XML Normal Form

Learning Objectives

- XML Functional Dependencies
- Definition of the XML normal form
- Intuition behind XML normal form
- Some examples
- Normalization algorithm

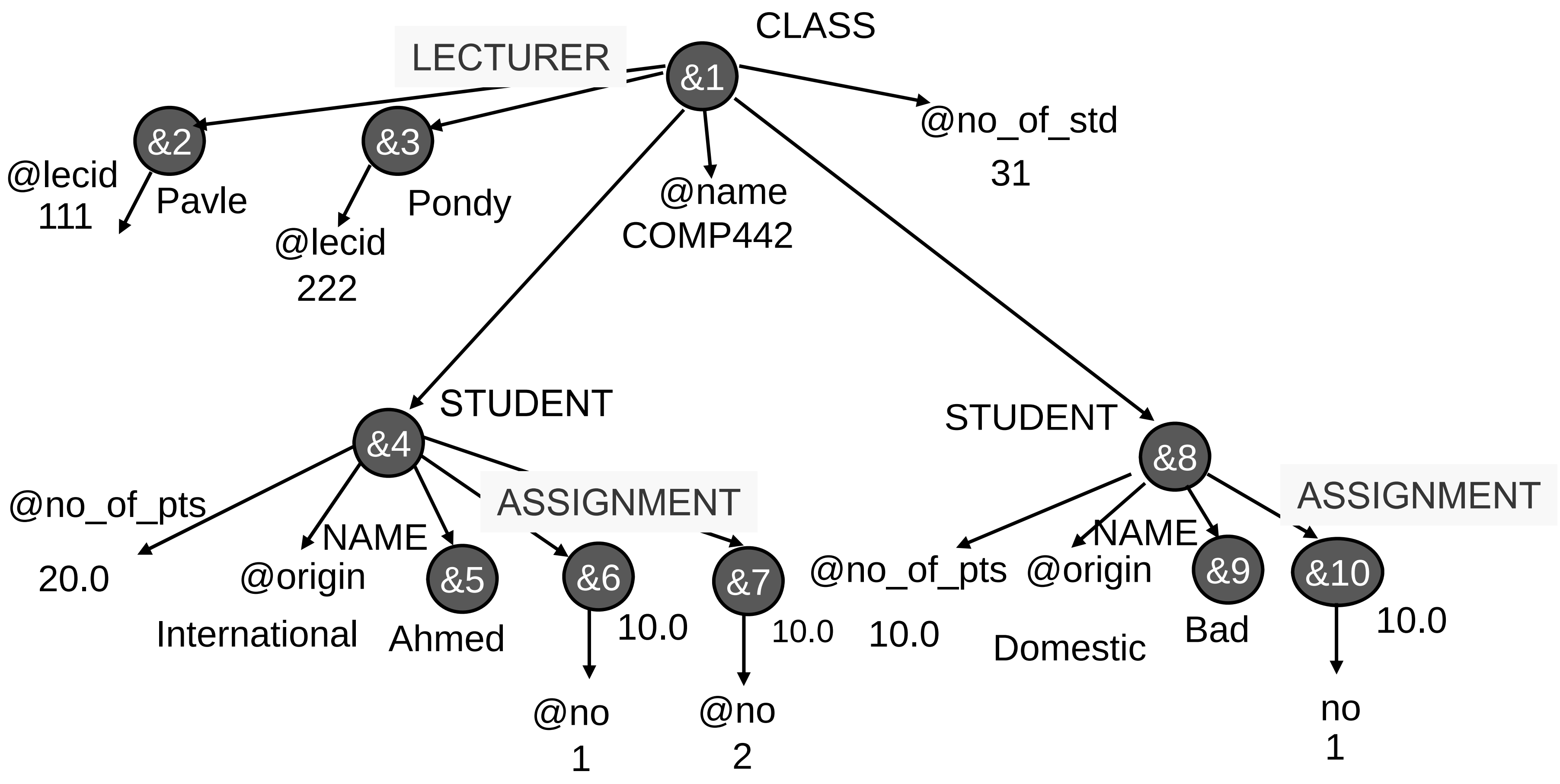
Why?

- Efficient use of native XML databases depends on the quality of documents’ meta data – a DTD or a Schema
- DTDs of a good design prohibit data redundancy in XML documents, whereas bad designed DTDs allow data redundancy
- Data redundancy cause update anomalies, which lead to an inefficient database use
- Quality of a DTD design is expressed by its normal form
- Like in relational databases, the XML normal form (XNF) is defined using functional dependencies
- Functional dependencies also make foundation of the normalization algorithm

Approach

- Defining XML normal form strongly relies on functional dependencies
- Defining XML functional dependencies strongly relies on tree tuples
- Work with tree tuples to eliminate possible redundancy
  - *M. Arenas, L. Lipkin: A Normal Form for XML Documents, ACM Transactions on Database Systems, Vol 29, No 1, March 2004, pp 195-232*

