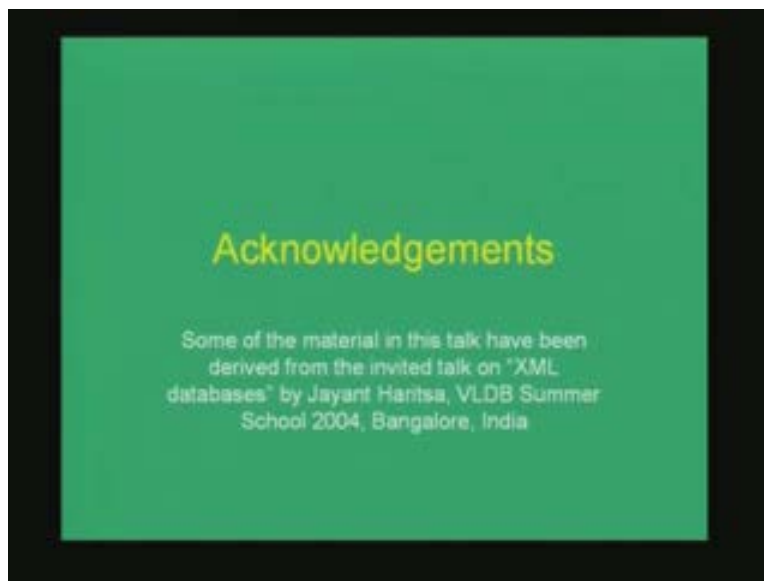


Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 39
XML-Advanced Concepts

Hello and welcome. In the previous session in the DBMS course, we have been looking into managing XML data. As we had mentioned earlier, XML is kind of a markup language which is in some way the de facto standard for information interchange over the internet. And the power of XML comes from its simplicity in the sense that it's a markup language which can be read and understood by both humans as well as machines. And it's independent of any operating platform or independent of any standards like how schema is describes and so on. And essentially it's a self-describing, it describes a self-describing data set in a sense.

Now in this session let us continue with XML by looking into some advanced aspects of how XML data are stored or how they are queried and so on. And XML as I had mentioned in the previous session has elicited wide area of interest and there are several different not just in computer science in the sense that XML is elicited interest across several different disciplines because information interchange or managing different facets of information, different points of information and integrating them, interchanging information is a common problem in several different domains. And there are different kinds of standards in specific domains like banking or finance or bio informatics or transportation or whatever and several different domains which have defined XML DTDs pertaining to their specific area of concern. So let us look into XML little more deeply in this session and see what we can do with XML.

(Refer Slide Time: 03:36)



First of all let me start by acknowledging that some of the material in the slides have been derived from an invited talk by Jayant Haritsa in the VLDB summer school held in Bangalore in June 2004. So let us have a brief recap of what we studied about XML and what are its main features like I had mentioned before XML is a platform independent and standardized extensible markup language. Now each of these different terms means a very specific thing. Platform independent means that you just store XML in plain vanilla text, character data essentially. And every platform, no matter what you use, what platform you use would support textual data regardless of whatever underlying encoding that you are using.

(Refer Slide Time: 05:04)



And it's a standardized markup language in the sense that there are specific rules that specify how an XML data set should look like. For example it should be a rooted tree and a begin tag should end with an end tag and there is this notion of well formedness and validness and so on. And it's an extensible markup language. The markup language does not define, what are the kinds of tags that you need to have in your XML data. You can define your own tags and we took up some examples where a notice was a kind of tag notice and slash notice or from and to and so on and so forth.

It's a self-describing dataset, in the sense that the structure of the data is implicit in the data set itself or meta data is embedded within the dataset itself. So you don't need a separate descriptor for how the data are to be described. Of course you may have a separate set of what might called as restrictions or which specify whether that description is valid or not. And it's a very simple standard for data interchange made up of simple building blocks like elements and attributes and nesting of different elements and so on. And we also saw the difference between what is a well formed XML fragment and what is a valid XML fragment.

A well formed XMLs fragment essentially conforms to the XML structural requirements, I mean structural requirements of an XML document in the sense that it's a rooted tree and the nesting is proper, every begin tag is closed by an end tag and tag names do not have spaces in them and they do not begin with a special character and so on and so forth. But a valid XML data has to be well formed of course that is unless data set is well formed it can't be valid. However valid is most stringent in the sense that it's not sufficient for an XML data set to be well formed but also it should conform to a given document type definition, a DTD or a XML schema that is a schema that is specified as part of this document.

(Refer Slide Time: 07:12)



Why XML or what is the significance of XML? It's a very convenient way, probably the most convenient way of exchanging data over the web. And like I had mentioned before it's easier for both machine understanding and human understanding. So in the worst case you can actually open an XML document in a text editor like notepad or Emax or whatever and understand what the XML data is all about and make changes if required. And it has a simple tree structure like a simple hierarchical structure which is easier to understand, easier to enforce because the tree has very specific constraints and well known constraints as to what makes up a tree structure.

And enforcement is also easier that is it's easy to build parsers which using stacks or whatever. There are several kinds of tree algorithms that can validate a given tree and it's also easier to navigate, nice directory structure is a tree structure and hence XML data sets are usually rendered in the form of a directory structure on browsers. And like I had mentioned before, XML is eliciting interest not just in the computing community but perhaps more so in several other communities. And information interchange is not just a problem with database researchers or computer science researchers but it's a problem in almost any domain of concern.

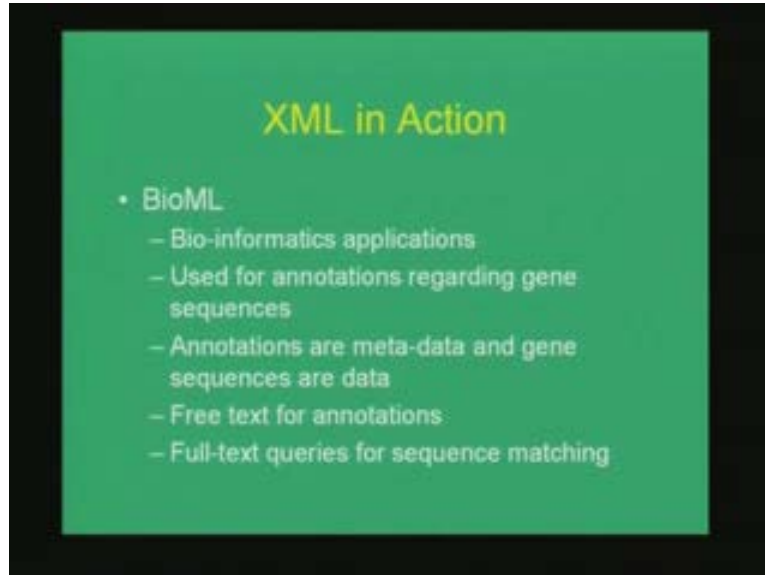
(Refer Slide Time: 08:42)



What is the information interchange problem? **Simply stated that** simply stated it means it is a problem of how to integrate the varying or desperate sources of information that exists in any given system. FpML for example as shown in the slide is an XML standard for the finance industry. And FpML essentially standardizes the different kinds of contracts or transaction data sets that are routinely specified, generated and specified in the finance domain pertaining to banking or stock trading or loaning and several different allied activities where contracts would have more or less some common features in the sense that or contract would have to specify what are the parties to a given contract, what is the validity period of the contract, when does the contract expire, what are the components of a contract and what's the amount involved in this contractual interchange and so on.

And there are several standards that are used independently by different sources and FpML in effect tries to standardize this contractual specifications so that it becomes easier to integrate different kinds of contracts that are generated in different places.

