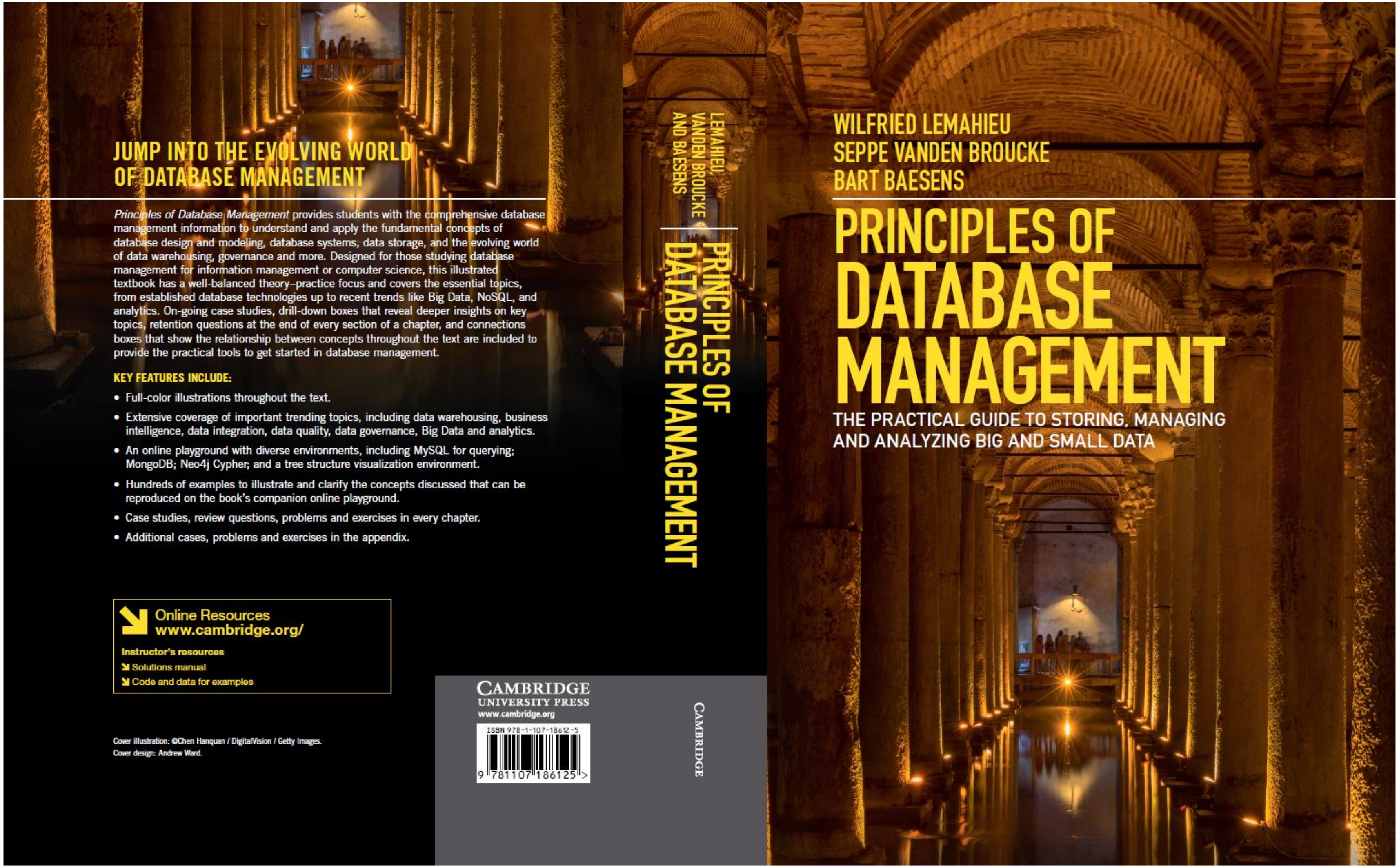
XML Databases



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources
Solutions manual
Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

LEMAHIEU, VANDEN BROUCKE AND BAESENS
PRINCIPLES OF DATABASE MANAGEMENT
CAMBRIDGE

CAMBRIDGE UNIVERSITY PRESS
www.cambridge.org
ISBN 978-1-107-18612-5
9 781107 186125

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com

Introduction

- Extensible Markup Language
- Processing XML Documents
- Storage of XML Documents
- Differences between XML and Relational Data
- Mappings Between XML Documents and (Object-) Relational Data
- Searching XML Data
- XML for Information Exchange
- Other Data Representation Formats

Extensible Markup Language

- Basic Concepts
- Document Type Definitions and XML Schema Definitions
- Extensible Stylesheet Language
- Namespaces
- XPath

Basic Concepts of XML

- Introduced by the World Wide Web Consortium (W3C) in 1997
- Simplified subset of the Standard Generalized Markup Language (SGML),
- Aimed at storing and exchanging complex, structured documents
- Users can define new tags in XML (↔ HTML)

Basic Concepts of XML

- Combination of a start tag, content and end tag is called an XML element
- XML is case-sensitive
- Example

```
<author>
<name>
<first name>Bart</first name>
<last name>Baesens</last name>
</name>
</author>
```

Basic Concepts of XML

- Start tags can contain attribute values

```
<author email="Bart.Baesens@kuleuven.be">Bart Baesens</author>


<author>
<name>Bart Baesens</name>
<email use="work">Bart.Baesens@kuleuven.be</email>
<email use="private">Bart.Baesens@gmail.com</email>
</author>
```

