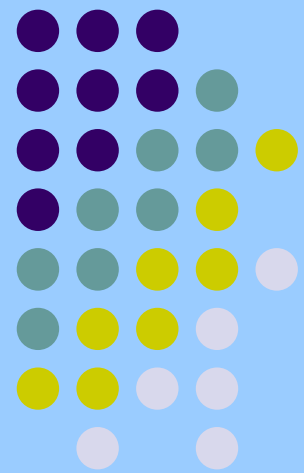
Data Modeling using XML

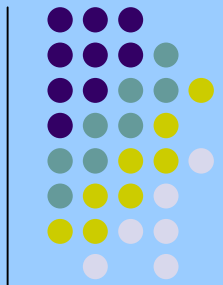
Murali Mani, WPI

Antonio Badia, University of Louisville

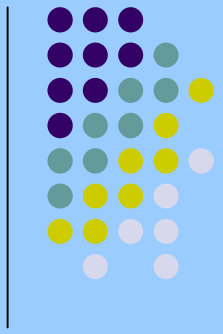
Oct 13, 2003



Outline



- **Part I:** How to come up with good XML designs for real world database applications?
- **Part II:** Translation between Relational and XML models.

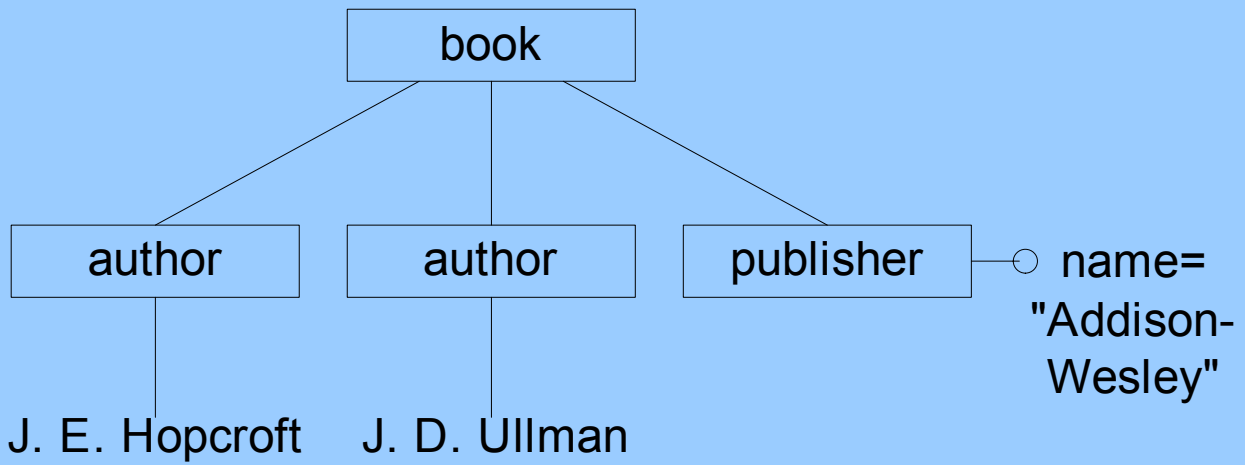


Part I:

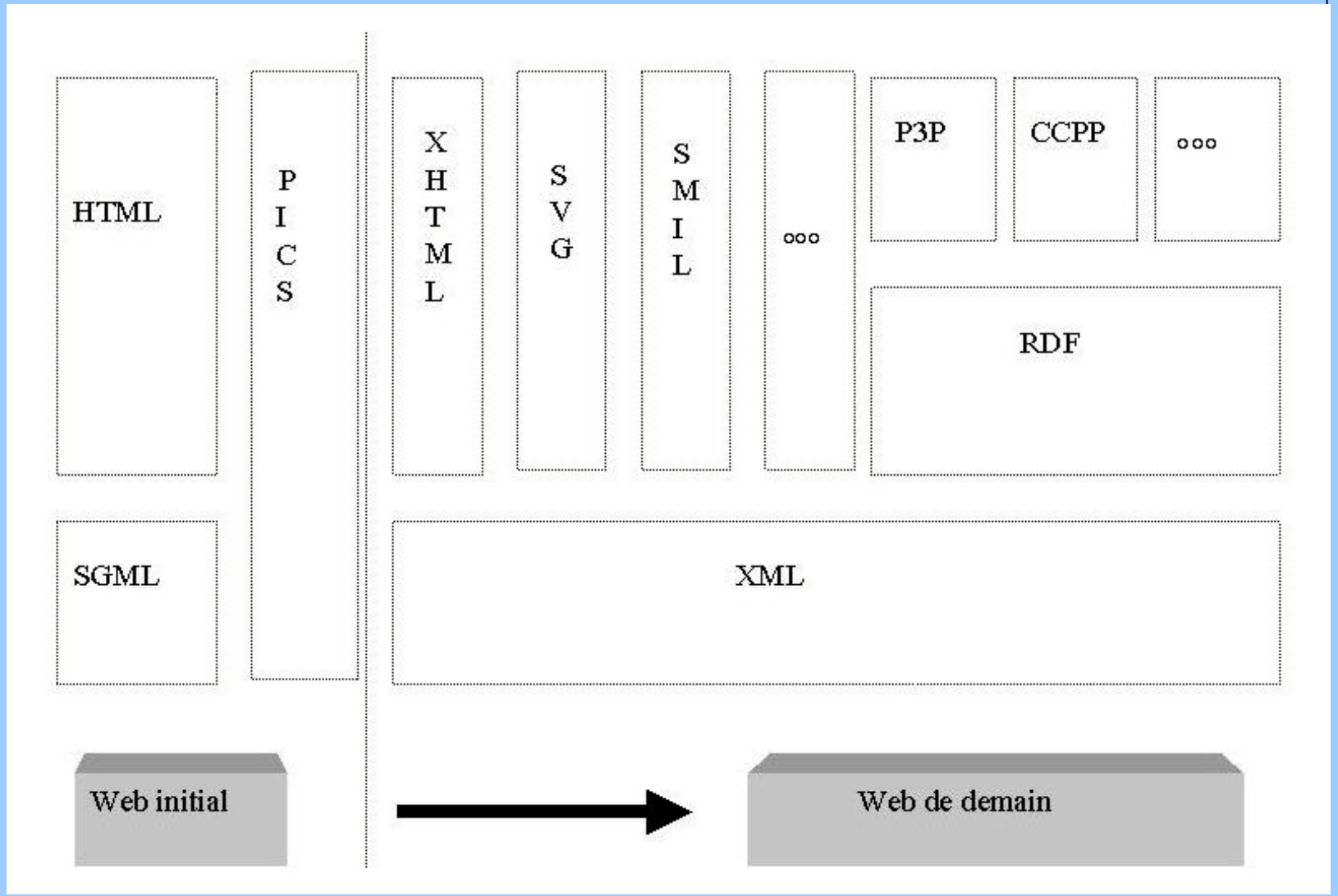
How to come up with good XML designs for real world database applications?

What is XML?

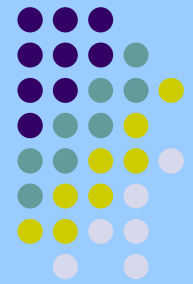
```
<book>
    <author>J. E. Hopcroft</author>
    <author>J. D. Ullman</author>
    <publisher name="Addison-Wesley"/>
</book>
```

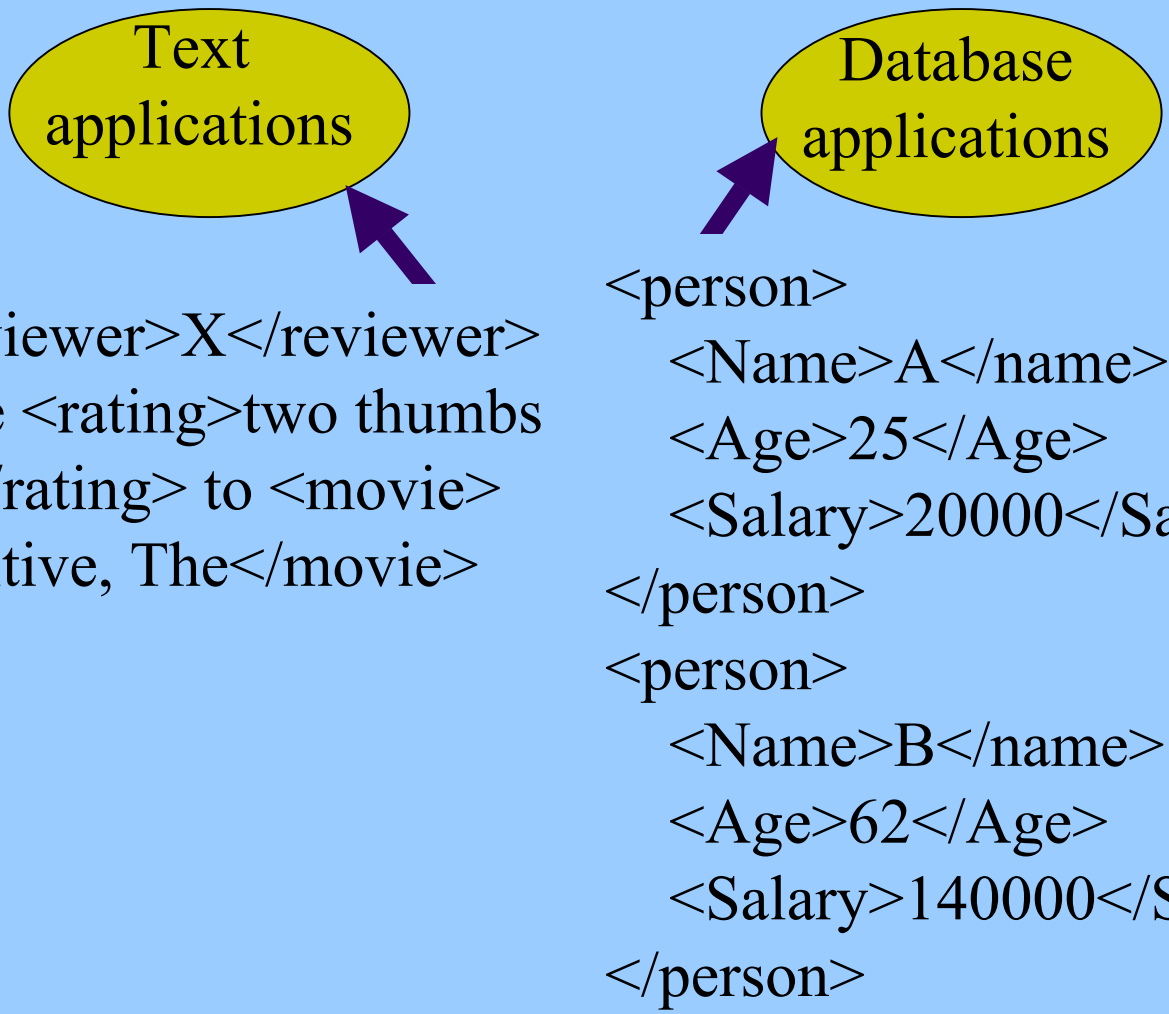


XML for information exchange



XML Publishing





Publishing Relational Databases

Name	Age	Salary
A	25	20000
B	62	140000

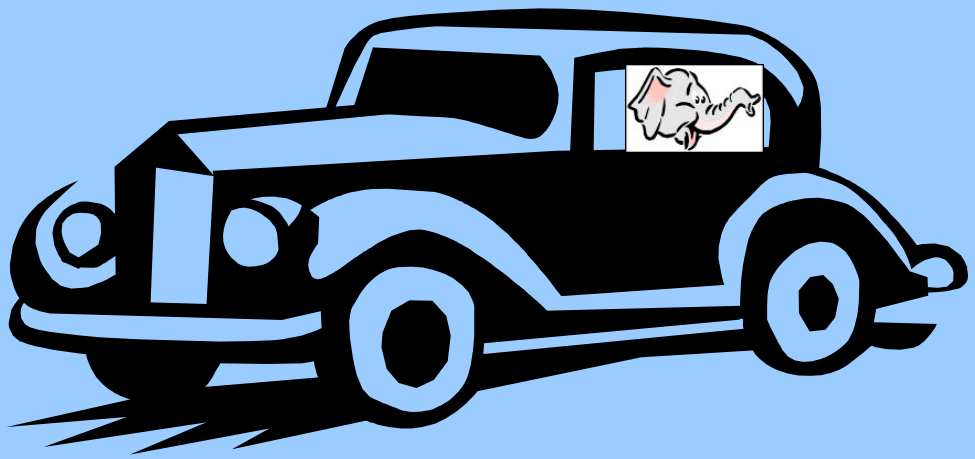
```
<person>
   <Name>A</name>
   <Age>25</Age>
   …
</person>
```

- Users/Applications see a uniform XML view
- Exchange data with other applications
- Querying XML is easier?

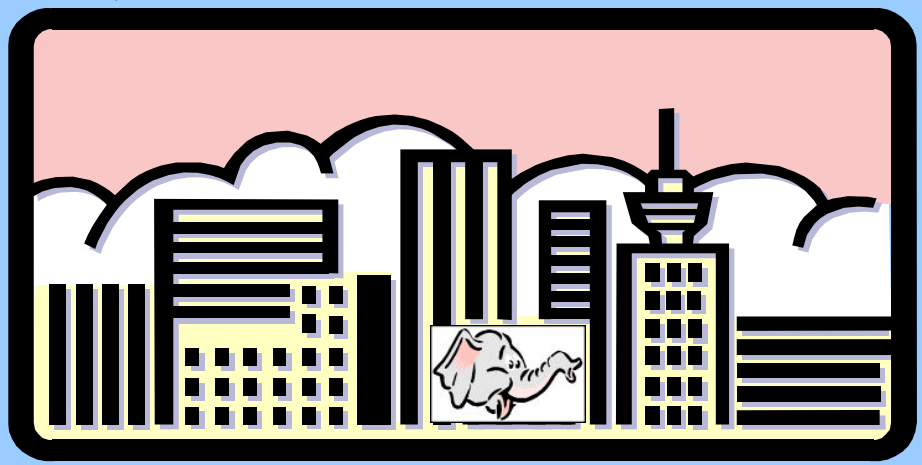
Problem:

What is a good XML schema for a relational schema?

XML for Data Modeling



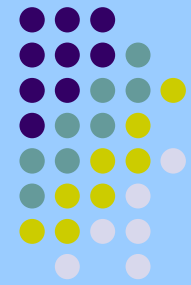
Location → location
(@val, @time, GPS)
GPS → gps (@satellite)



Location → location
(@val, @time, Bstation*)
Bstation → bstation
(@id, @sigStrength)

Location → location (@val, @time, (GPS | Bstation*))

XML as a logical data model



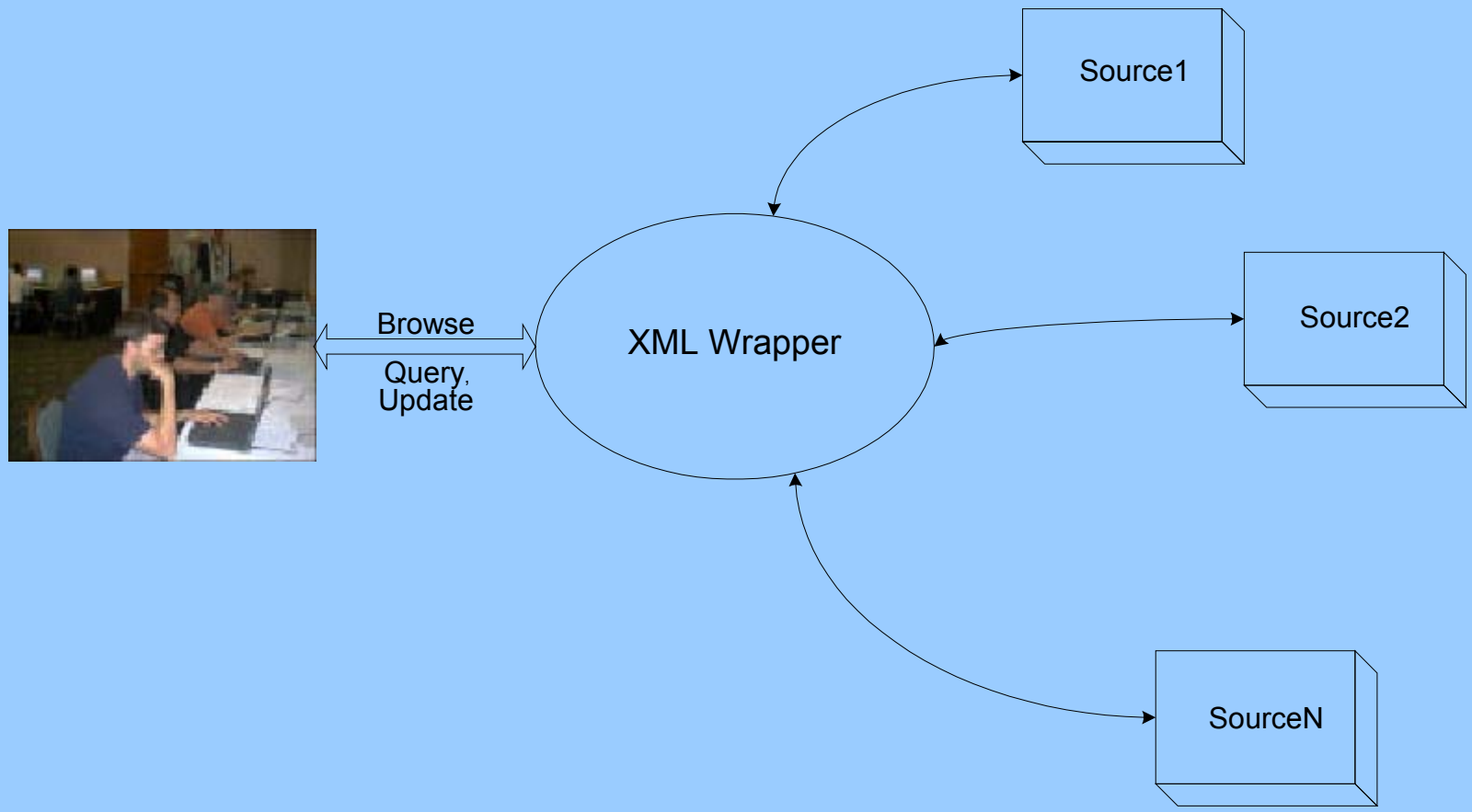
Location → location (@val, @time, (GPS | Bstation*))

- Use data modeling features provided by XML
  - Union types
  - Recursive types
  - Ordered relationships
- Easier to Query?

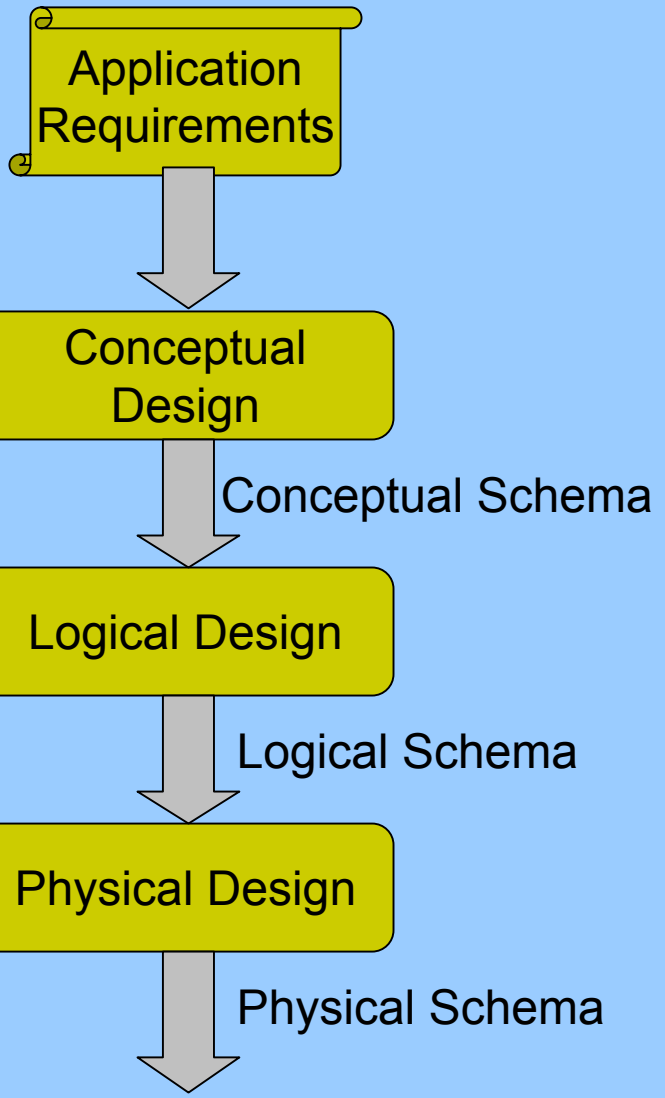
Problems:

- What is a good XML schema for an application?
- How do we store the data in relational databases?

