COS 326
Database Systems

Lecture 7

Semi-structured data and XML (1)

Chapter 31

16 August 2016

Admin matters

- Prac 4 progress?
- Essay topics & bookings

Week	**Date**	**Day**	**Topic**
5	**16 Aug**	**Tues**	**L7: Semi-structured data & XML databases**
	17 Aug	**Wed**	**L8: Semi-structured data & ORDBs** **(change to study guide schedule)**
	19 Aug	Fri	***Practical 4: XML DB (BaseX)***
6	23 Aug	Tues	**Class Test 1: OODB and ORDB** L9: Big data and NoSQL databases
	24 Aug	Wed	L10: NoSQL databases (MongoDB) **Presentation: Essay topic 1**
	26 Aug	Fri	*Practical 5:* XML data & ORDB (PostgreSQL)

In this lecture

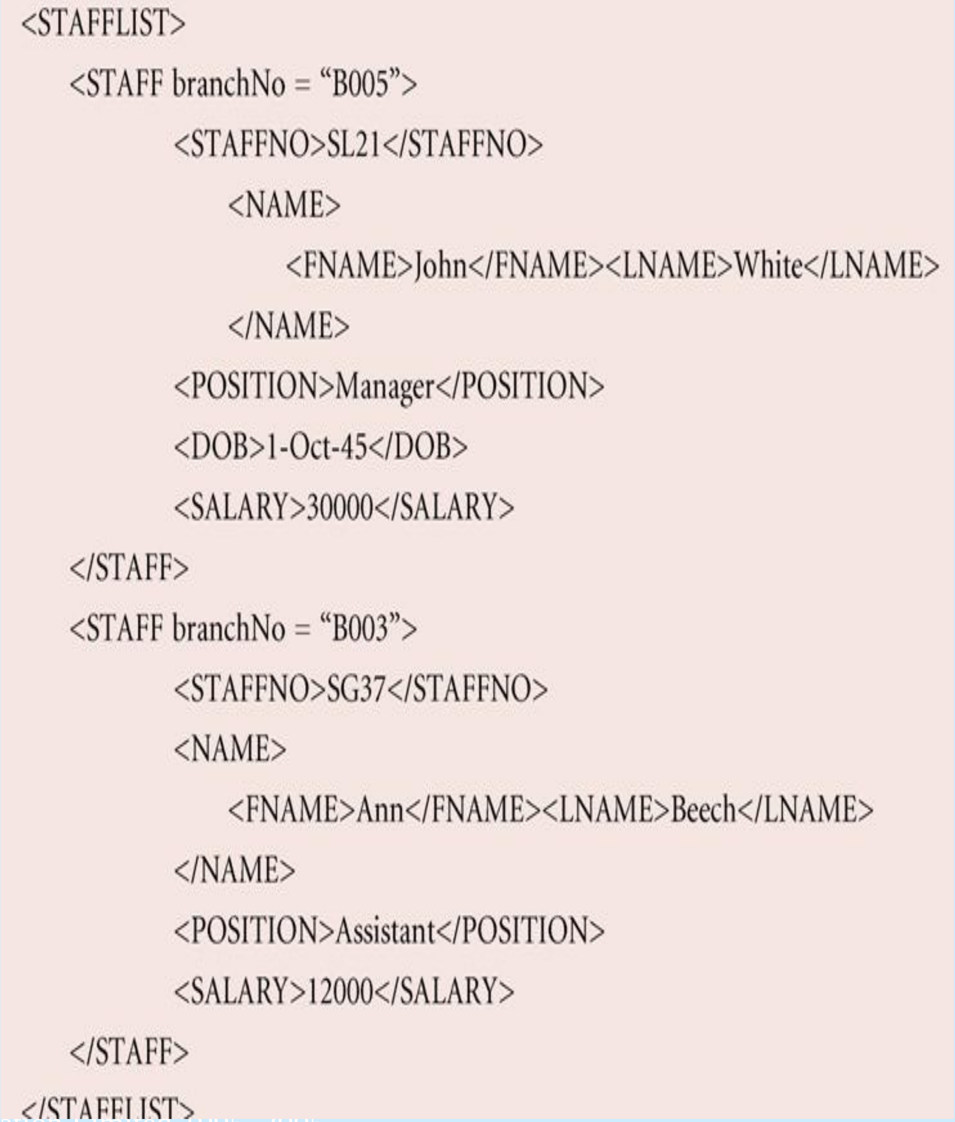
- Revision (COS216 and INF272)
    - HTML and the web
    - XML, XML Schema, XML related technologies
- Reading for the student
    - Section 31.2 Introduction to XML
    - Section 31.3 XML-related technologies
- **To be discussed in this lecture:**
    - XML query languages
        - XQuery
            - **XPath expressions**
            - **FLWOR queries**

Semi-structured data

Semi-structured data

e.g. XML documents

- may have a structure, but may change
- aka schema-less or self-describing
  - **no separate schema**
  - **schema is part of the data**
- RDBs, OODBs, ORDBs
  - **Require pre-defined schema**



```
<STAFFLIST>
    <STAFF branchNo = "B005">
            <STAFFNO>SL21</STAFFNO>
                <NAME>
                    <FNAME>John</FNAME><LNAME>White</LNAME>
                </NAME>
            <POSITION>Manager</POSITION>
            <DOB>1-Oct-45</DOB>
            <SALARY>30000</SALARY>
    </STAFF>
    <STAFF branchNo = "B003">
            <STAFFNO>SG37</STAFFNO>
            <NAME>
                <FNAME>Ann</FNAME><LNAME>Beech</LNAME>
            </NAME>
            <POSITION>Assistant</POSITION>
            <SALARY>12000</SALARY>
    </STAFF>
</STAFFLIST>
```

Importance of semi-structured data

- **Recent importance**
  - treat Web resources as database
    - cannot constrain with a database schema
  - flexible format for data exchange
    - many different databases in organisations
  - becoming the standard for data representation and data exchange   **e.g. on the Web**
- 1998: XML 1.0 by W3C
- Meta-language (language for describing other languages)
    - enables designers to create their **own customized tags**
    - provide **functionality not available with HTML**

XML: what you already know

1. XML file: e.g. *myfile.xml*

2. format of an XML document
   - header, tags, attributes, namespaces, etc

3. Document Object Model (DOM)
   - tree structure for an xml document

4. XML document validation
   - Document Type Definitions (DTDs)
   - Schemas (Microsoft XML schema, W3C schema)

5. Conversion of XML data to formatted text documents
   - Extensible Stylesheet Language (XSL, XSLT)

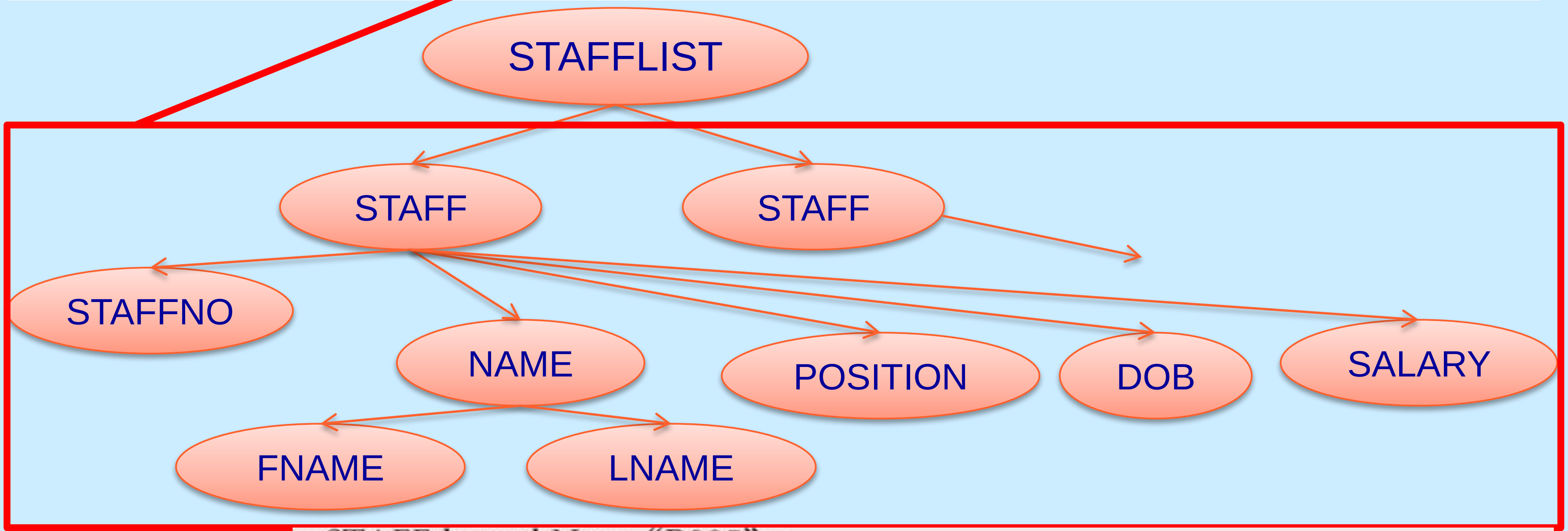
31.3.4 XML Path Language (XPath)

