Chapter 7: XQuery

References:

- Scott Boag, Don Chamberlin, Mary F. Ferníndez, Daniela Florescu, Jonathan Robie, Jérôme Siméon (Eds.): XQuery 1.0: An XML Query Language. W3C Recommendation 23 January 2007. [http://www.w3.org/TR/xquery/]
- Wolfgang Lehner, Harald Schöning: XQuery. Grundlagen und fortgeschrittene Methoden. dpunkt.verlag, 2004, ISBN 3-89864-266-6, 290 Seiten. [http://www.xquery-buch.de/]
- Howard Katz (Ed.), Don Chamberlin, Denise Draper, Mary Fernández, Michael Kay, Jonathan Robie, Michael Rys, Jérôme Siméon, Jim Tivy, Philip Wadler: XQuery from the Experts. A Guide to the W3C XML Query Language. Pearson Education Inc., 2004, ISBN 0-321-18060-7, 484 pages.
- Jim Melton, Stephen Buxton: Querying XML: XQuery, XPath, and SQL/XML in Context. Morgan Kaufmann/Elsevier, 2006, ISBN 1-55860-711-0, 815 pages.
- Rudolf Jansen: XQuery: Eine praxisorientierte Einführung. Software & Support Verlag GmbH, 2004, ISBN 3-935042-65-5, 167 Seiten.



After completing this chapter, you should be able to:

- read and understand queries in XQuery.
- write queries to XML data in XQuery.
- enumerate and explain the clauses of FLWOR expressions.
- explain the use of direct and computed constructors.
- compare XQuery with SQL.



1. Introduction

2. Basic Syntax, Constructors

3. FLOWR-Expressions

4. Examples, Comparison with SQL

5. Prolog, Functions



- One can view XML as a data model, and every data model should have a query language.
- XPath permits to
 - ◇ select nodes in a given XML document, and to
 ◇ compute values from the data in a document,
 but it does not permit to generate new nodes.
- An XML query language should be able to produce new XML documents as result.

I.e. to transform given XML documents into documents that are differently structured, contain only a subset of the data, or additional derived data.



• Joins are very limited in XPath (semijoins) or can be formulated only procedurally (with for-loops).

Note that many powerful constructs such as for-loops got into XPath only in Version 2.0 during the development of XQuery. Although today, XPath is sufficient for quite a lot of queries, the original XPath was much more restricted.

• One cannot sort in XPath.

Sometimes, although the result set can be determined by a simple Path expression, one must use the more advanced FLWOR-expression (an XQuery construct, see below) only for the purpose of sorting.

• Of course, a good XML query language must be at least as powerful as SQL.



 The XML format is a common interface to a lot of different data sources (documents, relational databases, object repositories).

The data might be physically stored as XML, or might be only viable as XML via a middleware.

• An XML query language permits to combine data from different sources.

This integrating function of an XML query language is natural and important. While also SQL can be used in distributed databases, and there exist relational interfaces to non-relational data, this is much more vendor-dependent (and typically expensive).



• In December 1998, the W3C organized a workshop about query languages for XML.

[http://www.w3.org/TandS/QL/QL98/].

• There was a lot of research about query languages for semi-structured data models and XML in particular (e.g., Lorel, XQL, XML-QL, YATL, Quilt). See, e.g.: XML Query Langauges, Experiences and Exemplars.

See, e.g.: XML Query Langauges, Experiences and Exemplars. [http://www.w3.org/1999/09/ql/docs/xquery.html]

 The XML Query Working Group started in 1999 the work on a W3C standard XML query language. [http://www.w3.org/XML/Query]

History (2)

- XPath 1.0 and XSLT 1.0 became a W3C Recommendation in November 1999.
- While it seemed natural that XPath-like expressions should be used also in XQuery, the XQuery committee had quite different ideas for the exact details of syntax and semantics.

XPath 1.0 came from the document processing community, not from the database community. But having two similar languages that differed in important details was obviously not good. This lead to difficult negotiations and ultimately the development of XPath 2.0.



- XML Schema became a W3C recommendation in May 2001.
- Steps of the XQuery standardization:
 - ♦ First Working Draft: February 15, 2001.
 - ♦ Last Call Working Draft: November 12, 2003
 The last call period ended on February 15, 2004, Several under

The last call period ended on February 15, 2004. Several updates were published afterwards.

- ♦ W3C Candidate Recommendation: Nov. 3, 2005 Update: June 8, 2006
- ♦ W3C Proposed Recommendation: Nov. 21, 2006
- ♦ W3C Recommendation: January 23, 2007



- The following eight documents were developed together:
 - ♦ XQuery 1.0
 - ◊ XQueryX 1.0: XML Syntax for XQuery 1.0
 - ♦ XPath 2.0
 - ♦ XSLT 2.0
 - ◊ XQuery 1.0/XPath 2.0 Data Model
 - ♦ XQuery 1.0/XPath 2.0 Formal Semantics
 - ♦ XQuery 1.0/XPath 2.0 Functions and Operators
 - ♦ XSLT 2.0/XQuery 1.0 Serialization



- Related documents (Working Group Notes):
 - ♦ XML Query Use Cases

[http://www.w3.org/TR/xquery-use-cases/]

◊ XML Query (XQuery) Requirements

[http://www.w3.org/TR/xquery-requirements/]

- Extensions (Candidate Rec. May/Aug. 2008):
 - ♦ XQuery and XPath Full Text 1.0

[http://www.w3.org/TR/xpath-full-text-10/]

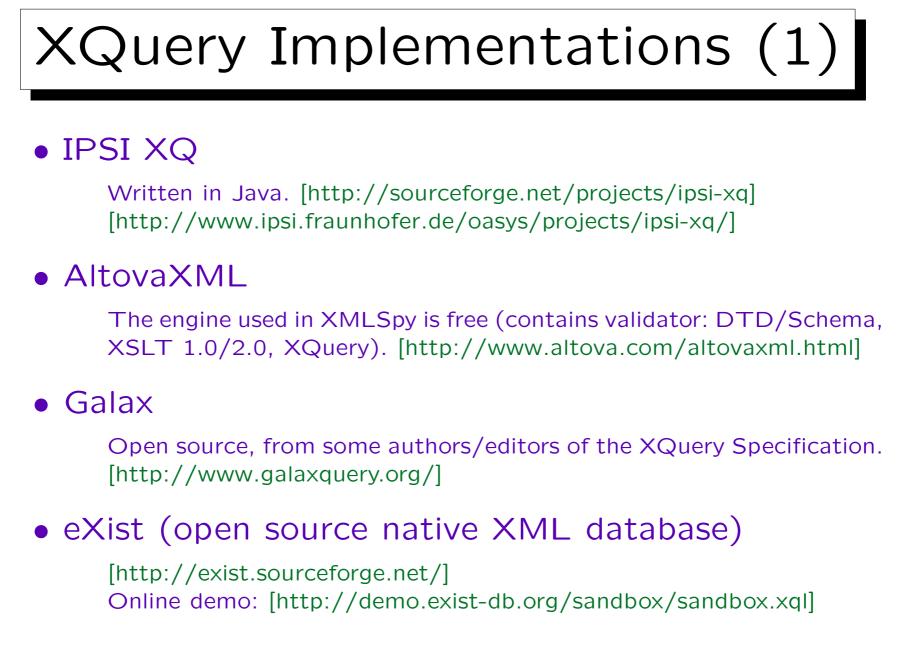
◊ XQuery Update Facility 1.0

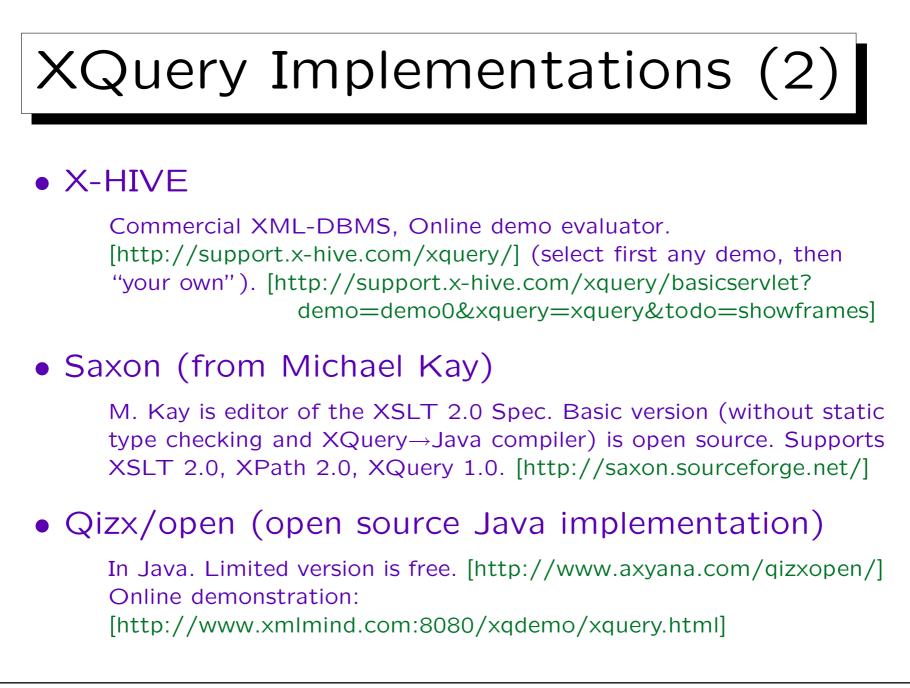
[http://www.w3.org/TR/xquery-update-10/]



- The two languages have overlapping, but not identical goals.
- XSLT was developed by the document processing community. Main use: rendering XML documents. Although it can also be used for selecting and restructuring data.
- XQuery is a database language.
- Databases store very many / very large documents: indexes and query optimization are important (the data does not fit completely into main memory).

The data is also more regularly structured (in most cases).





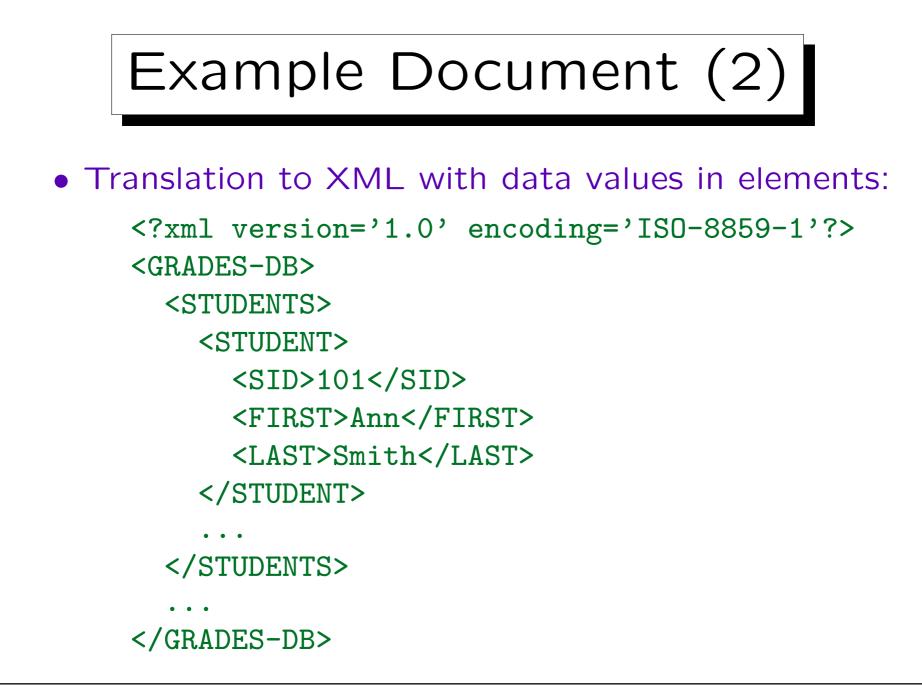
Example Document (1)

STUDENTS			
SID	FIRST	LAST	EMAIL
101	Ann	Smith	• • •
102	Michael	Jones	(null)
103	Richard	Turner	•••
104	Maria	Brown	• • •

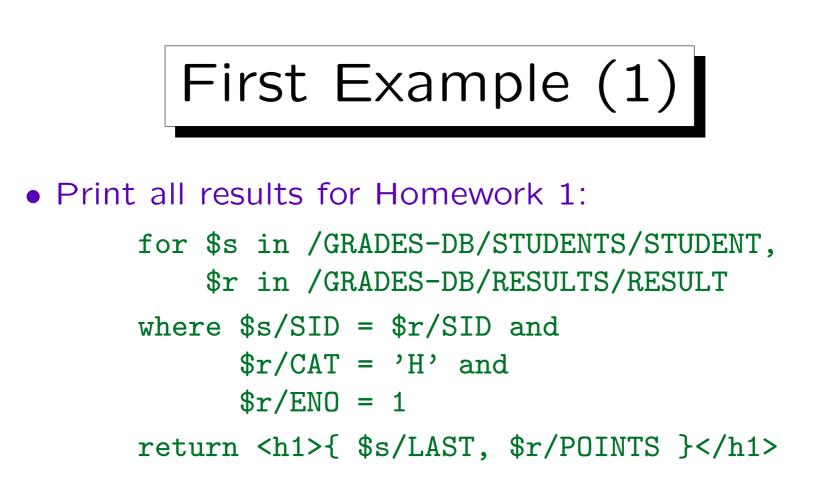
	EXERCISES		
CAT	<u>ENO</u>	TOPIC	MAXPT
Η	1	Rel. Algeb.	10
H	2	SQL	10
М	1	SQL	14

	RESULTS		
SID	CAT	<u>ENO</u>	POINTS
101	Η	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Stefan Brass: XML und Datenbanken

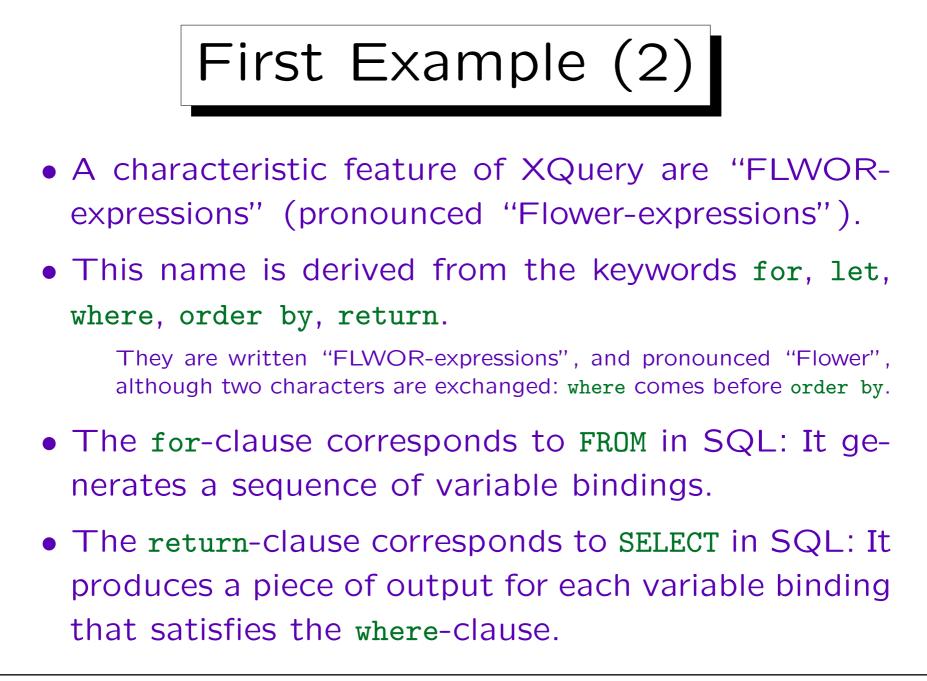


Stefan Brass: XML und Datenbanken



• Result:

<h1><LAST>Smith</LAST><POINTS>10</POINTS></h1><h1><LAST>Jones</LAST><POINTS>9</POINTS></h1><h1><LAST>Turner</LAST><POINTS>5</POINTS></h1>



First Example (3)

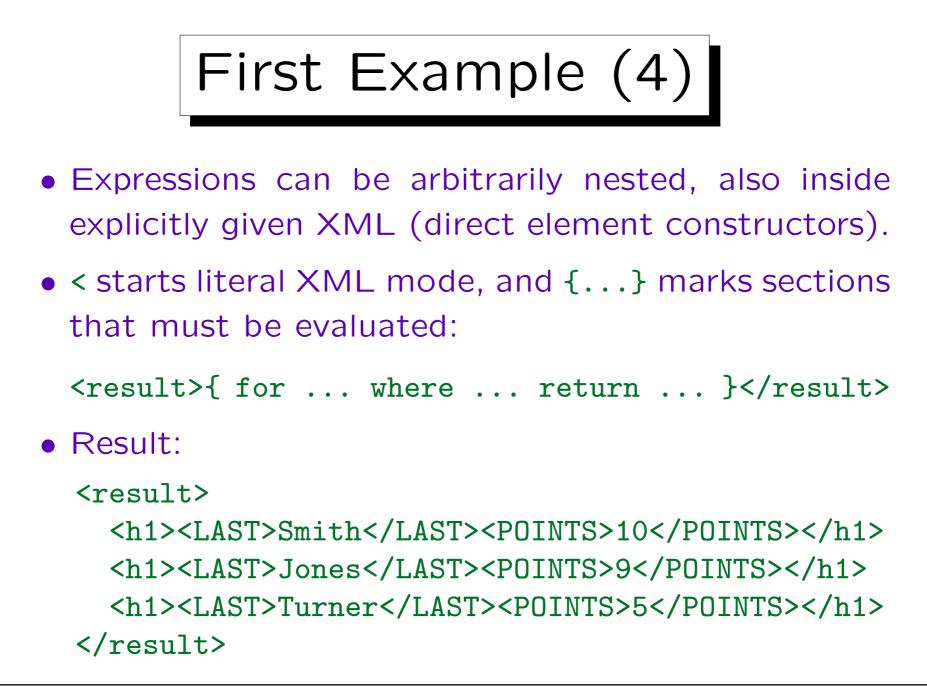
- Note that the order of the clauses in XQuery fits better with the evaluation sequence: Also in SQL, FROM is conceptually evaluated first, and SELECT last.
- In XQuery, all keywords are written in lower case.