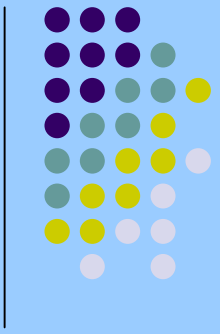
XML for data integration



Database Design Stages



Logical Data Model and Redundancy



Bad Design

Student_Professor

sname	BS	advisor	age
SD	CS	MXM	40
YC	EE	MXM	40

Professor

pname	age
MXM	40

Student

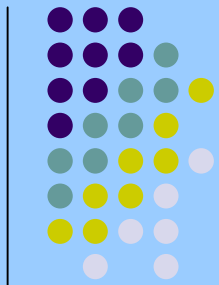
sname	BS	advisor
SD	CS	MXM
YC	EE	MXM

Good Design

Person

name	address	city	state	zip
AV	A1	Los An	CA	90034
AN	A2	Los An	CA	90034

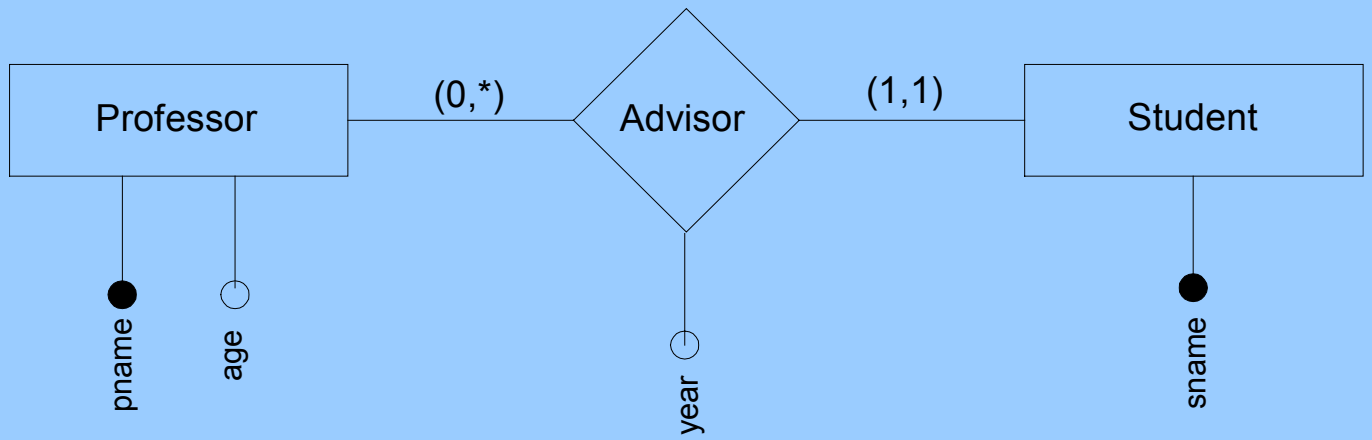
What is a Data Model?



- Structural Specification
- Constraint Specification
- Operations

Entity Relationship (ER) Model

- Structures: Entity Types, Relationship Types
- Constraints: Cardinality constraints



Relational Model

- Structures: Relations
- Constraints: Key, Foreign Key

Professor	
pname	age

Student	
sname	advisor

Key Constraints:

Key (Professor) = <pname>

Key (Student) = <sname>

Foreign Key Constraints:

Student (advisor) references Professor (pname)

Specifying Structures for XML

G = (N, T, P, S)

N = {Book, Author, Publisher, #PCDATA}

T = {book, author, publisher, pcdata}

S = {Book}

Book → book (Author +, Publisher)

Author → author (#PCDATA)

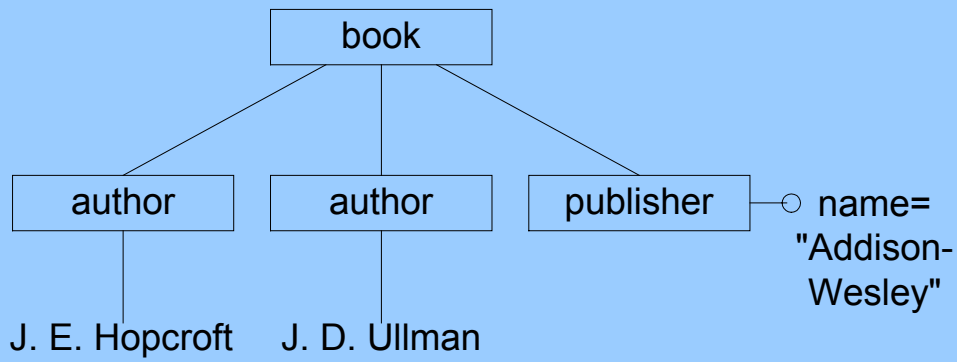
Publisher → publisher (@name::String)

#PCDATA → pcdata (ε)

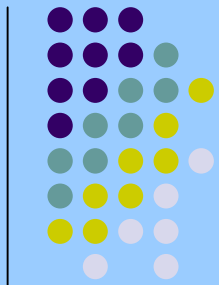
Regular Tree Grammar

Every production rule is of the form A → a X

A ∈ N, a ∈ T, X is a regular expression over N

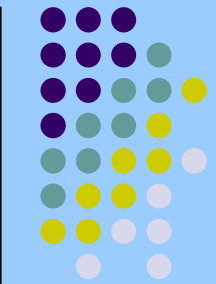


XML Schema Language Proposals



- W3C DTD: local tree grammar
- W3C XML Schema: single type tree grammar
- ISO/OASIS RELAX NG: full-fledged regular tree grammar

Properties of different Regular Tree Grammar classes



- Expressiveness
  - Regular tree grammar ***strictly more expressive*** than single type tree grammar
  - Single type tree grammar ***strictly more expressive*** than local tree grammar
- Closure properties
  - Regular tree grammar closed under union, intersection and difference
  - Single type tree grammar/local tree grammar closed only under intersection
- Type assignment
  - Type assignment can be ambiguous for regular tree grammar.
  - Type assignment is unambiguous for local tree grammar/single type tree grammar.

Ambiguous Type Assignment

G = (N, T, P, S)

N = {Book, Author1, Author2, Publisher, #PCDATA}

T = {book, author, publisher, pcdata}

S = {Book}

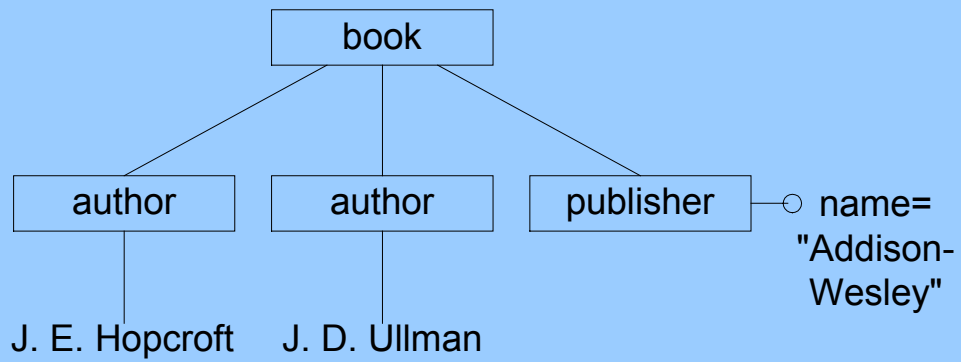
Book $\rightarrow$ book (Author1*, Author2*, Publisher)

Author1 $\rightarrow$ author (#PCDATA)

Author2 $\rightarrow$ author (#PCDATA)

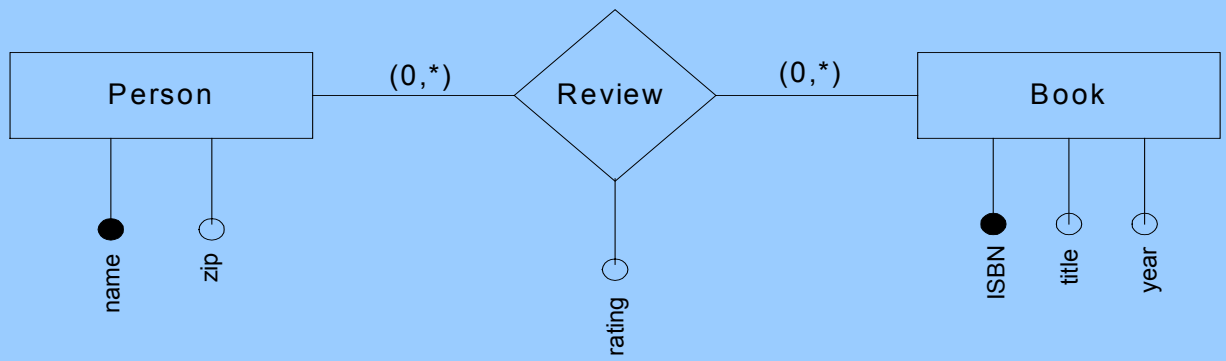
Publisher $\rightarrow$ publisher (@name::String)

#PCDATA $\rightarrow$ pcdata (ε)



Constraint Specification for XML – why?

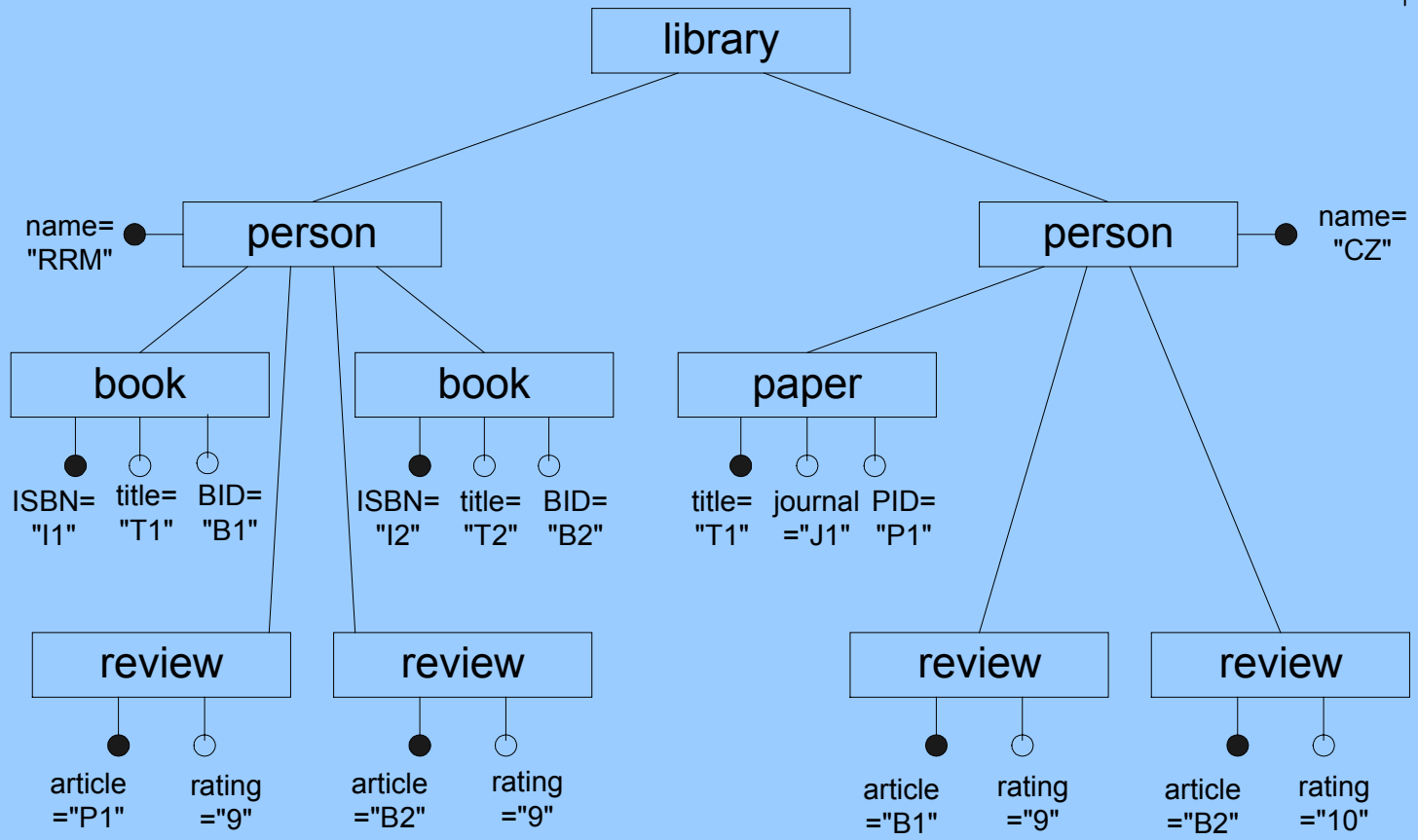
If we represent all relationships only by hierarchies, then the logical model will have redundancy.



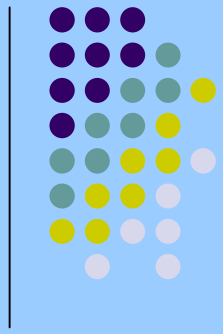
What constraint specification?

- Key, Foreign Key
- ID/IDREF

Specifying Constraints for XML: Example

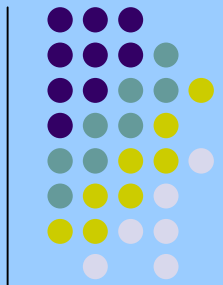


Specifying Constraints for XML



- Keys are specified using (rel, sel, field)
  - rel is relative axis
  - sel is selector axis
  - field is a set of path expressions
- For any element that “belongs to” rel, “sel” will give a set of elements. For this set of elements, field is the key.
- rel and sel can be types or path expressions
- Foreign keys are specified as (rel_1, sel_1, $field_1$) references (rel_2, sel_2, $field_2$)

Constraint Specification Proposals

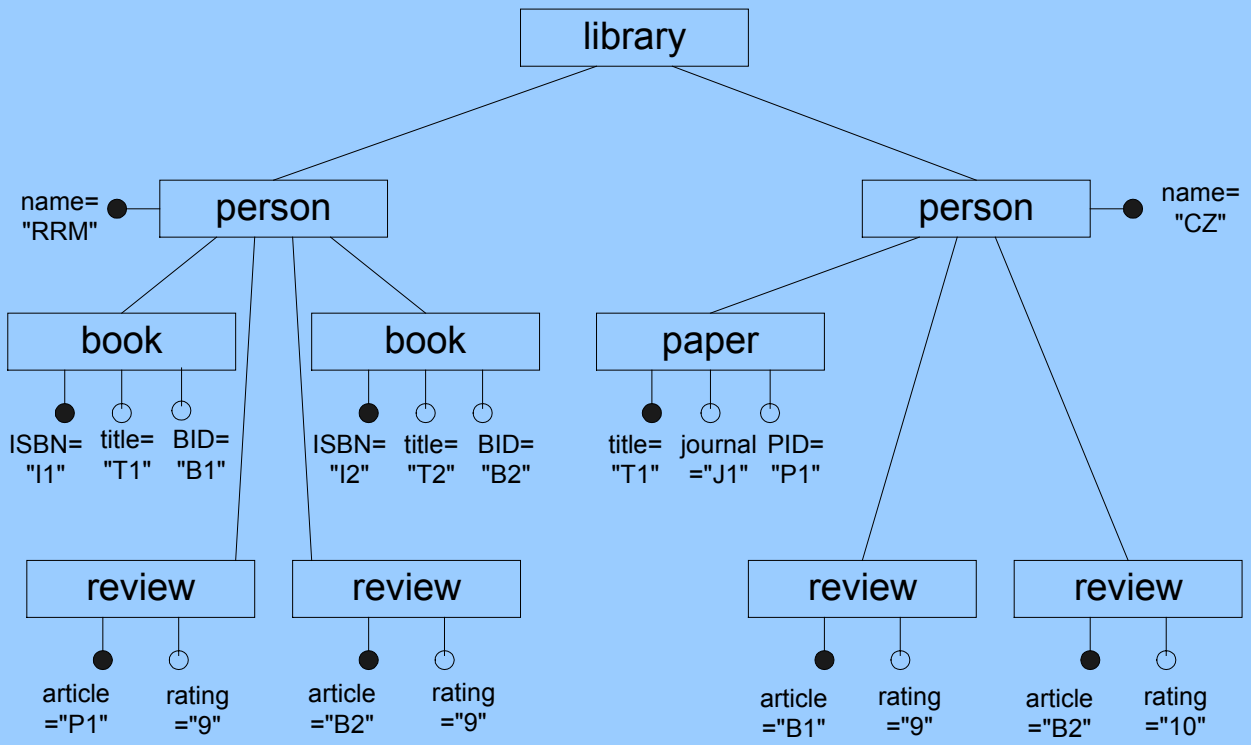


- W3C XML Schema
  - Relative axis = type
  - Selector axis = path expression
- Keys for XML – WWW10
  - Relative axis = path expression
  - Selector axis = path expression
- UCM – WWW10
  - No relative axis
  - Selector axis = type

Our proposal

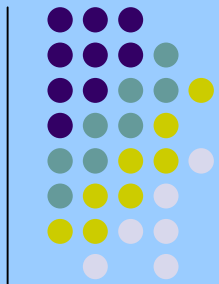
- Relative axis = type
- Selector axis = type
- IDREF and IDREFS identify target types

Example



(Library, Person, <@name>)
(Library, Book, <@ISBN>)
(Library, Paper, <@title>)
(Person, Review, <@article>)

@article::IDREF **references (Book | Paper)**



Why use XML as logical data model?

Path Expressions vs Joins

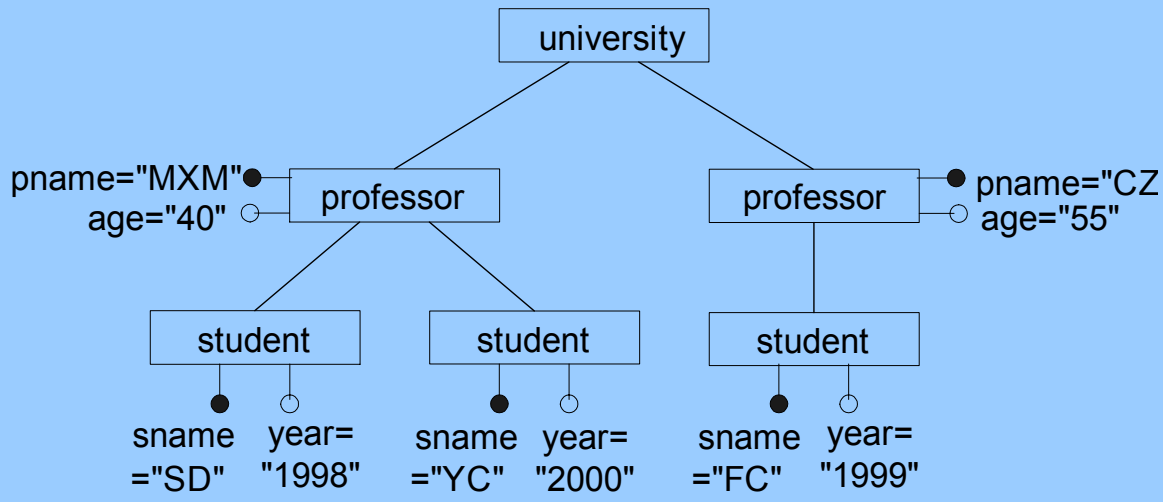
Professor

pname	age
MXM	40
CZ	55

Student

sname	year	advisor
SD	1998	MXM
YC	2000	MXM
FC	1999	CZ

Student (advisor) references Professor (pname)



Query: Give names of students of professors of age 40

$\pi_{name}((\sigma_{age=40}(Professor)) \otimes Student)$

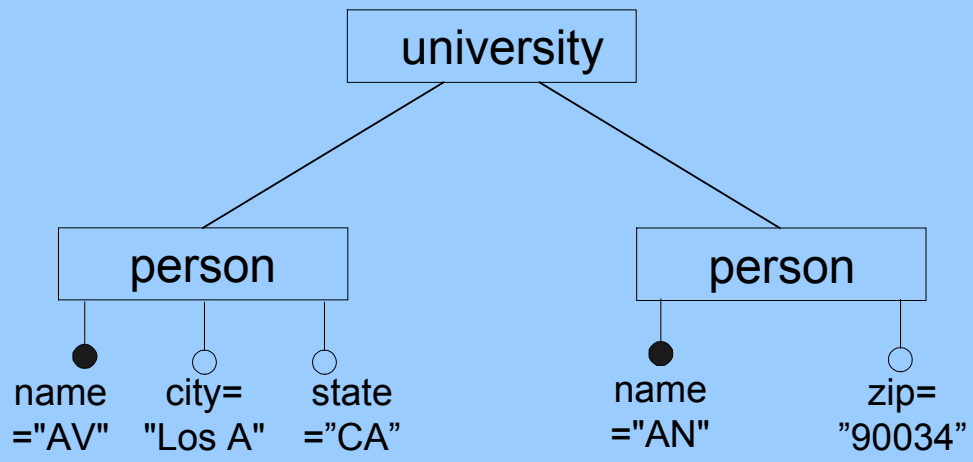
professor [@age=40]/student/@sname

Union Types - attributes

Person

name	city	state	zip
AV	Los A	CA	null
AN	null	null	90034

Person → person (@name, ((@city, @state) | @zip))



