DIPLOMAMUNKA

Kristóf Dávid

Debrecen

2009

Debreceni Egyetem

Informatikai Kar

XML és Adatbázisok

Témavezető:

Jeszenszky Péter

egyetemi adjunktus

Készítette:

Kristóf Dávid

programtervező matematikus

Debrecen

2009

Tartalomjegyzék

1. Bevezetés

Napjainkban az adatok tárolására és továbbítására többféle lehetőség kínálkozik előttünk. Ezek között találhatjuk az XML fájlok használatát is, melynek több előnye van. Könnyen értelmezhető ember és gép számára, szöveges megjelenéséből adódóan pedig egyszerűen bővíthető. Strukturált felépítése és platform-függetlensége miatt lehetőséget biztosít a különböző rendszerek közötti adatátvitel problémamentes megvalósítására. Számos előnye miatt az XML megtalálható az összes platform szinte valamennyi technológiájában, többek között a .NET-ben is használatos. Ezt jól példázza az is, hogy a .NET-es operációs rendszerekben található IIS (Internet Information Services) verziók adatbázisa, valamint a .NET nyelveiben használt konfigurációs fájlok már teljes egészében XML kódokból állnak. A diplomamunkám keretében szeretném bemutatni, hogy az XML állományokat hogyan lehet különböző XML-alapú – főleg XQuery – nyelvek segítségével kezelni. Ehhez szeretnék bemutatni néhány natív XML adatbázis-kezelő rendszert, valamint meg akarom ismertetni az olvasót a .NET technológiába beépíthető eszközök tárházával. Ezen kívül egy alkalmazáson keresztül be akarok mutatni egy lehetséges megoldást arra, hogy a .NET és az XML eszközeinek együttes felhasználásával hogyan oldhatunk meg egy – talán sokak számára felmerülő – problémát.

Az elkészítendő program ötletét egy, a munkahelyemen adódó probléma megoldásának igénye ihlette. A cég központjában naponta legenerálásra kerülnek bizonyos XML állományok, amelyek meghatározzák, hogy melyek azok az ügyfeleink, akiknek elektronikus levelet kell kapniuk az adott napon, és ebből milyen típusút kell küldeni. Ezek a fájlok feltöltésre kerülnek egy FTP szerverre, ahonnan a – mostani megoldás szerint – kézzel kerülnek áttöltésre egy lokális mappába. A munkatársak manuálisan ellenőrzik, hogy minden fájl megérkezett-e és azok szintaktikailag helyesek-e. A file-ok tartalmi ellenőrzésére semmilyen módszert nem alkalmaznak. Az XML állományokat minden nap letöltik, és ezeknek akár több ezer soros tartalmuk is lehet. Így előfordult már olyan eset, hogy egy ügyfél 4-5 példányban is megkapta ugyanazt a levelet az XML állományok generálási hibája miatt.

Erre a problémára, illetve ennek egy módosított változatára szeretnék megoldást nyújtani egy olyan alkalmazással, amely a fájlokat automatikusan, igény esetén a programból manuálisan is képes letölteni, azokat szintaktikailag és tartalmilag ellenőrizni és javítani. Ezek mellett felvetődött annak az igénye, hogy a nagy mennyiségű adatban való keresést is megvalósítsam. Az alkalmazásnak lehetővé kell tennie azt is, hogy külső szoftverek igénybevétele nélkül is képesek legyünk elektronikus levelet küldeni az ügyfeleknek.

2. Az XML technológia

2.1 Az XML-ről általánosan

Az XML megnevezés az „Extensible Markup Language”, vagyis a Kiterjeszthető Jelölőnyelv rövidítése. Az XML az SGML-ből származtatott jelölőnyelv. Az XML az SGML részhalmaza, szigorúbb szintaktikai szabályokkal, ezért a webes környezetben szélesebb körben terjedt el. Az XML-t a W3C fejlesztette ki az 1990-es évek második felében.

Egy XML dokumentum tartalma különböző összetevőkre bontható. Például elemeket, attribútumokat, megjegyzéseket és feldolgozási szabályokat tartalmazhat. Alapelemei az elemek, amelyek egy nyitó (<) és egy záró (>) jelölőelemből, valamint a közöttük elhelyezkedő elem nevéből épülnek fel. Minden elem egy logikai részét jelöli a dokumentumnak, és tartalmazhat további elemeket is. Van egy elem, ami mindegyik másikat tartalmazza, ezt gyökér-elemnek nevezik. Egy elem további információkat is tartalmaz, azáltal, hogy lehet több attribútuma is. Az attribútumok az elemre vonatkozó tulajdonságokat tárolhatják. Minden attribútum tartalmaz egy nevet és – egyenlőségjel megadása után – egy értéket.

Egy XML dokumentumnak jól formázottnak kell lennie, ami azt jelenti, hogy minden megnyitott jelölőelem le kell legyen zárva, az elemek egymásba ágyazhatók, de nem nyúlhatnak át egymáson és minden dokumentum csak egy gyökérelemet tartalmazhat. Az XML dokumentumok érvényesek is lehetnek, de ez nem feltétlenül szükséges.

Érvényes az a dokumentum, amely megfelel a megadott dokumentumtípus definíciójának. A DTD (Document Type Definition – dokumentum-típus definíció) írja le a használható elemtípusokat, az ezekhez tartozó tulajdonságokat, és azt, hogyan lehet az egyes elemeket egymásba ágyazni, vagyis meghatározza, hogy milyen módon kell egy XML dokumentumot felépíteni. Ha tehát egy XML dokumentum meghatároz egy DTD-t és követi az általa felállított szabályokat, akkor azt érvényesnek nevezzük. Azt az ellenőrzést, hogy a megadott dokumentum megegyezik-e az előre meghatározott dokumentumszerkezeti szabályokkal, az feldolgozó végzi. A DTD az SGML öröksége, így képességei is korlátozottak, primitívebb, nem ismeri a hierarchiát, nem XML szintaxist használ a sémák leírására. 2001-ben viszont megjelent az XML-sémaszabvány, amely segítségével

helyettesíthetjük a DTD-ket. Ha egy dokumentum megfelel egy sémának, akkor séma-érvényesnek nevezzük

2.2 Natív XML adatbázis

XML adatbázisról akkor beszélünk, ha XML adatok egy gyűjteményét egy – például relációs – adatbázisban tároljuk. Az XML adatbázis használatának előnye, hogy kihasználhatjuk az adatbázisok által nyújtott szolgáltatásokat: többek között megengedett a tranzakciók, zárolások, indexek, adat-manipulációs utasítások, API-k használata, adatok távoli elérése.

Egy XML adatbázist natívnak nevezünk, ha teljesíti a következő kritériumokat:

- Definiál egy modellt az XML dokumentum számára, és ennek a modellnek megfelelően tárolja és keresi vissza az adatokat. A modellnek tartalmaznia kell elemeket, attribútumokat, PCDATA-t és azok sorrendjét.
- A tárolás alapvető egysége a modellben meghatározottak alapján történik. Például egy relációs adatbázisban az adatokat tartalmazó sorokat egy táblában tároljuk.
- Nem szükséges semmilyen speciális fizikai tárolási modellt. Épülhet relációs, hierarchikus vagy objektum-orientált adatbázisra.

Ez a modell támogatja a tetszőleges mélységű beágyazást, a vegyes adattartalmat és a szemi-strukturált adatokat. A – széles körben elfogadott – natív XML adatbázisrendszerek definíciója nem zárja ki a különböző technológiák használatát. Például relációs adatbázison alapuló megvalósítás esetén az SQL alkalmazását is lehetővé teszi, ugyanis az SQL a tárolt adatokat, mint XML tartalmat fel tudja dolgozni.

Egy natív XML adatbázis képes egyszerre több XML dokumentumot kezelni, lehetővé téve, hogy lekérdezhetők, módosíthatók legyenek, mint a relációs adatbázisban egy tábla sorai.

2.3 XML adatbázisokból való lekérdezés

Az XML alkalmazásoknál fontos, hogy egy XML dokumentumban tudjunk lekérdezni, navigálni. Erre két módszert szeretnék ismertetni.

2.3.1 XPath

Az XML Path nyelvet először 1999-ben publikálta a W3C, az XSLT és XPointer definiálásának eredményeképpen. Az XPath kifejezéseket használ dokumentumokban való keresésre és kapcsolódások kialakítására. Az XPath különböző csomópontokból álló faként kezeli az adott XML-dokumentumot. Az XPath segítségével képesek vagyunk a csomópontok elérési útvonalát meghatározni típusuk, nevük, értékeik, valamint a csomópontoknak az egymáshoz viszonyított kapcsolatuk alapján.

Az XPath a kereséshez több féle lehetőséget kínál számunkra, ezek a következők lehetnek:

- Kereshetünk egyetlen csomópontra, amely a következő típusú lehet:
  - Gyökér csomópont
  - Elem csomópont
  - Attribútum csomópont
  - Névtér csomópont
  - Feldolgozási utasítás csomópont
  - Megjegyzés csomópont
- Egy csomópontra kereshetünk relatív elérési út megadásával.
- Egy egész szám megadásával, amely a csomópontot azonosítja a fában elfoglalt helye (mélysége), illetve mérete (gyermekeinek száma) alapján.
- Az XPath nyelvbe beépített függvényeket is segítségül hívhatjuk.

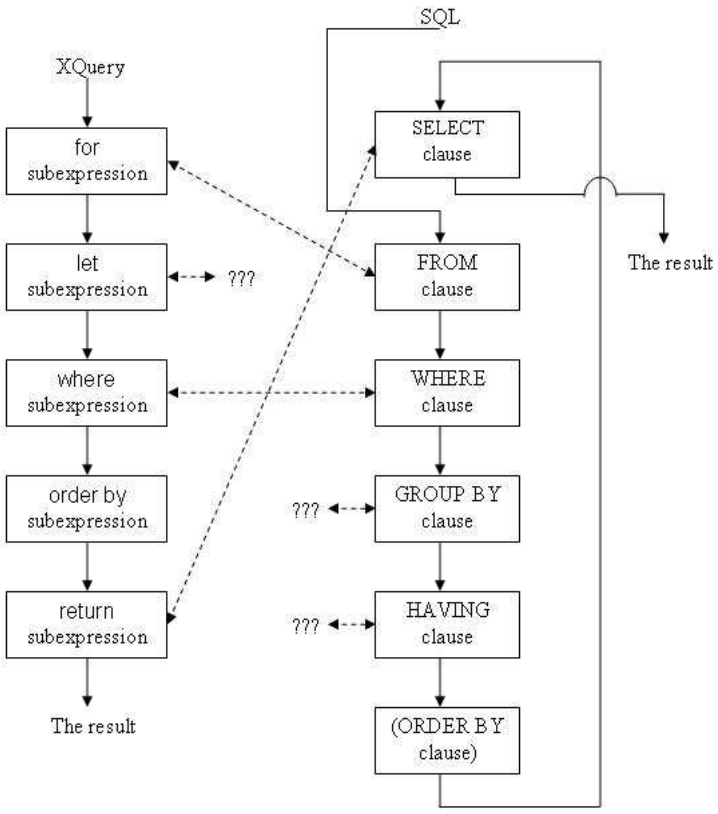
2.3.2 XQuery

Az XQuery, amely egy deklaratív, típusos, funkcionális nyelv, 2001-ban jelent meg. Elsődleges kifejezései a változók és a konstansok. A változók nevei $ karakterrel kell, hogy kezdődjenek, a konstansok numerikus értékek és sztringek lehetnek. Az XQuery-ben a legáltalánosabb kifejezésforma az ún. FLWOR kifejezés. A FLWOR kulcsszó egyes karakterei a következőt jelentik:

- *FOR*: iteráció a megadott csomópontokon. A *FOR* kulcsszó után meg kell adnunk egy változót, amely az utána következő útvonalon található csomópontokon halad végig. A változó megadása után lehetőségünk nyílik egy ún. helyzeti érték meghatározására is, amely egy pozitív egész szám 1-től kezdődően. Ezt az AT szó után adhatjuk meg.

- `LET`: egy változónak adhatunk vele értéket.
- `WHERE`: feltételek megadása, ezzel a lekérdezett csomópontokon végezhetünk egy szűrést.
- `ORDER BY`: az eredményhalmaz sorba rendezésére szolgál, a megadott változó szerint.
- `RETURN`: a lekérdezés eredményét szolgáltatja a megadott formátumban.

Az `FLWOR` kifejezések hasonló felépítésűek, mint az SQL nyelvben felhasznált kifejezések. Ezt a megfeleltetést az 1-es ábrán láthatjuk.



1. ábra: Az FLWOR és a SELECT közötti kapcsolat

A képen a tömör vonalak a lekérdezési haladási irányt, a szaggatott vonalak a nyelvekben található kifejezések megfelelését, míg a kérdőjelek („???”) azt jelölik, hogy az adott nyelvnek nincs a másik nyelvben megfelelője.

