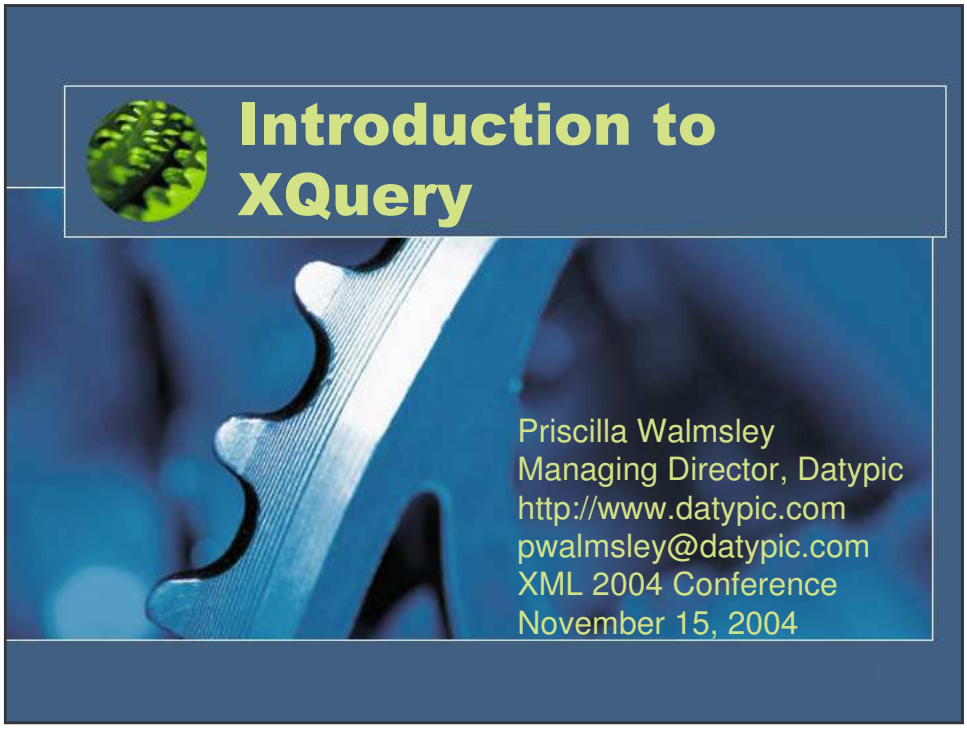
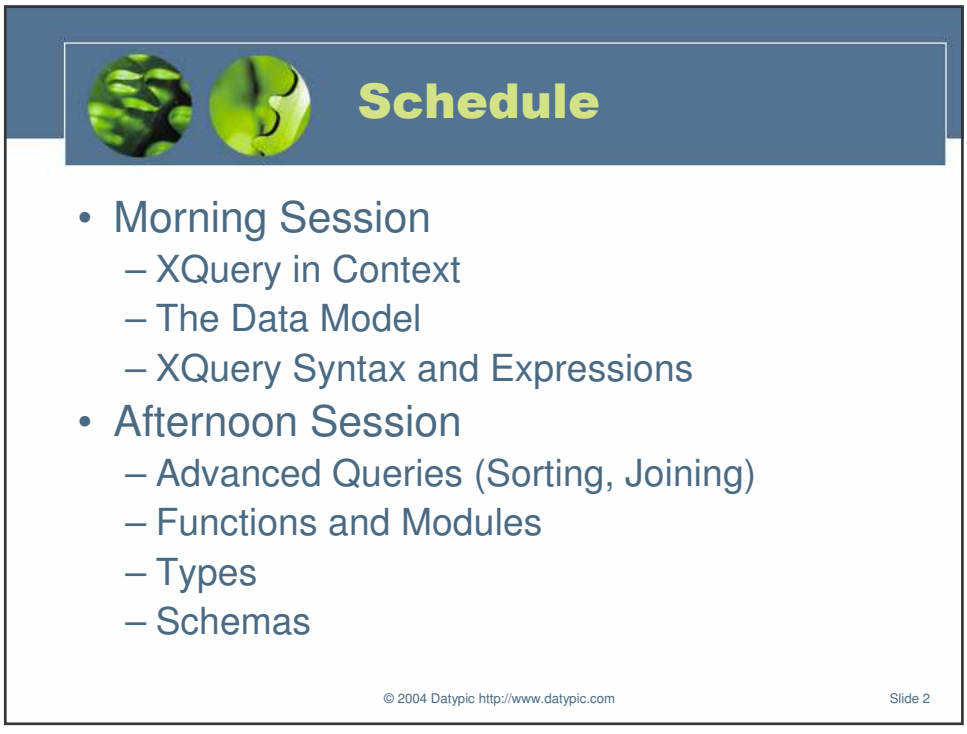
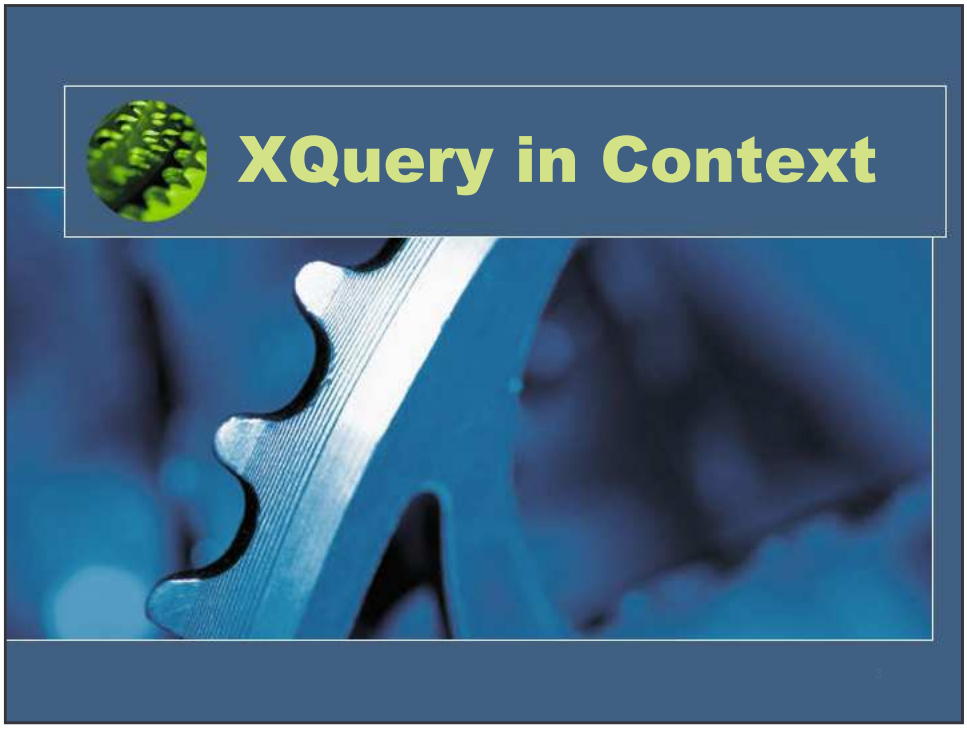

Introduction to XQuery
Priscilla Walmsley
Managing Director, Datypic
http://www.datypic.com
pwalmsley@datypic.com
XML 2004 Conference
November 15, 2004

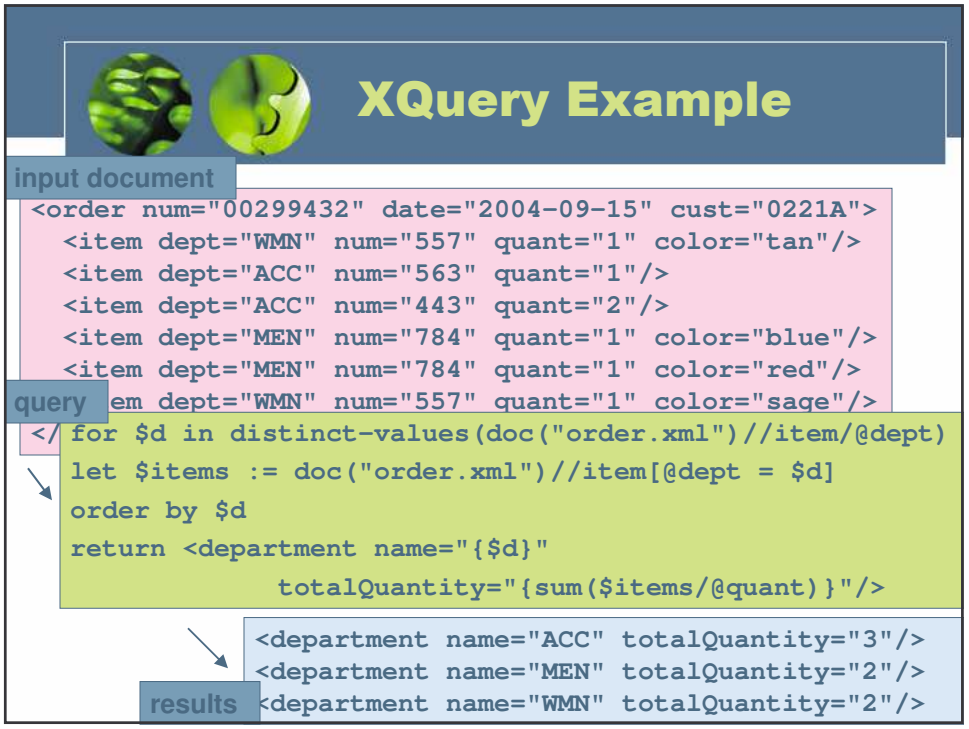

Schedule
• Morning Session
– XQuery in Context
– The Data Model
– XQuery Syntax and Expressions
• Afternoon Session
– Advanced Queries (Sorting, Joining)
– Functions and Modules
– Types
– Schemas
© 2004 Datypic http://www.datypic.com
Slide 2

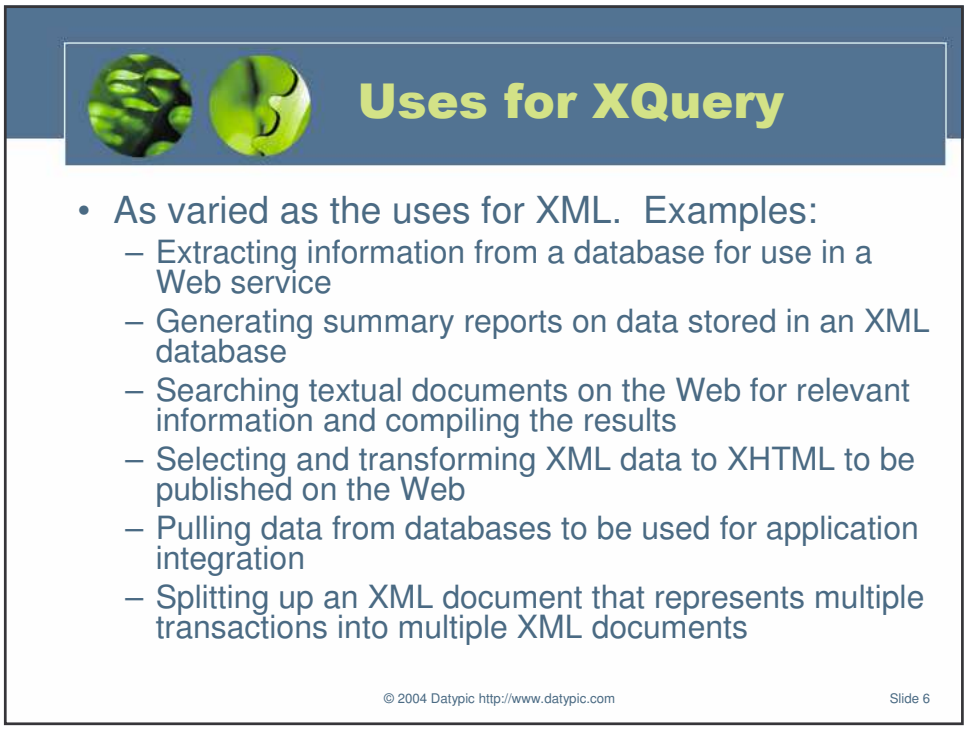


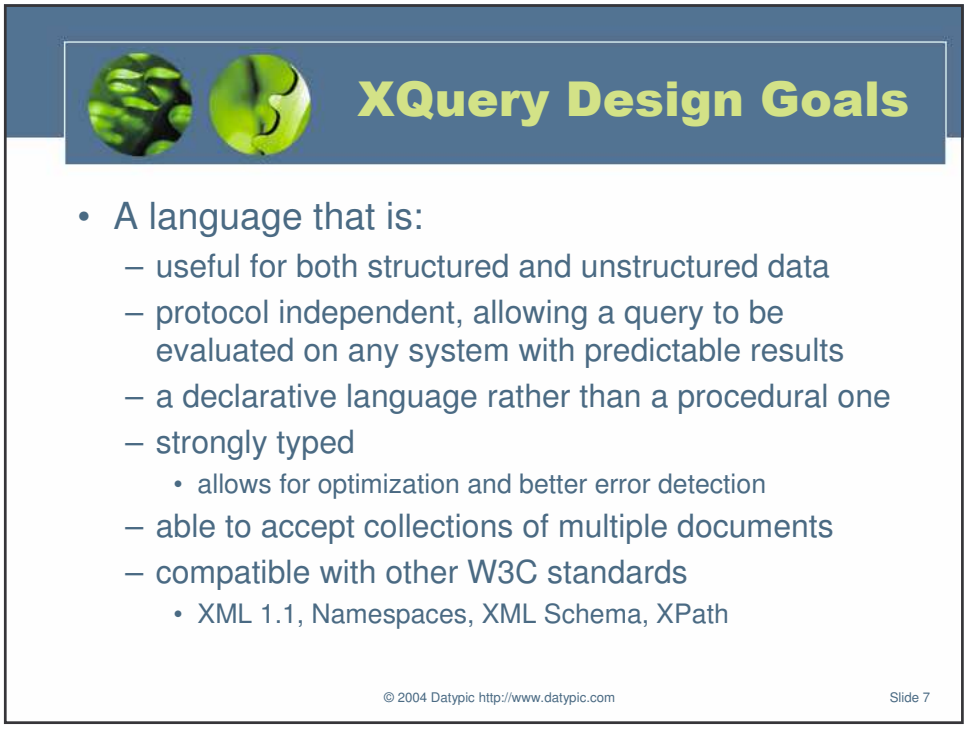
What is XQuery?

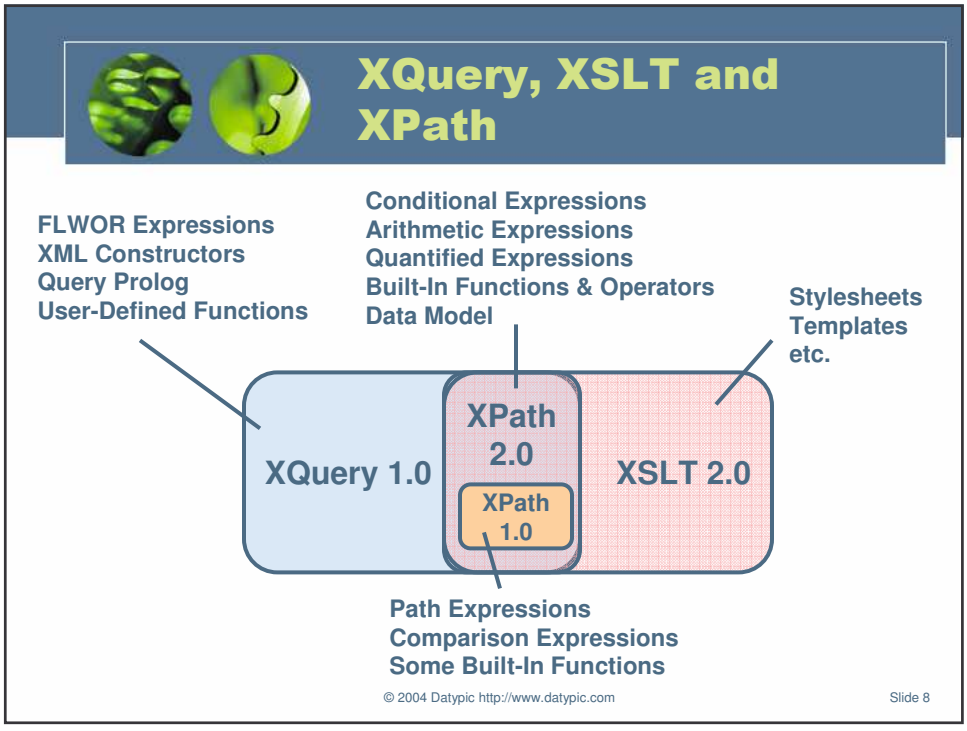
- A query language that allows you to:
  - select elements/attributes from input documents
  - join data from multiple input documents
  - make modifications to the data
  - calculate new data
  - add new elements/attributes to the results
  - sort your results

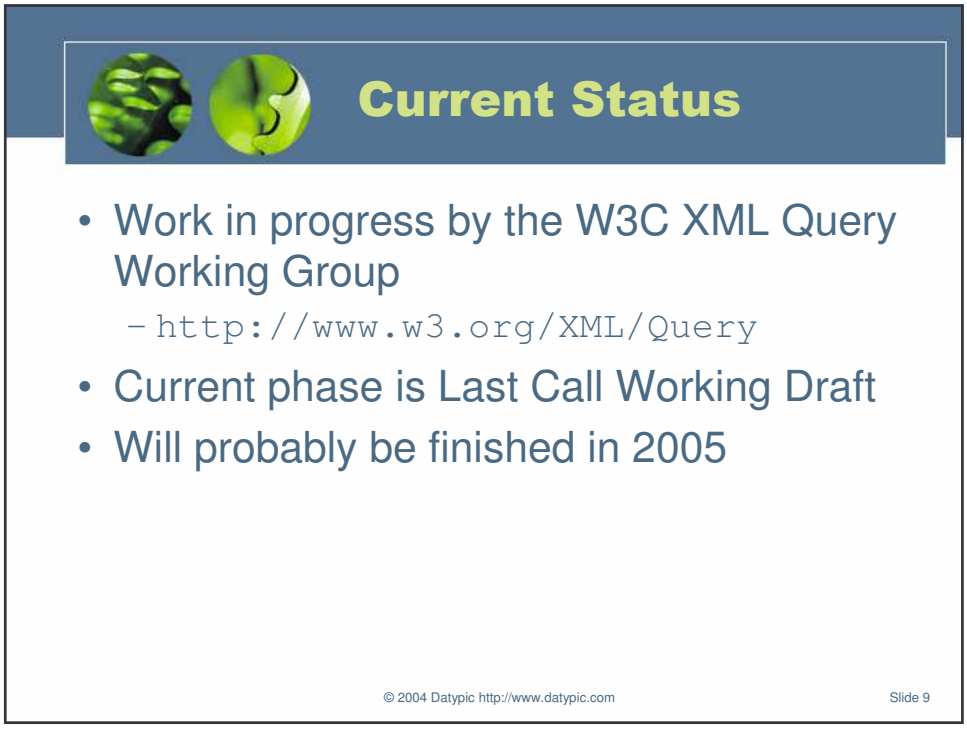
 Slide 4

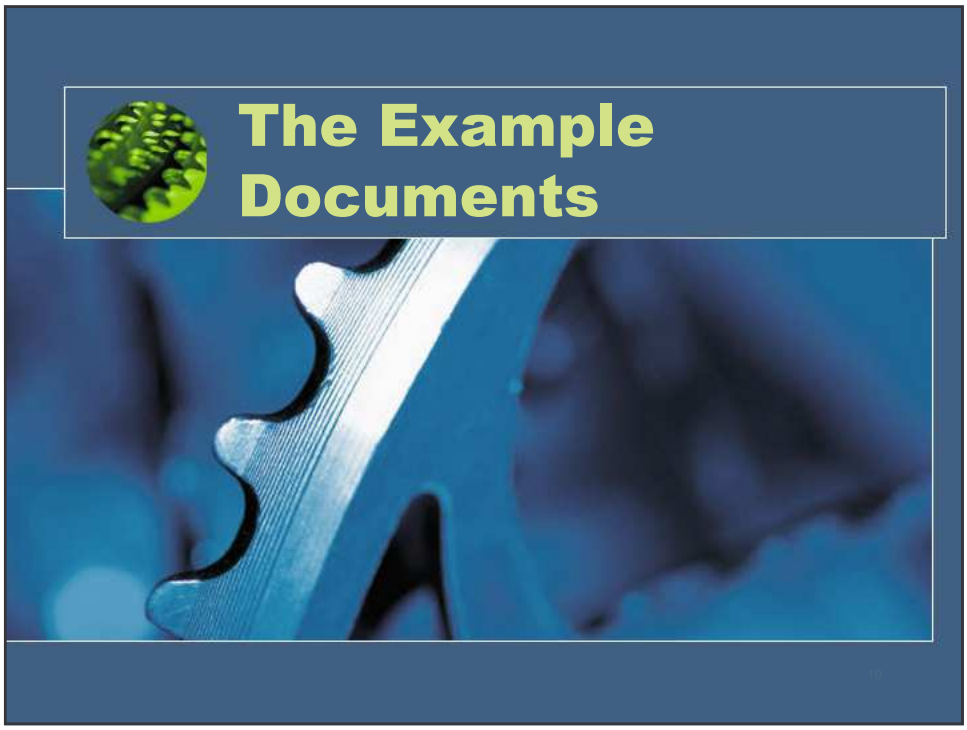

XQuery Example
input document
<order num="00299432" date="2004-09-15" cust="0221A">
<item dept="WMN" num="557" quant="1" color="tan"/>
<item dept="ACC" num="563" quant="1"/>
<item dept="ACC" num="443" quant="2"/>
<item dept="MEN" num="784" quant="1" color="blue"/>
<item dept="MEN" num="784" quant="1" color="red"/>
query
for $d in distinct-values(doc("order.xml")//item/@dept)
let $items := doc("order.xml")//item[@dept = $d]
order by $d
return <department name="{$d}"
totalQuantity="{sum($items/@quant)}"/>
<department name="ACC" totalQuantity="3"/>
<department name="MEN" totalQuantity="2"/>
<department name="WMN" totalQuantity="2"/>
results

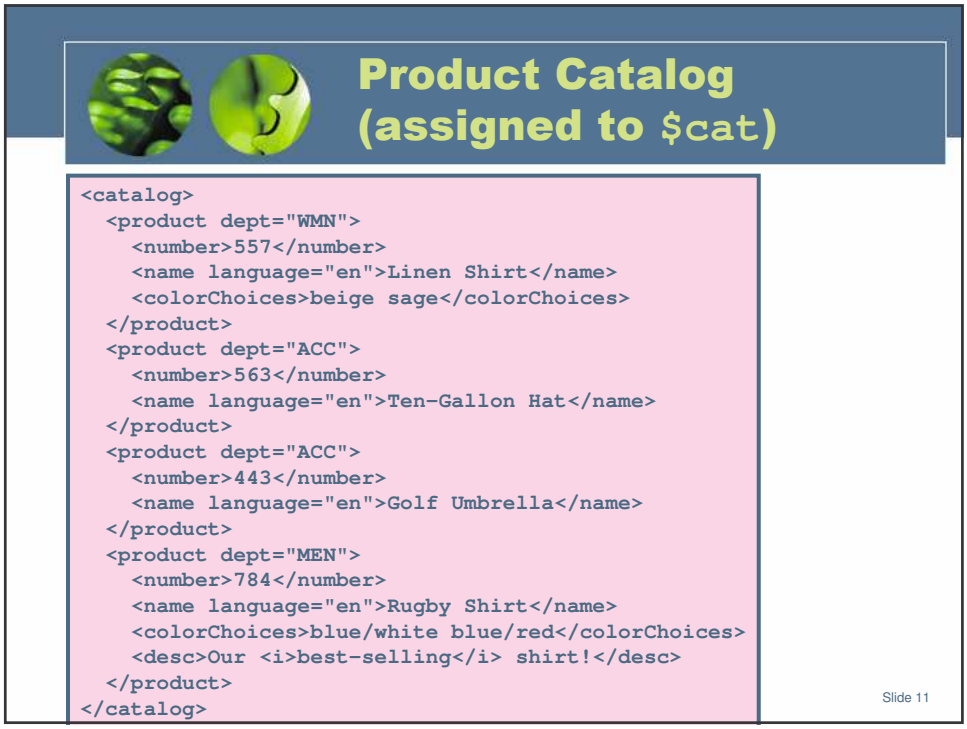

Uses for XQuery
• As varied as the uses for XML. Examples:
– Extracting information from a database for use in a Web service
– Generating summary reports on data stored in an XML database
– Searching textual documents on the Web for relevant information and compiling the results
– Selecting and transforming XML data to XHTML to be published on the Web
– Pulling data from databases to be used for application integration
– Splitting up an XML document that represents multiple transactions into multiple XML documents
© 2004 Datypic http://www.datypic.com
Slide 6

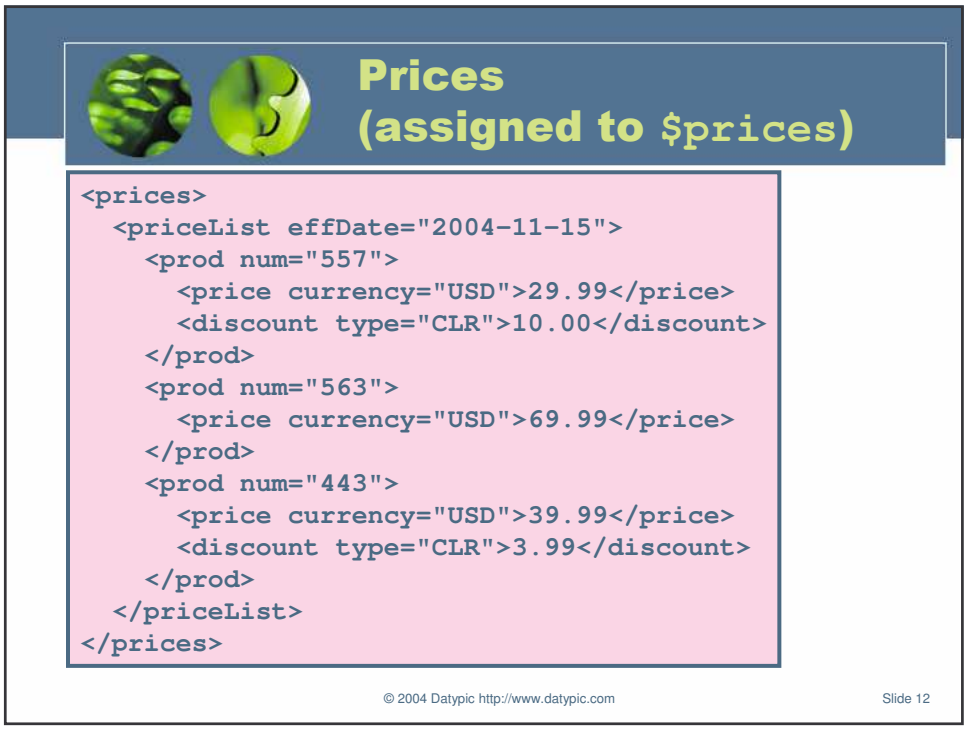

XQuery Design Goals
• A language that is:
– useful for both structured and unstructured data
– protocol independent, allowing a query to be evaluated on any system with predictable results
– a declarative language rather than a procedural one
– strongly typed
• allows for optimization and better error detection
– able to accept collections of multiple documents
– compatible with other W3C standards
• XML 1.1, Namespaces, XML Schema, XPath
© 2004 Datypic http://www.datypic.com
Slide 7

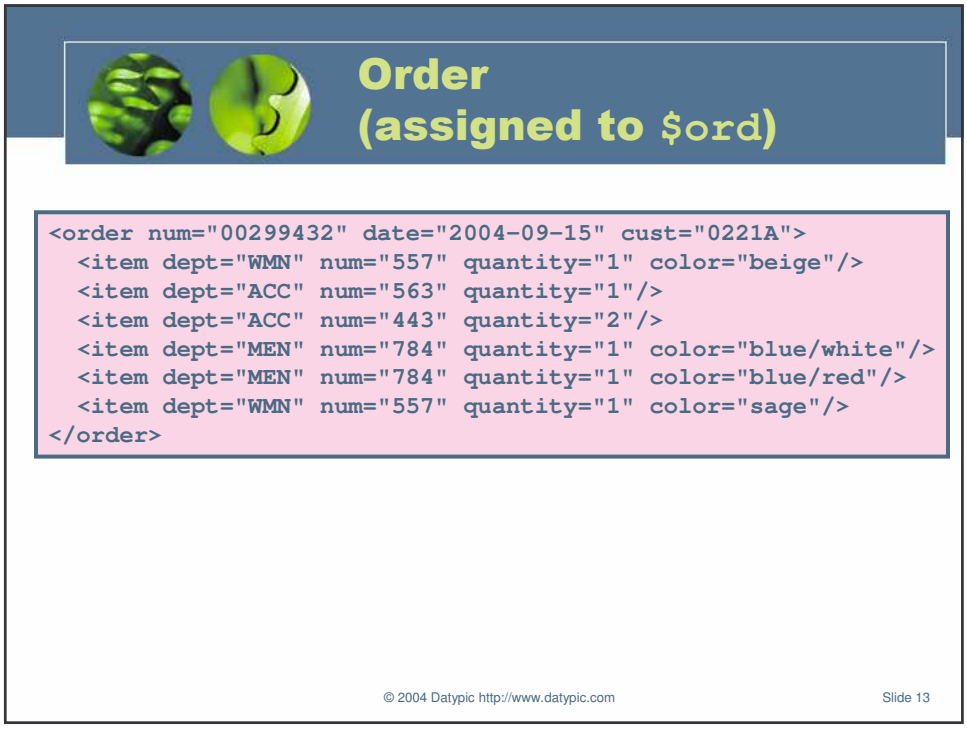

XQuery, XSLT and XPath
FLWOR Expressions
XML Constructors
Query Prolog
User-Defined Functions
Conditional Expressions
Arithmetic Expressions
Quantified Expressions
Built-In Functions & Operators
Data Model
Stylesheets
Templates
etc.
XQuery 1.0
XPath 2.0
XPath 1.0
XSLT 2.0
Path Expressions
Comparison Expressions
Some Built-In Functions
© 2004 Datypic http://www.datypic.com
Slide 8

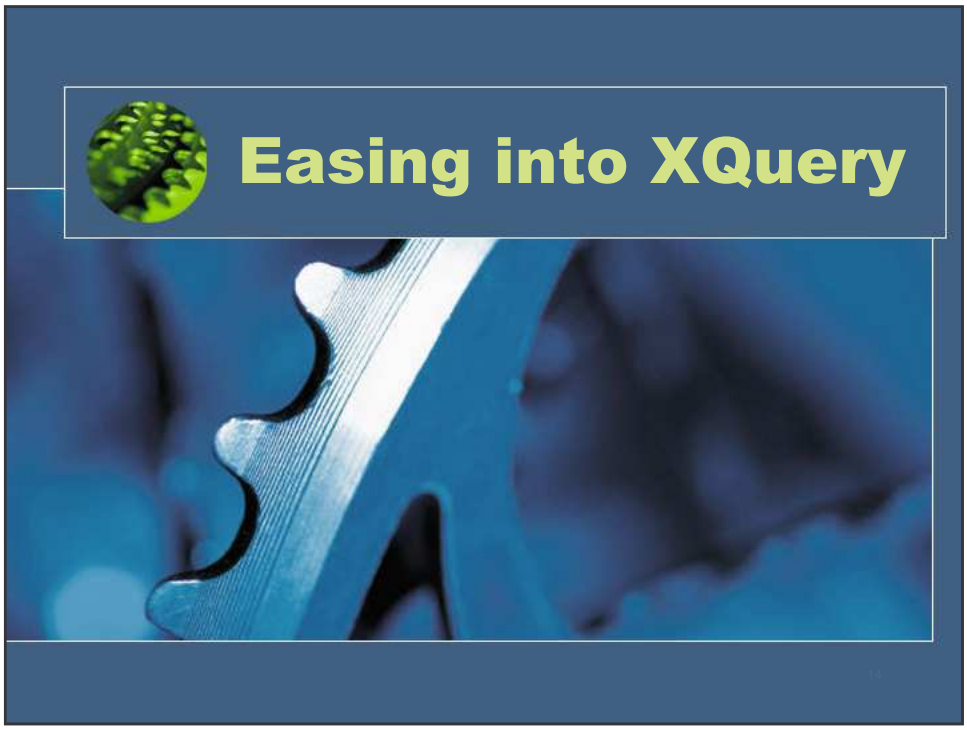

Current Status
• Work in progress by the W3C XML Query Working Group
– http://www.w3.org/XML/Query
• Current phase is Last Call Working Draft
• Will probably be finished in 2005
© 2004 Datypic http://www.datypic.com
Slide 9

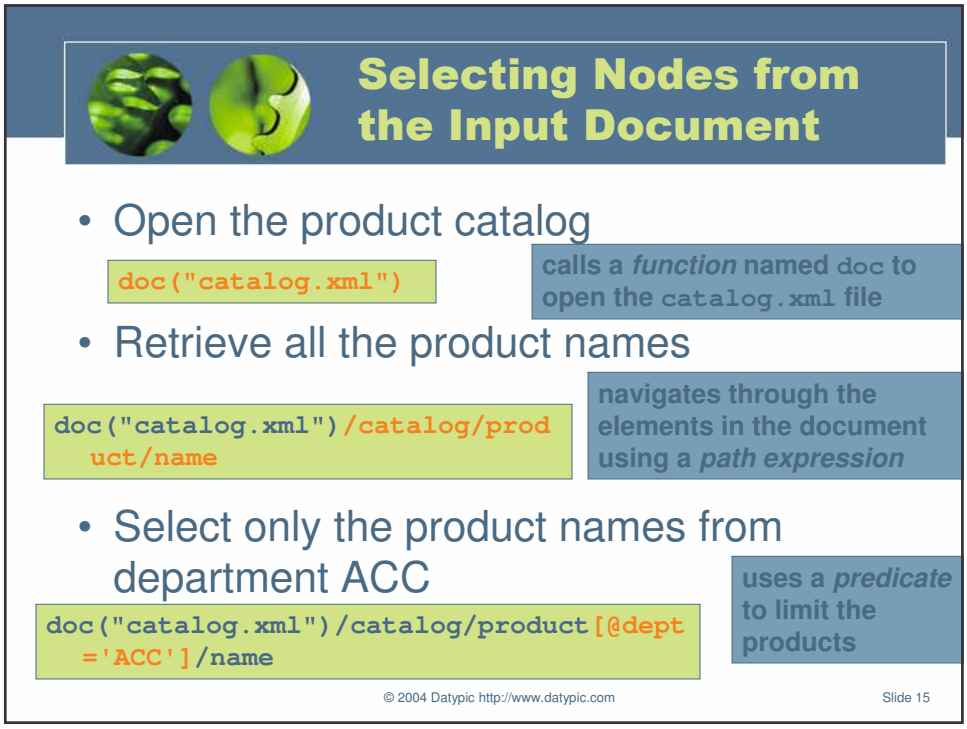

The Example Documents

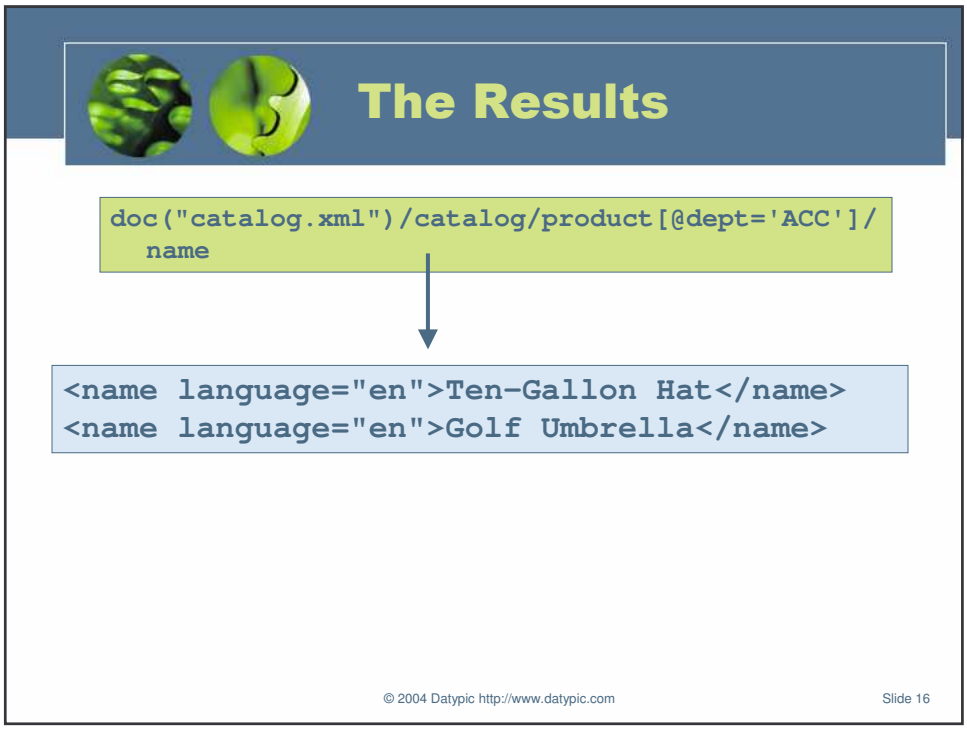

Product Catalog
(assigned to $cat)
<catalog>
<product dept="WMN">
<number>557</number>
<name language="en">Linen Shirt</name>
<colorChoices>beige sage</colorChoices>
</product>
<product dept="ACC">
<number>563</number>
<name language="en">Ten-Gallon Hat</name>
</product>
<product dept="ACC">
<number>443</number>
<name language="en">Golf Umbrella</name>
</product>
<product dept="MEN">
<number>784</number>
<name language="en">Rugby Shirt</name>
<colorChoices>blue/white blue/red</colorChoices>
<desc>Our best-selling shirt!</desc>
</product>
</catalog>
Slide 11

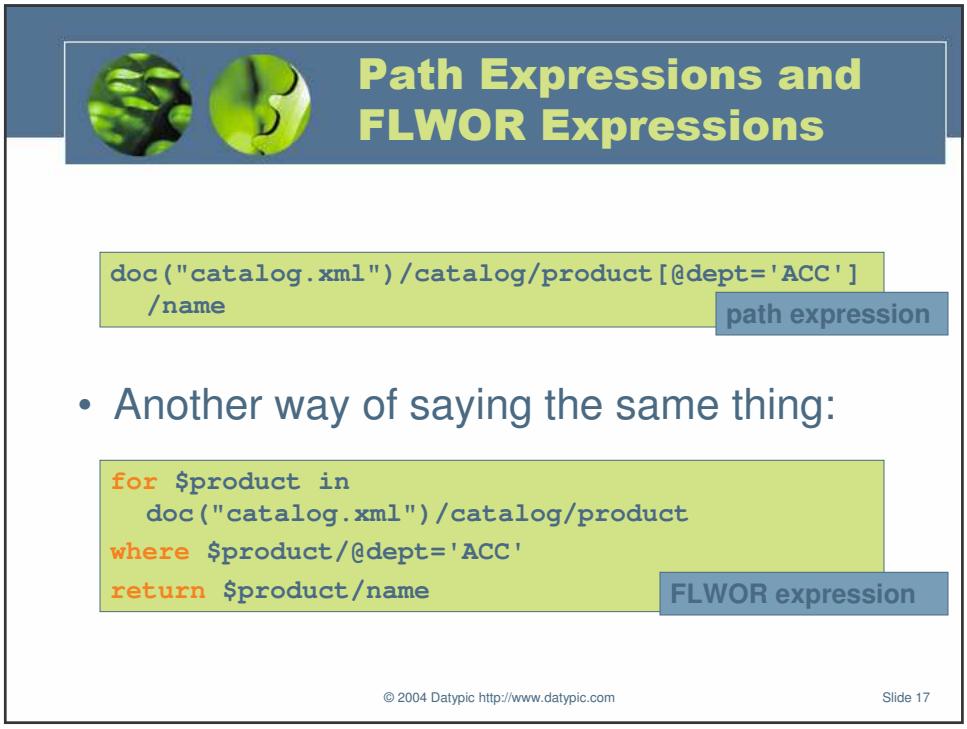

Prices
(assigned to $prices)
<prices>
<priceList effDate="2004-11-15">
<prod num="557">
<price currency="USD">29.99</price>
<discount type="CLR">10.00</discount>
</prod>
<prod num="563">
<price currency="USD">69.99</price>
</prod>
<prod num="443">
<price currency="USD">39.99</price>
<discount type="CLR">3.99</discount>
</prod>
</priceList>
</prices>
© 2004 Datypic http://www.datypic.com
Slide 12

