Towards a unified model for heterogeneous data

Matteo Magnani

Dept. of Computer Science
University of Bologna
Via Mura A.Zamboni, 7
40127 Bologna
Italy

Danilo Montesi

Department of Mathematics
and Informatics,
University of Camerino
Via Madonna delle Carceri, 9
Camerino (MC)
Italy

Introduction

- In a growing number of applications the classic relational model captures only a fraction of relevant data.
  - Biological databases.
  - XML data/document repositories.
  - The World Wide Web.
  - Multimedia data repositories.
- Specialized systems, models, and theories are available, customized to specific kinds of data.
- It is not unusual to have applications based on combinations of these systems.

Introduction

- In the (not so far) future, everything about our life will be stored in a (not very large) database [2].

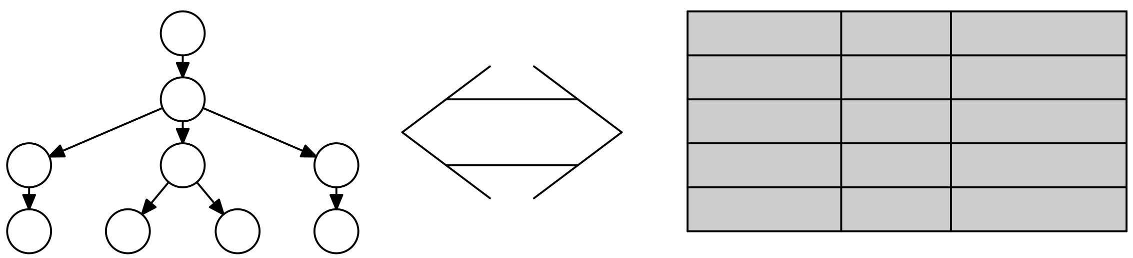
Introduction

- The *objective* is to have a single system storing heterogeneous data, and providing unified and homogeneous representation and manipulation functionalities.
- Existing system are already providing support for heterogeneous data.
- For instance, many information systems manipulate mixed XML and relational data.

The case of XML data

First main approach: Adaptation.

- Trees are converted to tables, and vice versa.
- SQL/XML (Oracle, DB2), FOR XML clause (SQL Server).



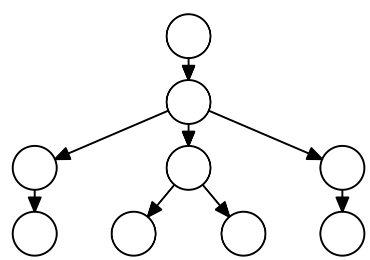
Adaptation

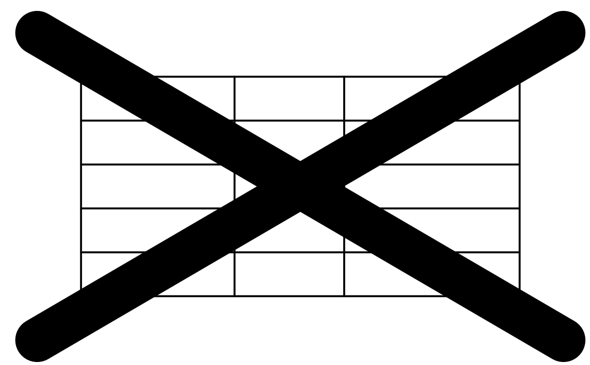
- XML-enabled systems readily available.
- Usable if the difference is *more on the format than on the model*.

The case of XML data

Second main approach: Rethinking.

- New ad hoc systems.
- Tamino, eXist, Galax.





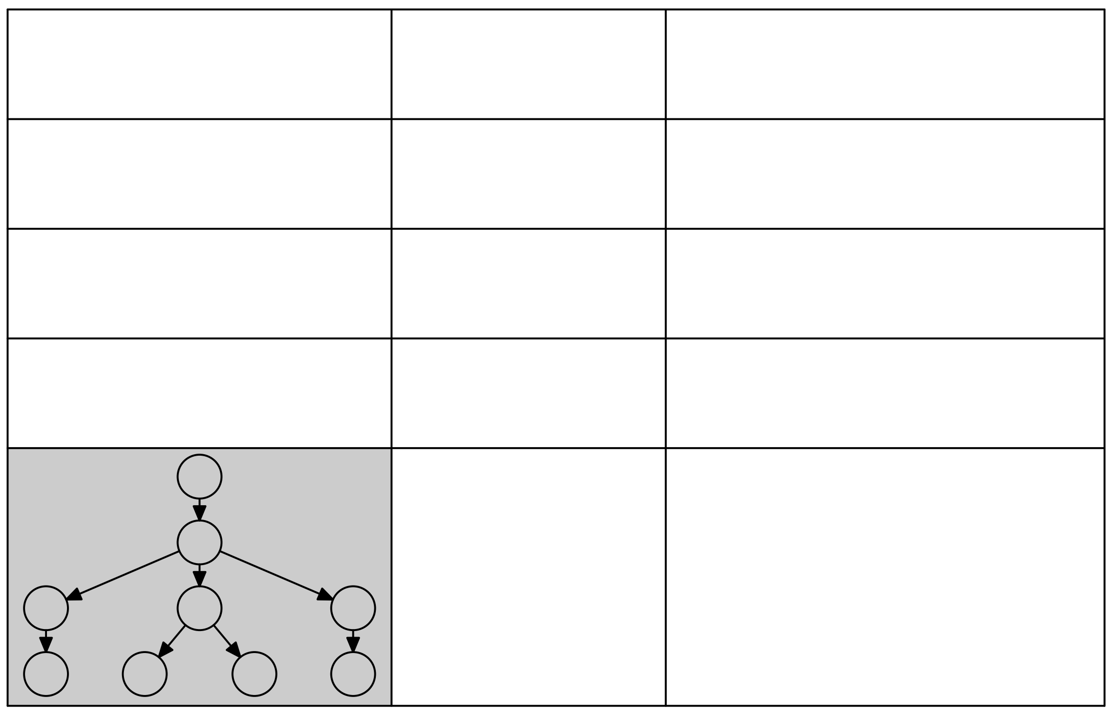
Rethinking

- It needs time (more suited to academics and small applications than commercial/critical systems).
- It promises better results.
- At the end, many functionalities are not substantially different from those found in traditional systems → We would like to identify and change only those features that are specific to XML.

The case of XML data

Third main approach: Extension.

- New complex types are defined.
- Oracle's XML Type.



Extension

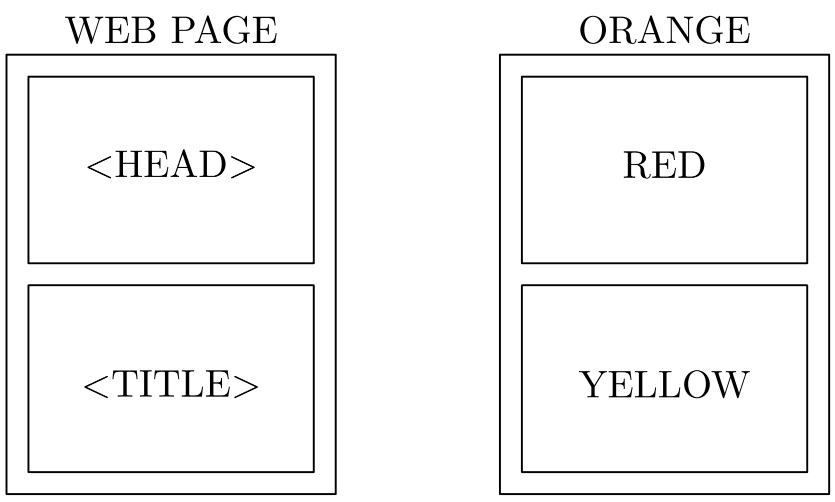
- Good choice to take care of object heterogeneity.
- Still constrained to be embedded into non-flexible relational structures.
- Very useful when the new data is not much structured.
- Otherwise, we must reimplement database functionalities inside the objects (=Rethinking).

Some basic considerations (I)

- There are operations which cannot be described by a simple, general, and compact model, as they are meaningful only when applied to particular kinds of data.
- For instance, the extraction of a color histogram from an image.
- Therefore, a model for heterogeneous data cannot describe everything, and must hide the details about elementary pieces of data.
- The level of detail is not absolute, but it depends on our requirements.

Some basic considerations (II)

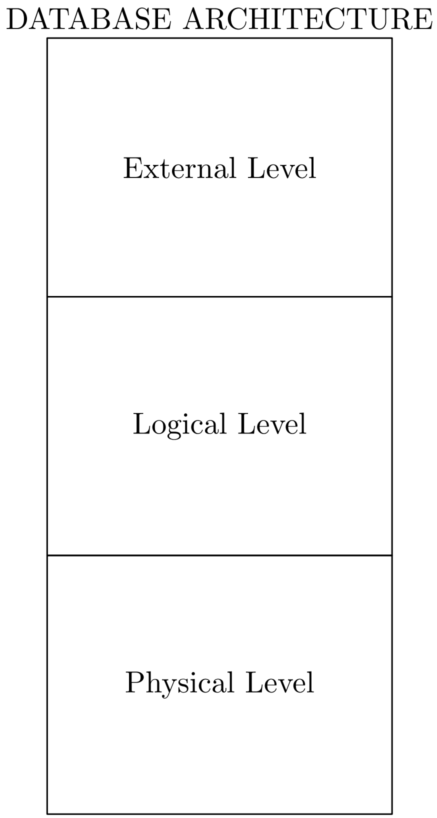
- There is a (limited) number of aggregation patterns for elementary kinds of data.



Some basic considerations (III)

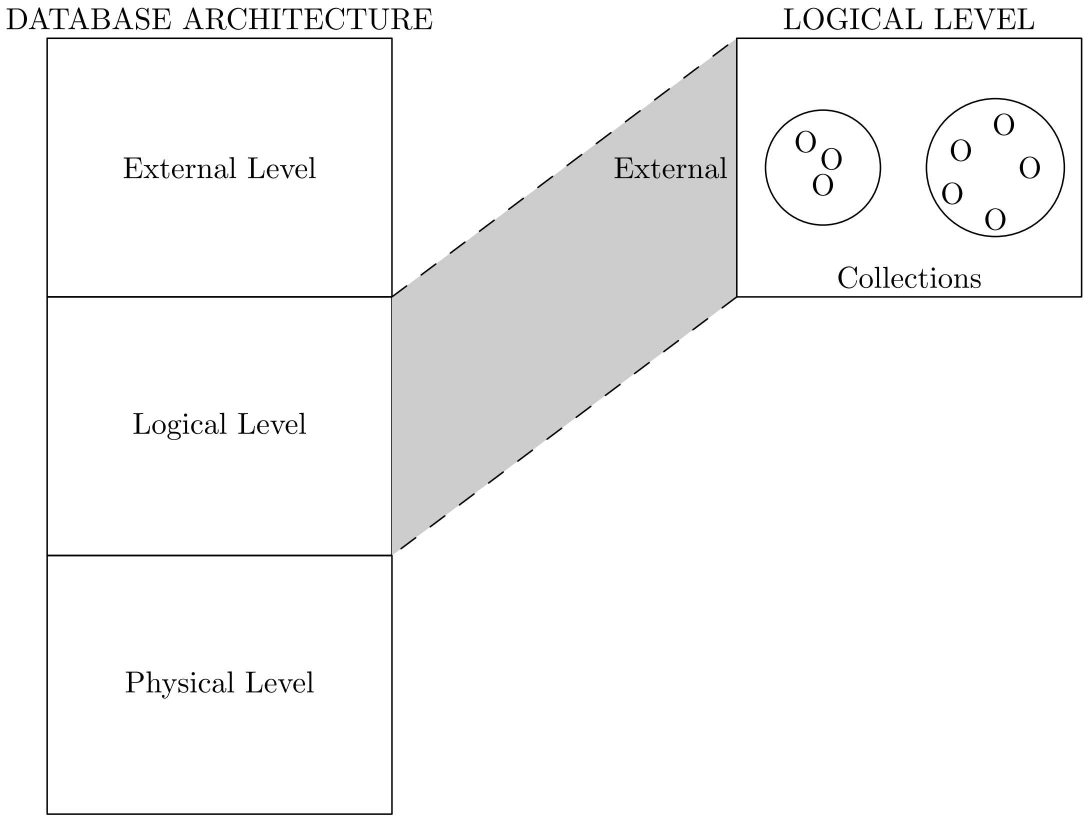
- There are operations that we can nearly always perform on collections of data, and that can be modeled aside from its peculiarities.
- 'Give me all O where P.'
- 'Give me all Web Pages where the Title is "home page".'
- 'Give me all Tuples where the Identifier is '001'.'
- 'Give me all Images where the prevalent color is Green.'

