

BINF 8211/6211
Design and Implementation of
Bioinformatics Databases
Lecture #19

Dr. D. Andrew Carr
Dept. Bioinformatics and Genomics UNCC
Spring 2016

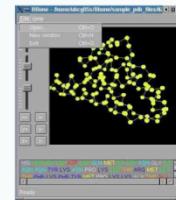
Review Relational Model

Today Topics

- SQL
- Project information review
 - Homework 6,7
- Data Representation

SQL

- TRIGGER
 - Used to automatically update after insert of data.
 - CREATE or REPLACE TRIGGER
 - AFTER INSERT or UPDATE OF ...
 - BEGIN
 - UPDATE ...
 - END
- Stored procedures
 - Internal SQL that are contained with the database
- Stored Functions
- Embedded SQL



Data Representation Styles

- Structure = Form + **Meaning** (interpretation)
 - Link together nouns and verbs to provide meaning
 - Format and context
 - An access path must be known (browsing, forms, dump files, parsing tools)
- Highly Structured Representation
 - Strongly typed: constrained, relations explicit
 - Data model is documented
 - Where is the data?
 - What does it mean?
 - What are the related parts and how are they represented?
- Semi-structured
 - Not strongly typed, flexible attributes, external sources may define structures.

What is it that you have to map? That is, the material, the form that you are working with – has a shape, or constraints, and properties

Strongly typed says what types of values are possible, what operations are valid, the meanings that can be associated with the data and how the data must be stored (floating point vs integer vs character for example).

What structures can the material support – what can it interact with

Representation means partly how thoroughly you understand it and how well you described it

If you only know a little bit then trying to put it into a framework that demands a great deal of detail can be frustrating (or you guess)

Semi-structured that allows for flexibility may be more efficient in the long run, even if it is not as elegant

Structuring data enforces careful assessment of its use(s).

Reminder:

- Scientific databases support research by providing structured data storage
 - Tasks:
 - Merge information across different databases
 - Add new datasets
 - Mediate data representations or create new types
- Accurate data fusion (integration) requires knowledge of
 - Meaning of the data (semantics)
 - Format of the data (syntax)
 - Location of the data

With scientific data the goal is to uncover new information, find new relationships by making new combinations and transformations

Merge outcomes, extract new values

Much easier to create a mud pie than a pot that can stand under its own weight or even contain other substances— the architecture matters, material and connections

Restructuring data alters the data representation – how it is interpreted partly depends on the structure so this is not cost-free.

- Data are highly structured in relational databases. We specify:
 - Data types, ranges, business rules, cardinality
- ‘Semi-structured’ representations format elements with defined tags into hierarchies
 - **Markup languages** use tag-value *pairs* – the overarching framework is the standard generalized markup language, SGML.
 - **RDF**, the resource description framework, is a metadata model using *triples* (subject-predicate-object) statements.
 - It can be *serialized* in an XML format.
- Complete communication of the meaning requires that format and context be given

