Debreceni Egyetem

Informatika Kar

PDF-GENERÁLÁS
ADATBÁZISBAN TÁROLT XML DOKUMENTUMOKBÓL

Konzulens:
Kollár Lajos
Egyetemi tanársegéd

Készítette:
Beke Zoltán
Programtervező matematikus

Debrecen
2007

Tartalomjegyzék

1 Bevezetés 4
2 Az ORACLE és az XML gyakorlati alkalmazásai 6
3 Kódolás XML-ben 8
3.1 Az XML és a HTML 8
3.2 XML alapok 9
3.2.1 Üres címkék: 11
3.2.2 Attribútumok 11
3.2.3 Feldolgozási utasítások 12
3.3 XML séma nyelvek 12
3.3.1 Adatok megadása DTD segítségével 13
3.3.2 Adatok megadása XML Schema segítségével 16
3.4 XPATH 21
4 Az XML használata programozói oldalról 23
4.1 Java és az XML 23
4.1.1 SAX API 23
4.1.2 DOM 25
4.2 XML az ORACLE-ben 27
4.2.1 XMLTYPE adattípus 28
4.2.2 XMLTYPE a gyakorlatban 30
4.2.3 XMLTYPE függvények 32
5 PDF generáló alkalmazás 36
5.1 Bemutatás 36
5.1.1 Esetleges felhasználási területek 36
5.1.2 Felhasználói felület 37
5.2 Technikai háttér 38
5.2.1 Az alkalmazás ORACLE háttere 38
5.2.2 Az alkalmazás XML háttere 42
5.2.3 Az alkalmazás Java háttere 45

5.3 XML lekérdezések tesztelése Java és ORACLE oldalról 50
5.3.1 XML lekérdezések Java oldalról 50
5.3.2 XML lekérdezések ORACLE oldalról 51
5.3.3 Összehasonlítás 51
5.3.4 Konklúzió 53
5.4 További fejlesztési lehetőségek 54
6 Összefoglalás 56
7 Irodalomjegyzék 58
8 Függelék 59
9 Köszönetnyilvánítás 60

1 Bevezetés

A Diplomamunka címében szereplő betűszó nagyon sokaknak elég misztikusnak hat. Többen tévhitben vannak, vagy egyáltalán nem tudják hova tenni, csak azt lehet tudni, hogy szakmai körökben előszeretettel használják. Ez a sokat használt betű hármas az XML (eXtensible Markup Language).

Az XML lényegében a webes technológiákkal jött létre, melyet először sokan félreértelmeztek, aztán pedig szabványos építőelemmé vált. Az XML ma már „hivatalosan" is jelen van a webes technológiák között, és megszámlálhatatlan webes szolgáltatás alapját adja. Az XML terjedése ma is folyamatosan növekszik, mivel egyre több fejlesztő és vállalat ismeri fel a benne rejlő lehetőségeket. Mindazonáltal az XML még mindig egy viszonylag fiatal technológia, így sokan vannak, akik csak vizsgálják, hogy mire is lenne jó ez nekik.

Az XML potenciális felhasználási területeit nagyon nehéz meghatározni nem úgy, mint a HTML vagy a Java technológia esetén. Az XML-t az emberi DNS szálhoz lehetne hasonlítani, mivel arról sem lehet könnyen megmondani, hogy milyen embert fog produkálni, ugyanúgy az XML használhatósága sem egyértelmű.

Ahogyan egyre többen fognak web alkalmazások fejlesztésébe, úgy egyre többen szembesülnek a HTML nyelv korlátaival. Az XML hivatott ezen korlátok leküzdésére, ezzel lehetővé téve, hogy az ábrázolás problémáit félretéve, a webes dokumentumok készítőinek csak a tartalommal kelljen törődniük. Mindezt a HTML-hez hasonlóan egy egyszerű, könnyen olvasható címkerendszerrel és egy szabályrendszerrel valósítja meg.

Az XML lényegében egy metanyelv, ami azt jelenti, hogy elsődleges feladata más nyelvek leírása. Lényegében egy olyan keret, egy olyan szabályrendszer amihez, minden belőle származtatott leírónyelvnek alkalmazkodnia kell. Az XML-t használva egy olyan egyedi nyelvet alkothatunk, amelynek segítségével gyakorlatilag bármilyen információ leírható.

A diplomamunkám során rá szeretnék mutatni, hogy az XML technológia mennyire széles körben használható, milyen alapokon nyugszik, és egyfajta áttekintést szeretnék nyújtani arról, hogyan használható ez a nyelvezet fejlesztői szemszögből nézve.

Az XML számos területen alkalmazható ezek között szerepel az adatbázisokban való tárolás

Be szeretném mutatni, hogy a rendkívül népszerű és fejlett adatbázis-kezelő rendszer egy olyan verzióját, ami lehetőséget ad számunkra, hogy XML dokumentumokat tároljunk az

adatbázis-tábláinkban. Fontos bemutatni, hogy milyen lehetőségeket ad ez a rendszer arra, hogy ilyen típusú dokumentumokkal dolgozzunk. Mivel ez egy olyan verzió, ami hallgatók és fejlesztők számára ingyenesen elérhető, érdemes rámutatni arra, hogy menyiben kínál többet a többi ingyenesen használható, és letölthető adatbázis-kezelő rendszerektől. Azért használtam az ORACLE adatbázis-kezelő rendszert, mert többi szoftverrel szemben egyedül ez nyújt natív támogatást az XML dokumentumok tárolására, illetve kezelésére. Ami azért jó, mert az XML dokumentumot már az adatbázisban való tárolás során XML típusú adatként kezelhetjük, ellentétben a többi szoftverrel, amik ezt a típust nem ismerik adatbázis oldalon, ott csak egyszerű karaktersorozatként jelenik meg.

Fontos megemlíteni, hogy ez a technológia, milyen fejlesztések felé halad tovább, milyen lehetőségek vannak a követező generációs adatbázis-kezelő rendszerekben. Ezért rámutatok a jelenlegi verzióban található hiányosságokra, és bemutatom, hogy ezeket hogyan lehet megoldani a nem régen kiadott legújabb verzióban.

Szeretnék bemutatni egy általam fejlesztett PDF generáló alkalmazást, aminek a fejlesztése során nagyon sok lehetőségre és rejtett hátrányokra bukkantam mind Java és mind adatbázis oldalon. Az alkalmazás kiaknázza azt a lehetőséget, hogy előre definiált szerződésformákat alkalmazva a felhasználónak csak a szerződő felek adatait, vagyis a változó tartalmat kelljen meghatároznia. A formai megkötéseket egy XML dokumentum tartalmazza.

Majd összehasonlítom, hogy bizonyos XML lekérdezések hogyan zajlanak le az adatbázis oldalán, illetve, hogyan valósítható meg ugyanez Java oldalon. Ezen összehasonlítás azért volt szükséges, mert a két technológia gyakorlatilag ugyanarra képes, de bizonyos feladatokra az egyik, míg másra a másik alkalmazható hatékonyabban.

2 Az ORACLE és az XML gyakorlati alkalmazásai

A diplomamunka fejlesztése során az ORACLE Database 10*g* Express Edition (ORACLE Database XE) belépő szintű, kis helyigényű adatbázis-kezelő rendszert használtam, amely ingyenesen használható fejlesztésre, szabadon telepíthető és továbbadható. Az adatbázis-kezelő adminisztrációja is egyszerű. Az ORACLE Database XE segítségével nagy teljesítményű, bevált, piacvezető szoftverinfrastruktúrán lehet alkalmazásokat fejleszteni és megvalósítani, amelyek később szükség esetén költséges és bonyolult migráció nélkül bővíthetők. Ez az adatbázis-kezelő rendszer alapvetően a más ingyenes nyílt forráskódú adatbázis-kezelő rendszerek, mint például a MySQL, PostgreSQL vagy a teljesen Java alapokra lépülő HSQLDB, ellenfele próbál lenni azzal, hogy egy belépő szintű adatbázis-kezelő rendszert kínál az alkalmazásfejlesztőknek, adatbázis-adminisztrátoroknak és diákoknak saját alkalmazásaik fejlesztéséhez. Lehetősége van a független szoftver és hardverszállítóknak, hogy a saját termékeikkel vagy azokba beépítve ingyenesen forgalmazhassák, ezzel alkalom nyílik arra, hogy a fejlesztők és diákok a világ vezető adatbázis-kezelő rendszerét használva alkalmazásokat fejlesszenek és üzemeltessék azokat.