- Comments are defined as follows

```
<!--This is a comment line -->
```

- Processing instructions are defined as follows

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

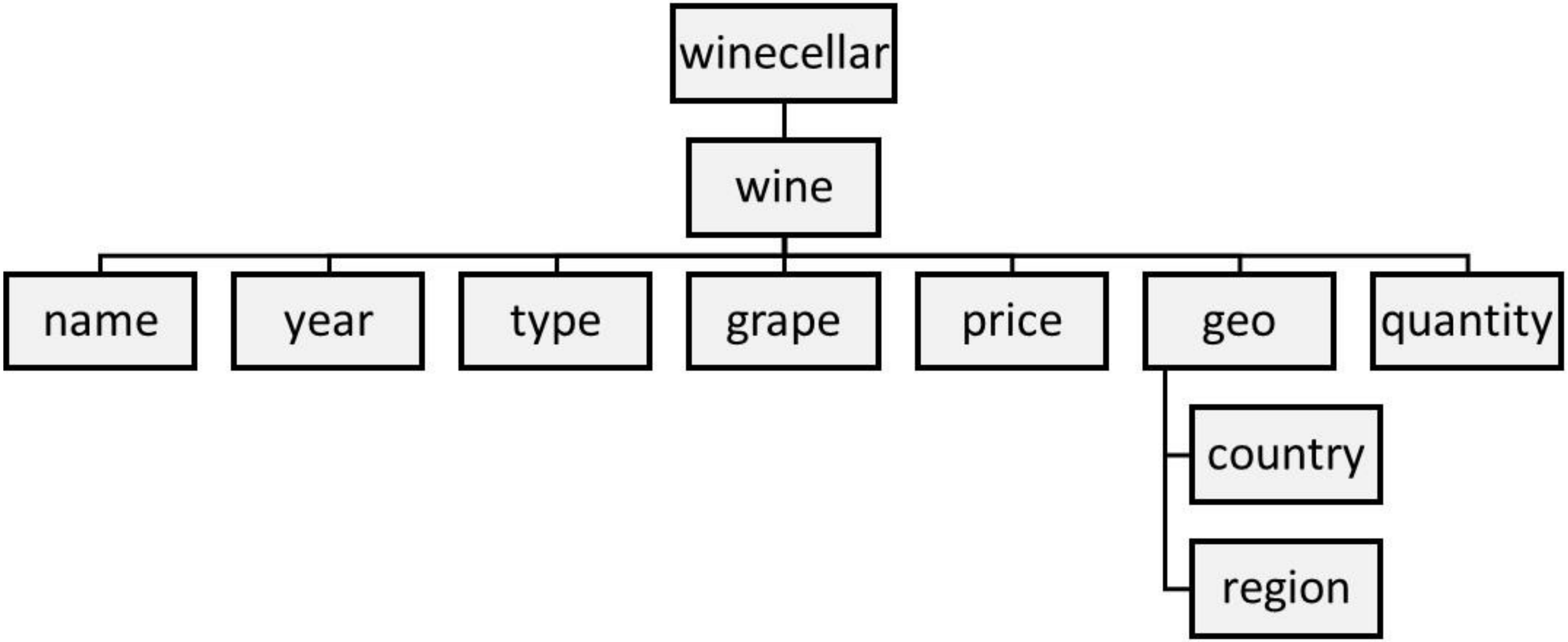
Basic Concepts of XML

- Self-defined XML tags can be used to describe document structure (↔ HTML)
  - can be processed in much more detail
- XML formatting rules
  - only one-root element
  - start tag should be closed with a matching end tag
  - no overlapping tag sequence or incorrect nesting

Basic Concepts of XML

```
<?xml version="1.0" encoding="UTF-8"?>
<winecellar>
   <wine>
      <name>Jacques Selosse Brut Initial</name>
      <year>2012</year>
      <type>Champagne</type>
      <grape percentage="100">Chardonnay</grape>
      <price currency="EURO">150</price>
      <geo>
         <country>France</country>
         <region>Champagne</region>
      </geo>
      <quantity>12</quantity>
   </wine>
<wine>
      <name>Meneghetti White</name>
      <year>2010</year>
      <type>white wine</type>
      <grape percentage="80">Chardonnay</grape>
      <grape percentage="20">Pinot Blanc</grape>
      <price currency="EURO">18</price>
      <geo>
         <country>Croatia</country>
         <region>Istria</region>
      </geo>
      <quantity>20</quantity>
   </wine>
</winecellar>
```

Basic Concepts of XML



Document Type Definitions and XML Schema Definitions

- Document Type Definitions (DTD) and XML Schema Definitions (XSD) specify structure of XML document
- Both define tag set, location of each tag, and nesting
- XML document which complies with DTD or XSD is referred to as valid
- XML document which complies with syntax is referred to as well-formed

Document Type Definitions and XML Schema Definitions

- DTD definition for winecellar

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE winecellar [
<!ELEMENT winecellar (wine+)>
<!ELEMENT wine (name, year, type, grape*, price, geo, quantity)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT grape (#PCDATA)>
<!ATTLIST grape percentage CDATA #IMPLIED>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency CDATA #REQUIRED>
<!ELEMENT geo (country, region)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT region (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
]>
```

Document Type Definitions and XML Schema Definitions

- Disadvantages of DTD
  - only supports character data (no support for integers, dates, complex types)
  - not defined using XML syntax
- XML Schema supports various data types and user-defined types

Document Type Definitions and XML Schema Definitions

- XML Schema definition for winecellar

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="winecellar">
<xs:complexType>
<xs:sequence>
<xs:element name="wine" maxOccurs="unbounded" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:element type="xs:string" name="name"/>
<xs:element type="xs:short" name="year"/>
<xs:element type="xs:string" name="type"/>
<xs:element name="grape" maxOccurs="unbounded" minOccurs="1">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute type="xs:byte" name="percentage" use="optional"/>
</xs:extension>
</xs:simpleContent>
```

Document Type Definitions and XML Schema Definitions

- XML Schema definition for winecellar (contd.)

```
</xs:complexType>
</xs:element>
<xs:element name="price">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:short">
<xs:attribute type="xs:string" name="currency" use="optional"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="geo">
<xs:complexType>
<xs:sequence>
<xs:element type="xs:string" name="country"/>
<xs:element type="xs:string" name="region"/>
</xs:sequence>
```

Document Type Definitions and XML Schema Definitions

- XML Schema definition for winecellar (contd.)

```
</xs:complexType>
</xs:element>
<xs:element type="xs:byte" name="quantity"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Extensible Stylesheet Language

- Extensible Stylesheet Language (XSL) can be used to define stylesheet specifying how XML documents can be visualized in a web browser
- XSL encompasses 2 specifications
  - XSL Transformations (XSLT): transforms XML documents to other XML documents, HTML web pages, or plain text
  - XSL Formatting Objects (XSL-FO): specify formatting semantics (e.g., transform XML documents to PDFs) but discontinued in 2012
- Decoupling of information content from information visualization

Extensible Stylesheet Language

- XSLT stylesheet for summary document with only name and quantity of each wine

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match='/'>
<winecellarsummary>
<xsl:for-each select='winecellar/wine'>
<wine>
<name><xsl:value-of select='name'/></name>
<quantity><xsl:value-of select='quantity'/></quantity>
</wine>
</xsl:for-each>
</winecellarsummary>
</xsl:template>
</xsl:stylesheet>
```

Extensible Stylesheet Language

```
<?xml version="1.0" encoding="UTF-8"?>
<winecellarsummary>
   <wine>
      <name>Jacques Selosse Brut Initial</name>
      <quantity>12</quantity>
   </wine>
   <wine>
      <name>Meneghetti White</name>
      <quantity>20</quantity>
   </wine>
</winecellarsummary>
```

