9. Semi-structured data and XML

Slides adapted from Pearson Ed.

Well-structured data

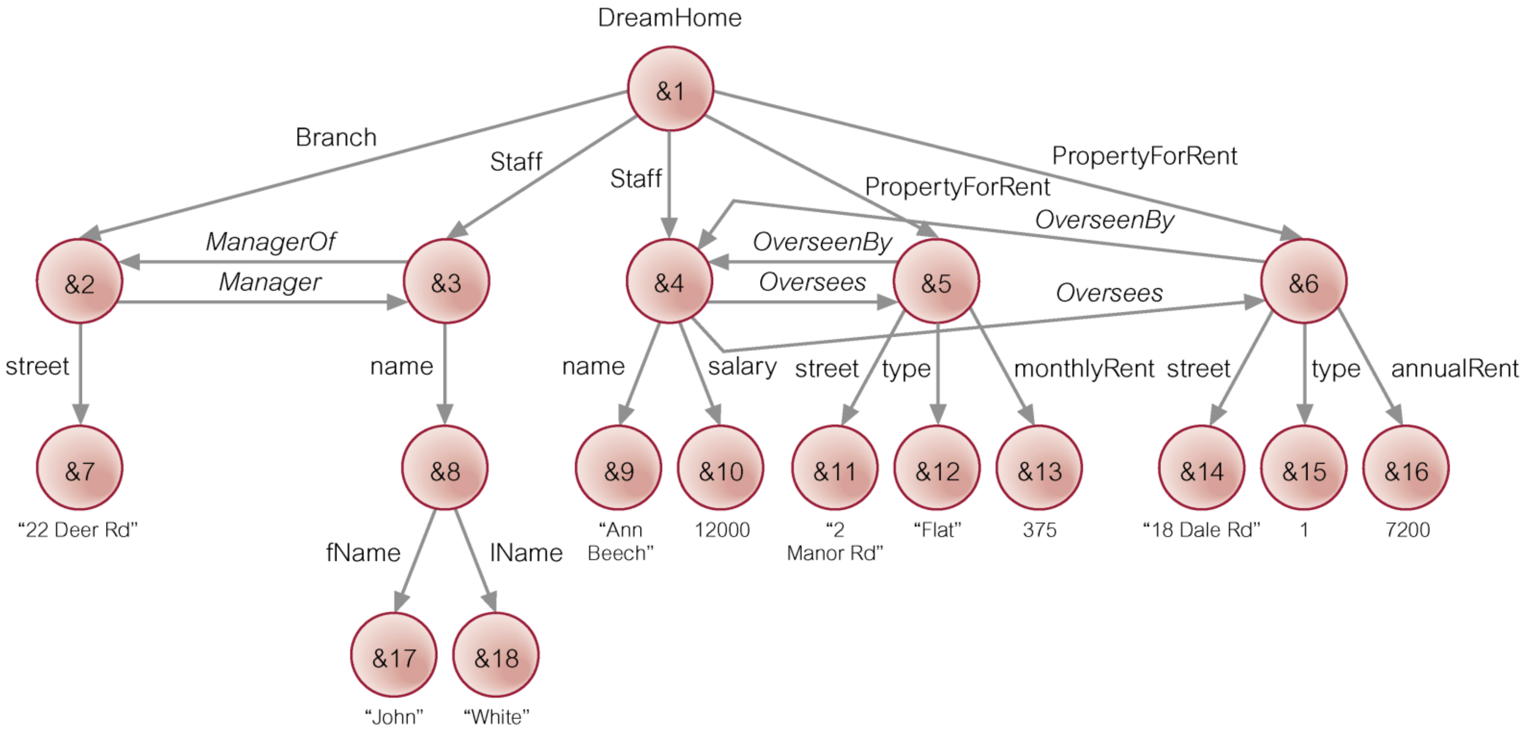
- Relational databases are best suited for well-structured and simple-structured data where all data of one type conforms to one structure.
  - Structure is known in advance.
  - Structure is not expected to change.
  - Data can be broken down into a finite number of rectangular tables.