The NewClass XML Tree



Tabular Rep of NewClass Tree Tuples

path(*D*)	t_1	t_4	t_5
CLASS	&1	&1	&1
CLASS.@name	COMP442	COMP442	COMP442
CLASS.@no of std	31	31	31
CLASS.LECTURER	&2	&2	&3
CLASS.LECTURER.@lecid	111	111	222
CLASS.LECTURER.S	Pavle	Pavle	Pondy
CLASS.AUDITOR	$\perp$	$\perp$	$\perp$
CLASS.AUDITOR.S	$\perp$	$\perp$	$\perp$

Tabular Rep of Class Tree Tuples

path(*D*)	t_1	t_4	t_5
CLASS.STUDENT	&4	&4	&8
CLASS.STUDENT.@no_of_pts	20.0	20.0	0.0
CLASS.STUDENT@origin	International	International	Domestic
CLASS.STUDENT.NAME	&5	&5	&9
CLASS.STUDENT.NAME.S	Ahmed	Ahmed	Bad
CLASS.STUDENT.ASSIGNMENT	&6	&7	&10
CLASS.STUDENT.ASSIGNMENT.S	10.0	10.0	10.0
CLASS.STUDENT.ASSIGNMENT.@no	1	2	1

XML Functional Dependency

- We define the XML functional dependency using the set of paths of a DTD and check its satisfaction by an XML document using maximal tree tuples
- For a DTD D a functional dependency (fd) over D is an expression of the form

  $$S_1 \rightarrow S_2,$$

  where S_1 and S_2 are non-empty subsets of $paths(D)$
- The set of all fds over D will be denoted by $F(D)$

Understanding a Functional Dependency

- For $S \subseteq paths(D)$, and $t_1, t_2 \in \tau(D)$, $t_1.S = t_2.S$ means $t_1.p = t_2.p$ for all $p \in S$
- Furthermore, $t_1.S \neq \perp$ means $t_1.p \neq \perp$ for all $p \in S$
- If $S_1 \rightarrow S_2 \in F(D)$ and T is an XML tree such that $T \lhd D$ and $S_1 \cup S_2 \subseteq paths(T)$, tree T satisfies $S_1 \rightarrow S_2$ if

  $$(\forall t_1, t_2 \in tuples_D(T))$$

  $$((t_1.S_1 = t_2.S_1 \wedge t_1.S_1 \neq \perp) \Rightarrow t_1.S_2 = t_2.S_2)$$

- If t_1, t_2 satisfy $S_1 \rightarrow S_2$ then either both $t_1.S_2$ and $t_2.S_2$ are null or both $t_1.S_2$ and $t_2.S_2$ are not null

FDs Satisfied in The NewClass Tree

- Each lecturer has a unique lecturer *id* (`@lecid` is a key of `LECTURER`)

  fd_1: `CLASS.LACTURER.@lecid→CLASS.LECTURER`

- Each student has a unique name (`NAME.S` is a key of `STUDENT`)

  fd_2: `CLASS.STUDENT.NAME.S→CLASS.STUDENT`

  fd_3: `CLASS.STUDENT.NAME.S→`
  `CLASS.STUDENT.@origin`

  fd_4: `CLASS.SUDENT.NAME.S→`
  `CLASS.STUDENT.@no_of_pts`

FDs Satisfied in The NewClass Tree

- Each student's assignment has a different number

  fd_5: {`CLASS.STUDENT`,
  `CLASS.STUDENT.ASSIGNMENT.@no`}→
  `CLASS.STUDENT.ASSIGNMENT`

  fd_6: {`CLASS.STUDENT`,
  `CLASS.STUDENT.ASSIGNMENT.@no`}→
  `CLASS.STUDENT.ASSIGNMENT.S`
- Note,
  - `CLASS.STUDENT` determines the context
  - `CLASS.STUDENT.ASSIGNMENT` determines the scope,
  - whereas `@no` values have to identify `CLASS.STUDENT.ASSIGNMENT` node identifiers, or `CLASS.STUDENT.ASSIGNMENT.S` values

Testing Satisfaction of fd_6

- S_1 = {CLASS.STUDENT,
  CLASS.STUDENT.ASSIGNMENT.@no}
  - t_1(CLASS.STUDENT) = &4,
  - t_1(CLASS.STUDENT.ASSIGNMENT.@no) = 1,
  - t_4(CLASS.STUDENT) = &4,
  - t_4(CLASS.STUDENT.ASSIGNMENT.@no) = 2,
  - t_5(CLASS.STUDENT) = &8,
  - t_5(CLASS.STUDENT.ASSIGNMENT.@no) = 1,