Overview of the Model



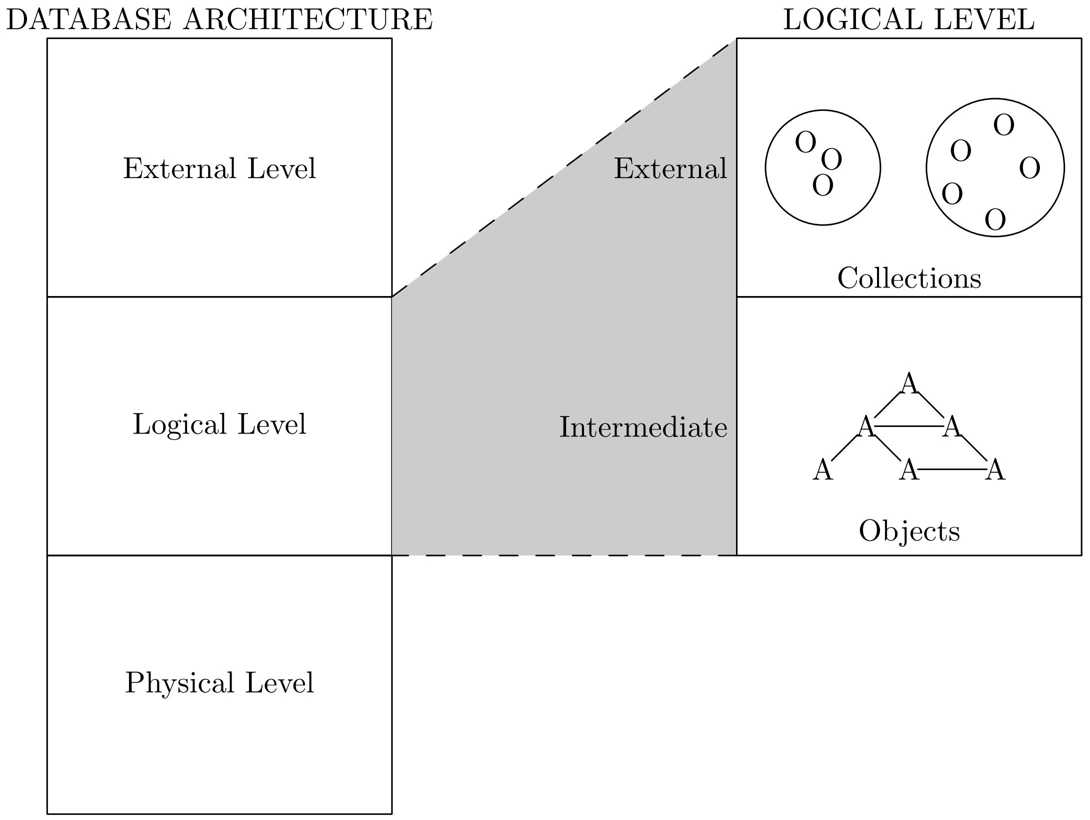
- 3-level architecture of (relational) databases.
- We re-define the logical level.

Overview of the Model



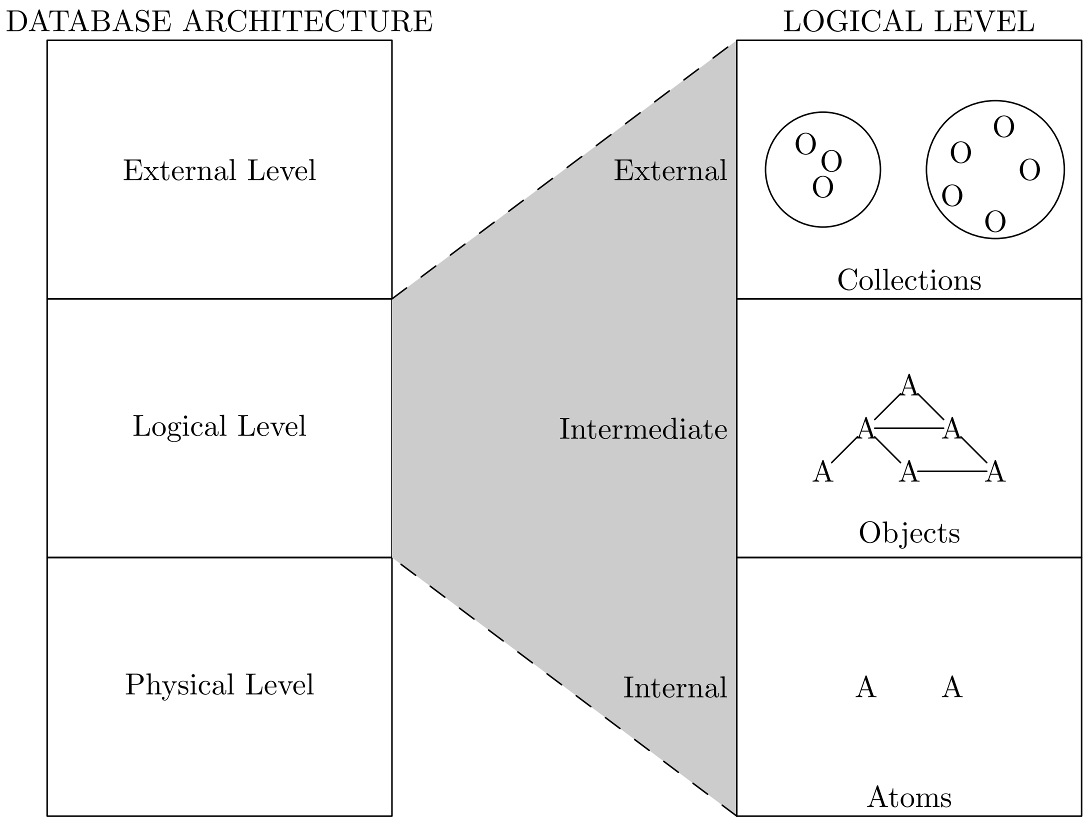
- Collections are sets, multisets, lists of objects.
- For example, relations are sets of tuples.

Overview of the Model



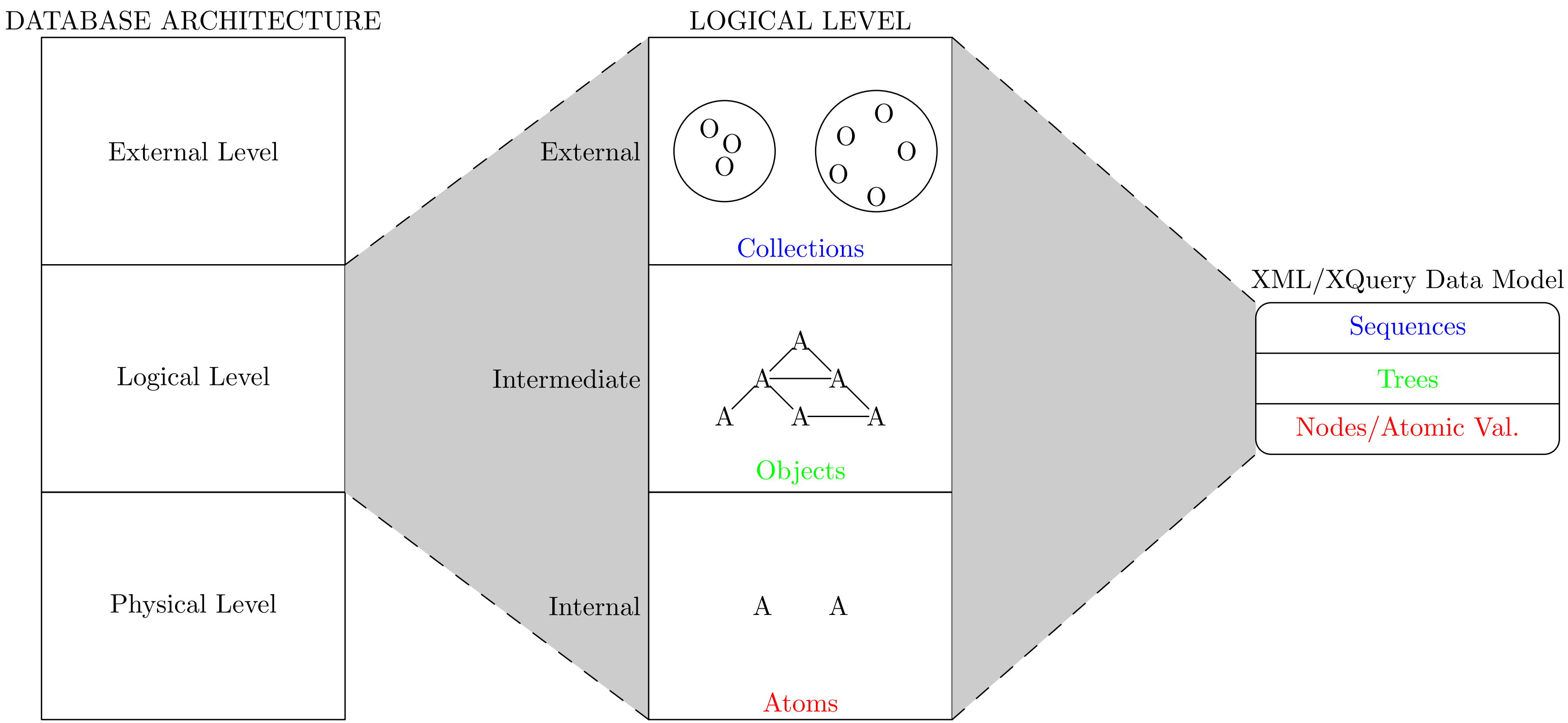
- Objects are aggregations of atoms.
- Objects can be tuples, trees, graphs.

Overview of the Model



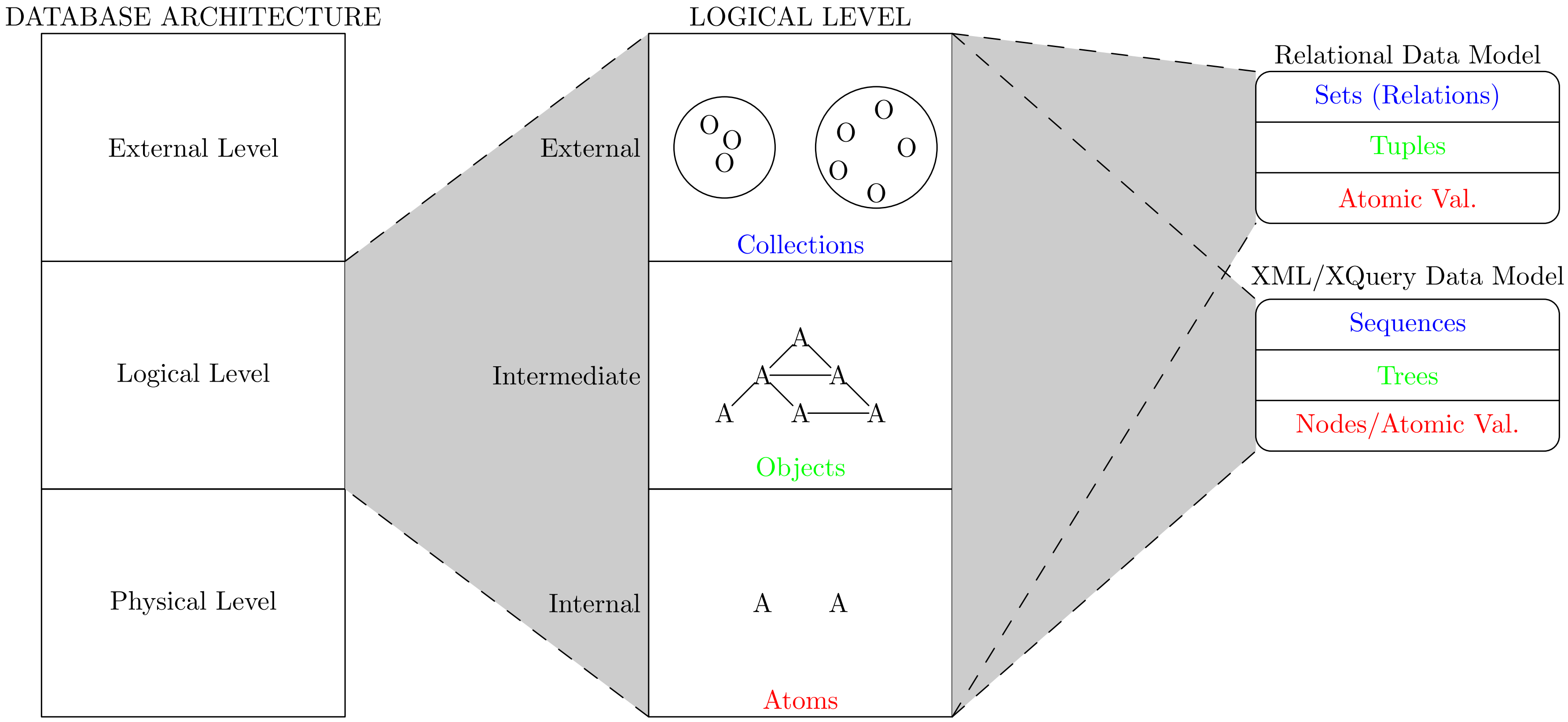
- Atoms hide data heterogeneity.

