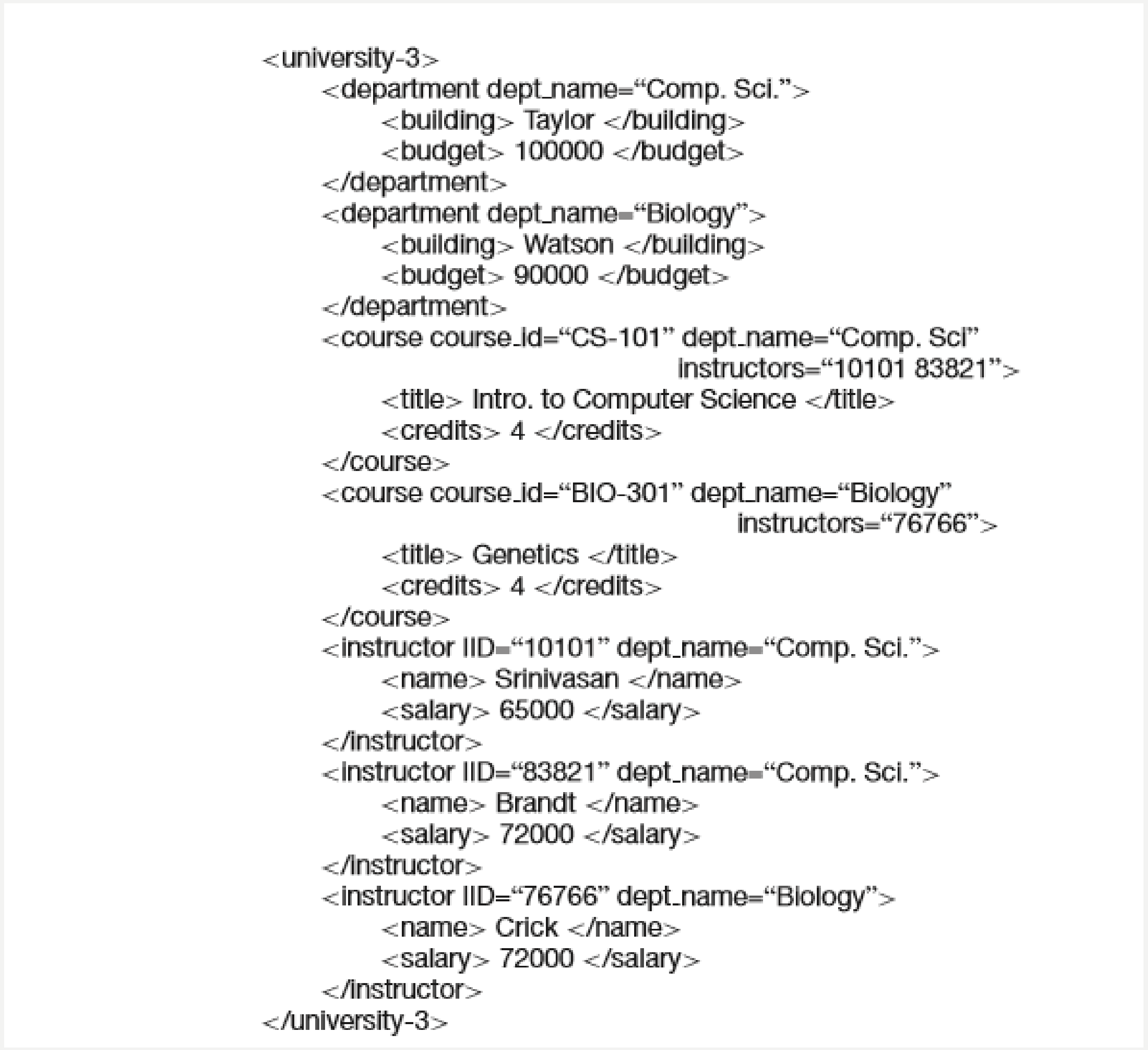
XQUERY

Compiled BY
M ABDUL JAWAD

TREE MODEL OF XML DATA

- Query and transformation languages are based on a **tree model** of XML data
- An XML document is modeled as a tree, with **nodes** corresponding to elements and attributes
  - Element nodes have child nodes, which can be attributes or subelements
  - Text in an element is modeled as a text node child of the element
  - Children of a node are ordered according to their order in the XML document
  - Element and attribute nodes (except for the root node) have a single parent, which is an element node



```
<university-3>
      <department dept_name=“Comp. Sci.”>
            <building> Taylor </building>
            <budget> 100000 </budget>
      </department>
      <department dept_name=“Biology”>
            <building> Watson </building>
            <budget> 90000 </budget>
      </department>
      <course course_id=“CS-101” dept_name=“Comp. Sci”
                                    instructors=“10101 83821”>
            <title> Intro. to Computer Science </title>
            <credits> 4 </credits>
      </course>
      <course course_id=“BIO-301” dept_name=“Biology”
                                          instructors=“76766”>
            <title> Genetics </title>
            <credits> 4 </credits>
      </course>
      <instructor IID=“10101” dept_name=“Comp. Sci.”>
            <name> Srinivasan </name>
            <salary> 65000 </salary>
      </instructor>
      <instructor IID=“83821” dept_name=“Comp. Sci.”>
            <name> Brandt </name>
            <salary> 72000 </salary>
      </instructor>
      <instructor IID=“76766” dept_name=“Biology”>
            <name> Crick </name>
            <salary> 72000 </salary>
      </instructor>
</university-3>
```

XPATH

- XPath is used to address (select) parts of documents using **path expressions**
- A path expression is a sequence of steps separated by "/"
  - Think of file names in a directory hierarchy
- Result of path expression: set of values that along with their containing elements/attributes match the specified path
- E.g. /university-3/instructor/name evaluated on the university-3 data returns

```
<name>Srinivasan</name>
<name>Brandt</name>
```

- E.g. /university-3/instructor/name/text( )
  returns the same names, but without the enclosing tags

XPATH (CONT.)

- The initial "/" denotes root of the document (above the top-level tag)
- Path expressions are evaluated left to right
  - Each step operates on the set of instances produced by the previous step
- Selection predicates may follow any step in a path, in [ ]