Mivel az XML adatok felvitelére, törlésére és módosítására még nem volt megfelelő eszköz, így az XQuery létrehozásakor felmerült az igény, hogy az erre képes legyen. Ezt a következő módszerekkel érhetjük el:

- `insert`: Egy XQuery kifejezés segítségével csomópontokat illeszthetünk be az XML dokumentumba. Ezen kívül megadhatunk neki különböző kulcsszavakat a beillesztendő új csomópont pozíciójára vonatkozóan.
- `delete`: Ennek segítségével törölni tudjuk az XQuery által visszaadott elemeket.
- `replace value of`: Ezzel lehetőségünk nyílik arra, hogy egy létező csomópont értékét módosítsuk. A módosításhoz az `of` kulcsszó után egy XQuery kifejezéssel tudatjuk, hogy az XML dokumentum melyik elemének, vagy attribútumának értékét szeretnénk módosítani, majd utána a `with` kulcsszó megadását követően a helyettesíteni kívánt értéket adjuk meg. A változtatás az összes olyan csomópontra végrehajtódik, amelyre teljesül a megadott XQuery kifejezés.

2.4 Adatbázis-kezelő rendszerek XML támogatása

A mai adatbázis-kezelő rendszerek törekednek az XML tárolás és feldolgozás natív támogatására. Két adatbázis-kezelő rendszerben való XML támogatást szeretnék bemutatni.

2.4.1 Oracle XML DB

Az XML adatok kezelésére az Oracle egy új komponenst fejlesztett ki, az Oracle XML DB-t. Ebben az XML adatok tárolására szolgáló típus az `XMLType`. Ez ugyanúgy használható, mint bármely más típus az Oracle-ben.

Az Oracle számos függvényt tartalmaz az XML adatok lekérdezésére. Ezek a következők:

- `XMLELEMENT`: Egy elemet kereshetünk meg vele, ha megadjuk a nevét és igény szerint az értékének részletét vagy egészét.
- `XMLATTRIBUTES`: Egy elem egy vagy több attribútumát tudjuk lekérdezni, az attribútum(ok) nevének és értékek listájának megadásával. Az `XMLELEMENT` függvénybe is beágyazhatjuk.
- `XMLQUERY`: A függvénynek paraméterül kell adnunk egy végrehajtandó XQuery lekérdezést.
- `XMLTABLE`: Szintén egy XQuery lekérdezést hajthatunk végre a segítségével, a különbség annyi, hogy ez egy SQL táblával azonos szerkezetű eredményt ad vissza.

- XMLEXISTS: Egy XQuery lekérdezést adunk meg, és eredményül egy logikai értéket kapunk, attól függően, hogy a lekérdezés visszaadott-e elemet, vagy sem.

2.4.2 Microsoft SQL Server

Az MS SQL Server az XML típusú adatokat XML típusú változóban képes tárolni. Az összes verzió támogatja a tetszőleges típusú adatok XML formában történő lekérdezését. Ezt egyszerűen úgy tehetjük meg, hogy a lekérdezéshez hozzáfűzzük a FOR XML záradékot. Ennek megadhatunk továbbá különböző opciókat is, az elkészült struktúrára vonatkozóan. Ezek a következőek lehetnek:

- RAW: Az összes elem neve „row" lesz, amelyen belül az attribútumok nevei a lekérdezett tábla oszlopainak nevei lesznek.
- AUTO: A szülő elem neve a tábla, gyermekeinek neve a lekérdezett oszlopok nevei lesznek.
- EXPLICIT: Ekkor mi határozhatjuk meg a XML felépítését úgy, hogy minden visszaadott sor tartalmaz két számot: egy azonosító számot, amellyel egy szülő gyermekeit azonosítjuk, valamint a szülő azonosítóját.
- PATH: Ebben az esetben is mi adjuk meg az XML struktúráját, azzal a különbséggel, hogy itt egy teljes elérési útvonalat adunk meg minden elemhez. Ennek hiányában minden lekérdezett sort egyenrangúnak tekint, és a „row" (vagy a megadott nevű) szülő gyermekeként adja vissza azokat.

Alapértelmezetten mindegyik struktúra úgy épül fel, hogy az adott oszlopok rekordjai a tábla nevű elemek attribútumaként jelennek meg. Ezen az ELEMENTS kulcsszó megadásával változtathatunk, amellyel elérhetjük, hogy minden rekord egy új, az oszlop nevű elem gyermekeként jelenjen meg. Ezen kívül többek között megadhatjuk még neki a használandó sémát (SCHEMA) és akár a gyökérkönyvtár nevét is (ROOT).

Egy XML fájl tartalmát az OPENROWSET függvény használatával vagyunk képesek eltárolni az adatbázisban. Ekkor a BULK kulcsszó után meg kell adnunk a dokumentum elérési útvonalát. Megadhatjuk neki továbbá a fájl formátumát is. Erre azért van szükség, hogy meghatározzuk a visszaadandó oszlop típusát. Ez a paraméter egy külső formátumfájlra mutathat, vagy értékül adhatjuk a SINGLE_BLOB, SINGLE_CLOB, vagy SINGLE_NCLOB típust – ilyenkor nincs szükség külön fájl megadására. Ezek között a különbség a fájl

kódolása. Ajánlott a `SINGLE_BLOB` használata az XML adatok tárolására, ugyanis ez a típus támogatja az összes Windows kódolási konverziót.

Az XML adatok megjelenítése és tárolása az `OPENXML` függvény használatával lehetséges. Ez egy változóból beolvasott XML adatokat tudja feldolgozni. Emellett megadhatjuk neki az XML adaton belüli elérési útvonalat, illetve azt is, hogy azon belül az elemeket, attribútumokat vagy csak az értékeket jelenítse meg.

A Microsoft SQL Server támogatja a táblában tárolt XML típusú változókon való lekérdezéseket. Ezeket a következő függvények használatával tehetjük meg:

- `query()`: Egy XQuery lekérdezést értékel ki, visszaadva a megfelelő csomópontokat. Lehetőséget nyújt a felhasználónak, hogy egy XML dokumentumból részeket vegyen ki.
- `value()`: Egy XML dokumentum csomópontjának értékét kaphatjuk vissza valamilyen SQL típusként.
- `exist()`: Egy ellenőrzést hajthatunk vele végre, hogy a megadott XQuery lekérdezésnek van-e eredménye, vagy sem. A visszatérési érték 1, 0 vagy `NULL` lehet.
- `nodes()`: Segítségével lehetőségünk van az XML dokumentumot SQL adatokra bontani oly módon, hogy a paraméterként megadott XQuery lekérdezés által visszaadott csomópontok mindegyike egy SQL sorként térjen vissza.
- `modify()`: Ezzel a függvénnyel az XML dokumentumba vihetünk be adatokat, törölhetjük és módosíthatjuk azokat. Ezt az XQuery által nyújtott insert, `delete` és `replace value of` kifejezésekkel tehetjük meg.

Az SQL Server használatával kapcsolatban meg kell jegyeznünk, hogy a 2005-s verzióban nincs lehetőségünk a let záradék használatára, valamint függvények írására sem. A 2008-s verzióban is csak az XQuery beépített függvényeit használhatjuk, viszont a let záradék már megjelent benne.

3. Az elkészült szoftver

3.1 Technológiai háttér

3.1.1 A keretrendszer

Az alkalmazás Microsoft .NET keretrendszerben készítem el, C# nyelven.

3.1.2 A fejlesztői környezet

Az alkalmazás elkészítése Visual Studio 2005 alatt történik, Windows Formokat használva. Az XML adatokból nyert, és eltárolt adatbázis kezeléséhez Microsoft SQL Server 2005 használatos.

3.1.3 További fejlesztői eszközök

Az XML adatokon végzett műveleteket, lekérdezéseket XQuery nyelven hajtom végre, a következő módszerrel:

- A lekérdezések megírásához Microsoft SQL Server 2005-t használok, mely támogatja az XML alapú lekérdező nyelvek használatát, többek között az XQuery-t is. Ezt XML adattípus deklarálásával érhetjük el.

Az XQuery használatához más módszerek is adottak:

- A .NET keretrendszer beépített moduljait, valamint a `Microsoft.Xml.XQuery` névteret használva (külső hivatkozás megadása).
- Más cégtől származó beépíthető modulok, szoftverek, névterek alkalmazása.

Ezeket a módszereket a diplomamunkában részletesebb elemzem, kipróbálom.

3.1.4 Hardver és szoftver követelmények

Mivel az alkalmazás adatbázis-kezelő része MS SQL-ben lett megírva, így a minimális követelmény a Microsoft SQL Server 2005 Workgroup Edition megléte. Erre azért van szükség, mert az alkalmazás megfelelő használatához elengedhetetlen az SQL Server Agent megléte is, amely ebben a változatban található meg először. A C#.NET-ben megírt kód miatt a program futtatásához .NET 2.0-ás Framework szükséges. A Microsoft hivatalos honlapja alapján az ehhez szükséges minimális hardver és szoftver követelmények az alábbiként

vannak feltüntetve: 600 MHz processzor, 512 MB RAM, Windows 2000 SP4 operációs rendszer.

3.2 Általános ismertetés

Adott 5 darab XML állomány egy tárhelyen. A fájlokra példát a függelékben láthatunk. Ezek a fájlok azon ügyfeleket listáját tartalmazzák, akik valamilyen elektronikus levelet kell hogy kapjanak az adott napon. Az XML állományok naponta érkeznek, és nagy mennyiségű adatot tartalmazhatnak, ezért többek között fontos az adatok visszakereshetősége. Vannak bizonyos szabályok, amelyek meghatározzák, hogy egy személy mikor és milyen levelet kaphat. Ha valaki nem felel meg ezen kritériumoknak, akkor szükség lehet adatainak módosítására, így a felületről történő adatváltoztatást is lehetővé kell tenni. Előfordulhat az is, hogy az XML állományok szintaktikailag hibásak, ilyenkor nem lehet őket használni, és a problémák javítása nehézkes. Erre is megoldást nyújt az automatizált XML feldolgozó rendszer kialakítása.
A kialakítandó rendszer céljai a következők:

- A napi rendszerességgel érkező adatok betöltése egy adatbázisba és azok ellenőrzése.
- Az XML állományok szintaktikai ellenőrzése.
- Visszakereshetőség biztosítása.
- Elektronikus levelek elküldése manuálisan.

3.3 A program első indítása

Az elkészült alkalmazás első indításakor több folyamat fut le. Mielőtt ezek végrehajtódnának, a képernyőn megjelenik egy ablak, a következő szöveggel:
„Az XML fájlok most le fognak töltődni a szerverről. Ez akár egy percig is eltarthat! Minden indításkor letöltődjenek?”. Ekkor kiválaszthatjuk, hogy az adatokat a program minden futtatásakor szeretnénk-e a szerverről áthelyezni a lokális mappánkba, vagy sem. A választ egy konfigurációs fájlban tároljuk, ahonnan minden induláskor kiolvassuk ennek értékét, és ettől függően töltődnek le a fájlok.

Ezek után megjelenik egy ablak, rajta egy, a műveletek állapotát jelző ProgressBar-ral. Ez folyamatosan mutatja, hogy a műveletek mekkora része fejeződött be. Eközben a program a következő pontokban leírt folyamatokat hajtja végre.

3.3.1 Adatbázis táblák és tárolt eljárások létrehozása

A program első indulásakor a következő SQL utasítások hajtódnak végre:

- Egy `XMLDB` nevű adatbázis létrehozása az adatok tárolására.
- Az `XMLDB` adatbázisban a következő táblák létrehozása:
  - `XMLDocs` tábla, amelyben az összes eddigi adat szerepelni fog.
  - `ActXMLDocs`, amely az aktuális napi adatokat tartalmazza.
  - `TiltottXMLDocs`, amely azon személyek adatait tartalmazza, akik a tiltólistán szerepelnek, vagyis nem kaphatnak levelet.

  A táblák mindegyike a következő szerkezettel rendelkezik:
    - Egy azonosító ID mezővel, amely automatikusan generálódik minden rekordhoz.
    - Egy XML típusú mezővel, amely az adatokat tartalmazza.

  A táblában egy rekord egy XML fájlnak felel meg. Vagyis egy fájl tartalmát egy sorban tároljuk.
- Ezután létrejönnek azon tárolt eljárások, amelyek az adatbázis megfelelő tábláiba töltik az adatokat.

