UNIT 3 OBJECT BASED DATABASE AND XML

1.0 Introduction

1. Objects and Identities

Q: Describe objects and Identities

A: Object – an abstract representation of real - world entity that has a unique identity.

- Each **real**-**world** entity is modelled as a database object
- Object is an instance of a class that encapsulates data & behavior.
- Object needs unique identity to be stored in system.
- Called as OID (Object Identifier)
- OID is used by system to identity each object uniquely and create and manage inter object references.

Q: what are complex objects?

A: Complex Objects

- It is an object that includes **set of attributes** associated to it.
- Value of an attribute can be an object or set of objects
- Complex objects are built by applying constructor to simple objects
- Which are data types such as integers, real numbers, characters, strings, bullion, etc.

Q: define encapsulation.

2. **Encapsulation**
   Internal structure and data of object is hidden from the outside world behind a limited set of well-defined interfaces

Q: define polymorphism

A: Polymorphism

It is the ability of an object to take on many forms.

Q: what are classes and inheritance

A: Classes and inheritances

A class is defined as a collection of similar objects with shared structure and behavior
Inheritances help create hierarchy of classes
A class is defined as instance of another class and will inherit attributes and methods of such class.
Sub class – inherits attributes of super class
Superclass – its attributes are inherited by sub class

Q: What is Object oriented Database Management System (OODBMS)?

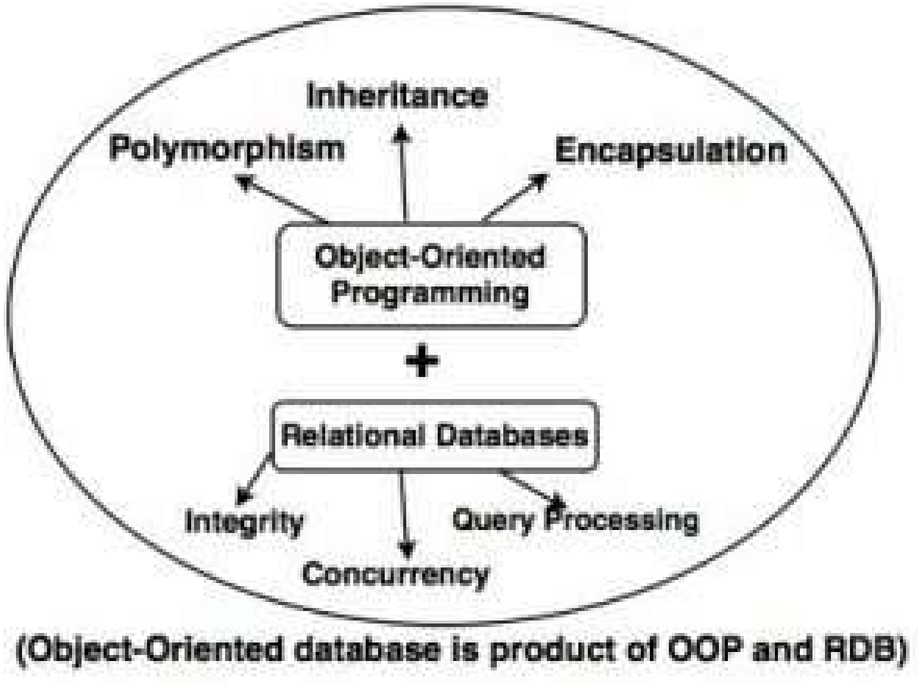
It is a data model that captures the semantics of objects supported in Object oriented programming.

OODBMS is manager of OODB.

It supports modelling and creation of data as object.

Examples of OODBMS – Versant Object Database, Objectivity DB, Object Store, ONTOS, V base, Orion, ZODB.

Features of OODBMS



Object-Oriented database

1. **Complexity:** it can represent complex internal structure with multilevel complexity
2. **Inheritances:** new object can be created from existing object that inherits all characteristics of existing object
3. **Encapsulation:** data hiding- which binds data and functions together – which can manipulate data and not visible to outside world
4. **Persistency:** OODBMS allows to create persistent object- object remains in the memory after execution→ solves problem of recovery and concurrency

Advantages of OODBMS

1. **More Expressive Query Language:** provides navigational access from one object to next data
2. **Reduced Redundancy and Increased Extensibility:** allows new datatype to be built from existing types-common properties of superclass are shared with sub class
3. **Enriched Modeling Capabilities:** allows the real-world to be modeled more closely
4. **Removal of Impedance Mismatch:** it provides single language interface between Data Manipulation Language (DML) and programming language
5. **Improved performance:** OODBMS improves overall performance of DBMS
6. **Applicability to Advanced Database Applications:** suitable for advanced database applications such as – CAD, OIS (Office Information System), CASE, Multimedia system etc.
7. **Support for Long-Duration Transaction:** uses different protocols to handle long duration transactions
8. **Support for Schema Evaluation:** schema evaluation is more feasible. Schema is better structured.

Disadvantages of OODBMS

1. OODBMS has lack of universal data model and standards
2. In OODBMS locking at object level may impact performance
3. OODBMS is complex due to increased functionality
4. OODBMS lack of support for views and also for security.

Q: Drawbacks/obstacles in relational Data model/ relational database

1. Limited type system supported by relational model
2. Difficulty in accessing database
3. Limited Operations- too restrictive for real world objects
4. Not support inheritance

Q: what is OO (Object Oriented Database) Advantages?

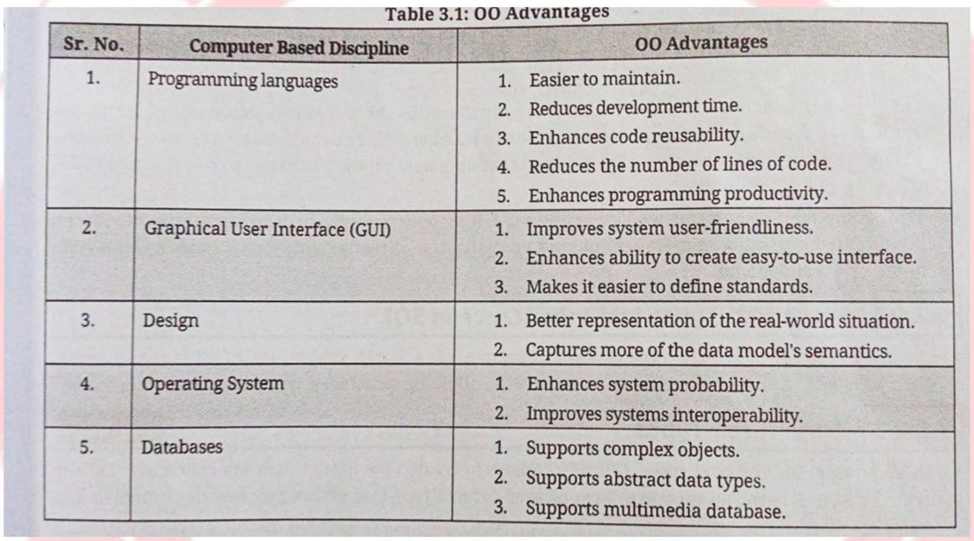
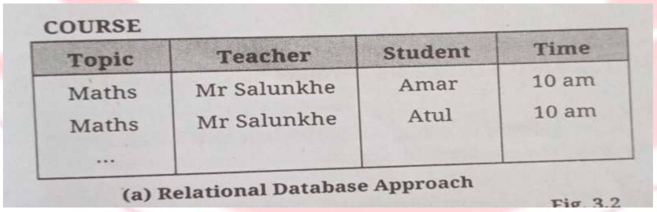


Table 3.1: OO Advantages

Sr. No.	Computer Based Discipline	OO Advantages
1.	Programming languages	1. Easier to maintain. 2. Reduces development time. 3. Enhances code reusability. 4. Reduces the number of lines of code. 5. Enhances programming productivity.
2.	Graphical User Interface (GUI)	1. Improves system user-friendliness. 2. Enhances ability to create easy-to-use interface. 3. Makes it easier to define standards.
3.	Design	1. Better representation of the real-world situation. 2. Captures more of the data model's semantics.
4.	Operating System	1. Enhances system probability. 2. Improves systems interoperability.
5.	Databases	1. Supports complex objects. 2. Supports abstract data types. 3. Supports multimedia database.

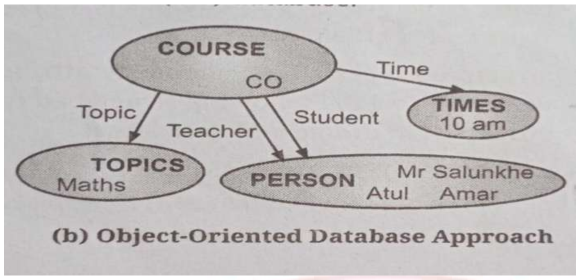
Q: Depict approaches of relational db and object oriented db


COURSE

Topic	Teacher	Student	Time
Maths	Mr Salunkhe	Amar	10 am
Maths	Mr Salunkhe	Atul	10 am
...			

(a) Relational Database Approach

Fig. 3.2



(b) Object-Oriented Database Approach

Q: Explain Complex Data types

A: - Traditional DB have simple datatypes.

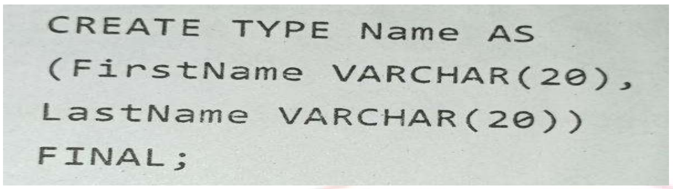
- Basic data items are small unstructured with few record types only.
- Recent years demand grown for more complex data types
- **Example** – Entire address can be viewed as an atomic data item (hide details such as street address, city, state, postal code)
- If address is broken into its components → querying would be more complicated.
- With complex data type we can represent →E-R model concepts such as composite attributes, multi valued attributes without a complex translation to relation model.

Q: Explain Structured types in SQL

A: - it is a form of **user defined type** that **allows representing complex data types.**

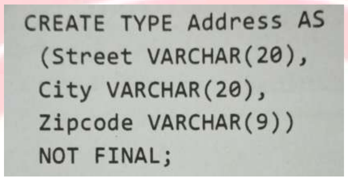
- Using structured type, we can represent **composite** attributes and **multivalued** attributes of E-R diagram efficiently.

