**.

Each **row** is a **tuple** that has a unique identifier: the **primary key**.

Each **column** is a **set of data values of a particular type** and can also be interpreted as an **attribute.**

Knight
id lastname firstname age

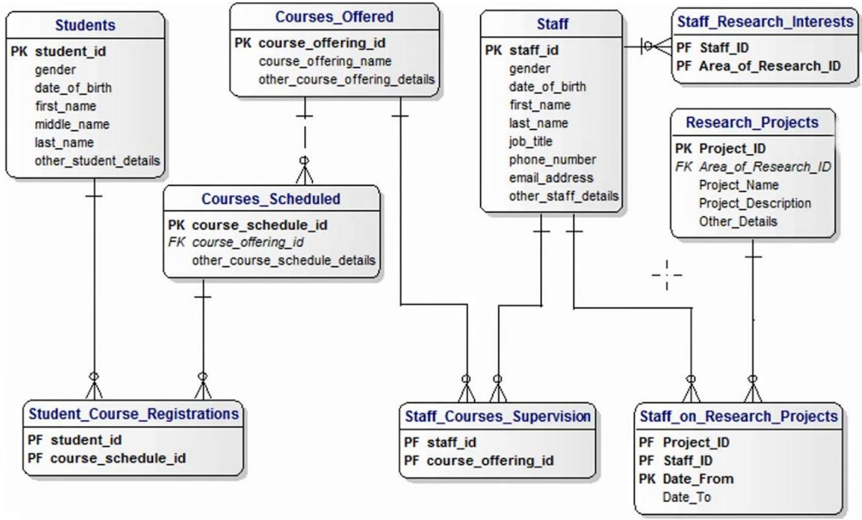
id	lastname	firstname	age
1	Pendragon	Arthur	40
2	Le gallois	Perceval	32
3	Du Lac	Lancelot	35

RDBMS Structure

Each entity contains only **directly-related information**. It is **linked** to other entities with **peripheral information**.

E.g.: only personal information is recorded in Students. Course information is stored in another entity. Courses and students will be connected via a relationship

Pros: No data duplication
Cons: Obligation to structure everything in advance



Normalization

After Normalization from 0-NF to 1-NF the Column **Dept** now has a clearly defined data type with only one value per row.

In the first form, how do you rename a department consistently?

EmpID	Employee	Age	Dept
1001	ABC	30	Sales,Finance
1002	CDE	30	Sales,Finance,DevOps

DeptID	DeptName
1	Sales
2	Finance
3	DevOps

EmpId	Employee	Age	DepID
1001	ABC	30	1
1001	ABC	30	2
1002	CDE	40	1
1002	CDE	40	2
1002	CDE	40	3

taken from: https://www.sqlshack.com/what-is-database-normalization-in-sql-server/

Properties of RDBMS

ACID: **A**tomicity, **C**onsistency, **I**solation, **D**urability

- ❖ **Atomicity** – completion of the transaction as a whole or none at all
- ❖ **Consistency** – assures the stable state of the database with or without changes
- ❖ **Isolation** – multiple transactions do not interfere with each other
- ❖ **Durability** – permanent effect on the database by the changes

Normalization: A process of designing databases with **several degrees** of **data integrity**

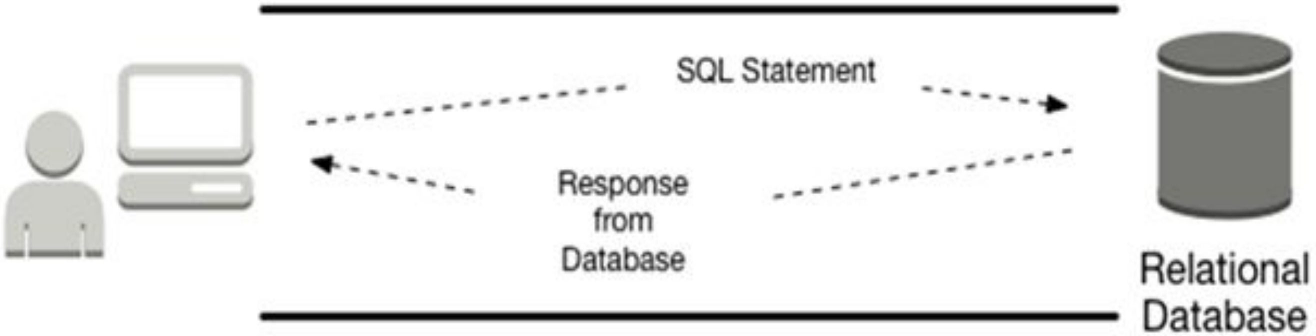
Scalability: RDBMS can scale-up, not scale out (**vertical scaling** vs. **horizontal scaling**)

Domains: **data types** with **optional constraints** (restrictions on the allowed set of values): support JOIN functionality, engineered for **data integrity**

Basics of RDBMS and SQL

SQL: Structured Query Language

- A standard, usable in all RDBMS
- Access to data and the storage structure of the data
- **Modify, update,** and **view data** (Data Manipulation Language, **DML**)
- **Modify** database **structure** (Data Definition Language, **DDL**)



Basics of SQL queries

What are you selecting?

* = all in SQL

From which table?

The name of the table

With what conditions applied?

AND is used for multiple statements

Queries end with a semicolon;

```
SELECT *
FROM customers
WHERE package = “Premium” AND
package_price = 100 AND
customer_location = ‘Seattle’;
```

Operators
=, > , >=
etc

The Relational Diagram

To model the **tables' fields, datatypes and relations** we use a relational diagram.

There are **3 types of relations**:

- one-to-one
- one-to-many
- many-to-many

Let's look at some examples!