(Refer Slide Time: 10:27)

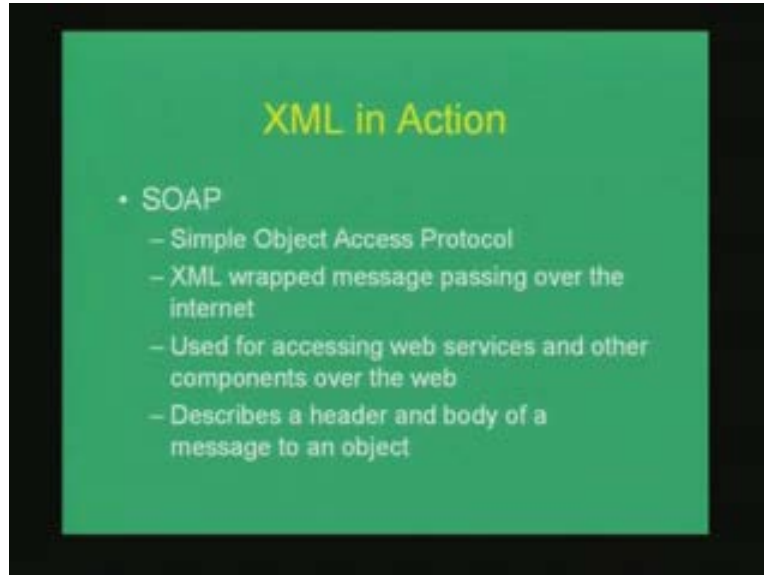


Similarly BioML as shown in the slide is an XML standard primarily for use in bioinformatics applications and that too in annotating gene sequence data. And gene sequences as you know are very long sequences comprising of one of four different characters 80 g and c sometimes u and sub sequences of this long sequence specify, this long sequence is what is called as the genome sequence and sub sequences of this genome sequences specify genes or sometimes what are called as some a set of codon which code for a particular kind of behavior in the organism or however when the gene folds. So such kinds of codon, such kinds of code on strings or genes are annotated.

And this annotation is done in several different ways by several different researchers and because there several researchers trying to sequence genomes of different kinds and including of course the human genome **and** or annotating different parts of a genome string.

Now BioML is a standard which specifies how this annotations had to be performed and of course it allows for free text for annotations. That is it does not really place any constraints on what should go into the annotations themselves but it specifies how and where annotations should have. And BioML in addition to its, for example FpML or BioML actually specify the schema, they basically specify the XML schema of this. So the XML schema also supports full text queries for different kinds of sequence matching problems.

(Refer Slide Time: 12:37)

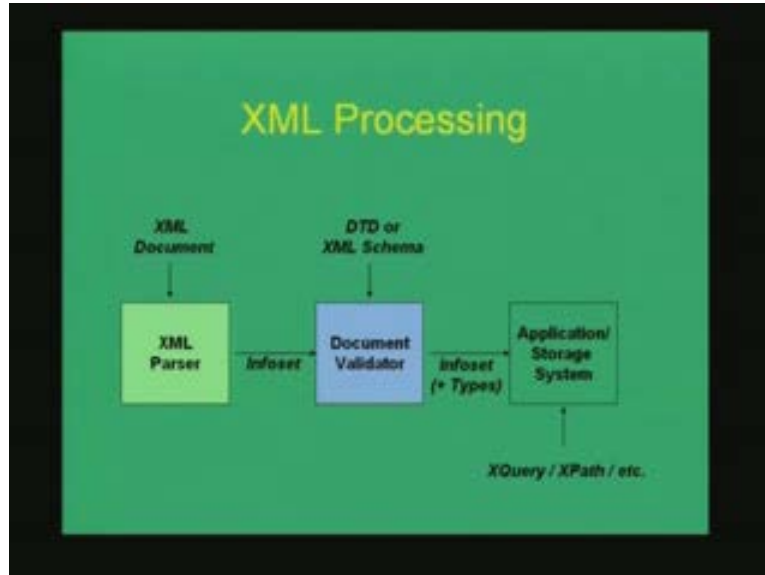


And then a very commonly used standard that uses XML is the SOAP protocol. You might have heard of the SOAP protocol which stands for simple object access protocol which forms in some sense the building blocks of web services. SOAP is a mechanism by which a software can access a remote object or invoke, send methods to remote object via the internet.

So what is the SOAP protocol do? It is simply a message passing or a message sent to a remote object that is wrapped in an XML document that is the message is sent as an XML document and at the other end where the services provided, the XML is parsed and the actual message is taken from the XML document and then sent to the object and question. So in this for example you might have some kind of an object that performs a given kind of calculation let us say currency conversion. So you might want to perform some kind of currency conversion and that is a service that you provide over the web.

Now if this currency conversion object is SOAP compliant then the clients speaking with your object or speaking with your component sends messages in an XML format. And the object contains an XML parser embedded into it which parses the XML document and divides the XML input into two different parts a header and a body. So the header contains meta data about the message saying what kind of message, where is it coming from and any other restrictions and the body contains the actual message itself.

(Refer Slide Time: 14:24)

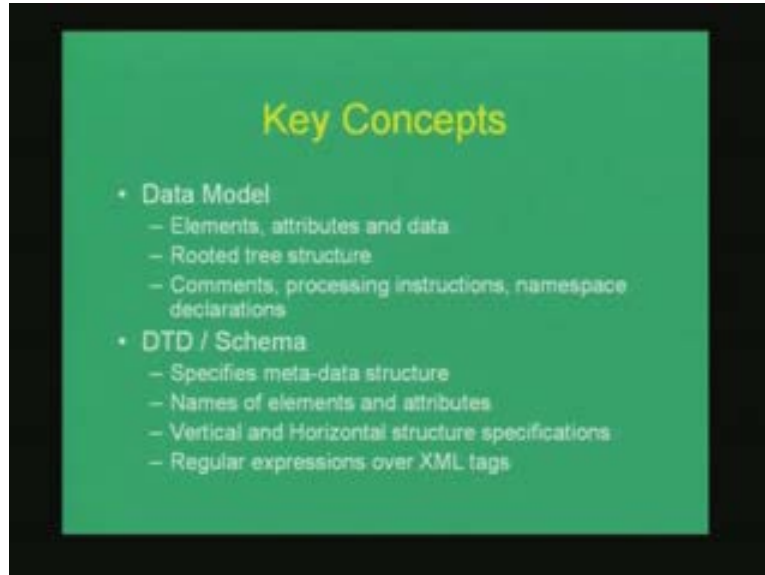


So this slide shows a schematic diagram of how an XML, how XML processing would probably look like. Given XML document, given an XML document we have what is called as an XML parser. When an XML parser can check for the well formedness of an XML document so that it can parse an XML tree and create some kind of a data structure here, also called as an info set which in turn goes into a document validator. Document validator essentially validates a given XML document against an input DTD or an XML schema.

So once the document is validated then the XML is ready for the application of the storage system where whatever it is being used for. And of course the application itself accepts different kinds of queries using XQuery or XPath and so on and can answer queries on the info set that it received from the document validator.

So what are the key concepts in XML? You have the data model which comprises of elements attributes and data and rooted tree structure plus of course comments and processing instructions and namespace declarations and so on and so forth.

(Refer Slide Time: 15:57)



In addition you have the DTD or the XML schema which specifies the meta data structure. And there are several different features that are available in a DTD in the sense that it can define regular expressions over XML tags. So you can say that, notice can have zero or more headers or one or more two elements and so on and so forth.

So here is an example XML fragment were this is a rooted tree again. It's an XML element but every element, every well-formed element is a rooted tree in itself.

(Refer Slide Time: 16:40)



So this is a rooted tree which begins at imdb and slash imdb and of course you might know that imdb stands for the internet movie database which is a huge source of XML information that is it stores a lot of movie related information in XML. And then imdb itself contains an element called show and show year equal to 1993 and comments are given like this. That is angular braces with an exclamation mark followed by two hyphens and then again two hyphens with the angular braces here.

So show is the element from here to here, so the show element contains one or more reviews. So review one starts from here to here and review two starts from here to here and then there is a set of box office numbers. And each review itself in turn says where the review is from, so this one says the review is from sun times. And the review itself is a mixed XML plus free text that is there is free text going around here within which there are some XML tags. That is Roger Ebert is surrounded by the reviewer tag and two thumbs up is given, is surrounded by the rating tag and so on.

So that's how a simple XML fragment would look like and a typical DTD for this would look like this where you start your DTD by a DOCTYPE declaration and then specify what is the root element of your XML fragment. Then within this specify each element and the set of attributes.

