Beyond relational databases

Matthew J. Graham
CACR

Methods of Computational Science
Caltech, 5 May 2011

matthew graham

Name	Distance	Brightness	Temperature
Sirius	2.64	-1.47	9960

HTML:

```
<TD> Sirius </TD>
<TD> 2.64 </TD>
<TD> -1.47 </TD>
<TD> 9960 </TD>
```

XML:

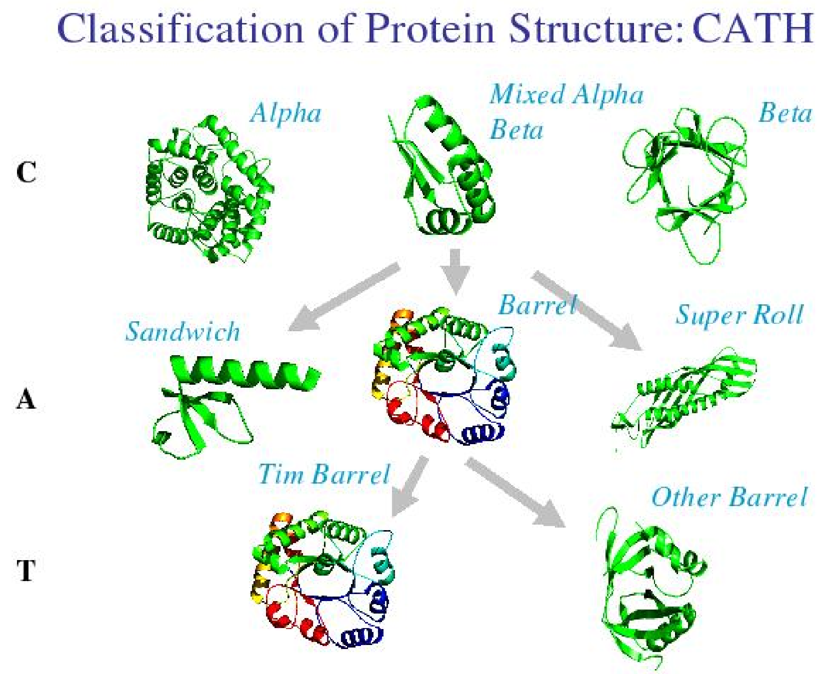
```
<Source>
  <Star>
    <Name> Sirius </Name>
    <Distance> 2.64 </Distance>
    <Brightness> -1.47 </Brightness>
    <Temperature> 9960 </Temperature>
  </Star>
</Source>
```

hierarchical model

- Data is organized in a tree structure
- Levels consist of records of the same type - same set of field values - with a sort field to ensure a particular order
- 1:N (parent-child) relationship between two record types: child may only have one parent but a parent can have many children
- Popular in the late 1960s/1970s with IBM's Information Management System (IMS)
- Structure of XML documents

hierarchical example

CATH database of protein structures in the Protein Data Bank:
Levels: Class, Architecture, Topology, Homologous Superfamily, Sequence Family



xml

- W3C standard markup language for structured data
- Hierarchical data model
- Consists of elements and attributes
- OS/Platform agnostic: just text
- Domain specific languages, e.g. MathML, MusicXML
- Validatable: DTD, XML Schema

xml example

```
<resource xsi:type="vs:DataCollection" created="2000-01-01T09:00:00" status="active">
    <title> The Catalogue of Palomar-Quest Transient Sources </title>
    <shortName> pqtrans </shortName>
    <identifier> ivo://nvo.caltech/pqtrans </identifier>
    <curation>
        <creator> Matthew Graham </creator>
        <version> 1.0 </version>
        <contact> mjg@caltech.edu </contact>
    </curation>
    <content>
        <subject> variable stars </subject>
        <description> This contains the transient sources discovered in the Palomar-Quest survey </description>
        <referenceURL> http://nvo.caltech.edu/catalogs/pqtrans </referenceURL>
    </content>
    <format> text/xml+votable </format>
    <format> text/plain+csv </format>
    <coverage> optical </coverage>
    <catalog>
        <table><column><name> ID </name>
            <description> Source identifier </description>
            <unit/>
        </column></table>
    </catalog>
</resource>
```

xpath

- W3C standard for pointing to elements, attributes and/or their values in an XML file
- v1.0 (1999); v2.0 (2007)
- Nodes identified by name and separated by '/'

```
/resource/content/description
```