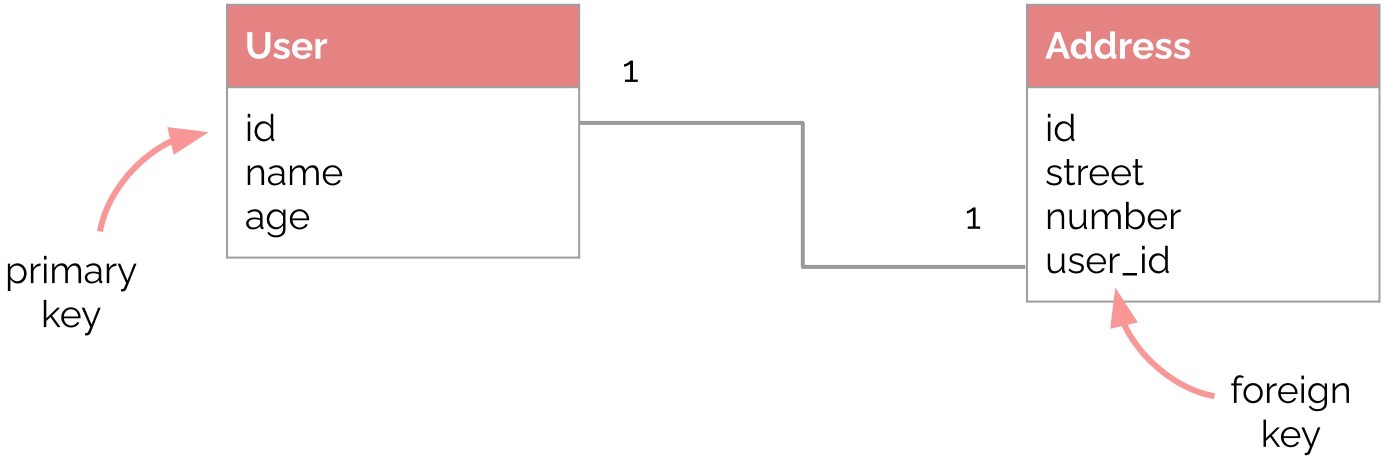
One-to-one relation

Let's start simply with the following data specs:

- **Users** have a name, and age
- Each User **has one and only one** address
- **Addresses** have street, number and city

How to model - One-to-one

Each User **has one and only one** address.
Called a one-to-one or **1:1 relation**.



How to model - One-to-many

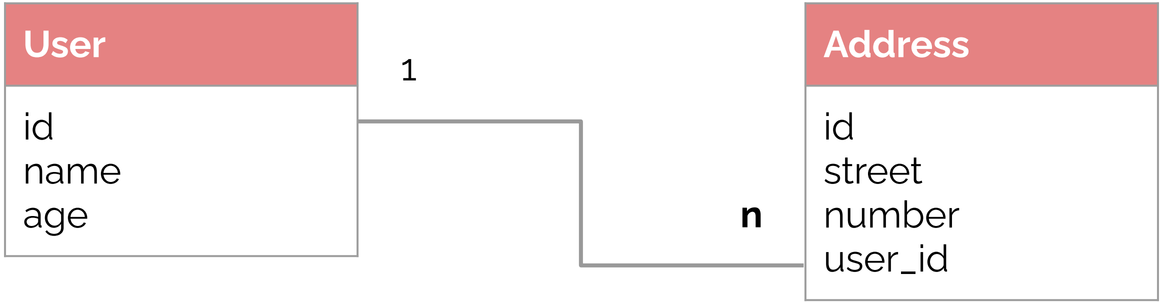
One-to-many relation

As with many modern apps like amazon, ebay, uber eats, etc:

- Users have a name, and age
- Each User **has many** addresses
- Addresses have street, number and city

How to model - One-to-many

Each User **has many** addresses. Called a one-to-many or **1:n relation**.



How to model - Many-to-many

Many-to-many relation

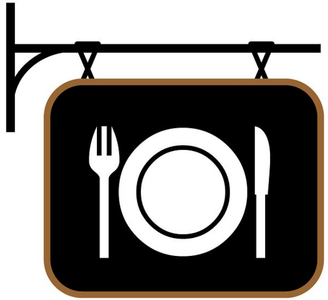
To learn about the last relation, let's imagine a delivery app for a restaurant.

The specs are:

❖ An **Order** has a date and user_id (let's not care about the users table)

❖ Each **Item** has name, price and description

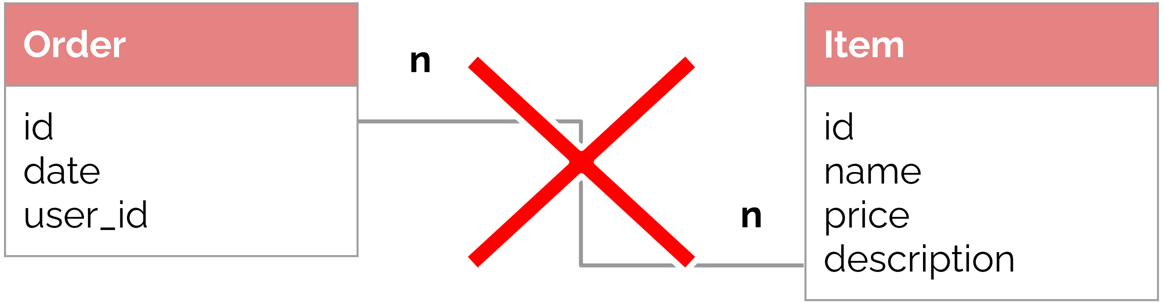
❖ An **order can have multiple items** and an **item can be in multiple orders**