(Refer Slide Time: 18:22)



So imdb is the root element can contain show star, show star essentially means that zero or more occurrences of the show element. Similarly the show element contains title, review star that is it has to contain exactly one title followed by zero or more reviews, review elements. And show can also contain an attribute called year which is character data and title is character data and so on. So I have not completed the DTD as yet but this is what the typical DTD would look like for this XML fragment. In contrast we have what is called as XML schema which is an emerging standard for which is fast replacing DTDs (Refer Slide Time: 20:00).

XML schema is far more expressive than a DTD in the sense and it supports, it's a strongly typed language in the sense that it supports a type of a particular kind. And most importantly an XML schema document is an XML document itself. That is you can use the same XML parser to parse an XML schema as you would use for parsing an XML document.

So how would an XML schema for this XML fragment look like? You see that it starts with an element declaration and the element declaration has an attribute called name equal to show. So I am not starting from the imdb declaration itself, I am just declaring the schema here for the show element. So this one says the show element is declared like this. That is the show element is a complex type having a sequence of a two or more elements where the first element in the sequence is the title and the type of this element is string.

(Refer Slide Time: 22:58)



And this is a single element, so the element is closed right here and then the second element is and then there is a sub sequence. So the first element of this sequence is an element of name title and the second element of the sequence is another sequence which can have repetitions by itself.

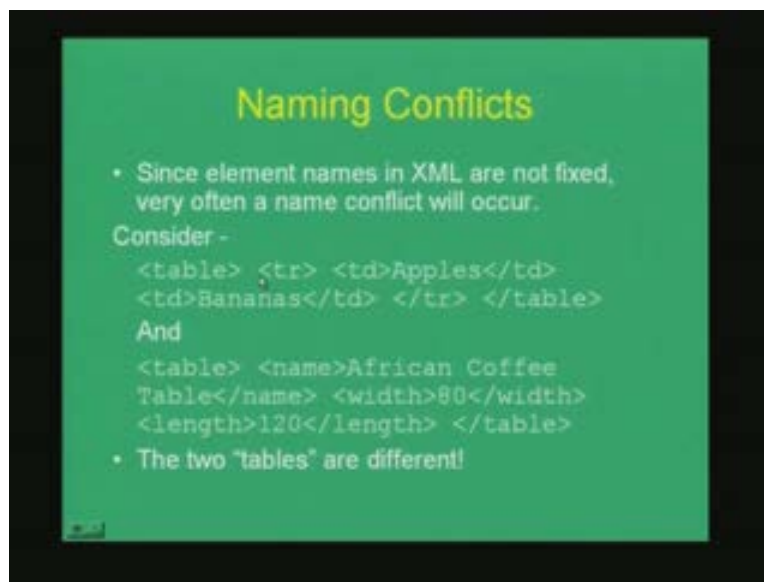
So min occurs zero and max occurs unbounded is another name for saying the star declaration in DTD that is something can occur zero or more times. So what is this sequence contain? This sequence contains an element called review and which is of mixed data that is a review can contain both free text and XML tags in itself.

And of course this sequence finishes now and then after this there is a choice element that is the third element of this sequence where I can have a choice between box office or seasons or so on. So one of these two, I can have either box office or seasons as the third element in the sequence. so note that show itself comprises of three different elements

title, review star in a sense and box office or seasons which was not specified completely in the DTD in the previous slide. And of course this sequence finishes and the next element, the next attribute of this element is year which is optional and slash element.

So that is typically what an XML schema would look like. As you can see here this is a simple XML document in itself and then you can parse this XML document using any XML parser. And then based on the outputs of this parser, you can enforce the constraints here on the given XML fragment.

(Refer Slide Time: 23:20)



Naming Conflicts

- Since element names in XML are not fixed, very often a name conflict will occur.

Consider -

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

And

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

- The two "tables" are different!

Let us now turn to a specific problem that occurs when parsing XML data namely the notion of naming conflicts. Now consider these two XML fragments here that are shown, one XML fragment is shown here and the second XML fragment is shown here. The first XML fragment, if you have worked with HTML you might have recognized that this is a HTML fragment and HTML fragment can also be treated as an XML fragment there is no harm in that.

So of course well-formed HTML fragments that is in the sense that every start tag is paired with an end tag and there is a nice hierarchical structure for this fragment. So this is a table fragment which begins at table and ends at slash table. And there is a table row which begins at tr and slash tr and table descriptor that is some table cell which begins at td and slash td.

And then somewhere down the line there is another fragment here which says table, slash table and name and width and length and so on and so forth, so African coffee table 80 and so on. Now these two could occur within a single XML document that is this could actually be some kind of PCDATA that is parsed character data for given XML element. And this could be another XML element by itself but these two tables are different. Now this is required because the data contains HTML and the HTML has to be rendered and

this is required because the semantics require that we need to declare something called table and these two tables are different. Now such a conflict is called a naming conflict that is when two or more tags mean different things but have the same name in them.

(Refer Slide Time: 25:55)



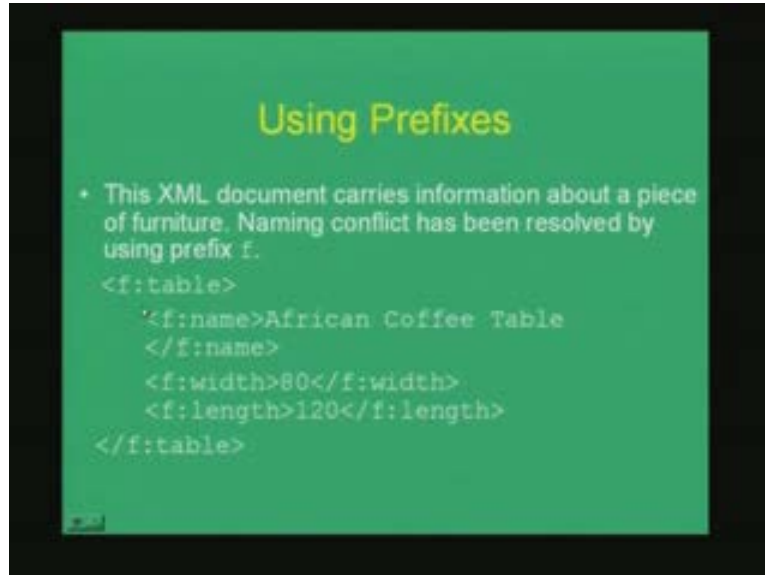
Using Prefixes

- Naming conflicts can be avoided using prefixes.
- This XML document carries information in a table. h is the prefix here.

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

So in order to resolve conflicts, how would we resolve conflicts. One simple way to resolve them is to prefix each of these, each occurrence of this meta data with a particular string. So for example this one shows (Refer Slide Time: 25:28) a prefix called h where it says, where the HTML table is prefixed by a character called h. So h table and slash h table and note the use of this colon character here. So the HTML table is discriminated or distinguished from the XML table by using a different prefix called h.

(Refer Slide Time: 25:60)



And similarly the XML table itself is prefixed by f. Now that is one simple way of doing that but again the question still remains as to what if the prefix also is the same, I mean if the name of the tag can be identical, the prefixes can also be identical when two or more XML documents are brought together and meant to be integrated into one schema.

So for this the notion of names spaces becomes valid or namespaces become important. A namespace essentially defines a unique space globally, worldwide across the web. So how does it define a unique name space across the web? Remember the notion of a URL or a URI uniform resource identifier.

A URL or the more general form which is the URI is a unique name for a resource across the web. Therefore whenever you see a address like this (Refer Slide Time: 27:09) <http://osl.iiitb.ac.in> whatever. it is unique that is this stands for, it has to map on to a single ip address or a single address across the world wherever in the world that it is referred to.