  Ez az eljárás a következőképpen néz ki az `XMLDocs` tábla esetén:

```
1. CREATE TABLE #temp (TempXMLData XML)
2. DECLARE @cv int, @today int, @insert nvarchar(max)
3. SET @cv = 1
4. WHILE @cv <= 5
5. BEGIN
6.    TRUNCATE TABLE #temp
7.    SET @insert = 'SELECT bulkcolumn FROM OPENROWSET (BULK
            ''Elérési_Útvonal\XMLFiles\XML' + CONVERT (varchar, @cv)
            + '.xml'', SINGLE_CLOB) AS TempXMLData'
8.    INSERT INTO #temp
9.    EXEC (@insert)
10.   SELECT @today = ref.value('.', 'int')
11.   FROM #temp CROSS APPLY
            TempXMLData.nodes ('/ROWS/HEADER_SECTION/CREATION_DATE')
            AS T(ref)
```

```
12.   IF (SELECT [XMLData].exist
            (' /ROWS/HEADER_SECTION[CREATION_DATE =
            sql:variable("@today")
            and XML_ID = sql:variable("@cv")]')
13.   FROM dbo.XMLDocs) = 1
14.   SET @insert = 'INSERT INTO XMLDocs ([XMLData]) SELECT * FROM
            OPENROWSET (BULK ''Elérési_Útvonal\XMLFiles\XML'
            + CONVERT (varchar, @cv) + '.xml'', SINGLE_BLOB)
            AS [XMLData]'
15.   EXECUTE (@insert)
16.   SET @cv = @cv + 1
17. END
18. DROP TABLE #temp
```

Első lépésként létrehozok egy ideiglenes táblát, amely XML típusú mezőjében ideiglenesen eltároljuk a fájlok tartalmát.
Ezek után deklarálok néhány változót:

- `@cv` – egész szám típusú változó, amely az eljárásban található ciklus változója.
- `@today` – egész szám típusú változó, amely a fájlokban található dátumot tárolja, amelynek a mai dátummal kell egyenlőnek lennie. Ezt ki lehetne számolni a `getdate()` függvény segítségével is, de mivel nem garantált, hogy valóban az adott nap adatait tartalmazza a fájl, – hanem akár korábbi adatok is lehetnek benne – így a `getdate()`-et használva az ellenőrzés során problémák merülnének fel, amit ezzel a módszerrel el tudunk kerülni.
- `@insert` – szöveges változó, amely egy `INSERT` parancsot tárol.

Beállítom a ciklusváltozó értékét 1-re, majd elindítok egy ciklust, amíg ezen változó értéke nem lesz egyenlő 5-tel. Ezen ciklus segítségével lépkedek végig az XML fájlokon.
A ciklus első lépéseként kiürítem az ideiglenes táblámat. Ezt minden fájl beolvasása előtt meg kell tennem.

Ezek után beállítom az `@insert` változó értékét úgy, hogy az ciklusváltozó alapján meghatározott XML fájlt bemásolja a `#temp` táblába. Ezt a parancsot azért kell szöveges változóban tárolni, mert az `OPENROWSET` parancsban a fájl elérési útvonalát és nevét egyetlen szöveges értékkel kell meghatározni. Mivel a fájl neve a ciklus minden lefutásakor más, ezért nem lehetséges egyetlen mezővel magadni, csak az elérési útvonalhoz konkatenálva a fájl nevét. Így az egész parancsot egyetlen szöveges mezőben tárolva, majd

ezt az EXEC parancsnak átadva le tudjuk futtatni az adatok felvitelére szolgáló parancsot . A táblába szúrandó sorok meghatározása a következőképpen történik:

Meghívom az OPENROWSET függvényt, a BULK operátorral, ezáltal elősegítve a nagy mennyiségű szöveges adatok bevitelét fájlból egy SQL Server táblába. A szöveges fájl jelen esetben az XML fájljaink közül lesz valamelyik. A fájl elérési útvonalát a programban úgy adjuk meg, hogy lekérdezzük a program aktuális útvonalát, majd hozzákonkatenáljuk az XMLFiles almappát, és azon belül a fájl nevét. Az OPENROWSET-nek még paraméterül adhatjuk a fájl formátumát, ami jelen esetben SINGLE_BLOB. Ezek után az aktuálisan vizsgált fájlból – az ideiglenes táblán keresztül – meghatározzuk a betöltendő adatok dátumát. Ezt a CROSS APPLY kulcsszó megadásával tehetjük meg, amely a táblák és tábla értékű függvények összekapcsolására szolgál. A tábla értékű függvényünk most egy XQuery utasítás, amely a következőt csinálja:

- A #temp táblánk XML típusú mezőjére (TempXMLData) meghívjuk a nodes nevű XQuery-s parancsot, melynek paramétere az XML dokumentumban szereplő azon elem, amelynek értékét meg szeretnénk kapni – jelen esetben a létrehozás dátuma (CREATION_DATE).
- Ezt a value parancs segítségével egy egész szám típusú értékként kiolvassuk, majd eltároljuk a @today nevű SQL változónkban.

Ekkor meghatározunk egy feltételt, amelynek megvalósítja, hogy csak akkor töltsük be az adatbázisba egy fájl tartalmát, ha az adott dátummal még nem szerepel az adott azonosítójú adat. Ez az azonosító (XML_ID) a fájl sorszámát tartalmazza, vagyis egy egész számot 1 és 5 között. Ezt a feltételt úgy tudjuk megvizsgálni, ha a tábla XML típusú mezőjére (XMLData) meghívjuk az exist nevű parancsot, amelyben egy XQuery lekérdezést hajtunk végre. Ez a lekérdezés visszaadja, hogy az adatbázisban van-e olyan dátummal és XML_ID-val rendelkező adat, mint amilyen a fájlban szerepel. Az XML_ID értékét a ciklus változója adja, ez pedig az éppen vizsgált fájl sorszáma. Ezen külső SQL változókat az XQuery lekérdezésnek az sql:variable(„változó_neve”) módszerrel adhatjuk át.

Ha tehát nem kapunk vissza olyan sort, amely megegyezne a fájl tartalmával, akkor azt be tudjuk tölteni az adatbázisunkba. Ez hasonlóképpen történik, mint a létrehozás dátumának lekérdezése, vagyis egy változóban tárolt `INSERT` utasítás végrehajtásával. Az egyetlen különbség most az, hogy nem a ideiglenes táblánkba töltjük az adatokat, hanem a ténylegesen létező `XMLDocs` táblába.

Amikor a ciklus lefutott, vagyis megvizsgálta és szükség esetén betöltötte az adatokat a megfelelő táblába, akkor töröljük az ideiglenes táblánkat.

Az adott napi adatokat tartalmazó táblához (`ActXMLDocs`) tartozó tárolt eljárás a következőképpen módosul: Létre kell hozni egy változót, amelyik azt fogja jelölni, hogy az adott tábla adatai törölve lettek-e a mai napon vagy sem. A változó értéke alapértelmezetten hamis. Az adatok betöltése (vagyis az `@insert` változóban tárolt parancs végrehajtása) előtt megvizsgáljuk, hogy a változó milyen értékű. Ha hamis, és van adat, amit be kell töltenünk, akkor törli a táblában lévő adatokat, majd a változó értékét igazra állítja. Ha a változó értéke igaz, akkor nem kell csinálni semmit.

A tiltott ügyfeleket tartalmazó tábla esetén a különbség annyi az `XMLDocs` táblával szemben, hogy itt nem kell ciklust létrehoznunk, mivel itt csak egyetlen fájl kell megvizsgálnunk. Ennek következményeképpen az `OPENROWSET` parancsnak át tudjuk adni a fájl elérési útvonalát és nevét, mivel az statikus. Ekkor az `@insert` változóra sincs szükségünk, mivel az `INSERT` utasítás is végrehajtható a megszokott módon.

3.3.2 Az adatok le- és betöltése

Miután létrehoztuk a tárolt eljárásokat, biztosítanunk kell, hogy ezek a megfelelőképpen, a jó időben le is fussanak. Ezért a program a következő lépésként létrehoz egy SQL Server Agent szerviz alatt futtatandó Job-ot, a következő lépésekkel:

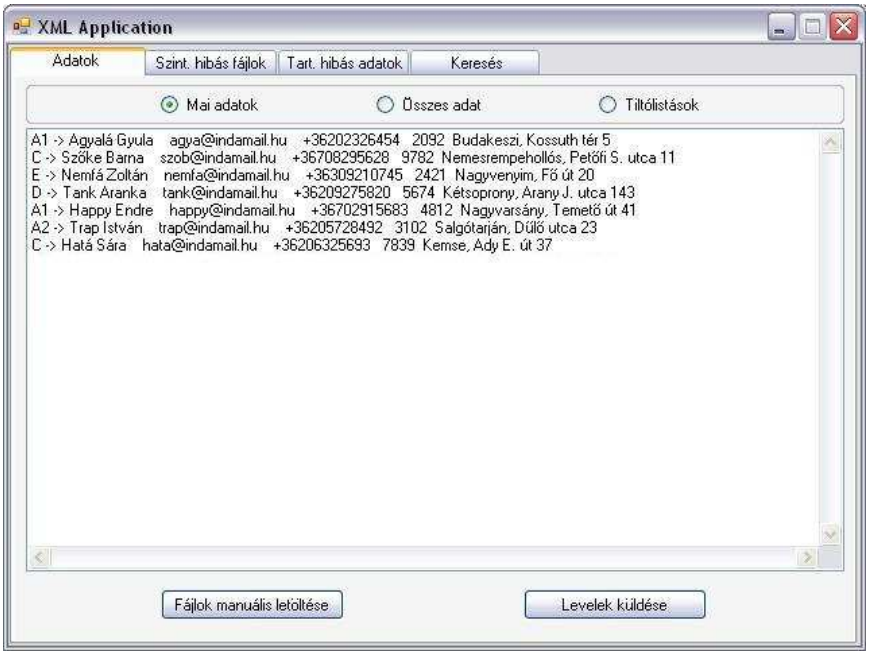
- Lefuttat egy SQL Server Integration Services (SSIS) feladatot, amely a következőket csinálja:
    - Létrehozza az `XMLFiles` mappát az alkalmazás mappájában. Ha a mappa már létezik, az abban lévő adatok automatikusan törlődnek.
    - Kapcsolódik az FTP szerverre.
    - Letölti a szerveren található XML fájlokat a saját gépünkre.
- Ha az SSIS csomag hiba nélkül futott le, a következő lépés a tárolt eljárások meghívása. Ezzel a fájlok tartalmát betöltjük a megfelelő adatbázisokba.

Ha valamiért nem futna le a fájlok letöltése, az adatbázisba való betöltés lépését automatikusan kihagyjuk, és az SQL Server Agent hibával tér vissza.
A fájlok le- és betöltése úgy lett beállítva, hogy minden hétköznap reggel 8 és este 8 óra között automatikusan fusson le.

A program első indítása közben ezek a műveletek hajtódnak végre. Természetesen bármilyen hiba esetén az alkalmazás a megfelelő üzenet kiíratása után befejezi működését. Ha minden végrehajtódott, a műveletet jelző ablak eltűnik, és megjelenik a program első ablaka.

3.4 Az adatok megjelenítése

Miután az alkalmazás elindult, és betöltötte az adatbázisba az aktuális fájlokat, a programnak meg kell jelenítenie az adatbázisban szereplő adatokat. Ez a 2. ábrán látható ablakban történik meg:



2. ábra: az ügyfelek listáját tartalmazó ablak

Ez az ablak egyben az alkalmazás fő ablaka is. Mint láthatjuk a program egyes funkciói külön fülekre kerültek. Ezek közül a köszöntő képernyő az adatok megjelenítésére szolgál. Az adatokat egy szöveges mezőben jelenítem meg, ahol egy személyhez tartozó információk a következők:

- A levél típusa.
- Az ügyfél neve.
- Az ügyfél elektronikus levél címe.
- Az ügyfél telefonszáma.
- Az ügyfél irányítószáma.
- Az ügyfél lakhelye (város, utca, házszám).

A levelek típusának jelentősségét a „Tartalmilag hibás adatok” fülnél részletezem (2.6-os pont).

Elöljáróban annyit, hogy a levél többféle típusú lehet: A1, A2, B, C, D, E.

3.4.1 Az adatok leválogatása

Mint láthatjuk, az adatbázisban lévő adatokat ki tudjuk válogatni aszerint, hogy melyik táblában találhatóak, melyik adat érdekel minket. Ez lehet az aktuális napi adatok (`ActXMLDocs`), az összes eddigi adat (`XMLDocs`), vagy a tiltólistán szereplő személyekre vonatkozó információk (`TiltottXMLDocs`).