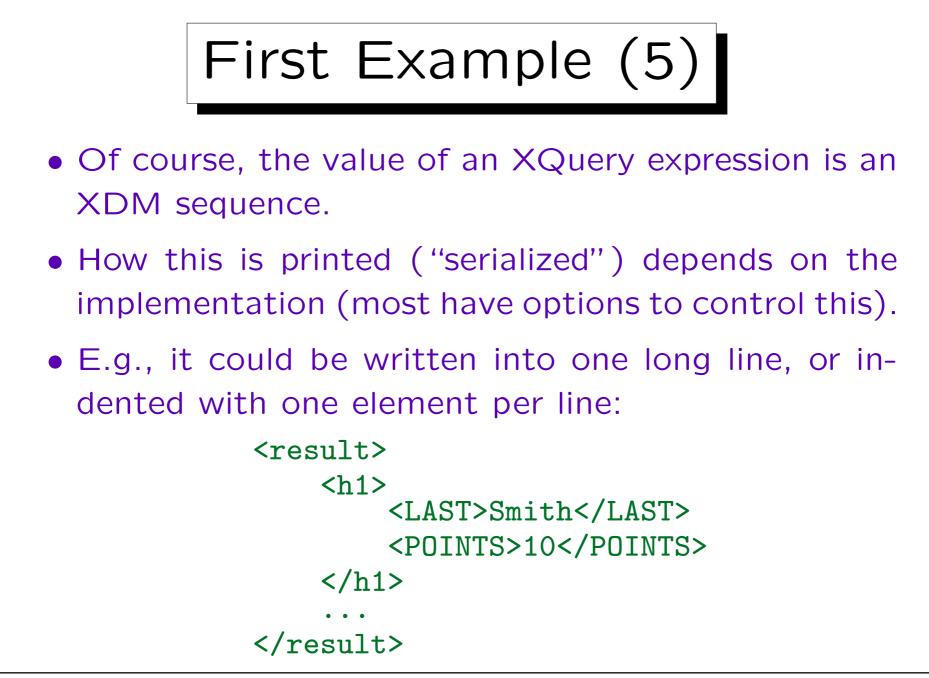
In contrast, SQL is case-insensitive. However, since case is important in XML, the choice for XQuery seems natural. As XPath, XQuery has no reserved words. It is possible to name an element for.

• XPath 2.0 is a subset of XQuery, i.e. FLWORexpressions are not the only type of queries.

The for-loop in XPath is a simplified version (special case) of the FLWOR-expression in XQuery.

```
7. XQuery
```





Stefan Brass: XML und Datenbanken



1. Introduction

2. Basic Syntax, Constructors

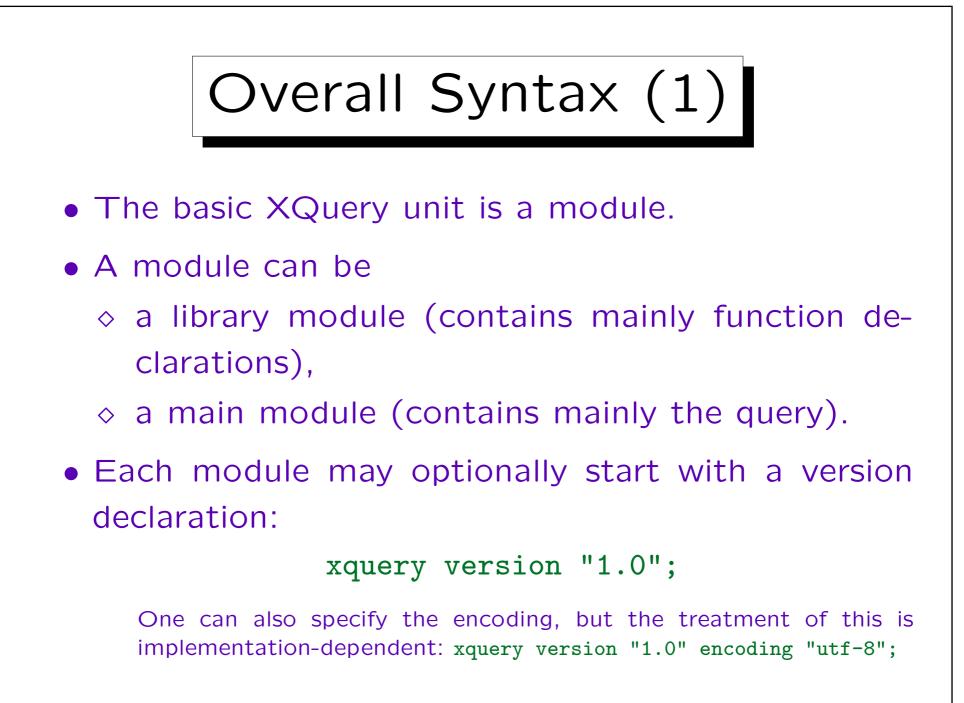
3. FLOWR-Expressions

4. Examples, Comparison with SQL

5. Prolog, Functions

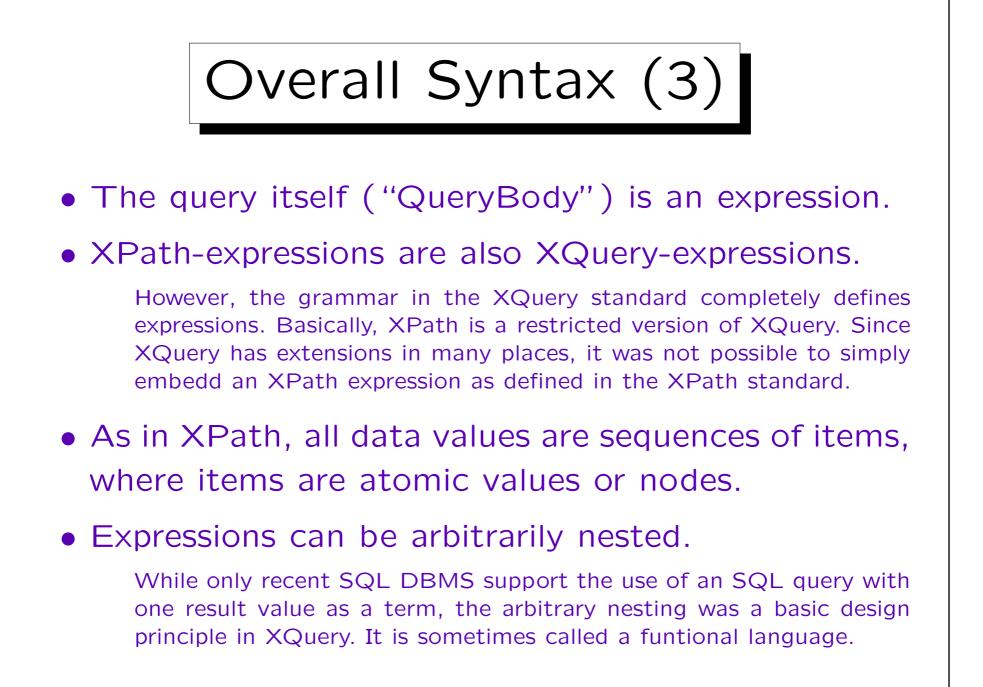
Stefan Brass: XML und Datenbanken

Universität Halle, 2008





- A main module consists of a prolog (which can be empty) and the query ("QueryBody").
- A library module consists of a module declaration and a prolog.
- The prolog can contain
 - First an arbitrary sequence of namespace declaractions, module import commands (for schemas and modules), and XQuery parameter settings,
 - ◊ and then an arbitrary sequence of variable, function, and option declarations.





- On the top level, an expression consists of one or more subexpressions ("ExprSingle") separated by "," (sequence concatenation operator).
- On the next level, an expression ("ExprSingle") is
 - ◊ a FLWOR-expression,
 - ◊ a quantified expression (some, every)
 - ◊ a typeswitch expression (see below)
 - \diamond an if-expression,
 - ◊ or an expression with the usual logical, comparison and arithmetic operators (see below).



- In comparison, the XPath 2.0 grammar has
 - ◊ a for-expression instead of the FLWOR-expression,
 - \diamond no typeswitch expression.
- Note that the for-expressions in XPath 2.0 are valid FLWOR-expressions in XQuery:
 - They have only the for and the return part.
 It is legal in XQuery to leave out the other parts.
 - The for-clause is simplified: XQuery permits to declare a type for the variable, and to add a positional variable (see below).

Expressions (3)

- The grammars for XQuery and XPath 2.0 are very similar (they are generated from a single source, only some possibilities are missing in XPath or replaced by simpler mechanisms).
- Continuing the comparison, one finds that
 Quantified expressions (some, every) permit a ty-

pe declaration for the variable in XQuery.

In XPath, no such type declaration is possible. In XQuery, it is optional (thus, XPath is still a subset of XQuery).

Expressions (4)

 The "valueExpression" (Argument of unary + and -, i.e. at the end of the operator hierarchy) is a path expression in XPath. In XQuery, there are two additional possibilities:

◇ validate (strict|lax) { {Expression} }

The expression must evaluate to exactly one document or element node. It is treated as an XML infoset (i.e. existing type annotations are ignored), validated according to the "in-scope schema definitions", and a new tree is built from the PSVI. However, the "Schema Import Feature" is optional in XQuery.

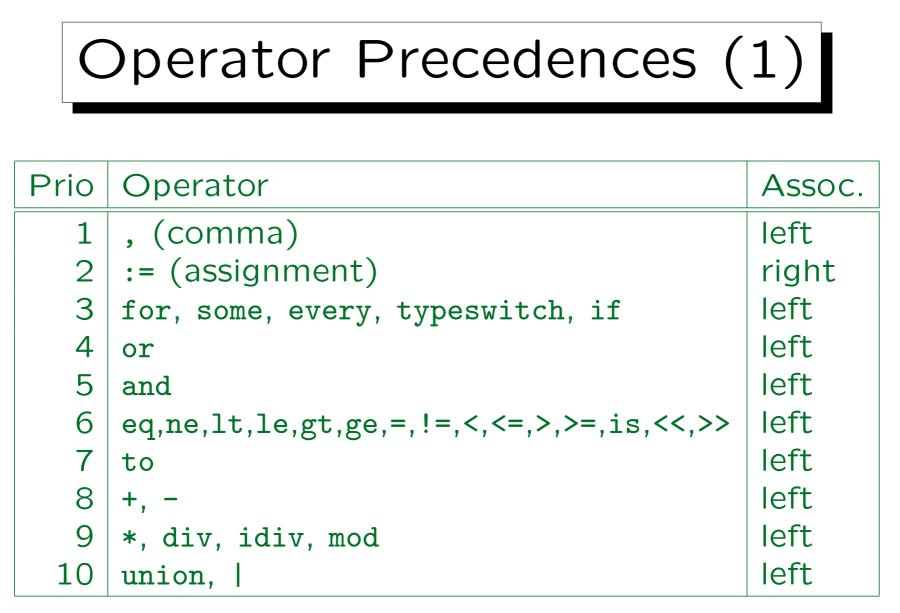
- ♦ An "extension expression" with a pragma:
 - (# ...#) { (Expression) }



• As explained above, the XPath grammar permits the namespace axis, with is not supported in XQuery.

But because it can be supported only in an inefficient way, it is anyway no longer recommended to use it.

- The next difference is in the "Primary Expression":
 - Soth languages permit numeric and string literals, variable references, expressions in (...), the context item ".", and function calls.
 - ◊ XQuery permits in addition constructors (see below), and "ordered unordered { (Expression) }".

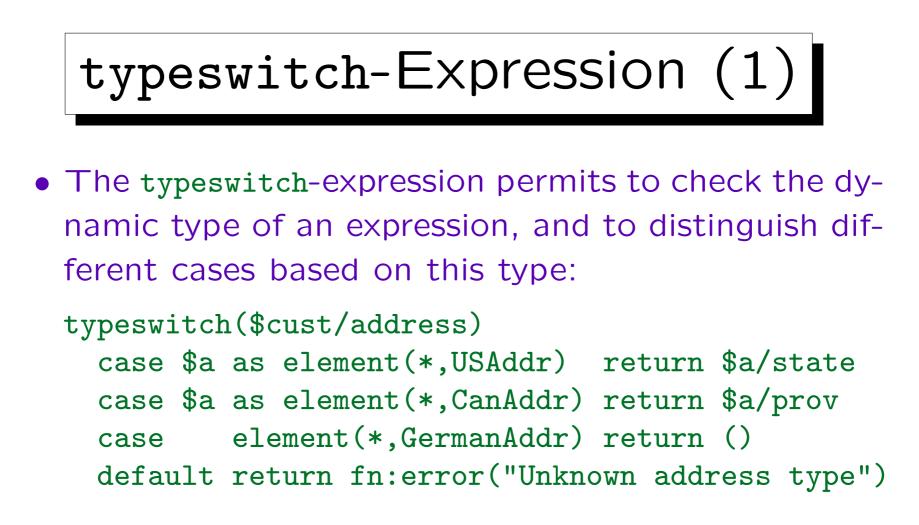


(continued on next slide)

	C	Operator Precedences	(2)
(continu	ed from previous slide)	
	Prio	Operator	Assoc.
	11	intersect, except	left
	12	instance of	left

11	intersect, except	left
12	instance of	left
13	treat	left
14	castable	left
15	cast	left
16	- (unary), + (unary)	right
17	<pre>?, *, + (Occurrence Indicators)</pre>	left
18	/, //	left
19	[],(),{}	left

Only differences (additions) to XPath: :=, typeswitch.



element(*,USAddr) matches any non-nilled element
 node with type annotation USAddr.

Or a type derived from that. This example needs schema validation.



- The first case-clause with a matching type is selected, or the default clause if non matches.
- A variable must be declared in the case only if the value of the original expression is needed to compute the return value.