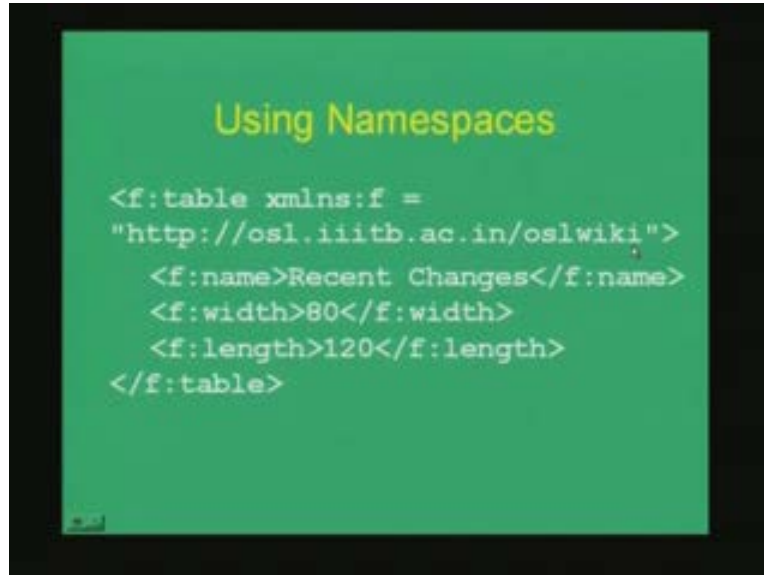
(Refer Slide Time: 28:15)



Now using the concept of URIs, we can ensure that naming conflicts in XML documents can be resolved. For example we can, when we say that when we put a prefix called h for example for this table meta data, for this table tag then we describe that h means a specific URI. Now this could be, this need not actually contain anything. The URI need not contain anything with respect to this table but let us say this is the URI in which we work in, let us say where we created the XML fragment.

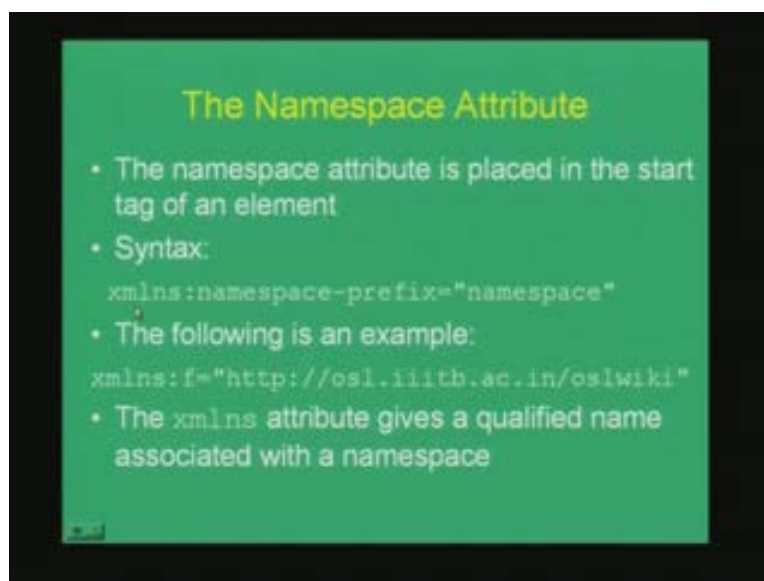
Now once we alias a given prefix to a URI it becomes unique. So in a sense whenever the XML parser looks at a prefix called h followed by colon, it replaces that with this URI. So if a different h comes from a different source and it maps to a different URI, it means that it is a different prefix. Therefore it basically means some other tag and not the same h as this prefix that is specified here.

(Refer Slide Time: 29:30)



So similarly we can use a different that is the same URI having a different prefixes so that different prefixes from the same URI can again be distinguished. Therefore what the XML parser actually does when it encounters name called f colon is that it replaces f colon by this whole thing slash f (Refer Slide Time: 29:20) internally. That is `http://osl.iiitb.ac.in/oslwiki/f`. So that several different prefixes that have been defined at this URI remains distinguishable.

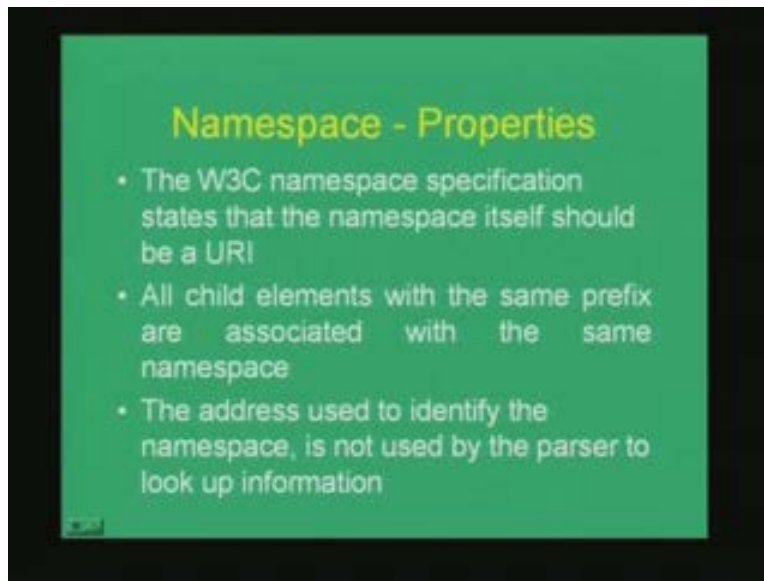
(Refer Slide Time: 30:00)



So the namespace attribute as we saw in the previous slides is placed in the start tag of an element.

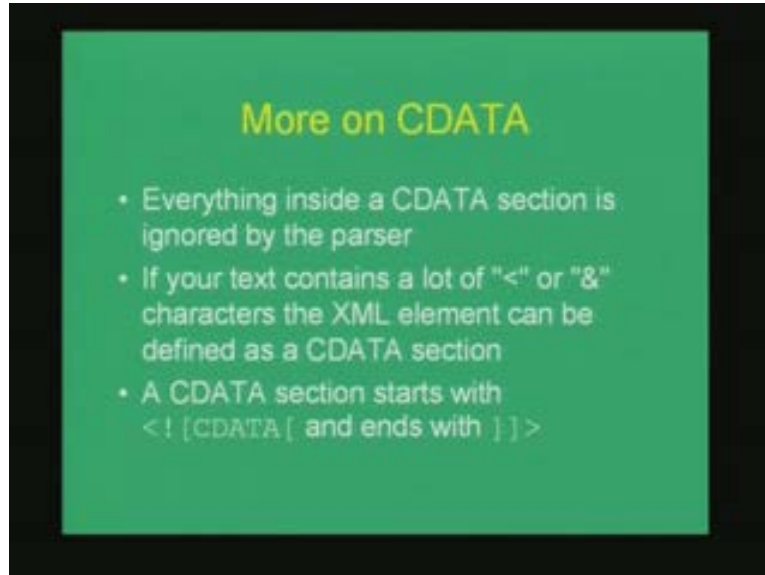
So for a given element, for every element you can define a separate name spaces where names for that element are uniquely describe globally across the world. And the syntax for name space begins with an xmlns, xml namespace declaration followed by a namespace prefix equal to a coated string following the namespace or coated string following the URI. And it basically gives a qualified name associated with a namespace.

(Refer Slide Time: 30:25)



So, all child elements if I define a namespace for a given element like in the example table there, all children elements of this element are associated with the same namespace. So their names are term to mean unique meta data that have been defined in this URI. The address is just simply the URI address that is specified is used to simply identify a namespace but the parser itself does not try to connect to that URI or look up that information or whatever. It is simply used for resolving naming conflicts and it doesn't necessarily have to check or validate whether the URI exists and whether the URI defines this names or anything of that sort.

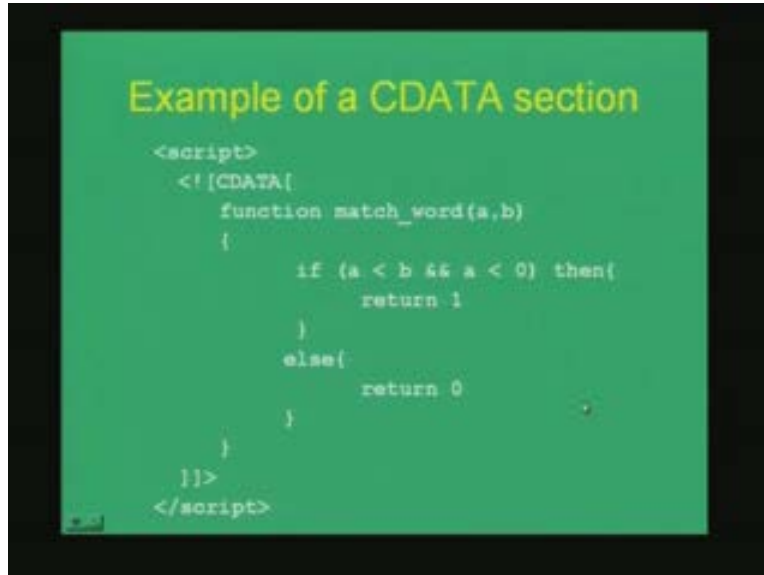
(Refer Slide Time: 31:19)



Similarly let us look at some more issues related to CDATA. CDATA if you remember is character data. Character data, when I define an element as character data then it means that everything until the slash of that element is taken in without parsing that is the data itself might contain other tags which are not parsed and everything is taken until the end of that element, so everything is ignored until the end of this element.