- So, $t_1.S_1 \neq t_4.S_1$ and $t_1.S_1 \neq t_5.S_1$ and $t_4.S_1 \neq t_5.S_1$, hence no indication of fd_6 being not satisfied

  fd_6: {CLASS.STUDENT, CLASS.STUDENT.ASSIGNMENT.@no}→
  CLASS.STUDENT.ASSIGNMENT.S

Testing Satisfaction of fd_6

- Suppose we infer a tree tuple t_x (having data about an assignment of student Ahmed) from the Class document

  t_x = (&1, COMP442, 31, &2, 111, Pavle, ⊥, ⊥, &4, 20.0, International, Ahmed, &11, **20.0, 1.0**)

  then:
  - t_1(CLASS.STUDENT) = &4,
  - t_1(CLASS.STUDENT.ASSIGNMENT.@no) = 1,
  - t_x(CLASS.STUDENT) = &4,
  - t_x(CLASS.STUDENT.ASSIGNMENT.@no) = 1,
- So, $t_1.S_1 = t_x.S_1$, but fd_6 is not satisfied, since $t_1.S_2 \neq t_x.S_2$ since
  - S_2 = {CLASS.STUDENT.ASSIGNMENT.S},
  - t_1(CLASS.STUDENT.ASSIGNMENT.S) = 10.0,
  - t_x(CLASS.STUDENT.ASSIGNMENT.S) = 20.0,

FDs not Satisfied in The Class Tree

- The Class tree does not satisfy functional dependencies
  {`CLASS.STUDENT.ASSIGNMENT.@no`}$\rightarrow$
  {`CLASS.STUDENT.ASSIGNMENT`}
  {`CLASS.STUDENT.ASSIGNMENT.@no`}$\rightarrow$
  {`CLASS.STUDENT.ASSIGNMENT.S`}
- Consider tuples t_1 and t_5:
  t_1(`CLASS.STUDENT.ASSIGNMENT.@no`) = `1`
  t_5(`CLASS.STUDENT.ASSIGNMENT.@no`) = `1`
  t_1(`CLASS.STUDENT.ASSIGNMENT`) = `&6`
  t_5(`CLASS.STUDENT.ASSIGNMENT`) = `&10`
- So, $t_1(S_1) = t_5(S_1)$ and $t_1(S_2) \neq t_5(S_2)$

More on Functional Dependencies

- Using tree tuple representation it is easy to combine node and value equality:
  - The node equality corresponds to equality between vertices,
  - The value equality corresponds to equality between strings
- Keys are naturally defined using functional dependencies
- E.g.
  - fd_1, fd_2, and fd_5 are key defining
- Also, using sets with more than one path leads to defining relative constraints (that are valid only within the context defined by a path)
- E.g.
  - fd_5 and fd_6: are relative constraints

Trivial Functional Dependencies

- A functional dependency *f* is trivial if it is implied solely by a DTD *D*, and not by other fds
- A DTD forces a number of trivial fds:
  - For each $p \in EPaths(D)$ and p' a prefix of p, follows $p \rightarrow p'$
    - e.g. `CLASS.STUDENT.ASSIGNMENT`→`CLASS.STUDENT`
    - (Follows from the fact that all vertices have unique identifiers and at most one parent)

More Trivial Functional Dependencies

- A DTD also forces some other trivial fds:
  - For each p, such that $p.@a \in \text{paths}(D)$, follows $p \rightarrow p.@a$
    - e.g. `CLASS.LECTURER→ CLASS.LECTURER.@lecid`
    - (Follows from the fact that each element cannot have two attributes with the same name)
  - If for every pair of tree tuples t_1, t_2 in a tree T, $t_1.S_1 = t_2.S_1$ implies they have a null value on some $p \in S_1$, then the fd $S_1 \rightarrow S_2$ is trivially satisfied (an example follows)

A Case With Trivial FDs

```
<!ELEMENT R ((A | B), C, D)>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
```

paths(*D*) = {R, R.A, R.B, R.C, R.D}

- For each tree *T* conforming to D and for each tree tuple *t* in *T:*
  - either R.A $= \perp$ AND R.B $\neq \perp$, or

    R.B $= \perp$ AND R.A $\neq \perp$
- So, for any *p* in *paths*(*D*), the functional dependencies

  {R.A, R.B}→*p*, is trivial

Inferring Functional Dependencies

- If an XML tree T satisfies a functional dependency f, that fact is denoted by $T \models f$
- An XML tree T satisfies a set of functional dependencies Σ, if it satisfies each fd f from Σ
- Given a DTD D, a set $\Sigma \subseteq F(D)$, and a $f \in F(D)$, we say that (D, Σ) implies f, denoted $(D, \Sigma) \vdash f$, if for any tree T such that $T \models D$ (T conforms to D) and $T \models \Sigma$, it is the case that $T \models f$
- The set of all functional dependencies that are implied by (D, Σ) is denoted $(D, \Sigma)^+$

Inferring Functional Dependencies

- Suppose the Class document satisfies the following functional dependency

  $\{\texttt{CLASS.LECTURER.@lecId}\} \rightarrow \{\texttt{CLASS.LECTURER}\}$
- Since the following functional dependency is trivially satisfied

  $\{\texttt{CLASS.LECTURER}\} \rightarrow \{\texttt{CLASS.LECTURER.S}\}$
- Then, according to the transitivity, it follows

  $\{\texttt{CLASS.LECTURER.@lecId}\} \rightarrow \{\texttt{CLASS.LECTURER.S}\}$

- $\{\texttt{CLASS.LECTURER.@lecId}\}$ is a key for `LECTURER`

XML FDs and Keys

- An XML functional dependency of the form

$$S \rightarrow p.e$$

  is a key generating functional dependency, since the path $p' = p.e$ on the right hand side has an element as $last(p')$, and vertices have unique identifiers
- So, from $S \rightarrow p.e$ follows S is a key for e
- E.g.