  - E.g. /university-3/course[credits >= 4]
    - returns course elements with credits >= 4
    - /university-3/course[credits] returns course elements containing a credits subelement
- Attributes are accessed using "@"
  - E.g. /university-3/course[credits >= 4]/@course_id
    - returns the course identifiers of courses with credits >= 4
  - IDREF attributes are not dereferenced automatically (more on this later)

FUNCTIONS IN XPATH

- XPath provides several functions
  - The function count() at the end of a path counts the number of elements in the set generated by the path
    - E.g. /university-2/instructor[count(./teaches/course)> 2]
      - Returns instructors teaching more than 2 courses (on university-2 schema)
  - Also function for testing position (1, 2, ..) of node w.r.t. siblings
- Boolean connectives and and or and function not() can be used in predicates
- IDREFs can be referenced using function id()
  - id() can also be applied to sets of references such as IDREFS and even to strings containing multiple references separated by blanks
  - E.g. /university-3/course/id(@dept_name)
    - returns all department elements referred to from the dept_name attribute of course elements.

MORE XPATH FEATURES

- Operator "|" used to implement union
  - E.g. /university-3/course[@dept name="Comp. Sci"] | /university-3/course[@dept name="Biology"]
    - Gives union of Comp. Sci. and Biology courses
    - However, "|" cannot be nested inside other operators.
- "//" can be used to skip multiple levels of nodes
  - E.g. /university-3//name
    - finds any name element *anywhere* under the /university-3 element, regardless of the element in which it is contained.
- A step in the path can go to parents, siblings, ancestors and descendants of the nodes generated by the previous step, not just to the children
  - "//", described above, is a short from for specifying "all descendants"
  - ".." specifies the parent.
- doc(name) returns the root of a named document

XQUERY

- XQuery is a general purpose query language for XML data
- Currently being standardized by the World Wide Web Consortium (W3C)
  - The textbook description is based on a January 2005 draft of the standard. The final version may differ, but major features likely to stay unchanged.
- XQuery is derived from the Quilt query language, which itself borrows from SQL, XQL and XML-QL
- XQuery uses a

  for … let … where … order by …return …

  syntax

  for ⇔ SQL **from**
  where ⇔ SQL **where**
  order by ⇔ SQL **order by**

  return ⇔ SQL **select**
  let allows temporary variables, and has no equivalent in SQL