Union Types - Relationships

Person

name	zip
RRM	01609
CZ	90095

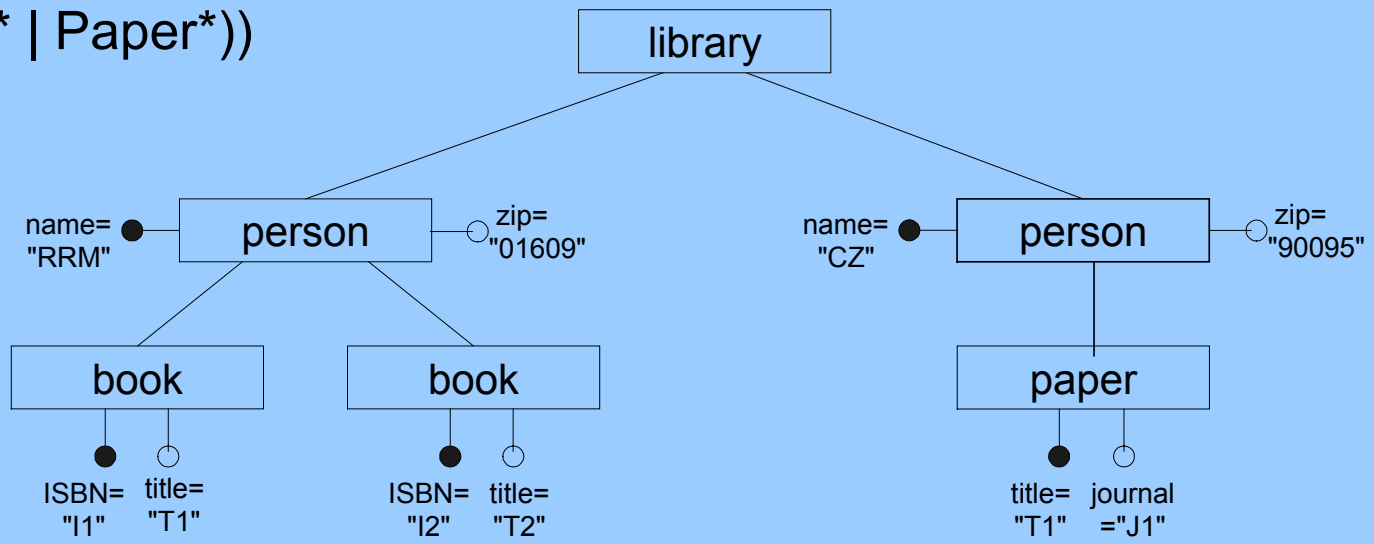
Book

ISBN	title	author
I1	T1	RRM
I2	T2	RRM

Paper

title	journal	author
T1	J1	CZ

Person → person (@name, @zip, (Book* | Paper*))



Union Types - Relationships

Conference

name	venue
ER	Chicago

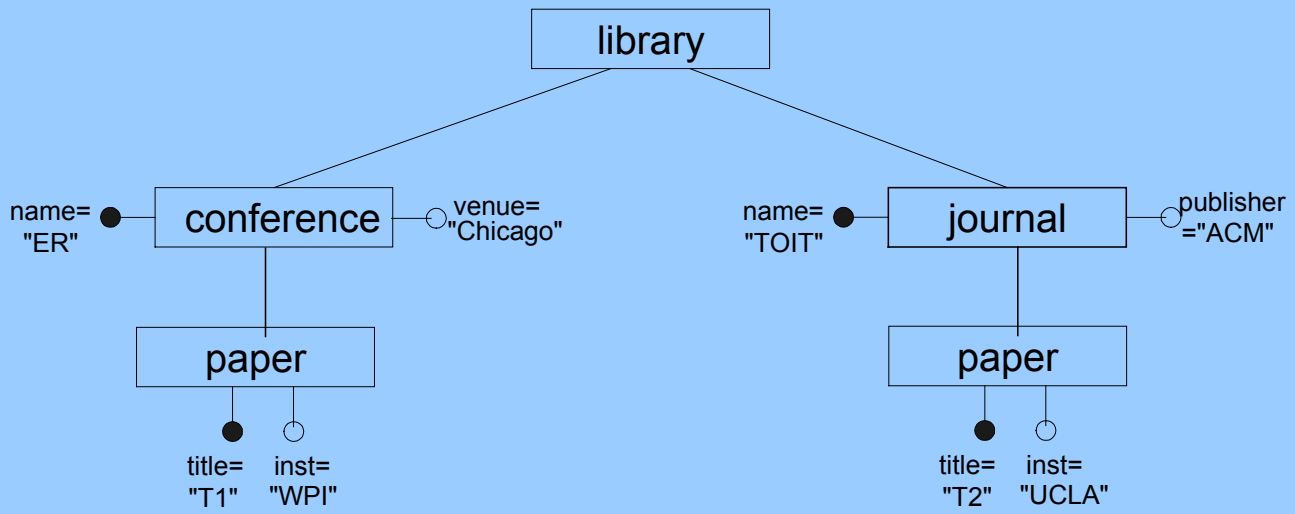
Journal

name	publisher
TOIT	ACM

Paper

title	inst	conf	journal
T1	WPI	ER	null
T2	UCLA	null	TOIT

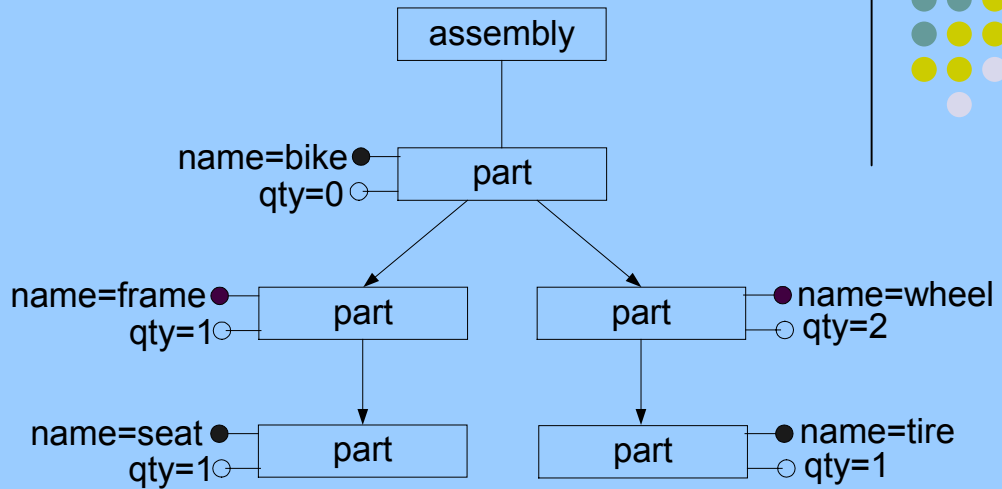
Conference → conference (@name, @venue, Paper*)
Journal → journal (@name, @publisher, Paper*)



Recursive Types

Assembly

name	superPart	qty
seat	frame	1
tire	wheel	1
frame	bike	1
wheel	bike	2
bike	null	0



Query: What are subparts of bike?

```
WITH RECURSIVE SubPart (name) AS
            (SELECT name FROM Assembly
             WHERE superPart=bike)
      UNION
            (SELECT R2.name
             FROM SubPart R1, Assembly R2
            WHERE R2.superPart = R1.name)
SELECT * FROM SubPart
```

part[@name=bike]//part/@name

IDREF vs Foreign Keys

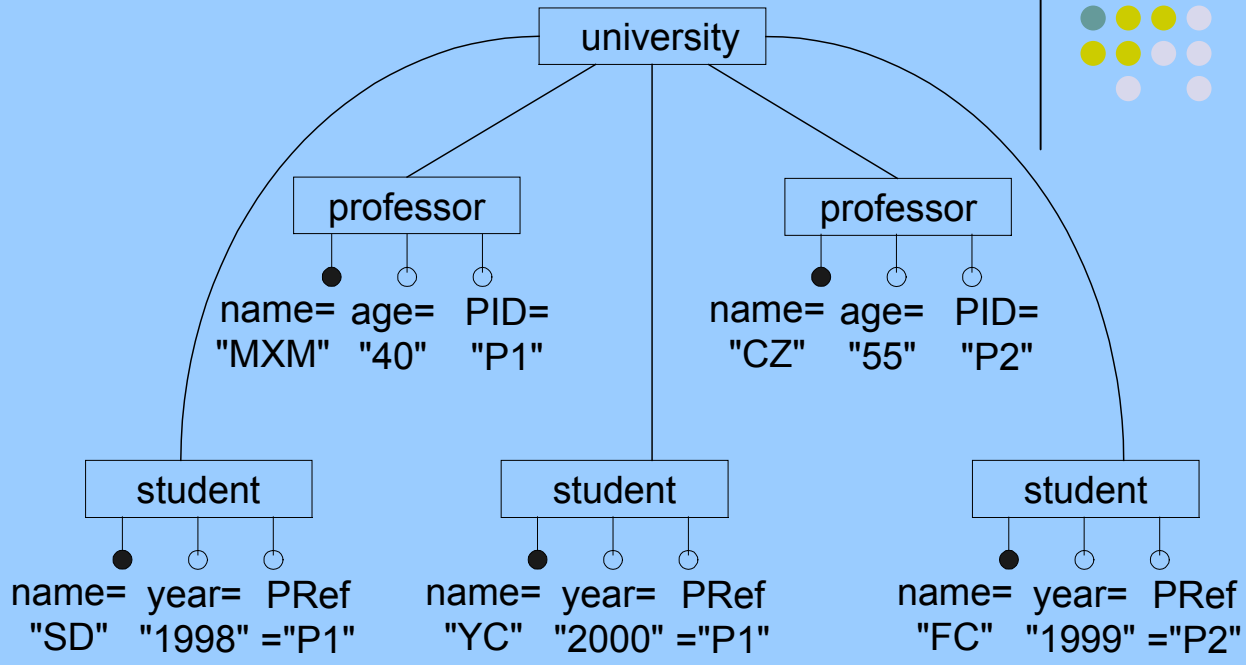
Professor

name	age
MXM	40
CZ	55

Student

name	year	advisor
SD	1998	MXM
YC	2000	MXM
FC	1999	CZ

Student (advisor) references Professor (name)

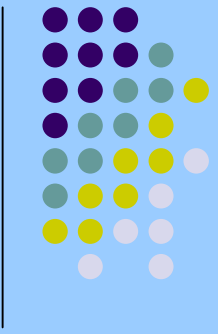


@PRef::IDREF references (Professor)

Query: Give names of students of professors of age 40

student[@PRef⇒professor/@age=40]/@name

IDREF as union of foreign keys



Book

ISBN	title
I1	T1
I2	T2

Person

name	zip
RRM	01609
CZ	90095

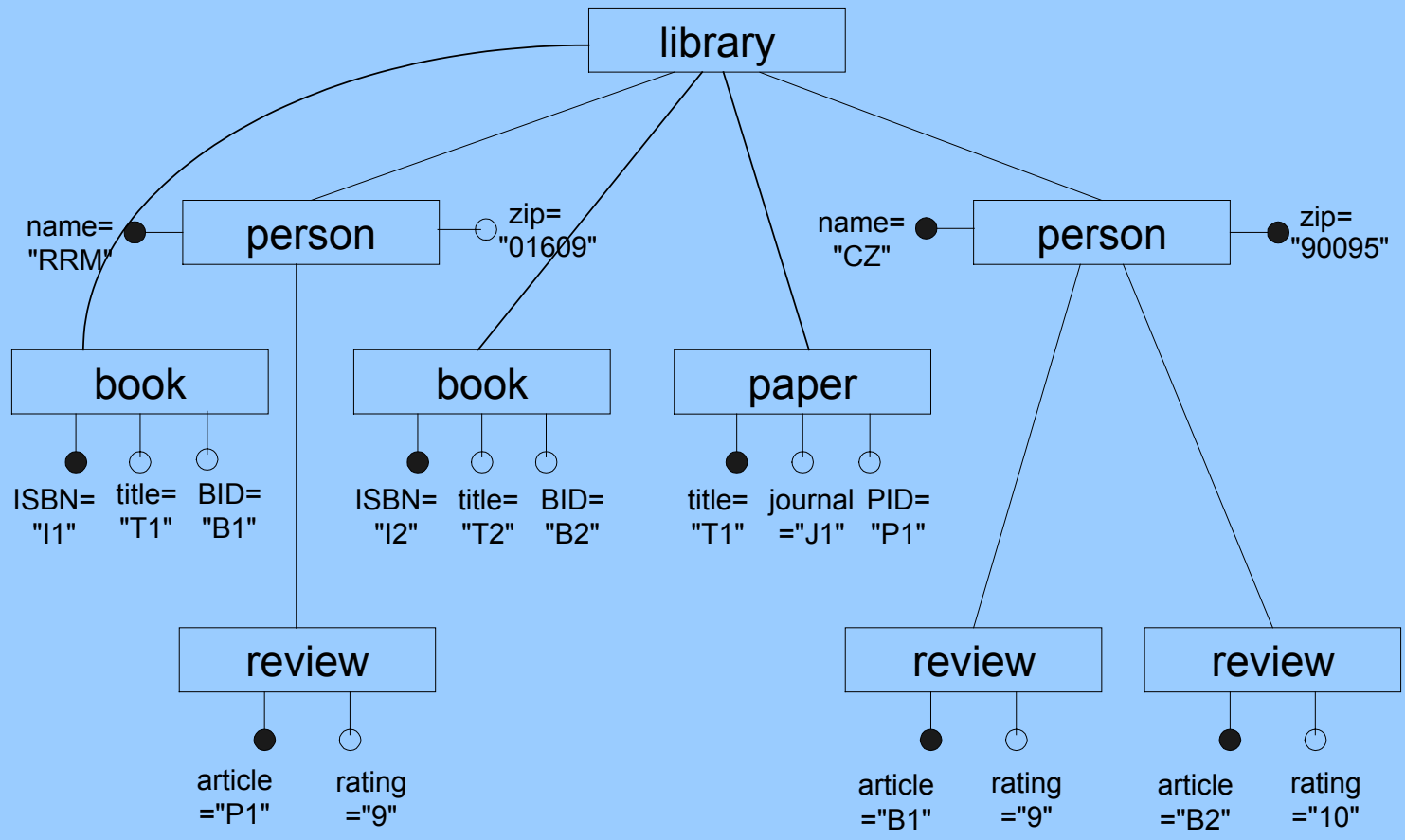
Paper

title	journal
T1	J1

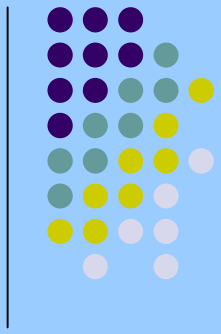
Review

name	book	paper	rating
RRM	null	T1	9
CZ	I1	null	9
CZ	I2	null	10

IDREF as union of foreign keys



@article::IDREF references (Book | Paper)

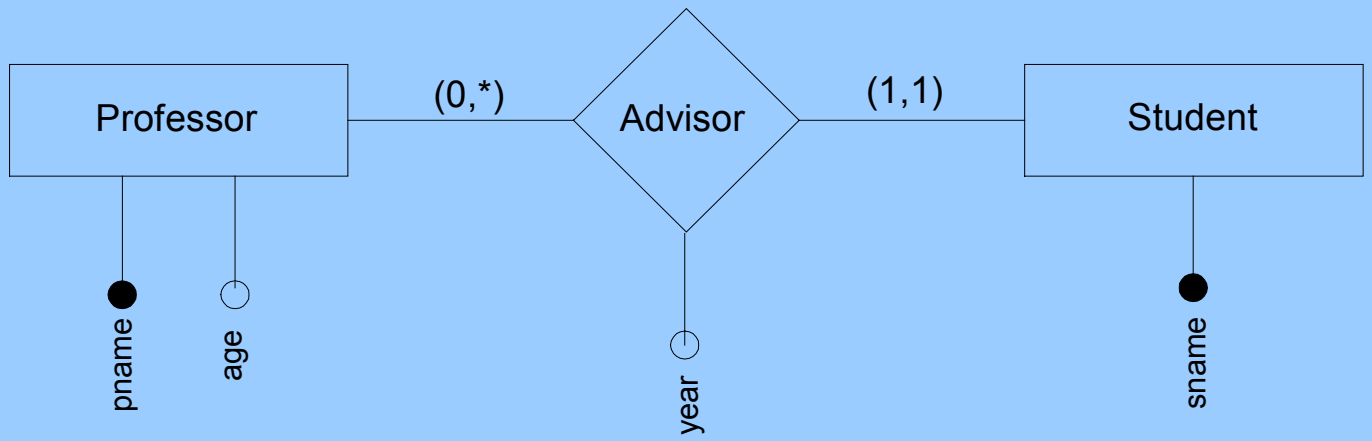


Conceptual Model:
ERex (ER extended for XML)

Murali Mani, Antonio Badia

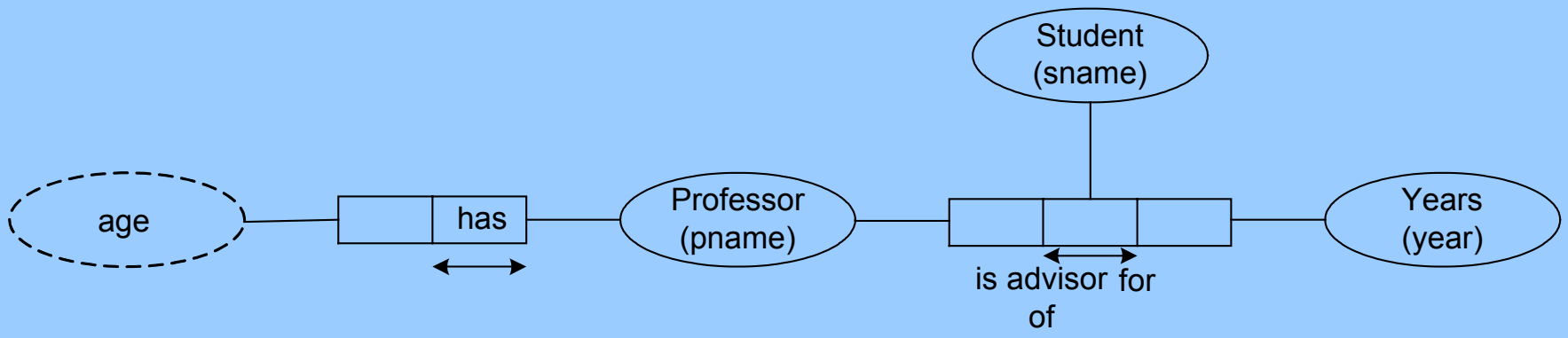
Entity Relationship (ER) model

- Entity Types, Relationship Types and their attributes



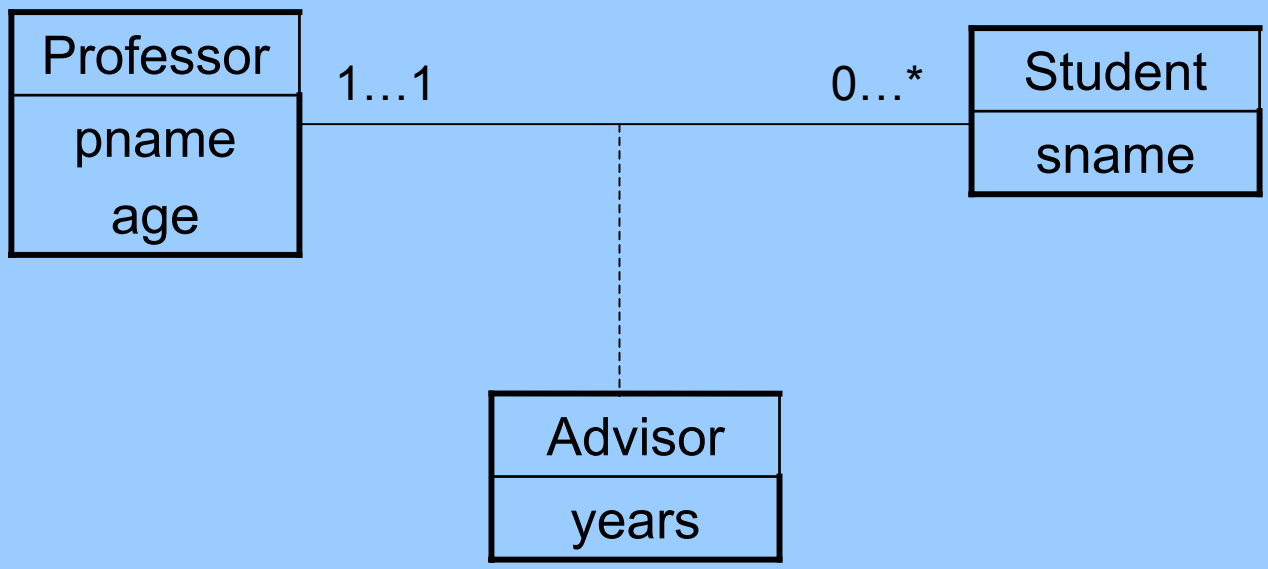
Object Role Modeling (ORM)

