(1) What is Semi Structured data? XML document is a semi structured model. Justify with suitable example.

OR

Discuss: XML fulfill requirements for semi structured data model.

OR

How semi-structure data model resolves issues in existing web data management?

Semi-structured data model

- A data model, based on graphs, for representing both regular and irregular data.
- Semi-structured data models are meant to represent information from very structured to very unstructured kinds, and, in particular, irregular data.
- In a structured data model such as the relational model, one distinguishes between the type of the data (schema in relational terminology) and the data itself (instance in relational terminology).

Self-describing data

- The content comes with its own description; contrast with the relational model, where schema and content are represented separately.

Flexible typing

- Data may be typed (i.e., "such nodes are integer values" or "this part of the graph complies with this description"); often no typing, or a very flexible one.

Serialized form

- The graph representation is associated to a serialized form, convenient for exchanges in a heterogeneous environment.

1) Self-describing data

Starting point

- association lists, i.e., records of label-value pairs.

```
{name: "Alan", tel: 2157786, email: "agb@abc.com"}
```

Natural extension

- values may themselves be other structures:

```
{name: {first: "Alan", last: "Black"},
     tel: 2157786,
     email: "agb@abc.com"}
```

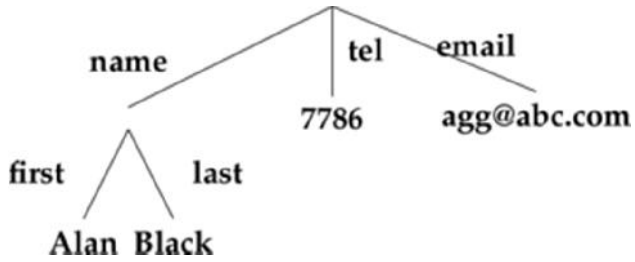
Further extension

- allow duplicate labels.

```
{name: "alan'', tel: 2157786, tel: 2498762 }
```

2) Tree-based representation

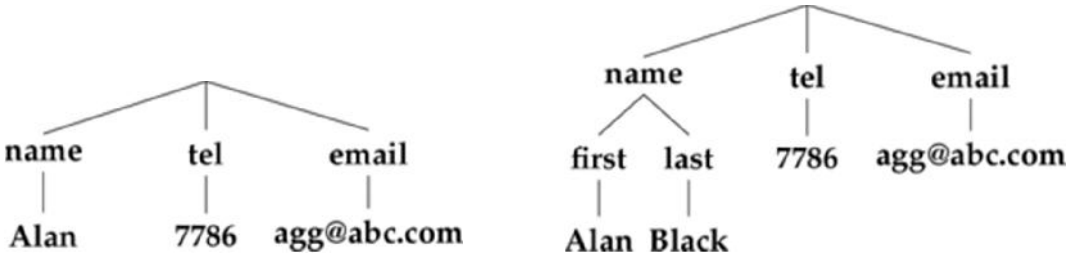
- Data can be graphically represented as trees.
- Label structure can be captured by tree edges, and values reside at leaves.



Tree representation, with labels on edges

Tree-based representation: labels as nodes

- Another choice is to represent both labels and values as vertices.



Tree representation, with labels as nodes

Representation of regular data

- The syntax makes it easy to describe sets of tuples as in:

```
{    person: {name: "alan", phone: 3127786, email: "alan@abc.com"},
     person: {name: "sara", phone: 2136877, email: "sara@xyz.edu"},
     person: {name: "fred", phone: 7786312, email: "fd@ac.uk"} }
```

Representation of irregular data

- Many possible variations in the structure: missing values, duplicates, changes, etc.

```
{ person: {name: "alan", phone: 3127786, email: "agg@abc.com"},
  person: &314
       {    name: {first: "Sara", last: "Green" },
            phone: 2136877,
            email: "sara@math.xyz.edu",
            spouse: &443 },
  person: &443
       {    name: "fred", Phone: 7786312, Height: 183,
            spouse: &314 }}
```

XML fulfill requirements for semistructured data model

- There have been different proposals for semistructured data models.
- They differ in choices such as: labels on nodes vs. on edges, trees vs. graphs, ordered trees vs. unordered trees.
- Most importantly, they differ in the languages they offer.
- Two quite popular models (at the time of writing) are XML, a de facto standard for exchanging data of any kind, and JSON ("JavascriptObject Notation"), an object serialization format mostly used in programming environments.
- XML, the Extensible Markup Language, is a semistructured data model that has been proposed as the standard for data exchange on the Web.
- XML meets the requirements of a flexible, generic, and platform-independent language, as presented earlier.
- Any XML document can be serialized with a normalized encoding, for storage or transmission on the Internet.
- It is well-established to use the term "XML document" to denote a hierarchically structured content represented with XML conventions.
- "document" means both the content and its structure, but not their specific representation which may take many forms.
- Also "document" is reminiscent of the SGML application area, which mostly focuses on representing technical documentation.
- An XML document is not restricted to textual, human-readable data, and can actually represent any kind of information, including images, of references to other data sources.
- XML is a standard for representing data but it is also a family of standards (some in progress) for the management of information at a world scale: XLink, XPointer, XML Schema, DOM, SAX, XPath, XSL, XQuery, SOAP, WSDL, and so forth.

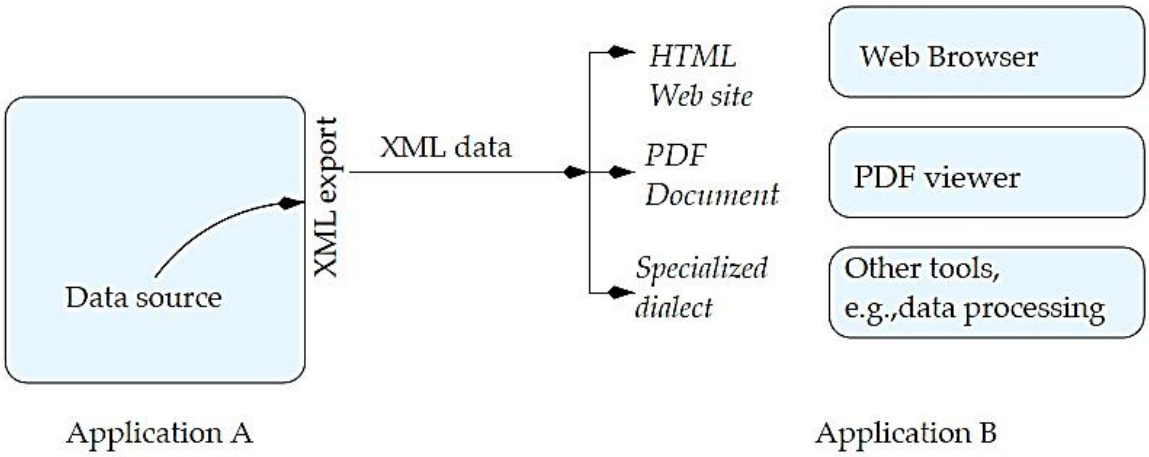
(2) Explain in Detail Web Data Management with XML.

- XML is a very flexible language, designed to represent contents independently from a specific system or a specific application.
- These features make it the candidate of choice for data management on the Web.
- XML enables data exchange and data integration, and it does so universally for (almost) all the possible application realms, ranging from business information to images, music, biological data, and the like.

Following are the two simple scenarios showing typical distributed applications based on XML that exploit exchange and integration:

1) Data exchange

- The typical flow of information during XML-based data exchange is illustrated in below Figure.



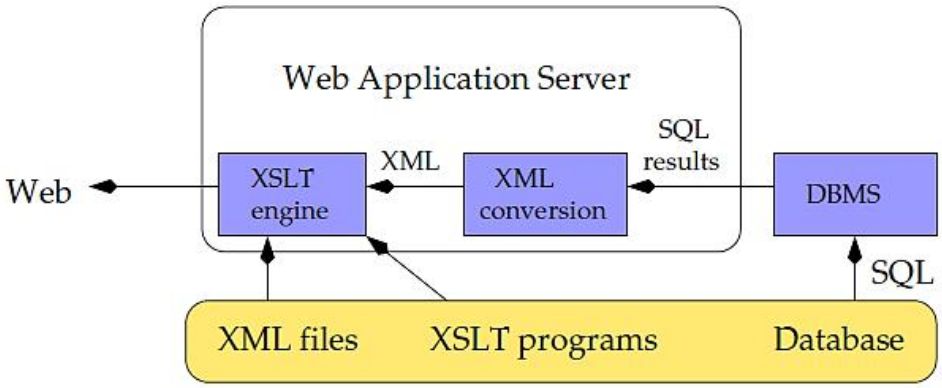
Flow of information in XML-based data exchange

- Application A manages some internal data, using some specialized data management software, (e.g., a relational DBMS).
- Exchanging these data with another application B can be motivated either for publication purposes, or for requiring from B some specialized data processing.
- The former case is typical of web publishing frameworks, where A is a web server and B a web client (browser, mobile phone, PDF viewer, etc.).
- The later case is a first step towards distributed data processing, where a set of sites (or "peers") collaborate to achieve some complex data manipulation.
- XML is at the core of data exchange. Typically, A first carries out some conversion process (often called "XML publishing") which produces an appropriate XML representation from the internal data source(s.)
- These XML data are then consumed by B which extracts the content, processes it, and possibly returns an XML-encoded result.

Several of the afore mentioned features of XML contribute to this exchange mechanism:

1. Ability to represent data in a serialized form that is safely transmitted on the Web;
2. Typing of document, which allows A and B to agree on the structure of the exchanged content;
3. Standardized conversion to/from the serialized representation and the specific tree based representation respectively manipulated by A and B.

A real Web Publishing architecture would be as shown in the below figure.