But then if the text or the character text contains a number of these characters, let us say less than or ampersand and so on, it is quite easy for XML parsers that especially if they are a little buggy it can be quite easy for XML parsers to get confused and look and try to parse them and especially if it is HTML data and not well formed and the parser might flag errors even though the XML itself is well formed and valid and so on. So there is another way of declaring CDATA where a CDATA section can be declared using a specific tag like this.

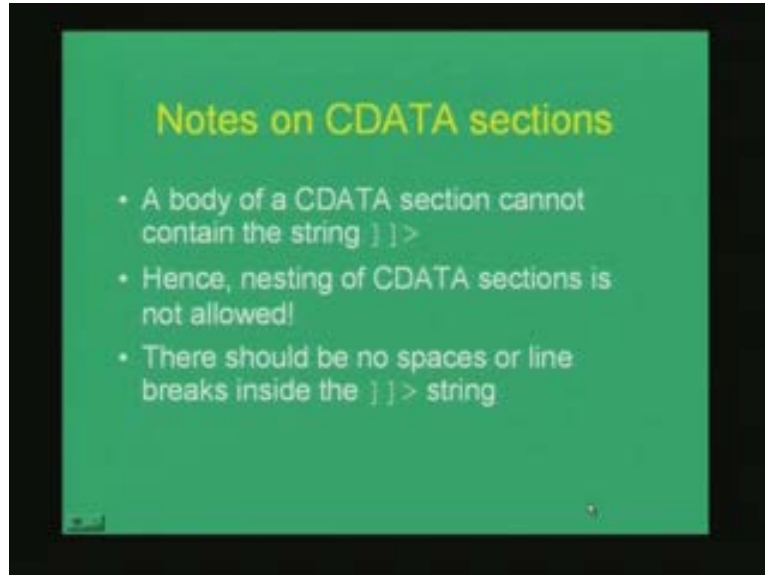
(Refer Slide Time: 32:44)



So this slide shows an example where script and slash script is an element that defines CDATA. And this element contains a CDATA section which says CDATA and then defines a function or defines some script here with less than and greater than symbols and ampersand symbols and then ends the CDATA section like this here. Now what are some of the rules for CDATA sections. Simply that the body of a CDATA section cannot contain this string which defines the end of the CDATA sections (Refer Slide Time: 33:28).

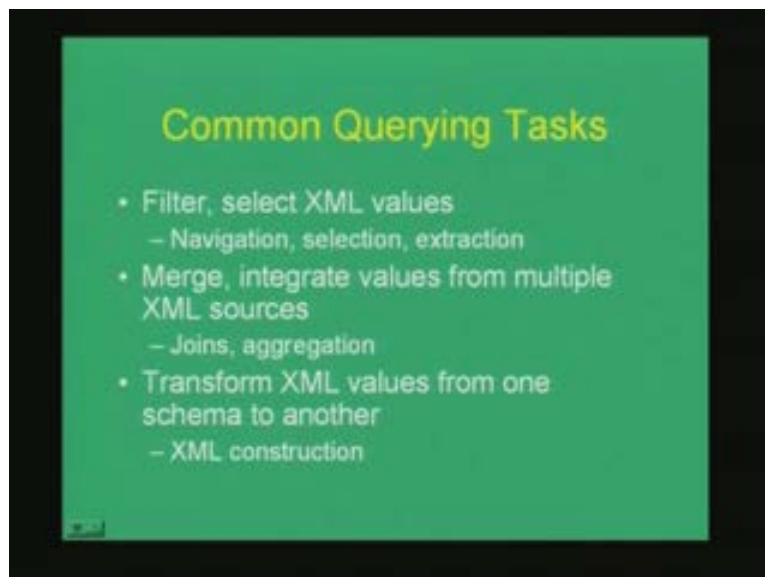
Hence, which also implies that nesting of CDATA sections is not allowed. So you can't have nested like you can't have nested comments in C, you cannot have a nested CDATA sections and there should be no spaces or line brakes inside this string.

(Refer Slide Time: 33:51)



Now let us look into querying XML data and what kinds of query languages are present and what paradigms of queries exist. The common querying tasks that are usually done over an XML data are something like filtering and selecting, navigation, selection, extraction and so on. In addition you could define some kind of joins or aggregation like we do in SQL and transformation that is convert one form of an SQL data to another.

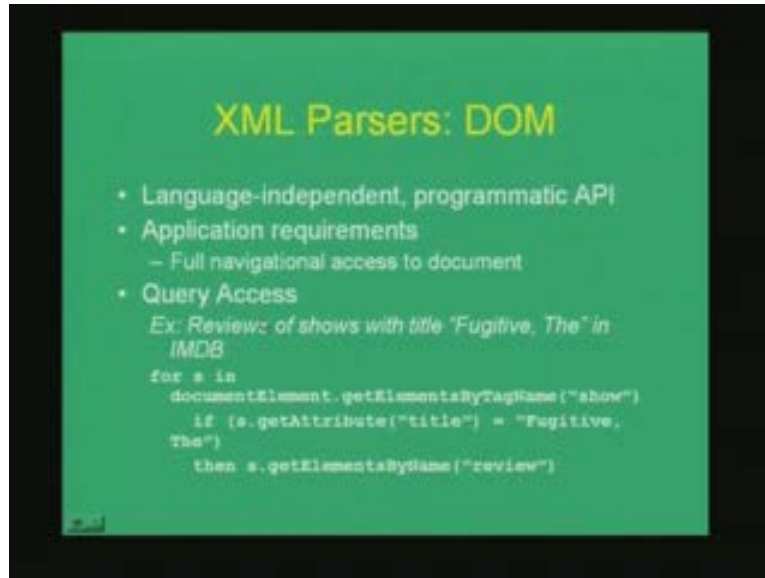
(Refer Slide Time: 34:11)



So before we start up with querying itself, we have to address a more fundamental problem or fundamental issue of XML parsing itself and **there is one**, there are two specific kinds of XML parsing which have implications on how querying is performed

and let us briefly look at these two paradigms of XML parsing and what kind of impacts they have on queries.

(Refer Slide Time: 35:27)

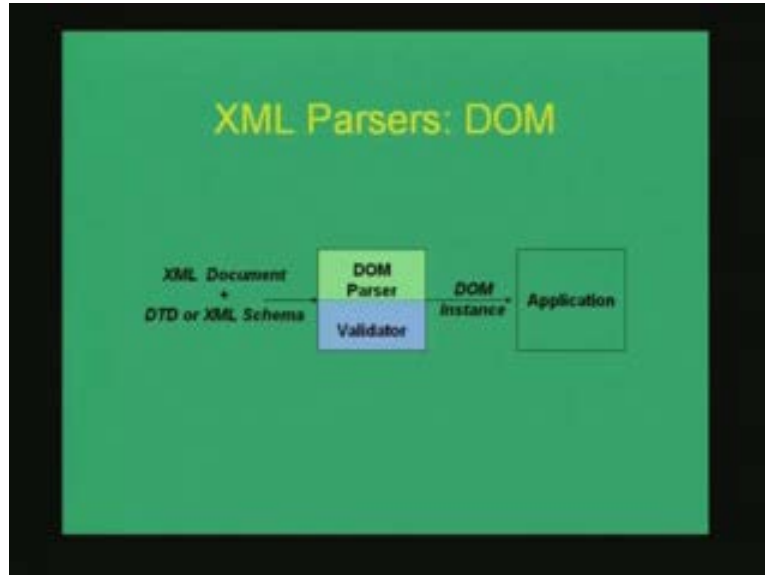


The first kind of XML parser is what is called as a full navigational parser that means of which you might have heard of this name called DOM or document object model. This is an example of such a parser which requires an entire XML document to be available for it before it can start parsing. That is the XML document should be full, complete and well-formed and so on and which can then be parsed and DOM basically creates some DOM object which can be called by application programs in order to query the XML data.