- closer to natural language sentences
- attributes/relationships are expressed uniformly using roles



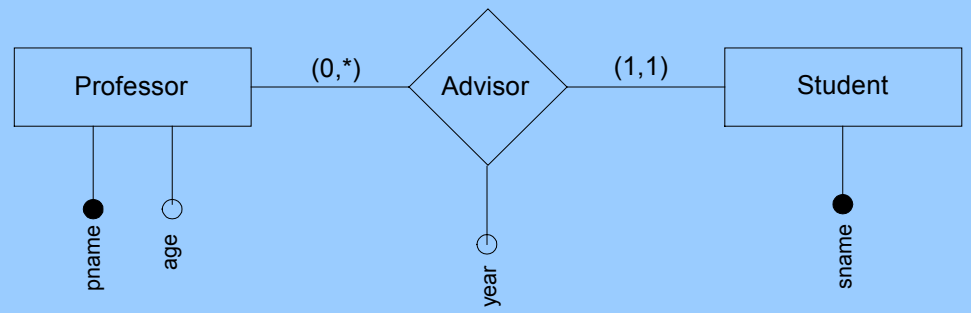
Unified Modeling Language (UML)

- Modeling software systems
- Class Diagrams, Association Classes

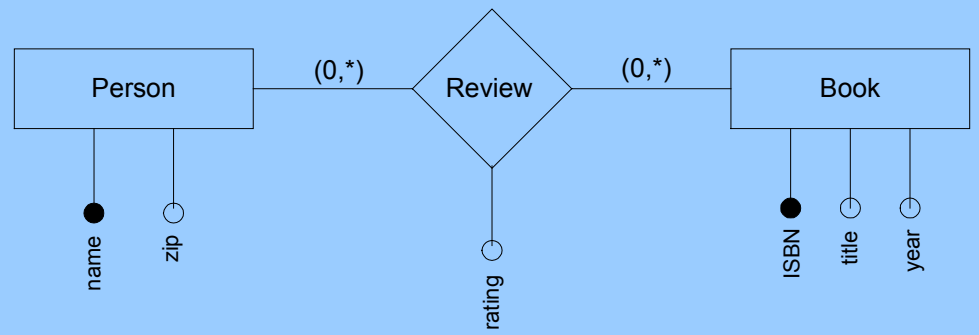


From ER model

Binary 1:n relationships

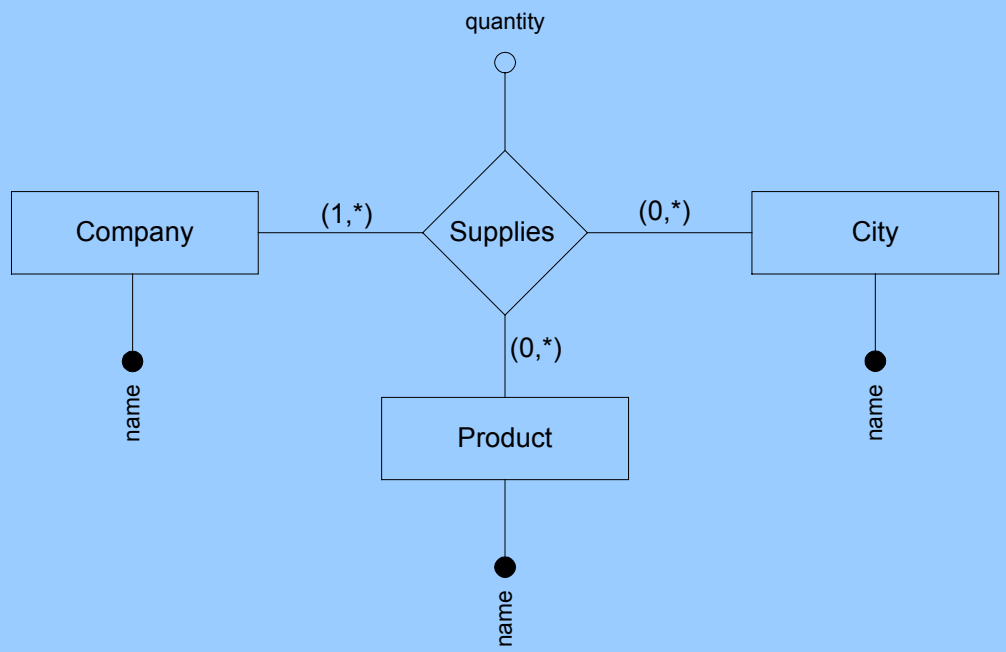


Binary m:n relationships

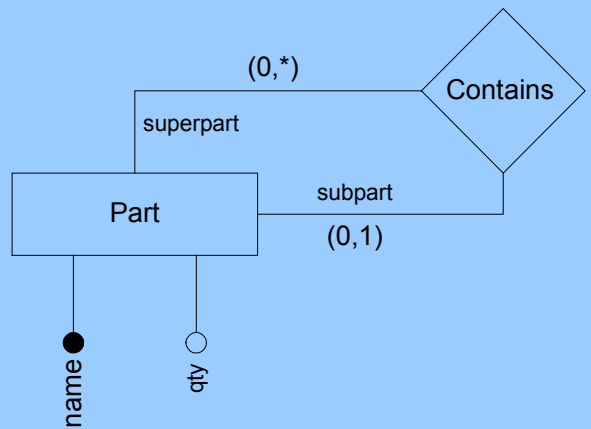


From ER Model

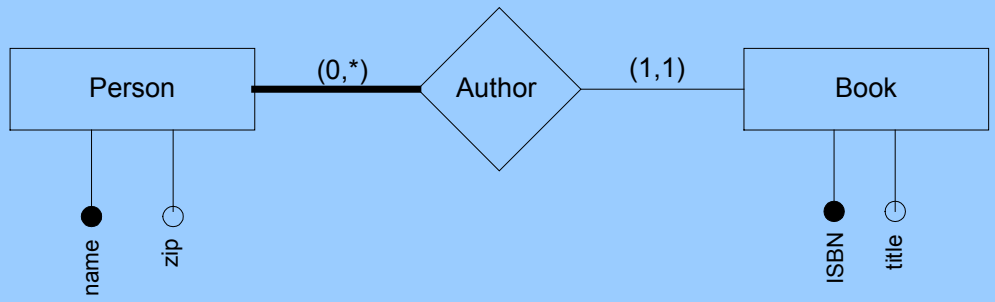
N-ary relationships



Recursive relationships



Ordered Relationships



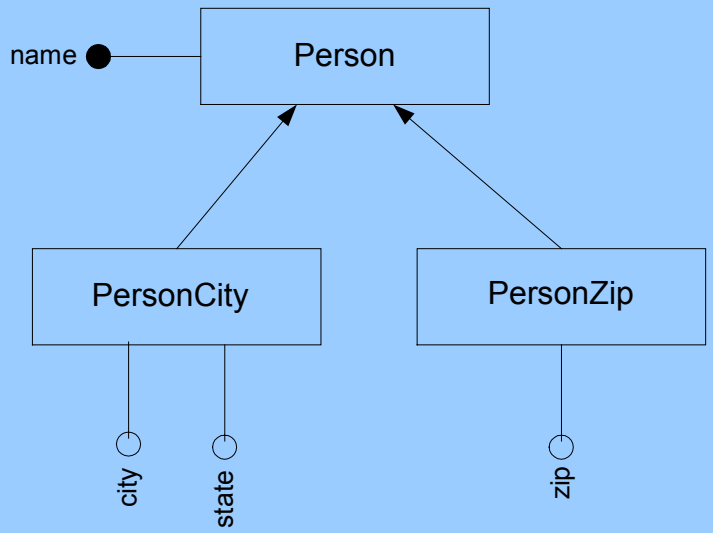
Person

name	zip
Ullman	95123
RRM	90095

Book

ISBN	title	order	author
B1	DB	2	Ullman
B2	Aut	1	Ullman
B3	MMSL	1	RRM

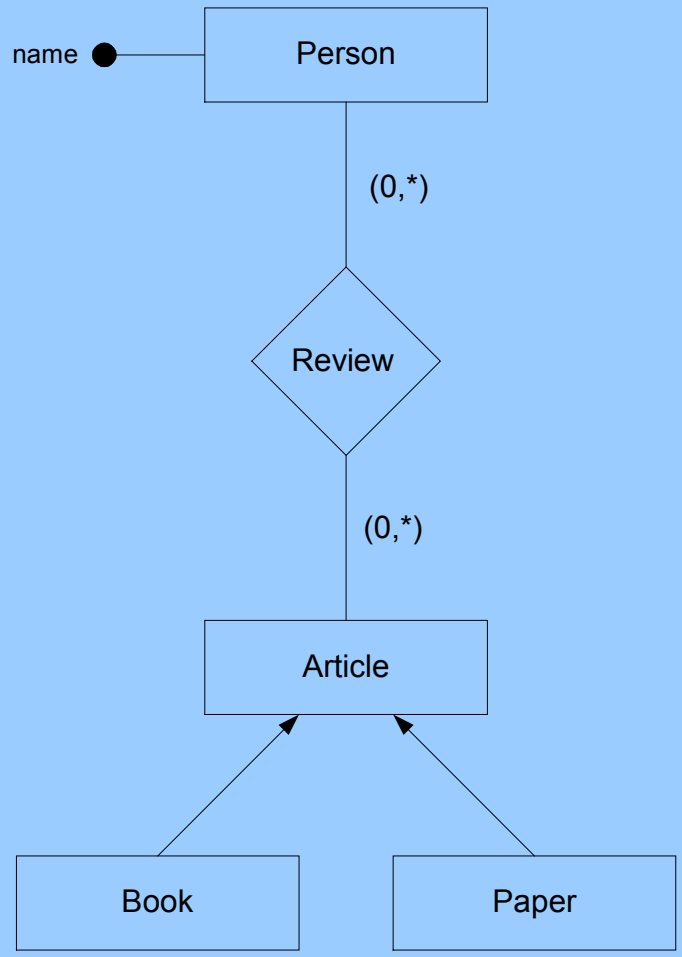
Categories and Set Constraints



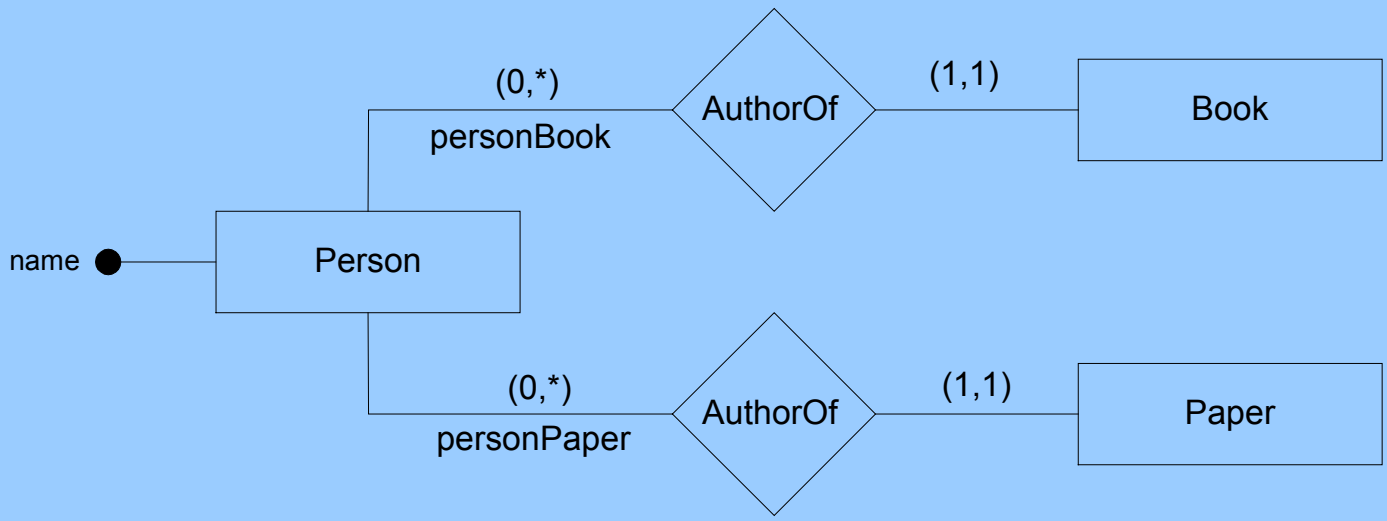
$PersonCity \cap PersonZip = \varnothing$

$PersonCity \cup PersonZip = Person$

Categories

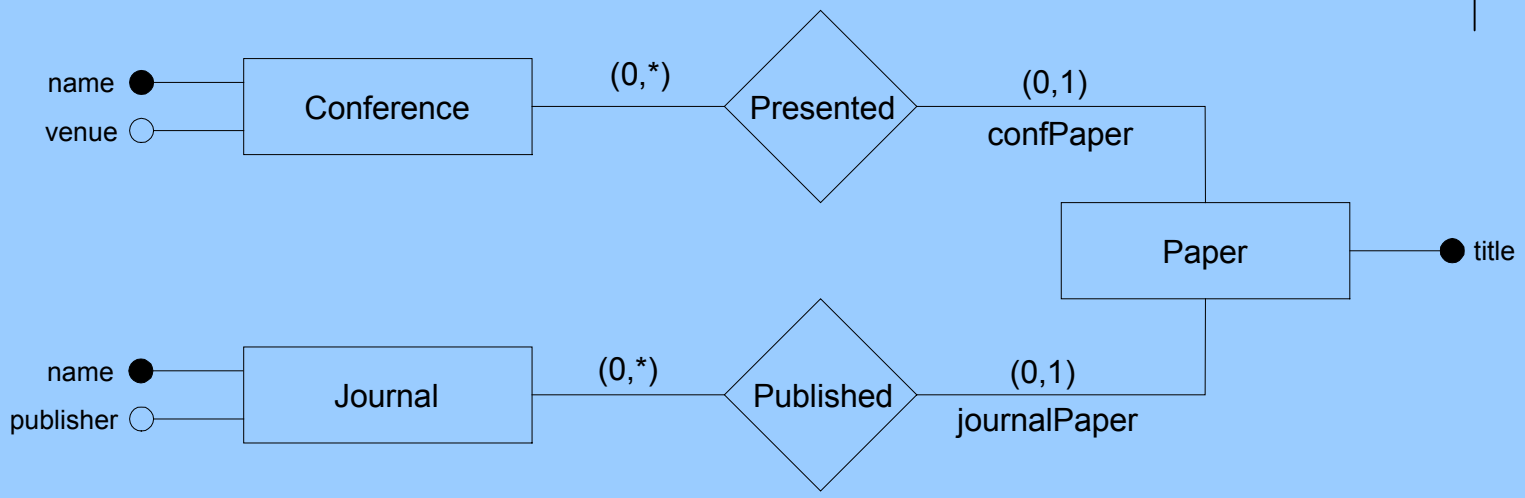


Set constraints on Roles



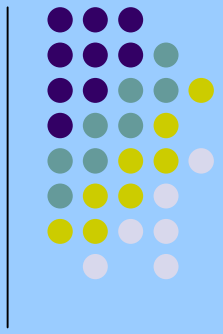
personBook ∩ personPaper = ∅

Set Constraints on Roles



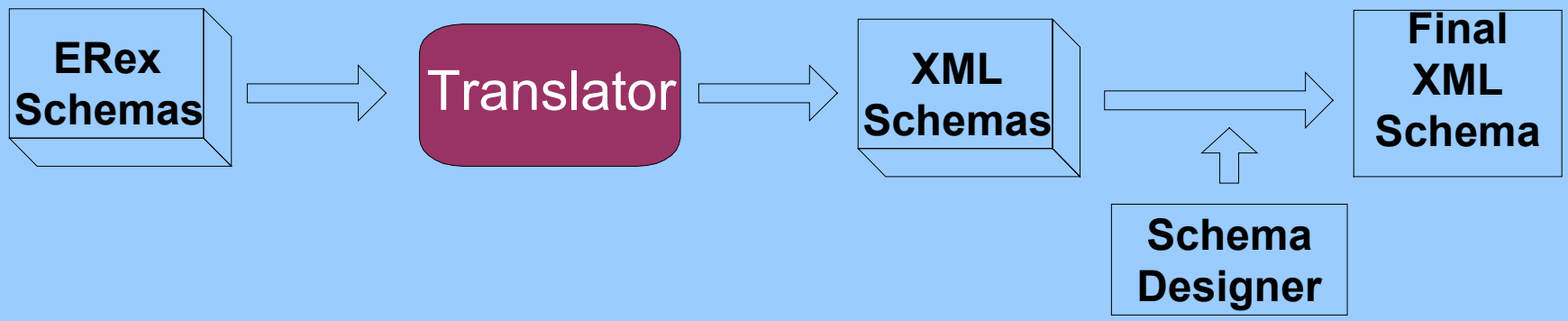
$confPaper \cap journalPaper = \varnothing$

$confPaper \cup journalPaper = Paper$

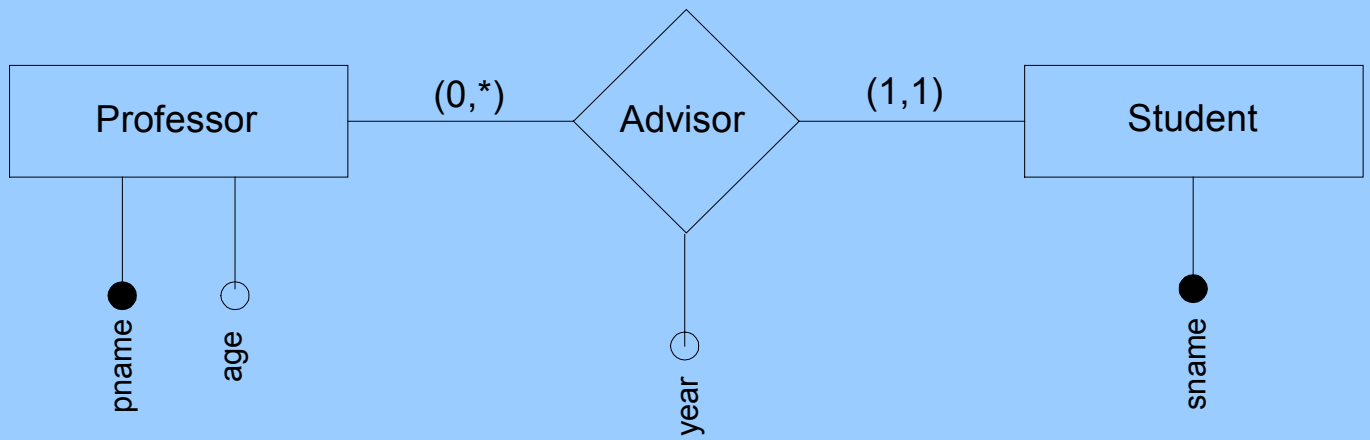


Translating ERex schemas to XML schemas

System Architecture



1:n relationships



Representing 1:n Relationships - subelement

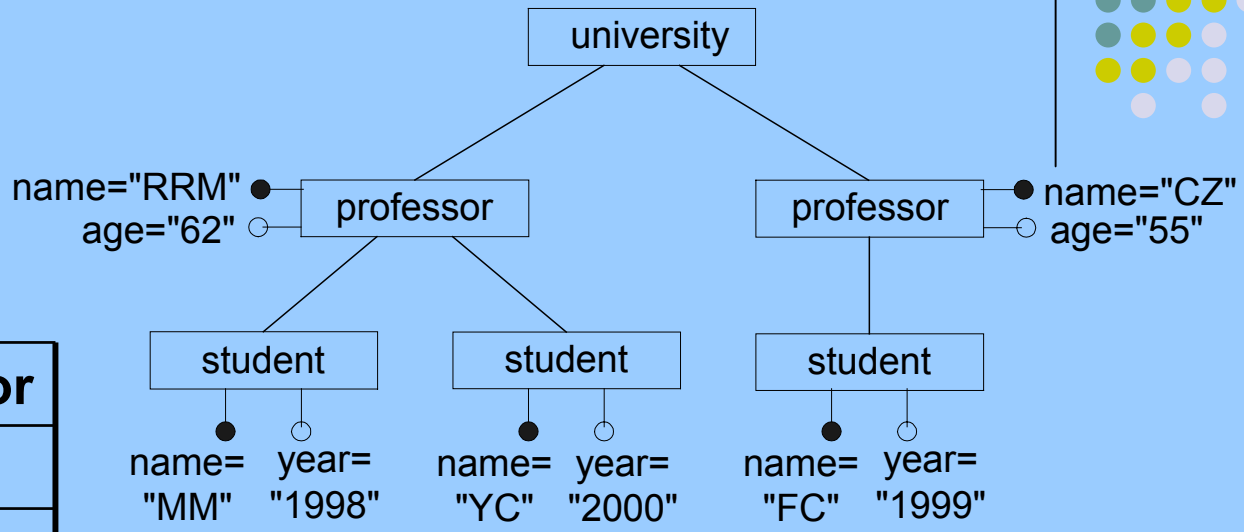
Professor

name	age
RRM	62
CZ	55

Student

name	year	advisor
MM	1998	RRM
YC	2000	RRM
FC	1999	CZ

Student (advisor) references Professor (name)



