Szakdolgozat

Kiss Tamás

Debrecen

2010

Debreceni Egyetem

Informatikai Kar

XML alapú szolgáltatások (Webszolgáltatások, XML adatbázisok)

Témavezető:

Adamkó Attila

egyetemi adjunktus

Készítette:

Kiss Tamás

Programtervező informatikus

Debrecen

2010

Bevezetés

Az informatika több évtizede tartó fejlődése napjainkban is virágkorát éli. Ma már szinte nem találni olyan háztartást, ahol ne lenne jelen a számítógép. És ami manapság együtt jár a számítógépek terjedésével, talán még annál is nagyobb fejlődést mutatva, az a web. A web, amelyen nap mint nap leellenőrizzük elektronikus postaládánkat, friss híreket böngészünk kedvenc hírportálunkon, vásárolunk, szeretett bloggereink újabb bejegyzéseit olvassuk, esetleg második életünket éljük közösségi oldalakon, vagy épp munkánkat végezzük.

Manapság mindenki ismeri az alapvető webbel kapcsolatos fogalmakat, rövidítéseseket. Ha egy átlagos felhasználó azt hallja, hogy *html*, nyomban az internetre asszociál. Azonban, ha a címben szereplő betűszót (XML) látja, nagy valószínűséggel nem tudja hova rakni, pedig akár az is elképzelhető, hogy többet használja, mint az általa ismert *html*-t. Esetleg eszébe juthat a netezőnek, hogy találkozott ő már ezzel a kifejezéssel, mikor épp kedvenc weboldalán folytatta mindennapi szörfözését, ám nem töltött be a kívánt tartalom, helyette egy érthetetlen hibaüzenetet dobott a böngésző, benne e rejtélyes három betűvel. Mindez nem véletlen, ugyanis az XML ott van mindenhol a weben, ha csak közvetetten is, de folyamatosan használjuk, azonban mindez rejtve marad előlünk. Nem nagy túlzás azt kijelenteni, hogy az XML az internetes kommunikáció nyelve.

Másik oldalról nézve, nem sok olyan fejlesztő van, aki ne találkozott volna e betűszóval. Az XML az a technológia, amelynek jelentőségét a szakemberek nem vonják kétségbe, amit minden nagyobb szoftvergyártó cég elismer, és megkérdőjelezhetetlen szabványként van jelen az informatikában.

Az XML, mint webes technológia jött létre, azonban hamar kinőtte önmagát. Felhasználási területeit nehéz lenne mind felsorolni, a benne rejlő lehetőségeket egyre többen ismerik fel, ennek köszönhető, hogy fejlődése továbbra sem állt meg.

Ezen területek közt található az adattárolás. Az XML segítségével saját magunk által definiált struktúrarendszert hozhatunk létre az XML állományokban, amely validálásához és lekérdezéséhez saját eszközöket, erre a célra kifejlesztett szabványokat biztosít számunkra a technológia. Ezen tulajdonságok egy jó, megbízható adatbázis alappilléreiként szolgálhatnak, és így juthatunk el az XML-adatbázisok fogalmához, amely egyik fő témaköre a dolgozatnak, így bővebben is lesz szó még róla a továbbiakban.

Másik fő téma a webszolgáltatások. Megjelenésükkel nagy változásokat hoztak az elosztott rendszerek világába. Ma már kevés olyan informatikai céget találni, ahol ne használnának vagy fejlesztenének webszolgáltatásokat. A nagyobb szoftvergyártók is elismerik, és ez annak köszönhető, hogy a szerver és a kliens közti kommunikációban az XML-nek jut a főszerep, és ez által platformfüggetlenséget biztosítanak.

Dolgozatom célja, hogy az imént pár mondatban bemutatott technológiákról átfogó képet alkossak, a bennük rejlő lehetőségeket feltárjam. Az első részben az XML részletes bemutatása a cél, majd ez adja az alapjait a további témáknak, az XML-adatbázisoknak, és az XML-alapú webszolgáltatásoknak. Mindez egy konkrét webszolgáltatás készítése mentén történik, amely alkalmazás egyfajta üzenőfal szerepét képes ellátni.

Az XML

Az olyan XML-alapú szolgáltatások, mint a XML-adatbázisok, vagy éppen a webszolgáltatások, igen összetett technológiák, ezért mielőtt e témák részletes kitárgyalására kerülne a sor, szükséges ezek alapjait is áttekinteni, azaz jelen esetben az XML-t.

Ahogy arról már a bevezetésben is szó volt, az XML nem azon technológiák közé sorolandó, amelyekkel nap mint nap találkozunk, amelynek ismerete előkövetelmény lenne a számítógépek és az internet hatékony használatához. Úgyis lehetne fogalmazni, hogy az XML amolyan színfalak mögött megbúvó technológia.

Az XML, teljes nevén Extensible Markup Language a World Wide Web Consortium (W3C) terméke. Azaz egy kiterjeszthető leíró nyelv, amelynek célja elsősorban adatok, információk és ezek struktúráinak leírása. Elődje, az SGML (Structured Generalized Markup Language) is hasonló célok mentén alakult ki, azonban az XML már jelentősen túlnőtte elődjét, felhasználási területeit órákig lehetne sorolni.

Az XML rövid története

Az XML nem egy felesleges újabb szabvány, nem egy alternatíva a HTML-re, vagy épp az SGML-re, az XML-re szükség volt, az XML-t az előbb említett nyelvekkel szembeni elégedetlenség szülte.

Nem sokkal az internet megjelenése után született meg a GML (Generalized Markup Language) nevű nyelv az IBM fejlesztésében 1960-ban, majd ennek utódja az SGML, azaz a szabványos általánosított jelölő nyelv, amely 1986-ban vált ISO szabvánnyá. Létrehozására azért volt szükség, mert az egymással kölcsönösen inkompatibilis formátumok nagyon nehézzé tették az elektronikus dokumentumok közzétételét, a közös munkát. Elsősorban szöveges alapú adatbázisok kezeléséhez és közzétételéhez tervezték a nyelvet, amely azonban túl komplexnek bizonyult, és nagyon nehézzé vált SGML értelmezőket és alkalmazásokat implementálni.
Így került sor a HTML (Hypertext Markup Language - hiperszöveges jelölő nyelv) megjelenésére, amely csakúgy, mint az XML, az SGML egy leegyszerűsített részhalmazaként fogható fel. Kezdetben csak egy nagy találmány volt, amely egyszerűségével végre megkönnyítette a munkát, azonban nem sokkal később már mindenki által elismert online

információközlés szabványává vált. Látva elterjedését, egyre többen foglalkoztak a nyelv által nyújtott lehetőségek bővítésén, újabb és újabb funkciók hozzáadásával. Legnagyobb mértékben a webböngészők fejlesztői járultak hozzá ehhez a munkához, azonban épp ez volt az oka annak, hogy a HTML elvesztette egyik legnagyobb előnyét, az egyszerűségét. Ennek következtében lett igény egy újabb nyelv megteremtésére. Az XML fejlesztését, akárcsak a HTML-t a W3C vállalta magára, és hozta létre az SGML és a HTML tapasztalataira alapozva. Az egyszerű felépítés, az átláthatóság az XML-re is jellemző, azonban a HTML-nél általánosabb funkciójú, rugalmasabb, ám szintaktikailag mégis szigorúbb nyelv. Tehát az XML levetkőzte magáról az SGML átláthatatlan bonyolultságát, megőrizte a HTML egyszerűségét, általánosabbá, ezáltal sokrétűbbé tette a nyelvben rejlő lehetőségeket.

XML 10 pontban

Az XML tervezésénél a következő 10 célt, nyelvvel szembeni elvárást fogalmaztak meg a fejlesztők:

1. Az XML célja az adatok strukturálása: Az XML szabályok gyűjteménye, melyek segítségével olyan szöveges formátumokat állíthatunk elő, melyek alkalmasak adatok strukturált leírására.
2. Az XML a HTML-hez hasonló legyen: Akárcsak a HTML, az XML is tagokból, és attribútumokból áll, azonban azoknak nincs előre definiált jelentése, csak az egyes adatcsoportok elválasztására szolgálnak, a tagok és egyéb elemek értelmezését meghagyja az XML-t olvasó alkalmazások számára.
3. Az XML legyen szöveges, de nem olvasásra való: Az XML-t szöveges fájlként tároljuk, ami lehetővé teszi a fejlesztőnek a szabad szemmel való olvasását, amennyiben ez szükséges, de elsősorban nem olvasási szándékkal készülnek az XML állományok.
4. Az XML legyen bőbeszédű: Mivel az XML fájlok szövegesek, így több tárterületet igényelnek, mintha bináris formában tárolnánk őket, azonban így olvashatóak, bőbeszédűek. Ez egy megengedhető hátrány, mert megfelelő tömörítésekkel elérhető a kisebb méret.

5. Az XML legyen technológiák egész családja: Az XML 1.0 definiálja a tag és attribútum fogalmát, azonban technológiák gyűjteménye segíti az XML-el kapcsolatos problémák kezelését. Ilyen például az XPointer, XLink, XQuery, XSL, XSD.
6. Az XML legyen új, de ne teljesen új: Az XML nem előzmények nélküli. Elődje az SGML, illetve részben a HTML is. Ezen nyelvek tapasztalata alapján épül fel az XML.
7. XML-en keresztül vezessen az út a HTML-től az XHTML-ig: AZ XHTML a HTML utódja. Mivel XML alapú, örökli annak szigorú szintaxisát, és szemantikát is társít az elemekhez.
8. Az XML legyen moduláris: Az XML lehetőséget nyújt arra, hogy új dokumentum-formátumokat meglévő formátumok újrahasznosításával hozzunk létre. Hogy elkerüljük az ilyen esetekből származó hibákat, mint például azonos formájú, de eltérő jelentéssel bíró elemek, tag-ok hibás értelmezését, az XML névterekkel és egyéb eszközökkel lehetőséget biztosít ezek kiküszöbölésére.
9. Az XML legyen az RDF és a Szemantikus Web alapja: Az RDF Erőforrás Leíró Nyelv (Resource Description Framework) egy adatleíró nyelv, amellyel erőforrásokról szóló információkat ábrázolhatunk a weben. Az RDF alkalmazásokat és úgynevezett ügynökprogramokat fog egy egységes Szemantikus Webbé formálni. Az RDF-nek, és ezáltal a Szemantikus Webnek egy jó alapra, egy közös nyelvre van szüksége, ez pedig az XML.
10. Az XML legyen licenszmentes, platform-független és jól támogatott: Az XML-t bármilyen platformon lehet használni, és teljesen ingyen. Emellett még nagy közösség által támogatott, ezáltal az XML fejlesztés során sok segítségre lelünk.

Az XML dokumentumok felépítése

Első ránézésre szembetűnik, hogy az XML fájlok felépítése nagyon hasonló a HTML fájloknál megszokottakhoz. Kezdő és ezekhez tartozó záró címkékből áll a fájl, köztük pedig szöveges tartalom lelhető fel. Azonban míg a HTML-nél a jól ismert <html>, <body>, <head>, <p> és egyéb tag-ek használhatóak fel, addig az XML-ben nincsenek határok erre vonatkozóan, tetszőleges címkéket alkalmazhatunk, találhatunk. Tehát az XML-nek

nincsenek előre definiált címkéi, így elsőre jóval engedékenyebbnek, lazábbnak tűnik, mint a HTML, de ez csak a látszat, szintaktikailag sokkal több megszorítás elé állítja a fejlesztőt.

Tekintsük a következő példát, a „*users.xml*” nevű fájlt, amely a fejlesztett rendszer egy adatbázisfájlja, célja a felhasználók tárolása.

```
<?xml version="1.0" encoding="UTF-8"?>
<users xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
      xsi:noNamespaceSchemaLocation="users.xsd">

      <user id="user-0" username="admin" password="admin" rang="admin">
            <full_name>Admin</full_name>
            <e-mail>admin@admin.com</e-mail>
      </user>

      <user id="user-1" username="bela" password="bela" rang="user">
            <full_name>Kovács Béla</full_name>
            <e-mail>kovacs_bela@probamail.hu</e-mail>
      </user>

      <user id="user-2" username="cecil" password="cecil" rang="user">
            <full_name>Nagy Cecil</full_name>
            <e-mail>Nagy_Cecil66@email.com</e-mail>
      </user>

</users>
```

A fenti kód legelső sorában az XML deklaráció található. Ez egy feldolgozási utasítás, amit minden esetben <? és ?> jelek határolnak. Három paraméterrel rendelkezhet. Az első a „version”, amely azt mondja meg a feldolgozónak, hogy a dokumentum az XML melyik verziójával van összhangban. Második az „*encoding*” paraméter, amely a készítésnél használt kódolási sémát adja meg, a harmadik pedig a „*standalone*”, amely azt jelzi, hogy a dokumentum feldolgozására hatással van-e külsőleg meghatározott deklarációkészlet. Ebből a második és harmadik megadása opcionális.

Az XML is, akárcsak számos más jelölőnyelv, három különböző építőelemre támaszkodik. Név szerint említve, tartalmazhat elemeket, attribútumokat és értékeket. Az elemek alapvetően valamilyen információt írnak le, vagy tartalmaznak. Ilyen elemek a fent látható

példában a „*user*”, „*full_name*”, „*e-mail*”, de még a „*users*” is. Minden elemnek kötelező kezdő illetve záró címkével rendelkezni, amennyiben valamelyik hiányzik a kettő közül, az hibát eredményez, az XML előbb említett szigorú szintaxisa nem engedi meg az effajta lazaságot. Szintén egy fontos megszorítás az is, hogy minden XML állományban szerepelnie kell egy gyökérelemnek, ami magában foglalja az összes többi elemet, ez a *users.xml* fájl esetében a „*users*” elem. Lehetőség van még úgynevezett üres elemek megadására, amelyet a következőképpen jelölhetünk: pl. <email />.

Az attribútumok mindig az elemek nyitó címkéjében foglalnak helyet, mindig rendelkeznek névvel és hozzá tartozó értékkel. A HTML-lel ellentétben, az XML megköveteli az attribútumértékek idézőjelekkel való elhatárolását. Az attribútumok az elemek fontosabb tulajdonságainak leírására szolgálnak. A felhasználókat tároló fájlban ilyen tulajdonságok a userek esetében az „*id*”, „*username*” a „*password*” és a „*rang*”. A felhasználónév és a jelszó a bejelentkezéshez elengedhetetlen adatok, mondhatni a legfontosabb tulajdonságai a felhasználóknak, míg a rang attribútum a felhasználó jogait határozza meg. Érdekes probléma, hogy mikor használjunk elemeket, illetve mikor használjuk attribútumot. Természetesen legtöbb esetben egyértelmű a választás, azonban előfordulnak olyan helyzetek, mikor nehezebb a döntés. Fontos szempont az átláthatóság is, de a nyelv által nyújtott lehetőségek ismeretei segítenek leginkább ilyen esetekben.

Lehetne még mélyebbre ásni az XML dokumentumokban, a felépítésük részletesebb vizsgálata érdekében, de most csak az alapok átvétele volt a fontos, ami megtörtént, így most érdemes lehet pár pontba szedni az XML főbb szabályait, amik a nyelv precízségének alapjait adják:

1. A címkék neveiben különbséget teszünk a kis- és nagybetűk között.
2. Minden nyitó címkéhez tartozik egy záró címke, kivéve az üres elemeket.
3. Egymásba ágyazás esetében a címkék nem lapolódhatnak át.
4. Az attribútumok értékeit minden esetben idézőjelek határolják.
5. Mindig van egy gyökérelem.

