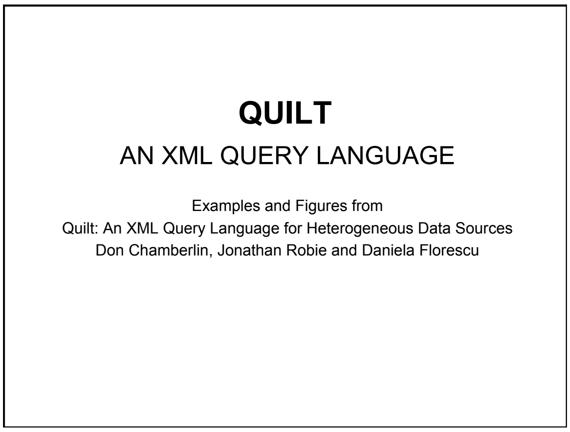
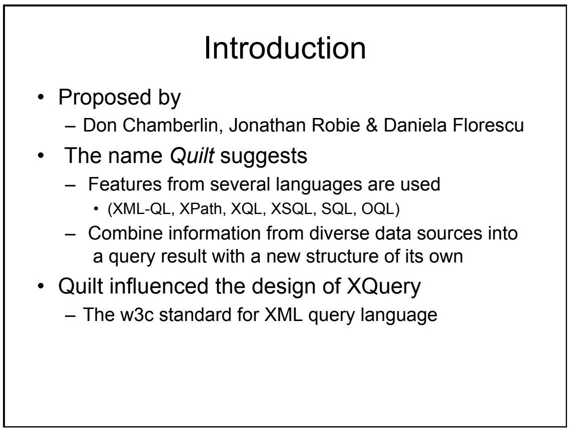


QUILT

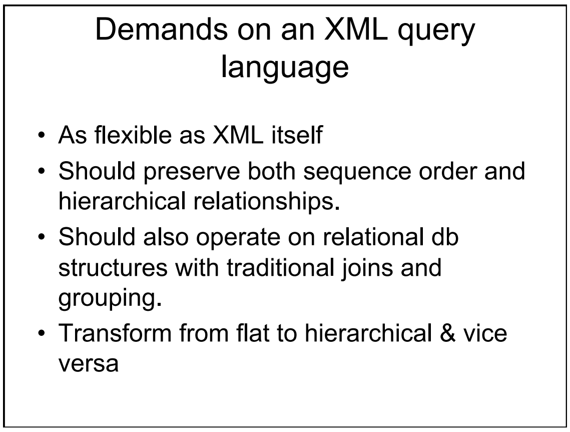
AN XML QUERY LANGUAGE

Examples and Figures from
Quilt: An XML Query Language for Heterogeneous Data Sources
Don Chamberlin, Jonathan Robie and Daniela Florescu



Introduction

- Proposed by
  - Don Chamberlin, Jonathan Robie & Daniela Florescu
- The name *Quilt* suggests
  - Features from several languages are used
    - (XML-QL, XPath, XQL, XSQL, SQL, OQL)
  - Combine information from diverse data sources into a query result with a new structure of its own
- Quilt influenced the design of XQuery
  - The w3c standard for XML query language



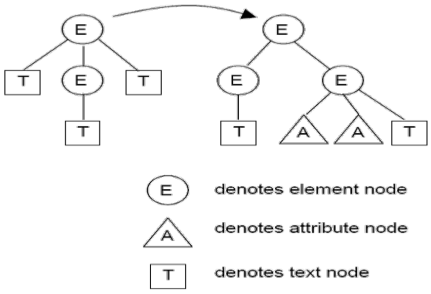
Demands on an XML query language

- As flexible as XML itself
- Should preserve both sequence order and hierarchical relationships.
- Should also operate on relational db structures with traditional joins and grouping.
- Transform from flat to hierarchical & vice versa

The Quilt Language

- QUILT borrows
  - XPATH, XQL – path expressions
  - XML-QL – variable bindings
  - SQL – series of clauses structure
  - OQL – functional language
- Query is represented as an expression.
- Input and output of a Quilt query are
  - XML documents,
  - fragments of XMLdocuments, or
  - collections of XML documents.