So the application requirements state that it should provide full navigational access to the document. So you cannot have partial XML string that is available and ask the DOM parser to start parsing. And kind of queries that DOM allows is something like this. that is DOM essentially creates an object which contains a XML document and then you can say something like if document element.getElementsByTagName show then, so which basically gets a given element of this thing then for s in this thing that is s is a element which is the set of all show elements. And in the show element if the title contains the fugitive then get the review and so on.

So you can address or dereference an element by the tag name and then it returns an element object to you and then you can get the data associated with that object and so on.

(Refer Slide Time: 37:02)



So this slide here shows how the DOM parser works that is you give an XML document plus a DTD or XML schema to the DOM parser. The DOM parser performs both parsing and validation and creates an object or a DOM instance which is then given to the application. The application then start calling the DOM instance by using it or integrating it with the other sets of its objects.

(Refer Slide Time: 37:47)



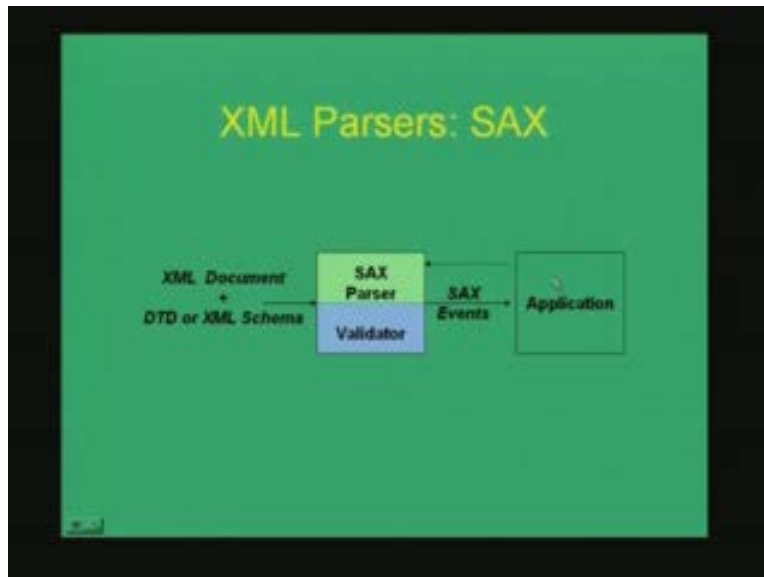
The other kind of XML parsing is what is called as stream based parsing. And example for such a stream based parser is SAX which is also widely available on the Microsoft windows platform.

Again this is a language independent and programmable, programmatic API and SAX in contrast to DOM does not require full navigational access to the given XML document. Instead, you can stream the XML data and it parses as the XML data parses through it. That means you can put an X, you can put a SAX parser on a network stream which is sending you XML data and as and when the XML data streams through, SAX creates a XML object, i mean it creates an object on the fly for the XML data that is parsing.

In other words it also means that SAX performs just one parse over the entire XML stream. And there are several applications where SAX is more important something like stock quotes which are sent in a streaming data over the web and as and when codes change, data streamed and an XML parser, a SAX like parser can parse them and what SAX does is it also has a feature of call backs. That is SAX not only parses the XML data but it also creates events that can call back into the application and interrupt the application and tell that something is happened, so take appropriate action and so on.

On the other hand in DOM it is the application who is in control and the DOM just creates a, DOM parser just creates an object and the application decides when to call the object and what to do with the object and so on. And of course in SAX its read only access for un typed notes and there is no in place updates that's possible.

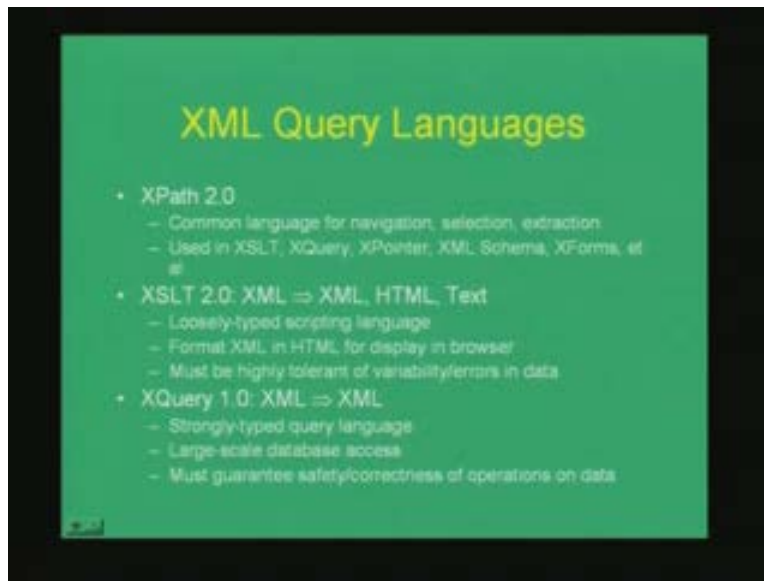
(Refer Slide Time: 39:56)



So, this slide shows how a schematic diagram of how the SAX parser works that is you have a XML document plus DTD which is streaming that is which need not be the entire document as such. Now this streaming XML document parses through a SAX parser and validator which in turn sends SAX events to the application. That is the application keeps interrupting, SAX parser keeps interrupting the application by sending appropriate signals or events to the application. And the application has to perform specific tasks associated with each of this events.

So it will say that found an element called show and which is valid and this is the element and so on. So, as and when a show element comes, the application says that a new show has arrived and this is the data that has to be rendered and so on. So the application begins the SAX parser but once the application begins, it's the SAX parser that's in control which calls back the application.

(Refer Slide Time: 41:00)



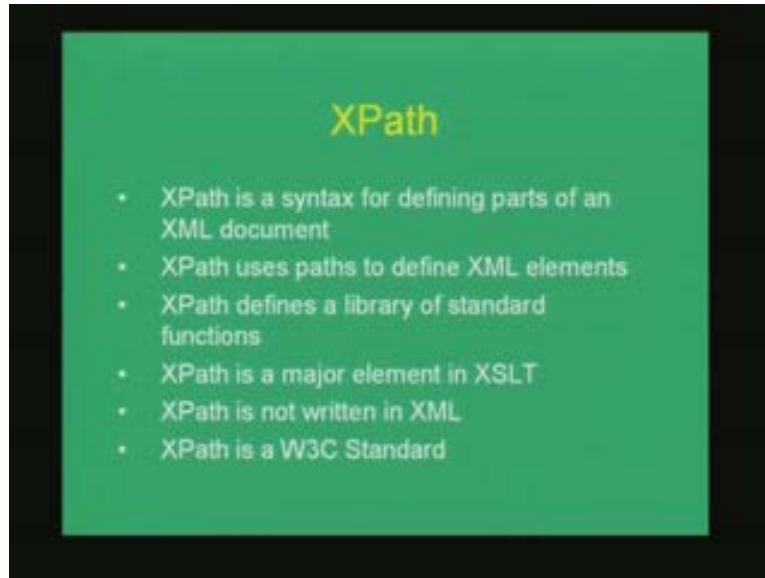
So, let us look at XML query languages. now look at these two kinds of parsing techniques let's see how they can affect queries as well but of course before that we should look at different kinds of XML query languages itself.

And some examples are shown in this slide here. you have query languages like XPath, XPath 2.0 which is very commonly used language for specifying navigations or selections or extractions from an XML tree and it's also used in, I mentioned the name called XSLT which is another kind of query language it's more like a transformation language. That is it can convert XML to HTML or it can convert XML to text or one form of XML to another form of XML and so on.

