databases, data engineering & big data

Introduction and main concepts

ESME SUDRIA

Luc Marchand - luc.marchand.pro@proton.me
Maxence Talon - maxencetallon@gmail.com

About me

Data Engineer / Backend Developer Consultant @ LUXCORP / HYPERCHANGE

luc.marchand.pro@proton.me

- 6 years at Octo Technology
- Work independently now
- 5th year as a teacher at ESME
- Main topics of interests
  - Kafka & streaming world
  - data engineering & backend programming
  - Best practices / Software craftsmanship
  - drums & music

If you have any question, feel free to drop me a mail at any time

Course Methodology

- This course features academic lectures and a long and modular practical work
  - All the practical works will be graded and the final course is a practical work as a final exam
- Academic lectures closely follow the course material
  - Questions are welcome throughout, do not hesitate to interrupt !
  - Course material is allowed during the exam and the practical exam
  - You you can use generative AI to help you
- Practical work will occur on the second part of the course
  - The context will be the same through all the practical work sessions
  - The idea is to show you a (kind of) real architecture to allow you to understand all the parts of a data architecture
- In the real worl, when you work as an IT / data engineer, there is no real exam but team work and sometimes dojos.
- All the course will use some specific technologies as practical examples but we will introduce you each concept and key points

Quick poll

Who has ever worked with databases before ?

Which technology ?

Where can I get this support ?

You'll find the support on http://databases.esme.s3-website.eu-west-3.amazonaws.com

TPs & TDs will also be available there as well as their answers

Global Syllabus

01 **Introduction and main concepts**

02 SQL, set up env and practical work

03 NoSQL world

04 Introduction to Big Data & Data Engineering

05 Kafka & event driven architectures

06 Spark & Delta

07 Warehouse, DBT & BI

08 IA - MLOps & RAG

Course syllabus

01 What is a database ?

02 History of databases

03 High level concepts

04 Who uses databases ?

05 DataBase Management Systems (DBMSes)

06 Usage & architecture

01

What is a database ?

There are multiples definitions

An organized collection of data.The data are typically organized to model relevant aspects of reality in a way that supports processes requiring this Information
- Wikipedia

A collection of data that represents some objects from the real world, and used as a support to a computer application.
– Georges Gardarin

A structured collection of data that is made accessible to a computer, in order to satisfy several users simultaneously in a timely fashion.
– Claude Delobel

A structured collection of data that is accessible from several users in a selective manner.

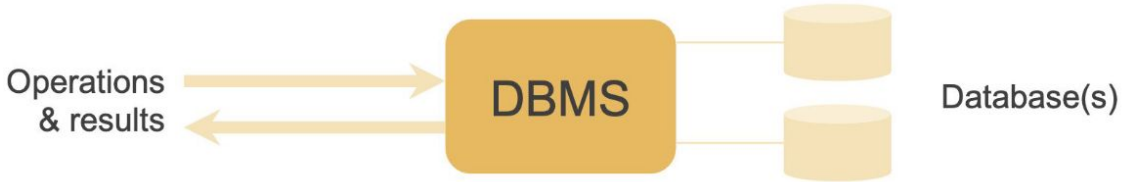
A collection of data managed by a DBMS and associated to a single application.

Our definition

- We adopt a more generic definition

A database is a collection of data that is accessible from a common entry point

- All the other aspects (storage medium, list of operations, concurrent access, existence of a computer application, ...) will depend on the context; most of them are under the responsibility of a **D**ata**B**ase **M**anagement **S**ystem (DBMS)
  - Examples of DBMSes : Oracle, MySQL, MongoDB, ...
  - An operating system is also a kind of DBMS, through its filesystem APIs



The French abbreviation for DBMS is SGBD (**S**ystème de **G**estion de **B**ases de **D**onnées)

02

Databases history

Late 50s', early 60'

In those days...

- Storage took place on magnetic tapes
- Data inpud used to be done with punch cards

Data was managed as files, with several limitations:

- Little flexibility in data structures
- Sequential access only (tapes)
  - Data had to be read or written in a predetermined order
- As a consequence, data was often redundant and inconsistent
- The applications had to take care of physical data access themselves
- No atomicity of data updates
  - Example: debit and credit of a same amount on two different accounts
- Simultaneous access by several users was hard to implement
- No security policy (access rights, confidentiality)

Late 60s', early 70'

In those days...

- Hard disks allowed direct access to random pieces data - no more sequential access

The navitation models (network & hierarchical) were the most common ones

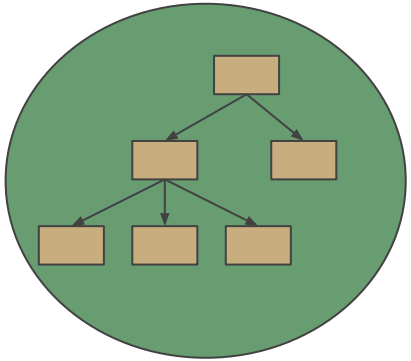
- Data navigation used abstractions such as "pointers" and "paths"

Limitations

- More complexity as the amount of data increases
  - Increase in the number of pointers
  - Possible redundancy (and thus inconsistency) with the hierarchical model
- No standard interface or language for defining and manipulating data
- Rigid data structures
  - Moving relationships around between entities was hard
  - The model was not fit for modelling many-to-many relationships
- Hence, complexity of developing applications on top of such databases
  - Besides, the access path the entities had to be coded into the applications

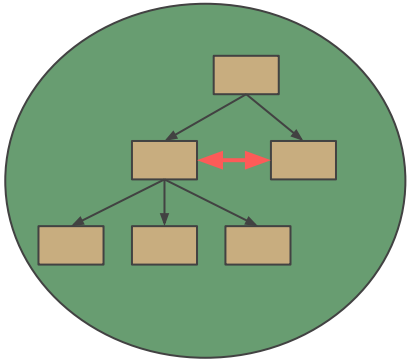
Several modern NoSQL databases feature network (graph) and hierarchical (document) models. Some limitations above still apply - they are no more constraint brought by technology but are considered a trade-off for performance. Beside, rigidity is less of an issue because many NoSQL DBMSes offer alternative access methods to pieces of data.

Database models from 1970s



Hierarchical

- **Only one owner per record**
- Information can have a complex structure (types, lists, nesting)



Network

- **The consumer needs to know the access path**
- e.g. relationship between people to model a social network

Late 60s', early 70s' (cont'd)

- Edgar Franck and Ted Codd formalize the relational model
  - Ted Codd received the Turing Price in 1981 for his work

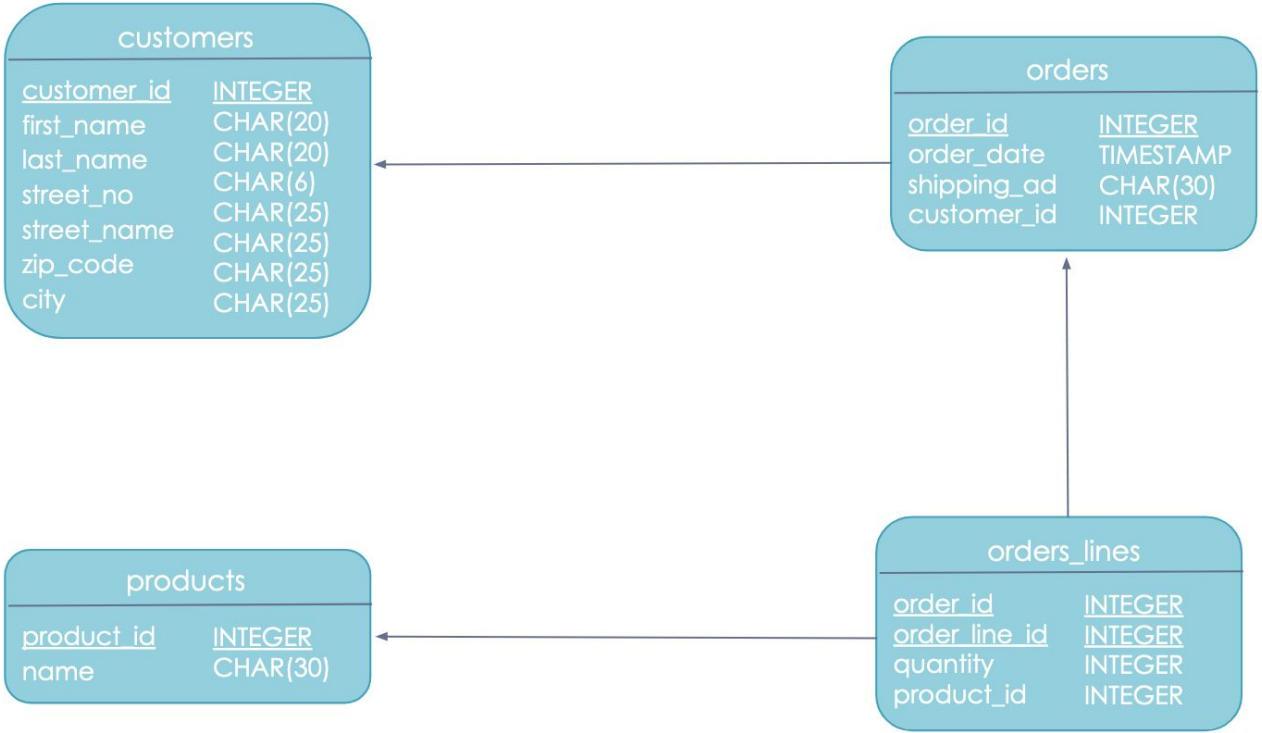
- Prototypes for relational DBMSes appear
  - System R at IBM
  - Ingres at UC Berkeley