Overview of the Model



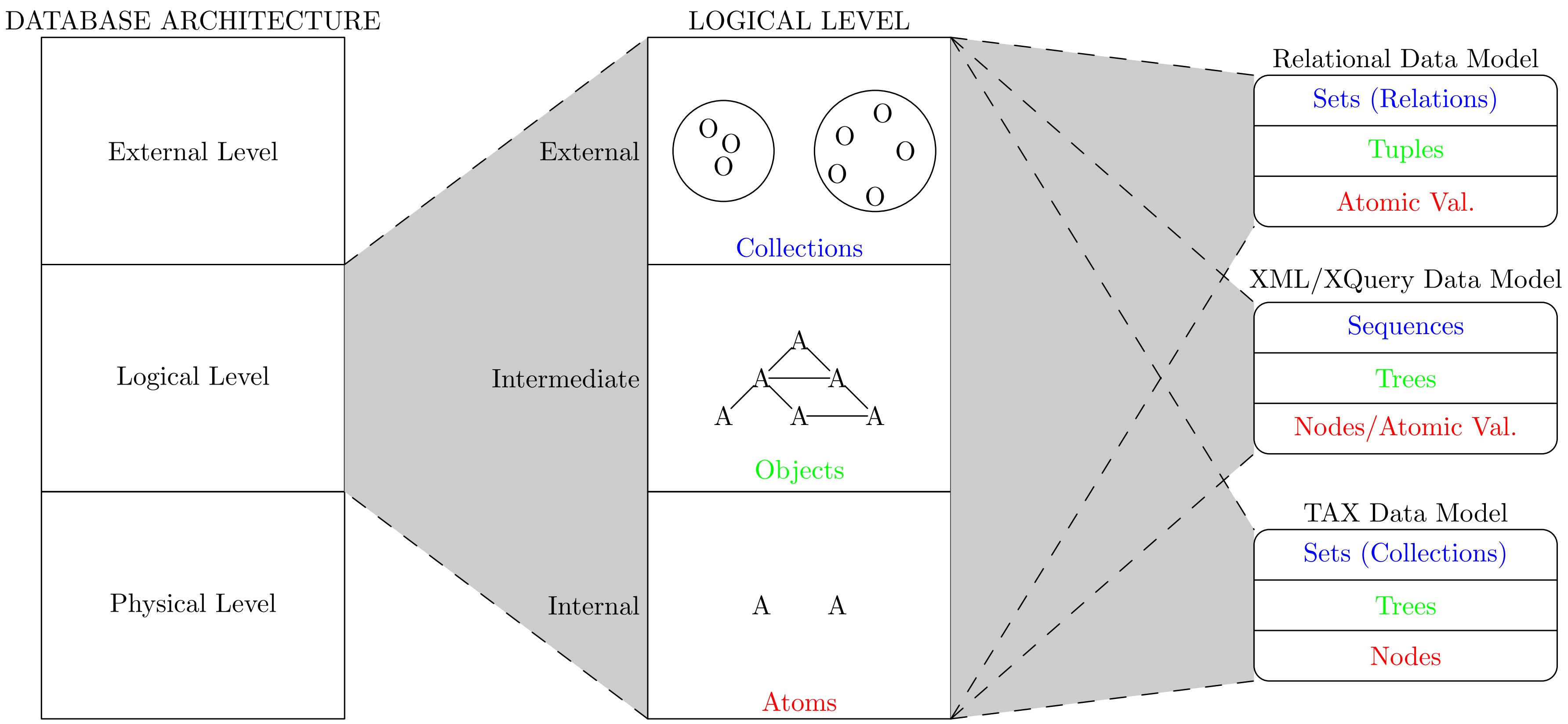
- The model instantiates to XQuery Sequences.

Overview of the Model



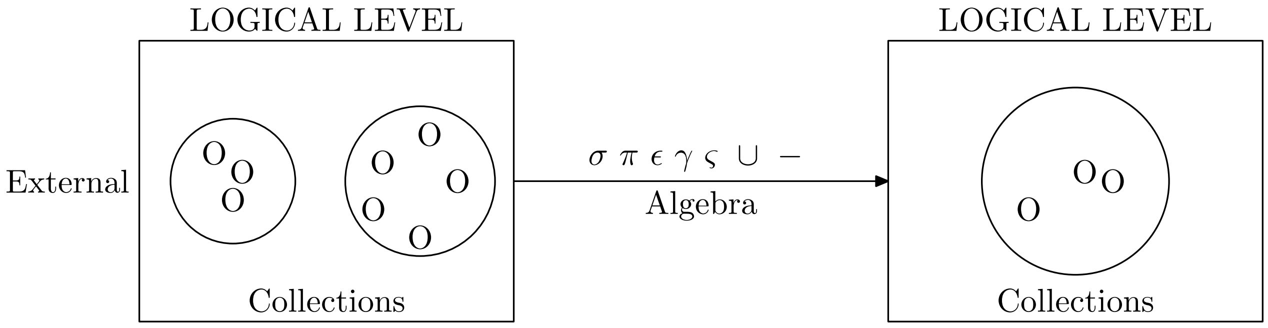
- The model instantiates to Relational Tables.

Overview of the Model



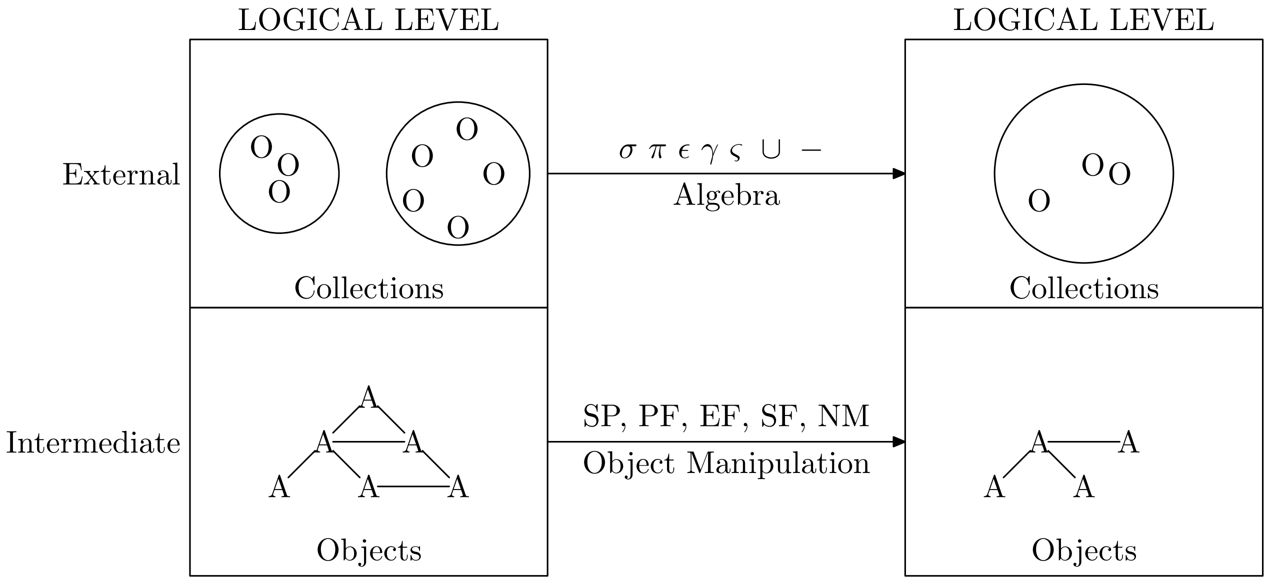
- The model instantiates to TAX Collections.

Overview of the Algebra



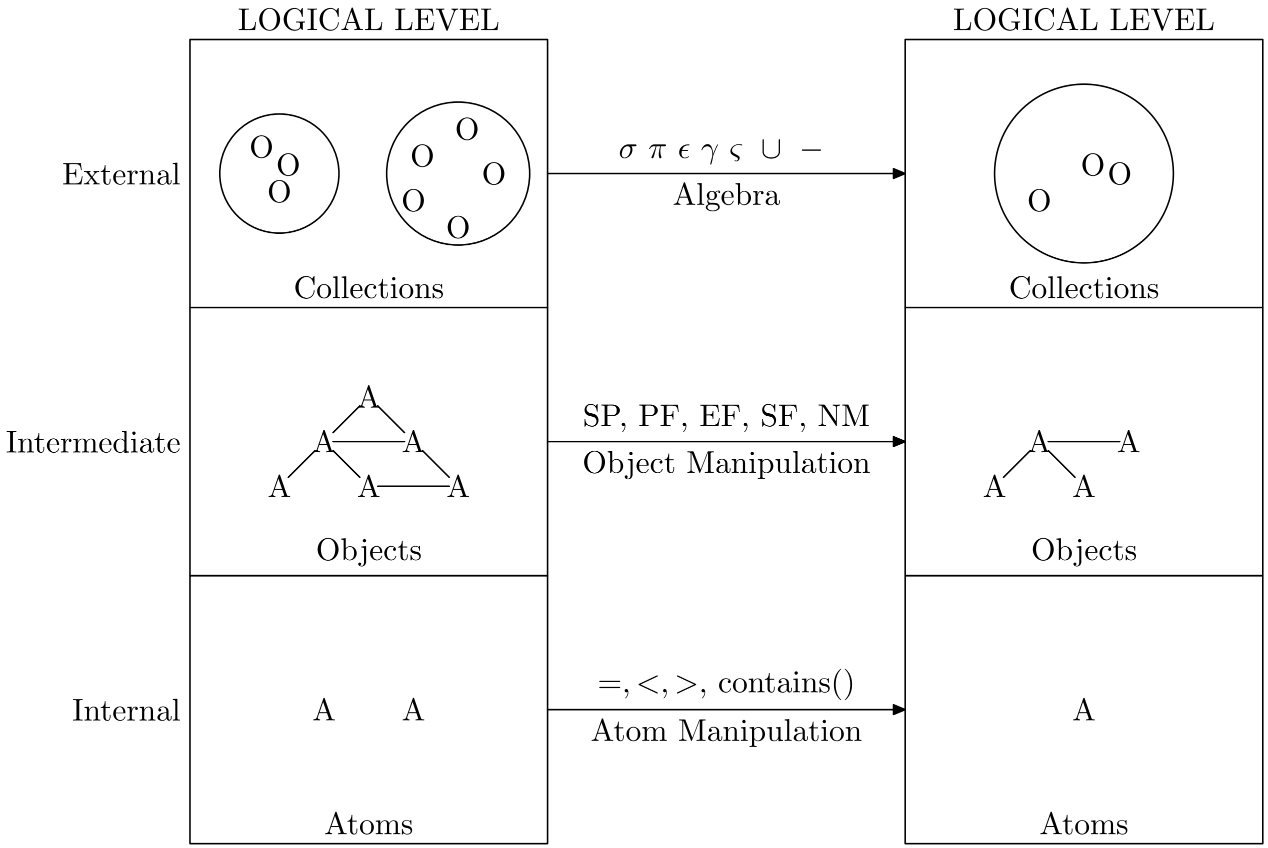
- Ex: $\sigma_P(C) = \{c \mid c \in C \wedge P(c)\}$
- Based on abstract properties of the parameters.
- P returns *true* if an object c satisfies some contraints, *false* otherwise.
- We want to define the algebra without specifying P and other object-dependent parameters.

