universität wien

Data Engineering

Chapter 1-3

VU Information Management & Systems Engineering

Peter Kalchgruber

MapReduce

A programming model and an associated implementation for processing and generating large data sets.

Dean, Jeffrey, and Sanjay Ghemawat

MapReduce Framework for Large-Scale Data Processing

- Simplifying Large-Scale Data Processing
- Harnessing the power of multiple CPUs for efficient analysis
- Distributing work across computing sites for scalability
- Built-in fault tolerance for resilient computations

- Key Features
  - 3 phases: Map, Shuffle, Reduce
  - Automatic parallelization and distribution of work
  - I/O Scheduling
  - Example "The Story of Sam"

MapReduce: The Essence of Divide and Conquer

MapReduce uses **key-value pair** approach

Key: A unique identifier for each data element

Value: The actual data or a reference to its location.

Map Function: Processing Individual Elements

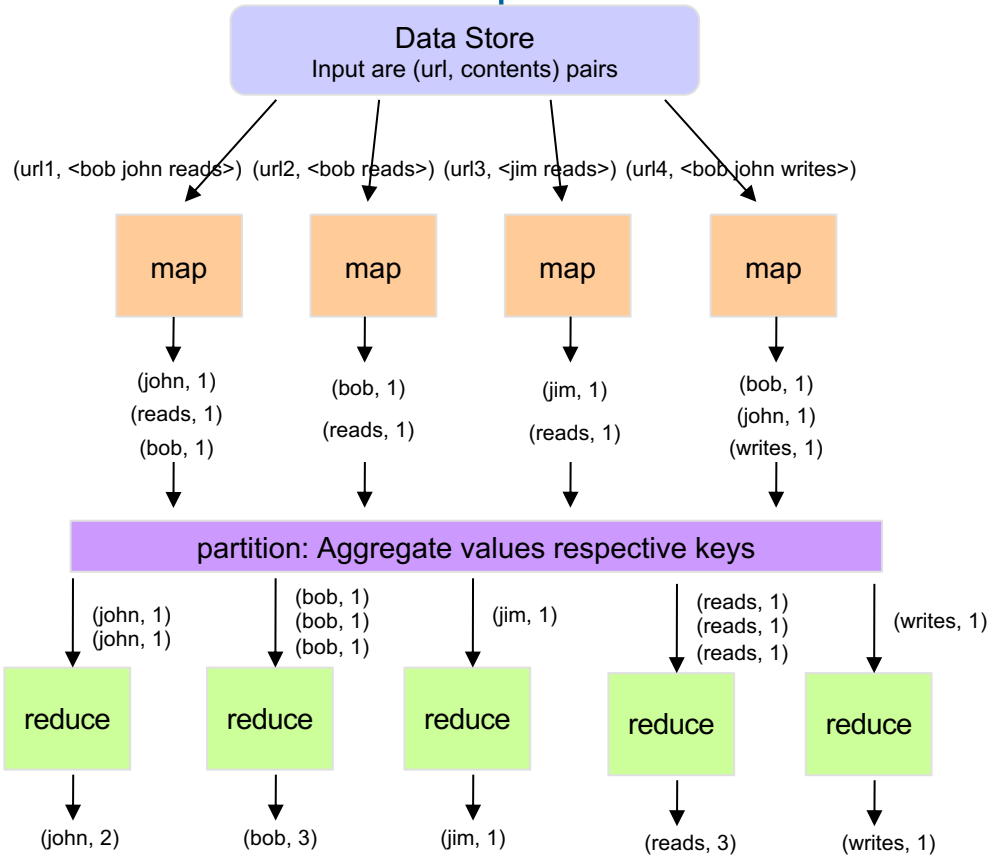
Executes on every dataset element, transforming into a new key-value pair.