- Example –composite attribute **name** with its component attributes – FirstName, LastName



```
CREATE TYPE Name AS
(FirstName VARCHAR(20),
LastName VARCHAR(20))
FINAL;
```

- Final – You cannot create sub types (Child types)
- Similarly composite attribute **address** is represented as follows:



```
CREATE TYPE Address AS
 (Street VARCHAR(20),
 City VARCHAR(20),
 Zipcode VARCHAR(9))
 NOT FINAL;
```

- Not final keyword signifies the type can be inherited
- These are user defined types with E-R diagram as follows:

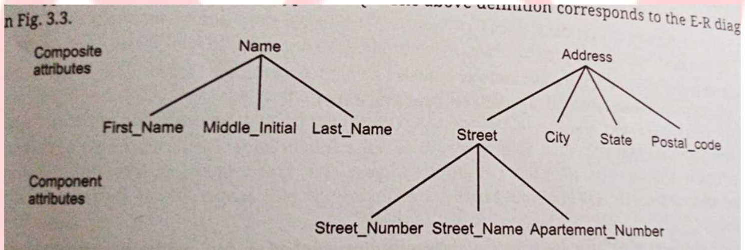
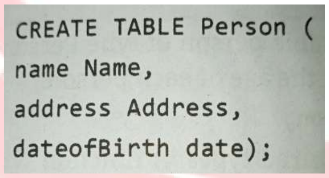


Fig. 3.3: Composite Attributes (Name and Address)

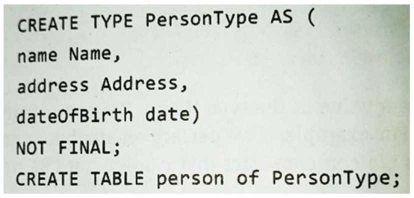
-
- These types can be used to create composite attributes in a relation.



```
CREATE TABLE Person (
name Name,
address Address,
dateofBirth date);
```

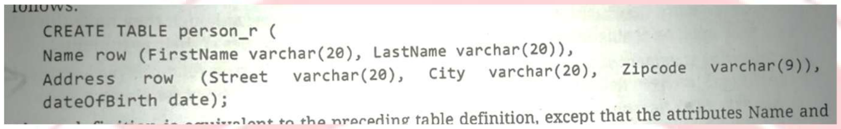
-
- The components of composite attributes can be accessed using a dot notation
- name.FirstName

- We can also create a table whose rows are of user defined type



```
CREATE TYPE PersonType AS (
name Name,
address Address,
dateOfBirth date)
NOT FINAL;
CREATE TABLE person of PersonType;
```

- Another way of defining composite attribute in SQL is using **unnamed row type**

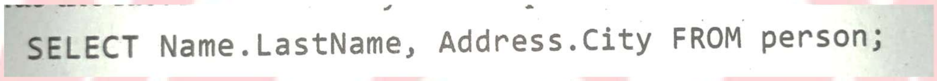


follows:

```
CREATE TABLE person_r (
Name row (FirstName varchar(20), LastName varchar(20)),
Address   row   (Street   varchar(20),   City   varchar(20),   Zipcode   varchar(9)),
dateOfBirth date);
```

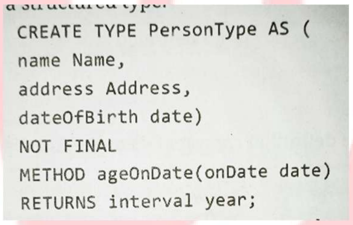
definition is equivalent to the preceding table definition, except that the attributes Name and

-
- Query to **access** components of attributes of a composite attribute is



```
SELECT Name.LastName, Address.City FROM person;
```

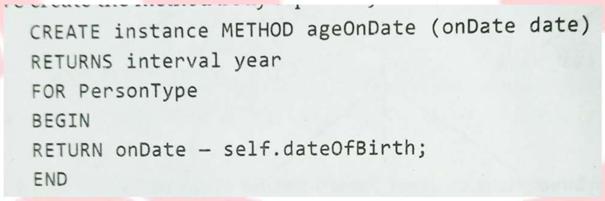
-
- A structured type can have methods defined as follows



a structured type:

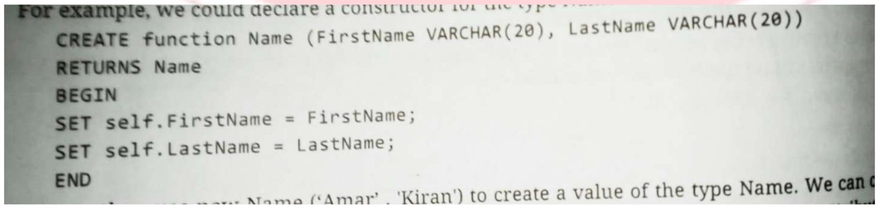
```
CREATE TYPE PersonType AS (
name Name,
address Address,
dateOfBirth date)
NOT FINAL
METHOD ageOnDate(onDate date)
RETURNS interval year;
```

-
- Method body can be created separately as follows



```
CREATE instance METHOD ageOnDate (onDate date)
RETURNS interval year
FOR PersonType
BEGIN
RETURN onDate - self.dateOfBirth;
END
```

-
- **Constructor** functions are used to create values of structured types

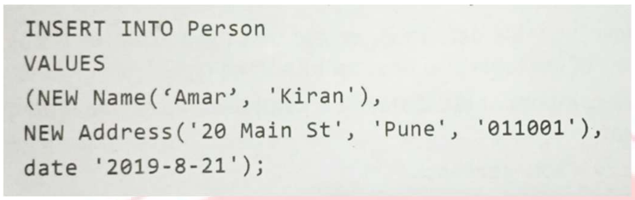


For example, we could declare a constructor for the type

```
CREATE function Name (FirstName VARCHAR(20), LastName VARCHAR(20))
RETURNS Name
BEGIN
SET self.FirstName = FirstName;
SET self.LastName = LastName;
END
```

new Name ('Amar', 'Kiran') to create a value of the type Name. We can

-

- There can be **more than one constructor** for the same structured type with same name.
- We can also create a **tuple**
- 

```
INSERT INTO Person
VALUES
(NEW Name('Amar', 'Kiran'),
NEW Address('20 Main St', 'Pune', '011001'),
date '2019-8-21');
```

Q: What is Inheritance in SQL?

A: - With Inheritance new objects can be defined as being a particular type of an existing object type (class) and inherit the pre - defined attributes and behavior of that type

- In OODB Inheritance is followed in **2 ways**:
  1. For **reusing** and **redefining** types (type Inheritance) and
  2. For creating **hierarchies of collection of similar objects** (table Inheritance)
- Inheritance allows a type or table to acquire properties of another type or table.
- Sub type or sub-table Inherits properties
- Supertype or super table – whose properties are Inherited
- Hierarchical relationship can exist between 2 structured types

Q: What are Types of inheritance in SQL

Ans: A type hierarchy provides the following **advantages**

1. It encourages modular implementation of the data model
2. It ensures consistent reuse of schema components
3. It ensures no data fields are accidently left out

Fig shows

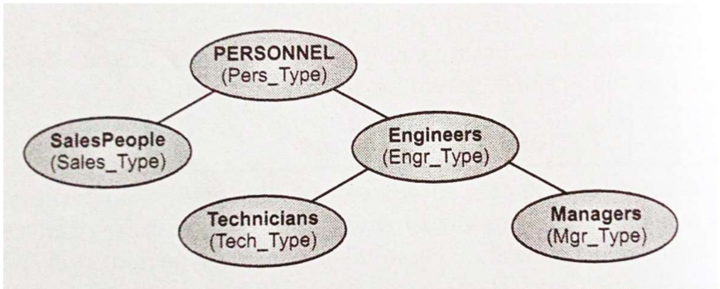


Fig. 3.4: Natural Class Hierarchy for a Personnel Application

1. How inheritance might work in a company model
2. All employees are members of class personnel and they all have attributes associated with being an employee (employee number, name, address)

- Inheritance can be applied to named row type only
- There are several forms of inheritance:

1. Single inheritance

Sub classes inherit only from one superclass

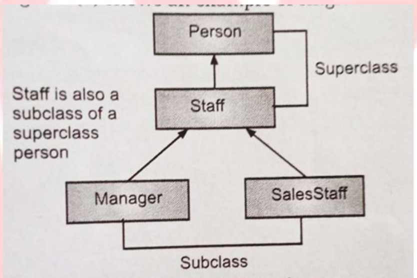


Fig. 3.5 (a): Single Inheritance

2. Multiple inheritance

- Subclass inherit from more than one superclass

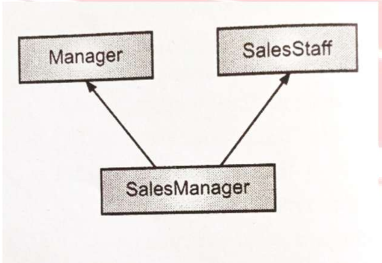


Fig. 3.5 (b): Multiple Inheritance

- Multiple inheritance is very problematic

- **Conflict** arises when superclass contains same attributes or methods.
- This can be handled by using name of superclass as a **qualifier** Use single inheritance to avoid conflict
- Eg. SalesManager→Manager→ SalesStaff
- Or
- SalesManager→ SalesStaff→ Manager

3. Repeated inheritance

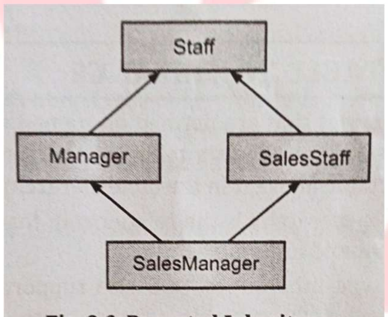
- It is a special case of multiple inheritance
- Superclasses inherit from **common superclass**
- Inheritance **ensure** that sub class does not inherit properties from superclass twice
- 

Fig. 3.6: Repeated Inheritance

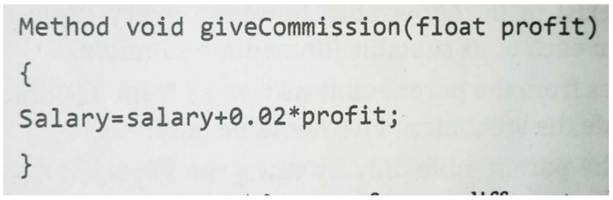
4. Selective inheritance

- It allows a sub class to inherit limited number of properties from superclass

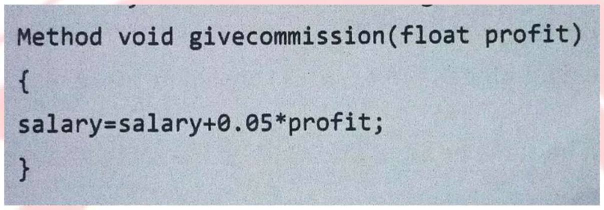
Q: Explain Concept of Overriding and Overloading

Overriding:

- Properties are automatically inherited by subclass from their superclass
- But it is possible to redefine a property in the subclass
- This is called overriding
- E.g. a method is defined in the staff class to increment salary

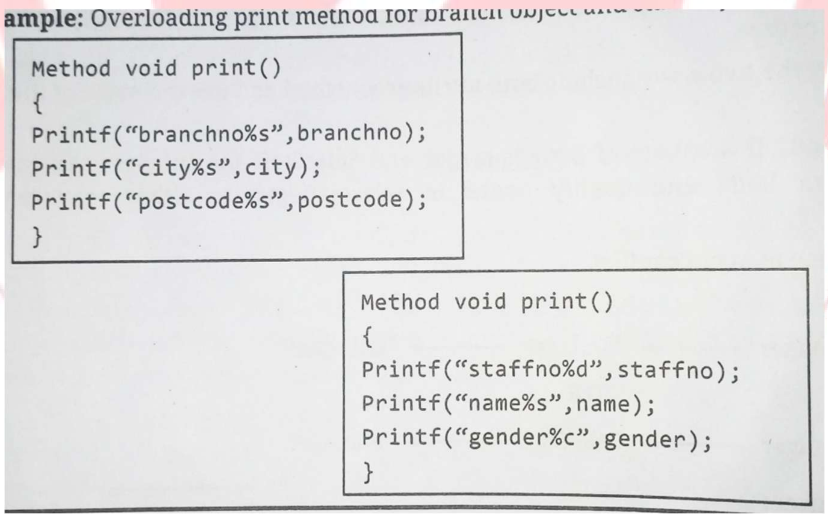
- 

```
Method void giveCommission(float profit)
{
Salary=salary+0.02*profit;
}
```

- If we perform different calculation for commission in manager subclass
- This can be done by redefine the overriding
- 

```
Method void givecommission(float profit)
{
salary=salary+0.05*profit;
}
```

Overloading

- **Name** of the method is **reused** within a class definition
- Single message can perform different functions <u>depending upon which object receives it</u>
- E.g. overloading print method for branch object and staff object
- 

ample: Overloading print method for branch object and ...

```
Method void print()
{
Printf("branchno%s",branchno);
Printf("city%s",city);
Printf("postcode%s",postcode);
}
```

```
Method void print()
{
Printf("staffno%d",staffno);
Printf("name%s",name);
Printf("gender%c",gender);
}
```

Q: What is Table inheritance? Write down its advantages.

