The Relational Model

Serge Abiteboul

INRIA Saclay, Collège de France et ENS Cachan

Organization

The principles

- Abstraction
- Universality
- Independence

Abstraction: the relational model

Universality: main functionalities

Independence: the views revisited

Optimization

Complexity and expressiveness

Conclusion

The principles

DBMS

Goal: the management of large amounts of data

- Large amounts of data: database
- Software that does this: DBMS

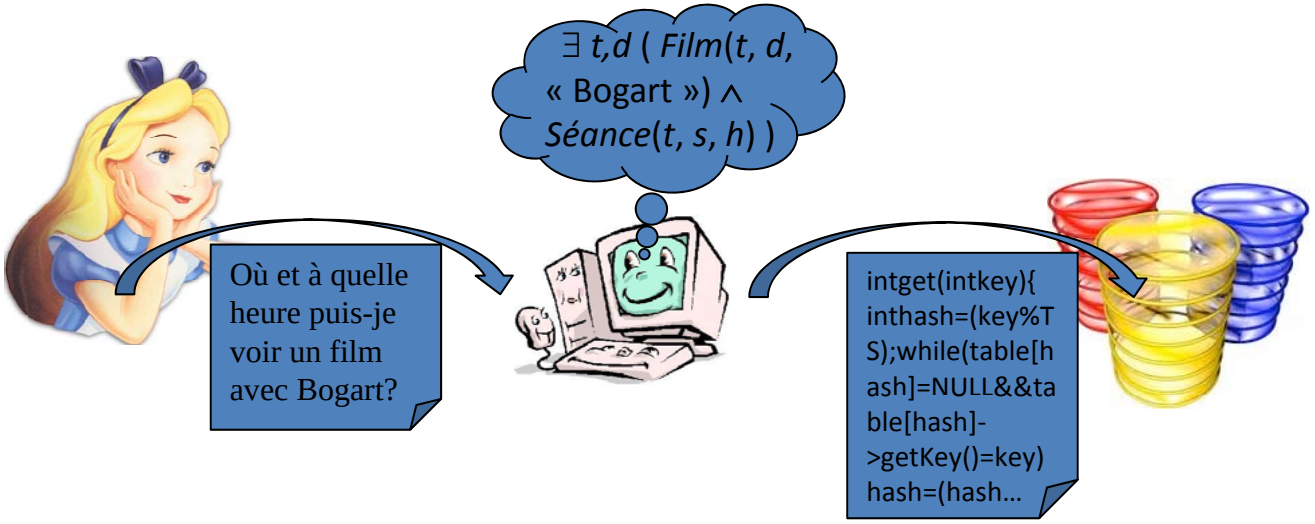
Management systems, databases

Characteristics of the data

- Persistence over time (years)
- Size (giga, tera, etc.).
- Shared among many users and programs
- May be distributed geographically
- Heterogeneous storage : hard disk, network

Mediation

The data management system acts as a mediator between intelligent users and objects that store information

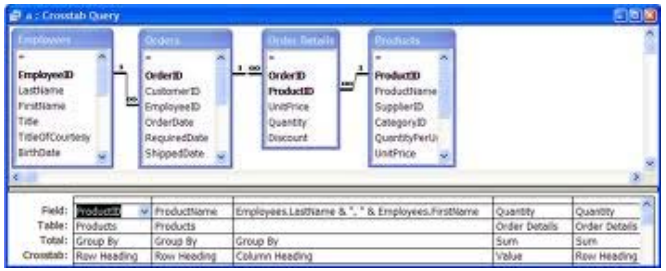


The questions are translated into first order logic and then into programs with precise and unambiguous syntax and semantics

Alice does not want to write this program; she does not have to

1st principle: abstraction

- Data model
  - Definition language for describing the data
  - Manipulation language (queries and updates)
- Simple data structure
  - Relations
  - Trees
  - Graphs
- Formal language for queries
  - Logics
  - Declarative vs. Procedural
  - Graphical languages



Complex graphical queries with MS Access

Towards abstraction: high-level data models

The relational model: Codd 1970

Data are represented as tables

Queries are expressed in relational calculus:
« declarative »

In practice, a richer language: SQL

Very successful both scientifically and industrially

- Commercial systems such as Oracle, IBM's DB2
- Popular free software like mySQL
- DBMS on personal computers such as MS Access

2nd principle: universality

DBMSs are designed to capture ***all data*** in the world for ***all*** kinds of ***applications***

- Powerful languages
- Rich functionalities: see further
- To avoid to multiply developments

In reality

- Less structured data are often stored in files
- Too intense applications require specialized software
- Today more and more specialized systems

Towards universality

We need services such as

- Concurrency and transactions
- Reliability and Security
- Data distribution
- More