Partition: Organizing Data

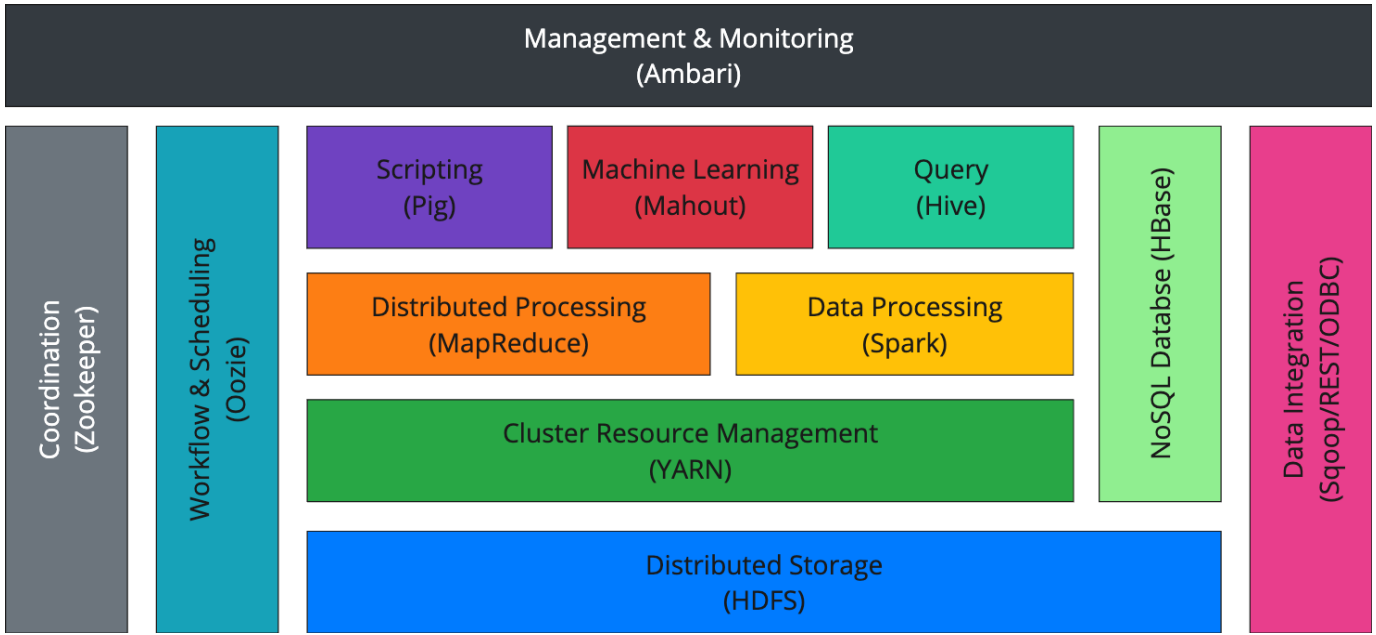
Shards the key-value pairs, grouping them based on the key for efficient processing.

Reduce Function: Aggregating Results

Operates on each unique key, aggregating or summarizing associated values.



Hadoop Ecosystem



MapReduce Framework: Hadoop Ecosystem

Apache Foundation Java project since 2008

- **HDFS**: distributed file system (similar to Google FS)
  - Error tolerance by 3-times (default) data replication
  - If no "heartbeat" of a node => the central node re-distributes data
  - Files are stored in chunks of fixed size (64 MB) => reasonable number of large files
  - Rack-aware file system => nobody knows where data is stored
  - Is not directly mountable by an operating system
- **Hadoop Map Reduce**: Parallel Programming Framework
- **HBase**: NoSQL database modeled after Google BigTable
- **YARN** (Yet another resource negotiator)

Semi-Structured Data

The information that is normally associated with a schema is contained within the data, which is sometimes called "self-describing".