Extensible Stylesheet Language

- XSLT stylesheet for transforming XML document to HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <body style="font-family:Arial;font-size:12pt;background-
color:#ffff">
<h1>My Wine Cellar</h1>
<table border="1">
    <tr bgcolor="#f2f2f2">
      <th>Wine</th>
      <th>Year</th>
```

Extensible Stylesheet Language

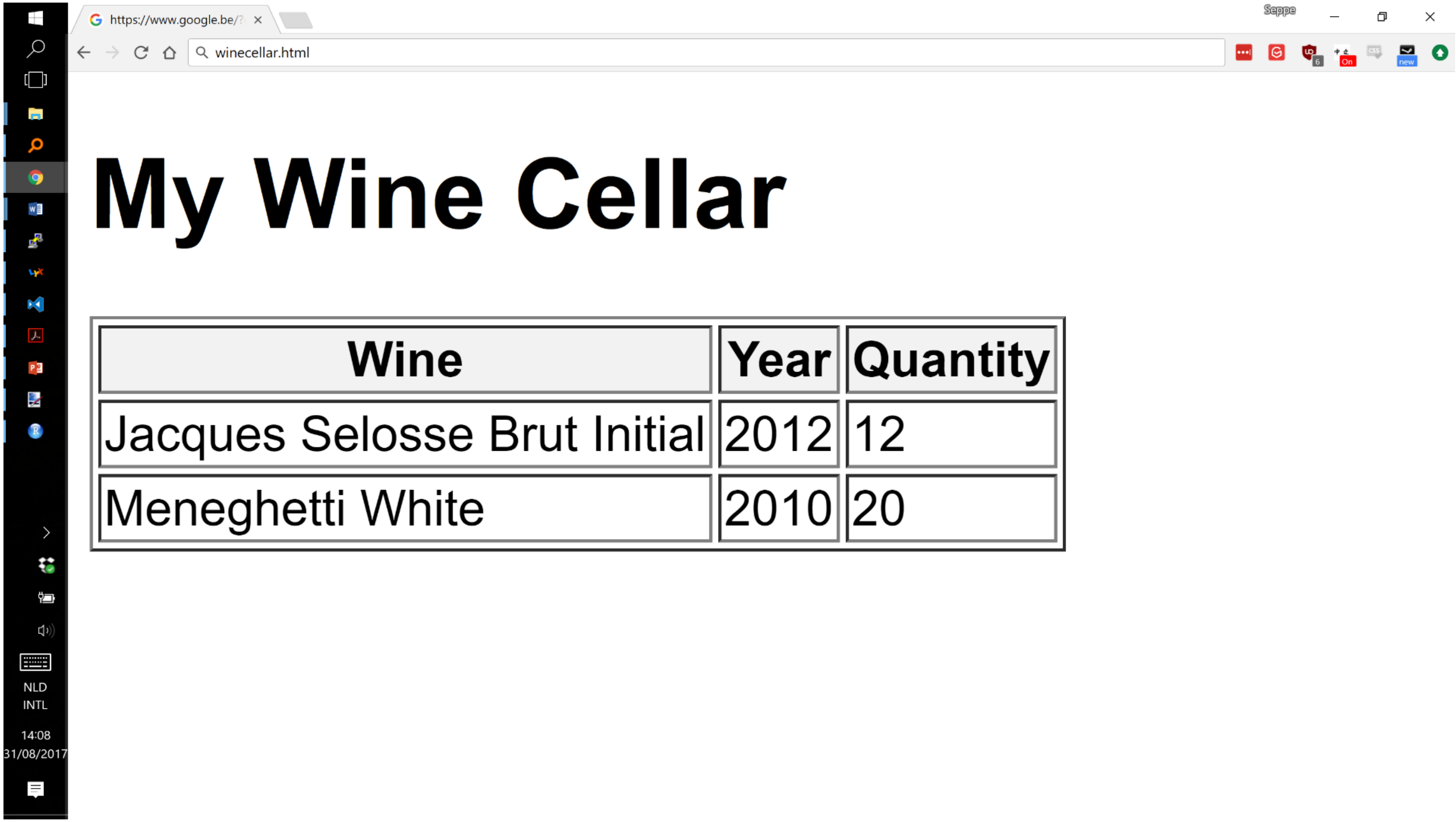
- XSLT stylesheet for transforming XML document to HTML (contd.)

```
<th>Quantity</th>
    </tr>
    <xsl:for-each select="winecellar/wine">
    <tr>
      <td><xsl:value-of select="name"/></td>
      <td><xsl:value-of select="year"/></td>
 <td><xsl:value-of select="quantity"/></td>
</tr>
     </xsl:for-each>
</table>
</body>
</html>
```

Extensible Stylesheet Language

```
<html>
   <body style="font-family:Arial;font-size:12pt;background-color:#ffff">
      <h1>My Wine Cellar</h1>
      <table border="1">
         <tr bgcolor="#f2f2f2">
            <th>Wine</th>
            <th>Year</th>
            <th>Quantity</th>
         </tr>
         <tr>
            <td>Jacques Selosse Brut Initial</td>
            <td>2012</td>
            <td>12</td>
         </tr>
         <tr>
            <td>Meneghetti White</td>
            <td>2010</td>
            <td>20</td>
         </tr>
      </table> </body></html>
```

Extensible Stylesheet Language



My Wine Cellar

Wine	Year	Quantity
Jacques Selosse Brut Initial	2012	12
Meneghetti White	2010	20

Namespaces

- To avoid name conflicts, XML introduced concept of a namespace
- Introduce prefixes to XML elements to unambiguously identify their meaning
- Prefixes typically refer to a URI (uniform resource identifier) which uniquely identifies a web resource such as a URL (uniform resource locator)
  - does not need to refer to physically existing webpage

Namespaces

```
<winecellar xmlns:Bartns="www.dataminingapps.com/home.html">
```

```
<bartns:wine>
<bartns:name>Jacques Selosse Brut Initial</bartns:name>
<bartns:year>2012</bartns:year>
</bartns:wine>
```

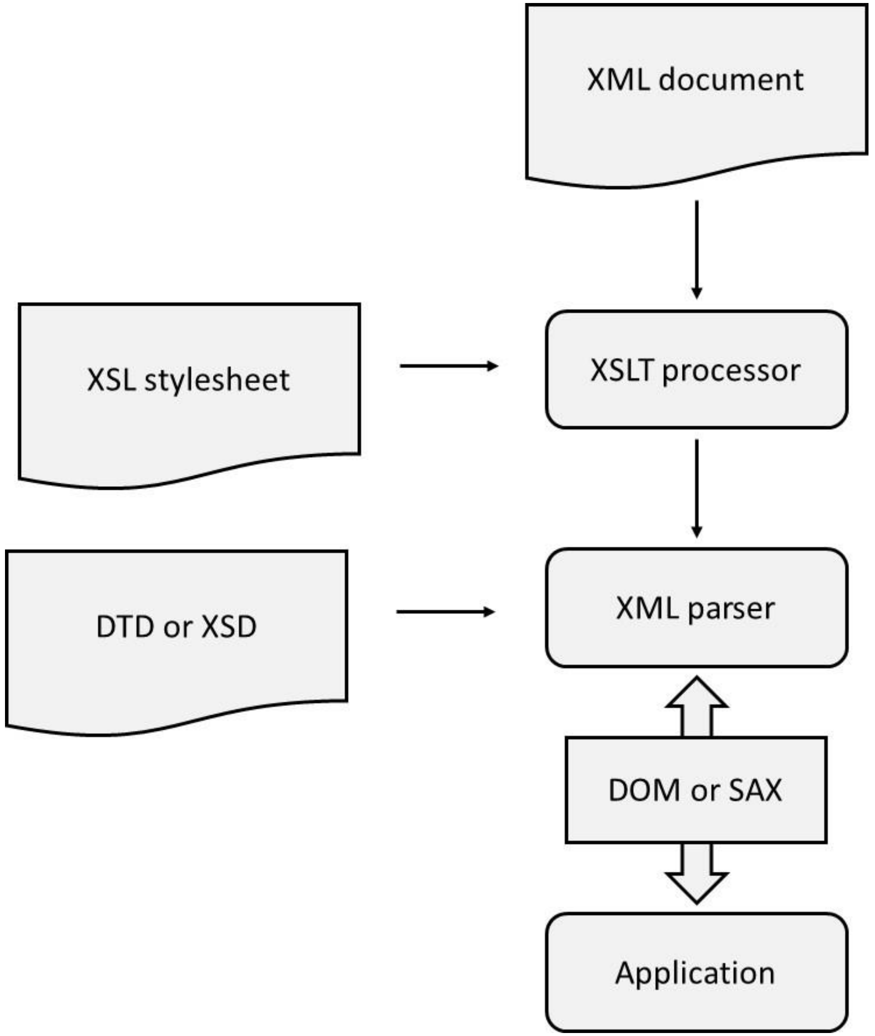
```
<winecellar xmlns="www.dataminingapps.com/defaultns.html">
```

XPath

- XPath is a simple, declarative language that uses path expressions to refer to parts of an XML document
  - considers an XML document as an ordered tree
- Example XPath expressions