  {`CLASS.LECTURER.@lecId`}→{`CLASS.LECTURER`}

  and {`CLASS.LECTURER.@lecId`} is a key for `LECTURER` (within the scope of the whole document)

XML Normal Form

- Given a DTD D and $\Sigma \subseteq F(D)$, (D, Σ) is in XML normal form (XNF) iff for every nontrivial functional dependency f in $(D, \Sigma)^+$ of the form $S \to p.@a$ or $S \to p.S$, it is the case that $S \to p$ is in $(D, \Sigma)^+$

- If there exists a nontrivial functional dependency $S \to p.@a$ or $S \to p.S$, such that $S \to p$ is not in $(D, \Sigma)^+$ that functional dependency is anomalous, because it causes update anomalies
- An anomalous functional dependency is not desirable

XNF And Trivial FDs

- It should be noted that a functional dependency violating XNF has to be nontrivial
- Indeed, the trivial fds
  - $p.@a \rightarrow p.@a$ and
  - $p.S \rightarrow p.S$

  are always in $(D, \Sigma)^+$, but often $p.@a \rightarrow p$ is not, which does not necessarily represent a bad design

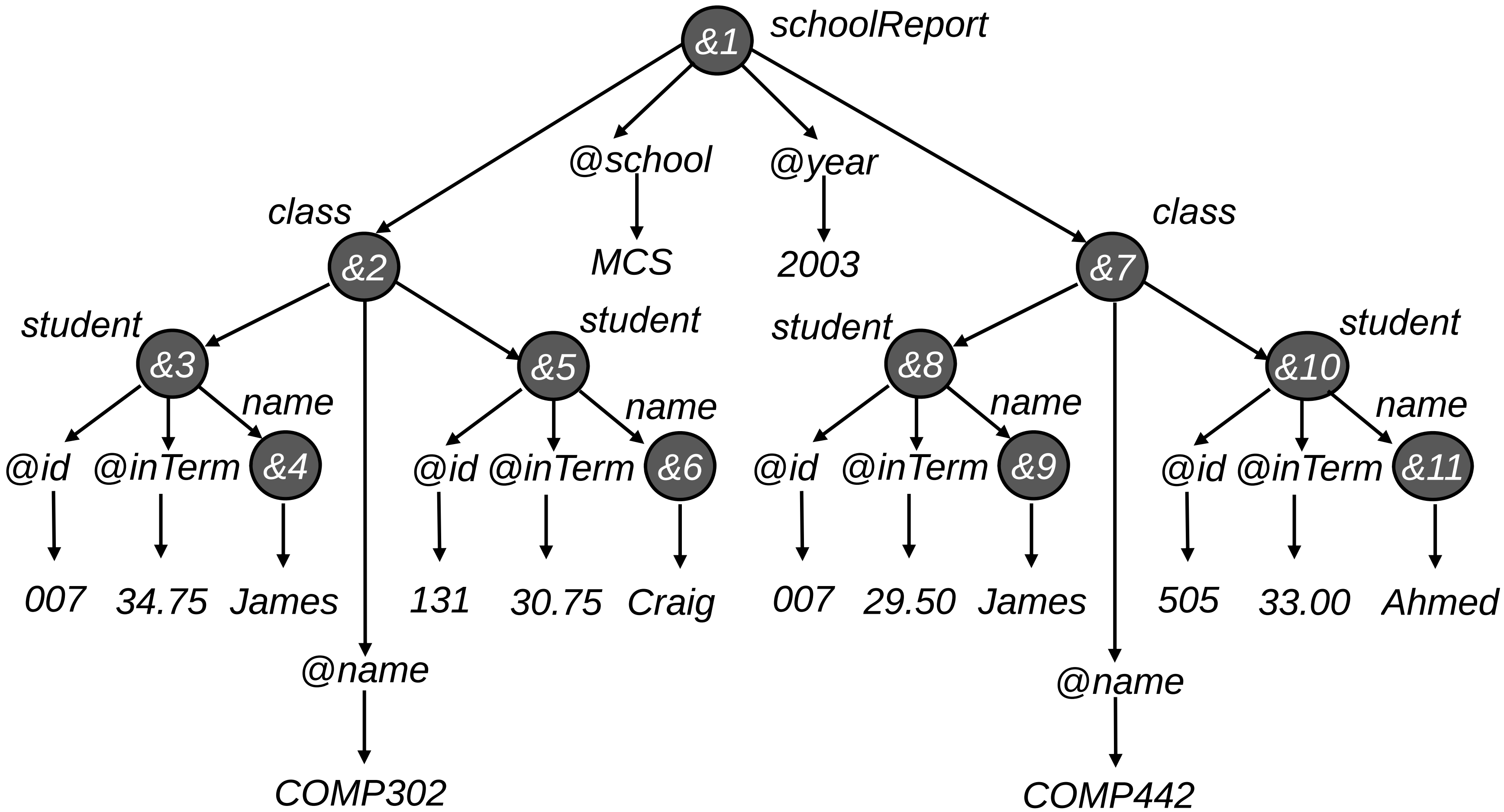
schoolReport.dtd

```
<!ELEMENT schoolReport (class*)>
<!ELEMENT class (student*)>
<!ELEMENT student (name)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST schoolReport school CDATA
#REQUIRED year CDATA #REQUIRED>
<!ATTLIST class name CDATA #REQUIRED>
<!ATTLIST student id CDATA #REQUIRED
inTerm CDATA #REQUIRED>
```