Scaling

- Volume of data
- Volume of requests

Performance

- Response time: The time per operation
- Throughput: The number of operations per time unit

Large variety of applications with important needs for data management

Two main classes

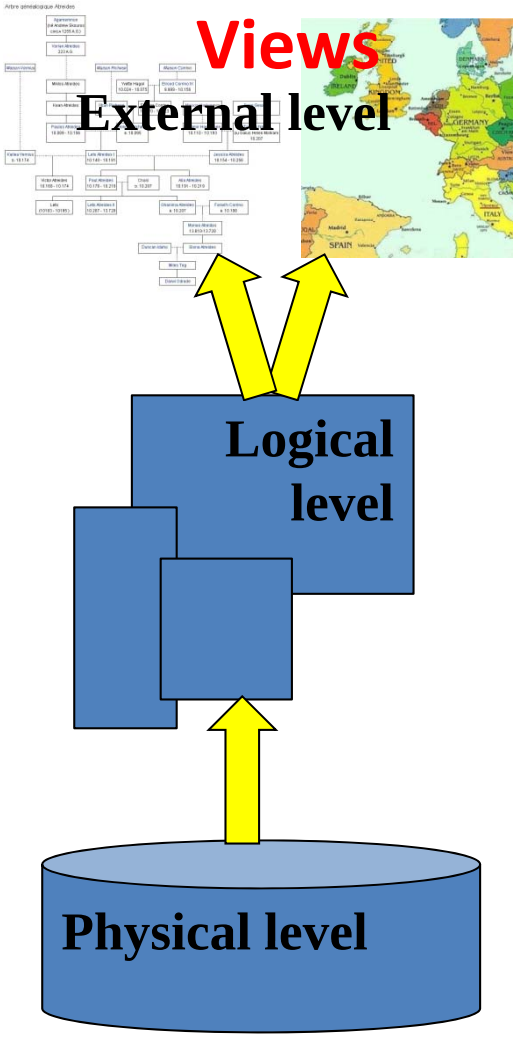
OLTP: Online Transaction Processing – Transactional

- E-commerce, banking, etc..
- Simple transactions, known in advance
- Very high load in number of transactions per second*

OLAP: Online Analytical Processing – Decision making

- Business intelligence queries
- Often very complex queries involving aggregate functions
- Multidimensional queries: e.g., date, country, product

3rd principle: Independence physical/logical/external



ANSI-SPARC Architecture (75): 3 levels

Separation into three levels

- Physical level: physical organization of data on disk, disk management, schemas, indexes, transaction , log
- Logic: logical organization of data in a schema, query and update processing
- Externally: views, API, programming environments

Independence

- Physical: We can change the physical organization without changing the logical level
- Logical: We can evolve the logical level without modifying the applications
- External: We can change or add views without affecting the logical level

Abstraction

The relational model

Data are organized in relations

Film

Titre	Directeur	Acteur
Casablanca	M. Curtiz	Humphrey Bogart
Les quatre cents coups	F. Truffaut	Jean-Pierre Léaud
Star wars	G. Lucas	Harrison Ford

Séance

Titre	Salle	Heure
Casablanca	Lucernaire	19 :00
Casablanca	Studio	20 :00
Star Wars	Sel	20 :30
Star Wars	Sel	22 :15

Queries are expressed in relational calculus

- q_{HB} = { s, h | ∃ d, t (*Film*(t, d, « Humphrey Bogart ») ∧ *Séance*(t, s, h) }