University → university (Professor*)
Professor → professor (@name, @age, Student*)
Student → student (@name, @year)

(University, Professor, <@name>)
(University, Student, <@name>)

Representing 1:n Relationships - IDREF

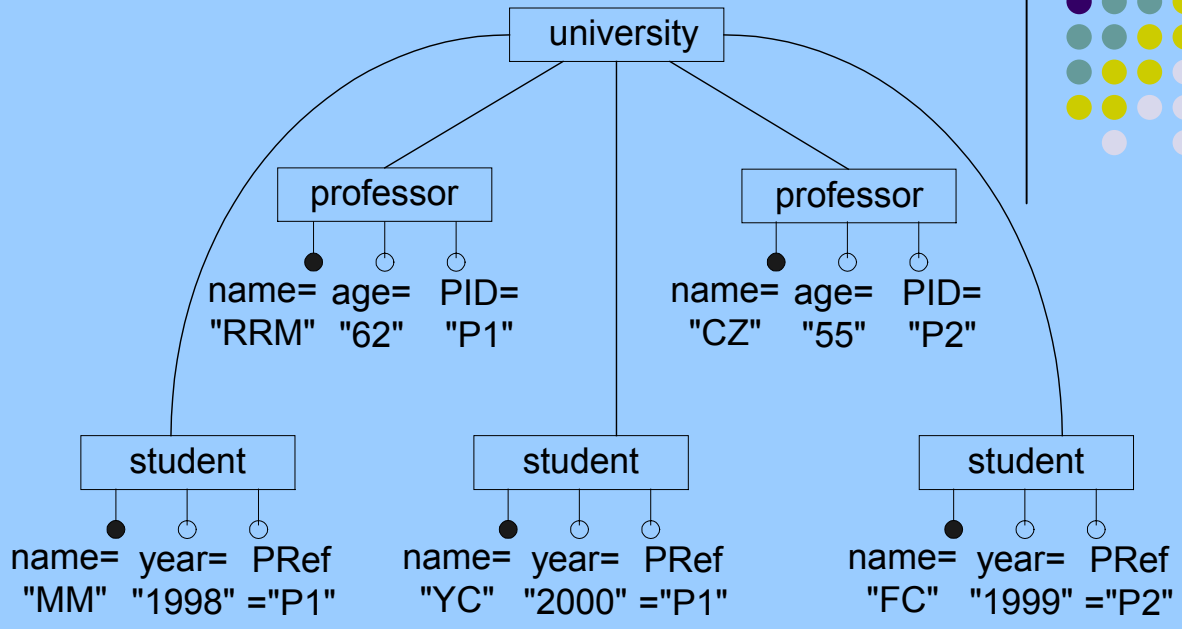
Professor

name	age
RRM	62
CZ	55

Student

name	year	advisor
MM	1998	RRM
YC	2000	RRM
FC	1999	CZ

Student (advisor) references Professor (name)



University → university (Professor*, Student*)
Professor → professor (@name, @age, @PID)
Student → student (@name, @year, @PRef)

(University, Professor, <@name>)
(University, Student, <@name>)
@PRef::IDREF references (Professor)

Representing 1:n Relationships – foreign keys

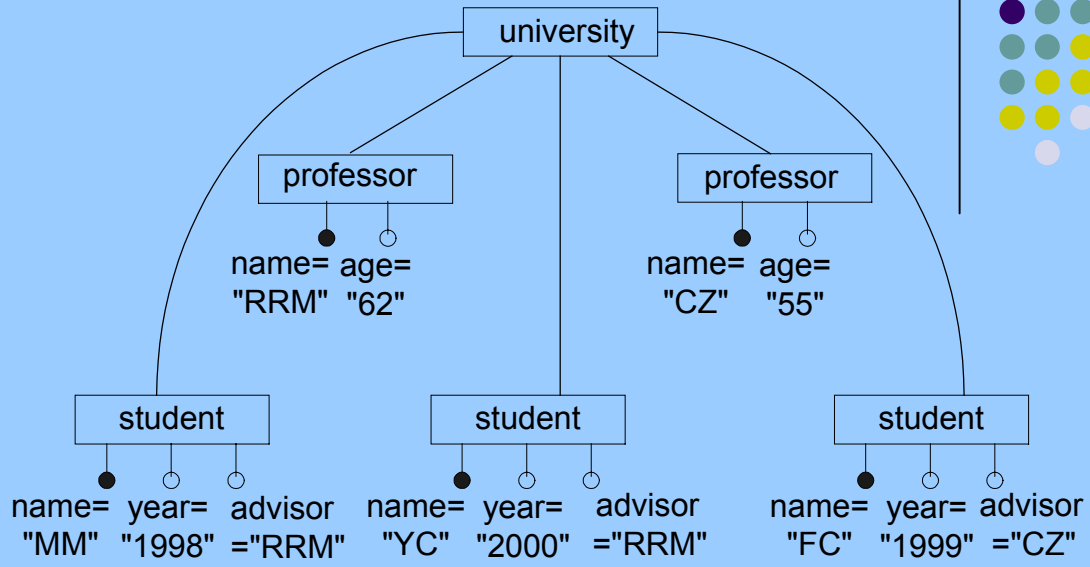
Professor

name	age
RRM	62
CZ	55

Student

name	year	advisor
MM	1998	RRM
YC	2000	RRM
FC	1999	CZ

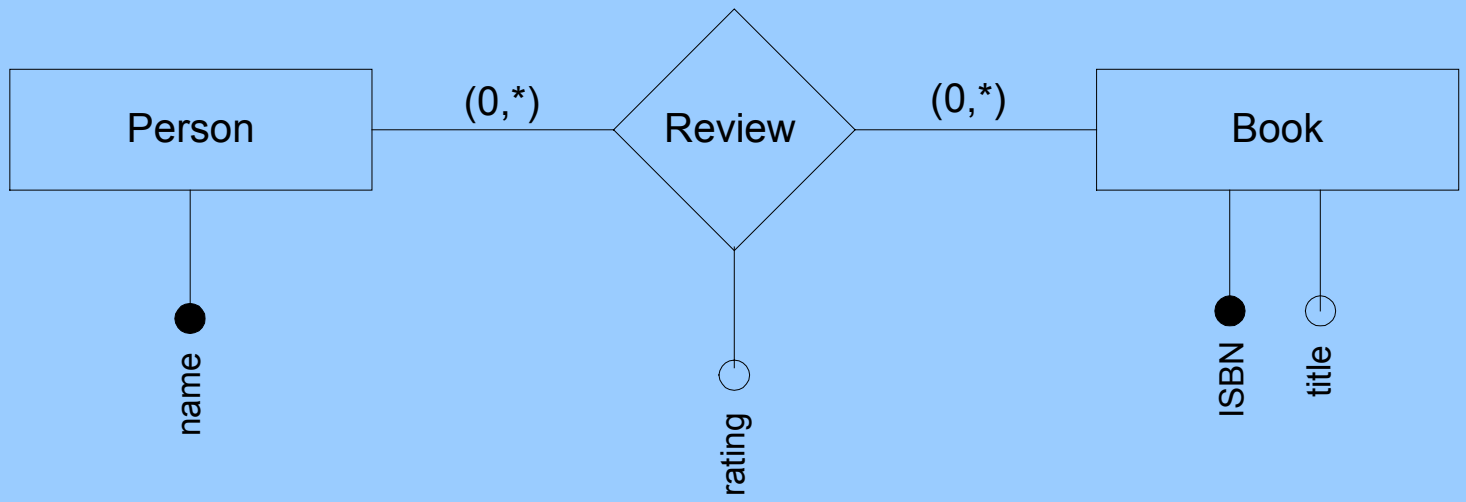
Student (advisor) references Professor (name)



University → university (Professor*, Student*)
Professor → professor (@name, @age)
Student → student (@name, @year, @advisor)

(University, Professor, <@name>)
(University, Student, <@name>)
(University, Student, <@advisor>) references (University, Professor, <@name>)

m:n relationships



Representing m:n relationships

Person

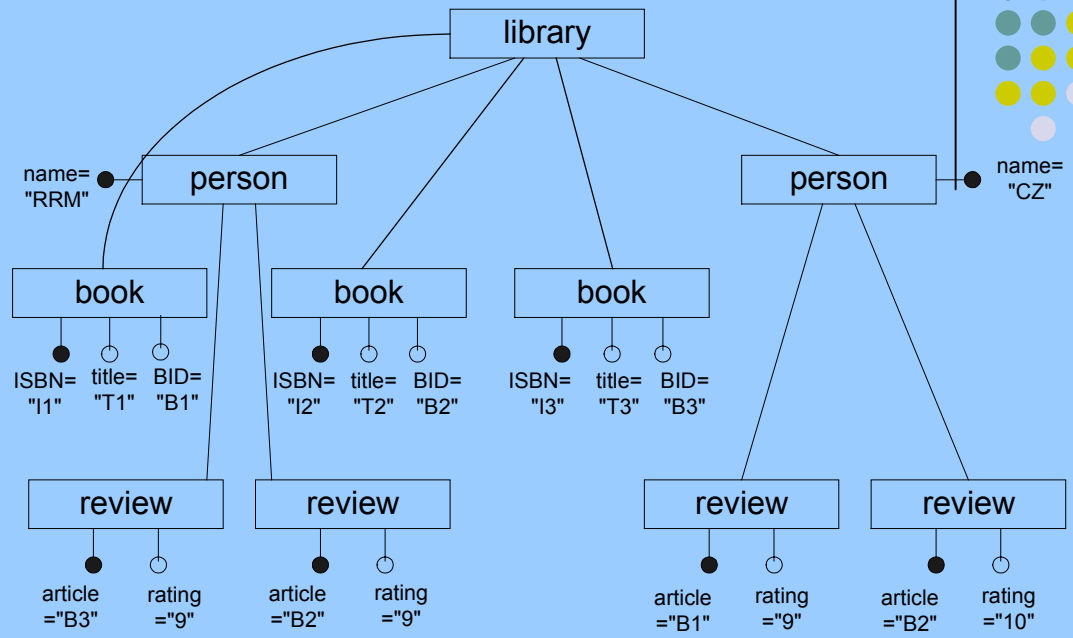
name
RRM
CZ

Book

ISBN	title
I1	T1
I2	T2
I3	T3

Review

pname	ISBN	rating
RRM	I3	9
RRM	I2	9
CZ	I1	9
CZ	I1	10



Library → library (Person*, Book*)
Person → person (@name, Review*)
Book → book (@ISBN, @title, @BID)
Review → review (@article, @rating)

Review (pname) references Person (name)
Review (ISBN) references Book (ISBN)

(Library, Person, <@name>) (Library, Book, <@ISBN>)
(Person, Review, <@article>)
@article::IDREF references (Book)

Representing m:n relationships

Person

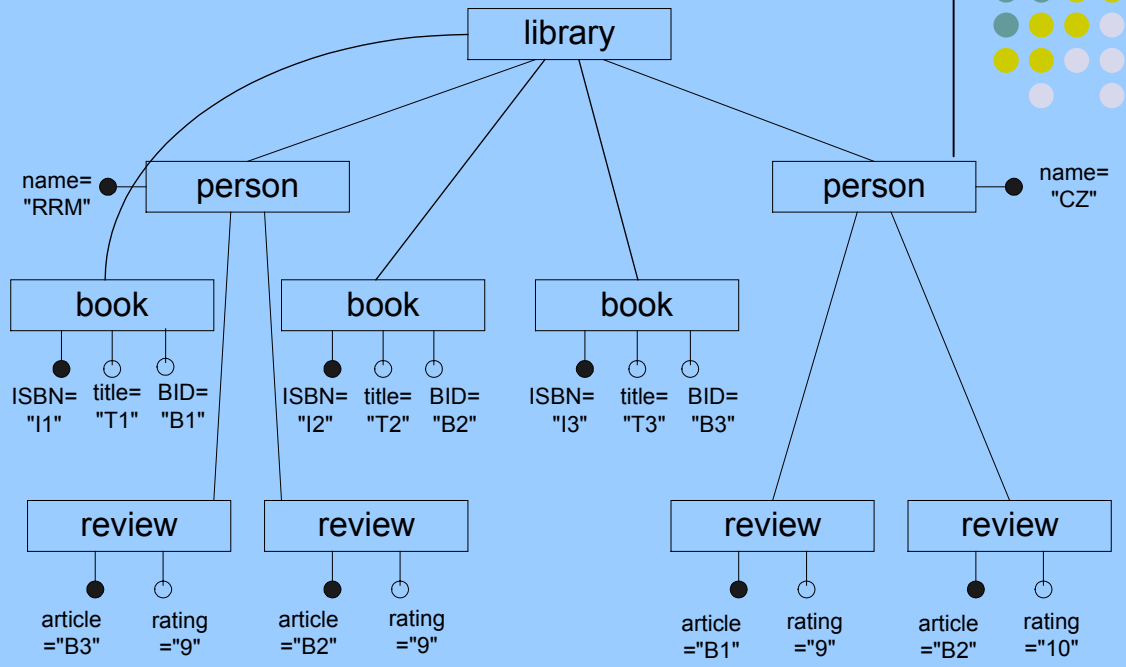
name
RRM
CZ

Book

ISBN	title
I1	T1
I2	T2
I3	T3

Review

pname	ISBN	rating
RRM	I3	9
RRM	I2	9
CZ	I1	9
CZ	I1	10

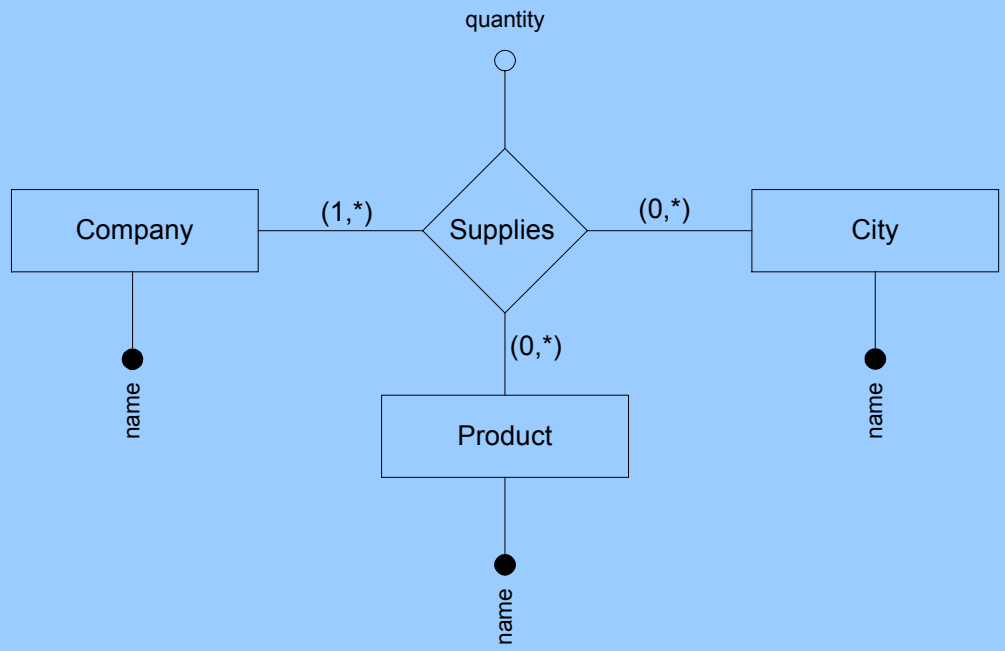


Query: What is the rating given by RRM for book T1

$\pi_{rating}((\sigma_{title=T1}(Book)) \otimes (\sigma_{pname=RRM}(Review)))$

person[@name=RRM]/review[@article⇒Book/title=T1]/@rating

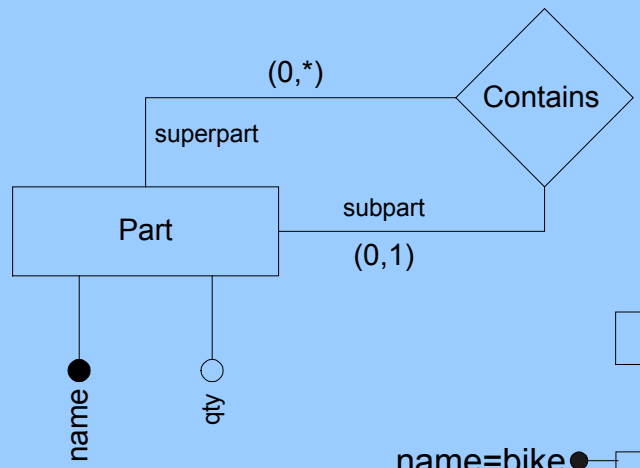
N-ary relationships

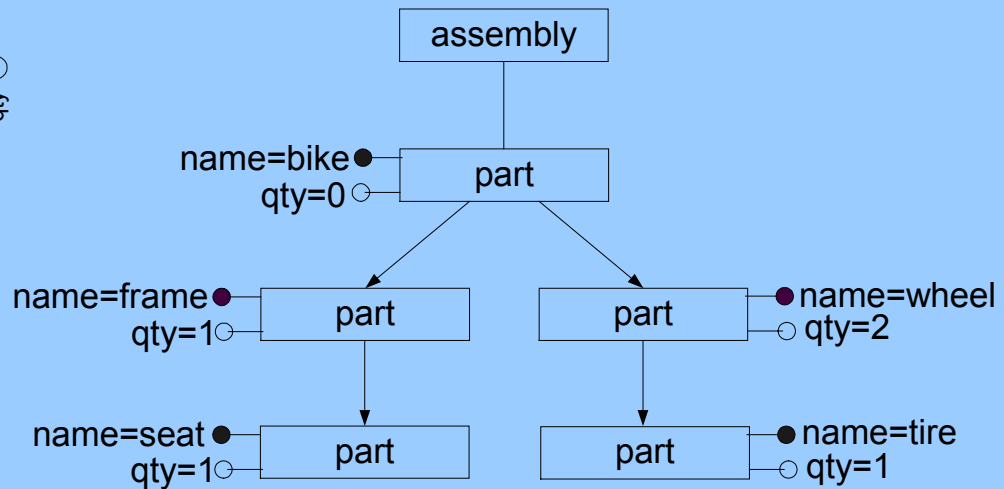


Root → root (Company*, Product*, City*)
Company → company (@name, Supply+)
Supply → supply (@ProdRef, @CityRef, @qty)
Product → product (@name, @ProdID)
City → city (@name, @CityID)

(Root, Company, <@name>)
(Root, Product, <@name>)
(Root, City, <@name>)
@ProdRef::IDREF references (Product)
@CityRef::IDREF references (City)

Recursive Relationships





Assembly

name	superPart	qty
seat	frame	1
tire	wheel	1
frame	bike	1
wheel	bike	2
bike	null	0

Assembly → assembly (Part*)

Part → part (@name, @qty, Part*)

(assembly, part, <@name>)