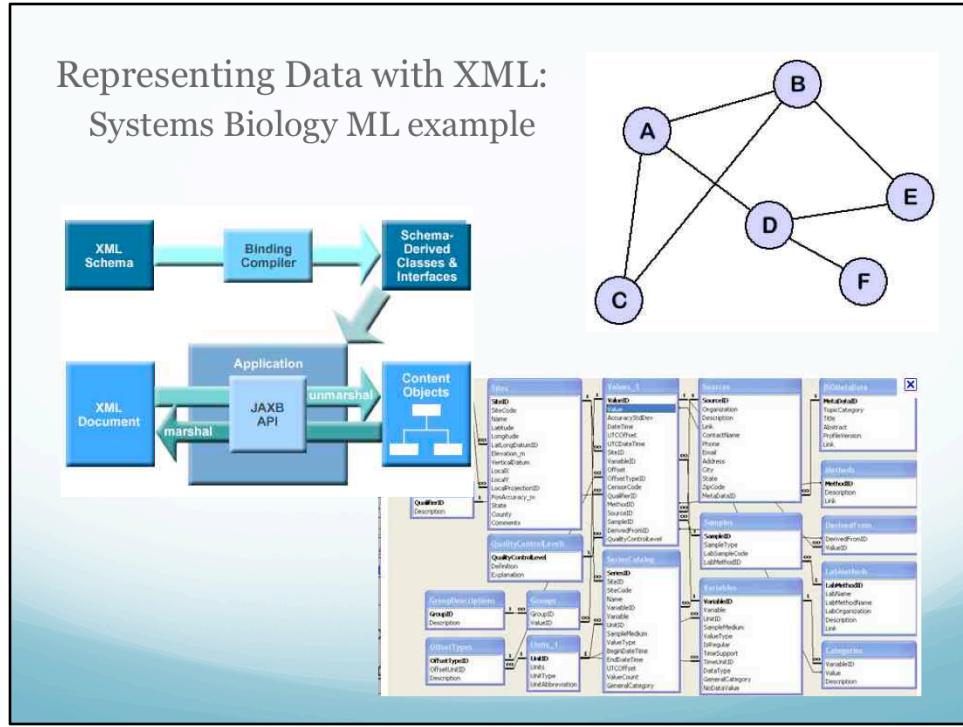
standard generalized markup language

Interpretation: substance vs style

- Tags are familiar:
 - HTML uses tags to describe how things should **look**
`<i>hi</i>` for
hi
 - XML uses tags to describe **meaning**
`<emphasis>hi</emphasis>` HI!
`<greeting>hi</greeting>` Hi
- An XML *processor* provides presentation
 - Read file, convert the data into representation appropriate for another program
 - Tags → symbols for actions
 - web browsers, XML editors, data systems, etc.

3/31/16

Representing Data with XML: Systems Biology ML example



Different ways to structure data allow choices for downstream applications, exchange, sharing.

Main ways to discriminate representation modes= structured vs unstructured

Markup languages use tags to map elements clearly.

- HTML is a *text*-markup language
 - Mapping appearance but not meaning.
 - <u>appearance</u> gives appearance in an application that can interpret the tags.
 - Grammatical structure and meaning (syntax and semantics) are *not* part of html.
- XML is a *data* markup language
 - Specifically designed for data sharing in documents in the Web client model.
 - In a microarray output file, you would like to know the probe sequence from the probe intensity, for example.
- You define tags that explain what data is in a document
 - Where are the bits that are usable.
 - What are reasonable things to do with them.

Syntax is the part of speech (noun, verb – do what to whom) and semantic is what it means in context.