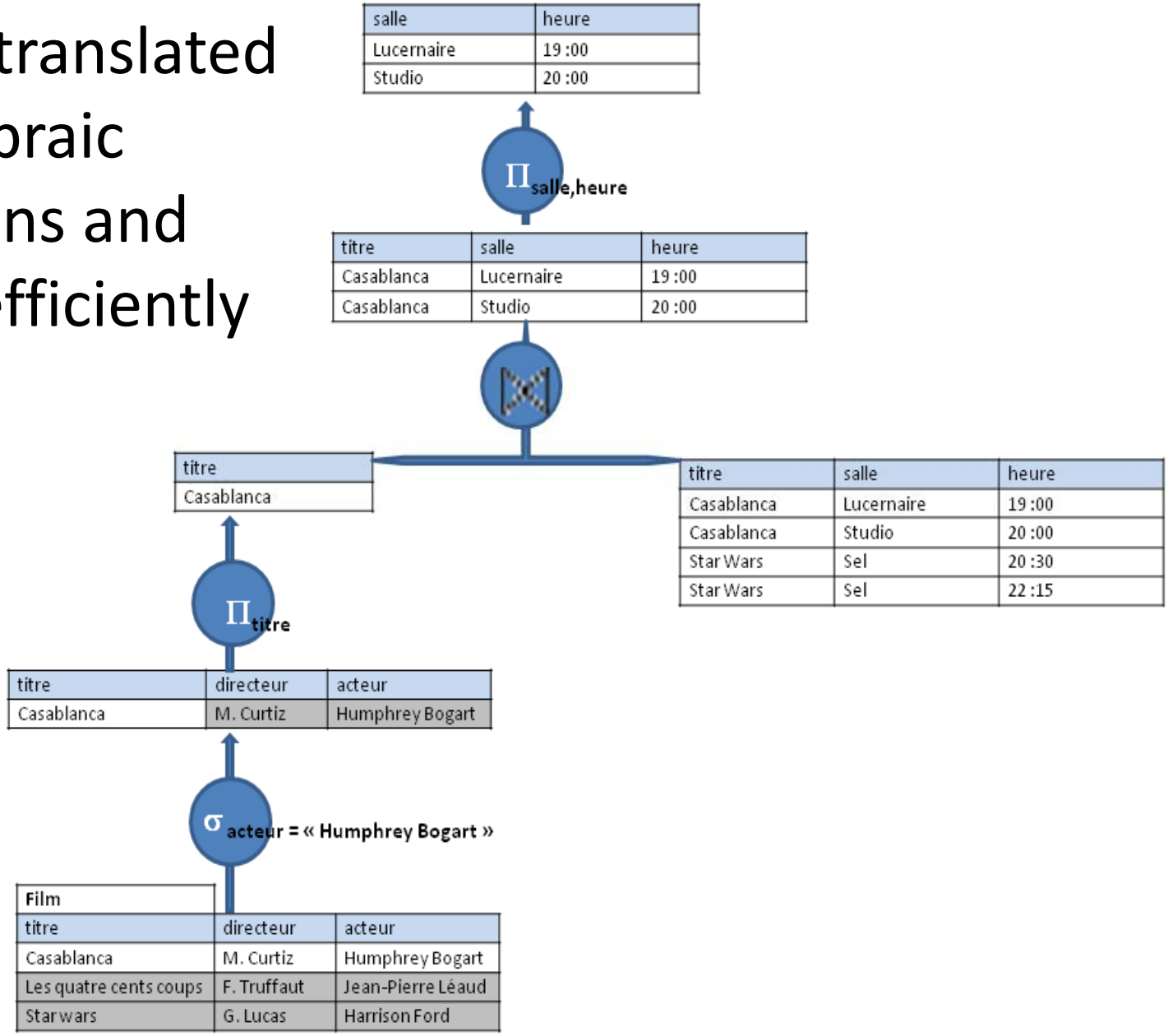
- In practice, using a syntax that is easier to understand:
- SQL:

  select salle, heure

  from Film, Séance

  where Film.titre = Séance.titre **and** acteur= «Humphrey Bogart»

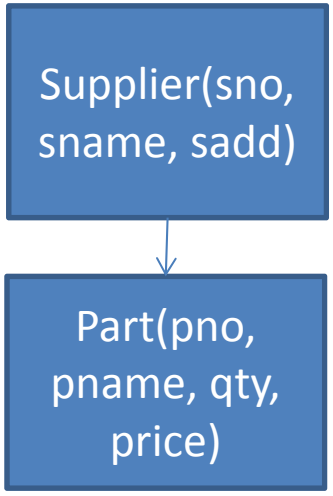
Queries are translated in algebraic expressions and evaluated efficiently



The main predecessors

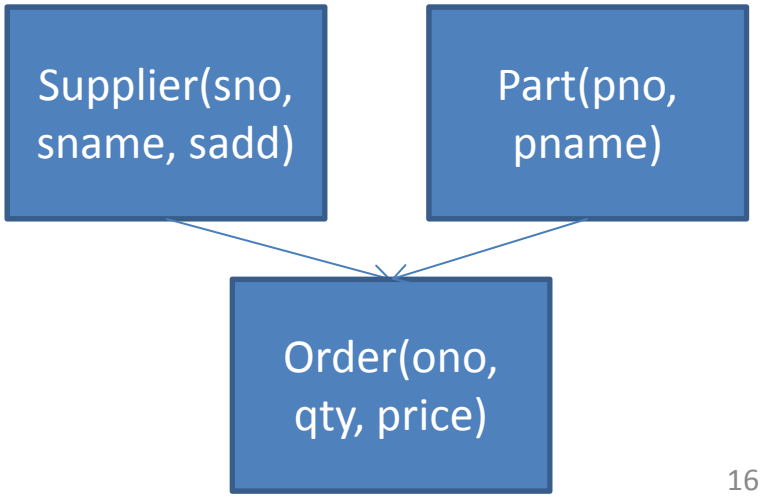
Trees

- IMS, IBM late 60s, 70s
- Still very used
- A hierarchy of records with keys



Graphs

- Codasyl
- A graph of records with keys



Little abstraction Languages

- Navigational
- Procedural
- Record-at-at-time

The main successors: semistructured data models

Trees

- XML
- Exchange format for the Web
- Standard
- Query languages: Xpath, Xquery
- Developing very fast