Ez az öt fő szabály kiegészülve még néhány szintén egyszerű megszorítással alkotják az XML jól formázottságának feltételeit, amit minden XML dokumentumnak teljesítenie kell. Ezt felfoghatjuk úgy is, hogy ez az XML „helyességének” első lépcsőfoka, ugyanis van egy

második is. Ez az XML érvényessége, ami a következő fejezet témájához, a sémákhoz köthető.

Adatok meghatározása sémák segítségével

Ahogy arról már korábban szó esett, az XML adatok, információk, és ezek struktúráinak leírására szolgál. A sémák felelnek az információk struktúráinak leírásáért. Amikor létrehozunk egy XML dokumentumot egy meglévő séma alapján, akkor nem csak az XML nyelvet használjuk ennek megírásához, hanem egy másik jelölőnyelvet, amit a séma határoz meg. Ez az XML nyelv valódi célja, valódi erőssége, hogy segítségével saját jelölőnyelvek ezreit hozhatjuk létre. Egy saját sémával tulajdonképpen azt tudjuk meghatározni, hogy milyen címkéket és attribútumokat használhatunk az XML dokumentumok létrehozása során, illetve ezek milyen viszonyban vannak egymással, tehát a dokumentum struktúráját hozzuk létre. Több nyelv is rendelkezésünkre áll sémák leírására, ilyen nyelv például a DTD, az XML Schema, vagy a Relax NG.

XML Schema

Bár általában a DTD-vel szokás a sémák tárgyalását kezdeni, én most mégis az XML Schema-val kezdeném a sort, mivel a fejlesztett projectben is e technológia segítségével létrehozott sémák vesznek részt az XML adatbázisfájlok validálásában.

Az XML Schema-t is, akárcsak a legtöbb XML technológiát, a W3C hozta létre azzal a céllal, hogy a DTD-nél fejlettebb sémalétrehozó technológiát alkossanak. Ez lett az XML Schema Definition Language, vagy röviden XSD. Az XML Schema segítségével megírt sémák, maguk is XML dokumentumok, így jól alkalmazkodnak az XML jelölőnyelv igényeihez.

A következő kódrészlet a „*users.xml*” felhasználók tárolásáért felelős fájl sémaleírásának, azaz a „*users.xsd*” fájlnak egy részlete.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="users">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="user" type="userType" minOccurs="0"
maxOccurs="unbounded"/>
```

```
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
…
…
…
</xsd:schema>
```

Nyitó címkék, záró címkék, ahogy azt már megszoktuk, valamint a legelső sorban egy XML deklaráció, így beigazolódott, hogy az XSD fájlok valóban XML dokumentumok. Ezután, mivel az XSD nyelv eszközeit akarjuk használni, szükséges egy kötelező névtér-deklaráció, ezt láthatjuk a második sorban. A névterekhez általában tartozik egy előtag, azaz egy prefix, aminek megválasztása természetesen tetszőleges, de az XML Schema esetében az „*xsd*” prefix a megszokott, ha megadtuk az előtagot, akkor ezzel kell majd kezdenünk, ezzel kell minősítenünk a nyelv elemeinek neveit. Az *xmlns:xsd*-t követő hivatkozás egy W3C által meghatározott szabványos URI. Tehát az XSD névterét, és ezzel eszközeit deklaráló *xsd:schema* elem az XSD fájlok kötelező gyökéreleme.

Ezen a főelemen belül található meg a konkrét sémaleírás, aminek a lelkét a típusok jelentik. Az XSD-ben alapvetően két főtípust különbözetünk meg: az egyszerű és az összetett típust. Az egyszerű típus alatt gyakorlatilag az elemi típusokat értjük, vagyis ide tartoznak a numerikus, vagy szöveges adatok, listák, dátumformátumok, stb. Az összetett típusok pedig a különböző egyszerű típusok kombinálásával érhetők el.

Az egyszerű adattípusok elemek és attribútumok definiálására egyaránt használatosak. Az elemeket: <xsd:element>, az attribútumokat <xsd:attribute> elemekkel definiálhatjuk. Ezek különféle attribútumokkal rendelkezhetnek, azonban ezek közül a két legfontosabb, a „*name*” és a „*type*”. A „*name*” nyilvánvalóan az elem illetve attribútum nevének, míg a „*type*” a típusának megadására szolgál. Az XSD nyelv számos egyszerű adattípust különböztet meg, például a „*xsd:integer*”, ami segítségével egész számokat modellezhetünk, vagy az „*xsd:string*”, ami karakterláncok definiálására szolgál, vagy az „*xsd:boolean*”, ami logikai típust reprezentál, és ezen kívül még sok egyéb adattípus áll rendelkezésünkre, amelyekről a W3C XSD-vel foglalkozó hivatalos oldalán bőven tájékozódhatunk. A következő rövid kódrészletben egy „*full_name*” nevű, „*xsd:string*”, azaz karakterlánc típusú elem, illetve egy „*id*” nevű, „*xsd:ID*” típusú attribútum XSD-beli definiálása látható.

```
<xsd:element name="full_name" type="xsd:string"/>
…
<xsd:attribute name="id" type="xsd:ID"/>
```

A legtöbb egyszerű típusú elem, illetve attribútum definiálása a fenti példában látható egyszerűséggel megtehető. Még szintén az egyszerű típusokon belül megkülönbeztetjük az úgynevezett egyéni típust. Az XSD egyik legvonzóbb tulajdonsága, hogy saját típusokat hozhatunk létre benne, ami valójában az alaptípusok célnak megfelelő finomítása. Az egyéni típusokat mindig <xsd:simpleType> címkével jelöljük. A legtöbb esetben egyes egyszerű típusok képességeinek korlátozásáról van szó, mint ahogy a következő példában s látható:

```
    <xsd:simpleType name="usernameType">
        <xsd:restriction base="xsd:string">
            <xsd:minLength value="4"/>
            <xsd:maxLength value="12"/>
        </xsd:restriction>
    </xsd:simpleType>
```

Tehát egy olyan „usernameType” nevű egyéni típus áll rendelkezésünkre, mely alapvetően sztring, de minimum 4, maximum 12 karakter hosszú lehet. Az <xsd:restriction> elemmel jelezzük, hogy a következőkben egy típus korlátozásáról lesz szó, és a „base” attribútum értékében tudjuk megadni a korlátozni kívánt típust. Természetesen nem csak karakterláncokra vonatkozóan adhatunk meg korlátozásokat, ugyanilyen módon lehetőségünk van számos más típus testreszabására is.

Az egyéni típus segítségével létrehozhatunk felsorolásos típust is:

```
    <xsd:simpleType name="rangType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="admin"/>
            <xsd:enumeration value="user"/>
        </xsd:restriction>
    </xsd:simpleType>
```

Ez mindössze annyit ad meg, hogy a „*rangType*” típus karakterláncot jelöl, amelynek értéke vagy „*admin*” vagy „*user*” lehet, más értéket nem vehet fel. Lehetőségünk van úgynevezett minta alapú típusok létrehozására is:

```
    <xsd:simpleType name="emailType">
```

```
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="(\S)+@(\S)+\.(\S)+"/>
        </xsd:restriction>
    </xsd:simpleType>
```

Az <xsd:pattern> elem leírásával lehet definiálni minta alapú típust, a „*value*” attribútumában pedig egy szabályos kifejezést, egy regexp-et kell megadni, amire illeszkednie kell az „*emailType*” típusú elemnek, vagy attribútumnak. A regexp részletesebb tárgyalásába most nem mennék bele, mert nem lényeges a továbbiakban, amit tudni kell erről, hogy ez a regexp azt garantálja, hogy a felhasználók a regisztrálásnál egy valóban e-mail cím formájú adatot adjanak meg.

Az összetett típusok az előbb tárgyalt egyszerű típusoknál összetettebbek, ahogy az a nevéből is kitűnik, azonban az alapjait az egyszerű típusok jelentik, azokra támaszkodva épülnek fel. Minden esetben az <xsd:complexType> elem segítségével hozunk létre összetett típusokat. Alapvetően négy nagy csoportba lehet tenni a különféle összetett típusokat, vannak üres elemek, csak elemeket tartalmazó elemek, kevert tartalmú elemek, sorozatok és választási listák. A következő „*users.xsd*” sémaleírásból kiragadott kódrészlet is egy komplextípust mutat be:

```
<xsd:complexType name="userType">
        <xsd:sequence>
            <xsd:element name="full_name" type="xsd:string"/>
            <xsd:element name="e-mail" type="emailType"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="username" type="usernameType"/>
        <xsd:attribute name="password" type="passwordType"/>
        <xsd:attribute name="rang" type="rangType"/>
</xsd:complexType>
```

Egy „*userType*” nevű komplex típus definiálása látható, amely egy kevert tartalmú elem struktúrájának leírása. Két fajta gyermekelemet tartalmaz, <full_name>, illetve <e-mail> elemet, illetve négy attribútumot. A két gyermekelem egy <xsd:sequence> nevű elembe van beágyazva, ez az elem egy sorozatot hoz létre a benne foglalt elemekből, azaz azoknak a fent leírt sorrendben kell következnie a sémához tartozó XML fájlban. Ha nincs előfordulások számára vonatkozó megszorítás, akkor pontosan egyszer kell, hogy szerepeljenek a

gyermekelemek a szülőelemben. Azonban ez nem mindig ideális a számunkra, tökéletes példa erre a felhasználókat tároló „*users.xml*” fájl esete. E dokumentum célja, hogy az újabb és újabb beregisztrált felhasználókat eltárolja, azaz <user> elemek sorozatára van szükségünk. Ennek leírása is megtalálható az XSD fájlban:

```
<xsd:element name="users">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="user" type="userType" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

A felhasználók tárolásért felelős fájlban a gyökérelem a <users> elem, ennek definiálását tartalmazza a fenti kódrészlet. Mint látható, csak „*user*” nevű gyermekelemet tartalmazhat, de abból minimum nullát, a maximum előfordulása pedig nincs meghatározva, akármennyi lehet belőle, e megszorításokat a „*minOccurs*” és „*maxOccurs*” attribútumok segítségével lehetett megtenni.

Az előzőekben tehát sikerült elkészíteni a felhasználókat tároló XML fájl XSD nyelvben megírt sémáját. Definiálva van, hogy egy <users> elem tárolja az <user>-ek sorozatát. A <user> elem akárcsak szülője, maga is komplex típus, méghozzá „userType” típusú, amely pontosan egy <full_name>, illetve egy <e-mail> elemet tartalmaz, valamint „*id*”, „*username*”, „*password*” és „*rang*” attribútumokat. Amely elemeknek, illetve attribútumoknak meg van írva a saját típusuk, az egyszerű sztringtől kezdve a „usernameType” nevű egyéni típusig.

Azonban arról még nem esett szó, hogy mindez, hogy lesz hozzárendelve az XML fájlhoz. A választ az XML fájlokban leljük, ugyanis azokat kell sémához kötni, nem pedig fordítva.

```
<?xml version="1.0" encoding="UTF-8"?>
<users xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
      xsi:noNamespaceSchemaLocation="users.xsd">
…
</users>
```

A sémára való hivatkozást az XML fájlban, méghozzá annak gyökérelemében kell megtenni. Ahhoz, hogy egy XML dokumentum sémáját tartalmazó XSD fájl nevét és helyét

megadhassuk, egy speciális attribútumot kell használnunk, amelynek értéke tartalmazza a sémát leíró fájl pontos helyét és nevét. Két ilyen attribútum is létezik, méghozzá a „*schemaLocation*”, illetve a „*noNamespaceSchemaLocation*” nevű attribútumok. A kettő közti különbség az, hogy míg az előbbiben a séma helyét és a hozzá tartozó névteret tudjuk megadni, addig az utóbbi olyan névtér elérési helyét adja meg, amelyhez nem tartozik névtér, tehát nem szükséges az ez által definiált elemekhez prefixet hozzácsatolni. De ahhoz, hogy bármelyiket is használhassuk, előtte explicit módon meg kell adnunk az ezeket tartalmazó névteret, azaz „*http://www.w3.org/2001/XMLSchema-instance*” névteret.

Document Type Definition

A Document Type Definition, röviden DTD, az XML dokumentumok struktúrájának leírására eredetileg alkalmazott technológia. Ez a legrégebbi, ami azt illeti, magánál az XML-nél is öregebb, még az SGML számára hozták létre. Bár ma már számos nála fejlettebb sémaleírásra használt eszköz létezik, éppen idős korának köszönheti, hogy manapság is sokan használják, széleskörű támogatottságot élvez. Bár maga a DTD nyelvezete nem igazán bonyolult, mondhatni tömörségre törekszik, mégis kicsit idegen tőlünk, főleg azok után, ha az ember megismerkedik a fejlettebb technológiákkal, mint például az előző fejezetben bemutatott XML Schema-val. A DTD legnagyobb hátránya talán abban rejlik, hogy nem nevezhető típusos nyelvnek, gyakorlatilag minden elem tartalma csak szöveg lehet, nem adhatjuk meg neki, hogy most épp egész számról van-e szó, vagy karakterláncról, esetleg logikai értékről, a DTD sajnos nem ennyire kifinomult nyelv. A fejlesztett projekthez én magam is azért választottam az XML Schema-t, mert ez jóval közelebb áll az XML-hez, számomra átláthatóbb, logikusabb felépítéssel rendelkezik, valamint típusai segítségével, sokkal részletesebb, konkrétabb struktúrát, sémát lehet létrehozni. Azonban nem lehet úgy XML-ről beszélni, hogy ne beszélnénk vele párhuzamosan a DTD-ről is. Kora és ezzel széleskörű támogatottsága miatt még ma is rengeteg olyan rendszer van, ahol az XML állomány felépítését DTD állomány, állományok írják le.

A DTD, akárcsak az XSD, a dokumentumok logikai szerkezetének alapvető részleteit írja le, azaz a jelölőnyelvben használható elemeket, attribútumaikat, és ezek egymáshoz való viszonyát.

Ha DTD-t használunk egy XML fájl helyességének ellenőrzésére, akkor ezt kötelező jelezni az XML állományban. Erre szolgál a dokumentumtípus-deklaráció. Itt két lehetőség közül

választhatunk, megadhatunk külső DTD fájlt, vagy a dokumentumtípus deklaráción belül elhelyezhetjük a dokumentum típus meghatározást, ezt nevezzük belső DTD-nek. Illetve még van egy harmadik eset, mikor külső DTD-re támaszkodunk, ugyanakkor egy belső DTD is jelen van, kiegészítve, vagy akár felülírva a külsőt.

```
<!-  Hivatkozás külső DTD-re -->
<?xml version="1.0" encoding="UTF-8" standalone=”no”?>
<!DOCTYPE  users SYSTEM "users.dtd”>
<users …>
…
</users>
```

A dokumentumtípus-deklarációnak az XML deklaráció után, de még az első elem előtt kell szerepelnie a dokumentumban. A „*DOCTYPE*” kulcsszó után kötelező megadni az XML fájl gyökéreleménének nevét. Külső DTD esetén ezt követi a „*SYSTEM*” szó, majd a dokumentum elérési helye, a DTD URI-ja.

```
<!-  Belső DTD használata -->
<?xml version="1.0" encoding="UTF-8" standalone=”yes”?>
<!DOCTYPE  users [
<!ELEMENT user …
…
]>
<users …>
…
</users>
```

Belső DTD esetén is meg kell adnunk a gyökérelem nevét, majd ezt követi szögletes zárójelek között a konkrét DTD, az elemek, illetve attribútumok leírása.

```
<!-  Belső és külső DTD együttes használata -->
<?xml version="1.0" encoding="UTF-8" standalone=”no”?>
<!DOCTYPE  users SYSTEM "users.dtd”> [
<!ELEMENT user …
…
]>
<users …>
…
</users>
```

A harmadik esetben először jeleznünk kell a külső DTD helyét, majd ezt követően megadni a belső DTD-t. A belső DTD elemei minden esetben felülbírálják a külső DTD azonos nevű elemeinek deklarációját.