So if you want to render an XML document, you generally use an XSLT query language which will take an XML document and give out a corresponding HTML document in return. and then there is a XQuery kind of language which is a, what may be termed as a composable language in the sense that one XQuery takes an XML document as input and gives out another XML document as output. It's very similar to the relational algebra queries which takes a relation as input and gives a relation as output.

And it's a strongly typed XML, it's a strongly typed query language and useful for large scale database accesses itself.

(Refer Slide Time: 42:40)



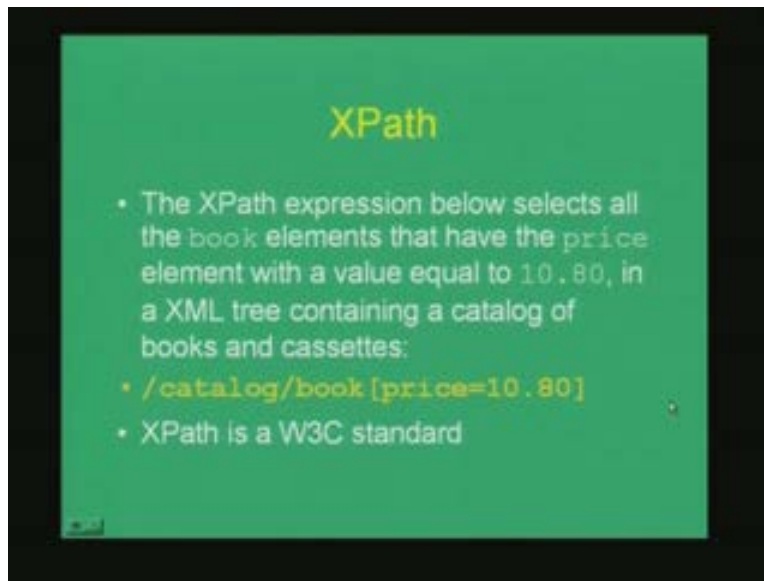
So let us briefly look at XPath. XPath is a syntax for defining parts of an XML document and basically the way it defines parts is to use a directory like parts structures where in order to define XML stands or XML elements.

(Refer Slide Time: 42:56)



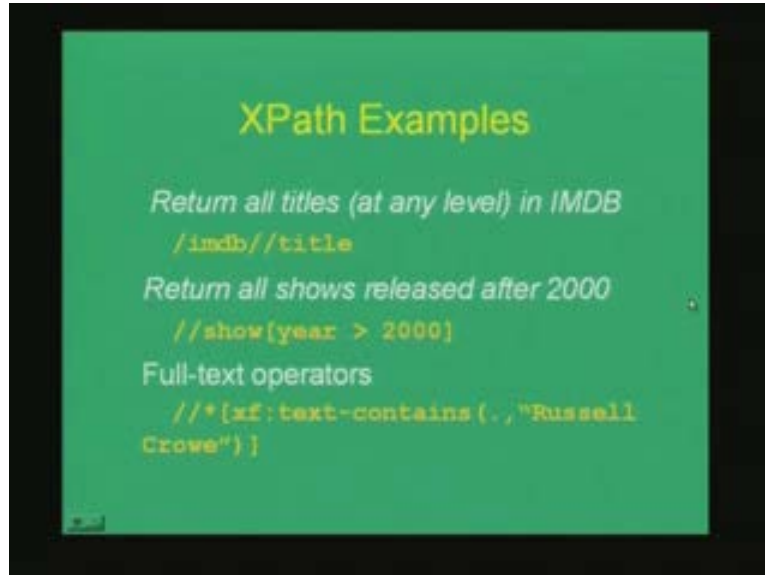
Let us quickly go to an example to illustrate this. This slide essentially says that path expressions in XPath look very similar to the directory structure in a computer file system. So I might have a directory called this slash this and slash this and so on. So if this is what to be an XPath expression, all this would be elements so some OSL.iiitb.ac.in would be an element and grace would be an element and so on.

(Refer Slide Time: 43:48)



So, this slide shows an example where an XPath selects a book element which lies under a catalog element where the price attribute of the book element is equal to 10.80. And the catalog element is the root element of this XML document. So this is just like the root directory in your simple file system language and so the root directory contains, defines the element called catalog, under the catalog look for an element called book and look for an attribute matching this particular criteria.

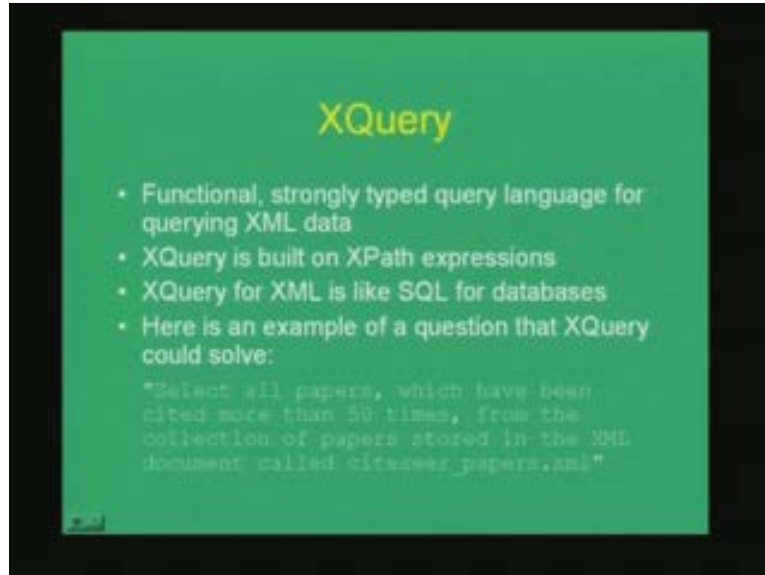
(Refer Slide Time: 45:14)



Similarly there are other examples here. If you have a double slash like this, as shown in this example it says that return all titles at any level in the imdb XML document. That is the root element is imdb and title can occur at any level, so double slash essentially means any level in your document. And similarly this one double slash at the beginning again says that return a show element at any level and doesn't matter what the root element also is but where the show year is greater than 2000 that is all shows released after 2000.

And you can also have full text operators like text contains Russell Crowe and so on anywhere in here... there is no element name as well that is given, so it just says star. So any element in this XML document at any level return that element where the text contains Russell Crowe. So that's about XPath where, it's a very brief introduction to XPath where which is a file system like language for representing different navigational aspects of an XML document.

(Refer Slide Time: 46:04)



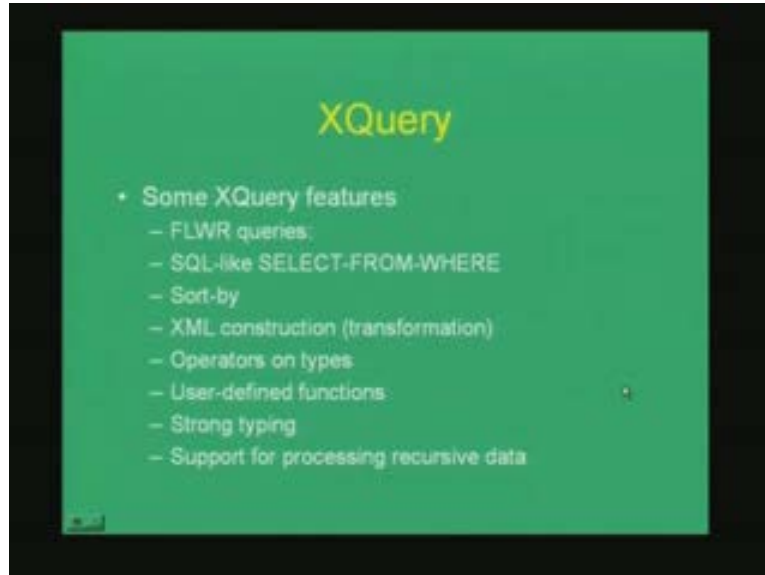
And this is where the hierarchy or tree structure of an XML document becomes significant in the sense that it is easy to express a tree structure in the form of a directory structure as done in XPath plus it is easy to enforce constraints. That is in a tree structure, a node can have at most one parent that is any non-root level node will have exactly one parent and there is exactly one path from the root to any given element in a XML tree. So you can actually specify one long file system like path which uniquely identifies each element in the XML document.