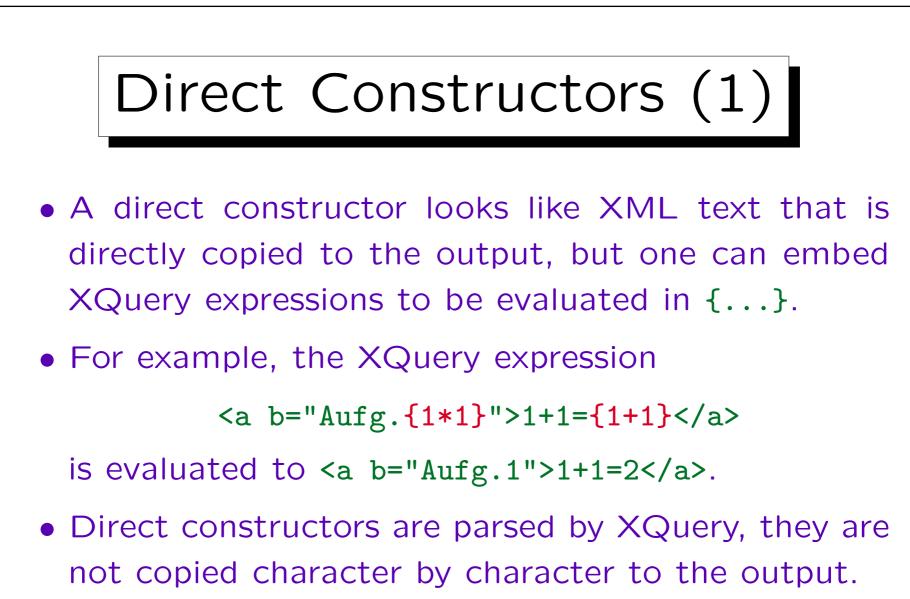
The scope of this variable declaration is this single case. Different cases can declare variables with the same name.

• The same effect can be achieved with conditional expressions (if) and "instance of".

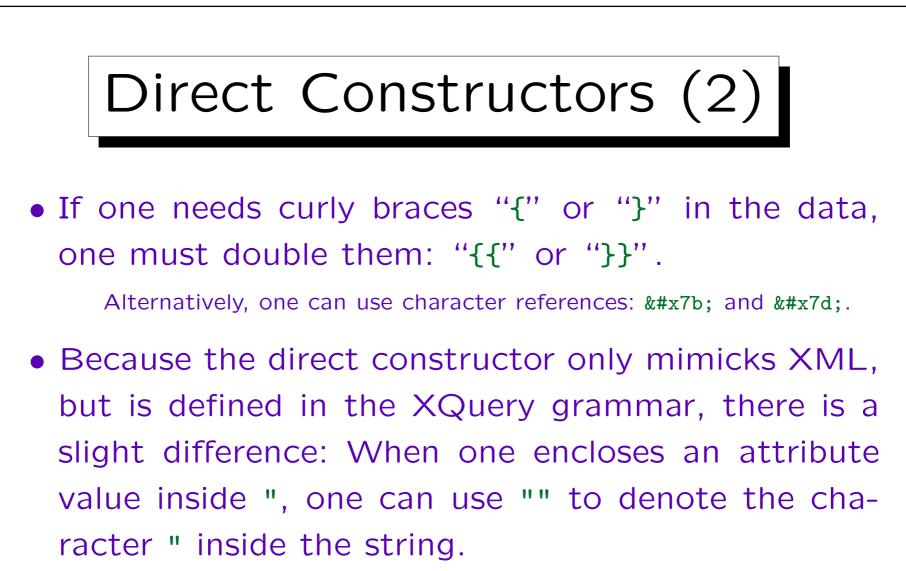
"treat as" is necessary in addition to use the value as a value of its real type. So in the end, the typeswitch simplifies the expression.

Constructors: Overview

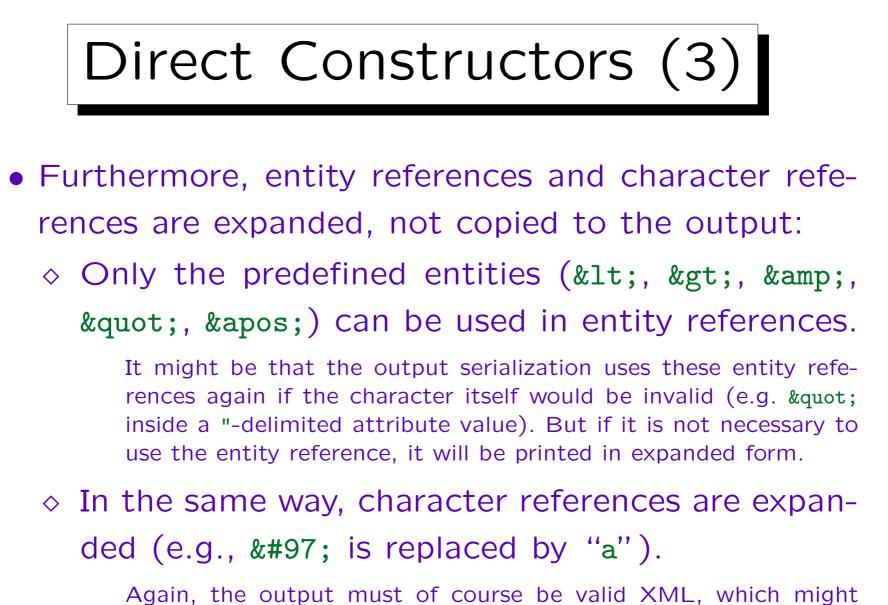
- An important difference between XPath and XQuery is that XQuery can generate new nodes, XPath can only select nodes from given documents.
- Creating new nodes is done by constructors.
- There are two types of constructors in XQuery:
 - ◇ Direct constructors, which look like XML text.
 There e.g. the node name is explicitly given.
 - Computed constructors, which have a new syntax, and permit to compute e.g. the node name by an expression.



The internal XDM representation of the output is constructed, thus e.g. information about extra whitespace/line breaks inside tags is lost.



Correspondingly, when one encloses it in ', the apostrophe is written ''. This is the XQuery/XPath convention, not the XML convention. There one must use an entity reference or a character reference.



require some form of escaping (entity / character references).

Direct Constructors (4)

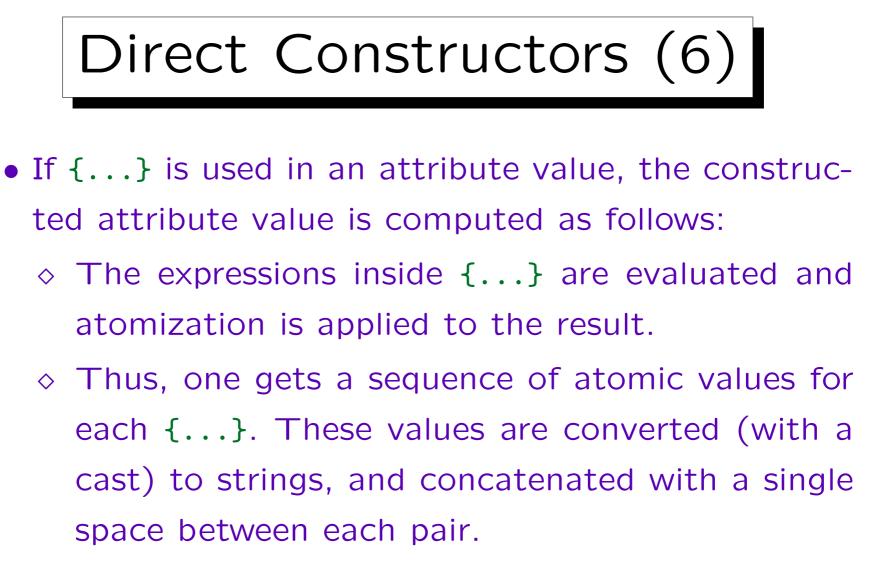
 XQuery comments (: ... :) cannot be used in the direct element constructor, neither in the tags nor in the content (except of course inside {...}).

In the tags they are a syntax error, in the content they are considered as text data. Formally, comments can appear everywhere where "ignorable whitespace" can appear. A few productions in the XQuery grammar are marked with /* ws:explicit */. Inside these productions, the nonterminal S (known form the XML grammar) is used to mark explicitly where whitespace is allowed. This nonterminal does not match the XQuery comment. The productions for the direct element constructor have this explicit whitespace. In this way they are made more compatible with the real XML grammar, although I personally do not see the advantage of forbidding comments inside tags.

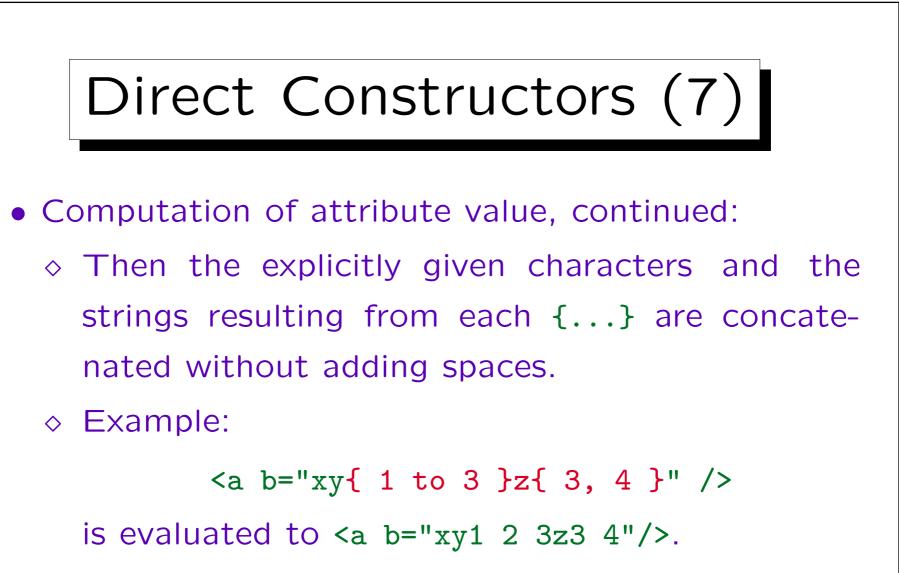
Direct Constructors (5)

- Allowed occurrences of embedded XQuery expressions ({...}) inside direct element constructors:
 - ♦ The element name (element type) and the attribute names must be given explicitly (a QName), and cannot be computed with {...}.

If one wants to compute these, one must use the computed element constructor (see below).



At the beginning and the end of the sequence, no space is inserted, thus the empty sequence gives the empty string.

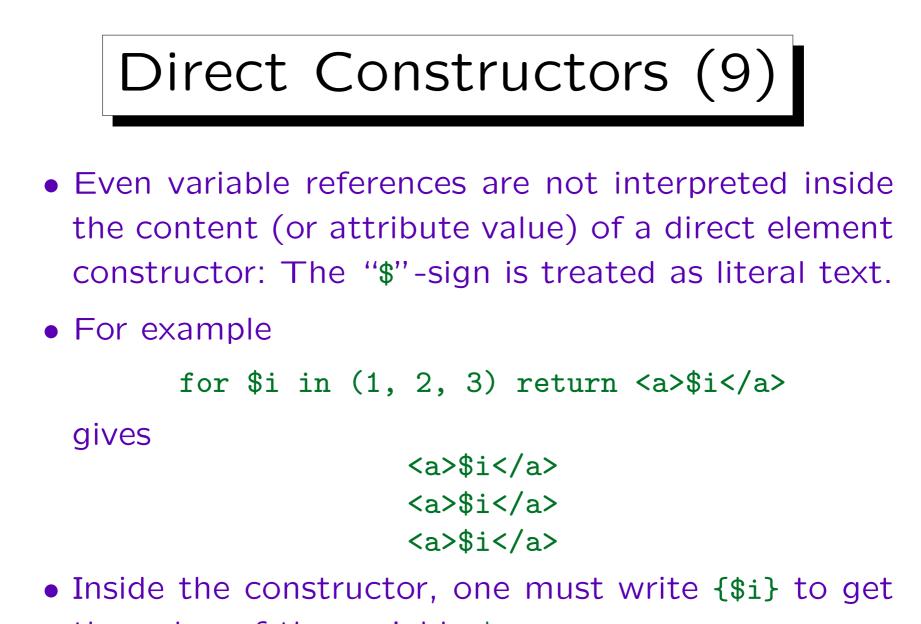


◊ If the attribute name is xml:id, the attribute value is treated specially (as an ID).

Direct Constructors (8)

- The content of a direct element constructor can contain (between start tag and end tag):
 - ◊ Literal text (without the characters <, {, }, &),</p>
 - $\diamond\,$ entity references for the five predefined entities,
 - ◊ character references,
 - ◊ CDATA sections: <![CDATA[...]]>,
 - ♦ enclosed expressions: {...},
 - ◇ other direct constructors (for element, comment, and processing instruction nodes).



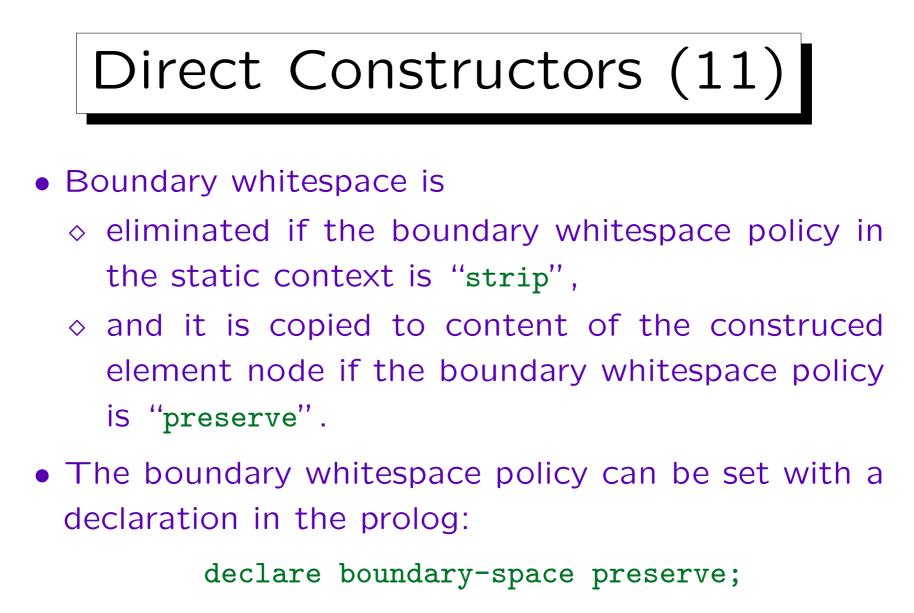


the value of the variable \$i.

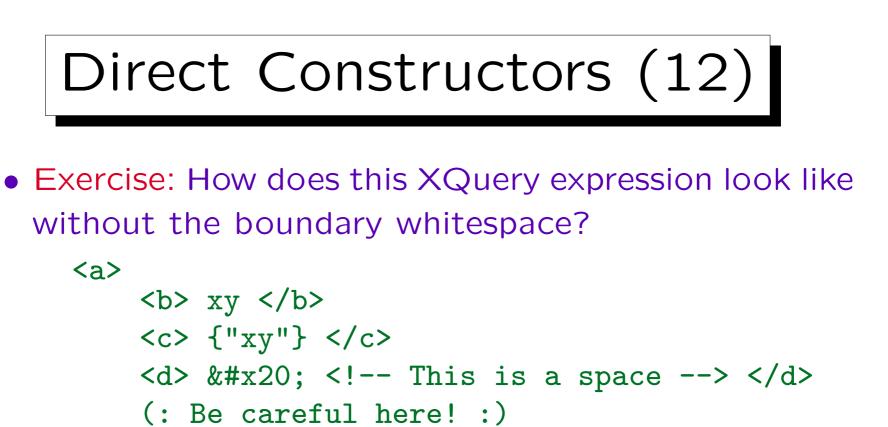
Direct Constructors (10)

- A sequence of whitespace characters (e.g. spaces, line breaks) within the content of a direct element constructor is considered "boundary whitespace" if it is delimited on both sides by
 - the start or end of the content (i.e. the start tag or end tag of the direct element constructor), or
 - ◊ and enclosed direct constructor (e.g. start and end tags of direct element constructors), or
 - \diamond an enclosed expression {...}.