```
doc("winecellar.xml")/winecellar/wine
doc("winecellar.xml")/winecellar/wine[2]
doc("winecellar.xml")/winecellar/wine[price > 20]/name
```

Processing XML Documents

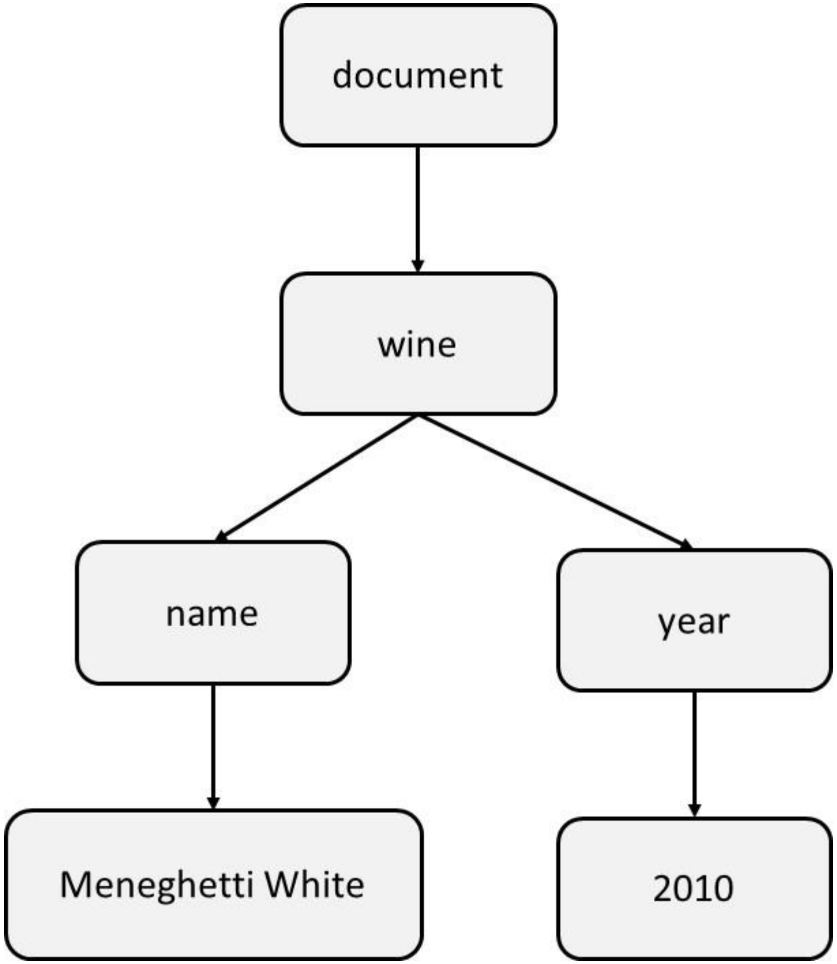


Processing XML Documents

- DOM API is a tree-based API and represents XML document as a tree in internal memory
  - developed by W3C
- DOM provides classes with methods to navigate through the tree and do various operations
- DOM is useful to facilitate direct access to specific XML document parts and when a high number of data manipulations are needed, but can get memory intensive

Processing XML Documents

```
<wine>
<name>Meneghetti White<name>
<year>2010<year>
<wine>
```



Processing XML Documents

- SAX API (Simple API for XML) is an event-based API

```
start document
start element: wine
start element: name
text: Meneghetti
end element: name
start element: year
text: 2010
end element: year
end element: wine
end document
```

Processing XML Documents

- Event stream can be passed on to application which will use an event handler
- SAX has smaller memory footprint and is more scalable than DOM
- SAX is excellent for sequential access, but less suited to support direct random access
- SAX is less performing for heavy data manipulation than DOM
- StAX (Streaming API for XML) is a compromise
  - StAX allows the application to pull XML data using a cursor mechanism

Storage of XML Documents

- XML documents stored as semi-structured data
- Approaches
  - document-oriented approach
  - data-oriented approach
  - combined approach

Document-Oriented Approach for Storing XML Documents

- XML document will be stored as a BLOB or CLOB in a table cell
  - RDBMS considers these as ‘black box’ data
  - querying based upon full-text search
  - (O)RDBMSs have introduced XML data type (SQL/XML extension)
- Simple approach
  - no need for DTD or XSD for the XML document
  - especially well-suited for storing static content
  - but: poor integration with relational SQL query processing

Data-Oriented Approach for Storing XML Documents

- XML document decomposed into data parts spread across a set of connected (object-) relational tables (shredding)
- For highly structured documents and fine-granular queries
- DBMS or middleware can do translation
- Schema-oblivious shredding (starts from XML document) versus schema-aware shredding (starts from DTD/ XSD)
- Advantages
  - SQL queries can now directly access individual XML elements
  - reconstruct XML document using SQL joins
- Object-relational DBMS as an alternative

The Combined Approach for Storing XML Documents

- Combined approach (partial shredding) combines document- and data-oriented approach
- Some parts stored as BLOBs, CLOBs, or XML objects, whereas other parts shredded
- SQL views are defined to reconstruct XML document
- Most DBMSs provide facilities to determine optimal level of decomposition
- Mapping approaches can be implemented using middleware or by DBMS (XML-enabled DBMS)

Differences Between XML Data and Relational Data

- Building block of relational model is mathematical relation which consists of 0, 1 or more unordered tuples
- Each tuple consists of 1 or more attributes
- The relational model does not implement any type of ordering (↔ XML model)
  - add extra attribute type in RDBMS
  - use list collection type in object-relational DBMS

Differences Between XML Data and Relational Data

- Relational model does not support nested relations (first normal form)
  - ↔ XML data is hierarchically structured
  - object-relational DBMS supports nested relations
- Relational model does not support multivalued attribute types (first normal form)
  - ↔ XML allows same child element to appear multiple times
  - additional table needed in relational model
  - object-relational model supports collection types

Differences Between XML Data and Relational Data

- RDBMS only supports atomic data types, such as integer, string, date, etc.
  - XML DTDs don’t support atomic data types (only (P)CDATA)
  - XML Schema supports both atomic and aggregated types
  - aggregated types modeled in object-relational databases using user defined types
- XML data is semi-structured
  - can include certain anomalies
  - change to DTD or XSD necessitates re-generation of tables

Mappings Between XML Documents and (Object-) Relational Data

- Table-Based Mapping
- Schema-Oblivious Mapping
- Schema-Aware Mapping
- SQL/XML

Table-Based Mapping

- Specifies strict requirements to the structure of the XML document