Az nyílt forráskódú rendszerek egyik nagy hátránya, hogy gyakran nem rendelkeznek ilyen részletes támogatottsággal és szakirodalommal, interneten fellelhető technikai anyaggal, professzionális szakértői háttérrel, emellett az alkalmazásfejlesztőknek később lehetőségük van átkonvertálás nélkül bővíteni éles üzemű, nagy adatmennyiségeket kezelő rendszerekké alkalmazásaikat, ekkor a fejlesztés sokkal kisebb kockázattal jár.

Az ORACLE Database XE az ORACLE Database 10g 2. verziójának alapjára épül, így teljes mértékben kompatibilis a cég többi adatbázis-szoftverének összes tagjával, köztük a Standard Edition One, Standard Edition és Enterprise Edition kiadással. Tehát kicsiben lehet kezdeni, és az igények növekedésével frissíteni lehet az Database 10g-re úgy, hogy az alkalmazások átírás nélkül átköltöztethetők a más kiadású ORACLE adatbázis-kezelőkre.

A Database XE ugyanazt a jól ismert SQL és PL/SQL programozási felületet biztosítja, mint a Database 10g többi verziója, de emellett a különböző fejlesztői csoportok igényeinek megfelelően más programnyelvek széles körét is támogatja. Például teljes körű fejlesztési és üzemeltetési támogatást nyújt a Java, .NET, PHP és Windows környezetben dolgozó fejlesztőknek. Emellett a Database XE révén a fejlesztők teljes mértékben kihasználhatják az

ORACLE HTML DB funkcióit a webes alkalmazások gyors fejlesztéséhez és rendszerbe állításához.
Az ORACLE Database XE 32 bites Linux vagy Windows operációs rendszeren futtatható és a következő paraméterekkel rendelkezik:

- Legfeljebb egy egymagos vagy kétmagos processzort használ;
- Egy gigabájt alatti memóriát kezel;
- Gépenként csak egy példányban használható; valamint
- Összesen 4 GB felhasználói adatot tárolhat.

Későbbi fejezetben szeretném tárgyalni, hogyan támogatja az ORACLE az XML nyelvezetet, ami újdonságnak számít az adatbázis-kezelő rendszerek terén. Itt akarom részletezni, hogy hogyan használhatjuk az XMLTYPE típust, valamint hogyan kezelhetjük az ilyen tulajdonsággal rendelkező mezőinket.
A későbbiekben részletezem az általam elkészített alkalmazást, ami interneten keresztül lehetővé teszi, hogy XML dokumentumokat tárolhassunk az adatbázisban, kihasználva az abban rejlő lehetőségeket és az így letárolt XML dokumentumokból dinamikusan, állíthassunk elő PDF dokumentumokat. Ezen konkrét alkalmazás alapján szeretném bemutatni, hogy milyen lehetőségeket kínál az ORACLE, milyen előnyei és hátrányai vannak. Szeretnék rámutatni a fejlesztés közben felmerülő problémákra, ezek lehetséges megoldására, valamint ez egyes technológiák hiányosságaira, és hogy ezek kiváltására, vagy pótlására milyen lehetőségeink vannak. A későbbi fejezetekben rávilágítok, arra hogy hogyan egészíti ki a Java nyelv a szolgáltatásaink repertoárját, és egy összehasonlító tesztelés után melyik szolgáltatása gyorsabb, és melyik használata egyszerűbb számunkra.

3 Kódolás XML-ben

3.1 Az XML és a HTML

Amióta egyre jobban elterjedt az Internet, számos új technológia jött létre, de ezek közül a leginkább szerteágazó hatással a XML rendelkezik. Ez az a technológia, amelyet mindegyik platform képviselője elismer és használ. Az XML egy olyan technológia, ami valójában arról gondoskodik, hogy az adatok jól strukturáltan megfelelő formában álljanak rendelkezésre. Nagyon kevés olyan helyzetet találunk, ahol a végfelhasználó találkozik az XML-lel.
Az XML sokban hasonlítható a HTML-hez, de ez a kijelentés csak nagy vonalakban igaz mert valójában sokkal, szélesebb hatáskörrel bíró adatstrukturáló nyelv, amely messze sokkal többre használható, mint weblapok készítése.
Ha azt mondjuk, hogy a HTML a weblapok forrása, azt nehéz megválaszolni, hogy a XML minek a forrása, nem egy konkrét célra találták ki, hanem egy olyan keretrendszer, amely a legkülönfélébb problémák megoldására ad segítséget. Mindezt egyedi fejlesztésű, XML alapú jelölőnyelvek teszik lehetővé. Az XML mindenképpen egy hasznos technológia, mert olyan háttér-technológiát képvisel, amellyel nagyon jól szervezett adattárakat hozhatunk létre. Azzal is érdemes tisztában lenni, hogy az XML valahol a HTML jövője, mivel a HTML egy kicsit alulstrukturált, és a fejlesztők elég szabadosan használják az egyes nyelvi elemeket, az így kialakult helyzet azzal okolható, hogy a HTML-ben nincs olyan szigorú szabályrendszer, ami az XML-nek a kezdetektől fogva a sajátja.
Az XML a HTML mellett a legnagyobb hatású webtechnológia, de a HTML-től eltérően ez rejtve marad a felhasználó előtt. Az XML egy úgynevezett metanyelv, ami azt jelenti, hogy elsődleges rendeltetése más nyelvek leírása. Ez lényegében azt jelenti, hogy az XML egy keret, egy olyan szabályrendszer, amihez minden belőle származtatott leírónyelvnek alkalmazkodnia kell. Az XML-t használva egy olyan egyedi nyelvet alkothatunk, aminek a segítségével gyakorlatilag bármilyen információ leírható. Eredetét tekintve az SGML egy leegyszerűsített részhalmaza, úgy ahogy a HTML is. Az XML és a hozzá kapcsolódó technológiák fejlesztését a World Wide Web Konzorcium (W3C) vállalta magára. Az XML tervezési szempontjai az XML-ajánlást tartalmazó dokumentum szerint a következők:

- Legyen egyszerűen használható az interneten;
- Alkalmazások széles körét támogassa;
- Legyen kompatibilis az SGML-lel;
- Egyszerűen lehessen XML dokumentumokat, feldolgozó programokat írni;
- Az XML opcionális lehetőségeit az abszolút minimumra kel csökkenteni;
- Legyenek az ember számára könnyen olvashatók, és világosan áttekinthetők;
- A tervezés folyamata legyen gyors;
- Szabályszerűen és tömören lehessen vele tervezni;
- A dokumentumokat könnyen létre lehessen hozni;

3.2 XML alapok

Az XML nem programozási nyelv, lényegében egy „nyelvtan” amelyet adatszerkezetek megadására használunk. Igaz, hogy a W3C-től származó ajánlás meglehetősen hosszú és összetett, maga a nyelv azonban rendkívül egyszerű. Ha valaki írt már valaha weboldalt, vagy böngészte egy oldal forráskódját, akkor egy XML-hez rendkívül hasonló struktúrát láthatott maga előtt. Az XML dokumentumok alapvetően két alapkomponensből épülnek fel: az adatok struktúráját kijelölő címkékből, és magukból az adatokból. Példa XML struktúrára:

```
<?xml version=”1.0”?>
<megszolitas stilus=”baráti”>
        <kitol>Feladó</kitol>
        <kinek>Címzett</kinek>
        <uzenet>Hello, hogy vagy?</uzenet>
        </alairas>
</megszolitas>
```

Az XML dokumentumok alapvető elemekből épülnek fel. Egy ilyen elemnek három összetevője van: nyitó- és zárócímke és valamilyen szöveges tartalom. Az XML elemeket egy fastruktúrára emlékeztető hierarchikus szerkezetbe rendezzük, ami az adatok logikai összefüggésére és tárolási struktúrájára is utal. A címkéket egy-egy pár relációs jel határolja

(<…>), ami közrefogja az elem nevét, attribútumait és értékeit. A záró címkék neve előtt (/) található.
Az XML elemeknek néhány egyszerű szabályt kell betartaniuk:

- Egy elemnek rendelkeznie kell nyitó- és záró cinkével, ha nincs zárócímke akkor, üres elemről beszélünk;
- A nyitó és záró címkéknek azonos névvel kell rendelkezniük;
- A kis és nagy betűk nem cserélhetők fel, külön kell kezelnünk őket;
- A címkék neve nem tartalmazhat úgynevezett „whitespace” karaktereket;

Az egyes elemek egymásba ágyazhatóak. Lényegében akkor jutunk egy igazi XML dokumentumhoz, ha van egy gyökér elemünk, és ez tartalmazza az összes többi elemet. Az XML címkék párokat alkotnak, ezért megnyitásukhoz képest fordított sorrendben, kell őket lezárni.

