A7B36XML, AD7B36XML

XML Technologies

Lecture 4

XPath, SQL/XML

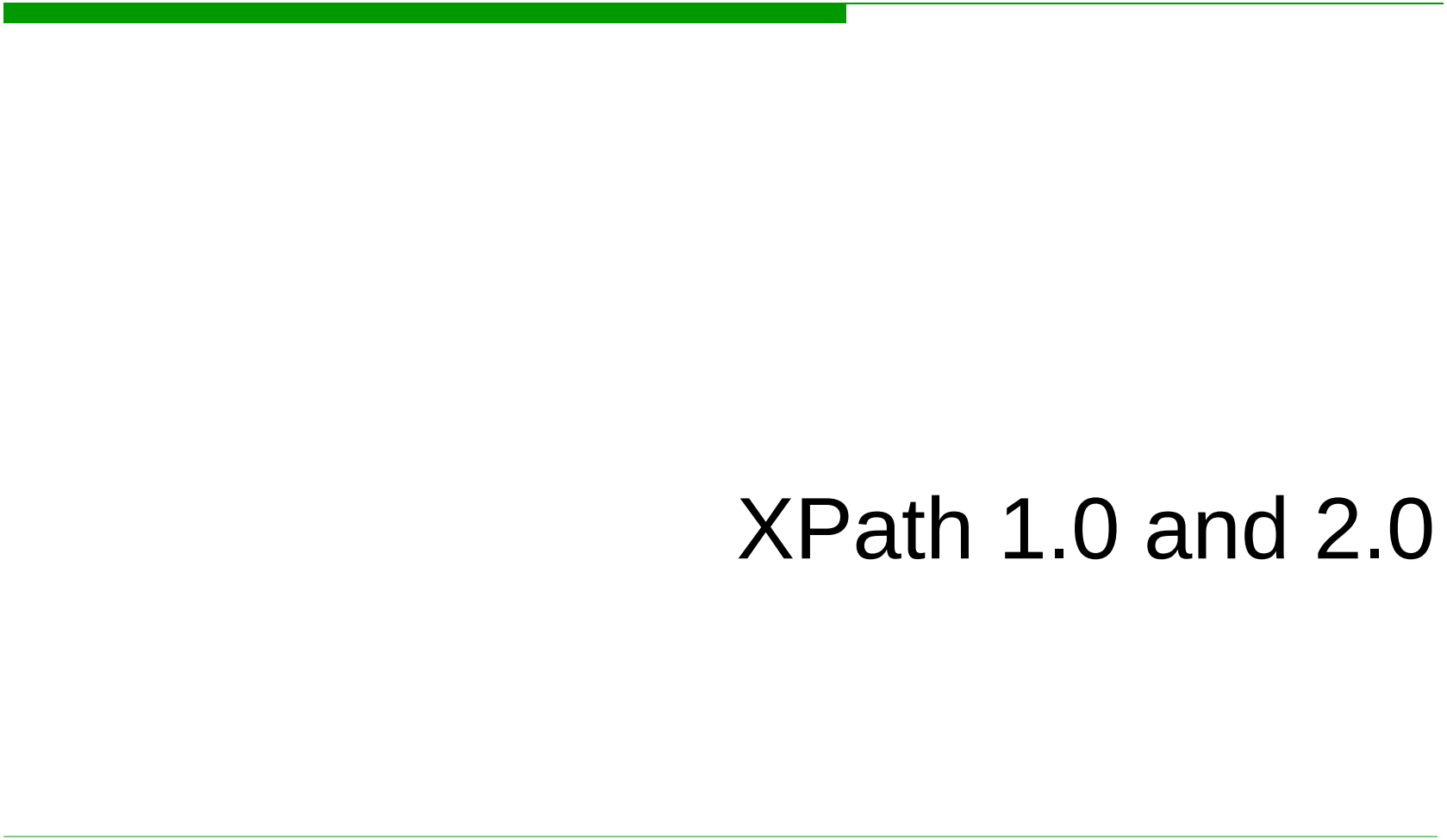
24. 3. 2017

Author: **Irena Holubová**
Lecturer: **Martin Svoboda**

http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-2-A7B36XML/

Lecture Outline

- **XPath**
  - Advanced constructs
- **SQL/XML**


XPath 1.0 and 2.0

XPath 1.0 – Brief Tutorial

- Path consists of steps

  /step1/step2/step3/…
  step1/step2/step3/…

- Step:

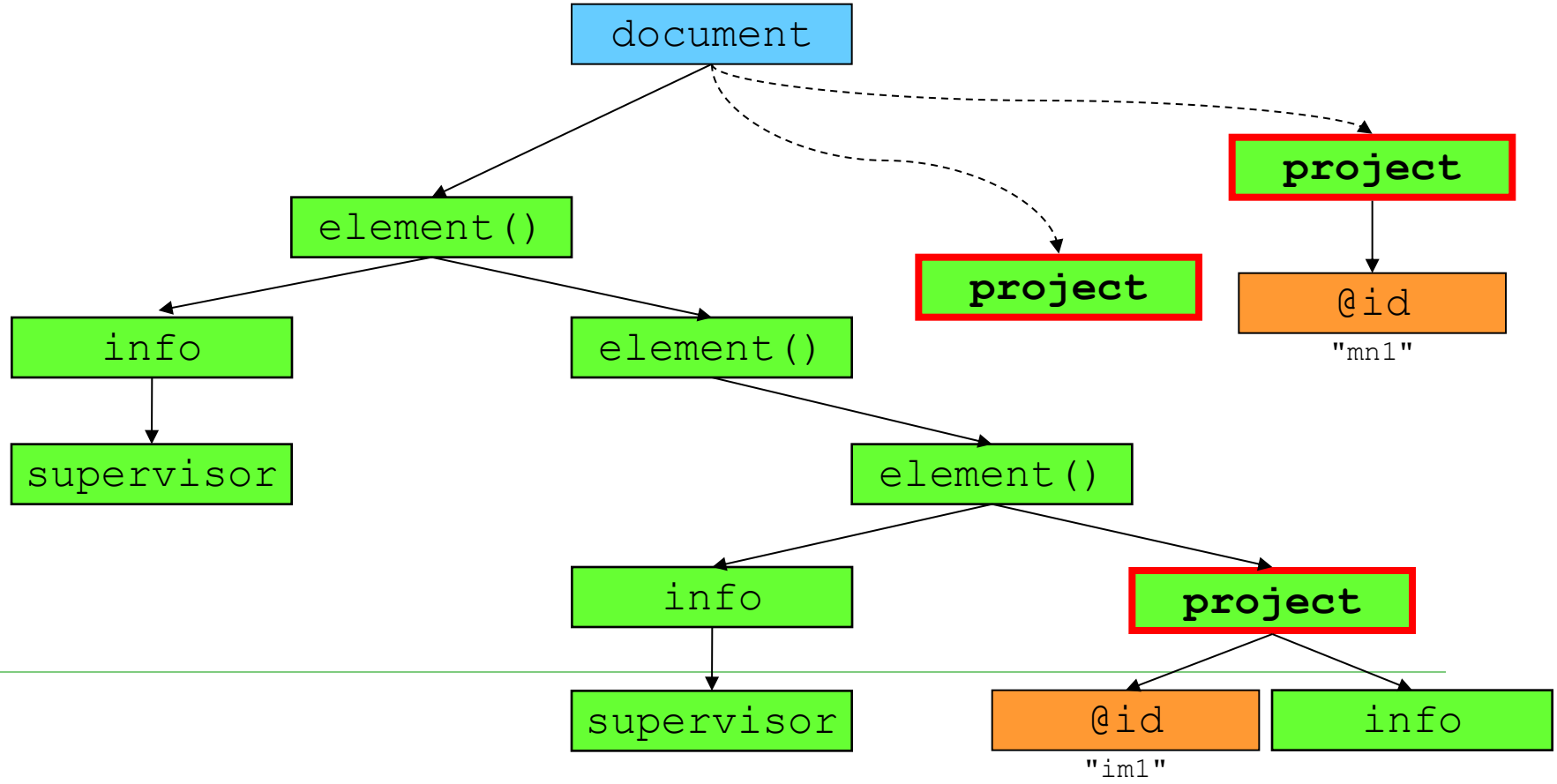
  axis::node-test predicate1 ... predicateN

  - Axis
    - Denotes the "direction" of the step
  - Node test
    - Denotes the type/name of nodes selected by the axis
  - Predicate
    - Logical condition further specifying requirements on the selected data