How to model - Many-to-many

An **order can have multiple items** and an **item can be in multiple orders**

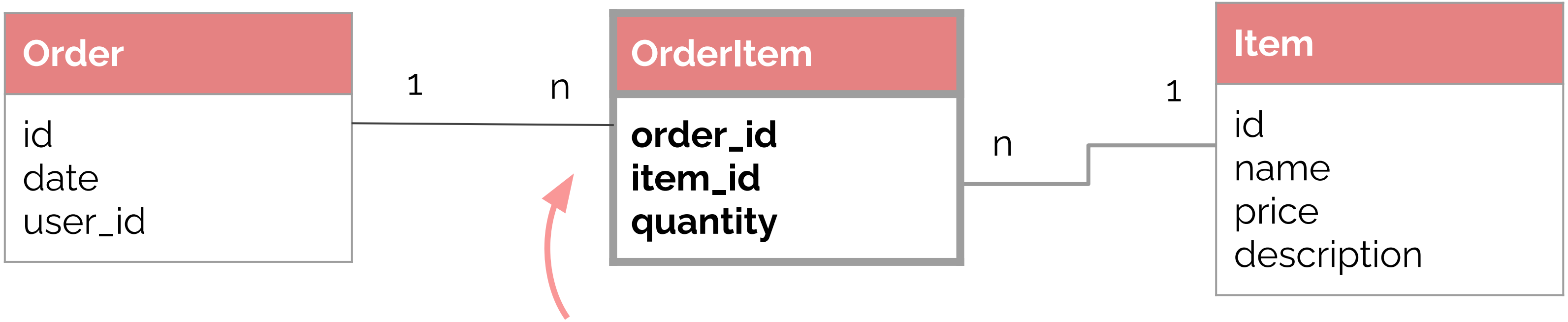
Called a many-to-many or **n:n relation**.



How to model - Many-to-many

No more n:m!

The extra table enables us to **get rid of n:m relations in favor of 1:n relations.**



Adding datatypes

It's useful to also specify the datatypes for each column in the diagram

User
id **INT AUTO_INCREMENT**
name **VARCHAR(100) NOT NULL**
age **INT NOT NULL**

1

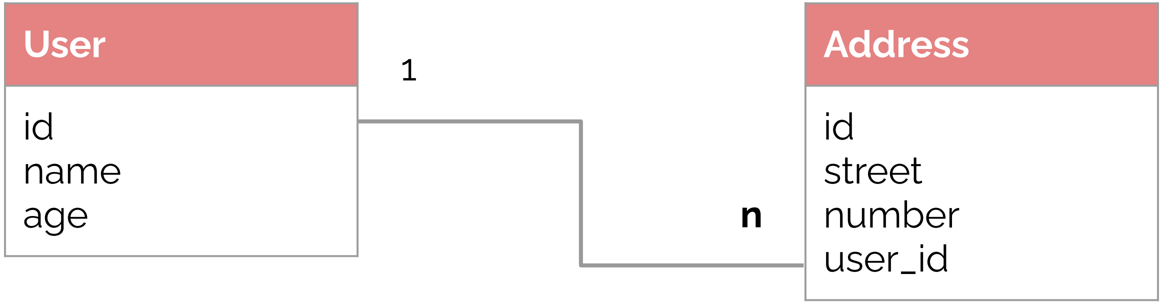
n

Address
id **INT AUTO_INCREMENT**
street **VARCHAR(140) NOT NULL**
number **INT NOT NULL**
user_id **INT**

Integrity constraints

What happens when related table records are **updated** or **deleted?**

Consider the following db:



What happens to an Address if I delete a User?

Integrity constraints

- **no constraints are defined**, the User is erased and the connected Addresses are *orphaned.*
- **ON DELETE NO ACTION** the deletion is refused because there are Addresses associated to that User
- **ON DELETE CASCADE** all Addresses associated will be automatically deleted, along with the User

```
mysql> CREATE TABLE address
     …
   PRIMARY_KEY(id)
   FOREIGN KEY (user_id)
     REFERENCES user(id)
     ON DELETE CASCADE
     ON UPDATE NO ACTION;
```

SQL Joins

Employee Table:

EmpID	EmpFname	EmpLname	Age	EmailID	PhoneNo	Address
1	Vardhan	Kumar	22	vardy@abc.com	9876543210	Delhi
2	Himani	Sharma	32	himani@abc.com	9977554422	Mumbai
3	Aayushi	Shreshth	24	aayushi@abc.com	9977555121	Kolkata
4	Hemanth	Sharma	25	hemanth@abc.com	9876545666	Bengaluru
5	Swatee	Kapoor	26	swatee@abc.com	9544567777	Hyderabad

Project Table:

ProjectID	EmpID	ClientID	ProjectName	ProjectStartDate
111	1	3	Project1	2019-04-21
222	2	1	Project2	2019-02-12
333	3	5	Project3	2019-01-10
444	3	2	Project4	2019-04-16
555	5	4	Project5	2019-05-23
666	9	1	Project6	2019-01-12
777	7	2	Project7	2019-07-25
888	8	3	Project8	2019-08-20

Client Table:

ClientID	ClientFname	ClientLname	Age	ClientEmailID	PhoneNo	Address	EmpID
1	Susan	Smith	30	susan@adn.com	9765411231	Kolkata	3
2	Mois	Ali	27	mois@jsq.com	9876543561	Kolkata	3
3	Soma	Paul	22	soma@wja.com	9966332211	Delhi	1
4	Zainab	Daginawala	40	zainab@qkq.com	9955884422	Hyderabad	5
5	Bhaskar	Reddy	32	bhaskar@xyz.com	9636963269	Mumbai	2

What project are employees working on?