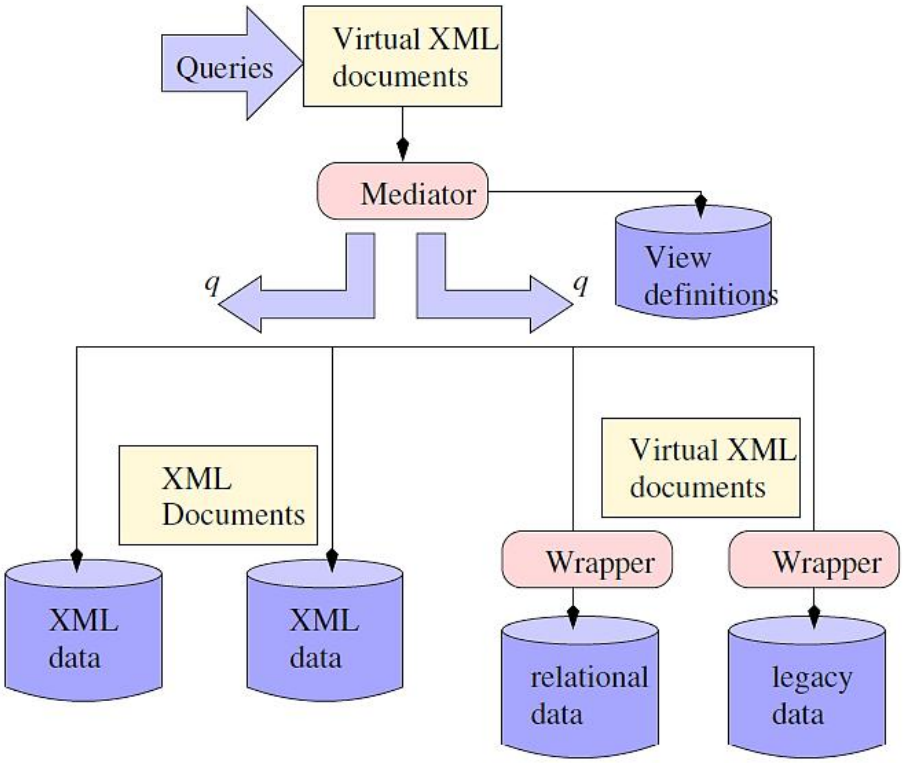


Software architecture for Web publishing applications

- Consider an example of application called Shows for publishing information about movie showings, in a Web site and in a Wap site.
- The application uses a relational database.
- Data are obtained from a relational database as well as directly from XML files.
- Some specialized programs, written with XSLT (the XML transformation language) are used to restructure the XML data, either coming from files, from the database through a conversion process, or actually from any possible data source.
- This is the XML publishing process mentioned above.
- It typically produces XHTML pages for a Web site.
- These pages are made available to the world by a Web server.
- The data flow, with successive transformations (from relational database to XML; from XML to a publication dialect), is typical of XML-based applications, where the software may be decomposed in several modules, each dedicated to a particular part of the global data processing.
- Each module consumes XML data as input and produces XML data as output, thereby creating chains of data producers/consumers.
- Ultimately, there is no reason to maintain a tight connection of modules on a single server.
- Instead, each may be hosted on a particular computer somewhere on the Internet, dedicated to providing specialized services.

2) Data integration

- A typical problem is the integration of information coming from heterogeneous sources.
- XML provides some common ground where all kinds of data may be integrated.



- For each (non-XML) format, one provides a wrapper that is in charge of the mapping from the world of this format to the XML world.
- Now a query (say an XQuery) to the global XML view is transformed by the mediator (using the view definitions) into queries over the local sources.
- A source wrapper translates the XML query to the source it receives into a query the source understands.
- That query is evaluated on the source, and some data are produced.
- The wrapper translates this data into XML data.
- The mediator combines the result it receives from all the wrappers to obtain the general result.

(3) Write short note on FLWOR expression with suitable example. Discuss advanced features of FLWOR with example.

- The most powerful expressions in XQuery. A FLWOR ("flower") exp.:
  - iterates over sequences (for);
  - defines and binds variables (let);
  - apply predicates (where);
  - sort the result (order);
  - construct a result (return).
- An example (without let):

```
for $m in collection('movies')/movie
where $m/year >= 2005
return <film>{$m/title/text()},
   (director: {$m/director/last_name/text()}) </film>
```

FLWOR expressions and XPath

- In its simplest form, a FLWR expression provides just an alternative to XPath expressions.

For instance:

```
let $year:=1960
for $a in doc('SpiderMan.xml')//actor
where $a/birth_date>=$year
return $a/last_name
```

- is equivalent to the XPath expression.

```
//actor[birth_date>=1960]/last_name
```

- Not all FLWOR expressions can be rewritten with XPath.

A complex FLWOR example

- "Find the description and average price of each red part that has at least 10 orders" (assume collections parts.xml and orders.xml):

```
for $p in doc("parts.xml")//part[color = "Red"]
let $o := doc("orders.xml")//order[partno = $p/partno] where
count($o) >= 10
order by count($o) descending return
<important_red_part> { $p/description }
<avg_price> {avg($o/price)} </avg_price> </important_red_part>
```

for and let

- Both clauses bind variables. However:
  - for successively binds each item from the input sequence.

    ```
    for $x in /company/employee
    ```

    binds each employee to $x, for each item in the company sequence.
  - let binds the whole input sequence.

    ```
    let $x := /company/employee
    ```

    binds $x to all the employees in company.
- Note the for may range over an heterogeneous sequence:

```
for $a in doc("Spider-Man.xml")//*
where $a/birth_date >= 1960
return $a/last_name
```

- Here, $a is bound in turn to all the elements of the document! (Does it work? Yes!)

for + return = an expression!

- The combination **for** and **return** defines an expression: **for** defines the input sequence, **return** the output sequence.

  A simple loop:

```
for $i in (1 to 10) return $i
```

  Nested loops:

```
for $i in (1 to 10) return
   for $j in (1 to 2) return $i * $j
```

  Syntactic variant:

```
for $i in (1 to 10),
   $j in (1 to 2) return $i * $j
```

  Combination of loops:

```
for $i in (for $j in (1 to 10) return $j * 2) return $i * 3
```

Defining variables with let

- **let** binds a name to a value, i.e., a sequence obtained by any convenient mean, ranging from literals to complex queries:

```
let $m := doc("movies/Spider-Man.xml")/movie
return $m/director/last_name
```

- A variable is just a synonym for its value:

```
let $m := doc("movies/Spider-Man.xml")/movie
for $a in $m/actor
return $a/last_name
```

- The scope of a variable is that of the FLWR expression where it is defined. Variables cannot be redefined or updated within their scope.

The where clause

- **where** is quite similar to its SQL synonym. The difference lies in the much more flexible structure of XML documents.
- "Find the movies directed by M. Allen"

```
for $m in collection("movies")/movie
where $m/director/last_name="Allen"
return $m/title
```

- Looks like a SQL query? Yes but predicates are interpreted according to the XPath rules:
  - if a path does not exists, the result is **false**, no typing error!
  - if a path expression returns several nodes: the result is true if there is at least one match.
- "Find movies with Kirsten Dunst" (note: many actors in a movie!)

```
for $m in collection("movies")/movie
where $m/actor/last_name="Dunst"
return $m/title
```

The return clause

- **return** is a mandatory part of a FLWR expression. It is instantiated once for each binding of the variable in the **for** clause.

```
for $m in collection("movies")/movie let $d :=
$m/director
where $m/actor/last_name="Dunst" return
  <div>
   {$m/title/text(), "directed by", $d/first_name/text(),
       $d/last_name/text()},
     with <ol>
      {for $a in $m/actor
        return <li>{$a/first_name, $a/last_name, " as ",
                     $a/role}</li>
      } </ol>
      </div>
```

Joins

- Nested FLWOR expressions makes it easy to express joins on document, à la SQL:

```
for $p in doc("taxpayers.xml")//person
  for $n in doc("neighbors.xml")//neighbor where $n/ssn
  = $p/ssn
  return
<person>
  <ssn> { $p/ssn } </ssn>
     { $n/name }
  <income> { $p/income } </income>
</person>
```

Join and grouping

- "Get the list of departments with more than 10 employees, sorted on the average salary"

```
for $d in doc("depts.xml")//deptno
 let $e := doc("emps.xml")//employee[deptno=$d] where count($e)
  >= 10
  order by avg($e/salary) descending return <big-dept>
     { $d, <headcount>{count($e)}</headcount>,
       <avgsal>{avg($e/salary)}</avgsal>
      }
</big-dept>
```

(4) Explain XML evaluation techniques.

Techniques for the efficient evaluation of XML queries, and in particular for tree pattern queries.

1) Structural join

- The first techniques concern structural joins and can be seen as foundational to all the others.
- Structural joins are physical operators capable of combining tuples from two inputs, much in the way regular joins in the relation case do, but based on a structural condition (thus the name).
- Formally, let p_1 and p_2 be some partial evaluation plans in an XML database, such that attribute X in the output of p_1, denoted $p_1.X$, and attribute Y from the output of p_2, denoted $p_2.Y$, both contain structural IDs.
- Let $\prec$ denote the binary relationship "isParentOf" and $\prec\prec$ denote the binary relationship "isAncestorOf".
- Formally then, the structural join of p_1 and p_2 on the condition that $p_1.X$ be an ancestor of $p_2.Y$ is defined as:

$$p_1 \bowtie_{X \prec\prec Y} p_2 = \{(t_1, t_2) \mid t_1 \in p_1, t_2 \in p_2, t_1.X \prec\prec t_2.Y\}$$

- and the structural join on the parent relation $\prec$ is similarly defined by:

$$p_1 \bowtie_{X \prec Y} p_2 = \{(t_1, t_2) \mid t_1 \in p_1, t_2 \in p_2, t_1.X \prec t_2.Y\}$$

Nested loop join

- The simplest physical structural join algorithms could proceed in nested loops.
- One iterates over the output of p1 and for each tuple, one iterates over the output of p2.
- However, this leads to CPU costs in O(|p1|x|p2|), since each p1 tuple is compared with each p2 tuple.

Hash join

- As in traditional relational database settings, one could consider hash joins that are often called upon for good performance, given that their CPU costs are in O(|p1|+|p2|).
- However, hash-based techniques cannot apply here, because the comparisons that need to be carried are of the form "is id1 an ancestor of id2?", which do not lead themselves to a hash-based approach.

Stack-based join

- To efficiently perform this task, Stack-based structural joins operators have been proposed originally for (start, end) ID scheme.
- They can be used for other labelling schemes as well.