- Abbreviations simplify the expressions

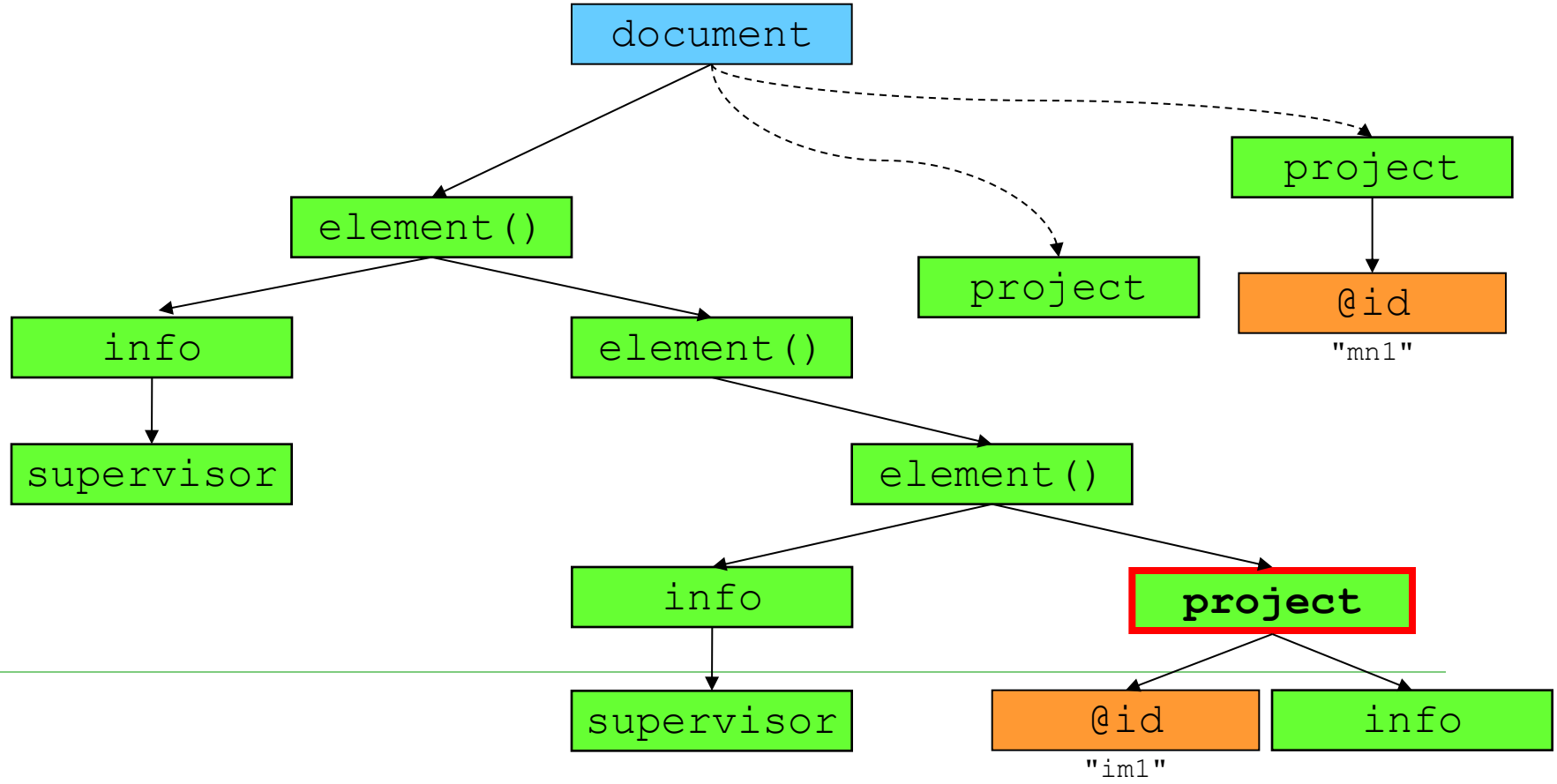
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



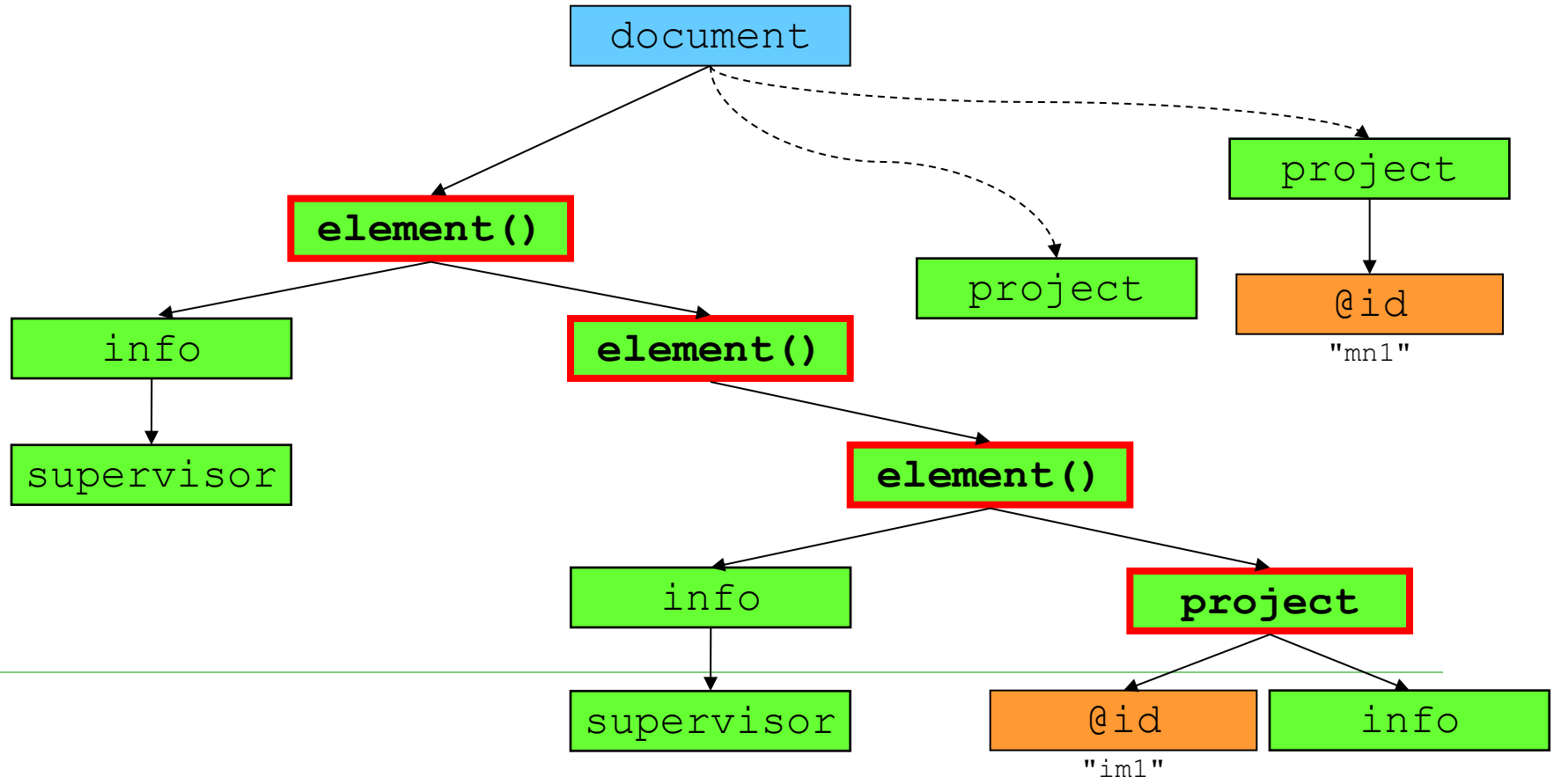
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