Hibás	Helyes
<?**xml version**=”1.0”?> <**meg szolitas stilus**=”baráti”> <**kitol**>Feladó <**kinek**>Címzett</**KiToL**> </**kinek**> </**megszolitas**>	<?**xml version**=”1.0”?> <**megszolitas stilus**=”baráti”> <**kitol**>Feladó</**kitol**> <**kinek**>Címzett</**kinek**> </**megszolitas**>

A táblázat bal oldalán szereplő kód nem érvényes XML jelölés, mert összekuszálódott benne a nyitó- és záró címkék sorrendje, a megfelelő címkenevekben nem következetesen lettek használva a kis és nagybetűk és meg nem engedett „whitespace” karaktereket alkalmaztunk bennük.

3.2.1 Üres címkék:

Előfordulhat, hogy az elemek nem tartalmaznak semmilyen szöveget, ennek több oka is lehet: vagy az elem attribútumai tartalmazzák az összes szükséges információt, vagy az elemnek üresnek kell lennie ahhoz, hogy a dokumentum érvényes legyen.
Az üres elemeket kétféleképp ábrázolhatjuk:

<**elem**></**elem**>
<**elem**/>

Vagyis egy üres elemet úgy hozunk létre, hogy a nyitócímkét közvetlenül a zárócímke követi, vagy egyetlen címke tartalmazza az elem nevét, és utána az elem lezárását, jelző perjelet.

3.2.2 Attribútumok

Olykor fontos lehet, hogy úgy rendeljünk információt egy XML-hez, hogy a kérdéses információt ne kelljen külön elembe foglalni.

<**img src**=”..//kepek/image00.jpg”
height=”640”
width=”480”/>

A fenti példa alapján, ha ezeket, az attribútumokat elemekké alakítanánk akkor, nemhogy nem lenne egyszerűbb a dokumentumunk szerkezete, de sokkal kevésbé lenne átlátható. Teljesen ránk van bízva, hogy mikor használunk az adatok tárolására elemeket, vagy attribútumokat, de előfordulhat, hogy egyes technológia, vagy feldolgozó sokkal hatékonyabb, ha különálló elemeket használ, de ha emberi felhasználónak mutatjuk be nyers XML-adatainkat, akkor az attribútumok sokkal áttekinthetőbbek. Az attribútumokkal meghatározhatjuk a lehetséges elemek listáját felsorolás formájában, lehet alapértelmezett értékük, típust adhatunk nekik így igen tömör formában, tárolhatunk információt. Ugyanakkor az attribútumoknak megvan a maga hátránya is, mivel nem alkalmasak hosszú karakterek tárolására, nem tudunk egymásba

ágyazni információkat és a „whitespace” karakterek is beleszámítanak az attribútumok értékébe. Az átláthatóság miatt nagyon fontos, hogy amikor lehet, mindig alkalmazzuk az attribútumokat kivéve, ha az információ csak hosszú karakterláncban adható meg, vagy a logikai szerkezet egymásba ágyazást igényel, vagy jelentéssel nem bíró de változó számú üres karaktert tartalmaz, amiket a rendszernek figyelmen kívül kell hagynia, akkor gyermek elemet kell alkalmaznunk.

3.2.3 Feldolgozási utasítások

A feldolgozási utasítások olyan információt tartalmaznak, amit az XML-forráskódot feldolgozó alkalmazásnak kell átadni. Ezek nem az XML jelölés részei, ezért ezeket <? ?> karakterkettősökkel határoljuk.

<?**xml version**=”1.0” **encoding**=”UTF-8” **standalone**=”no”?>

A fenti példában egy úgynevezett XML deklaráció szerepel, melyben azt adtuk meg, hogy a dokumentumunk az XML mely verziójával van összhangban. A jelenlegi alkalmazások az XML 1.0-ás verzióját ismerik. A kódolásra vonatkozó paraméter azt jelzi, hogy használtunk-e valamilyen kódolási sémát dokumentum létrehozásánál, ha ezt nem tüntetjük fel, akkor alapértelmezés szerint az UTF-8 karakterkódolást használjuk. A standalone paraméter arra utal, hogy alkalmazunk-e külső fájlt, mely a dokumentumot érintő deklarációkat tartalmazza.

3.3 XML séma nyelvek

Az XML dokumentumok megalkotásának oka, az emberi hibák számának csökkentése volt. Mivel az XML szerkezete meglehetősen merev, ezért a fejlesztőknek nagyon sok lehetőségük van, arra hogy hibázzanak. Az XML dokumentumokban esetleg megbújó hibák felfedezésének alapja a séma, amely egy olyan szerkezeti leírás, melynek segítségével az XML fejlesztők előre meghatározhatják, miféle adatok, milyen formátumban és milyen struktúrával szerepelhetnek az adott dokumentumban. A séma létrehozásához fel kell mérnünk, hogy milyen osztályba sorolhatók az adatok, ezeknek milyen elemek fognak

megfelelni, az elemekhez pedig milyen attribútumok tartozhatnak. A séma igazi jelentősége abban áll, hogy lehetővé teszi az XML dokumentum szerkezeti és logikai ellenőrzését. Ez azt jelenti, hogy a fejlesztő vagy esetleg egy alkalmazás tesztelheti, hogy a neki átadott XML dokumentum megfelel-e a sémában megadott szabályoknak. Ha valahol eltérés van, akkor biztos, hogy a dokumentum hibás. A sémák leírására alapvetően két jelölő nyelvet használhatunk:

- DTD (Document Type Definition)
- XSD (XML Schema)

3.3.1 Adatok megadása DTD segítségével

A DTD az XML dokumentumok logikai felépítésének leírására eredetileg alkalmazott megközelítés. Valójában nem az XML-el jött létre hanem, az elődjét jelentő SGML fejlesztői dolgozták ki. A DTD valójában úgy került be az XML világba, hogy annak születésekor már számos olyan eszköz létezett, ami képes volt DTD-ket kezelni. A legtöbb XML alapú jelölő nyelv leírása meglehetősen egyszerű, alapvetően azért, mert a DTD nyelve nagyon tömör. A DTD, a dokumentumok logikai szerkezetének alapvető részleteit írja le, vagyis az adott jelölőnyelvhez tartozó elemeket, attribútumokat és azok egymáshoz való viszonyát. Egy DTD-n belül a következő dolgokat adhatjuk meg:

- Az adott dokumentumtípusban megengedett elemek körét
- Az egyes elemekhez rendelhető attribútumokat
- A dokumentumban szereplő egyedeket
- A külső egyedekkel kapcsolatban használható jelöléseket

Amikor összerendelünk egy dokumentumot és egy DTD-t akkor két lehetőségünk van: vagy maga a deklaráció tartalmaz bizonyos leírásokat (internal DTD), vagy a deklaráció egy külső fájlra hivatkozik, ami a DTD-t tartalmazza (external DTD).