Az adatok lekérdezését a táblákból a következő eljárás hajtja végre:

```
1. SELECT
2.    ref.value('./@Template_Name', 'varchar(2)') AS [template_name],
3.    ref.value('./@Surname', 'varchar(30)') AS [surname],
4.    ref.value('./@Forename', 'varchar(30)') AS [forename],
5.    ref.value('./@Postcode', 'int') AS [postcode],
6.    ref.value('./@Town', 'varchar(40)') AS [town],
7.    ref.value('./@Street', 'varchar(40)') AS [street],
8.    ref.value('./@House_Number', 'int') AS [house_number],
9.    ref.value('./@Phone_Number', 'varchar(15)') AS [phone_number],
10.   ref.value('./@EMail', 'varchar(40)') AS [email]
11. FROM @Table CROSS APPLY
XMLData.nodes('/ROWS/LETTER_SECTION/LETTER_DETAILS') AS T(ref)
```

A lekérdezés a következőképpen zajlik: a `@Table` változóban tárolódik az a tábla, amelyikből le szeretnénk kérdezni az adatokat. Ez az `XMLDocs`, az `ActXMLDocs` és a `TiltottXMLDocs` táblák valamelyike lehet. Ezen a táblán keresztül, az XML adatokat tartalmazó (`XMLData`) mezőjéből lekérdezzük a `ROWS/LETTER_SECTION/LETTER_DETAILS` elem attribútumainak értékét. Az attribútumok nevét `@` jellel kell kezdeni az elemre hivatkozó `value()` függvényen belül, majd meg kell adni a visszatérési értékének típusát. Így kaphatjuk meg a számunkra fontos adatokat, amelyeket csak meg kell jelenítenünk.

3.4.2 Fájlok manuális letöltése

Az alkalmazásban lehetőség nyílik arra, hogy a fájlokat ne csak automatikusan, óránként töltsük le, hanem bármikor, kézzel is. Ezt nem az SQL Server Agent-ben létrehozott feladat meghívásával végezzük, hanem C# kódból, a System.Net névtér FTP műveletekre alkalmas metódusai segítségével. Ezzel képesek vagyunk a letöltés sebességét nagymértékben növelni, csökkentve ezáltal a letöltésre fordított időt. Az automatikus letöltésnél azért nem ezt a módszert alkalmazzuk, mert annál nem biztos, hogy szempont a letöltésre fordított idő. Az SQL Server Agent Job-jának létrehozására még azért is volt szükség, mert a C#-ban írt kóddal csak futó program esetén tudunk fájlokat letölteni a szerverről.

3.4.3 Elektronikus levelek küldése

Amennyiben az adott napra az összes állomány megérkezett, és azok mind szintaktikailag, mind tartalmilag rendben vannak, akkor (illetve amennyiben a felhasználó úgy dönt) a Levelek küldése funkció segítségével az alkalmazás automatikusan elküldi az elektronikus leveleket a címzetteknek.

Amennyiben vagy szintaktikai, vagy tartalmi hiba van a fájlokban, adatokban, akkor egy hibaüzenetet küld a program, amely felhívja a figyelmünket, hogy hibás adatokkal nem küld levelet. (A hibák okáról és javítási lehetőségeiről bővebben a 2.5. és 2.6. pontban írok.) Ezen hibák kijavítása után lehetőség van a gomb ismételt megnyomására, ezáltal a levelek küldésére.

Ha először próbáljuk kiküldeni a leveleinket, akkor feljön egy ablak, amelyen megadhatjuk a levelezésre használandó SMTP szerverünk nevét, port számát, valamint a saját felhasználói nevünket és jelszavunkat. A program ezután megpróbálja elküldeni az

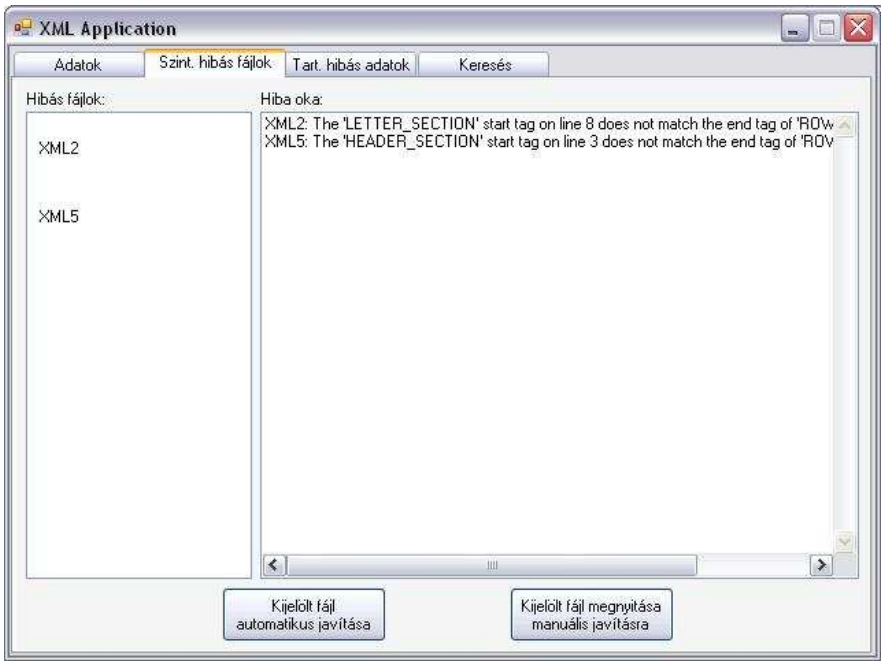
elektronikus leveleket. Hiba esetén – ha nem, vagy rosszul adtunk meg SMTP szerver tulajdonságait – a megfelelő üzenet kiírásával visszatérünk a kezdő ablakhoz.

3.5 Adatok szintaktikai vizsgálata

A letöltött fájloknak szintaktikailag helyesnek kell lenniük, ugyanis a szintaktikailag helytelen fájlok megakadályoznák a program megfelelő működését. Ezért meg kell találni a nem megfelelő szerkezetű fájlokat, és ezek javításának lehetőségét biztosítani kell a felhasználó számára.

Egy XML fájl szintaktikailag hibásnak tekintek, ha a szerkezete nem felel meg a függelékben található XSD sémával.

Az alkalmazás a saját mappájában lévő, vagyis a szerverről letöltött fájlokat vizsgálja. A nem megfelelő szerkezetű fájlokat a 3. ábrán látható módon jelenítjük meg:



3. ábra: szintaktikailag hibás adatokat megjelenítő ablak

Az ablak a következőképpen épül fel: a bal oldalon található azon fájlok neve, amelyek szintaktikailag helytelenek, a jobb oldalon pedig egy rövid leírás arra vonatkozóan, hogy mi a probléma oka.

Ezen kívül láthatunk két gombot, a hiba javításának megoldási technikájára vonatkozóan.

A fájlok szintaktikai hibáját a .NET keretrendszer `System.Xml` névterének segítségével ismeri fel a program. A kód röviden a következőkből áll:

Létrehozom az XML-t olvasó objektum tulajdonságait.

```
XmlReaderSettings settings = new XmlReaderSettings();
```

Majd beállítom, hogy séma alapján vizsgálja a fájlokat. Itt be lehetne azt is állítani, hogy DTD alapján vizsgáljon.

```
settings.ValidationType = ValidationType.Schema;
```

Ezek után megadom neki a séma elérési helyét és nevét.

```
settings.Schemas.Add("http://XMLDocSchema",
XmlReader.Create(Directory.GetCurrentDirectory()+"\\XMLSchema.xsd"));
```

Majd egy for ciklusban végigfutok a fájlok nevein, amiket ellenőrizni szeretnék.

```
for (int xml = 1; xml <= 5; xml++)
{
```

Egy feltételben vizsgálom, hogy az adott fájl létezik-e, ugyanis előfordulhat, hogy még nem töltöttük le.

```
if (File.Exists(Directory.GetCurrentDirectory() +
      "\\XMLFiles\\XML" + xml.ToString() + ".xml"))
{
      try
      {
```

Ekkor az adott állományhoz létrehozunk egy `XmlTextReader` objektumot, ami egy `while` ciklusban elkezdi beolvasni az állományt. Ha nem történik hiba, akkor a fájl szintaxisával nincs gond.

```
using (XmlReader xmlValidatingReader =
      XmlReader.Create(Directory.GetCurrentDirectory
      () + "\\XMLFiles\\XML" + xml.ToString() +
      ".xml", settings))
{
      while (xmlValidatingReader.Read()) { }
}
}
```

Hiba esetén kiíratjuk azt.

```
catch (Exception e)
{
      //kiíratás
```

```
            }
        }
    }
```

Ha az alkalmazás talál szintaktikailag hibás fájl(ok)at, akkor ahhoz, hogy a leveleket ki tudja küldeni, ki kell javítanunk ezen problémákat. Ezt kétféleképpen tehetjük meg. Fontos megjegyeznem, hogy az automatikus javítást egyszerre csak egy fájlon hajthatjuk végre, így az összes hibára egyenként meg kell hívnunk a javítás valamelyik módszerét.

3.5.1 A szintaktikailag hibás fájlok automatikus javítása

A javításokra szolgáló módszerek közül az egyik az, hogy a fájlokat automatikusan javíttatjuk, a módosításokat a programra bízzuk.

Az ilyen módú javítás a következő algoritmus alapján fut le:

- A javításra meghívott fájl ismételt megvizsgálása. Erre azért van szükség, mert időközben akár változhatott a fájl tartalma, szerkezete. Ez akkor lehetséges, ha például kézzel már javították a fájlt, vagy az óránkénti automatikus letöltés hajtódott végre ez idő alatt.
- Az ellenőrzés minden javítás után lefut, hiba esetén újból kezdetét veszi az algoritmus, mert az azt jelenti, hogy több hiba is volt a fájlban.
- Ha a hiba továbbra is fennáll, az XML fájl tartalmát beolvassa a program egy szöveges változóba.
- Ezek után a hiba szövegéből kiszűri, hogy hol és milyen probléma áll fenn.
- A program két esetet tud lekezelni:
    - Az XML dokumentum elemei nem megfelelő sorrendben vannak egymásba ágyazva. Ekkor a sorrendben előrébb álló elem záró részét kitörli a program, majd folytatja működését. Nyilvánvalóan a záró elem törlése nem nyújt megoldást a problémára, de végrehajtásával a következő esetet idézzük elő, amelyet meg tud oldani az alkalmazás.
    - Az XML dokumentum valamelyik eleme nincs lezárva. Ekkor a hibaüzenetből kiderül, hogy melyik sorba kellene lennie a záró elemnek, így egyszerűen beszúrjuk oda az – utóbbi hibaüzenetből kinyert – elem nevét a </ és > jelek között.

- Ezek után a program visszaírja a szintaktikailag helyes adatokat az XML fájlba, majd lezárja azt.
- Bármilyen egyéb hiba esetén (pl. nem tudta megnyitni a fájlt írásra, mert nincs hozzá jogunk, vagy másnál nyitva van), egy hibaüzenettel tér vissza az alkalmazás.

3.5.2 A manuális javítás

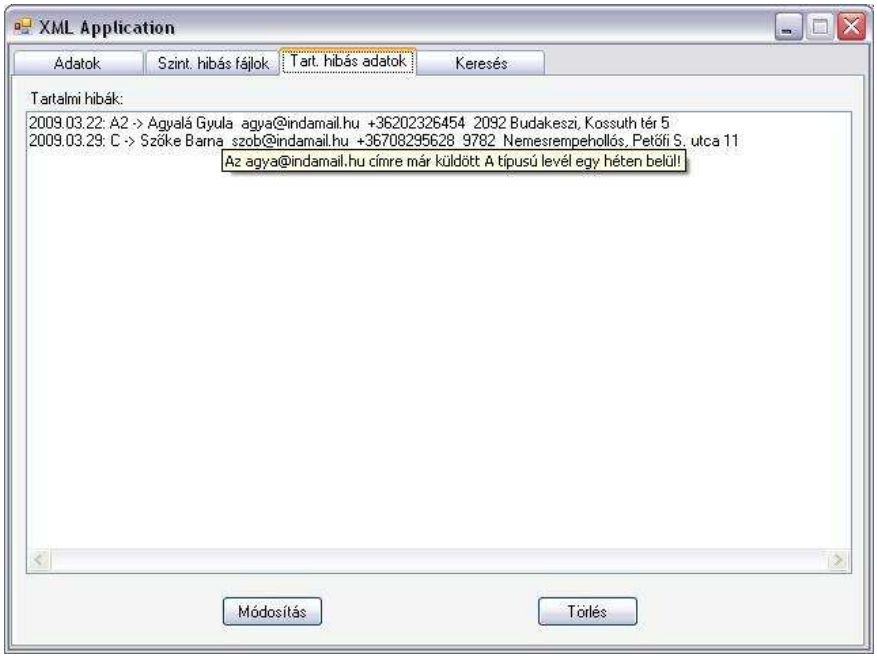
Ha úgy gondoljuk, hogy a szintaktikai hibák javítását kézzel jobban el tudjuk végezni, akkor a „Kijelölt fájl megnyitása manuális javításra” gomb megnyomása után ezt meg is tehetjük. Ekkor a kijelölt fájlunk tartalma egy új ablakban nyílik meg, ahol lehetőségünk van az adatok módosítására, majd a fájl mentésére is. Természetesen mentés nélkül is vissza tudunk lépni. Itt jegyezném meg, hogy egy fájl akár több száz sorból is állhat, így a manuális javításuk nehézkes is lehet. Ehhez a módszerhez viszont nagy segítségünkre lehet, hogy a hibákat nyomon követhetjük az ablakban, így a megfelelő sort megkeresve könnyedén korrigálhatjuk a fájl szerkezetét.

