# The Relational Model

Serge Abiteboul
INRIA Saclay & ENS Cachan


# The course

Wednesday afternoon: *Des données, à l'information, aux connaissances : Le Web de demain*

Thursday morning: data models (relational & beyond)

Thursday afternoon: web search

Friday morning: semantic

Friday afternoon: deduction (Datalog & webdamlog)

# Organization

The principles

- Abstraction
- Universality
- Independence

Abstraction: the relational model

Universality: main functionalities

Independence: the views revisited

Optimization, complexity and expressiveness

A jewel of databases

Conclusion

# The principles

# Database Management System

Goal: the management of large amounts of data

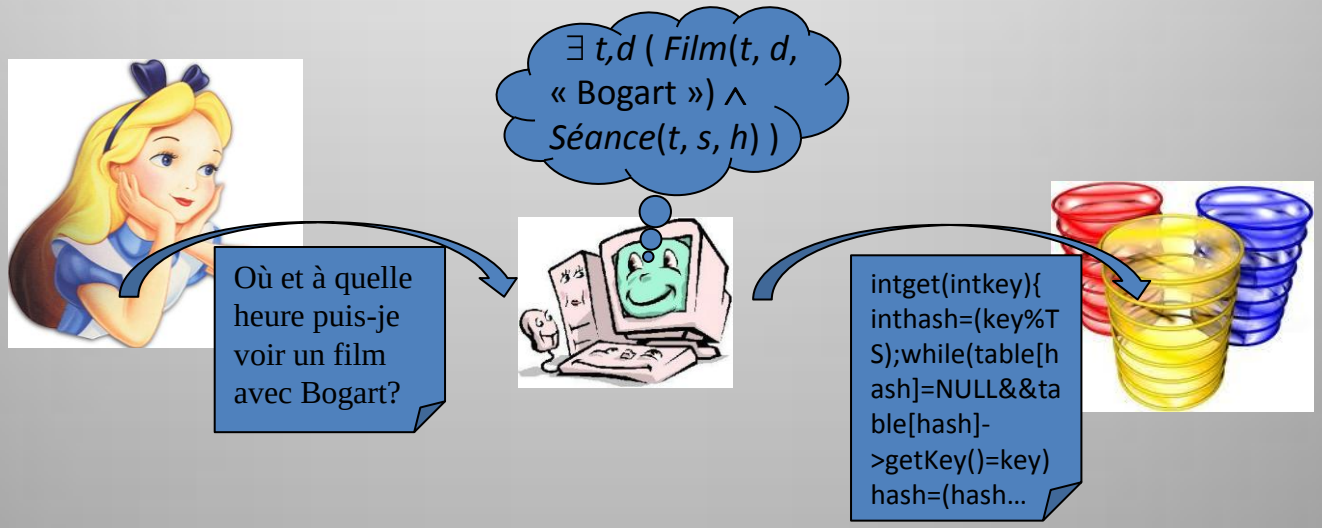
- **Large data**: **database**
- **Large software** to manage them: **DBMS**
  - Very complex systems
  - Years of research and development by large groups of researchers/engineers

Characteristics of the data

- Persistence over time (years)
- Possibly very large (giga, tera, etc.).
- Possibly distributed geographically
- Typically shared between many users and programs
- Typically stored on heterogeneous storage : hard disk, network

# Key role: Mediation between human and machines

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




First order logic: precise and unambiguous syntax and semantics

Program: Alice does not want to write it; she does not have to

# 1st principle: abstraction

High level data model

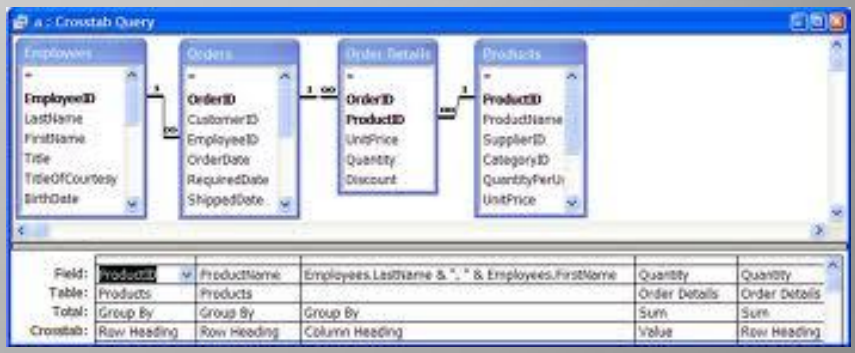
- **Definition language** for describing the data
- Manipulation language: **queries** and **updates**

Definition language based on simple data structure

- Relations
- Trees
- Graphs

Formal language for queries

- Logics
- Declarative vs. Procedural
- Graphical languages




Complex graphical queries with MS Access

# 2nd principle: universality

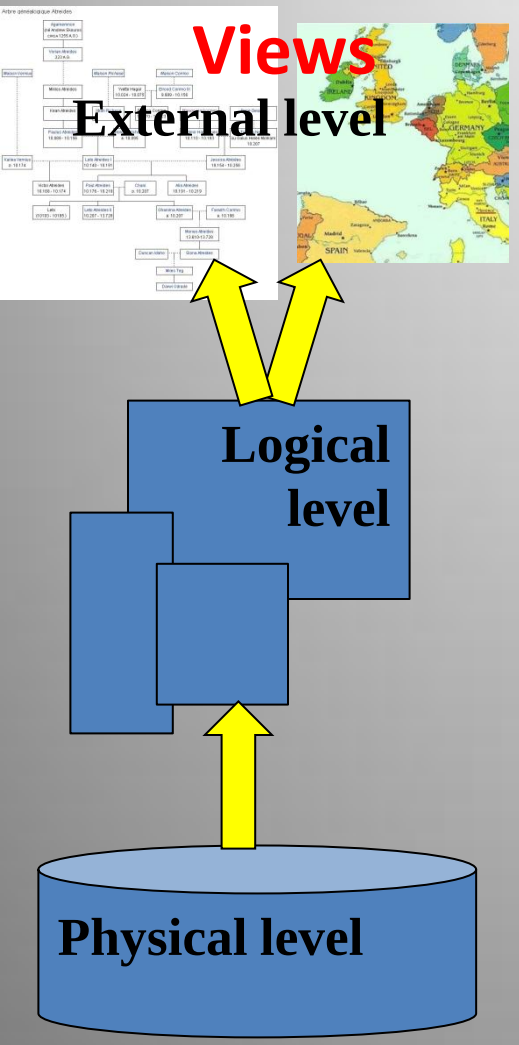
DBMSs are designed to capture **all data** in the world for **all kinds of applications**