- Tables that are defined on named rows type support table inheritance
- it allows the table to inherit the behavior from supertable above it in the table hierarchy
- table hierarchy→ relationship among tables → subtable inherits behavior of supertable
- SQL supports Table inheritance
- Ex. We define BOOK table as
  - CREATE TABLE BOOK OF Book Type;
- Now we can define Text Book as subtable of BOOK
  - CREATE TABLE TextBook OF TextBookType UNDER BOOK;
- Type of subtable must be the **subtype** of the type of the parent table
- Every tuple in subtable implicitly exist in supertable
- A **query** on the parent table finds all the tuples from **parent** table as well as from its **subtable**
- We can restrict the query to find tuples in the **parent table only**
  - Ex. SELECT Book_Title FROM BOOK ONLY;

Advantages of Table inheritance

- Encourages modular implementation of data model
- Ensures consistence reuse schema components
- Allows to construct queries whose scope can be sum or all the tables in table hierarchy

Q: Explain 'Array and Multiset types' in SQL

SQL supports 2 collection types→ Arrays and Multiset

Array

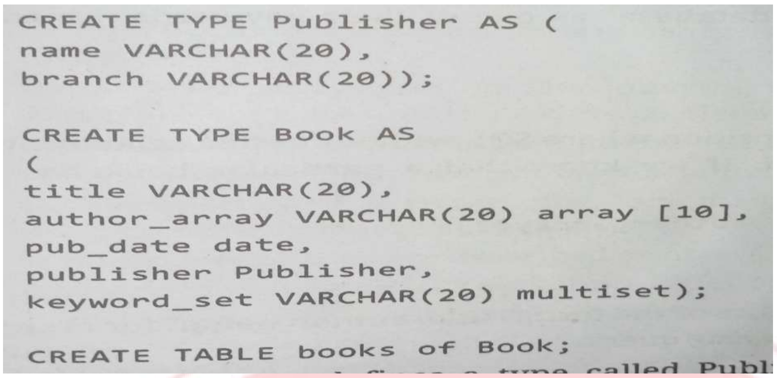
- It is an **ordered collection** of elements

- Array type violates 1NF(First Normal Form in DBMS)
- Array types enable us to **enhance** a field of an existing type by **putting more than one entry in to it**.
- This creates a repeating group
- Arrays are order such that each element in the array corresponds to exactly 1 ordinal position in the array
- CREATE TYPE colors_array AS VARRAY(5) OF VARCHAR(20);
- CREATE TABLE products (
-   id INT,
-   name VARCHAR(50),
-   colors colors_array
-);
- INSERT INTO products (id, name, colors)
- VALUES (1, 'Shirt', colors_array('Red', 'Blue', 'Green'));
- SELECT colors[2] FROM products WHERE id = 1;

-- Output: Blue

Multiset

- It is an **unordered collection of elements**
- An element may occur multiple times
- We cannot reference one element in multiset because their **location** in the collection is **unknown**.
- Ex. Array and Multiset valued attributes can be defined as



```
CREATE TYPE Publisher AS (
name VARCHAR(20),
branch VARCHAR(20));

CREATE TYPE Book AS
(
title VARCHAR(20),
author_array VARCHAR(20) array [10],
pub_date date,
publisher Publisher,
keyword_set VARCHAR(20) multiset);

CREATE TABLE books of Book;
```

```
-
- CREATE TYPE fruits_multiset AS TABLE OF VARCHAR(20);
-
- CREATE TABLE fruit_basket (
-   id INT,
-   fruits fruits_multiset
- );
-
- INSERT INTO fruit_basket (id, fruits)
- VALUES (1, fruits_multiset('Apple', 'Banana', 'Apple'));
```

- This allows storing multiple instances of the same value (e.g., two Apples).

Feature	**Array**	**Multiset**
Size	Fixed	Dynamic
Duplicates	Not allowed	Allowed
Order	Maintains order	Unordered
Use Case	Structured, indexed data	Flexible, repeated values

Creating and Accessing collection values

Array:

- array['abc','pqr','xyz']

Multiset:

- multiset['computer','database','sql']
- we can access or update elements of an array by specifying the array index
- array[2]

Q: Write down Features of OODBMS.

- Relationships are represented by reference via object identifier (OID)
- Employees **indexing** techniques to locate disk pages that store the object
- Handles larger and complex data
- **DML** (data manipulation language) is incorporated into a programming language
- Stored data entries are described as **object**
- Object oriented database can handle different types of data
- Data is stored in the form of **object**
- E.g. of OODBS – GemStone/s, ObjectDB, db4o, ObjectDatabase++

Q: what are Use cases /applications of OODBMS

- CAD applications (AutoCAD, SolidWorks)
- AIML (storing complex models)
- Real-time system (defense, aerospace)

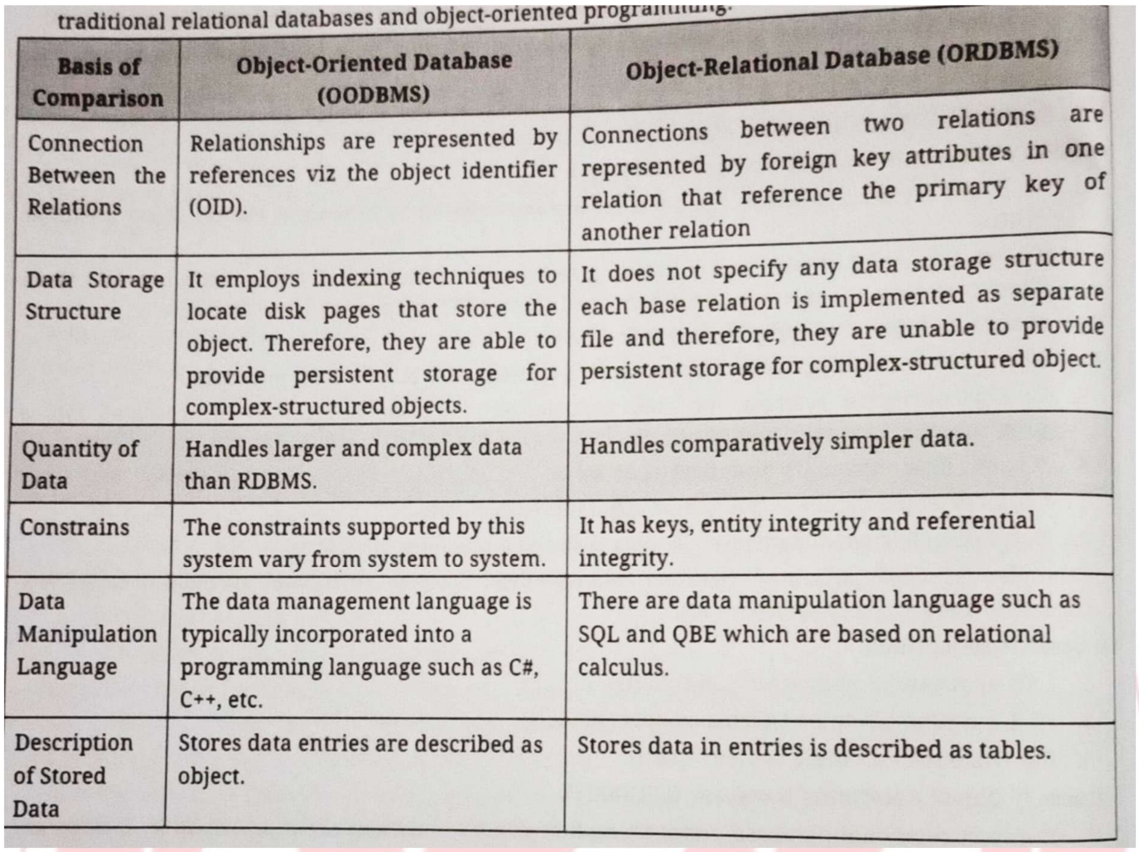
Q: write down Features of ORDBMS (object – relational database)

- Connections between two relations are represented by **foreign key** attributes in one relation that reference the **primary key** of another relation
- Relational database does not specify any data storage structure.
- It handles simpler data.
- Data Manipulation Languages (**DML**) used SQL, QBE
- It stores data in **entries** – describe as tables
- It can handle **single type** of data.
- Table contains rows and columns
- Ex. ORDBMS, PostgreSQL, Microsoft SQL server, IBM DB2.

Q: what are Use cases /applications of ORDBMS-

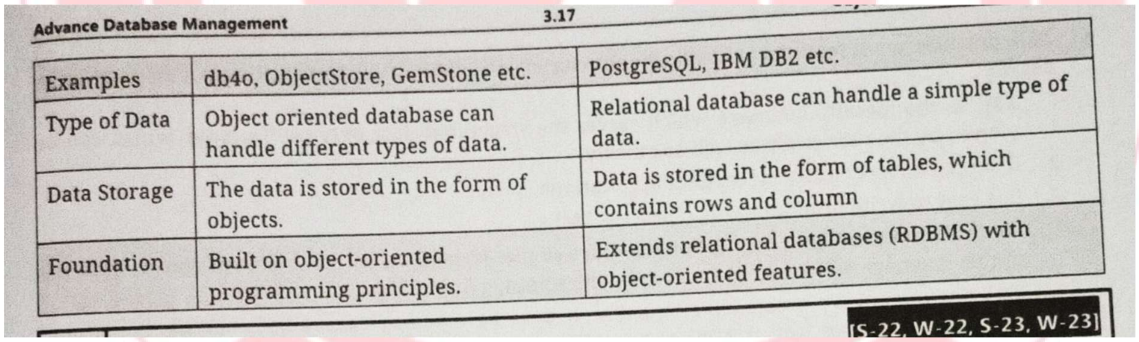
- Enterprise applications (Banking, Finance)
- E-commerce (storing structured and semi structured data)
- Geographic Information System (GIS) (Handling spatial data)

Q: Differentiate between OODBMS and ORDBMS



traditional relational databases and object-oriented programming.

Basis of Comparison	Object-Oriented Database (OODBMS)	Object-Relational Database (ORDBMS)
Connection Between the Relations	Relationships are represented by references viz the object identifier (OID).	Connections between two relations are represented by foreign key attributes in one relation that reference the primary key of another relation
Data Storage Structure	It employs indexing techniques to locate disk pages that store the object. Therefore, they are able to provide persistent storage for complex-structured objects.	It does not specify any data storage structure each base relation is implemented as separate file and therefore, they are unable to provide persistent storage for complex-structured object.
Quantity of Data	Handles larger and complex data than RDBMS.	Handles comparatively simpler data.
Constrains	The constraints supported by this system vary from system to system.	It has keys, entity integrity and referential integrity.
Data Manipulation Language	The data management language is typically incorporated into a programming language such as C#, C++, etc.	There are data manipulation language such as SQL and QBE which are based on relational calculus.
Description of Stored Data	Stores data entries are described as object.	Stores data in entries is described as tables.



Advance Database Management 3.17

Examples	db4o, ObjectStore, GemStone etc.	PostgreSQL, IBM DB2 etc.
Type of Data	Object oriented database can handle different types of data.	Relational database can handle a simple type of data.
Data Storage	The data is stored in the form of objects.	Data is stored in the form of tables, which contains rows and column
Foundation	Built on object-oriented programming principles.	Extends relational databases (RDBMS) with object-oriented features.

