

# Semistructured Data, Chapter 31



# Semistructured Data – Example

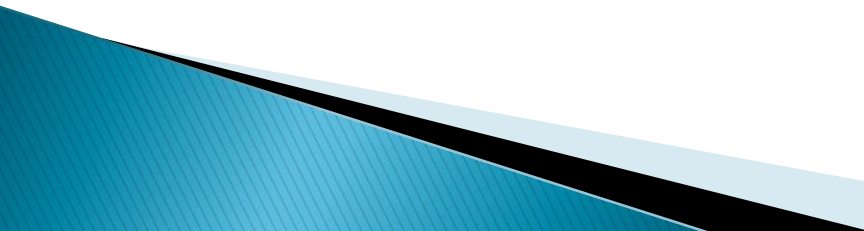
(pg. 1057)

```
DreamHome (&1)
  Branch (&2)
    street (&7) "22 Deer Rd"
    Manager &3
  Staff (&3)
    name (&8) -
      fName (&17) "John"
      lName (&18) "White"
    ManagerOf &2
  Staff (&4)
    name (&9) "Ann Beech"
    salary (&10) 12000
    Oversees &5
    Oversees &6
  PropertyForRent (&5)
    street (&11) "2 Manor Rd"
    type (&12) "Flat"
    monthlyRent (&13) 375
    OverseenBy &4
  PropertyForRent (&6)
    street (&14) "18 Dale Rd"
    type (&15) 1
    annualRent (&16) 7200
    OverseenBy &4
```

**Figure 31.1** Sample representation of semistructured data in the *DreamHome* database.

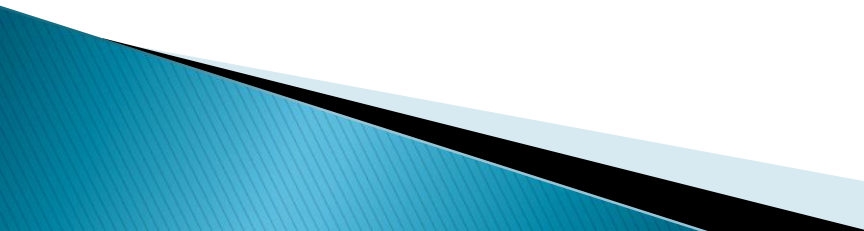
# Semistructured Data – Example

Example shows:

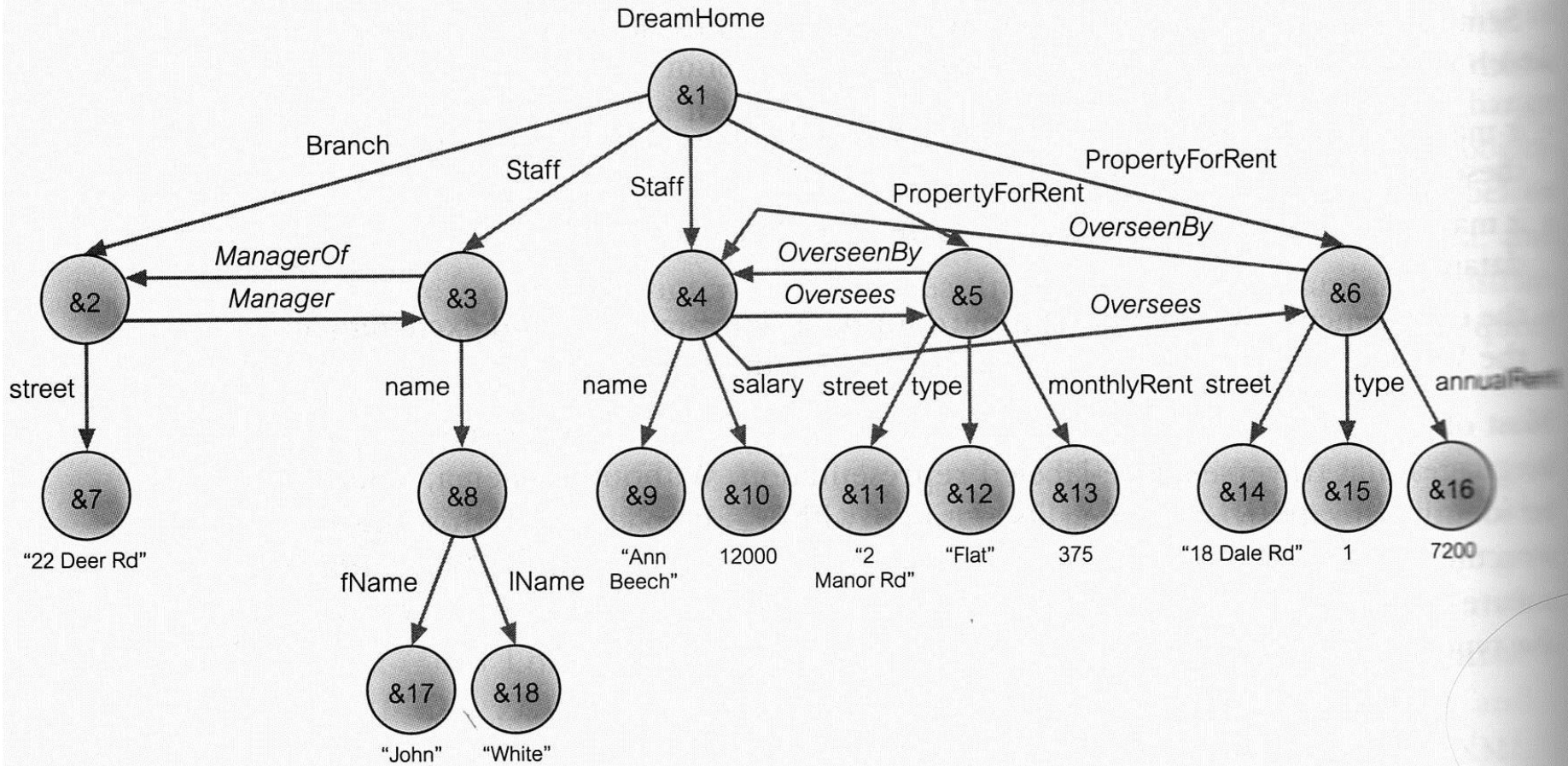
- ▶ 1 branch office (22 Deer Road)
  - ▶ 2 members of staff (John White and Ann Beech)
  - ▶ 2 pieces of properties for rent (2 Manor Rd and 18 Dale Rd)
  - ▶ Some relations amongst those above
- 