Model: An Ordered Forest



Principal forms of Quilt expressions

- Path expressions
- Element constructors
- FLWR expressions
- Expressions involving operators and functions
- Conditional expressions
- Quantifiers
- Variable bindings

Path Expressions

- Provides a way to navigate through a hierarchy of nodes.
- Use the operators of the XPath abbreviated syntax.

In the second chapter of the document named "zoo.xml", find the figure(s) with caption "Tree Frogs".

```
document("zoo.xml")/chapter[2]//
            figure[caption = "Tree Frogs"]
```

Dereference Operator("->")

- When a dereference operator follows an IDREF-type attribute or a key, it returns the element(s) that are referenced by the attribute or key.
- Can be used in the steps of a path expression.

Find captions of figures that are referenced by <figref> elements in the chapter of "zoo.xml" with title "Frogs".

```
document("zoo.xml")/chapter[title = "Frogs"]
                                //figref/@refid->/caption
```

Element Constructors

- Used to generate an element node in the output.
- Consists of a start tag and an end tag, enclosing an optional list of expressions

Generate an <emp> element containing an "empid" attribute and nested <name> and <job> elements. The values of the attribute and nested elements are specified by variables that are bound in other parts of the query.

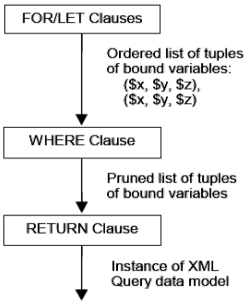
```
<emp empid = $id>
     <name> $n </name>,
     <job> $j </job>
</emp>
```

```
<$tagname ATTRIBUTES $attrs>
    <description> $d </description>,
    <price> $p </price>
</$tagname>
```

FLWR Expressions

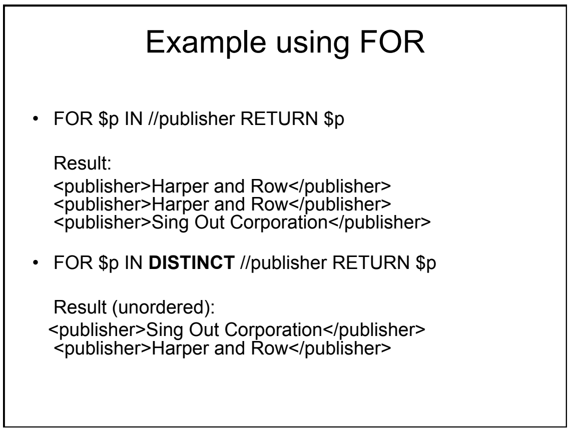
- Pronounced as *flower* expressions
- Constructed from FOR, LET, WHERE, and RETURN clauses
  - These clauses must appear in a specific order
- Used whenever it is necessary to iterate over the elements of a collection

Flow of data in a FLWR expression


FOR/LET Clauses
Ordered list of tuples of bound variables: ($x, $y, $z), ($x, $y, $z)
WHERE Clause
Pruned list of tuples of bound variables
RETURN Clause
Instance of XML Query data model

The FOR clause

- Generates bindings for one or more variables
- Variables bound by *for* stand for a single node + desc.
- Number of tuples generated is the product of the cardinalities of the node-sets returned by the respective expressions.
- Ordering among the tuples
  - derived from the ordering of their elements in the input document, with the first bound variable taking precedence, followed by the second bound variable, etc.
- Use of **DISTINCT** keyword
  - A node set generated using **DISTINCT** is unordered .