2) Optimizing structural join queries

- Algorithm STD (Stack-Tree Descendant) allows combining two inputs based on a structural relationship between an ID attribute of one plan and an ID attribute of the other.
- Using STD and the similar Stack-Tree Ancestor (STA), one can compute matches for larger query tree patterns, by combining sets of identifiers of nodes having the labels appearing in the query tree pattern.
- In general, it turns out that for any tree pattern query, there exist some plans using the STD and STA physical operators, and which do not require any Sort (also called fully pipelined plans).

- However, one cannot ensure a fully pipelined plan for a given join order.
- This complicates the problem of finding an efficient evaluation plan based on structural joins, because two optimization objectives are now in conflict:
    - avoiding Sort operators (to reduce the time to the first output, and the total running time) and
    - Choosing the join order that minimizes the sizes of the intermediary join results.

3) Holistic twig joins

- The previous Section showed; how one can reduce the total running time by avoiding sort operators and reduces the total running time and/or the size of the intermediary join results, by choosing the order in which to perform the joins.
- A different approach toward the goals of reducing the running time and the size of the intermediary results consists in devising a new (logical and physical operator), more precisely, an n-ary structural join operator, also called a holistic twig join.
- Such an operator builds the result of a tree pattern query in a single pass over all the inputs in parallel.
- This eliminates the need for storing intermediary results and may also significantly reduce the total running time.

PathStack Algorithm (for linear queries)

- PathStack works by continuously looking for the input operator (let's call it iop_{min}, corresponding to the query node n_{min} labeled a_{min}) whose first ID has the smallest *pre* number among all the input streams.
- This amounts to finding the first element (in document order) among those not yet processed, across all inputs.
- Let's call this element e_{min}.
- Once e_{min} has been found, PathStack inspects all the stacks, looking for nodes of which it can be guaranteed that they will not contribute to further query results.
- In particular, this is the case for any nodes preceding e_{min}, i.e., ending before the start of e_{min}.
- It is easy to see that such nodes cannot be ancestors neither of e_{min}, nor of any of the remaining nodes in any of the inputs, since such nodes have a starting position even bigger than e_{min}'s.
- PathStack pops all such entries, from all stacks.
- PathStack then pushes the current nmin entry on $s_{a_{min}}$, if and only if suitable ancestors of this element have already been identified and pushed on the stack of a^p_{min}, the parent node of amin in the query.
- If this is the case and a new entry is pushed on $s_{a_{min}}$, importantly, a pointer is stored from the top entry in $s_{a^p_{min}}$, to the new (top) entry in $s_{a_{min}}$
- Such pointers record the connections between the stack entries matching different query nodes, and will be used to build result tuples out.
- Finally, PathStack advances the input operator iop_{min} and resumes its main loop, identifying again the input operator holding the first element (in document order) not yet processed etc.

- When an entry is pushed on the stack corresponding to the query leaf, it is certain that the stacks contain matches for all its ancestors in the query, matches that are ancestors of the leaf stack entry.
- At this point, two steps are applied:
    - Result tuples are built out of the entries on the stacks, and in particular of the new entry on the stack of the query leaf;
    - This new entry is popped from the stack.

TwigStack Algorithm

- This algorithm generalizes PathStack with support for multiple branches.
- The ideas are very similar, as the main features (no intermediary results and space-efficient encoding of twig query results) are the same.
- if the query pattern edges are only of type ancestor/descendant, TwigStack is I/O and CPU optimal among all sequential algorithms that read their inputs in their entirety

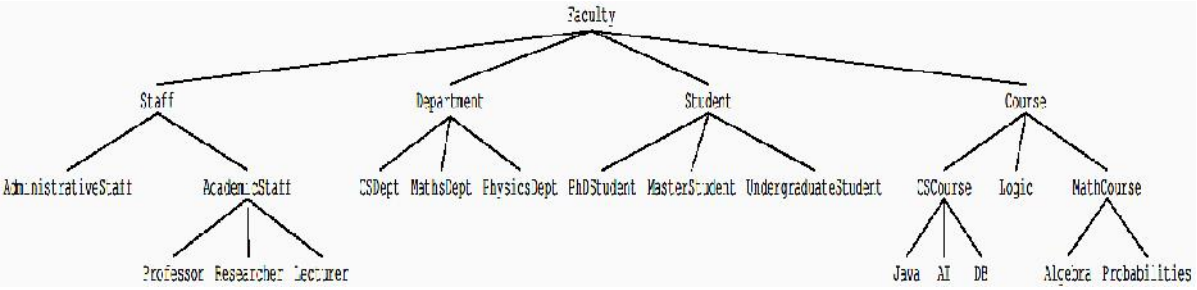
(5) What is Ontology? Describe RDF, RDFS in Brief.

Ontologies

- Formal descriptions providing human users a shared understanding of a given domain.
    - A controlled vocabulary.
- Formally defined so that it can also be processed by machines.
- Logical semantics that enables reasoning.
- Reasoning is the key for different important tasks of Web data management, in particular
    - to answer queries (over possibly distributed data).
    - to relate objects in different data sources enabling their integration.
    - to detect inconsistencies or redundancies.
    - to refine queries with too many answers, or to relax queries with no answer.

Classes and class hierarchy

- Backbone of the ontology
- `AcademicStaff` is a Class (A class will be interpreted as a set of objects)
- `AcademicStaff` isa Staff (isa is interpreted as set inclusion)



Relations

- Declaration of **relations** with their **signature** (Relations will be interpreted as binary relations between objects)

`TeachesIn(AcademicStaff, Course)`

if one states that "*X* `TeachesIn` *Y*", then *X* belongs to `AcademicStaff` and *Y* to `Course`,

`TeachesTo(AcademicStaff, Student),`

`Leads(Staff, Department)`

Instances

- Classes have **instances**
- `Dupond` is an instance of the class `Professor`
- it corresponds to the fact: `Professor(Dupond)`
- Relations also have **instances**
- `(Dupond,CS101)` is an instance of the relation `TeachesIn`
- it corresponds to the fact: `TeachesIn(Dupond,CS101)`
- The instance statements can be seen as (and stored in) a **database**

Ontology = schema + instance

- Schema
  - The set of class and relation names
  - The signatures of relations and also constraints
  - The constraints that are used for two purposes
    - checking data consistency (like dependencies in databases)
    - inferring new facts
- Instance
  - The set of facts
  - The set of base facts together with the inferred facts should satisfy the constraints
- Ontology (i.e., Knowledge Base) = Schema + Instance

3 ontology languages for the Web

- RDF: a very simple ontology language
- RDFS: Schema for RDF
  - Can be used to define richer ontologies
- OWL: a much richer ontology language

RDF: Resource Description Framework

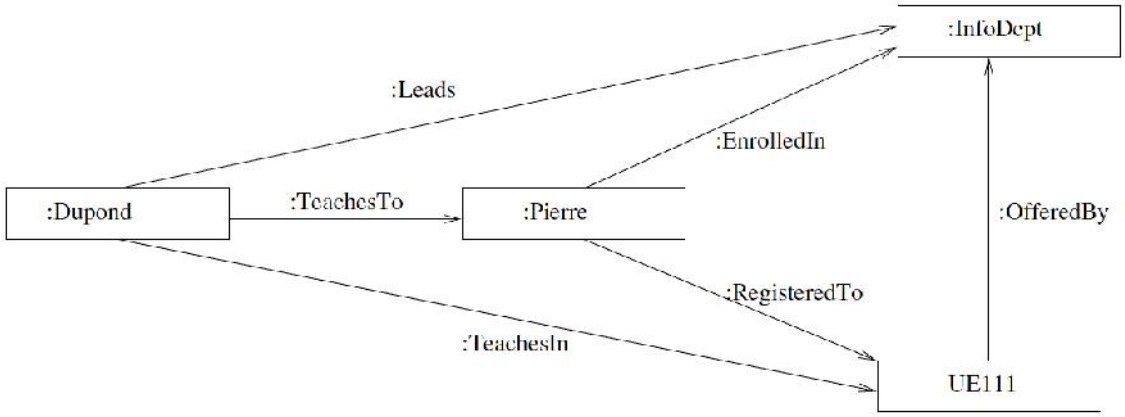
- RDF facts are triplets

```
<:Dupond :Leads :CSDept >
<:Dupond :TeachesIn :UE111 >
<:Dupond :TeachesTo :Pierre >
<:Pierre :EnrolledIn :CSDept >
<:Pierre :RegisteredTo :UE111 >
<:UE111 :OfferedBy :CSDept >
```

RDF graph

- A set of RDF facts defines

- a set of relations between objects
- an RDF graph where the nodes are objects:



RDF semantics

- A triplet hs P oi is interpreted in first-order logic (FOL) as a fact P (s, o)
- Example:

```
Leads(Dupond, CSDept)
TeachesIn(Dupond, UE111)
TeachesTo(Dupond,Pierre)
EnrolledIn(Pierre, CSDept)
RegisteredTo(Pierre, UE111)
OfferedBy(UE111, CSDept)
```

RDFS: RDF Schema

- the schema in RDF is super simplistic
- An RDF Schema defines the schema of a richer ontology

RDF Schema

- Do net get confused: RDFS can use RDF as syntax, i.e., RDFS statements can be themselves expressed as RDF triplets using some specific properties and objects used as RDFS keywords with a particular meaning.
- Declaration of classes and subclass relationships

```
< Staff rdf:type rdfs:Class >
< Java rdfs:subClassOf CSCourse >
```

- Declaration of instances (beyond the pure schema)

```
< Dupond rdf:type AcademicStaff >
```

- Declaration of relations (properties in RDFS terminology)

```
< RegisteredTo rdf:type rdf:Property >
```

- Declaration of subproperty relationships

```
< LateRegisteredTo rdfs:subPropertyOf RegisteredTo >
```

- Declaration of domain and range restrictions for predicates

```
< TeachesIn rdfs:domain AcademicStaff >
< TeachesIn rdfs:range Course >
TeachesIn( AcademicStaff , Course)
```

RDFS logical semantics

RDF and RDFS statements	FOL translation	DL notation
< i rdf:type C > < i P j >	$C(i)$ $P(i, j)$	$i : C$ or $C(i)$ $i\ P\ j$ or $P(i, j)$
< C rdfs:subClassOf D > < P rdfs:subPropertyOf R > < P rdfs:domain C > < P rdfs:range D >	$\forall X (C(X) \Rightarrow D(X))$ $\forall X \forall Y (P(X, Y) \Rightarrow R(X, Y))$ $\forall X \forall Y (P(X, Y) \Rightarrow C(X))$ $\forall X \forall Y (P(X, Y) \Rightarrow D(Y))$	$C \sqsubseteq D$ $P \sqsubseteq R$ $\exists P \sqsubseteq C$ $\exists P^{-} \sqsubseteq D$

(6) Describe Failure management in distributed system.