- Powerful languages
- Rich functionalities: see further
- To avoid multiplying developments

In reality

- Less structured data are often stored in files
- Too intense applications require specialized software
- More and more the case

# 3rd principle: independence




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

# 1st principle: abstraction

High level data model: The relational model

# The relational model

Codd 1970

Data are represented as tables, namely **relations**

Queries are expressed in **relational calculus**: « declarative »

- Essentially First order logic

In practice, a richer model/language: **SQL**

- Ordering between tuples, nesting, aggregate functions, nulls…

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

# 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:

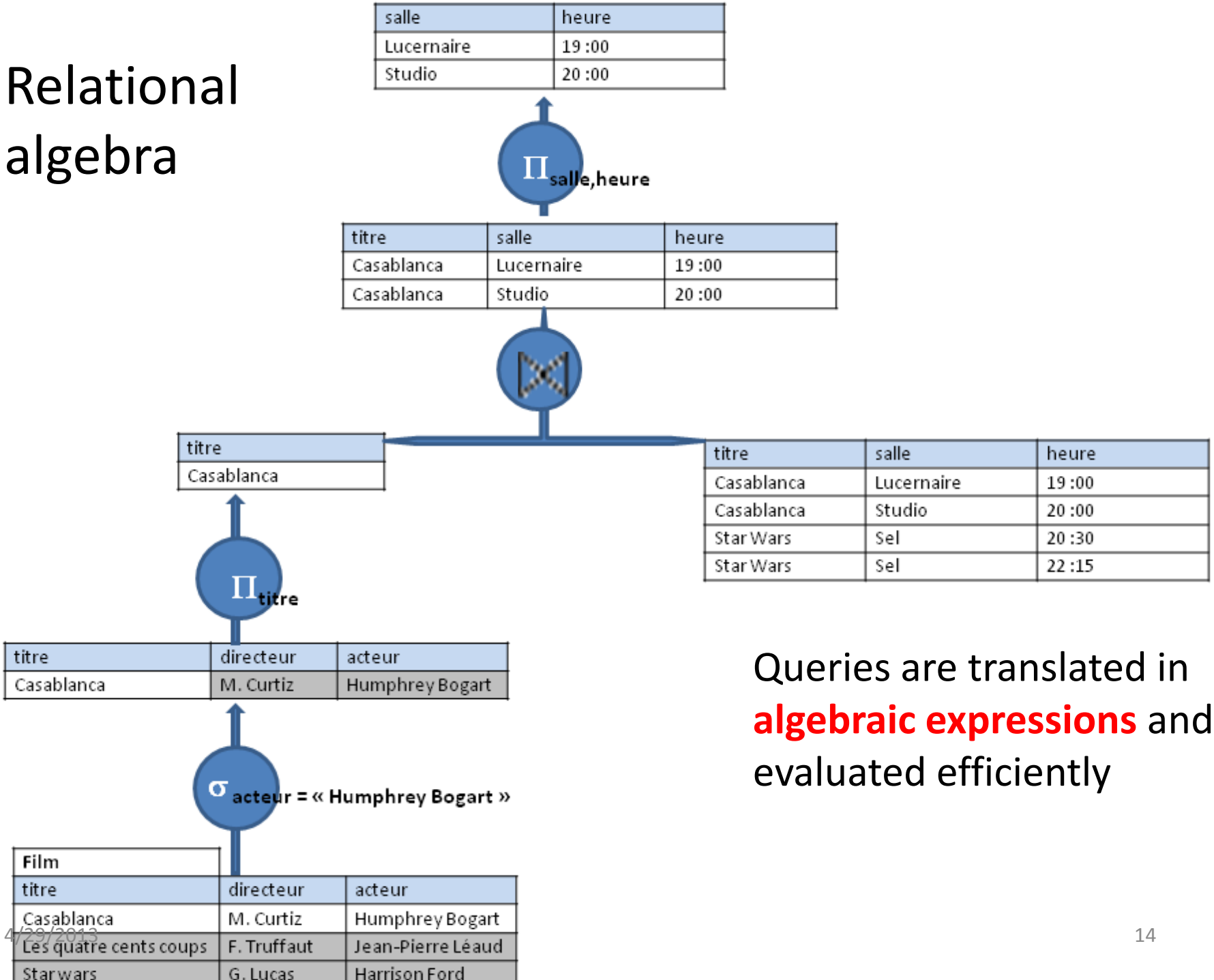
SQL:

**select** salle, heure

**from** Film, Séance

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

# Relational algebra




Queries are translated in **algebraic expressions** and evaluated efficiently

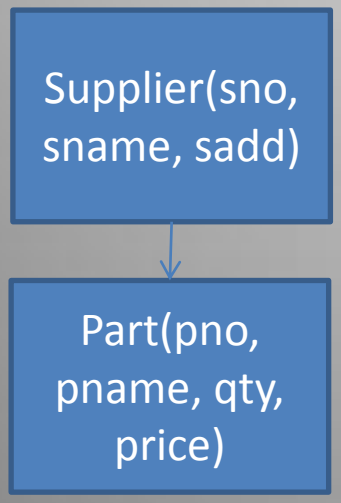
# The main predecessors

Trees

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

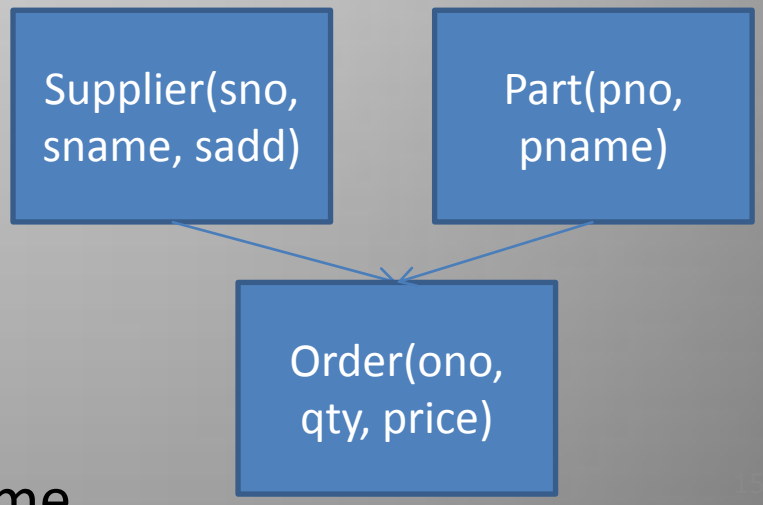
Graphs

- **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

Next two classes

# 2nd principle: universality

Must support many functionalities

# Two main classes of applications with important needs for data management

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

# Towards universality: more functionalities

Applications have essential functional requirements such as :

- Concurrency and transactions
- Reliability, security, access control
- Data distribution
- Performance and scaling

# Towards universality: performance and scaling

Many applications have severe performance requirements

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

We need to be able to scale

- Volume of data — Terabytes of data
- Volume of requests — 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 sophisticated developments in db theory

# Dependencies and schema design

Use simple dependencies up to complex semantic data models

Help choose a better relational schema

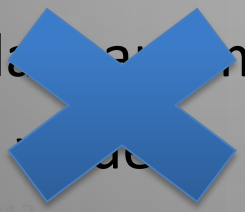
| Person | Child |
|---|---|
| John | Toto |
| John | Zaza |
| Sue | Lulu |
| Sue | Mimi |

| Person | Car |
|---|---|
| John | 2chevaux |
| John | BMW |
| Sue | BMW |
| Sue | 2chevaux |

| Sue | Lulu |
|---|---|
| Sue | Mimi |

Upda[illegible]nalies

Null [illegible]



# 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 survive 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.

# 3rd principle: independence

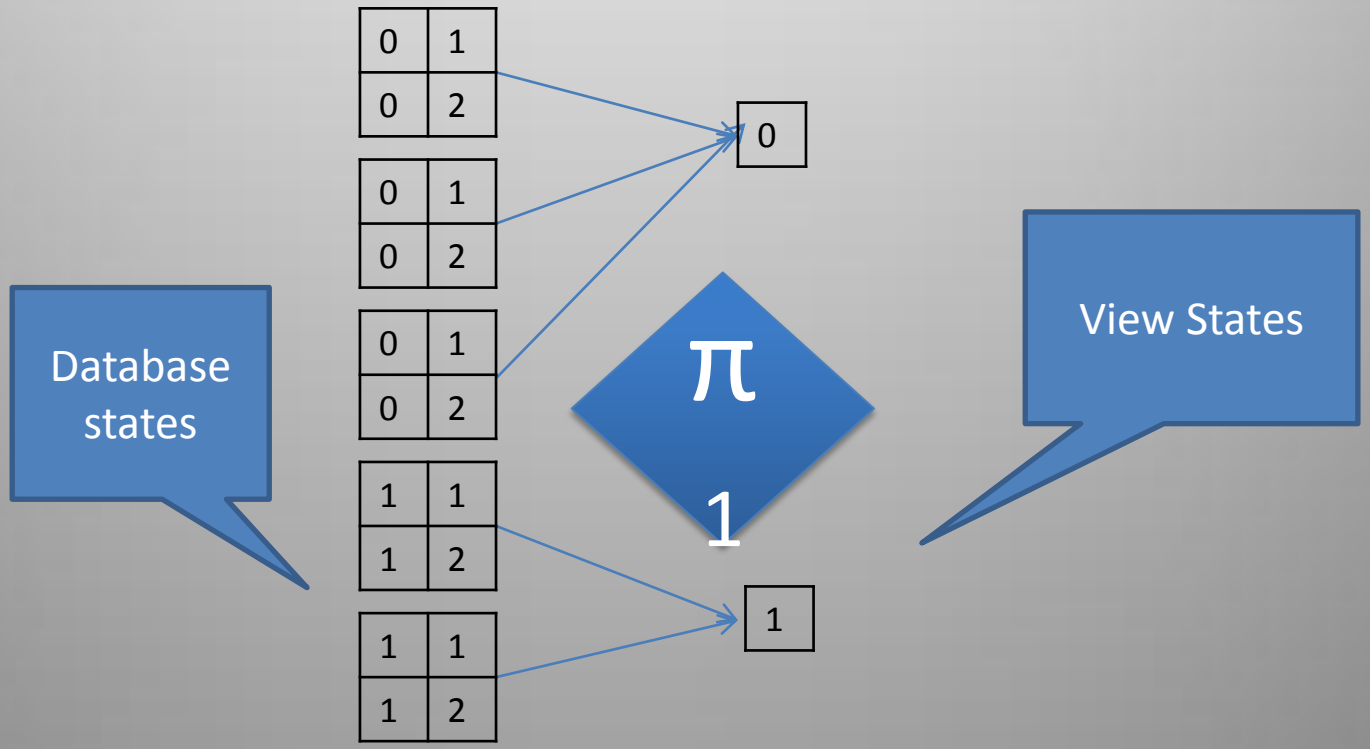
Views

# Views

Definition:

- **Function: Database → Database**

Perhaps the most fundamental concept in databases



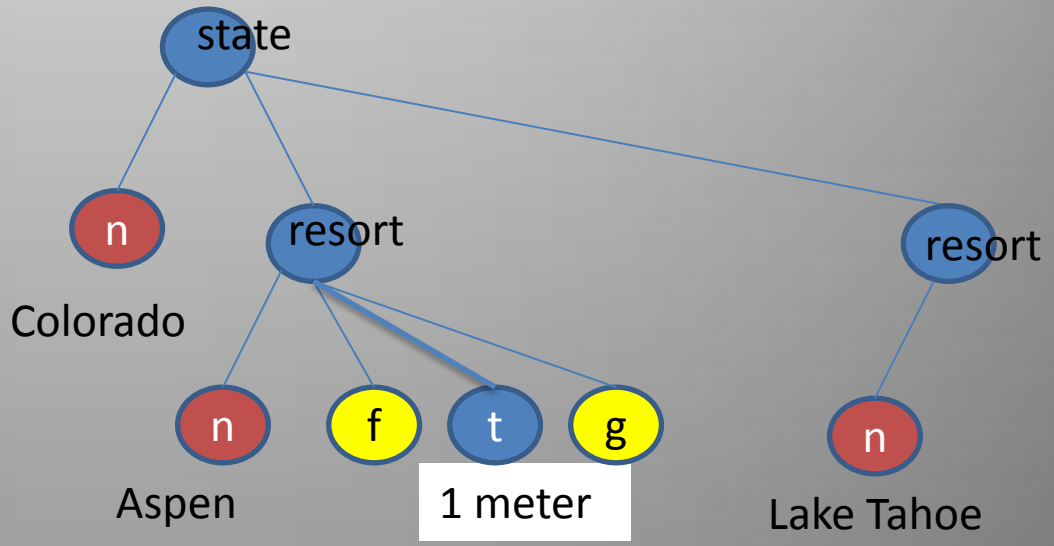
# View definition

Classical query
- Define view …

Implicit definition and recursion
- Datalog
- Dependencies (tgds)

Mix explicit/implicit: Active XML

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



# To materialize or not

Intentional

Update: do nothing

Query: complex

**Query vs. Update**
***The* database *trade-off***

Materialized

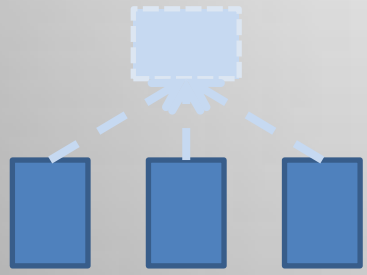
Update: propagate

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

Query: simple

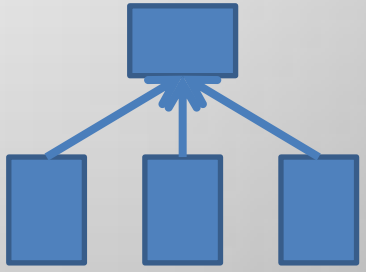
# Integration: view over several bases

Intentional: mediator



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
- Complex logical constraints between the database and the views

# Optimization, complexity and expressivity

# The reasons of the success

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

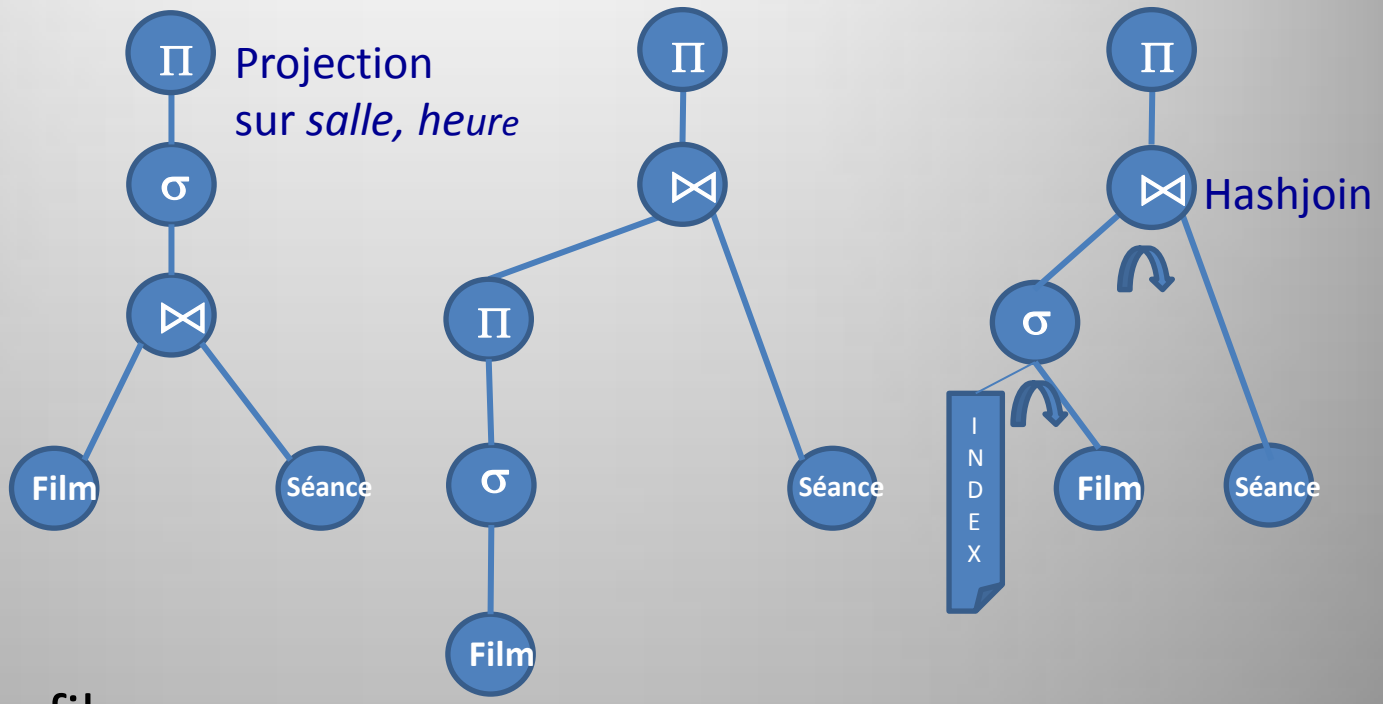
**Algebra**: A calculus query can easily be translated into an algebraic expression (Codd Theorem) that can be evaluated efficiently.

