Beyond the Relational Model

Serge Abiteboul

INRIA Saclay, Collège de France et ENS Cachan
Recall for first lecture:
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
  – To offer more in terms of reliability, security, etc..

We study here some of the main attempts to go beyond the relational model
Organization

Trees and XML
Graphs and object databases
NoSQL
OLAP (On-line analytical processing)
Conditional tables

Next class: Semantic Web
Trees and XML
Introduction

Trees are useless

A tree is a tree. How many more do you have to look at?

Ronald Reagan, governor of California, opposing the expansion of Redwood National Park (1966)

Knowledge lives in trees

But of the tree of the knowledge of good and evil, thou shalt not eat of it: for in the day that thou eatest thereof thou shalt surely die.

Genesis, 2. 17

We don’t need anything beyond relations. These things are useless. Reject!

Anonymous referee (circa 1990)

The Bible does not say “But of the two dimensional table of knowledge of good and evil …”
Using trees to represent data: an old idea

From the 60s and IMS (Hierarchical database model)
  – But fully procedural languages and records at a time
All really started in the 80s and Non-first-normal-form
  – François Bancilhon in France et Hans Schek in Germany
  – PhD thesis of Nicole Bidoit
The first class was on relations. Now what?

Data would prefer to live in infamous nested relations aka V-relations aka N1NF relations aka NF2 relations.

<table>
<thead>
<tr>
<th>Name</th>
<th>Child</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Toto Lulu</td>
<td>Jaguar 2CV</td>
</tr>
<tr>
<td>Bob</td>
<td>Mimi Zaza</td>
<td>Mustang Prius</td>
</tr>
</tbody>
</table>
The devil is in the details

V-relations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

A is not a key
The size is now possibly exponential in the size of the domain

A is a key
No new power

N1NF-relations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

A is not a key
The size is now possibly exponential in the size of the domain
Complex object model:
set and tuple constructors

Families

Name Peter Cars Name BMW Year 2010
Children

Name Peter Cars Name 2CV Year 1976
Children

Name Mimi Sex F

Name Zaza Sex F

Name Toto Sex M

24/2012
Logic for complex objects

**Logic**: main novelty – variables denoting sets

Example: AbouBanat query

\{ T.Father | \text{Families}(T) \land \forall \ X,x \ ( T.\text{Children} = X \land x \in X \implies x.\text{Sex} = F ) \} 

The father of only girls
Algebra for complex objects

Set of sets

Unnest

Nest

Unnest

Name | Child | Car
---|---|---
Alice | Toto |
Bob | Mimi |
Bob | Mustang |
Bob | Zaza |
Bob | Prius |

Name | Child | Car
---|---|---
Bob | Mimi |
Bob | Mustang |
Bob | Zaza |
Bob | Prius |

Identity
Results

Equivalence theorem: algebra and logic have same expressive power

Remark: one can compute transitive closure using algebra/logic (Cool!)

Each new level of nesting introduces one more exponential

- A query is in the algebra/calculus iff it has elementary time complexity (similarly space complexity)

\[2^2 \cdots 2^n\]
From complex objects to semistructured data

- **Families**
  - **Name** Peter
    - **Cars**
      - **Name** BMW
        - **Year** 2010
      - **Name** 2CV
        - **Year** 1976
  - **Name** Mimi
    - **Sex** F
  - **Name** Zaza
    - **Sex** F
  - **Name** Toto
    - **Sex** M

Date: 3/24/2012
Revolution 1: more flexibility

Families

Name Peter Cars

Name BMW Year 2010

Name Toto Sex M

Name Zaza Sex F

Annotations
Trash

Name Peter Cars

Name Mimi Sex F

Name 2CV Year 1976

Children

Children
Revolution 2: get ride of *-nodes and name all nodes
XML = ordered, labeled, unbounded trees

```
Families
    Family
        Name Peter
        Cars
            Car
                Name BMW
                Year 2010
        Children
            Name Toto
                Sex M
            Name Zaza
                Sex F
        Name Zaza
            Sex F
            Ann. Trash
        Cars
            Name 2CV
                Year 1976
```
This is better adapted to a Web context