Buneman, Peter

Limitations of Traditional Data Formats

- EDIFACT
- CSV / TSV
- Fixed-Width Text Files
- Binary Formats
- Proprietary Formats

Example: Order encoded in EDIFACT

```
UNB+UNOA:1+US::EE+50138::THEM+140505:0305+001934++ORDERS'
UNH+1+ORDERS:91:2:UN'
BGM+220+A761902+4:20140501:102+9'
RFF+CT:EUA01349'
RFF+AAV::C'
TXT+THIS IS WHAT AN EDI MESSAGE WOULD LOOK LIKE... '
NAD+BY++OUR NAME PLC::::+++++EW4 34J'
CTA+PD’
COM+01752 253939:TE+01752 253939:FX+0:TL'
CTA+OC+:A.SURNAME'
COM+2407:EX'
CTA+TI+:B.BROWN'
COM+0:EX'
CTA+SU'
COM+0161 4297476:TE+01752 670633:FX'
UNT+15+1'
UNZ+1+001934'
```

Characteristics of Semi-Structured Data

- Self-Describing: Metadata included
- Flexible Schema: Adaptable to changes
- Hierarchical Structure: Nested elements
- Inhomogeneous Structure: Varied formats
- Implicit Schema: Structured suggested, not enforced
- Graph-Like Model: Interconnected nodes
- Platform-Independent: Universally accessible
- Human and Machine-Readable: Easily processed and understood

Semi-Structured Data (Examples)

XML

```
<contact>
<name>Alice</name>
<email>
  a@ex.com
</email>
<phone>123-45</phone>
</contact>
```

JSON

```
{
 "name": "Alice",
 "email": "a@ex.com",
 "phone": "123-45"
}
```

RDF

```
@prefix c:
<http://ex.com/contact#> .
c:Alice a c:Person ;
 c:name "Alice" ;
 c:email " a@ex.com" ;
 c:phone "123-45" .
```

YAML

```
name: Alice
email: a@ex.com
phone: 123-45
```

EXtensible Markup Language

A markup language designed for storage, transmission, and reconstruction of data.

eXtensible Markup Language

- Standardized Data Interchange Format
- XML became W3C Recommendation 1998
- Tag-based Syntax
- Foundation of several web technologies
  - XHTML
  - RSS/ATOM
  - AJAX (the X in AJAX)

It takes ten minutes to understand (base) XML, and then ten month to understand the new technologies hung around it.

Peter Chen

eXtensible Markup Language

Factors that Led to the Rise of XML

- Flexibility and Simplicity: XML balances structure and ease of use compared to SGML or highly rigid formats like EDI.
- Human and Machine Readability: XML is relatively easy for humans and computers to understand.
- Web Compatibility: XML's integration with web technologies promoted its widespread adoption.

Goals of XML

- Human Readable: Understandable without specialized tools.
- Data Sharing: Simplifies data sharing across platforms and applications.
- SGML-Compatible: Maintains compatibility with its parent, SGML.
- Ease of Processing: Programmatically parsable with standard libraries.
- Support Diverse Applications: Adaptable to various use cases.

Document-centric XML vs. Data-centric XML

Document-Centric XML

- Focuses on representing the layout and formatting of a document.
- Often used for human-readable content like reports, articles, or ebooks.
- May contain large text sections with some embedded tags for structure or styling.

Example: A research paper with sections, paragraphs, and citations marked up using XML tags.

Data-Centric XML

- Focuses on representing the data itself in a structured way.
- Often used for machine-readable information exchange like invoices, purchase orders, or scientific data.
- Highly structured with well-defined elements and attributes containing specific data points.

Example: An invoice with elements for items, quantities, prices, and total amount.

Disadvantages of XML

- Not suitable for very large datasets (multiple MB of data)
- Images are not represented well
- XML can quickly become difficult to read when complex

Usecases

- RSS Feeds
- SOAP Protocols
- APIs (e.g., Google)
- Weather Services
- Healthcare Data Exchange
- Financial Transactions
- Microsoft Office

Types of XML Content

- XML Document Declaration
- Elements and attributes
- Comments
- Character Data
- Processing Instructions
- Entity References
- Namespace

XML Document Declaration

- Optional at the beginning of XML document
- Specifies XML version and character encoding

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Elements

- Primary building block
- Must have valid names
- Start tag and end tag
- Can be nested
- Must be properly closed