Az elemek és attribútumok deklarációjának formája független attól, hogy külső vagy belső DTD-ben tárolódnak. Az elemek leírásának általános alakja a következő:

```
<!ELEMENT ElemNév Típus>
```

Az „*ElemNév*” értelemszerűen a sémát használó fájlban egy használható címke nevét fogja jelölni, a típus pedig ennek típusát. Az előzőekben azt mondtam, hogy a DTD legnagyobb hátránya, hogy nem nagyon nevezhető típusosnak, most mégis láthatjuk, hogy rendelkezik típusokkal. Azonban az elemek esetében mindössze négy különféle típust különböztet meg a nyelv:

- Üres elem – EMPTY
- Csak gyermekemelet tartalmazó elem – ELEMENT ONLY
- Kevert tartalmú elem - MIXED
- Tetszőleges típusú elem – ANY

```
<!-   Üres elem deklarálása -->
<!ELEMENT data EMPTY>

<!-   Csak gyermekelemeket tartalmazó elem deklarálása -->
<!ELEMENT user (full_name, e-mail+, (school | workplace), hobbies*)>

<!-   Kevert tartalmú elem deklarálása -->
<!ELEMENT school (#PCDATA | Grammar_school | College | University)*>

<!-   Tetszőleges típusú elem deklarálása -->
<!ELEMENT any_data ANY>
```

Az üres tartalmú elemnek is van értelme természetesen. Bár gyermekelemi, vagy szöveges tartalma nem lehet, azonban attribútumok tárolására tökéletes. Deklarálásában az EMPTY szónak jut a kulcsszerep.

A csak gyermekemelet tartalmazó elemekben, ahogy neve is jelzi, csak további elemekre lelhetünk, önálló szöveges tartalom nem fordulhat elő. Itt rögtön az elem neve után

következik a lehetséges gyermekelemeinek felsorolása, ez a tartalmi modell. A tartalmi modell speciális elemdeklarációs szimbólumokból, és elemnevekből áll össze. Lehetőségünk van sorozatok, vagy választási csoportok létrehozására, emellett megmondhatjuk, hogy az adott gyermekelemből mennyi fordulhat elő a szülőjében. A sorozat az elemek sorrendjét írja elő, a választási csoportok elemek alternatív módozatait írják le. A fenti deklarációban, deklaráltunk egy „*user*” nevű elemet. A tartalmi modellje egy szekvenciát ír le. A „*user*” elemeknek minden esetben rendelkeznie kell egy „*full_name*” gyermekelemmel, majd ezt követi minimum egy e-mail cím megadása, ezután pedig vagy egy „*school*”, vagy egy „*workplace*” elem következik, és végül lehetőségünk van hobbijaink megadására. Tehát ebben a tartalmi modellben láthattunk példát szekvenciára, választási csoportra, valamint két speciális karakter használatára is. A „+” karakter, mint láthattuk, azt jelzi, hogy az adott elemnek legalább egyszer elő kell fordulnia, míg a „*” az elemek tetszőleges előfordulási számára utal, azaz az is megengedett, hogy egyetlen egyszer sem tűnik fel az adott elem. Ezen kívül még létezik a „?” karakter, amely jelentése, hogy a kérdéses gyermekelem pontosan egyszer fordul elő, vagy egyszer sem. A „ , ” szimbólumot használhatjuk a szekvencia egyes elemeinek elválasztására, a „|” karakter egy választási csoport egyes alternatíváinak elkülönítésére szolgál, míg a kerek zárójelek a gyermekelemek sorozatát vagy egy választási csoportok zárnak közre.

A kevert tartalmú elemek szöveget és gyermekelemeket egyaránt tartalmazhatnak. A „#PCDATA” jelentése Parsed Character Data, azaz értelmezett szöveg, tehát azt jelzi, hogy az elem tartalmazhat szöveges adatot, amit értelmezésre kerül. Ezt követheti a gyermekelemek listája, amennyiben a szöveges tartalmon kívül elemeket is tartalmaz a kérdéses elem.

A tetszőleges típusú elemek a legflexibilisebb szerkezeti egységek, semmiféle előre meghatározott belső tartalommal nem rendelkeznek, bármit tartalmazhatnak. Deklarálásuk az ANY kulcsszóval történik. Lazasága miatt alkalmazása kerülendő, mert ellentmond a XML alaptörekvéseinek, túl nagy szabadságot ad a fejlesztőnek, vagy épp a felhasználónak.

Az attribútumok deklarálását az elemektől külön kell elvégeznünk. Általános alakja a következő:

```
<!ATTLIST ElemNév AttribútumNév AttribútumTípus AlapértelmezettÉrték>
```

Minden esetben kötelező nevet adnunk attribútumainknak, azonban közvetlen ennek megadása előtt azt kell jelezni, hogy mely elemhez tartozzon a tulajdonság. Ha megadtuk az attribútum nevét, típust is kell adnunk neki, és lehetőségünk van alapértelmezett értéket hozzárendelni. Ezen alapértelmezés lehet egy konkrét számérték is, vagy karakterlánc, tehát egy valóban „*default*” érték, de kerülhet ide három kulcsszó is, amelyek az attribútumok használatáról adnak információt. A három kulcsszó a következő:

- #REQUIRED: Az attribútum használata, és értékének megadása kötelező.
- #IMPLIED: Az attribútum használata opcionális.
- #FIXED: Az attribútum egy fix értékkel rendelkezik.

Az attribútumok típusa is eléggé korlátozott, 10 különféle típust használhatunk:

- CDATA: Karakteres adat, az értelmező nem dolgozza fel.
- NOTATION: A DTD egy másik pontján megadott jelölés.
- ENTITY: Külső entitásra, bináris egyedre való hivatkozás.
- ENTITIES: Több entitás megadása whitespace karakterekkel elválasztva.
- ID: Az attribútumhoz tartozó elem egyértelmű azonosítására használható attribútum-típus
- IDREF: A DTD egy másik pontján deklarált ID-re mutató hivatkozás.
- IDREFS: Olyan IDREF, amely ID-k listáját tartalmazza.
- NMTOKEN: Számok, betűk és egyéb karakterekből felépített név, ami az attribútum értékéül szolgálhat.
- NMTOKENS: Több NMTOKEN együttes használata.
- Karakterláncok sorozata: Ebben az esetben nem valamilyen előre definiált kulcsszót kell használnunk, hanem egy értéklistát, amelyet karakterláncok sorozata alkot. Gyakorlatilag egy választási lista.

Egy attribútum-deklarációs utasításban egyszerre akár több attribútumot is deklarálhatunk. Végezetül tekintsünk egy példát attribútum-deklarációra:

```
<!ATTLIST user
id ID #REQUIRED
username CDATA #REQUIRED
rang (admin | user) ”user”>
```

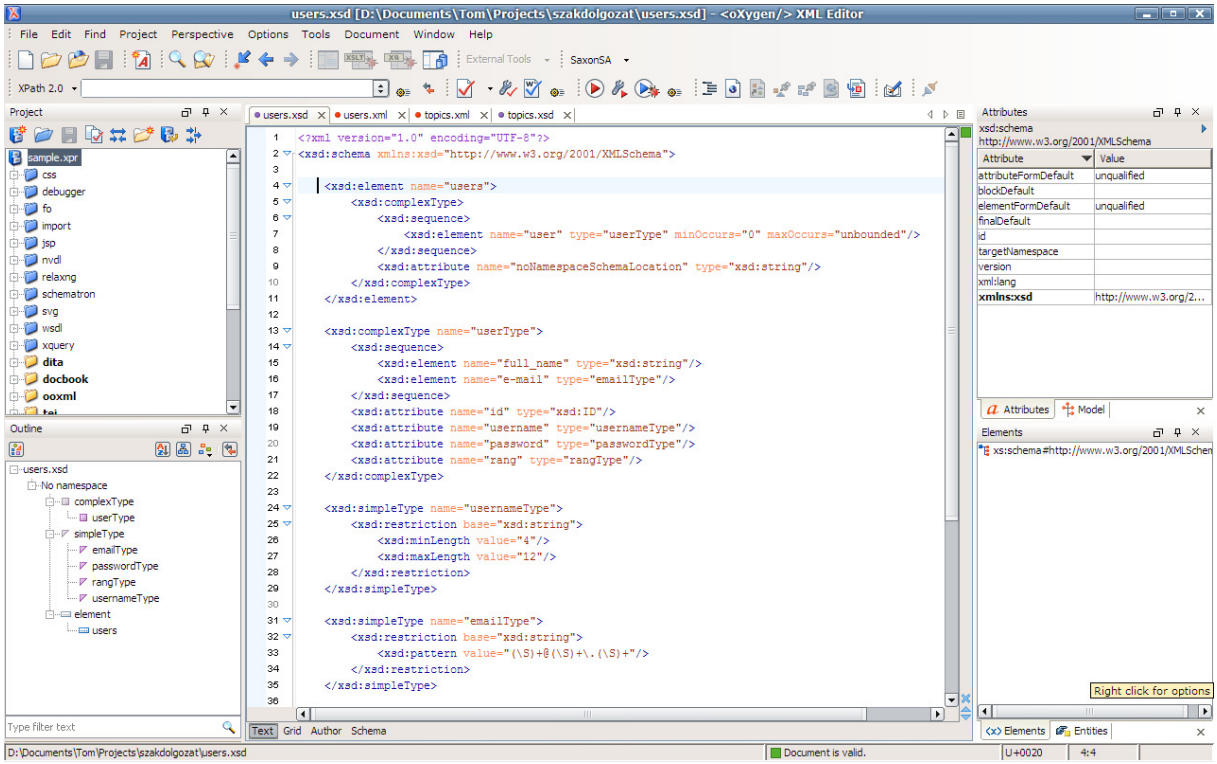
A fenti egyszerű példában a „*user*” elem attribútumainak egyidejű deklarálása történik. Rendelkezik egy ID típusú „*id*” nevű tulajdonsággal, melynek megadása kötelező, valamint egy „*username*” nevű egyszerű, nem értelmezett karakteres adattal, használata szintén kötelező, és végül egy „*rang*” attribútummal, mely két értéket vehet fel, és ezekből is alapértelmezett értéke a „*user*”.

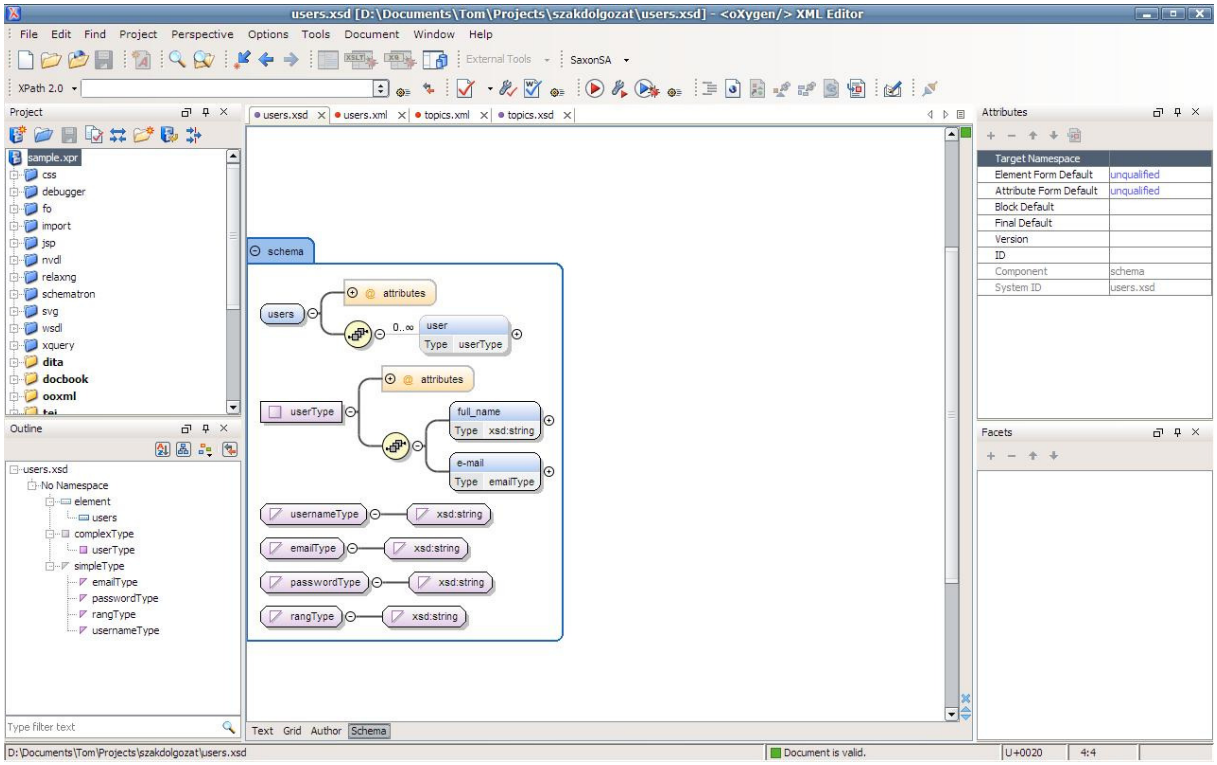
Most, hogy áttekintettük kettőt is az XML állományok sémaleírására használt technológiák közül, megállapítható, hogy mindkettő megállja a helyét, funkcionális mindkettő tökéletesen megfelel XML sémák létrehozására. Természetesen számos más eszköz áll még rendelkezésre, amelyek ugyanígy képesek ellátni az XML dokumentumok érvényességének ellenőrzését, azonban ennek a két technológiának összenőtt a neve az XML-el.

XML-szerkesztők

Az XML fájlok közönséges szöveges állományok, így ebből adódóan, bármilyen szövegszerkesztővel képesek vagyunk XML dokumentumokat létrehozni, szerkeszteni. Azonban vannak erre a célra fejlesztett szerkesztők, amelyek sok segítséget tudnak nyújtani a XML-el való munkánk során. Rendelkezésünkre állnak ingyenes szoftverek is, ilyen például a Butterfly XML Editor, de nem meglepő módon, léteznek kereskedelmi XML-szerkesztők is. Ezek az ismertebbek, méghozzá közülük is kettő emelkedik ki, az XML Spy, és az <oXygen/>.