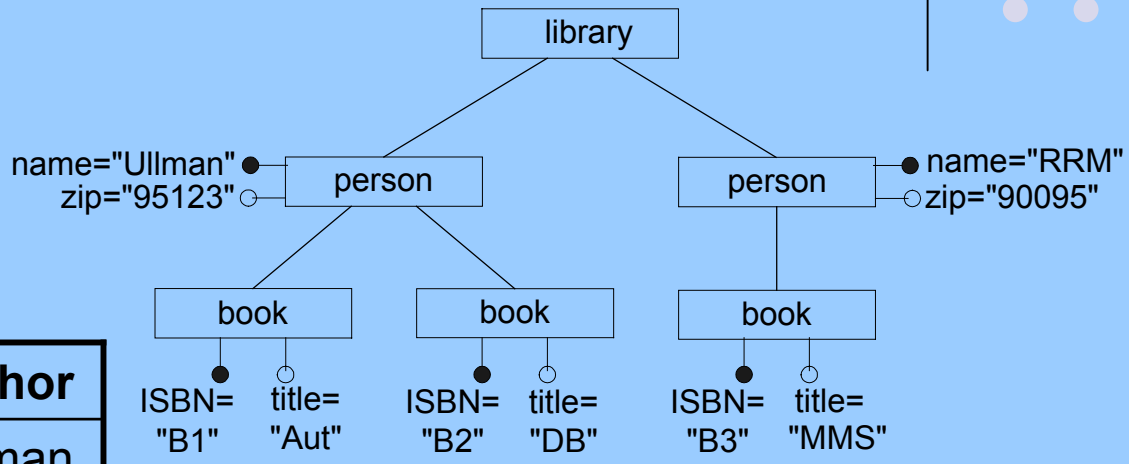
Ordered Relationships

Person

name	zip
Ullman	95123
RRM	90095

Book

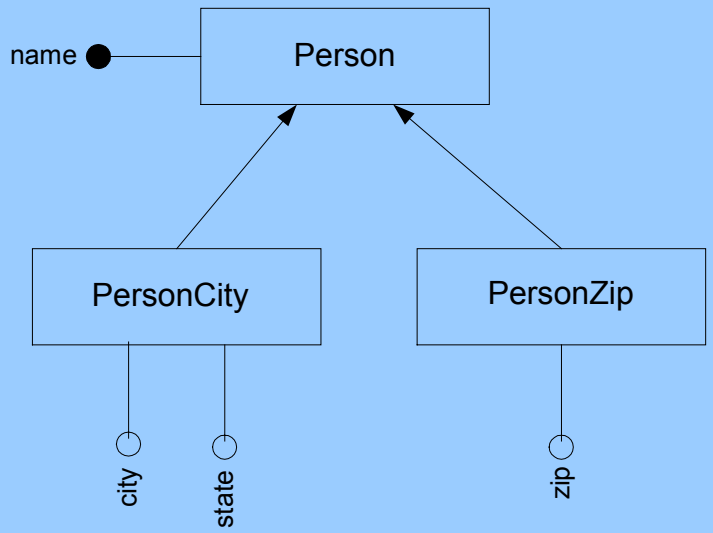
ISBN	title	order	author
B1	DB	2	Ullman
B2	Aut	1	Ullman
B3	MMS	1	RRM



Library → library (Person*)
Person → person (@name, @zip, Book*)
Book → book (@ISBN, @title)

(Library, Person, <@name>)
(Library, Book, <@ISBN>)

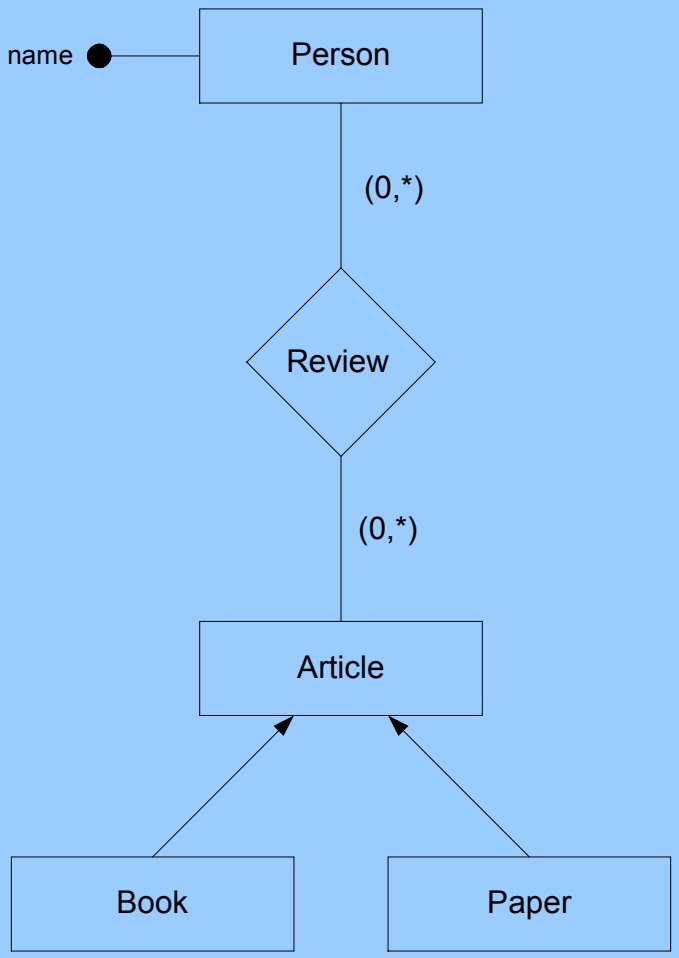
Categories and set constraints



$PersonCity \cap PersonZip = \varnothing$
$PersonCity \cup PersonZip = Person$

Person $\rightarrow$ person (@name, ((@city, @state) | @zip))

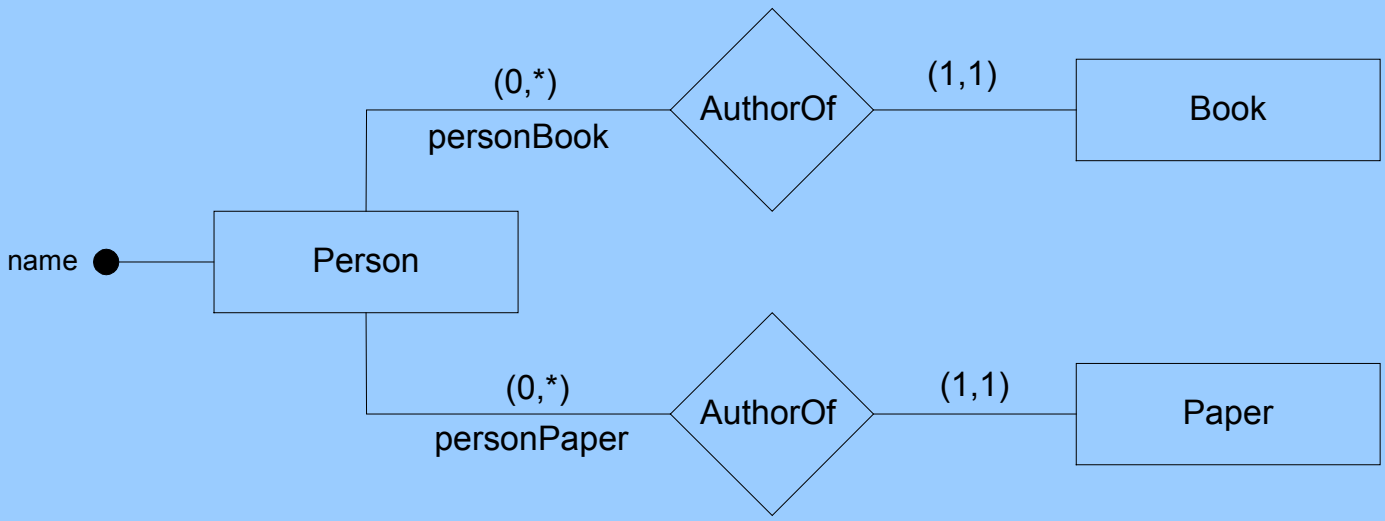
Categories and Set Constraints



Root → root (Person*, Book*, Paper*)
Person → person (@name, Review*)
Review → review (@article, @rating)

(Root, Person, <@name>)
(Person, Review, <@article>)
@article::IDREF references (Book | Paper)

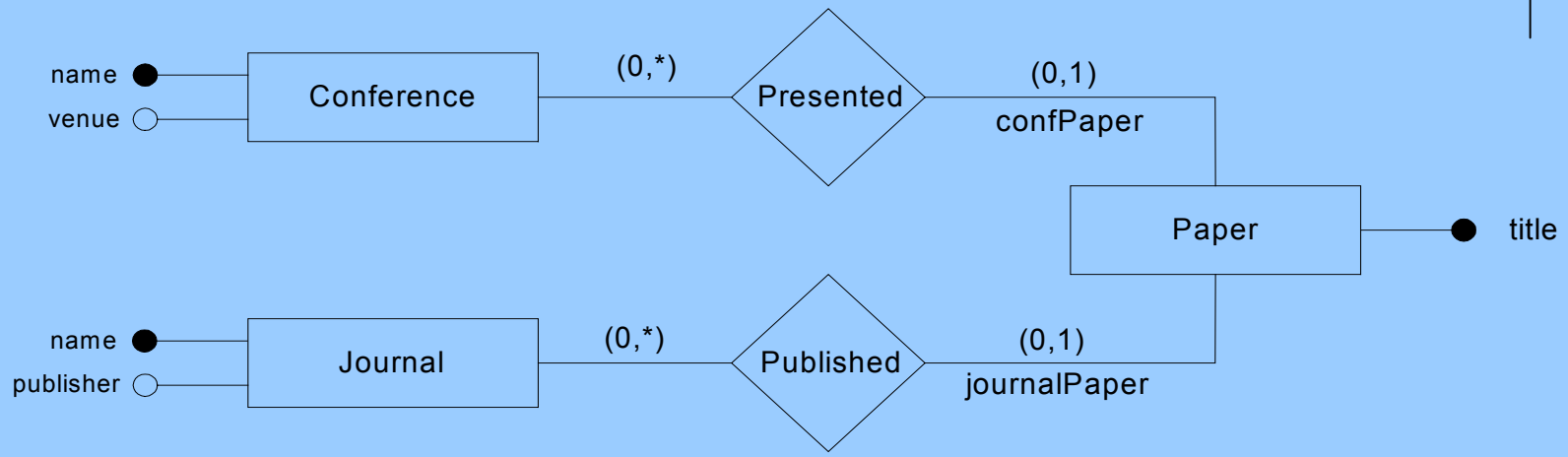
Set constraints on Roles



personBook ∩ personPaper = ∅

Person → person (@name, (Book* | Paper*))

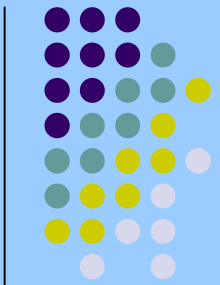
Set Constraints on Roles



confPaper ∩ journalPaper = ∅
confPaper ∪ journalPaper = Paper

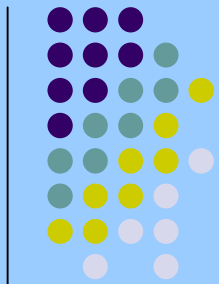
Conference → conference (@name, @venue, Paper*)
Journal → journal (@name, @publisher, Paper*)

Converting ERex → XML

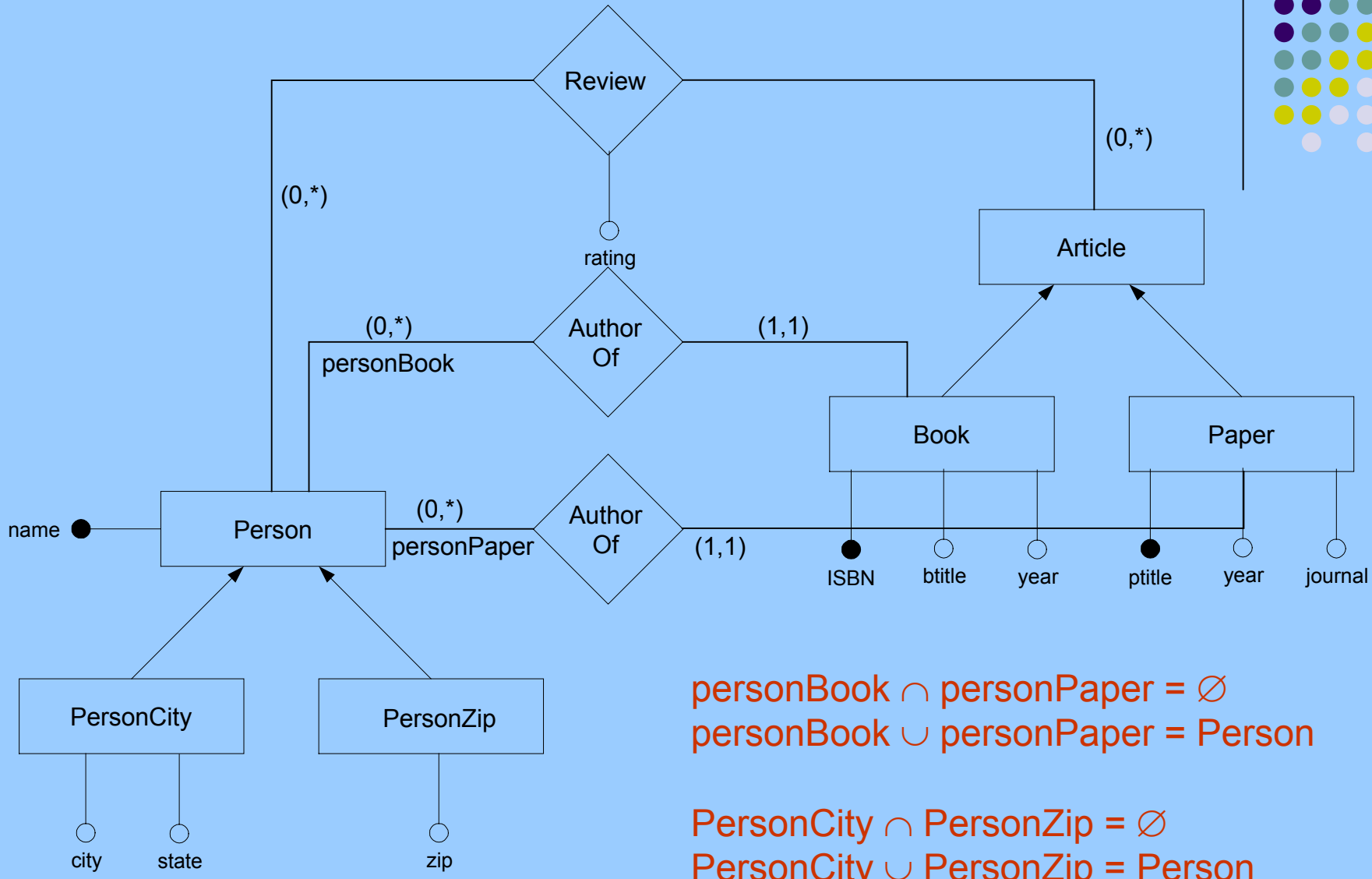


- Goals
  - Maximize relationships represented using subelement.
  - Others try to represent using IDREF

Algorithm: ERex → XML

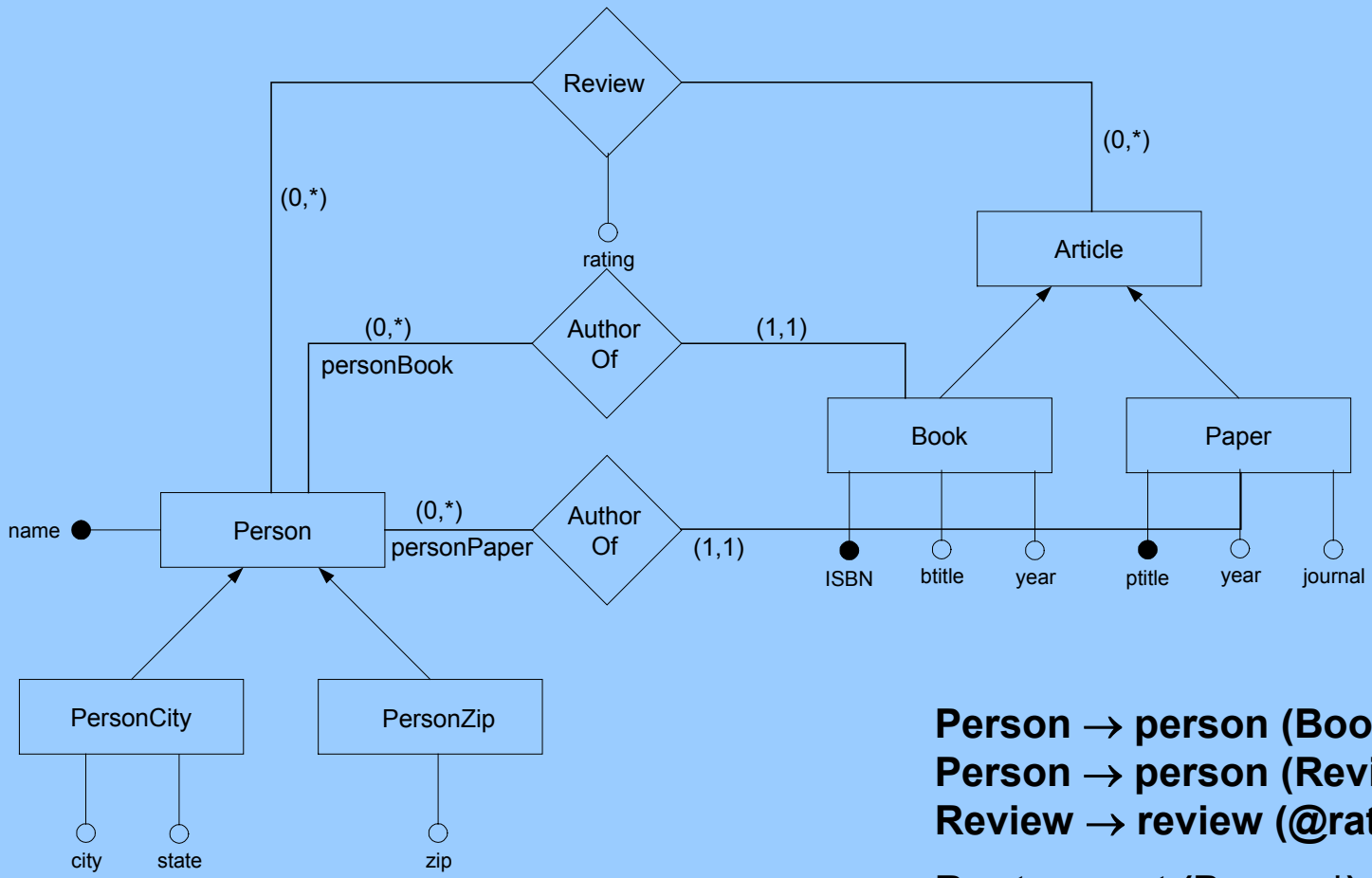


- A non-terminal symbol for each
  - entity type with key
  - m:n relationship
  - n-ary relationship
  - Root non-terminal symbol
- Represent attributes
- Represent relationships and identify top nodes
  - 1:1 and 1:n relationships
  - m:n relationships
  - n-ary relationships
- Identify key and IDREF constraints.



$personBook \cap personPaper = \varnothing$
$personBook \cup personPaper = Person$

$PersonCity \cap PersonZip = \varnothing$
$PersonCity \cup PersonZip = Person$



N = {Root, Person, Book, Paper, Review}

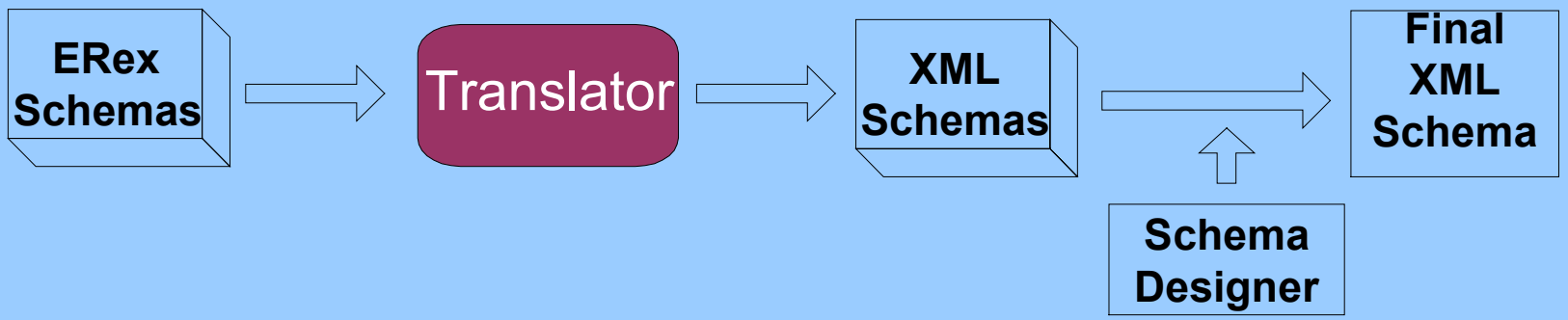
Book → book (@ISBN, @btitle, @year)
Paper → paper (@ptitle, @year, @journal)
Person → person (@name,
((@city, @state) | @zip))

Person → person (Book* | Paper*)
Person → person (Review*)
Review → review (@rating, @article)

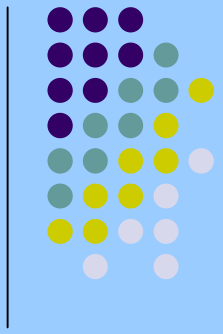
Root → root (Person*)

(Root, Person, <@name>)
(Root, Book, <@ISBN>)
(Root, Paper, <@ptitle>)
(Person, Review, <@article>)
@article::IDREF references (Book | Paper)

Conclusions



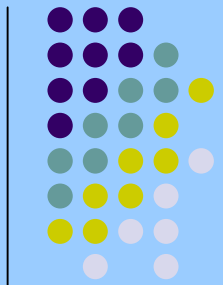
- Obtained good XML Schema from ERex schemas



Part II:

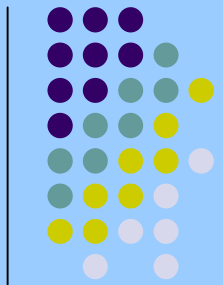
Translation between Relational and XML models.