schoolReport.xml

```
<schoolReport school=“MCS” year=“2003”>
  <class name=“comp302”>
      <student id=“007” inTerm=“34.75”>
            <name>James</name>
      </student>
      <student id=“131” inTerm=“30.75”>
            <name>Craig</name>
      </student>
  </class>
  <class name=“comp442”>
      <student id=“007” inTerm=“29.50”>
            <name>James</name>
      </student>
      <student id=“505” inTerm=“33.00”>
            <name>Ahmed</name>
      </student>
  </class>
</schoolReport>
```

schoolReport.xml



FD Satisfied by The Report Tree

- Consider the functional dependency

  `schoolReport.class.student.@id`→ `schoolReport.class.student.name.S`

  and the tree tuples:

  t_1 = (`&1, MCS, 2003, &2, COMP302, &3, 007, 34.75, &4, James`)

  t_2 = (`&1, MCS, 2003, &7, COMP442, &8, 007, 29.50, &9, James`)

- Since:
  - t_1(`schoolReport.class.student.@id`) = `007`
  - t_2(`schoolReport.class.student.@id`) = `007`
  - t_1(`schoolReport.class.student.name.S`) = `James`
  - t_2(`schoolReport.class.student .name.S`) = `James`

  the functional dependency is satisfied

FDs Satisfied by The Report Tree

- So, the functional dependency

  f_1: `schoolReport.class.student.@id`→
  `schoolReport.class.student.name.S`

  is satisfied in the Report document
- Now, let us check whether the functional dependency

  f_2: `schoolReport.class.student.@id`→
  `schoolReport.class.student.name`

  is also satisfied
- Since:
  - t_1(`schoolReport.class.student.@id`) = `007`
  - t_2(`schoolReport.class.student.@id`) = `007`
  - t_1(`schoolReport.class.student.name`) = `&4`
  - t_2(`schoolReport.class.student.name`) = `&9`

  the functional dependency f_2 is not satisfied

Data Redundancy In `schoolReport.xml`

- The fact that the fd f_1 is satisfied in the Report document and the fd f_2 is not, points to data redundancy
- Really, a student's name appears in each class where the student has enrolled
- The argument would be more evident if we had other student's personal data, like surname, address, phone,…
- The *`scoolReport.dtd`* is not in XNF and its instance has data redundancy
- Recall modification update anomaly:
  - Each change of the student's address has to be updated in each class

Normal Form of The Report Document

- The fact that the fd f_1 is satisfied in the Report document and the fd f_2 is not, points to data redundancy
- Really, a student's name appears in each class where the student has enrolled
- The argument would be more evident if we had other student's personal data, like surname, address, phone,…
- The Report document is not in the XNF
- Recall modification update anomaly:
  - Each change of the student's address has to be updated in each class

Bringing Report Document into XNF

- To eliminate redundancy, we separate student's personal data and data related to its performance in a particular course
- To achieve that we introduce a new `student` element to store personal data and leave only student's `id` and `inTerm` data within each `class` in the `class_student` element (which is renamed old `student` element)
- This way we avoid redundancy, since student's personal data appear in only one place, and when needed will be modified only there
- This way, the document is brought into XNF and the information in the document is preserved

Improved schoolReport.dtd

```
<!ELEMENT schoolReport (classes, students)>
<!ELEMENT classes (class*)>
<!ELEMENT class (class_student*)>
<!ELEMENT class_student EMPTY>
<!ELEMENT students (student*)>
<!ELEMENT student (name)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST schoolReport school CDATA #REQUIRED
year CDATA #REQUIRED>
<!ATTLIST class name CDATA #REQUIRED>
<!ATTLIST student id ID #REQUIRED>
<!ATTLIST class_student id IDREF #REQUIRED
inTerm  CDATA #REQUIRED>
```