Személy szerint én az <oXygen/> mellett tettem le a voksom, és ezt a szerkesztőt használtam a fejlesztés során. A legtöbb XML technológiát ismeri, tudja kezelni, a sémáktól, és egyszerű XML fájloktól kezdve a transzformációkat megvalósító XSLT-n keresztül a XML lekérdező nyelvéig, az XQuery-ig.


users.xsd [D:\Documents\Tom\Projects\szakdolgozat\users.xsd] - <oXygen/> XML Editor
File Edit Find Project Perspective Options Tools Document Window Help
Project
Outline
Attributes
Elements
Right click for options
Text Grid Author Schema


users.xsd [D:\Documents\Tom\Projects\szakdolgozat\users.xsd] - <oXygen/> XML Editor
File Edit Find Project Perspective Options Tools Document Window Help
Project
Outline
Attributes
Facets
schema
users
attributes
user
userType
full_name
e-mail
usernameType
emailType
passwordType
rangType
xsd:string
Text Grid Author Schema

XML adatbázisok

Az adatbázisok ma már alapvető fontosságúak minden szakterületen. Szinte bárhol járunk is a weben, legyen az egy közösségi oldal, hírportál, cégismertető honlap, vagy egy bank e-szolgáltatásait nyújtó lapjai, mindenhol meglapul a háttérben egy adatbázis, ami az általunk kért, keresett, vagy épp küldött adatokat kezeli. Pontosabban ezen adatok kezelését nem adatbázisok, hanem adatbázis-kezelő rendszerek végzik, de a két fogalmat gyakran egymás szinonimájaként szokás használni.

Nagyon fontos feladat egy rendszer fejlesztésének megkezdése előtt a megfelelő adattárolási módszer kiválasztása. Manapság rengeteg lehetőségünk van ezen a téren, számos adatbázis-kezelő rendszer áll rendelkezésünkre. A legismertebbek a relációs adatbázisok, de szintén gyakran használnak objektumorientált adatbázisokat, vagy épp hálós adatmodelleket kezelőeket is. Valamint egyre gyakoribb az XML adatbázisok használata.

XML, mint adatbázis

Az XML dokumentumok adatbázisok, a szó legszigorúbb értelmében. Ugyanis az adatbázis legalapvetőbb definíciója: adatok gyűjteménye, tárolója. Másfelől tekintve, nincs nagy különbség az XML fájlok és más fájlformátumok között, mindegyik valamilyen adatot tárol. Azonban, mint adatbázis-formátum, az XML rendelkezik pár előnnyel. Például önleíró (a címkék kifejezik az adatok struktúráját, és típusait), hordozható, illetve az adatokat képes fa-, illetve gráf struktúrában leírni. De vannak hátrányai is adatbázis szempontból. Az XML bőbeszédű, az adatok elérése lassú, akárcsak feldolgozásuk.

A nagyobb kérdés azonban az, hogy az XML, illetve az őt körülvevő technológiacsalád képesek-e egy adatbázis-kezelő rendszert alkotni. Az XML technológia sok lehetőséget kínál, amit egy szokványos adatbázis-kezelő rendszertől is megszokhattunk. Adatok tárolására ott vannak az XML állományok, sémák leírására alkalmas többek között a DTD, XSD, rendelkezik lekérdező nyelvekkel is (XPath, XQuery, XQL, stb.), valamint programozási felületekkel (SAX, DOM), és így tovább. Viszont vannak olyan dolgok is, amelyeket az adatbázis-rendszerek megvalósítanak, azonban az XML nem: Valóban hatékony adattárolás, indexelés, adatvédelem, tranzakciók, konkurenciakezelés.

Így az a következtetés vonható le, hogy olyan környezetben, ahol viszonylag kevés adatot kell kezelni, kevés felhasználó van jelen, szerény teljesítmény is elegendő, ott XML dokumentum, illetve dokumentumok betölthetik az adatbázis szerepét. Azonban ahol szigorú követelményeknek van kitéve az adatbázis, sok a felhasználó, szükség van az adatok integritásának megőrzésére, ott az XML dokumentumok önmagukban nem alkalmazhatóak. Mindenesetre az egyértelműen megállapítható, hogy az XML állományok felépítése, és a rendelkezésre álló egyéb XML technológiák alapjául szolgálhatnak adatbázis-rendszereknek.

Adat- és dokumentum-központú dokumentumok

Az XML dokumentumokat tartalmuk alapján két kategóriába tudjuk sorolni. Lehetnek adat-központúak, illetve dokumentum-központúak.

Az adat-központú dokumentumokat leggyakrabban adatátvitelre használják. Ezek gépi feldolgozásra készülnek, és maga az a tény, hogy ezek a dokumentumok XML dokumentumok, általában nem fontos. Ugyanis az ezt használó adatbázis, vagy alkalmazás számára egyáltalán nem lényeges, hogy az adatok egy rövid ideig XML-ben tárolódnak. Tipikusan ilyen dokumentumok lehetnek reptéri járatok, raktári lekérdezések, vagy tudományos adatok.

Az adat-központú dokumentumokra jellemző a meglehetősen szigorú struktúra, a finomszemcsézett adatok, a kevés vagy egyáltalán fel nem lelhető kevert tartalom. A sorrend, így például a testvérelemek sorrendje általában nem lényeges.

Az ilyen-jellegű adatokat legtöbb esetben relációs adatbázisban tárolják, az XML dokumentumok generálás útján állnak elő, jellemzően adatok továbbítása céljából, ekkor élettartamuk csak az adatcsere idejéig tart. De az is előfordulhat, hogy az adatokra nem csak ideiglenesen van szükségünk, mint például egy telefonregiszter, vagy egy betegnyilvántartó rendszerben, amikor az adatok tárolására a félig-strukturáltság a jellemző.

```
<!-   Egy tipikus adat-központú dokumentum -->
<Flights>
      <Airline>ABC Airways</Airline>
      <Origin>Dallas</Origin>
      <Destination>Fort Worth</Destination>
      <Flight>
```

```
         <Departure>09:15</Departure>
         <Arrival>09:16</Arrival>
      </Flight>
      <Flight>
         <Departure>11:15</Departure>
         <Arrival>11:16</Arrival>
      </Flight>
   </Flights>
```

A dokumentum-központú dokumentumok általában emberi feldolgozás céljából lettek létrehozva. Jellemző rájuk a lazább struktúra, a kevésbé szemcsézett adatok, és a sok kevert tartalom, valamint az elemek sorrendje is lényeges információ. Tipikusan kézzel írott dokumentumok, mint egy PDF, RTF vagy MS-World dokumentum, amelyeket majd XML-be konvertálunk. Az adat-központú dokumentumokkal ellentétben, ezek a dokumentumok általában nem adatbázisban jönnek létre.

```
<!-   Egy tipikus dokumentum-központú dokumentum -->
<Product>
   <Intro>
   The <ProductName>Turkey Wrench</ProductName> from <Developer>Full
   Fabrication Labs, Inc.</Developer> is <Summary>like a monkey wrench,
   but not as big.</Summary>
   </Intro>
  <Description>
   <Para>You can:</Para>
   <List>
   <Item><Link URL="Order.html">Order your own turkey wrench</Link></Item>
   <Item><Link URL="Wrenches.htm">Read more about wrenches</Link></Item>
   <Item><Link URL="Catalog.zip">Download the catalog</Link></Item>
   </List>
   <Para>The turkey wrench costs <b>just $19.99</b> and, if you
   order now, comes with a <b>hand-crafted shrimp hammer</b> as a
   bonus gift.</Para>
   </Description>
   </Product>
```

XML adatbázisok fajtái

Függetlenül attól, hogy az XML-t ideiglenes tárolóként használjuk-e információcsere céljából, vagy, hogy segítségükkel félig-strukturált dokumentum-centrikus modellt megvalósító dokumentumot akarunk-e létrehozni, sok esetben szükséges, hogy az XML-t adatbázisokban tároljuk, az adatok kifinomultabb tárolása, és visszanyerése céljából, valamint, ha az XML dokumentumok által tárolt adatok megosztott elérését szeretnénk biztosítani.

XML dokumentumok adatbázisban való tárolásának alapvetően két módja van, azaz az XML adatbázisokat két csoportba lehet osztani:

- XML-Enabled, azaz XML felülettel rendelkező adatbázis, mely képes XML formátumra alakítani belső tartalmát.
- Natív XML adatbázis, amely tárolási formájának alapjait XML állományok adják, azonban nem feltétlen szövegként tárolják azt.

Az XML-Enabled adatbázisok megőrzik a saját adatmodelljüket, ami lehet akár relációs, hierarchikus, vagy objektum-orientált, és az XML dokumentumok adatmodelljét a saját adatmodelljükbe képezik le. Tehát az XML-Enabled adatbázis lehet alapvetően relációs modellen alapuló adatbázis is, amely képes hagyományos lekérdezésekre XML formában megadni a választ, tehát az adatbázisrendszer elvégzi a transzformálást. Az ilyen jellegű adatbázisok lehetővé teszik az adataik XML nyelvre való leképezését. Használatuk akkor lehet hasznos, ha adatbázisban meglévő adatokat XML formátumban szeretnénk közzétenni, illetve ha már létező adatbázisba szeretnénk XML állományban lévő adatokat integrálni. Tehát az XML formátumok állhatnak input, illetve output oldalon is.

A natív XML adatbázisok közvetlenül az XML dokumentum modelljét használják. Léteznek olyan natív XML adatbázisok melyek konkrétan XML dokumentumokat tárolnak és használnak, azonban a gyakoribb az, hogy az XML dokumentumokban megtalálható adatok tárolására saját tárolási módot használnak, de mindkét esetben tökéletesen megőrzik az XML által meghatározott modellt.

Bár a választás tetszőleges, de általában kényelmesebb és hatékonyabb, ha az adat-központú dokumentumok tárolására és manipulálására XML-Enabled adatbázist, míg a dokumentum-központú dokumentumok esetében natív XML adatbázist használunk.

Azonban az is megfigyelhető, hogy a hagyományos adatbázisok és a natív XML adatbázisok közti határvonalak egyre inkább elmosódnak. Sok hagyományos adatbázisrendszer natív XML támogatást nyújt, és számos natív XML adatbázisrendszer támogatja a külső adatbázis használatát.

Adattárolási lehetőségek

Egyik adattárolási lehetőség a már fentebb említett mód, mikor is egy hagyományos adatbázisban tároljuk az XML dokumentumban található információkat. Ekkor, valamilyen leképezési mechanizmus szerint az XML által meghatározott sémát átalakítjuk az adatbázis modelljével összeegyeztethető sémává, az XML adatait konvertáljuk, és ugyanúgy táblákban, rekordokban tároljuk, mint bármilyen más adatot az adatbázisban. Ezt adat-központú dokumentumok esetében szokás használni.

Dokumentum-központú dokumentumok esetén más adattárolási lehetőségek ajánlottak. Tárolhatjuk egyszerűen fájlrendszerben az XML állományokat, amennyiben kevés van belőlük, és viszonylag kevés információt hordoznak. Ekkor alkalmazhatjuk rajtuk az XML lekérdező nyelveit, hogy adatokat nyerjünk ki belőlük. Ez a módszer például egy dokumentáció esetén alkalmazható.

De tárolhatjuk az XML dokumentum adatait relációs adatbázisban, mint szöveg, vagy bináris objektum. Szövegként nem túl szerencsés tárolni, ugyanis az XML állományok tetszőleges hosszúságúak lehetnek, és a relációs adatbázisok nem tudnak tetszőleges méretű adattípust biztosítani. Bináris objektumként, azaz BLOB-ként viszont megoldható a tárolásuk. Illetve létezik egy speciális bináris típus is, amelyet CLOB-nak hívunk. A CLOB szöveges adat tárolására lett létrehozva, és általában engedélyezett az ilyen típusú adatokban valamilyen keresés megvalósítása.

Valamint lehetőség van még XML dokumentumainkat natív XML adatbázisokban tárolni. Ebben az esetben is szoftverfüggő a tárolási mód konkrét formája, azonban erről még a későbbiekben lesz szó.

Adatok és dokumentumok leképezése

Az XML dokumentumok és adatbázis séma közötti leképezés az elemeket, attribútumokat és szöveges tartalmat érinti. XML állományok adatbázisba való leképezése során majdnem minden esetben elvesznek az XML-ben található fizikai konstrukciók (egyedek, CDATA adatok, kódolási információk), és némi logikai konstrukció is (feldolgozási utasítások, kommentek, elemek sorrendje). Ennek az oka, hogy az adatbázisok csak a tényleges adatokkal foglalkoznak. Következtetésképpen megállapíthatjuk, hogy ha egy XML dokumentumot egy hagyományos adatbázisba integrálunk, majd a későbbiekben az adatbázisból ugyanazokat az adatokat szeretnénk visszakapni XML formában, jelentős esély van rá, hogy nem ugyanazt a dokumentumot kapjuk vissza. XML dokumentumok adatbázis sémára való leképezésére leggyakrabban két módszert használnak: tábla-alapú leképezés, és objektum-relációs leképezés.

Tábla-alapú leképezés

A tábla-alapú leképezés az XML dokumentumokat, mint egyszerű táblákat, vagy táblák halmazát modellezi. Ebben az esetben az XML dokumentumok felépítése a következő:

```
<database>
      <table>
         <row>
            <column1>...</column1>
            <column2>...</column2>
            ...
         </row>
         <row>
            ...
         </row>
         ...
      </table>
      <table>
         ...
      </table>
</database>
```

Amennyiben egyetlen egy táblánk van, akkor a <database> gyökérelemre, valamint a további <table> elemre, elemekre nincs szükség.

A leképezés nyilvánvaló előnye, hogy rendkívül egyszerű. Implementálása könnyen megoldható, és az ezen a leképezésen alapuló programok, alkalmazások meglehetősen gyorsak, és igen hasznosnak bizonyulnak például adatbázisok közti adatcsere esetén, mikor egyszerre egy táblát mozgatunk a két adatbázis között.

A legnagyobb hátránya a tábla-alapú leképezésnek, hogy csak nagyon kevés XML állományon alkalmazható, mert az XML fájloknak a megadott formában kell lennie ahhoz, hogy az adatbázisba tudjuk integrálni adatait. Ráadásul, ahogy azt már korábban említettem, nem őrzi meg a fizikai struktúrákat, se a dokumentum információkat.

Korlátai ellenére is rengeteg middleware alkalmazás használja XML dokumentumok és adatbázisok közti adatcserére ezt a fajta leképezést, de Webes applikációs szerverek és XML-Enabled adatbázisok is alkalmazzák adataik XML-ben való exportálása céljából.

Objektum-relációs leképezés

Mivel a tábla-alapú leképezés nem alkalmas a legtöbb XML dokumentum leképezésére, ezért a XML-Enabled adatbázisok nagy része, de számos middleware alkalmazás is az objektum-relációs leképezést használja. Ez a leképezés az XML dokumentumokból egy objektumfát hoz létre, majd az objektumokat leképezi az adatbázisba. Az XML dokumentumok egyes elemeit osztályként, más elemeit skalár tulajdonságként reprezentálja.