Semi-structured data

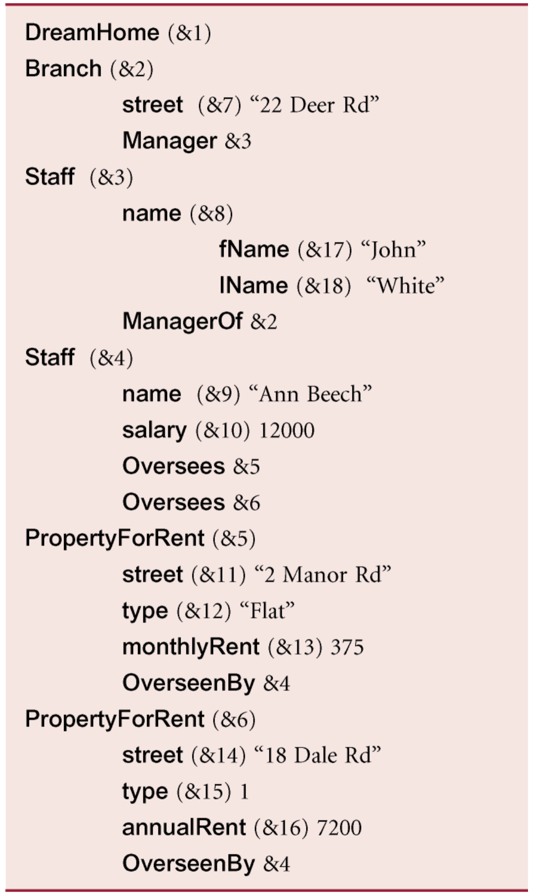
- Semi-structured, schema-less, or self-describing data: has structure, but that structure may change unpredictably, be irregular, or be incomplete.
- Schema-less / self-describing: no schema exists; the form the data comes in itself defines the schema.
  - OR a schema exists but allows for large range in variation.

Object Exchange Model (OEM)

- Early semi-structured database model.
  - Tree structure (nesting).
  - Objects have a label (“DreamHome”), a unique ID (&1), and sometimes a value (“Ann Beech”).
  - Node-to-node relationships.



OEM as text



```
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
```

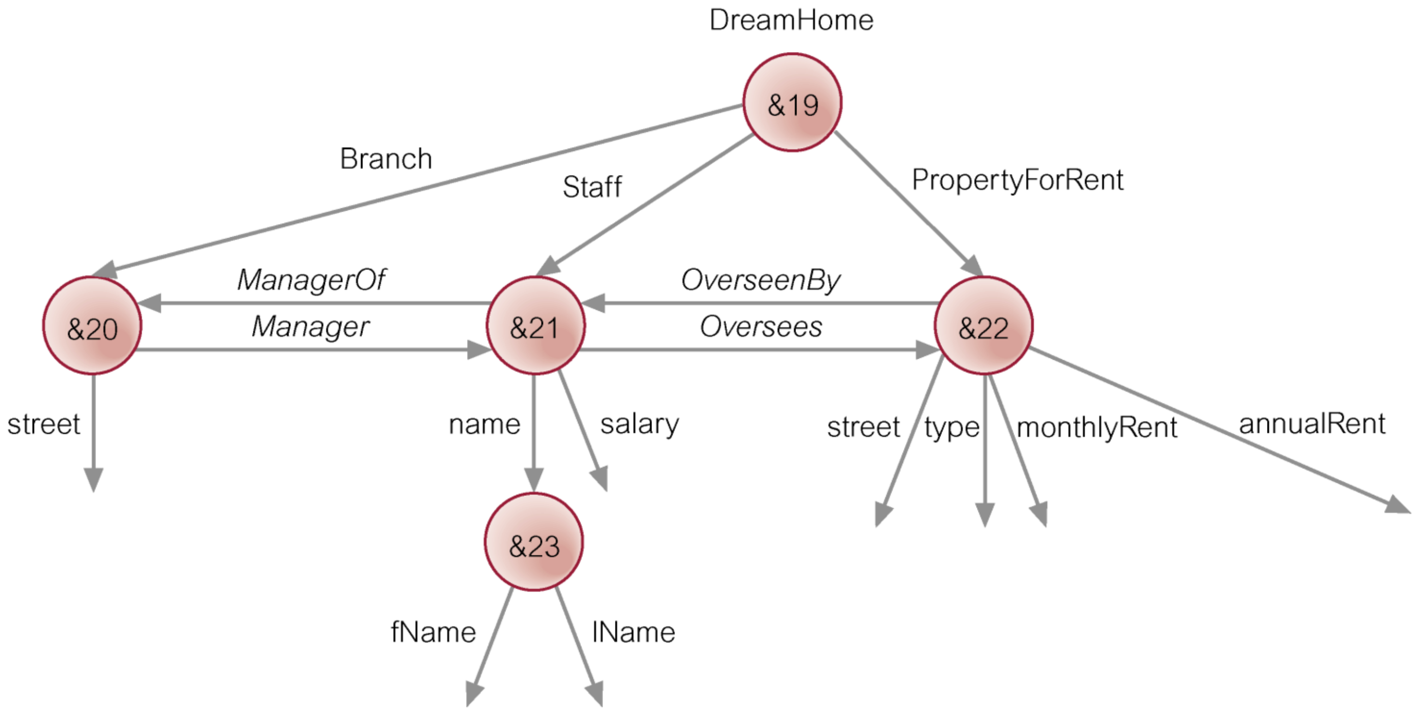
- Data is irregular:
  - Different format of name element.
  - Property can have monthly rent OR annual rent.
  - Property type can be stored as a string ("Flat") OR as a number (1).