Graphs

- Semantic Web & RDF
- Format for representing knowledge
- Standard
- Query language: SPARQL
- Developing very fast

Abstraction

- Logic foundations
- High-level languages
- We will discuss them

Universality: functionalities

Here with a very relational viewpoint

Performance and scaling

The core of the problem

Be able to support

- Terabytes of data
- Millions of requests per day

For this two main tools

- Optimization
- Parallelism

Dependencies

Laws about the data

- To protect data — To design schemas
- To optimize queries — To explain data

Examples

- Séance[titre] $\subseteq$ Film[titre] Inclusion dependency
  Only known films are shown
- Séance: salle heure $\rightarrow$ titre Functional dependencies
  Only one movie is shown at a time in a theater

Logical formulas

- $\forall$ t, s, h (*Séance*(t, s, h) $\Rightarrow \exists$ d, a (*Film*(t, d, a))) tgds
- $\forall$ t, t', s, h (*Séance*(t, s, h) $\wedge$ *Séance*(t', s, h) $\Rightarrow$ t=t') egds

Some of the most sophisticate developments in db theory

Dependencies and schema design

Use simple dependencies up to complex semantic data models

Help choose a better relational schema

Person	Child	Car
John	Toto	BMW
John	Toto	2chevaux
John	Zaza	BMW
John	Zaza	2Chevaux
Sue	Lulu	
Sue	Mimi	

Update anomalies

Null values

Concurrency and transactions - ACID

Atomicity: the sequence of operations is indivisible; in case of failure, either all operations are completed or all are canceled

Consistency: The consistency property ensures that any transaction the database performs will take it from one consistent state to another. (So, consistency states that only consistent data will be written to the database).

Isolation: When two transactions A and B are executed at the same time, the changes made by A are not visible to B until transaction A is completed and validated (commit).

Durability: Once validated, the state of the database must be permanent, and no technical problem should lead to cancelling of transaction operations

Recovery from failures

The DBMS must resist to failures

A variety of techniques

- Journal
- Back-up copies
- Shadow pages

“Hot-standby“: second system running simultaneously

Availability: users should not have to wait beyond what is seen as reasonable for an application

Distributed data

Typically the case

- When integrating several data sources
- Organizations with many branches
- Activities involving several companies
- When using distribution to get better performance

Query processing over distributed data

- Data localization & global query optimization
- Data fragmentation
- Typically horizontal partitioning

Distributed transactions

- Two-phase commit
- Typically too heavy for Web applications

More

Security

- Protect content against unauthorized users (humans or programs)
- Confidentiality: access control, authentication, authorization