De nézzünk is egy konkrét példát működésére:

```
<SalesOrder>
      <Number>1234</Number>
      <Customer>Gallagher Industries</Customer>
      <Date>29.10.00</Date>
      <Item Number="1">
         <Part>A-10</Part>
         <Quantity>12</Quantity>
         <Price>10.95</Price>
      </Item>
      <Item Number="2">
         <Part>B-43</Part>
```

```
         <Quantity>600</Quantity>
         <Price>3.99</Price>
      </Item>
</SalesOrder>
```

Ebből az XML állományból a következő objektumfát áll elő:

```
      object SalesOrder {
         number = 1234;
         customer = "Gallagher Industries";
         date = 29.10.00;
         items = {ptrs to Item objects};
      }           /   \
                 /     \
                /       \
   object Item {       object Item {
      number = 1;         number = 2;
      part = "A-10";      part = "B-43";
      quantity = 12;      quantity = 600;
      price = 10.95;      price = 3.95;
   }
```

Majd miután megkaptuk az objektumok alkotta fánkat, megtörténhet az objektumok adatbázisba való leképezése és a következő táblákat kapjuk:

```
   SaleOrders
   ----------
   Number   Customer                Date
   ------   --------------------    --------
   1234     Gallagher Industries    29.10.00
   ...      ...                     ...

   Items
   -----
   SONumber   Item   Part   Quantity   Price
   --------   ----   ----   --------   -----
   1234       1      A-10   12         10.95
   1234       2      B-43   600        3.99
   ...        ...    ...    ...        ...
```

Adatok lekérdezése

Természetesen, ha adatbázisban tárolt adatokon akarunk lekérdezéseket végezni, a lekérdezések minden esetben adatbázis függőek. Alapvetően XML dokumentumokról beszélünk, azonban ha az XML-ben tárolt adatokat már korábban leképeztük egy konkrét adatbázisba, például egy relációs adatbázisba, és a táblák rekordjai tartalmazzák a leképezett adatokat, akkor a konkrét adatbázis által támogatott lekérdezőnyelv segítségével tudunk adatainkon lekérdezéseket végrehajtani, példánk esetében az SQL nyelv szolgálná ki igényeinket. Azonban ezek a szokványos módszerek adatokat, értékeket adnak eredményül, minket viszont jobban érdekelne az, hogy miként tudunk XML formátumban eredményt kapni lekérdezéseinkre. Erre számos eszköz áll rendelkezésünkre, azonban kettővel fogunk foglalkozni ezek közül. Elsőnek az SQL/XML-el, amely nem önálló lekérdezőnyelv, hanem az SQL egy kiterjesztése, természetesen SQL centrikus. A nagyobb hangsúly azonban a második témán, azaz az XML lekérdező nyelvein lesz, amelyek teljes mértékben XML centrikusak.

SQL/XML

Az SQL/XML egy INCITS (The InterNational Committee for Information Technology Standards) által létrehozott szabvány. Fejlesztésében több óriáscég is közreműködött, többek között az Oracle, IBM, vagy épp a Microsoft, így támogatottságára nem lehet panasz. Az SQL/XML az SQL XML-t támogató kiterjesztése.

Olyan függvényeket biztosít számunkra, amelyek segítségével XML formátumban tudjuk adatainkat visszakapni a relációs adatbázisokból. Emellett egy XMLTYPE nevű adattípust is rendelkezésünkre bocsájt, amelyben a visszanyert XML tartalmat tudjuk elraktározni. Valamint még saját leképezési szabályokkal is el van látva, amelyek a rekordokban található adatok XML adatokká való átalakításáért felelnek.

Jó néhány XML visszanyerő funkcióval rendelkezik, ezek közül tekintsünk meg párat példaképpen.

Adott a következő két adatbázistábla:

Projects

ProjId	Name	CustId
1	Medusa	1
2	Pegasus	4
8	Typhon	4
10	Sphinx	5

Customers

CustId	Name	City
1	Woodworks	Baltimore
2	Software Solutions	Boston
3	Food Supplies	New York
4	Hardware Shop	Washington
5	Books Inc.	New Orleans

Majd ezeken a táblákon értelmezzük a következő lekérdezést:

```
select xmlelement(name "CustomerProj",
   xmlforest(c.CustId, c.Name as CustName, p.ProjId, p.Name as ProjName))
from Customers c, Projects p
where p.CustId=c.CustId
order by c.CustId
```

Az eredmény pedig egy XML formátumú adathalmaz (feltételezzük, hogy az „Book Inc.” nevű „customer” már nem volt az adatbázisban a lekérdezés ideje alatt):

```
<CustomerProj>
  <CustId>1</CustId>
  <CustName>Woodworks</CustName>
  <ProjId>1</ProjId>
  <ProjName>Medusa</ProjName>
</CustomerProj>
<CustomerProj>
  <CustId>4</CustId>
  <CustName>Hardware Shop</CustName>
  <ProjId>2</ProjId>
  <ProjName>Pegasus</ProjName>
</CustomerProj>
<CustomerProj>
  <CustId>4</CustId>
  <CustName>Hardware Shop</CustName>
  <ProjId>8</ProjId>
  <ProjName>Typhon</ProjName>
</CustomerProj>
```

Az eredménnyel elégedettek lehetünk, hiszen valóban XML formában kaptuk vissza a lekérdezett adatokat. A lekérdezésben látható pár SQL/XML által nyújtott metódus, mint az xmlelement(), amely egy XML elemet hoz nekünk létre nyitó és hozzá tartozó záró címkével. Emellett láthatunk még xmlforest() nevű metódust, amely oszlopokból XML elemeket gyárt, és mint láthatjuk, itt is specifikálható a létrehozandó XML elem neve. Bár a fenti lekérdezésben csak ezt a két metódust használtuk, de lehetőségünk van ezen kívül például attribútumok (xmlattribute()), vagy feldolgozási utasítások (xmlpi()), vagy komment (xmlcomment()) létrehozására is, de ez még mindig nem az összes lehetőség, amit az SQL/XML biztosít.

Másik eszköz, amit az SQL/XML megvalósít, az XMLTYPE adattípus. Ez valójában egy osztály, amely lehetőséget ad nekünk XML dokumentumok szövegének tárolására, és a dokumentum modelljének és ez által az elemeinek elérésére.

XML lekérdező nyelvek: XPath és XQuery

Bár az XML dokumentumok lekérdezését az XQuery lekérdezőnyelv tudja elvégezni, azonban ennek használatához elengedhetetlen az XPath ismerete, ugyanis az XQuery kifejezések XPath kifejezéseket tartalmaznak, ez szolgál a dokumentumban való navigálásra.

A *XPath* természetesen a W3C hivatalos szabványa, legújabb verziója a 2.0. Elsődleges feladata, hogy segítségével navigálni tudjuk az XML dokumentumokon belül.

Az XPath adatmodellje, csakúgy mint más modellek, felételezi, hogy az összes érték, amelyeket köztes lépésekben előállít, vagy szerkesztésre kerül, azonos általános alakkal rendelkezik. Ez az alak, az XPath esetében tételek szekvenciáját jelenti. Tehát egyszerűbben, a XPath kifejezések eredményei minden esetben tételek szekvenciája. Egy tétel a következők egyike lehet:

- Egyszerű típusú érték (például logikai, egész vagy szöveges érték)
- Csomópont: Több is van belőle, de a továbbiakban mi a következő hárommal fogunk foglalkozni:
    - Dokumentumok
    - Elemek
    - Attribútumok

Példáink számára a már megszokott „users.xml” adatbázisfájl tökéletes alany lesz, így ajánlatos lehet, ha szem előtt van:

```
<?xml version="1.0" encoding="UTF-8"?>
<users xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
      xsi:noNamespaceSchemaLocation="users.xsd">

      <user id="user-0" username="admin" password="admin" rang="admin">
            <full_name>Admin</full_name>
```

```
            <e-mail>admin@admin.com</e-mail>
      </user>

      <user id="user-1" username="bela" password="bela" rang="user">
            <full_name>Kovács Béla</full_name>
            <e-mail>kovacs_bela@probamail.hu</e-mail>
      </user>

      <user id="user-2" username="cecil" password="cecil" rang="user">
            <full_name>Nagy Cecil</full_name>
            <e-mail>Nagy_Cecil66@email.com</e-mail>
      </user>

</users>
```

A dokumentum-csomópontok alatt olyan állományokra kell gondolni, amelyek XML dokumentumot tartalmaznak. A „users.xml” fájlból a következő módon tudunk dokumentum-csomópontot képezni:

```
doc("users.xml")
```

A megjelölt fájlnak mindenképpen XML dokumentumnak kell lennie, a fájlt nem csak relatív módon adhatjuk meg, megadhatjuk az abszolút elérési útvonalát is, de egy teljes URL formájában is, ha távoli fájlról van szó.

Az elem és az attribútum csomópontok elérésésre útkifejezések megadásával van lehetőségünk. Egy útkifejezés lehet abszolút is, de relatív is. Ha a teljes útvonalat kívánjuk megadni, akkor a kifejezés minden esetben „/” jellel kezdődik, és ezt követik az egyes elemek, illetve attribútumok nevei, amelyek mentén el akarjuk érni a kívánt szekvenciát. Amennyiben elem csomópontot akarunk elérni, annak formája a következő:

```
/users/user/e-mail
```

Legelső esetben a *users* jelölőt használtuk, amivel a teljes dokumentum szekvenciáját kapjuk vissza, majd erre a szekvenciára alkalmazzuk a *user* jelölőt. Ennek eredménye az összes *user* elem, a benne található tartalommal együtt. És végezetül még alkalmazunk egy *e-mail* jelölőt is, amellyel az előzőleg megkapott *user* elemek alkotta szekvenciában található *e-mail* elemek szekvenciáját kapjuk vissza. Tehát az eredmény a következő:

```
<e-mail>admin@admin.com</e-mail>
<e-mail>kovacs_bela@probamail.hu</e-mail>
<e-mail>Nagy_Cecil66@email.com</e-mail>
```

Az attribútumok megtalálása is ugyanezen az elven működik, a különbség mindössze annyi, hogy az attribútumok neve elé „@” jelet kell tenni az útkifejezésben.

```
users/user/@username
<!-   Az eredmény pedig a username-ek szekvenciája -->
username="admin"
username="bela"
username="cecil"
```

A relatív útkifejezések annyiban különböznek a teljes útkifejezésektől, hogy azok nem kezdődhetnek „/” jellel. Az aktuális csomóponttól kezdve kell megadni az egyes jelölőket, amelyek mentén haladni akarunk.

Lehetősünk van tengelyek használatára is. Az előző kifejezés alakja például a következő lenne tengelyek használatával:

```
/child::users/child::user/attribute::username
```

De ezeken kívül még számos tengely létezik, például megadhatunk szülő, elől, leszármazott, utód, testvér vagy épp önmaga tengelyt is. De amelyet elég gyakran szokás használni, az a leszármazott vagy önmaga jelölő, amelyet a „//” szimbólummal jelzünk.

```
//e-mail
<!- Visszaadja az összes email elemet -->
/users//e-mail
<!- A kifejezésben bárhol használhatjuk a tengelyeket -->
```

Helyettesítő jeleket is használhatunk az útkifejezésben, nem feltétlen muszáj konkretizálnunk a jelölőket.

```
/users/*/e-mail
```

Ebben az estben az olyan *e-mail* elemeket is visszakapnánk, amelyet nem feltétlen a *user* elemekben találhatóak. Bár mi esetünkben ilyen nincs, de egy félig strukturált lazább szerkezetű XML állományban előfordulhat olyan eset, hogy az adatok kicsit szétszórtabbak.

Amikor egy kifejezést kiértékelünk, ezt korlátozhatjuk oly módon, hogy csak egy részhalmazát kövessük azoknak az utaknak, amelyek jelölői illeszthetők a kifejezés jelölőire. Ennek megvalósítását feltétellel ellátott jelölők biztosítják számunkra. A feltétel bármi lehet, ami logikai értékkel rendelkezik. Használhatjuk a megszokott operátorokat (=, !=, >, >=, …), valamint összetett feltételek esetén a feltételeket összeköthetjük *and* és *or* kulcsszavakkal.

```
/users/user[username=”bela”]
<!- A kapott szekvencia -->
<user id="user-1" username="bela" password="bela" rang="user">
            <full_name>Kovács Béla</full_name>
            <e-mail>kovacs_bela@probamail.hu</e-mail>
</user>
<!- Egy újabb kifejezés -->
/users/user[rang=”user”]/@username
<!- A kapott szekvencia -->
username="bela"
username="cecil"
```

De léteznek még ezen kívül egyéb hasznos feltételek. Például megadhatunk olyan feltételt, hogy egy adott szülőelemnek hányadik gyermekelemét kívánjuk visszakapni, vagy csak azokat az elemeket kapjuk vissza, amelyek rendelkeznek egy adott gyermekelemmel, vagy adott attribútummal.

```
<!- Adott gyermekelem meghatározása -->
/users/user[2]
<!- e-mail gyermekelemet tartalmazó user-ek -->
/users/user[e-mail]
```

Az XPath a bemutatott eszközein kívül még számos hasznos lehetőséget rejt magában, azonban ennyi alap már elegendő ahhoz, hogy az XQuery tárgyalásába kezdjünk.

Az *XQuery*-t akárcsak az XPath-t a W3C hozta létre. A XQuery a XPath egy kiterjesztése, minden XPath kifejezés egyben XQuery kifejezés is. Azonban túlmutat az XPath lehetőségein, tudása kiegészül a FLWOR kifejezésekkel. Az FLWOR (ejtsd, mint az angol Flower szót) rövidítés a *for*, *let*, *where*, *order by* és *return* záradékokból áll össze. Az XQuery nyelv XML dokumentumokon végrehajtható lekérdezésekre szolgál, azon adatbázisrendszerek lekérdezőnyelve lehet, amelyek az adatokat XML formában tárolják, mint a natív XML adatbázisok, vagy legalább megjeleníti képesek adataikat XML formában,

mint az XML-Enabled adatbázisok. Az előbb említett öt záradék - amelyek nagy segítségünkre vannak az XML dokumentumok lekérdezésében - adja az XQuery funkcionalitását. Az XQuery egy nagyon összetett téma, mi most csak az alapjaival foglalkozunk, amelyek egy része a fejlesztett rendszer adatbázisából való lekérdezésében is fontos szerepet játszik.

A XQuery áttekintését kezdjük a FLWOR záradékaival, elsőnek is a *for* utasítással.

```
for $felhasznalo in doc(”users.xml”)/users/user
```

A változók nevét minden esetben a „$” szimbólummal kell kezdeni. A *for* kulcsó után megadunk egy változót (ciklusváltozót), majd az *in* kulcsszó után pedig egy kifejezést. A cél a kifejezés kiértékelése, amelynek eredménye tételek szekvenciája lesz. A megadott változó értékül kapja egyesével mindegyik tételt, és minden alkalommal végrehajtódnak az ezt követő utasítások az éppen aktuális értékkel. A fenti példában a változó sorban értékül kapja a *users* elemben található *user* elemeket.