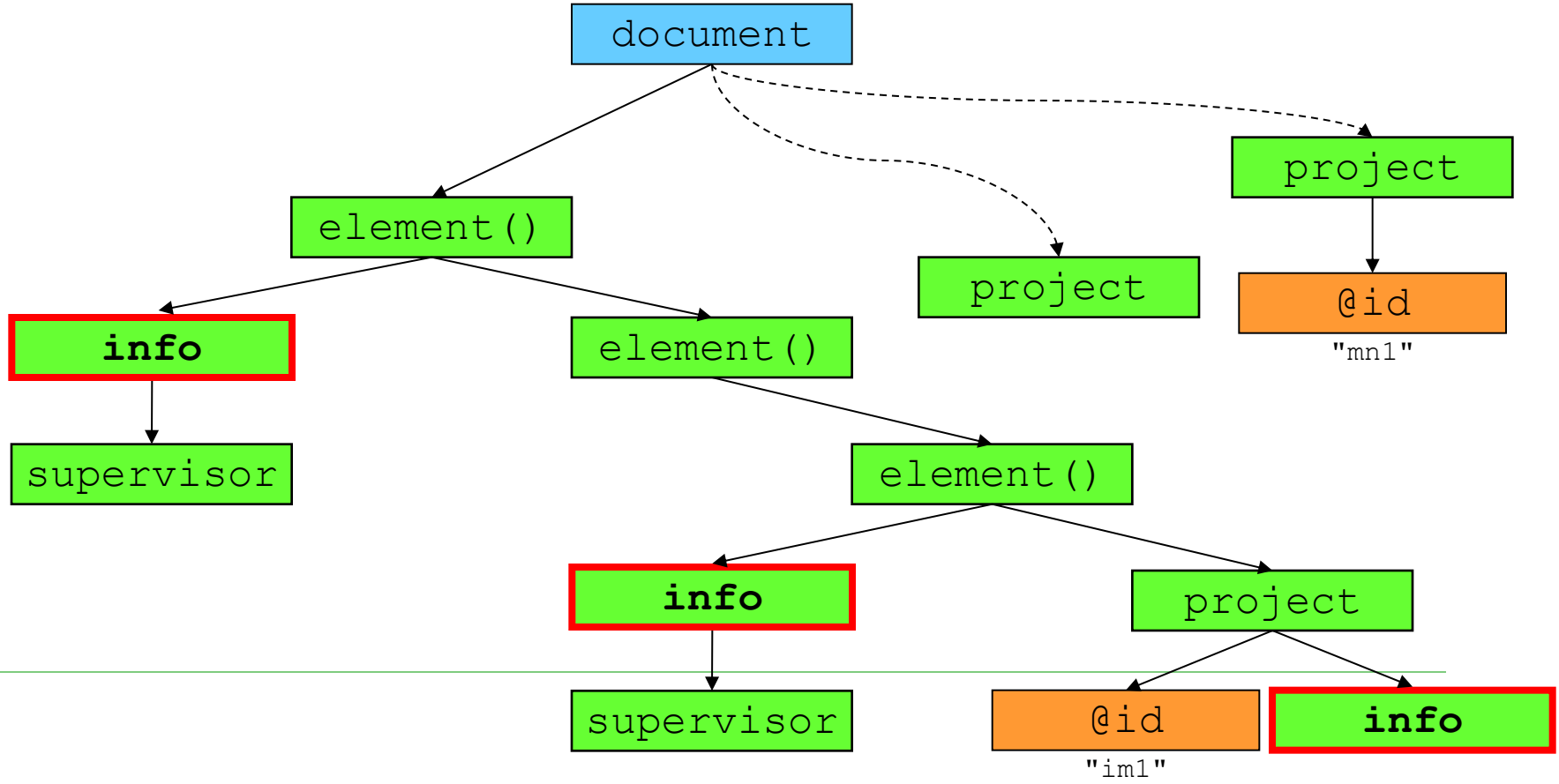
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