The next kind of query that we are going to look at or query paradigm that we are going to look at is the XQuery paradigm. XQuery is a functional language in the, it's a strongly typed query language for querying XML data. and XQuery as I said earlier is an XML to XML converter that is it can query an XML document and return an XML document just like the relational algebra queries that can query a relation and return a relation.

An XQuery in turn can use XPath expressions for its queries, so XPath can become a part of XQuery in specifying its query expressions. And many people would term XQuery as an SQL for XML databases. So it's analogous to SQL in the sense that it takes an XML document, returns an XML document, it contains several different operators, it can contain several different sophisticated query operators something like select from where or and so on.

So you can actually specify a query like select all papers which have been sighted more than 50 times from the collection of papers stored in the XML document called citeseer papers and so on. So you can give complex expressions based around a select from where kind of class.

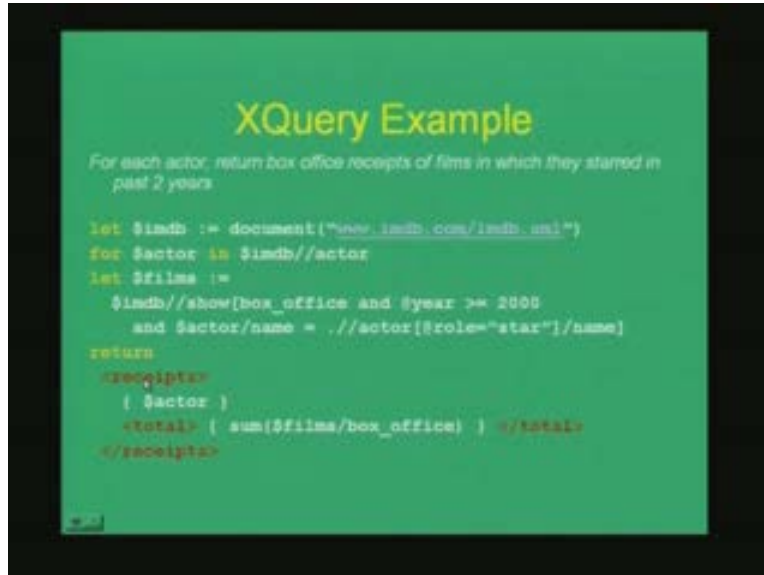
(Refer Slide Time: 48:20)



So what are some of the XML XQuery features? They are what are called as flower queries or FLWR queries which say that which stands for let where return repeat queries. That is it's a looping kind of queries where you can specify a for condition initial condition let where is similar to the SQL where and then repeat, so you can repeat these queries. We will see an example of this shortly.

Then there are SQL like select from where clauses were select so and so from this XML element where this condition matches. Then there are sort by operators, so you can sort elements based on certain attributes then XML construction that is transforming one XML document to another XML document. You can also have user defined functions where on this and XML XQuery basically supports strong typing. So you can say something like an integer or character or you can perform operations that are specific to integers versus character strings and so on. And it also has supports for processing recursive data sets.

(Refer Slide Time: 50:04)



Here is an example of an XQuery query that is a query written in the XQuery language. So the query essentially says that for each actor return box office receipts of films in which they starred in the past 2 years. And essentially let us go back to, let us go first to this last part of the XQuery document where here you say lot of let for and so on and so forth, for let where and so on and repeat and so. But this last one here is what is going to tell the Xquery engine what to return. Now here that imdb engine or the imdb XML document did not contain an element called receipt.

However what it does contain are elements like box office or actor and so on. And what the XQuery languages returning is an XML element called receipts and slash receipts which in turn is made up of actors and totals and so on. So receipt is, receipt essentially is an XML document comprising of character data which tells what is the actor and then one more XML element called total which contains the sum of all box office I mean films per box office receipts that they have obtained over the past 2 years.

And here an XQuery for example can first define variables like using a Pascal like syntax, so it says let dollar imdb equal to this document. So imdb basically specifies this particular document and then this is for any actor in imdb actor. That is actor is another variable. Note that all variables are prefixed by a dollar sign and where actor stands for any actor element at any level in the imdb document. So for any actor in this thing, let films equal to this one that is the show element where it contains a box office element and year greater than 2000 and actor name is the particular actor is entered as a star in this show, so star name and so on.

So essentially you let this one and for each actor this for let and the where is implicit here and return is also specified. So, this one iteratively performs for each actor at any level in the specified in the imdb document. And then it returns a set of this receipts elements which says actor and total that the actor has grossed.

(Refer Slide Time: 52:51)



And here are some www links or World Wide Web where you can get more information about DOM parsers or SAX parsers or XPath queries and XQuery standards and so on. But before this let us go back to the parser problem that we are talked about. That is how does a parser affect the queries that you give on an XML document. Both DOM and SAX parsers for example create documents or create objects which the application can access that is which the application can send messages to an access.

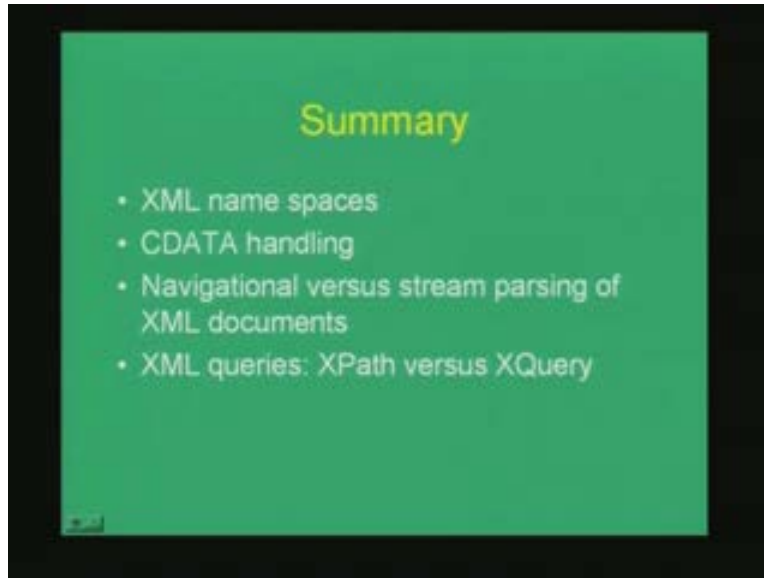
However the DOM parser requires a entire XML document to be present whereas the SAX parser calls back into the application as and when XML elements parse through the parser. Now what kind of implication does it have on queries? Let us take the query here. Let us say for each actor return box office receipts of films in which they starred in the past 2 years. This is when you give this, such a query to a XML document that is parsed using DOM, you essentially know that the entire XML document has been navigated and parsed and that is present in the object that is available here. So you can essentially go and look through the object and return the query results.

On the other hand if it is a SAX parser and it's a streaming XML data then you won't be sure whether you have encountered all possible actors in this loop. That is take a look at this (Refer Slide Time: 54:45) for actor in imdb slash actor, so you will have no idea whether all actors have been processed as part of this XML document.

So in such a sense, in such cases this query has to be in the form of what is called as a standing query. That is in the traditional database setting, the database is static and the query parses through the database and then returns query results. But here it's the other way around, the query is standing, the query is static and the database or data set parses through the query and the query returns backs or returns events or performs call back into the applications whenever a data set matching the query is available.

So as and when the query finds an actor and show satisfying this criterion, it returns a receipt kind of XML fragment by calling back into the application.

(Refer Slide Time: 56:14)



So let us summarize what we learnt today in this session. We started with XML namespaces and how to resolve naming conflicts by assigning a unique, globally unique URI for each name prefix in an XML document. And we also looked at some CDATA handling issues where especially if your document, if your CDATA contains angular braces and ampersand symbols and so on, you can embed it within a CDATA section.

Next we looked at two different kinds of XML parsing namely the navigational parsing of the entire document versus stream parsing and then we looked at XPath and XQueries as query languages or query paradigms over XML data sets. And how these kinds of parsing techniques can impact the kinds of queries that or how the query behaves in each of these parsing techniques. So that bring us to the end of this session.