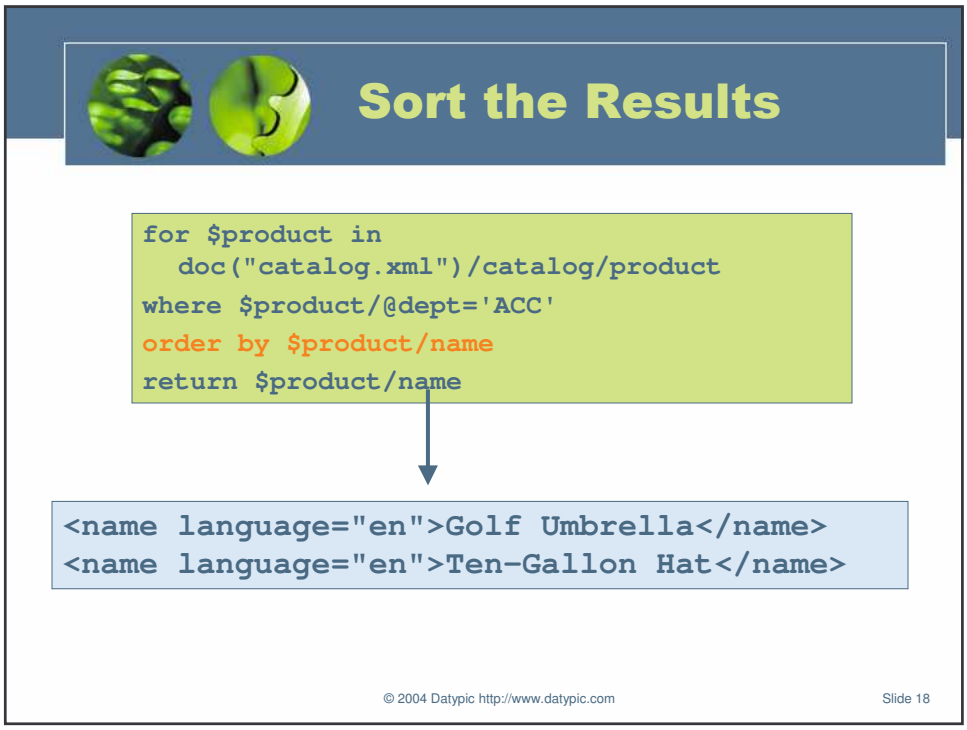

Order
(assigned to $ord)
<order num="00299432" date="2004-09-15" cust="0221A">
<item dept="WMN" num="557" quantity="1" color="beige"/>
<item dept="ACC" num="563" quantity="1"/>
<item dept="ACC" num="443" quantity="2"/>
<item dept="MEN" num="784" quantity="1" color="blue/white"/>
<item dept="MEN" num="784" quantity="1" color="blue/red"/>
<item dept="WMN" num="557" quantity="1" color="sage"/>
</order>
© 2004 Datypic http://www.datypic.com
Slide 13

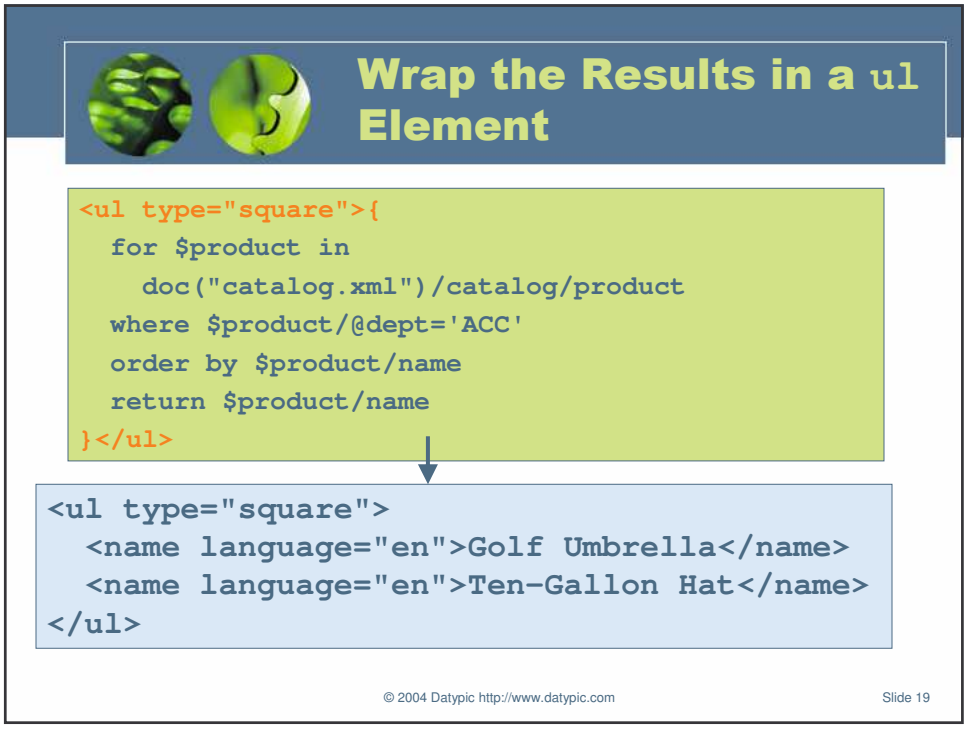

Easing into XQuery


Selecting Nodes from the Input Document
• Open the product catalog
doc("catalog.xml")
calls a function named doc to open the catalog.xml file
• Retrieve all the product names
doc("catalog.xml")/catalog/product/name
navigates through the elements in the document using a path expression
• Select only the product names from department ACC
doc("catalog.xml")/catalog/product[@dept='ACC']/name
uses a predicate to limit the products
© 2004 Datypic http://www.datypic.com
Slide 15

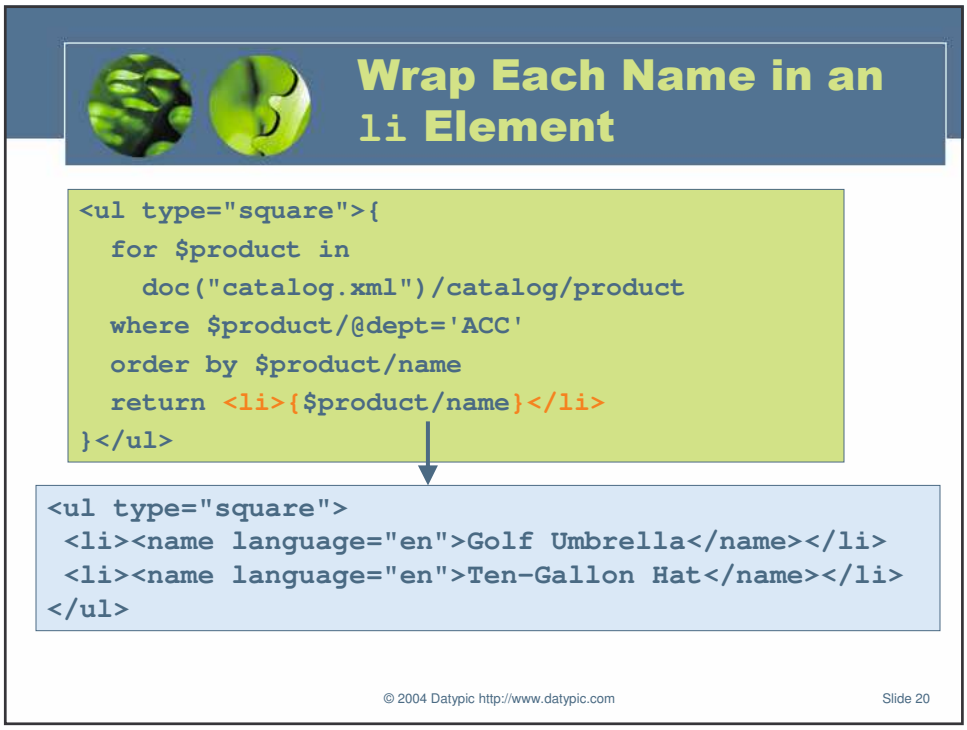

The Results
doc("catalog.xml")/catalog/product[@dept='ACC']/name
<name language="en">Ten-Gallon Hat</name>
<name language="en">Golf Umbrella</name>
© 2004 Datypic http://www.datypic.com
Slide 16


Path Expressions and FLWOR Expressions
doc("catalog.xml")/catalog/product[@dept='ACC']
/name
path expression
• Another way of saying the same thing:
for $product in
doc("catalog.xml")/catalog/product
where $product/@dept='ACC'
return $product/name
FLWOR expression
© 2004 Datypic http://www.datypic.com
Slide 17

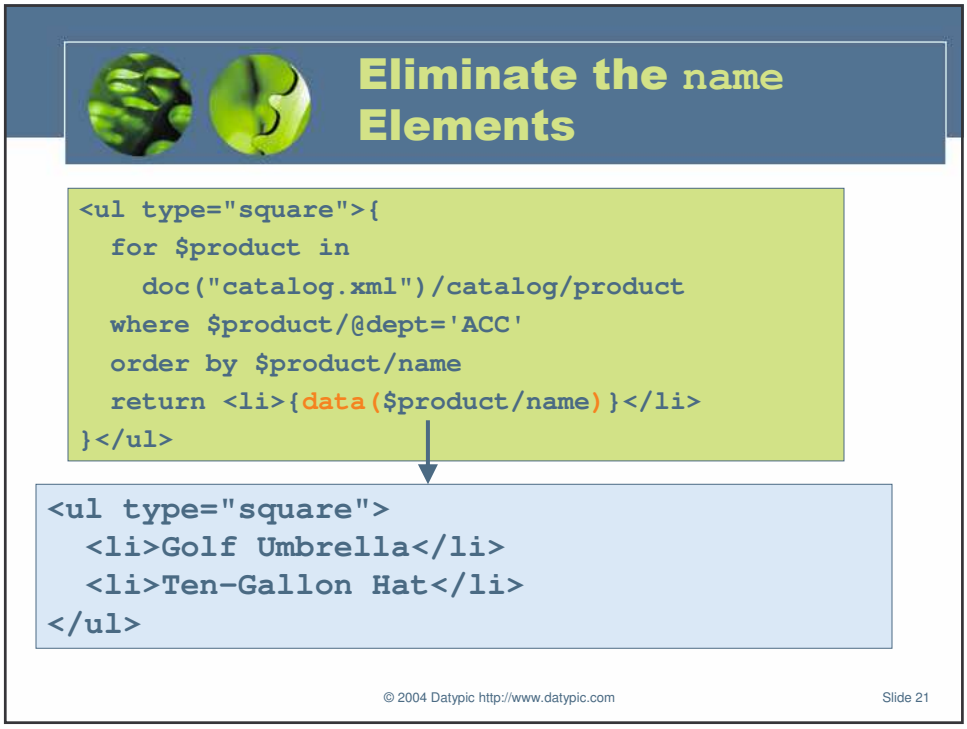

Sort the Results
for $product in
doc("catalog.xml")/catalog/product
where $product/@dept='ACC'
order by $product/name
return $product/name
<name language="en">Golf Umbrella</name>
<name language="en">Ten-Gallon Hat</name>
© 2004 Datypic http://www.datypic.com
Slide 18


Wrap the Results in a ul Element
<ul type="square">{
for $product in
doc("catalog.xml")/catalog/product
where $product/@dept='ACC'
order by $product/name
return $product/name
}
<ul type="square">
<name language="en">Golf Umbrella</name>
<name language="en">Ten-Gallon Hat</name>

© 2004 Datypic http://www.datypic.com
Slide 19

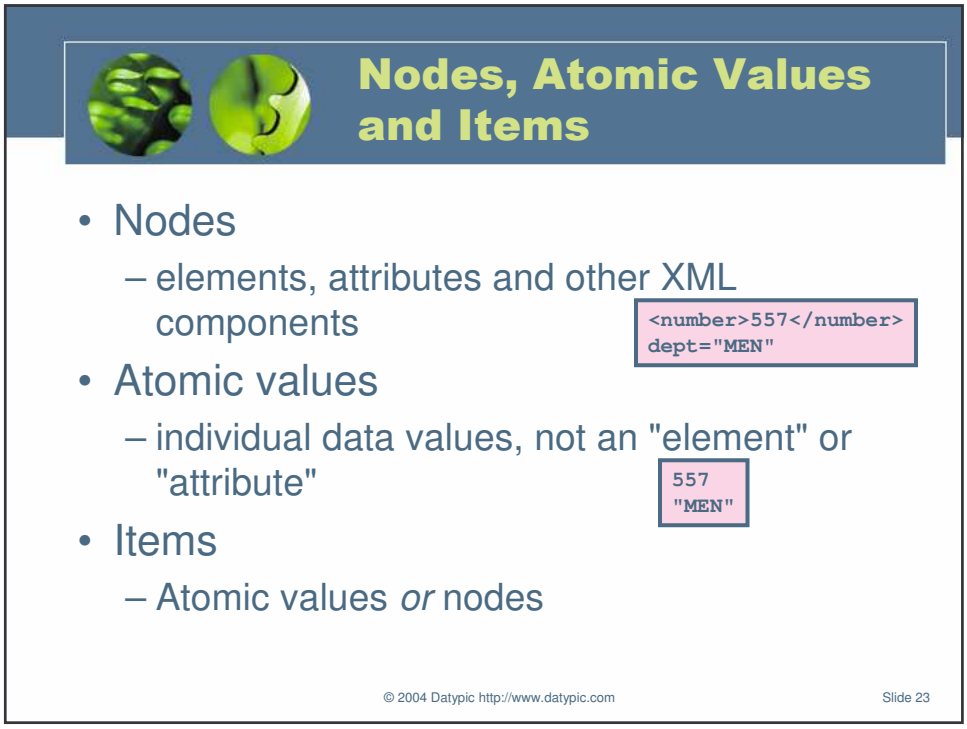

Wrap Each Name in an li Element
<ul type="square">{
for $product in
doc("catalog.xml")/catalog/product
where $product/@dept='ACC'
order by $product/name
return {$product/name}
}
<ul type="square">
<name language="en">Golf Umbrella</name>
<name language="en">Ten-Gallon Hat</name>

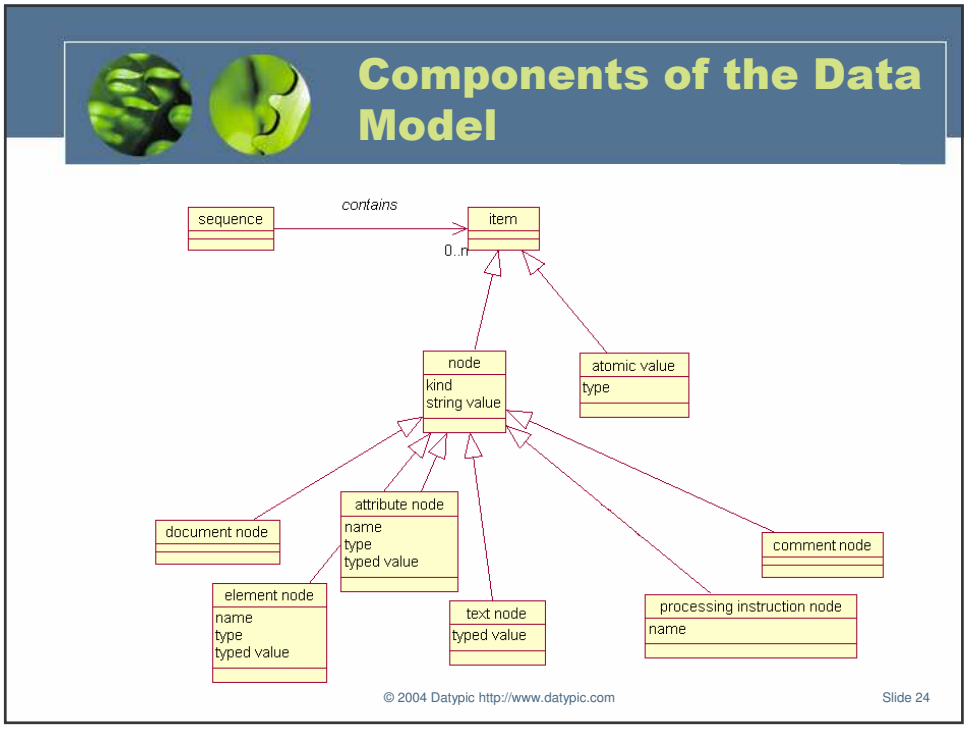
© 2004 Datypic http://www.datypic.com
Slide 20

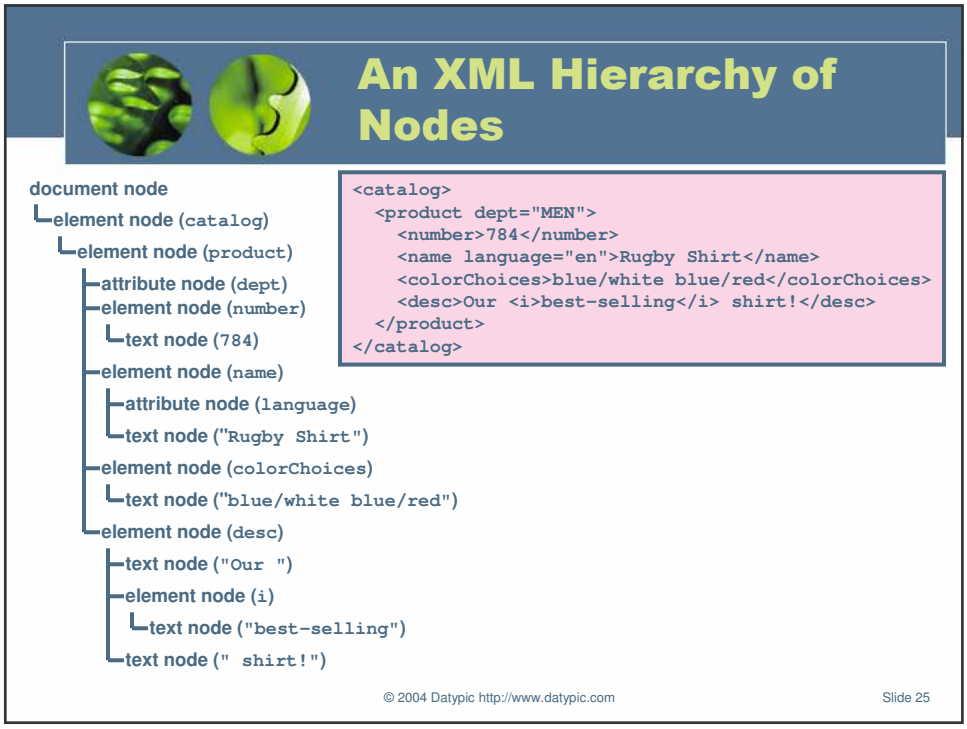

Eliminate the name Elements
<ul type="square">{
for $product in
doc("catalog.xml")/catalog/product
where $product/@dept='ACC'
order by $product/name
return {data($product/name)}
}
<ul type="square">
Golf Umbrella
Ten-Gallon Hat

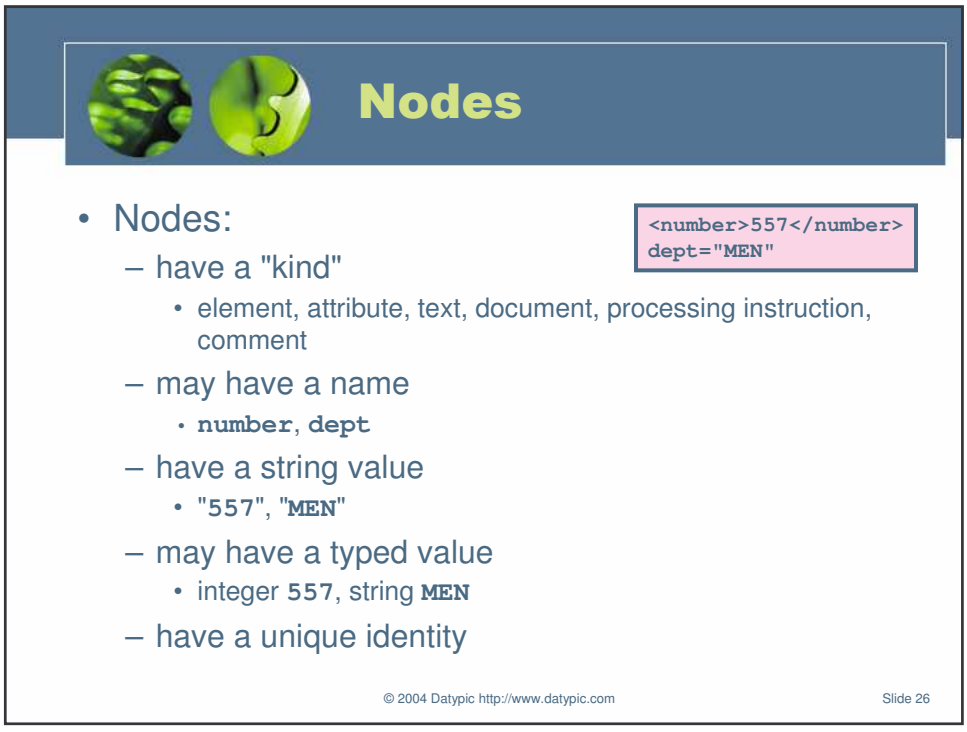
© 2004 Datypic http://www.datypic.com
Slide 21

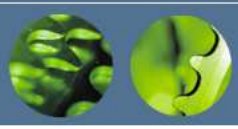

The Data Model


Nodes, Atomic Values and Items
• Nodes
– elements, attributes and other XML components
<number>557</number>
dept="MEN"
• Atomic values
– individual data values, not an "element" or "attribute"
557
"MEN"
• Items
– Atomic values *or* nodes
© 2004 Datypic http://www.datypic.com
Slide 23


Components of the Data Model
sequence
contains
item
0..n
node
kind
string value
atomic value
type
document node
attribute node
name
type
typed value
element node
name
type
typed value
text node
typed value
processing instruction node
name
comment node
© 2004 Datypic http://www.datypic.com
Slide 24


An XML Hierarchy of Nodes
document node
element node (catalog)
element node (product)
attribute node (dept)
element node (number)
text node (784)
element node (name)
attribute node (language)
text node ("Rugby Shirt")
element node (colorChoices)
text node ("blue/white blue/red")
element node (desc)
text node ("Our ")
element node (i)
text node ("best-selling")
text node (" shirt!")
<catalog>
<product dept="MEN">
<number>784</number>
<name language="en">Rugby Shirt</name>
<colorChoices>blue/white blue/red</colorChoices>
<desc>Our best-selling shirt!</desc>
</product>
</catalog>
© 2004 Datypic http://www.datypic.com
Slide 25

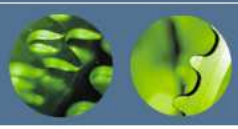

Nodes
Nodes:
have a "kind"
element, attribute, text, document, processing instruction, comment
may have a name
number, dept
have a string value
"557", "MEN"
may have a typed value
integer 557, string MEN
have a unique identity
<number>557</number>
dept="MEN"
© 2004 Datypic http://www.datypic.com
Slide 26



Family Relationships of Nodes

- Children
  - Element nodes can have zero, one or more children
    - Attributes are not considered children of an element node
- Parent
  - Each element and attribute node has one parent
    - Even though attributes are not children of elements, elements are parents of attributes!
- Ancestors
  - A node's parent, parent's parent, etc. all the way back to the document node
- Descendants
  - A node's children, children's children, etc.
- Siblings
  - Other nodes that have the same parent

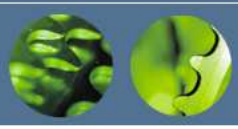
 Slide 27



Atomic Values

- Individual data values
  - no association with any particular node
- Every atomic value has a type
  - based on the XML Schema atomic types
  - e.g. **`xs:string`**, **`xs:integer`**
  - can also be the generic type **`xdt:untypedAtomic`**
    - for input data when no schema is in use

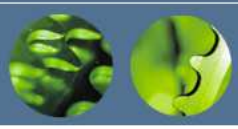
 Slide 28



How Atomic Values Are Created

- using a literal value
  - `"Catalog", 12`
- the result of a function
  - `count(//number)`
- explicitly extracting the value of a node
  - `data(<number>557</number>)`
- automatically extracting the value of a node
  - process is called *atomization*
  - `substring(<number>557</number>,1, 2)`

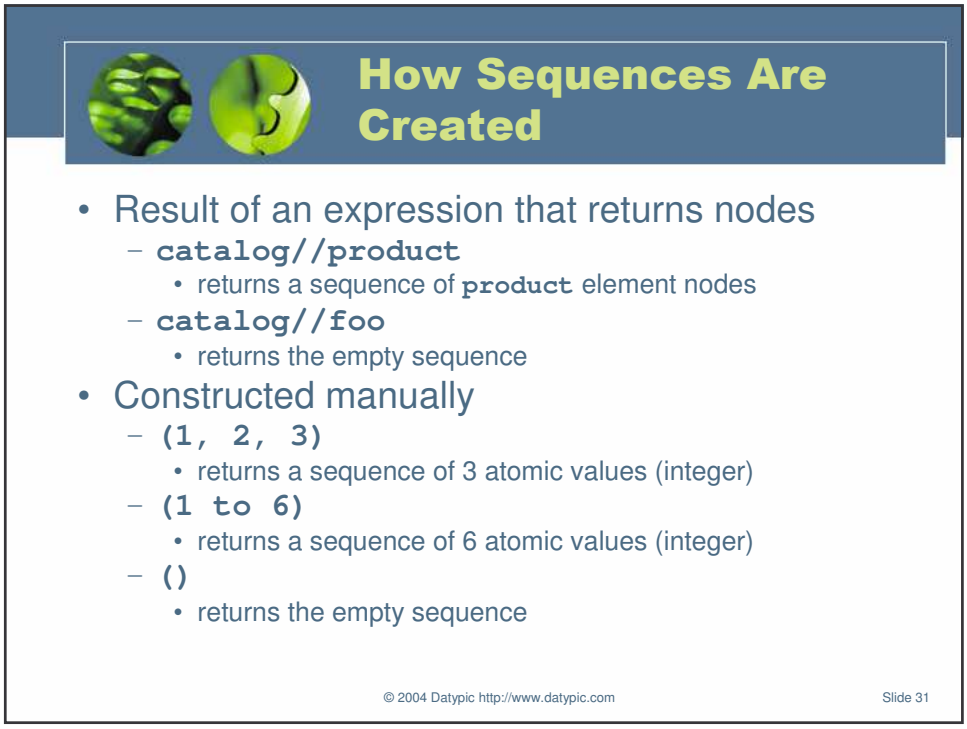
 Slide 29

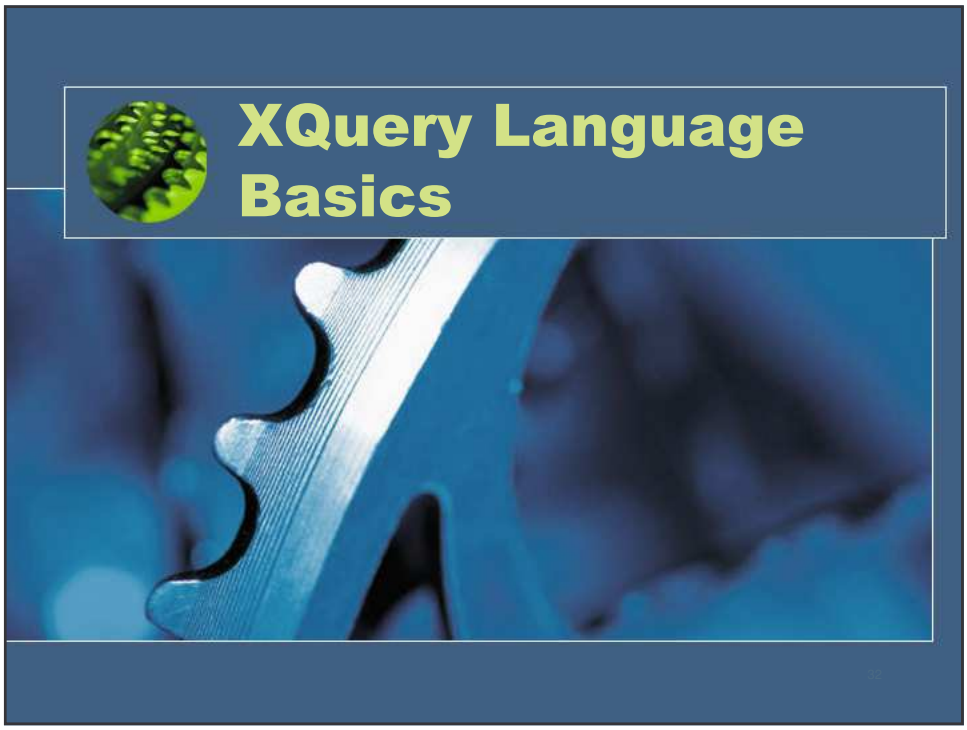


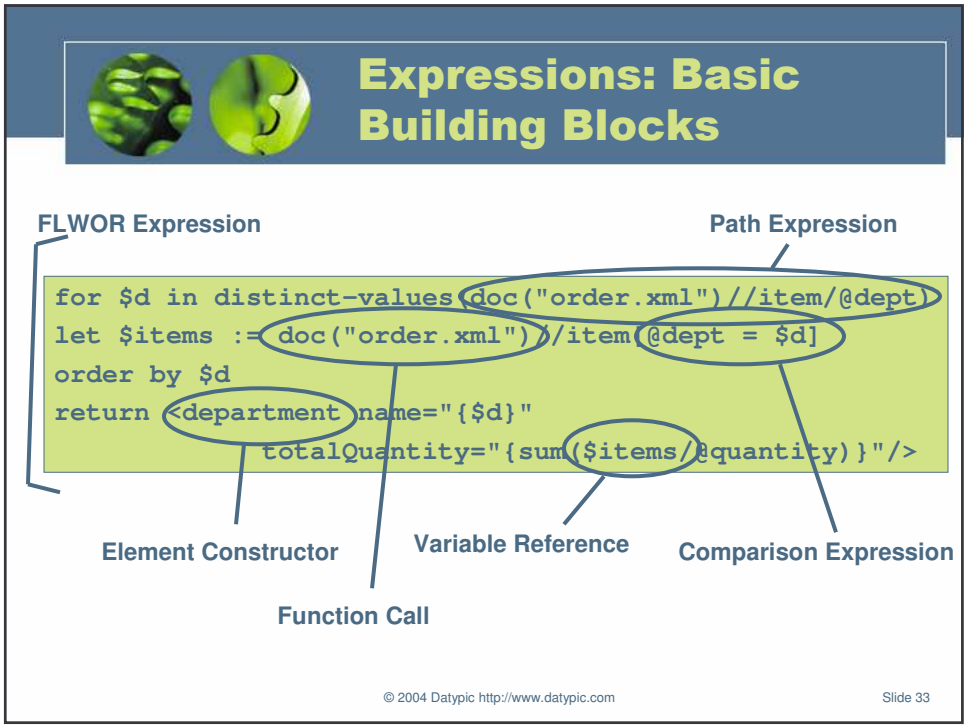
Sequences

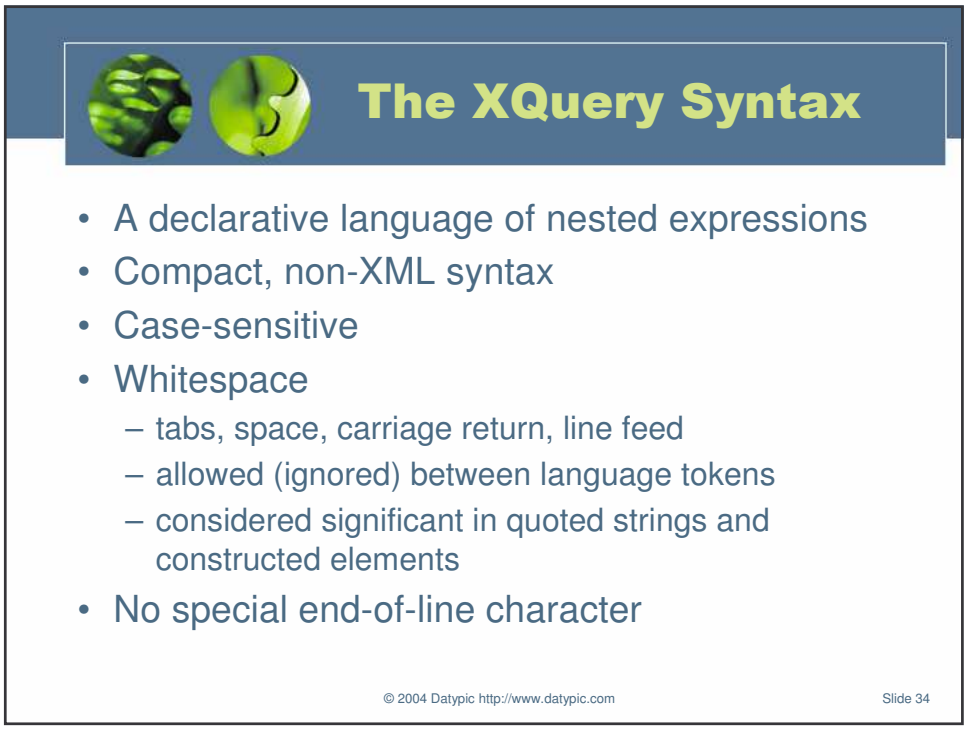
- Ordered lists of zero, one or more items
- A sequence of one item is exactly the same as the item itself
- A sequence of zero items is known as "the empty sequence"
  - different from zero, a zero-length string ("")
- There are no sequences within sequences

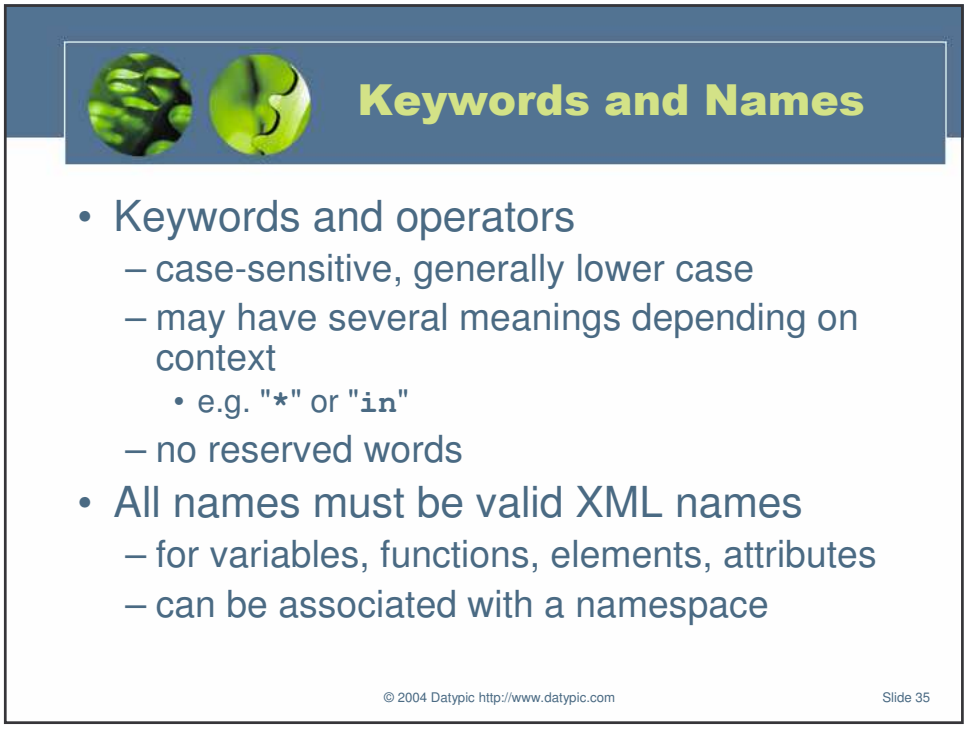
 Slide 30

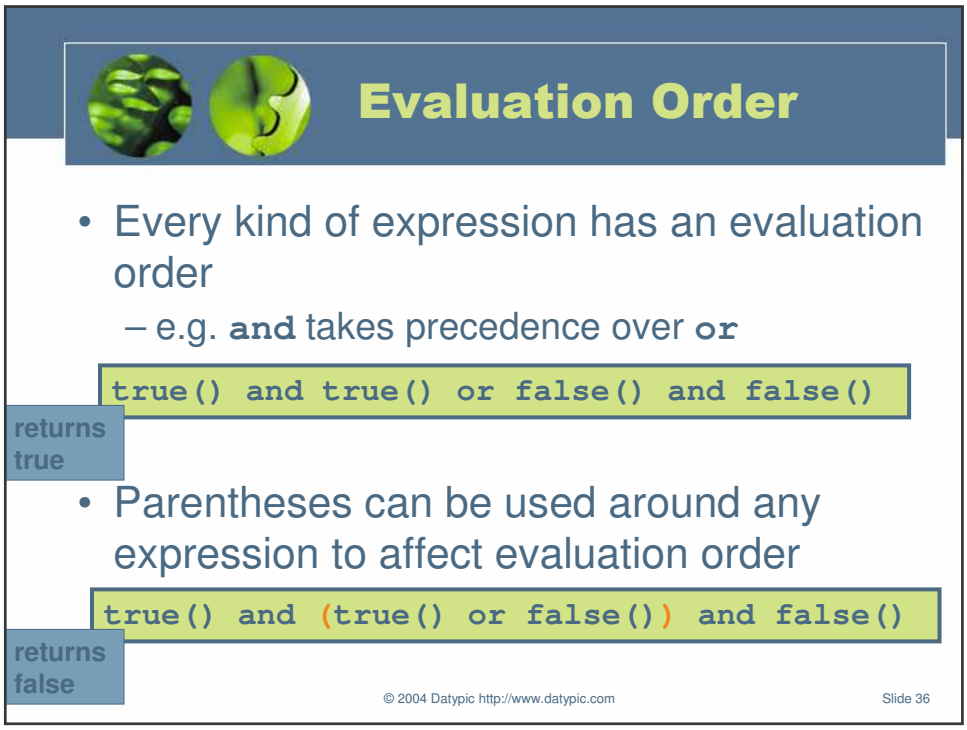

How Sequences Are Created
• Result of an expression that returns nodes
– catalog//product
• returns a sequence of product element nodes
– catalog//foo
• returns the empty sequence
• Constructed manually
– (1, 2, 3)
• returns a sequence of 3 atomic values (integer)
– (1 to 6)
• returns a sequence of 6 atomic values (integer)
– ()
• returns the empty sequence
© 2004 Datypic http://www.datypic.com
Slide 31

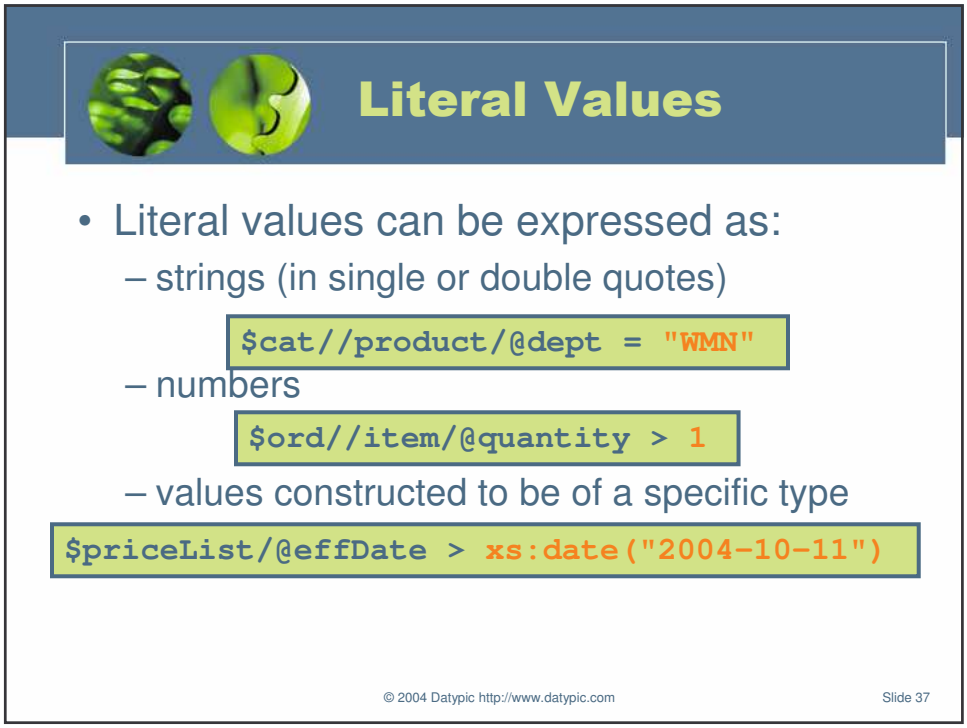

XQuery Language Basics

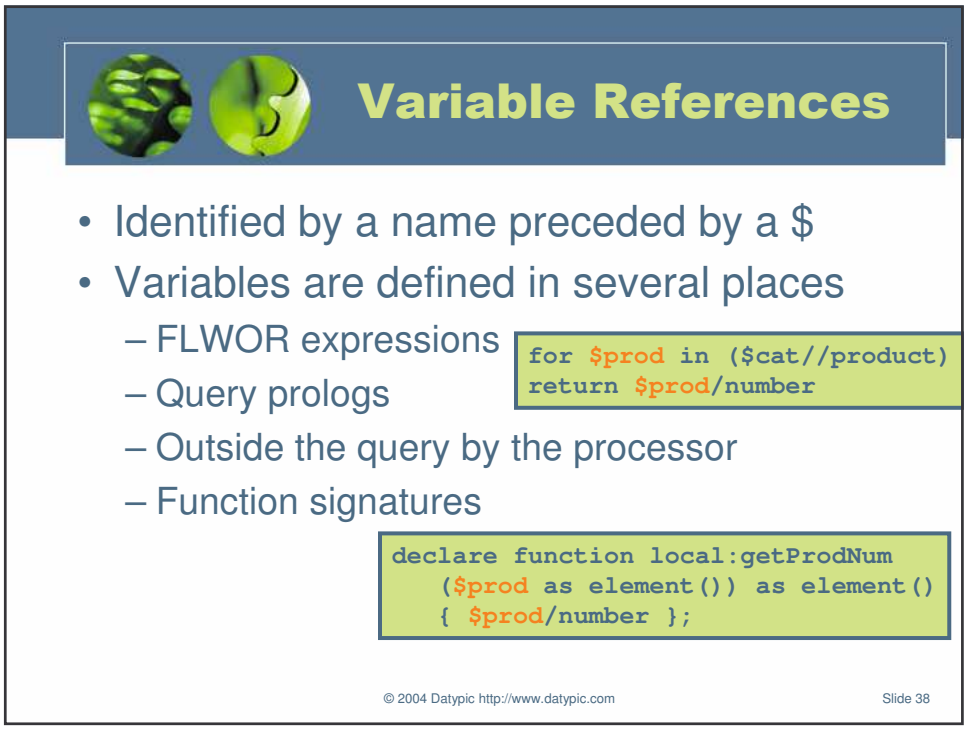

Expressions: Basic Building Blocks
FLWOR Expression
Path Expression
for $d in distinct-values(doc("order.xml")//item/@dept)
let $items := doc("order.xml")//item[@dept = $d]
order by $d
return <department name="{$d}"
totalQuantity="{sum($items/@quantity)}"/>
Element Constructor
Function Call
Variable Reference
Comparison Expression
© 2004 Datypic http://www.datypic.com
Slide 33