Space characters generated by character references, CDATA sections, or enclosed expressions do not count as whitespace here.

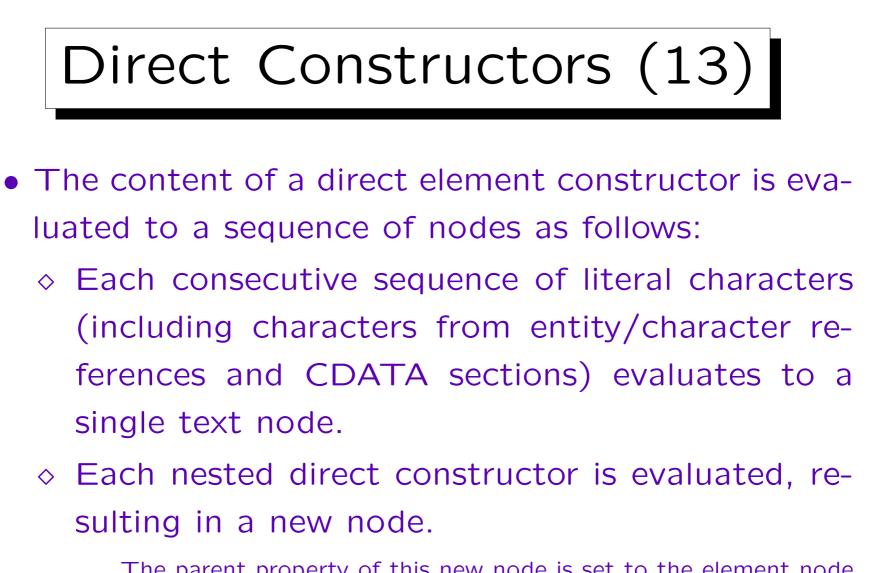


• The default is implementation-defined.

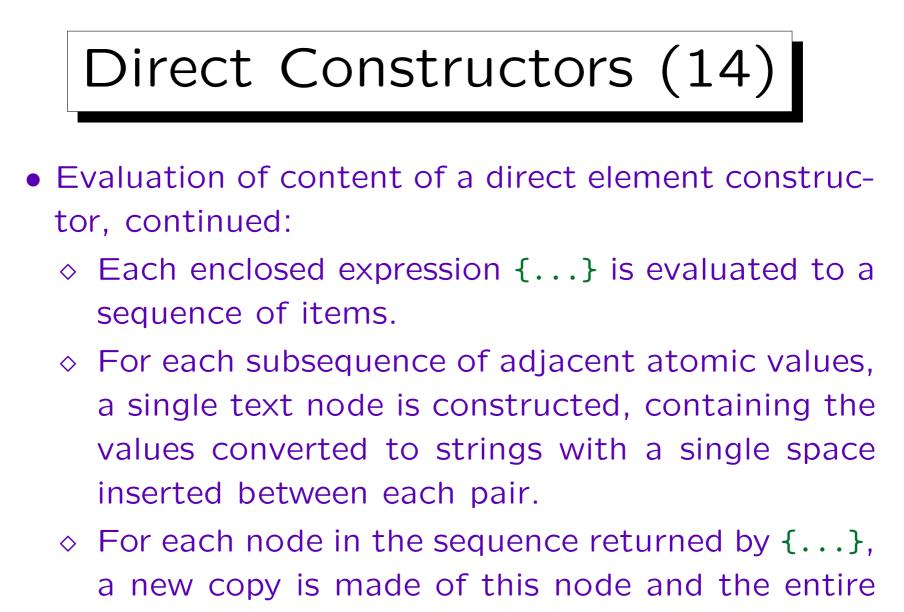


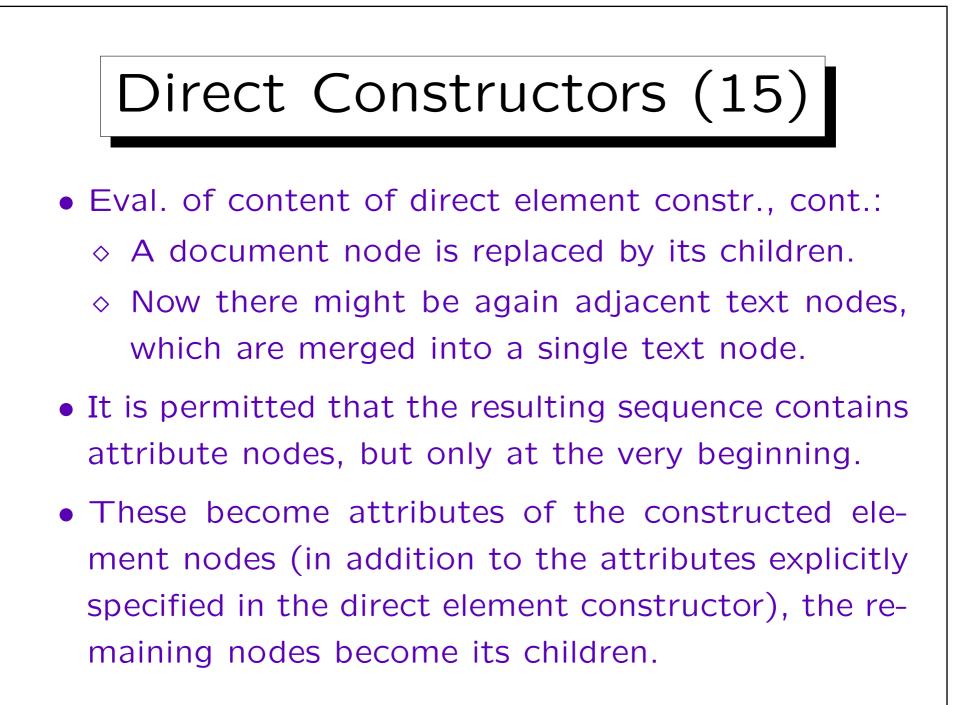
• The example shows also a direct comment constructor.

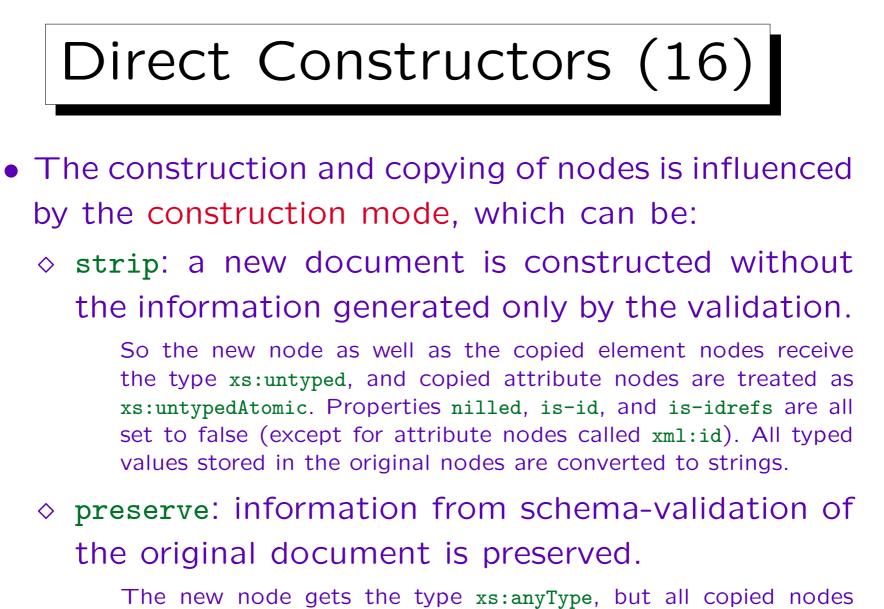
One cannot use enclosed expressions $\{\ldots\}$ in a direct comment constructor. One must write the comment explicitly. But there is of course also a computed comment constructor (see below).



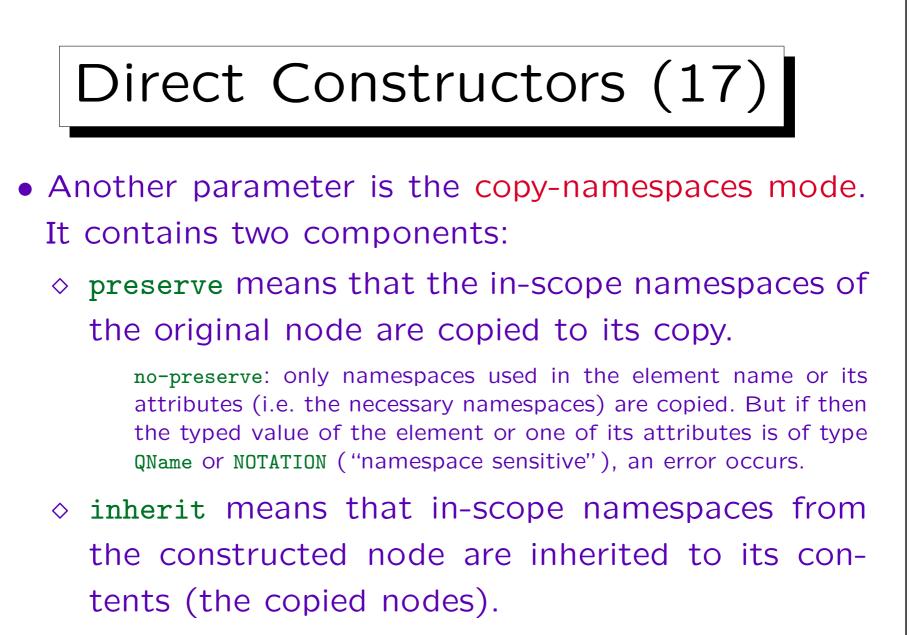
The parent property of this new node is set to the element node that is currently being constructed. The standard also explains how the base-uri-property is set (see Section 3.7.1.3).



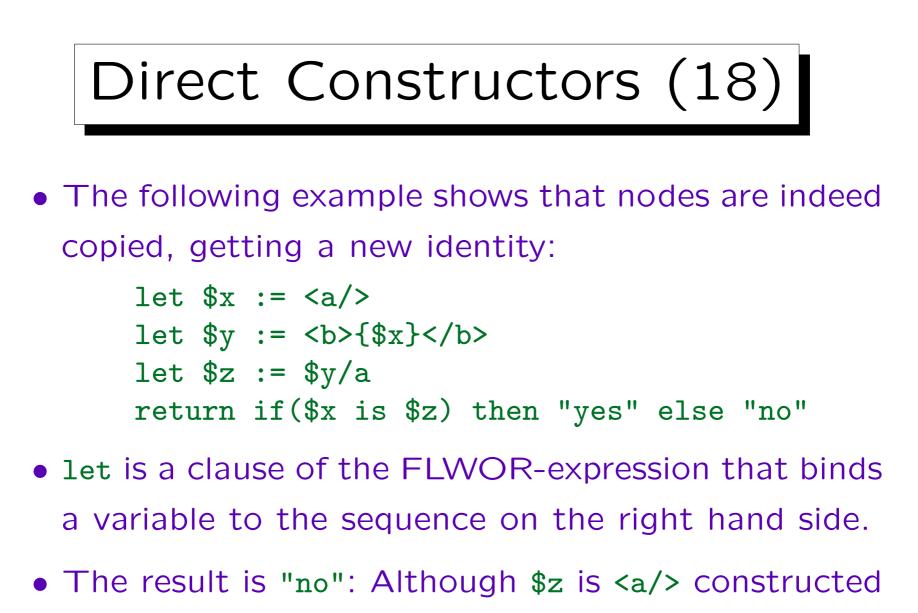




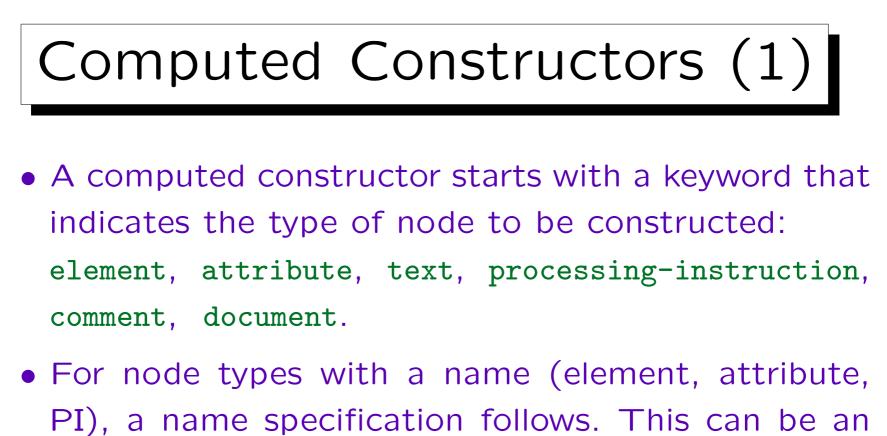
retain their original type. Properties like nilled are copied.



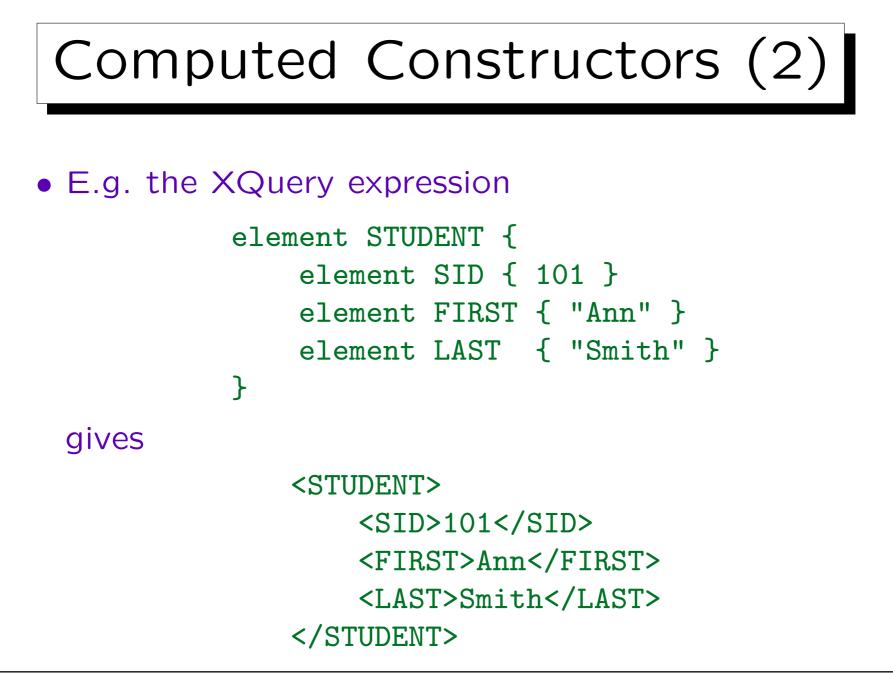
Possibly overridden by namespaces copied from original node.