Data monitoring

Data cleaning

Data mining

Data streaming

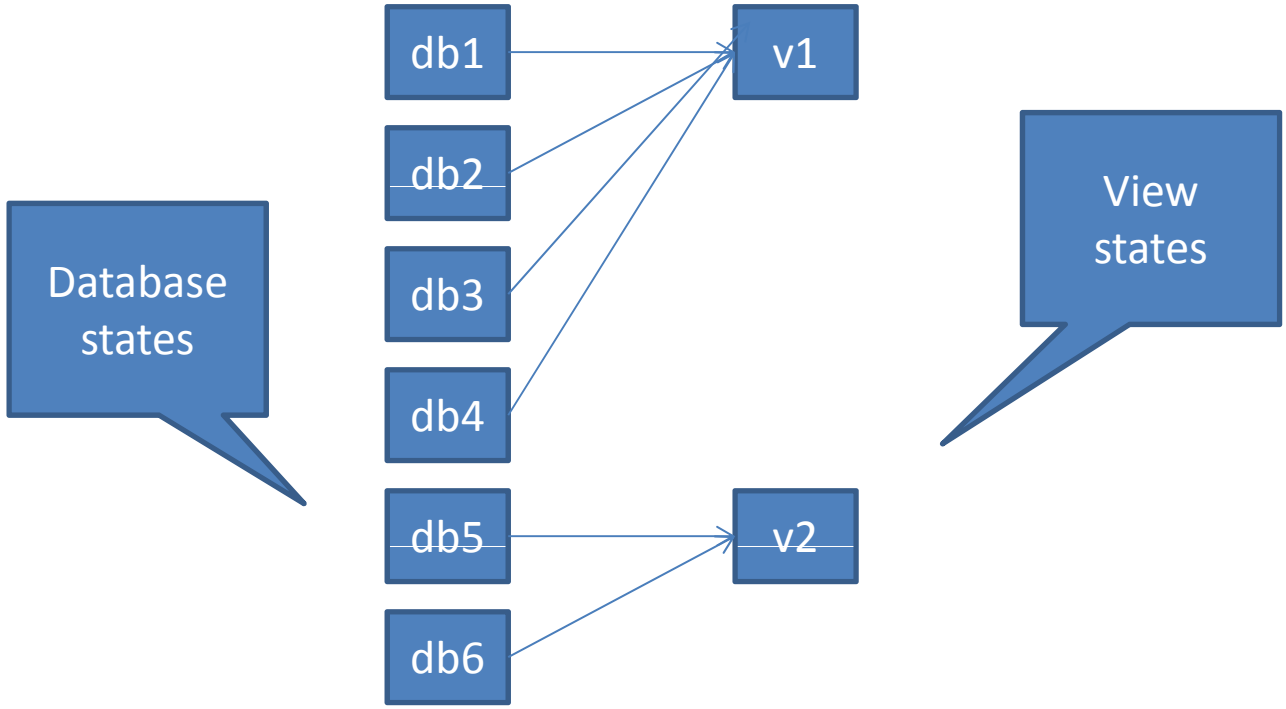
Spatiotemporal data

Etc.

Independence: views

Views

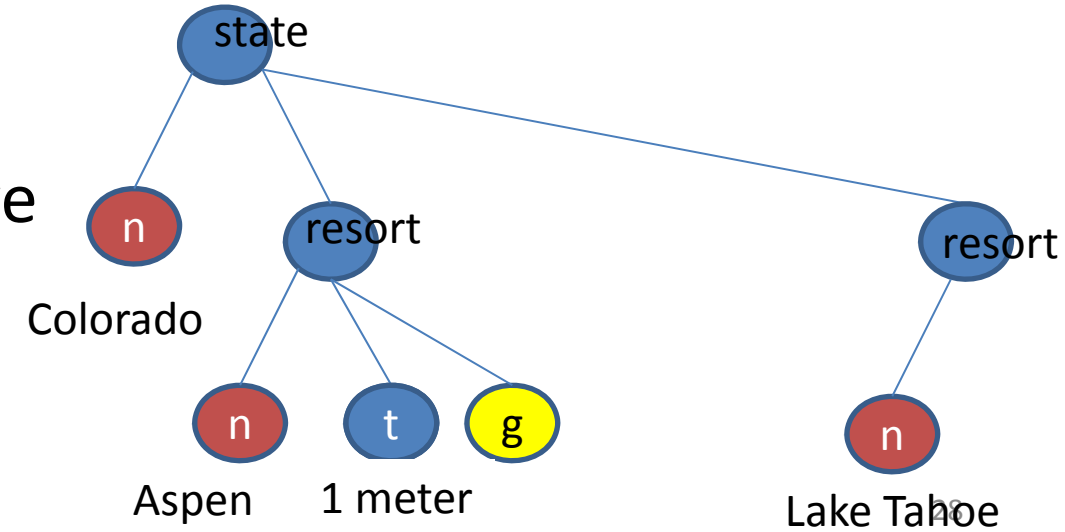
- Definition:
  - Function f: Database → View
- One of the most fundamental topics in databases



View definition

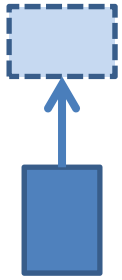
- Classical query
  - Define view …
- Implicit definition and recursion
  - Datalog
  - Dependencies (tgds)
- Mix between explicit/implicit: Active XML

```
<state s='Colorado'>
<resort n='Aspen'>
  <sc> Unisys.com/snow("Aspen") </sc>
 <sc>  Yahoo.com/GetHotels("Aspen")</sc>
  </resort>   … </resorts>
```