- Pros of the relational model
  - Relational modelling is simple - everybody understands tables
  - SQL (Structured Query Language) allos one the query data without knowledge of its physical layout on disk

<u>Limitations</u>

- Legibility: the simple model ("Everything is a table"), though versatile, makes some relationships less apparent (e.g. hierarchies, inheritance)
- With the upcoming of object-oriented language, the "impedance mismatch" between the relational and object models makes data mapping difficult

Example of a relational model



80s’

- Prototypes initiated in the 70s give way to commercial products : Oracle, Ingres, Sybase, ...
  - SQL is becoming an industry **standard**

- Beginning of parallel and distributed DBMSes

- Beginning of object-oriented DBMSes, with limited success
  - Main database vendors are reluctant to implement this new approach; the market sticks to the relational model
  - Steep learning curve
  - Low performance of the implementations the available
  - Maintenance operations are slow
  - No standard, hence a strong dependency on the vendor APIs
  - The object-relational model, described later, proved simpler and more valuable
  - Today, ORM and NoSQL datastores are much more common

90s’

- Appearance of *business intelligence* and *data mining* applications
- Appearance of large *data warehouses* (several terabytes)
- Emergence of the e-commerce industry
- Appearance of the object-relational model, to overcome some limitations of the relational model

Such limitations are :

  - Lack of pointers easing navigation in the model
  - Lack of complex types (user data structures, collections, inheritance, ...)
  - Lack of operations built into the model (i.e. methods)

The object-relational model (OR model) is not to be confused with object-relational mapping (ORM)! Despite the limitations overcome by the OR model, the relational model is still widely used today. The ORM offer an alternative workaround to those limitations, and addresses them in the application layer – not in the data layer as the OR model would do.

2000s’

- Database administration is more and more automated
- XML and related standards (XQuery...) are thriving : XML database appear
- But XML databases never really made it
  - Most relational databases now feature XML extensions,
  - When needed, XML is usually stored in the database as raw text or BLOBs

Late 2000s', 2010s'

- E-commerce, social media and online advertising are thriving : Amazon, Google, e-Bay, Yahoo!, Facebook, Twitter, ...
- For those "Internet giants", the relational model is completely unable to handle their traffic and the volume of data they generate - performance is a crucial issue directly linked to their revenue

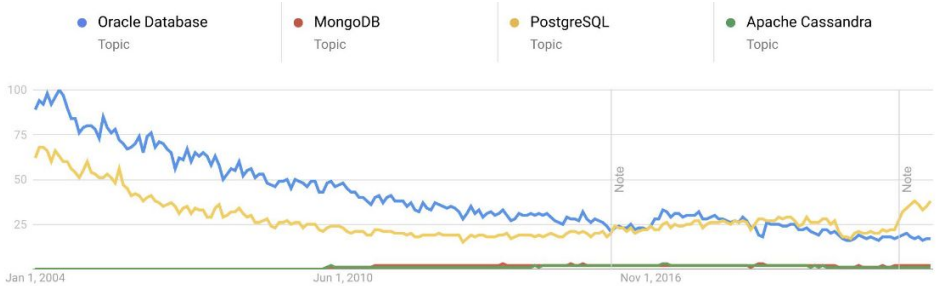
In 2006, Facebook has deployed a click stream monitoring, how much data do they get in one day ?

- NoSQL datastores appear in those years, as alternatives to traditional (relational) DBMSes
- Emergence of Big Data
  - Today's drivers for Big data: social networks, personalized experience, M2M (Machine-to-Machine traffic), IoT (internet of Things)

Relational databases are still widely used today and won't disappear anytime soon.
The mentality has shifted from "one size [relational] fits all" to "the right tool for the job"

2015 - 2017

- We see new storage patterns like the blockchain
  - The storage is massively distributed with distributed consensus
- NoSQL databases market is growing and there is a professional supports assured bu an editor behind every NoSQL systems (MongoDB, Cassandra, ...)
- Data Lake pattern appears and become a trend, at company scale
- The DMBSes are **commodities**. We don't look anymore as strategic assets.



We see a change in the mindset of companies. Board directories see the IS as a new vehicle for value creation. **The mentality has shifted from "cost reduction at every level" to "investment in innovation and new usages"**. We talk about digitalization.

Example of NoSQL data models

Document-oriented (e.g. MongoDB)

OBJECTS

```
{ '_id': 123456,
  'type': 'product',
  'name': 'computer',
  'features': {
     'cpu_GHz': 3,
     'ram_GB': 8,
     'brand': 'Dell'
  }
}
```

```
{ '_id': 123457,
  'type': 'product',
  'name': 'blender',
  'features': {
     'rpm': 10000,
     'voltage': '220V 50 Hz'
  }
}
```

```
{ '_id': 123458,
  'type': 'user',
  'login': 'choupi92',
  'password': 'AZnx403==',
  'shopping_history': [...]
}
```

Column-family aka BigTable (e.g. Cassandra)

PRODUCTS

_id	name	features		
123456	computer	cpu_GHz=3	ram_GB=8	brand=Dell
123457	blender	rpm=10000	voltage=220V 50 Hz	

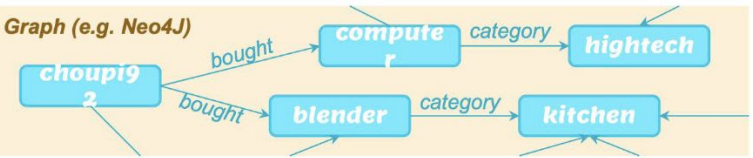
USERS

_id	authent		shopping_history		
123458	choupi92	password=AZnx403==	08/09/13=...	10/09/13=...	...

Key/Value pairs (e.g. Redis)

obj_123456	"type=product;name=computer;cpu_GHz=..."
obj_123457	"type=product;name=blender;rpm=10000..."
obj_123458	"type=user;login=choupi92;password=..."

Graph (e.g. Neo4J)



2017-2020

- DistributedSQL databases make their apparition
- The return of data warehouses despites hype around data lake pattern.
- Event driven architectures make their appearance
  - Confluent Kafka is one of the event architecture drivers
- Cloud computing is rising up. Move to cloud strategies are settled by companies to accelerate.

2020 - now

- MLOps patterns become trendy
  - In fact, 80% of data science projects never go to production

- databases-as-a-service (dbaas) services help a lot to create and manage multiple kinds of databases. They are a **de facto standard** to deploy and use databases.

- Modern SQL PRQL

- LLM and generative AI solutions arrive close to the developers
  - Github Copilot, Jetbrains AI, Cursor, ChatGPT, ...

To keep in mind

History of databases

1970s

- DBMS offers an abstraction to ensure **physical data independence** and expose data through a **conceptual data model**
- everybody understands tables

1980s

- Relational model helps to model complex business domain in a standard way.

2010s

- DBMS are **commodities**. They are easy to deploy, to maintains and requires less expertise that in the past
- Each DBMS has a learning curve for your organization.

2015s

- New paradigm as the Big Data or the Blockchain offer new ways of thinking about data. At the end, some concepts are still present as the research of physical data independence.
- Data are omnipresent and cross the border from the information system. It requires great effort to manage them.

2020s

- Data warehouses and data lakes are common design pattern in company but data management is still an area of research ;-)
- Cloud is a de facto standard when deploying / maintaining / using databases
- SQL is still the most principal approach to query data
- MLOps combine the both between data science & data-engineering

03

High-level concept

Structuration of data

Data comes in 3 flavors : structured, semi-structured and unstructured.

Structured

- The data conforms to a predetermined structure
- The structure is a strong invariant of the data model
- All records of the same entity type share the same format
- **Examples**: tables in the relational model, XML with DTD/XSD
- **Pros**
  - Easy processing since the structure is known in advance
  - Data quality is enforced
- **Cons**
  - Too restrictive for some types of data (emails, documents scanned as images, …)
  - When the structure changes, what about existing data still in the old format?