SQL Joins

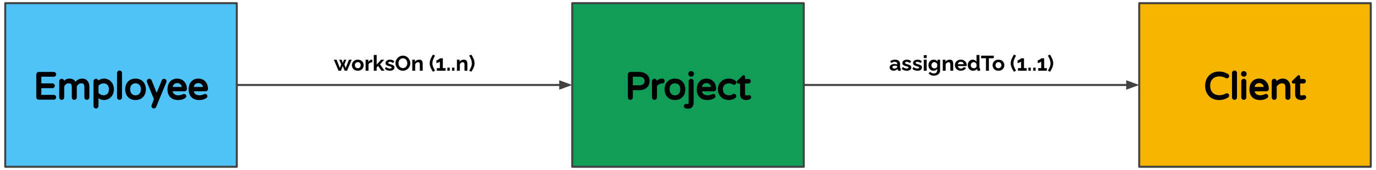
```
SELECT Employee.EmpID, Employee.EmpFname,
Employee.EmpLname, Projects.ProjectID, Projects.ProjectName
FROM Employee
INNER JOIN Projects ON Employee.EmpID=Projects.EmpID;
```

Output:

EmpID	EmpFname	EmpLname	ProjectID	ProjectName
1	Vardhan	Kumar	111	Project1
2	Himani	Sharma	222	Project2
3	Aayushi	Shreshth	333	Project3
3	Aayushi	Shreshth	444	Project4
5	Swatee	Kapoor	555	Project5

taken from https://medium.com/edureka/sql-joins-db6e76effac1

How to map a SQL Join result to OOP world?



Output:

EmpID	EmpFname	EmpLname	ProjectID	ProjectName
1	Vardhan	Kumar	111	Project1
2	Himani	Sharma	222	Project2
3	Aayushi	Shreshth	333	Project3
3	Aayushi	Shreshth	444	Project4
5	Swatee	Kapoor	555	Project5

?

```
class Employee {
  String id;
  String fullName;
  List<Project> worksOn;
}

class Project {
  String id;
  String title;
  Client assignedTo;
}

class Client {
  String id;
  String name;
}
```

ORM Impedance Mismatch

Tools & Libraries

Towards more
Business Logic

JDBC

object-relational mapping (ORM)
to be implemented by hand

Business Focus

```
public Car getByIds(String id) {
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    try {
        String sql = "select * from CAR where ID = ?";
        conn = DriverManager.getConnection();
        stmt = conn.prepareStatement(sql);
        stmt.setString(1, id);
        rs = stmt.executeQuery();
        if (rs.next()) {
            Car car = new Car();
            car.setMake(rs.getString(1));
        } else {
            return null;
        }
    } finally {
        try (
            if (rs != null) {
                rs.close();
            }
        } catch (Exception e) { }

        try (
            if (stmt != null) {
                stmt.close();
            }
        } catch (Exception e) { }

        try (
            if (conn != null) {
                conn.close();
            }
        } catch (Exception e) { }
    }
}
```

Hibernate

ORM is done automatically, but
DAO remains to be implemented
by hand

Hibernate
+
Spring Data JPA

ORM does it automatically: DAO is
implemented automatically

JPA

Standards & Specifications

JPA (Java Persistence API)

Set of Standards

Entities

Purpose: correspondence between the Java object and the SQL table

```java
@Entity
public class Question {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

}
```

eclipselink

OpenJPA

Hibernate

Repository (DAO)

Purpose: methods for CRUD operations

save()
findById()
findAll()
delete()

...

Spring Data JPA

NoSQL Databases

NoSQL (Not Only SQL) DBMS

cassandra

elasticsearch

neo4j

redis

mongoDB®

APACHE HBASE

Principles of NoSQL

Lack of Schema: **Flexible** and **lightweight** for development

Scalability: NoSQL can **scale-up and scale out** and were built for scalability

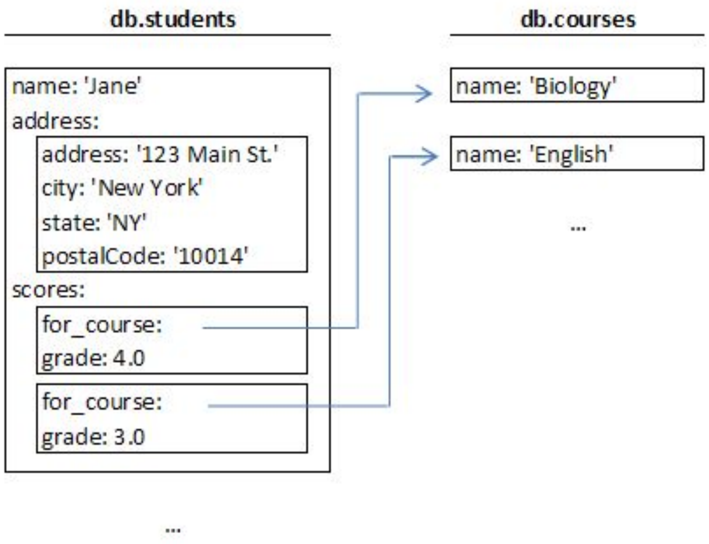
Specialized: Different flavors of NoSQL are **optimized for special use cases**

Big Data: Optimized for usage of **huge data volume** with **semi- or unstructured data**

Concepts of NoSQL

Each entity contains all the information it is related to.

E.g.: A student will have his personal information, as well as the list of courses he is attending.