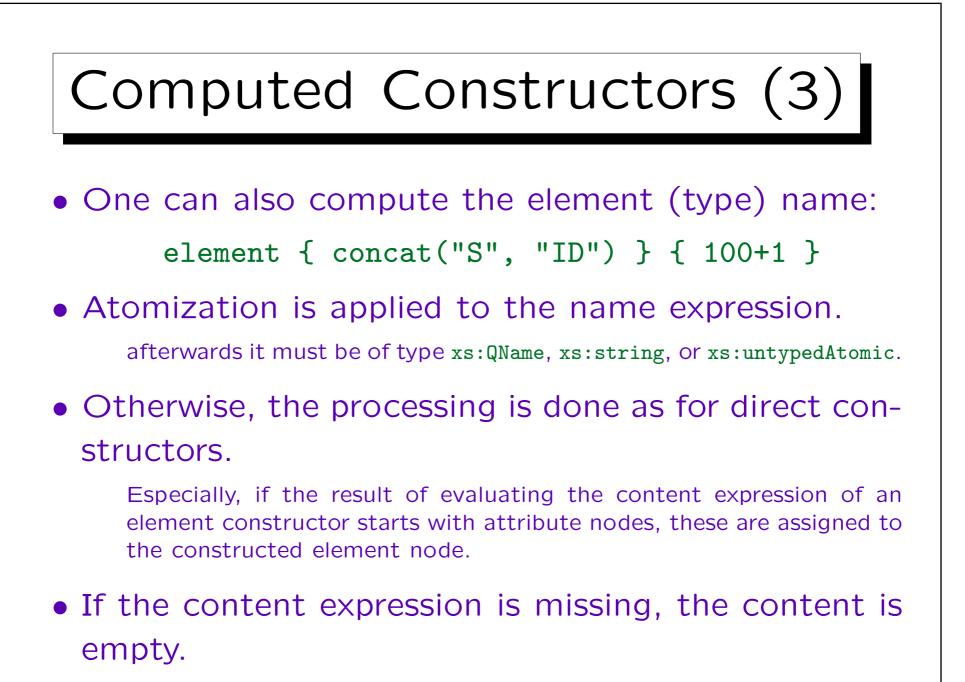


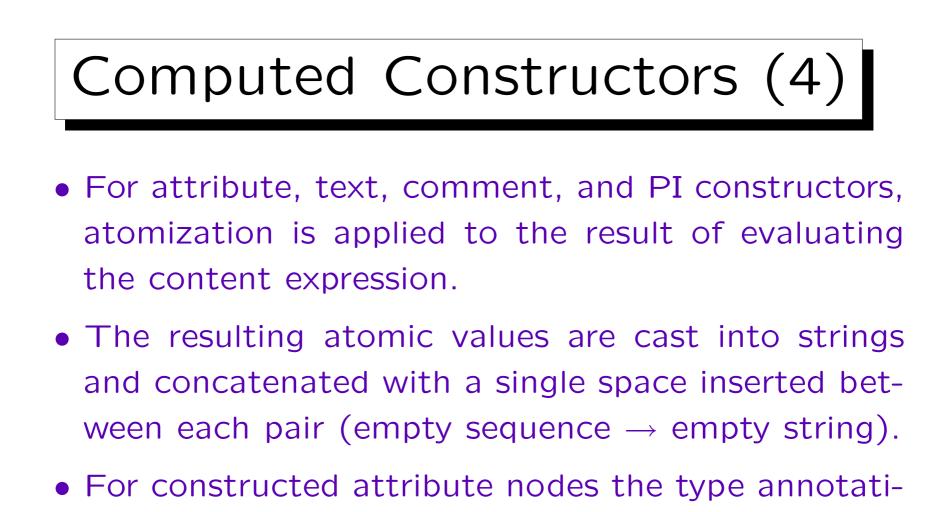
from \$x, it has a new identity.



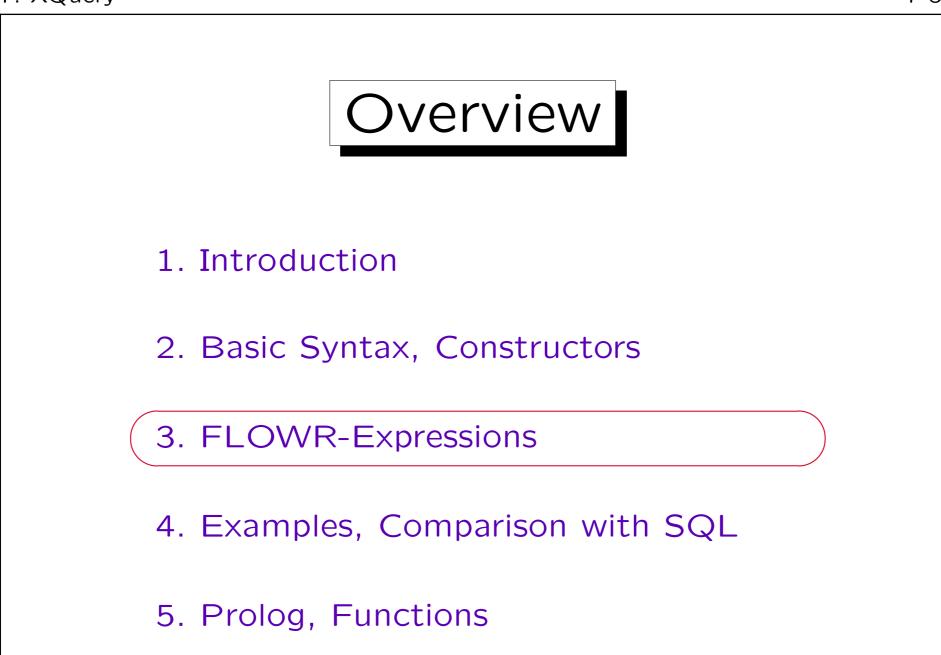
- explicitly given QName or an enclosed expression {...} ("name expression of the constructor").
- Next, the content is defined by an expression in {...} ("content expression") (optional).

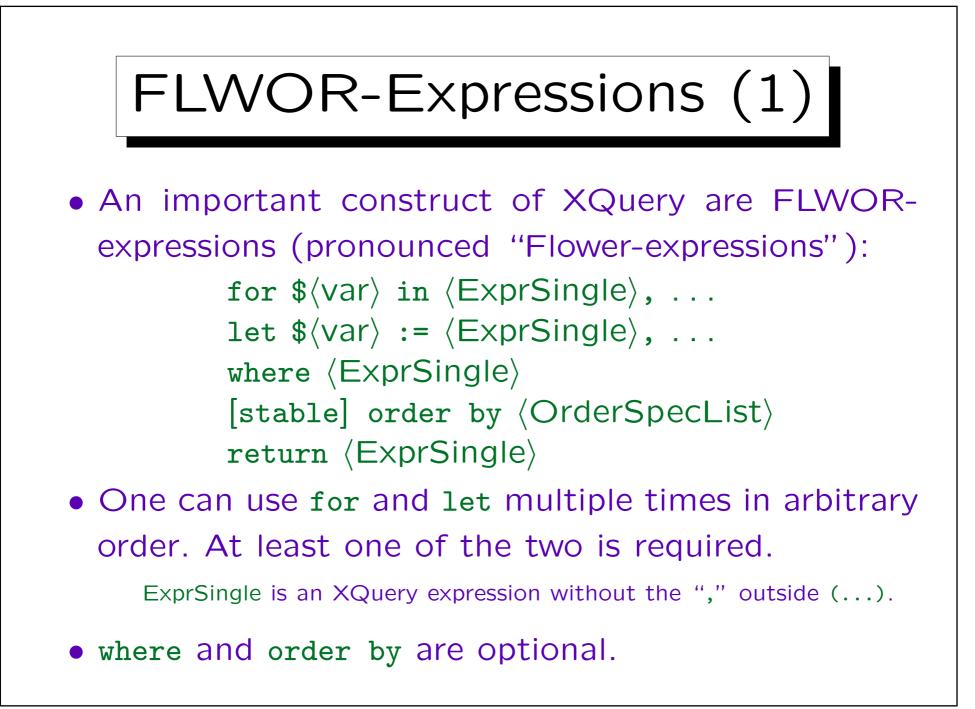


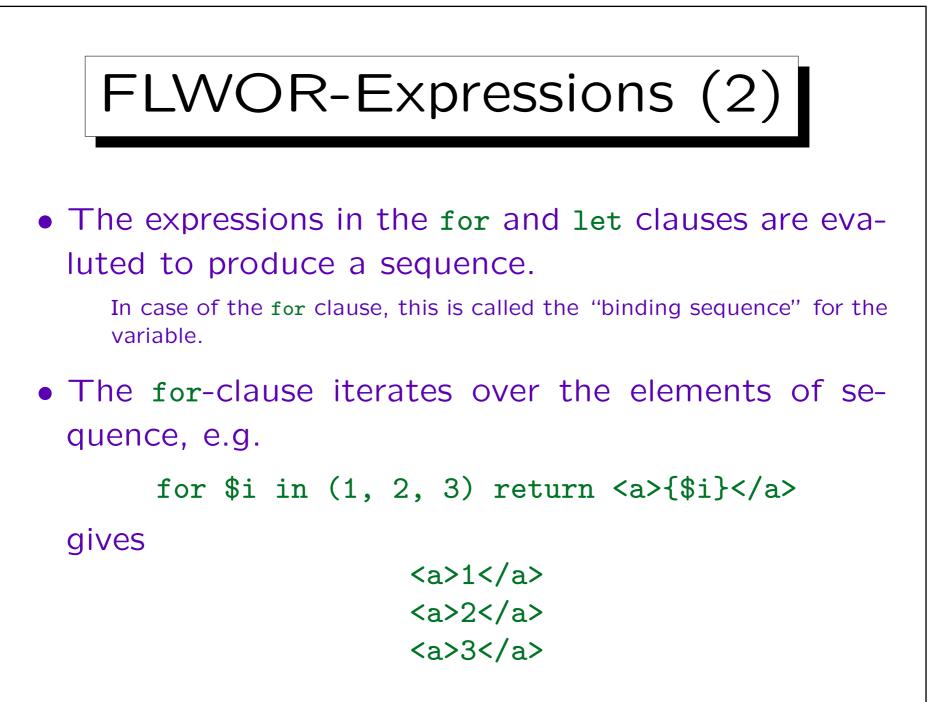


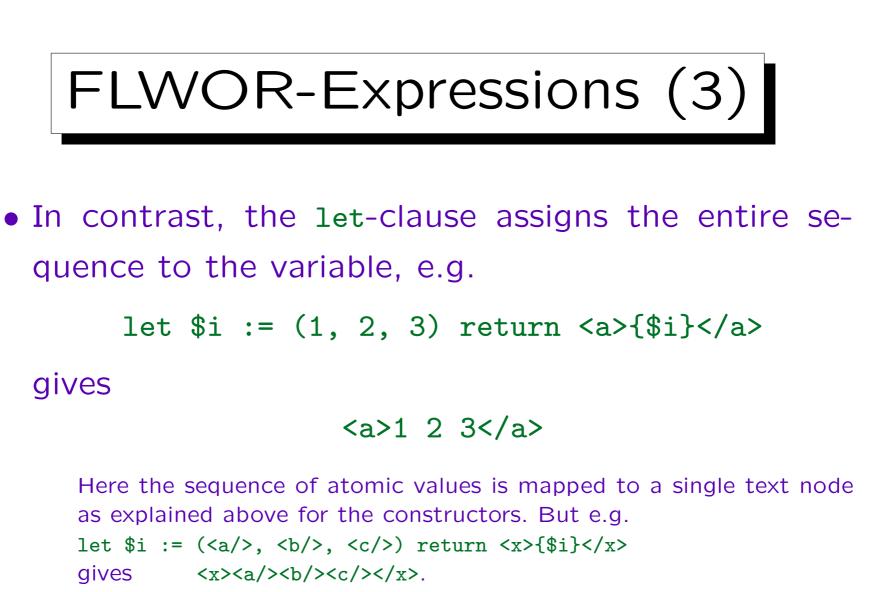


- ON is xs:untypedAtomic.
- Constructed text nodes are automatically deleted when their text is the empty string.