Unstructured

- The data has no structure; the only known type is 'object'
- Objects are considered opaque buckets of bytes, sometimes called BLOBs (Binary Large OBjects)
- **Examples**: raw files, data streams
- **Pros**
  - Can store anything including heterogeneous or changing data
- **Cons**
  - There is no metadata for the structure, so data cannot be interpreted without an accompanying program that knows how to read and write them

Semi-Structured

- A kind of hybrid between the first two flavors
- One still deals with unstructured objects, but they bear some meaningful metadata
  - Example metadata: tags (an object can have as many tags as necessary), relationships
- **Examples**: XML and JSON documents in general, objects from most NoSQL databases, documents in a search engine
- **Pros**
  - The metadata give meaning to objects, without the rigidity of a full structure (compare Gmail tags with Outlook folders ☺)
- **Cons**
  - Besides metadata, the contents of the objects themselves are unstructured so one still needs to know how to interpret them

Schema

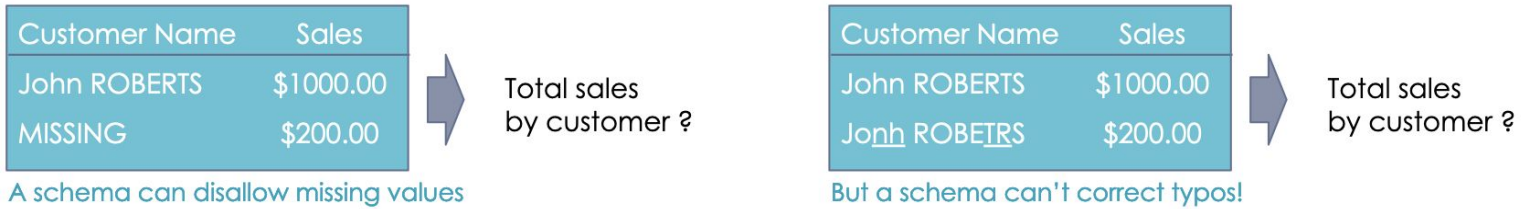
- A schema is an invariant on the data managed by a system : it imposes constraints on the data.
  - A fixed number of types (usually of entities manipulated in the business domain)
    - Car - Road - Driver
  - A fixed set of attributes for each type
    - Car: brand, color, wheels, licence plate #, owner
    - Road : name, GPS positions, traffic condition
    - Driver : identity, licence #, cars owned
  - Additional constraints that apply to entities, attributes and their relationships
    - The driver must be 18 years old
    - Driver's licence # is mandatory
    - A car is owned by 0, 1 or 2 drivers

STRUCTURED	UNSTRUCTURED	SEMI-STRUCTURED

• The schema is usually stored along with the data, so the DBMS can **enforce** it	• The data bears no schema, so the schema is **implicit** and encoded in the programs that manipulate the data	• There are **2 levels** of schemas
		  • The metadata schema, usually very simple, is managed by the DBMS (or search engine)
		  • The implicit schema for the unstructured payload, is encoded in programs

Data quality

- We have seen that only structured data and (to a lesser extent) semi-structured data have a schema that constrains their contents
- Even the most elaborate schema is designed and precise, it's a technical vision of how the data should conform : it cannot capture all the business constraints and processes that rely on good quality of data



- Data quality is the sum of 3 correlated concepts :
  - An **objective measurement of compliance** made on actual data and relying on schema and additional rules as
    - Completeness of data
    - Integrity checks + ...
  - A **process** where the above measurement is made on a regular basis, with corrective measures taken by designated "data owners"
  - A **set of** (usually expensive) **tools** implementing the process and measurements
- Data quality problems always happen but they are normally not a huge deal
  - Some exceptions : very bad quality, final reporting (data warehouse), regulatory and compliance reporting, customer relationship management, and... open data :)

Functional dependency

name	birth_date	address	festival	festival_address
Jean-Louis Berger	02/01/1980	83 rue Marie de Médicis BIARRITZ	Rock en seine	Saint-Cloud, France
Patrice Mercier	06/03/1965	26 rue des six frères Ruellan SARCELLES	Rock en seine	Saint-Cloud, France
Celestine Vernier	24/08/1997	83 rue Banaudon LYON	Rock en seine	Saint-Cloud, France
Jean-Louis Berger	02/01/1980	83 rue Marie de Médicis BIARRITZ	Aluna	Ruoms 07120, France

- In relational database theory, a functional dependency is a constraint between two or more columns in a table
- As a database designer, you have to identify the functional dependencies in your database schema to avoid data duplication. It's a basis for the database design rules called normal form.
- Normalization aims to free the database from update, insertion and deletion anomalies

name → birth_date, address
festival → festival_address

Entities, keys and identifiers

- Example: a database is used to store employees' characteristics
  - Their first and last names
  - Their birth date
  - Their social security number
  - Their salary, and so on
- We want to be able to create new records for new employees, to update a person's record upon modification (e.g. a raise), or to remove it if they leave the company

Entities, keys and identifiers (cont'd)

- In many cases, the data stored represents objects from the real world
  - There are exceptions: e.g. logs dumped for future analysis

- Those "real" objects are called **entities** – entities usually have a **type**
  - Example: "John Doe" is an entity of type "Person"

- In order to be retrieved and manipulated, each entity must be distinguished from the other entities stored alongside it. For this purpose, one of several attributes of the entity must be unique across all entities of the same type in the database: those attributes are called **keys**

- Among all the possible keys, one of them is usually chosen as "the" identifier. It is often called the **primary key**
  - Example for structured data: the social security number of a person
  - Example for unstructured data: a file name, including full path (metadata)

- **stability**: once created, the primary key of an entity must never change, because programs or other data may rely on it (cross-references...) !

Entities, keys and identifiers (cont'd)

- Sometimes, it is difficult to find a good primary key
  - Example: in some countries, the social security number cannot be stored in databases for privacy concerns. A person's name is not guaranteed to be unique or stable. Neither is the couple made from a person's name and their birth date
- In those cases, one can resort to use a unique, machine generated number as the primary key. This is often called a surrogate key

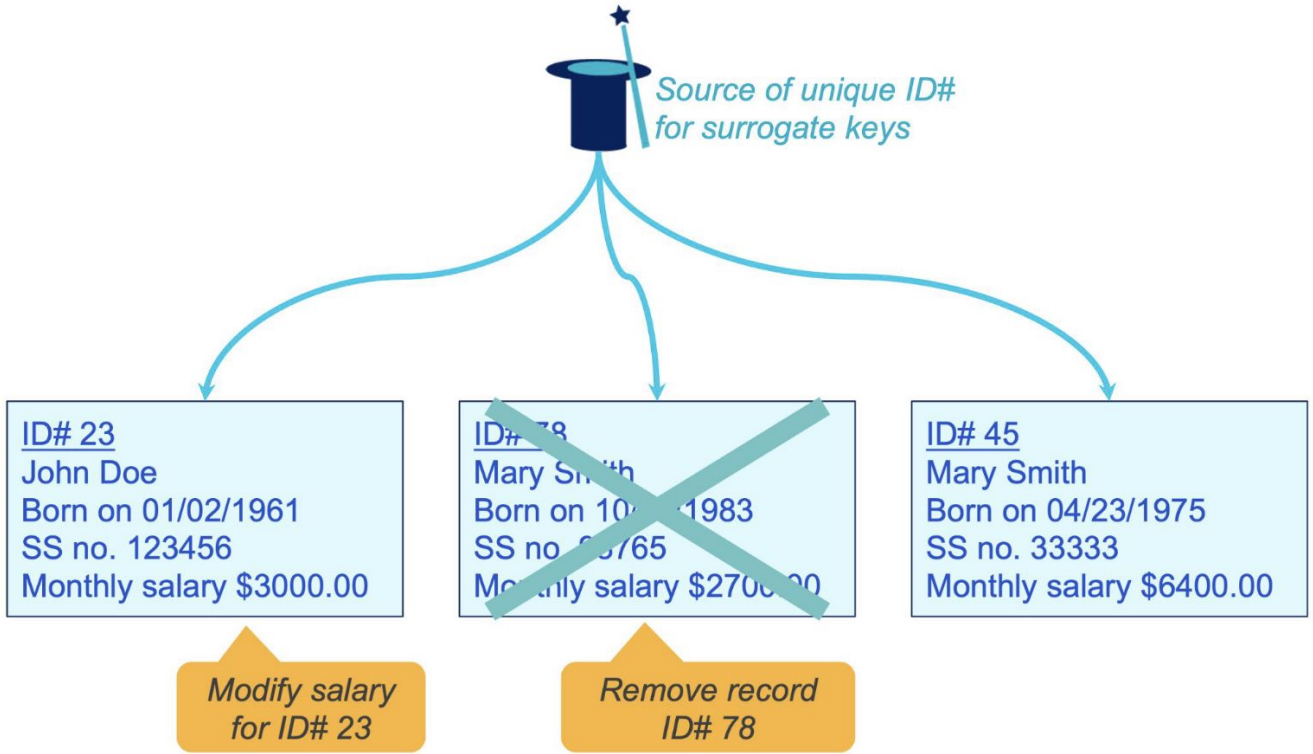
SURROGATE KEYS: PROS

- All other attributes may change without breaking the principle of stability of a primary key
- There is little risk of privacy issues, for the key at least
- A surrogate key may be used even when not strictly necessary – just in case, for convenience (DBMSes usually support them well) or performance reasons (manipulating integers is much faster than text data)
- A surrogate key is also a means to store several versions of an entity, each having its own primary key but with all other key fields being equal. This does not violate the principles of a primary key: uniqueness and stability