# Semistructured Data – Example

Irregularities include:

- ▶ First and last name of John White, Ann Beech name is a single component and has a salary
  - ▶ Monthly rent for property at 2 Manor Rd, 18 Dale Rd has annual rent;
  - ▶ Property type for 2 Manor Rd (flat), whereas 18 Dale Rd has the type (house) as an integer value
- 

# Semistructured Data – Graphical View of Example (pg. 1058)



**Figure 31.2** A graphical representation of the data shown in Figure 31.1.

# XML Representation of Staff Information (pg. 1066)

**Figure 31.5**

Example XML to represent staff information.

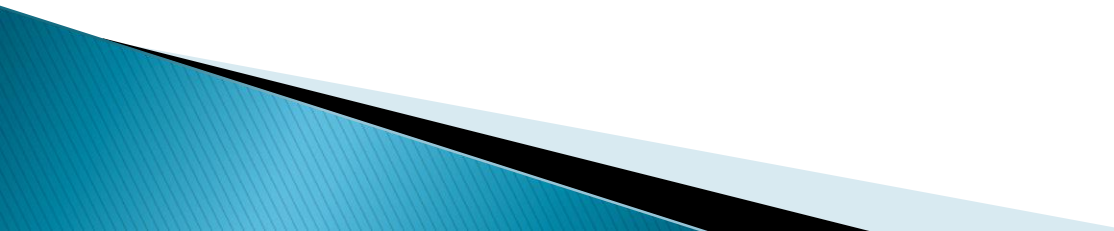
```
<?xml version= "1.1" encoding= "UTF-8" standalone= "no"?>
<?xml:stylesheet type = "text/xsl" href = "staff_list.xsl"?>
<!DOCTYPE STAFFLIST SYSTEM "staff_list.dtd">
<STAFFLIST>
  <STAFF branchNo = "B005">
    <STAFFNO>SL21</STAFFNO>
    <NAME>
      <FNAME>John</FNAME><LNAME>White</LNAME>
    </NAME>
    <POSITION>Manager</POSITION>
    <DOB>1945-10-01</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>
```

# XML versus JSON

XML	JSON
Markup Language	JavaScript Object Notation
<pre>&lt;?xml version="1.0"?&gt; &lt;book id="123"&gt;   &lt;title&gt;Object Thinking&lt;/title&gt;   &lt;author&gt;David West&lt;/author&gt;   &lt;published&gt;     &lt;by&gt;Microsoft Press&lt;/by&gt;     &lt;year&gt;2004&lt;/year&gt;   &lt;/published&gt; &lt;/book&gt;</pre>	<pre>{   "id": 123,   "title": "Object Thinking",   "author": "David West",   "published": {     "by": "Microsoft Press",     "year": 2004   } }</pre>
Extensible	Meant to fit naturally with code, so limited and lighter