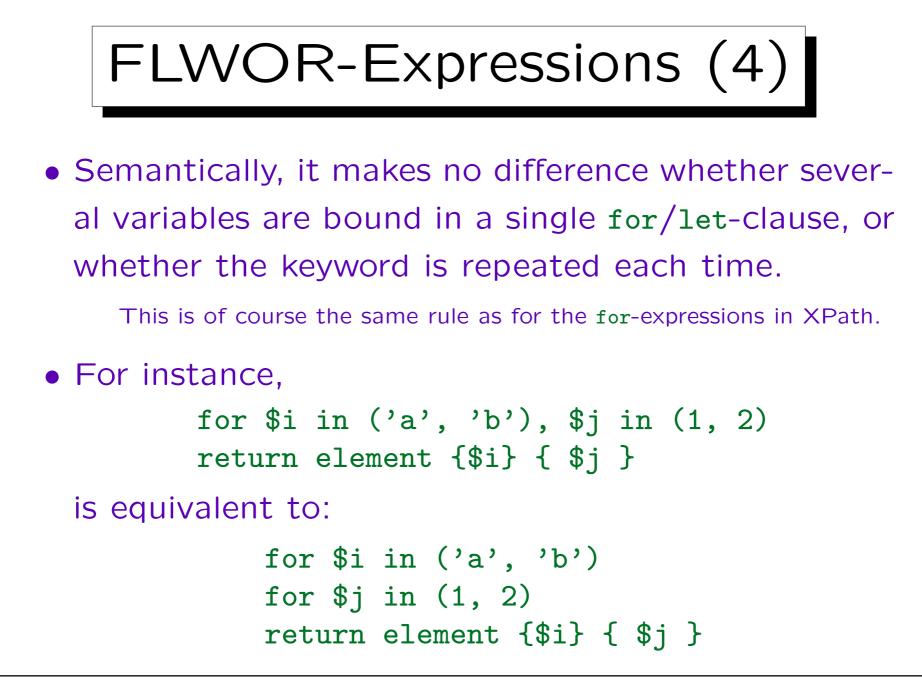


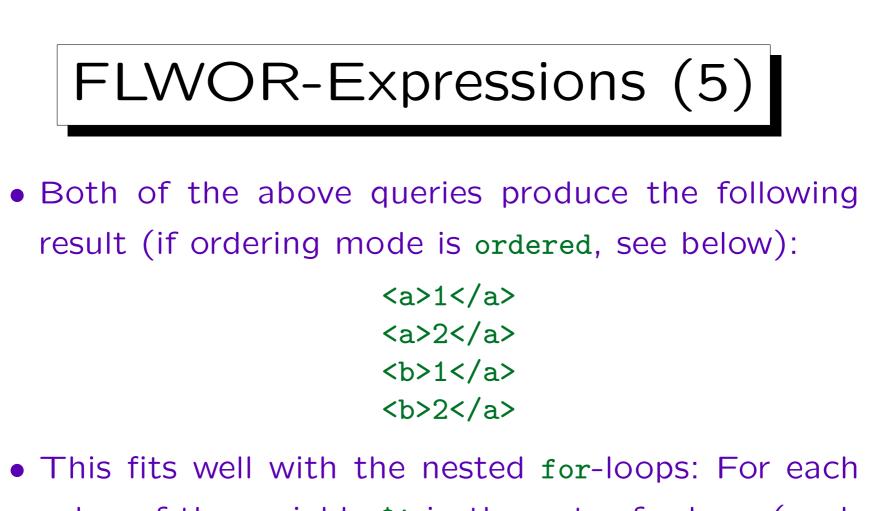




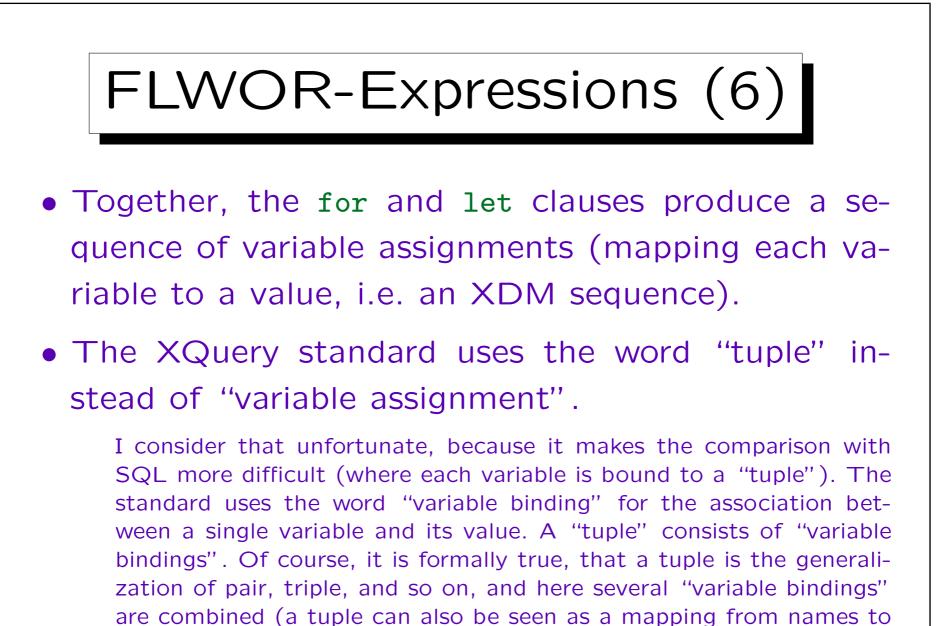


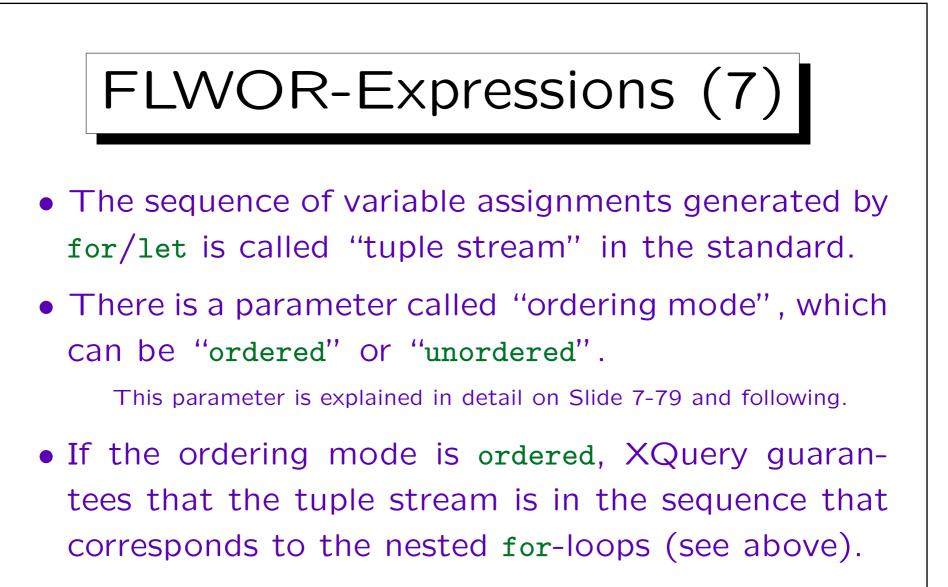






value of the variable \$i in the outer for loop (each element name), the inner for loop (over \$j) (the element content) is repeated once.



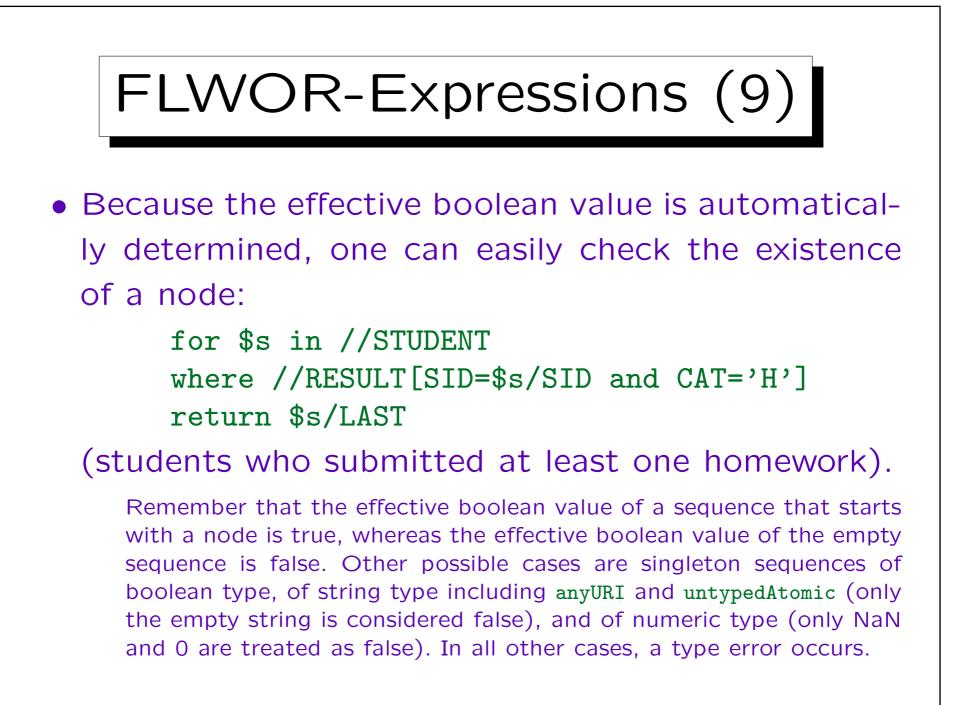


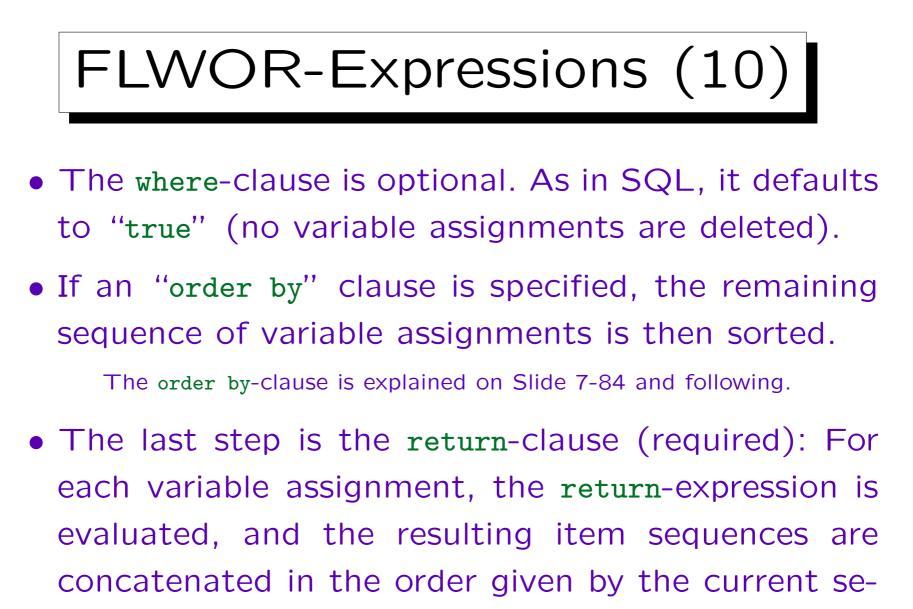
• Then the document order of the original document is retained in the output of the query.



- Next, the where-clause acts as a filter on the "tuple stream" (sequence of variable assignments).
- For each variable assignment, the expression under where is evaluated, and its effective boolean value is determined.
- If it is false, the variable assignment is deleted from the sequence.

I.e. the remaining sequence of variable assignments contains only those variable assignments for which the where-condition is true. The variable assignments remain in the same relative order in which they were generated by the for and let-clauses.



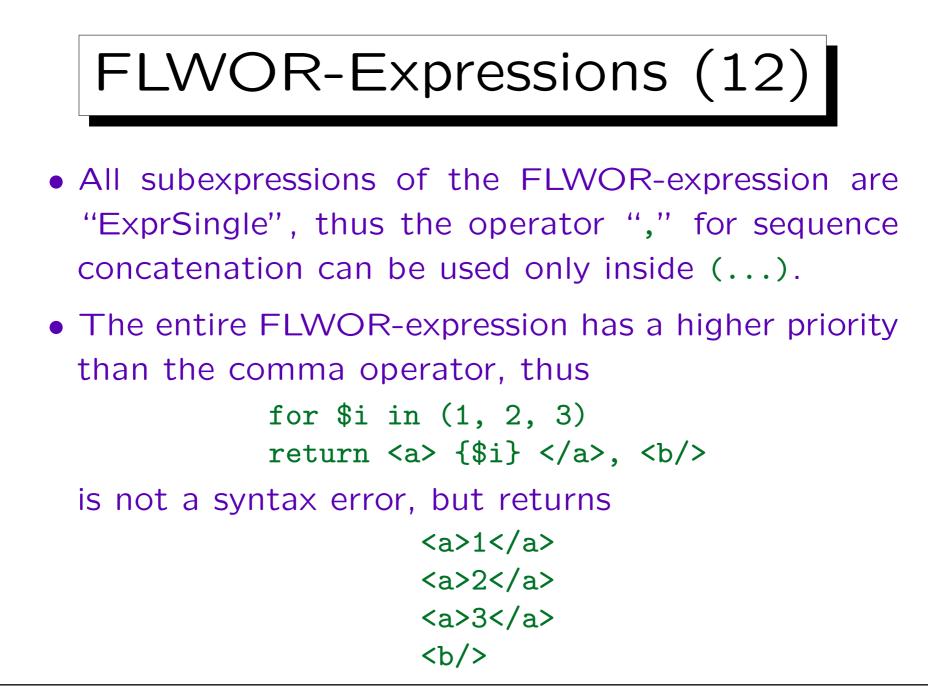


quence of variable assignments.

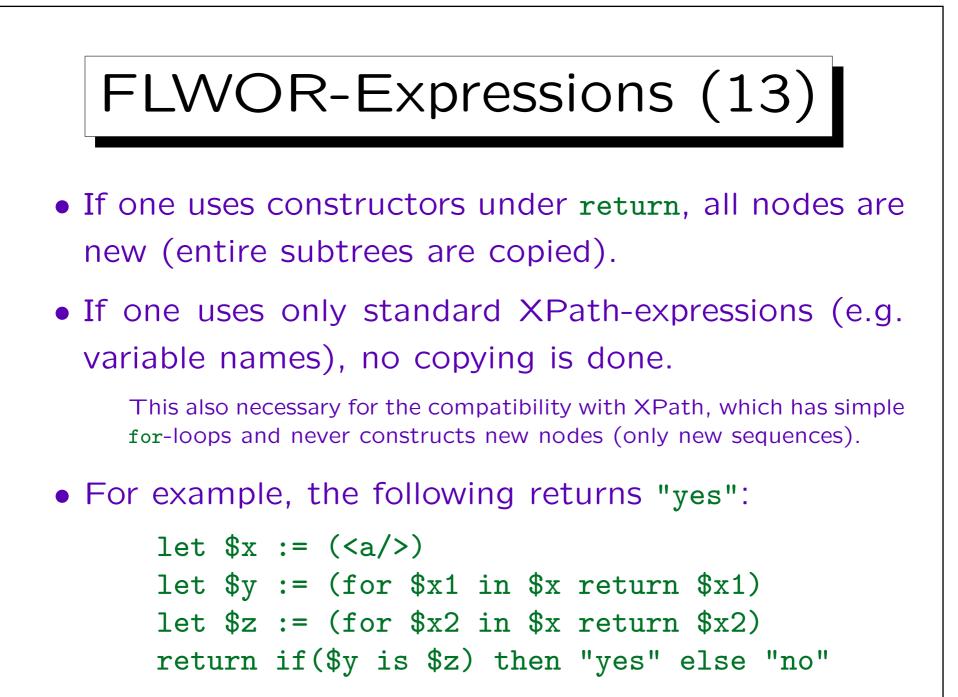
FLWOR-Expressions (11)

- The length of the result sequence can differ from the number of variable assignments considered, because the return-expression can evaluate to a sequence of arbitrary length (0, 1, or more).
- In the following query, there is one variable assignment per SQL exercise (2), but the output sequence contains one entry per solution (6 = 3 + 3):

7. XQuery



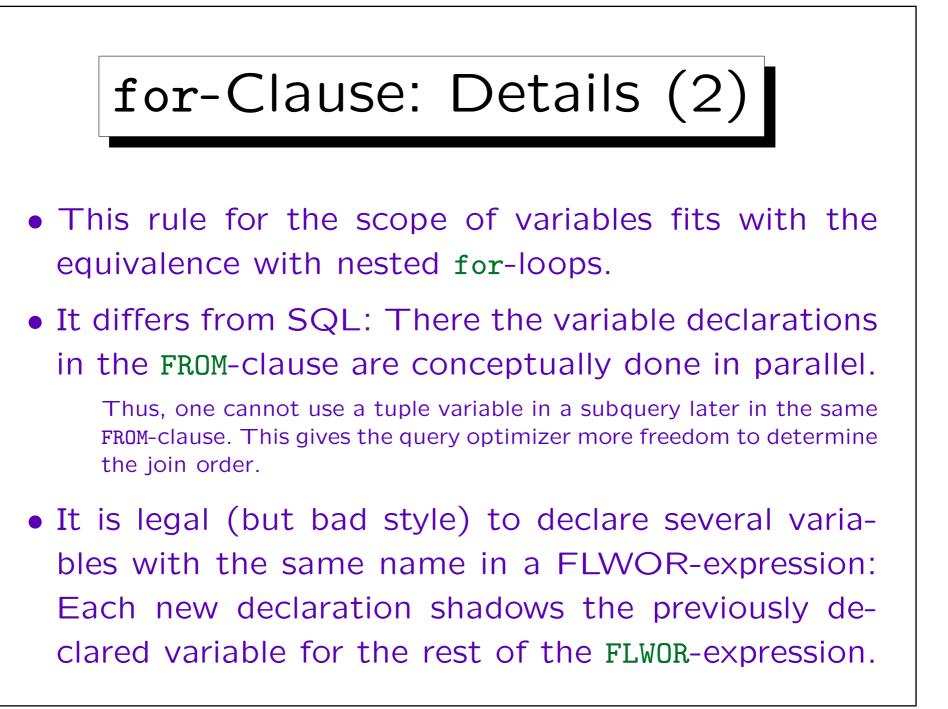
7. XQuery



for-Clause: Details (1)

- The scope of a variable declared with for or let extends from the point just after the binding expression (which defines the values for the variable) to the end of the FLWOR-expression.
- Thus, the variable can already be used in binding expressions for other variables declared later in the same for-clause:

for \$s in //STUDENT, \$r in //RESULT[SID=\$s/SID]
return element solved {\$s/LAST, \$r/CAT, \$r/ENO}

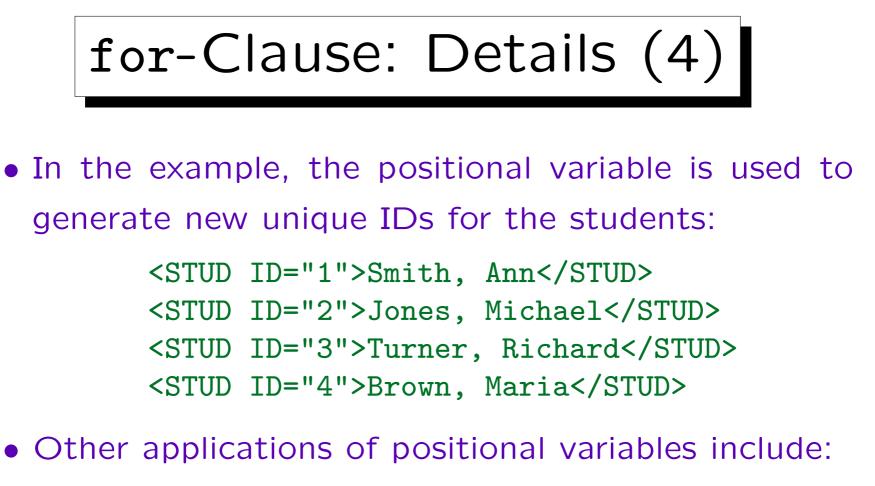




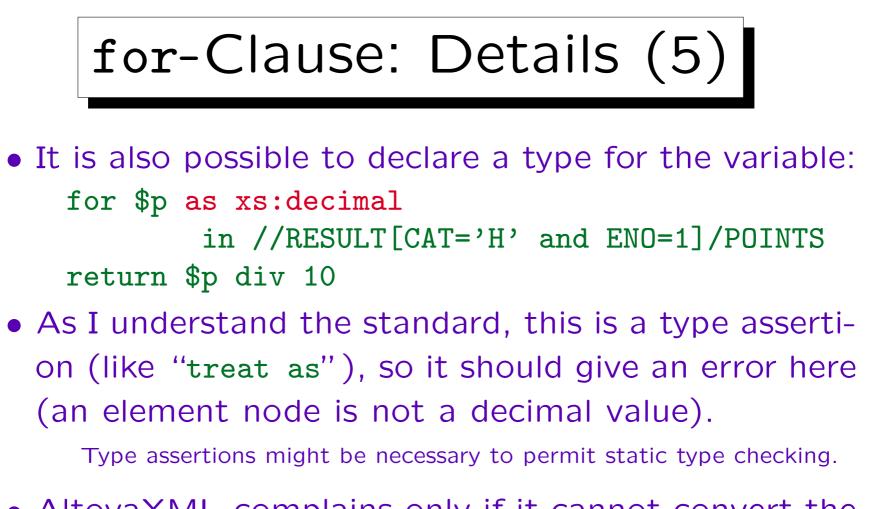
• One can define a "positional variable" associated with a variable declared in a for-clause, e.g.

```
for $s at $i in //STUDENT
return element STUD {
    attribute ID {$i},
    $s/concat(LAST, ", ", FIRST)
}
```

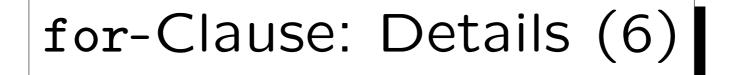
- \$i contains the position of the current value for \$s in the binding sequence, i.e. the value of //STUDENT.
- Positions are counted from 1. The result of the query is shown on the next slide.