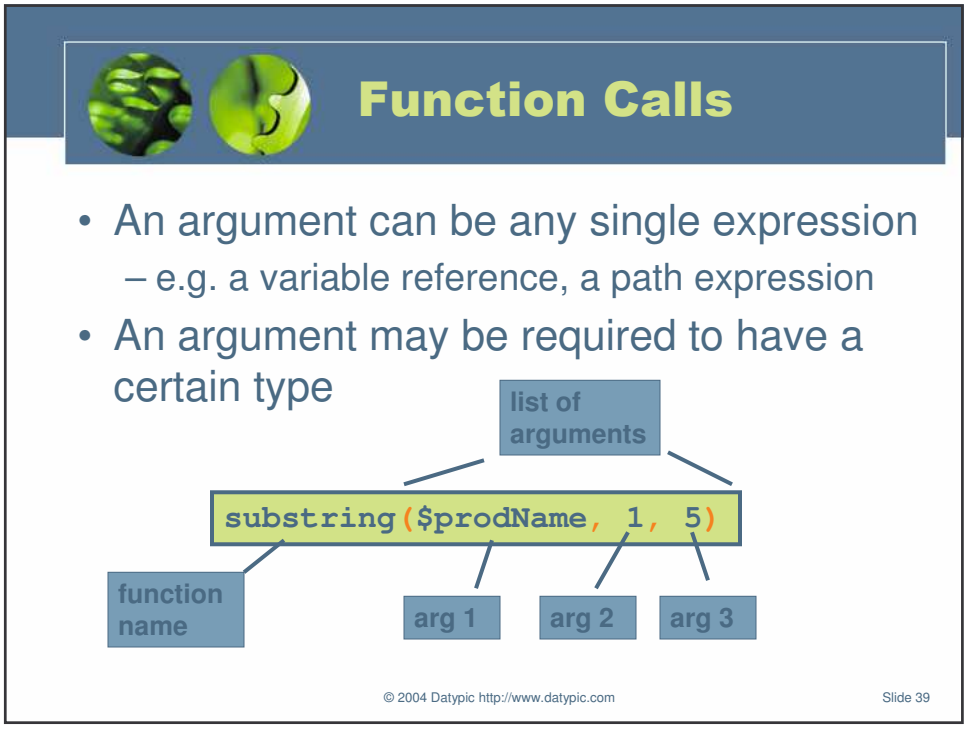

The XQuery Syntax
• A declarative language of nested expressions
• Compact, non-XML syntax
• Case-sensitive
• Whitespace
– tabs, space, carriage return, line feed
– allowed (ignored) between language tokens
– considered significant in quoted strings and constructed elements
• No special end-of-line character
© 2004 Datypic http://www.datypic.com
Slide 34

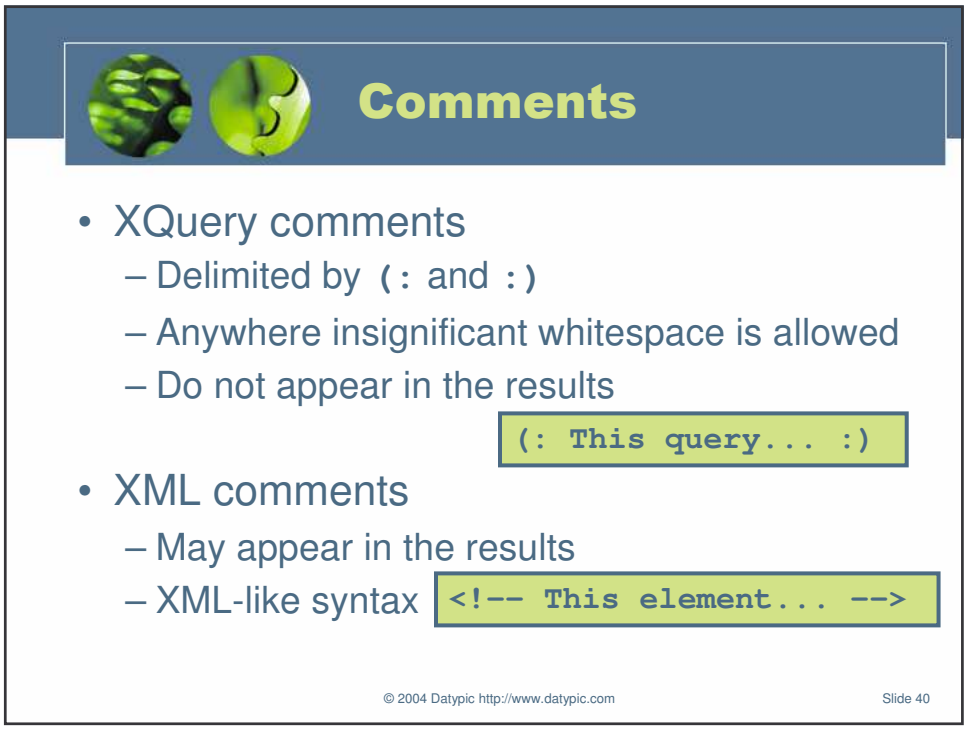

Keywords and Names
• Keywords and operators
– case-sensitive, generally lower case
– may have several meanings depending on context
• e.g. "*" or "in"
– no reserved words
• All names must be valid XML names
– for variables, functions, elements, attributes
– can be associated with a namespace
© 2004 Datypic http://www.datypic.com
Slide 35

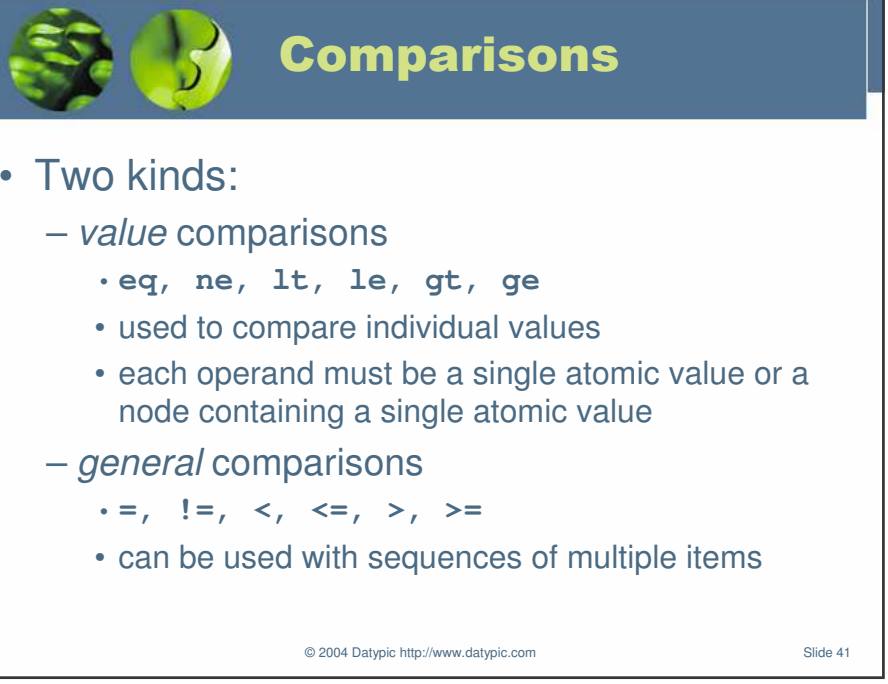

Evaluation Order
• Every kind of expression has an evaluation order
– e.g. and takes precedence over or
true() and true() or false() and false()
returns true
• Parentheses can be used around any expression to affect evaluation order
true() and (true() or false()) and false()
returns false
© 2004 Datypic http://www.datypic.com
Slide 36

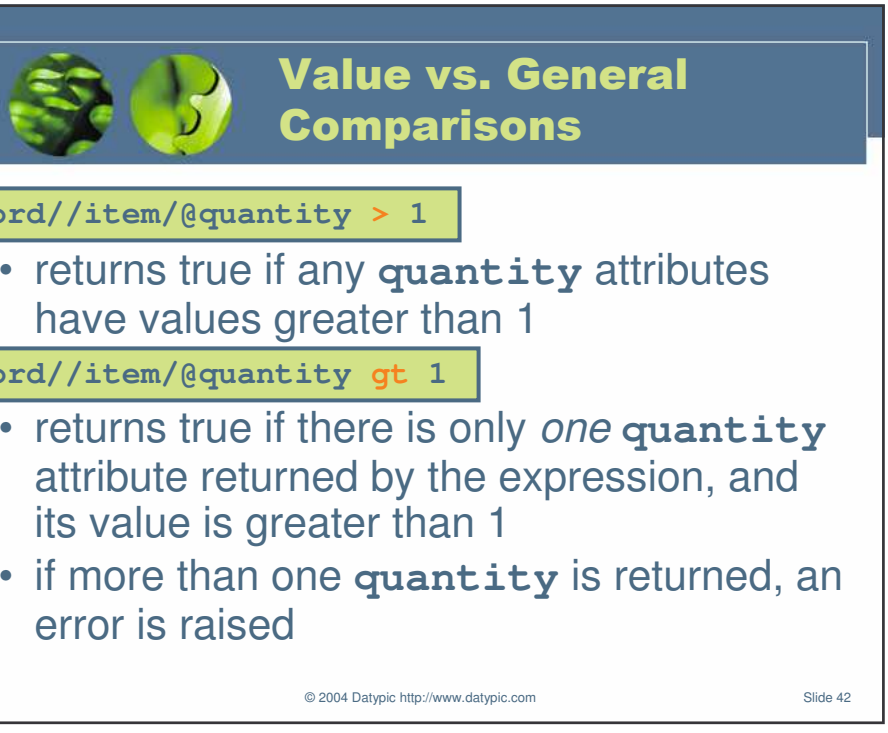

Literal Values
• Literal values can be expressed as:
– strings (in single or double quotes)
$cat//product/@dept = "WMN"
– numbers
$ord//item/@quantity > 1
– values constructed to be of a specific type
$priceList/@effDate > xs:date("2004-10-11")
© 2004 Datypic http://www.datypic.com
Slide 37

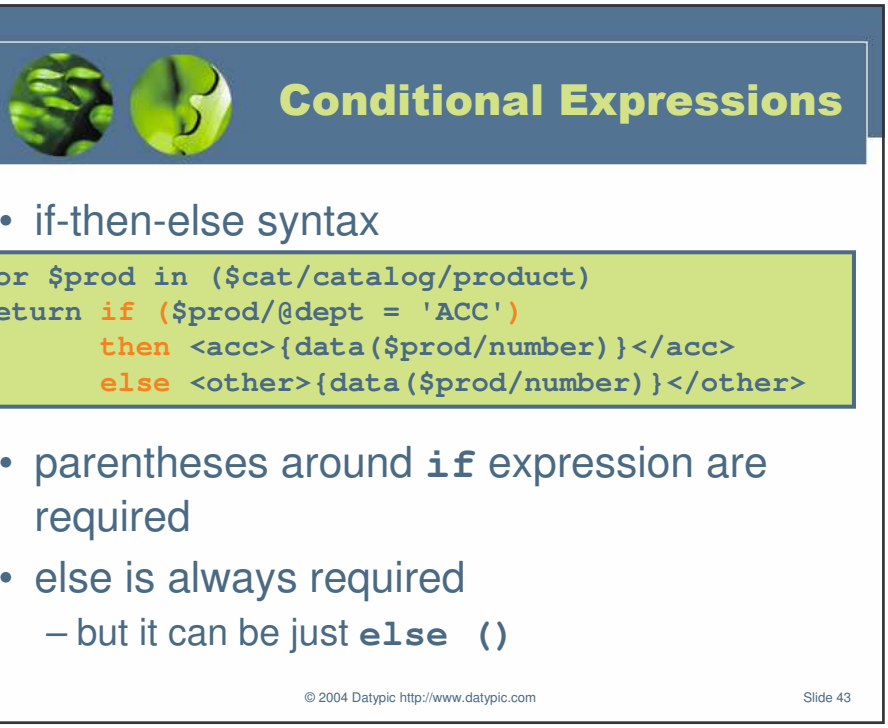

Variable References
• Identified by a name preceded by a $
• Variables are defined in several places
– FLWOR expressions
– Query prologs
– Outside the query by the processor
– Function signatures
for $prod in ($cat//product)
return $prod/number
declare function local:getProdNum
($prod as element()) as element()
{ $prod/number };
© 2004 Datypic http://www.datypic.com
Slide 38

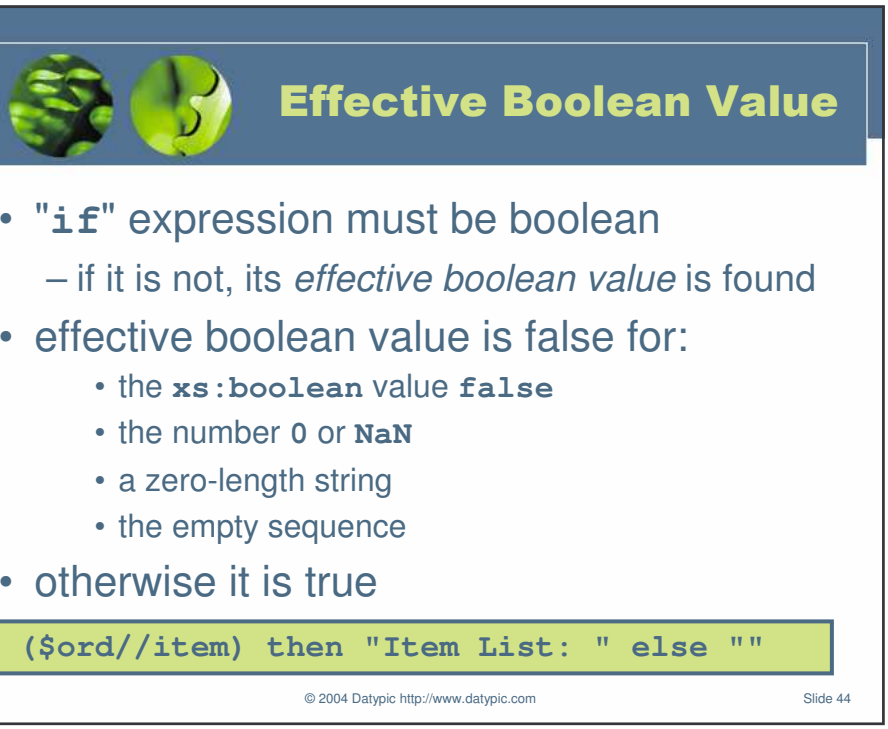

Function Calls
• An argument can be any single expression
– e.g. a variable reference, a path expression
• An argument may be required to have a certain type
list of arguments
substring($prodName, 1, 5)
function name
arg 1
arg 2
arg 3
© 2004 Datypic http://www.datypic.com
Slide 39

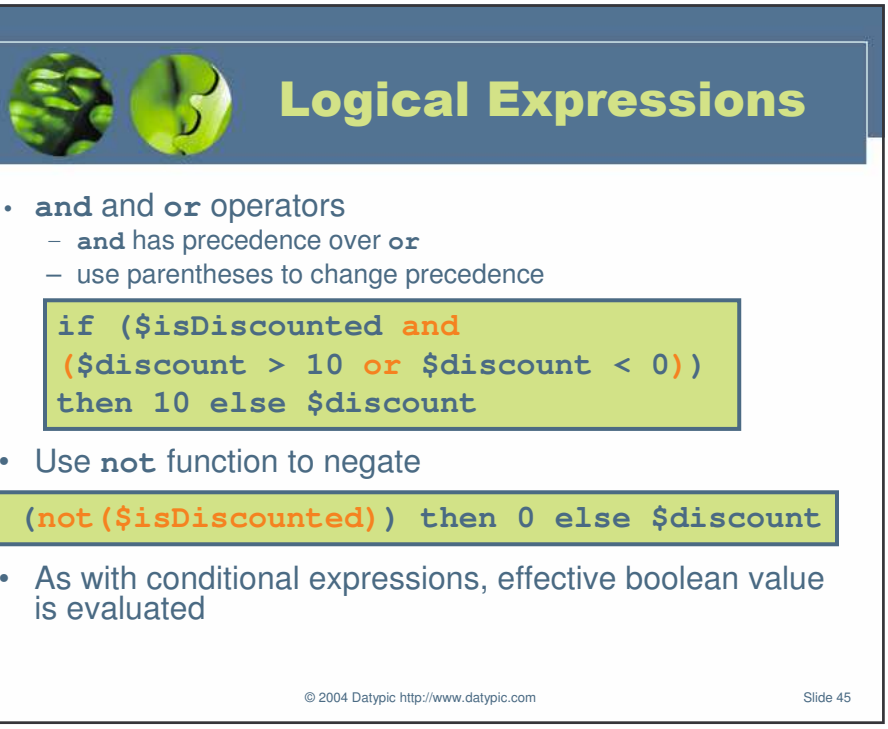

Comments
• XQuery comments
– Delimited by (: and :)
– Anywhere insignificant whitespace is allowed
– Do not appear in the results
(: This query... :)
• XML comments
– May appear in the results
– XML-like syntax
<!-- This element... -->
© 2004 Datypic http://www.datypic.com
Slide 40

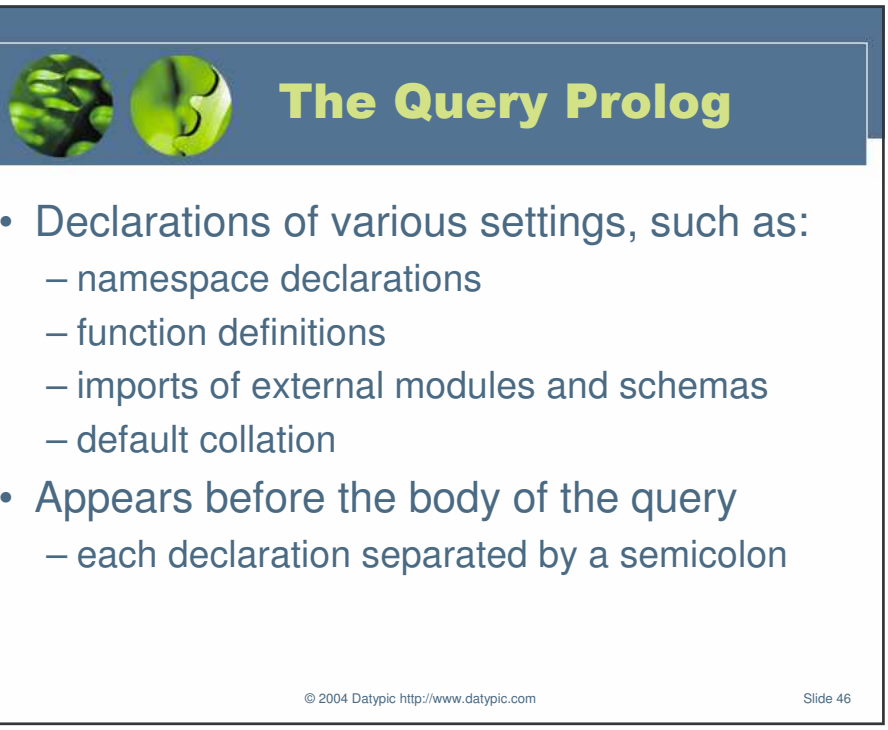

Comparisons
• Two kinds:
– value comparisons
• eq, ne, lt, le, gt, ge
• used to compare individual values
• each operand must be a single atomic value or a node containing a single atomic value
– general comparisons
• =, !=, <, <=, >, >=
• can be used with sequences of multiple items
© 2004 Datypic http://www.datypic.com
Slide 41

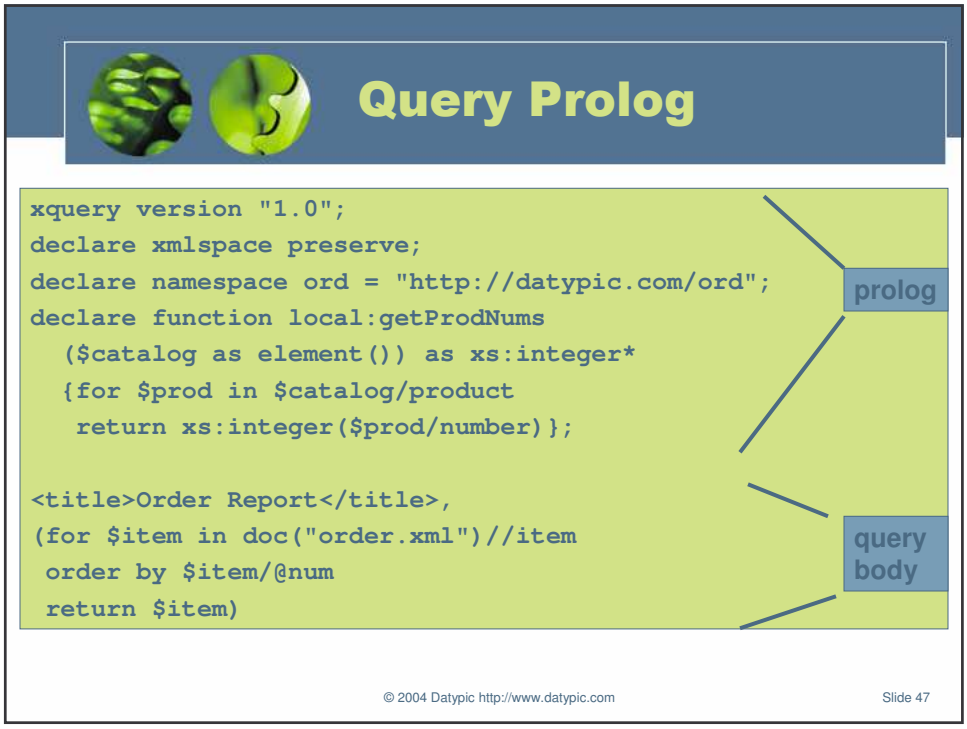

Value vs. General Comparisons
$ord//item/@quantity > 1
• returns true if any quantity attributes have values greater than 1
$ord//item/@quantity gt 1
• returns true if there is only one quantity attribute returned by the expression, and its value is greater than 1
• if more than one quantity is returned, an error is raised
© 2004 Datypic http://www.datypic.com
Slide 42


Conditional Expressions
• if-then-else syntax
for $prod in ($cat/catalog/product)
return if ($prod/@dept = 'ACC')
then <acc>{data($prod/number)}</acc>
else <other>{data($prod/number)}</other>
• parentheses around if expression are required
• else is always required
– but it can be just else ()
© 2004 Datypic http://www.datypic.com
Slide 43

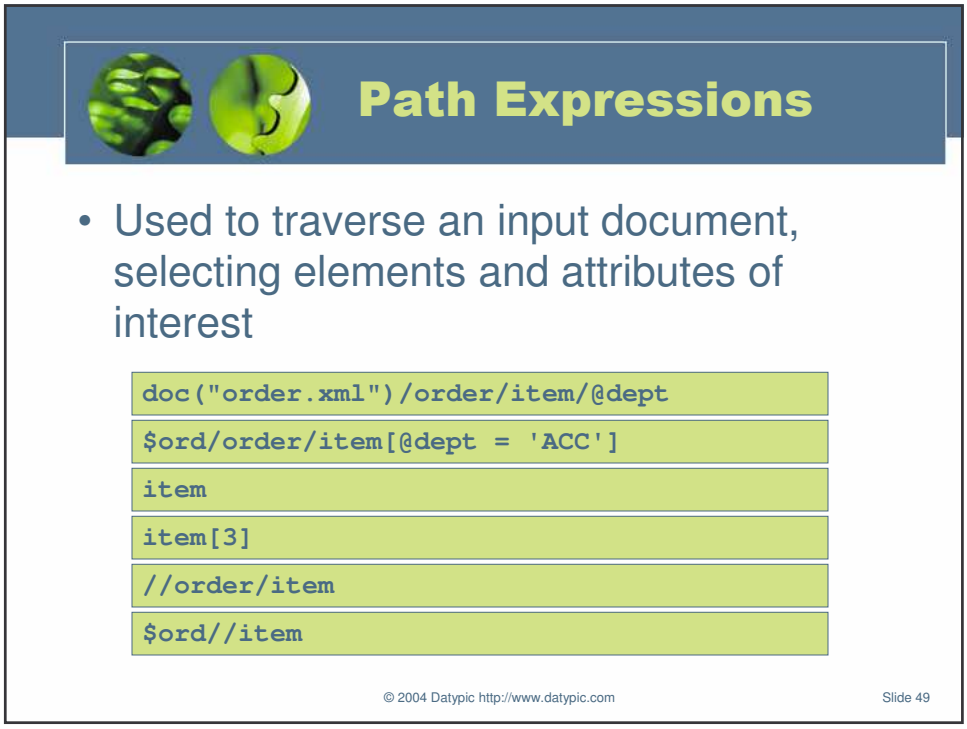

Effective Boolean Value
• "if" expression must be boolean
– if it is not, its effective boolean value is found
• effective boolean value is false for:
• the xs:boolean value false
• the number 0 or NaN
• a zero-length string
• the empty sequence
• otherwise it is true
if ($ord//item) then "Item List: " else ""
© 2004 Datypic http://www.datypic.com
Slide 44

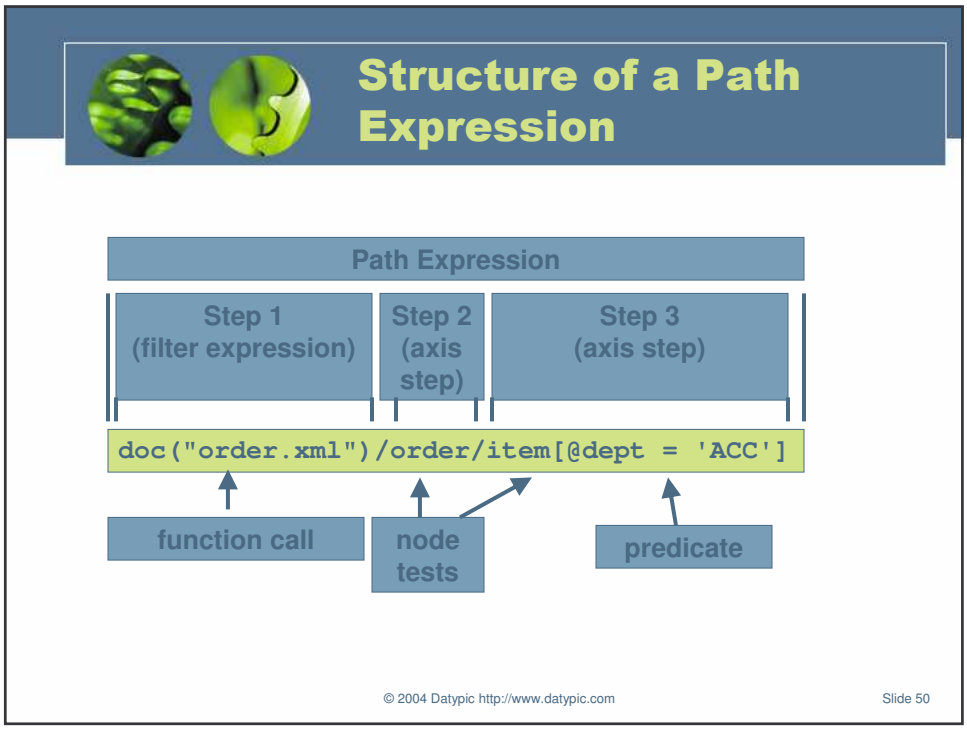

Logical Expressions
• and and or operators
– and has precedence over or
– use parentheses to change precedence
if ($isDiscounted and
($discount > 10 or $discount < 0))
then 10 else $discount
• Use not function to negate
if (not($isDiscounted)) then 0 else $discount
• As with conditional expressions, effective boolean value is evaluated
© 2004 Datypic http://www.datypic.com
Slide 45

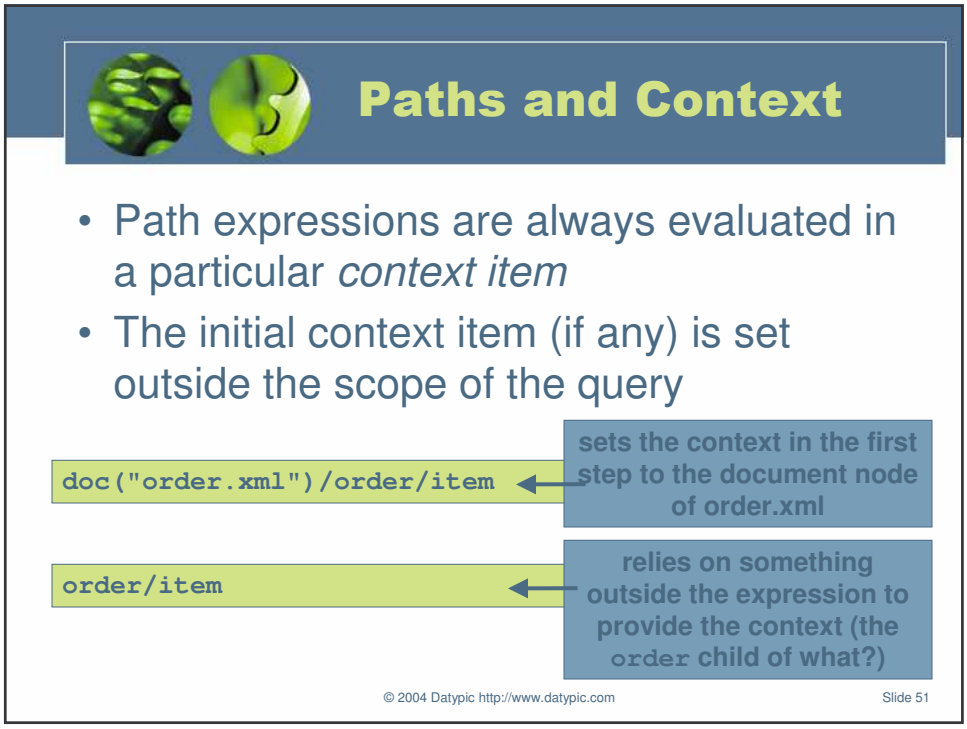

The Query Prolog
• Declarations of various settings, such as:
– namespace declarations
– function definitions
– imports of external modules and schemas
– default collation
• Appears before the body of the query
– each declaration separated by a semicolon
© 2004 Datypic http://www.datypic.com
Slide 46

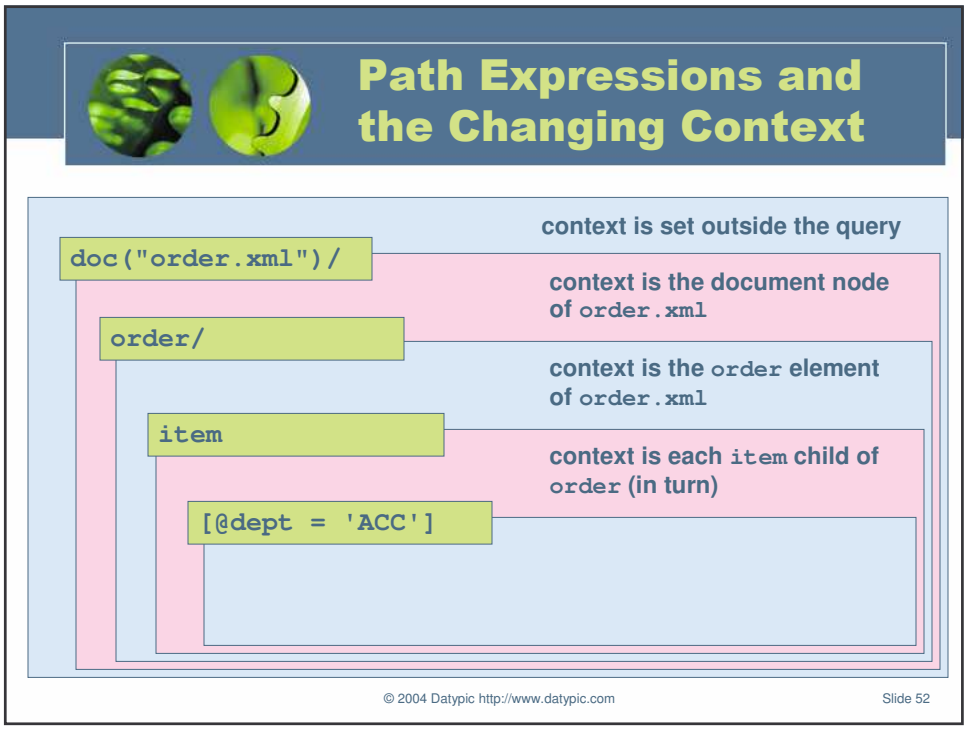

Query Prolog
xquery version "1.0";
declare xmlspace preserve;
declare namespace ord = "http://datypic.com/ord";
declare function local:getProdNums
($catalog as element()) as xs:integer*
{for $prod in $catalog/product
return xs:integer($prod/number)};
<title>Order Report</title>,
(for $item in doc("order.xml")//item
order by $item/@num
return $item)
prolog
query body
© 2004 Datypic http://www.datypic.com
Slide 47

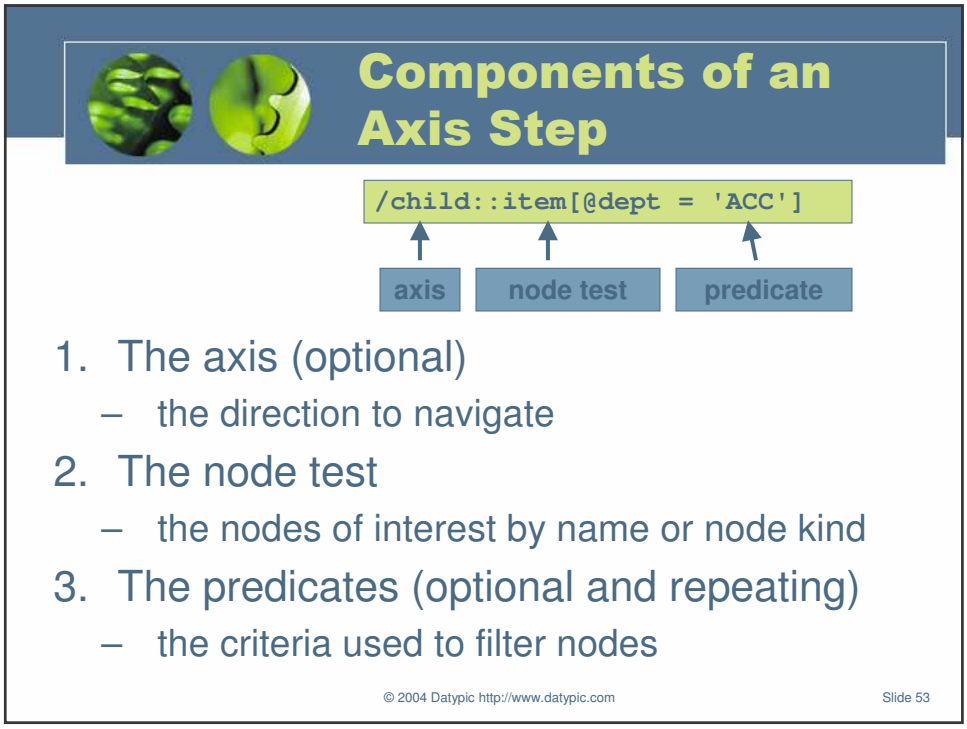

Path Expressions

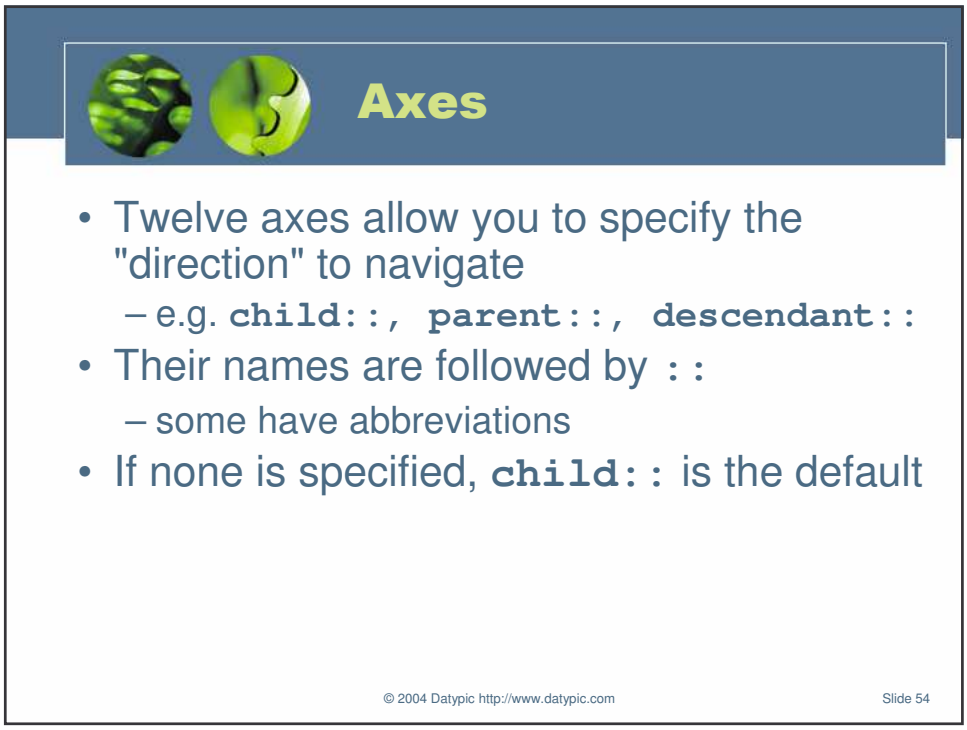

Path Expressions
• Used to traverse an input document, selecting elements and attributes of interest
doc("order.xml")/order/item/@dept
$ord/order/item[@dept = 'ACC']
item
item[3]
//order/item
$ord//item
© 2004 Datypic http://www.datypic.com
Slide 49

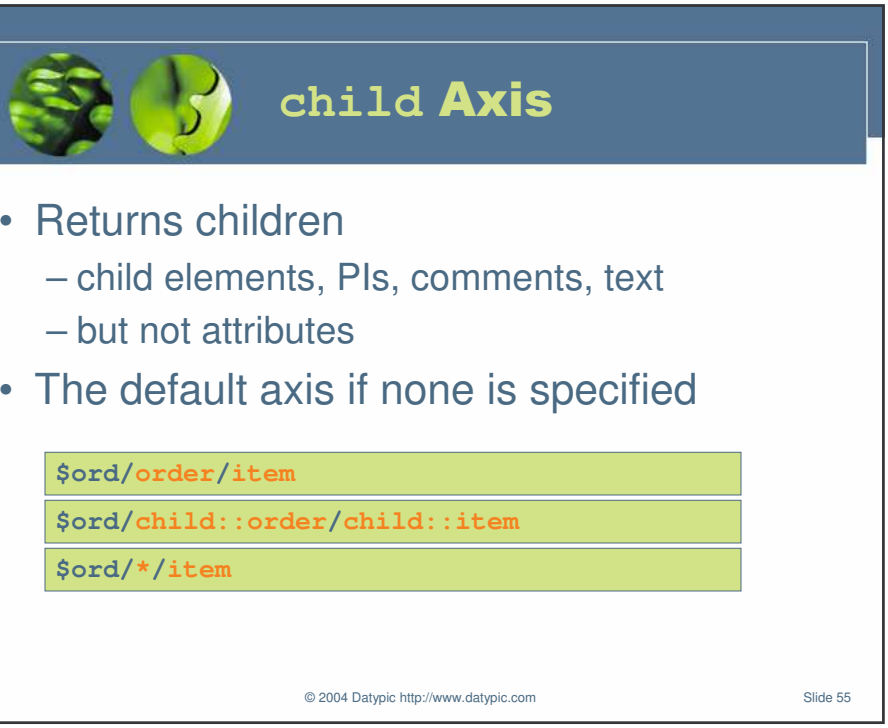

Structure of a Path Expression
Path Expression
Step 1 (filter expression)
Step 2 (axis step)
Step 3 (axis step)
doc("order.xml")/order/item[@dept = 'ACC']
function call
node tests
predicate
© 2004 Datypic http://www.datypic.com
Slide 50