Querying OEM database / files

- Objects can be accessed by node ID.
- Or, given nested/tree structure, by the path to the node.
- Lorel (Lightweight Object REpository Language): SQL-like query language using queries of the form:
  SELECT object/path [FROM path] [WHERE condition]
- Speed up queries by parsing the file and storing in memory an annotated representation of the data.

DataGuides

- Auxiliary data structure summarizing structure of OEM file/database.
- Annotated with IDs of nodes satisfying that path (not shown).
- Facilitates checking whether paths are valid and provides direct links to data.



Principles for semi-structured data

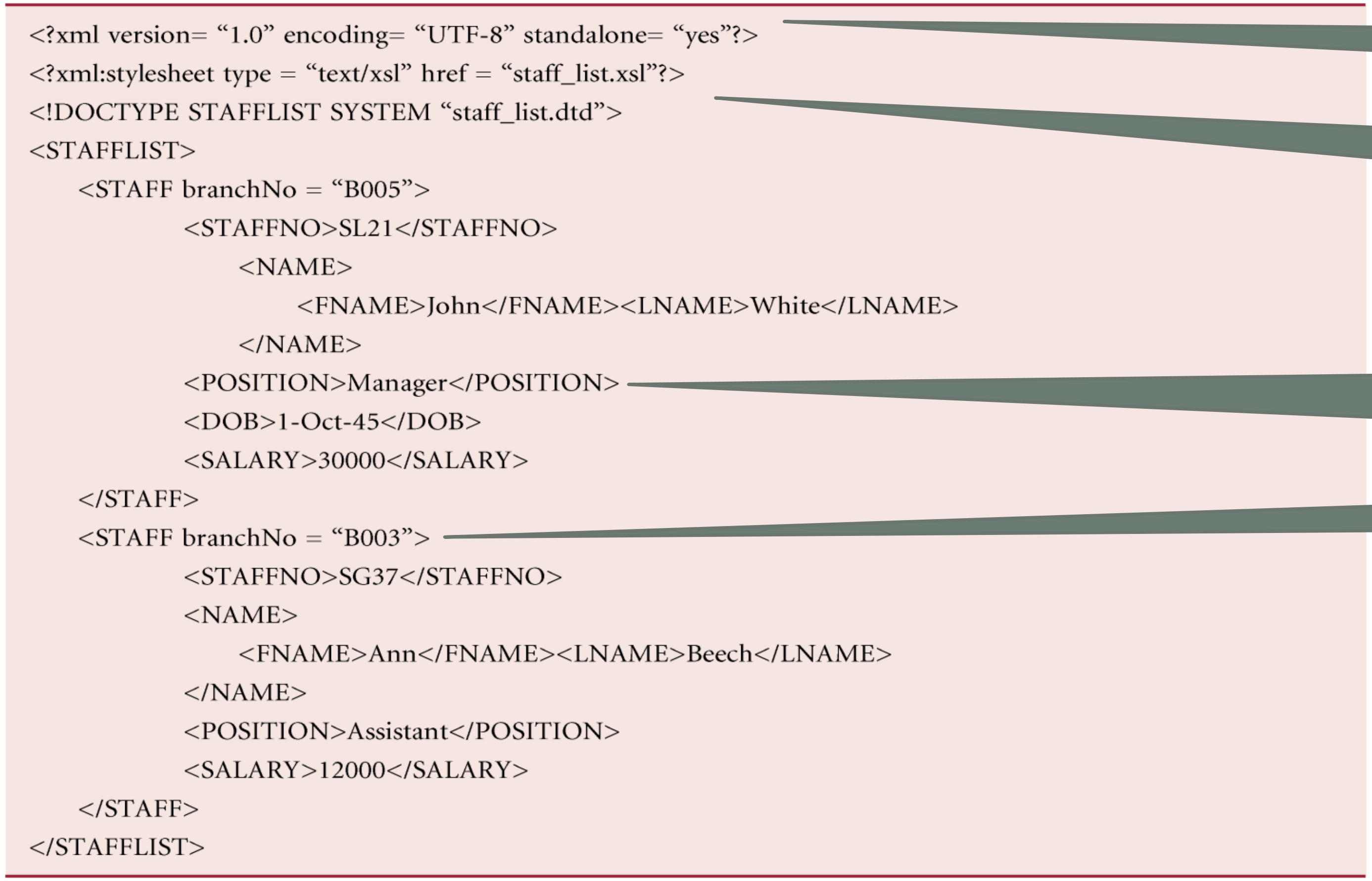
- Flexible, tree-like structure.
  - Querying is based on paths through the tree.
- Querying data efficiently requires a summary/understanding of its structure.
- Same general principles apply for XML.