OR

What is Distributed System? Discuss failure management in brief.

- A distributed system is an application that coordinates the actions of several computers to achieve a specific task.
- This coordination is achieved by exchanging messages which are pieces of data that convey some information.
  - "shared-nothing" architecture -> no shared memory, no shared disk.
- The system relies on a network that connects the computers and handles the routing of messages.
  - Local area networks (LAN), Peer to peer (P2P) networks.
- Client (nodes) and Server (nodes) are communicating software components:
  - Assimilate them with the machines they run on.

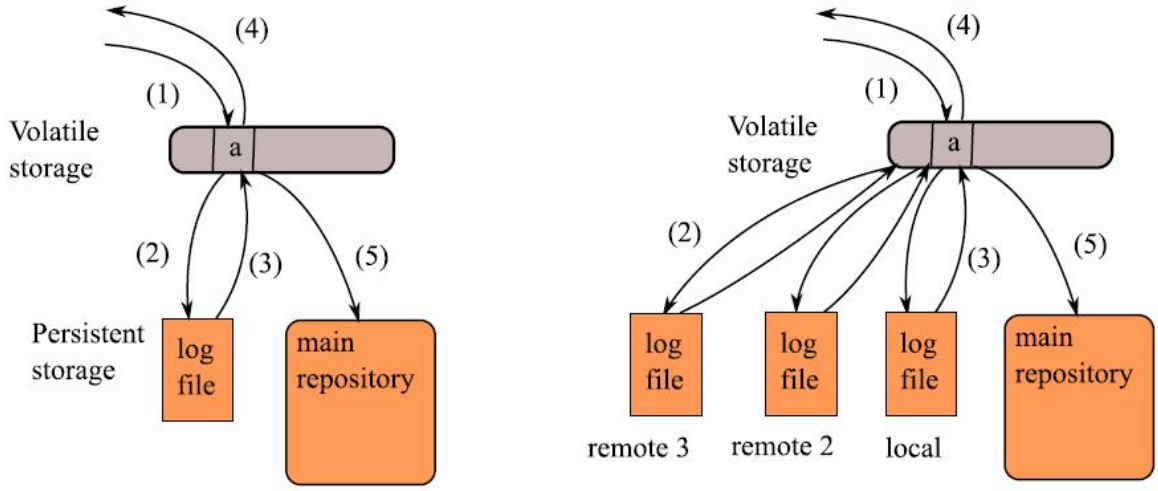
Failure management

- Common principles:
  1. The state of a (data) system is the set of item committed by the application.
  2. Updating "in place" is considered as inefficient because of disk seeks.
  3. Instead, update is written in main memory and in a sequential log file.
  4. Failure? The main memory is lost, but all the committed transactions are in the log: a REDO operation is carried out when the system restarts.
- Implemented in all DBMSs.

Failure and distribution

- First: when do we know that a component is in failed state?
  - Periodically send message to each participant.
- Second: does the centralized recovery still hold?
- Yes, providing the log file is accessible.

Failure recovery



Recovery techniques for centralized (left) and replicated architectures (right)

- Distributed logging can be asynchronous (efficient, risky) or asynchronous (just the opposite).
- Figure recalls the main aspects of data recovery in a centralized data management system, and its extension to distributed settings.
- Consider first a client-server application with a single server node (left part).
    - (1) The Client issues a write(a).
    - The server does not write immediately **a** in its repository.
    - Because this involves a random access, it would be very inefficient to do so.
    - Instead, it puts a in its volatile memory.
    - Now, if the system crashes or if the memory is corrupted in any way, the write is lost.
    - Therefore, the server writes in a log file (2).
    - A log is a sequential file which supports very fast append operations.
    - When the log manager confirms that the data is indeed on persistent storage (3), the server can send back an acknowledgment to the Client (4).
    - Eventually, the main memory data will be flushed in the repository (5).
- This is standard recovery protocol, implemented in centralized DBMSs.
- In a distributed setting, the server must log a write operation not only to the local log file, but also to 1, 2 or more remote logs.
- The issue is close to replication methods, the main choice being to adopt either a synchronous or asynchronous protocol.

Synchronous protocol

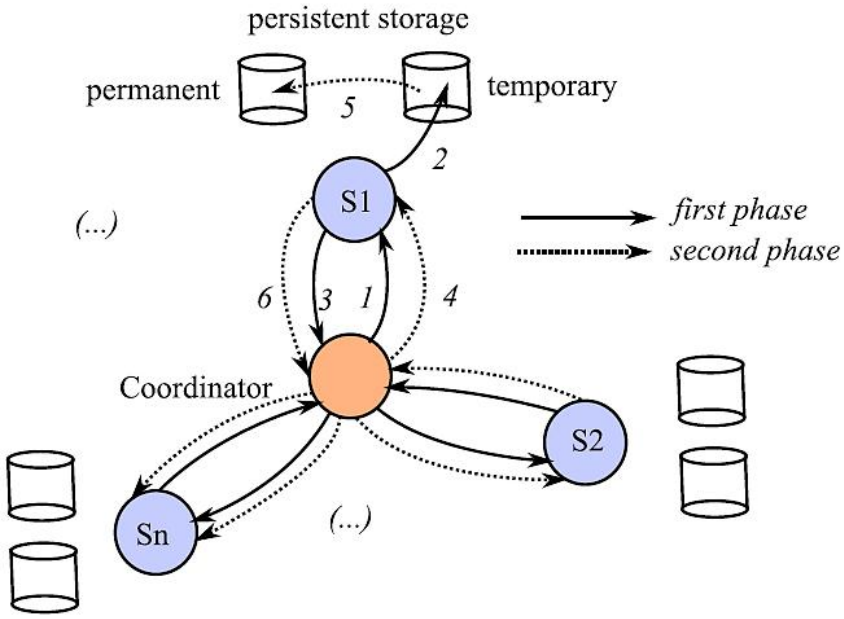
- The server acknowledges the Client only when all the remote nodes have sent a confirmation of the successful completion of their write() operation.

Asynchronous protocol

- The Client application waits only until one of the copies (the fastest) has been effectively written.
- Clearly, this puts a risk on data consistency, as a subsequent read operation may access an older version that does not yet reflect the update.

Distributed transactions

- A transaction is a sequence of data update operations that is required to be an "all-or-nothing" unit of work.
- That is, when a commit is requested, the system has to perform all the updates in the transaction. The transaction has been validated.
- In case of problem, the system has also the option to perform nothing of it. The transaction has been aborted.
- On the other hand, the system is not allowed to perform some of the updates and not others, i.e., partial validation is forbidden.
- The main algorithm that is used to achieve this goal is the two-phase commit (2PC) protocol:
    - first, the coordinator asks each participant whether it is able to perform the required operation with a Prepare message;
    - second, if all participants answered with a confirmation, the coordinator sends a Decision message: the transaction is then committed at each site.



The two-phase commit protocol (details are given for the Coordinator-Server 1 communications only)

(7) What is Ontology? Explain Ontology-based mediators.

OR

Explain Global-as-view (GAV) and Local-as-view (LAV) mediation with example.

Ontology

- Ontology is a formal description providing human users or machines a shared understanding of a given domain.
- Because of the logic inside, one can reason with ontologies, which is key tool for integrating different data sources, providing more precise answers, or (semi automatically) discovering and using new relevant resources.
- The main advantage of GAV is its conceptual and algorithmic simplicity.
- The global schema is simply defined using views over the data sources and specifies how to obtain tuples of the global relation G_i from tuples in the sources.

Two main approaches

- Ask queries to a global schema
- To answer, use data over local schemas
- In the two approaches, formulas relate the local schemas to the global schema
- Warehousing approach
    - Global instance is materialized
    - Data is transformed from the local instances and loaded in global instance
    - Queries are evaluated at the global instance
- Mediating approach
    - Global instance is virtual
    - Queries are evaluated using queries to the local instances

Data integration in the two approaches

- user query against a global schema that integrates
    - data source 1
    - data source 2
    - a data integration system that integrates
        - data source 3
        - data source 4
        - a data integration system that integrates
    - a data integration system that integrates
        - data source 5
        - data source 6
        - a data integration system that integrates

Mediation: Underlying principles

- Define a mediated schema (also called a global schema) that serves for the query interface for users.
- Declare the data sources: mappings between the global schema and the schemas of the local data sources.