Overview of the Algebra



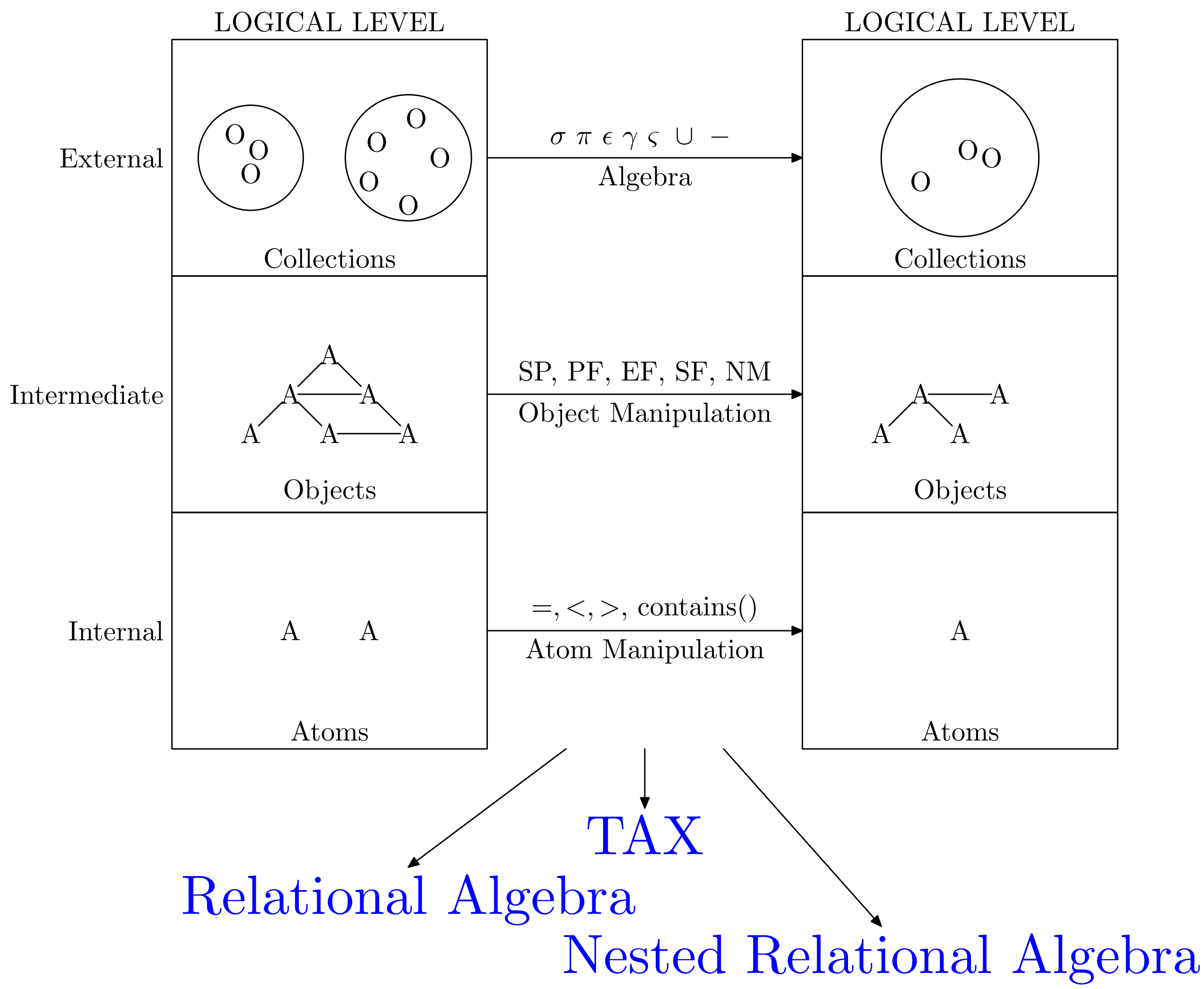
- A few kinds of parameters are enough to define the external algebra (SP,PF,EF,SF,NM).
- The intermediate language manipulates the internal structure of single objects (Ex: XPath).

Overview of the Algebra



- Atomic functionalities depend on the type of the atoms.

Overview of the Algebra



Expected Advantages

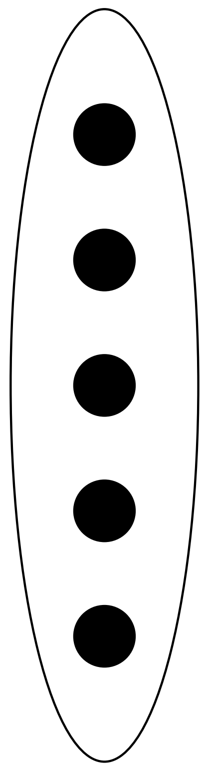
- For each layer, development of specific optimization techniques.
  - To manipulate large sets of objects.
  - To navigate XML trees.
  - To manipulate atoms (images, media, text).
- Easy extension of the system.
  - To add support for atomic types.
  - To add support for tuples, trees, graphs, and other aggregation patterns.

Parametric Algebra

- Relational Algebra (RA) works well with relational data.
- As the relational model is a specific case of ours, our parametric algebra should reduce to RA.
- Therefore, we use RA as a starting point to define our algebra.
- We first generalize its operators, then define new operators for missing functionalities.

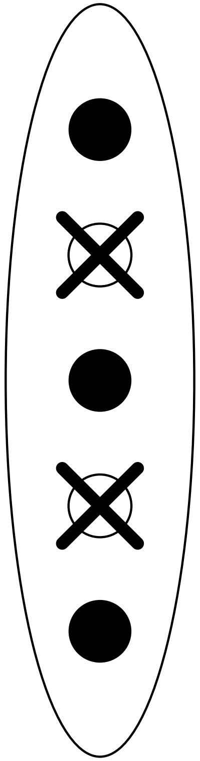
Parametric Algebra

- σ and π are two basic functionalities borrowed by the relational context.



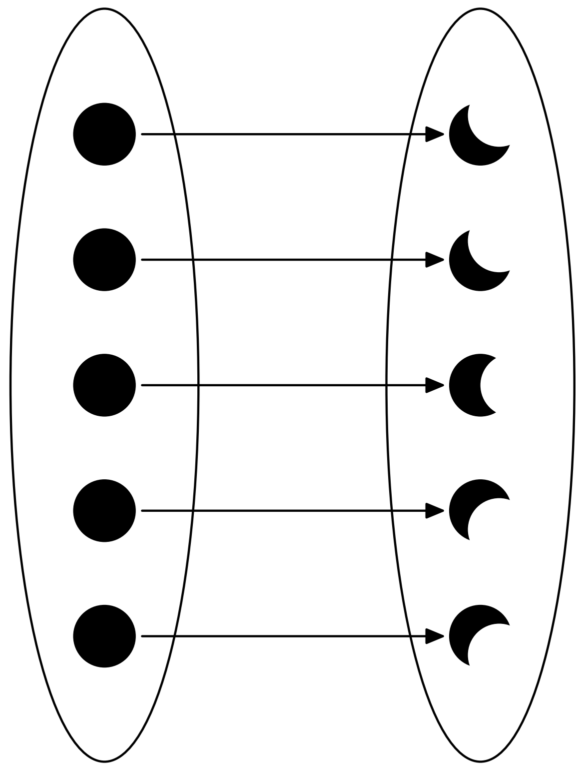
Parametric Algebra

- σ selects some of the objects in a collection.



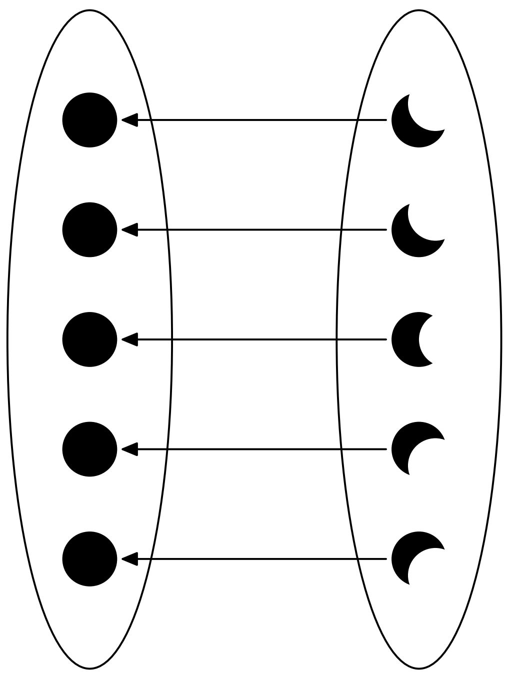
Parametric Algebra

- π extracts part of the objects.



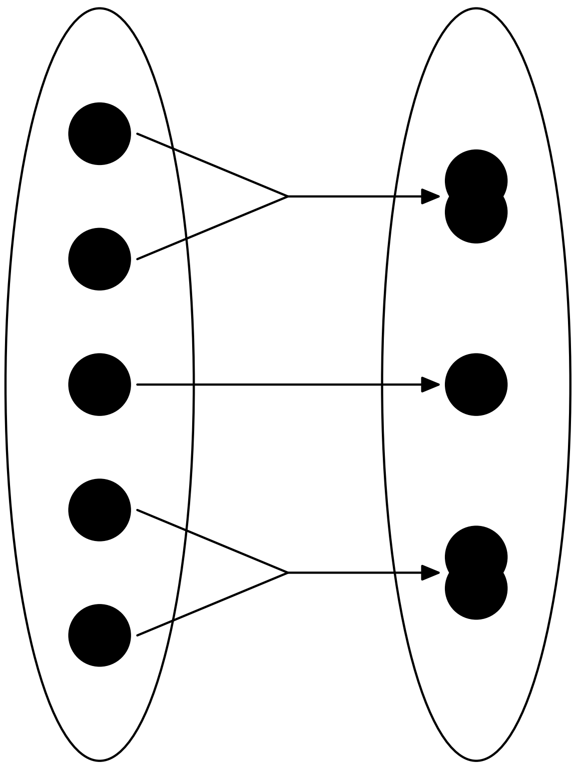
Parametric Algebra

- The opposite of projection is embedding (ϵ).
- Information to construct new data provided as a parameter of the operator.



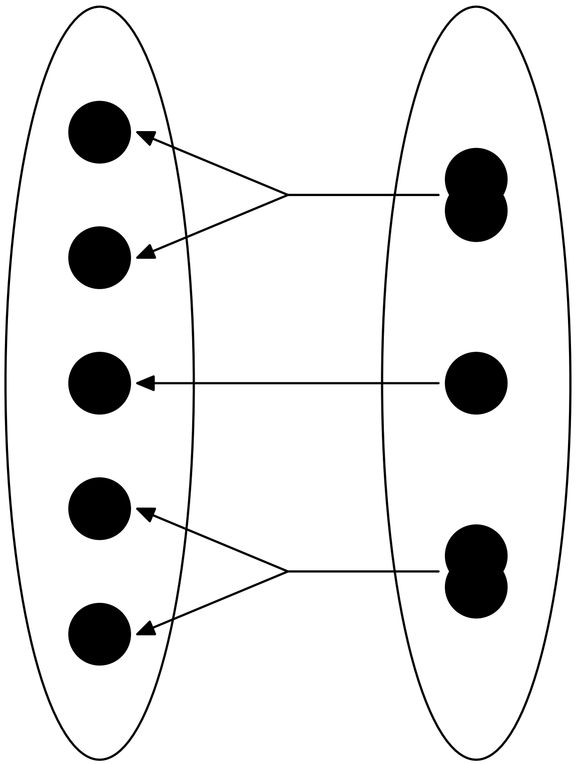
Parametric Algebra

- As objects are not supposed to be elementary, we also need a grouping operator (γ).