Pros: All information is retrieved in one call

Cons: Possibility of duplications

These are database systems that are not (only) SQL. They are often appreciated for the **flexibility** of the **data schema** and the **simplicity** of **horizontal scaling**.

Different types of NoSQL

Key-Value

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

redis

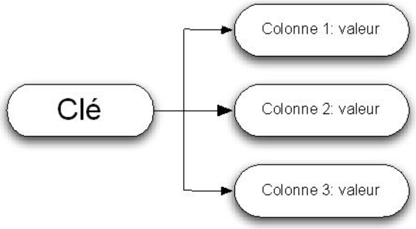
Document Based

```
{
  $id :          "Customer:04b24313-f210-4f0-989c",
  $type :        "entity",
  $table :       "Customer",
  C_ID :         "04b24313-f210-4f0-989c",
  C_FNAME :      "Homer",
  C_LNAME :      "Simpson",
  C_BANKACCOUNT : {
    IBAN :         "9876543210001234456",
    BIC :          "BICXXX",
    CREDITCARD:    "123456"
  }
}
```

mongoDB

elasticsearch

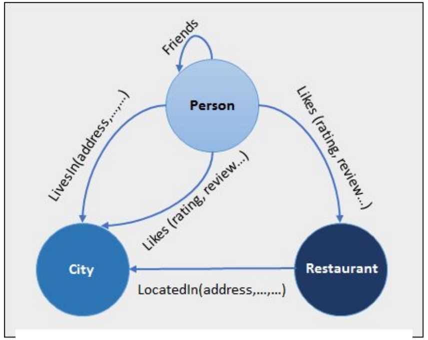
Column Based



APACHE HBASE

cassandra

Graph Based



neo4j

Good for the flexibility of the data schema and the simplicity of horizontal scaling.

NoSQL

```
knight : {
     $arthur : {
          name: Arthur Pendragon,
          age : 40,
          kingdom : {
               name : Logre,
               capital : Camelot,
               inhabitants : 100000
          }
     }
}
```

SQL

knight

id	name	age	kingdom
1	Arthur Pendragon	40	1
2	Léodagan	60	2

kingdom

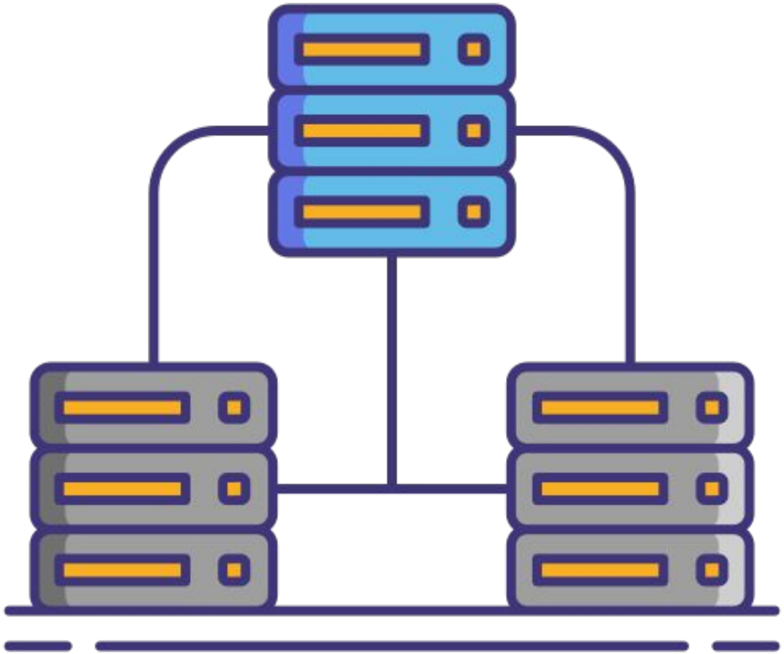
id	name	capital	inhabitants
1	Logre	Camelot	100000
2	Carmélide	Carohaise	50000

SQL vs NoSQL

Normalization

Normalization is a key principle of SQL databases

- **Avoids redundancy:** the information is written once and then referenced using a foreign key.
- **Fosters database maintenance:** the lack of redundancy allows information to be updated more quickly.



-> Basis of the join system

Denormalization in NoSQL

Denormalization

Due to the absence an efficient join system, denormalization is often applied when modeling NoSQL databases.

This is the exact opposite of normalization:

- ❖ Data is redundant
- ❖ Fosters quick access times

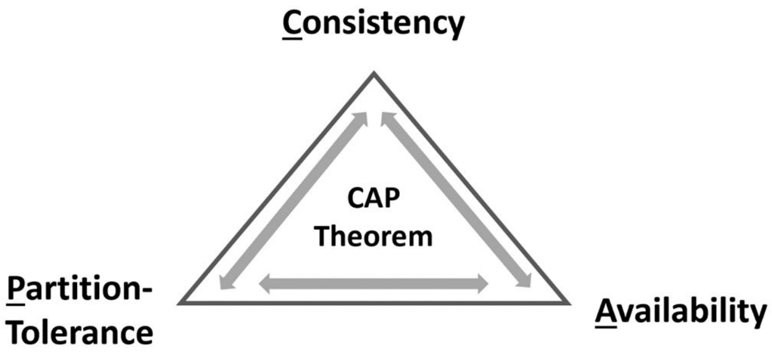
-> compromises storage size

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

CAP Theorem and CAP Samples



AP – Domain Name System (DNS) or Cloud Computing

DNS can be classified as **AP**. Availability must be very high, also partitioning of the network must be tolerable. Therefore consistency cannot be guaranteed, DNS changes are not visible immediately.