XML

XML

- eXtensible Markup Language: a generalization of HTML that allows people to define their own tags and document structure.
- Document structure defined using a Document Type Definition (DTD) file.
- Data stored in XML files that start with an <?xml> declaration and then (like HTML) tags with attributes, nested under a single root tag.

XML example



```
<?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”>
            <STAFFNO>SG37</STAFFNO>
            <NAME>
                <FNAME>Ann</FNAME><LNAME>Beech</LNAME>
            </NAME>
            <POSITION>Assistant</POSITION>
            <SALARY>12000</SALARY>
    </STAFF>
</STAFFLIST>
```

XML declaration

Stylesheet, document type definition

Elements with values between opening and closing tags

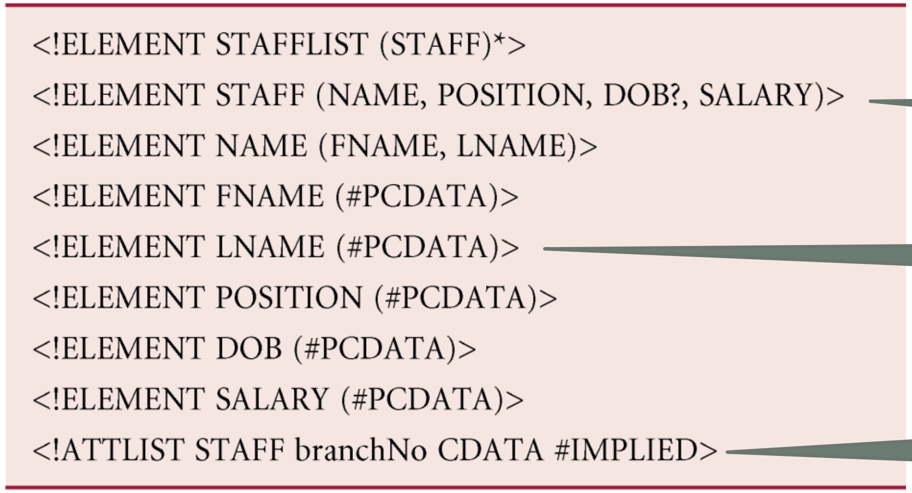
Element attribute

XML elements and attributes

- Elements / tags in angle brackets.
  - Order is important.
- Attributes: name-value pairs that contain descriptive information.
  - Appear on the inside of angle brackets, after element name.
  - Unordered.
  - Could alternatively choose to implement attributes as nested tags.
    - Only one attribute with a given name is allowed per tag, so if you need more than one, it *must* be implemented as a nested tag.