Parametric Algebra

- The opposite of a grouping is a splitting (ς).



Parametric Algebra

- Finally, we have binary operators, to combine two collections.
- $\cup$, $\bowtie$, $-$
- Some other operators cannot be defined until we instantiate objects, as they change their internal organization.
- We call them *presentation operators*.

Algebraic Equivalences

- The operators presented in the last few slides make little assumptions on the internal organization of objects.
- As a consequence, we can define abstract equivalences.
- We present three examples of equivalences concerning the selection operator.

Algebraic Equivalences

- Some equivalences are *generalizations* of the relational ones.
- Ex: **Pushing selections down into joins**.

$$\sigma_{\mathrm{P}}(C_1 \bowtie_{\mathrm{P}'} C_2) = C_1 \bowtie_{\mathrm{P}'} \sigma_{\mathrm{P}}(C_2)$$
$$\text{if } \forall c' \in C_1, c'' \in C_2 \; (\mathrm{P}(c' \cup c'') = \mathrm{P}(c'')) \quad .$$

- P is a selection predicate (Ex: book/author='Shelley'), C_1, C_2 are collections.
- The condition specifies that P does not depend on the objects in C_1.

Algebraic Equivalences

- Some equivalences are *relational-like*, but applied to new operators.
- Ex: **Inversion of selection and embedding**.

$$\sigma_{\mathrm{P}}(\epsilon_{\mathrm{FE}}(C)) = \epsilon_{\mathrm{FE}}(\sigma_{\mathrm{P}'}(C))$$

$$\text{if } \forall c \in C\ (\mathrm{P}(\mathrm{FE}(c)) = \mathrm{P}'(c))\ .$$

- FE specifies how to construct new data.
- With XML data, it corresponds to XQuery node constructors.

Algebraic Equivalences

- Some equivalences are *specific* for the new operators:
- Ex: **Pushing selections down into splittings**.

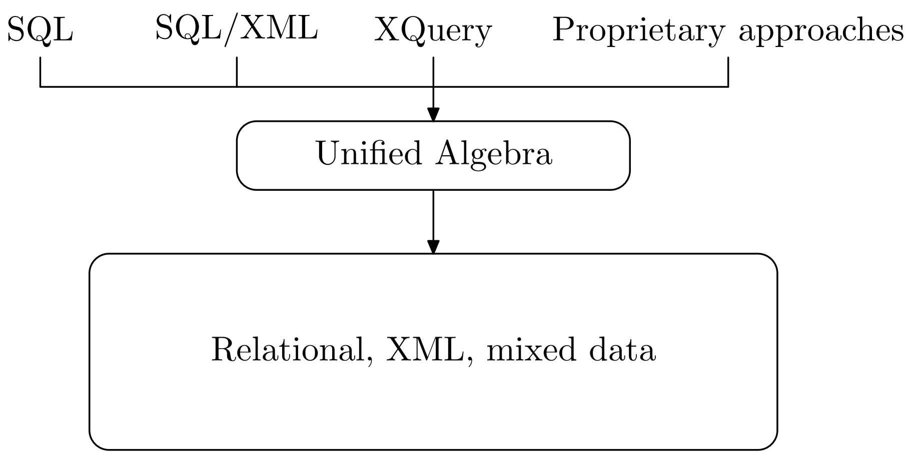
$$\sigma_{\mathrm{P}}(\varsigma(\mathrm{C})) = \varsigma(\pi_{\mathrm{FP}}(\mathrm{C}))$$
$$\text{if } \forall c \in \mathrm{C}\ (\mathrm{FP}(c) = \{e \mid e \in c \wedge \mathrm{P}(\{e\})\}) \ .$$

- FP is a projection function, specifying the part of the objects to be extracted.

A Concrete Example: XML/Relational Data Management

Objective

- Model for mixed XML/Relational data, as an instantiation of our abstract model.
- Homogeneous representation.
- The query algebra must support (significant subsets of) the main user languages.



Roadmap

- Parametric Algebra $\rightarrow$ Algebra for Mixed Data.
- To instantiate it, we must:
  - Define objects (the entities composing objects).
  - Define entity manipulation functions (we will use simple atoms).
  - Define application-specific presentation operators.

Entities: Data Trees

- An Object is a set of entities called Data Trees.
- A Collection is a set of Data Forests (Objects).
- A Data Forest can represent a Tuple with XML Trees.

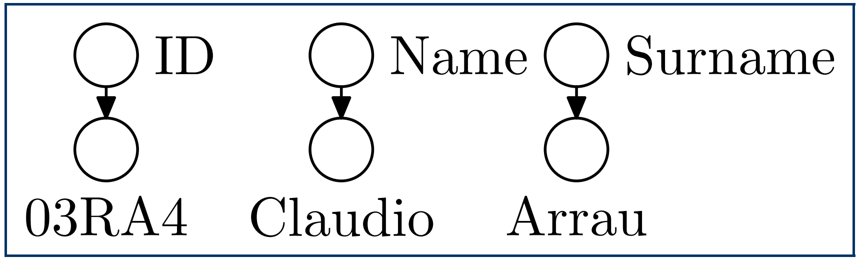
Entities: Data Trees

1. $V = \{v_1, \ldots, v_n\}$ is a finite set of vertices.
2. $E \subset \{(v_i, v_j) \mid v_i, v_j \in V\}$.
3. (V, E) is a directed tree.
4. $\preceq$ is a possibly empty partial order on V.
5. $\lambda : V \rightarrow (L \cup \{\text{null}\})$, where L is a set of labels.
6. $\tau : V \rightarrow T$, where T is a set of types.
7. $\forall v \in V \left(\delta(v) = \begin{cases} \oplus(c(v)) & \text{if outdegree(v) } \geq 1 \\ \delta'(v) & \text{o.w.} \end{cases} \right)$

   – $\oplus$ is a parametric concatenation operator, $c(v)$ is the set of v's children, and δ' is a content function defined on leaf nodes.

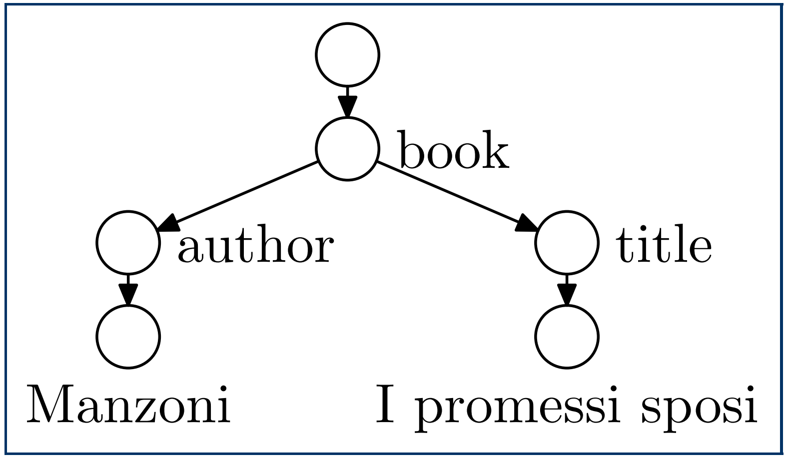
Objects (Examples)

- Relational Tuple:



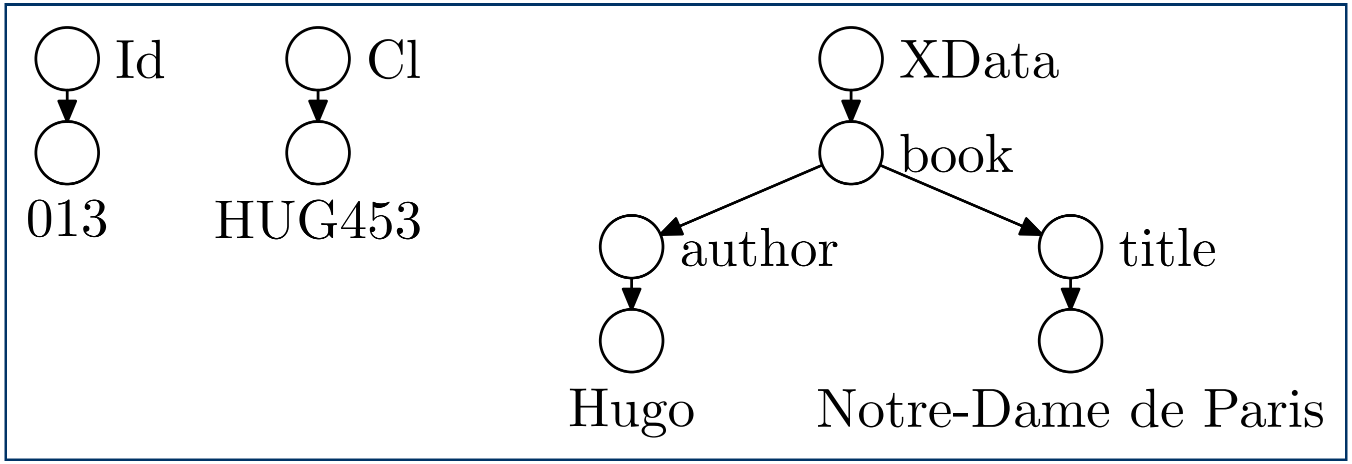
Objects (Examples)

- XML Tree:



Objects (Examples)

- Mixed Data:

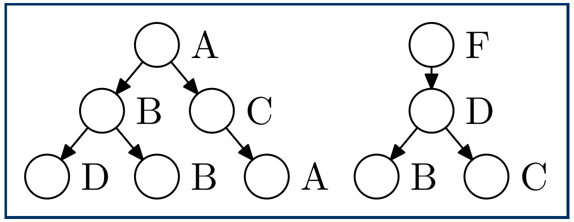


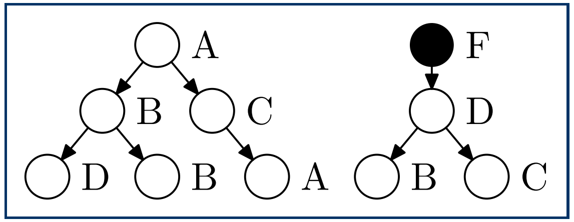
Predicates and Functions

- The parametric definition of our algebraic operators allows us to choose many languages to manipulate data forests.
- We need to define Object Manipulation Functions (SP, PF, EF, SF, NM).

P&F: Tree Selection Expressions

- Used to select the root nodes of data trees.
- $x \in \Sigma$ is a tree selection expression (TSE).
- (ϕ, ϕ) is a TSE if ϕ is a TSE.
- $\overline{\phi}$ is a TSE if ϕ is a TSE.