- Declarative query language
- Specifying a directory-like path
  - Possibly with conditions placed in the path
  - Retrieves collections of elements
- **Treats XML document as logical tree of nodes**

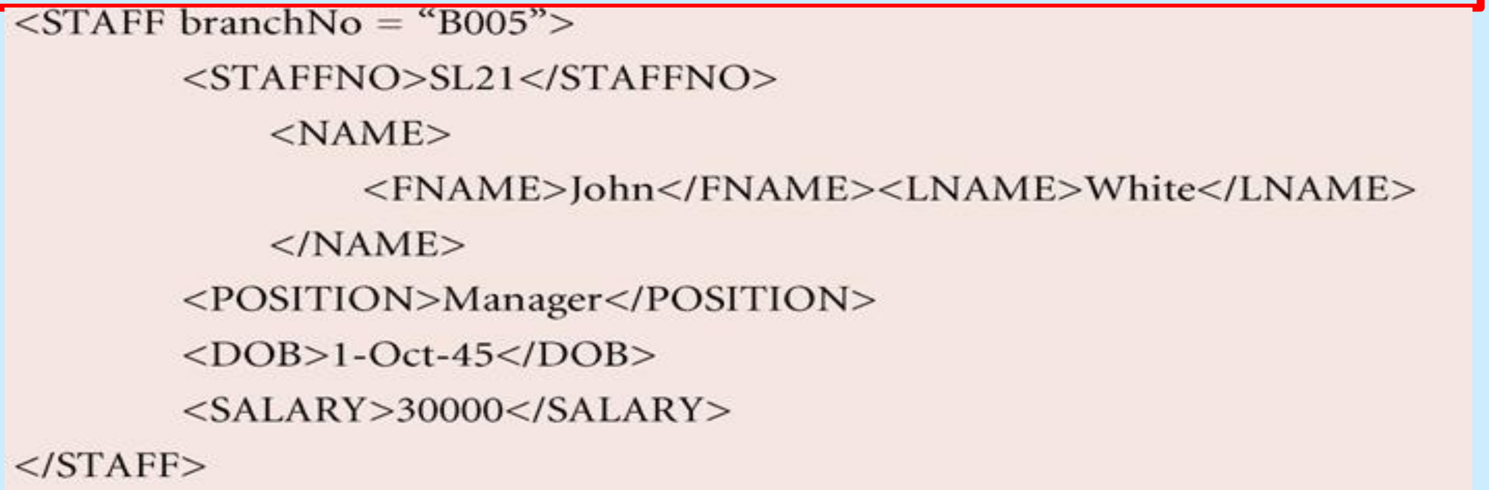
Query component	Description
context node	starting point
location path	path from one point in document to another, composed of steps
step	consists of basis and predicates
basis	axis and node test
axis	direction e.g. parent, ancestor, etc.
node test	identifies node type, e.g. element name or function text()
predicate	in square brackets after the basis

XPath: path expressions

Details of all staff? //STAFFLIST/child::STAFF *or* //STAFFLIST/STAFF



Part of the query result:

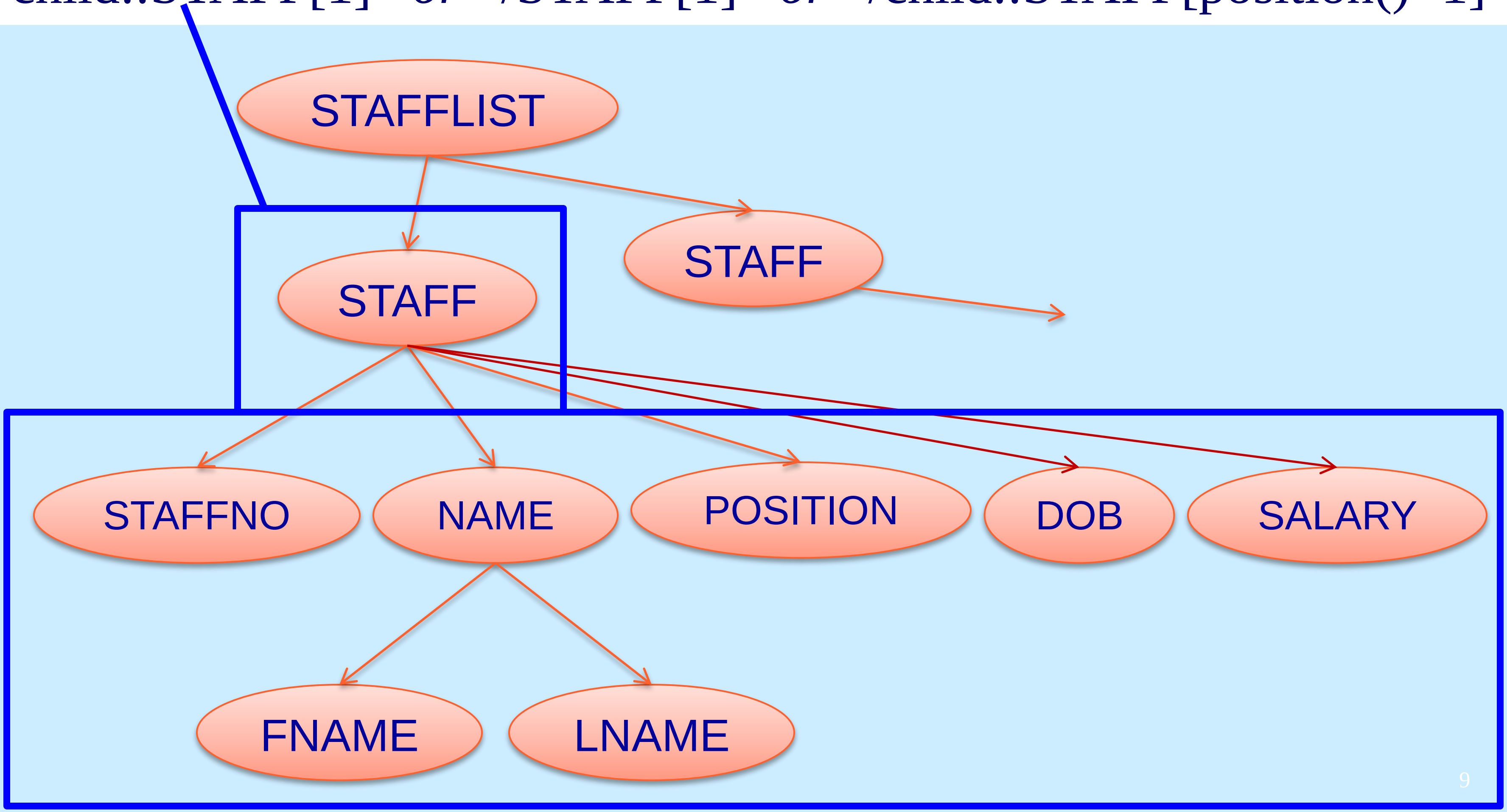


```
<STAFF branchNo = “B005”>
        <STAFFNO>SL21</STAFFNO>
            <NAME>
                <FNAME>John</FNAME><LNAME>White</LNAME>
            </NAME>
        <POSITION>Manager</POSITION>
        <DOB>1-Oct-45</DOB>
        <SALARY>30000</SALARY>
</STAFF>
```

XPath: path expressions

Details of 1st staff?