Two approaches:

- Global-As-Views (GAV) approach: the global relations are defined as views over the local relations
- Local-As-Views (LAV) approach: the local relations are defined as views over the global relations

The Global-As-Views approach: example

- 4 local schemas

- S1: a catalogue of teaching programs of (some) French universities
  - S1.Catalogue(nomUniv, programme)
- S2: Erasmus students enrolled in courses of (some) European
- universities
  - S2.Erasmus(student, course, univ)
- S3: Foreign students enrolled in programs of (some) French universities
  - S3.CampusFrance(student, program, university)
- S4: the course content of (some) international master programs
  - S4.Mundus(programTitle,course)

The global instance is a view of the local instances

- University (U): S1.Catalogue(U,P) ∨ S2.Erasmus(N,C,U) ∨ S3.CampusFrance(N',P',U)
- MasterStudent (N): S2.Erasmus(N,C,U), S4.Mundus(P,C) ∨ S3.CampusFrance(N,P',U'),S4.Mundus(P',C')
- MasterCourse (C): S4.Mundus(P,C)
- MasterProgram(P): S4.Mundus(P,C)
- EnrolledIn (N,P): S2.Erasmus(N,C,U), S4.Mundus(P,C) ∨ S3.CampusFrance(N,P,U'),S4.Mundus(P,C')
- RegisteredTo(N,U): S3.CampusFrance(N,P,U),

Semantics of GAV mappings

- MasterStudent (N)
  - `S2.Erasmus(N,C,U), S4.Mundus(P,C) ∨ S3.CampusFrance(N,P',U'),S4.Mundus(P',C')`
- Exact semantics (typically)
  - `∀ N [ (∃C∃U ∃P (S2.Erasmus(N,C,U) ∧ S4.Mundus(P,C)) ∨ (∃C'∃U' ∃P' ( S3.CampusFrance(N,P',U'), S4.Mundus(P',C'))) ⇔ MasterStudent (N) ]`
- Sound semantics
  - `∀ N [ (∃C∃U ∃P (S2.Erasmus(N,C,U) ∧ S4.Mundus(P,C)) ∨ (∃C'∃U' ∃P' ( S3.CampusFrance(N,P',U'), S4.Mundus(P',C'))) ⇒ MasterStudent (N) ]`

Query rewriting by unfolding

- The semantics tells how to populate the global relations
- A logical query plan for a given query is obtained by unfolding each atom in the query
- This leads to conjunctions of disjunctions
- Transform to disjunctions of conjunctions to obtain a set of conjunctive querie
- Example: q(N): MasterStudent(N)
  - `q(N): S2.Erasmus(N,C,U), S4.Mundus(P,C) ∨ S3.CampusFrance(N,P',U'), S4.Mundus(P',C')`
- Transform:
  - `q(N): S2.Erasmus(N,C,U), S4.Mundus(P,C)`
  - `q(N): S3.CampusFrance(N,P',U'),S4.Mundus(P',C')`

Main limitation of the GAV approach

- Adding or removing data sources requires to revise all the GAV mappings defining the global schema

- When a new data source arrives, we must consider how it may be combined with all the existing data sources to produce tuples of any global relation
- The Local-As-Views (LAV) approach
    - The mediated schema is designed to remain stable even when data sources join or leave the integration system
    - Only incremental changes for the sources that join/leave with no impact on the rest

The LAV approach

- Starts with a mediated schema, i.e., a set of global relations
- Example: Global schema
    - `Student(studentName),..., University(uniName)`
    - `Program(title), MasterProgram(title), Course(code)`
    - `EnrolledInProgram(studentName,title)`
    - `EnrolledInCourse(studentName,code), PartOf(code,title)`
    - `RegisteredTo(studentName, uniName)`
    - `OfferedBy(title, uniName)`

LAV mappings: defines the local relations

- Local as views: the local relations are defined as views of the global
- relations
- Example
    - S1.Catalogue(U,P):
        - FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P')
    - S2.Erasmus(S,C,U):
        - Student(S), EnrolledInCourse(S,C), PartOf(C,P),
        - OfferedBy(P,U), EuropeanUniversity(U), RegisteredTo(S,U') EuropeanUniversity(U'), U6=U'
    - S3. CampusFrance(S,P,U):
        - NonEuropeanStudent(S), EnrolledInProgram(S,P), Program(P), Offeredby(P,U), FrenchUniversity(U), RegisteredTo(S,U)
    - S4.Mundus(P,C):
        - MasterProgram(P), OfferedBy(P,U), OfferedBy(P,U'), EuropeanUniversity(U), NonEuropeanUniversity(U), PartOf(C,P)

Semantics of the LAV mappings

- S1.Catalogue(U,P):
    - FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P')
- Sound semantics (typically)
    - $\forall U \forall P$ [S1.Catalogue(U,P) $\Rightarrow \exists$ P' (FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P'))]
- Exact semantics
    - $\forall U \forall P$ [S1.Catalogue(U,P) $\Leftrightarrow \exists$ P' (FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P'))]

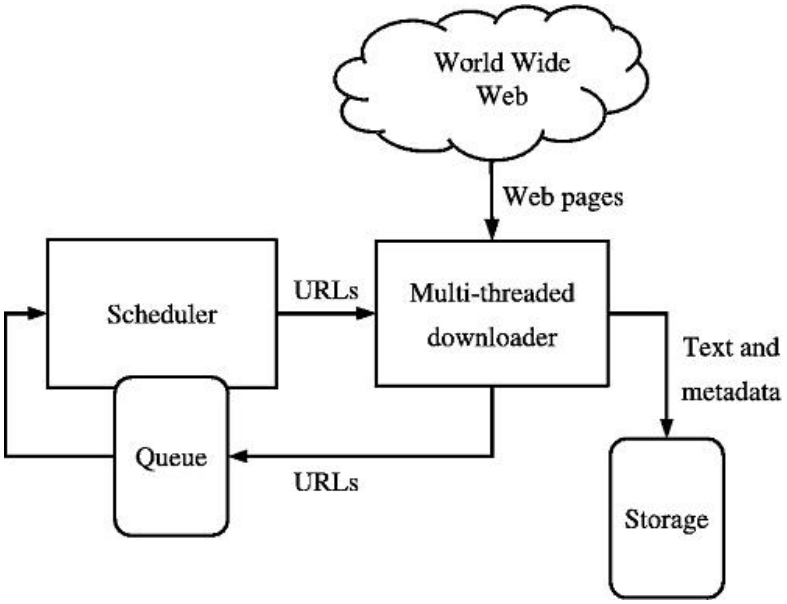
LAV vs. GAV

- With GAV: query processing is simple

  - Building the rewriting in LAV requires more work than the simple unfolding of the GAV approach
- With LAV: more flexibility and robustness
  - We can define the global schema without knowing the sources
  - We can define the mapping for one source without knowing the others
  - Allows a fine-grained description of the data sources, and a loose coupling between local and global relations
  - E.g.: if we are interested in Master students, we do not need to know in advance how to join the available data sources to obtain them like in the GAV approach; we just define them as a global query
  - MasterStudent(S):
    - Student(S), EnrolledInProgram(S,P), MasterProgram(P)

(8) Write short note on Web Crawler

Web Crawlers



- Crawlers, (Web) spiders, (Web) robots: autonomous user agents that retrieve pages from the Web.
- Basics of crawling:
  1. Start from a given URL or set of URLs
  2. Retrieve and process the corresponding page
  3. Discover new URLs (cf. next slide)
  4. Repeat on each found URL
- No real termination condition (virtual unlimited number of Web pages!)
- Graph-browsing problem
  - deep-first: not very adapted, possibility of being lost in robot traps
  - breadth-first
  - combination of both: breadth-first with limited-depth deep-first on each discovered website
- Sources of new URLs

- From HTML pages:

```
hyperlinks <a href="...">...</a>
media <img src="..."> <embed src="..."> <object data="...">
frames <frame src="..."> <iframe src="...">
JavaScript links window.open("...")
```

  etc.
- Other hyperlinked content (e.g., PDF files)
- Non-hyperlinked URLs that appear anywhere on the Web (in HTML text, text files, etc.): use regular expressions to extract them
- Referrer URLs Sitemaps

Scope of a crawler

- Web-scale
- The Web is infinite! Avoid robot traps by putting depth or page number limits on each Web server
- Focus on important pages.
- Web servers under a list of DNS domains: easy filtering of URLs
- A given topic: focused crawling techniques based on classifiers of Web page content and predictors of the interest of a link.
- The national Web (cf. public deposit, national libraries): what is this?
- A given Web site: what is a Web site?

Identification of duplicate Web pages

- Problem
  - Identifying duplicates or near-duplicates on the Web to prevent multiple indexing
  - trivial duplicates: same resource at the same canonized URL:
    - http://example.com:80/toto
    - http://example.com/titi/../toto
  - exact duplicates: identification by hashing
  - near-duplicates: (timestamps, tip of the day, etc.) more complex!

Crawling ethics

- Standard for robot exclusion: robots.txt at the root of a Web server.
  - User-agent: *
  - Allow: /searchhistory/
  - Disallow: /search
- Per-page exclusion (de facto standard).
  - <meta name="ROBOTS" content="NOINDEX,NOFOLLOW">
- Per-link exclusion (de facto standard).
  - Toto
- Avoid Denial Of Service (DOS), wait 100ms/1s between two repeated requests to the same Web server

(9) What is XPath? Describe XPath axes in brief. OR What is XPath Axes? Describe following Axes in brief: 1) Axes 2) Sibling 3) descendant 4) parent 5) attribute 6) self 7) preceding

XPath

- An expression language to be used in another host language (e.g., XSLT, XQuery).
- Allows the description of paths in an XML tree, and the retrieval of nodes that match these paths.
- Can also be used for performing some (limited) operations on XML data.
- Example

  2∗3 is an XPath **literal expression**.

  //∗[@msg="Hello world"]

  is an XPath **path expression**, retrieving all elements with a msg attribute set to “Hello world”.

XPath Data Model

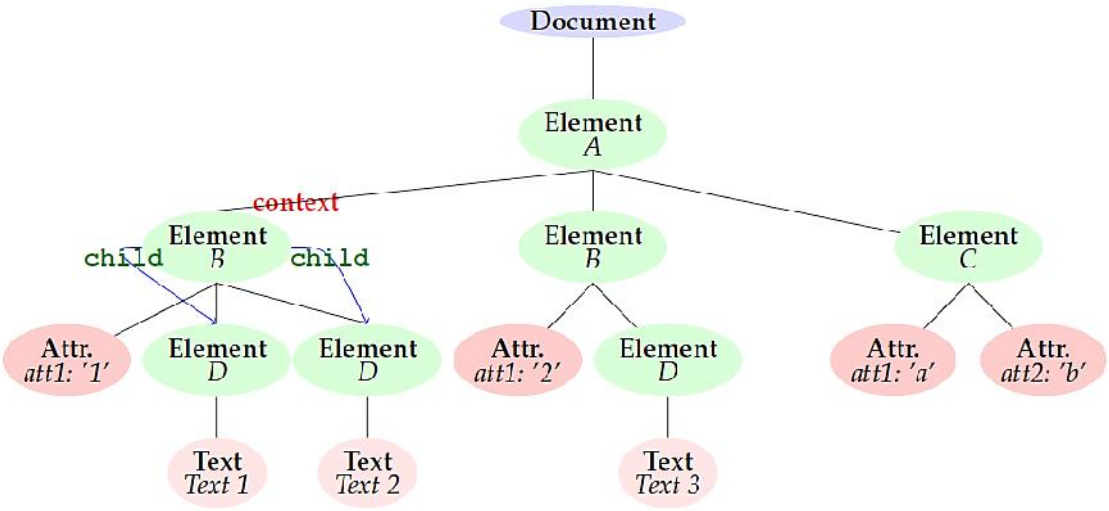
- XPath expressions operate over XML trees, which consist of the following node types:
- Document: the root node of the XML document;
- Element: element nodes;
- Attribute: attribute nodes, represented as children of an Element node;
- Text: text nodes, i.e., leaves of the XML tree.