3.3.1.1 Elemek és Attribútumok

A DTD legfontosabb összetevője az elem és az attribútum. Ezek az összetevők azért fontosok, mert ők képezik a dokumentum logikai szerkezetét. Az elem a tárolt információ logikai egysége, az attribútum pedig az elem által tárolt információ tulajdonságát hordozza. Az attribútumokkal szűkíthetünk a lehetséges elemek listáján, akár alapértelmezést is megadhatunk. Ahhoz, hogy DTD-ben megadjunk egy elemet, a következő formátumú elemdeklarációt kell alkalmaznunk:

<!**ELEMENT** Elemnév Típus>

Az *Elemnév* annak a címkének a nevét határozza meg, amit az XML dokumentumban az adott elemtípus azonosítására fogunk használni. Az elem típusát a *Típus* mezőben határozhatjuk meg. Összesen négy elemtípust különböztetünk meg:

- *Empty* – üres elem, aminek nincs semmilyen tartalma.
- *Element-only* – Olyan elem, ami csak gyerekeket tartalmaz
- *Mixed* – Kevert tartalmú, vagyis gyermekeket és szöveget egyaránt tartalmazhat
- *Any* – Tetszőleges típus, ez bármit tartalmazhat, amit a DTD megenged.

Az *Empty* üres elemek nem tartalmazhatnak sem gyermek elemet, sem szöveget, viszont attribútumokat igen. Az *Element-only* elemek csak más elemeket tartalmazhatnak, vagyis az ilyen elemekben nem bukkanhat fel önálló szöveges objektum, csak más elemekbe ágyazottan. Ilyen elemeket úgy hozhatunk létre, hogy az elem deklarációjában „ContentModel” segítségével felsoroljuk, hogy milyen elemeket tartalmazhat.

<!**ELEMENT** ElementName ContentModel>

A tartalmi modell elemnevekből és speciális szimbólumokból áll, ahol a szimbólumok az egymás közötti és a szülővel kapcsolatos logikai viszonyokat írják le.

<!**ELEMENT** eletrajz (bevezetes, (tanulmanyok | tapsztalatok+)+, hobbik?, referenciak*)>

- Kerek zárójelek – gyermekek sorozatát vagy választási csoportot ír le.
- Vessző – sorozat elemeit írja le, az elemek sorrendje nem változhat.
- | – az alternatívákat választja el
- Nincs szimbólum – pontosan egyszer fordulhat elő az elem
- ? – a kérdéses elem maximum egyszer fordulhat elő.
- + – az elemnek legalább egyszer elő kell fordulnia
- *– az adott elem tetszőlegesen sokszor fordulhat elő

A kevert elemek elemeket és szöveget egyaránt tartalmazhatnak. A tetszőleges tartalmú elemek a legrugalmasabb szerkezeti egységek, gyakorlatilag bármit tartalmazhatnak, amit az adott DTD megenged: szöveget, gyermekelemet, vagy ezek tetszőleges kombinációját.

3.3.1.2 Attribútumok

Az attribútumok lényegében az elemek tulajdonságát írják le, melyeket attribútum lista formájában adhatunk meg:

<!**ATTLIST** ElemNév AttrNév AttrTípus Default>

Az attribútumnak mindig van neve, típusa és lehet beállított alapértelmezett értéke. Az alapértelmezett érték helyén négy kulcsszó szerepelhet:

- **#REQUIRED** – az attribútum megadása kötelező
- **#IMPLIED** – az attribútum megadása opcionális
- **#FIXED** – az attribútum kötött értékkel rendelkezik
- **default** – az attribútum alapértelmezett értéke

Az alapértelmezett érték mellett meg kell adni, hogy az adott attribútum milyen típussal rendelkezik. Az alábbi típusokat használhatjuk:

- **CDATA** – nem értelmezett szöveges adat
- **Felsorolásos** – karakterláncok sorozata
- **NOTATION** – a DTD egy más pontján megadott jelölés
- **ENTITY** – Külső bináris egyed
- **ENTITIES** – több külső bináris egyed, üres karaktereket is használva
- **ID** – egyedi azonosító
- **IDREF** – egy a DTD más pontján deklarált ID-ra mutató hivatkozás
- **IDREFS** – több különböző, máshol deklarált ID-ra mutató hivatkozás
- **NMTOKEN** – XML tokenekből felépített név
- **NMTOKENS** – több XML tokenekből felépített név

<!**ELEMENT** photo (image, format)>
<!**ATTLIST**
photo image **ENTITY #IMPLIED**
photo format **NOTATION** (gif | jpeg) **#REQUIRED**
>

A fenti példában látható, hogy a photo nevű elemhez két attribútum is tartozik, és mindegyiket egyetlen attribútum listában adtunk meg.

3.3.2 Adatok megadása XML Schema segítségével

Az előző fejezetben tárgyalt sémaleíró technológia, nagyon sok mindenre használható, de létezik egy nála sokkal fejlettebb megoldás is, ez az XML Schema [8]. Az ezzel létrehozott séma pedig az XSD (XML Schema Definition Language). A XML Schema-t a W3C hozta létre, XML alapokra, épül, és nagyon hasonlóan működik, mint a DTD: leírja a dokumentum szerkezetét és lehetőséget ad a dokumentumok automatikus ellenőrzésére is. Mivel az XSD is XML nyelven íródott, ezért ennek ellenőrzésére is van egy séma. És mivel nem használhatjuk önmaga ellenőrzésére, erre használjuk a DTD-t. Mivel az XSD is egy szabványos XML dokumentum, ezért ez is egy szabvány XML deklarációval kezdődik. Az XSD nyelv összes eleme és attribútuma egy névtérnek a része. A névtér nem más, mint logikai konstrukciók

csoportja, amelynek a feladata az, hogy garantálja az elemek és attribútumok nevének egyediségét. A névterekhez tartozik egy prefix amit, annak jelzésére használunk, hogy egy elem vagy attribútum az adott névtér eleme. A XSD névtér előtagja xsd, ami azt jelenti, hogy az XSD nyelv minden attribútuma és eleme az xsd betűhármassal és egy kettősponttal kezdődik. A sémában az XML deklaráció után kötelezően a névtér deklarációnak kell következnie, az **XSD séma általános szerkezete a következőképpen néz ki:**

<?**xml version**=”1.0”?>
<**xsd:schema xmlns:xsd**=”http://www.w3.org/2001/XMLSchema”>
</**xsd**:**schema**>

3.3.2.1 Az XSD adattípusai

A XSD nyelv alapját azok az adattípusok adják meg, amelyek meghatározzák, hogy egy-egy nyelvi elemmel egyáltalán milyen adatokról adhatunk meg valamiféle leírást. A típusokat alapvetően két csoportra oszthatjuk, egyszerű és összetett típusokra. Az egyszerű típusokhoz tartoznak az elemi adattípusok úgymint a számok, dátumok, listák, idők különböző formái és így tovább. Az összetett típusok már sokkal bonyolultabb adatszerkezetek, amelyekről el lehet mondani, hogy az egyszerű típusokból építkeznek. Az XSD az alábbi egyszerű típusokkal rendelkezik:

- Karakterlánc típusok – **xsd:string**
- Logikai típusok – **xsd:boolean**
- Számtípusok – **xsd**:**integer**, **xsd**:**decimal**, **xsd**:**float**, **xsd**:**double**
- Dátum és idő típusok – **xsd**:**time**, **xsd**:**timeInstant**, **xsd**:**duration**, **xsd**:**date**, **xsd**:**month**, **xsd**:**year**, **xsd**:**century**, **xsd**:**recurringDate**, **xsd**:**recurringDay**
- Egyedi típusok – **xsd:simpleType**

Az egyszerű típusok, lényegében elemekkel és attribútumokkal kapcsolatban egyaránt használhatók, elem létrehozására „**xsd:element**”, attribútum létrehozására az „**xsd:attribute**” elemet használjuk. Ha elemet adunk meg, akkor annak két elsődleges attribútuma van: a