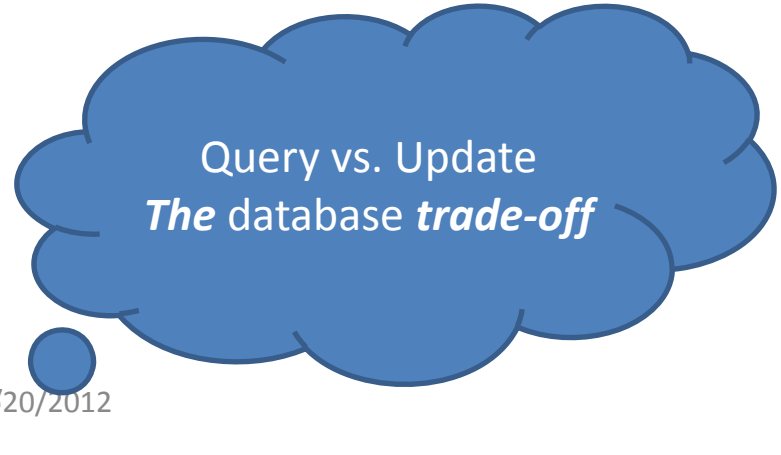
To materialize or not

Intentional



Update: do nothing

Query: complex



Materialized

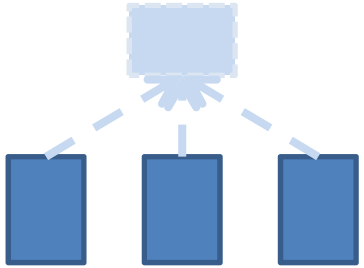
Update: propagate

- Base → view: costly view maintenance
- View → base: ambiguous

Query: simple

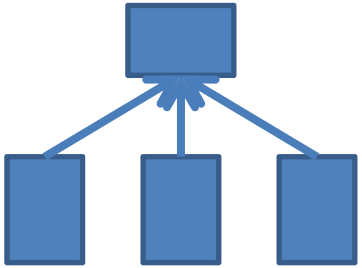
Integration: view over several bases

Intentional: mediatior



Queries are complex

- Materialized: warehouse



- Updates are complex

- Definitions
  - Global-as-view: $v = \varphi(db_1, \ldots, db_n)$
  - Local-as-view: $db_i = \varphi_i(v)$ for each I
  - Arbitrary complex constraints between the database and the views
    - Sometimes called alignments between them

Optimization

The reasons of the success

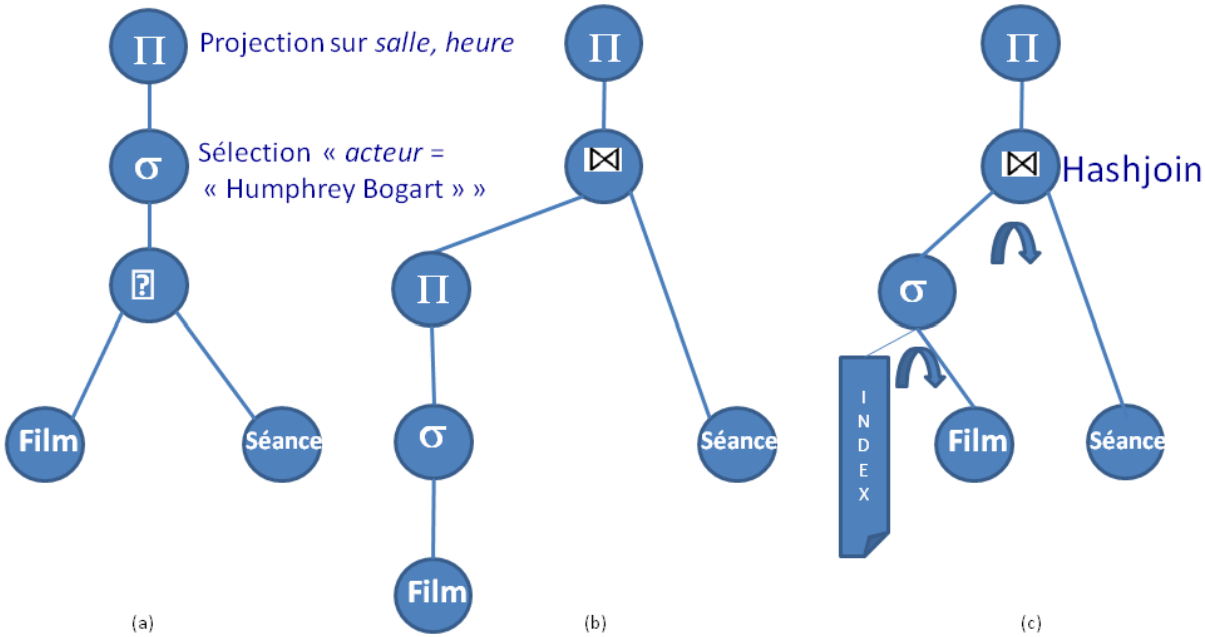
The queries are based on relational calculus, a logical language, simple and understandable by people especially in variants such as SQL

A calculus query can easily be translated into an expression of algebra that it is simple to evaluate (Codd Theorem)

Relational algebra is a limited model of computation (it does not allow computing arbitrary functions). That is why it is possible to optimize algebraic expressions evaluation

Finally, for this language, parallelism allows scaling to very large databases (class AC0)