- ♦ First-n queries (see below).
- ◇ Sampling: E.g. take only every 10-th student: \$i mod 10 = 1



- AltovaXML complains only if it cannot convert the value to the required type (treats it as type cast).
- Positional variables always have type xs:integer.



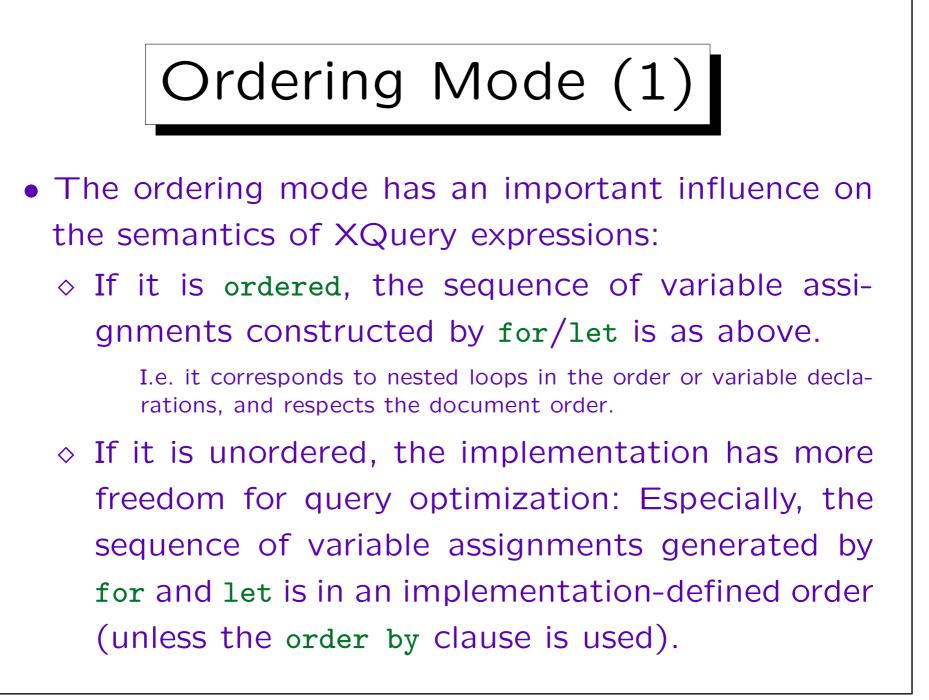
- The for-clause consists of a comma-separated list of one or more variable declarations, each consisting of:
 - ◊ The name of the variable (starting with "\$"),
 - optionally, a type declaration, consisting of the keyword "as" and a sequence type,

In the for-clause, the occurrence indicators ?, +, * are not relevant because the variable is bound to single sequence elements.

- optionally, a positional variable declaration, consisting of the keyword "at" and a variable name,
- \diamond the keyword "in", and an "ExprSingle".

let-Clause: Details

- The let-clause consists of a comma-separated list of one or more variable declarations with a slightly different syntax than under for (to emphasize that the entire sequence is bound to the variable):
 - ◊ Name of the variable (starting with "\$"),
 - optionally, a type declaration, consisting of the keyword "as" and a sequence type,
 - \diamond the symbol '':='', and an ''ExprSingle''.
- Of course, positional variables make no sense in the let-clause (and are therefore not permitted).



Ordering Mode (2)

- But the consequences of ordering mode unordered are even more drastic, because
 - in XPath expressions, document order does not have to be respected,
 - ◊ thus selecting specific positions becomes more or less meaningless (nondeterministic).
 - ◊ E.g. /a/b[1] gives any b-child of a, not necessarily the first.

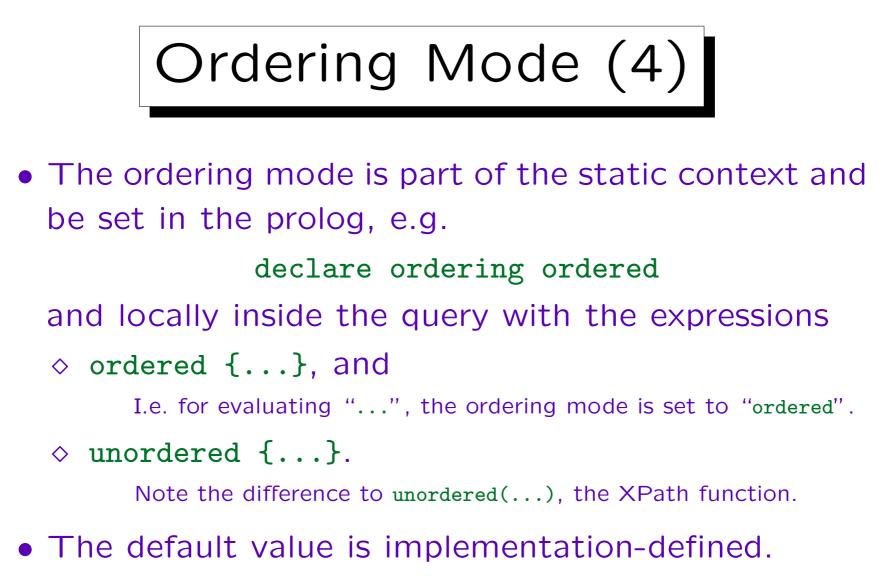
But /a/b[3] is non-empty only if there are at least three b-children.

Ordering Mode (3)

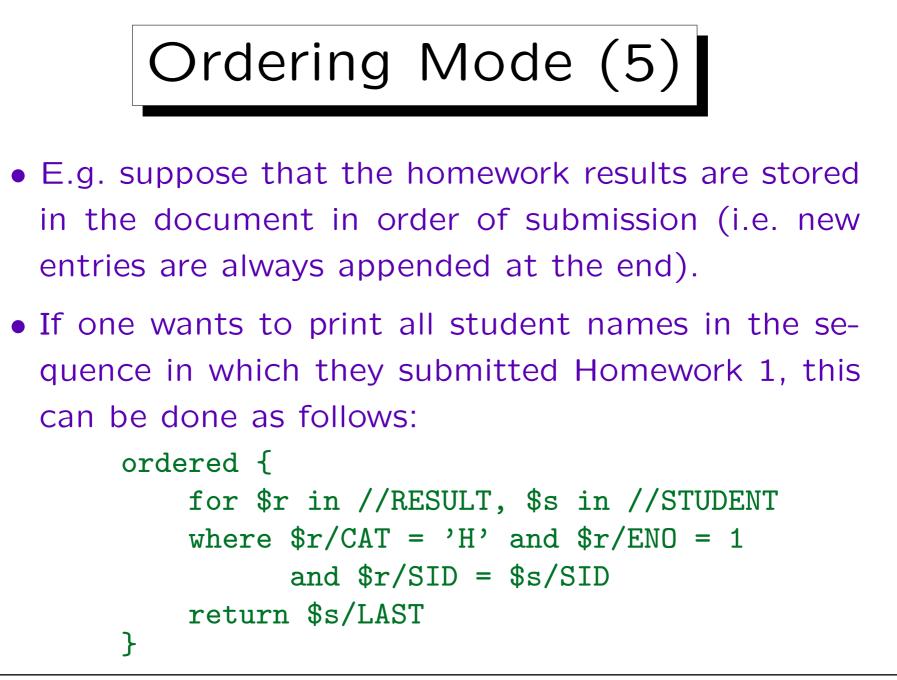
 Thus, ordering mode unordered is not only a question of the output sequence, but can modify also the selected values.

Actually, that is not so astonishing, because the arbitrary nesting of XQuery expressions means that as soon as one allows a different result sequence in FLWOR-expressions, one could anyway get an entirely different result for the entire query, not only a permutation. Everything in XQuery is a sequence, and the exact order matters in many places.

• When one uses the XPath function unordered(...) only at this single point an arbitrary permutation is allowed (not everywhere inside as with the ordering mode). E.g. positions inside remain meaningful.



This seems unfortunate, because it immediately causes portability problems: The ordering mode is important for nearly every query.





• With the order by clause, one can sort the tuple stream (variable assignments) generated by for/let.

After passing the filter of the where-clause and before the returnclause is evaluated.

- This is done by defining one or more expressions, the values of which are used for sorting, e.g.
 for \$s in //STUDENT, \$r in //RESULT[SID=\$s/SID] where \$r/CAT = 'H' and \$r/ENO = 1 order by \$r/POINTS
 - return \$s/LAST

order by Clause (2)

 More specifically, the expression(s) are evaluated for each variable assignment, and atomization is applied.

After that, each expression must return a sequence of length ≤ 1 (for a given variable assignment), i.e. a single value or the empty sequence. Otherwise (longer sequence) a type error occurs.

- Values of type untypedAtomic are treated as string.
- Then the values of each expression (for all variable assignments) must be comparable: The comparison is done with the operator gt of the least common supertype that has such an operator.

order by Clause (3)

- Suppose that n expressions are used as sort criteria, and the values for variable assignment \mathcal{A} are (x_1, \ldots, x_n) , and for variable assignment \mathcal{B} , the values are (y_1, \ldots, y_n) .
- Then $\mathcal A$ comes after $\mathcal B$ in the sort order if there is $i\in\{1,\ldots,n\}$ such that
 - \diamond neiter $x_j > y_i$ nor $y_j > x_j$ for $j = 1, \dots, i-1$, and
 - $\diamond \ x_i > y_i.$

I.e. the result of the first expression has highest priority, the second expression decides the relative order of two variable assignments when the values of the first expression are equal/uncomparable, and so on.

order by Clause (4)

- For each expression, one can specify ascending or descending:
 - ♦ ascending is the default: The least value is listed at the beginning, the greatest at the end.
 - descending selects the inverse order: The maxi- mum value is listed first, and then successively smaller values, until the minimum value.
- The abbreviations asc and desc known from SQL are not supported in XQuery.

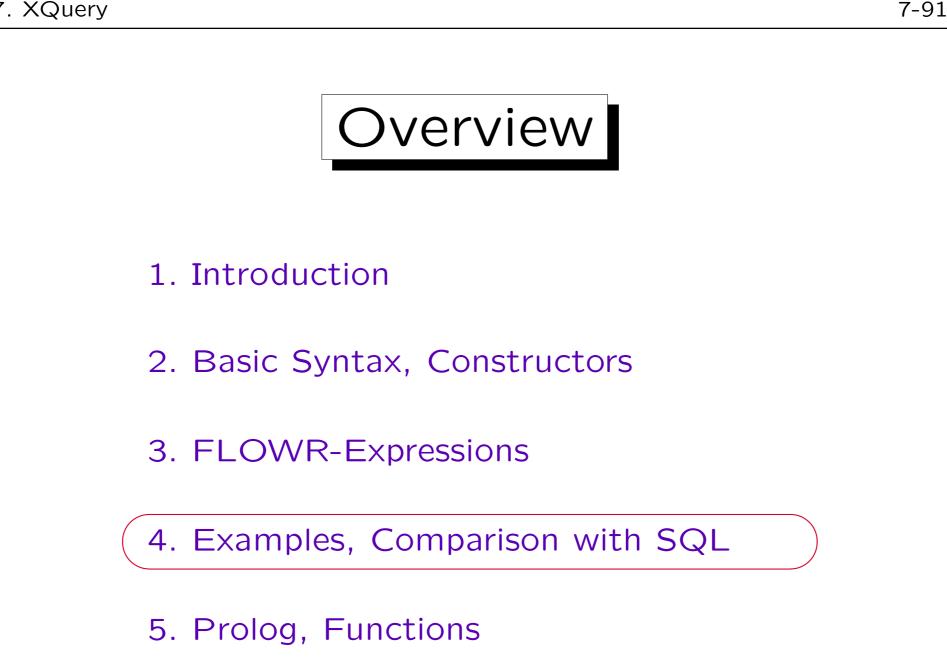
order by Clause (5) • Another difference to SQL is that XQuery does not require that the values used for sorting are also printed. However, modern SQL DBMS do not actually have this requirement. • For each column, one can specify \diamond empty greatest: The empty sequence is listed last in ascending order (first in descending order). And NaN comes immediately before the empty sequence in ascending order (immediately after it in descending order). ◊ empty least: The empty sequence comes first in ascending order, last in descending order.