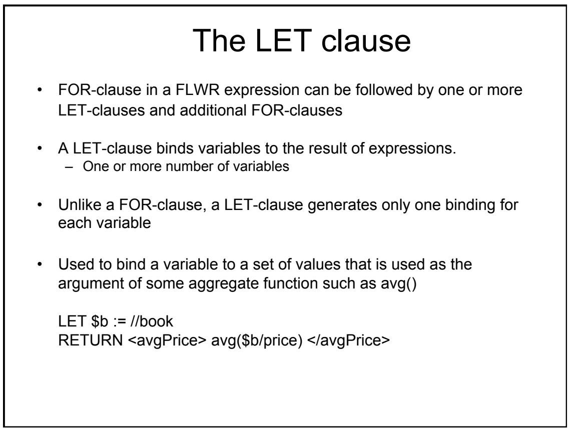
Example using FOR

- FOR $p IN //publisher RETURN $p

  Result:
  <publisher>Harper and Row</publisher>
  <publisher>Harper and Row</publisher>
  <publisher>Sing Out Corporation</publisher>

- FOR $p IN **DISTINCT** //publisher RETURN $p

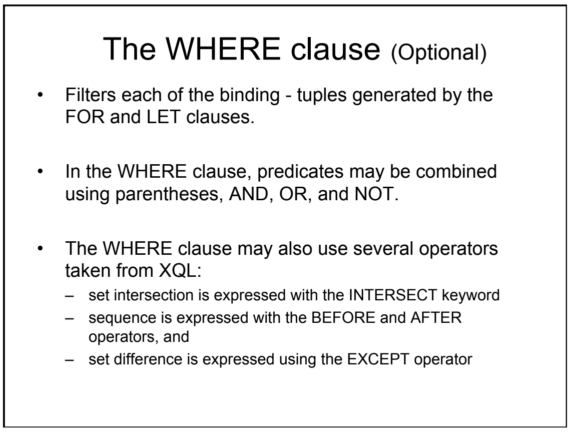
  Result (unordered):
  <publisher>Sing Out Corporation</publisher>
  <publisher>Harper and Row</publisher>



The LET clause

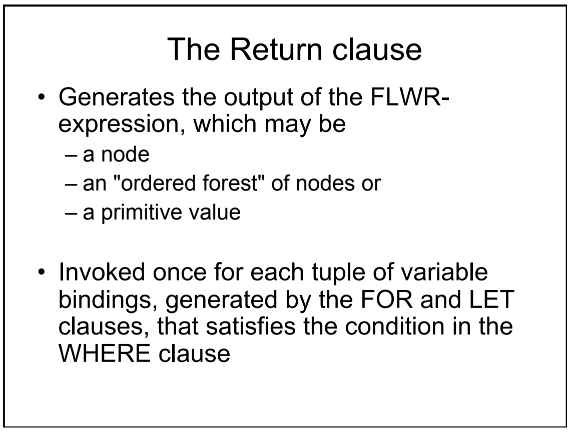
- FOR-clause in a FLWR expression can be followed by one or more LET-clauses and additional FOR-clauses
- A LET-clause binds variables to the result of expressions.
  - One or more number of variables
- Unlike a FOR-clause, a LET-clause generates only one binding for each variable
- Used to bind a variable to a set of values that is used as the argument of some aggregate function such as avg()

  LET $b := //book
  RETURN <avgPrice> avg($b/price) </avgPrice>



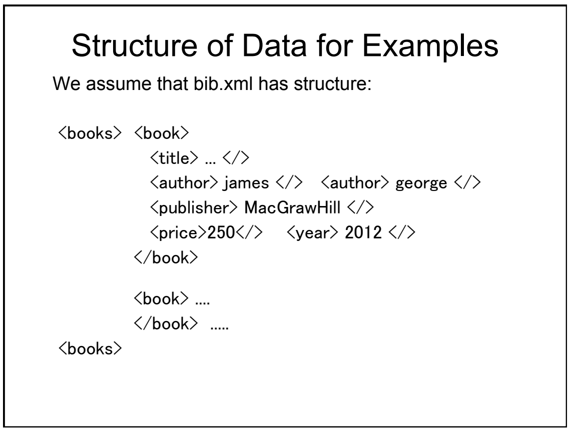
The WHERE clause (Optional)

- Filters each of the binding - tuples generated by the FOR and LET clauses.
- In the WHERE clause, predicates may be combined using parentheses, AND, OR, and NOT.
- The WHERE clause may also use several operators taken from XQL:
  - set intersection is expressed with the INTERSECT keyword
  - sequence is expressed with the BEFORE and AFTER operators, and
  - set difference is expressed using the EXCEPT operator