A *let* kulcsszóval újabb változókat hozhatunk be a kifejezésekbe, ezeknek értéket adhatunk.

```
let $user_doc := doc(”users.xml”)
```

A *where* záradékot egy szekvencia egy tételére tudjuk alkalmazni. Nem meglepő módon mindig feltétellel használjuk, amely feltétel kiértékelésre kerül.

```
where $nev = ’Bela’
```

Az *order by* záradék a kijelölt és esetlegesen szűrésen átesett szekvencia tételeit tudja valamilyen tetszőleges sorrend szerint rendezni. Alapértelmezett rendezés a növekvő, de a *descending* kulcsszó megadásával elérhetjük a csökkenő rendezést.

```
order by $felhasznalo/@username
```

A *return* záradék a kifejezések végén található meg, feladata, hogy visszaadja a kijelölt és megszűrt tételek szekvenciáját.

```
return $felhasznalo/full_name
```

Ezek az utasítások külön-külön önmagukban természetesen nem állják meg a helyüket, kombinálásukkal viszont megfelelő lekérdezéseket tudunk létrehozni. Most, hogy átvettük a

nyelv alapjait, nézzünk pár összetettebb példát, melyet alkalmazásunk is használ, ha nem is teljesen ugyanebben az alakban.

```
for $i in doc("users.xml")/users/user
where $i/@username eq 'admin' and $i/@password eq 'admin'
return $i/@id
```

Ennek a kifejezésnek az autentikációban jut fontos szerep. Egy adott felhasználó bejelentkezési kísérleténél ellenőrzi, hogy az általa megadott névvel és jelszóval létezik-e felhasználó az adatbázisban. A *$i* ciklusváltozó a *users.xml* fájlban található *user*-ek szekvenciájának tételeit kapja meg egyesével, azaz a *user* elemeket sorban, majd elvégez rajtuk egy szűrést, azaz kettőt, majd ha mindkét feltétel igaz értékkel értékelődik ki, akkor visszaadjuk a felhasználóhoz tartozó azonosítót. A feltételben két féle operátort használunk, egyik az *eq* (equals) operátor, amely annyiban különbözik a „=” operátortól, hogy míg az előbbi úgynevezett érték operátor, csak egy-egy atomi értéket tud összehasonlítani, addig az úgynevezett általános operátor, amely halmazokon, azaz pontosabban atomi értékek szekvenciáján is értelmezhető. Az XQuery általános hasonlító operátorai közt megtaláljuk a szokványos operátorokat, tehát a „=”, „!=”, „>”, „>=”, „<”, „<=” operátorokat. Az érték operátorok pedig a következők lehetnek: *eq* (equals), *ne* (not equals), *gt* (greater than), *ge* (greater or equals), *lt* (less than), *le* (less or equals). Valamint használunk még logikai operátort is a két feltétel összekötésére. A logikai operátorok (*and*, *or*) a tőlük megszokott módon működnek. Ebben a konkrét példában a felhasználónevet és a jelszavat mi adtuk meg, de a valódi implementációban ez nem így néz ki, ott a felhasználó által megadott paraméterek kerülnek a kifejezés szűrőrészébe.

```
for $i in doc("users.xml")/users
return count($i/user)
```

Ez a kifejezés arra szolgál, hogy visszaadja a regisztrált felhasználók számát. Az XQuery rengeteg függvénnyel segíti munkánkat, ezek közül egyik a *count()*, amely természetesen egy egész értéket fog nekünk visszaadni, méghozzá egy darabszámot, a példánkban a *users* elemben található *user* elemek előfordulásának számát. De ez csak egy függvény a sok közül, számos aritmetikai, sztringkezelő, dátumkezelő, stb. függvénnyel segíti a hatékonyabb munkát.

Az XQuery-ban lehetőségünk van két vagy több dokumentum összekapcsolására, hasonlóan az SQL tábla-összekapcsolásaihoz. Erre szolgál példaként a következő kifejezés.

```
for $i in doc("users.xml")/users/user[@username='admin']
for $j in doc("topics.xml")/topics/topic[@topic_id='topic-
0']/members/member_id
where data($j) eq data($i/@id)
return $j
```

A fenti lekérdezés szintén autentikációs feladatot lát el. Amennyiben egy felhasználó egy adott témát (*topic*ot) meg akar tekinteni, a szerver-oldali alkalmazás egy olyan kérést küld az adatbázisnak, hogy ellenőrizze le, hogy a felhasználó szerepel-e az adott téma tagjai között. Mivel a témákat másik állományban tároljuk (*topics.xml*), így szükséges a két állomány összekapcsolása. A felhasználónév és a témaazonosító itt is paraméterekből érkezik, nem előre definiáltak. Az összekapcsolást két *for* ciklussal oldjuk meg. Itt is láthatunk egy függvényt, a *data()* függvényt, mely az adott csomóponton található adatot, azaz mondhatni az értéket adja vissza, nem pedig a csomópontot.

Mindenképpen meg kell említeni az XQuery kapcsán, hogy lehetőséged ad tárolt alprogramok létrehozására, amelyeket a későbbiekben meghívhatunk, használhatunk. Ebben nagyon jó optimalizációs lehetőségek rejlenek. A deklaráció formája:

```
declare function local:add($x,$y) { $x + $y }
```

Ami sajnos még nem forrt ki igazán az XQuery eszközei közül, azok az update műveletek. Erre az XQuery Update Facility lett létrehozva, azonban mivel még nem teljesen tisztázott része a specifikációnak, ezért inkább nézzünk meg egy konkrét adatbázis-rendszer (Sedna) adta frissítési lehetőséget.

```
UPDATE insert <user id="user-22" username="steve" password="secret"
               rang="user"><full_name>"Steven Smith"</full_name>
                           <e-mail>"ssmith@gmail.com"</e-mail></user>
                            "into doc("users.xml")/users"
```

A fenti kifejezés egy felhasználó létrehozását végzi el az adatbázisban.

Az imént pár oldalban bemutatott XQuery technológia nyújtotta lehetőségek jóval sokrétűbbek, minthogy részletesebben foglalkozhassak vele ebben a dolgozatban, azonban az

alapjaival sikerült megismerkedni, jelentőségét, alkalmazásomban való használatát sikerült bemutatni.

Natív XML adatbázis

Mint ahogy arról már korábban szó volt a natív XML adatbázisok igazi előnyét az jelenti, hogy az XML dokumentumok modelljét tökéletesen megőrzik, így az adatok lekérdezésére, manipulálására az XML lekérdező nyelvei igen hatékonyan alkalmazhatóak. Ezt a jó tulajdonságot annak köszönhetik az natív XML adatbázisok, hogy erre a célra lettek kifejlesztve. Leginkább dokumentum-központú dokumentumok esetén használják, de szintén jó szolgálatot tesznek a félig strukturált adatok tárolásában.

A natív XML adatbázisoknak, azaz röviden NXD-nek, definíció szerint a következő követelményeket kell teljesítenie:

- Logikai modellt definiál az XML dokumentum számára, és ennek megfelelően tárolja és adja vissza az adatokat. Minimum elvárás, hogy a modell tartalmazzon elemeket, attribútumokat, és PCDATA-t, és ezeknek sorrendjét.
- A tárolás alapvető logikai egységéül XML dokumentumnak, azaz az általa meghatározott modellnek kell szolgálnia, csak úgy, mint egy relációs adatbázisban egy tábla tartalmazza az adatokat tároló sorokat.
- Nem szükséges rendelkeznie semmilyen egyéni, speciális fizikai modellel. Épülhet relációs, hierarchikus, vagy objektum-orientált adatbázisra. Vagy akár használhat saját tárolási módot, például indexelt, tömörített fájlokat.

Sok lényeges dolgot elárul ez a definíció, azonban a legfontosabb három pontnak talán a következőket lehetne kiemelni:

- A natív XML adatbázisok arra specializálódtak, hogy XML adatokat tároljanak, és ennek minden komponensét közvetlenül az XML modell szerint írják le.
- Dokumentumokat tudunk bevinni az adatbázisba, és dokumentumokat nyerünk is vissza.
- A natív XML adatbázisok nem feltétlenül különálló adatbázisok.

Ahogy ebből a definícióból tisztán kivehető, a natív XML adatbázisok nem egy új alacsony-szintű adatbázis modellt jelentenek, de nem is azért jöttek létre, hogy leváltsák a régebbi adatbázisrendszereket, hanem egyszerűen szólva csak egy eszköz, hogy segítse a fejlesztőt egy robosztus adatbázis létrehozásában és benne XML dokumentumok lekérdezésében, manipulálásában.

A natív XML adatbázisok architektúrájukat tekintve két kategóriába sorolhatók. Beszélhetünk szöveg alapú, és modell alapú architektúráról.

A szöveg alapú architektúrával rendelkező natív XML adatbázisok az XML-t, mint szöveget tárolják. Leginkább az olyan relációs adatbázisok tartoznak ebbe a kategóriába, melyek BLOB, illetve CLOB objektumokban raktározzák az XML dokumentumok szövegét. A közös az ilyen adatbázisokban, hogy hatékony indexelést valósítanak meg, így az XML dokumentumokon a navigálás könnyen megy, ez pedig meglehetősen nagy adatlekérdezés sebességi előnyöket biztosít az olyan esetekben, mikor teljes dokumentumokat vagy nagyobb dokumentum részleteket kérdezünk le.

A modell alapú architektúrát megvalósító natív XML adatbázisok egy belső objektummodellt építenek fel az XML dokumentumokból, és ezt a modellt tárolják. Hogy ezt milyen módon valósítják meg az adatbázisfüggő, tárolhatják akár relációs vagy objektum-orientált adatbázisban is. Az ilyen típusú natív XML adatbázisok, melyek más adatbázisra épülnek, teljesítményükben hasonlót nyújtanak, mint az adatbázis, amelyekre épültek. De van egy másik fajtája is a modell alapú architektúrával rendelkező natív XML adatbázisoknak. Ezek az adatbázisok egyéni módon oldják meg a dokumentumok tárolását, függetlenek más modellektől, adatbázisoktól. Mivel ezek egyediek, így konkrét tárolási módjukról nem nagyon lehet beszélni. Ami azonban a teljesítményüket illeti, a szöveg alapú architektúrájú adatbázisokra hajaznak, ugyanis elég gyakori, hogy fizikai mutatókat használnak az egyes bejegyzések között.

Példák XML adatbázisokra

MySQL

A MySQL egy nagyon elterjedt relációs adatbázis, a Sun Microsystems terméke. Nem konkrét XML adatbázis, csupán támogatást nyújt hozzá, azaz XML-Enabled adatbázisok közé lehet sorolni.

A *mysql* és *mysqldump* kiegészítők valósítják meg az XML támogatást. Tábla-alapú leképezést használnak, hogy az SQL lekérdezések eredményeképpen XML-ként adják vissza az adatokat. Emellett függvényeket is biztosítanak, amelyek az XML elemek elérését illetve frissítését segítik. Mivel nem rendelkezik kizárólagos XML típussal, így szövegként tárolja az XML dokumentumokban fellelhető adatokat.

Egyik ilyen függvény az *ExtractValue*, amely egy XPath kifejezés használatával képes visszaadni a kifejezésre illeszkedő elemeket. Amennyiben nem csak egy elem illeszkedik a XPath által meghatározott kifejezésre, akkor összefűzi az elemeket alkotó szöveget és azzal tér vissza.

Másik ilyen függvény az *UpdateXML*, amely három paraméterrel dolgozik. Egyik egy olyan XML „érték” melyet keresünk a dokumentumban, majd egy másik paraméter egy XPath kifejezés, majd végezetül egy új XML érték, mellyel le akarjuk cserélni az eredeti értéket, melyet az első paraméterben adtunk meg.

eXist

Az eXist egy olyan natív XML adatbázis, mely egyéni adattárolást valósít meg, melynek alapját B+ fák és lapozó fájlok alkotják. Java-ban lett megírva, amely nagy előnye is, de egyben hátránya is az adatbázisnak. Előnye, mert éppen a Java miatt, talán a legtöbben használják XML dokumentumok számára, bár ebben az is közrejátszik, hogy Open Source technológia. Hátránya leginkább az, hogy vagy Java-n, vagy http-n keresztül használjuk, más módunk nincs.

Az eXist teljes körű XQuery támogatást nyújt, emellett XSLT-t is ismeri, sőt szerveroldalon is meg tudja valósítani a vele történő transzformációt. Mivel nyílt forrású szoftverről van szó, így rengeteg felhasználók által írt modullal rendelkezik, amelyek nagy segítségünkre

lehetnek. Az XML dokumentumok frissítését is el tudjuk végezni, mivel implementálja az XUpdate-ot, valamint részben XInclude és XPointer támogatással is bír. Szintén erősségéről árulkodik, hogy hívható XML-RPC-n, SOAP-on keresztül is.

Hátránya, hogy nagyméretű állományokkal nem igazán boldogul, elég lassan végzi el ezeken a lekérdezéseket.

Sedna Native XML Database System

A Sedna egy nyílt forráskódú, C/C++-ban megírt, natív XML adatbázis. Minden tulajdonsággal rendelkezik, amit egy jó adatbázisnak tudnia kell, megvalósítja az ACID tulajdonságokat, megfelelő biztonságot biztosít, lehetőségünk van rendszer-visszaállításra, valamint felhasználó kezelése is kiemelkedő. Implementálja a W3C XQuery-t, valamint XML frissítő nyelvvel is rendelkezik. Ami még kiemelkedő, hogy lehetőségünk van XML-triggerek használatára, valamint beépített függvényeivel relációs adatbázisokat is elérhetünk.

Mivel nyílt forráskódú technológia így elég sok API-t hoztak létre hozzá, valamint rendelkezik egy igen hasznos, könnyen kezelhető grafikus felülettel is. Bár parancssorból is lehetőségünk van az adatbázis kezelésére, azonban a grafikus felület megkönnyíti munkánkat.

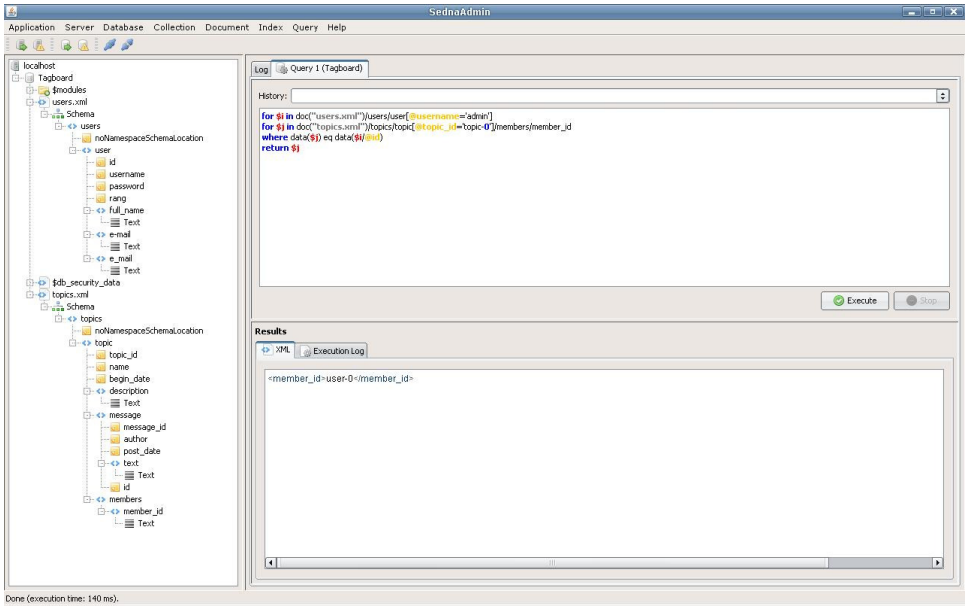
A felsoroltakban nem volt benne összes jó tulajdonsága az adatbázisnak, számos ilyennel rendelkezik, többek között ez volt az oka annak, hogy én magam is a Sedna-t választottam a rendszerem adatbázisának.