[S-22, W-22, S-23, W-23]

XML – eXtensible Markup Language

Q: What is XML? Write short note on XML

- XML: eXtensible Markup Language
- Text based markup language
- Derived from SGML (Standard Generalized Markup Language)
- HTML is suitable for formatting and structuring contents of webpages but not suitable for not representing structured data i.e. extracted from databases.
- Therefore, a new language XML is developed.
- XML provide information about how to structure data in the webpages and meaning of certain components of data.
- XML tags identify data and are used to store and organize data
- XML is not going to replace HTML.
- Data in XML database can be queried transformed exported and returned to calling system.
- It is used to store huge amount of information.
- Data can be queried using XQuery

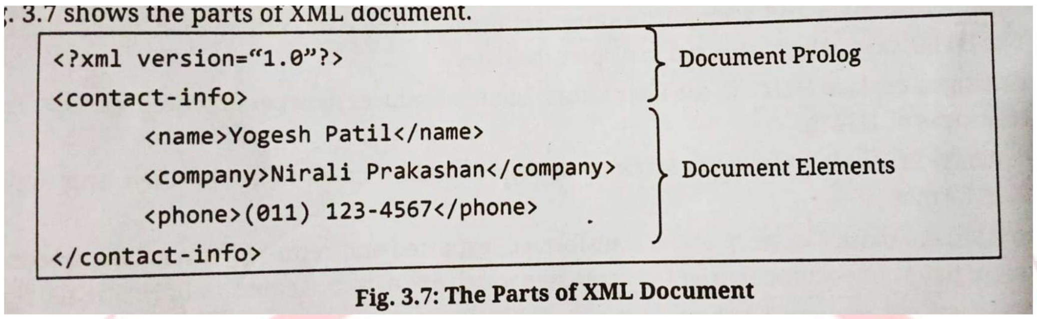
Features of XML

- Markup language which serves the structured data over the internet – can be viewed by users easily and quickly.
- Supports lots of different type of applications.
- It is easy to write programs which process XML.
- It is a simple language – user can use with minimal knowledge.
- Documents are created very quickly.
- One can create and view XML in notepad too.
- It can be used for floating and reloading databases.

Q: Explain parts of XML Documents

XML document is basic unit of XML.

It contains wide variety of data.



. 3.7 shows the parts of XML document.

```
<?xml version="1.0"?>
<contact-info>
        <name>Yogesh Patil</name>
        <company>Nirali Prakashan</company>
        <phone>(011) 123-4567</phone>
</contact-info>
```

Fig. 3.7: The Parts of XML Document

1. **Document prolog system**
    - Document prolog comes at the top of the document before the root element
    - This section contains XML declaration and document type declaration
2. **Document element section**
    - Document elements are building blocks of XML
    - They divide document into hierarchy of sections → each serving a specific purpose.
    - Document can be separated into multiple sections.
    - Elements can be containers with combination of text and other elements.

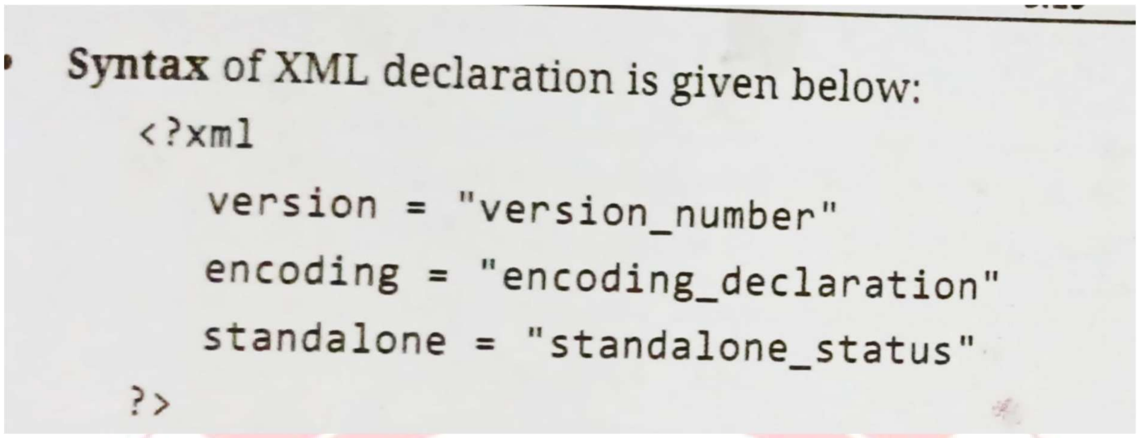
XML document can optionally have an XML Declaration as:

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

Where, version → XML version

Encoding → character encoding used in the document

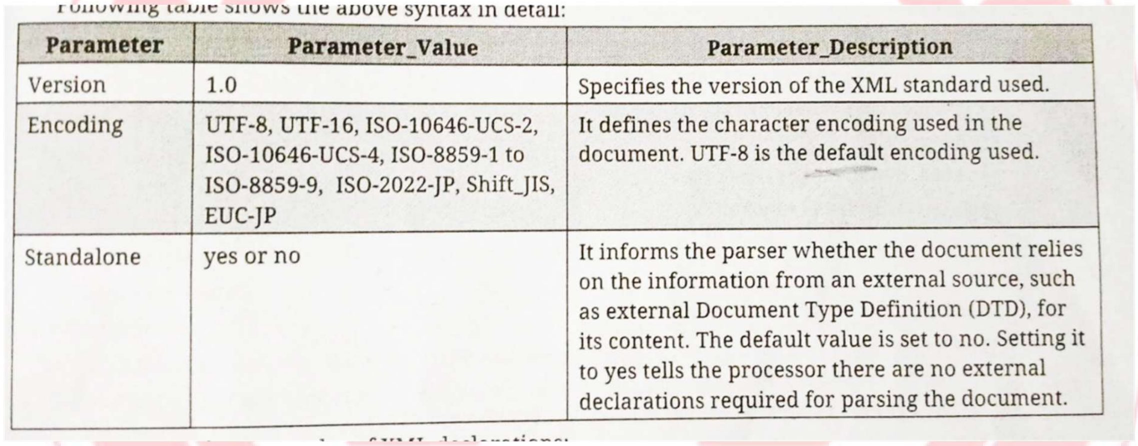
Syntax of XML declaration



- Syntax of XML declaration is given below:

```
<?xml
    version = "version_number"
    encoding = "encoding_declaration"
    standalone = "standalone_status"
?>
```

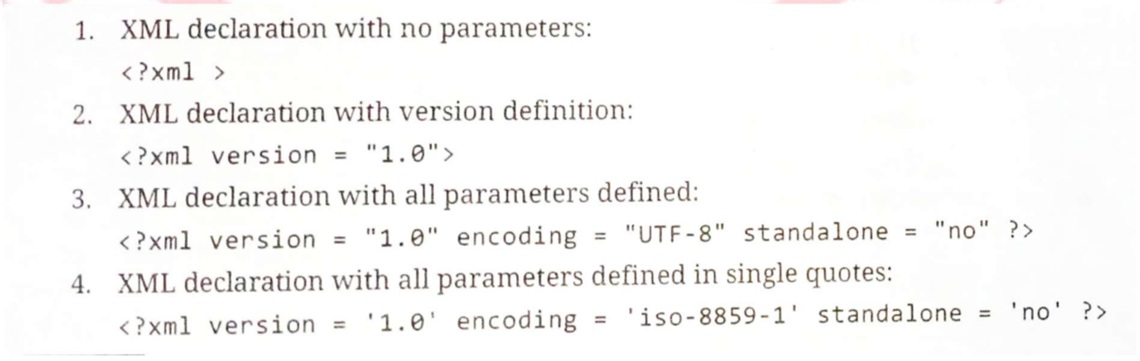
- **Syntax details**



Following table shows the above syntax in detail:

Parameter	Parameter_Value	Parameter_Description
Version	1.0	Specifies the version of the XML standard used.
Encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP	It defines the character encoding used in the document. UTF-8 is the default encoding used.
Standalone	yes or no	It informs the parser whether the document relies on the information from an external source, such as external Document Type Definition (DTD), for its content. The default value is set to no. Setting it to yes tells the processor there are no external declarations required for parsing the document.

- **Examples of XML declarations**



1. XML declaration with no parameters:
   ```
   <?xml >
   ```
2. XML declaration with version definition:
   ```
   <?xml version = "1.0">
   ```
3. XML declaration with all parameters defined:
   ```
   <?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
   ```
4. XML declaration with all parameters defined in single quotes:
   ```
   <?xml version = '1.0' encoding = 'iso-8859-1' standalone = 'no' ?>
   ```

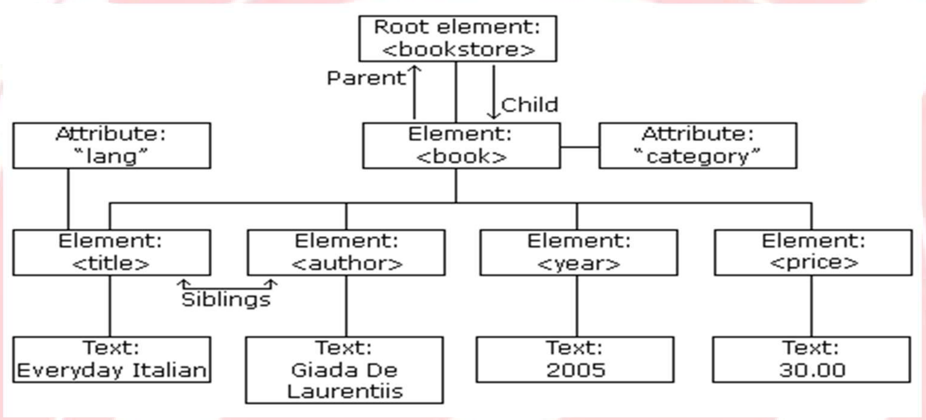
Q: Explain Structure of XML data with an example

- The fundamental construct in an XML document is the element.

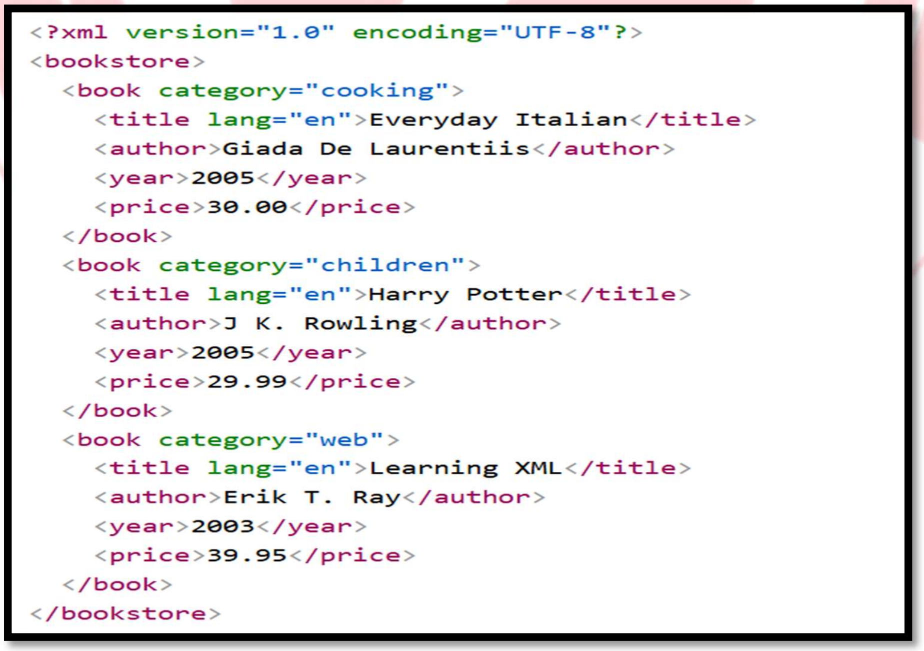
- An element is pair of matching start and end tags
- All the text appears between tags
- XML document must have a single root element
- Root element encompasses all other element in the document