```
<database>
<table>
<row>
<column1> data </column1>
…
</row>
<row>
<column1> data </column1>
…
</row>
…
</table>
<table>
…
</table>
…
</database>
```

Table-Based Mapping

- Actual data is stored as content of column elements
- Advantage is simplicity given the perfect one-to-one mapping
- Document structure can be implemented using an updatable SQL view
- Disadvantage is rigid structure of XML document
  - can be mitigated by XSLT

Schema-Oblivious Mapping

- Schema-oblivious mapping (shredding) transforms XML document without availability of DTD or XSD
- First option is to transform the document to a tree structure, whereby the nodes represent the data in the document
  - tree can then be mapped to a relational model
- Example table

```
CREATE TABLE NODE(
ID CHAR(6) NOT NULL PRIMARY KEY,
PARENT_ID CHAR(6),
TYPE VARCHAR(9),
LABEL VARCHAR(20),
VALUE CLOB,
FOREIGN KEY (PARENT_ID) REFERENCES NODE (ID)
CONSTRAINT CC1 CHECK(TYPE IN ("element", "attribute")));
```

Schema-Oblivious Mapping

```xml
<?xml version="1.0" encoding="UTF-8"?>
<winecellar>
   <wine winekey="1">
      <name>Jacques Selosse Brut Initial</name>
      <year>2012</year>
      <type>Champagne</type>
      <price>150</price>
   </wine>
   <wine winekey="2">
      <name>Meneghetti White</name>
      <year>2010</year>
      <type>white wine</type>
      <price>18</price>
  </wine>
</winecellar>
```

ID	PARENT_ID	TYPE	LABEL	VALUE
1	NULL	element	winecellar	NULL
2	1	element	wine	NULL
3	2	attribute	winekey	1
4	2	element	name	Jacques Selosse Brut Initial
5	2	element	year	2012
6	2	element	type	Champagne
7	2	element	price	150
8	1	element	wine	NULL
9	8	attribute	winekey	2
10	8	element	name	Meneghetti White
11	8	element	year	2010
12	8	element	type	white wine
13	8	element	price	18

Schema-Oblivious Mapping

- XPath or XQuery (see later) queries can be translated into SQL of which the result can be translated back to XML
- Example

```
doc("winecellar.xml")/winecellar/wine[price > 20]/name

SELECT N2.VALUE
FROM NODE N1, NODE N2
WHERE
N2.LABEL="name" AND
N1.LABEL="price" AND
CAST(N1.VALUE AS INT)> 20 AND
N1.PARENT_ID=N2.PARENT_ID
```

Schema-Oblivious Mapping

- Single table requires extensive querying (e.g., self-joins)
- More tables can be created
- Mapping can be facilitated by making use of object-relational extensions
- Due to extensive shredding, reconstruction of XML document can get quite resource intensive
  - middleware solutions offer DOM API or SAX API on top of DBMS
  - materialized views

Schema-Aware Mapping

- Steps to generate database schema from DTD or XSD
  - simplify DTD or XSD
  - map complex element type to relational table, or user-defined type, with corresponding primary key
  - map element type with mixed content to separate table where the (P)CDATA is stored; connect using primary-foreign key relationship
  - map single-valued attribute types, or child elements that occur only once, with (P)CDATA content to a column in the corresponding relational table; when starting from XSD, choose the SQL data type which most closely resembles
  - map multi-valued attribute types, or child elements that can occur multiple times, with (P)CDATA content to a separate table; use primary-foreign key relationship; use collection type in case of object-relational DBMS
  - for each complex child element type, connect the tables using a primary-foreign key relationship

Schema-Aware Mapping

- Generate a DTD or XSD from a database model
  - map every table to an element type
  - map every table column to an attribute type or child element type with (P)CDATA in case of DTD, or most closely resembling data type in case of XML Schema
  - map primary-foreign key relationships by introducing additional child element types
  - object-relational collections can be mapped to multivalued attribute types or element types which can occur multiple times

SQL/XML

- Extension of SQL which introduces
  - new XML data type with corresponding constructor that treats XML documents as cell values in a column of a relational table, and can be used to define attribute types in user-defined types, variables, and parameters of user-defined functions
  - set of operators for the XML data type
  - set of functions to map relational data to XML
- No rules for shredding

SQL/XML

```
CREATE TABLE PRODUCT(
PRODNR CHAR(6) NOT NULL PRIMARY KEY,
PRODNAME VARCHAR(60) NOT NULL,
PRODTYPE VARCHAR(15),
AVAILABLE_QUANTITY INTEGER,
REVIEW XML);
```

```
INSERT INTO PRODUCT VALUES("120", "Conundrum", "white", 12,
XML(<review><author>Bart
Baesens</author><date>27/02/2017</date> <description>This is
an excellent white wine with intriguing aromas of green apple,
tangerine and honeysuckle blossoms.<description><rating max-
value="100">94</rating></review>);
```

SQL/XML

- SQL/XML can be used to represent relational data in XML
  - default mapping whereby names of tables and columns are translated to XML elements and row elements are included for each table row
  - also adds corresponding DTD or XSD
- SQL/XML also includes facilities to represent the output of SQL queries in a tailored XML format
  - `XMLElement` defines XML element using 2 arguments: name of XML element and column name

SQL/XML

```
SELECT XMLElement("sparkling wine", PRODNAME)
FROM PRODUCT
WHERE PRODTYPE="sparkling";
```

```
<sparkling wine>Meerdael, Methode Traditionnelle
Chardonnay, 2014 </sparkling wine>
<sparkling wine>Jacques Selosse, Brut Initial,
2012</sparkling wine>
<sparkling wine>Billecart-Salmon, Brut Réserve,
2014</sparkling wine>
…
```

SQL/XML

```
SELECT XMLElement("sparkling wine", XMLAttributes(PRODNR AS "prodid"),
XMLElement("name", PRODNAME), XMLElement("quantity", AVAILABLE_QUANTITY))
FROM PRODUCT
WHERE PRODTYPE="sparkling";
```

```
<sparkling wine prodid="0178">
<name>Meerdael, Methode Traditionnelle Chardonnay, 2014</name>
<quantity>136</quantity>
</sparkling wine>
<sparkling wine prodid="0199">
<name>Jacques Selosse, Brut Initial, 2012</name>
<quantity>96</quantity>
</sparkling wine>
…
```

```
SELECT XMLElement("sparkling wine", XMLAttributes(PRODNR AS "prodid"),
XMLForest(PRODNAME AS "name", AVAILABLE_QUANTITY AS "quantity"))
FROM PRODUCT
WHERE PRODTYPE="sparkling";
```

SQL/XML

```
SELECT XMLElement("product", XMLElement(prodid, P.PRODNR), XMLElement("name",
P.PRODNAME, XMLAgg("supplier", S.SUPNR))
FROM PRODUCT P, SUPPLIES S
WHERE P.PRODNR=S.PRODNR
GROUP BY P.PRODNR
```