SURROGATE KEYS: CONS

- Manipulating machine-generated IDs can be difficult for non-technical people (e.g. end-users of an application)
- As a consequence, sometimes, people tend to adopt a surrogate key as a business, meaningful key (e.g. "the customer no. 12345"), so beware…
- In some contexts (distributed systems with several computers in interaction), generating truly unique numbers can be difficult

Entities, keys and identifiers (cont'd)



CRUD Operations

- All databases provide a minimum set of elementary operations that can be performed on an entity
- This set of operations is called CRUD

CREATE	i.e. Store a new object
READ	i.e. retrieve an object from the store
UPDATE	i.e. modifying the properties of an object
DELETE	i.e. remove an object permanently from the store

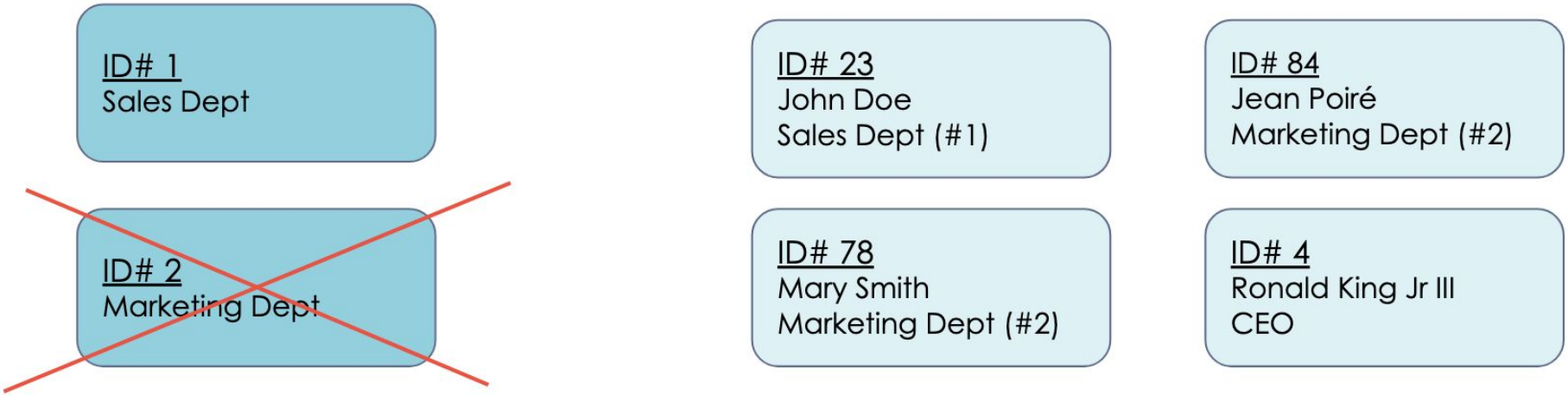
- The CRUD operations at least identify entities by their primary key; some databases offer additional, more flexible data access means
  - Example 1: raise all employees from the Sales department whose salary is below $2,000 and who have been in the company for more than 3 years
  - Example 2: compute the sum of all salaries in the company, broken down by department

Transactions

- Suppose the *marketing* department is being merged with the *sales* department. We want to :
  - delete the no longer existing *marketing* dept
  - then change the department of all marketing people to *sales*

Transactions (cont'd)

- Step 1 : remove the *marketing* department from the database



Transactions (cont'd)

- ~~Step 1 : remove the *marketing* department from the database~~
- Step 2 : move employee #78 to the *sales* dept

ID# 1
Sales Dept

ID# 23
John Doe
Sales Dept (#1)

ID# 84
Jean Poiré
Marketing Dept (#2)

ID# 78
Mary Smith
Sales Dept (#1)

ID# 4
Ronald King Jr III
CEO

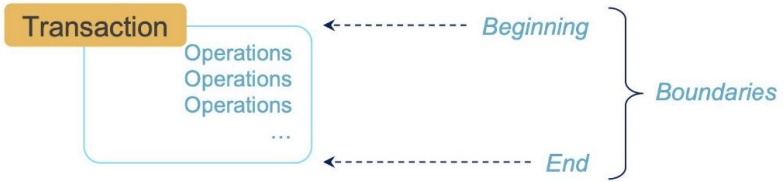
Transactions (cont'd)

- ~~Step 1 : remove the *marketing* department from the database~~
- ~~Step 2 : move employee #78 to the *sales* dept~~
- Step 3 : move employee #84 to the *sales* dept
  - The program crashes !

Transactions (cont'd)

- A **transaction** is a **unit of work** performed by a database. Data manipulation operations are said to be "wrapped" in transactions
  - **Every** operation, including elementary CRUD, occurs inside a transaction, even if the programmer didn't ask one (those are implicit transactions)
  - The programmer can explicitly open and close transactions, if she wants to wrap several operations in a single transaction

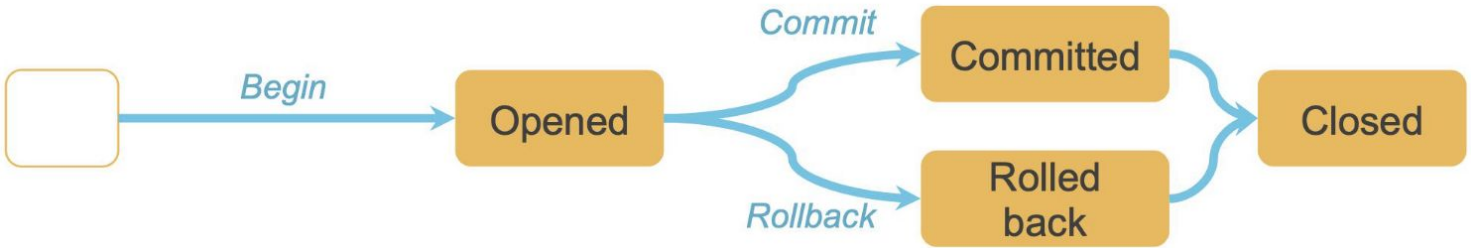
- With many DBMSes, especially the relational DBMSes, a transaction respects the **ACID** properties
  - Beware ! Most NoSQL databases use other models !



ATOMICITY	all operations in a transaction are either confirmed or cancelled at the end of the transaction
CONSISTENCY	before and after a transaction, the data is globally consistent
ISOLATION	simultaneous transactions don't see each other's modifications *(the data they see doesn't change unexpectedly)*
DURABILITY	once confirmed, the data is written to disk and will not be lost if the database crashes afterwards

Transactions (cont'd)

- When a transaction *begins*, it is said to be **opened**
- When it finishes *successfully*, it is said to be **committed**
- In contrary, in case of an error or a problem, it can be **rolled back** at any time during the course of its execution
  - To ensure atomicity, a rollback cancels all operations that were made by the transaction since its beginning



- *Opening*, *commit* and *rollback* are make
  - by y the **programmer**, explicitly
  - by the **DBMS**, for implicit transactions or when it encounters an error like a technical failure or a violation of the consistency property upon commit
    - Temporary violations are OK as long as they are cleaned before commit

Transactions (cont'd)

- Back to step 3 of our example, assuming we wrap our sequence in a transaction
  - After rollback, the database is left unchanged, ACID's A and C properties are satisfied

To Keep in mind

High-level concepts

STRUCTURATION OF DATA

- A **schema** is an invariant on the data managed by a system
- Unstructured data are difficult to manipulate

DATA QUALITY

- the most elaborate schema is but a technical vision of how the data should conform

ENTITY & KEYS

- In order to be retrieved and manipulated, each entity must be distinguished from the other entities through a primary key
- A surrogate key is a machine generated number as the primary key. It help to ensure stability on the primary key

TRANSACTION

- A transaction is a unit of work performed by a database.
- A transaction can be commit or rollback

04

Who use databases ?