- Predicate [. . .] : think 'where'

```
/resource[shortName = 'I/134/data']/identifier
```

Axis	Symbol	Example
attribute	@	@created
self	.	/identifier[. = '. . . ']
parent	..	../curation/publisher
descendant	//	//description

descendant-or-self	namespace
ancestor	ancestor-or-self
following	preceding
following-sibling	preceding-sibling

```
ancestor-or-self::description/identifier
```

operators

- and, or, not()

```
/resource[identifier = '...' and title = '...']
```

- +, -, *, div, mod
- =, !=, <, >, <=, >=,

```
/resource[@created > '2004-01-01T00:00:00']
```

- | (union)

```
/resource/content/description | /resource/catalog/table/description
```

functions

- concat(), substring(), contains(), substring-before(), substring-after(), translate(), normalize-space(), string-length()
- sum(), round(), floor(), ceiling()
- name(), local-name(), namespace-uri()
- position(), last(), count()
- string(), number(), boolean()

function examples

```
/resource[contains(shortName, 'data')]/identifier

/resource[substring-before(identifier, '://') = 'ivo']

/resource[string-length(normalize-space(shortName)) = 16]

/resource/format[2] (/resource/format[position() = 2])

/resource[count(format) > 1]

/resource/format[contains(preceding-sibling::format, 'xml')]
```

xpath 2.0

- Supports types - Built-in XML Schema and user-defined
- Expanded set of functions
- Sequences not node sets
- Conditional expressions
- Intersections and differences
- Subset of XQuery 1.0

Templates to convert XML into other forms

```
<xsl:template match="TABLE">
  <xsl:for-each select="FIELD">
    <xsl:value-of select="concat(@name, □"&amp;&amp;□"/>
  </xsl:for-each>
  <xsl:text>//</xsl:text>
  <xsl:for-each select="TR">
    <xsl:for-each select="TD">
      <xsl:value-of select="concat(., □"&amp;&amp;')"/>
    </xsl:for-each>
    <xsl:text>//</xsl:text>
  </xsl:for-each>
</xsl:template>
```

- W3C standard language for querying XML documents
- Does for XML documents what SQL does for tabular data
- Functional language - no statements
- No update mechanism

flwor

for - let - where - order by - return

```
for $x in (1 to 10) return $x * $x

for $res in /resource where contains($res/title, 'data')
return $res/identifier

let $pos := 2
return /resource/format[$pos]

for $res in /resource where matches($res/content/creator, "^M.*Graham$")
order by $res/@created
return $res
```

flwor example

```
declare namespace cs = "http://www.ivoa.net/xml/ConeSearch/v1.0";
let $res := /resource[capability/@xsi:type = 'cs:ConeSearch']
for $cs in $res where contains($cs/description, 'quasar')
order by $cs/identifier
return
<conesearch>
      <title> {string($cs/title)} </title>
      <url> {string($cs/capability/interface/accessURL)} </url>
</conesearch>
```

nested queries

```
let $res := /resource[@status = 'active']
return
<results>
    <hits> {count($re)} </hits>
    {for $res in subsequence($re, $offset, $hitsperpage)
        return
        <result>
            <identifier> {string($res/identifier)} </identifier>
            <title> {string($res/title)} </title>
         <summary> {local:summary($res/content/description)} </summary>
            {for $subject in $res/content/subject
                return
                <subject> {string($subject)} </subject>}
        </result>}
</results>
```

declare function *namespace:functionName*
($var1 (as *type1*), . . .) {. . .};

```
declare function local:double($x) {$x * 2};

declare function local:summary($description) {
    let $seq := tokenize(normalize-space($description), " ")
    return
    if (count($seq) > 50) then
        concat(string-join(subsequence($seq, 1, 50), " "), '...')
     else
        $seq
};
```

implementations

- C/C++: libxml2
- Java: Xalan, Saxon
- XQuery engine: DataDirect XQuery
- Open source native XML dbs: eXist, Sedna, Galax
- RDBMS with XML support: Virtuoso, MySQL (5.1+), PostgreSQL (8+), SQL Server 2005+

object-oriented model

- Adds database functionality to object-oriented languages by allowing persistent storage of programming objects
- Avoids overhead of converting information from database representation to application representation
- Applications require less code and use more natural data modelling
- Good for complex data and relationships between data
- Used by BaBar at SLAC (>1 PB of data)

object-relational model

- Adds new object storage capabilities to relational systems
- Objects, classes and inheritance are directly supported in database schemas and in the query language and supports extension of the data model with custom data-types and methods
- Allows management of complex object types such as time series and geospatial data as well as diverse binary media such as audio, video, images, and applets
- Leading vendors are Oracle and IBM

associative model

- Data is modelled as entities - having a discrete independent existence - and associations