Rewriting algebraic expressions



(a) For each f in film

For each s in séance do …

complexity in ~ n^2

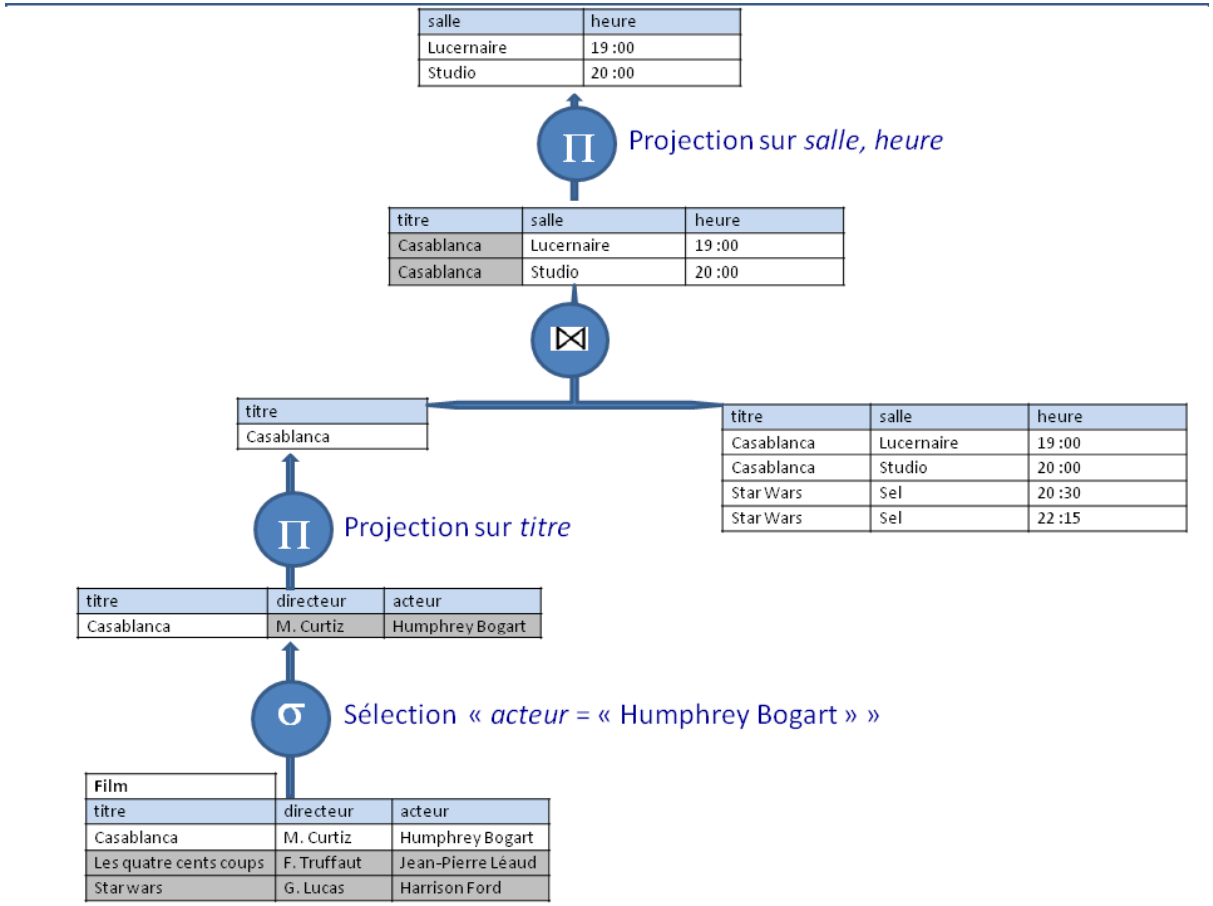
(b) If few tuples pass the selection

complexity in ~ n

(c) Using the index

complexity ~ constant

A possible query plan (without index)