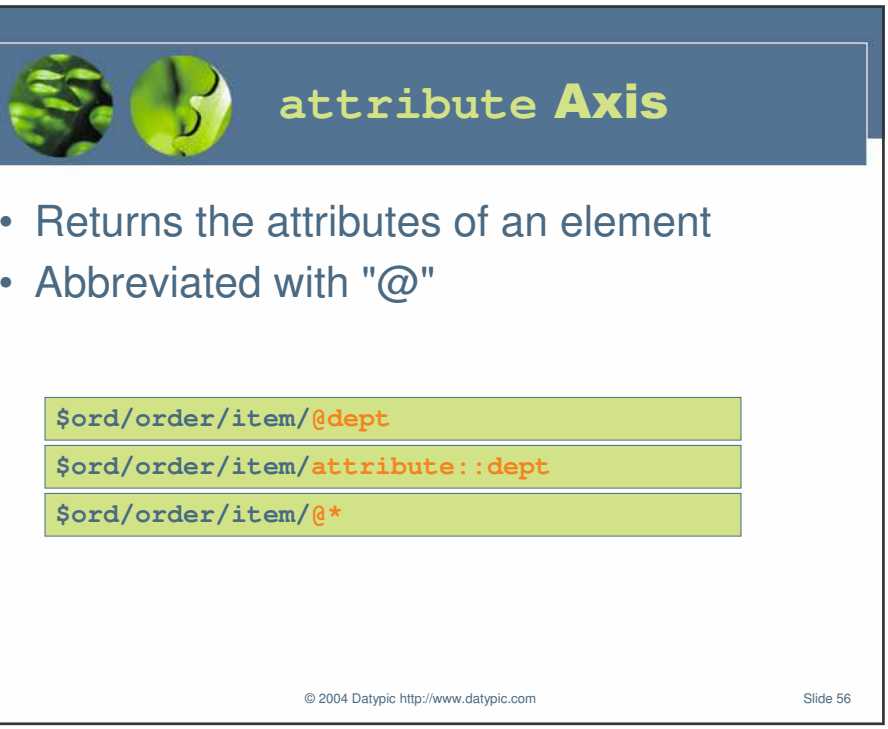

Paths and Context
• Path expressions are always evaluated in a particular *context item*
• The initial context item (if any) is set outside the scope of the query
doc("order.xml")/order/item
sets the context in the first step to the document node of order.xml
order/item
relies on something outside the expression to provide the context (the order child of what?)
© 2004 Datypic http://www.datypic.com
Slide 51

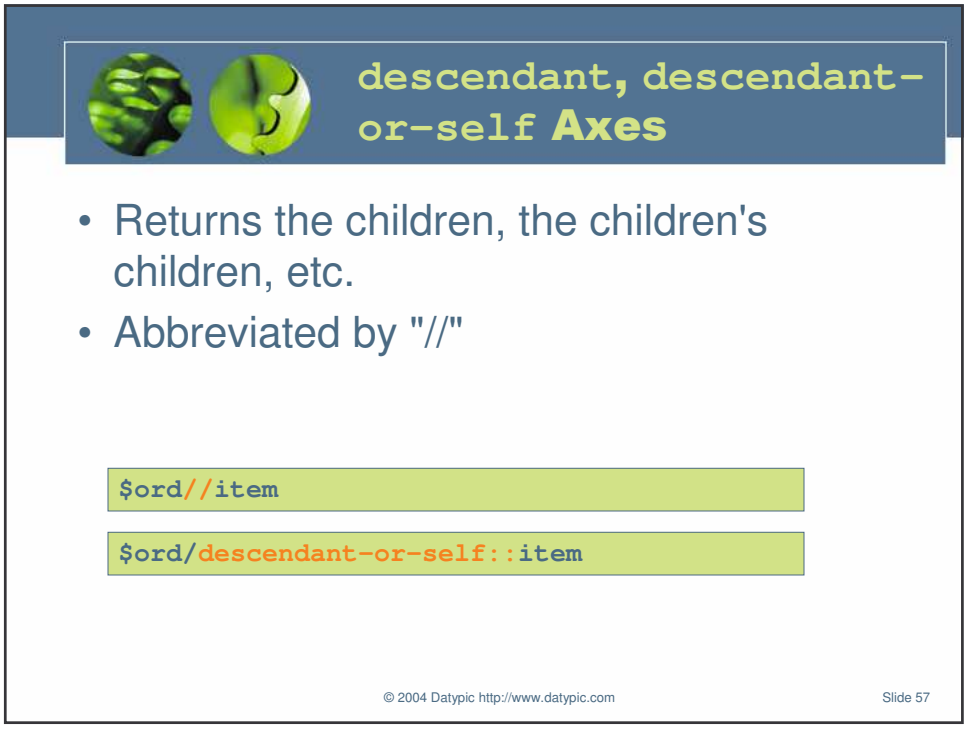

Path Expressions and the Changing Context
context is set outside the query
doc("order.xml")/
context is the document node of order.xml
order/
context is the order element of order.xml
item
context is each item child of order (in turn)
[@dept = 'ACC']
© 2004 Datypic http://www.datypic.com
Slide 52

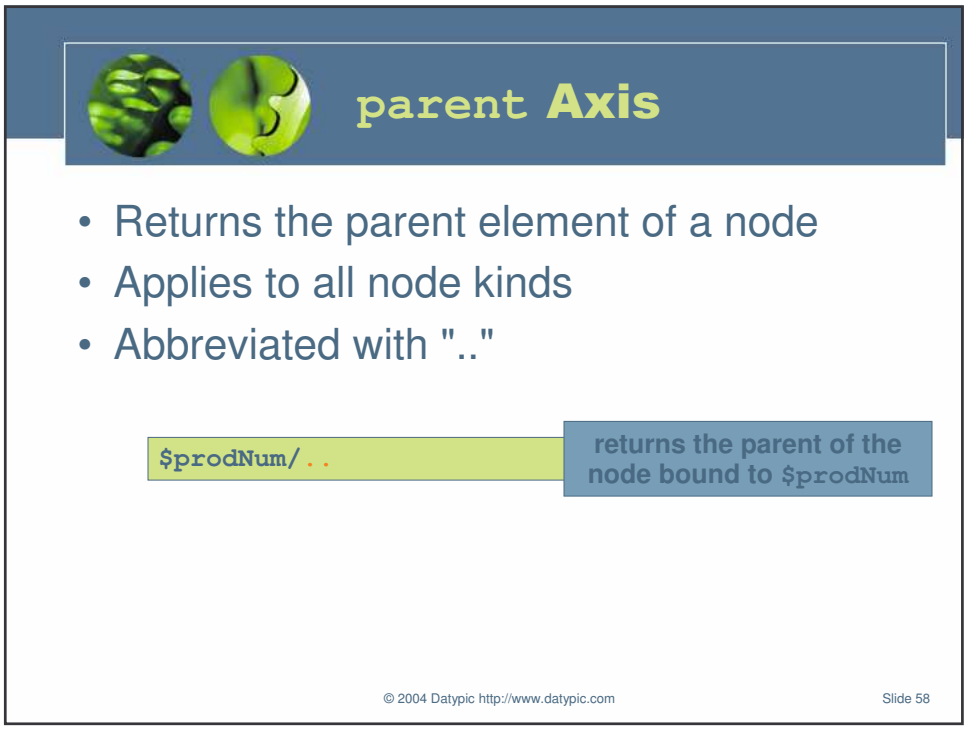

Components of an Axis Step
/child::item[@dept = 'ACC']
axis
node test
predicate
1. The axis (optional)
– the direction to navigate
2. The node test
– the nodes of interest by name or node kind
3. The predicates (optional and repeating)
– the criteria used to filter nodes
© 2004 Datypic http://www.datypic.com
Slide 53

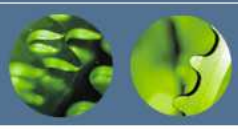

Axes
• Twelve axes allow you to specify the "direction" to navigate
– e.g. child::, parent::, descendant::
• Their names are followed by ::
– some have abbreviations
• If none is specified, child:: is the default
© 2004 Datypic http://www.datypic.com
Slide 54


child Axis
• Returns children
– child elements, PIs, comments, text
– but not attributes
• The default axis if none is specified
$ord/order/item
$ord/child::order/child::item
$ord/*/item
© 2004 Datypic http://www.datypic.com
Slide 55


attribute Axis
• Returns the attributes of an element
• Abbreviated with "@"
$ord/order/item/@dept
$ord/order/item/attribute::dept
$ord/order/item/@*
© 2004 Datypic http://www.datypic.com
Slide 56

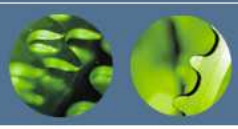

descendant, descendant-or-self Axes
• Returns the children, the children's children, etc.
• Abbreviated by "//"
$ord//item
$ord/descendant-or-self::item
© 2004 Datypic http://www.datypic.com
Slide 57


parent Axis
• Returns the parent element of a node
• Applies to all node kinds
• Abbreviated with ".."
$prodNum/..
returns the parent of the node bound to $prodNum
© 2004 Datypic http://www.datypic.com
Slide 58



Other Axes

- **`self`**
  - the node itself (abbreviated as ".")
- **`ancestor`**, **`ancestor-or-self`**
  - the parent, the parent's parent, and so on
- **`following`** and **`preceding`**
  - all nodes that follow/precede it, in document order
- **`following-sibling`** and **`preceding-sibling`**
  - siblings that follow/precede it, in document order



Node Tests

- Can consist of a:
  - node name (for elements/attributes)

```
$ord/order/item/@dept
```

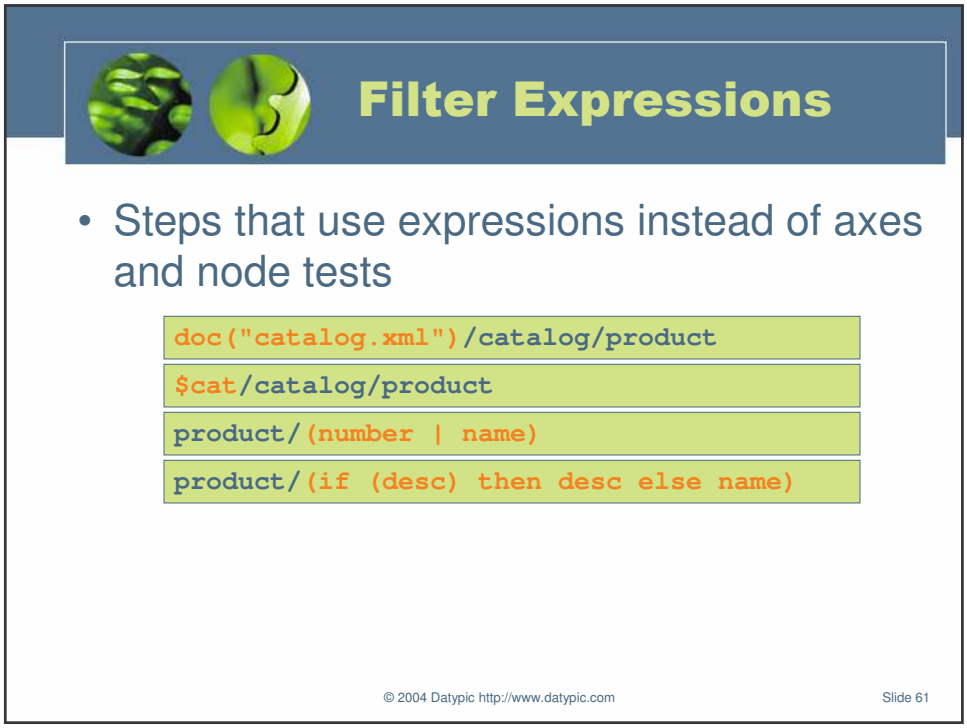
  - node kind

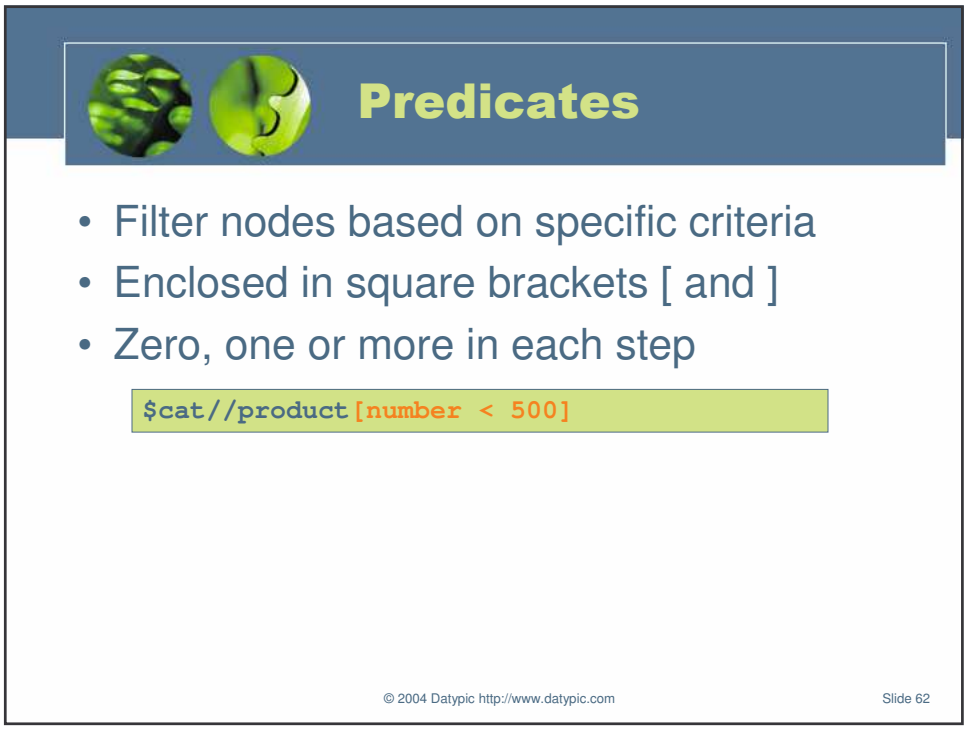
```
$cat/catalog/element()
$cat//number/text()
$cat/node()
```

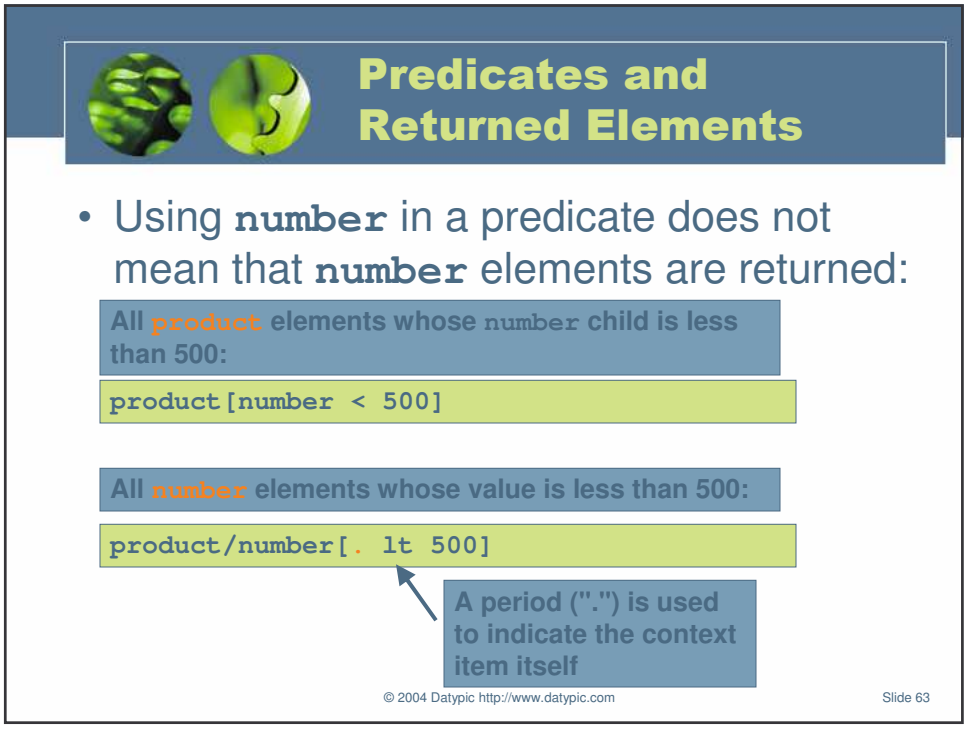
  - wildcard (*)

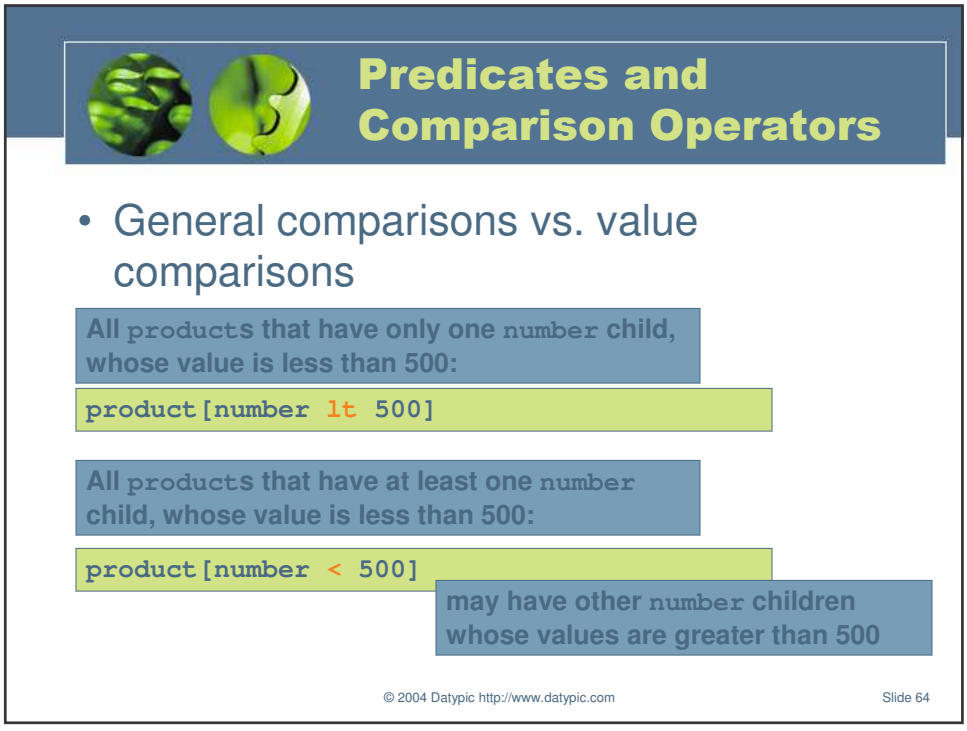
```
$ord/order/*/@*
```

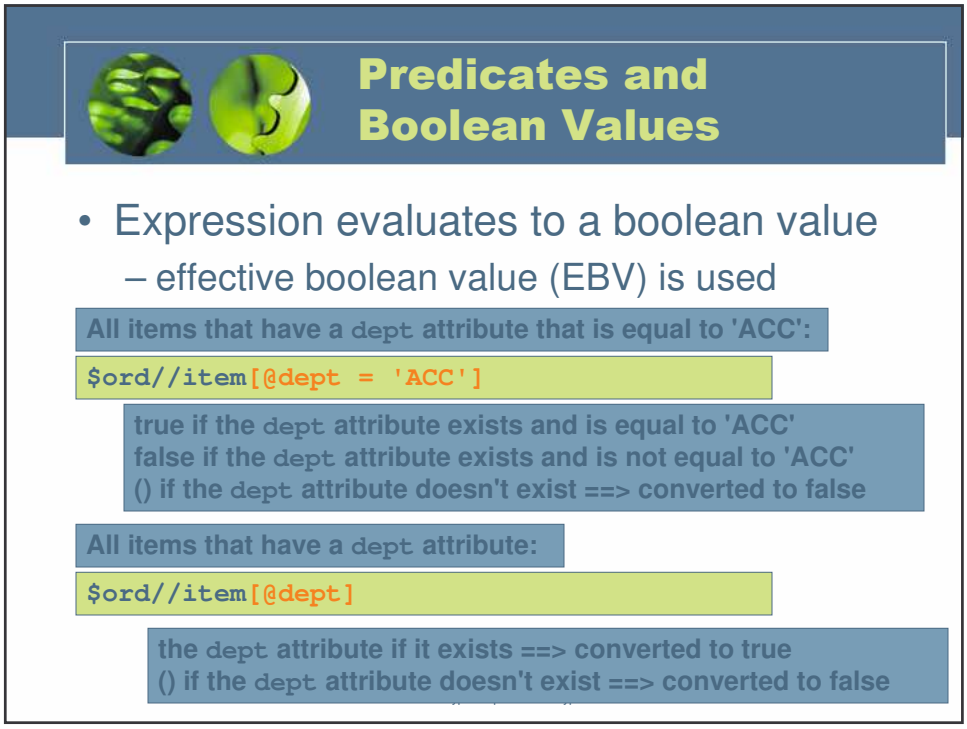
all the attributes of all the element children of `order`

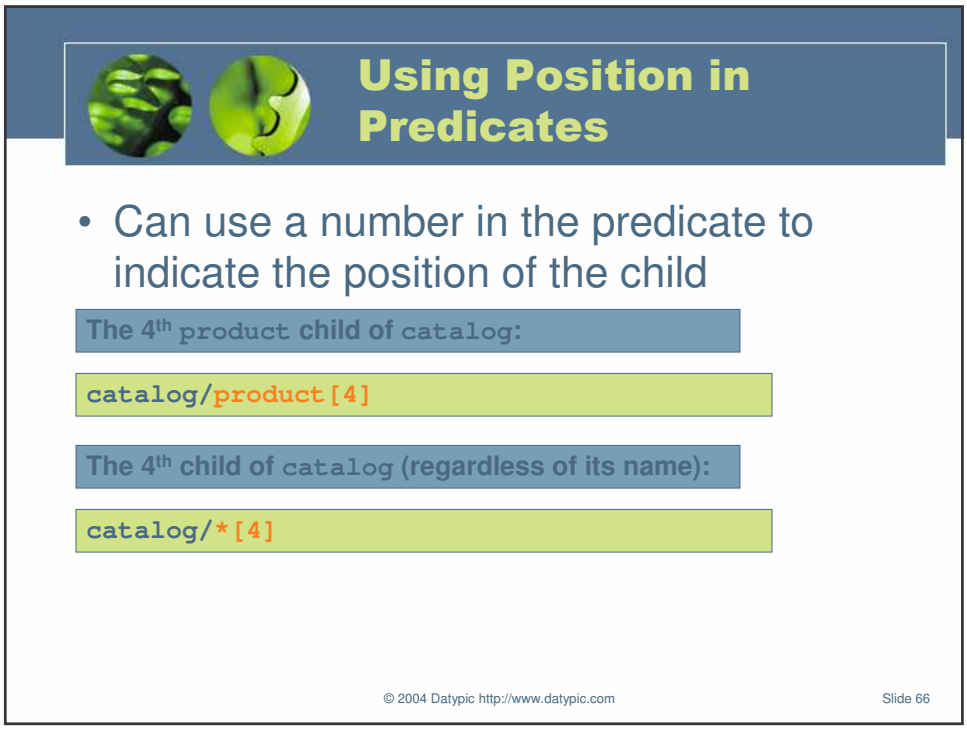

Filter Expressions
• Steps that use expressions instead of axes and node tests
doc("catalog.xml")/catalog/product
$cat/catalog/product
product/(number | name)
product/(if (desc) then desc else name)
© 2004 Datypic http://www.datypic.com
Slide 61

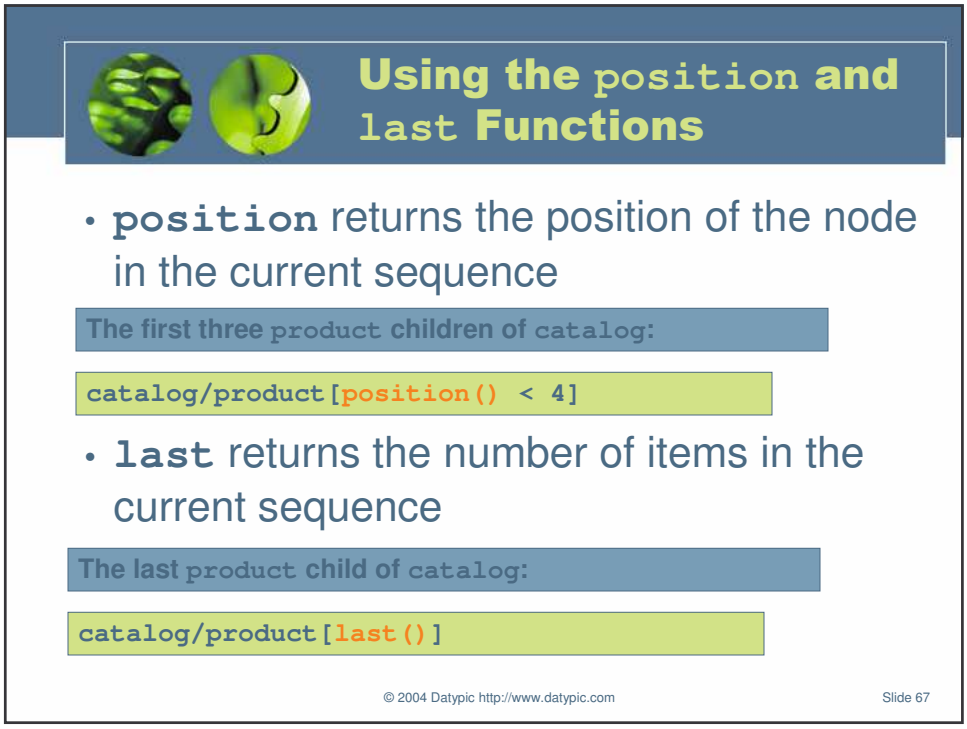

Predicates
• Filter nodes based on specific criteria
• Enclosed in square brackets [and]
• Zero, one or more in each step
$cat//product[number < 500]
© 2004 Datypic http://www.datypic.com
Slide 62

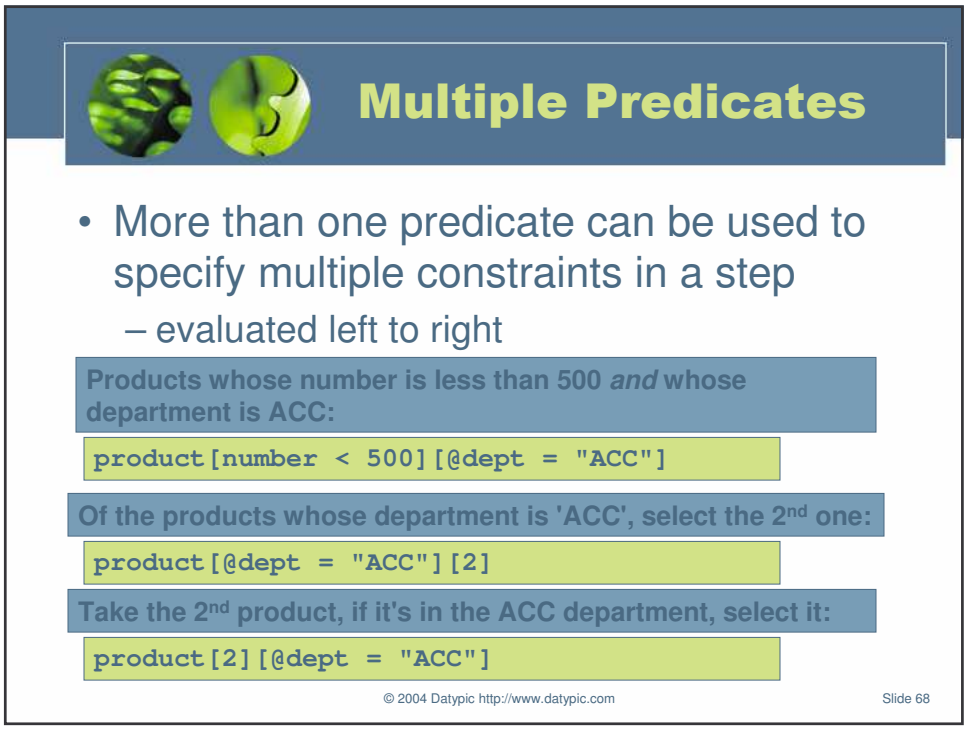

Predicates and Returned Elements
• Using number in a predicate does not mean that number elements are returned:
All product elements whose number child is less than 500:
product[number < 500]
All number elements whose value is less than 500:
product/number[. lt 500]
A period (".") is used to indicate the context item itself
© 2004 Datypic http://www.datypic.com
Slide 63

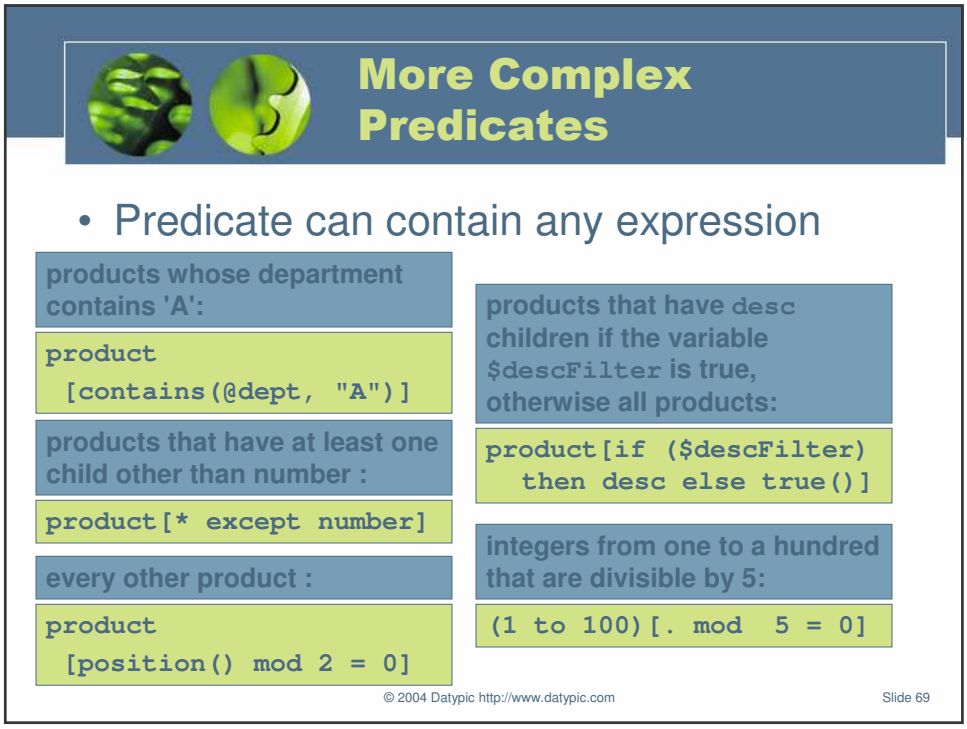

Predicates and Comparison Operators
• General comparisons vs. value comparisons
All products that have only one number child, whose value is less than 500:
product[number lt 500]
All products that have at least one number child, whose value is less than 500:
product[number < 500]
may have other number children whose values are greater than 500
© 2004 Datypic http://www.datypic.com
Slide 64

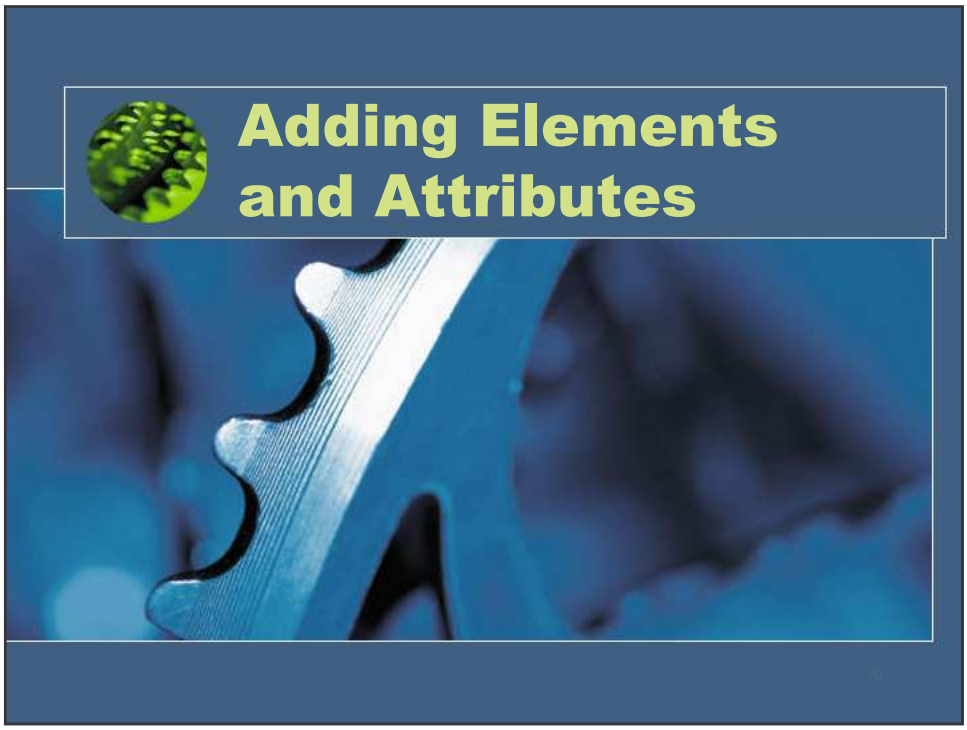

Predicates and Boolean Values
• Expression evaluates to a boolean value
– effective boolean value (EBV) is used
All items that have a dept attribute that is equal to 'ACC':
$ord//item[@dept = 'ACC']
true if the dept attribute exists and is equal to 'ACC'
false if the dept attribute exists and is not equal to 'ACC'
() if the dept attribute doesn't exist ==> converted to false
All items that have a dept attribute:
$ord//item[@dept]
the dept attribute if it exists ==> converted to true
() if the dept attribute doesn't exist ==> converted to false

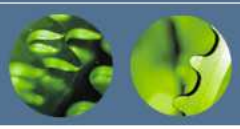

Using Position in Predicates
• Can use a number in the predicate to indicate the position of the child
The 4th product child of catalog:
catalog/product[4]
The 4th child of catalog (regardless of its name):
catalog/*[4]
© 2004 Datypic http://www.datypic.com
Slide 66


Using the position and last Functions
• position returns the position of the node in the current sequence
The first three product children of catalog:
catalog/product[position() < 4]
• last returns the number of items in the current sequence
The last product child of catalog:
catalog/product[last()]
© 2004 Datypic http://www.datypic.com
Slide 67


Multiple Predicates
• More than one predicate can be used to specify multiple constraints in a step
– evaluated left to right
Products whose number is less than 500 and whose department is ACC:
product[number < 500][@dept = "ACC"]
Of the products whose department is 'ACC', select the 2nd one:
product[@dept = "ACC"][2]
Take the 2nd product, if it's in the ACC department, select it:
product[2][@dept = "ACC"]
© 2004 Datypic http://www.datypic.com
Slide 68

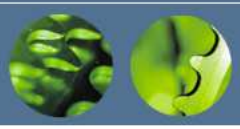

More Complex Predicates
• Predicate can contain any expression
products whose department contains 'A':
product [contains(@dept, "A")]
products that have at least one child other than number :
product[* except number]
every other product :
product [position() mod 2 = 0]
products that have desc children if the variable $descFilter is true, otherwise all products:
product[if ($descFilter) then desc else true()]
integers from one to a hundred that are divisible by 5:
(1 to 100)[. mod 5 = 0]
© 2004 Datypic http://www.datypic.com
Slide 69


Adding Elements and Attributes



3 Ways to Add Elements/ Attributes to Results

- Including them from the input document
  - like most of our previous examples
- Using direct constructors
  - XML-like syntax
- Using computed constructors
  - special syntax using curly braces
  - allows for dynamic names

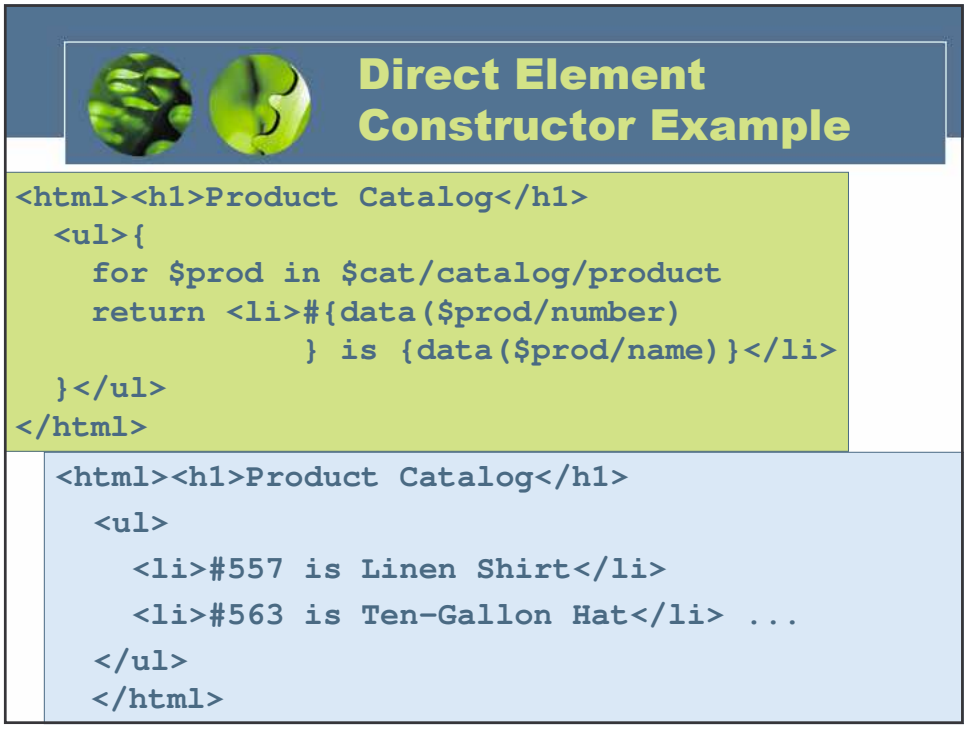


Including from the Input Document

```
for $prod in $cat/catalog/product[@dept='ACC']
return $prod/name
```

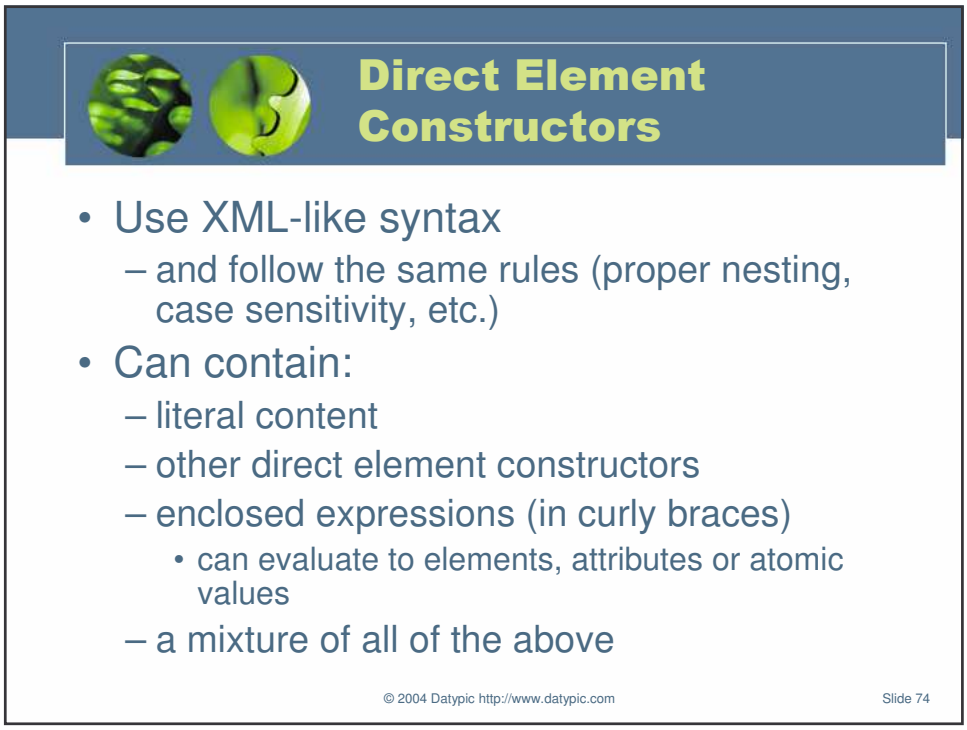
```
<name language="en">Ten-Gallon Hat</name>
<name language="en">Golf Umbrella</name>
```

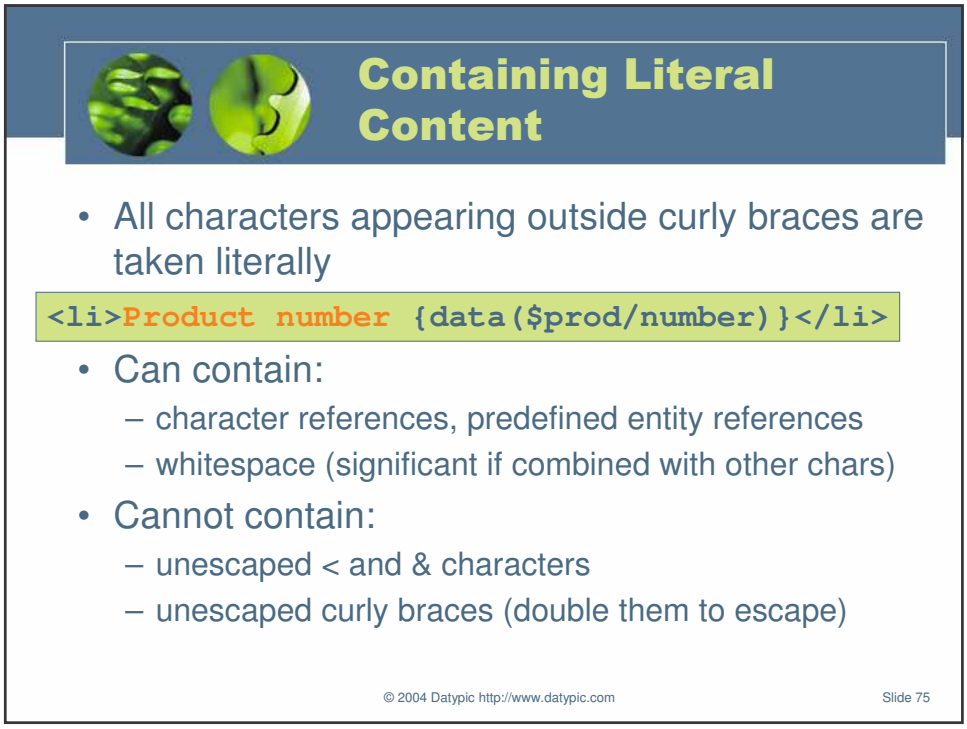
- **name** elements are included as is
  - along with their attributes (and children if any)
  - not just their atomic values
- no opportunity to change attributes, children, namespace

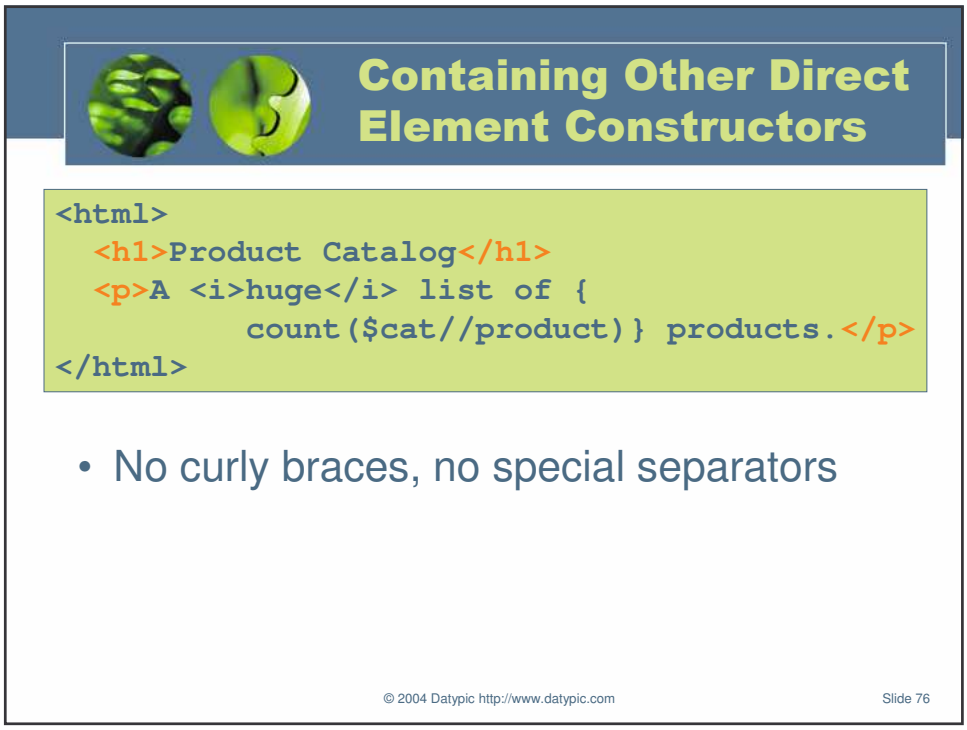

Direct Element Constructor Example
<html><h1>Product Catalog</h1>
{
for $prod in $cat/catalog/product
return #{data($prod/number)
} is {data($prod/name)}
}
</html>
<html><h1>Product Catalog</h1>

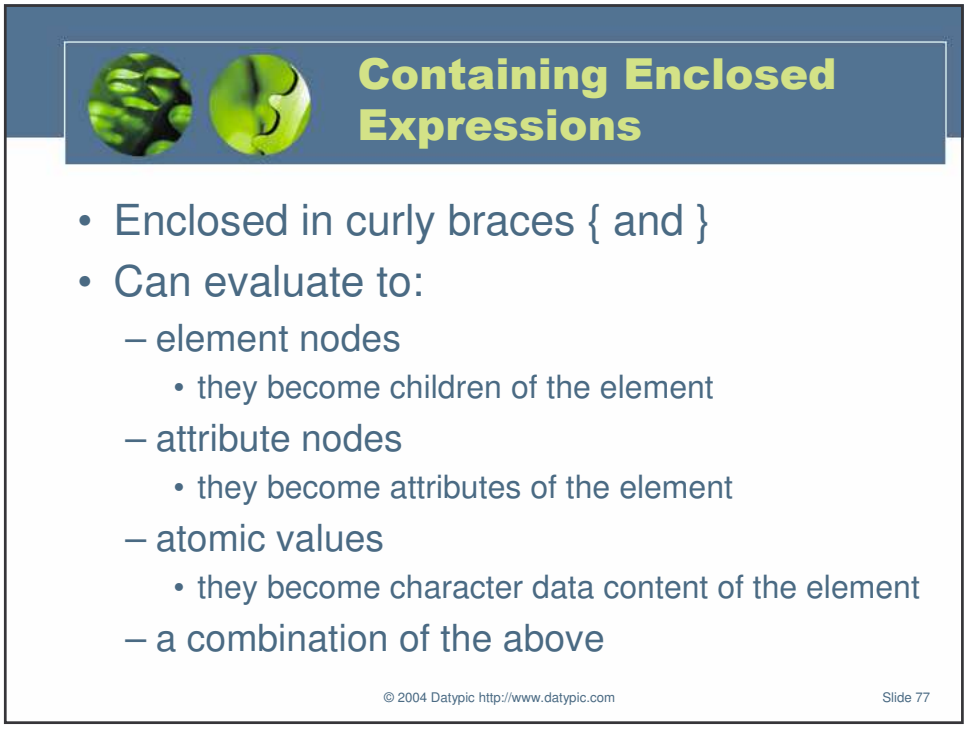
#557 is Linen Shirt
#563 is Ten-Gallon Hat ...