The Return clause

- Generates the output of the FLWR-expression, which may be
  - a node
  - an "ordered forest" of nodes or
  - a primitive value
- Invoked once for each tuple of variable bindings, generated by the FOR and LET clauses, that satisfies the condition in the WHERE clause

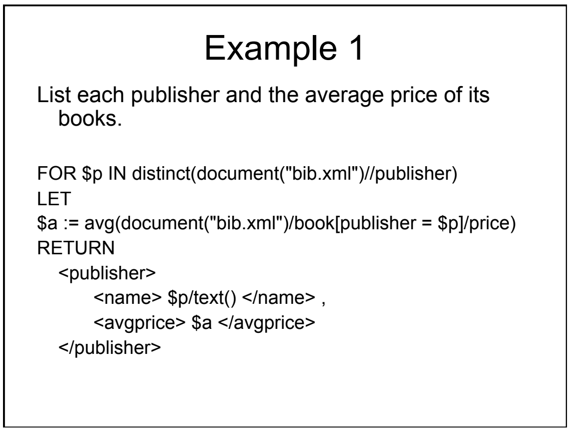


Structure of Data for Examples

We assume that bib.xml has structure:

```
<books>  <book>
            <title> … </>
            <author> james </>   <author> george </>
            <publisher> MacGrawHill </>
            <price>250</>    <year> 2012 </>
         </book>

         <book> ….
         </book>   …..
<books>
```



Example 1

List each publisher and the average price of its books.

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET
$a := avg(document("bib.xml")/book[publisher = $p]/price)
RETURN
   <publisher>
        <name> $p/text() </name> ,
        <avgprice> $a </avgprice>
   </publisher>
```

Example 2

List the publishers who have published more than 100 books.

```
<big_publishers>
   FOR $p IN distinct(document("bib.xml")//publisher)
   LET $b := document("bib.xml")/book[publisher = $p]
   WHERE count($b) > 100
   RETURN $p
</big_publishers>
```

Structural Transformation

FLOWER expressions are quite useful for structural inversion and other transformations

Get author-wise list of book titles:

```
<author_list>
   FOR $a IN distinct(document("bib.xml")//author)
   RETURN
   <author>
        <name> $a/text() </name>,
        FOR $b IN document("bib.xml")//book[author = $a]
        RETURN $b/title
   </author>
</author_list>
```

The SortBy clause

- To specify an ordering among the resulting elements
- Used after an element constructor or path expression
- Evaluation of arguments of the SORTBY clause
- ASCENDING is the default

An Example

Make an alphabetic list of publishers. Within each publisher, make a list of books, each containing a title and a price, in descending order of price.

```
<publisher_list>
      FOR $p IN distinct(document("bib.xml")//publisher
      RETURN
      <publisher>
         <name> $p/text() </name>,
         FOR $b IN document("bib.xml")//book[publisher = $p]
         RETURN
         <book>
                  $b/title, $b/price
         </book> SORTBY(price DESCENDING)
      </publisher> SORTBY(name)
</publisher_list>
```

Operators in Expressions

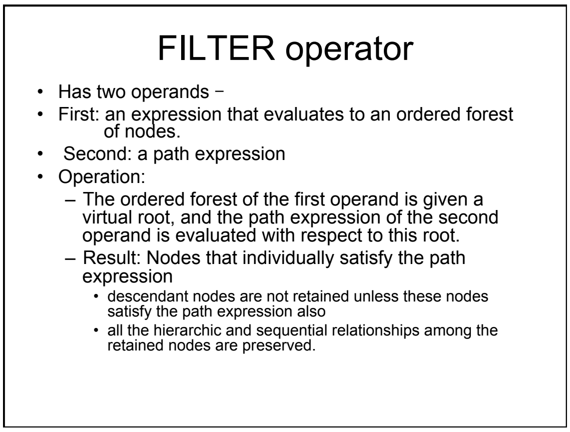
Quilt supports

- The usual set of arithmetic and logical operators
- The collection operators UNION, INTERSECT, and EXCEPT
- BEFORE and AFTER, useful in searching for information by its ordinal position
- FILTER operator
  The filtering process is based on node identity.

Conditional Expressions

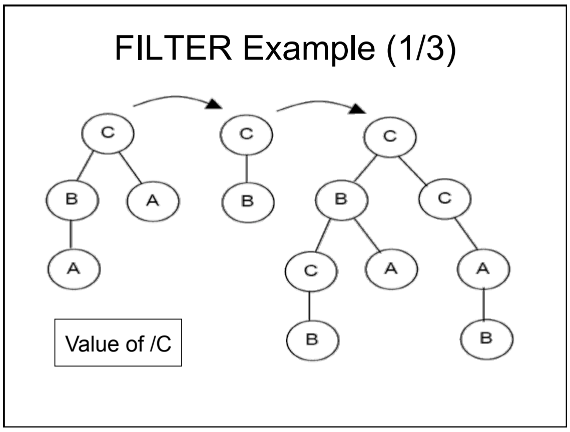
- Useful when the structure of the information to be returned depends on some condition.
- *Make a list of holdings, ordered by title. For journals, include the editor, and for all other holdings, include the author.*