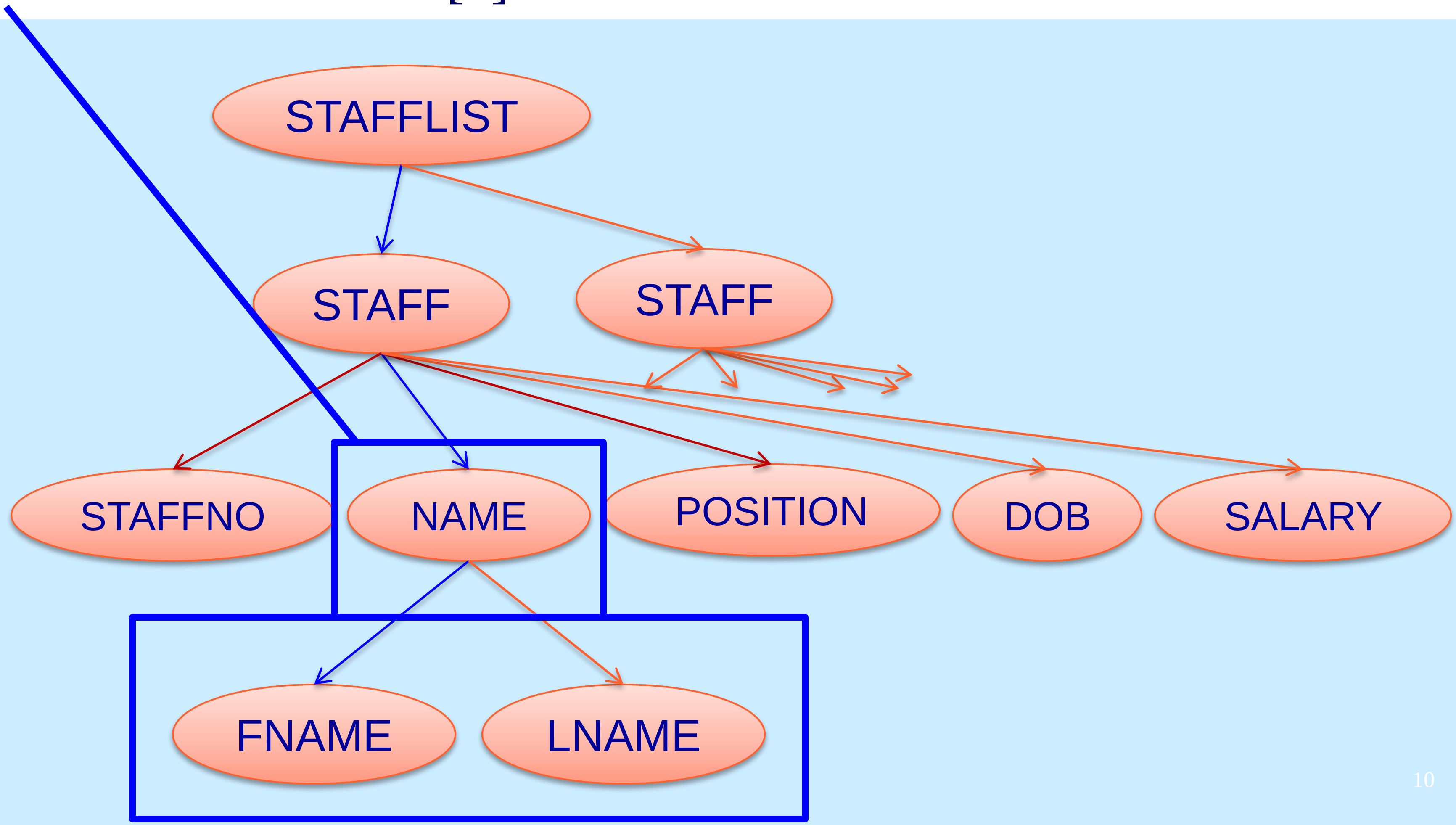
/child::STAFF[1] *or* /STAFF[1] *or* /child::STAFF[position()=1]



XPath: path expressions

name of 1st staff?

//STAFFLIST/STAFF[1]/NAME



XML Query Languages

- **XML Query languages**
  - SQL not good for XML (irregularity of XML data)
  - XML-QL, UnQL, XQL, XQuery …
- **XQuery by W3C XML Query Working Group**
  - Contributions from
    - database community
    - document community
    - programming language community

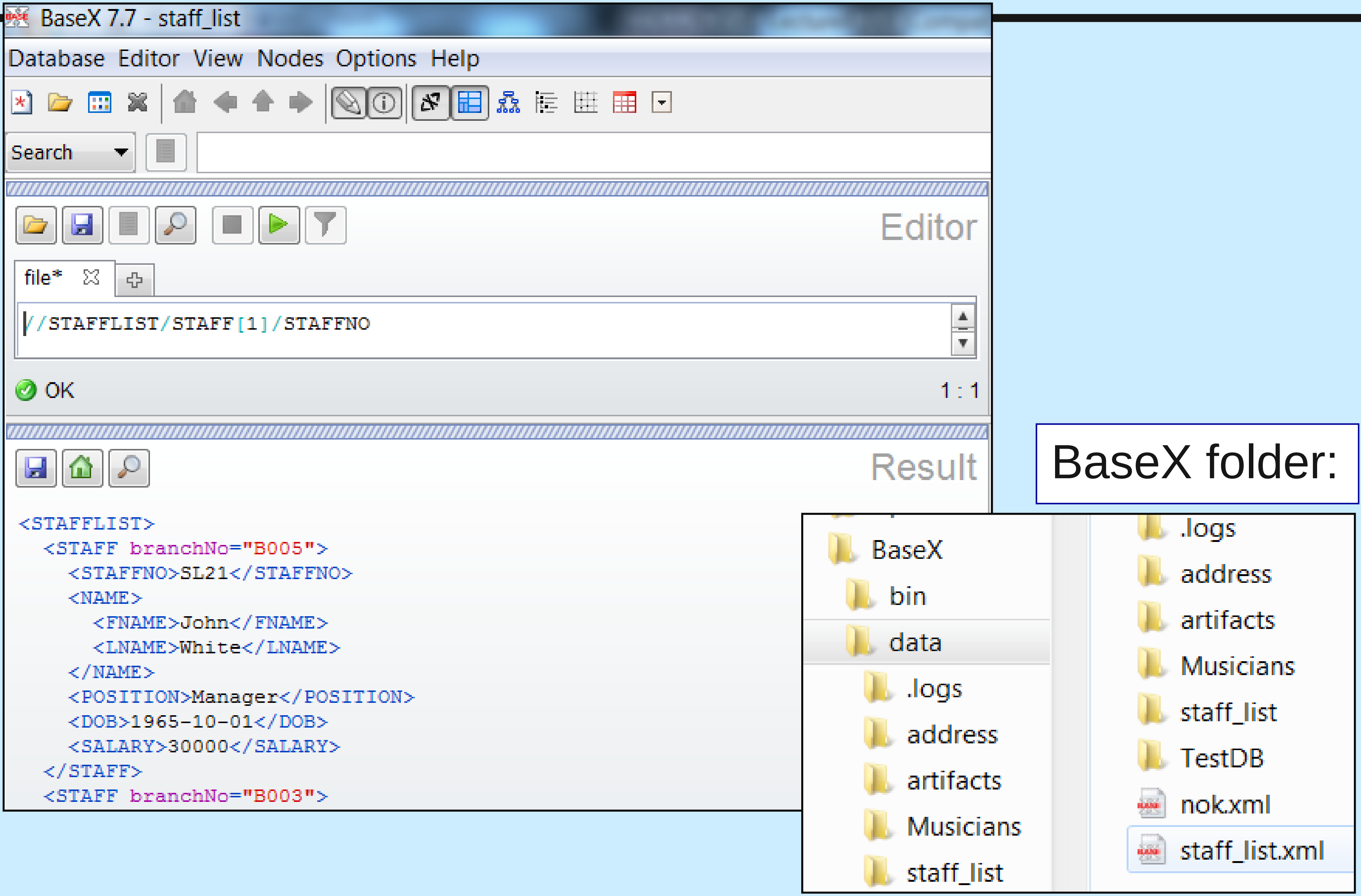
XQuery

- XQuery
  - a functional language
  - a query is represented as an expression
  - nested expressions possible