Axes

- An axis = a set of nodes determined from the context node, and an ordering of the sequence.
  - child: (default axis).
  - parent: Parent node.
  - attribute: Attribute nodes.
  - descendant: Descendants, excluding the node itself.
  - descendant-or-self: Descendants, including the node itself.
  - ancestor: Ancestors, excluding the node itself.
  - ancestor-or-self: Ancestors, including the node itself.
  - following: Following nodes in document order.
  - following-sibling: Following siblings in document order.
  - preceding: Preceding nodes in document order.
  - preceding-sibling: Preceding siblings in document order.
  - self: The context node itself.

Child axis

- The **child** axis denotes the **Element** or **Text** children of the context node.
- This is the default axis, used when the axis part of a step if not specified. So, **child::D** is in fact equivalent to **D**.



The Child axis

Parent axis

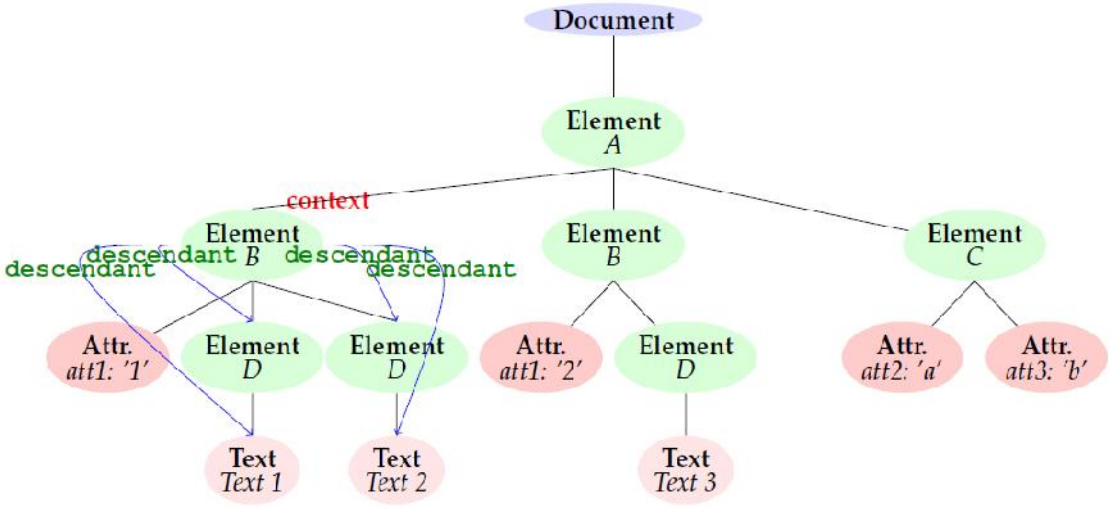
- The **parent** axis denotes the parent of the context node.
- The result is always an **Element** or a **Document** node, or an empty node-set (if the parent does not match the node test or does not satisfy a predicate).
- One can use as node test an element name.
- The node test ***** matches all names.
- The node test **node()** matches all node kinds.
- These are the standard tests on element nodes. For instance:
    - if the context node is one of the B elements,
        - the result of **parent::A** is the root element of our sample document; one obtains the same result with **parent::*** or **parent::node()**;
    - if the context node is the root element node,
        - then **parent::*** returns an empty set, but the path **parent::node()** returns the root node of the document.

Attribute axis

- The **attribute** axis retrieves the attributes of the context node.
- The node test may be either the attribute name, or **@*** which matches all attribute names.
- So, assuming the context node is the C element of our example,
    - **@att1** returns the attribute named att1;
    - **@*** returns the two attributes of the context node.

Descendant axis

- The **descendant** axis denotes all nodes in the subtree of the context node, except the **Attribute** nodes.
- The node test **text()** matches any **Text** node.
- Assume for instance that the context node is the first B element in the document order (as shown in Figure). Then :



*Result of **descendant::node()***

- **descendant::node()** retrieves all nodes descendants of the context node, except attributes;
- **descendant::*** retrieves all **Element** nodes, whatever their name, which are descendant of the context node;
- **descendant::text()** retrieves all **Text** nodes, whatever their name, which are descendant of the context node.
- Observe that the context node is not a descendant of itself. If one wants it in the resulting context list, one should use instead **descendant-or-self**.

Ancestor axis

- The **ancestor** axis denotes all ancestor nodes of the context node.
- The result of **ancestor::node()**, when the context node is the first B element, consists of both the element root and the root node.
- Again, if one wants the context node to belong to the result, one should use **ancestor-or-self** instead.

Following and preceding axes

- The **following** and **preceding** axes denote respectively all nodes that follow the context node in the document order, or that precede the context node, with the exception of descendant or ancestor nodes.
- **Attribute** nodes are not selected.

Sibling axes

- The siblings of a node N are the nodes that have the same parent as N.
- XPath proposes two axes: **following-sibling** and **preceding-sibling** that denote respectively the siblings that follow and precede the context node in document order.
- The node test that can be associated with these axes are those already described for **descendant** or **following**: a node name (for **Element**), ***** for all names, **text()** or **node()**. Note that, as usual, the sibling axes do not apply to attributes.

(10) What is XML Typing? Why it is essential in web data management.

- The structure of an XML document is included in the document in its label structure.
- As already mentioned, one speaks of self-describing data.
- This is an essential difference with standard databases:
    - In a database, one defines the type of data (e.g., a relational schema) before creating instances of this type (e.g., a relational database).
- In semi structured data (and XML), data may exist with or without a type.
- The “may” (in may exist) is essential.
- Types are not forbidden; they are just not compulsory and we will spend quite some effort on XML typing.
- But in many cases, XML data often presents the following characteristics:
    - the data are irregular: there may be variations of structure to represent the same information (e.g., a date or an address) or unit (prices in dollars or euros);
    - this is typically the case when the data come from many sources;
    - parts of the data may be missing, for example, because some sources are not answering, or some unexpected extra data (e.g., annotations) may be found;
    - the structure of the data is not known a priori or some work such as parsing has to be
- performed to discover it (e.g., because the data come from a newly discovered source);
    - part of the data may be simply untyped, (e.g., plain text).
- Another differences with database typing is that the type of some data may be quite complex.
- In some extreme cases, the size of the type specification may be comparable to, or even greater than, the size of the data itself.
- It may also evolve very rapidly.
- These are many reasons why the relational or object database models that propose too rigid typing were not chosen as standards for data exchange on the Web, but a semi structured data model was chosen instead.

Motivating Typing

- Perhaps the main difference with typing in relational systems is that typing is not compulsory for XML.
- It is perfectly fine to have an XML document with no prescribed type.
- However, when developing and using software, types are essential, for interoperability, consistency, and efficiency.
- ***Interoperability***
    - Schemas serve to document the interface of software components, and provide therefore a key ingredient for the interoperability between programs:
        - a program that consumes an XML document of a given type can assume that the program that has generated it has produced a document of that type.
- ***Consistency***
    - Similarly to dependencies for the relational model (primary keys, foreign key constraints, etc.), typing an XML document is also useful to protect data against improper updates.
- ***Storage Efficiency***
    - Suppose that some XML document is very regular, say, it contains a list of companies, with, for each, an ID, a name, an address and the name of its CEO.
    - This same information may be stored very compactly, for instance, without repeating the names of elements such as address for each company.

- Thus, a priori knowledge on the type of the data may help improve its storage.
- Query Efficiency. Consider the following XQuery query:

```
for $b in doc("bib.xml")/bib//*
where $b/*/zip = '12345'
return $b/title
```

- Knowing that the document consists of a list of books and knowing the exact type of book elements, one may be able to rewrite the query:

```
for $b in doc("bib.xml")/bib/book
where $b/address/zip = '12345'
return $b/title
```

- that is typically much cheaper to evaluate.
- Note that in the absence of a schema, a similar processing is possible by first computing from the document itself a data guide, i.e., a structural summary of all paths from the root in the document.
- There are also other more involved schema inference techniques that allow attaching such an a posteriori schema to a schema-less document.

(11) Discuss XPath and XQuery with example.

OR

Differentiate between XPath and XQuery with examples.

XPath

- An expression language to be used in another host language (e.g., XSLT, XQuery).
- Allows the description of paths in an XML tree, and the retrieval of nodes that match these paths.
- Can also be used for performing some (limited) operations on XML data.
- Example

  2∗3 is an XPath **literal expression**.

  //∗[@msg="Hello world"]

  is an XPath **path expression**, retrieving all elements with a msg attribute set to “Hello world”.

XPath Data Model

- XPath expressions operate over XML trees, which consist of the following node types:
- Document: the root node of the XML document;
- Element: element nodes;
- Attribute: attribute nodes, represented as children of an Element node;
- Text: text nodes, i.e., leaves of the XML tree.

Axes

- An axis = a set of nodes determined from the context node, and an ordering of the sequence.
  - child: (default axis).
  - parent: Parent node.

- attribute: Attribute nodes.
- descendant: Descendants, excluding the node itself.
- descendant-or-self: Descendants, including the node itself.
- ancestor: Ancestors, excluding the node itself.
- ancestor-or-self: Ancestors, including the node itself.
- following: Following nodes in document order.
- following-sibling: Following siblings in document order.
- preceding: Preceding nodes in document order.
- preceding-sibling: Preceding siblings in document order.
- self: The context node itself.