3.6 Az alkalmazás szempontjából tartalmilag hibás adatok

A tartalmi hibák az ügyfelek nem megfelelő adataiból keletkezhetnek. Mivel többféle levél generálásáról van szó, így ezek küldésének gyakoriságára és számosságára különböző feltételek vannak. Ezek a következők:

- A (A1 vagy A2) levelek: 1 héten belül ugyanaz az ügyfél ne kapjon A típusú elektronikus levelet.
- B: összesen kétszer kaphat, a kettő között legalább 3 hétnek kell eltelnie.
- C: legfeljebb egyszer kaphat ilyet.
- D, E: 1 héten belül ne kaphasson ugyanaz az ügyfél ugyanolyan típusú levelet, a D és E típusú e-mailek között 2 hónapnak kell eltelnie.

Ezenkívül még vannak olyan ügyfelek, akik bizonyos oknál fogva egyáltalán nem kaphatnak levelet. Ezek a személyek a `TiltottXMLDocs` nevű SQL táblában találhatók meg, akiket az aktuálisan betöltött adatok – vagyis az adott napon kiküldendő elektronikus levelek címzettjei – között fellelve szintén tartalmi hibát ad a program.
A hibás személyek adatai a 4. ábrán látható ablakban jelennek meg:



4. ábra: a tartalmilag hibás adatok ablak

Itt megjelenítjük a hibás személyhez tartozó információkat, kiegészítve azzal a dátummal, amikor a legutóbbi levelet küldtük neki, valamint a levél típusával. Továbbá a kurzort a hiba felé mozgatva megjelenik a hiba részletesebb leírása, vagyis annak oka, amiért a hibás adatokhoz került az ügyfél. Ide kerülnek azok a személyek is, akik a tiltólistán vannak. Ekkor az aktuális dátummal jelenik meg az ügyfél, a szövegdobozban az „Ő nem kaphat levelet” kiírással.

3.6.1 A tartalmilag hibás személyek kigyűjtése az adatbázisból

A tartalmilag hibás ügyfeleket az Transact SQL és az XQuery lekérdezések kombinációjával veszem ki az `XMLDB` adatbázisból a következő tárolt eljárás segítségével:

```
1. CREATE TABLE #temp (
      id int,
      template_name nvarchar(2),
      surname nvarchar(30),
      forename nvarchar(30),
      postcode int,
      town nvarchar(40),
      street nvarchar(40),
```

```
      house_number int,
      phone_number nvarchar(15),
      email nvarchar(40),
      dated datetime
)
2. CREATE TABLE #tempt (
      ...
)
3. CREATE TABLE #tempd (
      id int,
      date nvarchar(10),
      dated datetime
)
4. INSERT INTO #temp
5. SELECT
      id,
      ref.value('./@Template_Name', 'nvarchar(2)') AS [template_name],
      ref.value('./@Surname', 'nvarchar(30)') AS [surname],
      ref.value('./@Forename', 'nvarchar(30)') AS [forename],
      ref.value('./@Postcode', 'int') AS [postcode],
      ref.value('./@Town', 'nvarchar(40)') AS [town],
      ref.value('./@Street', 'nvarchar(40)') AS [street],
      ref.value('./@House_Number', 'int') AS [house_number],
      ref.value('./@Phone_Number', 'nvarchar(15)') AS [phone_number],
      ref.value('./@EMail', 'nvarchar(40)') AS [email],
      null
6. FROM XMLDocs CROSS APPLY
    XMLData.nodes('ROWS/LETTER_SECTION/LETTER_DETAILS') AS T(ref)
7. INSERT INTO #tempt
8. SELECT
      ...
9. FROM TiltottXMLDocs CROSS APPLY
      XMLData.nodes('ROWS/LETTER_SECTION/LETTER_DETAILS') AS T(ref)
10. INSERT INTO #tempd
11. SELECT
      id,
      ref.value('.', 'nvarchar(10)') AS [date],
      null
```

```
12. FROM XMLDocs CROSS APPLY
      XMLData.nodes('ROWS/HEADER_SECTION/CREATION_DATE') AS T(ref)
13. UPDATE #tempd
14. SET dated = convert(datetime, date)
15. UPDATE #temp t1
16. SET dated = t2.dated
17. FROM
18.         (SELECT id, dated FROM #tempd) AS t2
19. WHERE t1.id = t2.id
20. SELECT
      t2.dated,
      t2.template_name,
      t1.surname,
      t1.forename,
      ...
21. FROM #temp t1
22.   JOIN #temp t2
23.         ON t1.email = t2.email
                AND t1.id < t2.id
                AND (t1.template_name like 'A%'
                      AND t2.template_name like 'A%'
                      AND datediff(day, t2.dated, t1.dated) <= 7
                )
                AND (t1.template_name like 'B'
                      AND ...
                ...
24. UNION ALL
25. SELECT
      ...
26. FROM #tempt tt
27. JOIN #temp t1
28.         ON t1.email = tt.email
```

A kód működése soronként:

Az első sorban létrehozok egy ideiglenes táblát, amelyben a tartalmilag hibás ügyfelek adatait fogom letárolni. Ezután létrehozok egy újabb táblát ugyanazzal a szerkezettel, amelyben a későbbiekben a tiltólistás személyek lesznek. Majd a harmadik sorban egy újabb

tábla definiálása következik. Ebben lesznek az ügyfeleket tartalmazó XML fájlok létrehozásának dátumai, az összes ügyfélhez kapcsolva.

Ezután a 4. sortól kezdődően a `#temp`, majd a `#tempt` táblába beszúrom az összes ügyfél adatait az `XMLDocs`, illetve a `TiltottXMLDocs` táblákból. Ezt a már korábban megismert módon hajtom végre.

Majd a 10. sortól kezdve a `#tempd` táblába szúrom be az összes ügyfélhez tartozó XML fájl létrehozásának dátumát.

Ezt követően (a 13. sortól) a `#tempd` (dátumokat tartalmazó tábla), majd ezen keresztül a `#temp` tábla dated mezőjének értékét módosítom. Erre azért van szükség, mert az XML fájlokban – és ezáltal az adatokat tartalmazó táblákban is – a létrehozás dátuma egy egész szám típusú mező. Így a dátumok összehasonlítása nehézkes lenne. Ezért az ott található számot szöveges értékben kérem le az adatbázisból, majd dátummá konvertálom, és így megkapom a létrehozás dátumát, dátum típusúként.

A 20. sortól kezdve a visszaadandó sorokat kérdezem le. Ehhez az összes adatot tartalmazó táblát (`#temp`) össze kell fűzni saját magával, így megkapva azokat a sorokat, amelyek többször szerepelnek benne. Ezen kívül még feltételnek kell szabnom az összekapcsolás során, hogy csak a valamilyen okból hibás adatokat adja vissza, tehát azokat, amelyek a fent említett feltételeknek eleget tesznek. Ilyen például az A típusú leveleket kapó ügyfelek egy héten belül. Ennek a lekérdezésnek az eredményéhez még hozzá kell adni a 25. sortól kezdődő `SELECT` eredményét, ami a levelet nem kapó személyeket adja vissza oly módon, hogy a tiltott és az összes személy adatait fűzi össze. Az összehasonlítások alapjául mindig az ügyfelek e-mail címe szolgál, mivel ez egyedi mező.

3.6.2 A hibás adatok módosítása

A hibás adatok javítását is biztosítja a program. Erre két módszer van kidolgozva, az egyik az adatok módosítása. Ezt akkor használhatjuk, ha másfajta levelet szeretnénk küldeni az adott ügyfélnek. A változtatandó ügyfél kijelölése után a módosítás gombra kattintva – a felbukkanó új ablakban – meg kell adnunk az levél új típusát, ezzel megváltoztathatva az adott ügyfél adatait a táblában. Ez a következő kóddal valósul meg:

```
1. UPDATE XMLdocs
2. SET XMLData.modify('replace value of
(/ROWS/LETTER_SECTION/LETTER_DETAILS[@EMail=sql:variable("@mail")]/@Templat
e_Name)[1] with sql:variable("@template_name")')
```

Először meghívjuk az SQL `UPDATE` parancsát, amivel elérjük azt, hogy módosíthassunk az adatbázisban. Természetesen az adott táblában az `XMLData`, vagyis az XML adatokat tartalmazó mezőt kell megváltoztatnunk, így a `SET` kulcsszó után ezt adjuk meg. Erre az oszlopra kiadjuk az XQuery modify parancsát, majd zárójelek a közé beírjuk, hogy milyen módosítást szeretnénk végezni az adatokon. Ebben az esetben egy értéket szeretnénk kicserélni egy másikra, amit a `replace value` kulcsszó jelez. Az `of` után megadjuk, hogy a fájl melyik értékét szeretnénk megváltoztatni. Itt a szögletes zárójelpár között megadhatunk egy szűkítést, ami ebben az esetben a tárolt eljárásban megkapott, és az XQuery számára – SQL változóként – átadott e-mail címre vonatkozik. Miután megtalálta a program a megfelelő – adott elektronikus levél címmel rendelkező – személyt, a szögletes zárójelekben lévő tag figyelmen kívül hagyásával folytathatjuk a változtatandó mező elérési útvonalát az XML adaton belül. Most a levél típusát, vagyis a `Template_Name` értéket szeretnénk módosítani, így azt írjuk be. A `with` kulcsszó után megadhatjuk, hogy mire szeretnénk módosítani ennek értékét. Ezt szintén egy SQL változóból kapjuk meg.

3.6.3 A hibás adatok törlése

A hibák javításának másik eszköze a törlés. Ekkor szintén ki kell jelölnünk a törlendő személyt, majd a törlés gomb lenyomásával kitörölhetjük őt az adatbázisunkból. Ezt a következőképpen valósíthatjuk meg:

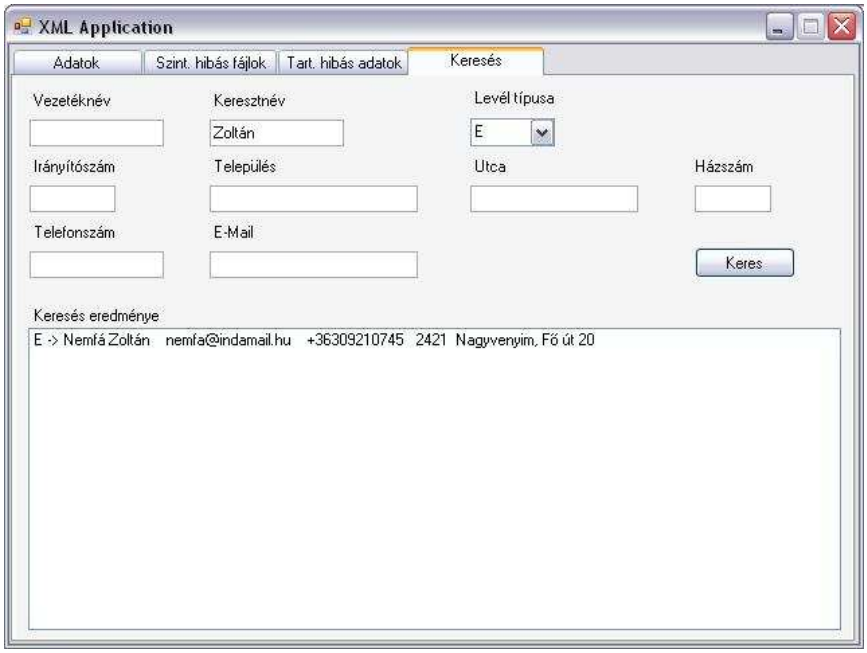
```
1. UPDATE XMLdocs
2. SET XMLData.modify('delete
/ROWS/LETTER_SECTION/LETTER_DETAILS[@EMail=sql:variable("@mail")]')
```

Az `UPDATE` parancs kiadása után – ismét az `XMLData` mezőre – meghívjuk az XQuery modify parancsát. Ennek most a delete kulcsszót adjuk meg, majd a fájlon belül a törlendő ügyfél kiválasztásához – SQL változóként - átadjuk a személy e-mail címét. Fontos megjegyezni, hogy ilyenkor nem fog kitörlődni az egész rekord, csak a mező `LETTER_DETAILS` része. Ennek oka, hogy az elérési útvonalban ez az utolsó elem. Ezt

azért valósítottam meg így, mert az `XMLDocs` táblában egy sorban egy XML fájl tartalma található, így több személy adatai is lehetnek benne, amiket nem akarunk kitörölni. Ez vonja maga után azt is, hogy nem `DELETE` parancsszót adtunk át az SQL-nek. Az, hogy nem töröljük az egész sort – akár további `LETTER_DETAILS` adatok hiányában – nem jár semmilyen következménnyel, ugyanis egy üres `LETTER_SECTION` elem nem okoz sem szintaktikai, sem más hibát a programban. A tiltólistán szereplő ügyfelekre csak a törlés funkciót hívhatjuk meg. Ellenkező esetben hibaüzenettel figyelmeztet bennünket a program a nem megfelelő parancs hívásáról. A törlés persze nem a `TiltottXML` táblából, hanem az `XMLDocs` táblából való eltávolítást jelenti.