Schema specification with DTD

- Can optionally specify schema for a set of XML documents.
  - DTD is older, largely replaced by XML Schema.
- DTD: Specify elements and attributes, and number of elements that can be used/nested.



```
<!ELEMENT STAFFLIST (STAFF)*>
<!ELEMENT STAFF (NAME, POSITION, DOB?, SALARY)>
<!ELEMENT NAME (FNAME, LNAME)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT LNAME (#PCDATA)>
<!ELEMENT POSITION (#PCDATA)>
<!ELEMENT DOB (#PCDATA)>
<!ELEMENT SALARY (#PCDATA)>
<!ATTLIST STAFF branchNo CDATA #IMPLIED>
```

Ordered list of child elements allowed

Data type of value inside element / attribute

Attribute can be set to be required (#REQUIRED) or optional (#IMPLIED)

DTD: other features

- * = zero or more; + = one or more; ? = optional.
- Can specify that an attribute is functioning as an ID (identifier for this element) or IDREF(S) (link(s) to another element’s ID).
  - Can capture parent/child relationships *outside* of the nesting structure by having children refer to their parent’s ID.

XML Schema

- Updated way of specifying format of XML documents.
- Unlike DTD, XML Schema docs are themselves valid XML.
  - Different syntax:

```
<xs:element name=“STAFFLIST”>
  <xs:complexType>
    <xs:sequence>
      <xs:group ref=“STAFFTYPE” minOccurs=“0” maxOccurs=“unbounded” />
    </xs:sequence> </xs:complexType> </xs:element>
<xs:group name=“STAFFTYPE”> <xs:element name=“STAFF”>
    <xs:complexType> <xs:sequence>
      <xs:element name=“STAFFNO” type = “xs:string”/>
      <xs:attribute name=“POSITION” type = “xs:string”/>
      <xs:element name=“DOB” type = “xs:date”/>
      <xs:element name=“SALARY” type = “xs:decimal”/>
    </xs:sequence> </xs:complexType>
</xs:element> </xs:group>
```

STAFFLIST consisting of 0 or more STAFFTYPE elements

Custom definition of STAFFTYPE listing out allowed children

XML Schema: other features

- More types: more default types, and can also define and use custom types.
- Allows specifying uniqueness and primary key constraints.

Querying in XML

- Similar to OEM: tree structure allows for referring to nodes by path.
- Querying done with SQL-like syntax for XML (XQuery) that makes use of XML-style paths (XPath).

XPath notation

- Paths separated by slashes (/).
- @ used to refer to attributes.
- [] used to specify conditions.
  - [3] or [position()=3]: third element.
  - [@branchNo="B003"]: condition on attribute.

Notation	Meaning
doc("filename")	Opens an XML file
//	Selects descendants of current node ("by any path")
/PATH or /child::PATH	Selects children of current node with specified tag name
..	Selects parent of current node
@ or attribute::	Refers to an attribute of a node
[cond]	Use to apply conditions to the selection
*	Wildcard (selects all)