People around databases

Because "It's not just a software"

Business Analyst (BA)

BAs usually don't use the database themselves, but design conceptual models from which database models are derived.

Application User

Most applications use a database to host their data. Application servers or software are clients to the DBMS; each application then has its own private database.

Database Administrator (DBA)

The DBA is a DBMS specialist who ensures that databases are working properly at all times. They are responsible for backing them up, and for tuning the DBMS to maintain acceptable performances. As experts, they often advise developers of best practices.

Reporting User

Reporting users use specialized (often expensive) software to produce business reports from the database. Such users range from the SQL-proficient business user hacking in Excel ("power user"), to the CEO who can't tell a mouse from a USB key but still wants a financial report on his desk every Monday morning.

Architects, developers

Architects and developers design the database schema (structure) of applications, so they meet users' business needs. They write the application code that interfaces with the database, with help from the DBAs or other architects.

Databases and apps

Sometimes databases and applications consume each other's data. Common examples are multi-database applications and datawarehouses.

Data Scientist / Data Engineer

Those users are both computer and mathematically literate. They crunch data directly from the database, often with specialized software that allows them to build statistical models after the data or transform pipelines.

05

Database Management System (or DBMS)

Summary

SHARING DATA

CONSISTENCY & INTEGRITY

SERVICES TO THE CLIENT

ANATOMY OF A DBMS

OF QUERIES AND HOW THEY ARE HANDLED

A LIST OF COMMON DBMSES

An analogy

A RESTAURANT

VS

A DBMS

Sharing data

A RESTAURANT	A DBMS
A restaurant can serve dinner to several people at once	A DBMS can serve data to several clients at once
A waiter can take orders from several tables independently	It processes requests from several clients independently
What's available for dinner is written on the menu: pick your favorite!	The databases and the objects they contain (tables, collections, ...) are available in the catalog of the DBMS; the catalog can be queried at any time
The course of dinner at the next table does not interfere with our dinner	Thanks to isolation, work done by a transaction does not interfere with the other transactions that are opened at the same time
A restaurant can have VIP customers; the management decide if they are offered free appetizer, a free dinner, or granted their favorite table	Many DBMS have security mechanisms that allow fine-tuning of the access rights for a particular client

More on security

Fine-grained security is not really an issue any more, except in strict environments like financial or government institutions. In the old days, users and applications would share a single database so security was a concern. Nowadays, the standard is "one application with services – one database". The application and its services, as façades, take care of security and nobody else accesses the database directly.
Most NoSQL databases were designed with no security features at all (sometimes this is a concern, because of the privacy issues brought by big data).

Consistency & Integrity

A RESTAURANT

- The kitchen only keeps ingredients to prepare the meals that are on the menu, in correct proportions so as to avoid wasting food
  - > Exception: when a meal is being prepared, the balance of ingredients is temporarily not respected
  - > This temporary imbalance does not prevent other customers' meals from being prepared
- Food can be found either as raw ingredients (in the kitchen), or as final dishes served to the customer
  - > Exception: when a meal is being prepared, some food is in an intermediate state
- When a customer is done eating, the food is not returned to the restaurant
  - > Exception: some restaurants are so bad that this property may be violated

A DBMS

- Data managed by the DBMS is consistent, integrity constraints are respected at any time (Consistency)
  - > Exception: when a transaction is in progress, some constraints may not be fully respected until the transaction commits
  - > This is temporary and invisible to other transactions (Isolation)
- Transactions are eventually fully committed or rolled back (Atomicity)
  - > When a transaction is in progress, its own view of the data is changing
- When a transaction is committed, the data is definitely written to a persistance medium (Durability)
  - > Exceptions: some DBMSes are so bad that a crash at the wrong moment can lead to data loss

Relational vs NoSQL

With many NoSQL databases, those duties are not fulfilled by the DBMS but by the applications (or the client driver). In fact, in the NoSQL world, paradigms other than ACID are used to deal with data consistency. This is because ACID, though theoretically sound, is an obstacle to the extreme performance that NoSQL databases aim at achieving. More on this later.

Service to the clients

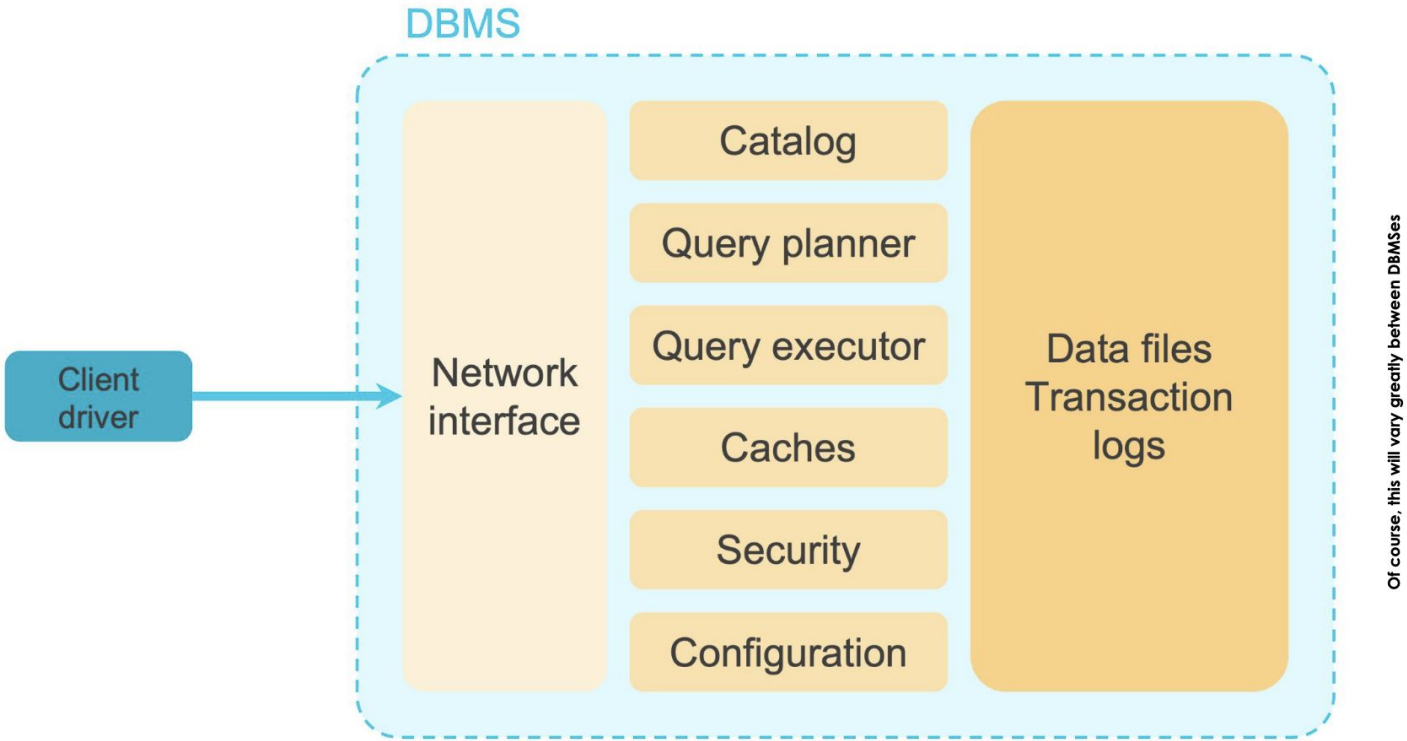
A RESTAURANT

- In case of a problem (burning, spilling, ...), a dish can be prepared again, keeping the customer satisfied
- Customers expect reasonable delays for serving meals, even when the restaurant is full
  - > Eventually, if the customers are too demanding, waiters will lose time satisfying whims instead of doing their job
- Customers don't have to know what happens exactly in the kitchen, nor do they have to know the exact recipe of the dishes they are served
  - > But hygiene inspectors have to know that for their verification duties
- Besides, some dishes may not be on the menu but available on request
  - > In other words, the "real" menu may be different from customer to customer

A DBMS

- A good DBMS features recovery strategies, so data is not lost and is still consistent after a crash
- A DBMS has special data structures (indices) and optimization strategies to keep response time acceptable most of the time, even when the amount of data is big or when there are many clients
  - > Eventually, if many clients issue requests that conflict with each other, the server will lose some time coordinating in order to satisfy the ACID properties. Performance will deteriorate
- Clients don't know how the data is physically stored on disk, nor do they know the strategy of the DBMS to retrieve it efficiently
  - > But DBAs have to know that for their tuning duties
- Besides, the logical view of the database may be a subset only of the actual catalog
  - > Different users can have different views of the database, depending on their rights