- It is organised as items - an identifier, name and type - and links - an identifier, source, verb and target

Example:

Flight BA1234 arrived at LAX on 04-May-11 at 1:25pm

Items: Flight BA1234, LAX, 04-May-11, 1:25pm, arrived at, on, at

Links: (((Flight BA1234 arrived at LAX) on 04-May-11) at 1:25pm)

rdf

```
<rdf:RDF xmlns:vos="http://www.net.ivoa/xml/VOSpaceTypes-v1.0"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:tns1="urn://myprop#"
         xmlns:tns2="ivo://ivoa.net/vospace/properties/">
   <rdf:Description rdf:about="vos://nvo.caltech!vospace/mytable1">
      <n1 xmlns="urn://myprop#">value1</n1>
      <n2 xmlns="urn://myprop#">value2</n2>
    <datasize xmlns="ivo://ivoa.net/vospace/properties#">145Mb</datasize>
      <format xmlns="ivo://ivoa.net/vospace/properties#">jpeg</format>
   </rdf:Description>
</rdf:RDF>
```

sparql

```
PREFIX myprop: <urn://myprop\#>
PREFIX vos: <ivo://ivoa.net/vospace/properties\#>
SELECT DISTINCT ?ivoid
FROM <http://nvo.caltech.edu/vospace-1.1/find>
FROM <http://ast.cam.ac.uk/astrogrid/myspace/find>
WHERE \{?source myprop:n1 "value1" ;
               vos:format "jpeg" ;
               vos:ivoid ?ivoid . \}
LIMIT 25
```

- DBpedia (http://dbpedia.org)
- Virtuoso (http://demo.openlinksw.com/sparql_demo)

Turn large badly-formed tables into smaller, well-formed ones

- 1^{st} normal form: no repeating elements or groups of elements table has a unique key (and no nullable columns)
- 2^{nd} n.f.: no columns dependent on only part of key

Star Name	Constellation	Area

- 3^{rd} n.f.: no columns dependent on other non-key columns

Star Name	Magnitude	Flux

denormalized databases

- Normalization means lots of joins when performing queries
- Adding redundant data or grouping data will improve reads
- Denormalisation imposes heavy write cost with consistencies
- Various techniques to handle this:
  - -- Materialized views
  - -- Store writes in memory, sort and process and write sequentially to disk

where rdbms fail

RDBMS are great for small-medium volume applications:

- ease of use
- maturity
- flexibility
- powerful feature set

Scale up in terms of:

- dataset size
- read/write concurrency

decline of a rdbms

- Too many reads: add memcached to cache common queries -> reads not strictly ACID, cached data must expire
- Too many writes: scale vertically with beefed up hardware -> costly
- Too many joins: denormalize data to reduce joins
- Server swamped: stop any server-side computations
- Some queries still slow: prematerialize most complex queries, stop joining in most cases
- Writes getting ever slower: drop secondary indexes + triggers

"select fun, profit from real_world where relational = false"

Structured storage:

- document store
- graph (superset of triple store)
- key-value store (FB Cassandra)
- multivalue databases
- object database
- tabular (BigTable, HBase, Hypertable)
- tuple store

- Invented by Google fellows
- Based on functional programming map() and reduce() functions
- Canonical example is word count:

```
map(key:uri, value:text):
  for word in tokenize(value)
    emit(word, 1)

reduce(key:word type, value:list of 1s):
  emit(key, sum(value))
```

Apache framework for MapReduce

- HDFS: distributed file system
- HBase: column-based db (webtable)
- Hive: Pseudo-RDB with SQL
- Pig: Scripting language
- Zookeeper: Coordination service
- Whirr: Running cloud services
- Cascading: Pipes and filters
- Sqoop: RDB interface
- Mahout: ML/DM library