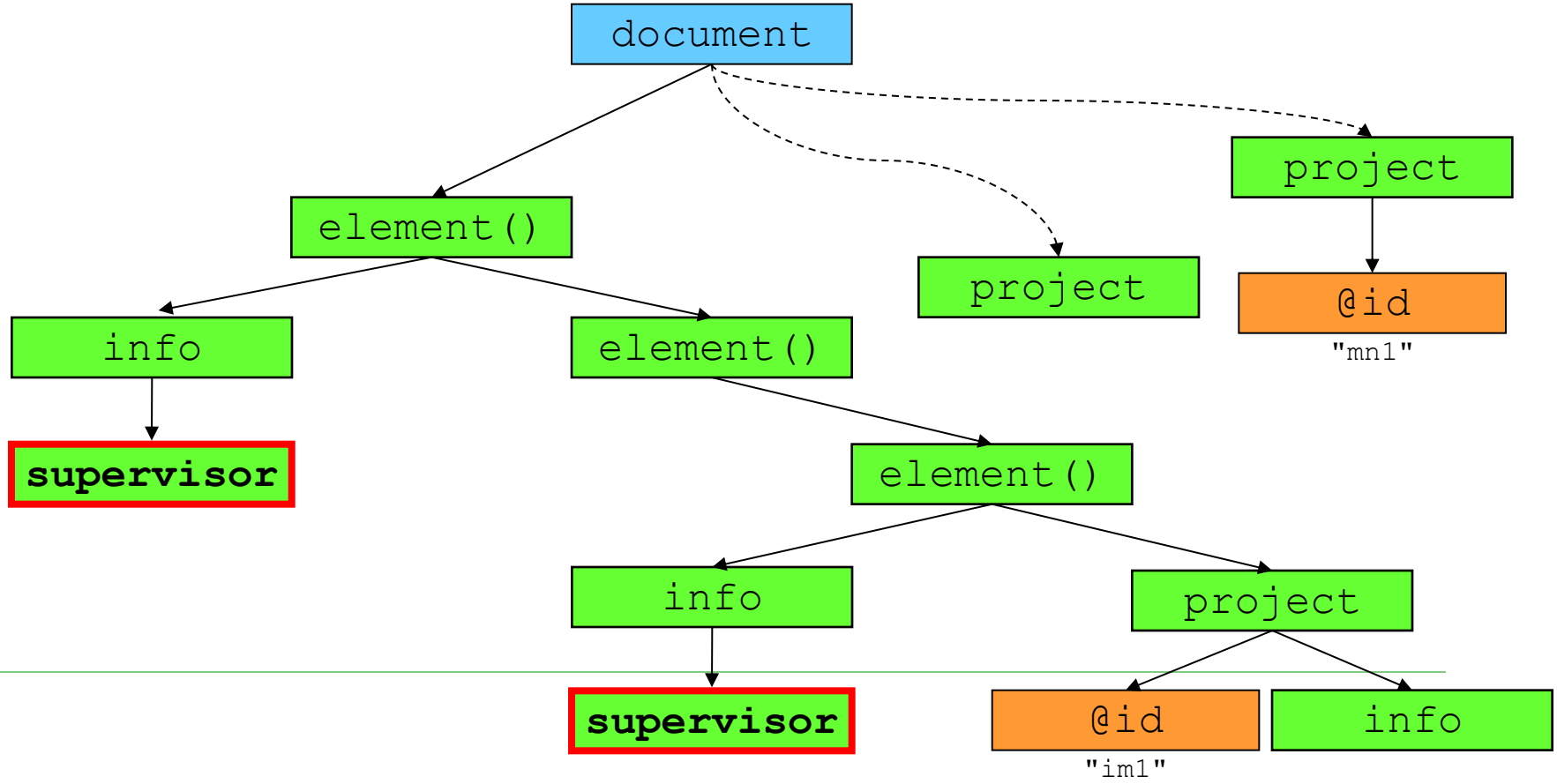
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