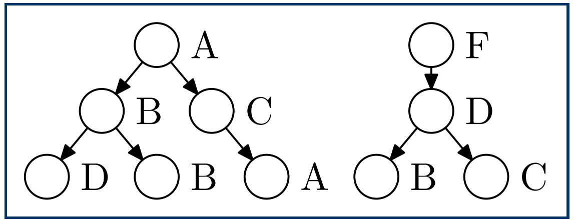
Input: 

F: 

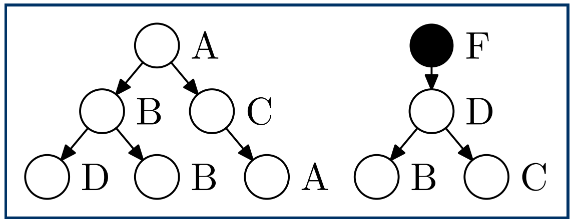
Trees with root F

P&F: Node Markers

Input:

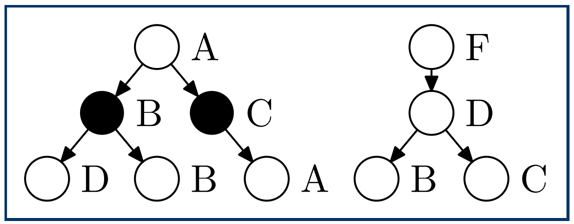


F:



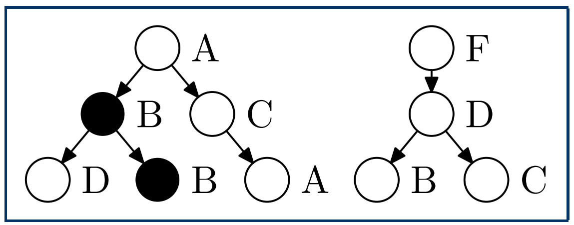
Trees with root F

$\overline{F}/*$:



Children where root is not F

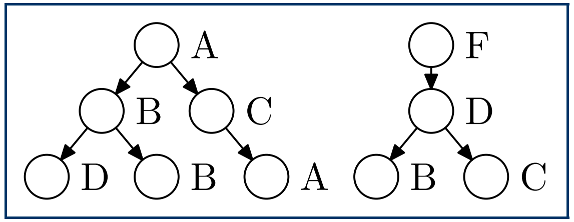
$A//B$:



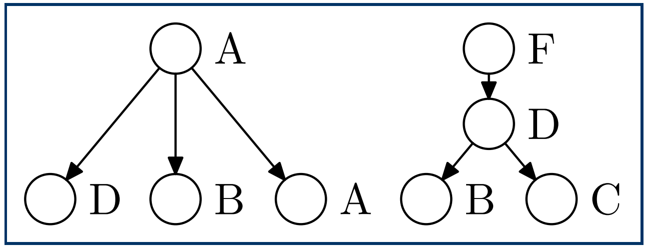
B descendants of A

P&F: Projection Functions

Input:

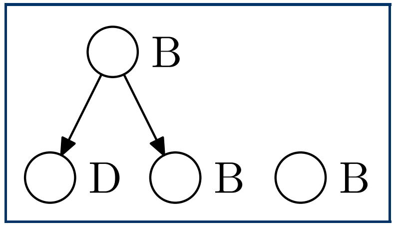


$-A/*$:



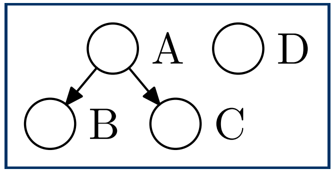
Deletes children of A

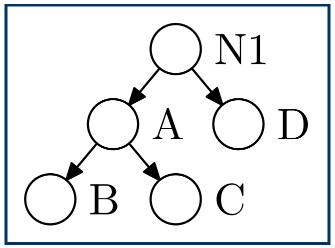
$+A//B$:



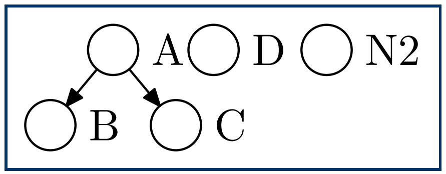
Extracts B descendants of A

P&F: Embedding Functions

Input: 

$<N1>\ (*)$: 

New node N1 with * (all) children

$<N2>\ ()$: 

New node N2 with no children

Presentation Operators

- To complete the algebra instantiated by the predicates and functions just defined, we still need presentation operators.
- Looking at the definition of data trees, we can identify two missing operators, used to change order and labels.
- Renaming (ρ), changes node and column names.
- Ordering (ω), changes the relative order of nodes.

SQL/XML

- Conservative extension of SQL, augmented with functions that build XML data [3].
- We focus on a subset of SQL/XML that includes the two basic functionalities of the language:
  - XMLELEMENT()
  - XMLAGG()

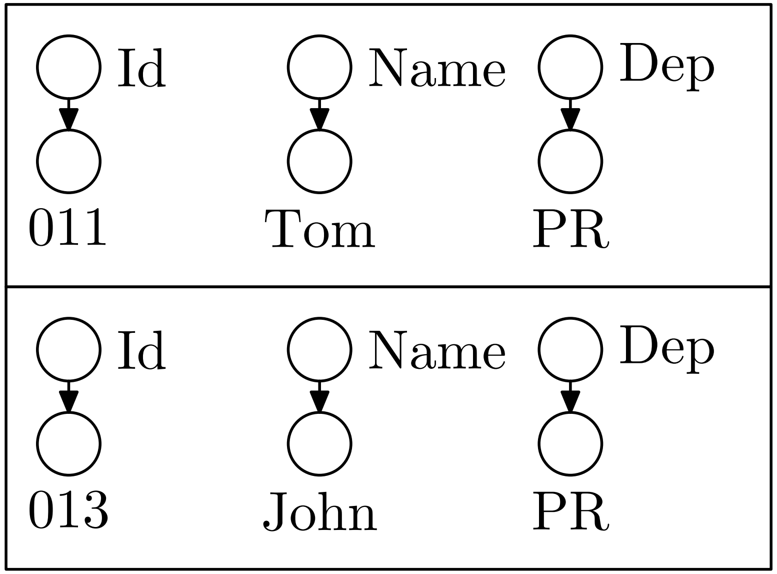
SQL/XML (Example)

```
SELECT XMLELEMENT(NAME "dep",
        XMLELEMENT(NAME "name",dep),
        XMLAGG(XMLELEMENT(NAME "emp",id))
        ) AS result
FROM EMP
GROUP BY dep
```

$$\rho_{\text{result} \leftarrow \$2}(\pi_{+\$2}(\pi_{-\$2/\text{dep}/*}(\epsilon_{<\$2>(<\text{dep}>(\$3,\$0))}($$
$$\pi_{-\$3/\text{name}/*}(\epsilon_{<\$3>(<\text{name}>(^{+}\text{dep}))}(\gamma_{\text{dep}}(\pi_{-\$0/*}(\epsilon_{<\$0>(\$1)}($$
$$\pi_{-\$1/\text{emp}/*}(\epsilon_{<\$1>(<\text{emp}>(^{+}\text{id}))}(\text{EMP})))))))))))$$

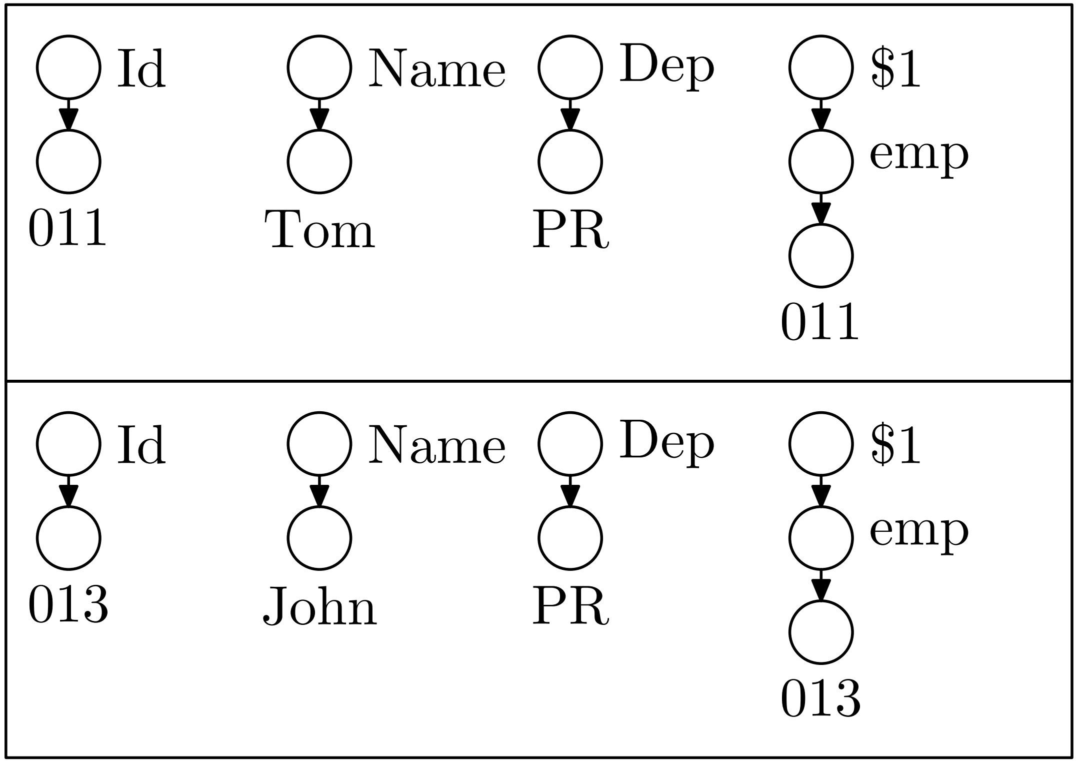
SQL/XML (Example)

EMP



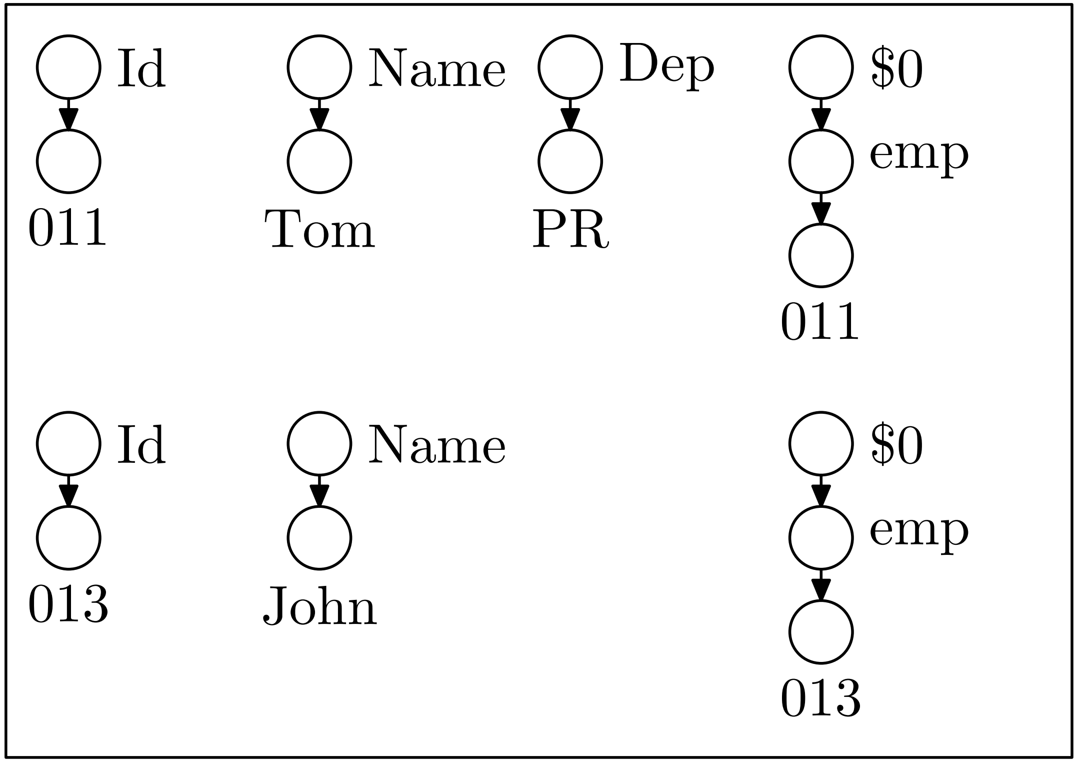
SQL/XML (Example)

$$\pi_{-\$1/\mathrm{emp}/*}(\epsilon_{<\$1>(<\mathrm{emp}>(^{+}\mathrm{id}))}(\cdot))$$