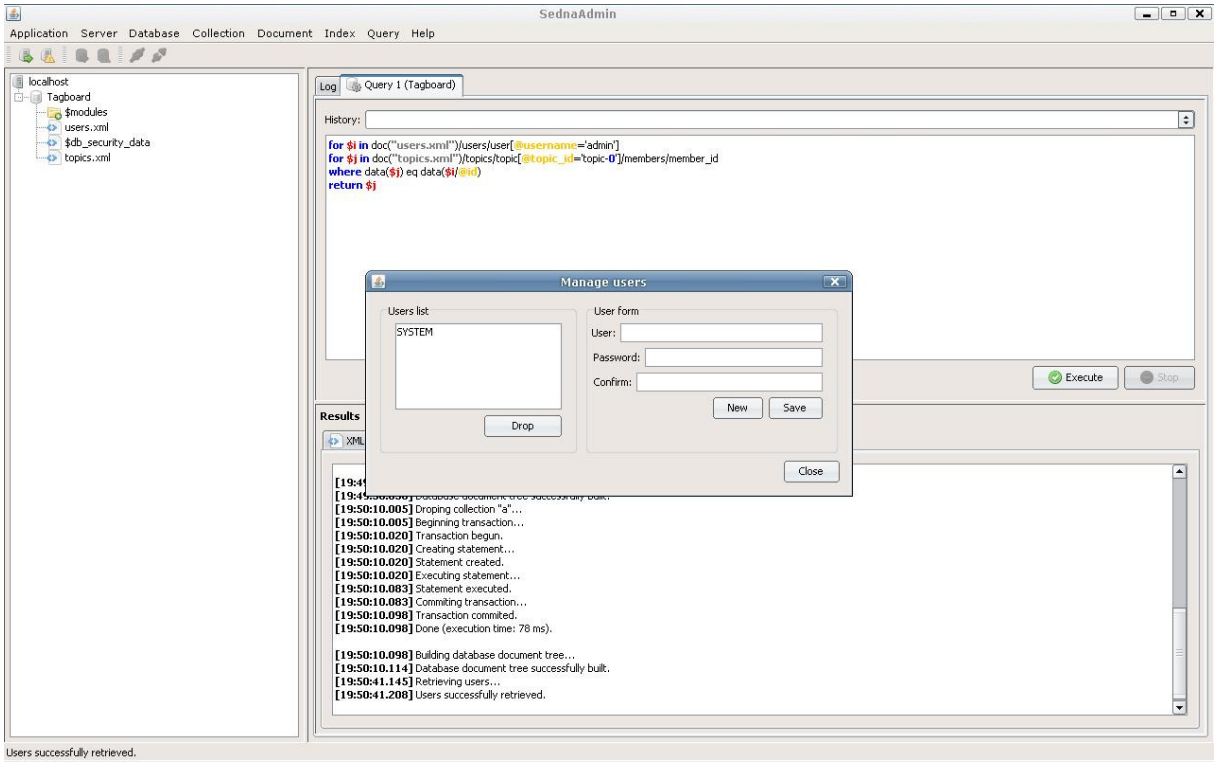
Mint említve lett, elég sok API áll rendelkezésünkre, így Java-ban sem okozhat gondot a használata. Erre konkrét példát is megtekinthetünk.

```
import ru.ispras.sedna.driver.*;

public class SednaManager {
    private SednaConnection conn = null;
      private String host = "localhost";
      private String dbName = "Tagboard";
      private String userName = "SYSTEM";
      private String password = "MANAGER";

    public SednaManager(String host) {
        if (host != null && !host.isEmpty())
```

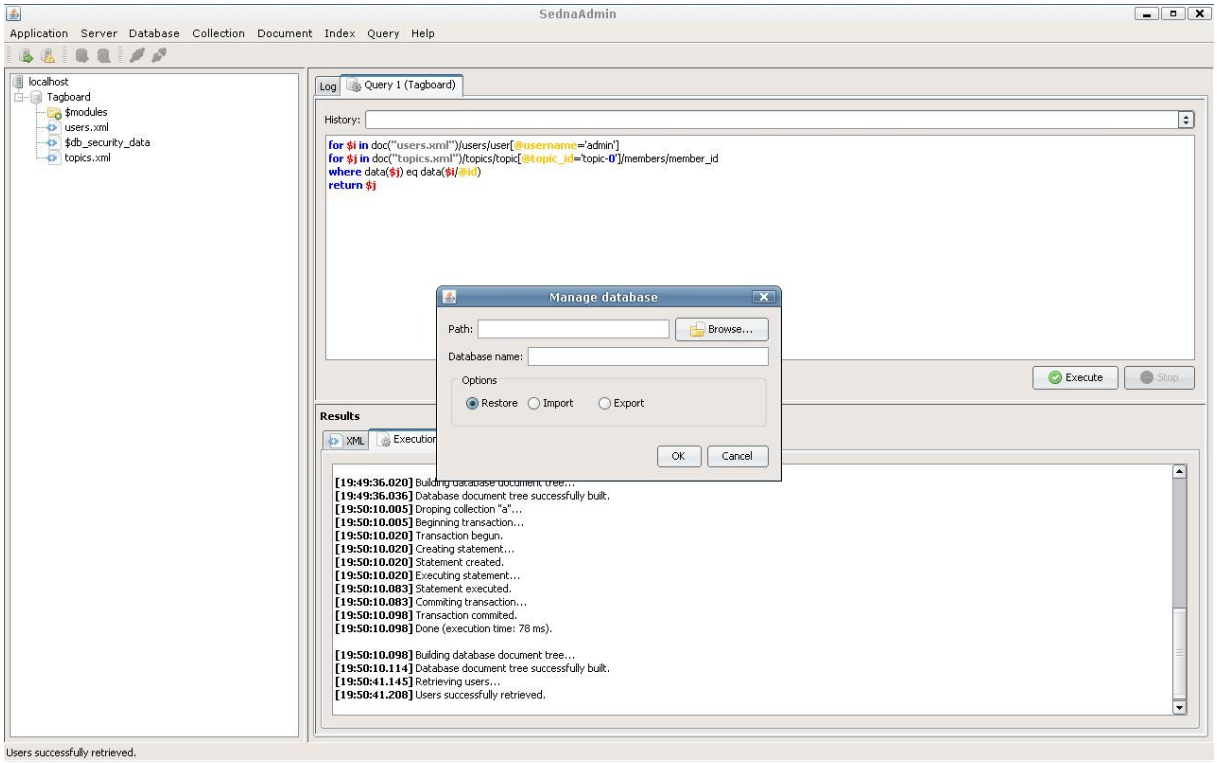
```
            this.host=host;
        try {
            conn = DatabaseManager.getConnection(this.host, dbName,
userName, password);
        }
        catch(DriverException ex) {
            ex.printStackTrace();
        }
    }

      public String executeQuery_2(String query){
        String s = null;
        try {
            conn.begin();
            SednaStatement st = null;
            st = conn.createStatement();
            st.execute(query);
            SednaSerializedResult sr = st.getSerializedResult();
            s = sr.next();
            conn.close();
        } catch (DriverException ex) {
            ex.printStackTrace();
        }
        return s;
    }
```

A Sedna Java-ban használható osztályairól és metódusairól, valamint azok működéséről is könnyű információkat szereznünk, mivel elég jól használható dokumentációt adnak hozzá.





Felhasználókezelés a Sedna grafikus felülete segítségével



Adatbáziskezelés a Sedna grafikus felülete segítségével

Webszolgáltatások

Bevezetés a webszolgáltatások világába

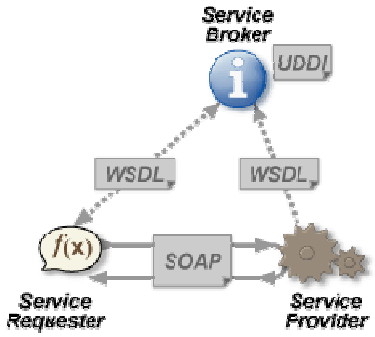
A wikipedia szerint, a webszolgáltatás vagy angolul webservice alkalmazások közötti adatcserére szolgáló protokollok és szabványok gyűjteménye. Különböző programnyelveken írt és különböző platformokon futó szoftveralkalmazások számítógép-hálózatokon (mint az Internet) keresztül történő adatcserére használják a webszolgáltatásokat.

Egy webszolgáltatás alapvetően egy szerver oldali alkalmazásból áll, amely egy szolgáltatást, vagy szolgáltatások halmazát nyújtja. Ezt a szolgáltatást vehetik igénybe kliens oldali alkalmazások. A kommunikáció menete abból áll, hogy a kliens-alkalmazás meghívja a webszolgáltatás egyes szolgáltatásait, egyes metódusait, és ezekre a szerver oldali alkalmazás megadja a megfelelő válaszokat.

A kérdés az, hogy honnan tudja a kliens oldalon lévő alkalmazás, hogy milyen szolgáltatásokat nyújt a webszolgáltatás, és ezeket milyen metódusokkal hívhatja meg. Erre szolgál a WSDL (Web Service Definition Language), amely a szerver által nyújtott szolgáltatásokat definiálja a kliensek számára. De lehetőség van a webszolgáltatások (azaz az ezeket leíró WSDL állományok) nyilvántartására is, erre az UDDI (Universal Description, Discovery, and Integration) lett kitalálva, ami ezáltal azt is lehetővé teszi, hogy böngésszünk az elérhető webszolgáltatások között.

A webszolgáltatás nyílt szabványokat használnak, így az adatcserére is egy nyílt szabványt, az XML-t használják. Mivel minden kommunikáció XML formátumban történik, így platformfüggetlenség biztosítható, ami a webszolgáltatások egyik legnagyobb előnye. Ezen XML felépítésű üzenetekre több szabvány is létezik, mi ezek közül talán a két leggyakrabban használttal, az XML-RPC-vel (XML-Remote Procedure Call), és a SOAP-al (Simple Obejct Acces Protocol) fogunk foglalkozni.

Mivel szerver-kliens kommunikációról beszélünk, tehát hálózatokon keresztül történik az adatok küldése, így szükséges valamilyen hálózati protokollt használnunk. A leggyakrabban használt protokollok a http (HyperText Transfer Protocol), smtp (Simple Mail Transfer Protocol), ftp (File Transfer Protocol), vagy xmpp (Extensible Messaging and Presence Protocol), ezek közül is a legelső fordul elő a leggyakrabban.



Webszolgáltatások

A webszolgáltatások előnyei:

- platform és nyelvfüggetlenség
- nyílt szabványok és protokollok használata
- http használatával tűzfalak megkerülése
- szolgáltatások és komponensek újrafelhasználása
- lehetővé teszik különböző gyártóktól származó szoftverek és szolgáltatások kombinálását új, integrált szolgáltatások létrehozására

A webszolgáltatások hátrányai:

- tranzakciókezelés hiánya, vagy körülményes használata
- az XML szöveges mivolta miatt lassabbak, mint az elosztott rendszerek
- a http protokoll használata miatt nagyobb biztonsági kockázat

XML-RPC

Mivel a fejlesztett webszolgáltatásom nem ezt a protokollt használja az üzenetek továbbítására, így ezzel a technológiával csak érintőlegesen foglalkozunk.

Az XML-RPC a távoli eljáráshívást XML üzenetekkel megvalósító protokollok egyike. A kérések és a válaszok is XML-ben vannak kódolva. A kéréseket a HTTP POST útján küldik el, a válaszok pedig a HTTP válasz *body* részében találhatóak meg. Az XML-RPC üzenetek rendkívül egyszerű felépítésűek. Az XML-RPC hátránya, hogy kevés típussal rendelkezik,

ezáltal az összetettebb típusokat nem igazán tudják ábrázolni. Ez a protokoll nincs semmilyen kapcsolatban a szolgáltatás leíró nyelvvel, azaz a WSDL-el.

Most pedig tekintsünk meg egy egyszerű példát XML-RPC kérésre:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>pelda.melyikVaros</methodName>
  <params>
    <param>
        <value><int>4025</int></value>
    </param>
  </params>
</methodCall>
```

Az elküldött kérés egy olyan szolgáltatást vesz igénybe, mely képes visszaadni egy adott irányítószámhoz tartozó város nevét. A kérés felépítése nagyon egyszerű, a <*methodCall*> elemben kell kijelölni a meghívandó metódust, majd a <*params*> elemben lehet megadni a paramétereket.

A kérésre adott válasz a következő:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>Debrecen</string></value>
    </param>
  </params>
</methodResponse>
```

A <*methodResponse*> elemben található meg a kérésre adott válasz, ami jelen esetben egy *string* típus, értéke pedig Debrecen.

SOAP

A SOAP számítógépek közti információcseréhez alkalmazott XML alapú protokoll. Bár az SOAP-ot számos átviteli protokoll igénybe veheti, központi szerepét a távoli eljárások HTTP protokollon keresztül történő meghívásában fejti ki. Egyik előnyének éppen ez számít, hogy

nem csak egy protokollhoz köthető, hanem például használható SMTP-vel is. Szerencsénkre sok programozási nyelv implementál eszközöket a SOAP üzenetek kezelésére.

Egy SOAP kérés a következőképpen néz ki:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A kérésre adott SOAP válasz:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Mint látható a kérés és a válasz is egyaránt egy *SOAP-Envelope*-ban található. Akárcsak a boríték, úgy a benne található *Body* elem használata is kötelező. Az Envelope elem nyitó címkéjében tudjuk megadni a névtér deklarációinkat. A SOAP üzenetek a kötelező *Envelope* és a *Body* elemeken kívül még opcionálisan tartalmazhatnak egy *Header* elemet is. Az *Envelope* elem minden esetben a gyökéreleme az üzeneteknek. Az opcionális fejléc, azaz a *Header* elem egy rugalmas keretrendszert biztosít a kiegészítő alkalmazás szintű követelmények megadásához. A legtöbb SOAP alkalmazás nem használja ki a *Header* elem nyújtotta lehetőségeket, pedig például hitelesítésre is lehetne használni. A *Body* elem veszi körül a SOAP üzenet lényegi tartalmát, itt tudjuk meghatározni a kívánt szolgáltatást, és meghívni annak metódusait, valamint átadni paramétereinket, illetve válasz esetén itt lesz megtalálható a kért érték. Ha hiba következik be, akkor a *Body* elembe belekerül egy *Fault*

elem, amely egy hibakóddal, azaz inkább egy hibaazonosítóval és egy hozzá tartozó hibaüzenettel fogja informálni a klienst a hiba okáról.

A SOAP üzenetek alakja kissé kaotikusnak, átláthatatlannak tűnhet elsőnek, főleg, ha további névtér deklarációk és további elemek kerülnek még bele. Azonban a jó hír az, hogy mivel a programozási nyelvek nagy része rendelkezik SOAP üzenetek kezelésére szolgáló eszközökkel, így általában konkrétan a SOAP üzenetekkel nekünk nem kell foglalkozni.

A SOAP bár nehezebben érthető, mint az XML-RPC, ez épp azért van, mert tudásban az előbbi a befutó. Jóval több adattípust ismer, mint az XML-RPC, és adattípusai segítségével az összetettebb típusokat is képes kezelni.