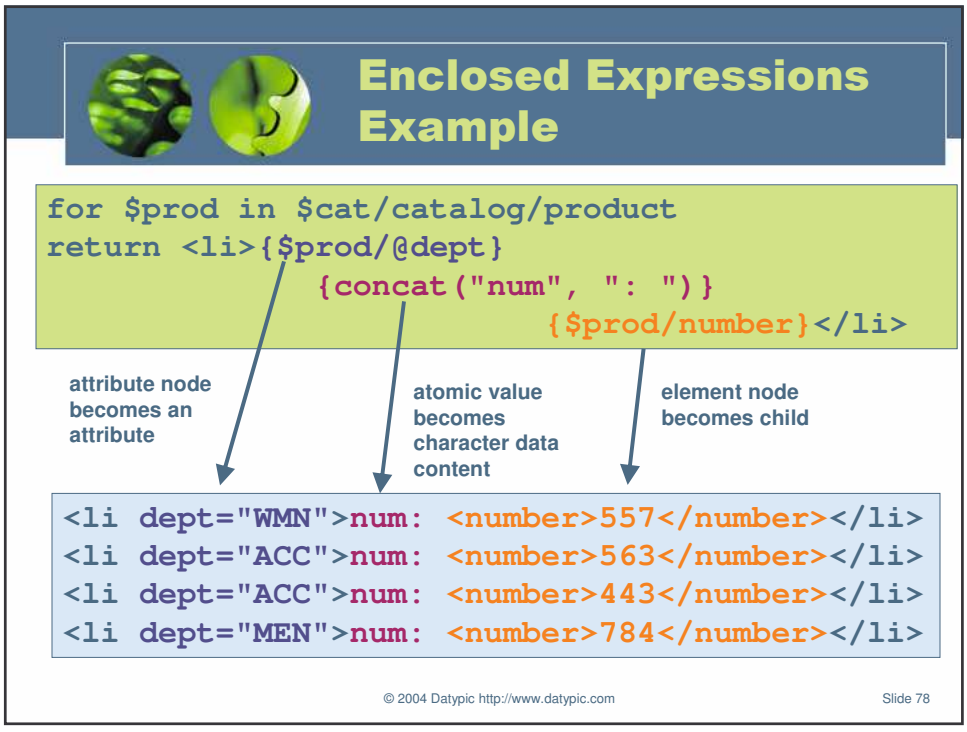
</html>

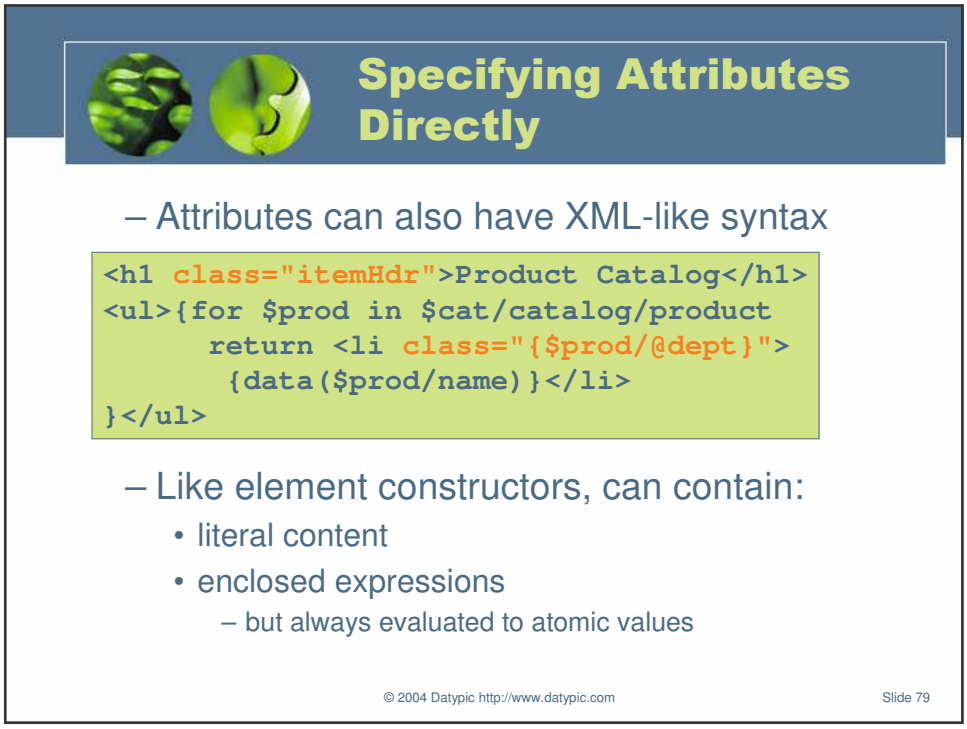

Direct Element Constructors
• Use XML-like syntax
– and follow the same rules (proper nesting, case sensitivity, etc.)
• Can contain:
– literal content
– other direct element constructors
– enclosed expressions (in curly braces)
• can evaluate to elements, attributes or atomic values
– a mixture of all of the above
© 2004 Datypic http://www.datypic.com
Slide 74

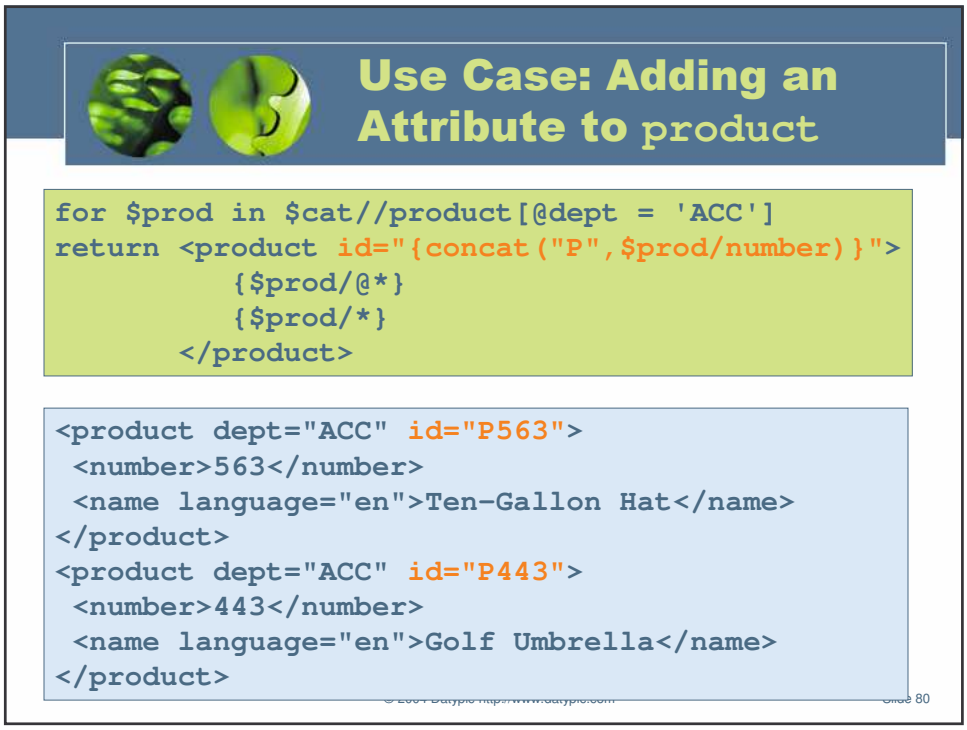

Containing Literal Content
• All characters appearing outside curly braces are taken literally
Product number {data($prod/number)}
• Can contain:
– character references, predefined entity references
– whitespace (significant if combined with other chars)
• Cannot contain:
– unescaped < and & characters
– unescaped curly braces (double them to escape)
© 2004 Datypic http://www.datypic.com
Slide 75

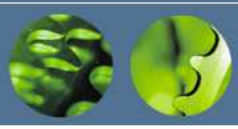

Containing Other Direct Element Constructors
<html>
<h1>Product Catalog</h1>
A huge list of {
count($cat//product)} products.
</html>
• No curly braces, no special separators
© 2004 Datypic http://www.datypic.com
Slide 76


Containing Enclosed Expressions
• Enclosed in curly braces { and }
• Can evaluate to:
– element nodes
• they become children of the element
– attribute nodes
• they become attributes of the element
– atomic values
• they become character data content of the element
– a combination of the above
© 2004 Datypic http://www.datypic.com
Slide 77


Enclosed Expressions Example
for $prod in $cat/catalog/product
return {$prod/@dept}
{concat("num", ": ")}
{$prod/number}
attribute node becomes an attribute
atomic value becomes character data content
element node becomes child
<li dept="WMN">num: <number>557</number>
<li dept="ACC">num: <number>563</number>
<li dept="ACC">num: <number>443</number>
<li dept="MEN">num: <number>784</number>
© 2004 Datypic http://www.datypic.com
Slide 78

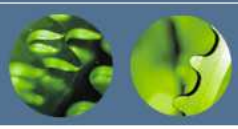

Specifying Attributes Directly
– Attributes can also have XML-like syntax
<h1 class="itemHdr">Product Catalog</h1>
{for $prod in $cat/catalog/product
return <li class="{$prod/@dept}">
{data($prod/name)}
}
– Like element constructors, can contain:
• literal content
• enclosed expressions
– but always evaluated to atomic values
© 2004 Datypic http://www.datypic.com
Slide 79


Use Case: Adding an Attribute to product
for $prod in $cat//product[@dept = 'ACC']
return <product id="{concat("P",$prod/number)}">
{$prod/@*}
{$prod/*}
</product>
<product dept="ACC" id="P563">
<number>563</number>
<name language="en">Ten-Gallon Hat</name>
</product>
<product dept="ACC" id="P443">
<number>443</number>
<name language="en">Golf Umbrella</name>
</product>
Slide 80



Computed Constructors

- Allow dynamic names and values
- Useful for:
  - copying elements from the input document but make minor changes to their content
    - e.g. add an **id** attribute to every element, regardless of name
  - turning content from the input document into element or attribute names
    - e.g. create an element whose name is the value of the **dept** attribute in the input document
  - looking up element names in a separate dictionary
    - e.g. for language translation



Computed Constructor Simple Example

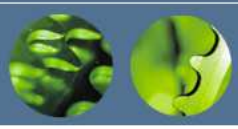
```
element product {
    attribute dept { "ACC" },
    element {concat("num","ber")}  { 563 },
    element name { attribute language { "en"},
                   "Ten-Gallon Hat"}
  }
```

```
<product dept="ACC">
  <number>563</number>
  <name language="en">Ten-Gallon Hat</name>
</product>
```



Use Case: Turning Content into Markup
for $dept in
distinct-values($cat/catalog/product/@dept)
return element {$dept}
{$cat/catalog/product[@dept = $dept]/name}
<WMN>
<name language="en">Linen Shirt</name>
</WMN>
<ACC>
<name language="en">Ten-Gallon Hat</name>
<name language="en">Golf Umbrella</name>
</ACC>
<MEN>
<name language="en">Rugby Shirt</name>
</MEN>

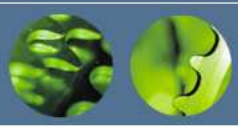

Selecting and Filtering



2 Ways to Select

- Path Expressions
  - great if you just want to copy certain elements and attributes as is
- FLWOR Expressions
  - allow sorting
  - allow adding elements/attributes to results
  - more verbose, but can be clearer

 Slide 85

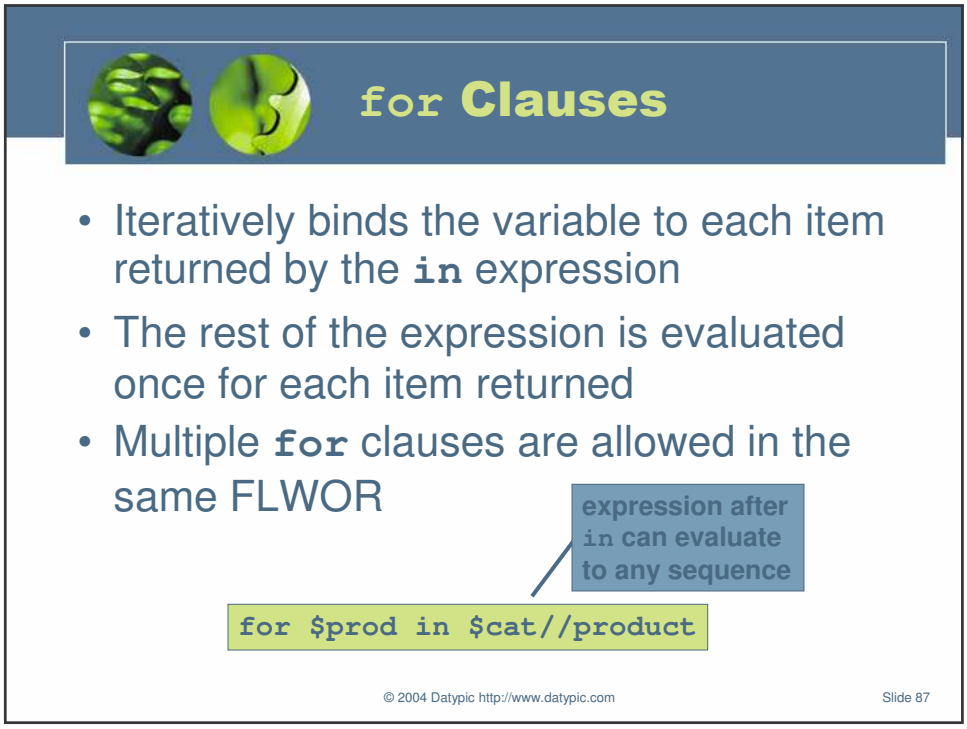


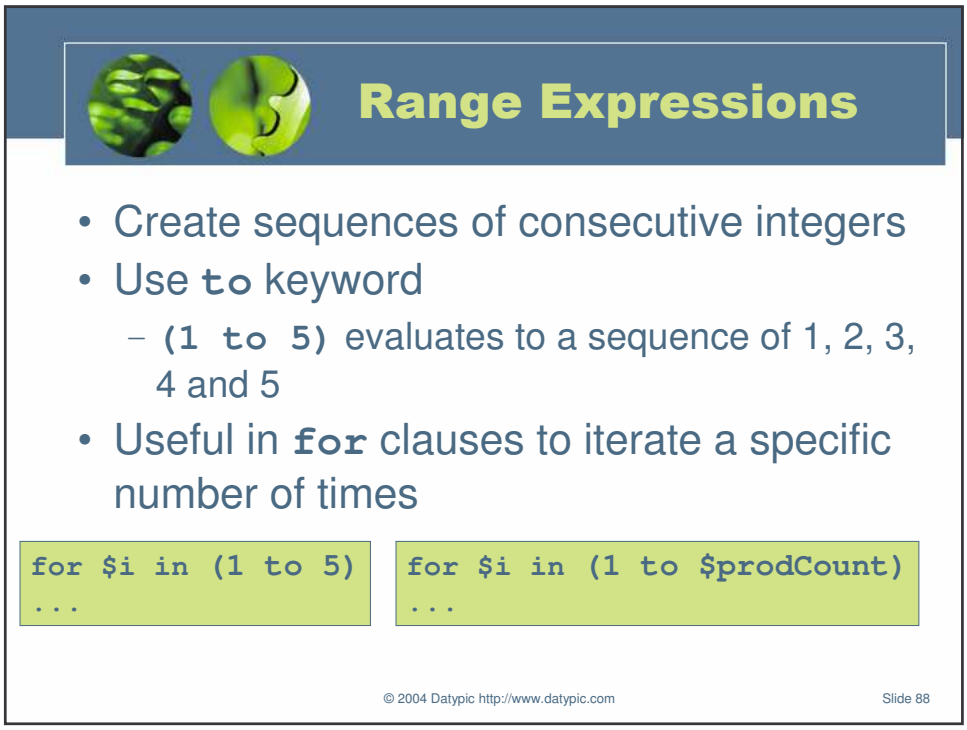
Clauses of a FLWOR Expression

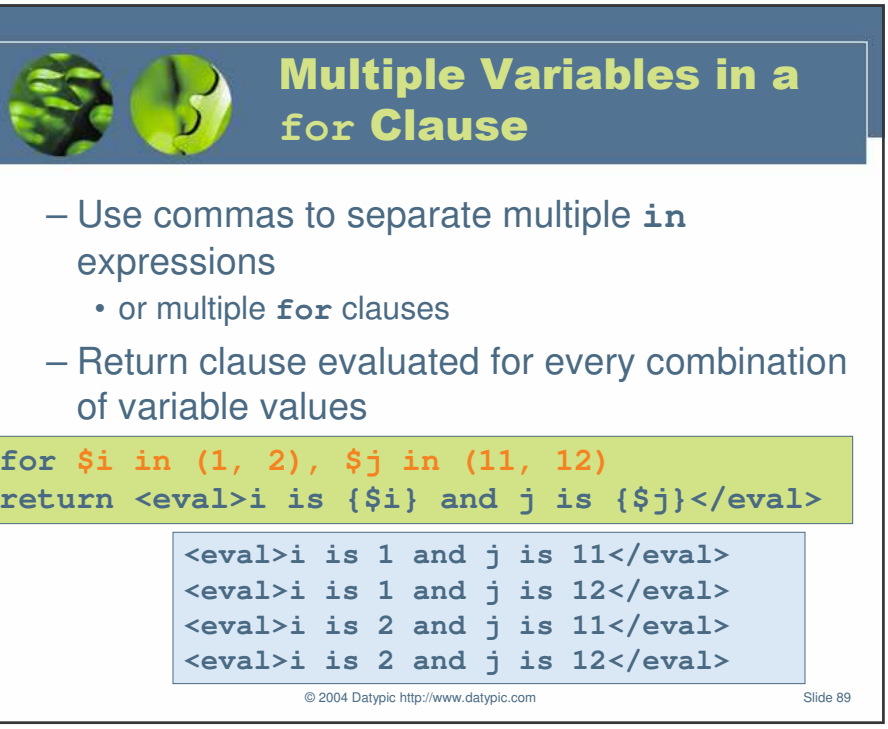
- **`for`** clause
  - iteratively binds the **`$prod`** variable to each item returned by a path expression.
- **`let`** clause
  - binds the **`$prodDept`** variable to the value of the **`dept`** attribute
- **`where`** clause
  - selects nodes whose **`dept`** attribute is equal to "**`WMN`**" or "**`ACC`**"
- **`return`** clause
  - returns the **`name`** children of the selected nodes

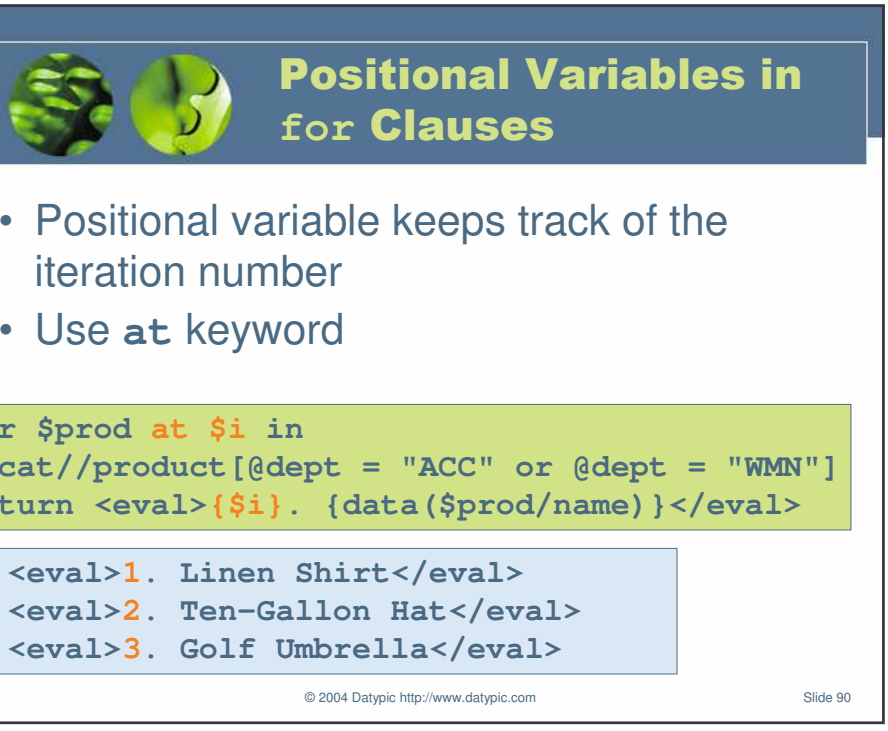
```
for $prod in $cat//product
let $prodDept := $prod/@dept
where $prodDept = "ACC" or $prodDept = "WMN"
return $prod/name
```

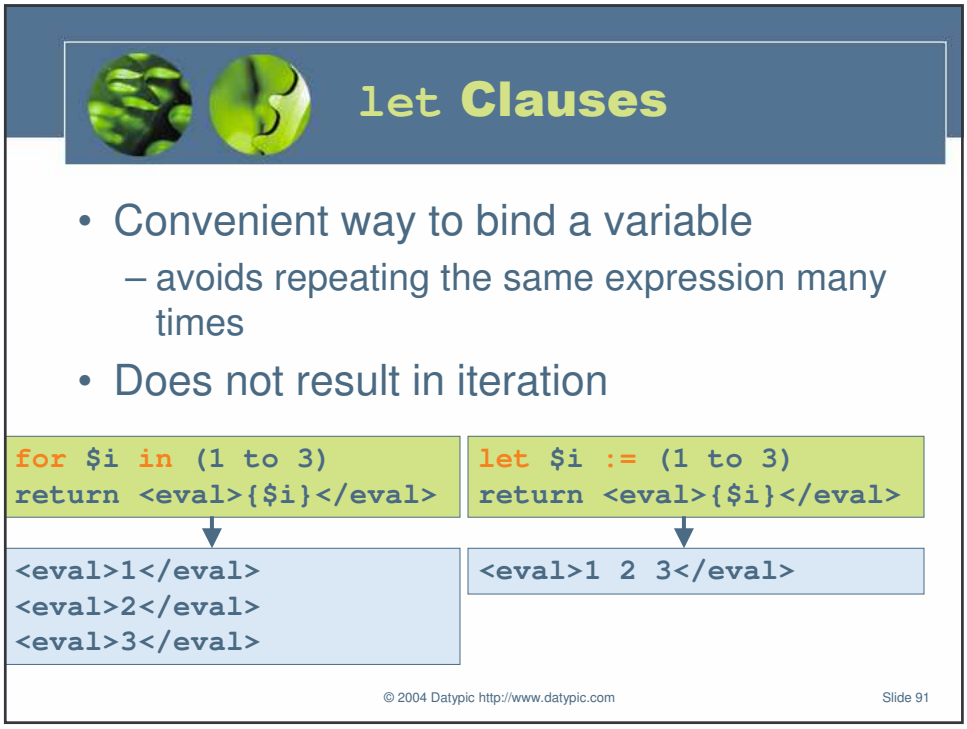
 Slide 86

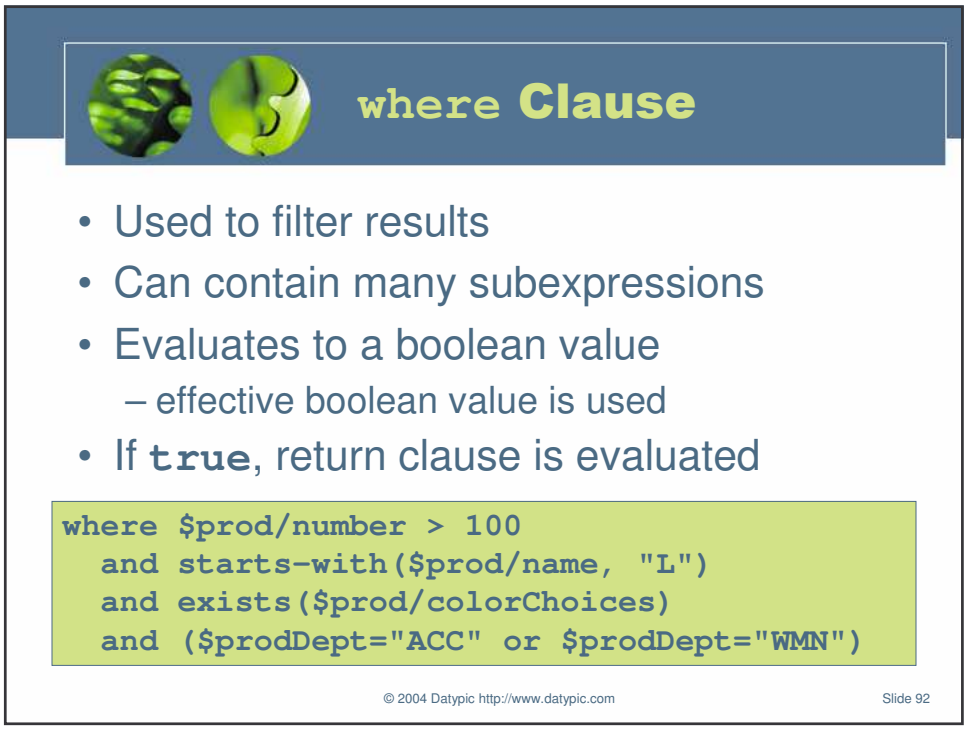

for Clauses
• Iteratively binds the variable to each item returned by the in expression
• The rest of the expression is evaluated once for each item returned
• Multiple for clauses are allowed in the same FLWOR
expression after in can evaluate to any sequence
for $prod in $cat//product
© 2004 Datypic http://www.datypic.com
Slide 87

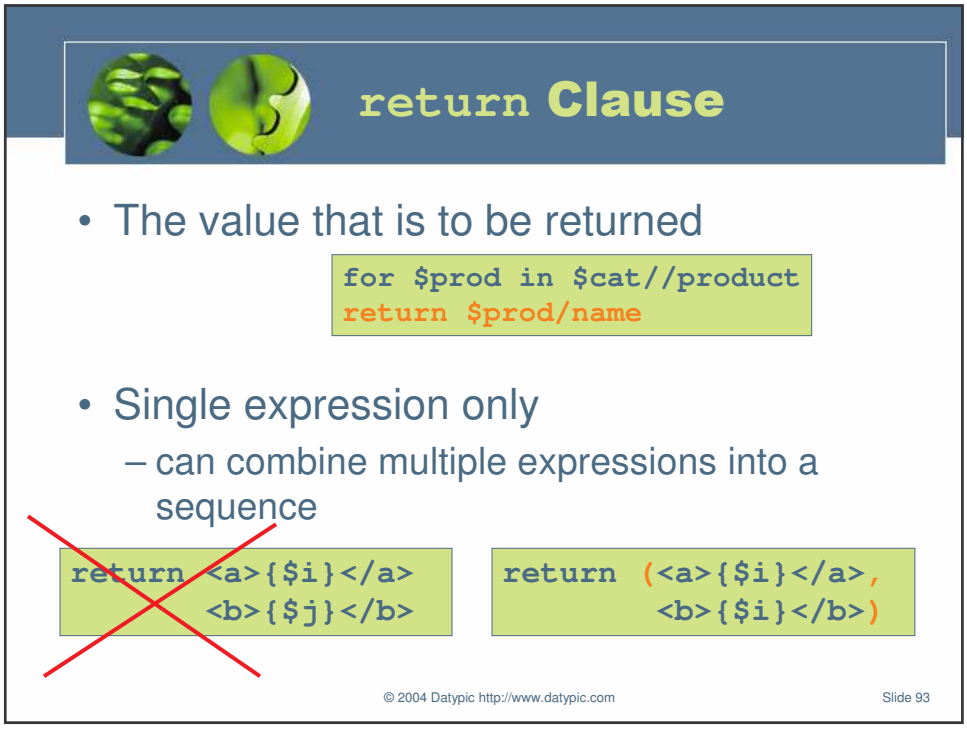

Range Expressions
• Create sequences of consecutive integers
• Use to keyword
– (1 to 5) evaluates to a sequence of 1, 2, 3, 4 and 5
• Useful in for clauses to iterate a specific number of times
for $i in (1 to 5)
...
for $i in (1 to $prodCount)
...
© 2004 Datypic http://www.datypic.com
Slide 88

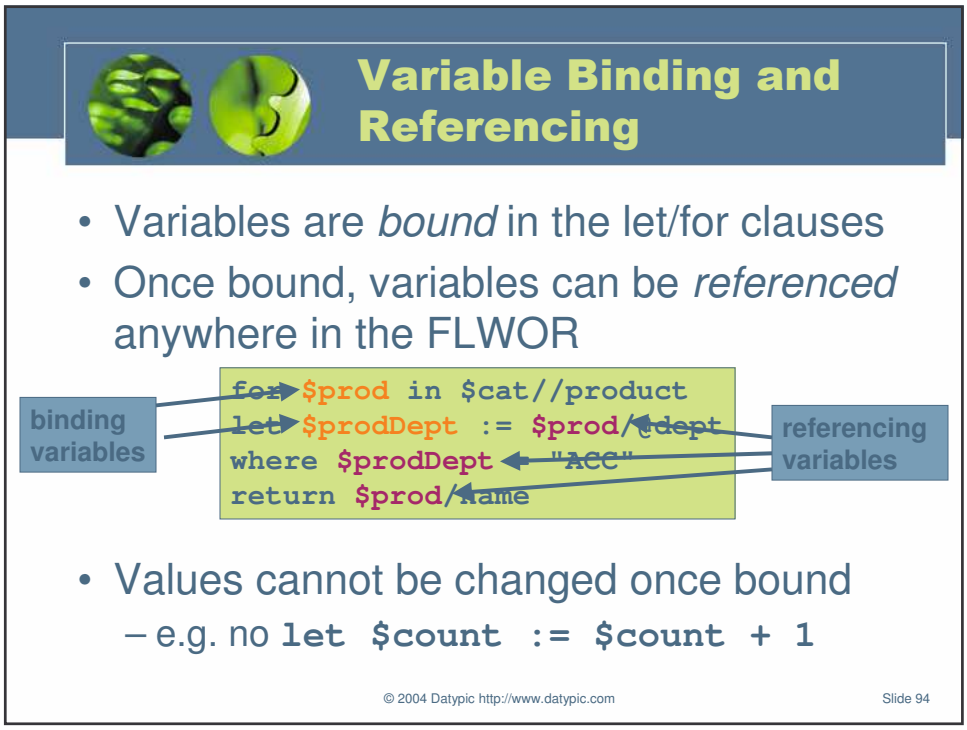

Multiple Variables in a for Clause
– Use commas to separate multiple in expressions
• or multiple for clauses
– Return clause evaluated for every combination of variable values
for $i in (1, 2), $j in (11, 12)
return <eval>i is {$i} and j is {$j}</eval>
<eval>i is 1 and j is 11</eval>
<eval>i is 1 and j is 12</eval>
<eval>i is 2 and j is 11</eval>
<eval>i is 2 and j is 12</eval>
© 2004 Datypic http://www.datypic.com
Slide 89

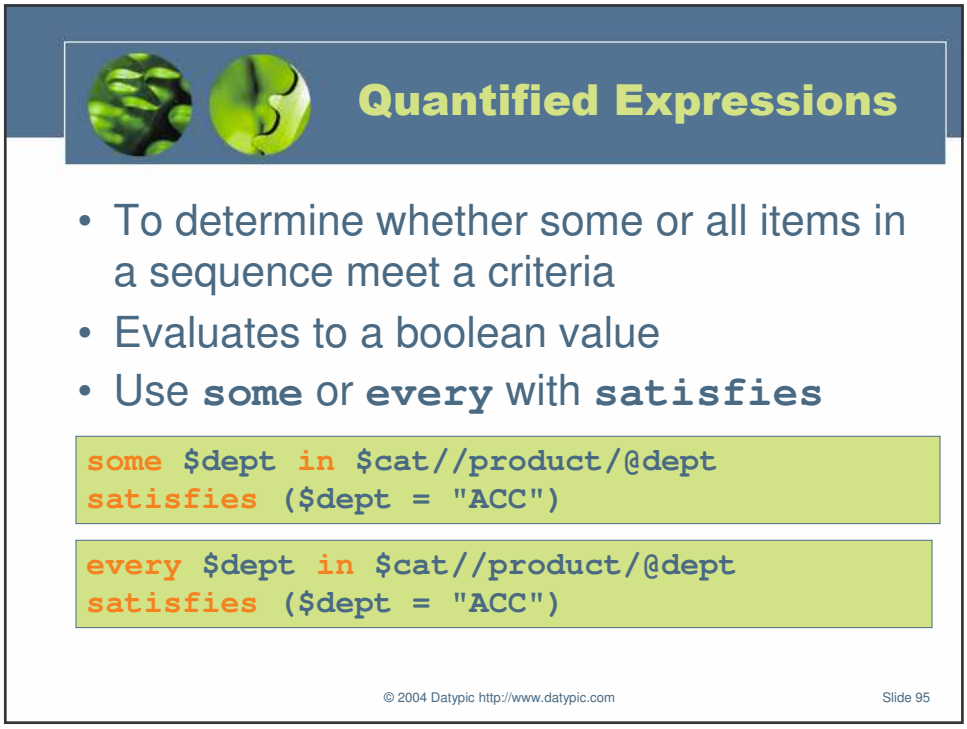

Positional Variables in for Clauses
• Positional variable keeps track of the iteration number
• Use at keyword
for $prod at $i in
$cat//product[@dept = "ACC" or @dept = "WMN"]
return <eval>{$i}. {data($prod/name)}</eval>
<eval>1. Linen Shirt</eval>
<eval>2. Ten-Gallon Hat</eval>
<eval>3. Golf Umbrella</eval>
© 2004 Datypic http://www.datypic.com
Slide 90

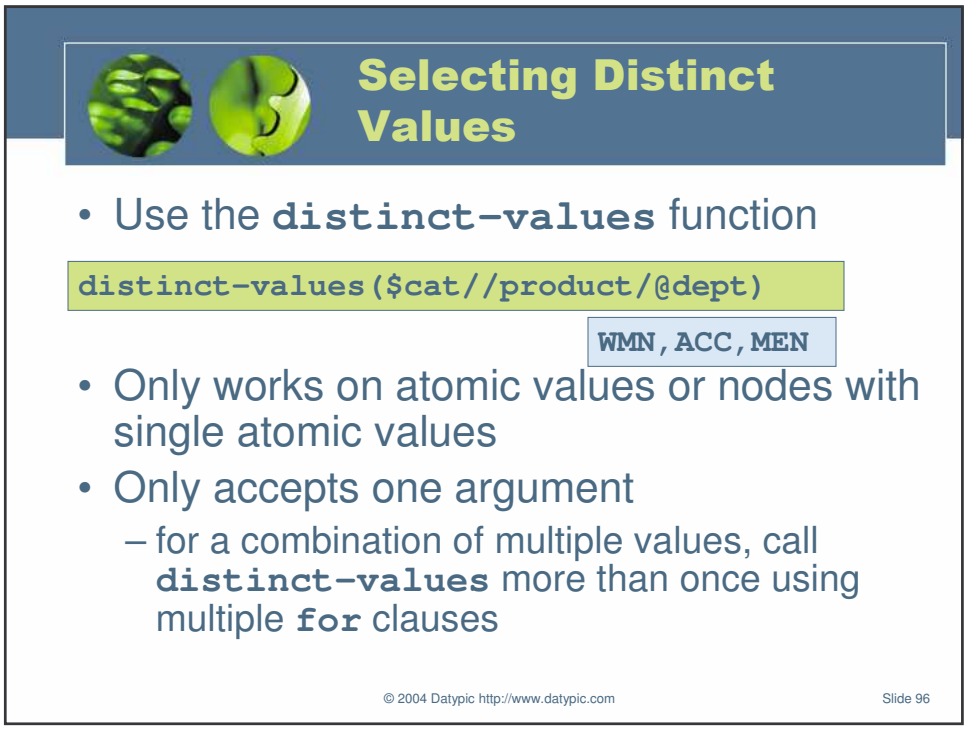

let Clauses
• Convenient way to bind a variable
– avoids repeating the same expression many times
• Does not result in iteration
for $i in (1 to 3)
return <eval>{$i}</eval>
<eval>1</eval>
<eval>2</eval>
<eval>3</eval>
let $i := (1 to 3)
return <eval>{$i}</eval>
<eval>1 2 3</eval>
© 2004 Datypic http://www.datypic.com
Slide 91