Optimization

Using access structures

- Hash
- B-trees

Using sophisticated algorithm

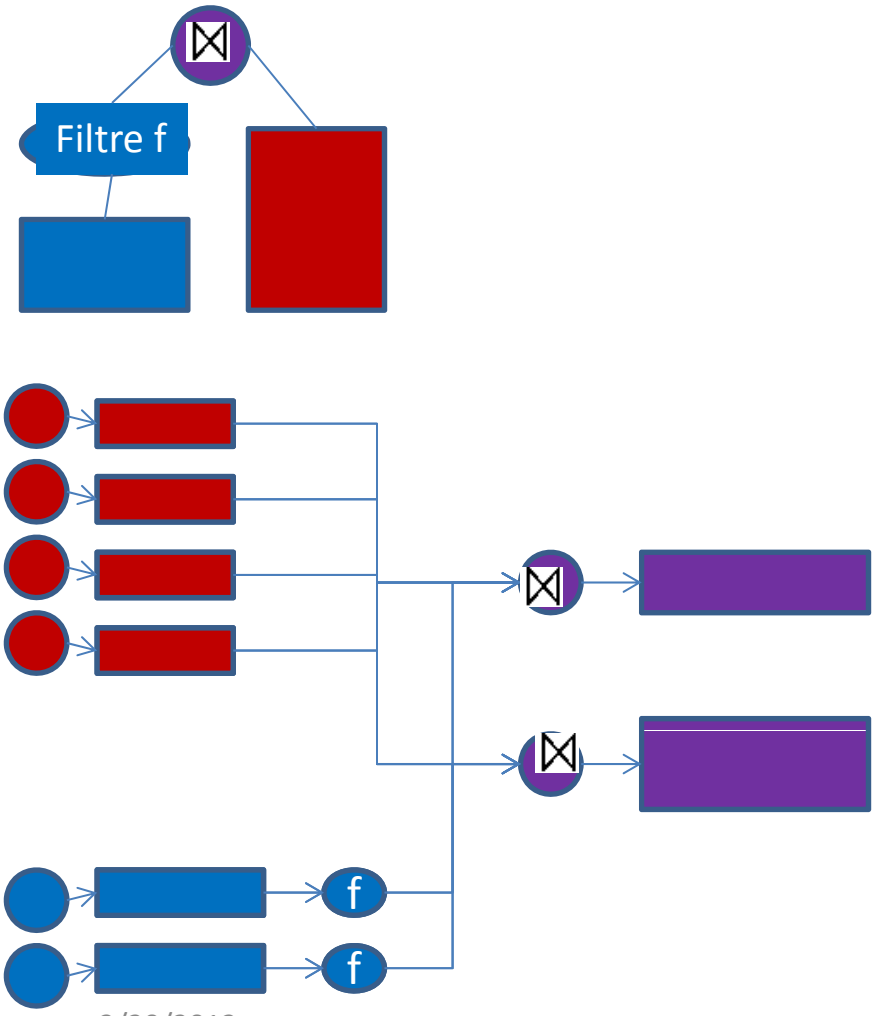
- Join

Cost evaluation to select an execution plan

Problem: search space is too large

Technique: Rewrite queries based on heuristics to explore only part of it

Optimization & scaling using parallelism



These problems can greatly benefit for parallelism

Typically divide the data

This is not true for all problems

Complexity and expressivity

Complexity
http://www.cs.rice.edu/~vardi/papers/sigmod08.pdf

Complexity: for a fixed query q,

- Testing given (I,t) whether t is in q(I) as a function of the size of I
- Focus on Boolean query to not depend on output size

Separate the dependency on sizes of data/query

- Very different and if mixed the dependency on query typically hides the dependency on the data
- Data complexity as a function of the size of the data (query fixed)
- Query complexity as a function of the size of the query (data fixed)

Data complexity

Relational calculus is in logspace

- The test can be performed using a space logarithmic in the size of the data
- This is primarily because the arity of tables is fixed; so a tuple uses logspace

logspace $\subseteq$ NC ($\subseteq$ ptime)

- Good potential for parallelization; see further

Query complexity

The complexity is pspace

Intuition: an intermediary result may be very large is it is the join of many relations

- Depends more on the number of variables used in the query that in its actual size
- Naive evaluation of (PiA(R join S)) requires more space that that of (PiA(R) cap^PiA(S))

Polynomial in the tree width

Parallel complexity

Data complexity: Constant parallel time

AC^0

A complexity class used in circuit complexity

The problems that may be solved with circuits of constant depth and polynomial size, with unlimited-fanin AND gates and OR gates.

Expressivity

One can not compute transitive closure

Add a fixed point

- Inflationary: fixpoint
- Or not: while

Vardi theorem: with an order on the domain

fixpoint = ptime and

while = pspace

Expressivity in absence of order

One cannot test if a relation has an even number of tuples

Abiteboul-Vianu

- Characterization of what can be computed with fixpoint and while
- Theorem: fixpoint = while iff ptime = pspace

Conclusion

Conclusion

And then: always question everything

- Revisit the models, languages, principles

Why?

- To scale to always more data and queries
- To support extreme applications that cannot be supported by standard technology:
  - Google
  - Visa transactions
- To facilitate application development
- To offer more in terms of performance, reliability, security, etc..

Conclusion

Relational model	Beyond
Entries in relations = atomic values	Entries are set of values
	Missing data, probabilisitic data
Data are regular	Semistructured
ACID	Weaker concurrency
Universal	Specialized: noSQL
Data are persistant	Queries on data flows
Data are static	Data & behavior: Object databases Active databases
Constraints are static (FDs, etc.)	Triggers
…	


Merci !
FRAPAR.