A document must declare the type of markup it is using (DTD)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html lang="en">
<head>
<table style="border-style: solid; border-color: black; border-width: 1px 1px 0px; background: whitesmoke none ;
width: 1056px; height: 498px;" border="0" cellpadding="4" cellspacing="0">
<tbody>
<tr>
<td colspan="2" rowspan="1" style="vertical-align: top;">
<table style="text-align: left; width: 100%;" border="1" cellpadding="2" cellspacing="2">
<tbody>
<tr>
<td colspan="5" style="vertical-align: middle; background-color: rgb(204,204,255); text-align: center;">
<big><big><style="color: rgb(0, 0, 153);">
Design and Implementation of Bioinformatics Databases</big><br>
BINF 8211/BINF 6211</big> <br>
UNCC<br>
Spring 2012
</td>
</tr>
```



Design and Implementation of Bioinformatics Databases
BINF 8211/BINF 6211
UNCC
Spring 2012

Document Type Definition: In this case I declare that I am using an HTML declaration, for version 4.0 and English conventions.

This is self-describing

So now we have a document that is a collection of data and that is self-describing and that is portable (Unicode based) and that can describe data in tree (graph) structures – is that a database?

It is pretty verbose (at least triple the size) and getting to the data can require text conversion and does require parsing.

Instead XML and its technologies provide a lot of the technology of a database management system.

Markup tags are **not** intended to be part of the output.

- The markup language defines what part is content, what the allowed tags are.
 - Applications use the markup to direct processes on the content
 - Processing can be modified for a particular context
 - Uniform formatting
 - Consistency of extraction
- The tags are paired brackets containing text:
`<gene> HER2/NEU</gene>`
- Tags + DTD make the file self-documenting
 - Some context for each value
 - To use: sender and receiver must agree on the tags and their interpretation – editors, parsers.

An application will look for tags that are meaningful to it, extract the values but not the tags (parse) and then do something with those values – perhaps reformat, perhaps carry out calculations. You might pass values to the application that alters how it filters some of the values, or which tags it selects.

Data- Vs. Document-centric DTDs

- Data-centric: XML is used as a data transport – the 'documents' are designed for machine consumption.
 - Data is fine-grained, has a regular structure, no mixed content in the fields
 - Data often originates in a database and is being transported to an application, or is intended to be inserted into a database for storage and before use
- Document-centric: XML is used to improve human consumption of the content.
 - Data is coarse-grained, irregular, size varies greatly and has mixed content

Essentially – the normalization has been pretty complete and the records are pretty complete. We know the expected size of each record.

eXtensible Markup Languages

- eXtensible Markup Language – “a syntax foundation for mediation on document contents” (W3 definition)
- Protocol for a self-describing information container
 - To enforce the protocol you need a toolkit for *creating the document with correct structure, and then for using documents created in the markup languages*
 - Documents often contain several types of data with differing structures.
- *Tags* delineate parts:
 - Boundaries (start and stop)
 - Roles (specific regions to be treated in a special way)
 - Positions
 - Containment – nesting (→ hierarchical)
 - Relationships - bring in *something from another file* at a position – example in downstream slides

UNCC BINF Instructor: Weller

3/31/16

A file is a package of data treated as a continuous unit (a physical structure)

An XML document is a logical structure that integrates the contents of multiple files.

It is allowable to make up a freeform XML, following a basic set of rules prescribing the proper form and using the tags.

It is more common to provide a DTD for document modeling to achieve:

A set of explicit rules (declarations) about how the tags look and what they can contain.

XML Schema is a standard for such document modeling, providing templates as examples to follow for good practice.

XML is a *framework*, not a single language, each domain defines the elements needed.

- ‘MedML’ is a set of self-describing elements within a framework – in this case it is medicine.
 - Each **element** within the language is explicitly named
 - <Interview FirstData=“2001-01-15” BMI=“16.66” Height = “62”...>
- The element represents a concept within your framework
 - The element has a **name**, or tag
 - **Attributes** appear as **name-value pairs**.
 - The data type of the Attributes is text or strings (CDATA = character data, PC data = Parsed Character Data) and each attribute can appear only once in a given element
- In a reconstructed document, the attribute name (the tag) does not appear, the value of the attribute (the field) does.

In a database the distinction between subelement and attribute is less clear – generally only identifiers are stored as attributes.

eXaMpLe

Element
Attribute and value

- A particular XML ('MedML') is a set of self-describing elements
- Example
 - <Interview FirstDate =“2001-01-15” BMI=“16.66” Height = “62”...>
- Elements can vary in number and there is only one type
- The element represents a concept
 - The order of *elements* is set, the order of attributes within elements is not set
 - Additional attributes can be added
 - Not all elements have to have the same set of attributes (you don't have to specify attributes for which no values are provided)
- There can be Entity references (indicated with '&') that specify replacement /inserted content from a linked file
 - &spdb might mean Swiss-Prot database

The elements resides in some knowledge domain, each element within the language is explicitly named

An XML document is self-describing: when you exchange it the necessary meta-data comes along.

- Representation is flexible and *extensible*.
 - Can change order within level of container
 - Allows multi-valued attributes
 - Omit Attributes
 - Comments can be inserted at any point
- A lot of scientific examples are provided and full descriptions of tags in particular application areas are here:

<http://mariovalle.name/sdm/scientific-data-management.html>
- Allows computer processing of file with a generic tool that does not care *at all* about specific content (values), only how they should be treated.

XML DTD structure

- In order to understand the way tags are used and organized in an XML file you provide a second file, which can be either a DTD (file.dtd) or XML Schema (file.xsd)

- Document Type Definition (has a .dtd Extension)

- Provides the logical model and set of valid tags

- First line declares the root element

```
<!ELEMENT Root (child +)>
<!ELEMENT Children (sub1, sub 2, sub3,)>
<!ELEMENT sub1(#PCDATA)> → leaf
```

- In the XML document you reference the dtd as follows

```
<?xml version =“1.0”?>
<!DOCTYPE sbml SYSTEM “sbml.dtd”>
<sbml> corresponds to root element of XML documents
that will use the DTD for validation
```

XML DTD file structure

```
<!ELEMENT ProductList (Product+)>
<!ELEMENT Product (P_code, P_Describe, P_indate?, P_expire?, P_Min?, P_Price)>
<!ELEMENT P_code (#PCDATA)>
<!ELEMENT P_Describe (#PCDATA)>
<!ELEMENT P_indate (#PCDATA)>
<!ELEMENT P_expire (#PCDATA)>
<!ELEMENT P_Min (#PCDATA)>
<!ELEMENT P_Price (#PCDATA)>

<?xml version ="1.0"?>
<!DOCTYPE ProductList SYSTEM "ProductList.dtd">
<ProductList>
    <Product>
        <P_code>23109-AA</P_code>
        <P_Describe>Hammer</Describe>
        <P_price> 6.00</P_price>
    </Product>
    <Product>
        <P_code>40002-AB</P_code>
        <P_Describe>Hex Wrench</Describe>
        <P_price> 12.95</P_price>
    </Product>
</ProductList>
```

In the xml document the dtd has been *referenced* in the second line.

- The top has the dtd and the bottom the resulting xml
 - + means the element occurs one or more times (required multivalued element)
 - * means the element occurs zero or more times (optional multivalued element)
 - ? Means the element is optional (optional single-valued element)
 - If no symbol is given the element must occur exactly once (required single-valued element)
 - a|c gives you the OR option (Minimum price could be a= 0.99 Or c= 6.00)
 - (e) lets you group elements (as in the second line)

The DTD is a set of explicit rules (declarations) about how the tags look and what they can contain.

This has limited expressivity – root, parent, child, mandatory, optional and no good way to state data type or data validation rules.

Also, this has a different syntax from the xml document itself, so it is not self-describing. Since DTD elements must follow the specified order you cannot actually use the unordered elements that were meant to be a strength of the XML approach.

Therefore we move on to the XML Schema construction.

Specifying Attributes in the DTD

- The previous example did not include any attributes for the Elements – what do you do with those?