XQuery

- **XQuery** is an XML query language that makes use of XPath to query XML structures.
- However it also allows for functions to be defined and called, as well as complex querying of data structures using **FLWOR** expressions.
- FLWOR allows for join functionality between data sets defined in XML.
- FLWOR article from wikipedia Sample XQuery (using some XPath) is:

```
declare function local:toggle-boolean($b as xs:string)
as xs:string
{
    if ($b = "Yes") then "true"
    else if ($b = "No") then "false"
    else if ($b = "true") then "Yes"
    else if ($b = "false") then "No"
    else "[ERROR] @ local:toggle-boolean"
};

<ResultXML>
    <ChangeTrue>{ local:toggle-
boolean(doc("file.xml")/article[@id="1"]/text()) }</ChangeTrue>
    <ChangeNo>{ local:toggle-
boolean(doc("file.xml")/article[@id="2"]/text()) }</ChangeNo>
</ResultXML>
```

XQuery vs XPath

- XQuery is a functional programming language that is used to query a group of XML data.
- It is able to manipulate and extract data from either XML documents or relational databases and MS Office documents that support an XML data source.
- It is a language that helps in creating syntax for new XML documents.
- XQuery is represented in the form of a tree model with seven nodes, namely processing instructions, elements, document nodes, attributes, namespaces, text nodes, and comments.

- All values are referred to as sequences.
- Even a single value is considered as a sequence of length one.
- The sequence can consist of either nodes or atomic values like integers, strings, or Booleans.
- It has the following features that are used for the transformation of XML data:
    - Side effect free.
    - Logical/physical data independence.
    - Strongly typed.
    - High level.
    - Declarative
- XPath is the XML Path Language that is used for selecting nodes from an XML document using queries.
- It can also compute values like strings, numbers, or Boolean type from another XML document.
- The expression in case of XML is known as XPath.
- It is represented as a tree structure with the ability of XPath to navigate it by selecting different nodes.
- It was created to define a common syntax and behavior model for XPointer and XSLT.
- XPath has the following features:
    - XPath defines the syntax for an XML document.
    - It has the capability to navigate path expressions in XML documents.
    - It has its own library defining standard functions.
    - It is a major component of XSLT.
- Other differences between XPath and XQuery:
    1. XPath is viewed as a regular expression whereas XQuery is like a C-programming language w.r.t. XML documents.
    2. XPath is a filter for an XML dataset and is the transformational component of XSLT. XQuery is used to select several nodes from an XML document for the purpose of processing using different queries.
    3. XQuery uses XPath syntax for addressing different parts of an XML document. The joins are performed using the FLWOR expression. This expression has five clauses, namely, WHERE, ORDER BY, FOR, LET, and RETURN.

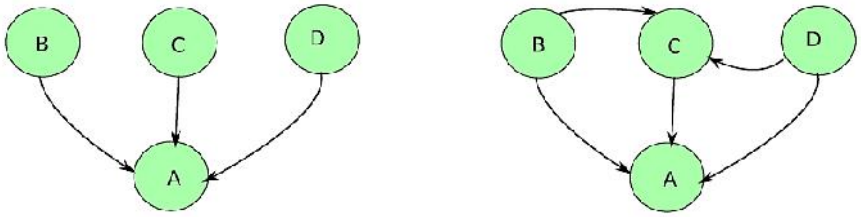
(12) Explain Web graph mining and hot topics in web search.

Web Graph Mining

- As all hyperlinked environments, the World Wide Web can be seen as a directed graph in the following way: Web pages are vertices of the graph, whereas hyperlinks between pages are edges.
- This viewpoint has led to major advances in Web search, notably with the PageRank and HITS algorithms presented in this section.
- Extraction of knowledge from graphs, or graph mining, has been used on other graph structures than the Web, for instance on the graph of publications, where edges are the citation links between publications; cocitation analysis relies on the observation that two papers that are cited by about the same set of papers are similar.
- Other graphs susceptible to this kind of analysis include graphs of dictionaries, or encyclopedias, or graphs of social networks.

PageRank

- PageRank, which was introduced with much success by the founders of the Google search engine, is a formalization of this idea.
- The PageRank of a page i can be defined informally as the probability pr(i) that the random surfer has arrived on page i at some distant given point in the future.
- Consider for instance the basic graph on the left of Figure.



- A random surfer will reach node A at step i if it reaches node B, C or D at step i - 1. Therefore:

$$pr(A) = pr(B) + pr(C) + pr(D)$$

Online Computation

- The computation of PageRank by iterative multiplication of a vector by the dampened transition matrix requires the storage of, and efficient access to, the entire Web matrix.
- This can be done on a cluster of PCs using an appropriate distributed storage technique (see the chapters devoted to distributed indexing and distributed computing).
- An alternative is to compute PageRank while crawling the Web, on the fly, by making use of the random walk interpretation of PageRank.
- This is what the following algorithm, known as OPIC for Online PageRank Importance Computation, does:
    1. Store a global cashflow G, initially 0.
    2. Store for each URL u its cash C(u) and history H(u).
    3. Initially, each page has some initial cash c_0.
    4. We crawl the Web, with some crawling strategy that accesses repeatedly a given URL.
    5. When accessing URL u:
        i. we set H(u) := H(u) + C(u);
        ii. we set C(u') := C(u)/n_u for all URL u' pointed to by u, with n_u the number of outgoing links in u.
        iii. we set G := G + C(u) and C(u) := 0.
    6. At any given time, the PageRank of u can be approximated as H(u)/G .

HITS

- The HITS algorithm (Hypertext Induced Topic Selection) is another approach proposed by Kleinberg.
- The main idea is to distinguish two kinds of Web pages: hubs and authorities.
- Hubs are pages that point to good authorities, whereas authorities are pages that are pointed to by good hubs.
- As with PageRank, we use again a mutually recursive definition that will lead to an iterative fix point computation.

Spamdexing

- The term spamdexing describes all fraudulent techniques that are used by unscrupulous Webmasters to artificially raise the visibility of theirWeb site to users of search engines.
- As with virus and antivirus, or spam and spam fighting, spamdexing and the fight against it is an unceasing series of techniques implemented by spamdexers, closely followed by countertechniques deployed by search engines.

Discovering Communities on the Web

- The graph structure of the Web can be used beyond the computation of PageRank-like importance scores.
- Another important graph mining technique that can be used on the Web is graph clustering:
    - Using the structure of the graph to delimitate homogeneous sets of Web pages (e.g., communities of Web sites about the same topic).
    - The assumption is that closely connected set of pages in a Web page will share some common semantic characteristic.

Hot Topics in Web Search

- ***Web 2.0***
- Web 2.0 is a buzzword that has appeared recently to refer to recent changes in theWeb, notably:
    - Web applications with rich dynamic interfaces, especially with the help of AJAX technologies (AJAX stands for Asynchronous JavaScript And XML and is a way for a browser to exchange data with a Web server without requiring a reload of a Web page); it is exemplified by GMail8 or Google Suggest9;
    - User-editable content, collaborative work and social networks (e.g., in blogs, wikis such asWikipedia10, and social networkWeb sites like MySpace11 and Facebook12);
    - Aggregation of content from multiple sources (e.g., from RSS feeds) and personalization, that is proposed for instance by Netvibes13 or YAHOO! PIPES (see Chapter 12).
- ***Deep Web***
- The deep Web (also known as hidden Web or invisible Web) is the part of Web content that lies in online databases, typically queried through HTML forms, and not usually accessible by following hyperlinks.
- As classical crawlers only follow these hyperlinks, they do not index the content that is behind forms.
- There are hundreds of thousands of such deep Web services, some of which with very high-quality information:
    - all Yellow pages directories, information from the U.S. Census bureau, weather or geo-location services, and so on.
- ***Information Extraction***
- Classical search engines do not try to extract information from the content of Web pages, they only store and index them as they are.
- This means that the only possible kind of queries that can be asked is keyword queries, and results provided are complete Web pages.
- The purpose of Web information extraction is to provide means to extract structured data and information from Web pages, so as to be able to answer more complex queries.

(13) Explain XLink and XPointer with example.

XLink

- What XLink can do
  - Xlink is a proposal for more powerful links between documents.
  - Xlink achieves everything possible with HTML's URL-based hyperlinks and anchors.
  - Beyond this, it supports multidirectional links where the links run in more than one direction.
- Any element can become a link, not just the element.
  - Links do not even have to be stored in the same file as the documents they link.
  - Furthermore, Xpointers allow links to point to arbitrary positions in an XML document.
- These features make XLinks more suitable.
- **Linking Elements**
  - xlink:type attribute values
    - simple
    - extended
    - locator
    - arc
    - resource
    - Title
  - Xlink namespace
    - The xlink prefix must be bound to the http://www.w3.org/1999/xlink namespace URI.
    - The xlink prefix is customary and should be used unless you have got a really good reason to change it.
- **Examples:**

```
<composer xmlns:xlink=" http://www.w3.org/1999/xlink"
          xlink:type="simple"
          xlink:href=" http://www.users.net/~beand/">
     Beth Anderson
</composer>
```

- **Simple Link**
  - Definition: A simple Xlink is encoded in an XML document as an element of arbitrary type that has an xlink:type attribute with the value simple and an xlink:href attribute. The attribute's value is the link target's URI.
  - Example

```
<novel xmlns:xlink=" http://www.w3.org/1999/xlink"
     xlink:type="simple" xlink:href="urn:isbn:0688069444">
     <title> The Wonderful Wizard of Oz </title>
     <author> L. Frank Baum </author>
     <year> 1900 </year>
</novel>
```