WSDL

A WSDL a webszolgáltatások leírását hivatott elvégezni. Szintén egy XML alapú nyelv, amely a webszolgáltatásoknak mint egy publikus interfésze van jelen. A kliens oldali alkalmazások ez alapján informálódnak minden fontos jellemzőjéről a szolgáltatásnak. Elég részletesen írja le a webszolgáltatás jellemzőit, nem csak a metódusok neveit találjuk meg a *wsdl* állományokban, hanem leírja a webszolgáltatás elérési helyét, a szükséges adattípusokat, de az összekapcsolási információkat is itt leljük. A WSDL nincs a SOAP-hoz kötve, más üzenetküldési protokollt is használhat, azonban a SOAP szolgáltatások leírásához tartalmaz beépített elemeket is. Természetesen kézzel is elkészíthetjük *wsdl* állományainkat, de erre a gyakorlatban általában nincs szükség, ugyanis a fejlettebb fejlesztői környezetek legenerálják számunkra a szükséges WSDL-t.

```
<?xml version="1.0" encoding="UTF-8">
<definitions name="WeatherService"
targetNamesapce="http://www.ecerami.com/wsdl/WeatherService.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.ecerami.com/wsdl/WeatherService.wsdl"
xmlns:tns="http://www.w3.org/2001/XMLSchema">
<message name="getWeatherRequest">
<part name="zipcode" type="xsd:string"/>
</message>
<message name="getWeatherResponse">
            <part name="temperature" type="xsd:int"/>
```

```
</message>
<portType name="Weather_PortType">
            <operation name="getWeather">
                  <input message="tns:getWeatherRequest">
                  <output message="tns:getWeatherResponse"/>
            </operation>
</portType>
<binding name="Weather_Binding" type="tns:Weather_PortType">
<soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getWeather">
      <soap:operation soapAction=""/>
            <input><soap:body encodingStyle= "http://schemas.xmlsoap.org/
soap/encoding/" namespace="urn:examples:weatherservice"use="encoded"/>
            </input>
            <output><soap:body encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" namespace="urn:examples:weatherservice" use="encoded"/>
            </output>
</operation>
</binding>
<service name="Weather_Service">
<documentation>WSDL allomany idojarasjelento szolgaltatashoz
</documentation>
<port binding="tns:Weather_Binding" name="Weather_Port">
<soap:address
location="http://localhost:8080/soap/
servlet/rpcrouter"/>
</port>
</service>
</definitions>
```

Szintén nem egy könnyen átlátható állománnyal van dolgunk, de a főelemeket nem nehéz fellelni. A gyökérelem a *definitions* elem, mely a webszolgáltatás nevének, és a felhasznált névterek megadására szolgál. Bár ebben az állományban nincs *types* nevű elem, de ez az elem szolgálhat arra, hogy benne a kliens és szerver között alkalmazott adattípusokat leírjuk. A *message* elem egyirányú üzenetek definiálására szolgál, míg a *portType* tudja összekötni az egyes *message* elemeinket. A *binding* a szolgáltatás konkrét megvalósításának leírására szolgál, és utolsóként a *service* elem a meghívandó szolgáltatás helyét határozza meg.

UDDI

Az UDDI a webszolgáltatások harmadik eleme, célja, hogy segítségével webszolgáltatásokat tudjunk felkutatni. Lényegében technikai specifikációk publikálására, illetve üzleti alkalmazások, webszolgáltatások megtalálására szolgál.

Az UDDI két részéből áll. Egyik része előírja a webszolgáltatások rendezési módját, természetesen XML formátumban vannak tárolva az adatok. Másik része jelenti az üzleti nyilvántartót. Az UDDI nyilvántartók mindenki számára hozzáférhetőek, bárkinek lehetősége van szolgáltatásainak regisztrálására.

Az UDDI-t úgy tervezték, hogy SOAP üzenetekkel legyen lekérdezhető, valamint hogy hozzáférést biztosítson a WSDL dokumentumokhoz, amelyek a könyvtárban felsorolt webszolgáltatások használatához szükséges protokollkötéseket és üzenetformátumokat írják le.

Az UDDI nyilvántartásában található adatokat és információkat három kategóriába lehet sorolni:

- Fehér oldalak: Alapvető információkat tartalmaz a regisztrált cégről.
- Sárga oldalak: A regisztrált céget, vagy az általa nyújtott szolgáltatást osztályozza.
- Zöld oldalak: Technikai információk a webszolgáltatásról.

Webszolgáltatás szerver oldalon

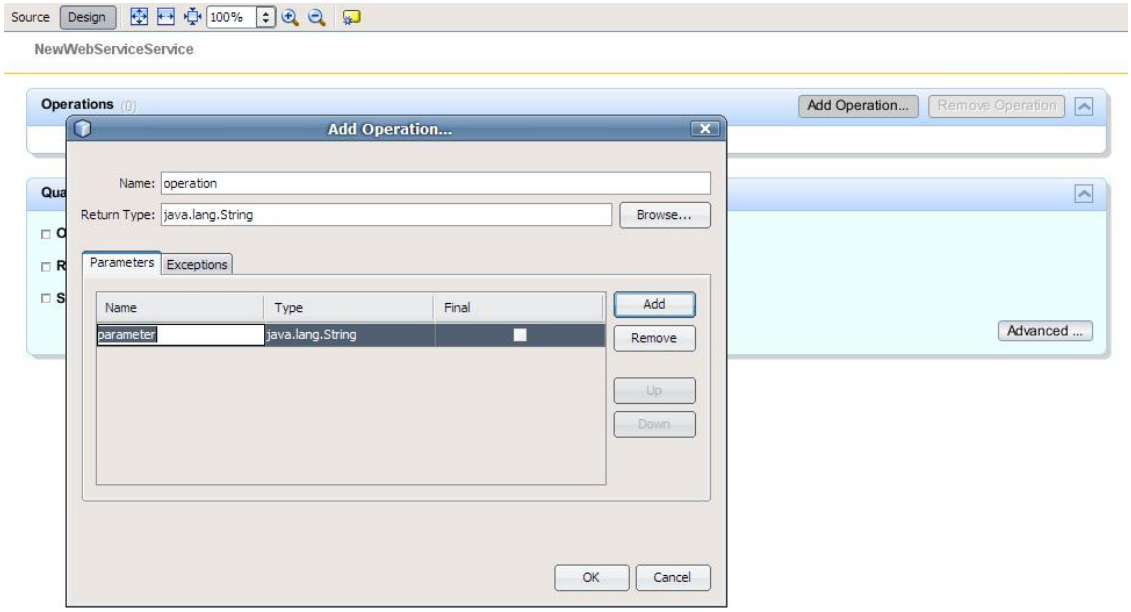
Szerencsékre ma már nem kell hosszas és körülményes gépelésekbe belemerülnünk, manuálisan létrehozni webszolgáltatásunkat leíró *wsdl* állományt, vagy épp a webszolgáltatást nyújtó teljes alkalmazást. Ehhez rengeteg támogatást kapunk a fejlett fejlesztői környezetektől, a wsdl-t legyártják maguktól az alkalmazásnak megfelelően, kezdve a használt egyedi típusoktól, a kötések leírásáig.

Webszolgáltatásom létrehozásához a Netbeans IDE 6.8-at, és Java-t használtam. A webszolgáltatások létrehozásához pedig egy elég jó API áll rendelkezésünkre, méghozzá a *Java API for XML Web Services (JAX-WS).* De nézzük is meg inkább, milyen egyszerű dolgunk van ezen eszközök segítségével.

Webszolgáltatást pár kattintással létre is hozhatunk, új project menüpont alatt a *Web* kategóriát választjuk, majd kiválasztjuk a *Web Application*-t. Ha itt megadtuk a szükséges adatokat, mint a project nevét, a használni kívánt alkalmazásszervert (*Glassfish* ajánlott), akkor a project létre is jött. Ezután a projectben létrehozunk egy új *Web service*-t. Ezzel a Netbeans máris legenerál nekünk egy rövidke kódot.

```
import javax.jws.WebService;

@WebService()
public class NewWebService {

}
```

Ilyenkor azonban még hibát jelez az ablak, mert minden webszolgáltatásnak rendelkeznie kell legalább egy metódussal. Ennek létrehozására nagyon elegáns módon van lehetőségünk. A forrásprogram szövegéről átválthatunk *desing* módba, ahol új metódusokat adhatunk hozzá webszolgáltatásunkhoz, megadva visszatérési értékét és bejövő paraméterek típusát, nevét, de akár még kivételeket is.



Természetesen ezt követően a kód generálás útján előáll, nekünk csak az érdemi részt kell megírni.

```
@WebMethod(operationName = "login")
public boolean login(@WebParam(name = "username")
String username, @WebParam(name = "password")
String password) {
    QueryManager qm = new QueryManager();
    return qm.loginQ(username, password);
}
```

Ha létrehoztuk webszolgáltatásunkat, vagy 1-1 metódusát, lehetőségünk van letesztelni a működését. Feltöltjük az alkalmazásunkat az alkalmazásszerverre, majd ezt követően a project *Web Services* mappájában a tesztelni kívánt webszolgáltatásra kattintva láthatunk *Test Web Service* menüpontot. Ezt követően tudjuk tesztelni a webszolgáltatásunk működését.

A tesztelés folyamata során még a generált SOAP üzeneteket is megtekinthetjük, de *wsdl* állományunk is könnyen elérhető a tesztelő oldalról.

Mint látható, a fejlesztői környezetek valóban széles körű támogatást nyújtanak a webszolgáltatások létrehozásához.

Webszolgáltatás kliens oldalon

A webszolgáltatások egyik nagy előnye a platform és nyelvfüggetlenség. Ez azt jelenti, hogy a Java-ban megírt webszolgáltatásunkhoz bármilyen nyelven megírhatnánk kliensünket. Mi azonban most egyszerűségre törekszünk, így a kliens is Java nyelvű lesz. Netbeans-t használva ez sem egy nagyon nehéz feladat.

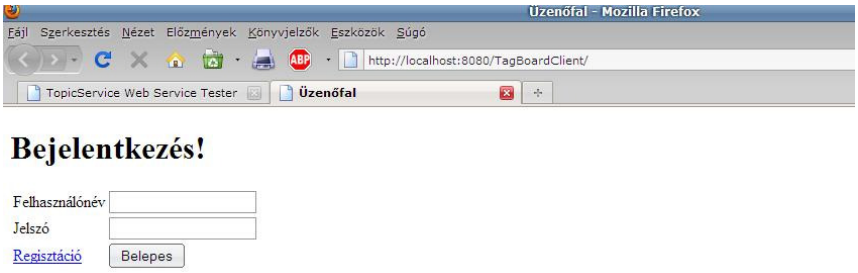
Csakúgy, mint a szerver esetén, új *Web Application* projektet hozunk létre. A projecten belül lehetőségünk nyílik új *Web service client-et* létrehozni. Ez egy Web service referenciát hoz létre a projecten belül, letölti a szükséges *wsdl* állományt, és innentől kezdve tudja hol található a webszolgáltatás, milyen metódusai hívhatóak meg, milyen paraméterekkel.

Amint használni akarjuk a webszolgáltatás egy metódusát a programunkban, egyszerűen megkeressük a navigator ablakban a webszolgáltatás referenciát, és azt kinyitva látjuk annak

metódusait. Ezt követően már csak annyi a dolgunk, hogy a szükséges metódust behúzzuk a használni kívánt pozícióba, és máris kapunk pár sor generált kódot.

```
try { // Call Web Service Operation
            webservice.TagboardWebServiceService service = new
webservice.TagboardWebServiceService();
            webservice.TagboardWebService port =
service.getTagboardWebServicePort();
            // TODO initialize WS operation arguments here
            java.lang.String username = "";
            java.lang.String password = "";
            // TODO process result here
            boolean result = port.login(username, password);
            System.out.println("Result = "+result);
        } catch (Exception ex) {
            // TODO handle custom exceptions here
}
```

Ha megírtuk a kliensünket, a projekt futtatásával az alkalmazás alkalmazásszerverre való feltöltése is megtörténik, majd a webböngészőben már láthatjuk is alkalmazásunk kezdőoldalát.



Amennyiben kliensünket a fent leírt módon hozzuk létre, a szerve-kliens kommunikáció megfelelően fog működni, kényelmesen, számunkra láthatatlanul fognak SOAP üzenetekkel kommunikálni.

Összefoglalás

Dolgozatom megírása elején célomnak azt tűztem ki, hogy az XML-alapú szolgáltatásokról egy viszonylag átfogó képet sikerüljön alkotnom, annak lehetőségeit feltárjam.

Ennek a célnak az elérése érdekében az XML alapjainak megismerésével kezdtem a dolgozatom, amely a további technológiák megértése szempontjából igen fontosnak bizonyult. Sikerült megismernünk az XML alapjait, és sémái közül talán a két legfontosabbat. Megismerkedtünk a nyelv előnyeivel, és ezáltal már itt érezhető volt, hogy egy ilyen szabvány miért is képezi sok technológia alapját.

Egy ilyen előny, az XML dokumentumok felépítése, ami adatok strukturált tárolására alkalmas. Így jutottunk el az XML adatbázis fogalmáig. Az XML adatbázisok ma is terjedő félben vannak. Némely adatbázis XML felülettel rendelkezik, azaz képes XML dokumentumok kezelésére, leképezésére, manipulálására, vagy épp saját sémáján tárolt adatait képes XML formában közzétenni. Ez adatbázisok közt kommunikáció során igen hasznos lehet, nem véletlen, hogy egyre több hagyományos adatbázis integrál valamilyen XML leképező, manipulációs módszert. Ezen kívül egyre több natív XML adatbázis lát napvilágot, amelyek az XML terjedésével fontos szerephez jutnak. Legnagyobb előnyei az ilyen adatbázisnak, hogy megőrzik az XML dokumentum modelljét, így eredeti formában kaphatjuk vissza adatainkat az adatbázisból, valamint használhatjuk az XML család lekérdezőnyelveit, amelyek a velük való munkánk során bizonyították a hatékonyságukat.

Utolsó témakörünket a webszolgáltatások jelentették. Ez egy viszonylag újabb technológia a szerver-kliens kommunikációt megvalósító technológiák között. A webszolgáltatásoknak azért jut kitüntetett szerep, mert XML alapú szabványokra támaszkodik. Ezekkel a technológiákkal (SOAP, WSDL, UDDI) oly módon lehet az alkalmazások közti kommunikációt megvalósítani, hogy független bármilyen programozási nyelvtől, operációs rendszertől, és hardvertől.

Részben sikerként könyvelhetem el az XML-alapú technológiák megismerésére irányuló próbálkozásomat, mert munkám során rengeteg technológiával találkoztam, és ennek egy részét részletesen, vagy kevésbé részletesen, de be is mutattam. Azonban az is kiderült, hogy egy olyan technológiával van dolgunk, ami folyamatosan fejlődik, mindenhol jelen van,

felhasználása rendkívül sokrétű, így egyetlen dolgozatban nem lehet lefedni ezt a témát, nem véletlen foglalkoznak könyvek százai az XML technológiákkal.

Összességben megállapítható, hogy az XML kihagyhatatlan elem lett az web világában, a benne rejlő lehetőségek már-már megkerülhetetlenné tették.

Irodalomjegyzék

Tanuljuk meg az XML használatát 24 óra alatt – Michael Morrison

XML Elmélet és gyakorlat – Chris Bates

XML lépésről lépésre – Michael J. Young

Adatbázisrendszerek: Alapvetés – Jeffrey D. Ullman, Jennifer Widom

Webszolgáltatások: XML alapú kommunikáció az Interneten – Gottdank Tibor

Java alapú webszolgáltatások: XML, SOAP, WSDL, UDDI - Steve Graham - Simeon Simeonov

XML Data Management: Native XML and XML-Enabled Database Systems - Akmal B. Chaudhri, Awais Rashid, Roberto Zicari

XQuery – Priscilla Walmsley

SGML: http://hu.wikipedia.org/wiki/Standard_Generalized_Markup_Language

HTML: http://hu.wikipedia.org/wiki/HTML

XML: http://hu.wikipedia.org/wiki/XML

XML 10 pontban: http://www.w3.org/XML/1999/XML-in-10-points

XML: http://www.xml.com

XML adatbázisok: http://www.rpbourret.com/

SQL/XML: http://www.stylusstudio.com/sqlxml_tutorial.html

UDDI: http://hu.wikipedia.org/wiki/UDDI

SOAP: http://en.wikipedia.org/wiki/SOAP

XML adatbázis: http://en.wikipedia.org/wiki/XML_database