Anatomy of a DBMS

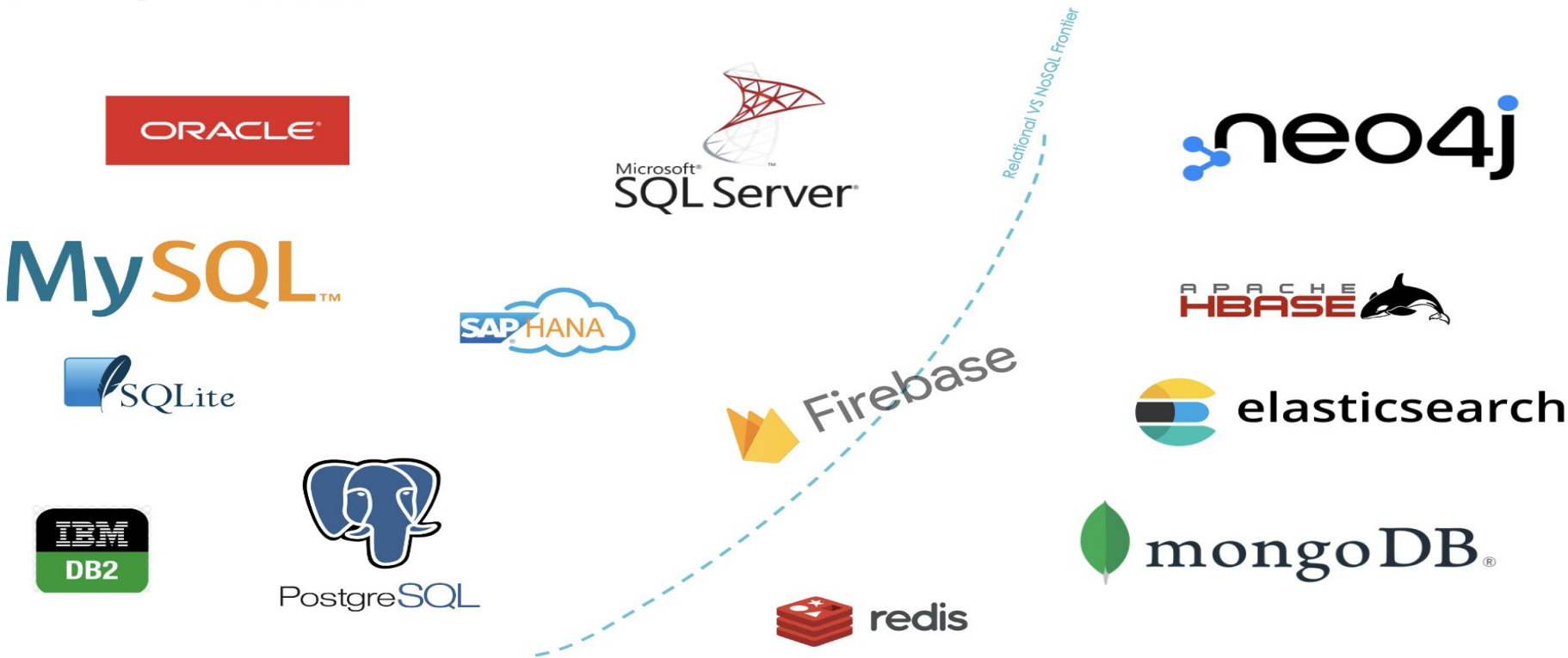


Of course, this will vary greatly between DBMSes

Queries and how they are handled

- **Queries** are the basic unit of communication between a database client and the DBMS. Examples :
  - “Get the characteristics of employee whose ID# is 78”
  - “Raise all employees from the Sales department by 5%”
  - “Get the total sales for region ‘Europe’ that were closed during the last month but for which the product wasn’t shipped yet 24 hours ago, except when the ‘Slow delivery’ option was chosen by the customer. Group and sort figures by customers weight in decreasing order”
- When the DBMS receives a query, the following operations occur :
  - **Analysis**: the query is checked for validity. For text-based query languages (e.g. SQL), the text is syntactically parsed. The result is checked against the catalog: are we querying objects that don’t exist?
  - **Control**: security rules (if any) are applied
  - **Optimizing**: several execution paths are examined, and the “best” one chosen, according to some criterion that depends on the DBMS
  - **Execution**: the query is finally executed, data is fetched and if requested, modified. Transactions are committed or rolled back
  - **Results:** (execution status and possible data requested) are sent back to the client

Common DBMS


ORACLE
Microsoft SQL Server
neo4j
MySQL
SAP HANA
APACHE HBASE
SQLite
Firebase
elasticsearch
IBM DB2
PostgreSQL
mongoDB
redis
Relational VS NoSQL Frontier

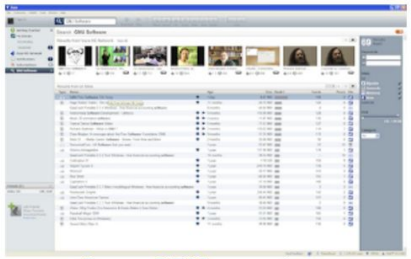
06

Usage and architecture

Summary

- DATABASES AS A STORAGE BACKEND FOR APP
- PRIMARY / SECONDARY ARCHITECTURE
- EMBEDDED DATABASE
- DATA WAREHOUSES, DATA MARTS
- CLIENT-SERVER ARCHITECTURE
- 3-TIER ARCHITECTURE
- PEER-TO-PEER (P2P) ARCHITECTURE
- DATA HUB / DATA LAKE / LAKE HOUSE

databases as a storage backend for an application



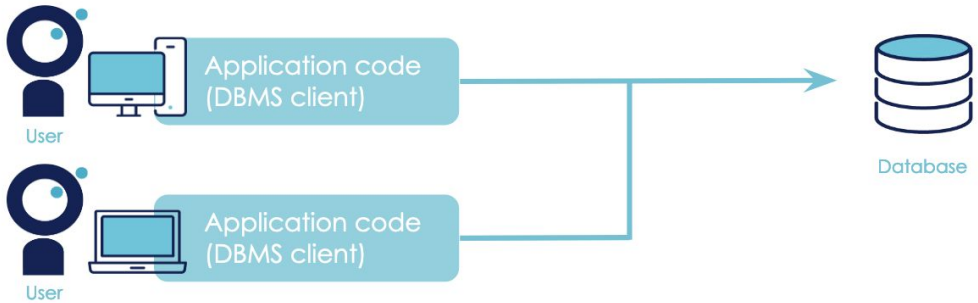
SuperCRM, a customer relationship application

- This is the most common usage for a database
- In this case, a piece of software uses the database as a persistent storage
  - Data is not lost when the application crashes, is shut down or the user disconnects
  - Several users of the same application share the same data
- This access pattern is called OLTP – **O**n**L**ine **T**ransactional **P**rocessing

OLTP =

- Many read and write requests to the DB (say, 50% reads, 50% writes)
- A majority of precise requests, each affecting a limited number of entities (cf CRUD)

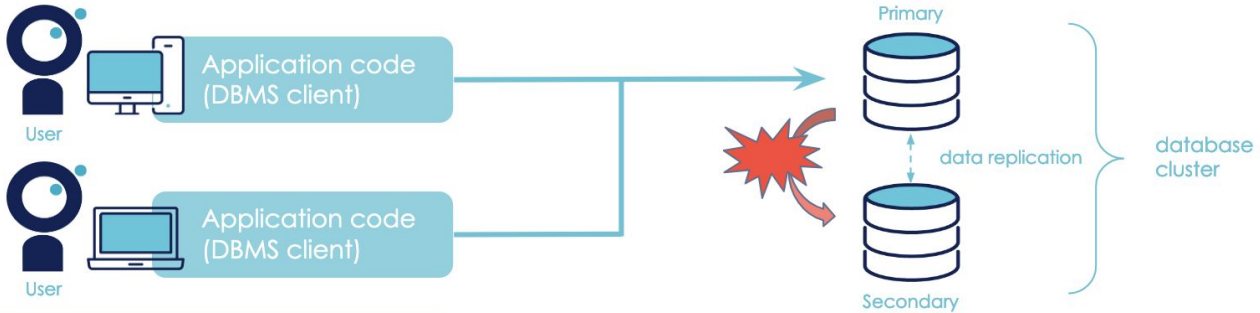
databases as a storage backend for an application (cont'd)



CLIENT-SERVER ARCHITECTURE

- One server, several clients to the database connecting at once
- Old-fashioned architecture, typical from the 1990's, before the emergence of the web architectures. We don't get to see it often nowadays
- All data is stored in one place, so all clients (all users) see the same version of the data
  - Except when they are in the middle of a transaction, with the isolation property
- The DBMS must be able to handle as many concurrent connections as necessary; this can be challenging if there are many users

databases as a storage backend for an application (cont'd)