- XQuery uses
  1. Path expressions (**use the syntax of XPath**)
  2. **FLWOR** expressions

- **XML Queries**
  - operate on ***single documents***
  - operate also on ***fixed collections of documents***
  - **select *sub-trees* of documents**

BaseX



```
//STAFFLIST/STAFF[1]/STAFFNO
```

OK 1 : 1

```
<STAFFLIST>
  <STAFF branchNo="B005">
    <STAFFNO>SL21</STAFFNO>
    <NAME>
      <FNAME>John</FNAME>
      <LNAME>White</LNAME>
    </NAME>
    <POSITION>Manager</POSITION>
    <DOB>1965-10-01</DOB>
    <SALARY>30000</SALARY>
  </STAFF>
  <STAFF branchNo="B003">
```

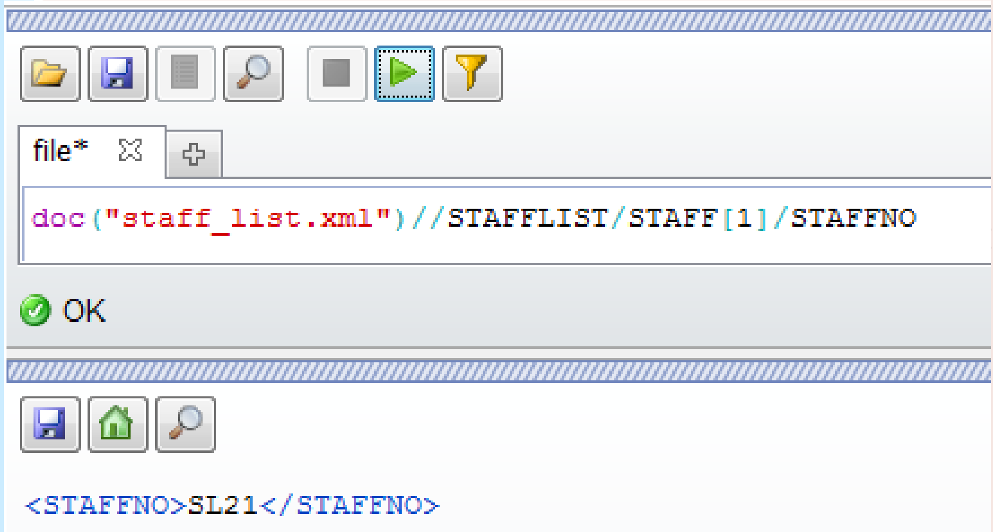
BaseX folder:

```
BaseX
  bin
  data
    .logs
    address
    artifacts
    Musicians
    staff_list
```

```
.logs
address
artifacts
Musicians
staff_list
TestDB
nok.xml
staff_list.xml
```

Find staff number of first member of staff in our XML document

```
<?xml version= “1.0” encoding= “UTF-8” standalone= “yes”?>
<?xml:stylesheet type = “text/xsl” href = “staff_list.xsl”?>
<!DOCTYPE STAFFLIST SYSTEM “staff_list.dtd”>
<STAFFLIST>
     <STAFF branchNo = “B005”>
               <STAFFNO>SL21</STAFFNO>
                    <NAME>
                         <FNAME>John</FNAME><LNAME>White</LNAME>
                    </NAME>
               <POSITION>Manager</POSITION>
               <DOB>1-Oct-45</DOB>
               <SALARY>30000</SALARY>
```



```
doc("staff_list.xml")//STAFFLIST/STAFF[1]/STAFFNO
```

✓ OK

```
<STAFFNO>SL21</STAFFNO>
```

```
/LNAME>
```

```
doc("staff_list.xml")//STAFFLIST/STAFF[1]/STAFFNO
```

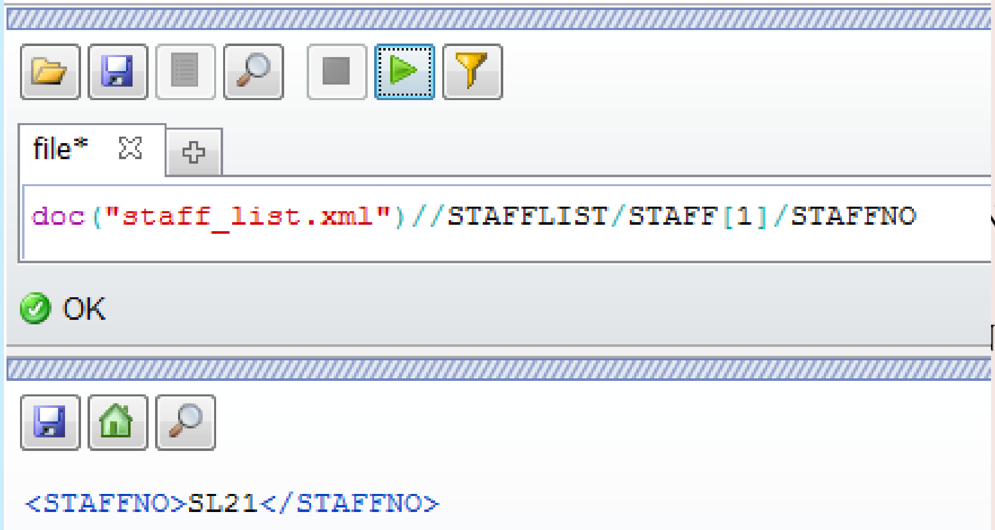
```
1. <?xml version= "1.0" encoding= "UTF-8      lalone= "yes"?>
   <?xml:styleshe
   <!DOCTYPE
2. <STAFFLIST>
3.   <STAFF branchNo = "B005">
4.       FFNO>SL21</STAFFNO>
         <NAME>
              <FNAME>John</FNAME><LNAME>          AME>
         </NAME>
       <POSITION>M
       <DOB>1-Oct-4
       <SALARY>300(
   </STAFF>
                         NAME>Beech</L
                         ]>
```

1. Opens **staff_list.xml** and returns its document node

2. uses //STAFFLIST to select STAFFLIST element

3. locates first STAFF element that is child of STAFFLIST element

4. finds STAFFNO elements occurring anywhere within this STAFF element



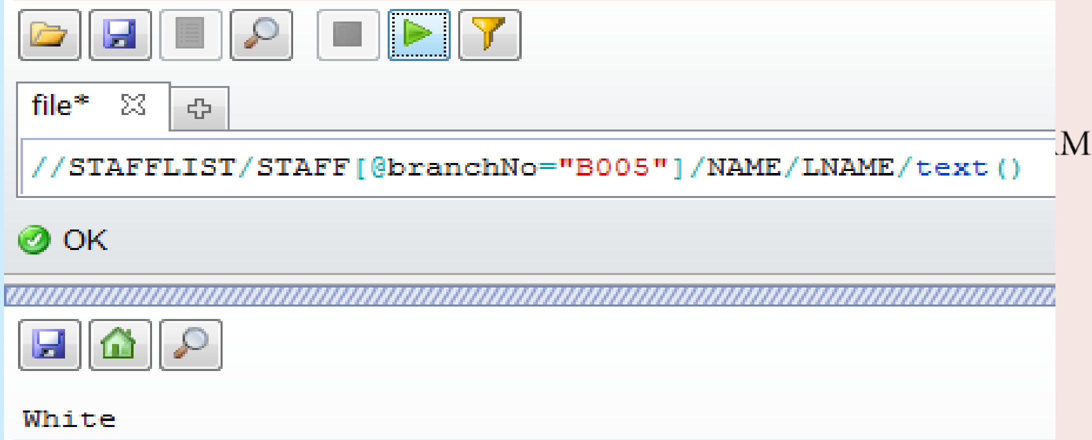
```
doc("staff_list.xml")//STAFFLIST/STAFF[1]/STAFFNO
```

```
<STAFFNO>SL21</STAFFNO>
```