```
<!ELEMENT height (#PCDATA)>
<!ATTLIST height
      Dimension   CDATA  #REQUIRED
      accuracy    CDATA  #IMPLIED >
```

- In this case Implied means Optional.
 - CDATA is the type, which is character, or string.
- The syntax is
 - <!ATTLIST element_name
 - Attribute_name attribute_type default_value>
- Attribute types include
 - CDATA, ENUMERATION, ID, IDREF, IDREFS, NOTATION, NMTOKEN, NMTOKENS, ENTITY, ENTITIES
- Attribute Default Values include
 - #IMPLIED
 - #REQUIRED
 - #FIXED
 - #LITERAL

Interpretation requires DTDs and Schemas.

Entity References in the DTD

- There are sometimes very long names that you would like to abbreviate – the method for doing this is to declare an entity reference
- Declare the ENTITY as an element in the DTD and then use the abbreviation in the XML document.

```
<?xml version ="1.0" encoding ="UTF-8"?> The DTD is specified in the DOCTYPE
<!DOCTYPE long-words [
<!ELEMENT long-words (word+)> The element long-words contains at least one word.
<!ELEMENT word (#PCDATA)> Define 3 entities – an abbreviation, followed by the
<!ENTITY ade "antidepressant"> word in quotes
<!ENTITY pph "pseudohypoparathyroidism">
<!ENTITY pums "pneumonia">
]> The abbreviation can be used in the XML document
<long-words> – it is preceded by an ampersand and followed by the
    <word>&ade;</word> semi-colon.
    <word>&pph;</word>
    <word>&pums;</word>
</long-words> If you want to use the reserved characters you do
something similar:
&lt; &gt; &amp; &apos; &quot;
```

Interpretation requires DTDs and Schemas.

When names collide: XML -Namespaces

- XML documents can reference one another - the strategy assumes that useful information is available in XML format at a known location.
 - Element names are defined by the developer of a DTD.
 - When you link documents names are likely to conflict
- *Namespaces* are registered – their purpose is to explain how elements and attributes are used in a DTD or Schema.
 - An XML Namespace is designated xmlns in the DTD or Schema

XML is one of a set of semantic Web technologies. Resources are addressable at Web-accessible servers.

- A namespace mechanism lets groups specify globally unique names to use as element tags in their context.
 - In a namespace you pre-pend each tag or attribute with a **URI, followed by a colon** (like those http:// identifiers for Web URLs).
 - The designation here is xmlns.

```

<root
  xmlns:h="http://www.w3.org/TR/html4/"
  xmlns:f="http://www.w3schools.com/furniture">

  <table>
    <tr>
      <td>Apples</td>
      <td>Bananas</td>
    </tr>
  </table>

  <table>
    <name>African Coffee Table</name>
    <width>80</width>
    <length>120</length>
  </table>

</root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>

```

You can define an abbreviation for the namespace here we have bound the prefix h to the first URL and f to the second URL.
 Use of the prefix is optional in the subelements of each set (unless I am switching out in the middle)

Uniform Resource Identifier (URI)

- A URI is a string that unambiguously denotes a resource
 - This could be a URN (name) or a URL (locator).
 - Many books use urn:isbn: #####-##### to identify a book edition.
 - The syntax is
 - <scheme>:<authority><path>?<query>
 - http://www.w3schools.com/Schema/schema_intro.asp
 - Hierarchical URIs use the ‘/’ character as a separator, with paths over a network being ‘//’.
 - http://www.example.org/path/to/somewhere?id_number
- A URI *reference* is a string that represents a URI
 - An optional fragment identifier can be attached by using a postfix ‘#’ and the fragment
 - <http://www.example.org#fragment>

Universal Resource Identifiers

URL is a special case specifying the location of a Web resource, used by a Web browser to find and download the specified resource

The URI identifies but does not necessarily download the resource (a resource might have a URI not the same as the URL)

A Uniform Resource Name omits the protocol specifying how to obtain the

Reading about XML

- Chapter 14 Section 3, Rob and Coronel (8th edition) “Database Systems”
- Online: www.w3schools.com – has tutorials on XML, XML DTD and XML Schema