3.7 Ügyfél keresése az adatbázisban

Az alkalmazás lehetőséget nyújt arra is, hogy megkeressünk bizonyos ügyfeleket a rájuk vonatkozó adatok alapján. A keresés ablakot az 5. ábrán láthatjuk.



5. ábra: a keresést megvalósító ablak

Mint láthatjuk, az ablak megfelelő mezőibe beírhatjuk a keresendő személy nevét, levelének típusát, elérhetőségi adatait. A Keres gombra való kattintáskor a háttérben meghívunk egy tárolt eljárást, annak átadva az összes paramétert, attól függetlenül, hogy azt üresen hagytuk-e, vagy sem. A személy adatait a következő kóddal kaphatjuk meg:

```
SELECT XMLData.query('for $let_det in /ROWS/LETTER_SECTION/LETTER_DETAILS
     where (sql:variable("@temp_name") = null
            or contains($let_det/@Template_Name,
                                        sql:variable("@temp_name")))
     and (sql:variable("@surname") = null
            or contains($let_det/@Surname, sql:variable("@surname")))
     and (sql:variable("@forename") = null
            or contains($let_det/@Forename, sql:variable("@forename")))
     and (sql:variable("@postcode") =  0
            or $let_det/@Postcode = sql:variable("@postcode"))
     and (sql:variable("@town") = null
            or contains($let_det/@Town,sql:variable("@town")))
     and (sql:variable("@street") = null
            or contains($let_det/@Street, sql:variable("@street")))
     and (sql:variable("@house_num") = 0
            or $let_det/@House_Number = sql:variable("@house_num"))
     and (sql:variable("@phone") = null
            or contains($let_det/@Phone_Number, sql:variable("@phone")))
     and (sql:variable("@email") = null
            or contains($let_det/@EMail, sql:variable("@email")))
     return $let_det')
FROM dbo.XMLDocs
```

A kódban az `XMLDocs` táblában, az `XMLData` mezőben keresek. A fájlon belüli elérési útvonalat az `in` kulcsszó után láthatjuk. A `where` részben minden egyes értékre, amire kereshetünk két feltétel vonatkozik. Az első, hogy az értéke `null`, illetve szám típusú mező esetén `0`. A program a meghíváskor, ha nem talál értéket egy szám típusú mezőben, akkor `0`-t ad át a SQL eljárásnak. Erre a vizsgálatra azért van szükség, mert ha nem töltjük ki a mezőt, akkor a feltétel további vizsgálata itt megáll, és az SQL nem vizsgálja a második kritériumot. Ezzel többek között gyorsíthatjuk az eljárást, és megóvhatjuk a programot az esetleges hibáktól. A feltétel második része a szám típusú mezőnél egy egyenlőségvizsgálat, a szöveget tartalmazó mezőknél egy XQuery-s összehasonlító függvény. Itt a `contains`

kulcsszó megadása után zárójelben meg kell adnunk, hogy miben, majd vessző után, hogy mit keresünk. A tartalmazó rész jelen esetben mindig az XML fájl `LETTER_DETAILS` elemének egy attribútuma, a keresendő szöveg pedig egy SQL változó értéke. Így minden esetben megkaphatjuk az(oka)t az eleme(ke)t, amely(ek)re a feltételek mindegyike teljesül. Természetesen ez történhet egy, vagy akár az összes érték megadásával.

4. XML adatbázis-kezelő szoftverek, eszközök

Az XML dokumentumokon végrehajtott lekérdezéseket természetesen nem csak az SQL Server segítségével lehet végrehajtani. Számos program, és egyéb eszköz létezik ezek használatára. Ezek közül először a .NET keretrendszerben elérhető lehetőségeket szeretném bemutatni, majd az olyan szoftvereket, amelyek használatához nem szükséges más eszköz.

4.1 .NET technológiát használó eszközök

4.1.1 A Saxon XQuery Processor

A Saxonica Limited XQuery-t feldolgozó termékei közül a legutóbbi verzió a 9.1.0.6., melyet 2007-ben adott ki a készítője Micheal H. Kay, a W3C támogatásával. Ez a verzió egyidejűleg lett kibocsájtva a Java és .NET keretrendszerekhez. A Saxon két csomagot készített el. Az egyik a Saxon-B, amely az alapvető megfeleltetést valósítja meg az XSLT 2.0-hoz és az XQuery-hez. A másik a Saxon-SA, amely egy séma-központú XQuery és XSLT feldolgozó. Mindkét csomag mindkét platformon elérhető, a két csomag közötti eltérés az, hogy a Saxon-B nyílt forráskódú, míg a Saxon-SA egy kereskedelmi forgalomban lévő termék, 30 napos ingyenes használati engedéllyel. A Saxon-SA és Saxon-B közötti legnagyobb különbség az, hogy az SA séma-központú, így megengedi az olyan lekérdezéseket, amelyben XML sémákat használnak. A SA csomag ezen felül tartalmaz egy XML Schema érvényesítőt, valamint olyan haladó kiegészítőket, amelyeket a Saxon-B nem. Ezek például a következők:

- Az SA csomag magába foglal egy olyan eszközt, amely az XPath kifejezéseket, XQuery FLWOR lekérdezéseket és XSLT sablonok kapcsolatait optimalizálja. Míg a Saxon-B ezeket mindig egymásba ágyazott ciklusokkal valósítja meg, addig az SA különböző indexelésen és hash-összekapcsoláson alapuló stratégiákat alkalmaz. Ez a végrehajtási idő óriási csökkenését eredményezi, egy 10 MB-os XML állomány esetében ez akár 300-szoros is lehet.
- A Saxon-SA a dokumentumokat képes adatfolyamként kezelni, így azokat a fájlokat is fel tudjuk dolgozni, amiket nem tudunk a memóriában tárolni.

- Az SA csomagnak van olyan hibakezelő része, amellyel képes kezelni és diagnosztizálni a dinamikus hibákat, valamint le tudja kezelni az XQuery formázási, csoportosítási és egyéb végrehajtási műveleteiből származó hibákat.

A Saxon API lehetőséget nyújt, hogy .NET-es környezetben írjunk XPath, XQuery lekérdezéseket, XSLT átalakításokat és XML Schemat használjunk. Ez az összes .NET-alapú nyelven elérhető. Ehhez csak le kell töltenünk a saxonapi.dll-t, és a programon belül létrehozni egy hivatkozást a `Saxon.Api` névtérre.

Az alkalmazásban az első, amit meg kell tennünk, hogy létrehozunk egy `Processor` nevű objektumot. Ezt be kell állítanunk az alapján, hogy a feldolgozandó XML fájlunk séma-központú-e, továbbá meg kell adnunk az XML és séma fájl elérési útvonalát és további beállítási lehetőségeket. Lehetőségünk van egyidejűleg több feldolgozót futtatni, de a Saxon eljárásoknak általában egy alkalmazáson belül gazdaságosabb ugyanazt a feldolgozót használnia. A következő folyamatok az XQuery, XPath és XSLT esetében is egy mintára épülnek:

- A feldolgozóból létre kell hoznunk egy `Compiler` objektumot a megfelelő nyelv módszerét használva. Ez a következő lehet: `NewXQueryCompiler`, `NewXPathCompiler` vagy `NewXsltCompiler`.
- Az elkészült `Compiler` objektumban be kell állítani az összes tulajdonságot és konfigurációs opciót, a kiválasztott nyelvnek megfelelően. Ezután meg kell hívni a `Compile` metódust, amely létrehoz egy `Executable` objektumot. A `Compile` metódusok túl vannak terhelve annak érdekében, hogy többféle forrást fogadjanak el inputként.
- Az `Executable` objektum egy stíluslapot, egy lekérdezést vagy egy kifejezést képviselhet. Ezt olyan gyakran lehet kiértékelni, amennyire szükség van rá, megtéve ezt egy, vagy több szálon egyszerre is. A kiértékelés első lépése a `Load` metódus, amellyel betölthetjük az `Executable` objektumot. A betöltés eredménye egy `XQueryEvaluator`, `XPathSelector` vagy `XsltTransformer` lehet, a választott nyelvtől függően.
- Az utolsó függvény, amit meg kell hívnunk, a `Run`, amely lefuttatja az általunk hívott lekérdezést. Ennek visszatérési értéke egy új XML dokumentum, egy érték vagy egy

elem lehet. Ezeket az `Evaluate`, `EvaluateSingle` és `GetEnumerator` függvények segítségével érhetjük el.

A Saxon API sok osztályt tartalmaz, ami tükrözi az XQuery/XPath/XSLT adatmodellt (XDM). Ezek a következők:

- `XdmValue`: Lehet egy XPath érték. Ez egy általános szekvenciában található, aminek az elemei csomópontok vagy atomi értékek. Egy stíluslap értéke vagy egy lekérdés eredménye is lehet, egy XQuery lekérdezés vagy egy XPath kifejezés kiértékelésekor.
- `XdmItem`: Lehet egy XPath elem, vagy egy `XmdValue` egy altípusa. Meg lehet vele hívni a `GetEnumeration` függvényt egy `XdmValue` osztályon, amely végigfut és feldolgozza a szekvenciában levő tételeket.
- `XdmNode`: egy csomópont. Ez az objektum hozzáfér az XDM modellben meghatározott csomópontok legfontosabb tulajdonságaihoz. Ezek alapján lehet: csomópont típusú, szöveges érték, név, típusos érték vagy URI. Az `OuterXml` tulajdonság lehetőséget nyújt arra, hogy sorba rendezhessük a csomópontokat.
- `XdmAtomicValue`: az XDM modellben meghatározott elemi érték. Elemi értékeket közvetlenül létre lehet hozni egész számból, szövegből, dupla pontosságú számból vagy egy URI-ból. Ezen kívül készíthetünk egyet úgy is, hogy megadunk egy szöveget, ami tartalmazza a lexikális ábrázolást és egy azonosítót, a szükséges típussal ellátva.

További osztályok által nyújtott szolgáltatások:

- XML dokumentumokat fel tudunk építeni sokféle forrásból, a `DocumentBuilder` osztály felhasználásával.
- XML dokumentumokat validálhatunk egy XML Schema alapján. Ezt a `SchemaValidator` osztály segítségével tehetjük meg. Ha az ellenőrzés közben hiba lép fel, a dokumentum nem érvényes, szintaktikailag hibás. Ellenkező esetben, vagyis ha lefut az ellenőrzés, akkor az XML fájl séma-érvényes.
- Az `XmlDestination` osztály segítségével meghatározhatjuk, hogy egy lekérdezés, átalakítás vagy validálás eredményét hogyan kezeljük.

Bár a Saxon API Java nyelven készült el, ettől függetlenül semmilyen akadálya nincs annak, hogy bármely .NET nyelvben használjuk. Amikor .NET platformon futtatjuk a Saxon-t, két XML szintaktikai elemző áll rendelkezésre: a `System.Xml`, a .NET platform beépített

elemzője, és a JAXP szintaktikai elemző, amely az OpenJDK könyvtárban található. Ez gyakran hivatkozik a `System.Xml` névtérre, bár a Saxon termék részeként kapjuk meg.

4.1.2 A Microsoft XQuery motorja

Ebben a részben azt szeretném bemutatni, hogy a Microsoft saját XQuery motorját hogyan lehet használni .NET-ben. A Saxon termékéhez hasonlóan itt is egy hivatkozással történik a dolog. Hátrány viszont a Saxon API-val szemben, hogy ez csak ASP.NET alatt használható.