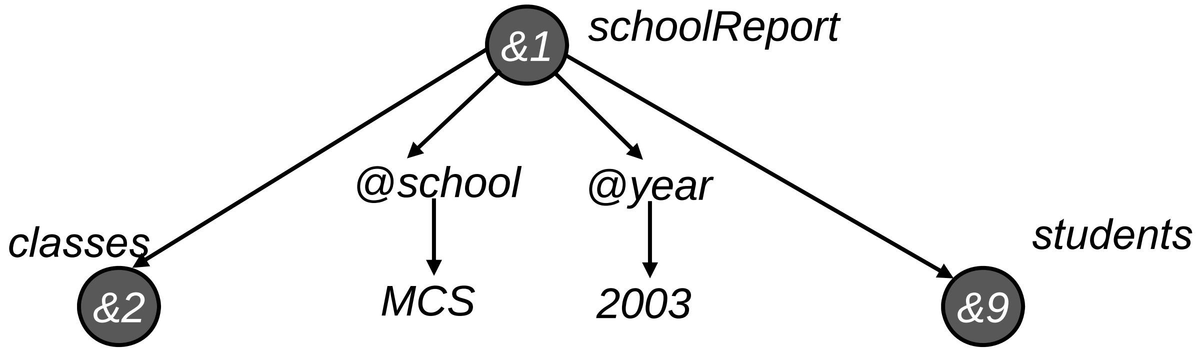
Improved schoolReport.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE schoolReport SYSTEM "schoolReportDTD.dtd">
<schoolReport school="MCS" year="2003">
  <classes>
    <class name="comp302">
     <class_student id="007" inTerm="34.75"/>
     <class_ student id="131" inTerm="30.75"/>
    </class>
    <class name="comp442">
     <class_student id="007" inTerm="29.50"/>
     <class_student id="505" inTerm="33.00"/>
    </class>
  </classes>
```

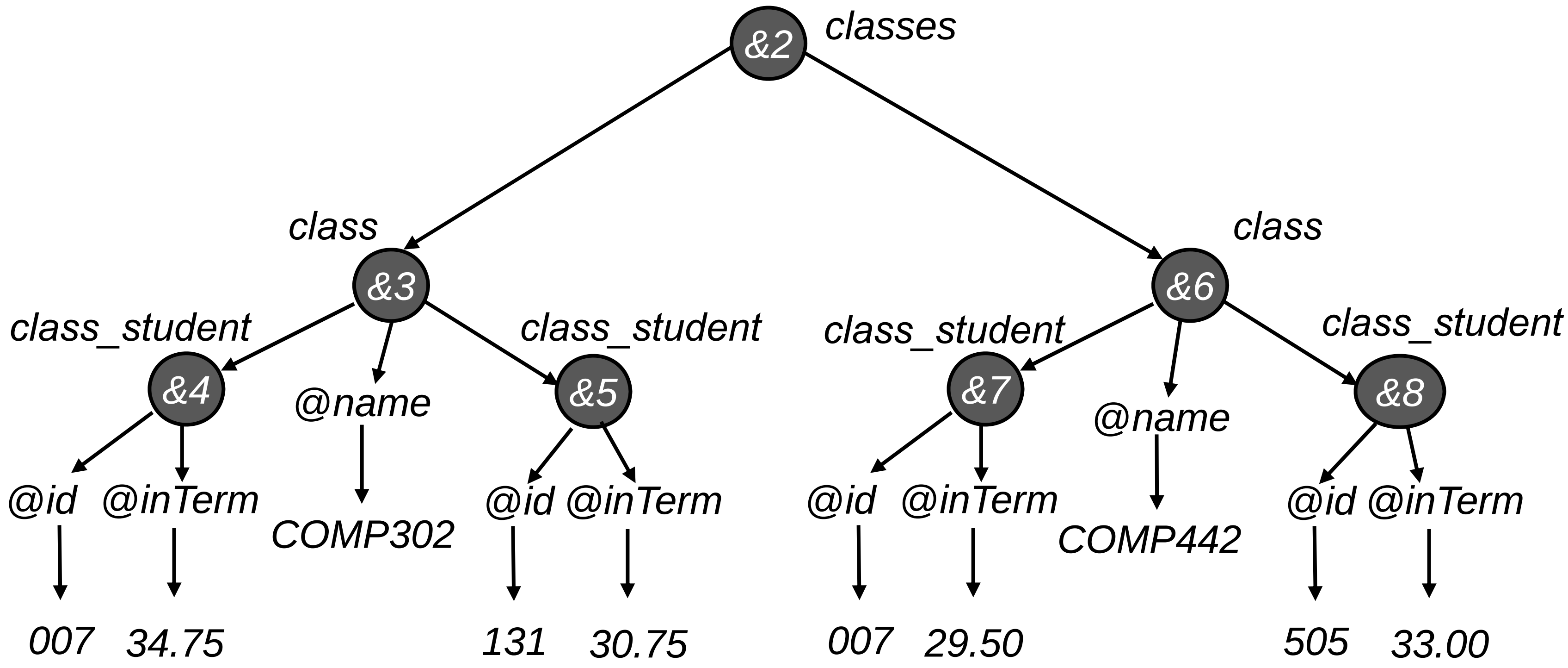
Improved `schoolReport.xml`