Why publish relational databases as XML?



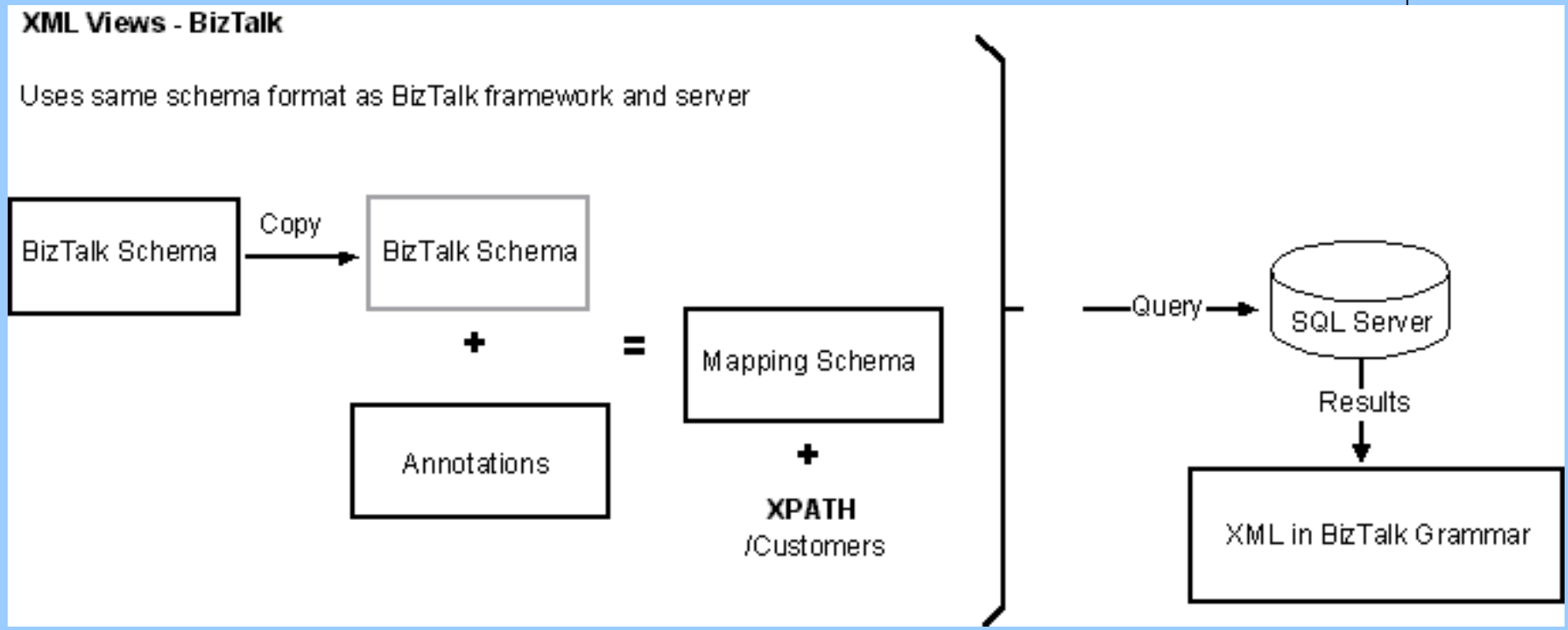
- Provide an XML view for our legacy data
  - Users/Applications can query our data over the web using standards.
  - Easier to query XML than legacy (relational) data?
- Convert our legacy data to XML
  - We can exchange data with applications.
  - Store data in XML databases?
  - Easier to query?

Application Scenarios

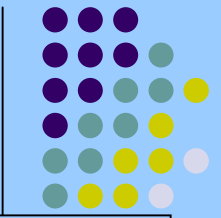


- Schema Matching Problem
  - Given a relational schema and an XML schema by a standards body, how do we map this relational schema to XML?
  - Tools such as XML Extender from IBM, Clio (University of Toronto and IBM), MS SQL Server
- Schema Mapping Problem
  - Given a relational schema, how do we come up with a ***good*** XML schema?

Schema Matching: MS SQL Server Architecture



Schema Mapping Problem



Name	Age	Salary
A	25	20000
B	62	140000

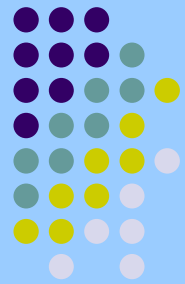
```
<person>
   <Name>A</name>
   <Age>25</Age>
   …
</person>
```

- Users/Applications see a uniform XML view
- Exchange data with other applications
- Store XML in native XML databases
- Querying XML is easier?

Problem:

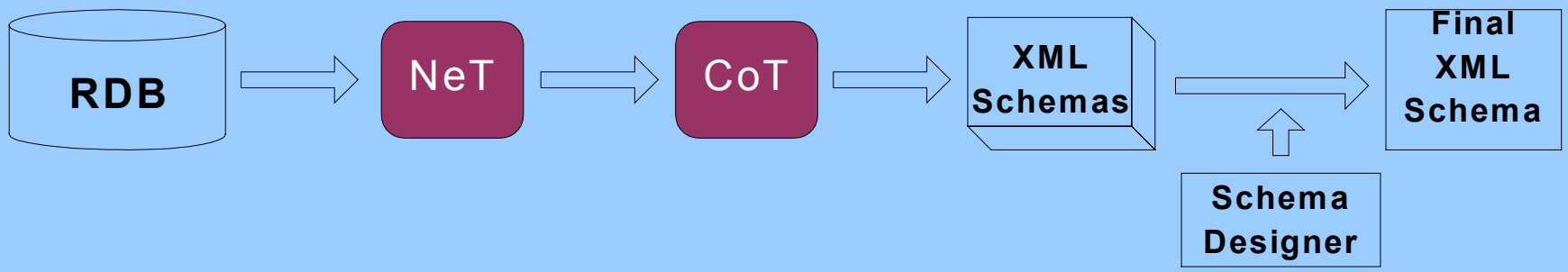
What is a good XML schema for a relational schema?

Goals



- XML Schema should maintain constraints.
- Resulting XML should not introduce redundancies.
- Most relationships can be navigated using path expressions, rather than joins.
- Minimal user interaction: Our translator should suggest good XML schemas to the database designer.

System Architecture



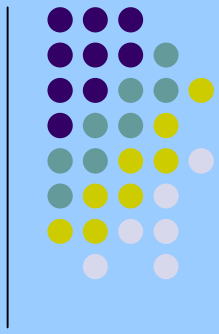
Related Work

- XML-DBMS
  - Template driven mapping language
- SilkRoute
  - Declarative Query Language (RXL) for viewing relational data as XML
- Xperanto
  - User specifies query in XML Query Language

Algorithms

- Naïve
  - FT (Flat Translation)
- Consider relational data
  - NeT (Nesting-based Translation)
- Consider relational schema
  - CoT (Constraint-based Translation)

FT: Flat Translation



- 1 : 1 mapping from relational to XML
- Idea
  - A type (non-terminal) corresponding to every relation
  - Attributes of a relation form attributes of the type
  - Keys and foreign keys are preserved

FT: Flat Translation - Example

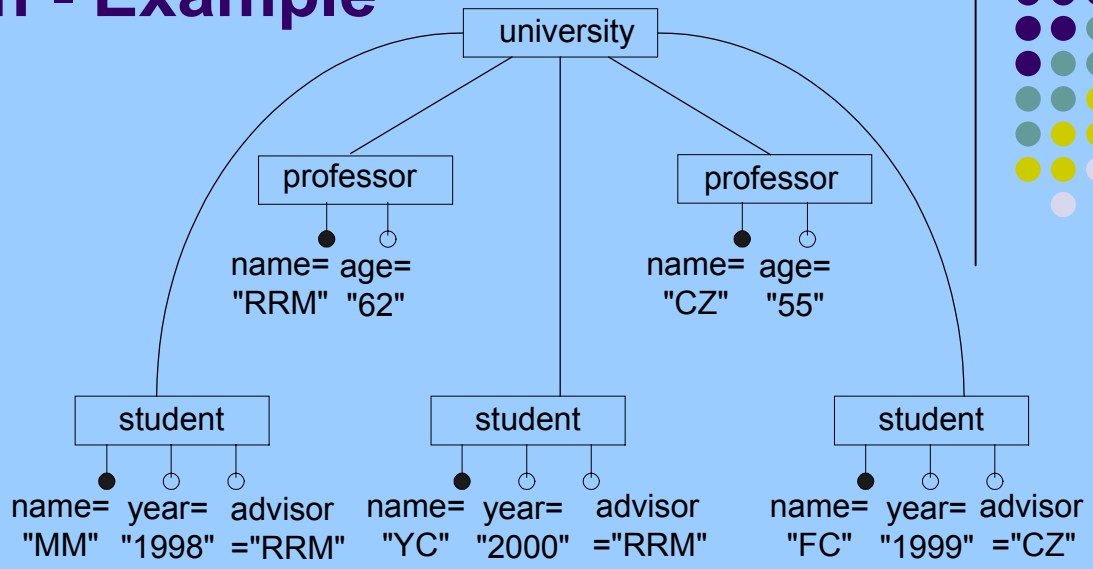
Professor

name	age
RRM	62
CZ	55

Student

name	year	advisor
MM	1998	RRM
YC	2000	RRM
FC	1999	CZ

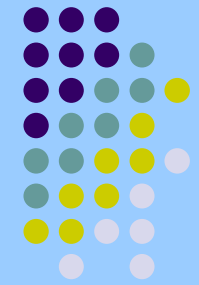
Student (advisor) references Professor (name)



University → university (Professor*, Student*)
Professor → professor (@name, @age)
Student → student (@name, @year, @advisor)

(University, Professor, <@name>)
(University, Student, <@name>)
(University, Student, <@advisor>) references
(University, Professor, <@name>)

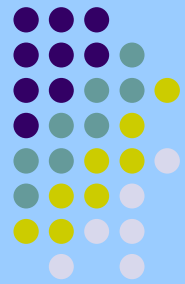
NeT: Nesting-based Translation



- Idea:

  Make use of non-flat features provided by XML: represent repeating groups using *, +

NeT: Example



Course (<u>cname, prof, text</u>)

cname	prof	text
Algorithms	Gafni	Udi Manber
Algorithms	Gafni	CLR
Algorithms	Majid	Udi Manber
Algorithms	Majid	CLR

NeT: Example

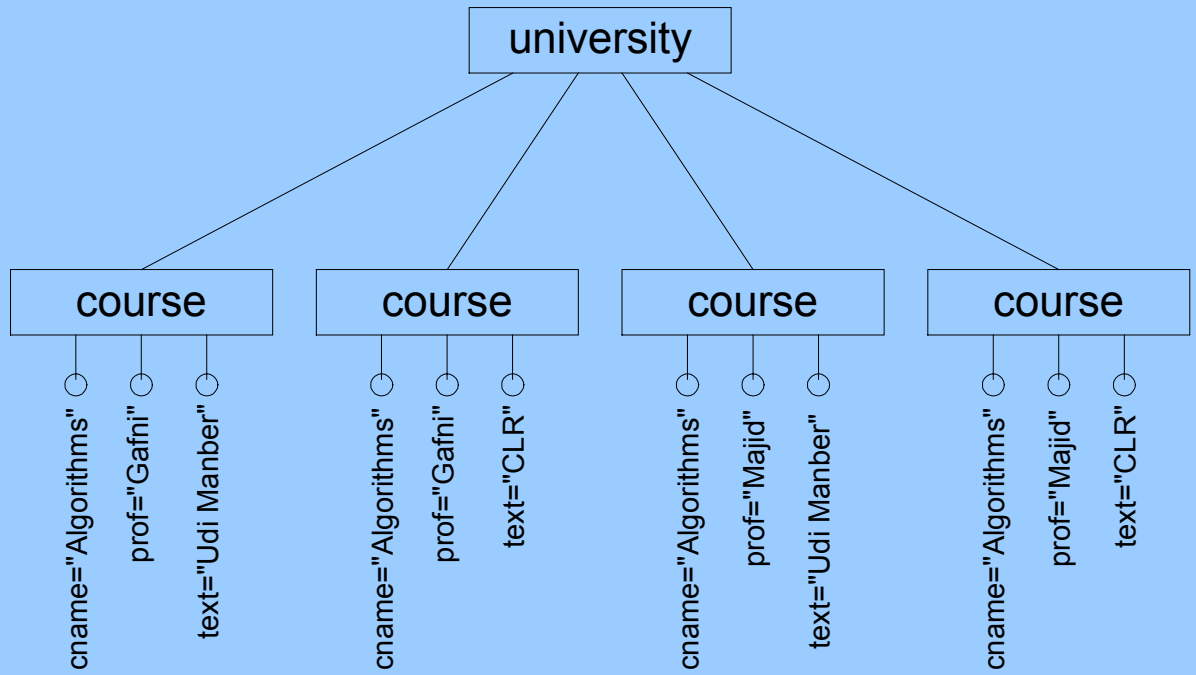
Course (<u>cname, prof $^{\pm}$</u>, text)

cname	prof	text
Algorithms	{Gafni, Majid}	Udi Manber
Algorithms	{Gafni, Majid}	CLR

Course (<u>cname, prof $^{\pm}$, text $^{\pm}$</u>)

Cname	prof	text
Algorithms	{Gafni, Majid}	{Udi Manber, CLR}

NeT: Example



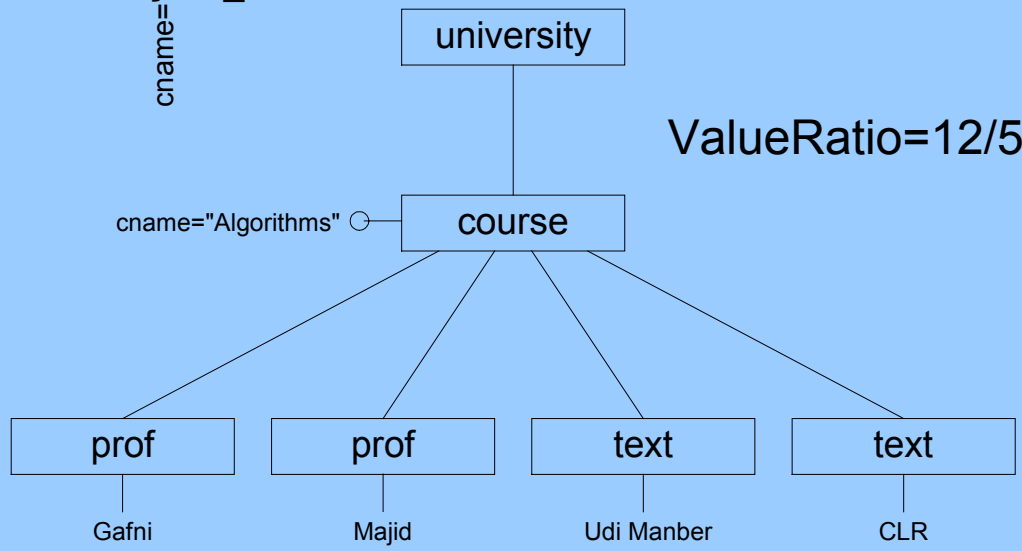
University → university (Course*)
Course → course (@cname, @prof, @text)

(University, Course, <@cname, @prof, @text>)

ValueRatio=12/5

University → university (Course*)
Course → course (@cname, Prof+, Text+)
Prof → prof (#PCDATA)
Text → text (#PCDATA)
#PCDATA → pcdata (ε)

(University, Course, <@cname, prof, text>)



NeT: Example

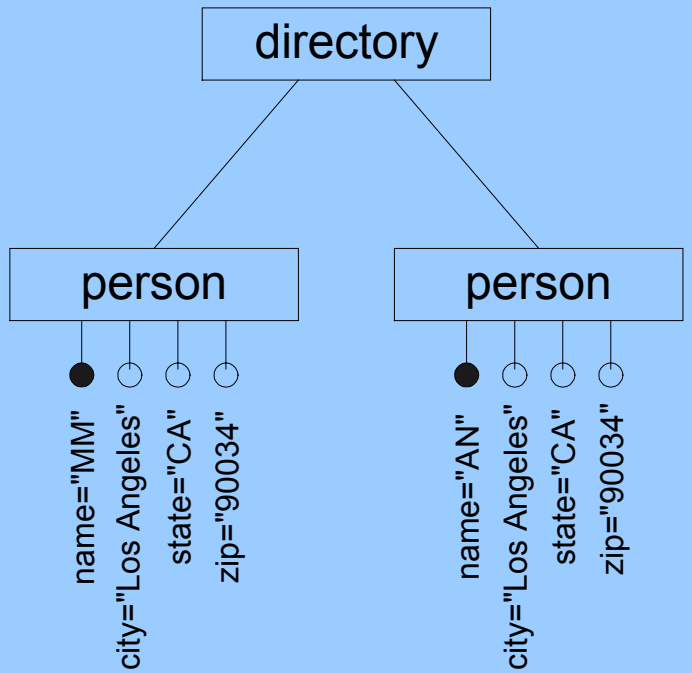
Person (<u>name</u>, city, state, zip)

name	city	state	zip
MM	Los Angeles	CA	90034
AN	Los Angeles	CA	90034

Person (<u>name</u>$^{\pm}$, city, state, zip)

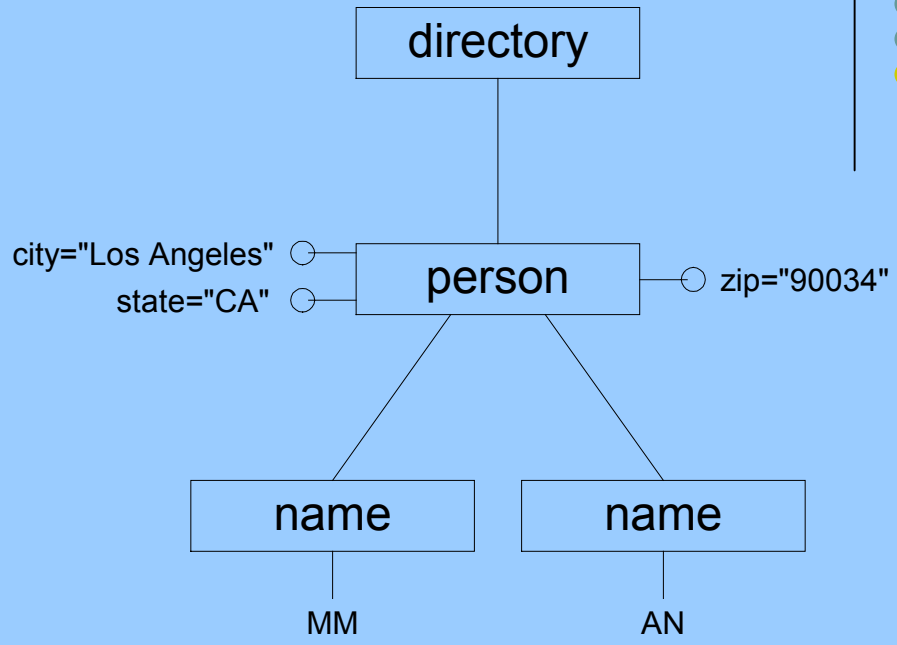
{MM, AN}	Los Angeles	CA	90034

NeT: Example



Directory → directory (Person*)
Person → person (@name, @city, @state, @zip)

(directory, person, <@name>)

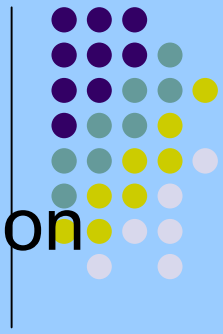


Directory → directory (Person*)
Person → person (@city, @state, @zip, Name*)
Name → name (#PCDATA)

(directory, person, <@name>)

ValueRatio=8/5

NeT: Summary



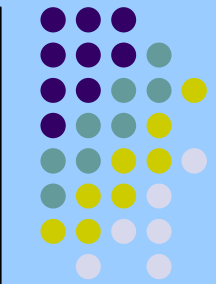
- Consider Table *t* with column set *C*. Nesting on column *X* is defined as:

  Any two tuples with the same values for

  (*C* – *X*) will be combined to one tuple

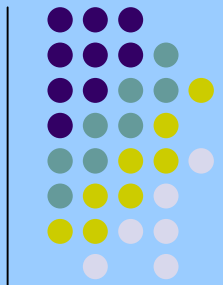
- **Observation: We need to nest only on key columns**
- Advantages of NeT
  - NeT removes redundancy if relation is not in 4NF
  - NeT provides more intuitive XML schemas with less redundancy

NeT: Experimentation



Test Set	#attr	#tuples	ValueRatio	#nested attributes	Time (sec)
Balloons1	5	16	3.64	3	1.08
Hayes	6	132	1.52	1	1.01
Bupa	7	345	1	0	4.40
Balance	5	625	2.79	4	21.48
TA_Eval	6	110	1.24	5	24.83
Car	7	1728	15.53	6	469.47
Flare	13	365	1.67	4	6693.41

CoT: Constraint-based Translation



- Translating relational schema
- Idea:

  Use foreign key constraints and our knowledge of how to represent relationships to come up with “better” XML models.