„**name**” és a „**type**”, ahol a „**name**” az adott elmetípus nevét, vagyis a címkét fogja tartalmazni, a „**type**” az elem tartalmának a típusát határozza meg, ami lehet egyszerű és összetett. Az attribútumok meghatározása lényegében ugyanígy történik, azzal az eltéréssel, hogy az XSD elem neve most „**xsd:attribute**” lesz.
Az XSD egyik legjobb tulajdonsága az, hogy lehetőségünk van egyedi típusok létrehozására. Ilyenkor az elemi típusok értelmezését finomítjuk tovább saját igényeinknek megfelelően. Lehetőségünk van arra, hogy például a „**string**” típusra adunk egy olyan megszorítást, hogy mely értékeket veheti fel egy felsorolással, vagy egy szám típusú objektum esetén egy intervallum megadásával. A deklarációt minden esetben az „**xsd:simpleType**” elemmel kell kezdeni. Az esetek többségében valamilyen elemi típus értéktartományának szűkítésére van szükségünk, amire az „**xsd:restriction**” elemet kell használnunk. Ennek a „**base**” nevű attribútuma tartalmazza, hogy melyik típust akarjuk korlátozni. A korlátozó szabályok megadására a számtípus esetén az alábbi elemeket használhatjuk:

- **xsd:minInclusive** – a legkisebb megengedett érték
- **xsd:minExclusive** – a legkisebb megengedett érték úgy, hogy a határ nem esik bele az intervallumba
- **xsd:maxInclusive** – a legnagyobb megengedett érték
- **xsd:maxExclusive** – a legnagyobb megengedett érték úgy, hogy a határ nem esik bele az intervallumba

Az alábbiakban egy olyan példát mutatok be, amiben az egész típusra adok egy megszorítást és létrehozunk egy „onetotenType”-ot:

```
<xsd:simpleType name=”onetotenType”>
        <xsd:restriction base=”xsd:integer”>
                <xsd:minInclusive value=”1”/>
                <xsd:maxExclusive value=”11”/>
</xsd:restriction >
</xsd:simpleType>
```

A szám típusok értelmezési tartományának csökkentése mellet lehetőség van a szöveges típusok tartományának a csökkentésére is:

- **xsd:length** – a karakterek pontos száma
- **xsd:minLength** – a karakterek legkisebb megengedett száma
- **xsd:maxLength** – a karakterek legnagyobb megengedett száma

A felsorolt típusok segítségével mindegyik elemi típusra tehetünk megszorítást, kivéve a logikai típust. A lehetséges értékeket az „**xsd:enumeration**” elem segítségével adhatjuk meg és ennek „**value**” attribútumában lehet meghatározni a felvehető értékeket. Ezeket, az értékeket az „**xsd:restriction**” elemen belül kell megadnunk.
Míg a felsorolt típusoknál meg kellet adnunk az összes felvehető értéket, a listák lehetővé teszik számunkra, azt hogy egy elemnek több értéke is lehessen. A lista típust az „**xsd:list**” elemmel hozhatunk létre. Ugyanúgy a „**base**” attribútumban kell megadni, hogy melyik adattípust akarjuk használni, valamint szöveges típusnál alkalmazott hosszbeli megszorításokat tehetünk, de ha ezeket elhagyjuk, akkor tetszőleges listákkal dolgozhatunk.
A minta alapú elemtípusok az egyik legbonyolultabbak, de az egyik leghasznosabbak is. Úgy juthatunk minta alapú adattípushoz, hogy az adott típusú elemek tartalmának felépítését egy maszkkal adjuk meg. Minta megadása az „**xsd:pattern**” elemmel lehetséges. A pattern elemnek a „**value**” attribútumában kell megadni, hogy nézzen ki a szabályos kifejezés. Egy szabályos kifejezés az alábbi elemekből épülhet fel:

- . – bármely karakter
- \d – bármely számjegy
- \D – bármely nem számjegy
- \s – bármely üres karakter
- \S – bármely nem üres karakter
- x? – az x egy vagy nulla előfordulása
- x+ – az x egy vagy több előfordulása
- x* – az x tetszőleges számú előfordulása
- (xy) – az x és y csoporttá kapcsolása
- x|y – x vagy y

- [xyz] – az x, y, vagy z, közül bármelyik
- [x-y] – az x és y által meghatározott tartományból bármi
- x{n} – a x n számú előfordulása egy sorban
- x {n,m} – a x legalább n, de legfeljebb m számú előfordulása

Az összetett típusok valójában az egyszerű típusokra épülnek. Mindegyik összetett típust a „**xsd:complexType**” elemmel hozhatjuk létre, ennek az elemnek a „**name**” attribútumában adhatunk nevet a típusnak, ahhoz hogy a későbbiekben használhassuk, de mint az egyszerű típusnál lehetőség van arra is, hogy elemen belül hozzuk léte. Az összetett típusokat alapvetően négy csoportra oszthatjuk:

- Üres elemek
- Csak elemeket tartalmazó elemek
- Kevert tartalmú elemek
- Sorozatok és választási listák

Az üres elemeknek sem szöveges, sem gyermekelemeik nincsenek, de lehetnek attribútumaik. Az üres elemet az „**xsd:complexType**” és az „**xsd:complexContent**” elemek kombinálásával kaphatjuk meg.
A csak elemeket tartalmazó elemek olyan elemek, amelyek csak elemeket tartalmaznak szöveget nem. A csak elemeket tartalmazó elemet az „**xsd:complexType**” használatával hozhatjuk létre.
Kevert tartalmú elemek szöveget és elemet egyaránt tartalmazhat, ez a típus a legösszetettebb. Léteznek tisztán szöveges elemek, de ezek tekinthetők a kevert tartalmú elemek egy alfajának, ezek az „**xsd:complexType**” és az „**xsd:simpleContent**” elemek kombinálásával kaphatók. Ha kevert típusú elemet szeretnénk kapni, akkor a „**xsd:complexType**” „**mixed**” attribútumát kell igazra alítani.
A sorozat a gyermek elemek olyan listáját adja, ahol az elemek pontos megjelenítési sorrendje nem változhat. A sorozatot az „**xsd:sequence**” elem segítségével deklarálhatjuk, ebben fel kell sorolni azokat az elemeket, amelyek a halmazt alkotják és abban a sorrendben, ahogy egymás után kell következniük.

Ha olyan elemet akarunk létrehozni, amely egy sor opcionálisan választható elem közül csak egyet tartalmazhat, akkor választási listát kell alkalmazni. Ezekben a listákban gyermekelemek és más listák lehetnek, de ezek közül csak egy elem szerelhet a dokumentumban. Létrehozásához az „**xsd:choice**” elem deklarálására van szükség.

3.4 XPATH

Az XMLTYPE típusú adatok lekérdezésére, és bizonyos részeinek a kinyerésére az *extract* és az *existsNode* beépített tagfüggvények használhatók. A dokumentumban való navigálásra mind a kettő az XPATH kifejezéseket használja, ami a W3C által megvalósított szabvány [9]. Az XPATH olyan technológia, amely lehetővé teszi, hogy megcímezzük a dokumentumok egyes részeit. Ez a rész lehet csupán egyetlen csomópont, de lehet több csomópont is. Az XML dokumentumot egy fának tekinti, és lehetőségek széles skáláját adja ennek a fának a bejárására. Tekintsük az alábbi XML dokumentumot:

```
<dokumentum>
    <sor>
        <szoveg betu_tipus="Normal" dinamikus="false"
                igazitas="Bal" meret="10">Első sor
        </szoveg>
    </sor>
    <sor>
          <szoveg meret="10" igazitas="Jobb" dinamikus="false"
               betu_tipus="Normal">Masodik sor
          </szoveg>
    </sor>
</dokumentum>
```

A / jel a fa gyökerét jelenti, de ezen túl elválasztóként is funkcionál, ugyanis ezzel a jellel választjuk el a fa egy elemét a gyermek elemeitől is:

A „/**dokumentum**” jelölés a dokumentum logikai gyökerének a dokumentum címkéjű gyermekét jelöli. A „/**dokumentum/sor**” jelölés azon „**sor**” elemeket azonosítja a dokumentumban, amelyek a „**dokumentum**” elem leszámozottjai.

A // egy adott csomópont összes leszármazottjának az azonosítására szolgál:
A „/**dokumentum**//**szoveg**” a dokumentum elem összes „**szoveg**” leszármazottját kiválasztja.