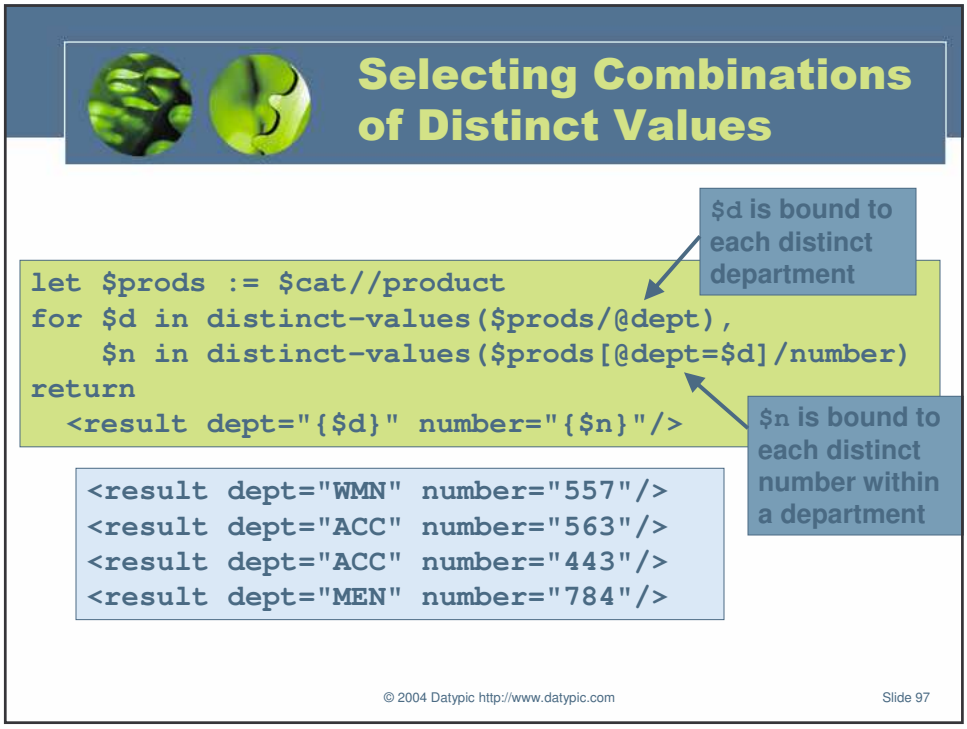

where Clause
• Used to filter results
• Can contain many subexpressions
• Evaluates to a boolean value
– effective boolean value is used
• If true, return clause is evaluated
where $prod/number > 100
and starts-with($prod/name, "L")
and exists($prod/colorChoices)
and ($prodDept="ACC" or $prodDept="WMN")
© 2004 Datypic http://www.datypic.com
Slide 92


return Clause
• The value that is to be returned
for $prod in $cat//product
return $prod/name
• Single expression only
– can combine multiple expressions into a sequence
return <a>{$i}
{$j}
return (<a>{$i},
{$i})
© 2004 Datypic http://www.datypic.com
Slide 93

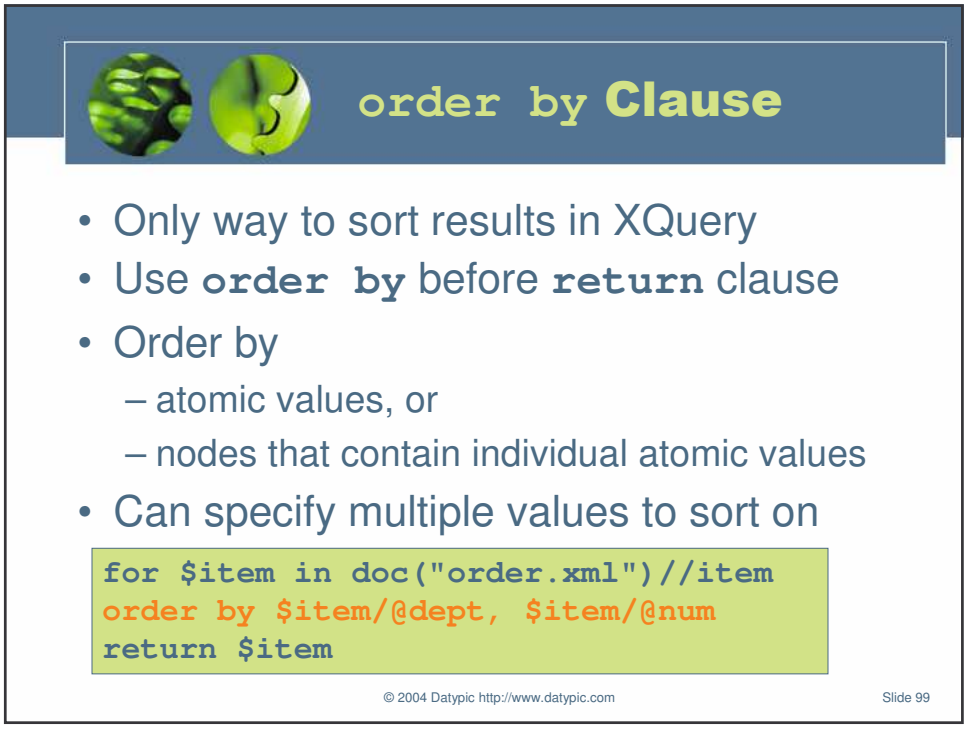

Variable Binding and Referencing
• Variables are *bound* in the let/for clauses
• Once bound, variables can be *referenced* anywhere in the FLWOR
binding variables
for $prod in $cat//product
let $prodDept := $prod/@dept
where $prodDept = "ACC"
return $prod/name
referencing variables
• Values cannot be changed once bound
– e.g. no let $count := $count + 1
© 2004 Datypic http://www.datypic.com
Slide 94

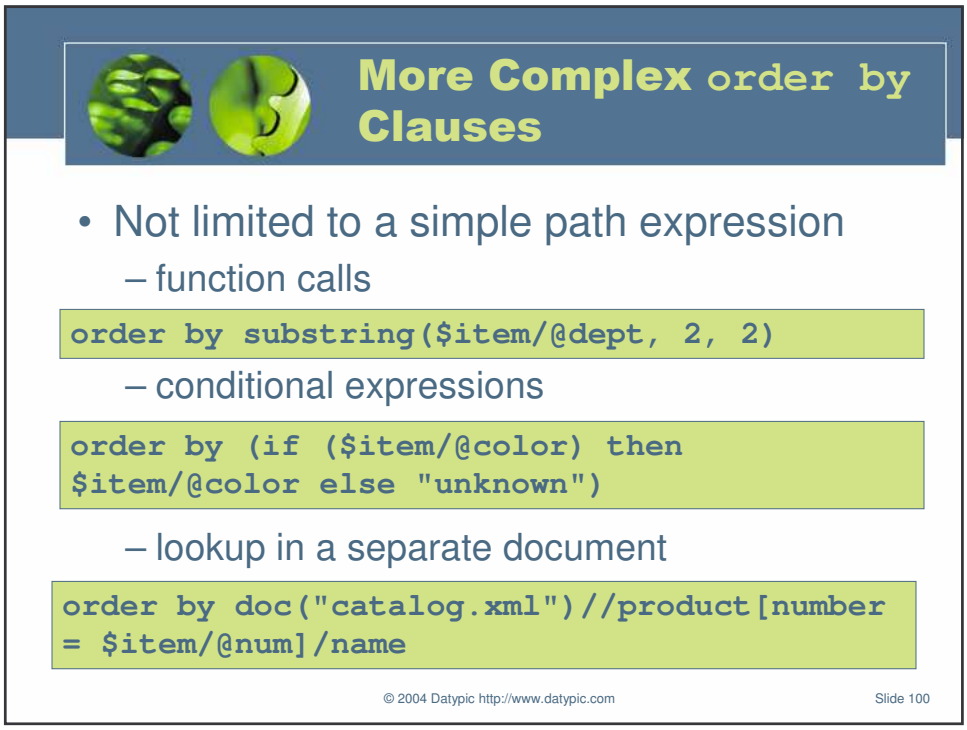

Quantified Expressions
• To determine whether some or all items in a sequence meet a criteria
• Evaluates to a boolean value
• Use some or every with satisfies
some $dept in $cat//product/@dept
satisfies ($dept = "ACC")
every $dept in $cat//product/@dept
satisfies ($dept = "ACC")
© 2004 Datypic http://www.datypic.com
Slide 95

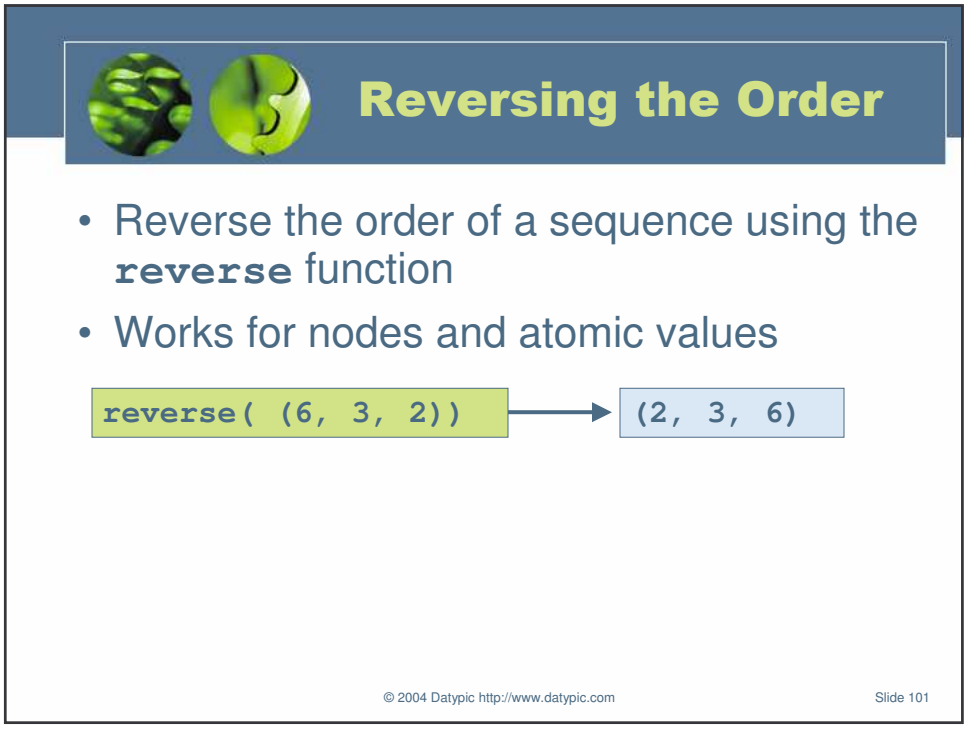

Selecting Distinct Values
• Use the distinct-values function
distinct-values($cat//product/@dept)
WMN,ACC,MEN
• Only works on atomic values or nodes with single atomic values
• Only accepts one argument
– for a combination of multiple values, call distinct-values more than once using multiple for clauses
© 2004 Datypic http://www.datypic.com
Slide 96

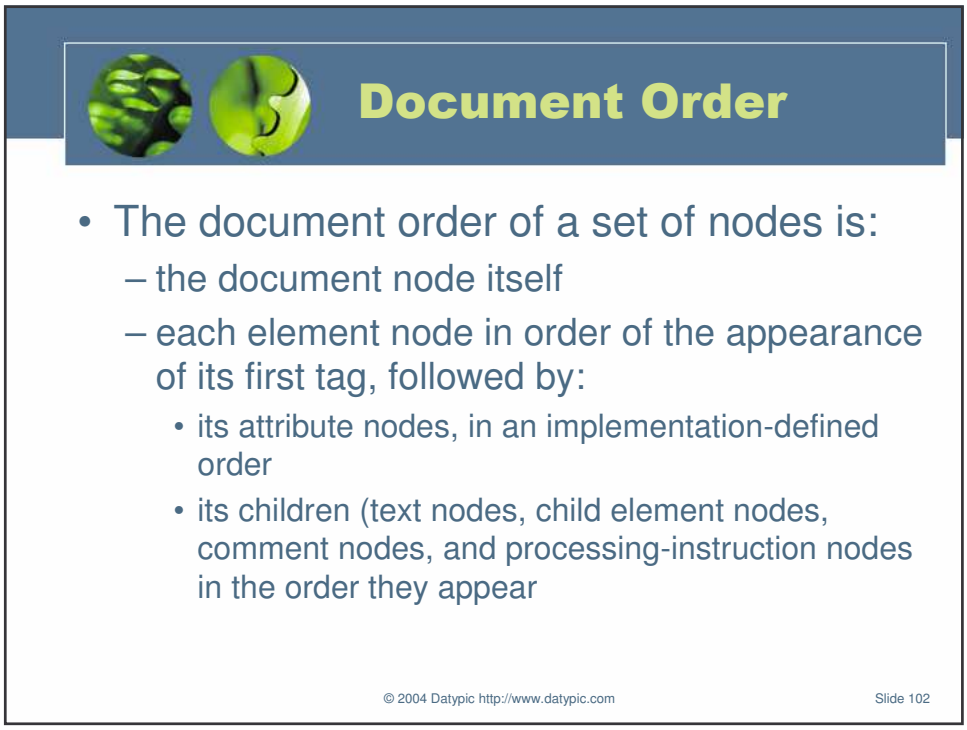

Selecting Combinations of Distinct Values
$d is bound to each distinct department
let $prods := $cat//product
for $d in distinct-values($prods/@dept),
$n in distinct-values($prods[@dept=$d]/number)
return
<result dept="{$d}" number="{$n}"/>
$n is bound to each distinct number within a department
<result dept="WMN" number="557"/>
<result dept="ACC" number="563"/>
<result dept="ACC" number="443"/>
<result dept="MEN" number="784"/>
© 2004 Datypic http://www.datypic.com
Slide 97

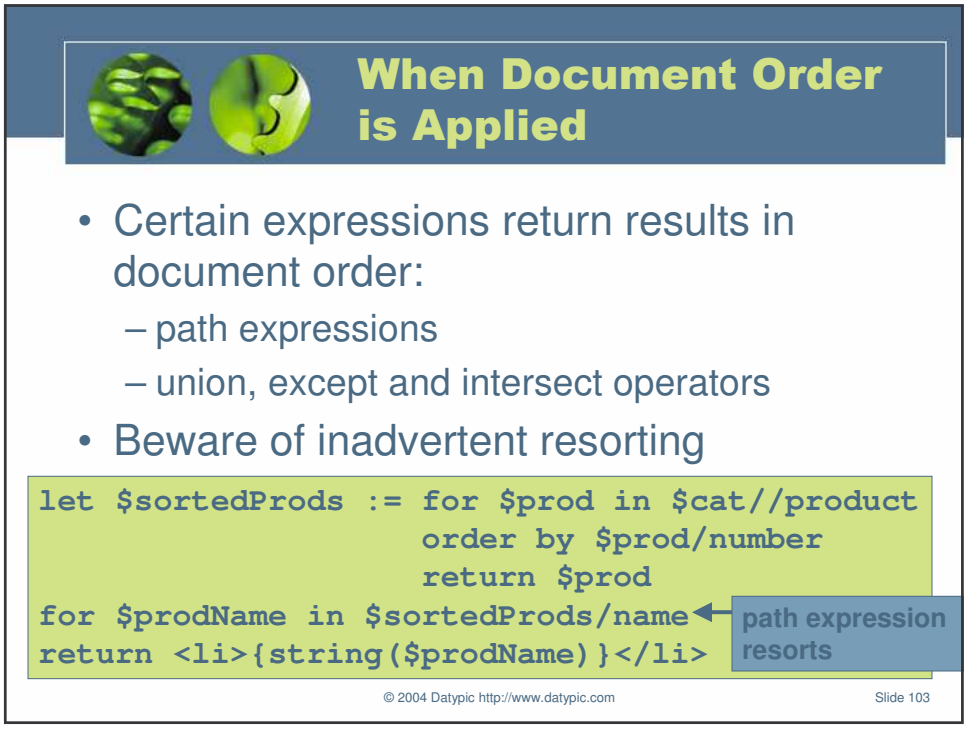

Sorting Results

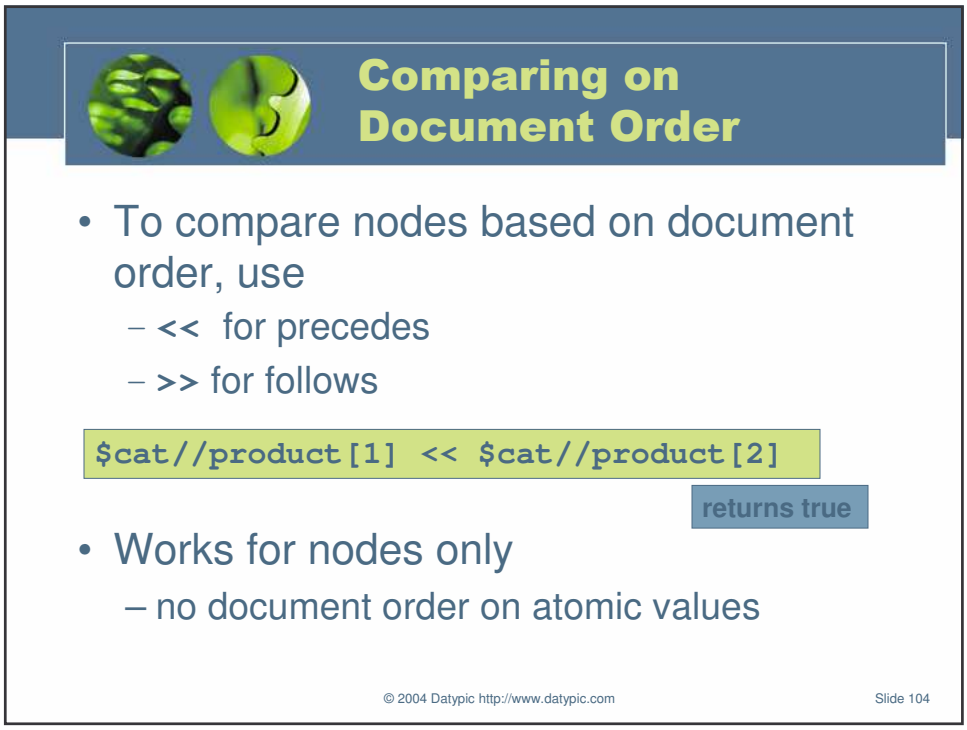

order by Clause
• Only way to sort results in XQuery
• Use order by before return clause
• Order by
– atomic values, or
– nodes that contain individual atomic values
• Can specify multiple values to sort on
for $item in doc("order.xml")//item
order by $item/@dept, $item/@num
return $item
© 2004 Datypic http://www.datypic.com
Slide 99


More Complex order by Clauses
• Not limited to a simple path expression
– function calls
order by substring($item/@dept, 2, 2)
– conditional expressions
order by (if ($item/@color) then
$item/@color else "unknown")
– lookup in a separate document
order by doc("catalog.xml")//product[number
= $item/@num]/name
© 2004 Datypic http://www.datypic.com
Slide 100


Reversing the Order
• Reverse the order of a sequence using the reverse function
• Works for nodes and atomic values
reverse((6, 3, 2))
(2, 3, 6)
© 2004 Datypic http://www.datypic.com
Slide 101


Document Order
• The document order of a set of nodes is:
– the document node itself
– each element node in order of the appearance of its first tag, followed by:
• its attribute nodes, in an implementation-defined order
• its children (text nodes, child element nodes, comment nodes, and processing-instruction nodes in the order they appear
© 2004 Datypic http://www.datypic.com
Slide 102

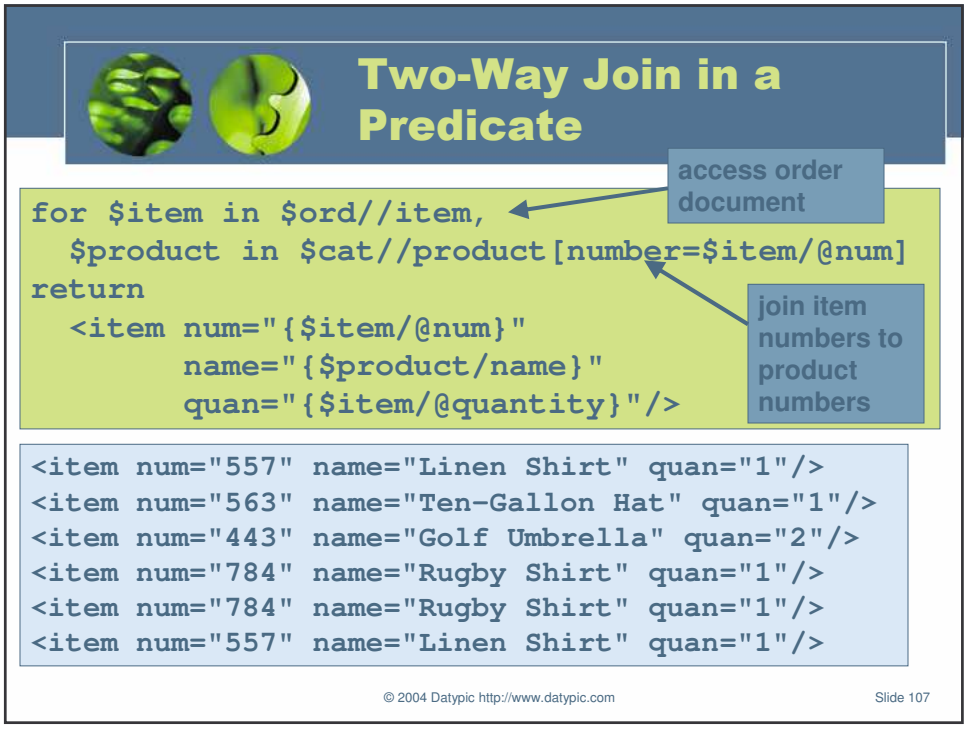

When Document Order is Applied
• Certain expressions return results in document order:
– path expressions
– union, except and intersect operators
• Beware of inadvertent resorting
let $sortedProds := for $prod in $cat//product
order by $prod/number
return $prod
for $prodName in $sortedProds/name
return {string($prodName)}
path expression resorts
© 2004 Datypic http://www.datypic.com
Slide 103

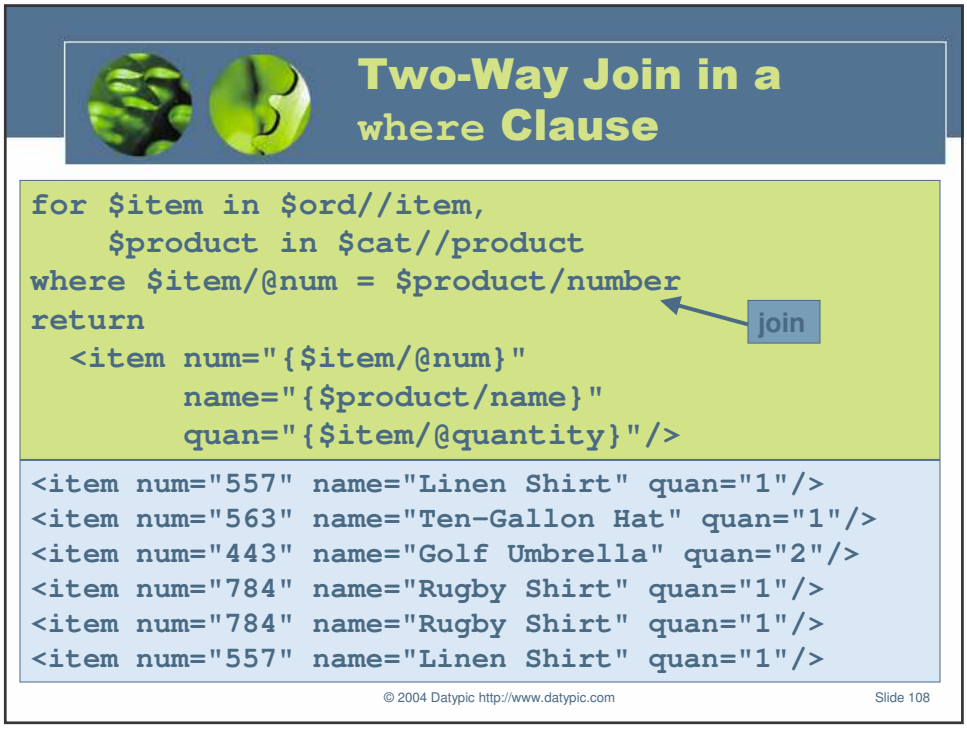

Comparing on Document Order
• To compare nodes based on document order, use
– << for precedes
– >> for follows
$cat//product[1] << $cat//product[2]
returns true
• Works for nodes only
– no document order on atomic values
© 2004 Datypic http://www.datypic.com
Slide 104

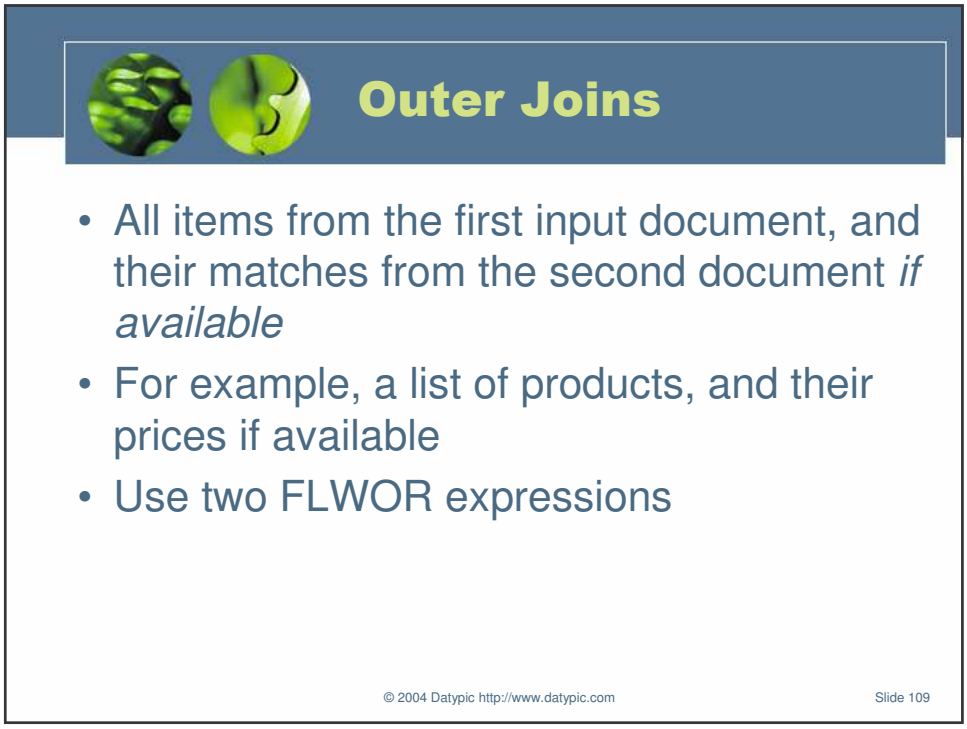
Joins

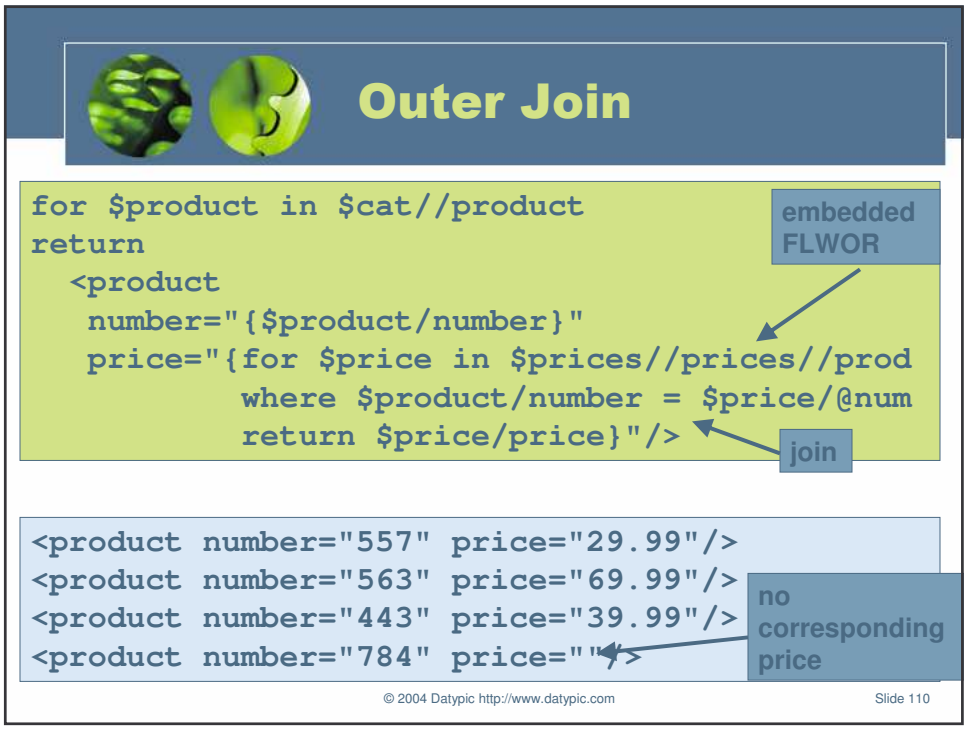
- Combining two or more input documents into one result set
- Another use for multiple `for` clauses

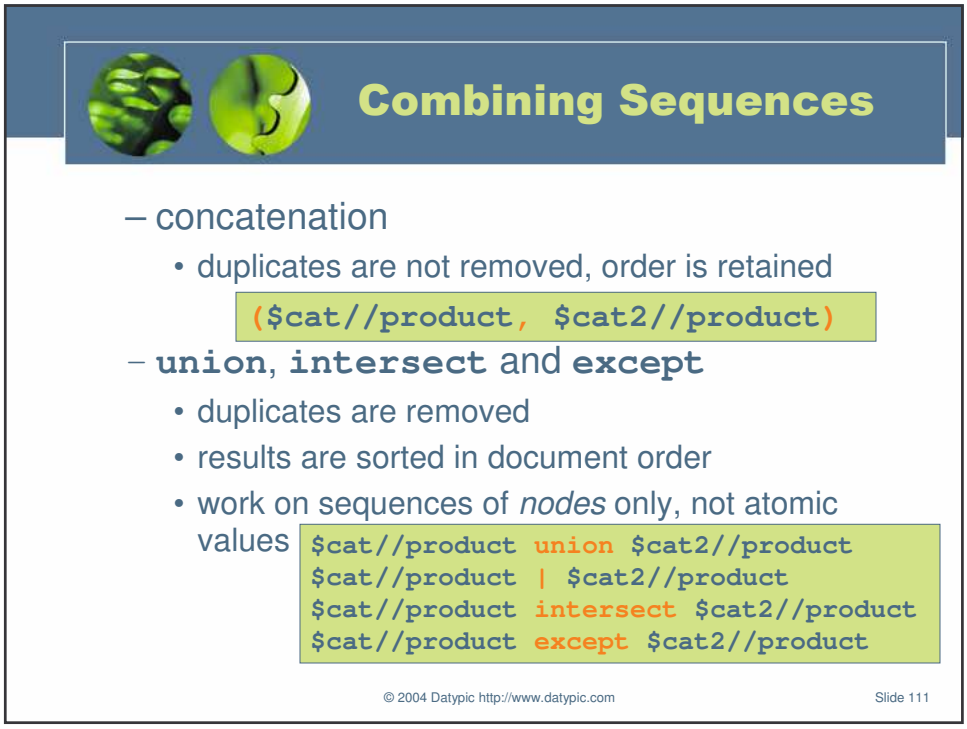
 Slide 106

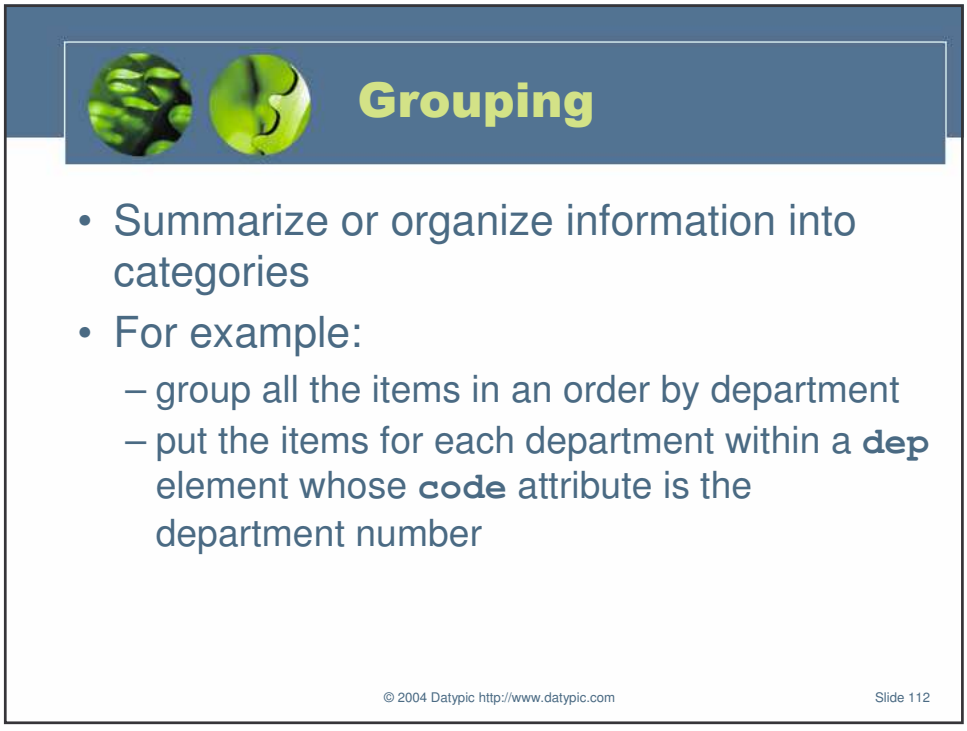

Two-Way Join in a Predicate
access order document
for $item in $ord//item,
$product in $cat//product[number=$item/@num]
return
<item num="{$item/@num}"
name="{$product/name}"
quan="{$item/@quantity}"/>
join item numbers to product numbers
<item num="557" name="Linen Shirt" quan="1"/>
<item num="563" name="Ten-Gallon Hat" quan="1"/>
<item num="443" name="Golf Umbrella" quan="2"/>
<item num="784" name="Rugby Shirt" quan="1"/>
<item num="784" name="Rugby Shirt" quan="1"/>
<item num="557" name="Linen Shirt" quan="1"/>
© 2004 Datypic http://www.datypic.com
Slide 107

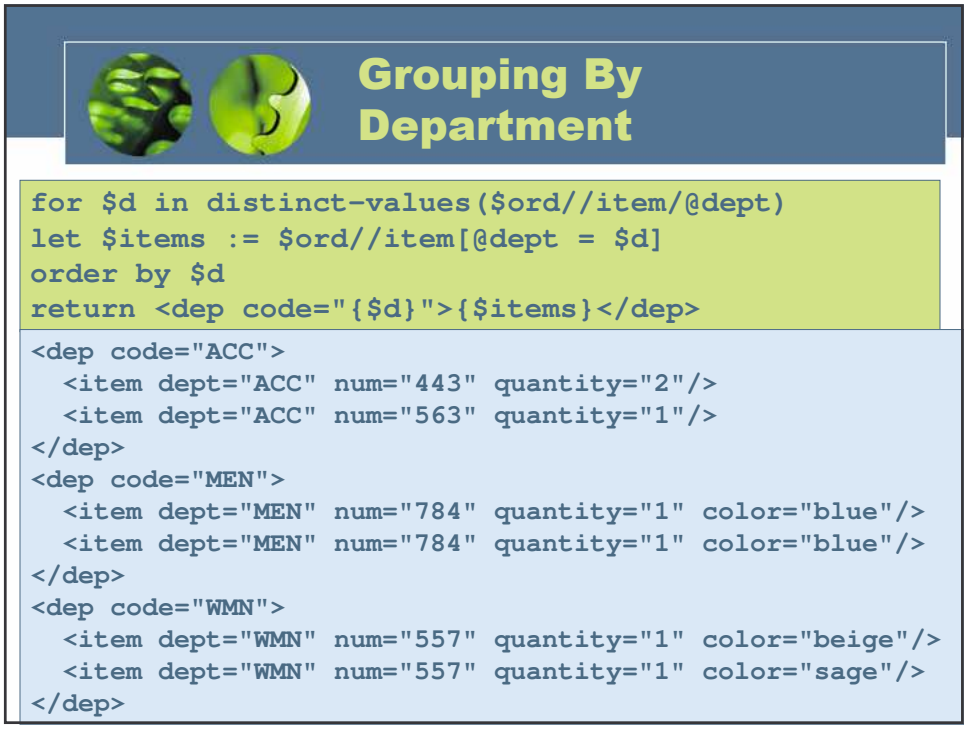

Two-Way Join in a where Clause
for $item in $ord//item,
$product in $cat//product
where $item/@num = $product/number
return
<item num="{$item/@num}"
name="{$product/name}"
quan="{$item/@quantity}"/>
join
<item num="557" name="Linen Shirt" quan="1"/>
<item num="563" name="Ten-Gallon Hat" quan="1"/>
<item num="443" name="Golf Umbrella" quan="2"/>
<item num="784" name="Rugby Shirt" quan="1"/>
<item num="784" name="Rugby Shirt" quan="1"/>
<item num="557" name="Linen Shirt" quan="1"/>
© 2004 Datypic http://www.datypic.com
Slide 108

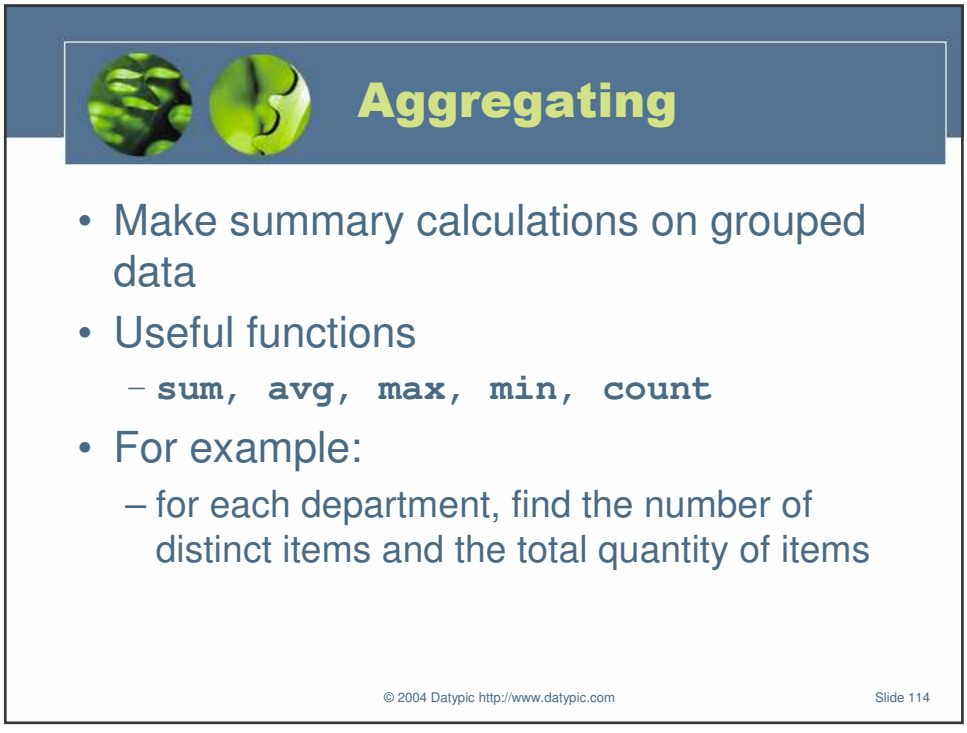

Outer Joins
• All items from the first input document, and their matches from the second document *if available*
• For example, a list of products, and their prices if available
• Use two FLWOR expressions
© 2004 Datypic http://www.datypic.com
Slide 109

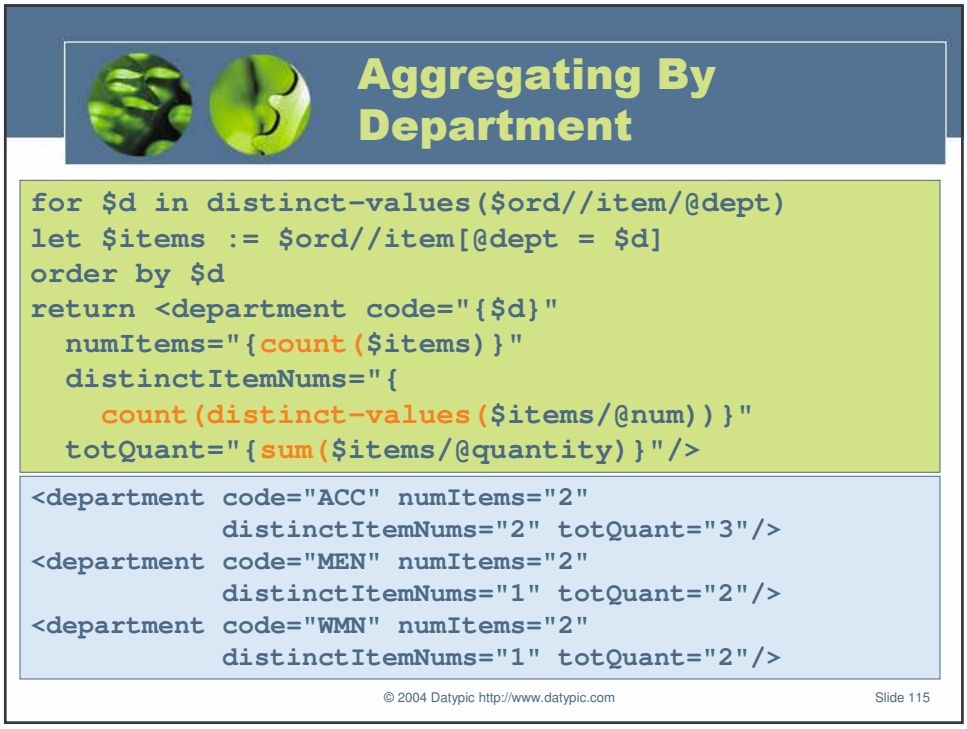

Outer Join
for $product in $cat//product
return
<product
number="{$product/number}"
price="{for $price in $prices//prices//prod
where $product/number = $price/@num
return $price/price}"/>
embedded FLWOR
join
<product number="557" price="29.99"/>
<product number="563" price="69.99"/>
<product number="443" price="39.99"/>
<product number="784" price=""/>
no corresponding price
© 2004 Datypic http://www.datypic.com
Slide 110