```
<product>
<prodid>178</prodid>
<name>Meerdael, Methode Traditionnelle Chardonnay</name>
<supplier>21</supplier>
<supplier>37</supplier>
<supplier>68</supplier>
<supplier>69</supplier>
<supplier>94</supplier>
</product>
<product>
<prodid>199</prodid>
<name>Jacques Selosse, Brut Initial, 2012</name>
<supplier>69</supplier>
<supplier>94</supplier>
</product>
…
```

SQL/XML

```
SELECT PRODNR, XMLElement("sparkling wine", PRODNAME),
AVAILABLE_QUANTITY
FROM PRODUCT
WHERE PRODTYPE="sparkling";
```

```
0178, <sparkling wine>Meerdael, Methode Traditionnelle
Chardonnay, 2014</sparkling wine>, 136
0199, <sparkling wine>Jacques Selosse, Brut Initial,
2012</sparkling wine>, 96
0212, <sparkling wine>Billecart-Salmon, Brut Réserve,
2014</sparkling wine>, 141
…
```

SQL/XML

- Template-based mapping
  - embed SQL statements in XML documents using tool-specific delimiter (e.g., `<selectStmt>`)

```
<?xml version="1.0" encoding="UTF-8"?>
<sparklingwines>
<heading>List of Sparkling Wines</heading>
<selectStmt>
SELECT PRODNAME, AVAILABLE_QUANTITY FROM PRODUCT WHERE
PRODTYPE="sparkling";
</selectStmt>
<wine>
<name> $PRODNAME </name>
<quantity> $AVAILABLE_QUANTITY </quantity>
</wine>
</sparklingwines>
```

SQL/XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sparklingwines>
<heading>List of Sparkling Wines</heading>
<wine>
<name>Meerdael, Methode Traditionnelle Chardonnay, 2014</name>
<quantity>136</quantity>
</wine>
<wine>
<name>Jacques Selosse, Brut Initial, 2012</name>
<quantity>96</quantity>
</wine>
..
</sparklingwines>
```

Searching XML Data

- Full-text search
- Keyword-Based Search
- Structured Search with Xquery
- Semantic Search with RDF and SPARQL

Full-text search

- Treat XML documents as textual data and conduct brute force full-text search
- Does not take into account any tag structure
- Can be applied to XML documents that have been stored as files or as BLOB/CLOB objects
- Usually by means of object-relational extension
- No semantically-rich queries targeting individual XML elements

Keyword-Based Search

- Assumes XML document is complemented with a set of keywords describing document metadata
- Keywords can be indexed by text search engines
- Document still stored in a file or as BLOB/CLOB
- Still not full expressive power of XML for querying

Structured Search with XQuery

- Structured search uses structural metadata which relates to actual document content
- E.g., XML book reviews
  - document metadata: properties of the document such as, author of the review document (e.g., Wilfried Lemahieu) and creation date (e.g., June 6th, 2017)
  - structural metadata: role of individual content fragments within the overall document structure, e.g., title of book (‘Analytics in a Big Data World’), author of book (‘Bart Baesens’), …

Structured Search with XQuery

- Structured search queries query document content by means of structural metadata
  - E.g., search for reviews of books authored by Bart Baesens
- XQuery formulates structured queries for XML documents
  - can consider both document structure and elements' content
  - XPath path expressions are used for navigation
  - includes constructs to refer to and compare content of elements
  - syntax similar to SQL

Structured Search with XQuery

- XQuery statement is formulated as a FLOWR instruction

```
FOR $variable IN expression
LET $variable:=expression
WHERE filtercriterion
ORDER BY sortcriterion
RETURN expression
```

Structured Search with XQuery

```
LET $maxyear:=2012
RETURN doc("winecellar.xml")/winecellar/wine[year <$maxyear]
```

```
FOR $wine IN doc("winecellar.xml")/winecellar/wine
ORDER BY $wine/year ASCENDING
RETURN $wine
```

```
FOR $wine IN doc("winecellar.xml")/winecellar/wine
WHERE $wine/price < 20 AND $wine/price/@currency="EURO"
RETURN <cheap wine> {$wine/name, $wine/price}</cheap wine>
```

```
FOR $wine IN doc("winecellar.xml")/wine
    $winereview IN doc("winereview.xml")/winereview
WHERE $winereview/@winekey=$wine/@winekey
RETURN <wineinfo> {$wine, $winereview/rating} </wineinfo>
```

Semantic Search with RDF and SPARQL

- Example of semantically-complicated query
  “*Retrieve all spicy, ruby colored red wines with round texture raised in clay soil and Mediterranean climate which pair well with cheese*”
- Semantic web technology stack
  - RDF
  - RDF Schema
  - OWL
  - SPARQL

Semantic Search with RDF and SPARQL

- Resource Description Framework (RDF) provides data model for semantic web
  - encodes graph-structured data by attaching semantic meaning to relationships
  - data model consists of statements in subject-predicate-object format (triples)

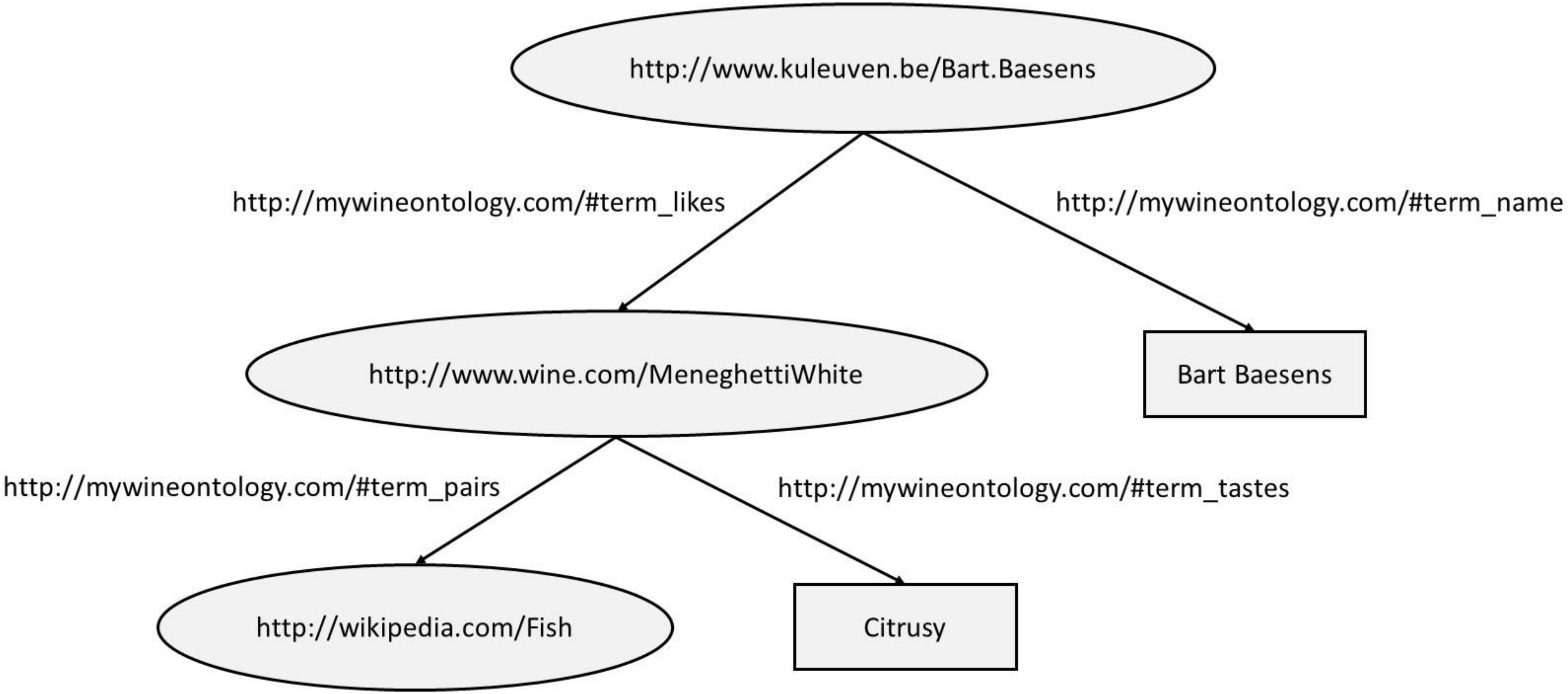
Subject	**Predicate**	**Object**
Bart	name	Bart Baesens
Bart	likes	Meneghetti White
Meneghetti White	tastes	Citrusy
Meneghetti White	pairs	Fish