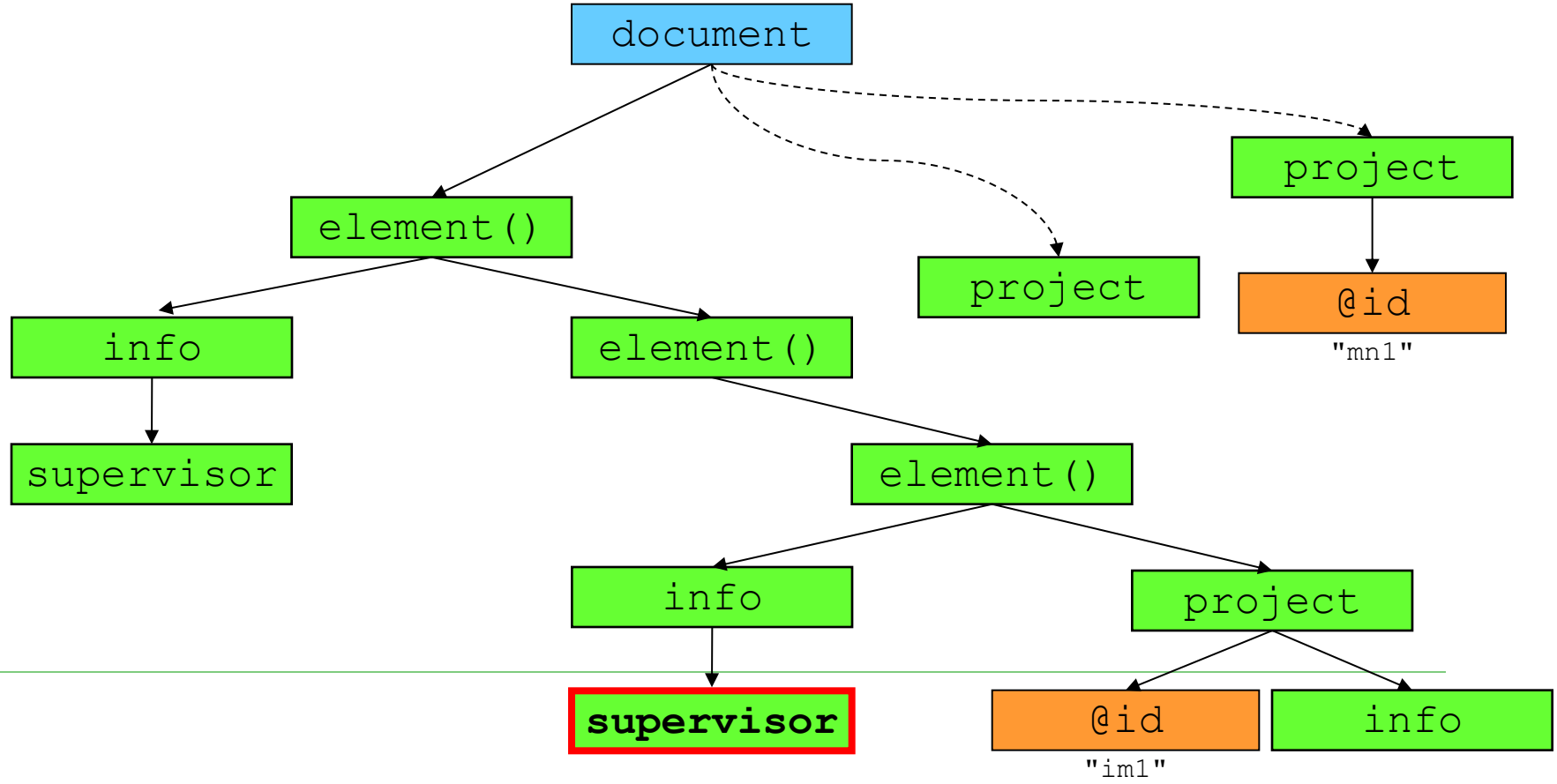
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



XPath 2.0

- Adds a huge number of new functions
  - see http://www.w3.org/TR/xpath-functions/
  - Prefixed with fn:
    - Namespace http://www.w3.org/2005/xpath-functions
- Works with ordered collections
  - Adds new constructs
    - Iterations of sequences (for loop), merging of sequences (union, intersect, except), conditions (if-then-else), quantifiers (some/every)
- Relation to XML Schema
  - Nodes are assigned with data types in the sense of XML Schema language
- Backward compatibility with XPath 1.0
  - Expressions from 1.0 return the same value
  - Few exceptions

http://www.w3.org/TR/xpath20/

XPath Data Types

- XPath 1.0
  - node-set, Boolean, number, string
- XPath 2.0
  - sequence, XML Schema data types

XPath 2.0 – Data Model

- sequence is an ordered collection of items
  - The result of an XPath 2.0 expression is a sequence
- item is either an atomic value or a node
  - atomic value = a value of any simple data type of XML Schema
  - node = an instance of any type of node
    - attribute, element, text, …

XPath 2.0 – Nodes

- Node has
  - Identity
  - Data type
    - XML Schema simple/complex data type
  - Typed value
    - Value according to a data type
      - Returned by fn:data()
  - String value
    - Type value converted to string (xs:string)
      - Returned by fn:string()

XPath 2.0 – Sequence

- Constructor ()
  - (1, 2, 3, 4)
- Constructor to
  - (1, 5 to 8) = (1, 5, 6, 7, 8)
- Constructor can contain XPath expressions
  - (//book, //cd)
- Constructor can be used as a step in an XPath expression
  - (1 to 100)[. mod 5 = 0]
  - //item/(price,value)
  - orders[fn:position() = (5 to 9)]

XPath 2.0 – Sequence

- Everything is a sequence
  - 1 = (1)
- Sequences are shallow = do not contain subsequences
  - (1, (2, 3), 4) = (1, 2, 3, 4)
- Sequences can contain duplicities
  - (1, 2, 1 to 2) = (1, 2, 1, 2)
- Sequences can contain atomic values and nodes together
  - (1, 2, //book)

XPath 2.0 – Iteration

- Construct for for iteration of sequences within expressions

```
for $i in (1,2,3)
return $i
```

- (1,2,3)

```
for $i in (10,20),
    $j in (1,2)
return ($i+$j)
```

- (11,12,21,22)

variables

```
for $varname1 in expression1,
    ...,
    $varnameN in expressionN
return expression
```