Self describing data: No separation between schema and data
Flexibility
Not such a big deal
A syntax for inlining and exchanging data

```
<families><family><name>Peter</name><Cars><Car><Name>BMW</Name><Year>2010</Year></Car></Cars><Children><Child> ...
```

The more things change, the more they stay the same
What else? The trees are unbounded

Like nested relations, trees are **unbounded in width**
Unlike nested relations, they are **unbounded in depth**
One can simulate 2 counter machines with 2 branches

- I am still looking for a real application that simulate 2 counter machines with XML documents?
- XML documents are rarely deep

But even for bounded trees there are fun questions

- Rich study of query languages
- Typing and semantics
What else? the trees are ordered
Unranked labeled ordered trees = XML

Ignore order
Classical optimization

Respect order
Totally new ball game

Reconcile?

Order is often painful for optimization
The XML world

Typing
  - **Tree automata**, DTD, XML Schema, Relax NG...

Query languages
  - **XPATH**
    ```xml
    article[1]/auteurs/auteur[2]
    ```
  - **Xquery**
    ```xml
    FOR $ p IN document("bib.xml") // publisher
    LET $ b: = document("bib.xml") // book [publisher = $ p]
    WHERE count($ b)> 100
    RETURN $ p
    ```
  - **Monadic datalog, FO, Pebble automata**...

Transformation language: XSLT

Other standards around XML
  - **SOAP, DOM**
  - **XML dialects**: RSS, WML, SVG, XLink, MathML

Lots of open source software
Query containment
(continuing jewel of 1st class)

- Recall **Homomorphism Theorem**

\[ q_1 \subseteq q_2 \text{ iff there is a homomorphism from } q_2 \text{ to } q_1 \]
Tree pattern query – semantics

Tree pattern query
Tree pattern query – semantics

Tree pattern query

- Graph representation of a tree pattern query with nodes and edges labeled 'r', 'a', 'b', 'c', and '#'.
Tree pattern query containment

Tree pattern containment

There is no homomorphism from q2 to q1
Tree pattern query containment

• But $q_1 \subseteq q_2$

$q_2 =$ there is a path of length at least 2 from the root $r$ to a leaf $c$

$q_1$ & the $#$ is not an $a$
  – There is such a path

$q_1$ & the $#$ is not a $b$
  – There is such a path
XML storage

In a file system
   - A directory is now becoming a searchable database

In a native XML DBMS
   - eXist: open source
   - MonetDB

In a relational DBMS
   - Blades for storing XML

Several types of API
   - XQJ XQuery API for Java specification (XQJ)
   - XML:DB JDBC for XML databases

Trend: reduce the separation between DBMS and file systems
Graphs and object databases
Object databases =
Object-oriented languages + Databases

- Object-oriented language
  - Object = data + behavior
  - Objects encapsulate data

- Standard database features
  - Transactions
  - Queries, etc.

- Object data model
  - Object identity
  - Complex structure (typically set & tuple constructors)
  - Classes: type and class hierarchies
  - Inheritance
Architecture: relational vs. object

- **Relational server**
  - Application
  - JDBC / ODBC
  - Each read is to the server

- **Object DB server**
  - Application
  - Object cache & cache manager
  - Some reads are local
  - Queries or OIDs
  - Some reads are local
The same object from disc to memory

Greatly facilitates developing applications
- A single data model (richer)
- Integration with an object programming language,

Performance because of complex objects
- Join between multiple tables replaced by navigation between objects
- Object often in local cache

In memory object

Query Navigation

Same object in object database
Moderate industrial success

• Object database systems
  – 1989: Object Database Manifesto (Atkinson, Bancilhon et al)
  – Pioneers: O2, ObjectStore, Objectivity, Versant...
  – ODMG Standard, OQL