Nested XML

An Example XML Document



- The image above represents books in this XML



```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Q: What are XML Elements, explain with syntax

- XML file is structured by several XML elements
- Also called as XML tags
- XML elements are enclosed in triangular brackets <> such as <element>
- XML elements can have start and end tags
  - <element>……….</element>

    OR
  - In simple cases
    - </element>

Nesting of elements

- XML elements can contain multiple elements as children
- Children element must not overlap

Below are the examples which shows the correct and incorrect usage of nested elements.

Example: This is Wrong

```
<customer>
        <City>
                <Location>
        </City>
                </Location>
</customer>
```

Example: **This is Correct**

```
<customer>
        <City>
                <Location>
                </Location>
        </City>
</customer>
```

Root element

- XML document can have only one root element
- Eg.

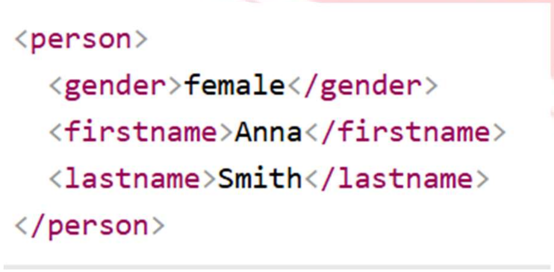
```
<root>
   <x>…….</x>
   <y>…….</y>
</root>
```

Case sensitivity

- XML elements are case sensitive
- Start and end elements need to be in same case
- Eg. <contact_info> is different from <Contact_info>

Q: Explain XML attributes with example

- Attribute specifies single property for the elements
- Uses main/value pair
- XML element can have one or more attributes
- Element can have descriptive attributes → provides additional information about element.
- attribute name = value:-appears inside start tag of the element just after name of the element
- Eg. Generally we use



```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

- Using XML attributes

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

In the first example, gender is an attribute. In the last example, gender is an element. Both examples provide the same information.

Q: How to give comments in XML?

- Message in XML document is easy and simple to understand with user defined tag
- But we can add notes/comments in XML document
- Comments can be added anywhere in the document
- Every comment begins with <!—and ends with -->
- Comments are ignored by XML processor
- Eg.

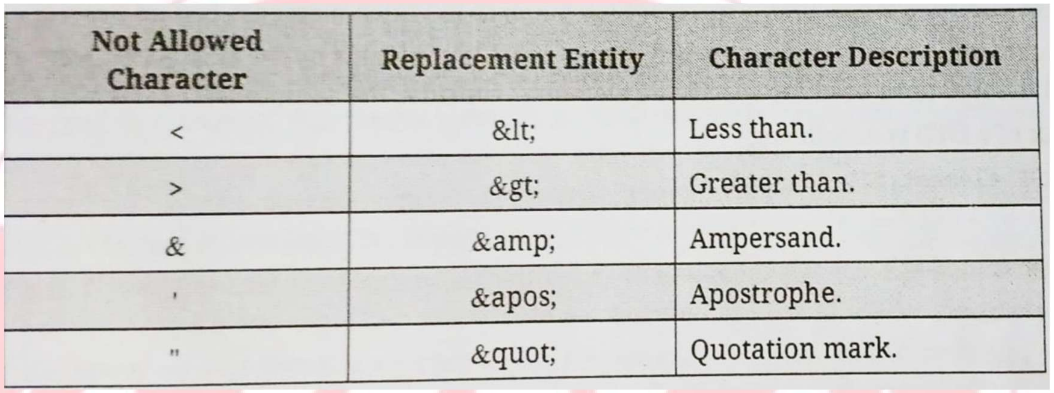
```
<note>
    <!-- This is the title of the note -->
    <title>Reminder</title>
    <body>Don't forget the meeting at 10 AM.</body>
</note>
```

Q: What are Elements of XML references?

- References allow to add or include additional text or markup in an XML document.
- References begin with “&” symbol.
- It’s a reserved character
- References end with “;” symbol.
- There are two types of references

1. Entity references
   Contains a name between start and end delimiters
2. Character references

- Contains references such as **&#65;**
  - # followed by a number (Unicode code of character)
  - 65 refers to “A”



Not Allowed Character	Replacement Entity	Character Description
<	&lt;	Less than.
>	&gt;	Greater than.
&	&amp;	Ampersand.
'	&apos;	Apostrophe.
"	&quot;	Quotation mark.

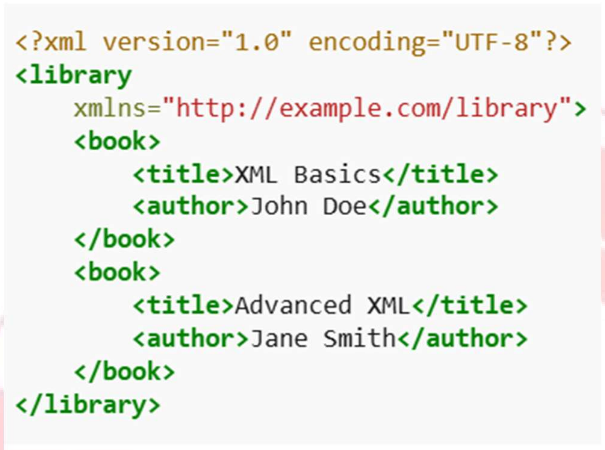
- This will generate an XML error:
  <message>salary < 1000</message>
- To avoid this error, replace the "<" character with an **entity reference**:
  <message>salary &lt; 1000</message>

Q: What is Namespace in XML?

- It is a set of unique names
- It is a mechanism by which element and attribute name can be assigned to a group
- Namespace is identified by URI(Uniform Resource Identifiers)

Namespace Declaration:

- XML documents are designed to be exchanged between applications

- Namespace allows to specify globally unique names
- Namespace is declared using reserved attributes
- <element xmlns:name=”URL”>
- 

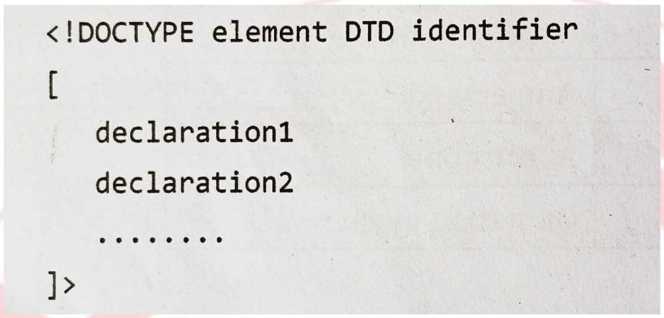
```xml
<?xml version="1.0" encoding="UTF-8"?>
<library
    xmlns="http://example.com/library">
    <book>
        <title>XML Basics</title>
        <author>John Doe</author>
    </book>
    <book>
        <title>Advanced XML</title>
        <author>Jane Smith</author>
    </book>
</library>
```

Q: Explain XML document schema

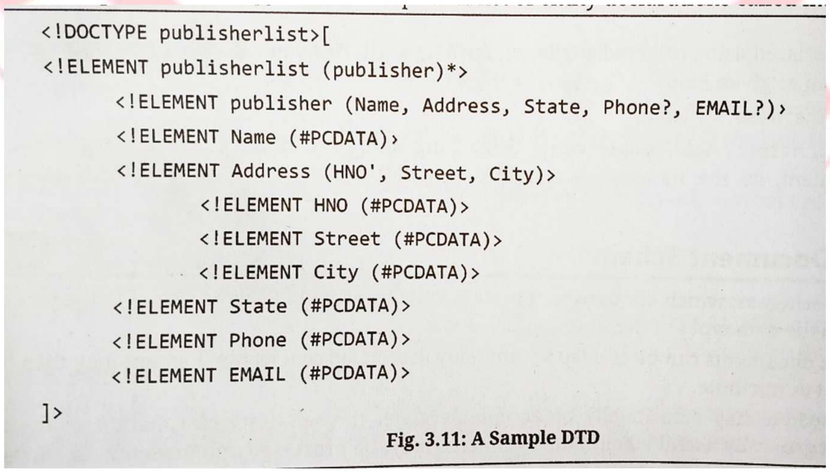
- Databases have schema
- Used to constrain what information can be stored in the database
- Also to constrain data types of stored information.
- XML document can be created without any associated schema

DTDs (Document Type Definition)

- Describes XML language precisely
- Checks vocabulary validity grammatical rules of XML language
- DTD can be specified inside the document or can be kept in a separate document.
- DTD is an optional part of XML
- It constrains appearance of sub elements and attributes within an element

- It is a set of rules which specifies structure in which elements can be nested.
- XML document is considered valid if it confronts to DTD associated with it.
- Basic syntax of DTD:
- 

```
<!DOCTYPE element DTD identifier
[
    declaration1
    declaration2
    ........
]>
```

- DTD starts with <!DOCTYPE delimiter
- An element tells the parser to parse the document from root element
- DTD identifier is for document type definition → may be a path to a file on the system or URL to a file in an internal
- If DTD points to external path → it is called as external sub net
- [] – enclose optional list of entity declaration called internal subnet
- 

```
<!DOCTYPE publisherlist>[
<!ELEMENT publisherlist (publisher)*>
      <!ELEMENT publisher (Name, Address, State, Phone?, EMAIL?)>
      <!ELEMENT Name (#PCDATA)>
      <!ELEMENT Address (HNO', Street, City)>
            <!ELEMENT HNO (#PCDATA)>
            <!ELEMENT Street (#PCDATA)>
            <!ELEMENT City (#PCDATA)>
      <!ELEMENT State (#PCDATA)>
      <!ELEMENT Phone (#PCDATA)>
      <!ELEMENT EMAIL (#PCDATA)>
]>
```

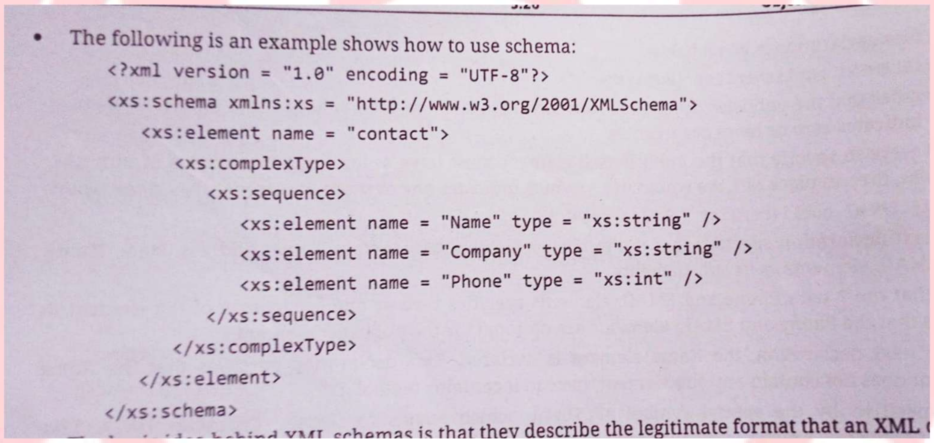
Fig. 3.11: A Sample DTD

- Every DTD is in the form of - <!DOCTYPE root [declaration] >
    - Where root is the name of root element in the document
    - Declaration are the rules of DTD
- Here root element is publisher list
- <!ELEMENT publisherlist (publisher)*>
    - Specifies that publisher list element contains 0 or more occurrence of publisher element
    - * indicates 0 or more occurrence
- If we want to specify publisher list must have at least one occurrence of publisher element then * is replaced by +
    - <!ELEMENT publisherlist (publisher)+>
- Line 3 in fig explains sub elements
- ? with phone and email specifies zero or more occurrences (optional)
- Name element does not contain any sub element
- It contains textual data specified by the symbol **#PCDATA** → **P**arsed **C**haracter **DATA**
- Two more special symbols are allowed
    - **EMPTY** – element has no contents, does nor need end tag, specified as <ELEMENT/>
    - **ANY** – specifies no restrictions on sub elements, can contain any elements even not specified in DTD as its sub elements
- Adress element contains only textual data

-

Q:What is XML schema?