CA – Relationales Datenbank Management System (RDBMS)

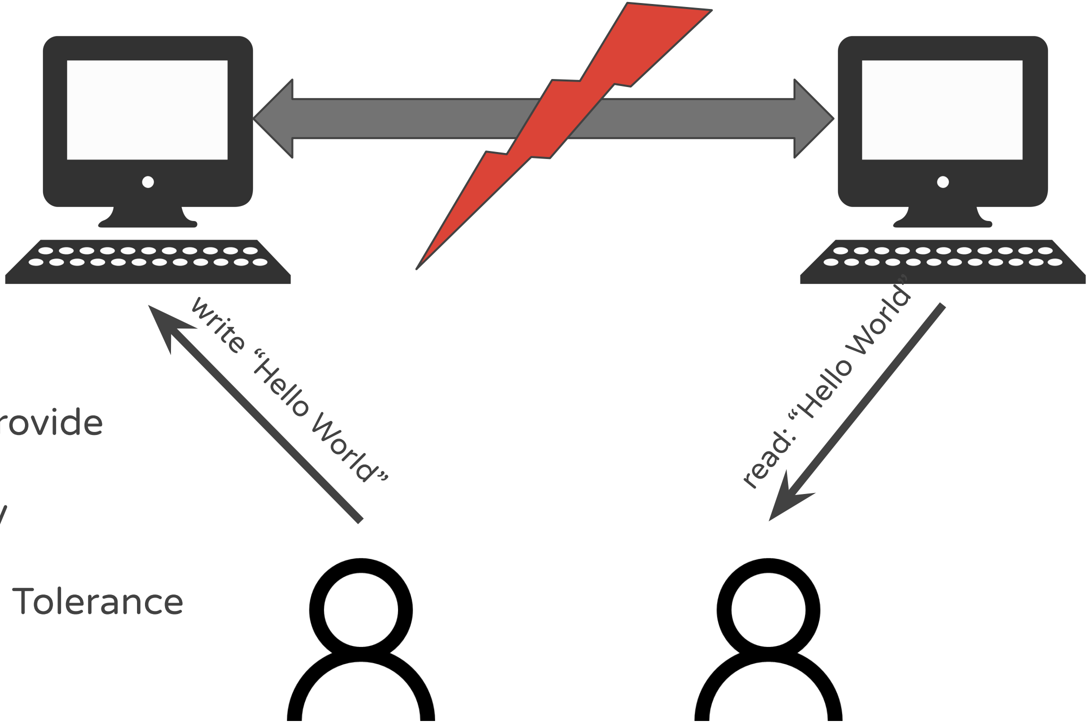
RDBMS like Oracle or DB2 aim for the highest possible degree of consistency and ca be classified as **CA**. RDBMS should be always available and consistent. However, only vertical scaling is used.

CP – Transaction processing in Finance

For distributed finance applications like ATM consistence has highest priority: a transaction must always be sound and complete. This should even hold in case of network partitioning. Therefore, availability is of lesser importance, so the classification is **CP**.

taken from https://data-science-blog.com/blog/2021/10/14/cap-theorem/

But what is it? Prove the CAP Theorem!