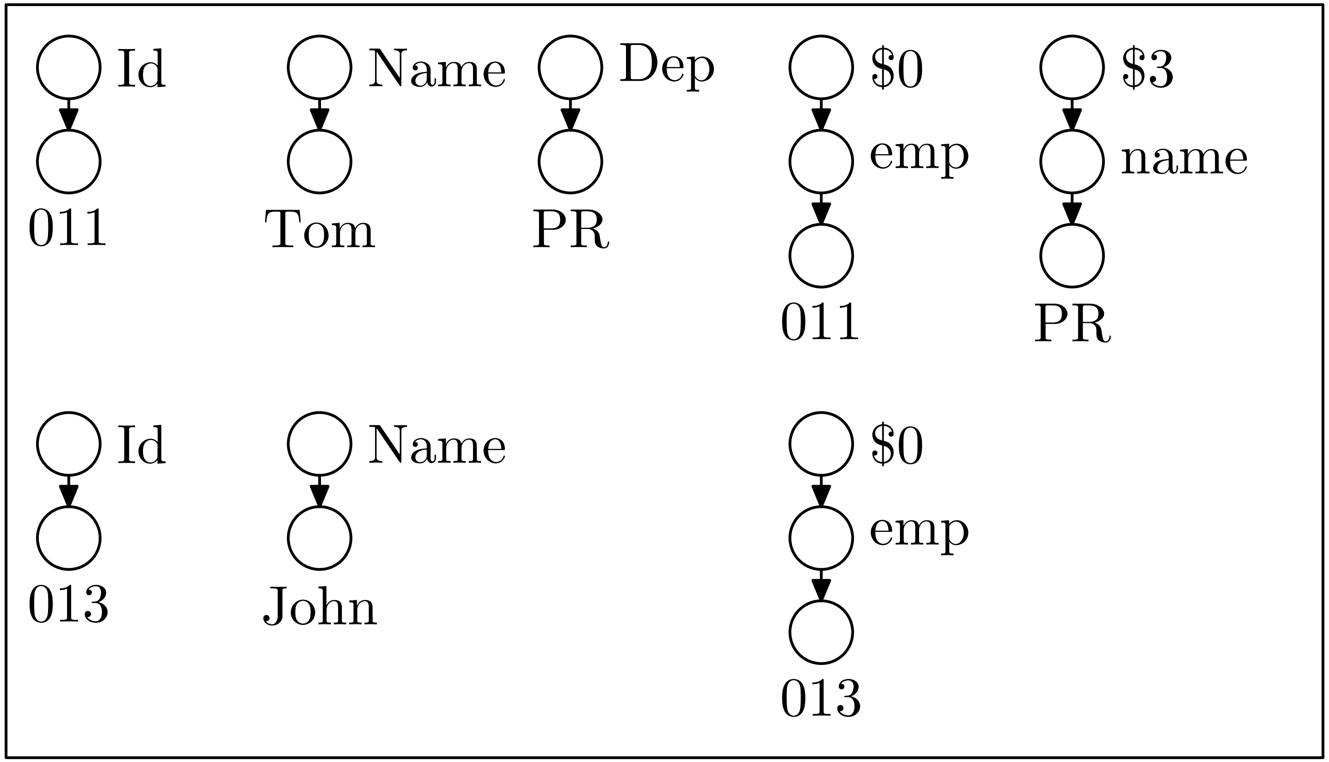
SQL/XML (Example)

$$\gamma_{\text{dep}}(\pi_{-\$0/*}(\epsilon_{<\$0>(\$1)}(.)))$$



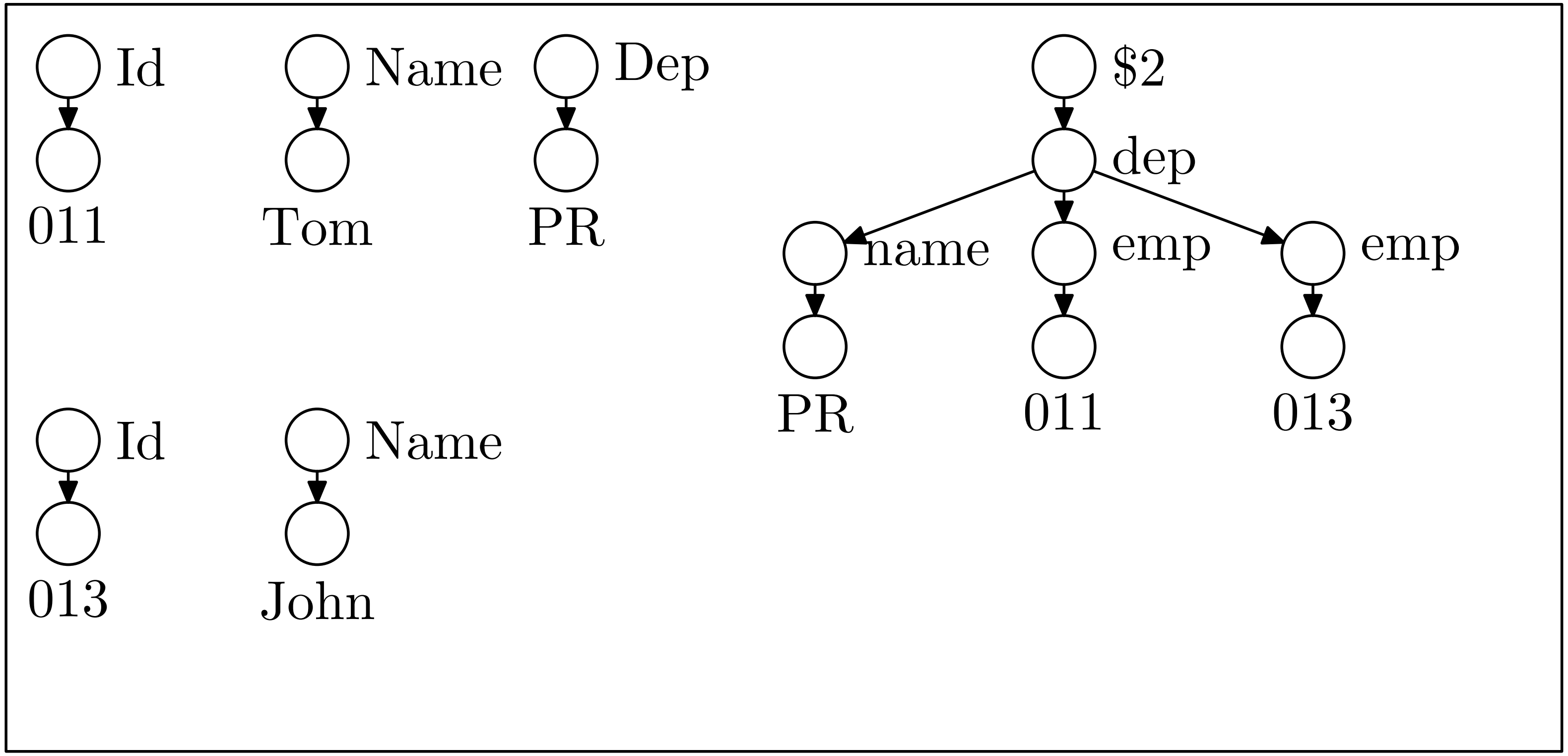
SQL/XML (Example)

$$\pi_{-\$3/name/*}(\epsilon_{<\$3>(<name>(^{+}dep))}(\cdot))$$



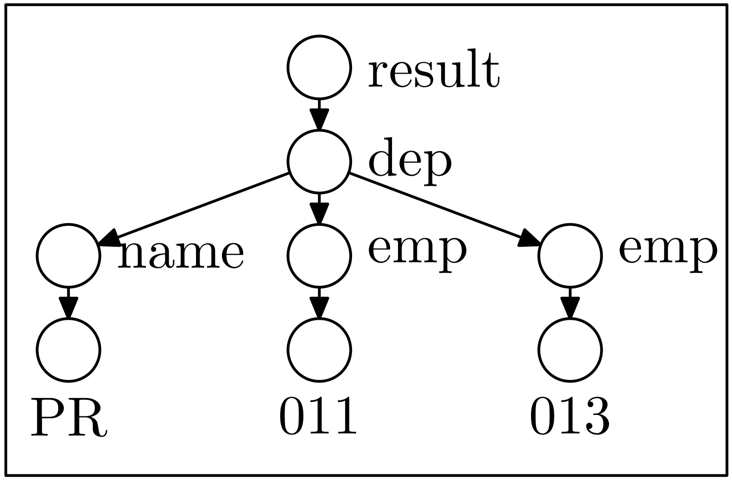
SQL/XML (Example)

$$\pi_{-\$2/\mathrm{dep}/*}(\epsilon_{<\$2>(<\mathrm{dep}>(\$3,\$0))}(\cdot))$$



SQL/XML (Example)

$$\rho_{\text{result}\leftarrow\$2}(\pi_{+\$2}(.))$$



Conclusion

- Data modeling: making it both generic and easily specializable is one of the main challenges.
- Parametric logical level that generalizes the main existing logical models.
- Application of the model to XML and Relational data.
- Some missing features:
  - Constraints.
  - Mapping to physical level.

Towards a unified model for heterogeneous data

Matteo Magnani

Dept. of Computer Science
University of Bologna
Via Mura A.Zamboni, 7
40127 Bologna
Italy

Danilo Montesi

Department of Mathematics and Informatics,
University of Camerino
Via Madonna delle Carceri, 9
Camerino (MC)
Italy

Additional Slides

- A step-by-step XQuery example.
- An example using Oracle XMLType.
- Other XQuery examples.
  - Input and Path Expressions.
  - FLWR Expressions.
  - Constructors.
  - A Complex Example.
- Grammars.
  - A Subset of SQL/XML.
  - A Subset of XQuery.

XQuery

- W3C proposal for an XML query language [1].
- We focus on a subset of XQuery that includes:
  - Simple path expressions.
  - For-Let-Where-Return expressions.
  - Constructors.
  - Arbitrarily nested sub-queries.

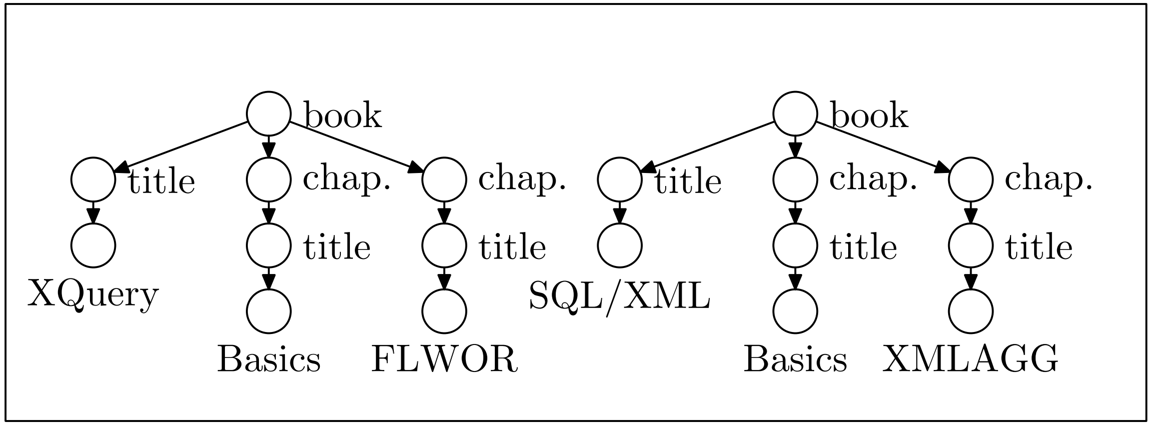
XQuery (Example)

```
for $b in collection('books')
let $c := $b//chapter
where $b/title = 'XQuery'
return $c/title
```

$$\gamma(\pi_{+\overline{\$c,\$b}}(\pi_{+\$c;\$b;\$c/*/title}(\sigma_{\$b/*/title='XQuery'}(\epsilon_{<\$c>(\overline{\$b})}(\pi_{+\$b;\$b/*//chapter}(\epsilon_{<\$b>(*)}(\varsigma(IN_{'books'}([\,])))))))))$$