- **Link Behavior**
  - How the connection is described by XLink presented to the end user or what dose it make software reading the document do?
  - They don't have just one answer.
  - Page authors can offer suggestions to browsers about how links should be handled by using the xlink:show and xlink:actuate attributes.
- **Xlink:show**
  - The xlink:show tells the browser or other application what to do when the link is activated.
  - It has five possible values:
  - New: opens a new window and shows the content at the link's URI (the ending resources) in that window.
- **Xlink**:actuate
  - The xlink:actuate attribute tells the browser when to show the content.
  - The optional xlink:actuate has four possible value
  - OnLoad: The link should be followed as soon as the application sees it.
  - OnRequest: The link should be followed when the user asks to follow it.
  - Other: Other markup in the document, not specified by xlink, determines when to follow the link.
  - None: No details are available.
- **Example**

```
<novel xlink:type="simple"
xlink:href=" ftp://archive.org/pub/etext93/wizoz10.txt"
xlink:actuate="onRequest" xlink:show="replace">
     <title> The Wonderful Wizard of Oz </title>
     <author> L. Frank Baum </author>
</novel>
```

  - This code says to wait for an explicit user request to follow the link and then replace the existing document with the document found at ftp://archive.org/pub/wizoz10.txt.
- **Link Semantics**
- Xlink elements can have xlink:title and xlink:role attributes to specify the meaning the connection between the resources
  - Xlink:title
    - this attribute contains a small amount of plain text describing the remote resources.
  - Xlink:role
    - this attribute contains a URI pointing to a longer description of the remote resource.
- **Example**

```
<novel xlink:type="simple"
xlink:href=" ftp://archive.org/pub/etext93/wizoz10.txt "
xlink:title="The complete text of the novel from project gutenberg"
xlink:role=" http://promo.net/pg/">
     <title> The Wonderful Wizard of Oz </title>
     <author> L. Frank Baum </author>
```

```
</novel>
```

- **Extended Links**
  - An extended link describes a collection of resources and paths between those resources.
  - Each path connects exactly tow resources.
  - Any individual resource may be connected to another resource, two other resources, no resources, all other resources, or any subset of other resources in the collection. It may even connect back to itself.
  - An extended link is a directed, labeled graph in which the paths are arcs, the documents are vertices, and the labels are URIs.

XPointer

- Xpointers address the individual parts of an XML document.
- Xpointer syntax builds on the Xpath syntax used by XSLT.

- **Xpointers in Links**
  - Xpointers in URL
- **Example**:
  - http://www.ibiblio.org/xml/people.xml#xpointer(//name[position()=1])
  - This URL points to the first name element in the document at
  - http://www.ibiblio.org/xml/people.xml
- **Xpointers in Xlinks**
  - Xpointers are more frequently used in Xlinks.
  - The following example points to the first book child of the bookcoll child of the testament root element in the document at the relative URL ot.xml:

```
<link xlink:type="simple"
      xlink:href="ot.xml#xpointer(/testament/bookcoll/book[position()=1])">
      Genesis
</link>
```

- **Bare Names**
  - A bare name Xpointer is similar to an HTML named anchor; it identifies the element it points at by its name.
  - However, an ID attribute of the element being pointed at, rather than a special an element with a name attribute, supplies this name.
- **Child Sequences**
  - Many xpointers descend exclusively along the child axis, selecting elements by their position relative to their siblings
  - `xpointer(/child::*[position()=1]/child::*[position()=2]/child::*[position()=3])`
  - selects the third child element of the second child element of the document's root element.
- **Points and Ranges**
  - Xpaths, bare names, and child sequences let you point only to entire nodes or sets of nodes.
  - Sometimes you may want to point to something that isn't a node, such as the third word of the second paragraph.

- Xpointer adds points and ranges to the Xpath syntax to make this possible.

- **Points**
  - If the node contains child nodes, then points exist before and after each of its children.
  - If the node does not contain child nodes, then a point is present before and after each character in the node's string value.
- **Example**
  - Eight points are located directly inside the novel element.

```
<novel copyright="public domain">*                    Point0
Point1  *<title>The Wonderful Wizard of Oz</title>*  Point2
Point3  *<author>L. Frank Baum</author>*
Point4
Point5  *<year>1990</year>*
Point6
Point7*</novel>
```

- **Ranges**
  - A range is the span of parsed character data between two points.
  - Ranges are created with four functions that XPointer adds to XPath:
    - range ( )
    - range-inside ( )
    - range-to ( )
    - string-range ( )

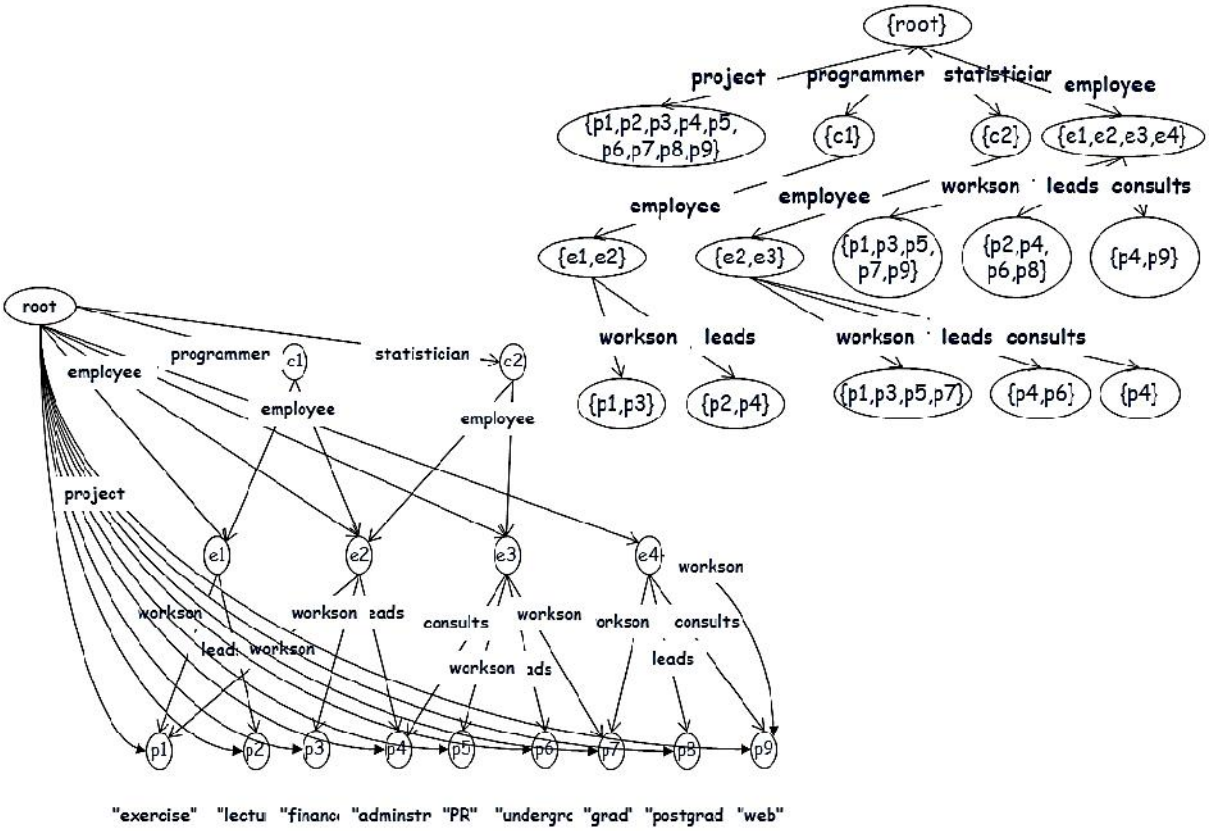
(14) Compare DTD and XSD.

- DTD, or Document Type Definition, and XML Schema, which is also known as XSD, are two ways of describing the structure and content of an XML document.
- DTD is the older of the two, and as such, it has limitations that XML Schema has tried to improve.
- The first difference between DTD and XML Schema, is namespace awareness; XML Schema is, while DTD is not.
- Namespace awareness removes the ambiguity that can result in having certain elements and attributes from multiple XML vocabularies, by giving them namespaces that put the element or attribute into context.
- Part of the reason why XML Schema is namespace aware while DTD is not, is the fact that XML Schema is written in XML, and DTD is not.
- Therefore, XML Schemas can be programmatically processed just like any XML document.
- XML Schema also eliminates the need to learn another language, as it is written in XML, unlike DTD.
- Another key advantage of XML Schema, is its ability to implement strong typing.
- An XML Schema can define the data type of certain elements, and even constrain it to within specific lengths or values.
- This ability ensures that the data stored in the XML document is accurate.

- DTD lacks strong typing capabilities, and has no way of validating the content to data types.
- XML Schema has a wealth of derived and built-in data types to validate content.
- This provides the advantage stated above.
- It also has uniform data types, but as all processors and validators need to support these data types, it often causes older XML parsers to fail.
- A characteristic of DTD that people often consider both as an advantage and disadvantage, is the ability to define DTDs inline, which XML Schema lacks.
- This is good when working with small files, as it allows you to contain both the content and the schema within the same document, but when it comes to larger documents, this can be a disadvantage, as you pull content every time you retrieve the schema.
- This can lead to serious overhead that can degrade performance.

(15) Explain following terms: Graph Bisimulation.

- A typing of a graph may be seen in some sense as a classification of its nodes, with nodes in a class sharing the same properties.
- Such a property is that they “point to” (or are “pointed by”) nodes in particular classes.
- For instance, consider below Figure.



- A set of nodes forms the class employee. From an employee node, one can follow workson, leads and possibly consults edges to some nodes that form the class project.
- This is the basis for typing schemes for graph data based on “simulation” and “bisimulation”.
- A simulation S of (E, r) with (E’, r’) is a relation between the nodes of E and E’ such that:
    - S(r, r’) and
    - if S(s, s’) and E(s, t, l) for some s, s’, t, l, then there exists t’ with S(t, t’) and E’(s’, t’, l’).
- The intuition is that we can simulate moves in E by moves in E’.
- Given (E, r), (E’, r’), S is a bisimulation if S is a simulation of E with E’ and S^{-1} is a simulation of E’ with E.
- To further see how this relates to typing, take a very complex graph E.
- We can describe it with a “smaller” graph E’ that is a bisimulation of E.
- There may be several bisimulations for E including more and more details.
- At one extreme, we have the graph consisting of a single node with a self-loop.
- At the other extreme, we have the graph E itself.
- This smaller graph E’ can be considered as a type for the original graph E. In general, we say that some graph E has type E’ if there is a bisimulalation of E and E’.