Semantic Search with RDF and SPARQL

- Represent subjects and predicates using URIs, and objects using URIs
  - universal unique identification becomes possible
- Note: predicate refers to vocabulary or ontology

Subject	Predicate	Object
http://www.kuleuven.be/Bart.Baesens	http://mywineontology.com/#term_name	“Bart Baesens”
http://www.kuleuven.be/Bart.Baesens	http://mywineontology.com/#term_likes	http://www.wine.com/MeneghettiWhite
http://www.wine.com/MeneghettiWhite	http://mywineontology.com/#term_tastes	“Citrusy”
http://www.wine.com/MeneghettiWhite	http://mywineontology.com/#term_pairs	http://wikipedia.com/Fish

Semantic Search with RDF and SPARQL


http://www.kuleuven.be/Bart.Baesens
http://mywineontology.com/#term_likes
http://mywineontology.com/#term_name
http://www.wine.com/MeneghettiWhite
Bart Baesens
http://mywineontology.com/#term_pairs
http://mywineontology.com/#term_tastes
http://wikipedia.com/Fish
Citrusy

Semantic Search with RDF and SPARQL

- RDF data can be serialized by means of RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/TR/PR-rdf-syntax/"
xmlns:myxlmns="http://mywineontology.com/" />
<rdf:Description
rdf:about="http://www.kuleuven.be/Bart.Baesens">
<myxlmns:name>Bart Baesens</ myxlmns:name>
<myxlmns:likes
rdf:resource="http://www.wine.com/MeneghettiWhite"/>
</rdf:Description>
</rdf:RDF>
```

Semantic Search with RDF and SPARQL

- RDF is one of the key technologies to realize Linked Data
- RDF Schema enriches RDF by extending its vocabulary with classes and subclasses, properties and subproperties, and typing of properties
- Web Ontology Language (OWL) is an even more expressive ontology language which implements various sophisticated semantic modeling concepts

Semantic Search with RDF and SPARQL

- RDF data can be queried using SPARQL (“SPARQL Protocol and RDF Query Language”)
- SPARQL is based upon matching graph patterns against RDF graphs
- Examples

```
PREFIX: mywineont: <http://mywineontology.com/>
SELECT ?wine
WHERE {?wine, mywineont:tastes, "Citrusy"}

PREFIX: mywineont: <http://mywineontology.com/>
SELECT ?wine, ?flavor
WHERE {?wine, mywineont:tastes, ?flavor}
```

XML for Information Exchange

- Message Oriented Middleware (MOM)
- SOAP-Based Web Services
- REST-Based Web Services
- Web Services and Databases

Message Oriented Middleware (MOM)

- Enterprise Application Integration (EAI): set of activities aimed at integrating applications within an enterprise
- EAI can be facilitated by 2 types of middleware
  - Remote Procedure Call (RPC): communication is established through procedure calls (e.g., RMI, DCOM); usually synchronous; strong coupling
  - Message Oriented Middleware (MOM) integration is established by exchanging XML messages; usually asynchronous; loose coupling

SOAP-Based Web Services

- Web services: self-describing software components, which can be published, discovered and invoked through the web
- Simple Object Access Protocol (SOAP)
  - Extensible, neutral, and independent XML-based messaging framework

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<GetQuote xmlns="http://www.webserviceX.NET/">
<symbol>string</symbol>
</GetQuote>
</soap:Body>
</soap:Envelope>
```

SOAP-Based Web Services

- Before a SOAP message can be sent to a web service, it must be clear which type(s) of incoming messages the service understands and what messages it can send in return
- Web Services Description Language (WSDL) is an XML-based language used to describe the interface or functionalities offered by a web service

SOAP-Based Web Services

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://www.webserviceX.NET/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.webserviceX.NET/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/">
<wsdl:types><s:schema targetNamespace="http://www.webserviceX.NET/" elementFormDefault="qualified">
<s:element name="GetQuote">
<s:complexType>
<s:sequence>
<s:element type="s:string" name="symbol" maxOccurs="1" minOccurs="0"/></s:sequence>
</s:complexType></s:element>
<s:element name="GetQuoteResponse">
<s:complexType>
<s:sequence>
<s:element type="s:string" name="GetQuoteResult" maxOccurs="1" minOccurs="0"/>
</s:sequence>
</s:complexType>
</s:element>
<s:element type="s:string" name="string" nillable="true"/>
</s:schema>
</wsdl:types>
…
</wsdl:definitions>
```

SOAP-Based Web Services

- Web service represented as set of port types that define set of abstract operations
  - operation has input message and optional output message (SOAP based)
  - message specifies attributes and their types using XML Schema
  - port types can be mapped to an implementation (port) by specifying URL
  - same WSDL document can refer to multiple implementations
- E-business transactions take place according to predefined process model based on web services and XML

REST-Based Web Services

- REST (Representational State Transfer) is built on top of HTTP and is completely stateless and light
  - less verbose than SOAP
  - based on request-reply functionality, for which HTTP is already perfectly suited
  - has become the architecture of choice by “modern” web companies to provide APIs
  - REST is tightly integrated with HTTP whereas SOAP is communication agnostic

REST-Based Web Services

```
GET /stockquote/IBM HTTP/1.1
Host: www.example.com
Connection: keep-alive
Accept: application/xml
```