```
FOR $h IN //holding
RETURN
   <holding>
         $h/title,
         IF $h/@type = “Journal” THEN $h/editor
         ELSE $h/author
   </holding> SORTBY (title)
```



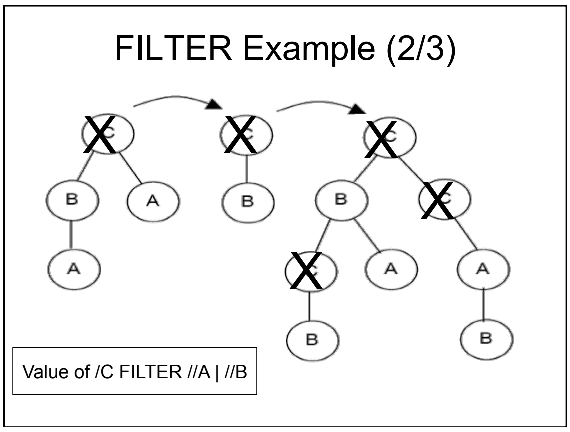
FILTER operator

- Has two operands –
- First: an expression that evaluates to an ordered forest of nodes.
- Second: a path expression
- Operation:
  - The ordered forest of the first operand is given a virtual root, and the path expression of the second operand is evaluated with respect to this root.
  - Result: Nodes that individually satisfy the path expression
    - descendant nodes are not retained unless these nodes satisfy the path expression also
    - all the hierarchic and sequential relationships among the retained nodes are preserved.



FILTER Example (1/3)

Value of /C

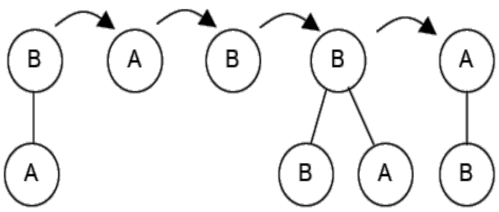


FILTER Example (2/3)

Value of /C FILTER //A | //B

FILTER Example(3/3)

Value of /C FILTER //A | //B



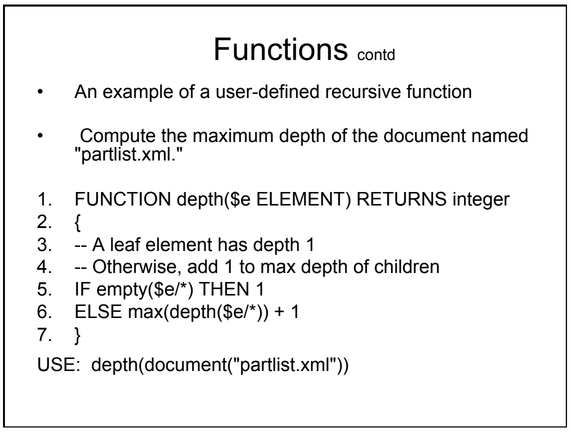
Another Example

The following simple query
generates data for the table of contents of "cookbook".