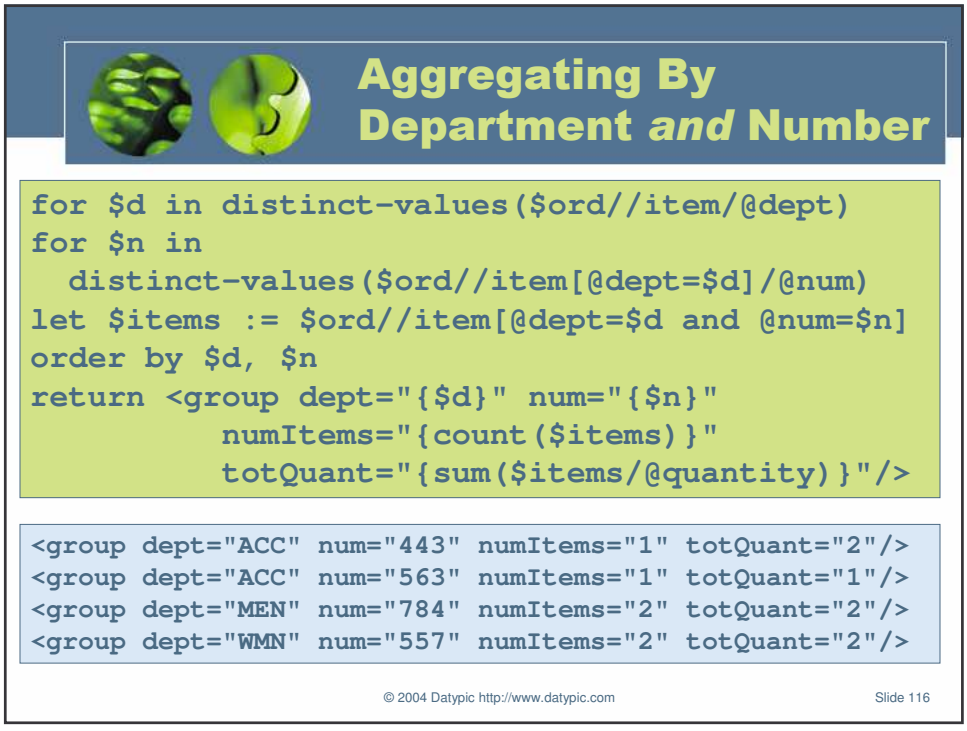

Combining Sequences
– concatenation
• duplicates are not removed, order is retained
($cat//product, $cat2//product)
– union, intersect and except
• duplicates are removed
• results are sorted in document order
• work on sequences of nodes only, not atomic values
$cat//product union $cat2//product
$cat//product | $cat2//product
$cat//product intersect $cat2//product
$cat//product except $cat2//product
© 2004 Datypic http://www.datypic.com
Slide 111


Grouping
• Summarize or organize information into categories
• For example:
– group all the items in an order by department
– put the items for each department within a dep element whose code attribute is the department number
© 2004 Datypic http://www.datypic.com
Slide 112


Grouping By Department
for $d in distinct-values($ord//item/@dept)
let $items := $ord//item[@dept = $d]
order by $d
return <dep code="{$d}">{$items}</dep>
<dep code="ACC">
<item dept="ACC" num="443" quantity="2"/>
<item dept="ACC" num="563" quantity="1"/>
</dep>
<dep code="MEN">
<item dept="MEN" num="784" quantity="1" color="blue"/>
<item dept="MEN" num="784" quantity="1" color="blue"/>
</dep>
<dep code="WMN">
<item dept="WMN" num="557" quantity="1" color="beige"/>
<item dept="WMN" num="557" quantity="1" color="sage"/>
</dep>


Aggregating
• Make summary calculations on grouped data
• Useful functions
– sum, avg, max, min, count
• For example:
– for each department, find the number of distinct items and the total quantity of items
© 2004 Datypic http://www.datypic.com
Slide 114

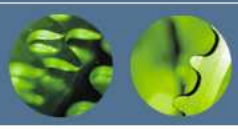

Aggregating By Department
for $d in distinct-values($ord//item/@dept)
let $items := $ord//item[@dept = $d]
order by $d
return <department code="{$d}"
numItems="{count($items)}"
distinctItemNums="{
count(distinct-values($items/@num))}"
totQuant="{sum($items/@quantity)}"/>
<department code="ACC" numItems="2"
distinctItemNums="2" totQuant="3"/>
<department code="MEN" numItems="2"
distinctItemNums="1" totQuant="2"/>
<department code="WMN" numItems="2"
distinctItemNums="1" totQuant="2"/>
© 2004 Datypic http://www.datypic.com
Slide 115


Aggregating By Department and Number
for $d in distinct-values($ord//item/@dept)
for $n in
distinct-values($ord//item[@dept=$d]/@num)
let $items := $ord//item[@dept=$d and @num=$n]
order by $d, $n
return <group dept="{$d}" num="{$n}"
numItems="{count($items)}"
totQuant="{sum($items/@quantity)}"/>
<group dept="ACC" num="443" numItems="1" totQuant="2"/>
<group dept="ACC" num="563" numItems="1" totQuant="1"/>
<group dept="MEN" num="784" numItems="2" totQuant="2"/>
<group dept="WMN" num="557" numItems="2" totQuant="2"/>
© 2004 Datypic http://www.datypic.com
Slide 116

A Product Catalog with a Namespace

```
<catalog xmlns="http://datypic.com/cat"
       xmlns:prod="http://datypic.com/prod">
  <number>1446</number>
  <prod:product>
    <prod:number>563</prod:number>
    <prod:name language="en">Ten-Gallon
         Hat</prod:name>
  </prod:product>
</catalog>
```

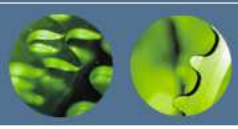
- For examples in this course, this document is bound to **`$nscat`** variable



Qualified Names

- Names in XQuery queries are namespace-qualified:
  - elements and attributes
    - from input documents, newly constructed elements
  - functions
  - types
  - variables

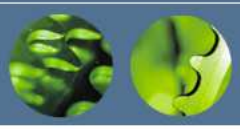
 Slide 119



Built-in Prefixes and Namespaces

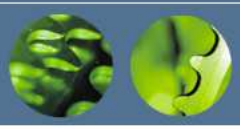
Prefix	Namespace	Uses
`xml`	`http://www.w3.org/XML/1998/namespace`	XML attributes such as `xml:lang` and `xml:space`.
`xs`	`http://www.w3.org/2001/XMLSchema`	XML Schema built-in types
`xsi`	`http://www.w3.org/2001/XMLSchema-instance`	XML Schema instance attributes such as `xsi:type` and `xsi:nil`.
`fn`	`http://www.w3.org/200?/??/xpath-functions`	All built-in functions
`xdt`	`http://www.w3.org/200?/??/xpath-datatypes`	XQuery built-in types
`local`	`http://www.w3.org/200?/??/xquery-local-functions`	locally declared functions that are not in a specific namespace

 Slide 120



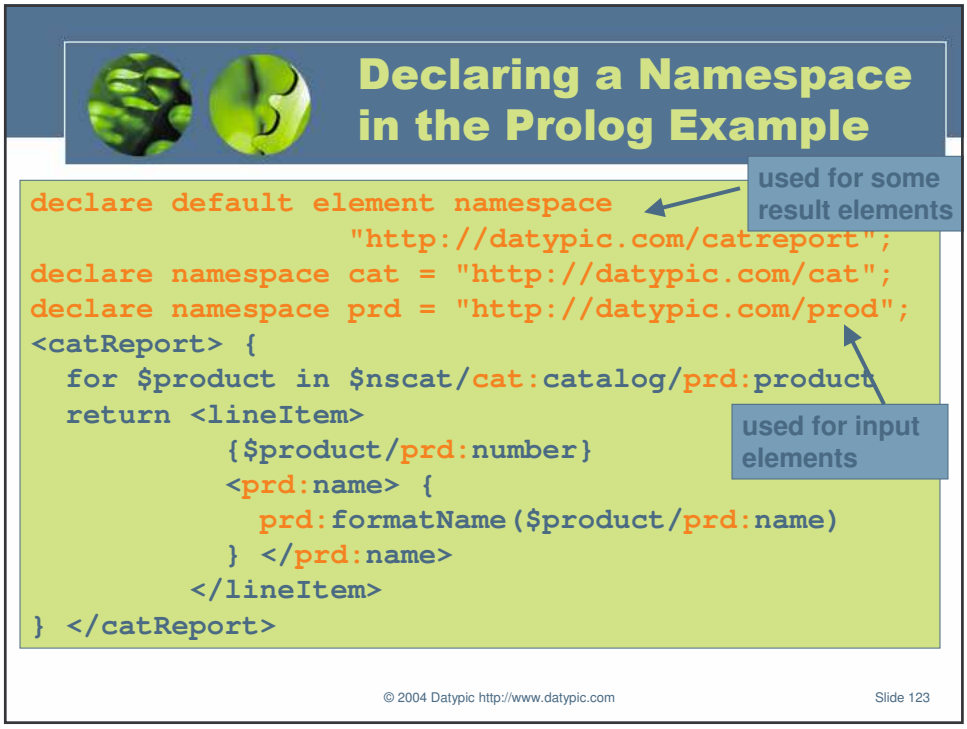
Declaring Namespaces in Queries

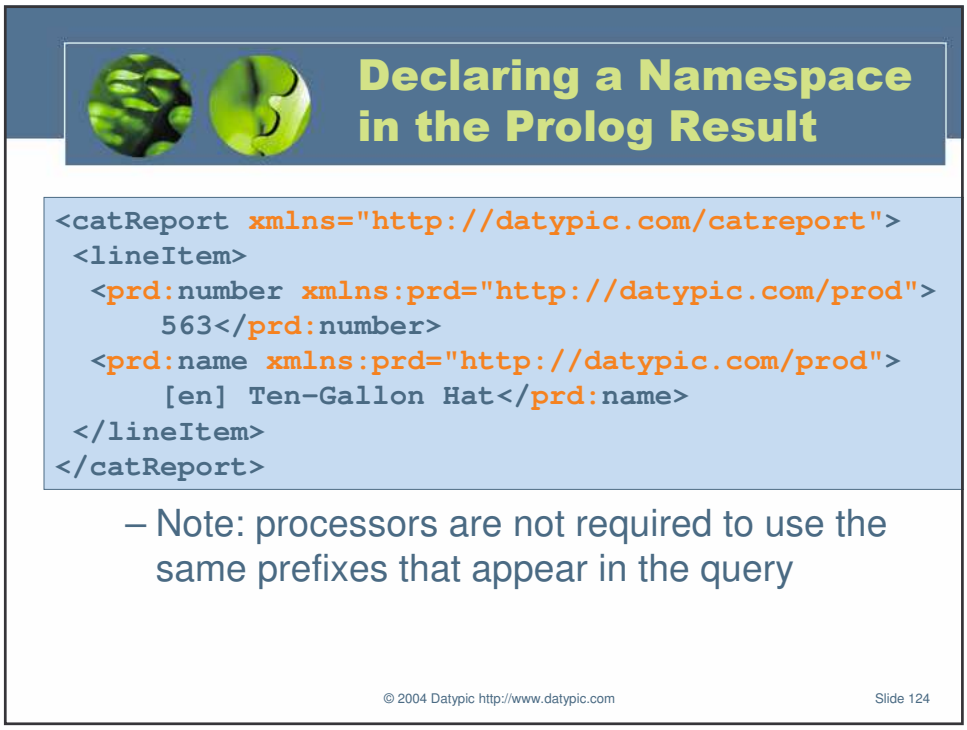
- 2 ways:
  - in the query prolog
  - in a direct element constructor

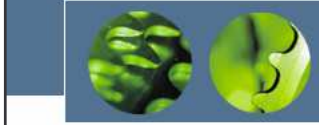


Declaring a Namespace in the Prolog

- In scope throughout the query body
- If it associated with a prefix:
  - prefix can be used with any name
- If it is a default namespace declaration:
  - applies to elements, types
    - *even in path expressions* (unlike XPath1.0)
  - not to variables, functions


Declaring a Namespace in the Prolog Example
used for some result elements
used for input elements
declare default element namespace
"http://datypic.com/catreport";
declare namespace cat = "http://datypic.com/cat";
declare namespace prd = "http://datypic.com/prod";
<catReport> {
for $product in $nscat/cat:catalog/prd:product
return <lineItem>
{$product/prd:number}
<prd:name> {
prd:formatName($product/prd:name)
} </prd:name>
</lineItem>
} </catReport>
© 2004 Datypic http://www.datypic.com
Slide 123


Declaring a Namespace in the Prolog Result
<catReport xmlns="http://datypic.com/catreport">
<lineItem>
<prd:number xmlns:prd="http://datypic.com/prod">
563</prd:number>
<prd:name xmlns:prd="http://datypic.com/prod">
[en] Ten-Gallon Hat</prd:name>
</lineItem>
</catReport>
– Note: processors are not required to use the same prefixes that appear in the query
© 2004 Datypic http://www.datypic.com
Slide 124



Declaring a Namespace in a Direct Constructor

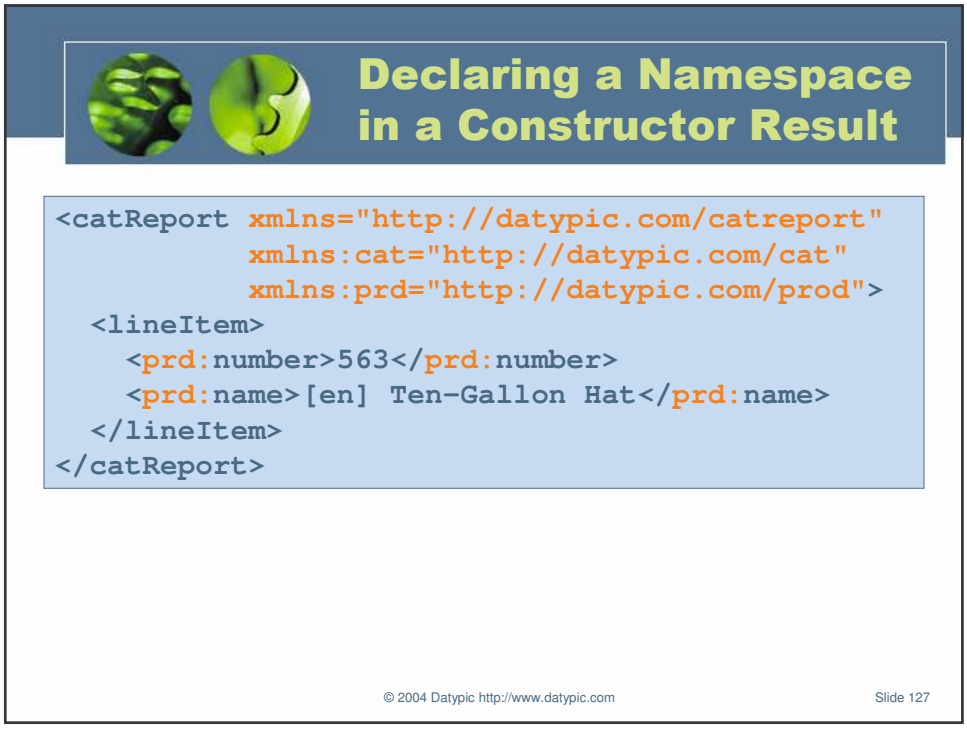
- Uses XML-like syntax
- Must be static
  - unlike attributes, value cannot be expressed as an enclosed expression
- Scope is the element being constructed

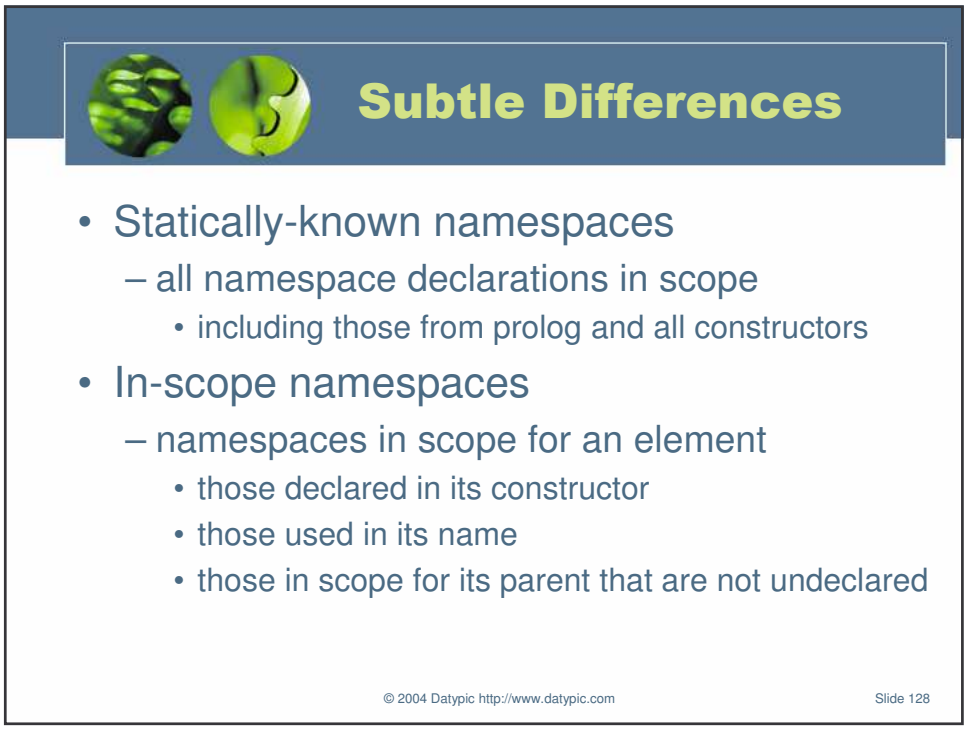
```
<catReport xmlns="http://datypic.com/catreport"
           xmlns:cat="http://datypic.com/cat"...
```


Declaring a Namespace in a Direct Constructor

```
<catReport xmlns="http://datypic.com/catreport"
           xmlns:cat="http://datypic.com/cat"
           xmlns:prd="http://datypic.com/prod"> {
for $product in $nscat/cat:catalog/prd:product
  return <lineItem>
           {$product/prd:number}
           <prd:name> {
             prd:formatName($product/prd:name)
           } </prd:name>
         </lineItem>
} </catReport>
```

- Same results as previous example
  - but namespace declarations are different

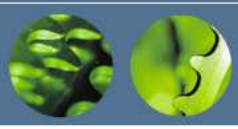

Declaring a Namespace in a Constructor Result
<catReport xmlns="http://datypic.com/catreport"
xmlns:cat="http://datypic.com/cat"
xmlns:prd="http://datypic.com/prod">
<lineItem>
<prd:number>563</prd:number>
<prd:name>[en] Ten-Gallon Hat</prd:name>
</lineItem>
</catReport>
© 2004 Datypic http://www.datypic.com
Slide 127


Subtle Differences
• Statically-known namespaces
– all namespace declarations in scope
• including those from prolog and all constructors
• In-scope namespaces
– namespaces in scope for an element
• those declared in its constructor
• those used in its name
• those in scope for its parent that are not undeclared
© 2004 Datypic http://www.datypic.com
Slide 128

Functions

- Built-in functions
  - over 100 functions built into XQuery
  - names do not need to be prefixed when called
- User-defined functions
  - defined in the query or in a separate module
  - names must be prefixed

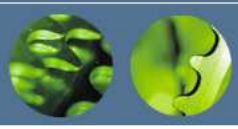
 Slide 130



Built-In Functions: A Sample

- String-related
  - `substring, contains, matches, concat, normalize-space, tokenize`
- Date-related
  - `current-date, months-from-date, adjust-time-to-timezone`
- Number-related
  - `round, avg, sum, ceiling`
- Sequence-related
  - `index-of, insert-before, reverse, subsequence, distinct-values`

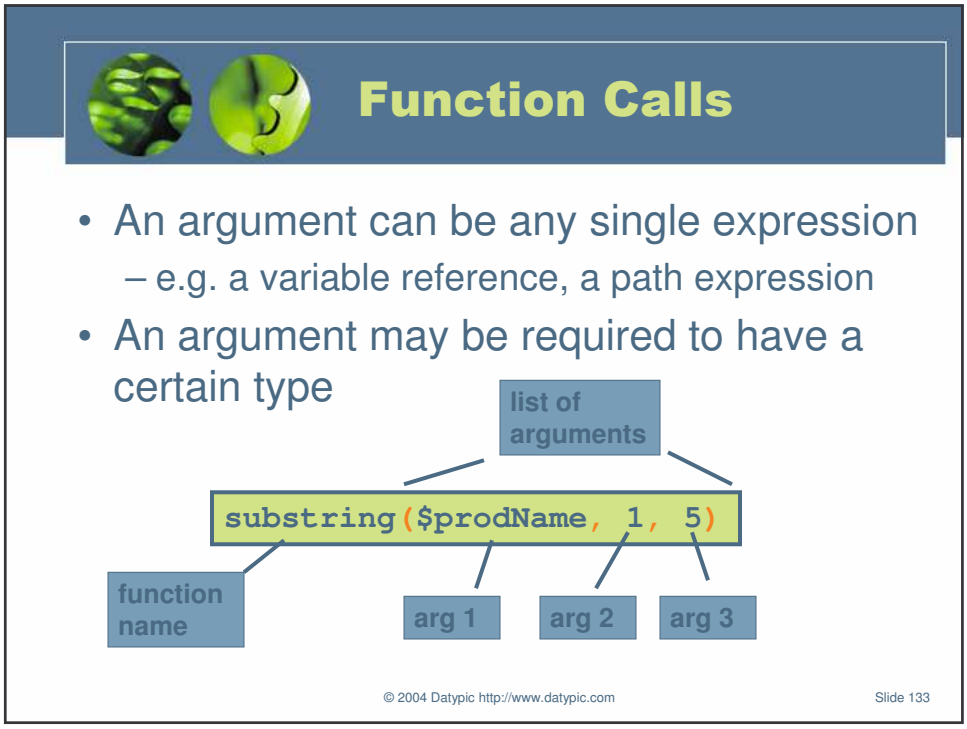
 Slide 131

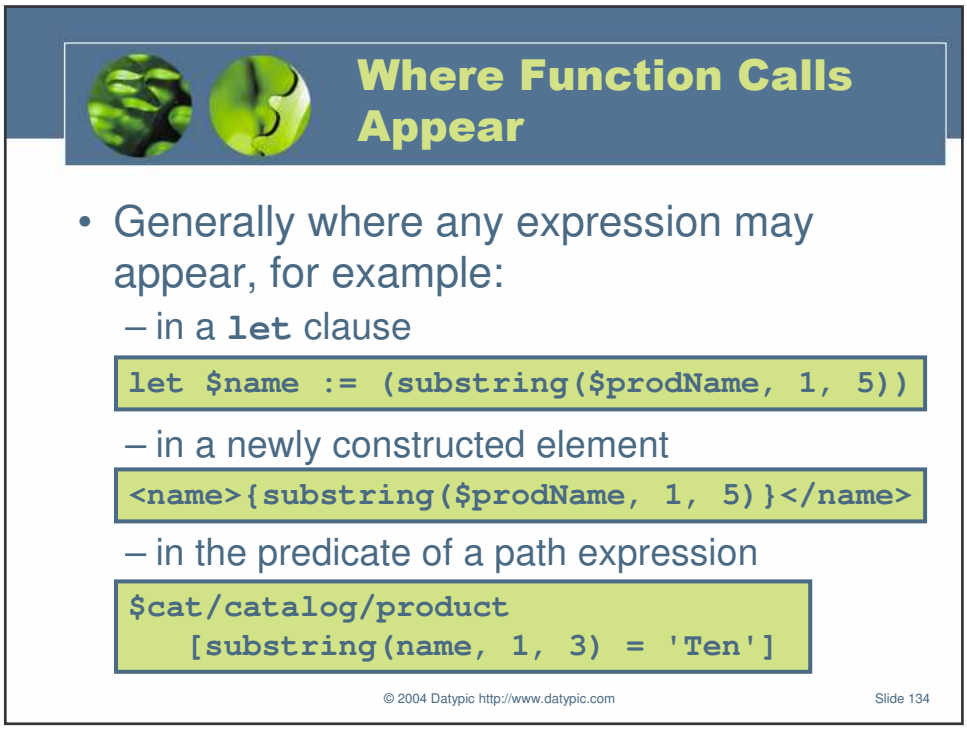


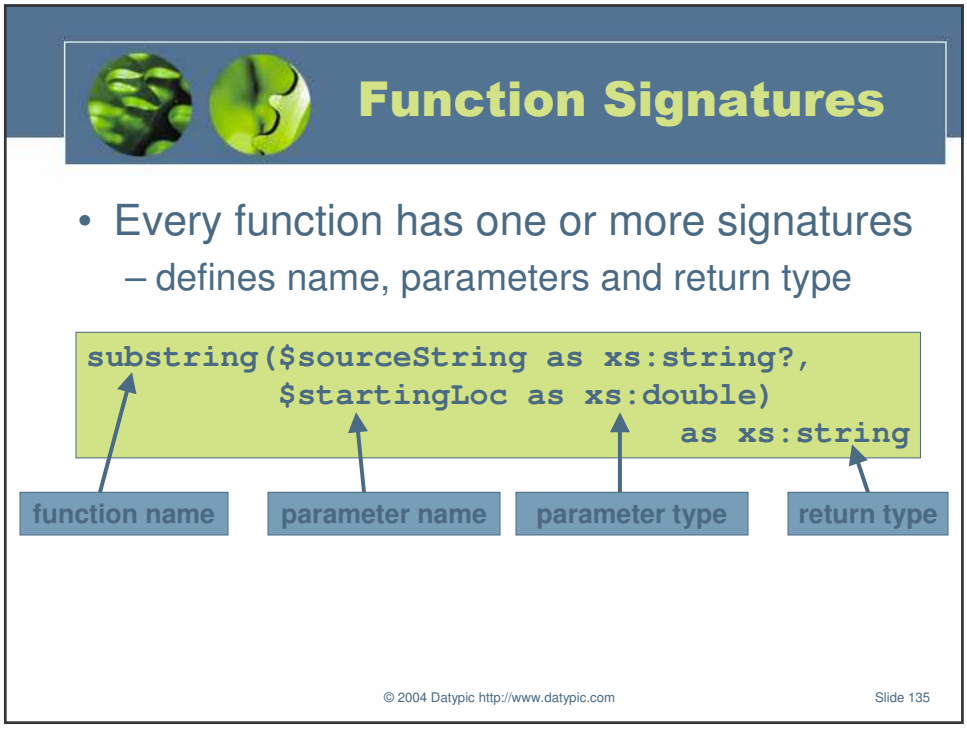
More Built-In Functions

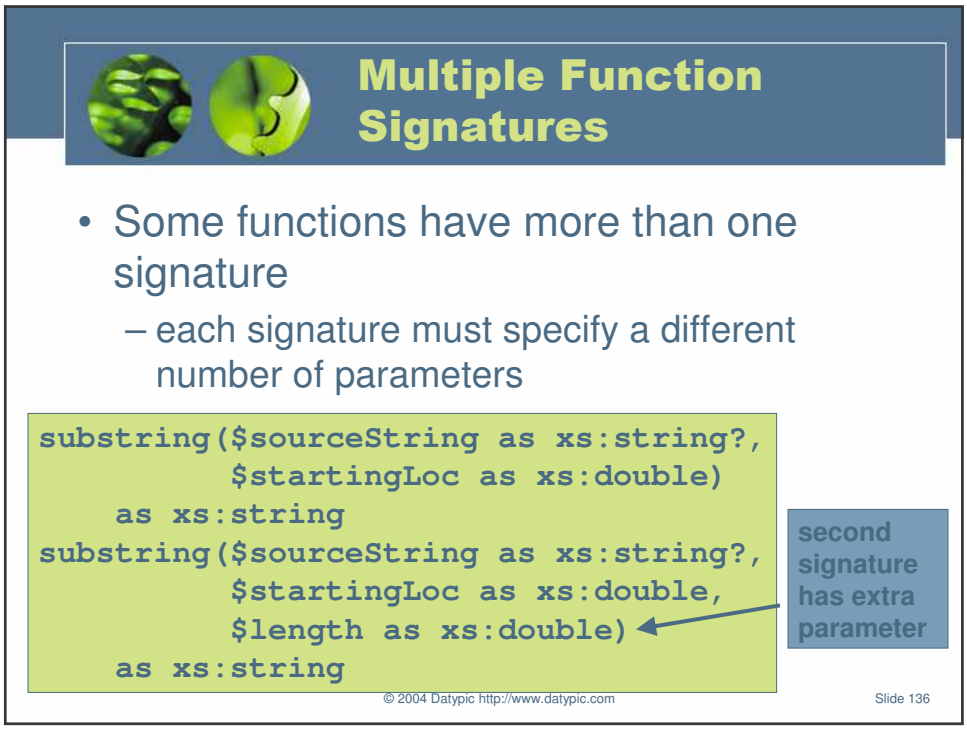
- Node-related
  - `data, empty, exists, id, idref`
- Name-related
  - `local-name, in-scope-prefixes, QName, resolve-QName`
- Error handling and trapping
  - `error, trace, exactly-one`
- Document- and URI-related
  - `collection, doc, root, base-uri`

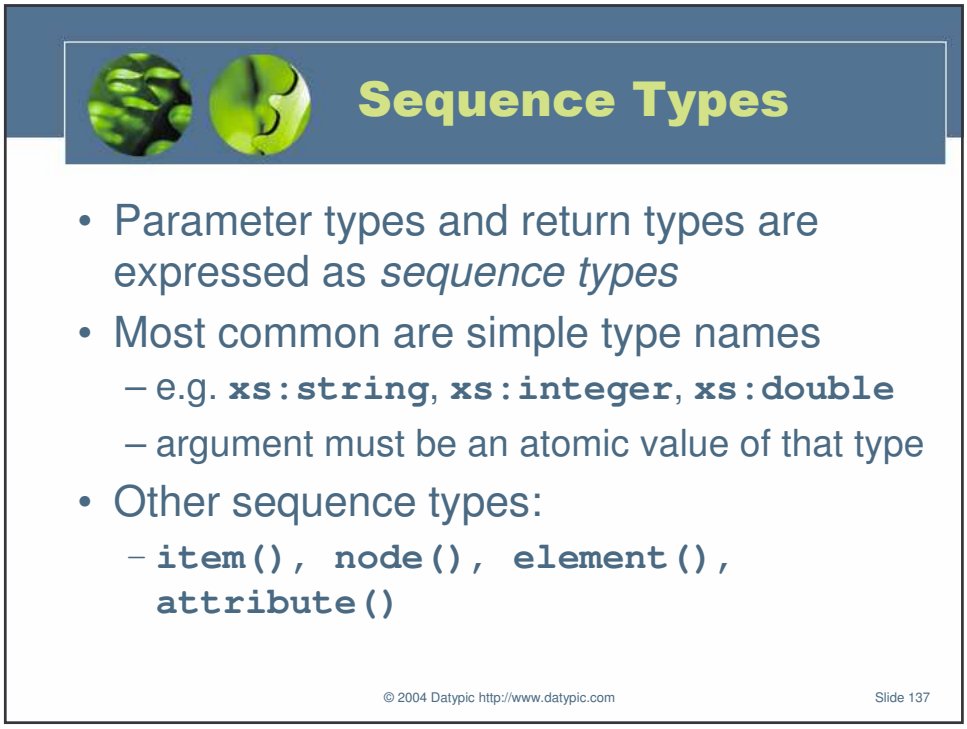
 Slide 132

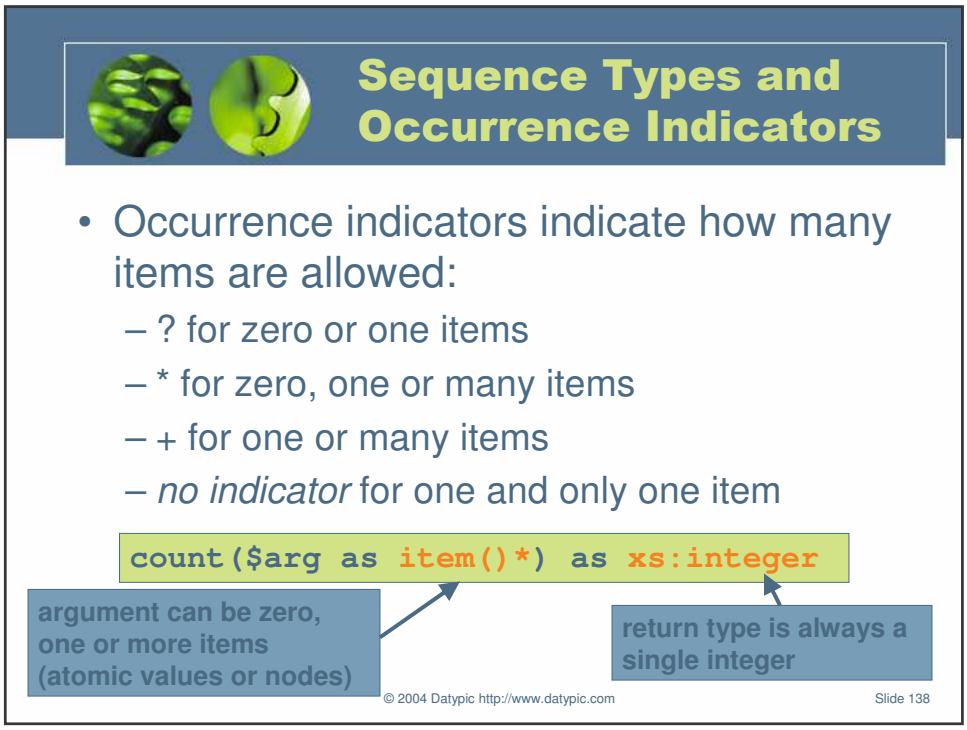

Function Calls
• An argument can be any single expression
– e.g. a variable reference, a path expression
• An argument may be required to have a certain type
list of arguments
substring($prodName, 1, 5)
function name
arg 1
arg 2
arg 3
© 2004 Datypic http://www.datypic.com
Slide 133

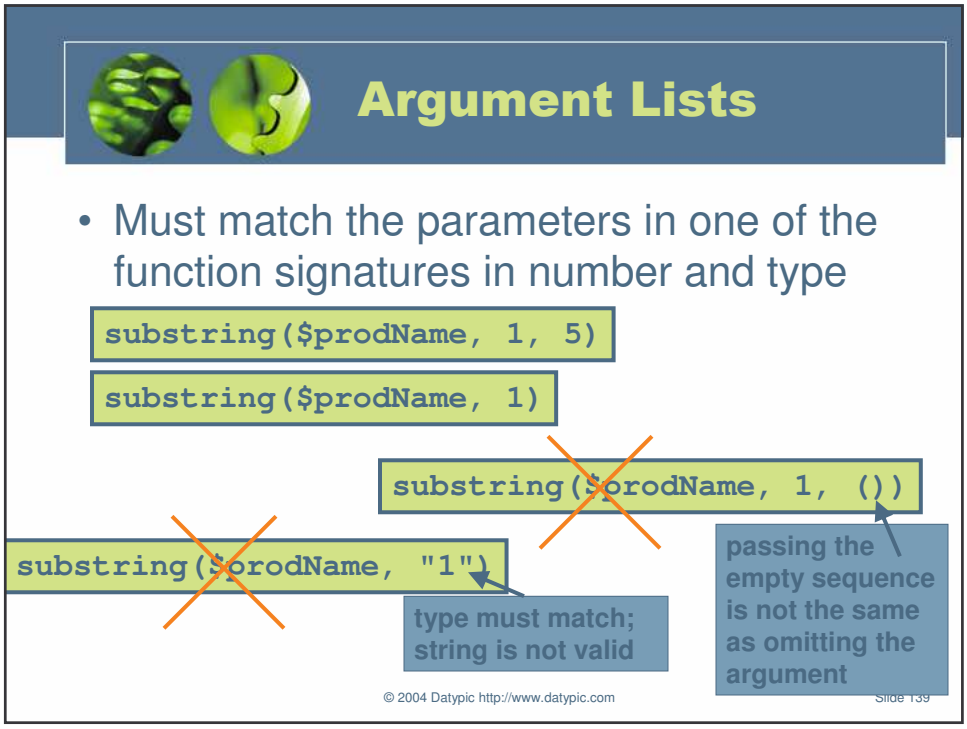

Where Function Calls Appear
• Generally where any expression may appear, for example:
– in a let clause
let $name := (substring($prodName, 1, 5))
– in a newly constructed element
<name>{substring($prodName, 1, 5)}</name>
– in the predicate of a path expression
$cat/catalog/product
[substring(name, 1, 3) = 'Ten']
© 2004 Datypic http://www.datypic.com
Slide 134

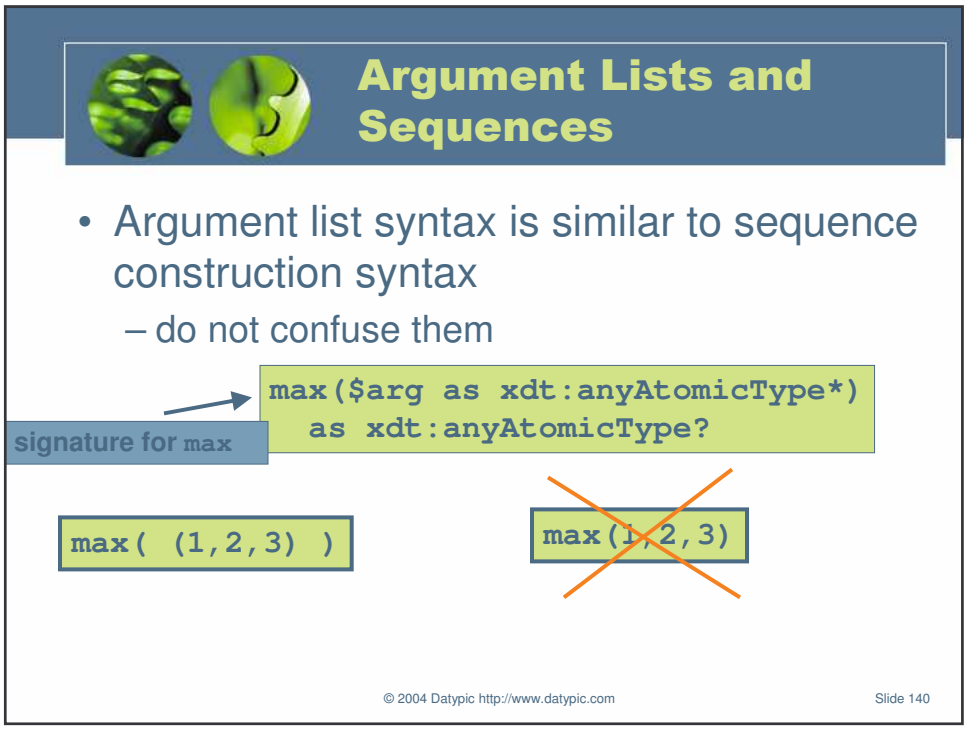

Function Signatures
• Every function has one or more signatures
– defines name, parameters and return type
substring($sourceString as xs:string?,
$startingLoc as xs:double)
as xs:string
function name
parameter name
parameter type
return type
© 2004 Datypic http://www.datypic.com
Slide 135

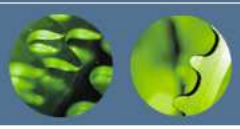

Multiple Function Signatures
• Some functions have more than one signature
– each signature must specify a different number of parameters
substring($sourceString as xs:string?,
$startingLoc as xs:double)
as xs:string
substring($sourceString as xs:string?,
$startingLoc as xs:double,
$length as xs:double)
as xs:string
second signature has extra parameter
© 2004 Datypic http://www.datypic.com
Slide 136


Sequence Types
• Parameter types and return types are expressed as *sequence types*
• Most common are simple type names
– e.g. xs:string, xs:integer, xs:double
– argument must be an atomic value of that type
• Other sequence types:
– item(), node(), element(), attribute()
© 2004 Datypic http://www.datypic.com
Slide 137


Sequence Types and Occurrence Indicators
• Occurrence indicators indicate how many items are allowed:
– ? for zero or one items
– * for zero, one or many items
– + for one or many items
– *no indicator* for one and only one item
count($arg as item()*) as xs:integer
argument can be zero, one or more items (atomic values or nodes)
return type is always a single integer
© 2004 Datypic http://www.datypic.com
Slide 138