```
  <students>
     <student id="007">
          <name>James</name>
     </student>
     <student id="131">
          <name>Craig</name>
     </student>
     <student id="505">
          <name>Ahmed</name>
     </student>
  </students>
</schoolReport>
```

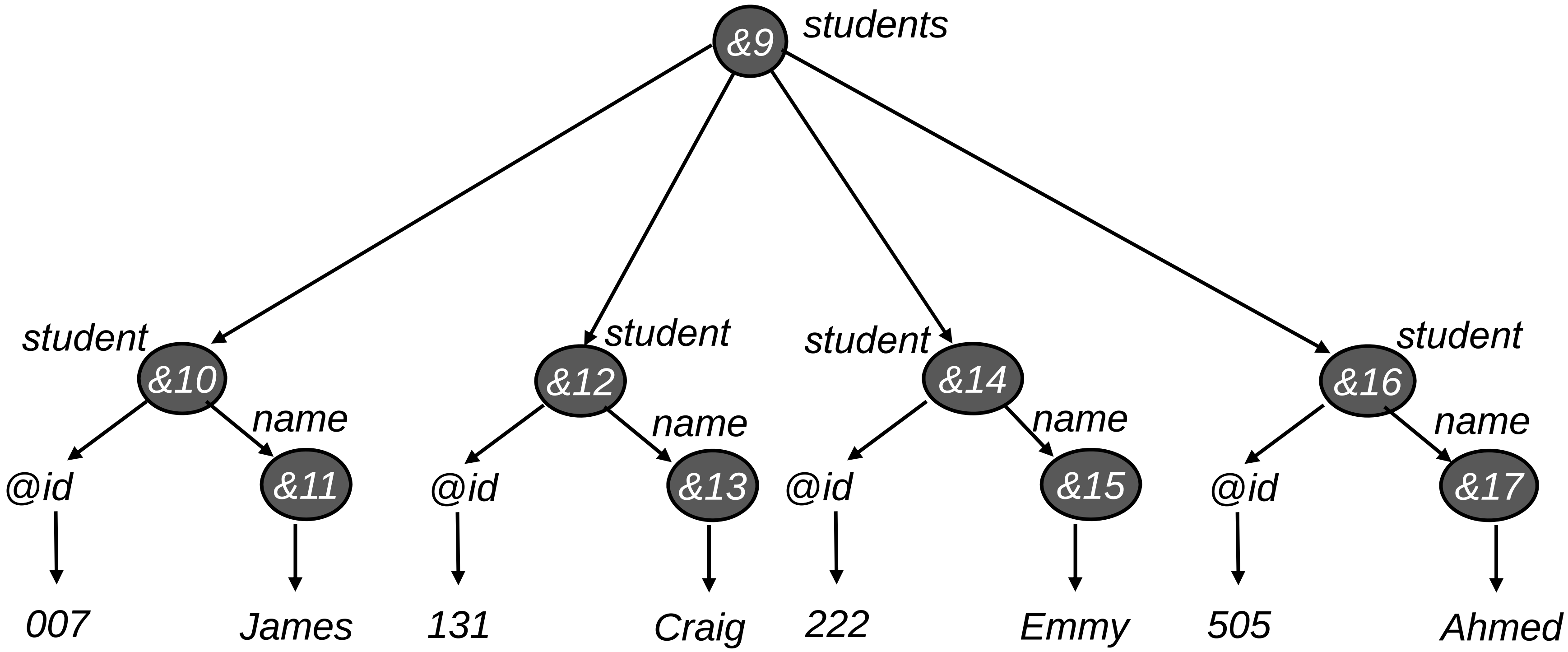
schoolReport.xml



schoolReport.xml



schoolReport.xml



FDs Satisfied by The Improved Report

- These are some of the functional dependencies satisfied by the improved Report document:

f_1: {`schoolReport.classes.class`, `schoolReport.classes.class.class_student.@id`}→ `schoolReport.classes.class.class_student.@inTerm`

f_2: {`schoolReport.classes.class`, `schoolReport.classes.class.class_student.@id`}→ `schoolReport.classes.class.class_student`

f_3: `schoolReport.students.student.@id`→ `schoolReport.students.student.name.S`

f_4: `schoolReport.students.student.@id`→ `schoolReport.students.student`

- The improved Report document is in XNF

Reasoning About Anomalous FD's

- If both $S \rightarrow p.@a$ (or $S \rightarrow p.S$) and $S \rightarrow p$ functional dependencies are satisfied in an XML document then there are no two different vertices v_1 and v_2 such that $lab(v_1) = lab(v_2) = e$ and $e = last(p)$ with the same value for $p.@a$ or $p.S$ (hence, no data redundancy)

- Otherwise, if a functional dependency $S \rightarrow p.@a$ (or $S \rightarrow p.S$) is satisfied in an XML document, but $S \rightarrow p$ is not, then there may be two different vertices v_1 and v_2 such that $lab(v_1) = lab(v_2) = e$ and $e = last(p)$ with the same value for $p.@a$ or $p.S$ (hence, data redundancy)

Two Normalization Procedures

- Apply the following two procedures to improve the design of XML documents:
  - Creating new elements or/*and*
  - Moving attributes
- This way we removed a “bad” functional dependency from the document
- All identified “bad” functional dependencies were consequences of a wrong placement of data within nested elements

Two Normalization Procedures

- If an element B is nested within an element A, then an attribute *@a* or S data of the element B should carry information only about the properties of the association between A and B, otherwise it incurs redundancy:
  - If an attribute *@a* or S data carry information about a property that belongs solely to element B, we create a new element that will contain only specific information about the element B (including the attribute *@a* or S data) and reference that new element from within old element B
  - If an attribute *@a* or S data carry information about a property that belongs solely to the element A, we move it to the element A, since that is the place where it really belongs to

Normalization Algorithm

- Given a DTD D and a set Σ of functional dependencies:
  1. If (D, Σ) is in XNF, return
  2. Otherwise, find an anomalous fd and use one of the two procedures to modify D and eliminate the anomalous fd from Σ
  3. Continue the step 3 until a new (D, Σ) is in XNF

Journal DTD