Ahhoz, hogy a Microsoft által nyújtott ingyenes XQuery motort ASP.NET-es web alkalmazásainkban használni tudjuk, először le kell tölteni az xquery.msi fájlt a megfelelő helyről. Ezt elindítva a telepítő eljárás létre fog hozni egy új könyvtárat, amiben több alkönyvtár és fájl között megtalálhatjuk a Microsoft.Xml.XQuery.dll-t. Ahhoz, hogy az XQuery kifejezéseket egy ASP.NET web alkalmazásban használni tudjuk, egyszerűen át kell másolni ezt a fájlt az ASP.NET alkalmazás bin könyvtárába. Ezt következően importálni kell a `Microsoft.Xml.XQuery` névteret a kódot használó osztályban. Amint végrehajtottuk ezeket a lépéseket, el tudjuk kezdeni az XQuery kifejezéseket használni.

A kódunkban létre kell hoznunk egy `XQueryNavigatorCollection` példányt, és paraméterként megadni a felhasználandó XML dokumentum elérési útvonalát és nevét. Ezt a következőképpen tehetjük meg:

```
XQueryNavigatorCollection col = new XQueryNavigatorCollection();
col.AddNavigator(fileName, alias);
```

Itt egyszerre több XML dokumentumot is megadhatunk, mivel ezzel a módszerrel össze tudjuk kapcsolni a fájlokat, és így egy XQuery lekérdezésben egyszerre több állományból tudunk adatokat kinyerni. Amikor végre szeretnénk hajtani egy lekérdezést, előtte létre kell hoznunk az `XQueryExpression` osztály egy példányát és át kell neki adnunk a végrehajtandó XQuery lekérdezést tartalmazó fájlt. Ennek a fájlnak az elérési útvonalát a következő példában a `query` nevű változóban tároljuk:

```
XQueryExpression expr = new XQueryExpression(query);
```

Ezek után az `XQueryExpression` objektumra meghívjuk az `Execute` metódust. Ez egy `XQueryNavigatorCollection` példánnyal tér vissza, amelyre meghívjuk a `ToXml` függvényt, így segítségével XML formátumban kaphatjuk vissza a lekérdezésünk eredményét.

Ez a következő módon történik:

```
(expr.Execute(col)).ToXml();
```

A későbbiekben ezzel az eredménnyel dolgozhatunk.

4.1.3 Az AltovaXML 2009

Az Altova nevű cég is kiadott több XML-állományokat kezelő, feldolgozó szoftvert. Az AltovaXML 2009 rendszerük támogatja az XQuery lekérdezéseket többek között .NET keretrendszerben is. Ennek megvalósításához a már eddigiekhez megszokott módon le kell töltenünk egy .dll kiterjesztésű fájlt a cég honlapjáról, majd a betöltés után hivatkoznunk kell rá. Ez a hivatkozás ebben az esetben az `Altova.AltovaXML` nevet viseli. Az eddigiektől eltérő módon viszont egy regisztráció szükséges a névtér használatához.

Miután ez megtörtént, elérhető lesz számunkra négy osztály, amelyből nekünk kettő lehet igazán lényeges. Az egyik az `XMLValidator`, amelynek metódusai segítségével megadhatunk egy ellenőrizendő XML állományt, vagy egy URL-t, amely az XML adatokat hivatkozza. Ezek után meg kell adnunk az ellenőrzéshez szükséges séma objektumot, ami lehet XMLShema, vagy DTD. Majd az `isWellFormed` és `isValid` metódusok meghívásával megkaphatjuk, hogy az adott XML fájlunk a séma alapján helyesen formázott és helyes-e. A visszatérés egy logikai értékkel történik.

A másik fontos osztály az XQuery nevű, amellyel természetesen lekérdezéseket hajthatunk végre. Ennél meghatározhatjuk, hogy az XML dokumentumot változó, vagy fájl segítségével szeretnénk megadni. Ugyanez igaz az XQuery lekérdezésre is, vagyis megadhatjuk egy fájllal, illetve egy változóval is. Ezen kívül még beállíthatjuk az eredmény kódolását, a betűérzékenységet, valamint azt is, hogy a lekérdezés eredményét szöveges változóban, vagy fájlban szeretnénk viszontlátni. Miután megadtuk a szükséges tulajdonságokat, az `Execute` függvény hívásával hajthatjuk végre a lekérdezést.

Az Altova ezen kívül még számos olyan szoftvert adott ki, amelyek segíthetik az XML állományok feldolgozását, megjelenítését, kezelhetőbbé tételét. Az egyik ilyen az UModel, amely legfőbb funkcionalitása, hogy egy XML állományt adhatunk át a .NET-es projektünknek. Ebből automatikusan generálja le a C# (vagy más nyelvű) kódot, tartalmazva az XML fájl szerkezetét, valamint a feldolgozásához szükséges függvények vázát is. A projekt fájl megadásával ezt egy új osztályként importálja be.

4.1.4 Az XQSharp

Nemrégiben (a 2009-es év első negyedévében) megjelent egy új eszköz az XQuery lekérdezések .NET-ben való használatához. Ezt XQSharp-nak nevezik, Visual Studio 2008 alatt lehet használni. A honlapon található leírás alapján a következő tulajdonságokkal rendelkezik a szoftver:

- Nagy teljesítmény: a fejlesztés során nagy gondot fordítottak arra, hogy a kód hatékonyan, gyorsan fusson.
- Statikus típus-ellenőrzés: biztosítják, hogy soha ne okozzon futási idejű hibát a nem megfelelő típus használata, így ezt még fordítási időben ellenőrzik.
- Séma-vizsgálat: XQSharp támogatja egy séma importálását és általa egy XML dokumentum validálását. Ezzel az eszközzel be tudjuk importálni az XML Shema-t a lekérdezésbe.
- Haladó XQuery támogatás: lehetőség van egyszerre több XML dokumentumból való lekérdezésre, XPath használatára, ezen kívül több kimeneti fájltípust is alkalmazhatunk (XML, XHTML, HTML vagy sima szöveg).

A nyílt forráskódú XQSharp a következő elemeket tartalmazza:

- XQSharp parancssori eszköz: támogatja a fent leírt jellemzőket, és lehetőséget nyújt, hogy bármilyen lekérdezést futtassunk.
- XQuery példák: a letöltött alkalmazás tartalmaz XQuery példákat is, ezáltal segítséget nyújtva a használatához.
- Dokumentáció: a csomag része egy teljes online dokumentáció, ami elmagyarázza, hogy hogyan lehet az eszközöket helyesen használni.

Az XQSharp teljes támogatást nyújt az XQuery lekérdezések használatához.

4.2 Natív XML adatbázis-kezelő rendszerek

Ebben a részben elszakadok a .NET-es technológiától, és olyan adatbázis-kezelő rendszereket mutatok be, melyek elsősorban önmagukban, vagy esetleg Java IDE-be ágyazva használhatóak.

4.2.1 Az <oXygen/>

Az <oXygen/> egy olyan XML szerkesztő szoftver, amely különböző lehetőségeket nyújt XML állományok feldolgozására. Ilyen például: XML átalakítás, XML Schema használat, XPath, XSLT, XQuery támogatás. Az <oXygen/> elérhető Eclipse-be beépülő bővítményként is.

Az <oXygen/> alapértelmezetten háromféle szerkesztői lehetőséggel rendelkezik. Az egyik egy szöveg alapú szerkesztő, amelynek használata mellett láthatjuk az egyes tagokat egy strukturáltabb felépítésben. Ezt egy külön ablak, az Outliner teszi számunkra lehetővé, azáltal hogy az egyes elemeket, attribútumokat és értékeiket szerkezetük szerinti felépítésben, formázva újra ki. A másik szerkesztési mód egy formázott XML szerkesztő, amelynek megjelenítésével jobban átláthatjuk az XML struktúrát és könnyebben módosíthatjuk. A harmadik szerkesztési módban az XML adatokat vizuálisan társíthatjuk egy CSS stíluslaphoz, amely meghatározza azt, hogy az adatok hogyan jelenjenek meg.

A program támogatja az XPath lekérdezéseket, és tartalmazza az 1.0 és 2.0-s függvényeket is. Mivel az összetett függvények megjelenítése a mennyiségük miatt egy eszköztárban nehézkes lenne, így a program ezt úgy valósítja meg, hogy a függvény beírásánál automatikusan felajánlja egy felugró listában a választható lehetőségeket. Az eredményeket egy külön panelben jeleníti meg, amelyben egy sorra való kattintással rá tudunk ugrani a dokumentumban. Az lekérdezés eredményét egy külön fájlban is elmenthetjük.

Az <oXygen/>-ben lehetőségünk van XQuery lekérdezések írására is. Ehhez minden támogatást meg is kapunk a szoftvertől. Az Outline ablakban az összes összetevő megjelenik: névtér deklarációk, változók és függvények. Ezeket rendezhetjük név, típus, hely és névtér alapján, valamint szűrést is alkalmazhatunk rájuk a nevük alapján. Mint a program többi szerkesztésénél, itt is egy listában ajánlja fel a használható kifejezés listáját, amely bármikor előhívható a Ctrl+Enter billentyűkombinációval. A függvények különböző modulokból (eXist, MarkLogic, Tiger Logic, Oracle, SQL Server, Tamino, stb.) is érkezhetnek. A választható függvények listában természetesen ezek is megjelennek. Miközben az XML fájljainkat szerkesztjük, a program érzékeli az inputként használt fájlokat, majd szerkezetüket egy külön panelben jeleníti meg. Lehetőségünk van a szoftverben arra, hogy az XQuery lekérdezéseinkhez Drag and Drop listából válasszuk ki és adjuk hozzá FLWOR és XPath kifejezéseinket. A lekérdezés futtatása előtt a program végrehajt egy szintaktikai ellenőrzést, amely felderítheti például az FLWOR kifejezésben szereplő hibákat. Az XQuery

lekérdezésünket a Saxon SA motor segítségével tudjuk végrehajtani. A futtatásaink során használhatjuk az <oXygen/> debuggerét, amely segítségével a következő műveleteket tudjuk végrehajtani:

- Megadhatunk különböző töréspontokat.
- Stack és Trace nézetének segítségével kideríthetjük a hiba helyét, a Trace logját egy fájlba is lementhetjük.
- A lekérdezésben részt vevő csomópontok, változók és paraméterek értékét tudjuk futás közben is nyomon követni.
- Le lehet kérdezni a futási időket a lekérdezés minden egyes részletére.

Az <oXygen/> a felsoroltakon kívül még más lehetőséget is kínál. Ezek a következők:

- Fa-szerkesztő: Nagy méretű dokumentumoknál lehet a szerkezetet faként megjeleníteni, és csak a számunkra fontos részeket megjeleníteni. Ezáltal csökkentjük a memória-kihasználtságot, valamint hatékonyabbá tehetjük akár az XQuery lekérdezéseket is.
- Séma-konvertáló: Segítségével XML sémákat tudunk átalakítani egy másik típusú séma-dokumentummá.
- Adatbázis-konvertáló: Adatbázisban tárolt táblákat tudunk XML fájlokká és séma dokumentumokká konvertálni szerkezetük alapján. Sajnos SQL Server táblából ez nem lehetséges.

4.2.2 Az eXist

Az eXist-db egy nyílt forráskódú XML adatbázis-kezelő rendszer, amely az index-alapú feldolgozást veszi alapul. Az eXist honlapján találhatunk egy XQuery SandBox nevezetű alkalmazást, amely segítségével a saját adatbázisában tárolt XML adatokon hajthatunk végre XQuery lekérdezéseket. Itt számos példát is találhatunk. A web-alapú felületen kívül létezik egy Java-alapú kliens része is. Ha a honlapon található jar fájlt letöltjük és a leírásoknak megfelelően feltelepítjük, akkor rendszeresen használhatjuk a szoftvert XML dokumentumainkból való lekérdezések megírására és végrehajtására. A program használatához a Java5 megléte szükségeltetik. A telepítés befejeztével be kell lépnünk az eXist adminisztrációs oldalára, ahol először létrehozhatjuk az admin felhasználót. A dokumentum gyűjtemények és az import fájlok feltöltése, valamint az adatbázisba való bejelentkezés után megjelenik előttünk egy új ablak, amelyben az adatbázis-gyűjtemények

szerepelnek. Ezen az ablakon egy másik panelben manuálisan megadhatjuk a lekérdezésünket. Mivel az eXist egy többfelhasználós rendszer, így a felhasználók kezelését is meg kellett valósítania. Ezt az Edit User ablakban tehetjük meg. Itt felvehetünk új, törölhetünk és módosíthatunk már meglévő felhasználókat.

Ahhoz, hogy az eXist-t használva optimális kódot tudjunk írni XPath-ban vagy XQuery-ben, lehetőségünk szerint el kell kerülnünk a több ezer soros kódokat. Ehhez a függvények és operátorok használata az első lépés. Rengeteg függvényt el tudunk érni a Function nevű interfész implementálásával. Mivel az adatbázis nem képes korlátlan számú függvényt eltárolni, így néhány más elérésű függvényt és operátort szeretnék bemutatni.