Valid:

```
<_card>
<Card.Info>
<Card-Info_box>
```

Invalid

```
<xML_book>
<.tag>
<Card Header>
<1Header>
<Address:Street>
```

Attributes and Text

Attributes

- Additional information about elements
- Defined within start tag of an element
- Name-value pairs
- Must have valid names
- Appear only once on a given element
- Must always be quoted

Invalid

```
<BusinessCard id="32332" id="ATR_3293" nr=1>
```

Valid

```
<BusinessCard nr="32332" code="ATR_3293">
```

Text

- Actual data content within XML elements
- Format: Characters, numbers or other data types

Comments

- Embed human-readable information
- Used for adding notes or explanations
- Not visible in output
- Enclosed in <!-- and -->
- Not allowed
  - Before document declaration
  - Inside element brackets

Processing Instructions (PIs)

- Provide instructions for the XML processor
- Contain application-specific directives
- Form: `<?targetName instructions ?>`

```
<?application-name action="perform-calculation" value="42"?>

or

<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

CDATA Sections

- CDATA (Character Data) sections allow inclusion of text data that should not be parsed by XML processor like script or style code
- Ensures special characters or sequences in the text do not interfere with the XML structure
- CDATA sections are treated as plain text by the XML parser
- Enclosed in <![CDATA[... unescaped text data ...]]

```
<user-input>
   <![CDATA[
      The user entered: I don’t know if this <value> is correct.
   ]]>
</user-input>
```

Entity References

- General Entities
- Character Entities

```
&copy; &CompanyName;

&amp; &quot; &lt; &gt; &#060; &#062;
```

Namespace

- Namespaces ensure the unique identification of elements and attributes
- Enable XML documents from different sources to be combined without name conflicts
- Use URI references to differentiate similar elements with distinct meanings
- A default namespace can be declared and applied to all unqualified elements
- Prefixes before element names indicate the namespace and prevent ambiguity
- Essential for extending XML languages, such as in XHTML or SVG
- Support XML's extensibility and reusability across applications

Example without namespaces

```
<information>
  <note>
    <author>Jane Doe</author>
    <content>Meeting rescheduled to 3 PM.</content>
    <date>2024-03-28</date>
  </note>
  <note>
    <name>C Sharp</name>
    <octave>5</octave>
    <duration>Quarter note</duration>
  </note>
</information>
```

Example with Namespaces

```
<information xmlns:music="http://www.example.com/music"
xmlns:writing="http://www.example.com/writing">
  <writing:note>
    <writing:author>Jane Doe</writing:author>
    <writing:content>Meeting rescheduled to 3 PM.</writing:content>
    <writing:date>2024-03-28</writing:date>
  </writing:note>
  <music:note>
    <music:name>C Sharp</music:name>
    <music:octave>5</music:octave>
    <music:duration>Quarter note</music:duration>
  </music:note>
</information>
```

XML Syntax

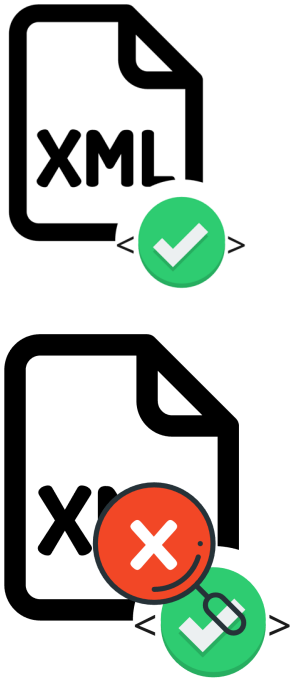
- Must be well-formed
- XML documents require a single root element to encapsulate all content
- Every opening tag must be matched with a closing tag
  - Empty Tags must be closed <hr />
- Attributes values cannot be minimized

  `<option selected>` use `<option selected="selected">`
- Tags are case-sensitive and must be used consistently
- Attribute values must always be quoted.

  `<li id=1>` use `<li id="1">`
- Nested elements must be correctly closed in the order they are opened.

XML Well-formed and Valid