A „*” karakter, helyettesítő karakterként szerepel:
A „/**dokumentum**/***/**szoveg**” kiválaszt minden a dokumentum elem unokájaként szereplő „**szoveg**” elemet.

A „@” jellel az attribútumokat hivatkozhatjuk:
A „/**dokumentum**/**sor**/**szoveg**/@**meret**” a „**dokumentum**” gyökér, „**sor**” gyermekének a „**szoveg**” leszármazottjának a „**meret**” attribútumát hivatkozza.

Lehetőség van index alapú jelölésre is, erre szolgál a „[]” jel:
A „/**dokumentum**/**sor[2]**” jelölés a „**dokumentum**” gyökér, második „**sor**” nevű gyermekét jelöli.

A „[]” jel, állítás tesztekre is használható:
A „/**dokumentum**/**sor**/**szoveg**/[@**meret=10** AND **igazitas**="Bal]/**text()**” kifejezés a „**dokumentum**” gyökér, „**sor**” gyermekének a „**szoveg**” leszármazottjának az értéket határozza meg, amelynek „**meret**” attribútuma „10” és „**igazitas**” attribútuma „Bal”

4 Az XML használata programozói oldalról

Ebben a fejezetben szeretném bemutatni, hogyan lehet kezelni az XML dokumentumokat. Programozói oldalról, szükségünk lesz azokra az alapvető szabályokra, amelyeket az „XML alapok” című fejezetben tárgyaltunk, úgymint mi a jólformáltság kritériuma, vagy mikor beszélhetünk egy dokumentum érvényességéről.

4.1 Java és az XML

Ebben a fejezetben az XML feldolgozás Java oldali megközelítését tárgyalom, ezen belül két XML feldolgozó API-t mutatnék be, az első a SAX (Simple API for XML) a második a DOM (Document Object Model).

4.1.1 SAX API

A SAX nem egy konkrét program, hanem egy programozási felület. Lényegében nem tesz semmit az adatokkal, csak kivált bizonyos eseményeket. Az API leírja, hogy hogyan kell egy SAX elemzőt megvalósítani. A SAX-ot xml-dev levelezési lista tagjai fejlesztették ki. Lényegében bármilyen programozási nyelven elérhető. Jelenleg a 2.0-ás verziónál tart, amit 2004-ben adtak ki, az 1.0-ás verzió 1998-ban jelent meg.
A SAX 1.0 az XML dokumentumban előforduló összes elemhez biztosítja a tartalomkezeléshez szükséges eseményeket, lényegében csak a névtereket nem támogatja. Egy SAX 1.0 elemzőnek a következő eljárásokat kell tartalmaznia, amelyeket a SAX automatikusan meghív, ha a megfelelő esemény a dokumentumban bekövetkezett:

- **characters()** – az elemben található karaktereket adja vissza
- **endDocument()** – akkor következik be ez az esemény, ha a dokumentum végére értünk
- **endElement()** – akkor következik be ez az esemény, ha egy elem záró címkéjét érzékelte a feldolgozó

- **ignorableWhitespace()** – akkor következik be ez az esemény, ha az elemző figyelmen kívül hagyható üres helyet érzékelt az elemek között
- **processingInstruction()** – ha az elemző feldolgozási utasítást talál, ekkor ezt az eljárást hívja meg
- **startElement()** – akkor következik be ez az esemény, ha egy elem nyitó címkéjét találja meg az elemző

A SAX 1.0-ban az attribútumok feldolgozására is van lehetőség, az általa nyújtott felület azon elemein keresztül, amelyek a **startElement()** metódus meghívásával egyidejűleg használhatók. A SAX 1.0 ma már elavultnak számít, 2.0 verzió óta inkább azt használják, bár a legtöbb 2.0-ás SAX támogatja az 1.0-ás verzió hívásait, de ma már mindenképp az újabb technológiát érdemes használni.
A SAX 2.0 az 1.0 olyan kiterjesztése, amely már a névtereket is támogatja, ezért az erre támaszkodó alkalmazásoknak további metódusokat kell tartalmaznia:

- **startPrefixMapping**() – ez akkor következik be ha, elkezdődik egy előtag leképezése, vagyis egy névtér leképezése egy adott egyedre
- **endPrefixMapping**() – ez a névtér előtag leképezés végén fut le
- **skippedEntity**() – ez a metódus akkor fut le, ha valamilyen okból az egyed kimarad a feldolgozásból

A SAX elemzőből több változat is létezik, ezek közül elsősorban a fejlesztés igénye szerint választhatunk. Java nyelven a Xerces elemzőt lehet használni. A Xerces XML elemzőt az Apache Software Foundation készítette el.
Egy általunk megírt Java feldolgozónak két interfészt kell megvalósítania a „ContentHandler” és az „ErrorHandler”. A „ContentHandler” interfész azon metódusokat valósítja meg, amelyeket a SAX elemző a normál feldolgozás során meghívhat. Az „ErrorHandler” metódusai azokra a hibákra kell reagálnia, amelyek feldolgozási hibákkal kapcsolatosak.

4.1.2 DOM

A DOM [7] egy olyan programozási felületet biztosít, amivel az XML dokumentum teljes tartalmát elérhetjük. Akár csak az XML-t is a W3C fejleszti, de DOM működésének számos részlete megvalósítás függő, vagyis az döntheti el, aki a konkrét elemzőt megírja. Fontos megemlíteni, hogy a DOM-nak több különböző szintje van. A DOM Level 1 kizárólag a HTML és XML tartalmakkal foglakozik. A DOM Level 2 bevezet néhány kiegészítést a névterekkel, CSS-sel és eseményekkel kapcsolatban, valamint tartalmaz néhány bejárási sémát. A DOM Level 3 már képes együtt működni az XML 1.1-es verziójával is, valamint ismeri számos más szabvány legfrissebb változatát is például: XML Information Set, XML Schema 1.0, SOAP 1.2. Az XML dokumentumok, adataik, illetve azok egy bizonyos része platform és nyelvfüggetlen módon elmenthetővé, illetve betölthetővé válnak a Load and Save ajánlását használva.

A DOM csak egy szabványt ad, így az implementációról a fejlesztőknek kell gondoskodnia. Az alkalmazásfejlesztőknek, akik DOM–ban akarják bemutatni dokumentumaikat, alkalmazkodniuk kell a megfelelő formához. A DOM logikai szerkezete egy fa, és megoldhatónak kell lennie, hogy egy alkalmazás átalakíthassa ezt a fát. Adatszerkezeti megközelítésből ez nem jelenti azt, hogy faként kell implementálni, hanem csak faként kell tudni kezelni, úgy hogy összhangban maradjon az ajánlással.

A DOM-nak egy adott programozási nyelvhez szánt megvalósítása, az adott nyelvhez való nyelvi csatolása. Amikor a DOM elemző beolvas egy dokumentumot, annak tartalmát ebbe a nyelvspecifikus objektumfába képezi le, és ezen a nyelven keresztül teszi elérhetővé.

Maga a DOM specifikáció csupán két nyelvhez tartalmaz közvetlen csatolást, az egyik a Java a másik az ECMAScript. Az ECMAScript lényegében a JavaScript szabványosított változata. Természetesen bármelyik nyelvi környezethez elkészíthető a megfelelő nyelvi csatolás csak követni kell a DOM specifikációban található leírást.

Miután az elemző előállította a dokumentumból a DOM fát, elkezdhetjük használni a csomópontokhoz tartozó programozási felületeket. Ezek a felületek a DOM fa egy-egy csomópont típusához tartoznak.

A *Node* az a DOM interfész, amelyből az összes többi programozási felület származik. Attól függetlenül, hogy a DOM fa adott csúcspontja milyen információt képvisel, mindig rendelkezik a *Node* összes tulajdonságával. A DOM fa összes csomópontja rendelkezik a

Node interfész összes metódusaival és attribútumaival, ezek lényegében az adott csúcspont szülőjével, gyerekeivel és testvéreivel kapcsolatos információkat tartalmazzák. Az attribútumok neve között szerepel a csomópont neve, értéke, valamint egy mutató arra a dokumentumra, amely tartalmazza a csomópont által tartalmazott információt.