XPath 2.0 – Iteration

```
fn:sum( for
                $item in //item
        return
                $item/amount * $item/price )
```

XPath 2.0 – Quantifiers

```
some/every $variable in expression satisfies test_expression
```

- If the quantifier is some (every), the expression is true if at least one (every) evaluation of the test expression has the value true; otherwise the expression is false

```
every $part in /parts/part satisfies $part/@discounted
```

```
some $x in (1, 2, 3), $y in (2, 3, 4)
     satisfies $x + $y = 4
```

XPath 2.0 – Merging

- Union of sequences
  - union or | (| is already in XPath 1.0)
- Intersection of sequences
  - intersect
- Exception of sequences
  - except

```
expression1 union expression2

expression1 intersect expression2

expression1 except expression2
```

- Only for sequences of nodes
  - If the sequence includes an item which is not - error
- All operators eliminate duplicities
  - Two nodes are duplicate if they have the same identity

XPath 2.0 – Merging

```
for
        $item in /order//item
return
        $item/* except $item/price
```

XPath 2.0 – Merging

```
//item[color="blue"]
intersect
//order[ordernumber>1000]//*

<order number="O233" ordernumber="2911">
  <customer>
   <name>Martin Necasky</name>
   <email>martinnec@gmail.com</email>
  </customer>
  <items>
   <item code="V289348">
    <name>Name of item 289348</name>
    <color>blue</color>
    <count>1</count><price-item>1234</price-item>
   </item>
   ...
  </items>
</order>
```

XPath 2.0 – Comparison

- We already know operators for sets
  - =, !=, …
- New type: comparison of nodes
  - expression1 **is** expression2
    - true, if both the operands evaluate to the same node
  - expression1 **<<** expression2, resp. expression1 **>>** expression2
    - true, if the node on the left precedes/succeeds the node on the right (in the document order)
  - If any of the operands is converted to an empty sequence, the result is an empty sequence
  - If any of the operands is converted to a sequence longer than 1, error

XPath 2.0 – Comparison

- New type: comparison of values
  - lt, gt, le, ge, eq, ne meaning "less than", "greater than", "less or equal", "greater or equal", "equal", "non equal"
  - If any of the operands is converted to an empty sequence, the result is an empty sequence
  - If any of the operands is converted to a sequence longer than 1, error

XPath 2.0 – Comparison

```
//order[
        customer/name = "Martin Necasky"
          and
        . << (
            //order[
                    customer/name = "Martin Necasky"
                      and
                    fn:sum(
                            for $p in .//item
                            return $p/amount * $p/price
                            ) > 100000
                    ]
            )[1]
      ]
```

XPath 2.0 – Conditions

```
if (expression1)
  then (expression2)
  else (expression3)
```

```
for $product in /catalogue//product
return
 if ($product/discount = "yes")
  then $product/discount-price
  else $product/full-price
```

SQL/XML

What is SQL/XML?

- Extension of SQL which enables to work with XML data
  - New built-in data type: XML
  - Publication of relational data in XML
  - XML data querying
- Node: SQL/XML ≠ SQLXML
  - Technology of Microsoft used in SQL Server
  - Not a standard from the SQL family
- Key aspect: XML value
  - Intuitively: XML element or a set of XML elements

Functions for Data Publication

- SQL expressions → XML values
  - XMLELEMENT – creating XML elements
  - XMLATTRIBUTES – creating XML attributes
  - XMLFOREST – creating XML elements for particular tuples
  - XMLCONCAT – from a list of expressions creates a single XML value
  - XMLAGG – XML aggregation

Employees

id	first	surname	dept	start
1001	Brad	Pitt	HR	2000-05-24

XMLELEMENT

- Creates an XML value for:
  - Element name
  - Optional list of namespace declarations
  - Optional list of attributes
  - Optional list of expressions denoting element content

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    E.first || ' ' || E.surname) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp>Brad Pitt</emp>
…	…

XMLELEMENT – Subelements

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname),
    XMLELEMENT (NAME "start_date", E.start)
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp> <name>Brad Pitt</name> <start_date>2000-05-24</start_date> </emp>
…	…

XMLELEMENT – Mixed Content

```
SELECT E.id,
  XMLELEMENT (NAME "emp", 'Employee ',
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname ), ' started on ',
    XMLELEMENT (NAME "start_date", E.start)
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp>Employee <name>Brad Pitt</name> started on <start_date>2000-05-24</start_date></emp>
…	…

Children

id	parent
37	1001

XMLELEMENT – Subqueries

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname ),
    XMLELEMENT (NAME "children",
      (SELECT COUNT (*) FROM Children D WHERE D.parent = E.id ))
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp> <name>Brad Pitt</name> <children>3</children> </emp>
…	…

XMLATTRIBUTES

- Specification of attributes can occur as
  - 3rd argument if namespace declarations are present
  - 2nd argument otherwise

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLATTRIBUTES (E.id AS "empid"),
    E.first || ' ' || E.surname) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp empid="1001">Brad Pitt</emp>
…	…

XMLELEMENT – Namespaces