- XML schema commonly known as XML Schema Definition (XSD)
- It defines number of built in types-string integer decimal Boolean
- It allows user defined types(complex type and sequence)
- It is used to describe and validate the structure and content of XML data
- It Defines elements and attributes of data types
- Schema element supports namespaces



- The following is an example shows how to use schema:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
   <xs:element name = "contact">
      <xs:complexType>
         <xs:sequence>
            <xs:element name = "Name" type = "xs:string" />
            <xs:element name = "Company" type = "xs:string" />
            <xs:element name = "Phone" type = "xs:int" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

schemas is that they describe the legitimate format that an XML

-
- XML schema describes the format that XML document can take

Elements

- Building blocks of XML document
- Element can be defined within XSD as:
  - <xs:element name="x" type="y"/>

Definition types

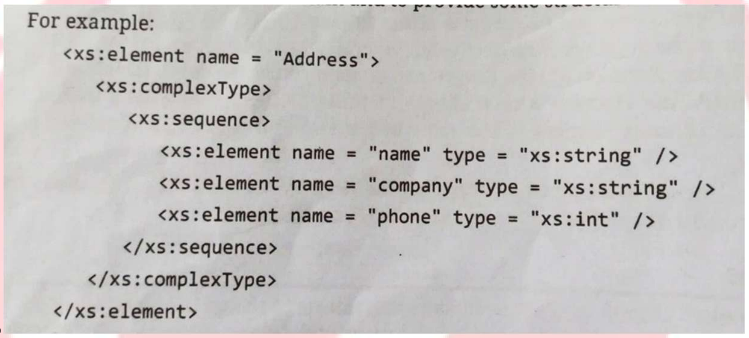
- XML schema elements can be defined as:

1. **Simple type**

- Text elements
- Predefined simple types- xs:integer, xs:boolen, xs:string, xs:date
- E.g. <xs:element name="phone_number" type="xs:int"/>

2. **Complex type**

- It's a container for other elements definitions
- Allow us to specify child elements of an element
- And provide some structure within XML documents
- **E.g.**



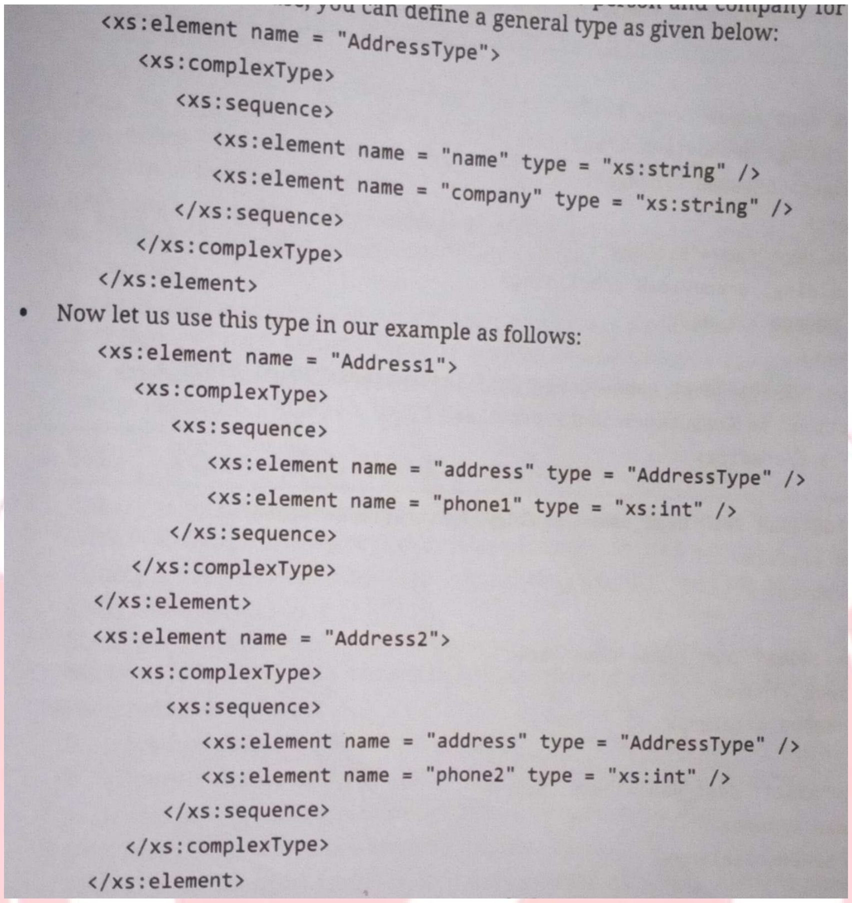
For example:

```
<xs:element name = "Address">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "name" type = "xs:string" />
         <xs:element name = "company" type = "xs:string" />
         <xs:element name = "phone" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

- Address a=element consists of child elements
- It's a simple hierarchy of elements

3. **Global type**

- We can define single type of our document
- Which can be used by other references

- 

you can define a general type as given below:

```
<xs:element name = "AddressType">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "name" type = "xs:string" />
         <xs:element name = "company" type = "xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

- Now let us use this type in our example as follows:

```
<xs:element name = "Address1">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "address" type = "AddressType" />
         <xs:element name = "phone1" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name = "Address2">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "address" type = "AddressType" />
         <xs:element name = "phone2" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Attributes

- Attributes provide extra information within an element
- It has name and type
  - <xs:attribute name=”x” type=”y”/>

Q: Explain the term XPath

- It’s a language for path expression
- Building block of XQuery
- XPath can be used to navigate through elements and attributes
- It’s a major element of XSLT- eXtensible Stylesheet Language Transformation

- XPath addresses part of XML document
- Current version of XPath standard is XPath 2.0
- Result of path expression is a set of nodes
- **Selecting a specific element**
- **XML Document:**

```
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J.K. Rowling</author>
    <price>29.99</price>
  </book>
  <book>
    <title lang="fr">Le Petit Prince</title>
    <author>Antoine de Saint-Exupéry</author>
    <price>15.99</price>
  </book>
</bookstore>
```

XPath Expression:

```
/bookstore/book/title
```

Output:

```
Harry Potter
Le Petit Prince
```

Selecting an element with a specific attribute

XPath Expression:

```
/bookstore/book/title[@lang='en']
```

Output:

```
Harry Potter
```

Selecting a specific node

XPath Expression:

/bookstore/book[2]/author

Output:

Antoine de Saint-Exupéry

Q: Explain the term XQuery (FLOWER Expression)

- It is designed to query XML data
- W3C (World Wide Web Consortium) has developed XQuery
- Used to query records in XML document
- It is independent of format of XML document

Characteristics of XQuery

- It is a functional language used to retrieve XML data from any document or database
- XQuery is similar to SQL in RDBMS → queries XML data.
- Uses XPath to traverse through XML data to fetch records
- XQuery is supported by almost all database systems

Advantages of XQuery

- It can be used to traverse both tabular or hierarchical data (provided they are in XML format)
- It can be used to traverse graphical and treelike structure.
- It can be used to transform XML documents
- It can also be used to traverse and build Web Pages

XQuery (FLOWER Expression)

- XQuery are organized in to “FLOWOR” expressions

F(FOR): Selects a collection of all nodes

L(LET): puts the result in an XQuery variable

w(WHERE): Selects the nodes specified by the condition

O(ORDER BY): Orders the nodes specified as per criteria

R(RETURN): Returns the final result

Example 1: Retrieve Book Titles with Price Greater Than $30

XQuery

```
for $book in doc("books.xml")/bookstore/book

where $book/price > 30

return $book/title
```

Explanation:

- for: Iterates through each <book> element.
- where: Filters books with a price greater than 30.
- return: Outputs the <title> of the filtered books.

Example 2: Retrieve Authors of Books Priced Below $20

XQuery

```
for $book in doc("books.xml")/bookstore/book

where $book/price < 20

return $book/author
```

Explanation:

- Filters books with a price less than 20.
- Returns the <author> of those books.

Example 3: Order Books by Price and Return Titles

XQuery

```
for $book in doc("books.xml")/bookstore/book
order by $book/price ascending
return $book/title
```

Explanation:

- Orders books by their <price> in ascending order.
- Returns the <title> of the books in the sorted order.

Q: Explain Joins in XML

- Joins are specified in XQuery
- Similar to SQL joins
- **Joins in XML** are used to combine data from multiple XML documents or nodes based on a common key or condition.
- This is particularly useful when working with hierarchical data structures in XML.
- Joins in XML are typically implemented using **XQuery** or **XPath**.

Types of Joins in XML

1. **Inner Join**: Combines data where the key matches in both XML documents.
2. **Left Outer Join**: Includes all data from the left XML document and matching data from the right XML document.

3. **Cartesian Join**: Combines every element from one XML document with every element from another.

Example: Inner Join in XQuery

Suppose we have two XML documents:

Document 1: Books.xml

Xml

```
<books>
  <book>
    <id>1</id>
    <title>XML Basics</title>
  </book>
  <book>
    <id>2</id>
    <title>Advanced XML</title>
  </book>
</books>
```

Document 2: Authors.xml

Xml

```
<authors>
  <author>
    <book_id>1</book_id>
    <name>John Doe</name>
  </author>
  <author>
    <book_id>2</book_id>
    <name>Jane Smith</name>
  </author>
</authors>
```

XQuery to Perform Inner Join:

Xquery

```
for $b in doc("Books.xml")//book,
    $a in doc("Authors.xml")//author
where $b/id = $a/book_id
return
<result>
  <book-title>{ $b/title/text() }</book-title>
  <author-name>{ $a/name/text() }</author-name>
</result>
```

Output:

Xml