**Optimization**: The 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.

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

# Rewriting algebraic expressions

Sélection
« *acteur* =
« Humphrey
Bogart » »



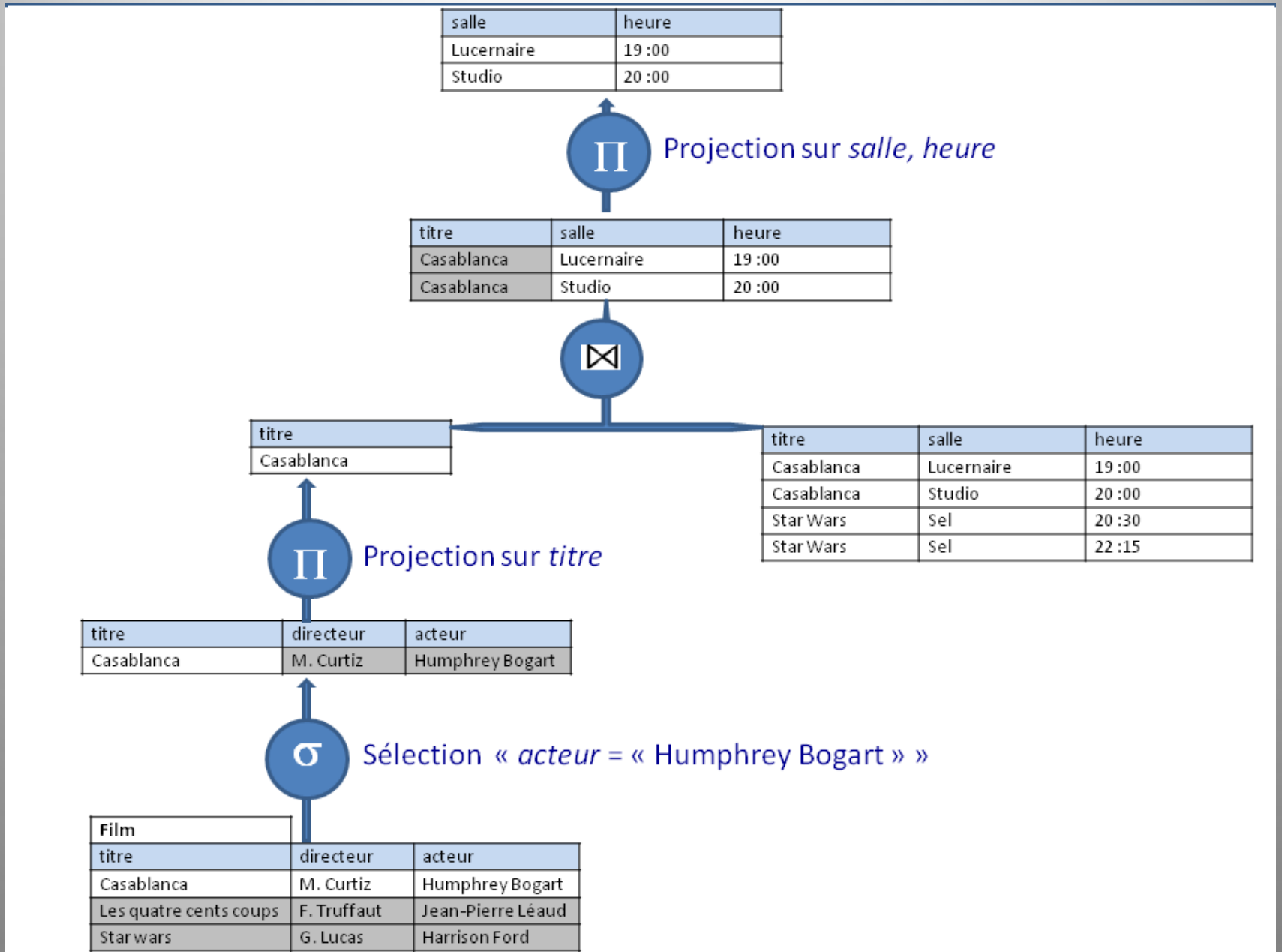

(a) For each f in film

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 algorithms

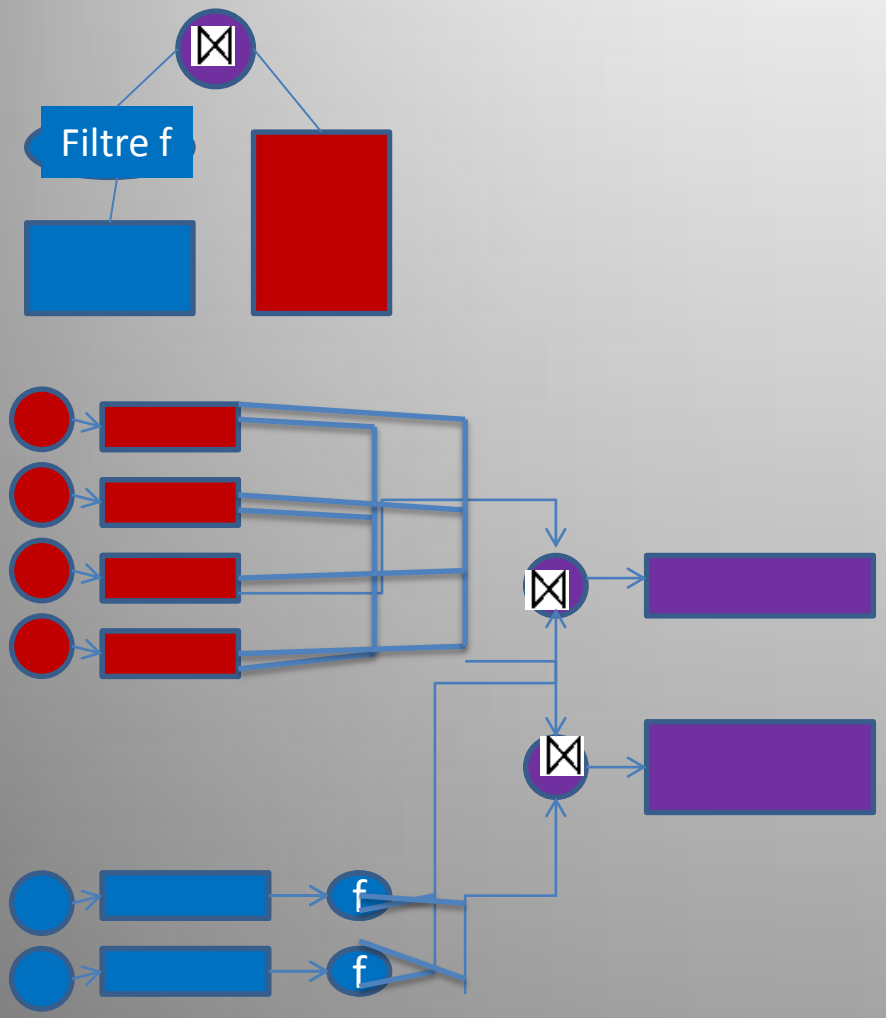
- E.g., merge join

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



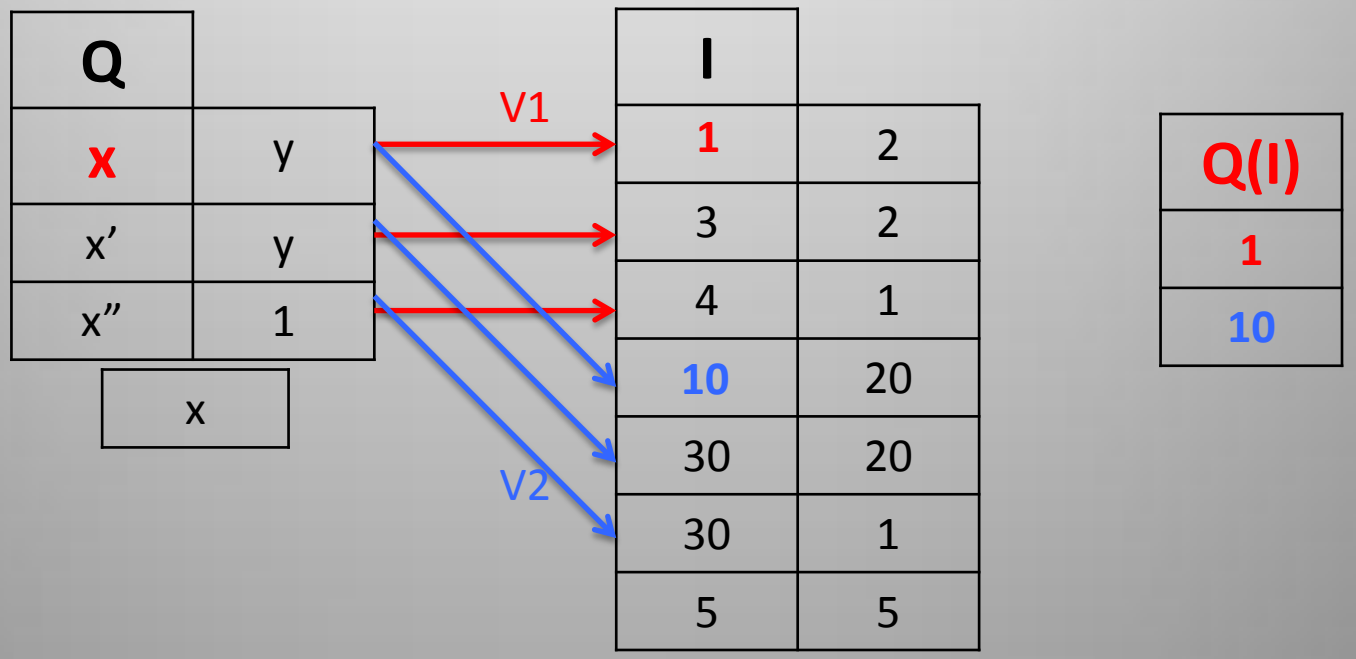