Find surnames of staff at branch B005

```
<?xml version= “1.0” encoding= “UTF-8” standalone= “yes”?>
<?xml:stylesheet type = “text/xsl” href = “staff_list.xsl”?>
<!DOCTYPE STAFFLIST SYSTEM “staff_list.dtd”>
<STAFFLIST>
    <STAFF branchNo = “B005”>
            <STAFFNO>SL21</STAFFNO>
                <NAME>
                    <FNAME>John</FNAME><LNAME>White</LNAME>
                </NAME>
            <POSITION>Manager</POSITION>
            <DOB>1-Oct-45</DOB>
            <SALARY>30000</SALARY>
    </STAFF>
    <STAFF branchNo = “B003”>

                                                                  ME>
```



```
//STAFFLIST/STAFF[@branchNo="B005"]/NAME/LNAME/text()
```

OK

```
White
```

```
//STAFFLIST/STAFF[@branchNo="B005"]/NAME/LNAME/text()
```

```
<?xr l version= "1.0" e ling= "UTF-8" standalone= "yes"?>
<? l:stylesheet type =            href = "staff_list.xsl"?>
<  OCTYPE ST
  TAFFLIST>
    <STAFF bra
      <ST
        <NAME>
          <FNAME>John</FNAME><LN ME>White</LN
        </NAME>
      <POSITION>Manager</POSITION>
      <DOB>1-Oct-45</DOB>
      <SALARY>30000</SALARY>
    STAFF>
    STAFF branchNo = "B003">
                                    ME>Beech</LNAME>
```

1. if staff_list.xml is the currently open database file, no need to specify doc

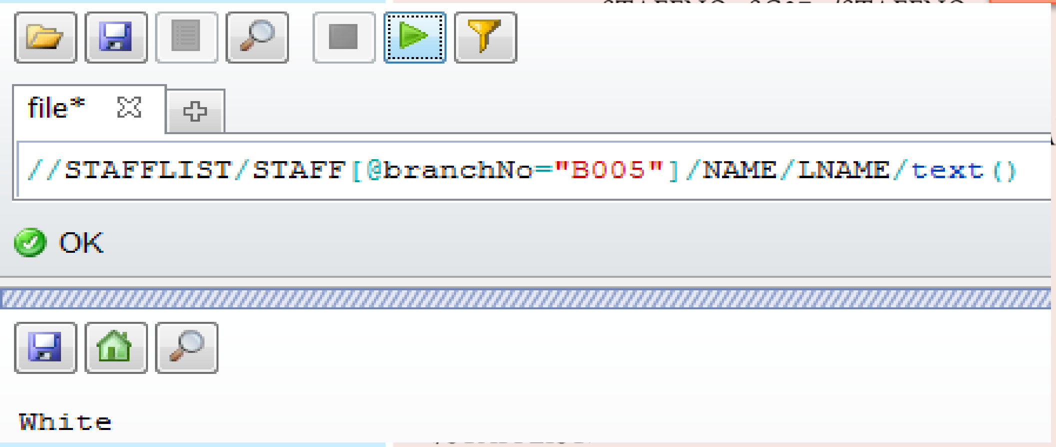
2. uses //STAFFLIST to select STAFFLI element

3.

3. uses /STAFF to select STAFF elements within STAFFLIST element

4. predicate: restricts STAFF elements to those with branchNo attribute = B005

5. selects LNAME element in NAME & extracts text



```
//STAFFLIST/STAFF[@branchNo="B005"]/NAME/LNAME/text()
```

```
White
```

XQuery – FLWOR Expressions

FLWOR ("flower") expression construction:

(1) starts with one or more **FOR or LET** clauses (any order)

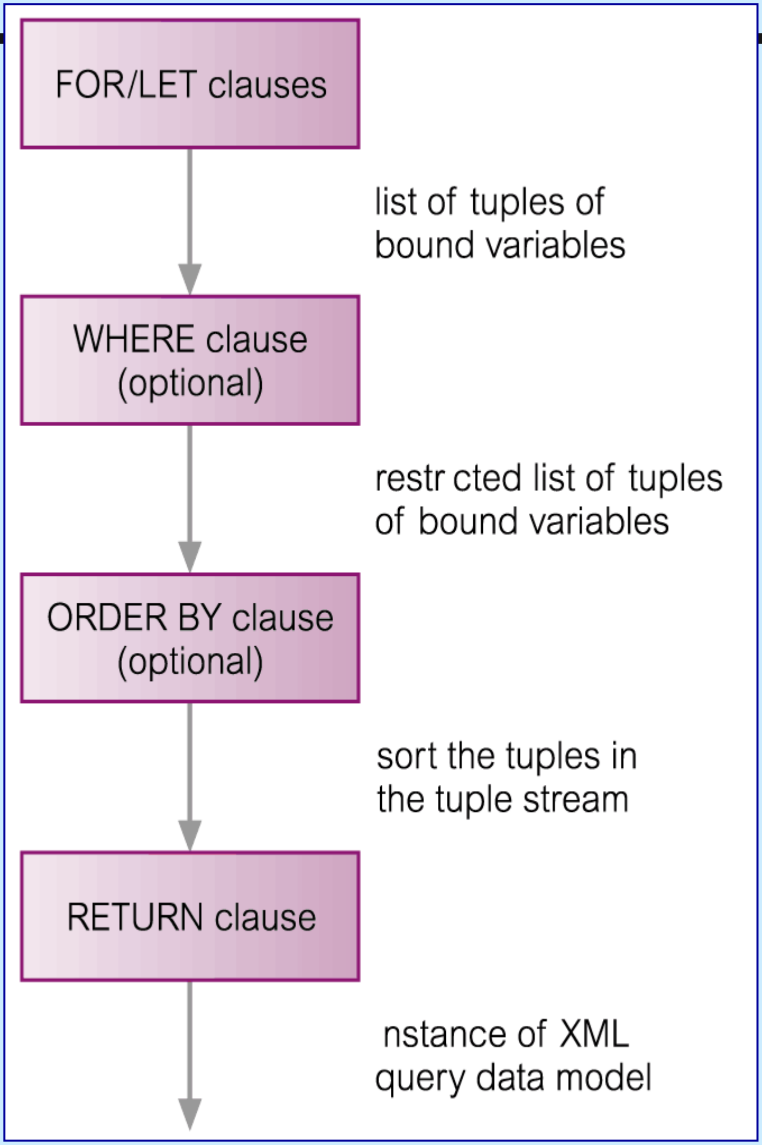
followed by:

(2) optional **WHERE** clause

(3) optional **ORDER BY** clause

(4) required **RETURN** clause

```
FOR        forVar IN inExpression
LET        letVar := letExpression
[WHERE          filterExpression ]
[ORDER BY  orderSpec ]
RETURN          expression
```



FOR/LET clauses

list of tuples of bound variables

WHERE clause (optional)

restr cted list of tuples of bound variables

ORDER BY clause (optional)

sort the tuples in the tuple stream

RETURN clause

nstance of XML query data model

FLWOR Expressions

- **FOR** **e.g. for $S in //STAFF**
  - iteration: associates each variable with expression
  - result is tuple stream
  - each tuple binds a variable to one of the items in the expression

- **LET** **e.g let $sal := 30000**
  - binds one or more variables to one or more expressions
  - without iteration (single binding for each variable)

```
FOR          forVar IN inExpression
LET          letVar := letExpression
[WHERE       filterExpression]
[ORDER BY    orderSpec]
RETURN       expression
```

