Handout - DB2 and XML

1. What is Xquery?

Xquery is used to query XML data. This may be either a database or a file. It used xpath expressions that can be used to extract elements and attributes from XML documents.

2. What is XML?
   - Extensible Markup Language (XML) is the universal language for data on the Web
   - XML is a technology which allows us to create our own markup language.
   - XML documents are universally accepted as a standard way of representing information in platform and language independent manner.
   - XML is universal standard for information interchange.
   - XML documents can be created in any language and can be used in any language.

3. What is a Processing Instruction in XML?

A Processing Instruction is the information which we would like to give to application. Through a Processing Instruction an application would get idea about how to process the document. A Processing Instruction can appear anywhere and any no. of times in a document.

4. What are namespaces? Why are they important?

A simple element is an XML element that can contain only text.

- Namespaces are a simple and straightforward way to distinguish names used in XML documents, no matter where they come from.
- XML namespaces are used for providing uniquely named elements and attributes in an XML instance
- They allow developers to qualify uniquely the element names and relationships and make these names recognizable, to avoid name collisions on elements that have the same name but are defined in different vocabularies.
- They allow tags from multiple namespaces to be mixed, which is essential if data is coming from multiple sources.

Example: a bookstore may define the <TITLE> tag to mean the title of a book, contained only within the <BOOK> element. A directory of people, however, might define <TITLE> to indicate a person's position, for instance:<TITLE>President</TITLE>. Namespaces help define this distinction clearly.

Note: a) Every namespace has a unique name which is a string. To maintain the uniqueness among namespaces a IRL is most preferred approach, since URLs are unique.

b) Except for no-namespace Schemas, every XML Schema uses at least two namespaces:
1.thetargetnamespace.
2. The XMLSchema namespace (http://w3.org/2001/XMLSchema)

5. What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes. There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

6. What is a Simple Element?

A simple element is an XML element that can contain only text.

- A simple element cannot have attributes
- A simple element cannot contain other elements
- A simple element cannot be empty
- However, the text can be of many different types, and may have various restrictions applied to it

7. How does the XML structure is defined?

XML document will have a structure which has to be defined before we can create the documents and work with them. The structural rules can be defined using many available technologies, but the following are popular way of doing so-

- Document Type Definition (DTD)
- Schema

8. What are differences between DTDs and Schema?

Schema	DTD
Schema document is an XML document i.e., the structure of an XML document is specified by another XML document.	DTDs follow SGML syntax.
Schema supports variety of dataTypes similar to programming language.	In DTD everything is treated as text.
In Schema, It is possible to inherit and create relationship among elements.	This is not possible in DTD without invalidating existing documents.
In Schema, It is possible to group elements and attributes so that they can be treated as single logical unit.	Grouping of elements and attributes is not possible in DTD.
In Schemas, it is possible to specify an upper limit for the number of occurrences of an element	It is not possible to specify an upper limit of an element in DTDs

9. Simple column name passing with XMLEXISTS, XMLQUERY, or XMLTABLE

To simplify using the XMLEXISTS predicate, the XMLQUERY scalar function, or the XMLTABLE table function, you can use a column name as a parameter in the XQuery expression specified by XMLEXISTS, XMLQUERY, or XMLTABLE without specifying the name in the **passing** clause.

You must use the **passing** clause that passes the column name as a parameter, if the parameter name being used is different from the column name being passed.

If a variable is specified explicitly in the **passing** clause and if the name conflicts with any variable referenced by the XQuery expression, precedence will be given to the variable in the **passing** clause. Assuming in the CUSTOMER table contains two XML columns named INFO and CUST, the following example retrieves XML data from INFO column:

SELECT XMLQuery('$CUST/customerinfo/name' PASSING INFO as "CUST") FROM customer

The variable CUST specified in the **passing** clause will be used to substitute the column INFO in the XQuery expression. The column CUST from the CUSTOMER table will not be used.

Examples: XMLQUERY and XMLEXISTS

Note that column names are case-sensitive in these examples, since they are enclosed by double quotes. When not surrounded by double quotes, regular column name rules apply: the name of the column is case-insensitive and stored in upper case.

The following example shows several SELECT statements that return the same sequence of documents from the PURCHASEORDER table. The XML documents are in column PORDER. The first two SELECT statements use the **passing** clause to pass the column name PORDER to the XQuery expression within the XMLQUERY scalar function. The third SELECT uses the PORDER column name as an implicitly passed parameter:

```
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name'  PASSING porder AS "PORDER")
  FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name' PASSING porder AS "porder")
  FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name') FROM purchaseorder
```

The following two examples shows several function calls that use both XMLQUERY and XMLEXISTS. Both examples return the same sequence of documents from the CUSTOMER table.

The following example uses the **passing** clause to explicitly specify the INFO column name as a parameter in the XMLQUERY scalar function and the XMLEXISTS predicate:

```
SELECT XMLQUERY ('$INFO/customerinfo/addr' passing Customer.INFO as "INFO")
FROM Customer
WHERE XMLEXISTS('$INFO//addr[city=$cityName]'
              passing Customer.INFO as "INFO",
              'Aurora' AS "cityName")
```

In the following example, the XMLQUERY function does not use a passing clause and XMLEXISTS **passing** clause does not specify the INFO column. The column name INFO is passed implicitly to XQuery expression in both the XMLQUERY scalar function and the XMLEXISTS predicate:

```
SELECT XMLQUERY ('$INFO/customerinfo/addr')
FROM Customer
WHERE XMLEXISTS('$INFO//addr[city=$cityName]'
              passing 'Aurora' AS "cityName")
```

Examples: XMLTABLE

The following two examples show two INSERT statements that use the XMLTABLE table function. Both examples insert the same data into the table CUSTOMER from the table T1. The table T1 contains an XML column named CUSTLIST. The XMLTABLE function retrieves the data from the column T1.CUSTLIST and returns a table with three columns, Cid, Info, and History. The INSERT statement inserts data from the XMLTABLE function into three columns of the table CUSTOMER.

The following example uses the **passing** clause to explicitly specify the CUSTLIST column name as a parameter in the XMLTABLE table function:

```
INSERT INTO customer SELECT X.* FROM T1,
   XMLTABLE( '$custlist/customers/customerinfo' passing T1.custlist as "custlist"
   COLUMNS
   "Cid" BIGINT PATH '@Cid',
   "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```

In the following example, the XMLTABLE table function does not use a **passing** clause. XMLTABLE uses the column name CUSTLIST from the table T1 as an implicitly passed parameter:

```
INSERT INTO customer SELECT X.* FROM T1,
    XMLTABLE( '$custlist/customers/customerinfo'
    COLUMNS
    "Cid" BIGINT PATH '@Cid',
    "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```