```
<?xml version="1.0" ?>
<bookstore>
  <book>
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
    <year>1925</year>
  </book>
</bookstore>
```



Why Validation of XML Makes Sense:

- Ensures XML documents conform to a predefined structure.
- Improves data integrity and reliability in data exchange.
- Facilitates interoperability between systems and applications.
- Catches errors early in the development process, reducing costs.
- Enables automated parsing and processing of XML documents.
- Provides clear specifications for data formats and types.

Document Type Definitions (DTDs)

- Original schema language for defining XML document structure.
- Can be embedded in XML documents or defined externally.
- Does not support data types other than strings.
- Strict order in which elements appear.
- Offers entity mechanism for reusing content.
- Lacks support for namespaces.
- More widely supported in legacy systems.

DTD Syntax and Building Blocks

- Element declarations
- Attribute declarations
- Entity declarations
- Notation declarations
- PCDATA
- Element quantifiers (?, *, +)
- Choice (|) and sequence (,) operators

DTD Example I

```
<?xml version="1.0"?>
<!DOCTYPE book [
   <!ELEMENT book (title, author+)>
   <!ELEMENT title (#PCDATA)>
   <!ELEMENT author (firstname, lastname, birthdate?)>
   <!ELEMENT firstname (#PCDATA)>
   <!ELEMENT lastname (#PCDATA)>
   <!ELEMENT birthdate (#PCDATA)>
   <!ATTLIST book id CDATA #REQUIRED>
]>
<book id="123">
   <title>The Great Gatsby</title>
   <author>
      <firstname>F. Scrott</firstname>
      <lastname>Fitzgerald</lastname>
   </author>
</book>
```

DTD Example II

```
1  <?xml version="1.0"?>
2  <!DOCTYPE book SYSTEM "book.dtd">
3  <book id="123">
4     <title>The Great Gatsby</title>
5     <author>
6        <firstname>F. Scrott</firstname>
7        <lastname>Fitzgerald</lastname>
8     </author>
9  </book>
10 <book id="456">
11    <title>1984</title>
12    <author>
13       <firstname>George</firstname>
14       <lastname>Orwell</lastname>
15    </author>
16 </book>
17
```

XML Schema

- Richer and more powerful than DTDs.
- Supports XML namespaces and multiple schemas in a single document.
- Allows definition of custom data types and data type inheritance.
- Can enforce the order of child elements.
- Facilitates creation of reusable schema modules.
- Enables default values and fixed values for elements and attributes.
- Better suited for modern, complex applications.

XML Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="title" type="xs:string"/>
     <xs:element name="author" maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="firstname"
          type="xs:string"/>
        <xs:element name="lastname"
          type="xs:string"/>
        <xs:element name="birthdate" type="xs:date"
          minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
    <xs:attribute name="id">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:pattern value="BK-\d{4}"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

XSD Validation Example