XPath example

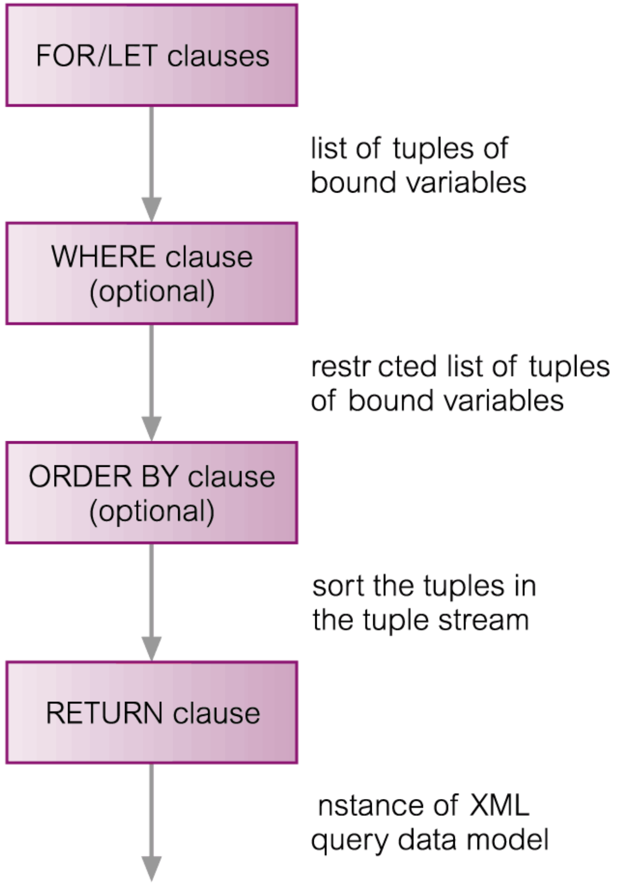
- Task: Find staff number of first member of staff in our XML document.
- doc(“staff_list.xml”)/STAFFLIST/STAFF[1]//STAFFNO
- Four steps:
  - First step opens staff_list.xml and returns its document node.
  - Second step uses /STAFFLIST to select STAFFLIST element at top.
  - Third step locates the first STAFF element that is the child of root element.
  - Fourth step finds STAFFNO elements occurring anywhere within this STAFF element.

Multiple equivalent ways of specifying the same data

- These paths also get the same data:
  - doc("staff_list.xml")//STAFF[1]/STAFFNO
  - doc("staff_list.xml")/STAFFLIST/STAFF[1]/STAFFNO

XQuery: FLWOR expressions

- FLWOR (“flower”) expression is constructed from FOR, LET, WHERE, ORDER BY, RETURN clauses.
- Minimal expression contains one FOR/LET clause and one RETURN clause.
- FOR and LET clauses bind values to one or more variables using expressions (e.g., path expressions).


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

XQuery example

- Task: List all staff at branch B005 with salary > £15,000.

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

XQuery example: nested FOR/LET and XML construction

- Task: List branches that have more than 20 staff.
-
```
<LARGEBRANCHES> {
   FOR $B IN distinct-values(doc(“staff_list.xml”)//@branchNo)
      LET $S := doc(“staff_list.xml”)//STAFF/[@branchNo = $B]
   WHERE count($S) > 20
   RETURN <BRANCHNO>{ $B/text() }</BRANCHNO>
 }
</LARGEBRANCHES>
```

Data model

- Similar to OEM, querying XML requires using an auxiliary data structure to speed up queries/update operations.
- Document Object Model (DOM) is an in-memory tree representation of the structure of an XML file.
  - Allows navigation and updates.
- XPath and XQuery data model (XDM) is the format for inputs to an XQuery engine.
  - Flattened tree structure: list of nodes in a standard document order.
  - Nodes contain parent/child information as well as attributes.

XML and relational databases

Multiple ways to make use of XML in a database

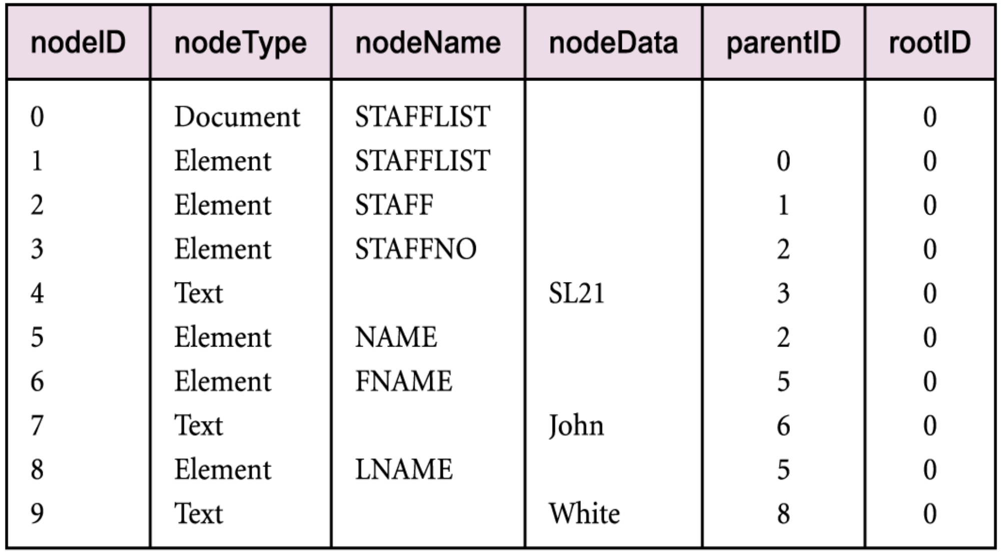
- **Data-centric XML model**: XML used as a data interchange format only, allowing diverse databases to communicate with each other by passing along files in XML format.
- **Document-centric XML model**: the database itself is stored in XML format – a native XML database (NXD).