• Object-Relational
  – Dirty attempts to use relational back-ends to store objects
But the ideas are spreading

Standard around Java: JDO
Popular open source software such as Db4o
Frameworks for languages with persistence: JPA, DataObjects.NET
NoSQL
Motivations for NoSQL

DBMSs pay a high overhead for their universality
Avoid this overhead for very demanding applications

Major overheads to avoid:

1. Buffer Management: cache disk blocks in memory
2. Locking: for the management of concurrency. Transactions must wait for the release of locks
3. Latching: Short term locks used for access structures that are shared as B-tree
4. Logging: Every update is written in the log that is forced to disk

Analysis of OLTP applications [Harizopoulos & AL08]:

- 35% buffer management
- 21% locking
- 19% latching
- 17% logging
Specialized data management systems

Specialized for certain types of queries
Specialized for certain aspects such as scalability
In return: sacrifice universality
  – Sacrifice certain types of queries like the join
  – Sacrifice some features, such as concurrency

No SQL
  – Non-standard systems for data management
  – Typically simpler data models
  – (Support sometimes SQL)

Warning: the term NoSQL is also used sometimes for systems based on the contrary, more complex models: Object /XML / RDF – not here
NoSQL: different flavors

**Extreme performance**
- Massive scalability
- Massive distribution
- Total availability

**Specialization**
- High transaction rates
- Simple OLAP queries on very large volumes

**No universality**

**Less independence**
- No 3 levels

**Less abstraction**
- Not relational and SQL
- Simple Data: key / value
- Simple queries

**Loss of functionality**
- No ACID (strict)
- Less typing and integrity
- Simple access structures
- Simplistic API - no JDBC
Examples

Key / value store with weak consistency
   – Cassandra (Apache), Dynamo (Amazon)

Key / value store on disk
   – **Hadoop** Hbase (Apache), BigTable (Google)

Document store with N1NF
   – MongoDB (free software)

Main memory database single-threaded for OLTP
   – VolTDB

Massively parallel database for analysis
   – Greenplum, MySQL Cluster

And many more ...
The OLAP multidimensional model
Data get organized in cubes

<table>
<thead>
<tr>
<th></th>
<th>USA</th>
<th>Canada</th>
<th>France</th>
<th>UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pain</td>
<td>12</td>
<td>25</td>
<td>14</td>
<td>86</td>
</tr>
<tr>
<td>Fromage</td>
<td>23</td>
<td>68</td>
<td>45</td>
<td>25</td>
</tr>
<tr>
<td>Yaourt</td>
<td>12</td>
<td>95</td>
<td>65</td>
<td>42</td>
</tr>
<tr>
<td>Chocolat</td>
<td>44</td>
<td>22</td>
<td>33</td>
<td>18</td>
</tr>
</tbody>
</table>

+ more dimensions:
- Kind of customer
- Kind of sale (web, ...
Discussion

Ted Codd 1995
Evolution from spreadsheet
Provide multidimensional views for analysis
  – Hierarchical domains – Time: day, week, month, year
  – Aggregation
Example of queries
  – 5 top demography groups buying videos
  – Products sold in France where rejection rate diminished by more than 5%
Querying, navigation, reporting
Standard query language: MDX (MSFT, 1997)

**SQL**

*select, from, where, group-by*

Yields a table (2-dim)

Select columns from some tables

Filter lines with predicates in *where* clause

Aggregation using *group by*

**MDX**

*with, select, from, where*

Yields a cube (N-dim)

Select: select cube dimensions

With: specification on selected dimensions

Where: specification on non selected dimensions

Implicit aggregation

```sql
with member Measures.profit as Measures.StoreSales = Measures.cost
select
  {Measures.StoreSales, Measures.Profit} on columns,
  non empty filter(Product.ProductDepartment.members,
   (Product.currentMember, Measures.StoreSales) > 2000.0) on rows
from [Sales] where ([Time].[1997])
```
Conditional tables
Uncertainty