A *Document* nevű interfész a DOM fa gyökércsomópontja, de az a csomópont nem feleltethető meg az XML dokumentum gyökerének. A *Document* interfész lényegében eggyel több szintet képez, mint amennyi szint az XML dokumentumban van. Ez azért van, mert ez az interfész számos olyan tulajdonsággal rendelkezik, ami magával a dokumentummal kapcsolatos. A *Document* interfész lényegében egy gyökérelem feletti szintet képvisel, maga a gyökér pedig csak a *Document* egyik gyermeke. A gyökérelem mellett még két gyermekre tartalmaz egy-egy mutatót a *DocumentType* és a *DOMImplementation*.

Az *Element* interfész az XML dokumentum egy elemét reprezentálja. Egyetlen erre az interfészre vonatkozó specifikus tulajdonság a címke neve, az összes többi attribútumot a felsőbb szintű *Node* interfésztől örökli. Tartalmaz még az elemhez tartozó attribútumok lekérdezésére, új attribútumok létrehozására illetve meglévők törlésére szolgáló metódusokat is. Lehetőség van egy adott névvel rendelkező gyermek elem lekérdezésére is, de lehetőség van hozzáférni az összes gyermekelemhez is, mivel az ehhez szükséges metódusokat örökli a *Node* interfésztől.

Az *Attr* interfész egy elem attribútumát reprezentálja, de nem tekinthető a DOM fa csomópontjának mivel egyik csomópontnak sem a gyermeke. Az *Attr* egy elem részének tekinthető. Mivel a *Node* interfész leszármazottja, az összes olyan metódus, ami a gyermekekkel kapcsolatos *null* értékkel tér vissza. Ezért nem lehet lekérdezni egy attribútum szülőjét, testvéreit vagy gyermekeit. Az attribútumoknak három tulajdonságuk van. Van nevük, értékük valamint egy logikai értékkel hivatkozik arra, hogy az adott attribútum explicit szerepel a dokumentumban vagy egy előre beállított értékről, van szó.

A *Nodelist* interfész nem a DOM fa egyetlen objektumára hivatkozik, hanem csomópontok egy csoportjára. Lehetőség van egy elemhez egy adott névvel rendelkező gyermek elemeinek a lekérdezésére. Egy eljárás és egy attribútum van ebben az interfészben, amit külön meg kell valósítani. Egy olyan attribútum, ami megmondja, hogy milyen hosszú az adott lista, és egy olyan metódus, amivel egy adott indexű elemhez lehet hozzáférni.

4.2 XML az ORACLE-ben

Az ORACLE 9i [1] verziója óta lehetőség van XML dokumentumok adatbázisban tárolására. Ebben a fejezetben be szeretném mutatni, hogyan tárolhatunk és kérdezhetünk le XML dokumentumokat, és hogy erre milyen eszközök állnak rendelkezésünkre.
A következőkben be szeretném mutatni a SYS.XMLTYPE típust, ami egy új rendszer szintű objektum, valamint ki szeretnék térni arra, hogy az ORACLE milyen XML jellemzőket támogat natívan.
Az ORACLE által natívan támogatott XML jellemzők:

XML eszköz	Leírás
XMLTYPE	Ez egy olyan előre definiált adattípus, ami az XML adatokhoz való hozzáférést biztosítja. Lehetőségünk van ilyen típussal rendelkező oszlopok létrehozására, ilyen típusú paraméterrel vagy visszatérési értékkel rendelkező PL/SQL függvények és létrehozására vagy ezeken az oszlopokon indexek létrehozására.
DBMS_XMLGEN	Az SQL lekérdezéseket XML formátumba alakíthatjuk.
SYS_XMLGEN	Egy SQL lekérdezésen belül generál egy XML-t. Az előzővel ellentétben nem egy eredményhalmazon operál, hanem egy skalár értéket, egy objektum típust vagy egy XMLTYPE típust konvertál egy XML dokumentummá. További paraméterként formátuma vonatkozó információk is megadhatók.
SYS_XMLAGG	Paraméterként kapott XML dokumentumokat vagy dokumentumtöredékeket fűz össze egy új gyökér elem segítségével.

UriType típusok	Ezek a típusok Uri-ref értékek adatbázisban tárolását teszik lehetővé. A SYS.UriType két altípussal rendelkezik: A SYS.HttpUriType és a SYS.DBUritype az első HTTP URL–ek míg a második az adatbázison belüli hivatkozások tárolására képes.

4.2.1 XMLTYPE adattípus

Ez egy új adattípus, ami táblák és nézetek oszlopaiban használható. Lehet változó, PL/SQL tárolt alprogram paraméter és függvény visszatérési érték típusa is. Beépített tagfüggvényei segítségével férhetünk hozzá a tartalmához, így ezek használatával az SQL lekérdezésekben elérhetővé válik az ebben letárolt XML tartalom.

4.2.1.1 Tábla létrehozás

Ilyen típusú oszlop ugyanúgy hozható létre, mint bármely másik.

```
CREATE TABLE DOC (
ID NUMBER(4)
XML_adat SYS.XMLTYPE);
```

4.2.1.2 Beszúrás

A Beszúráskor az XMLTYPE beépített tagfüggvényét kell használni, ami egy VARCHAR–ból vagy egy CLOB-ból hoz létre egy XMLTYPE példányt. Erre a feladatra a createXML függvény használható, ami leellenőrzi a bemenetként kapott XML adat jólformáltságát is.

```
INSERT INTO DOC (ID,XML_adat)
VALUES (01,SYS.XMLTYPE.createXML('
               <doc>
                    <adat format="dőlt">Szöveg</adat>
               </doc>
                                   '));
```

4.2.1.3 Lekérdezés

Lekérdezésekhez az extract tagfüggvényt kell használni, melynek paramétere egy XPATH elérési út.

```
SELECT    d.XML_adat.extract('/doc/adat/text()').getStringVal()
AS Eredmény FROM DOC d;
```

Eredmény
Szöveg

4.2.1.4 Módosítás

Ha módosítani szeretnénk az XML dokumentumot, az egészet ki kell cserélni, mivel az ORACLE pakolt CLOB ként tárolja azt. Az XML dokumentum részenként történő kicserélése nem támogatott. A módosítás egy egyszerű UPDATE segítségével mehet végbe, azzal a kiegészítéssel, hogy egy új XMLTYPE példányt kell létrehoznunk.

```
UPDATE DOC SET XML_adat=
SYS.XMLTYPE.createXML('
     <doc>
          <adat format="félkövér">Módosított szöveg</adat>
     </doc>
                         ');
```

Az ORACLE 11g verziójában lehetőség van arra, hogy ne csak CLOBJavaként tároljuk le adatainkat az XMLTYPE-ban, így a módosítások lehetővé válnak a dokumentum egy részén is, nem kell lecserélni az egész dokumentumot egy UPDATE művelet esetén.

4.2.1.5 Törlés

A törlés teljesen analitikusan történik, törlésre egyaránt felhasználható az extract és az existsNode beépített tagfüggvény is.

```
DELETE FORM DOC d WHERE
d.XML_adat.extract('/doc/adat/text()').getStringVal()='Text';
```

4.2.2 XMLTYPE a gyakorlatban

Az XMLTYPE lehetőséget ad arra, hogy XML adatokat tartalmazó táblákkal dolgozhassunk, úgy hogy a beépített függvényei segítségével SQL utasításokat hajthatunk végre. Az XMLTYPE szabadon használható, SQL utasításban más típusú adattípusokkal.
Az XMLTYPE nem mindig fa formában jelenik meg, mert úgy van optimalizálva, hogy ha nem szükséges a faforma, akkor szerializált formában van jelen, a faforma olyan beépített tagfüggvényeknél áll elő, mint például az extract és az existsNode. Így egyszerre használhatóvá válik számunkra a dokumentum szerializált és fa reprezentációja. Az XMLTYPE másik nagy előnye, hogy indexeket hozhatunk létre rajta.
Mikor mutatkozik meg az XMLTYPE előnye:

- Ha az XML dokumentumot egészben kell feldolgozni.
- Ha szeretnénk használni az extract és existsNode függvényeket a dokumentum egy részének a kinyerésére.
- Ha nagyon fontos hogy egy SQL utasításban vagy egy PL/SQL alprogramban egy átadott szöveges adat ne tetszőleges legyen, hanem csak XML érték.
- Ha ki szeretnénk használni az XPATH funkcionalitását, az extract és existsNode utasításokban.