CoT: Step 1

Professor

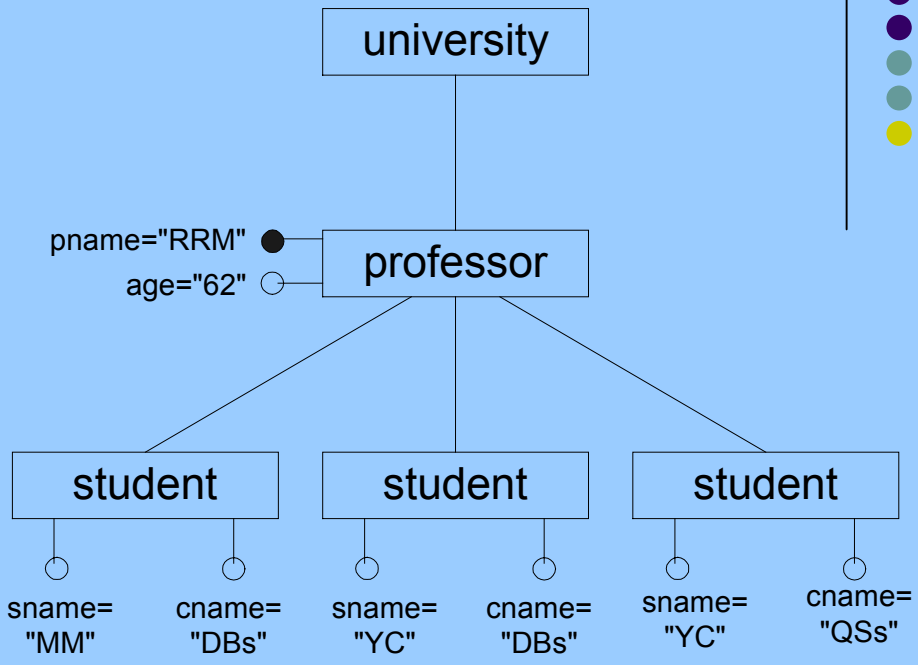
pname	age
RRM	62

Student

sname	advisor	cname
MM	RRM	DBs
YC	RRM	DBs
YC	RRM	QSs

Student (advisor) references Professor (pname)

ValueRatio=11/8



University → university (Professor*)

Professor → professor(@pname, @age, Student*)

Student → student (@sname, @cname)

(University, Professor, <@pname>)

(Professor, Student, <@sname, @cname>)

CoT: Step 2

University → university (Professor*, Course*)

Professor → professor (@pname, @age, Student*)

Student → student (@sname, @CRef)

Course → course (@cname, @since, @CID)

(University, Professor, <@pname>)
(University, Course, <@cname>)
(Professor, Student, <@sname, @Cref>)
CRef::IDREF references (Course)

Professor

pname	age
RRM	62

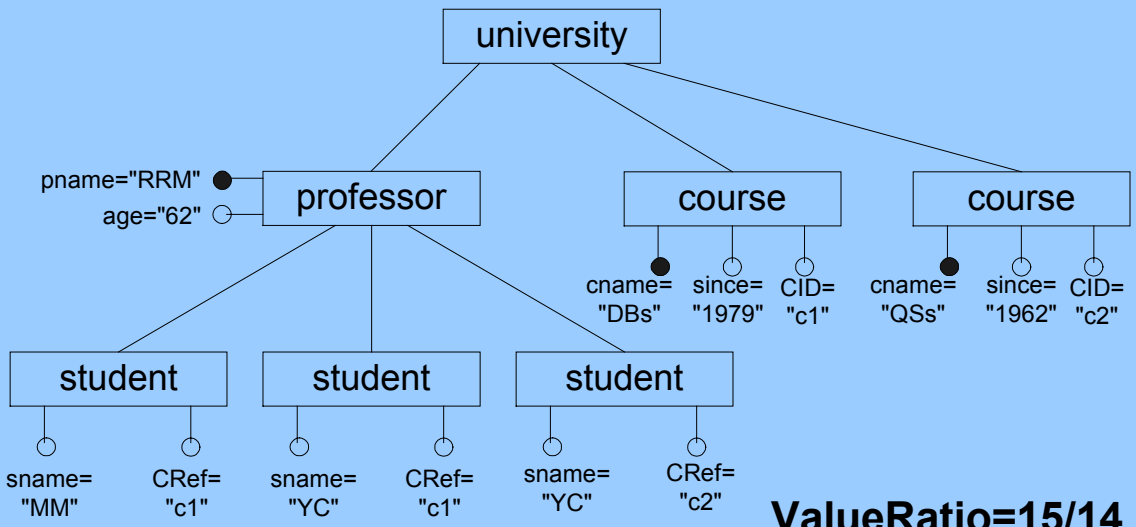
Course

cname	since
DBs	1979
QSs	1962

Student

sname	advisor	cname
MM	RRM	DBs
YC	RRM	DBs
YC	RRM	QSs

Student (advisor) references Professor (pname)
Student (cname) references Course (cname)

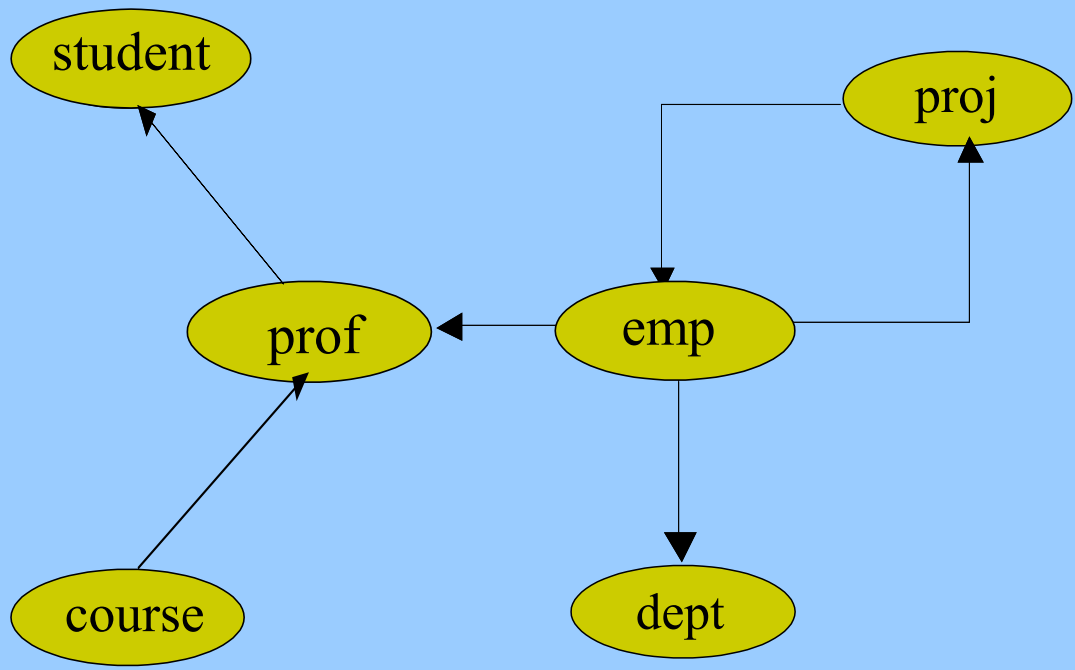


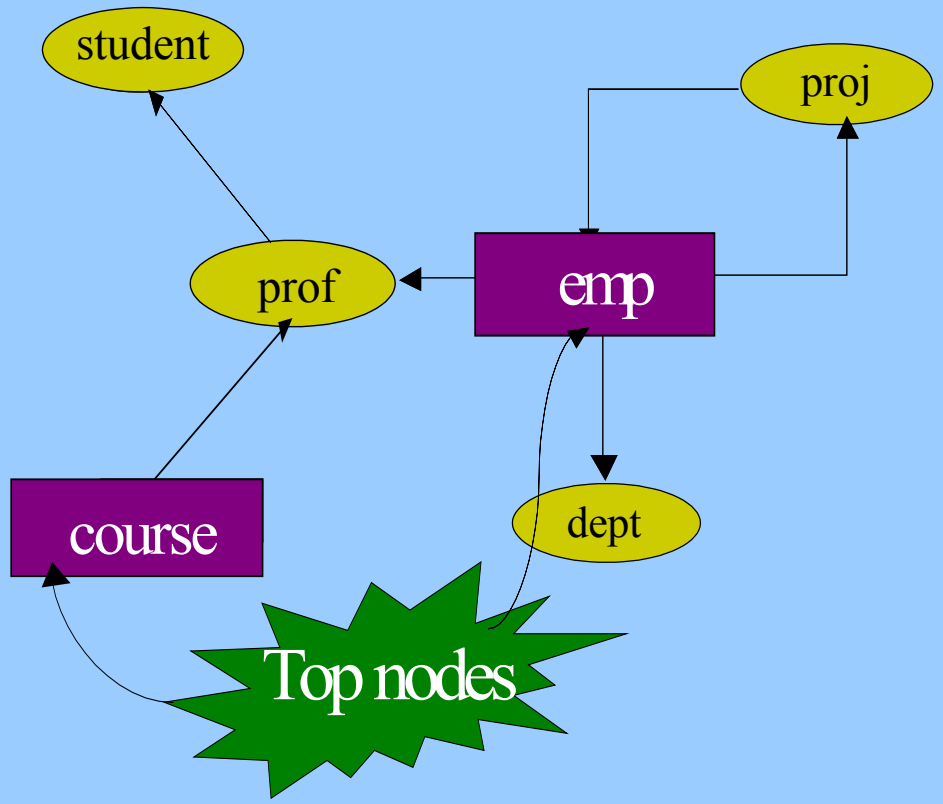
ValueRatio=15/14

CoT: Example

Student (SID, name, advisor)
Emp (EID, name, projName)
Prof (EID, name, teach)
Course (CID, title, room)
Dept (dno, mgr)
Proj (pname, pmgr)

Student (advisor) references
Prof (EID)
Emp (projName) references
Proj (pname)
Prof (teach) references
Course (CID)
Prof (EID, name) references
Emp (EID, name)
Dept (mgr) references
Emp (EID)
Proj (pmgr) references
Emp (EID)





(University, Course, <@CID>)
(University, Emp, <@EID>)
(University, Prof, <@EmpRef>)
(University, Student, <@SID>)
(University, Proj, @pname)
(University, Dept, @dno)

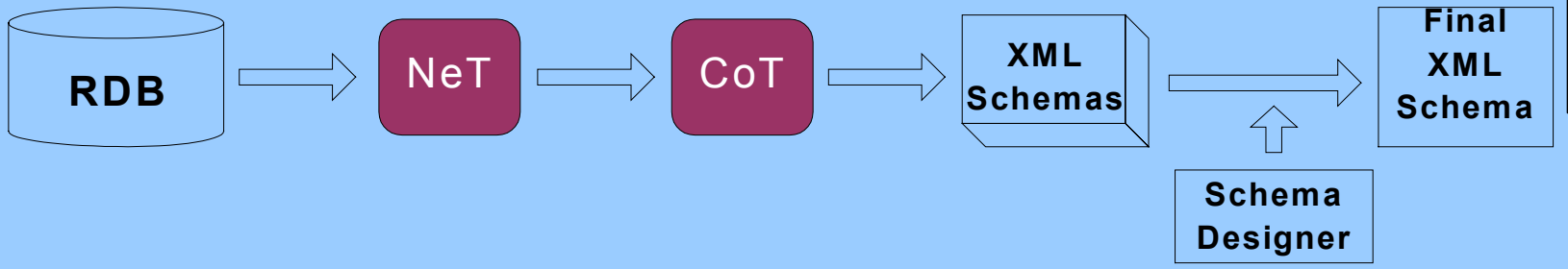
@EmpRef::IDREF references (Emp)
@ProjRef::IDREF references (Proj)

University → university (Course*, Emp*)
Course → course (@CID, @title, @room, Prof*)
Prof → prof (@EmpRef, Student*)
Student → student (@SID, @name)
Emp → emp (@EID, @name, @ProjRef, @EmpID, Dept*, Proj*)
Proj → proj (@pname, @ProjID)
Dept → dept (@dno)

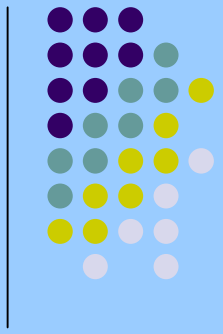
CoT Experimentation

- Ran on TPC-H data
- Value ratio > 100/88 (size decreased by more than 12%)

Conclusions



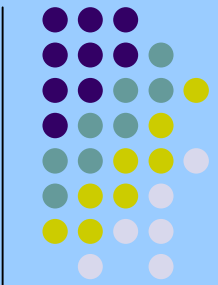
- We obtained “good” XML schemas from relational schemas
- Constraints are maintained
- Redundancies are decreased
- Most relationships can be navigated using path expressions.
- Minimum user interaction



Storing XML data in relational databases

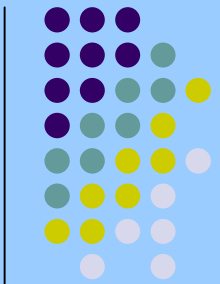
Murali Mani, Antonio Badia

Options for storing XML data



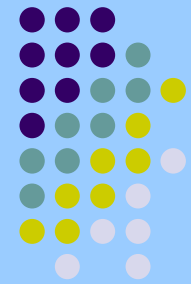
- Store in relational databases
  - Relational databases are robust and efficient (IBM XML Extender, Oracle, MS SQL Server)
- Store in native XML databases
  - More efficient for XML than relational databases (Natix, eXist, Tamino)
- Store in a combination of both
  - Structured portion of XML data in relational database, and unstructured portion in native XML store.

Related Work



- STORED
  - No Schema
  - Use data mining techniques to find structured and frequent patterns
  - These are stored in relational DB, others in semi-structured overflow store
  - Drawback: Requires integration of relational DB and semi-structured store
- Storing paths
  - One relation for storing nodes, one for storing edges.
  - Drawback: Type information is lost.

Type-based relational storage

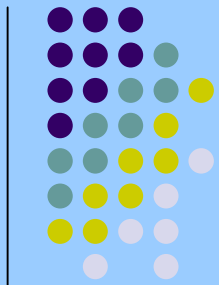


- Jayavel Shanmughasundaram
  - Several key ideas such as schema simplification, inlining, handling recursion
- LegoDB
  - Use the query workload to come up with an “efficient” relational schema.

Main features

- The entire XML document is shredded and stored in a relational database.
- All semantic constraints in XML schema are not captured in relational schema.
- We do not discuss how operations on XML are translated to SQL.

Why not capture all constraints?



A → a (@b, ((@c, D)* | (@e, F)*))

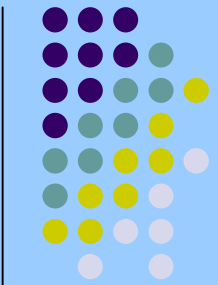
(Root, A, <@b>)

A (@b, @c, @e)
D (@aRef)
F (@aRef)

Semantic constraints lost

- if D refers to an A, then the corresponding @c should be non-null
- if F refers to an A, then the corresponding @e should be non-null

NF2 representation of regular tree grammars



Eliminate union, "|"

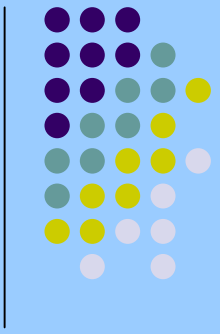
paper → (@ptitle, @journal | @conference)

paper → (@ptitle, @journal)

paper → (@ptitle, @conference) **(or)**

paper → (@ptitle, @journal, @conference)

Schema Simplification



$A \rightarrow a$ (@d, (B, C, B)*)

$A \rightarrow a$ (@d [1, 1], (B [2, 2], C [1, 1]) [0, *])
$A \rightarrow a$ (@d [1, 1], B [2, 2] [0, *], C [1, 1] [0, *])
$A \rightarrow a$ (@d [1, 1], B [0, *], C [0, *])

$A \rightarrow a$ (@d, B*, C*)

Semantic information lost
- The number of B's is two times the number of C's
- for every C, there is a B that occurs before it, and one that occurs after it.

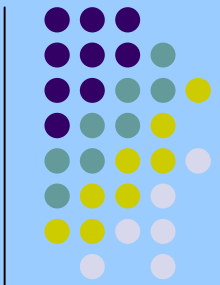
Inlining

Conf → conf (@ctitle, @date, Venue)
Venue → venue (@city, @country)

Conf → conf (@ctitle, @date, @city, @country)

Why inlining?
- Lesser joins, hence more efficient

Mapping Collection Types

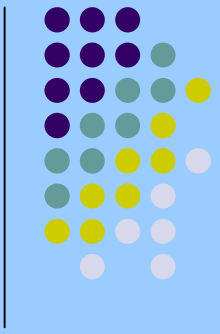


Conf → conf (@ctitle, @date, Paper*)
Paper → paper (@ptitle, @author)

Conf (@ctitle, @date)
Paper (@ptitle, @author, @confRef)

- Separate relation with foreign key for every collection type

IDREF attribute



Person → person (@name, @zip, Review*)
Book → book (@ISBN, @btitle, @BID)
Paper → paper (@ptitle, @journal, @PID)
Review → review (@article, @rating)

@article::IDREF references (Book | Paper)

Person (@name, @zip)
Book (@ISBN, @btitle, @BID)
Paper (@ptitle, @journal, @PID)
Review (@personRef, @bookRef, @paperRef, @rating)

Recursion using ?

A $\rightarrow$ a (@d, A?)

A (@d, ARef)
ARef refers to the child of A, and can be null

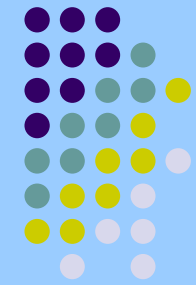
A $\rightarrow$ a (@d, ARef)
ARef refers to the parent of A and can be null.

Recursion using *

$A \rightarrow a$ (@d, A*)

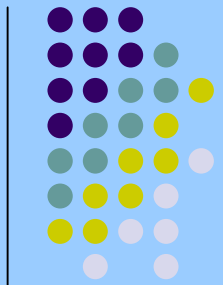
A (@d, ARef)
ARef refers to the parent of A, and can be null

Recursion – General technique



- For every cycle, we must have a separate relation
- Algorithm
  - For every strongly connected component, define a separate relation for one of the types.
  - In a strongly connected component, if there is a type which can be children of multiple types, then define a separate relation for that type.

Capturing Order in the Document



- Through order attributes
- Corresponding to each type in XML schema, say A, we have an associated order attribute, say aOrder

Conf → conf (@ctitle, @date, Venue)
Venue → venue (@city, @country)

Conf → conf (@ctitle, @date, @confOrder, @city, @country, @venueOrder)

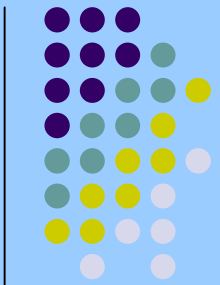
Conclusions

- XML schema with no recursion can be translated to relational schema with no nulls.
- XML schema with recursion cannot be translated to relational schema with no nulls.
- If recursion, separate relation needed for every cycle.
- All semantic constraints in XML cannot be captured in relational schema.
- XML resulting from CoT can be translated to the original relational schema; all semantic constraints are maintained.

Open Problems

- Standards specification: What structural and constraint specification schemes for XML are needed for database applications?
- XML used for text/document publishing: Keyword Search in XML documents
- Storing data consisting of structured and unstructured portions: integrating relational and XML stores.

Open Problem (contd…)



- Translating operations in XML model to underlying sources (relational)
  - Use annotated schema (MS SQL Server)
  - Use implicit annotations (LegoDB)
  - Query minimization: When we do automatic translation, we might perform unnecessary joins?