```
<result>
  <book-title>XML Basics</book-title>
  <author-name>John Doe</author-name>
</result>
<result>
  <book-title>Advanced XML</book-title>
  <author-name>Jane Smith</author-name>
</result>
```

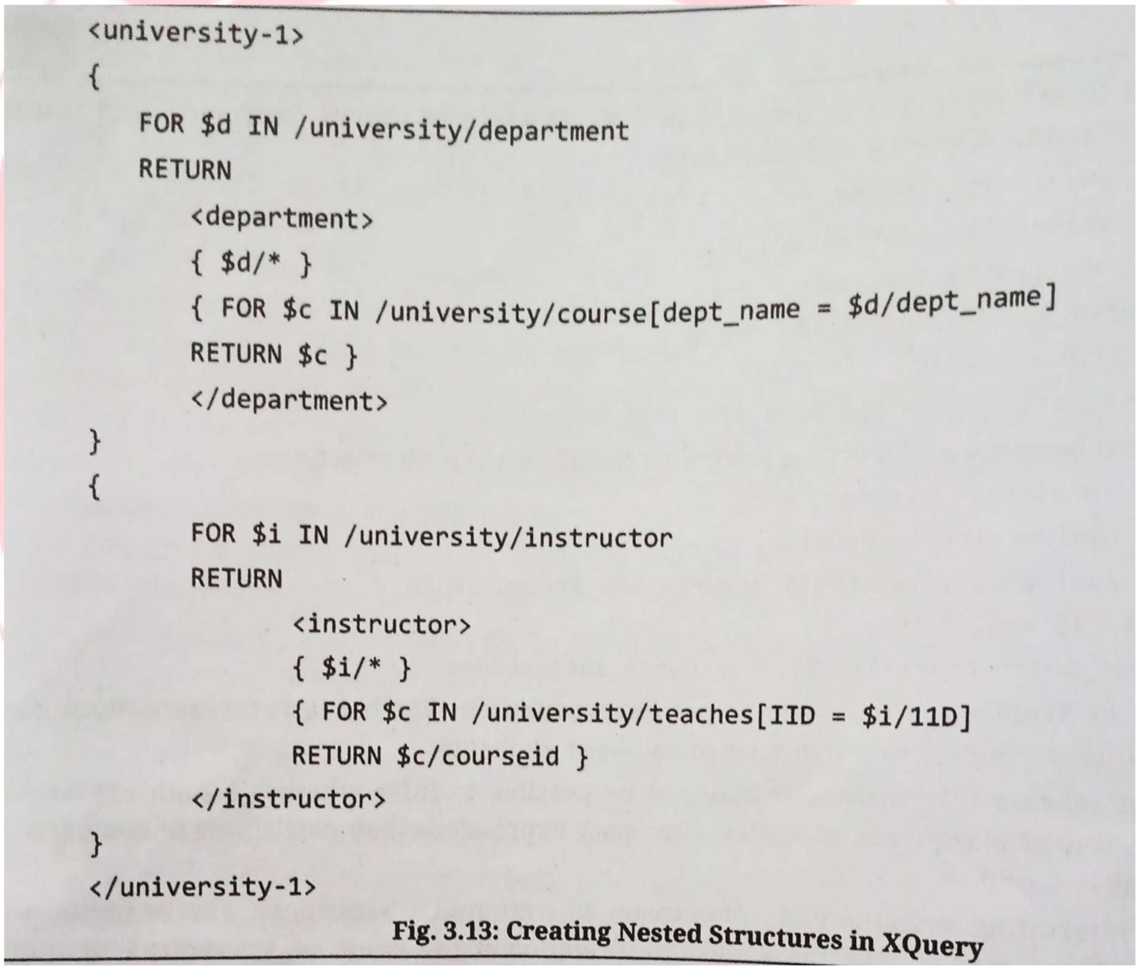
Explanation:

1. The for clause iterates over the <book> elements in Books.xml and <author> elements in Authors.xml.
2. The where clause ensures that only elements with matching id and book_id are joined.
3. The return clause constructs the resulting XML with the combined data.

This approach can be adapted for other types of joins by modifying the logic in the where clause or including unmatched elements as needed.

Q: Nested Queries

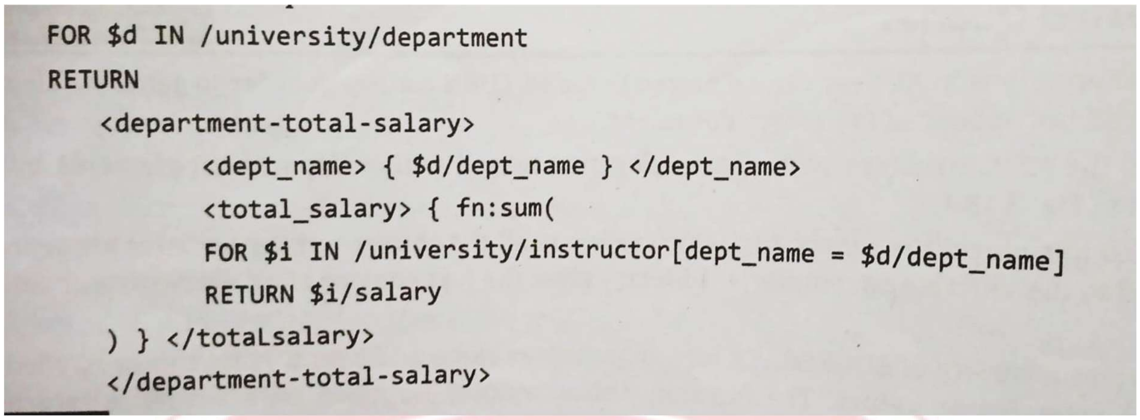
- The FLWOR expression in XQuery can be nested in RETURN clause
- To generate nesting that do not appear in the source document



```
<university-1>
{
    FOR $d IN /university/department
    RETURN
        <department>
        { $d/* }
        { FOR $c IN /university/course[dept_name = $d/dept_name]
        RETURN $c }
        </department>
}
{
        FOR $i IN /university/instructor
        RETURN
                <instructor>
                { $i/* }
                { FOR $c IN /university/teaches[IID = $i/11D]
                RETURN $c/courseid }
        </instructor>
}
</university-1>
```

Fig. 3.13: Creating Nested Structures in XQuery

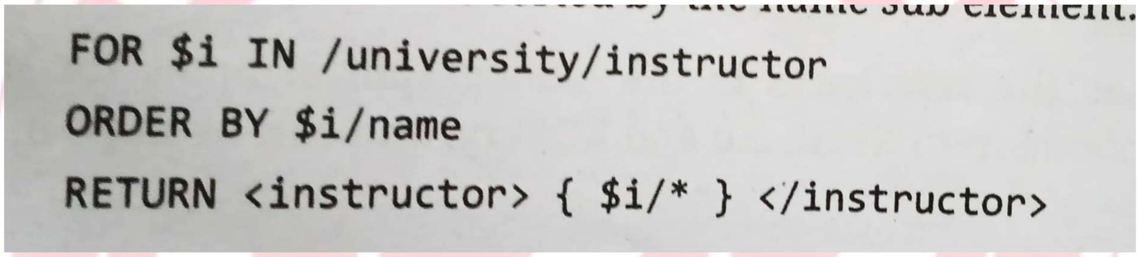
- XQuery does not provide any group by construct
- Aggregate query can be written by using AGGREGATE functions on path or FLWOR expression nested within the written clause.



```
FOR $d IN /university/department
RETURN
    <department-total-salary>
           <dept_name> { $d/dept_name } </dept_name>
           <total_salary> { fn:sum(
           FOR $i IN /university/instructor[dept_name = $d/dept_name]
           RETURN $i/salary
    ) } </totaLsalary>
    </department-total-salary>
```

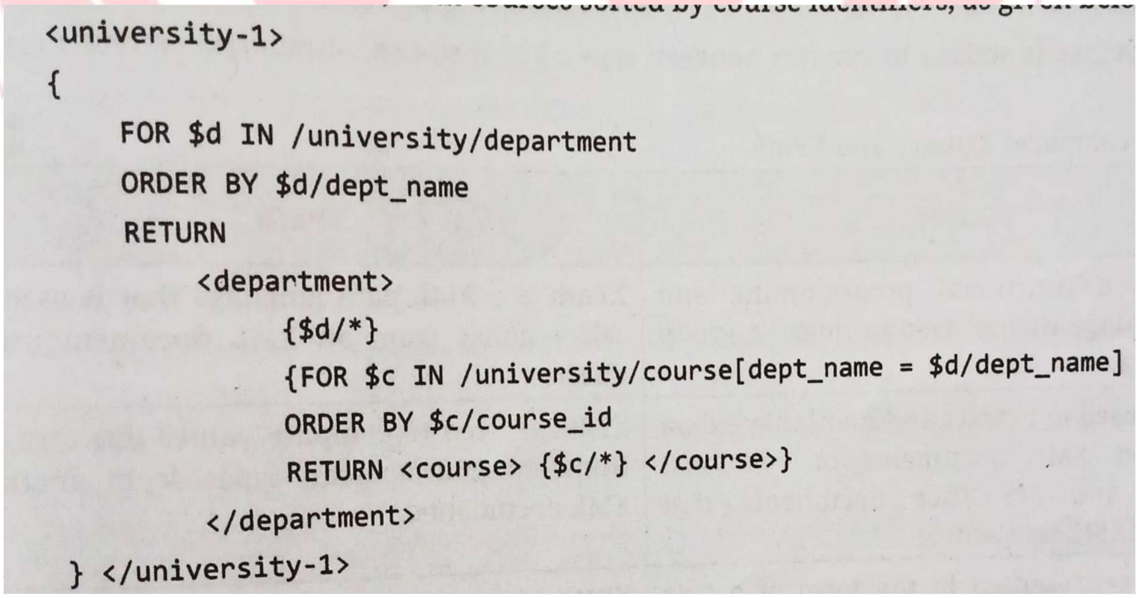
Q: How is Sorting of Results/Functions used in XML

- Results can be sorted in XQuery by using ORDER BY clause



```
FOR $i IN /university/instructor
ORDER BY $i/name
RETURN <instructor> { $i/* } </instructor>
```

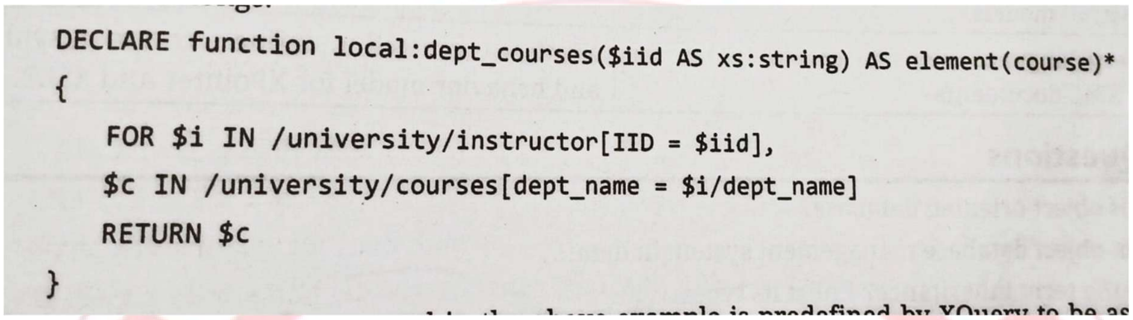
- To sort in descending order,
  we can use ORDER BY $i/name descending
- Sorting can be done at multiple levels of nesting



```
<university-1>
{
     FOR $d IN /university/department
     ORDER BY $d/dept_name
     RETURN
          <department>
                {$d/*}
                {FOR $c IN /university/course[dept_name = $d/dept_name]
                ORDER BY $c/course_id
                RETURN <course> {$c/*} </course>}
          </department>
} </university-1>
```

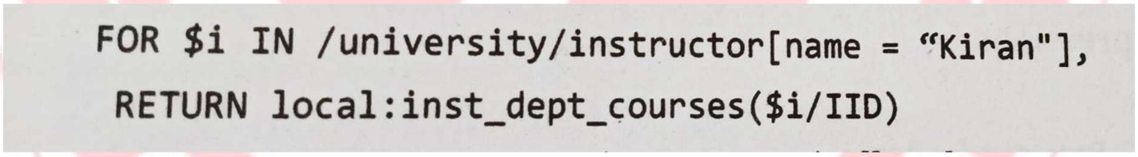
Functions and types

- XQuery provides a variety of built in functions
- Numeric functions, string matching, manipulation function
- It also supports user defined functions