# XML Query Languages (pg. 1091)

Many languages proposed:

- ▶ XML-QL (example on next slide)
  - ▶ UnQL
  - ▶ XQL by Microsoft
  - ▶ XML Query, XQuery by W3C (next examples)
- 



# Example XML Query Language

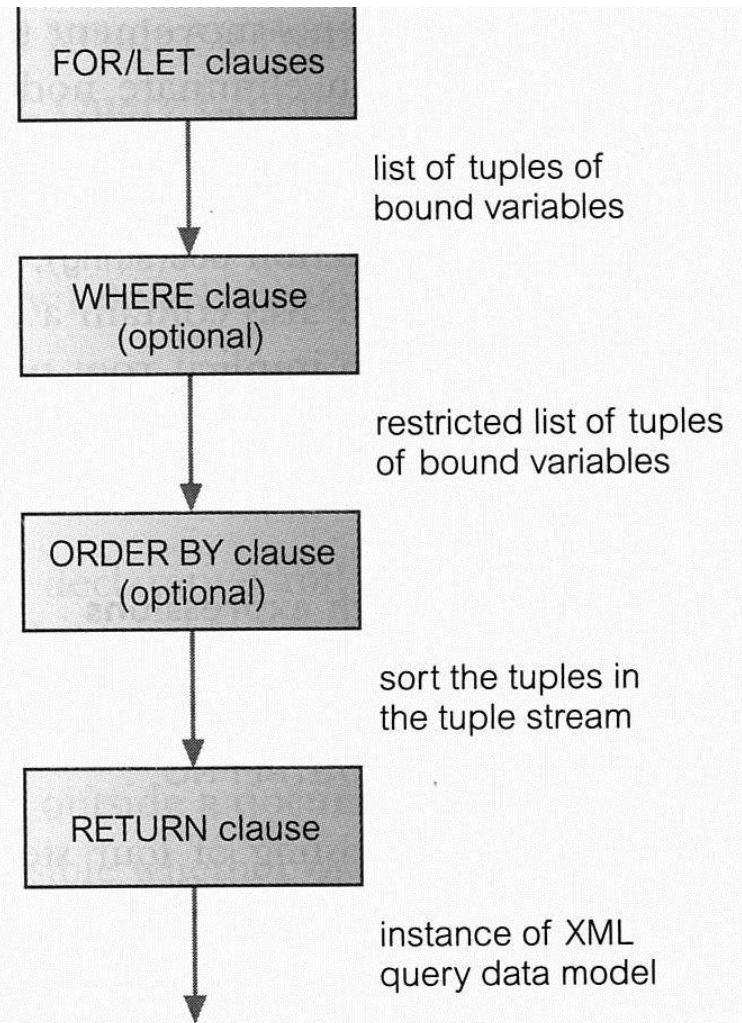
Query in XML-QL which gets the surnames of staff who earn more than 30,000.

```
WHERE <STAFF>  
  <SALARY>$S</SALARY>  
  <NAME><FNAME>$F</FNAME> <LNAME>$L</LNAME> </NAME>  
</STAFF> IN "http://www.dreamhome.co.uk/staff.xml"  
  $S > 30000  
CONSTRUCT <LNAME>$L</LNAME>
```

# Flow of data in FLWOR (pg. 1096)

**Figure 31.17**

Flow of data in a FLWOR expression.



# FLWOR expressions on Xquery

## (pg. 1095)

### **FLWOR expressions**

A FLWOR (pronounced “flower”) expression is constructed from FOR, LET, WHERE, ORDER BY, and RETURN clauses. The syntax of a FLWOR expression is:

<b>FOR</b>	forVar <b>IN</b> inExpression
<b>LET</b>	letVar := letExpression
<b>[WHERE</b>	filterExpression]
<b>[ORDER BY</b>	orderSpec]
<b>RETURN</b>	expression

# FLWOR expressions on Xquery (pg. 1097)

*(1) List staff with a salary of £30,000.*

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

This is a simple extension of a path expression with a variable used to represent the value of the salary we wish to restrict. For the XML document of Figure 31.5, only one STAFF element satisfies this predicate, so the result of this query is:

```
<STAFF branchNo = "B005">  
  <STAFFNO>SL21</STAFFNO>  
  <NAME>  
    <FNAME>John</FNAME> <LNAME>White</LNAME>  
  </NAME>  
  <POSITION>Manager</POSITION>  
  <DOB>1945-10-01</DOB>  
  <SALARY>30000</SALARY>  
</STAFF>
```