Lots of uncertain data
Studied in academia
Not much in industry
  – Null values in SQL – Trash semantics
  – No clear standard
We will see here in brief
  – Conditional tables
  – How to turn them probabilistic
Conditional tables & uncertainty

<table>
<thead>
<tr>
<th>Friend</th>
<th>Location</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>London</td>
<td>E</td>
</tr>
<tr>
<td>Bob</td>
<td>London</td>
<td>E \land F</td>
</tr>
<tr>
<td>Alice</td>
<td>Paris</td>
<td>\neg E</td>
</tr>
<tr>
<td>Lucile</td>
<td>London</td>
<td>F</td>
</tr>
</tbody>
</table>

4 possible worlds
## Conditional tables & probabilities

<table>
<thead>
<tr>
<th>Friend</th>
<th>Location</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>London</td>
<td>E</td>
</tr>
<tr>
<td>Bob</td>
<td>London</td>
<td>E ∧ F</td>
</tr>
<tr>
<td>Alice</td>
<td>Paris</td>
<td>¬E</td>
</tr>
<tr>
<td>Lucile</td>
<td>London</td>
<td>F</td>
</tr>
</tbody>
</table>

- E is 80%
- F is 40%

<table>
<thead>
<tr>
<th>Friend</th>
<th>Location</th>
<th>Location</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>London</td>
<td>Alice</td>
<td>Paris</td>
</tr>
<tr>
<td></td>
<td>London</td>
<td>Alice</td>
<td>Paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lucile</td>
<td>London</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alice</td>
<td>Paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lucile</td>
<td>London</td>
</tr>
</tbody>
</table>

- 32%
- 48%
- 8%
- 12%
A jewel of databases

The worst way I know of computing transitive closure
Calculus for complex objects

The points reachable from $a$ in a graph $G$

\[
\{ x \in \forall R \ ( ( R(a) \land \forall y,z \ ( R(y) \land G(y,z) \Rightarrow R(z) ) ) \Rightarrow R(x) ) \} 
\]

$x$ is reachable from $a$ if $x \in R$

for each set $R$ containing $a$ and “closed under” $G$
Algebra for complex objects

The points reachable from $a$ in a graph $G$

$$D := \mathbb{P}_1(G) \cup \mathbb{P}_2(G) : \text{the nodes in } G$$

$$P := 2^D : \text{the powerset of } D$$

$\Theta$ an algebraic query (in classical relational algebra) equivalent to:

$$R(a) \land \forall x,y \ ( R(x) \land G(x,y) \Rightarrow R(y) )$$

$$Q := \sigma_\Theta(P) : \text{the subsets of } D \text{ satisfying } \Theta$$

$$Q' := \mathbb{P}_1(\sigma_1 \triangleright_2 (Q \times Q)) : \text{the non-minimal elements in } Q$$

$$Q'' := Q - Q' : \text{the minimal elements in } Q \text{ (unique)}$$

unnest(Q'') : the points reachable from $a$ in $G$
Complexity

- Calculus
- Quantify Over Sets
- Quantify Over Sets of sets
- Quantify Over Sets of sets of sets
- Quantify Over Sets of sets of sets of sets
- ptime
- exptime
- 2exptime

Calculus + order + FP
Quantify Over Sets + FP
Quantify Over Sets of sets + FP
Quantify Over Sets of sets + FP
Quantify Over Sets of sets of sets + FP
Quantify Over Sets of sets of sets of sets + FP
Conclusion
Conclusion

Regain the 3 principles
   – Is this desirable?

Build a unifying theory
   – Is this desirable?

Develop new systems

Develop new theories

Consider richer semantics
   – Semantic Web: next time
Merci !