```
<toc>
     document("cookbook.xml") FILTER
             //chapter | //chapter/title | //chapter/title/text() |
             //section | //section/title | //section/title/text()
</toc>
```

Functions

- Provides a library of built-in functions
  - such as *document*, which returns the root node of a named document
- Quilt allows users to define functions of their own
- Each function definition must declare the types of its parameters and result
- A function may be defined recursively

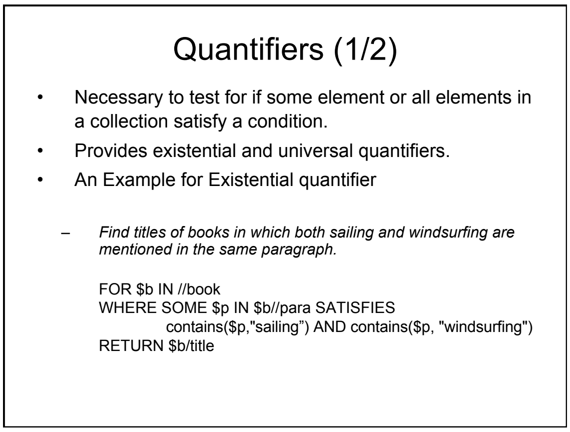


Functions contd

- An example of a user-defined recursive function
- Compute the maximum depth of the document named "partlist.xml."

```
FUNCTION depth($e ELEMENT) RETURNS integer
{
-- A leaf element has depth 1
-- Otherwise, add 1 to max depth of children
IF empty($e/*) THEN 1
ELSE max(depth($e/*)) + 1
}
```

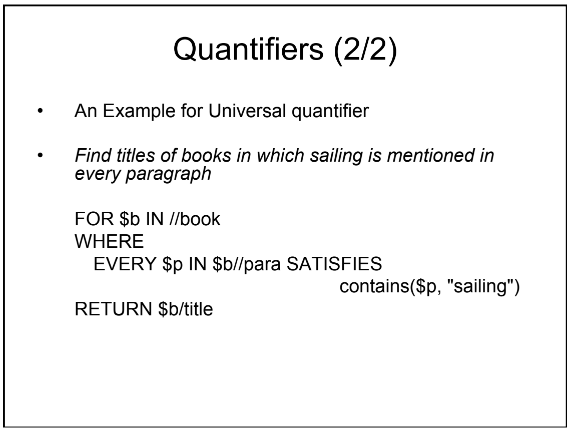
USE: depth(document("partlist.xml"))



Quantifiers (1/2)

- Necessary to test for if some element or all elements in a collection satisfy a condition.
- Provides existential and universal quantifiers.
- An Example for Existential quantifier
  - *Find titles of books in which both sailing and windsurfing are mentioned in the same paragraph.*

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
          contains($p,"sailing”) AND contains($p, "windsurfing")
RETURN $b/title
```



Quantifiers (2/2)

- An Example for Universal quantifier
- *Find titles of books in which sailing is mentioned in every paragraph*

```
FOR $b IN //book
WHERE
   EVERY $p IN $b//para SATISFIES
                              contains($p, "sailing")
RETURN $b/title
```

Variable Bindings

- To bind the value of the expression to a variable so that the definition of the expression does not need to be repeated.
  - – Defined by the word EVAL

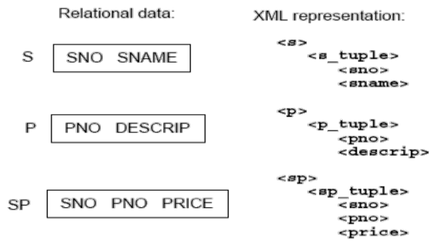
Example of EVAL

For each book whose price is greater than the average price, return the title of the book and the amount by which the book's price exceeds the average price.

```
LET $a := avg(//book/price)
EVAL
<result>
        FOR $b IN /book
        WHERE $b/price > $a
        RETURN
            <expensive_book>
                        $b/title,
                        <price_difference>
                                    $b/price - $a
                        </price_difference>
            </expensive_book>
</result>
```

Querying Relational Data

Consider the following schema



Relational data:

S | SNO SNAME

P | PNO DESCRIP

SP | SNO PNO PRICE

XML representation:

```
<s>
   <s_tuple>
      <sno>
      <sname>
<p>
   <p_tuple>
      <pno>
      <descrip>
<sp>
   <sp_tuple>
      <sno>
      <pno>
      <price>
```

A Simple Query

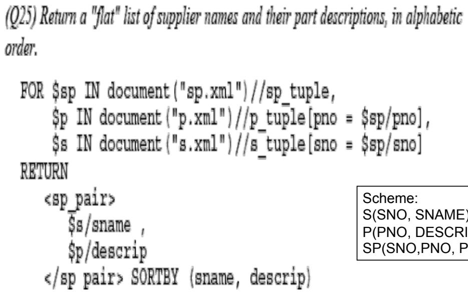
(Q23) Find part numbers of gears, in numeric order.

```
SQL:   SELECT pno
       FROM p
       WHERE descrip LIKE 'Gear'
       ORDER BY pno;
```

Scheme:
S(SNO, SNAME)
P(PNO, DESCRIP)
SP(SNO,PNO, PRICE)

```
Quilt: <gears-pNos>
           FOR $p IN document("p.xml")//p_tuple
           WHERE contains($p/descrip, "Gear")
           RETURN $p/pno SORTBY(.)
       </gears-pNos>
```

Joins

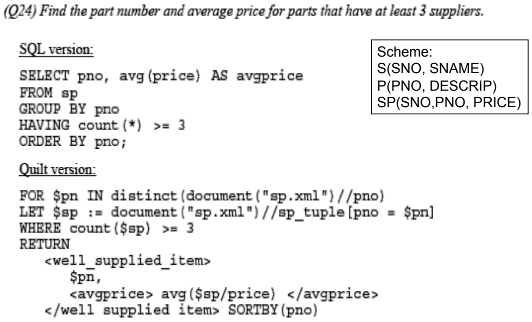


(Q25) Return a "flat" list of supplier names and their part descriptions, in alphabetic order.

```
FOR $sp IN document("sp.xml")//sp_tuple,
    $p IN document("p.xml")//p_tuple[pno = $sp/pno],
    $s IN document("s.xml")//s_tuple[sno = $sp/sno]
RETURN
   <sp_pair>
      $s/sname ,
      $p/descrip
   </sp pair> SORTBY (sname, descrip)
```

Scheme:
S(SNO, SNAME)
P(PNO, DESCRIP)
SP(SNO,PNO, PRICE)

Grouping



(Q24) Find the part number and average price for parts that have at least 3 suppliers.

SQL version:

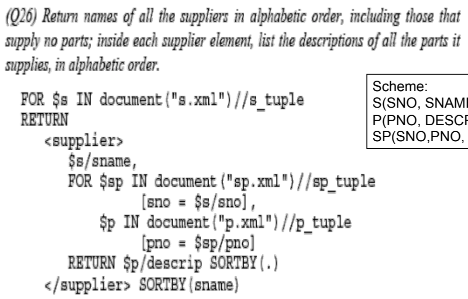
```
SELECT pno, avg(price) AS avgprice
FROM sp
GROUP BY pno
HAVING count(*) >= 3
ORDER BY pno;
```

Quilt version:

```
FOR $pn IN distinct(document("sp.xml")//pno)
LET $sp := document("sp.xml")//sp_tuple[pno = $pn]
WHERE count($sp) >= 3
RETURN
   <well_supplied_item>
      $pn,
      <avgprice> avg($sp/price) </avgprice>
   </well supplied item> SORTBY(pno)
```

Scheme:
S(SNO, SNAME)
P(PNO, DESCRIP)
SP(SNO,PNO, PRICE)

Left-outer join

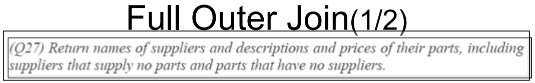


(Q26) Return names of all the suppliers in alphabetic order, including those that supply no parts; inside each supplier element, list the descriptions of all the parts it supplies, in alphabetic order.

```
FOR $s IN document("s.xml")//s_tuple
RETURN
   <supplier>
      $s/sname,
      FOR $sp IN document("sp.xml")//sp_tuple
               [sno = $s/sno],
          $p IN document("p.xml")//p_tuple
               [pno = $sp/pno]
      RETURN $p/descrip SORTBY(.)
   </supplier> SORTBY(sname)
```