```
1  <?xml version="1.0"?>
2  <book id="BK-123" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="book.xsd">
3      <title>The Great Gatsby</title>
4          <author>
5              <firstname>F. Scrott</firstname>
6              <lastname>Fitzgerald</lastname>
7              <birthdate>1896</birthdate>
8          </author>
9  </book>
```

DTD vs. XML Schema

Feature	DTD	XML Schema
Namespaces	Not supported	Full namespace support
Data Types	Only string	Rich support for data types (string, integer, date, etc.)
Syntax	Unique, SGML-derived syntax	Based on XML
Modularity	Limited	Highly modular
Reusability	Entity mechanism	Schema imports and includes
Element Order	Enforced	Flexible
Validation Precision	Less precise	More precise
Default/Fixed Values	Not supported	Supported
Complexity	Simpler, less powerful	More complex, more features
Support	Older standard, declining usage, but universal in XML processors	Modern standard, widely used

XML related technologies

- XPATH
- XSLT
- Xquery
- Xpointer, Xlink

XPath

- **Definition**: A language used for navigating through elements and attributes in an XML document.

- Key concept:
  - Context Node
  - Axis (way from context to selected node)
  - Predicates (further refinement)

XPath Syntax I

XPath Function	Description
/	The root node or absolute path; from the root of the document
/name	All nodes with the name 'name' directly under the root node
//name	Selects all nodes with the name 'name' anywhere in the document
.	The current node
..	The parent of the current node
@	Selects attributes. For example, //@id selects all 'id' attributes
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind
[]	Predicate; filters a node-set based on a condition, like [1] for the first element
name[1]	Selects the first 'name' element child of the context node
name[last()]	Selects the last 'name' element child of the context node

XPath Syntax II

XPath Function	Description
name[position()<3]	Selects the first two 'name' elements
//name[text()='specific text']	Selects 'name' elements with a specific text
//name[starts-with(@attr, 'prefix')]	Selects 'name' elements with an attribute 'attr' that starts with 'prefix'
//name[contains(@attr, 'substring')]	Selects 'name' elements with an attribute 'attr' containing 'substring'
//name[@attr='value']	Selects 'name' elements that have a certain attribute with a specific value
name/following-sibling::node()	Selects all the following siblings of 'name'
name/preceding-sibling::node()	Selects all the preceding siblings of 'name'
name/ancestor::node()	Selects all ancestors (parent, grandparent, etc.) of 'name'
name/descendant::node()	Selects all descendants (children, grandchildren, etc.) of 'name'
name/following::node()	Selects everything in the document after the closing tag of 'name'
name/preceding::node()	Selects everything in the document before the start tag of 'name'

XPath Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="Science">
    <title>The Universe in a Nutshell</title>
    <author>Stephen Hawking</author>
    <year>2001</year>
    <price>25.99</price>
  </book>
  <book category="Fiction">
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adams</author>
    <year>1979</year>
    <price>12.50</price>
  </book>
  <book category="Data Science">
    <title>Python for Data Analysis</title>
    <author>Wes McKinney</author> ...
```

Simple Selection

```
//book/title
```

```
...<year>2012</year>
    <price>39.95</price>
  </book>
  <book category="Science">
    <title>A Brief History of Time</title>
    <author>Stephen Hawking</author>
    <year>1988</year>
    <price>18.00</price>
  </book>
  <book category="Fiction">
    <title>1984</title>
    <author>George Orwell</author>
    <year>1949</year>
    <price>10.95</price>
  </book>
</bookstore>
```

XPath Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="Science">
    <title>The Universe in a Nutshell</title>
    <author>Stephen Hawking</author>
    <year>2001</year>
    <price>25.99</price>
  </book>
  <book category="Fiction">
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adams</author>
    <year>1979</year>
    <price>12.50</price>
  </book>
  <book category="Data Science">
    <title>Python for Data Analysis</title>
    <author>Wes McKinney</author> ...
    ...<year>2012</year>
    <price>39.95</price>
  </book>
  <book category="Science">
    <title>A Brief History of Time</title>
    <author>Stephen Hawking</author>
    <year>1988</year>
    <price>18.00</price>
  </book>
  <book category="Fiction">
    <title>1984</title>
    <author>George Orwell</author>
    <year>1949</year>
    <price>10.95</price>
  </book>
</bookstore>
```

Attribute Selection

```
//book[@category="Science"]/title
```

XPath Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="Science">
    <title>The Universe in a Nutshell</title>
    <author>Stephen Hawking</author>
    <year>2001</year>
    <price>25.99</price>
  </book>
  <book category="Fiction">
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adams</author>
    <year>1979</year>
    <price>12.50</price>
  </book>
  <book category="Data Science">
    <title>Python for Data Analysis</title>
    <author>Wes McKinney</author> ...
```

Predicate Filtering

```
//book[price>20]
```

```
...<year>2012</year>
    <price>39.95</price>
  </book>
  <book category="Science">
    <title>A Brief History of Time</title>
    <author>Stephen Hawking</author>
    <year>1988</year>
    <price>18.00</price>
  </book>
  <book category="Fiction">
    <title>1984</title>
    <author>George Orwell</author>
    <year>1949</year>
    <price>10.95</price>
  </book>
</bookstore>
```