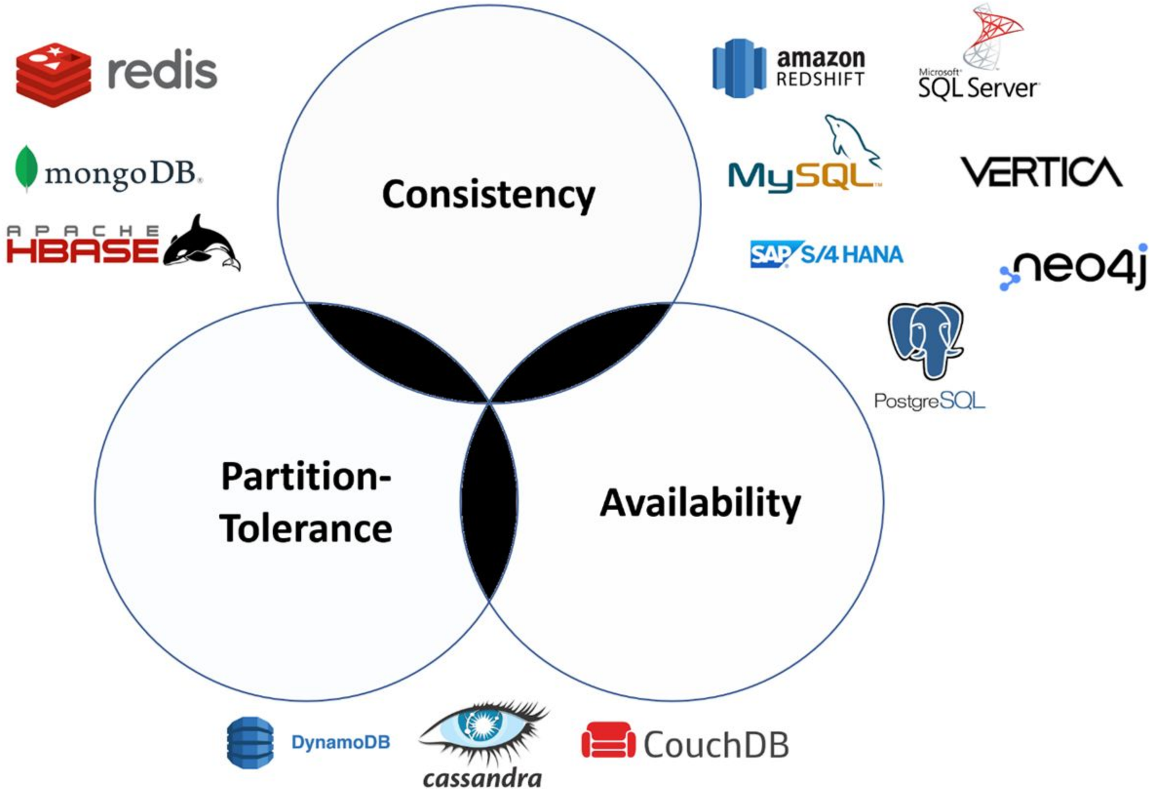


System cannot provide

- ❖ Consistency
- ❖ Availability
- ❖ Partitioning Tolerance

at the same time

CAP Classification of different Databases

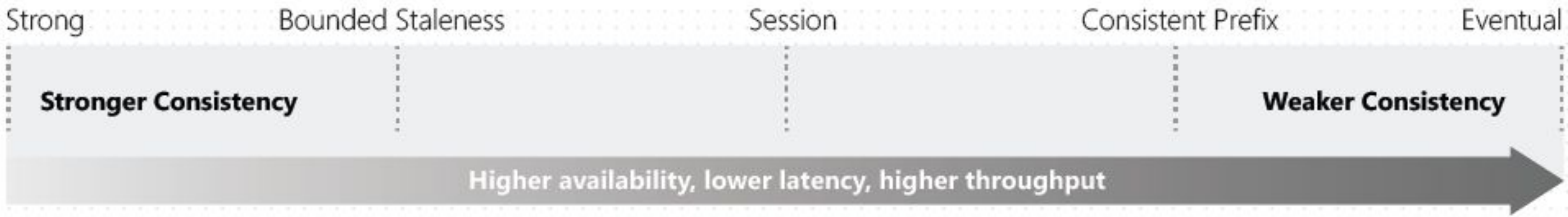


taken from https://data-science-blog.com/blog/2021/10/14/cap-theorem/

CAP with Gradual Properties

There can be **different levels** of **Consistency**

- ❖ Strong
- ❖ Bounded staleness
- ❖ Session
- ❖ Consistent prefix
- ❖ Eventual



taken from: https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

SQL vs NoSQL: Five Main Differences (1)

Language

- SQL has been around for over 40 years, is **highly established** and with **huge adoption and market share**
- NoSQL databases have unique, **optimized data manipulation languages**, which are **constrained by particular structures and capabilities**.

Scalability

- SQL databases typically use **vertical scaling** which aligns more to schemata, relational structure and constraints
- NoSQL technologies are typically lacking schemata, relations and constraints, **horizontal scaling** fits naturally

Structure

- SQL database schemata always represent **relational**, **tabular data**, with rules about **consistency and integrity**.
- NoSQL databases need not stick to this format, but generally fit into one of four broad categories:
    - **Column-oriented databases** transpose row-oriented RDBMSs, allowing efficient storage of high-dimensional data and individual records with varying attributes.
    - **Key-Value stores** are dictionaries which access diverse objects with a key unique to each.
    - **Document stores** hold semi-structured data: objects which contain all of their own relevant information, and which can be completely different from each other.
    - **Graph databases** add the concept of relationships (direct links between objects) to documents, allowing rapid traversal of greatly connected data sets.

SQL vs NoSQL: Five Main Differences (2)

Properties

At a high level, SQL and NoSQL comply with **separate rules for resolving transactions**. RDBMSs must exhibit four **ACID** properties

- **Atomicity** means all transactions must succeed or fail completely. They cannot be partially-complete, even in the case of system failure.
- **Consistency** means that at each step the database follows invariants: rules which validate and prevent corruption.
- **Isolation** prevents concurrent transactions from affecting each other. Transactions must result in the same final state as if they were run sequentially, even if they were run in parallel.
- **Durability** makes transactions final. Even system failure cannot roll-back the effects of a successful transaction.

NoSQL technologies adhere to the “CAP theorem” and implement the **BASE** properties:

- **Basically available**: reading and writing operations are available as much as possible (using all nodes of a database cluster), but might not be consistent (the write might not persist after conflicts are reconciled, the read might not get the latest write)
- **Soft-state**: without consistency guarantees, after some amount of time, we only have some probability of knowing the state, since it might not yet have converged
- **Eventually consistent**: If we execute some writes and then the system functions long enough, we can know the state of the data; any further reads of that data item will return the same value

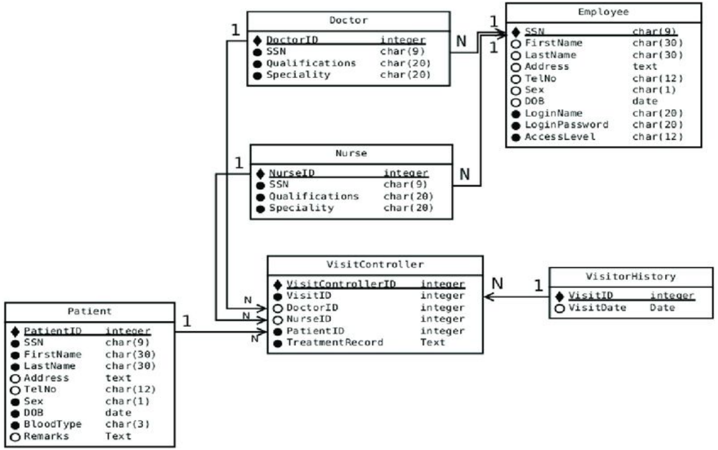
SQL vs NoSQL: Five Main Differences (3)

Support and communities

- SQL databases represent **massive communities**, **stable codebases**, and **proven standards**. Multitudes of examples are posted online and **experts are available** to support those new to programming relational data.
- NoSQL technologies are being adopted quickly, but communities **remain smaller and more fractured**. However, many SQL languages are proprietary or associated with large single-vendors, while NoSQL communities **benefit from open systems** and concerted **commitment to onboarding users**.