FLWOR Expressions

- **WHERE** **e.g. where $S/SALARY > 10000**
  - one or more conditions to restrict tuples generated by FOR and LET (*optional*)

```
FOR         forVar IN inExpression
LET         letVar := letExpression
[WHERE      filterExpression]
[ORDER BY   orderSpec]
RETURN      expression
```

FLWOR Expressions

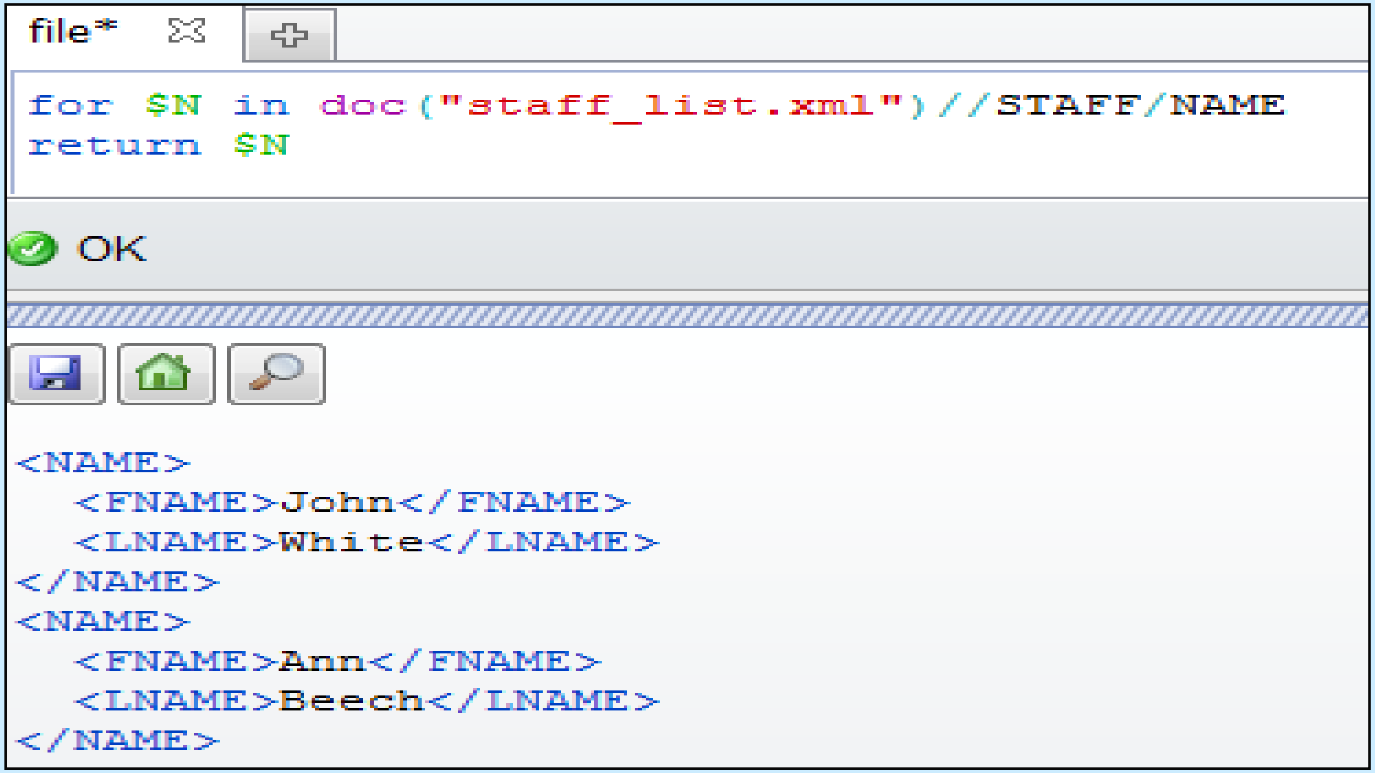
- **ORDER BY** **e.g. order by $S/STAFFNO descending**
  - order of the tuple stream
  - determines order in which RETURN clause is evaluated
- **RETURN** **e.g return $S**
  - evaluated once for each tuple in tuple stream
  - *query results concatenated to form returned result*

```
FOR         forVar IN inExpression
LET         letVar := letExpression
[WHERE      filterExpression]
[ORDER BY  orderSpec]
RETURN      expression
```

Example – XQuery FLWOR Expressions

List the names of all staff

```
for $N in doc("staff_list.xml")//STAFF/NAME
return $N
OR:
for $N in //STAFF/NAME  return $N
```



```
for $N in doc("staff_list.xml")//STAFF/NAME
return $N
```

```
<NAME>
   <FNAME>John</FNAME>
   <LNAME>White</LNAME>
</NAME>
<NAME>
   <FNAME>Ann</FNAME>
   <LNAME>Beech</LNAME>
</NAME>
```

Example 31.4 – XQuery FLWOR Expressions

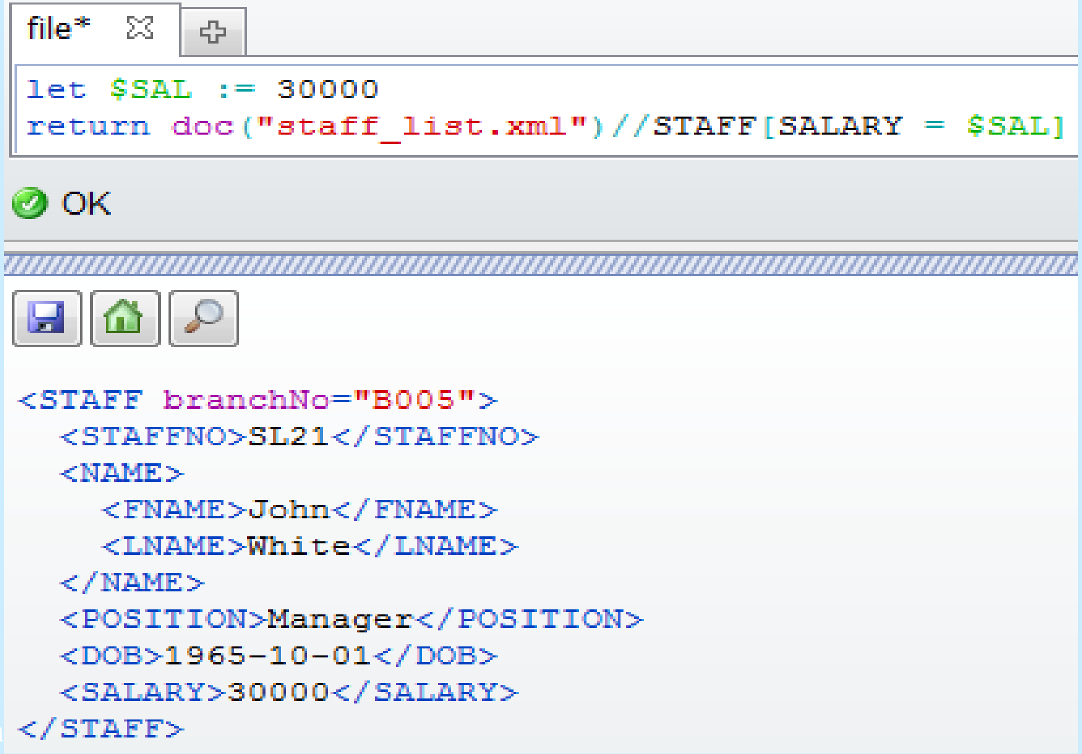
List staff with salary = R30,000

```
let $SAL := 30000
return doc("staff_list.xml")//STAFF[SALARY = $SAL]
(ALTERNATIVELY:
let $SAL := 30000
return //STAFF[SALARY = $sal] )
```

= extracts typed value of element (based on schema)

- resulting in a decimal value
- then compared with 30000

- other operators
  - ‘eq’, ‘ne’, ‘lt’, ‘le’, ‘gt’, ‘ge’: *value comparison operators*



```
let $SAL := 30000
return doc("staff_list.xml")//STAFF[SALARY = $SAL]
```