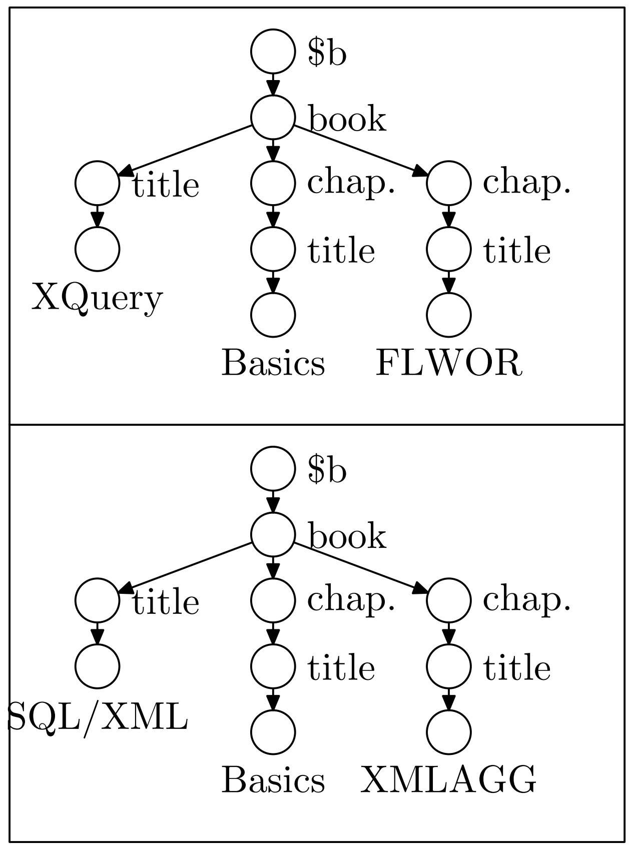
XQuery (Example)

$$\mathrm{IN}_{'\mathrm{books}'}([\,])$$



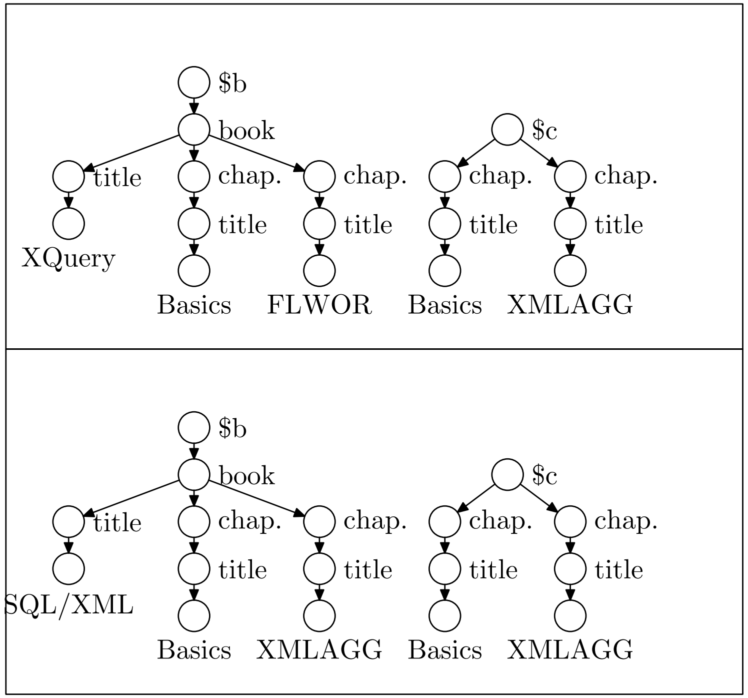
XQuery (Example)

$$\epsilon_{<\$b>(*)}(\varsigma(.))$$



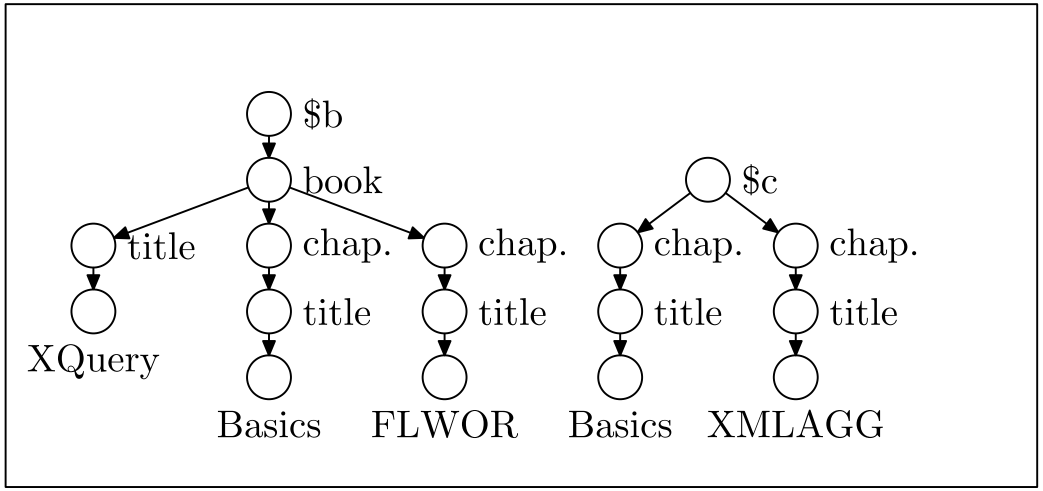
XQuery (Example)

$$\epsilon_{<\$c>(\overline{\$b})}(\pi_{+\$b;\$b/*//chapter}(\cdot))$$



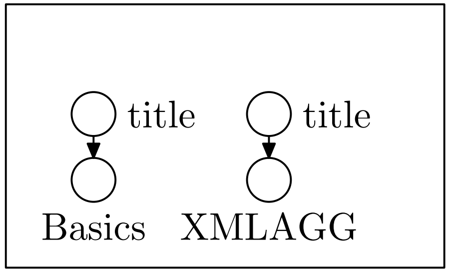
XQuery (Example)

$$\sigma_{\$b/*/title='XQuery'}(\cdot)$$



XQuery (Example)

$$\gamma(\pi_{+\overline{\$c,\$b}}(\pi_{+\$c;\$b;\$c/*/\text{title}}(\cdot)))$$



XMLType

Consider the following Oracle10*g* SQL statements:

create table DEPS_AND_EMPS(XDATA sys.XMLTYPE);

create table DEP_INFO(Name VARCHAR2(10), Info VARCHAR2(80));

XMLType

```
select D.XDATA.extract('/dep/name')
                .getClobVal()
from DEPS_AND_EMPS D
where D.XDATA.extract('/dep/id/text()')
                .getStringVal() = '0001';
```

$$\pi_{XDATA/dep/name}(\sigma_{XDATA/dep/id/*='0001'}(D))$$

XMLType

```
select D.XDATA, I.Info
from DEPS_AND_EMPS D, DEP_INFO I
where D.XDATA.extract('/dep/name/text()')
                    .getStringVal() = I.Name;
```

$$\pi_{\mathrm{XDATA,Info}}(\mathrm{D} \bowtie_{\mathrm{XDATA/dep/name/*=Name/*}} \mathrm{I})$$

Input and Path Expressions

doc('bib.xml')//author/surname

$$\pi_{+*//\mathrm{author/surname}}(\mathrm{IN}_{'\mathrm{bib.xml}'}([\,]))$$

FLWR Expressions

for \$b in collection('books')
let \$a := \$b//author
where \$b/title/* = 'Moby Dick'
return \$a

$$\gamma(\pi_{+\overline{\$a,\$b}}(\pi_{+\$a;\$b;\$a/*}(\sigma_{\$b/*/title/*='Moby\ Dick'}($$
$$\epsilon_{<\$a>(\overline{\$b})}(\pi_{+\$b;\$b/*//author}($$
$$\epsilon_{<\$b>(*)}(\varsigma(IN_{'books'}([\,])))))))))$$

Constructors

<greetings>
Hello <planet>World</planet>
</greetings>

$\epsilon^{-}_{<\text{greetings}>(\text{'Hello '},\$0)}(\epsilon_{<\$0>(*)}(\epsilon_{<\text{planet}>(\text{'World'})}([\,])))$

A More Complex Example

```
for $a in collection('Authors')
return
<author>
$a/name
{for $b in collection('Books')
where $b/author/* = $a/id/*
return $b/title}
{for $j in collection('Journals')
where $j/author/* = $a/id/*
return $j/title}
</author>
```

A More Complex Example

$$\gamma(\pi_{+\overline{\$a}}(\epsilon^{-}_{<author>('\hookleftarrow \quad \$a/name\hookleftarrow \quad ',\$0,'\hookleftarrow \quad ',\$1,'\hookleftarrow')}(\epsilon_{<\$1>(\overline{\$a,\$}$$

$$\bowtie \; ($$

$$\gamma_{\$a,\$0}(\pi_{+\overline{\$j}}(\pi_{+\$j;\$0;\$a;\$j/*/title}(\sigma_{\$j/*/author/*=\$a/*/id/*}(\epsilon_{<\$j>}$$

$$\varsigma_{\$a,\$0}(IN_{'Journals'}(\epsilon_{<\$0>(\overline{\$a})}((B2) =\bowtie$$

$$(\gamma_{\$a}(\pi_{+\overline{\$b}}(\pi_{+\$b;\$a;\$b/*/title}(\sigma_{\$b/*/author/*=\$a/*/id/*}($$

$$\epsilon_{<\$b>(\overline{\$a})}(\varsigma_{\$a}(IN_{'Books'}(\epsilon_{<\$a>(*)}(\varsigma(IN_{'Authors'}([\,]))))))))))$$

Grammar of a SQL/XML Subset I

```
CompilationUnit ::= SelectStatement ";"
TableColumn ::= Name ( "." Name )?
Name ::= ( <S_IDENTIFIER> | <S_QUOTED_ID> )
TableReference ::= Name
SelectStatement ::= "SELECT" SelectList
  FromClause ( WhereClause )? ( GroupByClause )
SelectList ::= ("*" | SelectItem
               ("," SelectItem )* )
SelectItem ::= (SQLPrimaryExpression |
                XMLElement|XMLAgg)(As)?
As ::= "AS" <S_IDENTIFIER>
FromClause ::= "FROM" TableReference
               ( "," TableReference )*
WhereClause ::= "WHERE" SQLExpression
```

Grammar of a SQL/XML Subset II

```
GroupByClause ::= "GROUP" "BY" TableColumn
                  ( "," TableColumn )*
SQLExpression ::= SQLComparisonExpr
          ( "AND" SQLComparisonExpr )*
SQLComparisonExpr::= SQLPrimaryExpr
                     <Relop> SQLPrimaryExpr
SQLPrimaryExpr ::= (TableColumn | <S_NUMBER> |
                    <S_CHAR_LITERAL>)
XMLElement ::= "XMLELEMENT(NAME" <S_QUOTED_ID>
  "," ElementContent ("," ElementContent)* ")"
ElementContent ::= (Name | XMLElement | XMLAgg
                    | <S_CHAR_LITERAL>)
XMLAgg ::= "XMLAGG(" XMLElement ( ","
                     XMLElement )* ")"
```

Grammar of an XQuery Subset I

```
Expr ::= InputExpr | FLWORExpr | Literal
         | Constructor
InputExpr ::= (InputFunctionCall | VarRef)
              (PathExpr)?
InputFunctionCall ::= ("doc(" | "collection(")
                      <StringLiteral> ")"
VarRef ::= <VarName>
PathExpr ::= ( ChildStep | DescendantStep )
             ( ChildStep | DescendantStep )*
ChildStep ::= "/" NameTest
DescendantStep ::= "//" NameTest
NameTest ::= <QName> | "*"
FLWRExpr ::=  (ForClause | LetClause)+
              (WhereClause)? "return" Expr
```

Grammar of an XQuery Subset II

```
ForClause ::= "for" <VarName> "in" Expr
LetClause ::= "let" <VarName> ":=" Expr
WhereClause ::= "where" Expr <CompOp> Expr
Constructor ::= "<" <TagQName> ("/>" |
                 (">" ElementContent*
                  "<" <TagQName> ">"))
ElementContent ::= <ElementContentChar> |
                   Constructor | EnclosedExpr
Literal ::= NumericLiteral | <StringLiteral>
NumericLiteral ::= <IntegerLiteral> |
                   <DecimalLiteral> |
                   <DoubleLiteral>
EnclosedExpr ::= "{" Expr "}"
```

References

[1] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language (working draft, nov 12, 2003). Technical report, W3C, 2003. http://www.w3.org/TR/xquery/.

[2] Jim Gemmel, Gordon Bell, Roger Lueder, Steven Drucker, and Curtis Wong. MyLifeBits: Fulfilling the memex vision. In *Multimedia'02*, pages 235–238, Juan-les-Pins, France, December 2002. ACM.

[3] Jim Melton. SQL - part 14: SQL/XML. Technical report, ISO/ANSI, 2003.