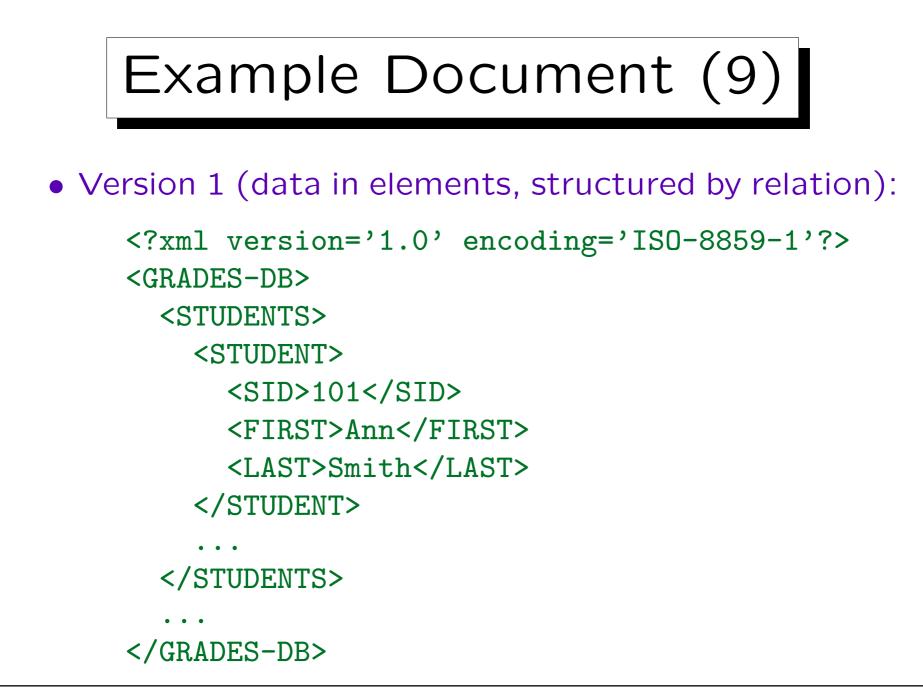
order by Clause (6)

- Thus, the order by clause consists of a commaseparated list of one or more "order specs", each of which consists of
 - ◊ an "ExprSingle" (values used for sorting),
 - optionally, one of the keywords "ascending" or "descending",
 - ◇ optionally, one of the phrases "empty greatest" Or "empty least",
 - ◊ optionally, the keyword "collation" and an URI literal (this defines the sort order for strings).

order by Clause (7)

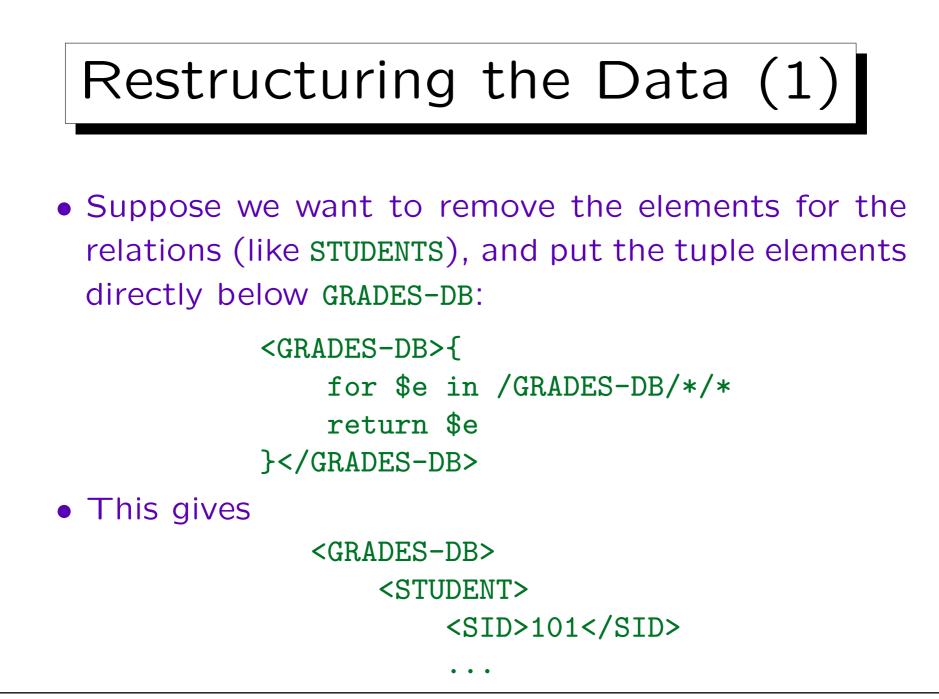
- Finally, instead of "order by", one can also write "stable order by".
- This means that if the sort criteria give no decision, the original order of the variable assignments must be kept (derived from the document order and the sequence/nesting of for-loops).
- Of course, if an order by clause is specified, this takes precedence over the ordering mode "unordered".



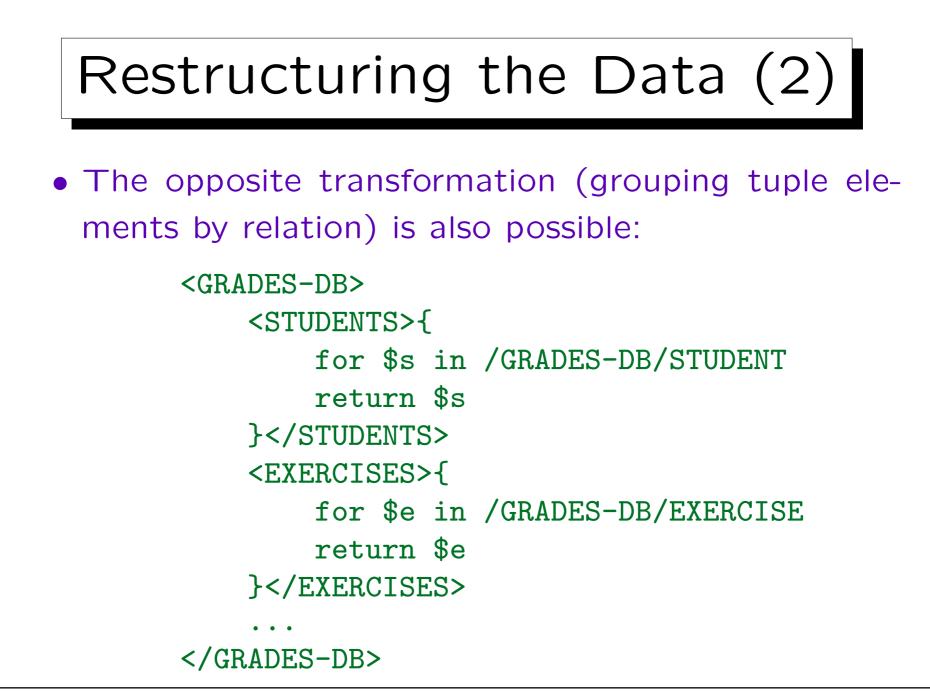


Stefan Brass: XML und Datenbanken

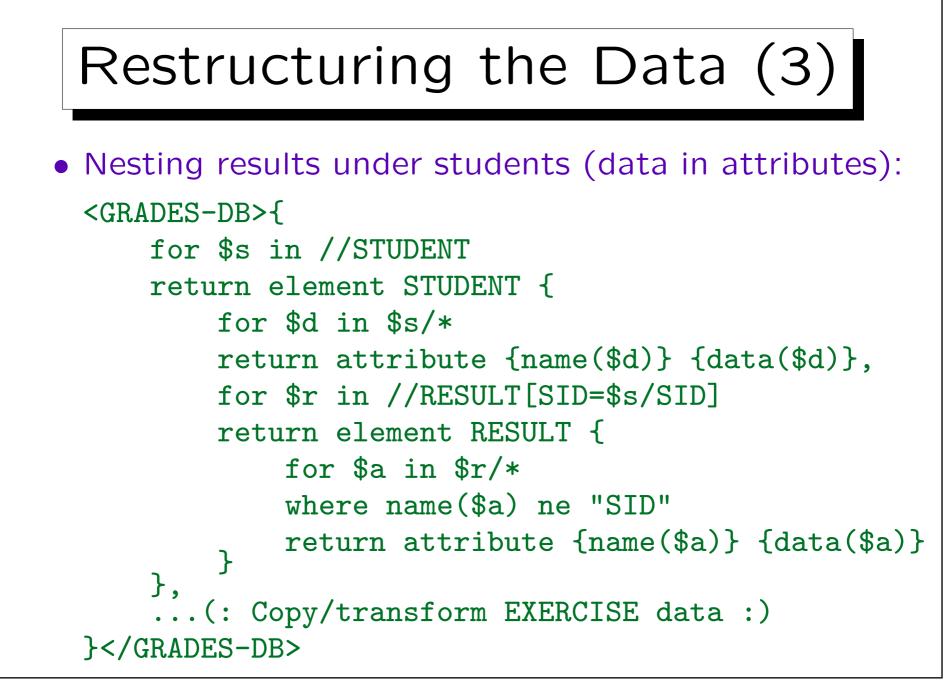


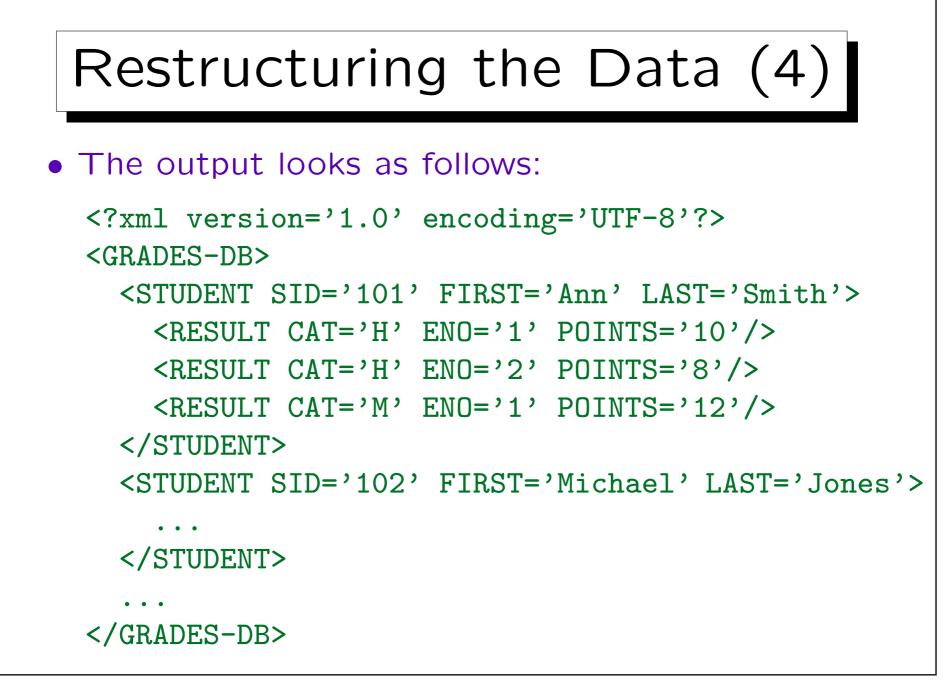




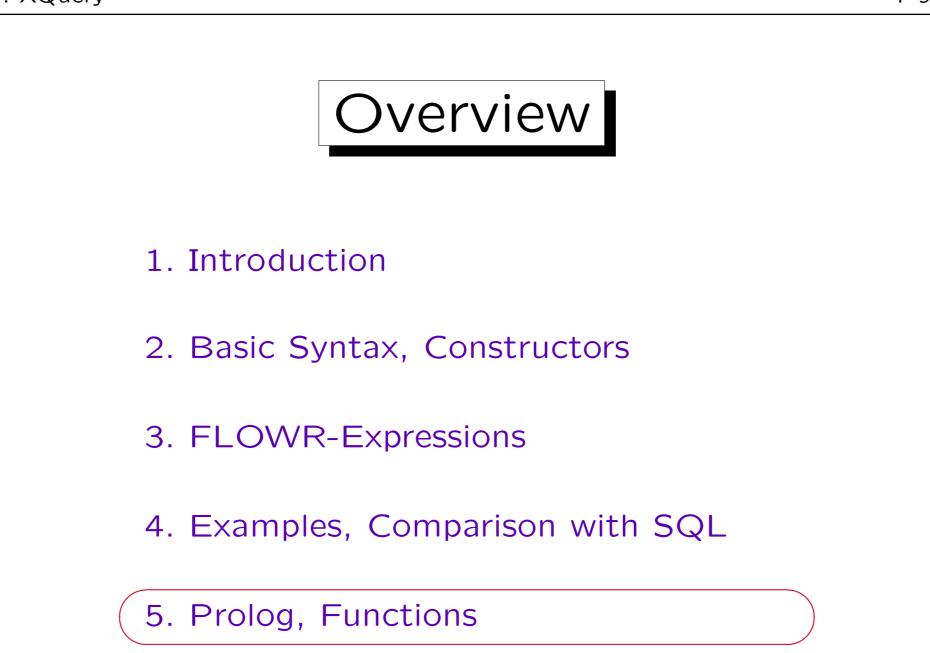


Stefan Brass: XML und Datenbanken



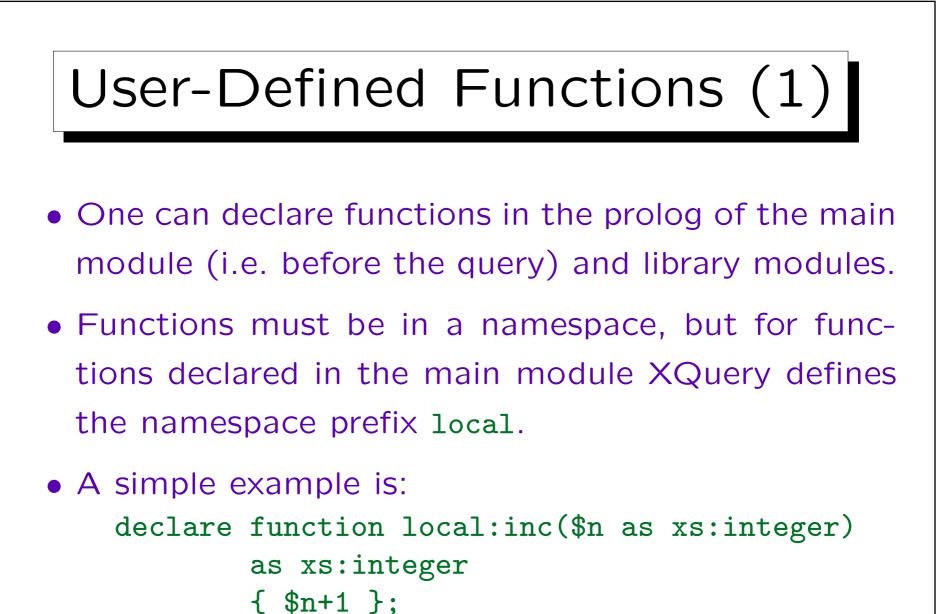


Stefan Brass: XML und Datenbanken

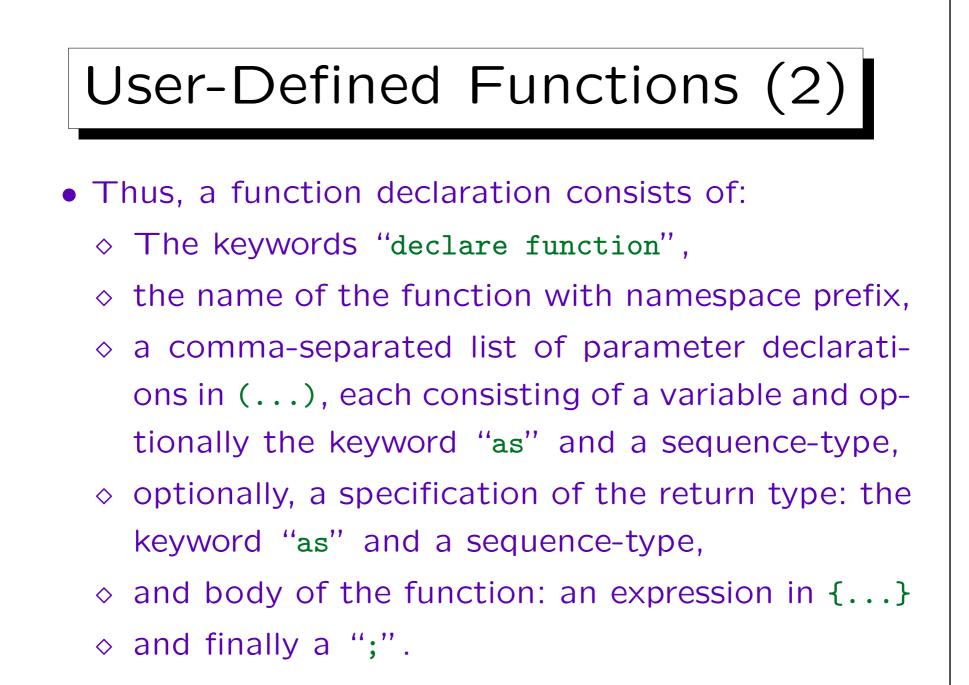


Namespaces

- Namespaces can be defined in the Prolog:
 - ◇ declare namespace Prefix = "URI";
 - ◇ declare default element namespace "URI";
 - ◇ declare default function namespace "URI";
- The following namespace prefixes are predeclared:
 - ◊ xml = http://www.w3.org/XML/1998/namespace
 - ◊ xs = http://www.w3.org/2001/XMLSchema
 - ◇ xsi = http://www.w3.org/2001/XMLSchema-instance
 - ◊ fn = http://www.w3.org/2005/xpath-functions
 - ◇ local = http:// www.w3.org/2005/xquery-local-functions



local:inc(1) (: This is the query :)





- If the types are not specified, item()* is assumed (the most general type).
- Instead of a function body, one can also specify the keyword "external".

It is implementation-dependent if an how functions written in some other language (e.g., C) can be linked to the XQuery evaluator.

• Functions can be recursive.

And also mutually recursive. XQuery becomes in this way computationally complete, but then it cannot guarantee termination.