OK

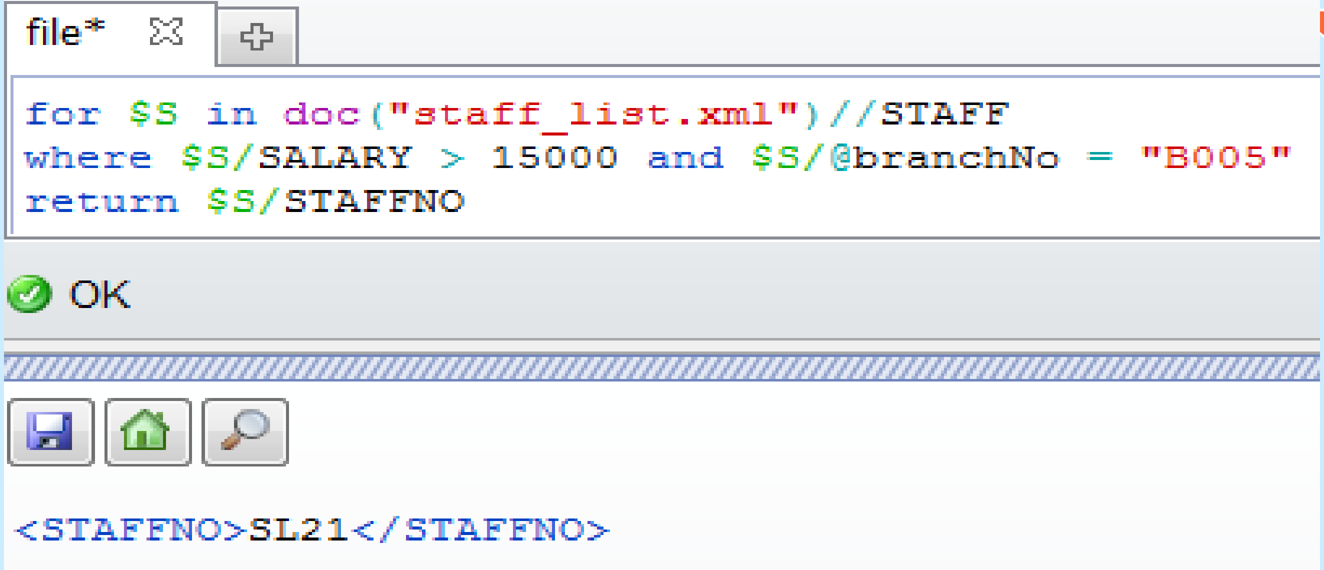
```
<STAFF branchNo="B005">
  <STAFFNO>SL21</STAFFNO>
  <NAME>
    <FNAME>John</FNAME>
    <LNAME>White</LNAME>
  </NAME>
  <POSITION>Manager</POSITION>
  <DOB>1965-10-01</DOB>
  <SALARY>30000</SALARY>
</STAFF>
```

Example 31.4 – XQuery FLWOR Expressions

Find staff number at branch B005 with salary > R15,000

```
for $S in doc("staff_list.xml")//STAFF
where $S/SALARY > 15000 and $S/@branchNo = "B005"
return $S/STAFFNO
```

```
ALTERNATIVELY:
for $S in //STAFF
where $S/SALARY > 15000 and $S/@branchNo = "B005"
return $S/STAFFNO
```



```
for $S in doc("staff_list.xml")//STAFF
where $S/SALARY > 15000 and $S/@branchNo = "B005"
return $S/STAFFNO
```

```
<STAFFNO>SL21</STAFFNO>
```

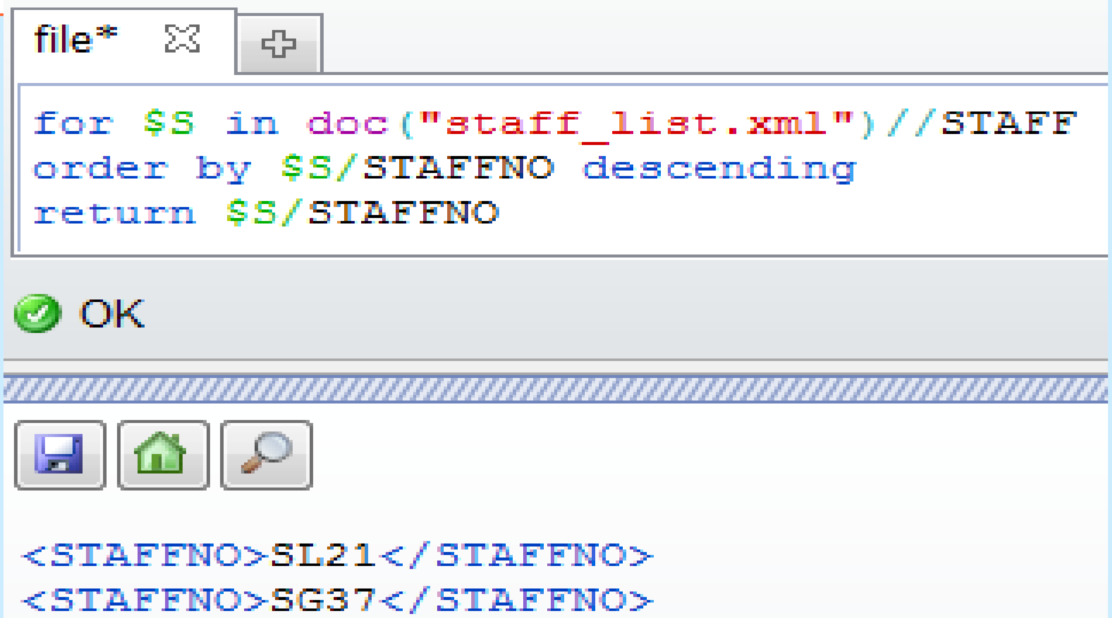
Example 31.4 – XQuery FLWOR Expressions

List staff numbers of all staff in descending order of staff number

```
for $S in doc("staff_list.xml")//STAFF
order by $S/STAFFNO descending
return $S/STAFFNO
```

OR:

```
for $S in //STAFF
order by $S/STAFFNO descending return $S/STAFFNO
```



```
for $S in doc("staff_list.xml")//STAFF
order by $S/STAFFNO descending
return $S/STAFFNO
```

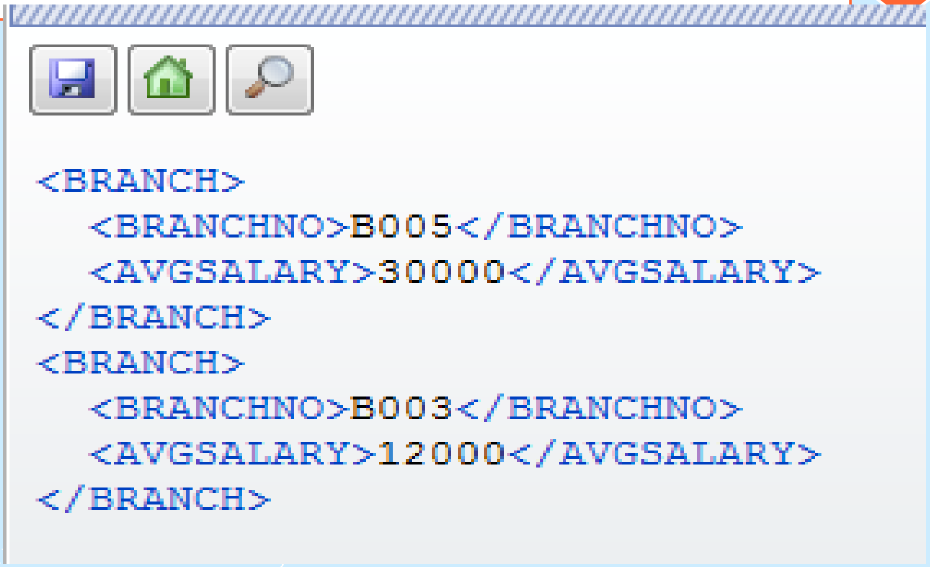
```
<STAFFNO>SL21</STAFFNO>
<STAFFNO>SG37</STAFFNO>
```