XPath Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="Science">
    <title>The Universe in a Nutshell</title>
    <author>Stephen Hawking</author>
    <year>2001</year>
    <price>25.99</price>
  </book>
  <book category="Fiction">
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adams</author>
    <year>1979</year>
    <price>12.50</price>
  </book>
  <book category="Data Science">
    <title>Python for Data Analysis</title>
    <author>Wes McKinney</author> ...
    ...<year>2012</year>
    <price>39.95</price>
  </book>
  <book category="Science">
    <title>A Brief History of Time</title>
    <author>Stephen Hawking</author>
    <year>1988</year>
    <price>18.00</price>
  </book>
  <book category="Fiction">
    <title>1984</title>
    <author>George Orwell</author>
    <year>1949</year>
    <price>10.95</price>
  </book>
</bookstore>
```

Axes

```
//book[author="Stephen Hawking"]/following-sibling::book/title
```

XPath Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="Science">
    <title>The Universe in a Nutshell</title>
    <author>Stephen Hawking</author>
    <year>2001</year>
    <price>25.99</price>
  </book>
  <book category="Fiction">
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adams</author>
    <year>1979</year>
    <price>12.50</price>
  </book>
  <book category="Data Science">
    <title>Python for Data Analysis</title>
    <author>Wes McKinney</author> ...
```

Functions

```
//book[contains(title, 'Data')]
```

```
...<year>2012</year>
    <price>39.95</price>
  </book>
  <book category="Science">
    <title>A Brief History of Time</title>
    <author>Stephen Hawking</author>
    <year>1988</year>
    <price>18.00</price>
  </book>
  <book category="Fiction">
    <title>1984</title>
    <author>George Orwell</author>
    <year>1949</year>
    <price>10.95</price>
  </book>
</bookstore>
```

eXtensible Stylesheet Language Transformations (XSLT)

- Purpose
  - Transforms XML documents into different XML, HTML, or text formats
  - Enables the separation of content and presentation.
- Uses XSLT stylesheets to define transformation rules.
- Operates as a template engine, matching patterns in the source XML.
- Written in XML

XSLT Key Components

- <xsl:template>: Defines rules to apply when a specified node is matched.
- <xsl:value-of>: Extracts the value of a selected node.
- <xsl:for-each>: Iterates over each node in a selected set of nodes.
- <xsl:if> or <xsl:choose>: Provides conditional processing.

XSLT UseCases

- Use Cases:
  - Generating dynamic web pages from XML data.
  - Converting XML data to PDF or other document formats.
  - Migrating data from one database to another.

XQuery

- Is to XML what SQL is to databases
- Language for querying XML data
- Built on XPath expressions
- Supported by all major databases
- W3C Recommendation

XQuery Key Features

- Functional: Built on functional programming concepts.
- Rich Expressions: FLWOR (For, Let, Where, Order by, Return) for complex queries.
- Versatile: Queries data that is fully structured, unstructured, or semi-structured.

XQuery Use cases

- Transforming XML documents.
- Aggregating data from multiple XML sources.
- Searching text within XML documents for web services.

Literature

- White, Tom. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012.
- Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003. Link
- Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113. Link
- Abiteboul, Serge. "Querying semi-structured data." Database Theory—ICDT'97: 6th International Conference Delphi, Greece, January 8–10, 1997 Proceedings 6. Springer Berlin Heidelberg, 1997.
- Harold, Elliotte Rusty, and W. Scott Means. XML in a nutshell: a desktop quick reference. " O'Reilly Media, Inc.", 2004.
- W3School XML Tutorial: https://www.w3schools.com/xml/
- X-Path: https://w3.org/TR/xpath
- XSLT-Path: https://w3.org/TR/xslt
- ECMA-404 (JSON): https://ecma-international.org/publications-and-standards/standards/ecma-404/