FLWOR SYNTAX IN XQUERY

- For clause uses XPath expressions, and variable in for clause ranges over values in the set returned by XPath
- Simple FLWOR expression in XQuery
  - find all courses with credits > 3, with each result enclosed in an <course_id> .. </course_id> tag

```
for  $x in /university-3/course
let   $courseId := $x/@course_id
where $x/credits > 3
return <course_id> { $courseId } </course id>
```

  - Items in the **return** clause are XML text unless enclosed in {}, in which case they are evaluated
- Let clause not really needed in this query, and selection can be done In XPath. Query can be written as:

```
 for $x in /university-3/course[credits > 3]
return <course_id> { $x/@course_id } </course_id>
```

- Alternative notation for constructing elements:

```
return element course_id { element  $x/@course_id }
```

JOINS

- Joins are specified in a manner very similar to SQL

```
    for $c in /university/course,
        $i in /university/instructor,
        $t in /university/teaches
  where $c/course_id= $t/course id and $t/IID = $i/IID
  return <course_instructor> { $c $i } </course_instructor>
```

- The same query can be expressed with the selections specified as XPath selections:

```
    for $c in /university/course,
        $i in /university/instructor,
        $t in /university/teaches[ $c/course_id= $t/course_id
                                    and $t/IID = $i/IID]
  return <course_instructor> { $c $i } </course_instructor>
```

NESTED QUERIES

- The following query converts data from the flat structure for university information into the nested structure used in university-1

```
  <university-1>
{     for $d in /university/department
       return <department>
                    { $d/* }
                    { for $c in /university/course[dept name = $d/dept name]
                      return $c }
                 </department>
}
{      for $i in /university/instructor
       return  <instructor>
                     { $i/* }
                     { for $c in /university/teaches[IID = $i/IID]
                       return $c/course id }
                  </instructor>
}
</university-1>
```

- $d/* denotes all the children of the node to which $d is bound, without the enclosing top-level tag

GROUPING AND AGGREGATION

- Nested queries are used for grouping

```
for $d in /university/department
return
      <department-total-salary>
           <dept_name> { $d/dept name } </dept_name>
            <total_salary> { fn:sum(
                 for $i in /university/instructor[dept_name = $d/dept_name]
                 return $i/salary
               ) }
            </total_salary>
       </department-total-salary>
```

SORTING IN XQUERY

- The **order by** clause can be used at the end of any expression. E.g. to return instructors sorted by name

```
 for $i in /university/instructor
order by $i/name
return <instructor> { $i/* } </instructor>
```

- Use **order by** $i/name **descending** to sort in descending order
- Can sort at multiple levels of nesting (sort departments by dept_name, and by courses sorted to course_id within each department)

```
<university-1> {
 for $d in /university/department
 order by $d/dept name
 return
      <department>
          { $d/* }
          { for $c in /university/course[dept name = $d/dept name]
            order by $c/course id
            return <course> { $c/* } </course> }
      </department>
} </university-1>
```

FUNCTIONS AND OTHER XQUERY FEATURES

- User defined functions with the type system of XMLSchema

```
  declare function local:dept_courses($iid as xs:string)
          as element(course)*
{
   for $i in /university/instructor[IID = $iid],
          $c in /university/courses[dept_name = $i/dept name]
   return $c
}
```

- Types are optional for function parameters and return values
- The * (as in decimal*) indicates a sequence of values of that type
- Universal and existential quantification in where clause predicates
  - **some** $e **in** *path* **satisfies** *P*
  - **every** $e **in** *path* **satisfies** *P*
  - Add **fn:exists($e)** to prevent empty $e from satisfying **every** clause
- XQuery also supports If-then-else clauses

- For example, to find departments where every instructor has a salary greater than $50,000, we can use the following query:
- **for $d in /university/department**

  where every $i in /university/instructor[dept name=$d/dept name] satisfies $i/salary > *50000*

  return $d
- Note, however, that if a department has no instructor, it will trivially satisfy the
- above condition. An extra clause:

fn:exists(/university/instructor[dept name=$d/dept name])