```
SELECT E.id,
  XMLELEMENT (NAME "IBM:emp",
    XMLNAMESPACES ('http://a.b.c' AS "IBM"),
    XMLATTRIBUTES (E.id AS "empid"),
    E.first || ' ' || E.surname
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<IBM:emp xmlns:IBM="http://a.b.c" empid="1001">Brad Pitt</IBM:emp>
…	…

XMLFOREST

- Constructs a sequence of XML elements for
  - Optional declaration of namespaces
  - List of named expressions as arguments
    - If any of the expressions returns NULL, it is ignored
    - If all the expressions return NULL, the result is XML value NULL
- Each element in the result can be named implicitly or explicitly

XMLFOREST

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLFOREST (
      E.first || ' ' || E.surname AS "name",
      E.start AS "start") ) AS xvalue
FROM Employees E
WHERE ...
```

id	xvalue
1001	<emp> <name>Brad Pitt</name> <start>2000-05-24</start> </emp>
…	…

XMLCONCAT

- Creates an XML value as a concatenation of results of multiple expressions

```
SELECT E.id,
  XMLCONCAT (
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname),
    XMLELEMENT (NAME "start_date", E.start)
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<name>Brad Pitt</name><start_date>2000-05-24<start_date>
…	…

XMLAGG

- XMLAGG is an aggregation function
  - Similar to SUM, AVG from SQL
- The argument for XMLAGG must be an XML expression
- For each row in a group G, we evaluate the expression and the resulting XML values are concatenated so that they form a single XML value for the whole group G
- For sorting we can use clause ORDER BY
- All NULL values are ignored for the concatenation
  - If all the concatenated values are NULL or the group is empty, the result is NULL

XMLAGG

```
SELECT XMLELEMENT (
  NAME "department",
  XMLATTRIBUTES (E.dept AS "name"),
  XMLAGG (
    XMLELEMENT (NAME "emp", E.surname)  ) ) AS xvalue
FROM Employees E
GROUP BY E.dept
```

xvalue
<department name="HR"> <emp>Pitt</emp> <emp>Banderas</emp> </department>
<department name="PR"> ... </department>
…

XMLAGG – Sorting

```
SELECT XMLELEMENT (
  NAME "department",
  XMLATTRIBUTES (E.dept AS "name"),
  XMLAGG (
    XMLELEMENT (NAME "emp", E.surname)
    ORDER BY E.surname ) ) AS xvalue
FROM Employees E
GROUP BY E.dept
```

xvalue
<department name="HR"> <emp>Banderas</emp> <emp>Pitt</emp> </department>
…

XML Data Type and Querying

- XML data type can be used anywhere, where SQL data types (e.g. NUMBER, VARCHAR, …)
  - Type of column, parameter of a function, SQL variable, …
  - Its value is an XML value
- XML Infoset modification: XML value is
  - XML element
  - Set of XML elements

  → Not each XML value is an XML document
- Querying:
  - XMLQUERY – XQuery query, results are XML values
  - XMLTABLE – XQuery query, results are relations
  - XMLEXISTS – testing of conditions

Example: table EmployeesXML

id	EmpXML
1001	<employee> <first>Brad</first> <surname>Pitt</surname> <start>2000-05-24</start> <department>HR</department> </employee>
1006	<employee> <first>Antonio</first> <surname>Banderas</surname> <start>2001-04-23</start> <department>HR</department> </employee>
…	…

XMLQUERY

```
SELECT
  XMLQUERY (
    'for $p in $column/employee return $p/surname',
    PASSING EmpXML AS "column"
    RETURNING CONTENT NULL ON EMPTY ) AS result
FROM EmployeesXML WHERE …
```

result
<surname>Pitt</surname>
<surname>Banderas</surname>
…

XMLTABLE

```
SELECT result.*
  FROM EmployeesXML, XMLTABLE (
    'for $p in $column/employee return $p/surname',
    PASSING EmployeesXML.EmpXML AS "column"
  ) AS result
```

result
<surname>Pitt</surname>
<surname>Banderas</surname>
…

XMLTABLE

```
SELECT result.*
  FROM EmployeesXML, XMLTABLE (
    'for $p in $column/employee return $p',
    PASSING EmployeesXML.EmpXML AS "column"
    COLUMNS name VARCHAR(40) PATH 'first'
                             DEFAULT 'unknown',
            surname VARCHAR(40) PATH 'surname'
  ) AS result
```

Xpath query

Xpath query

Assumption: We do not know the first name of Craig.

name	surname
Brad	Pitt
unknown	Craig

XMLEXISTS

```
SELECT id
FROM EmployeesXML
WHERE
  XMLEXISTS('/employee/start lt "2001-04-23"'
    PASSING BY VALUE EmpXML)
```

id
1001
1006
…

- Its argument is an XQuery expression
  - Returning true / false
- Usage: WHERE clause
- It does not extend the expressive power of the language
  - The same can be done via XMLQUERY or XMLTABLE

Other Constructs

- XMLPARSE – transforms the given SQL string value to XML value
- XMLSERIALIZE – transforms the given XML value to SQL string value
- IS DOCUMENT – tests whether the given XML value has a single root element
- …