PRIMARY / SECONDARY ARCHITECTURE

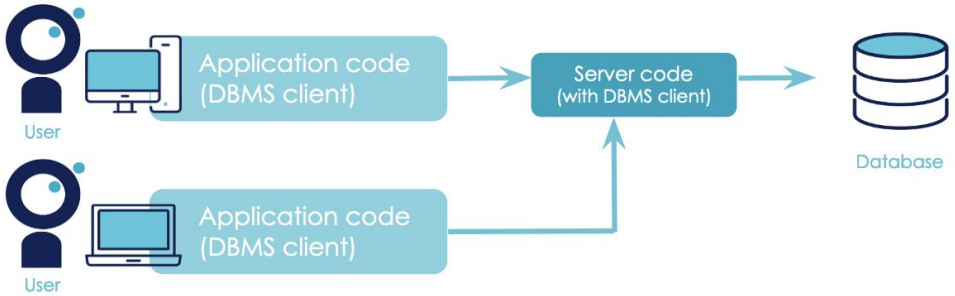
- Several servers (called **nodes**) form a **database cluster**; there is one **primary** node and one or several **secondary** nodes
- All data is replicated between the primary and the secondary nodes
- Active / Passive cluster
  - Clients connect to the primary node
  - The secondary node just mirrors data
  - When the primary node crashes, the secondary node takes over so clients can still be served
  - In most cases, data modification on the secondary node is forbidden

→ Availability and load-balancing for **read** operations only

- Active/active cluster (also called multi-master cluster)
  - Clients connect to any node, primary or secondary, and can make modifications
  - Some rules must be declared to handle **modification conflicts**
  - If a node crashes, any other node can serve clients

→ Availability and load-balancing for **read** and **write** operations

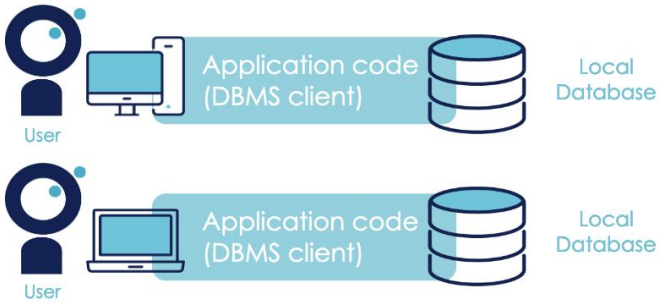
databases as a storage backend for an application (cont'd)



3-TIER ARCHITECTURE

- Clients don't connect to the database directly, but to an application server, e.g. a node server, java spring mvc. The application server talks to the database
- This is the standard architecture for web applications (but not only)
- All data is stored in one place, so all clients (all users) see the same version of the data
  - Except when they are in the middle of a transaction, with the isolation property
- The application server, as a database client, has a pool of connections that are shared among clients, and reused as necessary. This way the DBMS only sees a predictable and lower number of connections, whatever the number of end users

databases as a storage backend for an application (cont'd)



EMBEDDED DATABASES

- Each user's computer has its own local database. The database usually comes as a set of files of reasonable size (they hardly ever reach the GB size)
- Convenient for **small** applications, with no or limited connectivity (think of a salesman travelling from place to place)
- No concurrency issues with this architecture: for any given database there is at most one user connected to it
- Backups are very simple: copy the files somewhere else
- It's difficult to share data with other users. The local databases can be synchronized on demand with a central database, but there is always the risk of modification conflicts

technology → *sqlite, rockdb, couchbase mobile*
use case → *mobile application, utility*

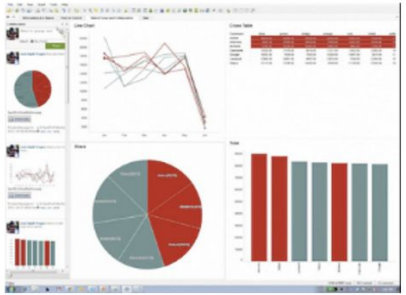
databases as a storage backend for an application (cont'd)

PEER-TO-PEER (P2P) ARCHITECTURE

- Several servers (called nodes) form a database cluster, to which several clients connect
- Two possible data distribution strategies:
  - All servers hold complete copies of the same data
  - ... or each server have a portion of the data (a shard); sharding is typical with NoSQL databases
- With sharding, when many nodes form the cluster, a huge total amount of data can be stored
- Concurrency issues are minimized because clients connect to any node in the cluster
- Because of this, it can be difficult (or plain impossible) to ensure concurrency: such clusters are usually not ACID

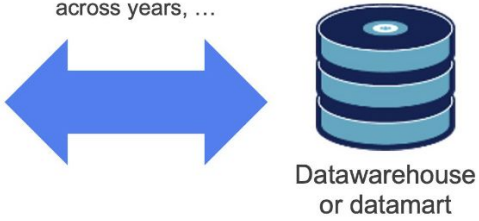
technology → *cassandra, hbase, ...*

data warehouses & data marts



Very serious financial report for the CEO

Top salesmen by generated revenue, evolution of sales across years, …



Datawarehouse or datamart

- This is the **second most common** usage for a database
- Reporting softwares produce... reports that query the database
  - The database usually contains pre-aggregated data, that is refreshed on a regular basis (every day/week/month/...) from other OLTP databases
- This access pattern is called OLAP – **O**n**L**ine **A**nalytical **P**rocessing

OLAP =

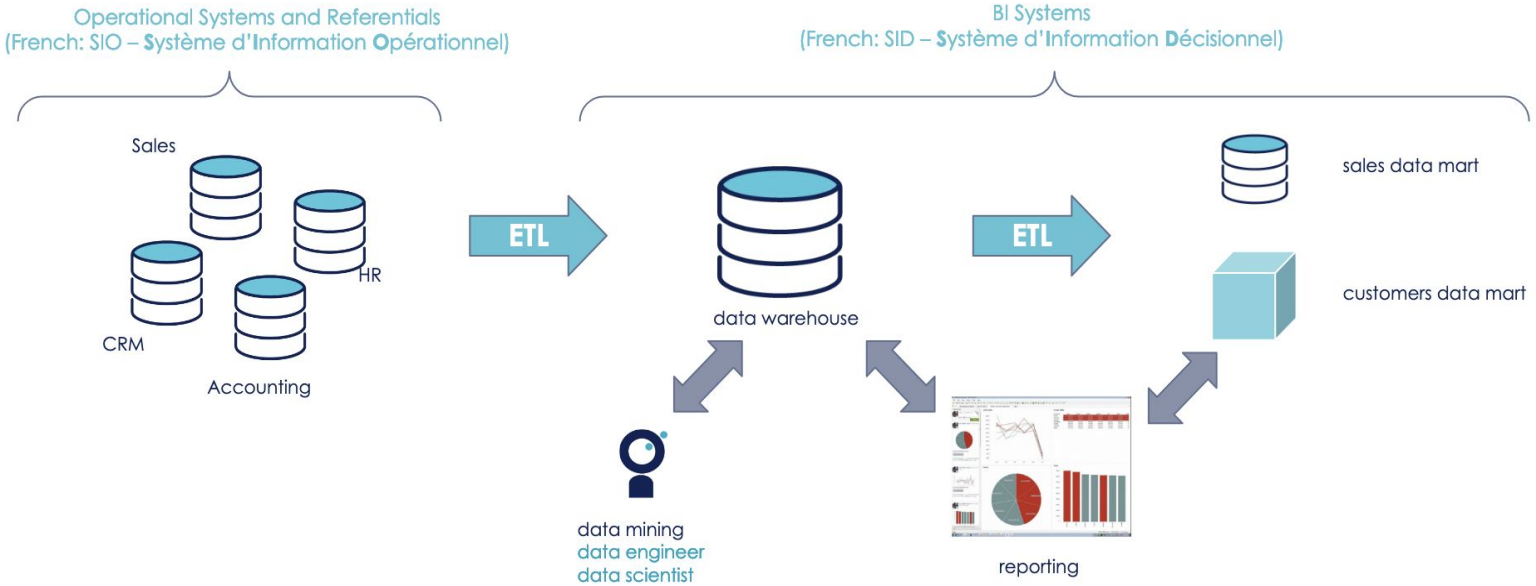
- Mostly reads; writes are very rare or occur upon refresh
- Those reads often span a large portion of the data, aggregating on the fly

data warehouses & data marts (cont'd)

- Data warehouses and data marts serve two purposes
  - **Concentrate** all (most of) the data of an organization, produced by independent departments, in one place
  - Give a single point of truth of the data, i.e., that data is **clean**, **approved** and **shared** across the organization
- Those purpose ultimately satisfy the need to make strategic decisions based on complete and accurate data: this is called **Business Intelligence** (BI)
  - In French: Décisionnel
- While a data warehouse contains "all" the data of the organization, smaller subsets specialized by domain are sometimes extracted from it; they are called **data marts**
- The data in a data warehouse often comes from several OLTP databases. The data **flows** from source applications to the data warehouse
  - Most of the time, data warehouses and data marts are refreshed on a daily, weekly, monthly... basis; data is sometimes streamed but this is rare
  - Except for the unattended refresh process, the data is usually read-only, for reporting purposes
- Most of data warehouses and data marts are either relational databases, or specialized data structures optimized for OLAP querying (e.g. cubes)
  - NoSQL in the common sense are still rarely seen as data warehouses, because reporting software strongly rely on
- SQL for querying. This is changing as some NoSQL databases are beginning to support at least some subset of SQL
  - For this reason, data warehouses rarely capture important unstructured data of an organization, e.g. emails, documents, ...