Scheme:
S(SNO, SNAME)
P(PNO, DESCRIP)
SP(SNO,PNO, PRICE)

Full Outer Join(1/2)

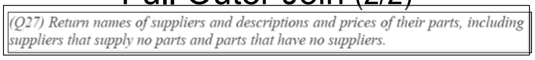


(Q27) Return names of suppliers and descriptions and prices of their parts, including suppliers that supply no parts and parts that have no suppliers.

```
<master_list>
        (FOR $s IN document("s.xml")//s_tuple
        RETURN
                <supplier>
                    $s/sname,
                    FOR $sp IN document("sp.xml")//sp_tuple[sno = $s/sno],
                        $p IN document("p.xml")//p_tuple [pno = $sp/pno]
                        RETURN
                        <part>
                            $p/descrip,
                            $sp/price
                        </part> SORTBY (descrip)
                 </supplier> SORTBY(sname)
        ) UNION …
```

Scheme:
S(SNO, SNAME)
P(PNO, DESCRIP)
SP(SNO,PNO, PRICE)

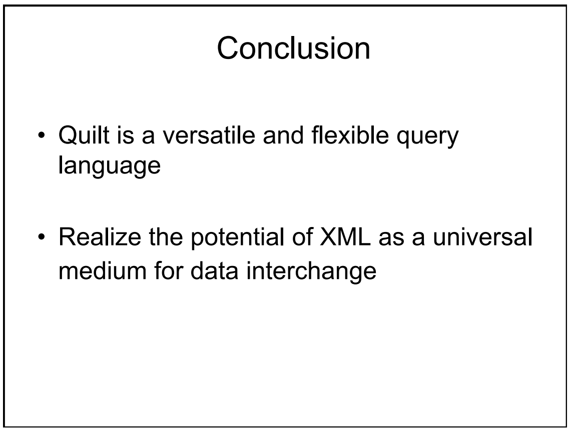
Full Outer Join (2/2)



(Q27) Return names of suppliers and descriptions and prices of their parts, including suppliers that supply no parts and parts that have no suppliers.

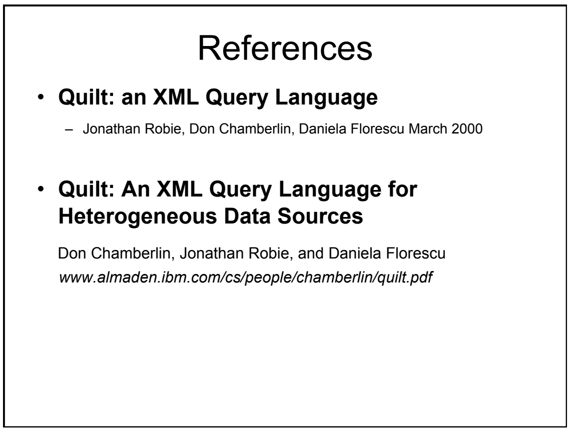
```
        -- parts that have no supplier
        <orphan_parts>
                FOR $p IN document("p.xml")//p_tuple
                WHERE empty(document("sp.xml")//sp_tuple[pno = $p/pno] )
                RETURN $p/descrip SORTBY(.)
        </orphan_parts>
</master_list>
```

Scheme:
S(SNO, SNAME)
P(PNO, DESCRIP)
SP(SNO,PNO, PRICE)



Conclusion

- Quilt is a versatile and flexible query language
- Realize the potential of XML as a universal medium for data interchange



References

- **Quilt: an XML Query Language**
  - Jonathan Robie, Don Chamberlin, Daniela Florescu March 2000
- **Quilt: An XML Query Language for Heterogeneous Data Sources**

  Don Chamberlin, Jonathan Robie, and Daniela Florescu
  www.almaden.ibm.com/cs/people/chamberlin/quilt.pdf