- Not all problems can take advantage of parallelism
- Data management can greatly benefit from parallelism
  - Relational calculus is in AC0
  - Acyclic join queries are embarrassingly parallelizable
  - Typically divide the data and work separately on pieces

# A jewel of databases

Containment of conjunctive queries

# Containment of conjunctive queries

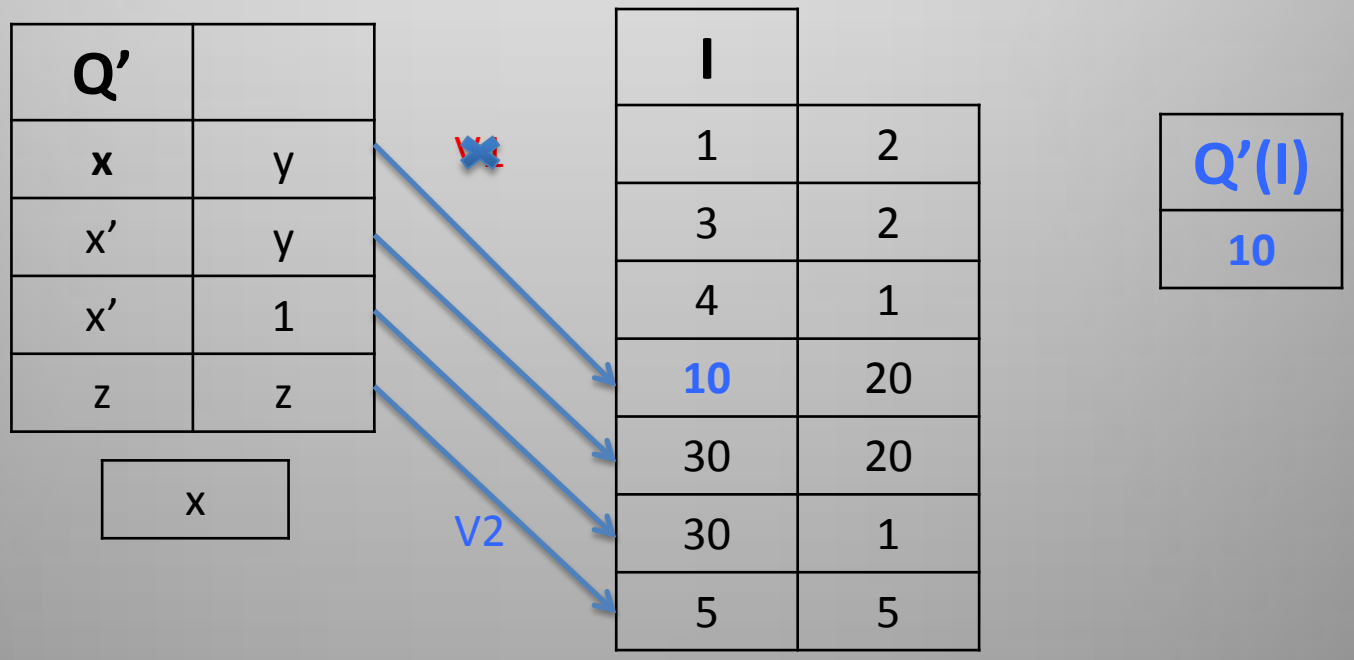
$Q = \{ x \mid \exists x', x'', y \; ( R(xy) \wedge R(x'y) \wedge R(x'',1)) \}$