```
DECLARE function local:dept_courses($iid AS xs:string) AS element(course)*
{
    FOR $i IN /university/instructor[IID = $iid],
    $c IN /university/courses[dept_name = $i/dept_name]
    RETURN $c
}
```

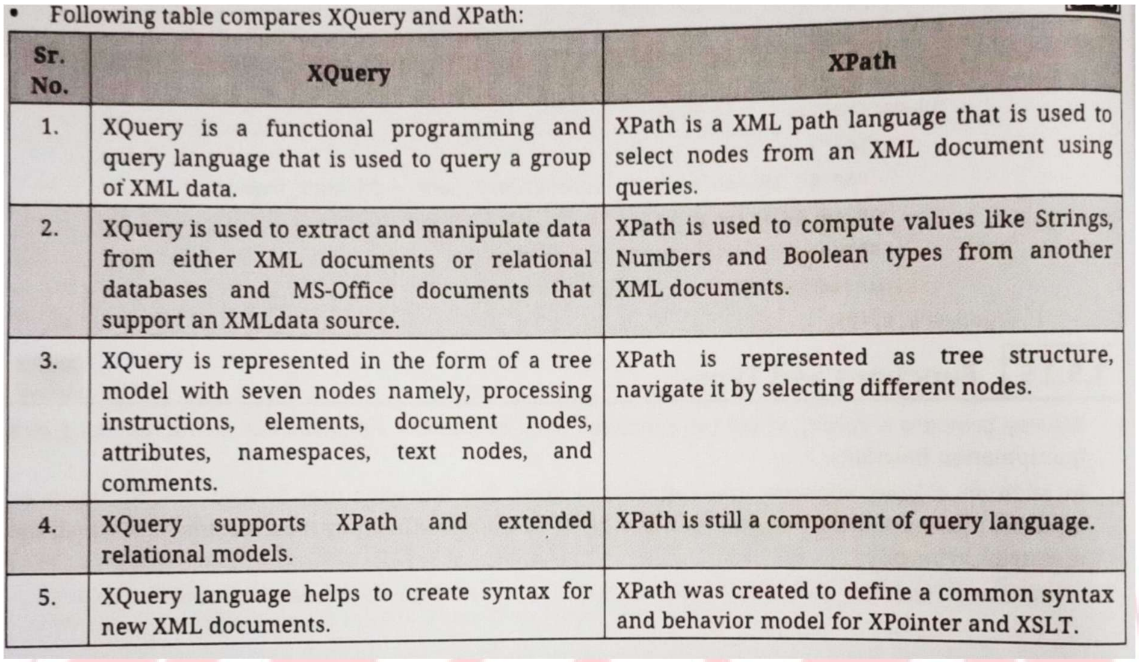
- Here Namespace prefix XS: used is predefined by XQuery → associated with XML schema namespace
- The namespace local: is predefined to be associated with XQuery local function.
- Following query illustrates function invocation → prints department courses for instructor name Kiran



```
FOR $i IN /university/instructor[name = "Kiran"],
 RETURN local:inst_dept_courses($i/IID)
```

- XQuery performs type conversion automatically whenever required
- If numeric value represented by a string is compared to numeric data type, type conversion from string to numeric type is automatically done.
- If element is passed to a function that expects a string value → type conversion to string is done by concatenating all the text values contained within the element

Q: Difference between XQuery and XPath



- Following table compares XQuery and XPath:

Sr. No.	XQuery	XPath
1.	XQuery is a functional programming and query language that is used to query a group of XML data.	XPath is a XML path language that is used to select nodes from an XML document using queries.
2.	XQuery is used to extract and manipulate data from either XML documents or relational databases and MS-Office documents that support an XMLdata source.	XPath is used to compute values like Strings, Numbers and Boolean types from another XML documents.
3.	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.	XPath is represented as tree structure, navigate it by selecting different nodes.
4.	XQuery supports XPath and extended relational models.	XPath is still a component of query language.
5.	XQuery language helps to create syntax for new XML documents.	XPath was created to define a common syntax and behavior model for XPointer and XSLT.

ADDITIONAL NOTES FOR EASY UNDERSTANDING

XML-eXtensible Markup Language

- You can create your own tags
- Used to store and transport data
- XML is self-descriptive
- XML is used to carry data
- XML is NOT used to display data
- We can use self-define tags
- XML is platform and language independent
- It helps in easy communication between two platforms
- E.g. SQL→uses XML to communicate with application
- ORACLE → uses XML for communication with application

Features of XML:

- Separates data from HTML
- Simplifies data sharing
- Simplifies data transport
- Increases data availability
- Simplifies platform change
- XML is written in hierarchical structure
- XML is case sensitive

e.g. college(classes, students)

```
<?xml version “1.0” encoding=””?>      →declaration
<College>        → root
        <Class1>            →child element
                <Name>Amit</Name>
                <RollNo>1</RollNo>
        </Class1>
        <Class2>            →child element
                <Name>Mohan</Name>
                <RollNo>2</RollNo>
        </Class2>
</College>
```

XML DTD:

- Document type definition
- Used to describe xml language precisely
- Used to define structure of a XML document
- Contains list of legal elements
- Used to perform validation
- **DTD syntax:**

- <!DOCTYPE element DTD identifier [declaration1 declaration 2]>
- **Types of DTD:**
- Internal DTD: (PCDATA – data that can be parses)
    - Elements are declared within the xml files
    - Syntax:
    - <!DOCTYPE root-element [element-declaration]>
    - **E.g.**
    - <?xml version="1.0" encoding ,"UTF-8">
    - <!DOCTYPE Address →this is DTD

```
  [<!Element Address(Name, Company, Phone)>
  <!Element Name(#PCDATA)
 <!
]>
<Address>          →root →this is XML
      <Name>---------</Name>
      <Company>-------<Company>
      <Phone>---------</Phone>
</Address>
```

- External DTD:
    - Elements are declared outside xml file
    - Syntax:
    - Add DTD
    - **XML file**
    - <!DOCTYPE Address SYSTEM "Add.dtd">

XML Namespaces

- Used to avoid element name conflict in XML document
- It is a set of unique names
- It is identified by URI(Uniform Resource Identifier)
- Attributes name must start with “xmlnx”
- Syntax:
- <element xmlns:name=”URI”>
- Elements and attribute names belong to URI
- **Conflict:** generally conflict occurs when we try to mix XML documents from different XML applications
- **E**.g. of conflict-
- **1.xml**
  - <class>

    <name>-------</name>

    </class>
- **2.xml**
  - <class>

    <name>-------</name>

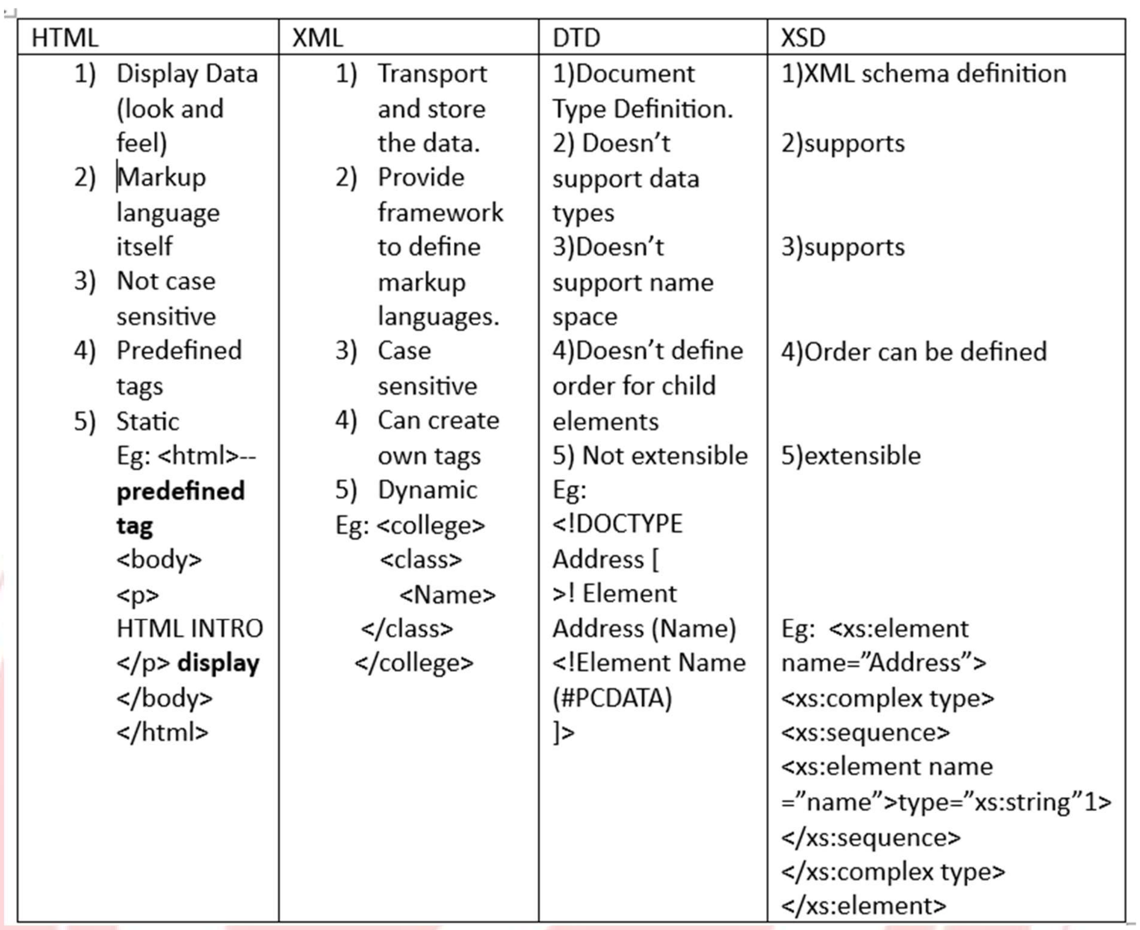
    </class>
- Conflict occurs due to same element name
- **Example of namespace:**
- **1.xml:**
- <?xml version=”1.0” encoding =”UTF-8:?>
  <c1:class xmlns:c1=”class1……”>   c1→prefix
          <c1:name>Aman</c1:name>
  </c1:class>

- **2.xml:**
- <?xml version=”1.0” encoding =”UTF-8:?>
  <c2:class xmlns:c2=”class2……”>   c2→prefix
          <c2:name>Aman</c2:name>
  </c2:class>
- Now there will be no conflict due to namespace

XML Schemas:

- Commonly known as XML schema Definition (XSD)
- It is used to describe and validate the structure and content of XML data
- It is a method of expressing constraints about XML documents
- It is like DTD but provides more control on XML structure
- **Syntax:-**
- <XS:Schema xmlns:xs=”--------“>
- **Definition types**
- Simple Type:
- Used only in context of text
- E.g. xs:Int,xs:String
-     <xs:element name=”Phone” type=”xs:Int/>
- Complex Type:
- It is the container for other element definitions.
- Allows you to specify which child elements an element can contain to provide same structure within your XML document
- Eg : of complex type   (ADD.xsd)
- <?xml version = “1.0”encoding=”UTF-8?>
- <xs:schemea xmlns:xs = “Schemal……”>

- <xs:element name : "Address">
-    <xs: complex Type>
-   <xs:sequence> ----------------------→**child elements should in sequence**
-   <xs:element name ="Name" type="xs:string"1>
- <xs:element name ="phone" type="xs:int"1>
- </xs:sequence>
- </xs:complex type>
- </xs:element>
- </xs:schema>
-
- (Add.xml)
- <?xml version = "1.0 encoding = "UTF-8"?>
- <Address
- xsi:schemalocation:"------Add xsd">---------→path of xms schema def
-       <Name> Aman</Name>
-       <Phone>9810</Phone>
-       <Address>
-
- (if u will type number in name place and alphabets in name place it will show an error)
-
-
-
-
- **DIFFERENCE BETWEEN HTML XML DTD XSD**



HTML	XML	DTD	XSD
1) Display Data (look and feel) 2) Markup language itself 3) Not case sensitive 4) Predefined tags 5) Static Eg: <html>-- **predefined tag** <body> <p> HTML INTRO </p> **display** </body> </html>	1) Transport and store the data. 2) Provide framework to define markup languages. 3) Case sensitive 4) Can create own tags 5) Dynamic Eg: <college> <class> <Name> </class> </college>	1)Document Type Definition. 2) Doesn't support data types 3)Doesn't support name space 4)Doesn't define order for child elements 5) Not extensible Eg: <!DOCTYPE Address [>! Element Address (Name) <!Element Name (#PCDATA)]>	1)XML schema definition 2)supports 3)supports 4)Order can be defined 5)extensible Eg: <xs:element name="Address"> <xs:complex type> <xs:sequence> <xs:element name ="name">type="xs:string"1> </xs:sequence> </xs:complex type> </xs:element>

•