```
HTTP/1.0 200 OK
Content-Type: application/xml
<StockQuotes>
<Stock>
<Symbol>IBM</Symbol>
<Last>140,33</Last>
<Date>22/8/2017</Date>
<Time>11:56am</Time>
<Change>-0.16</Change>
<Open>139,59</Open>
<High>140,42</High>
<Low>139,13</Low>
<MktCap>135,28B</MktCap>
<P-E>11,65</P-E>
<Name>International Business Machines</Name>
</Stock>
</StockQuotes>
```

Web Services and Databases

- Web service can make use of underlying database
- Database can act as web service provider or web service consumer
- Stored procedures can be extended with WSDL interface and published as web services
  - results can be returned as XML (e.g., SQL/XML)
- Stored procedures or triggers can include calls to external web services
  - E.g., trigger which monitors (local) stock data and if safety stock level is reached automatically generates a (e.g. SOAP) message with a purchase order to the web service hosted by the supplier
- Implications on transaction management (e.g. WS-BPEL)!

Other Data Representation Formats

- JSON and YAML are optimized for data interchange and serialization
- JavaScript Object Notation (JSON) provides a simple, lightweight representation based on name-value pairs
  - JSON provides 2 structured types: objects and arrays
  - primitive types supported: string, number, Boolean, and null
  - JSON is human and machine readable and models data in hierarchical way
  - structure of JSON specification can be defined using JSON Schema
  - JSON is not a markup language and not extensible
  - JSON documents can be parsed using the `eval()` function
  - native and fast JSON parsers in modern day web browsers

Other Data Representation Formats

```
{
  "winecellar": {
    "wine": [
      {
        "name": "Jacques Selosse Brut Initial",
        "year": "2012",
        "type": "Champagne",
        "grape": {
          "_percentage": "100",
          "__text": "Chardonnay"
        },
        "price": {
          "_currency": "EURO",
          "__text": "150"
        },
```

Other Data Representation Formats

```
"geo": {
          "country": "France",
          "region": "Champagne"
        },
        "quantity": "12"
      },
      {
        "name": "Meneghetti White",
        "year": "2010",
        "type": "white wine",
        "grape": [
          {
            "_percentage": "80",
            "__text": "Chardonnay"
          },
```

```
{
            "_percentage": "20",
            "__text": "Pinot Blanc"
          }
        ],
        "price": {
          "_currency": "EURO",
          "__text": "18"
        },
        "geo": {
          "country": "Croatia",
          "region": "Istria"
        },
        "quantity": "20"
      }
    ]
  }
}
```

Other Data Representation Formats

- YAML Ain’t a Markup Language (YAML) is a superset of JSON with support for relational trees, user-defined types, explicit data typing, lists and casting
  - better alternative for object serialization
  - uses inline and white space delimiters
  - works with mappings, which are sets of unordered key/value pairs and sequences which correspond to arrays
  - supports numbers, strings, Boolean, dates, timestamps, and null

Other Data Representation Formats

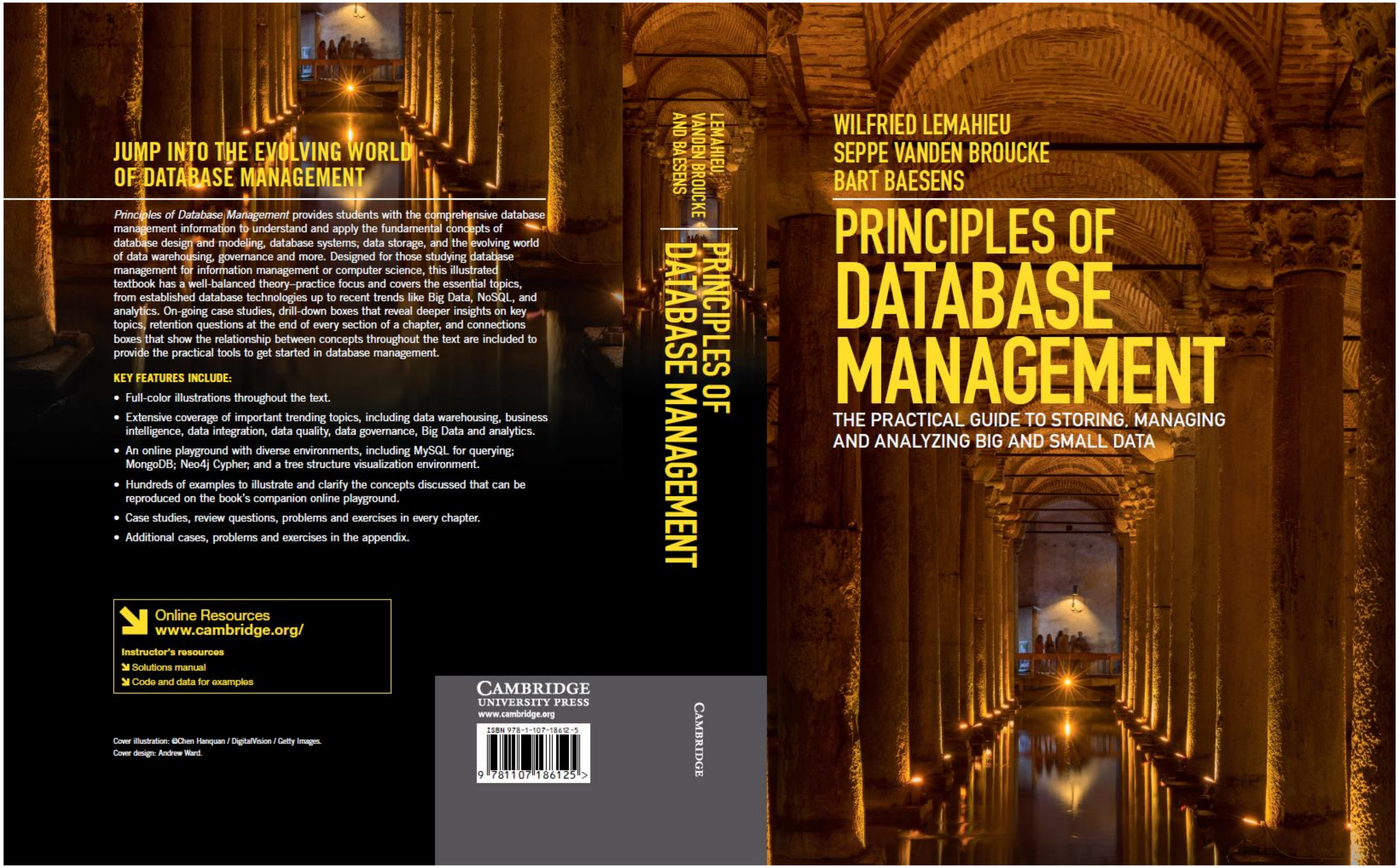
```
winecellar:
 wine:
  -
   name: "Jacques Selosse Brut
Initial"
   year: 2012
   type: Champagne
   grape:
    _percentage: 100
    __text: Chardonnay
   price:
    _currency: EURO
    __text: 150
   geo:
    country: France
    region: Champagne
   quantity: 12
  -
```

```
name: "Meneghetti White"
   year: 2010
   type: "white wine"
   grape:
    -
   _percentage: 80
     __text: Chardonnay
    -
   _percentage: 20
     __text: "Pinot Blanc"
   price:
    _currency: EURO
    __text: 18
   geo:
    country: Croatia
    region: Istria
   quantity: 20
```

Conclusions

- Extensible Markup Language
- Processing XML Documents
- Storage of XML Documents
- Differences between XML and Relational Data
- Mappings Between XML Documents and (Object-) Relational Data
- Searching XML Data
- XML for Information Exchange
- Other Data Representation Formats

More information?



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources
Solutions manual
Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-18612-5

LEMAHIEU
VANDEN BROUCKE
AND BAESENS

PRINCIPLES OF
DATABASE MANAGEMENT

CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com