# Another query

$\exists x', x'', y\ (\ R(xy) \wedge R(x'y) \wedge R(x'',1) \wedge x' = x'' \wedge R(z,z)\ )$

$Q' = \exists x', y, z\ (\ R(xy) \wedge R(x'y) \wedge R(x',1) \wedge R(z,z)\ )$



# Question

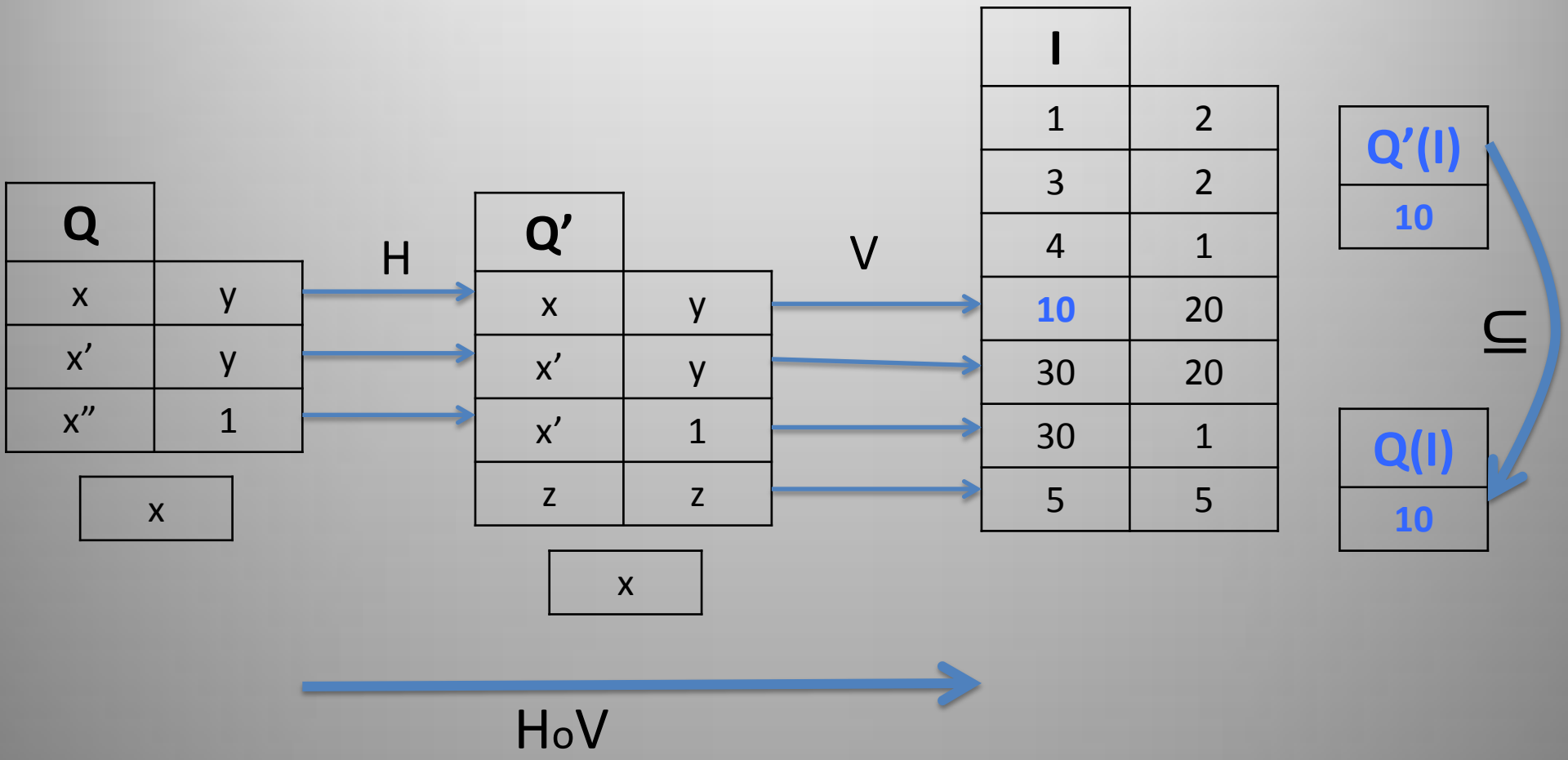
Definition: $Q' \subseteq Q$ if for all I, $Q'(I) \subseteq Q(I)$