SQL Databases

- ❖ Data stored in **Entity-Relationships**
- ❖ **Structured data schema**, uses **SQL**



NoSQL Databases

- ❖ Database **without SQL**
- ❖ **Unstructured** or **semi-structured data schema**

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
          }
}
```

Embedded sub-document

Embedded sub-document

SQL vs NoSQL

SQL is best suited if

- the data structure can be identified in advance
- data integrity is essential, and more important than speed.
- the transactional nature is strongly present

Example: A slow trading site, but whose stock is calculated in real time between all servers for all products.

NoSQL is best suited if

- the structure of the data is independent, indeterminate and scalable.
- the structure requires a high degree of agility
- speed trumps integrity

Example: A fast e-commerce site, but with inconsistent inventory contingencies.

NewSQL Databases to the Rescue?

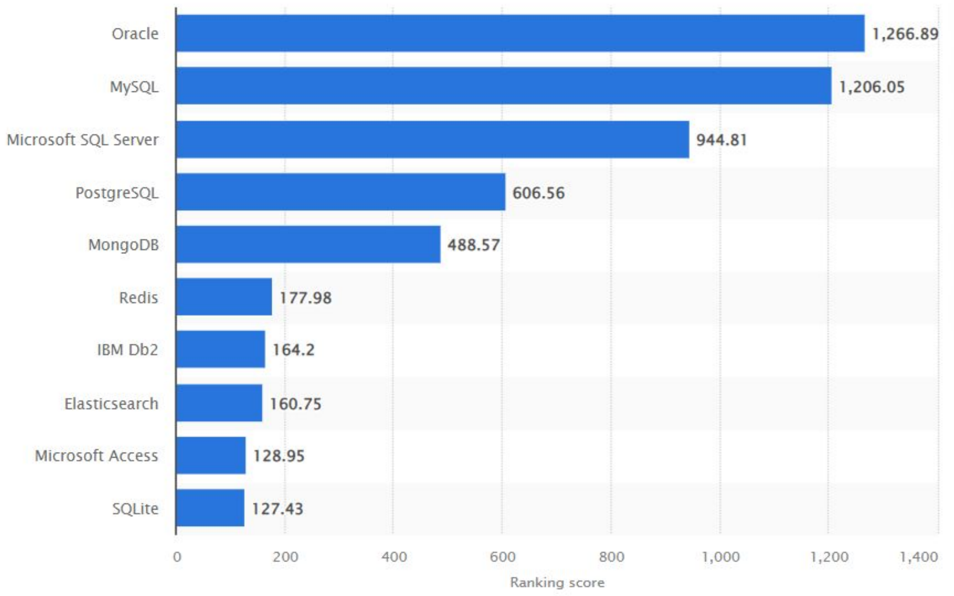
What is NewSQL?

The “**NewSQL**” is a movement promoting a new type of relational databases (or extensions for existing relational databases). It seeks to provide the same **scalable performance** of NoSQL systems but it’s still based on a **relational paradigm** and it keeps the good old **SQL** as the query language. Moreover it guarantees **ACID** transactions (Atomicity, Consistency, Isolation, and Durability).

	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No

taken from: https://labs.sogeti.com/newsql-whats/

What is NewSQL?



“There is a **good reason why there are so many different kinds of databases**. If individual categories tend to break down further and further into custom build solutions to specific use cases. This is because data is incredibly varied, and it **often pays to deploy different kinds of technology for different kinds of data**.”

taken from: https://arctype.com/blog/newsql/, diagram: statista.com

Important for all Databases: Use indexes!

> "An index is an automatically maintained structure that allows records to be easily located in a file.
>
> The use of indexes is based on the following observation: to find a book in a library, instead of examining each book one by one (which corresponds to a sequential search), it is faster to consult the catalogue where they are classified by theme, author and title. Each entry in an index has a value extracted from the data and a pointer to its original location. A recording can be easily retrieved by searching for its location in the index."
>
> Credit: Wikipedia

The indexes depend on the DBMS used. They drastically improve access to data. Some indexes have limited functionality, you have to index the data according to the queries that will be made to consult them.

There are dozens of index types: Hash, GeoSpatial, Text, Compound… (Example: List of MongoDB indexes, PostgreSQL indexes)

Warning: Having too many indexes can impact the database performance, especially during POST operations.

Links and other information

CAP Theorem

- A Beginners Guide to CAP Theorem: https://www.analyticsvidhya.com/blog/2020/08/a-beginners-guide-to-cap-theorem-for-data-engineering/
- A Critique of the CAP Theorem: https://jvns.ca/blog/2016/11/19/a-critique-of-the-cap-theorem/
- Different Consistency Levels: https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels
- CAP Theorem: https://data-science-blog.com/blog/2021/10/14/cap-theorem/

Types of NoSQL Databases

- Understanding the Differences: https://phoenixnap.com/kb/nosql-database-types

Links and other information

Criticism of SQL

- Why Relational Databases are not the Cure-All. Strength and Weaknesses: https://phauer.com/2015/relational-databases-strength-weaknesses-mongodb/

Cypher vs. SQL

- Transitive Records in SQL and Cypher: https://dzone.com/articles/finding-directtransitive-reports-in-sql-and-neo4js

NewSQL

- Too good to be true: https://arctype.com/blog/newsql/

Hybrid Models

- JSON with PostgreSQL: https://blog.sql-workbench.eu/post/json-path/