Storing XML documents in a relational database

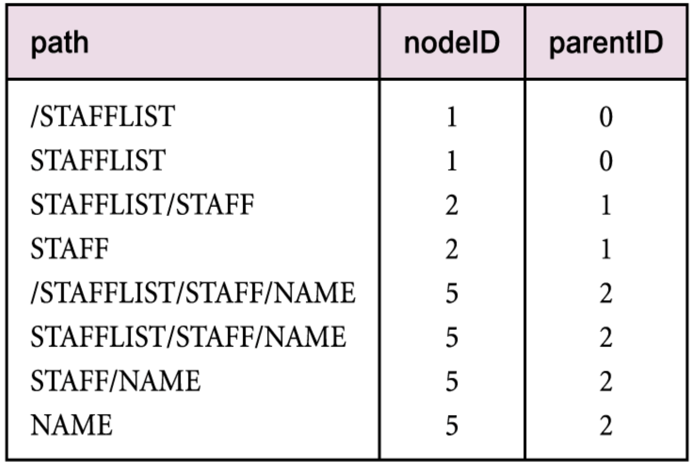
- Several options:
  - Store as a single attribute in a table as a CLOB (character large object) or newer XMLType data.
  - Use the structure of the XML to decide how to distribute (shred) XML data across multiple attributes/tuples/tables as necessary.
  - Convert XML to a generic form that works for all schemas.

XML in a schema-independent form

- Possible to parse XML file into tree structure and convert that tree into a relational table.



nodeID	nodeType	nodeName	nodeData	parentID	rootID
0	Document	STAFFLIST			0
1	Element	STAFFLIST		0	0
2	Element	STAFF		1	0
3	Element	STAFFNO		2	0
4	Text		SL21	3	0
5	Element	NAME		2	0
6	Element	FNAME		5	0
7	Text		John	6	0
8	Element	LNAME		5	0
9	Text		White	8	0



path	nodeID	parentID
/STAFFLIST	1	0
STAFFLIST	1	0
STAFFLIST/STAFF	2	1
STAFF	2	1
/STAFFLIST/STAFF/NAME	5	2
STAFFLIST/STAFF/NAME	5	2
STAFF/NAME	5	2
NAME	5	2

Other semistructured data formats

JSON (JavaScript Object Notation)

- Derived from JavaScript in 2000s but used as a data exchange format by multiple programming languages.
- Main data structures:
  - Objects: list of key-value pairs enclosed in curly brackets {}.
  - Arrays: comma-separated, ordered list of data of non-uniform type, enclosed in square brackets [].
  - Primitive types: numbers, strings, Boolean, null.
- JSON specification is lightweight: https://www.json.org/json-en.html

XML vs. JSON

- JSON much more compact.
  - Can be preferable as a data interchange format.
- Flexible nested arrays and objects allows for similar functionality.
- Structure can change unpredictably.
  - XML has more facilities for pre-defining schemas.

```
{
 staff: [
  {
   branchNo: “B005”
   staffNo: “SL21”,
   name: { fname: “John”, lname: “White” },
   position: “Manager”,
   DOB: “1-Oct-45”,
   salary: 30000
  },
  {
   branchNo: “B003”,
   name: { fname: “Ann”, lname: “Beech” },
   position: “Assistant”,
   salary: 12000
  }
 ]
}
```

Semi-structured data vs. relational data: summary

- More complicated data structures possible.
  - Nested objects, list data nested within single entities vs. regular tables with workarounds to represent multi-valued data, parent-child relationships, etc.
- Structure is understood by reading the data.
  - Changing the file directly changes structure.
- More complicated structure makes querying also more complicated.
  - Requires tree traversal via paths.