```
<!ELEMENT journal (volume*)>
<!ELEMENT volume (issue*)>
<!ELEMENT issue (paper*)>
<!ELEMENT paper (#PCDATA)>
<!ATTLIST journal name CDATA #REQUIRED>
<!ATTLIST volume no CDATA #REQUIRED>
<!ATTLIST issue no CDATA #REQUIRED>
<!ATTLIST paper year CDATA #REQUIRED>
```

A Journal Document

```
<journal name="ACM TODS">
  <volume no="29">
      <issue no="1">
            <paper year="2003">
                  Pavle, XML Databases, 1-18
            </paper>
            <paper year="2003">
                  Pondy, Mobile Agents, 19-36
            </paper>
      </issue>
      <issue no="2">
            <paper year="2003">
                  Lindsay, Concurrency, 37-55
            </paper>
      </issue>
  </volume>
```

A Journal Document

```
<volume no=“30”>
    <issue no=“1”>
          <paper year=“2004”>
                Neil, Predicates in Prolog, 1-
20
          </paper>
    </issue>
    <issue no=“1”>
          <paper year=“2004”>
                Ray, Regular Expressions, 1-20
          </paper>
    </issue>
</volume>
</journal>
```

FDs in The Journal Document

- The functional dependency

  f_1: `journal.volume`→
  `journal.volume.issue.paper.@year`

  is satisfied in the Journal document, since each paper in a volume has the same value of the year attribute
- But, the functional dependency

  `journal.volume`→
  `journal.volume.issue.paper`

  is not satisfied, since each `volume` can have several `issues` each having several `papers`
- The functional dependency f_1 is a bad one, hence the Journal document is not in XNF

Bringing Journal Document into XNF

- To eliminate redundancy, we move the `@year` attribute from `paper` element to `volume` element
- In fact, year is a property of volume, not of a paper
- This way we avoid redundancy, since volume's year data appear in only one place,
- This way, the document is brought into XNF and the information in the document is preserved

Improved Journal DTD

```
<!ELEMENT journal (volume*)>
<!ELEMENT volume (issue*)>
<!ELEMENT issue (paper*)>
<!ELEMENT paper (#PCDATA)>
<!ATTLIST journal name CDATA #REQUIRED>
<!ATTLIST volume no CDATA #REQUIRED year
CDATA #REQUIRED>
<!ATTLIST issue no CDATA #REQUIRED>
```

Improved Journal Document

```
<journal name="ACM TODS">
  <volume no="29" year="2003">
      <issue no="1">
            <paper>
                  Pavle, XML Databases, 1-18
            </paper>
            <paper>
                  Pondy, Mobile Agents, 19-36
            </paper>
      </issue>
      <issue no="2">
            <paper>
                  Lindsay, Concurrency, 37-55
            </paper>
      </issue>
  </volume>
```

Improved Journal Document

```
<volume no="30" year="2004">
    <issue no="1">
          <paper>
                Neil, Predicates in Prolog, 1-
20
          </paper>
    </issue>
    <issue no="1">
          <paper>
                Ray, Regular Expressions, 1-20
          </paper>
    </issue>
</volume>
</journal>
```

FDs of The Improved Journal Document

- The transformed Journal document does not satisfy the bad functional dependency

  f_1: `journal.volume` → `journal.volume.issue.paper.@year`

  but satisfies the trivial functional dependencies

  f_2: `journal.volume` → `journal.volume`

  f_3: `journal.volume` → `journal.volume.@year`

  and also a number of nontrivial ones, like

  f_4: `journal.volume.@year` → `journal.volume.@no`

  f_5: `journal.volume.@no` → `journal.volume.@year`

Summary

- It is desirable for an XML document stored in a Native XML database to be in XNF
- An XNF XML document has no data redundancy
- Documents that are not in XNF have data redundancy and exhibit update anomalies
- Data redundancy is recognized by “bad” anomalous functional dependencies
- Two operations to remove “bad’ functional dependencies are:
  - Creating a new element and
  - Moving an attribute (or S data element)

  both processes are information preserving