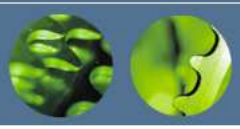

Argument Lists
• Must match the parameters in one of the function signatures in number and type
substring($prodName, 1, 5)
substring($prodName, 1)
substring($prodName, 1, ())
substring($prodName, "1")
type must match; string is not valid
passing the empty sequence is not the same as omitting the argument
© 2004 Datypic http://www.datypic.com
Slide 139


Argument Lists and Sequences
• Argument list syntax is similar to sequence construction syntax
– do not confuse them
max($arg as xdt:anyAtomicType*) as xdt:anyAtomicType?
signature for max
max((1,2,3))
max(1,2,3)
© 2004 Datypic http://www.datypic.com
Slide 140



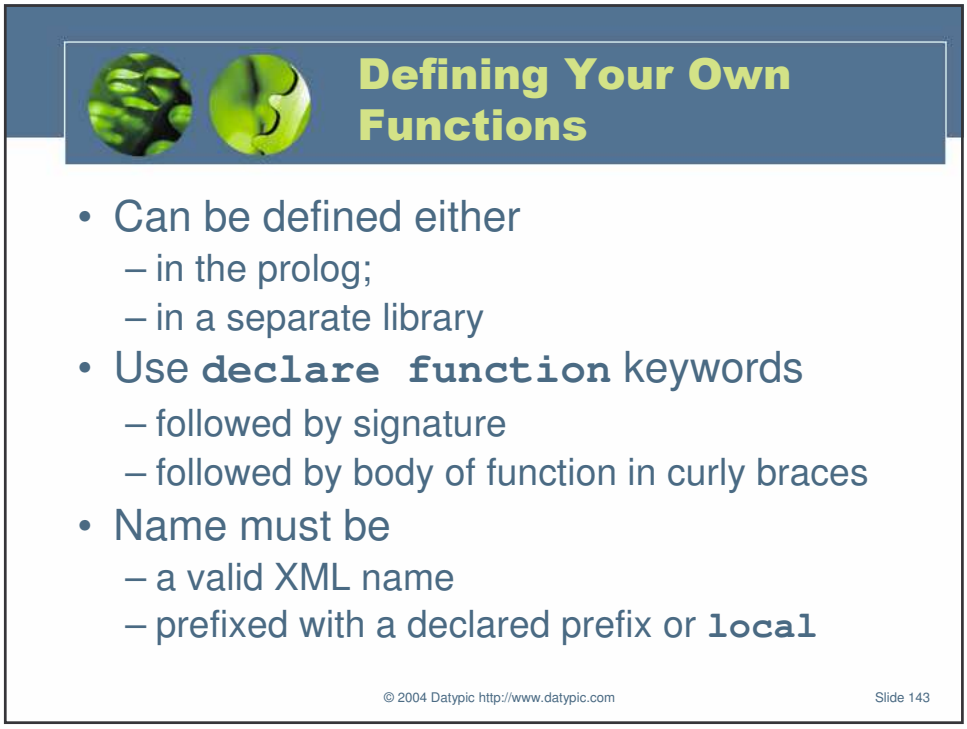
Function Conversion Rules

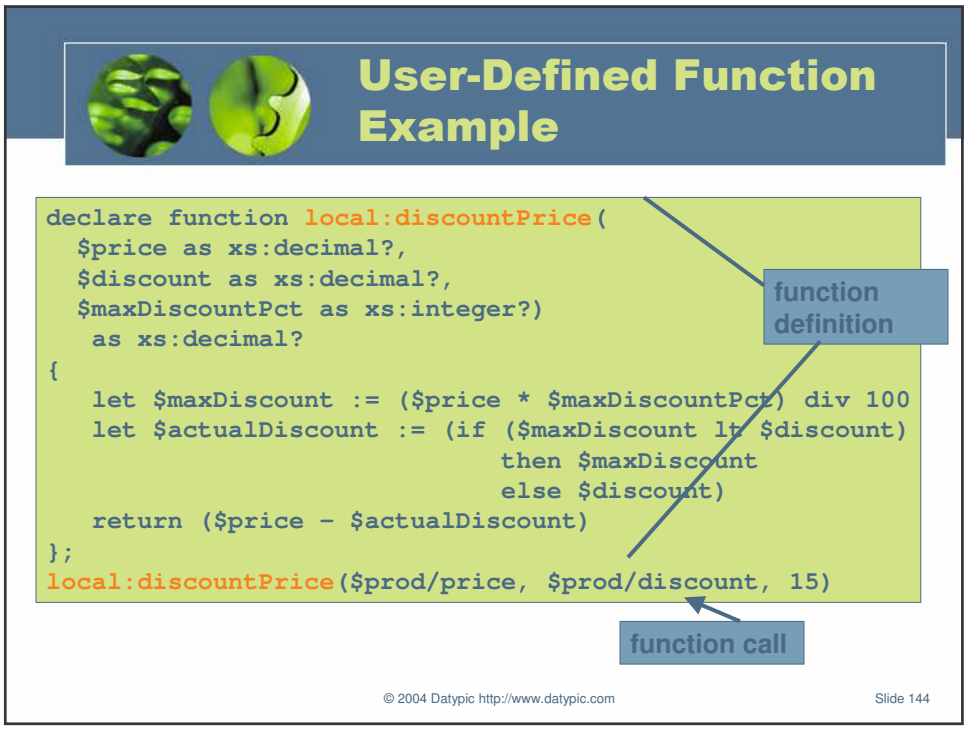
- Used when the type of an argument doesn't match the type in the signature
- Rules
  - Step 1: values are *atomized*
    - if it is expecting an atomic value and it receives a node
    - atomic value is automatically extracted from the node
  - Step 2: untyped values are cast
    - if it is expecting an integer and it receives an untyped value
  - Step 3: numeric values are promoted
    - if it is expecting a decimal and it receives an integer

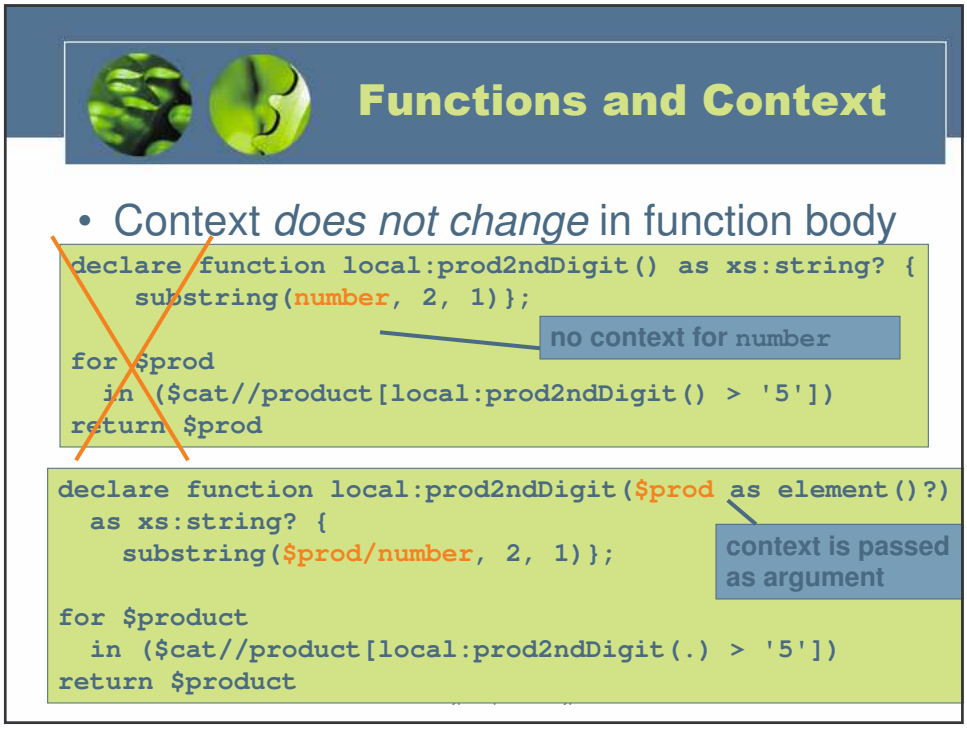


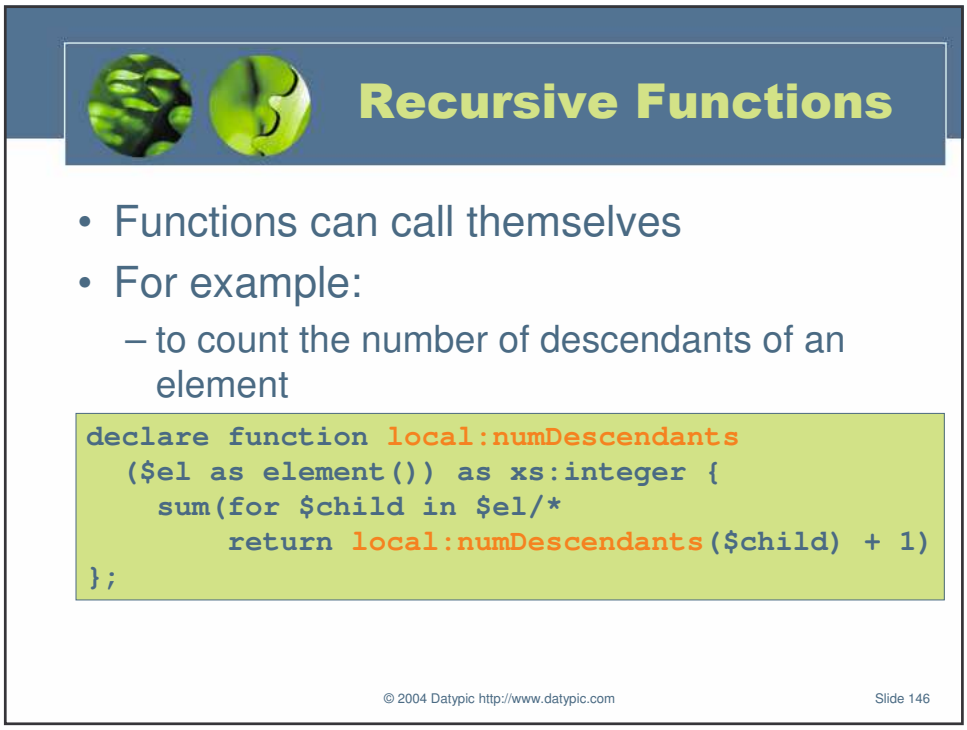
Function Conversion Rules Example

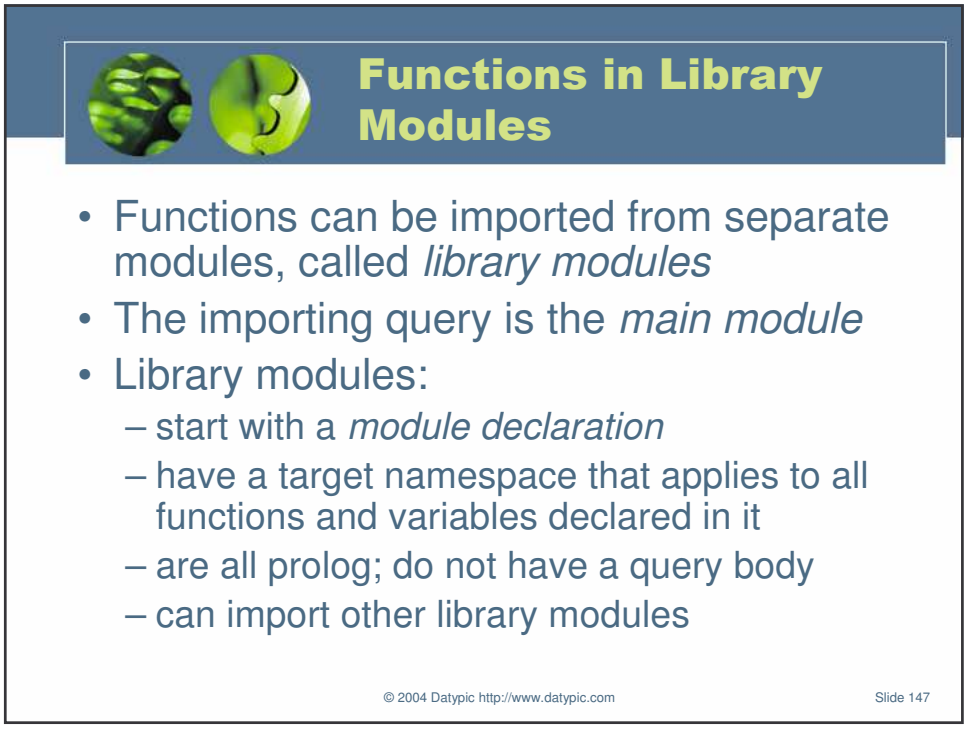
- If a function expects an argument of type **`xs:decimal?`** it accepts all of the following:
  - An atomic value of type **`xs:decimal`**
  - The empty sequence, because the occurrence indicator (?) allows for it
  - An atomic value of type **`xs:integer`**, because **`xs:integer`** is derived from **`xs:decimal`**
  - An untyped atomic value, whose value is 12.5, because it is cast to **`xs:decimal`** (Step 2)
  - An element node of type **`xs:decimal`**, because its value is extracted (Step 1)
  - An untyped element node whose content is 12.5, because its value is extracted (Step 1) and cast to **`xs:decimal`** (Step 2)
- It does not accept
  - an atomic value of type **`xs:string`**

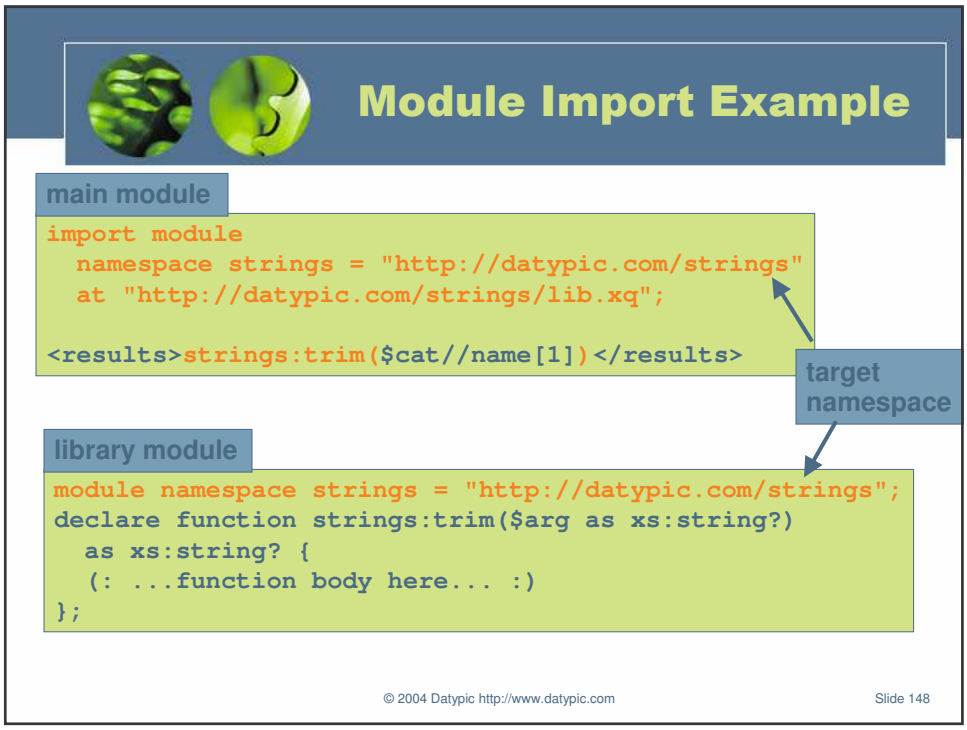

Defining Your Own Functions
• Can be defined either
– in the prolog;
– in a separate library
• Use declare function keywords
– followed by signature
– followed by body of function in curly braces
• Name must be
– a valid XML name
– prefixed with a declared prefix or local
© 2004 Datypic http://www.datypic.com
Slide 143

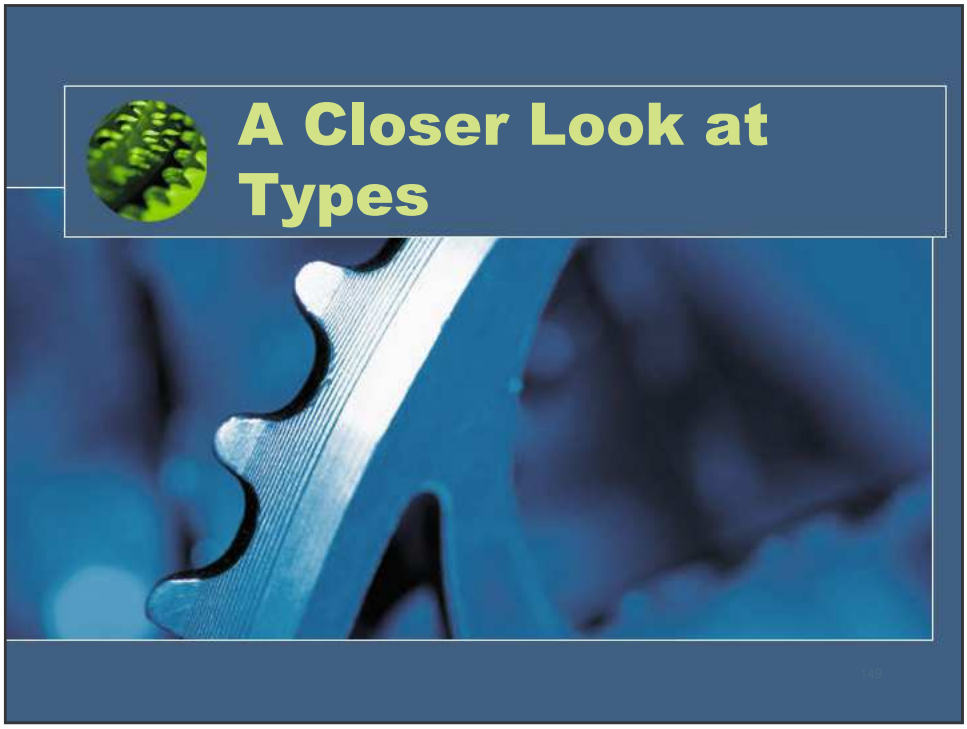

User-Defined Function Example
declare function local:discountPrice(
$price as xs:decimal?,
$discount as xs:decimal?,
$maxDiscountPct as xs:integer?)
as xs:decimal?
{
let $maxDiscount := ($price * $maxDiscountPct) div 100
let $actualDiscount := (if ($maxDiscount lt $discount)
then $maxDiscount
else $discount)
return ($price - $actualDiscount)
};
local:discountPrice($prod/price, $prod/discount, 15)
function definition
function call
© 2004 Datypic http://www.datypic.com
Slide 144


Functions and Context
• Context *does not change* in function body
declare function local:prod2ndDigit() as xs:string? {
substring(number, 2, 1)};
for $prod
in ($cat//product[local:prod2ndDigit() > '5'])
return $prod
no context for number
declare function local:prod2ndDigit($prod as element()?)
as xs:string? {
substring($prod/number, 2, 1)};
for $product
in ($cat//product[local:prod2ndDigit(.) > '5'])
return $product
context is passed as argument


Recursive Functions
• Functions can call themselves
• For example:
– to count the number of descendants of an element
declare function local:numDescendants
($el as element()) as xs:integer {
sum(for $child in $el/*
return local:numDescendants($child) + 1)
};
© 2004 Datypic http://www.datypic.com
Slide 146


Functions in Library Modules
• Functions can be imported from separate modules, called *library modules*
• The importing query is the *main module*
• Library modules:
– start with a *module declaration*
– have a target namespace that applies to all functions and variables declared in it
– are all prolog; do not have a query body
– can import other library modules
© 2004 Datypic http://www.datypic.com
Slide 147

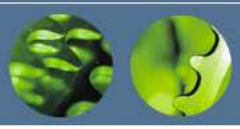

Module Import Example
main module
import module
namespace strings = "http://datypic.com/strings"
at "http://datypic.com/strings/lib.xq";
<results>strings:trim($cat//name[1])</results>
target namespace
library module
module namespace strings = "http://datypic.com/strings";
declare function strings:trim($arg as xs:string?)
as xs:string? {
(: ...function body here... :)
};
© 2004 Datypic http://www.datypic.com
Slide 148



The XQuery Type System

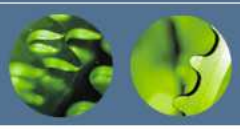
- Two kinds of types:
  - built-in types
    - simple types mostly based on XML Schema
  - imported types
    - simple and complex types defined by users in XML schemas

 Slide 150



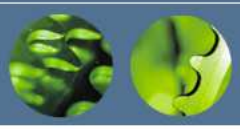
44 Types Built Into XML Schema

- String types
  - e.g. `xs:string`, `xs:token`, `xs:language`
- Numeric types
  - e.g. `xs:integer`, `xs:decimal`, `xs:float`
- Date and time types
  - e.g. `xs:date`, `xs:time`, `xs:duration`
- XML 1.0 types
  - e.g. `xs:NMTOKEN`, `xs:ID`, `xs:ENTITY`
- Others
  - e.g. `xs:boolean`, `xs:anyURI`, `xs:QName`



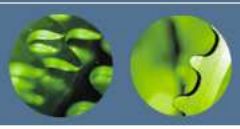
4 Types Added by XQuery/XPath

- Ordered duration types
  - `xdt:dayTimeDuration`
  - `xdt:yearMonthDuration`
- Untyped data (for XML data with no schema)
  - `xdt:untyped`
  - `xdt:untypedAtomic`
- Generic atomic type (for function signatures)
  - `xdt:anyAtomicType`



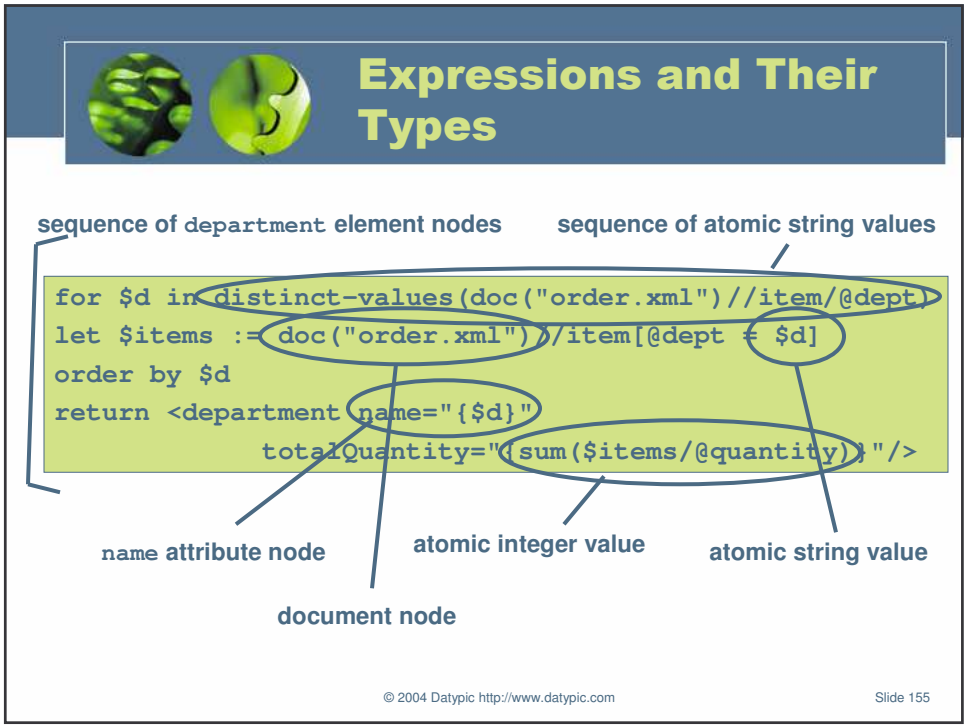
Constructors and Casting

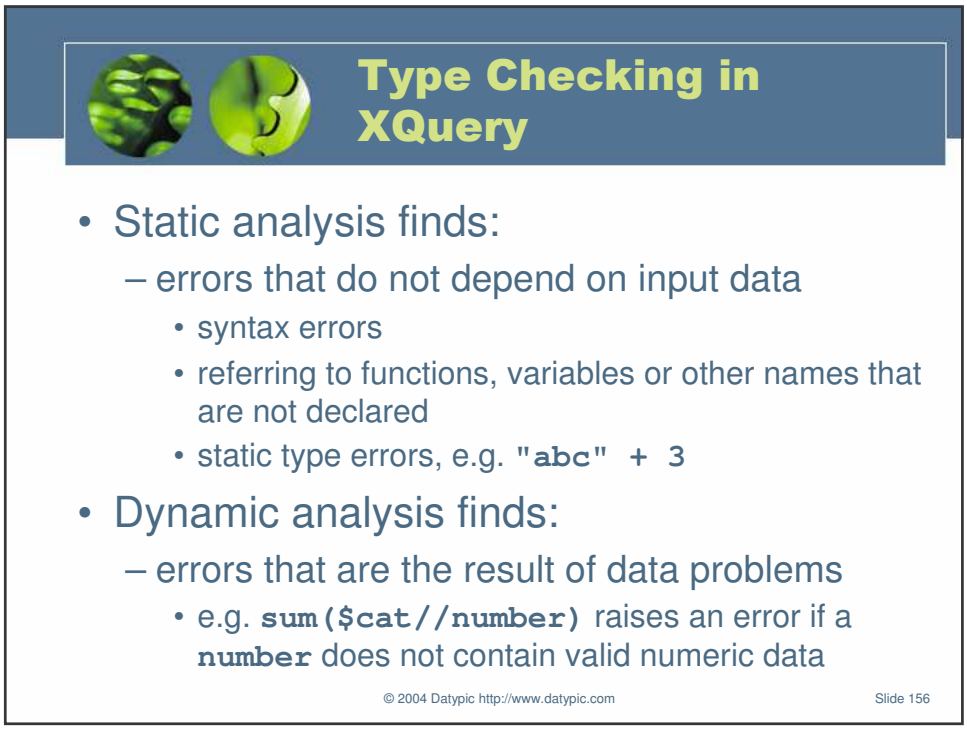
- Every atomic type has a corresponding *constructor* function
  - allows you to create values of that type
- For example, `xs:date("2004-12-15")`
  - creates an atomic value of type `xs:date`
- Argument can be an `xs:string`, and in some cases other types
  - for example, `xs:float(12)` creates an `xs:float` value from an `xs:integer` value

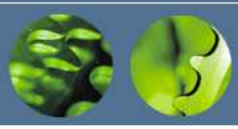


XQuery is Strongly Typed

- Every expression evaluates to zero, one or more nodes or atomic values
- Every node and atomic value has a type
  - Elements & attributes have *type annotations*
  - Atomic values have *types*


Expressions and Their Types
sequence of department element nodes
sequence of atomic string values
for $d in distinct-values(doc("order.xml")//item/@dept)
let $items := doc("order.xml")/item[@dept = $d]
order by $d
return <department name="{$d}"
totalQuantity="{sum($items/@quantity)}"/>
name attribute node
atomic integer value
atomic string value
document node
© 2004 Datypic http://www.datypic.com
Slide 155

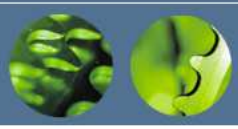

Type Checking in XQuery
• Static analysis finds:
– errors that do not depend on input data
• syntax errors
• referring to functions, variables or other names that are not declared
• static type errors, e.g. "abc" + 3
• Dynamic analysis finds:
– errors that are the result of data problems
• e.g. sum($cat//number) raises an error if a number does not contain valid numeric data
© 2004 Datypic http://www.datypic.com
Slide 156



Strong Typing: Pros and Cons

- Pro
  - easy identification of static errors
    - saves time debugging and testing
    - identifies errors that even testing may not ever uncover
- Con
  - adds complexity because explicit casting is required in some cases

 Slide 157



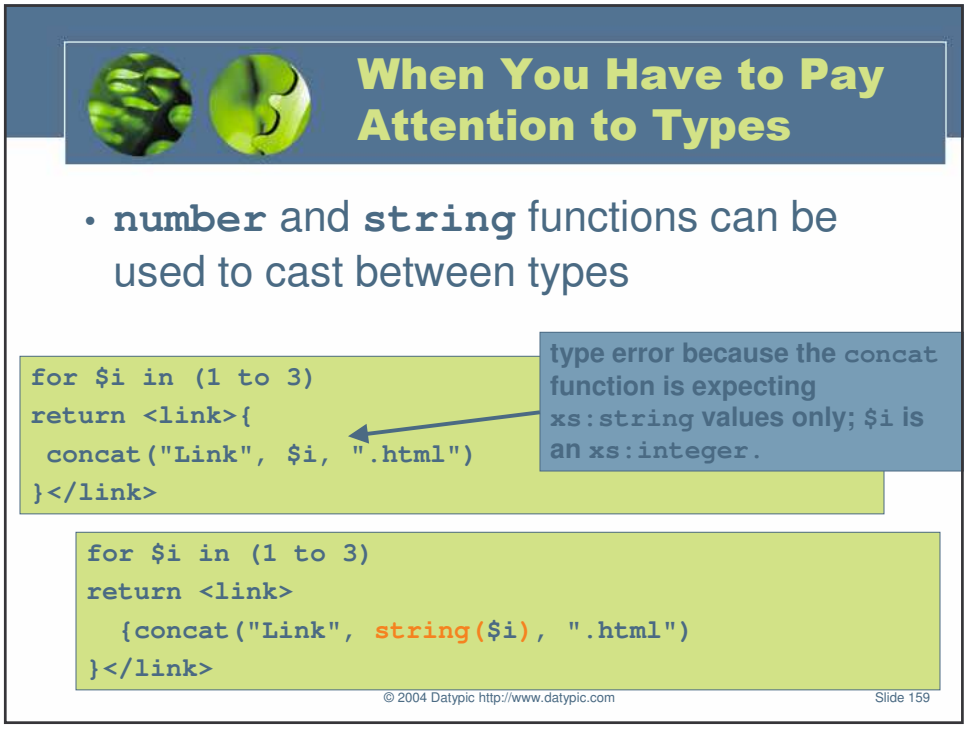
Do I Have to Pay Attention to Types?

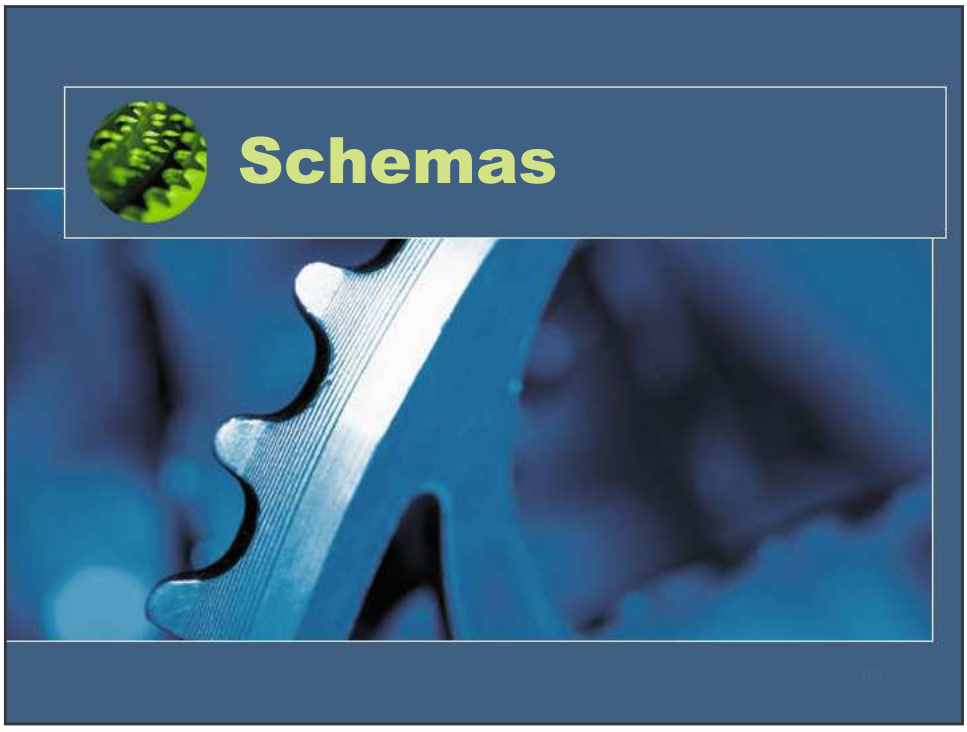
- Usually, no, not if you don't want to.
  - without a schema, your XML data is "untyped"
    - it has one of the "untyped" types:
      - `xdt:untyped` for elements,
      - `xdt:untypedAtomic` for attributes and atomic values
  - in most expressions, untyped values are cast to the expected types

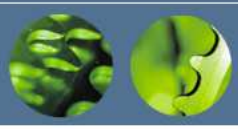
```
product/number + 23
sum(product/number)
```

if `number` is untyped, it is cast to a numeric type automatically

 Slide 158

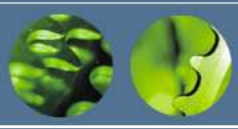

When You Have to Pay Attention to Types
• number and string functions can be used to cast between types
for $i in (1 to 3)
return <link>{
concat("Link", $i, ".html")
}</link>
type error because the concat function is expecting xs:string values only; $i is an xs:integer.
for $i in (1 to 3)
return <link>
{concat("Link", string($i), ".html")
}</link>
© 2004 Datypic http://www.datypic.com
Slide 159


Schemas



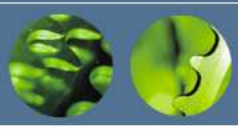
What's a Schema?

- Describes the structure and data content of an XML document
- Can be used to validate XML documents
  - input documents or result documents
- Can be used to pass information to the query about types



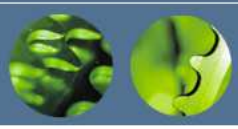
Schema Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://datypic.com/prod.xsd"
   elementFormDefault="qualified">
  <xs:element name="product" type="ProductType"/>
  <xs:complexType name="ProductType">
    <xs:sequence>
      <xs:element name="number" type="xs:integer"/>
      <xs:element name="name" type="NameType"/>
    </xs:sequence>
    <xs:attribute name="dept" type="xs:string"/>
  </xs:complexType>
  <xs:complex name="NameType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="language" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:simpleType></xs:schema>
```



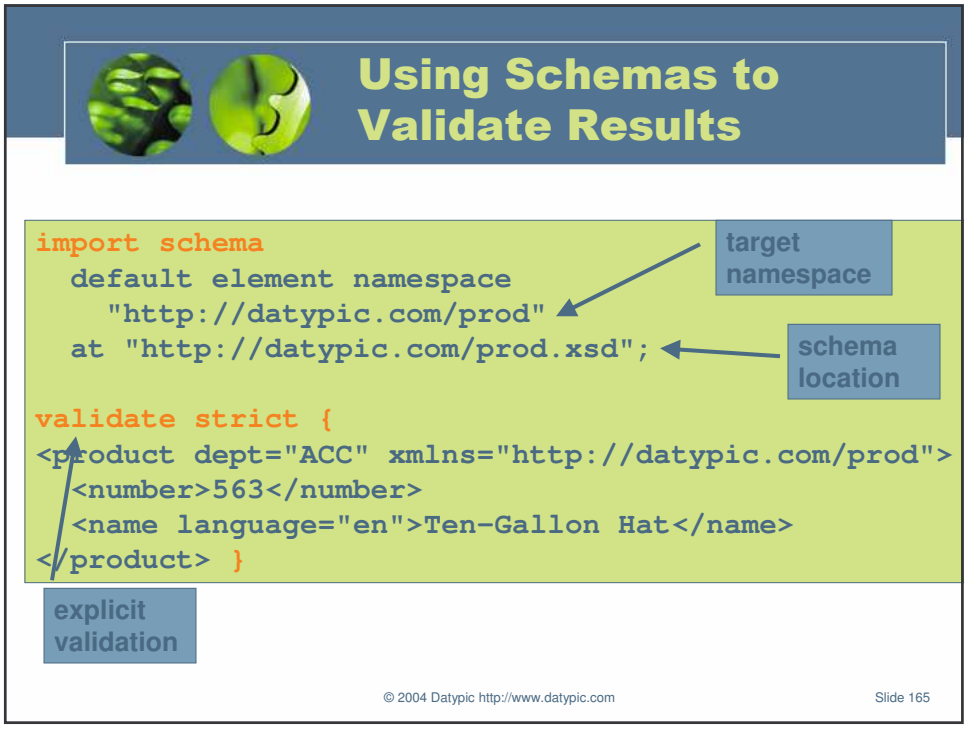
Benefits of Using Schemas

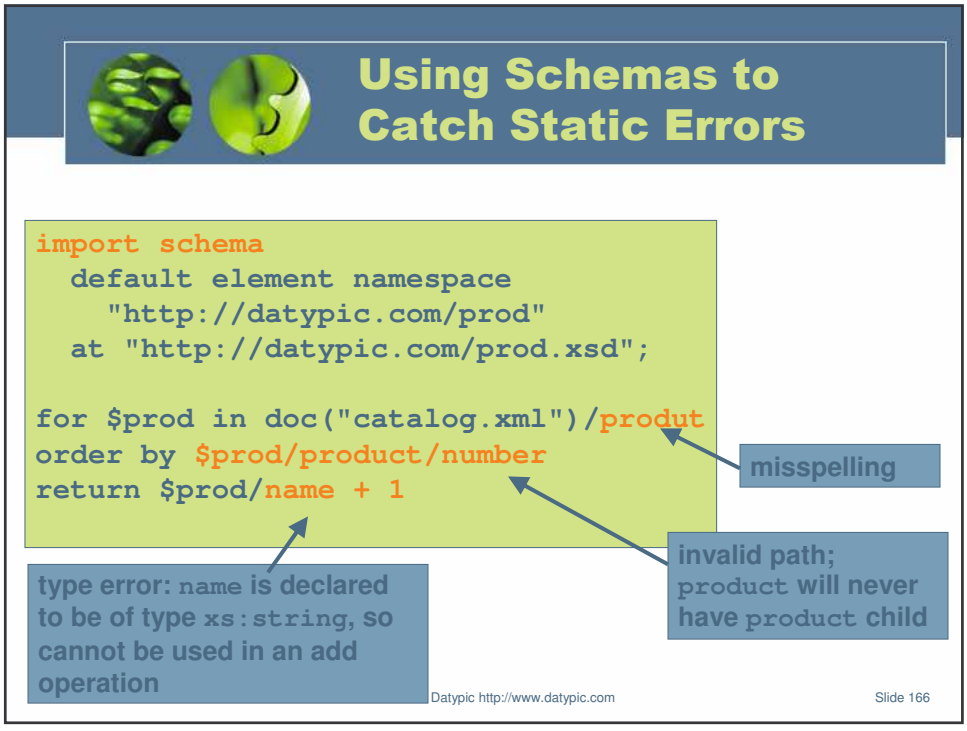
- Predictability of input documents
  - no need to use if-thens to check input data before processing
- Better identification of static errors
  - allows discovery of errors in the query that were not otherwise apparent
- Validity of query results
  - provides guarantee that a query will produce results that always conform to, for example, XHTML
- Special processing based on type
  - you could write an expression that processes the address element differently depending on whether it is of type USAddressType, UKAddressType, etc.



Schemas and XQuery

- Use of schemas is *optional*
- Some processors may:
  - validate documents when they are accessed e.g. using the `doc` function
    - or even outside the scope of the query
  - allow you to import schema documents
  - allow you to explicitly validate input or result documents using these imported definitions

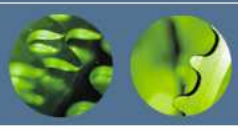

Using Schemas to Validate Results
import schema
default element namespace
"http://datypic.com/prod"
at "http://datypic.com/prod.xsd";
validate strict {
<product dept="ACC" xmlns="http://datypic.com/prod">
<number>563</number>
<name language="en">Ten-Gallon Hat</name>
</product> }
target namespace
schema location
explicit validation
© 2004 Datypic http://www.datypic.com
Slide 165


Using Schemas to Catch Static Errors
import schema
default element namespace
"http://datypic.com/prod"
at "http://datypic.com/prod.xsd";
for $prod in doc("catalog.xml")/produt
order by $prod/product/number
return $prod/name + 1
misspelling
invalid path; product will never have product child
type error: name is declared to be of type xs:string, so cannot be used in an add operation
Datypic http://www.datypic.com
Slide 166

W3C Recommendations

- Many documents; varying degrees of readability
- The most useful are:
  - XQuery 1.0: An XML Query Language
    - the basic syntax of XQuery
  - XQuery 1.0 and XPath 2.0 Functions and Operators
    - the built-in functions
  - XML Query Use Cases
    - good examples of queries for specific uses
- See `http://www.w3.org/XML/Query#specs` for links to all the recommendations

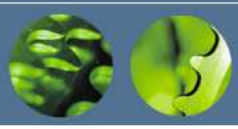
 Slide 168



Mailing Lists

- talk@xquery.com
  - practical discussion about XQuery
- www-ql@w3.org
  - for general discussion on query languages
- public-qt-comments@w3.org
  - for comments on the W3C recommendations only

 Slide 169



Books

- Kay, Michael. *XPath 2.0 Programmer's Reference*. Wrox, 2004.
  - a great reference for where XQuery overlaps with XPath
- Katz, Howard et. al. *XQuery from the Experts.* Addison-Wesley, 2003.
  - very interesting for implementers, or those who want to understand the design of the language on a deeper level
- Walmsley, Priscilla, *Definitive XQuery.* Prentice Hall, 2005.
  - intended as both a tutorial and a reference; XQuery made easy!

 Slide 170


Thank You