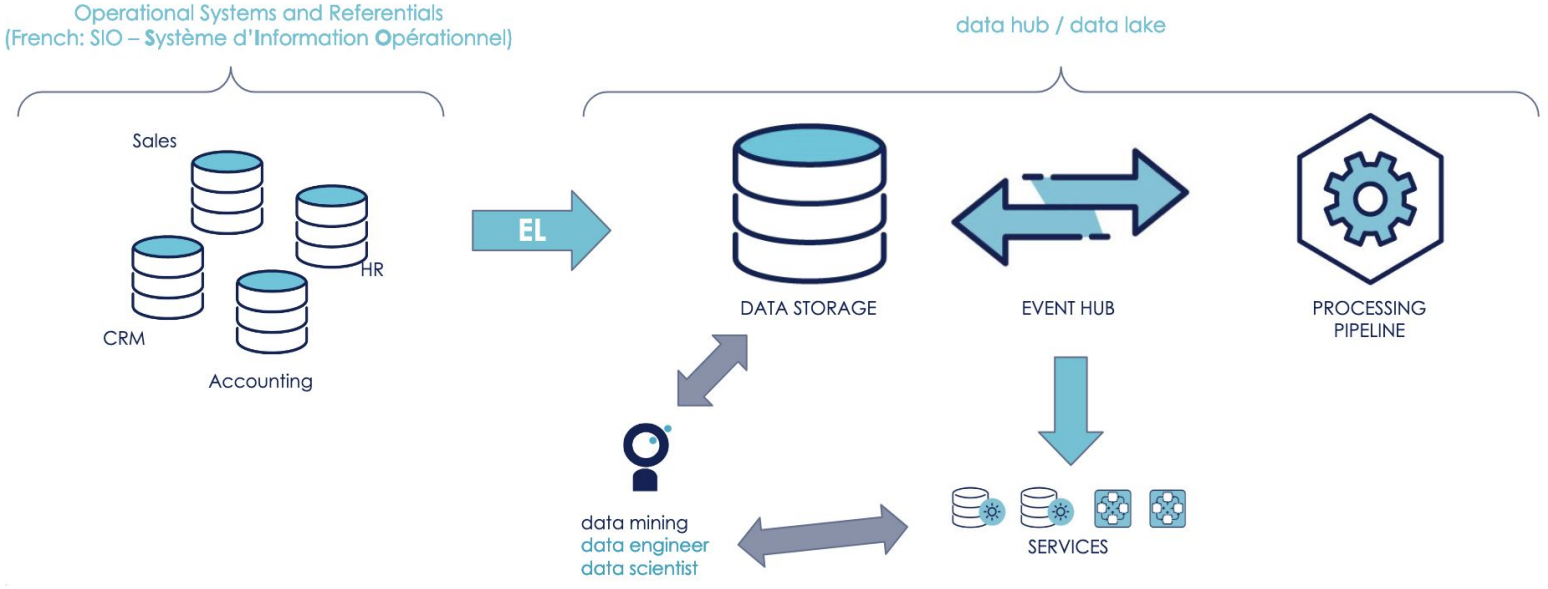
data warehouses & data marts (cont'd)

- Data is transported by specialized bulk software called ETL – **E**xtract, **T**ransform, **L**oad; reflecting the way that data is handled
  - > The data model in the various databases differs from each other, so transformations are needed to map, enrich, clean, ... the incoming data

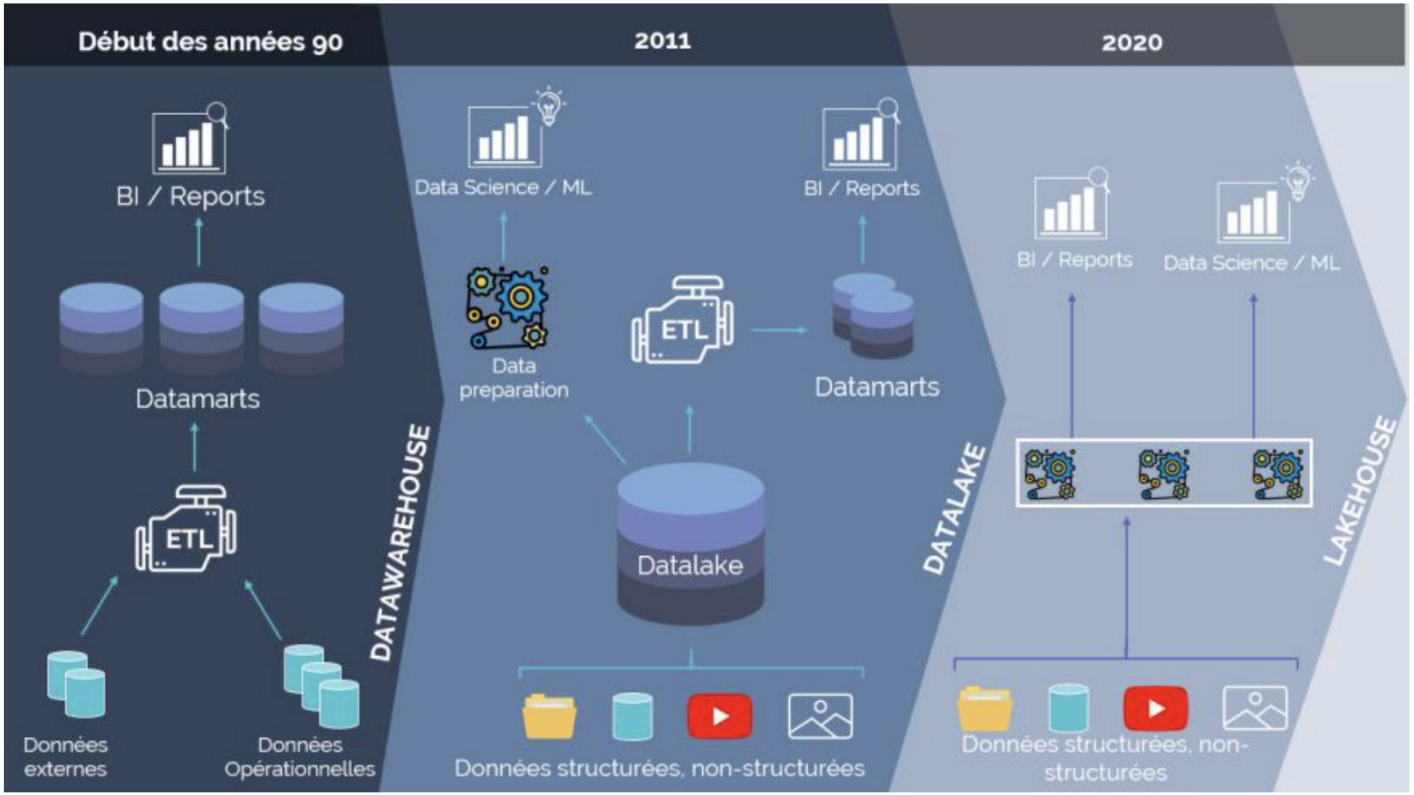


data hub & data lake

- Data is transported by specialized bulk software called EL – Extract and Load, reflecting the way that data is handled
  - > On the platform itself, transformation and modelisation process happens on the datastorage directly and don't rely on operational systems
  - > Distribution of data and applications are built over the platform in a layer called services



LakeHouse


Début des années 90
2011
2020
BI / Reports
Datamarts
ETL
Données externes
Données Opérationnelles
DATAWAREHOUSE
Data Science / ML
Data preparation
ETL
BI / Reports
Datamarts
Datalake
Données structurées, non-structurées
DATALAKE
BI / Reports
Data Science / ML
Données structurées, non-structurées
LAKEHOUSE

To Keep in mind

DATABASES AS A STORAGE BACKEND FOR APP

PRIMARY / SECONDARY ARCHITECTURE

EMBEDDED DATABASE

DATA WAREHOUSES, DATA MARTS

DATA HUB / DATA LAKE / LAKE HOUSE

DOJO

THE END

See you next week ;-)