# FLWOR expressions on Xquery (pg. 1098)

(2) *List the staff at branch B005 with a salary greater than £15,000.*

```
FOR $S IN doc("staff_list.xml")//STAFF  
WHERE $S/SALARY > 15000 AND $S/@branchNo = "B005"  
RETURN $S/STAFFNO
```

In this example we have used a FOR clause to iterate over the STAFF elements in the document and, for each one, to test the SALARY element and branchNo attribute. The result of this query is:

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

# FLWOR expressions on Xquery

## (pg. 1098)

*(3) List all staff, ordered in descending order of staff number.*

```
FOR $S IN doc("staff_list.xml")//STAFF  
ORDER BY $S/STAFFNO DESCENDING  
RETURN $S/STAFFNO
```

This query uses the ORDER BY clause to provide the required ordering. The result of this query is:

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

# FLWOR expressions on Xquery (pg. 1098)

*4) List each branch office and the average salary at the branch.*

```
FOR $B IN distinct-values(doc("staff_list.xml")//@branchNo)
LET $avgSalary := avg(doc("staff_list.xml")//STAFF[@branchNo = $B]/SALARY)
RETURN
  <BRANCH>
    <BRANCHNO> {$B/text()} </BRANCHNO>
    <AVGSALARY> $avgSalary </AVGSALARY>
  </BRANCH>
```

This example demonstrates the use of the built-in function `distinct-values()` to generate a set of unique branch numbers and how element constructors can be used within the `RETURN` clause. It also shows the use of an aggregate function applied to the `SALARY` elements to calculate the average salary at a given branch. As we noted in the first example, atomization is used to extract the typed value of the `SALARY` elements to compute the average.

# Advantages of XML (pg. 1065)

**TABLE 31.1** Advantages of XML.

Simplicity

Open standard and platform-/vendor-independent

Extensibility

Reuse

Separation of content and presentation

Improved load balancing

Support for the integration of data from multiple sources

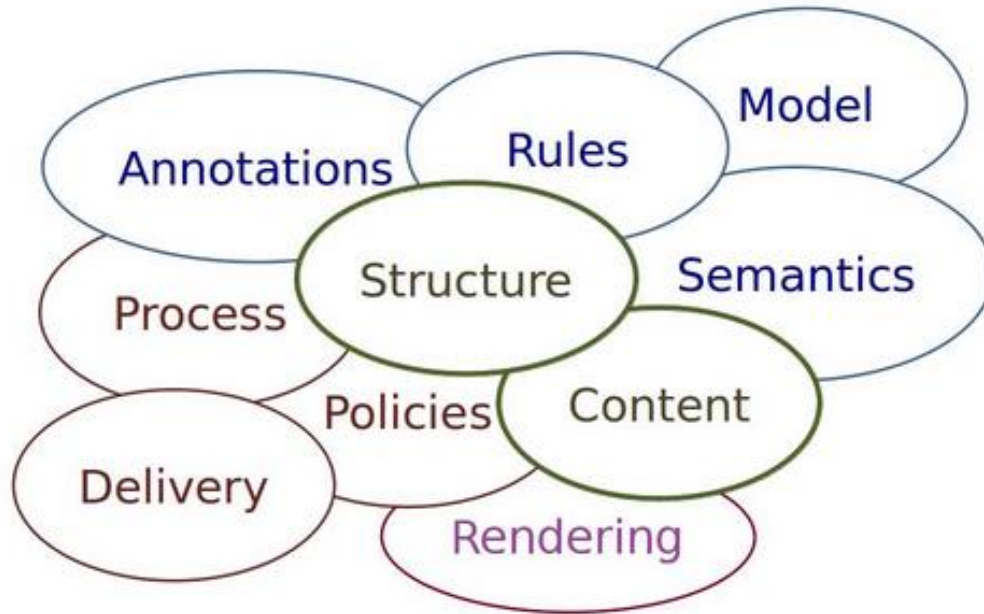
Ability to describe data from a wide variety of applications

More advanced search engines

New opportunities



# XML comparison with JSON



JSON handles structure and context, whereas XML handles all

[https://blogs.oracle.com/xmlorb/entry/analysis\\_of\\_json\\_use\\_cases](https://blogs.oracle.com/xmlorb/entry/analysis_of_json_use_cases)