- A dokumentumok feldolgozását segítő
  - XPath/XQuery függvények: `doc()` és `collection()`
  - eXist-specifikus függvények: `xmldb:document()` és `xmldb:xcollection()`

  Ezeket főleg relatív útvonalak megadásánál használjuk. A `doc()` és `xmldb:document()` közötti eltérés az, hogy a `doc()` egyetlen, míg az `xmldb:document()` egyszerre akár több dokumentum beolvasására is képes. A `collection()` és `xmldb:xcollection()` közötti különbség az, hogy míg az előbbinek átadott kollekció al-kollekcióit is feldolgozza a függvény, addig az utóbbinak csak a paraméterül adott kollekciót fogja használni, az al-kollekcióit nem.
- Sztring-kezelő függvények: az XPath/XQuery függvénykönyvtár tartalmazza a legtöbb programozási nyelv által használt szöveg-manipulációs függvényeket. Ezek viszont nem biztos, hogy elegendőek a számunkra szükséges adatok visszanyeréséhez, például arra, hogy bizonyos szavakat megtaláljanak egy adott szövegkörnyezetben.
  - Az `&=` operátor segítségével az előtte megadott csomópontok halmazából választja ki azokat, amelyek tartalmazzák az utána következő szavak mindegyikét.
  - Az `|=` operátor az előzőhöz hasonló felépítéssel rendelkezik, viszont ennél az operátornál megadott csomópontoknak elegendő az utána következő kulcsszavak valamelyikét tartalmazniuk.

  Mindkét operátorra igaz, hogy nem érzékeny a kis- és nagybetűk közötti különbségekre.

    - A `near()` függvény hasonlóan működik, mint az `&=` operátor, azzal a különbséggel, hogy ennél megadhatjuk a szavak egymástól való maximális távolságát is. A távolság alatt a két kulcsszó közötti szavak számát kell érteni. Ez alapértelmezetten 1. A `near()` függvényben már használhatunk ún. joker-karaktereket. Ez lehet `?`, ami egyetlen, valamint `*`, ami egyszerre több karakter helyettesítésére szolgál.
    - A `text:match-all()` és `text:match-any()` függvények nagyban hasonlítanak az `&=` és az `|=` operátorokra, az eltérés annyi, hogy ezek reguláris kifejezéseket is tartalmazhatnak. Reguláris kifejezések használatakor egy mintát adunk meg, s azt vizsgáljuk, hogy a feldolgozandó adatok melyik része illeszkedik a megadott mintára.
- Az XML:DB kiterjesztett funkciói között megtalálhatjuk az adatbázisok kezelésére vonatkozóakat is. Ezen függvények közül szeretnék néhányat bemutatni:
    - Az `xdb:register-database` segítségével egy új adatbázis-vezérlőt tudunk implementálni.
    - Az `xdb:create-collection` függvény meghívásával egy új adatbázis-kollekciót tudunk létrehozni.
    - Az `xmldb:login`-t használva pedig be tudunk jelentkezni a paraméterként megadott adatbázisba, a megfelelő felhasználói névvel és jelszóval.

Ahhoz, hogy XQuery függvényeket használhassunk Java nyelven, implementálnunk kell az `org.exist.xquery.Function` vagy az `org.exist.xquery.BasicFunction` interfészt.

Az XQuery lekérdezés hatékonyabbá tételének másik módja a program egy model-rendszerének használata. Ezeket a modulokat egy

- URI,
- adatbázis kollekció,
- Java archívum-fájl (jar),
- Java osztály (ha a modul implementálva van benne)

megadásával is be tudjuk szerezni. Az XQuery modulok végrehajtása során az `XQueryServlet` vagy az `XQueryGenerator` automatikusan egy gyorsítótárba kerül. Így a következő alkalommal, amikor az adott XQuery lekérdezést vagy a modult használni szeretnénk az adott helyről, ezek nem fognak újra betöltődni, hanem a már lefordított kódot

fogják újra felhasználni. A kód csak abban az esetben fog újra betöltődni, ha az eXist jelzi, hogy változott, vagy hosszabb ideje használaton kívül van. Ha a lekérdezés egyszerre több szálon fut, akkor az egyik a kódot be fogja másolni a gyorsítótárba. Az eXist ezen kívül támogatja az XQuery alapú modulkönyvtárak betöltését is. Ezek általában függvények egy csoportjából és változódeklarációkból állnak. Ezek a modulok lehetnek külsők, amelyek XQuery lekérdezéseket tartalmaznak, valamint belsők, Java-ban implementált modulok. A modul importálása funkció segítségével különböző XQuery lekérdezéseket tudunk betölteni. A Java modulok felvétele viszonylag egyszerű, csak meg kell adnunk egy névteret tartalmazó URI-t, amelyben definiálva van a modulok és a függvények listája. Ezenkívül csak elérhetővé kell tennünk a Java osztály számára az XQuery motor használatát, valamint implementálnunk kell az `org.exist.xpath.InternalModule` interfészt.

Az eXist támogatja a pragmák használatát, amelynek segítségével implementáció-specifikus adatokat tudunk küldeni az XQuery motor felé. A pragmákat a korábbi eXist verzióban egyszerűen úgy kellett használni, hogy – egy változóhoz hasonlóan – deklaráltuk azt. Az új verzióban ezt egy különleges XQuery kifejezésbe ágyazzuk bele. Ezen pragmák segítségével képesek vagyunk:

- A futási időt mérni (`exist:timer`)
- Frissíteni az adatbázist (`exist:batch-transaction`)
- Elősegíteni az indexek működését, hibáiknak megtalálását (`exist:force-index-use`)
- Az indexek használatának mellőzését, ha más megoldással könnyebben, gyorsabban kereshetünk (`exist:no-index`)
- A lekérdezéseinket optimalizálni, implementálva az `org.exist.xquery.Optimizable` interfészt (`exist:optimize`).

A bemutatott szoftvereken, technológiákon kívül találhatunk még sok hasonlót, de az általam ismert és tesztelt eszközök nem rendelkeznek olyan lehetőségekkel, amelyeket a fenti alkalmazásokból teljes egészében hiányozna.

5. Összefoglalás

Úgy gondolom, hogy a diplomamunka kellő betekintést nyújt az XML fájlok felhasználásába, feldolgozásába. Ez egyrészt a .NET keretrendszerben jártas emberek kíváncsiságát elégítheti ki, az elkészített alkalmazás, valamint az XML adatbázisok kezelését elősegítő eszközök leírása által. Másrészt a Java fejlesztők is kaphattak egy kisebb betekintést az XML fájlokkal való manipulációba, remélhetőleg felkeltve érdeklődésüket az említett technológiák használatára.

Miután elkészítettem egy .NET-es XML állományokat feldolgozó alkalmazást, valamint részletesebben megismerkedtem azokkal az eszközökkel, amelyek ebben segítségünkre lehetnek, úgy gondolom, hogy szinte bármilyen hasonló problémát meg lehet oldani ezen eszközök használatával vagy kombinációjukkal. Szerintem, ennek bizonyítéka az is, hogy a felmerült igény alapján a szoftver elkészítése is megfelelően sikerült, hiszen a leírt kritériumoknak megfelel, a célul kitűzött feladatokat maradéktalanul ellátja. Bár a szoftver tesztelése még nem teljes, mégis merem azt mondani, hogy a kellő biztonsággal működik. Az alkalmazás tehát elvégzi az XML állományok adatainak megjelenítését, szintaktikai ellenőrzését, esetleges javítását, a megszabott feltételeket nem teljesítő, vagyis tartalmi hibás adatok kiszűrését és javítását (az adatok módosításával vagy törlésével), valamint az adatokban való kereshetőséget is. Ezen kívül külső szoftverek igénybe vétele nélkül képes automatikusan és manuálisan letölteni fájlokat FTP szerverekről, valamint azok tartalma alapján elektronikus levelet küldeni a megfelelő személyeknek.

Felhasznált irodalom

Könyvek

[1] Bradley, Neil: The XML Companion. In: Kis Balázs (szerk.): Az XML-kézikönyv. Veszprém, Szak Kiadó, 2005.

[2] Melton, Jim – Buxton, Stephen: Querying XML – Xquery, XPath, and SQL/XML in Context. San Francisco, Elsevier Inc., 2006.

[3] Walmsley, Priscilla: XQuery. Las Vegas, O'Reilly Media, 2007.

Internet

[1] Bray, Tim – Paoli, Jean - Sperberg-McQueen, C. M. – Maler, Eve – Yergeau, François: Extensible Markup Language (XML) 1.0 (Fifth Edition): http://www.w3.org/TR/xml. W3C, 2008.

[2] Boag, Scott – Chamberlin, Don - Fernández, Mary F. – Florescu, Daniela – Robie, Jonathan – Siméon, Jérôme: XQuery 1.0: An XML Query Language: http://www.w3.org/TR/xquery. W3C, 2007.

[3] Frequently Asked Questions About XML:DB: http://xmldb-org.sourceforge.net/faqs.html. TheXML:DB Initiative, 2000-2003.

[4] Vithanala, Prasadarao K.: Introduction to XQuery in SQL Server 2005: http://msdn.microsoft.com/en-us/library/ms345122.aspx. Microsoft Comporation, 2005.

[5] Kay, Michael H.: SAXON – The XSLT and XQuery Processor: http://saxon.sourceforge.net. Saxonica Limited, 2009.

[6] ASP.NET. 4 Guys From Rolla.com: Querying XML Data with XQuery: http://www.4guysfromrolla.com/articles/071603-1.aspx. 2003.

[7] AltovaXML 2009 .NET Interface: http://www.altova.com/manual2009/AltovaXML/index.html?ax_netinterface.htm. Altova, 2009.

[8] XQSharp: XQuery 1.0 for the Microsoft .NET Framework: http://www.xqsharp.com/xqsharp. Clinical & Biomedical Computing Ltd., 2008.

[9] <oXygen/> XML Editor: http://www.oxygenxml.com/xml_editor.html. SyncRO Soft Ltd., 2002-2009.

[10] eXist – Open Source Native XML Database: http://exist.sourceforge.net. 2009.

Függelék

Példa a feldolgozandó XML dokumentumra:

```
<?xml version="1.0" encoding="UTF-8"?>
<ROWS>
	<HEADER_SECTION>
		<CREATION_DATE>20090329</CREATION_DATE>
		<CREATION_TIME>2142</CREATION_TIME>
		<XML_ID>1</XML_ID>
	</HEADER_SECTION>
	<LETTER_SECTION>
		<LETTER_DETAILS Template_Name="A1" Surname="Agyalá"
			Forename="Gyula" Postcode="2092" Town="Budakeszi"
			Street="Kossuth tér" House_Number="5"
			Phone_Number="+36202326454" EMail="agya@indamail.hu" />
		<LETTER_DETAILS Template_Name="C" Surname="Szőke"
			Forename="Barna" Postcode="9782" Town="Nemesrempehollós"
			Street="Petőfi S. utca" House_Number="11"
			Phone_Number="+36708295628" EMail=szob@indamail.hu />
	</LETTER_SECTION>
</ROWS>
```

Az XML dokumentumok sémája:

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
	xmlns:company="http://XMLDocSchema"
	targetNamespace="http://XMLDocSchema" elementFormDefault="qualified">
  <xs:element name="ROWS">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="HEADER_SECTION">
          <xs:complexType>
            <xs:all>
              <xs:element name="CREATION_DATE" type="xs:string" />
              <xs:element name="CREATION_TIME" type="xs:string" />
              <xs:element name="XML_ID" type="xs:int" />
            </xs:all>
          </xs:complexType>
        </xs:element>
```

```
        <xs:element name="LETTER_SECTION">
          <xs:complexType>
            <xs:choice maxOccurs="unbounded">
              <xs:element name="LETTER_DETAILS">
                <xs:complexType>
                  <xs:attribute name="Template_Name" type="xs:string"
                        use="required" />
                  <xs:attribute name="Surname" type="xs:string"
                        use="required" />
                  <xs:attribute name="Forename" type="xs:string"
                        use="required" />
                  <xs:attribute name="Postcode" type="xs:string" />
                  <xs:attribute name="Town" type="xs:string" />
                  <xs:attribute name="Street" type="xs:string" />
                  <xs:attribute name="House_Number" type="xs:string" />
                  <xs:attribute name="Phone_Number" type="xs:string" />
                  <xs:attribute name="EMail" type="xs:string"
                        use="required" />
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani a diplomamunka létrejöttéhez nyújtott segítségért témavezetőmnek, Jeszenszky Péter egyetemi adjunktusnak.

Továbbá szeretném megköszönni főnökömnek, Pászty Györgynek az alkalmazás elkészítésének ötletét, valamint szakmai segítőkészségét.

Köszönettel tartozom szüleimnek, hogy évekig támogatták egyetemi tanulmányaimat, hiszen nélkülük nem sikerült volna elkészítenem a diplomamunkát sem.