Questions and Answers:

Q1. Is XQL also a popular query language for XML? What's the difference between XQL and XML-QL?

A1: Yes, XQL is also a query language for XML. XQL stands for XML Query Language. It is a natural extension to the XSL pattern language. It builds upon the capabilities XSL provides for identifying class of nodes. Microsoft actively proposes the XQL.

According to my understanding, XQL is specially designed for the XML documents. Compared with XML-QL, XQL has a narrower application. XML-QL uses different query structure from the one that XQL uses. For more information, you could refer to the W3C website for their syntax in detail.

Q2. As I know, there are several query languages over XML have been proposed, such as XQuery, Quilt, Lorel etc. Also they are being applied in all kinds of applications. Could you please explain what is the difference between XQuery and XML-QL? Is XQuery one replacement for other XML query langaues?

A2. XQuery is designed to meet the requirements identified by the W3C XML Working Group and the use cases in [XML Query Use Cases]. XQuery is derived from an XML query language called Quilt which in turn borrowed features from several other languages, including Xpath 1.0, XQL, XML-QL, SQL and OQL. XQuery is designed to be a flexible query language that can query a broad spectrum of XML information sources, including both databases and documents.

XQuery version 1.0 contains XPath 2.0 as a subset. Any expression that is syntactically valid and executes successfully in both XPath 2.0 and XQuery 1.0 will return the same result in both languages.

For more detail information about XQuery, please follow this link to the authority website: http://www.w3.org/TR/xquery/

XQuery is designed to be a potential superset of all current XML based query languages. There are still hundreds of issues being addressed for the XQuery standard. Its future development is subjected to the solutions of these problems and also depends on the XML development.

Q3. We know that data source of a condition in the "**where**" statement must be a URI. What if our data source is not a URI?

A3. The basic picture about URI, URL, and URN of RFC 2396, looks like this:

```
 _________________________________________________________
|                                                         |
|         _________________                               |
|        |  ftp:            |                             |
|        |  gopher:         |                             |
|        |  http:         __|_____________                |
|        |  etc          |  |  urn:         |             |
|        |_______________|__|               |             |
|                URLs    |                  |             |
|                        |__________________|             |
|                               URNs                      |
|_________________________________________________________|
                       URIs
```

URI

Uniform Resource Identifier. The generic set of all names/addresses that are short strings that refer to resources.

URL

Uniform Resource Locator. An informal term (no longer used in technical specifications) associated with popular URI schemes: http, ftp, mailto, etc.

URN

Uniform Resource Name.

Therefore, the data source in the query statement in general, is a URI. If it is not a URI, it might cause problem to complete the query.

Q4. XML-QL allows two different ways to do "**group-by**" operation of SQL. One is nested queries, the other is skolem functions. All queries are stored internally in Normal Form. Normal Form must satisfy the condition "there are no nested queries". Can we say that using skolem function is always better than using nesting queries since nested queries need to be unnested?

A4. It is not necessary to un-nest the nested queries. To construct the tree structure with Skolem functions, the following three rules are needed to be satisfied.

Rule 1: Nested elements must have Skolem functions with arguments such that arg1 $\subseteq$ arg2.

```
CONSTRUCT <tag1 id=F([arg1])>
                <tag2 id=G([arg2])> … </>
          </>
```

Rule 2: An element that has an atomic content must have a Skolem function with \$X $\in$ args.

```
CONSTRUCT …<tag id=F([atgs])> $X </>
```

Rule 3: If a Skolem function occurs in two different places, then the following must be hold

```
G=H, arg1=arg2, tag1=tag2
{CONSTRUCT <tag1 id=G([arg1])> <tag id=F([rags])>…</></>}
{CONSTRUCT <tag2 id=H([arg1])> <tag id=F([rags])>…</></>}
```

Q5: What are the differences between XML-QL and LOREL in terms of efficiency and structure?

A5: Based on OQL, LOREL provides powerful path traversal operators and makes extensive use of type coercion to help yield "intuitive" results for all queries over XML data.

LOREL and XML-QL are the OQL-like and XML-like representatives of Class 2 of expressive query languages for XML, playing the same role as high-level SQL standards and languages (*e.g.*, SQL2) in the relational world. Study indicates that they need certain additions in order to become equivalent in power, in which case it would be possible to translate between them. Currently, a major portion of the queries that they accept can be translated from anyone language to another.

These two have strikingly similarity in the query structures. A document has compared the slight differences of them. No paper has discussed the efficiency differences of the two.

Q6: Can XML-QL combine results from different queries? If so, how? If there are two elements that have the same name but different attributes from two queries, how can XML-QL combine them together?

A6: XML-QL can combine results from different queries. For example, you can do the following

```
<result>
        <articles>
                WHERE <article> <title> $t </><year> $y </>
                          </> IN "www.a.b.c/bib.xml", $y > 1995
                CONSTRUCT <title> $t </>
        </>
        <books>
                WHERE <book> <title> $t </> <year> $y </>
                           </> IN "www.a.b.c/bib.xml", $y >1995
                CONSTRUCT <title> $t </>
        </>
</>
```

But you cannot put two query results in the same structure with the same name. You can either specify another name for the tags or put the previous attribute's name as a prefix.

Q7: What does it mean that XML-QL is relationally complete?
A7: XML-QL is relationally complete because of all the current XML base query languages. A relational complete query language should include five basic relational algebra operations: selection, projection, cross-product, union and difference. XML-QL has joints and can be implemented with known database techniques. Although it is not a standard yet, it is a relational complete query language.

Q8: How will XML data be exchanged between user communities using different but related DTDs?

A8: You can specify different DTDs like this

```
function findDeclaredIncome (
        $Taxpayers:"www.irs.gov/tp.dtd",
        $Employees:"www.employees.org/employeess.dtd"
        :"www.my.site.com/myresult.dtd"    )
        {
                WHERE  ….
                CONSTRUCT ….
}
```

Q9: Please explain how to join elements by value using **ELEMENT_AS**?

A9: Using **ELEMENT_AS** is kind of doing aliasing. You do a query for a certain document and produce the query results. Using **ELEMENT_AS** will give the result to a variable. Later using this variable to construct a new structure by just referencing the variable name.

Q10: If each element directly under the root element has a very deep nested structure and we are asking for the data in the deepest level, will the XML-QL query very complicated?

A10: It is not very complicated. Any part element can contain other nested part elements to an arbitrary depth. To query such a structure, XML-QL provides *regular path expressions*, which can specify element paths of arbitrary depth. For example, the following query produces the name of every part element that contains a brand element equal to ``Ford'', regardless of the nesting level at which r occurs.

```
WHERE <part*> <name>$r</> <brand>Ford</> </> IN "www.a.b.c/bib.xml"
CONSTRUCT <result>$r</>
```

XML-QL's regular path expressions provide the alternation (|), concatenation (.), and Kleene-star (*) operators, similar to those used in regular expressions. A regular path expression is permitted wherever XML permits an element. For example, this query considers any sequence of nested <part> elements and produces any nested <subpart> element or any <piece> element contained in a nested <component> element.

```
WHERE <part+.(subpart|component.piece)>$r</> IN "www.a.b.c/parts.xml"
CONSTRUCT <result> $r</>
```

Q11: Why XPath is not used in XML-QL?
A11: When designing the XML-QL, XPath is considered as another separate feature and was not put as a part of it. Now XPath 2.0 is a subset of XQuery.