Example 31.4 – XQuery FLWOR Expressions

List each branch office and average salary at branch

```
for $B in  distinct-values(//@branchNo)
let $avgSalary :=
             avg(//STAFF[@branchNo = $B]/SALARY)
return
      <BRANCH>
            <BRANCHNO>{ $B }</BRANCHNO>
            <AVGSALARY>{ $avgSalary }</AVGSALARY>
      </BRANCH>
```

Functions for aggregates are: avg,........................



```
<BRANCH>
   <BRANCHNO>B005</BRANCHNO>
   <AVGSALARY>30000</AVGSALARY>
</BRANCH>
<BRANCH>
   <BRANCHNO>B003</BRANCHNO>
   <AVGSALARY>12000</AVGSALARY>
</BRANCH>
```

Example 31.4 – XQuery FLWOR Expressions

List branches with at least one member of staff with salary > R15,000

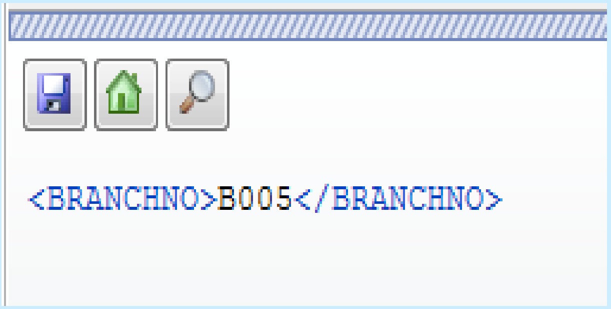
```
for $B in distinct-values(//@branchNo)
let $S := //STAFF[@branchNo = $B]
   where some $sal in $S/SALARY
             satisfies ($sal > 15000)

   return <BRANCHNO>{ $B}</BRANCHNO>
```

sub-query

some: existential qualifier (cf: WHERE EXISTS for SQL)

every: universal qualifier



```
<BRANCHNO>B005</BRANCHNO>
```

Example 31.5 – Joining Two Documents

File nok.xml --------------------->

Files:

staff_list.xml , nok.xml

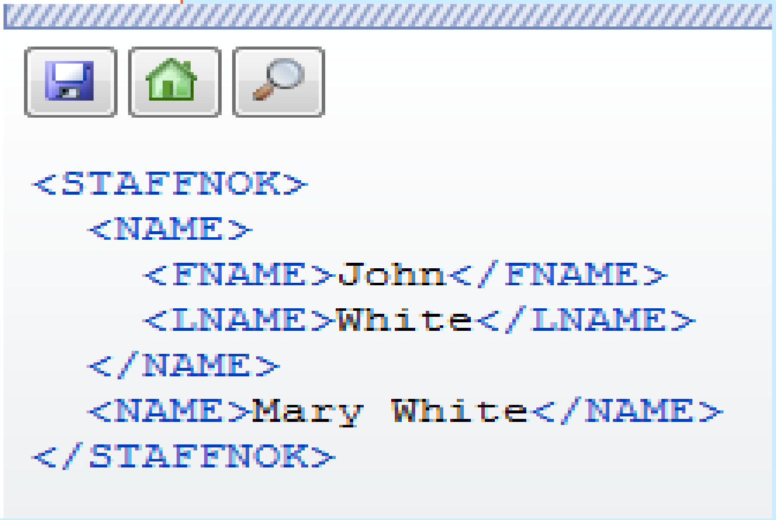
in folder

Prog files/BaseX/data

```
<NOKLIST>
    <NOK>
        <STAFFNO>SL21</STAFFNO>
        <NAME>Mary White</NAME>
    </NOK>
</NOKLIST>
```

List names of staff and names of their next of kin

```
for $S in
   doc("data/staff_list.xml")//STAFF,
   $NOK in doc("data/nok.xml")//NOK
where $S/STAFFNO = $NOK/STAFFNO
return
    <STAFFNOK>
     { $S/NAME, $NOK/NAME }
    </STAFFNOK>
```



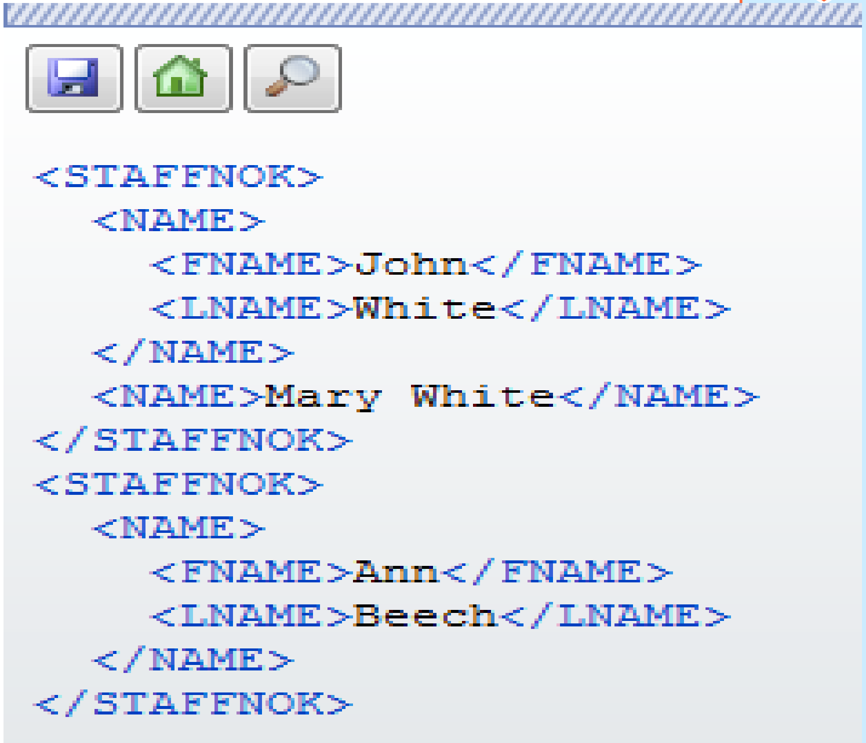
```
<STAFFNOK>
  <NAME>
    <FNAME>John</FNAME>
    <LNAME>White</LNAME>
  </NAME>
  <NAME>Mary White</NAME>
</STAFFNOK>
```

Example 31.5 – Joining Two Documents

List names of staff and names of their next of kin

```
for $S in doc("data/staff_list.xml")//STAFF
return <STAFFNOK>
            { $S/NAME }
            { for $NOK in doc("data/nok.xml")//NOK
                  where $S/STAFFNO = $NOK/STAFFNO
                  return $NOK/NAME }
       </STAFFNOK>
```

Correlated sub-query



```
<STAFFNOK>
  <NAME>
    <FNAME>John</FNAME>
    <LNAME>White</LNAME>
  </NAME>
  <NAME>Mary White</NAME>
</STAFFNOK>
<STAFFNOK>
  <NAME>
    <FNAME>Ann</FNAME>
    <LNAME>Beech</LNAME>
  </NAME>
</STAFFNOK>
```

Example 31.6 – User-Defined Function

Simple arithmetic function in BaseX

FUNCTION DEFINITION:

```
let $mult := function($x, $y) {$x * $y}
```

FUNCTION CALL:

```
return $mult(10,10)
```

RESULT: ???

Built-in functions: Reading for the student

Example 31.6 – User-Defined Function – BaseX?

Function to return staff at a given branch

```
define function staffAtBranch($bNo) AS element()*
{      for $S in doc("data/staff_list.xml")//STAFF
       where $S/@branchNo = $bNo
       order by $S/STAFFNO
       return $S/STAFFNO, $S/NAME, $S/POSITION,
$S/SALARY
}

for  $B in
       distinct-values(doc("data/staff_list.xml")//@branchNo)
order by $B
return
     <BRANCHNO> { $B/text() }
       { staffAtBranch($B) }  ←  **the function call*
     </BRANCHNO>
```

Exercise for student: check if this works in BaseX