$Q' \equiv Q$ if $Q' \subseteq Q$ and $Q \subseteq Q'$

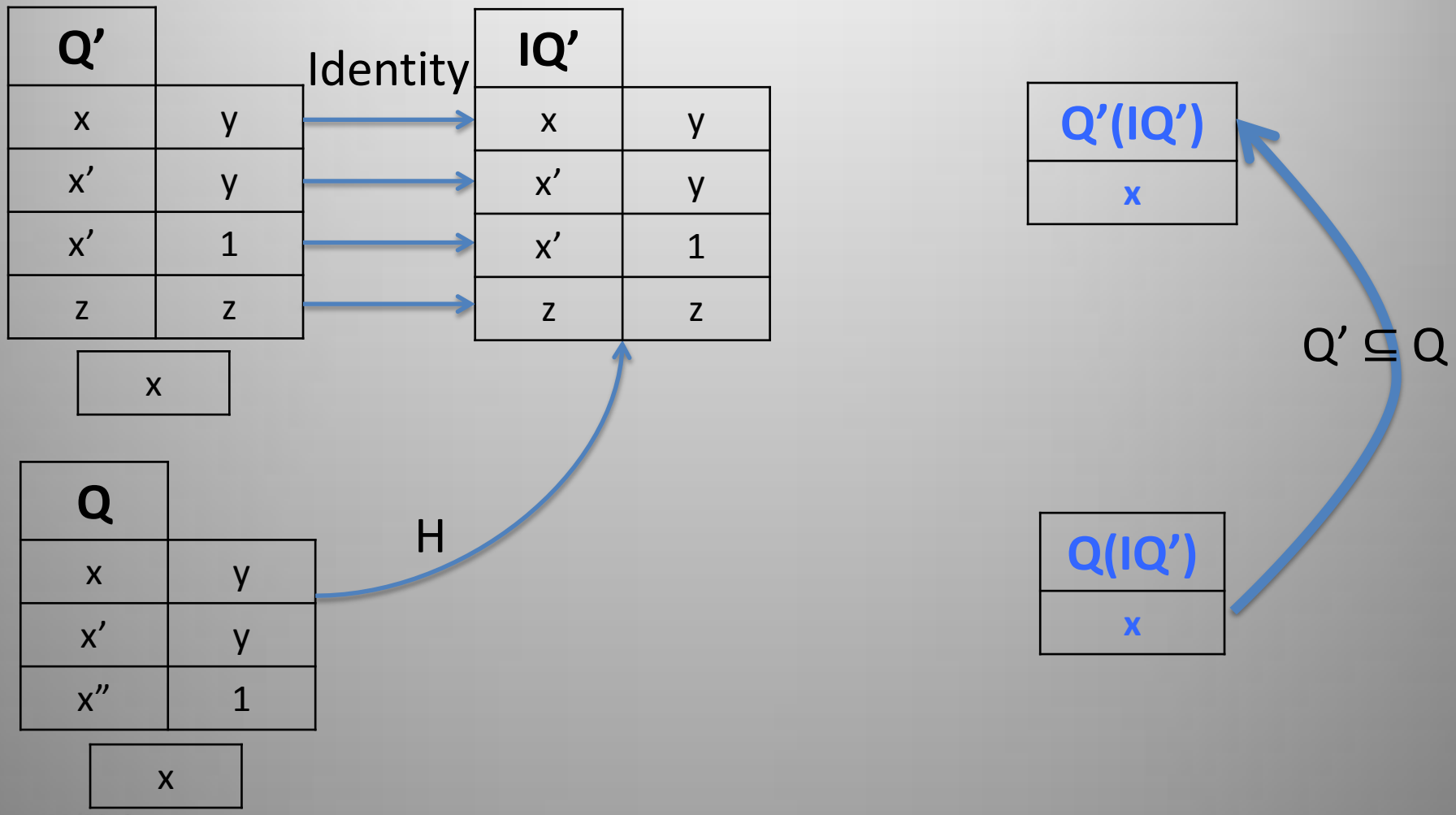
Problem: given Q', Q, test whether $Q' \subseteq Q$

Central issue for query optimization

# If there is a homomorphism from Q to Q', Q' ⊆ Q



# If Q’ ⊆ Q, there is a homomorphism from Q to Q’



$Q' \subseteq Q$ iff there is a homomorphism from Q to Q'

The problem is NP-complete

# Disjunction

Thm: $\cup Q'_i \subseteq \cup Q_j$ iff for each i, there exists j, $Q'_i \subseteq Q_j$

$\Leftarrow$) Suppose for each i, there exists j, $Q'_i \subseteq Q_j$

Let I be some instance: consider some I; there exists j, $Q'_i(I) \subseteq Q_j(I)$

$Q'_i(I) \subseteq \cup Q_j(I)$; so, $\cup Q'_i(I) \subseteq \cup Q_j(I)$; so, $\cup Q'_i \subseteq \cup Q_j$

$\Rightarrow$) Suppose $\cup Q'_i \subseteq \cup Q_j$

(to simplify assume wlg they are Boolean queries)

Consider some i and the instance $IQ'_i$

$\cup Q'_i(IQ'_i)$ not empty; so $\cup Q_j(IQ'_i)$ not empty;

For some j, $Q_j(IQ'_i)$ not empty;

So there is a homomorphism from $Q_j$ to $Q'_i$

# Beyond

- Negation: containment of FO queries is undecidable
  - co r.e. for finite relations  (r.e. for arbitrary relations)
- Recursion: containment of datalog is undecidable
  - Decidable for monadic datalog
  - Undecidable for monadic datalog over trees or strings (for infinite alphabet)

- Tableau homorphism is a particular case of subsumption in resolution theorem proving

# Conclusion

# Relational databases

Very successful in industry

Lots of results in academia

# Blend with other technologies

Logic programming

Production rule systems

Object-oriented languages

Workflows

Clusters of machines

Peer-to-peer architecture

# Always question everything

In industry: to challenge the well established guys

In academia: to discover new problems

Revisit the models, languages, principles

Main motivations

- To facilitate application development
- Performance to scale to always more data and queries
  - For extreme applications that cannot be supported by standard technology
  - Web search engines with huge volumes of data/queries
  - World wide banking with huge volumes of transactions
- To offer more in terms of reliability, security, etc..

# For instance, what can be questioned

| Relational model | Beyond |
|---|---|
| Entries in relations are atomic values | Entries are set of values |
| | Missing data, probabilistic data |
| | Spatio-temporal data |
| Data are regular | Semistructured |
| ACID | Weaker concurrency |
| Universal | Specialized: noSQL |
| Data are persistent | Persistent queries on data flows |
| Data are static | Data & behavior: Object databases<br>Active databases |
| Constraints are static (FDs, etc.) | Triggers |
| … | |


*Foundations of databases*, S. Abiteboul, R. Hull, V. Vianu. *Addison-Wesley*. 1995. www.webdam.inria.fr/Alice

*Database Systems: The Complete Book,* J.Widom, J.D. Ullman, H.Garcia-Molina

informatiques mathématiques

C A C H A N