- Ha nincs szükség az XML dokumentum részenként történő frissítésére (Csak ha CLOB ként tároljuk).
- Ha a későbbi fejlesztésekre gondolunk, akkor az XML dokumentumok tárolására, a legjobb választás az XMLTYPE mivel az ORACLE újabb verzióiban jobb és fejlettebb indexelési és optimalizációs technikák várhatók.

XMLTYPE használatának menete:

- Ha XMLTYPE típusú adatokkal szeretnénk dolgozni, akkor első lépésként egy ilyen típusú oszlopot kell létrehoznunk, ahol megadhatunk bizonyos tárolási jellemzőket is.
- Egy XMLTYPE példányt az XMLTYPE konstruktorával hozhatunk létre, vagy használhatjuk a SYS_XMLGEN és SYS_XMLAGG függvényeket is.
- Egy példány lekérdezésére az ORACLE két beépített tagfüggvényt biztosít, az első az extract, amit a tartalom kinyerésére használhatunk, a másik az existsNode, aminek segítségével egy bizonyos csomópont meglétét ellenőrizhetjük.
- Az ORACLE lehetőséget ad arra, hogy ORACLE Text indexeket hozzunk létre az XMLTYPE típusú oszlopokon, ezzel olyan szöveges operátorokat is használhatunk rajta, mint a CONTAINS, HASPATH, INPATH és más szöveges operátorok. Lényegében minden LOBJavaon operáló szöveges operátort használhatunk rajta.

Az XMLTYPE tárolása valójában CLOB ként történik, az ORACLE 9i verzióban csak ez az egy tárolási lehetőség van számunkra, de az ORACLE 11g verzióban az alábbi tárolási módok elérhetők: [6]

- CLOB – az ORACLE ezt a tárolási módot használja alapértelmezettként. Akkor használjuk, ha a tárolás dokumentum centrikus. Ez azt jelenti, hogy nincs szükségünk a dokumentum részenként történő frissítésére, valamint a lekérdezések az egész XML dokumentumra vonatkoznak. Az ORACLE 11g verzió előtt egyik hátránya ennek a tárolási módnak, hogy XML töredékek lekérdezése elég lassú, még úgy is, hogy a XPATH kérések optimalizálására létrehozhatunk függvény alapú indexeket, de ez nem mindig hozza a várt eredményt. Az ORACLE 11g egy új index típus létrehozásával ad megoldást a problémára. Ez az index a XMLIndex aminek, a segítségével, sokkal gyorsabban nyerhetünk ki töredékeket az XML dokumentumból.

- OBJECT RELATIONAL – az ORACLE 11g ezt a tárolási módját tartalom centrikusnak mondhatjuk. Ezt akkor ajánlott használni, ha a dokumentum egy adott részét akarjuk csak lekérdezni vagy módosítani. Ez a tárolási mód lényegében feldarabolja a dokumentumot, és ez azzal az előnnyel jár, hogy az adatbázis optimalizáló segítségével az XML töredékek lekérdezésére úgy alakítsa át az SQL lekérdezést, hogy az jóval gyorsabb legyen. Másik előnye ennek a tárolási módnak, hogy ez megszabadul a gyakran sok gondot okozó felesleges „whitespace" karakterektől. Ez többek között azzal az előnnyel jár, hogy így kevesebb adatot kell feldolgozni és ezáltal a feldolgozás sebessége is nőhet. Ennek a tárolási módnak az a hátránya, hogy nem hagyja sértetlenül a dokumentumot, a „whitespace" karakterek kihagyása által és többletmunkával jár a dokumentum feldarabolása és aztán az XML szerkezet újraépítése.
- BINARY XML Ez az új XML tárolási mód az XML Schema használatát teszi lehetővé, de használható nélküle is. Tárterület hatékonynak mondható. Mint ahogy a XMLDB Developer Guide szól: az XMLTYPE típusú adat előfeldolgozott állapotban van tárolva, amit kifejezetten XML adat tárolására fejlesztettek ki. Az OBJECT RELATIONAL tárolással szemben olyan előnnyel rendelkezik, hogy ebből hiányzik az XML elemző bit. A CLOB tárolási móddal szemben sokkal hatékonyabb a tárolás, a módosítás, az indexelés és az XML töredékek lekérdezése terén.

4.2.3 XMLTYPE függvények

Az ORACLE 9i verziójában két újabb végrehajtható SQL függvényt vezetett be, ez egyik az existsNode a másik az extract.
Az *existsNode* SQL függvény az XMLTYPE existsNode tagfüggvény segítségével van implementálva. Szintaktikája a következő:

```
existsNode (XMLTYPE_példány IN SYS.XMLTYPE,
    XPATH_sztring IN VARCHAR2) RETURN NUMBER;
```

Az *extract* egy XPATH kifejezést alkalmaz, és az eredményül kapott XML dokumentum töredéket XMLTYPEJavaként adja vissza.
Mivel az ORACLE 9i verzió óta már két újabb verziójú adatbázis kezelője is megjelent az ORACLE-nek, a függvények funkcionalitását megőrizve, újabb indexeket is tudnak használni és tovább lettek optimalizálva. Az XML adatok feldolgozása egy optimalizált, memóriában tárolt DOM fa alapján történik. Az XMLTYPE Statikus és tagfüggvényei:

XMLTYPE függvény	Leírás
`createXML(xmlval IN VARCHAR2)`	Ez egy statikus függvény, ami egy sztringből XMLTYPE példányt hoz létre, úgy hogy ellenőrzi, hogy az adat jólformált-e. Egyetlen paramétere van, egy sztring, aminek egy jólformált XML dokumentumot kell tartalmaznia. A paraméterként megadott adatnak megfeleltetett XML példánnyal tér vissza.
`createXML(xmlval IN CLOB)`	Ez egy statikus függvény, ami egy CLOB-ból XMLTYPE példányt hoz létre úgy, hogy ellenőrzi, hogy az adat jólformált-e. Egyetlen paramétere van egy jólformált dokumentumot tartalmazó CLOB. A paraméterként megadott adatnak megfeleltetett XML példánnyal tér vissza.
`existsNode`	Ez egy adott XPATH kifejezésre megadja, hogy a dokumentumra alkalmazva érvényes csomóponttal tér-e vissza vagy sem. Paraméterként egy XPATH kifejezést kap. Három visszatérési értéke van, ha 0 akkor az adott csomópont nem szerepel a dokumentumban, ha 1 akkor az adott csomópont szerepel a dokumentumban, ha NULL akkor az adott dokumentum üres.

`extract`	Ez a függvény egy XPATH kifejezést alkalmaz a dokumentumon, és az így kapott dokumentum töredéket XMLTYPE Java ként adja vissza. Bemenő paraméter egy XPATH kifejezés VARCHAR2Javaként. A visszatérési érték az eredményül kapott dokumentumtöredék XMLTYPE Java ként, vagy NULL, ha nem eredményez csomópontot az XPATH kifejezés.
`isFragment`	Ez a függvény megvizsgálja, hogy az adott XMLTYPE példány által reprezentált dokumentum töredék-e. Dokumentum töredékhez úgy juthatunk, hogy extract függvényt hajtunk végre, amely több csomópontot eredményez. A visszatérési érték 1, ha az adott példány egyetlen csomópontot vagy egy jólformált dokumentumot tartalmaz, egyébként 0.
`getClobVal`	A dokumentumot CLOB ként adja vissza. A Visszatérési érték egy szerializált XML dokumentumot tartalmazó CLOB, amit használat után fel kell szabadítani.
`getStringVal`	A dokumentumot sztring ként adja vissza. A Visszatérési érték egy szerializált XML dokumentumot tartalmazó sztring vagy szöveges csomópont esetén maga a csomópont. Ha a visszaadott csomópont méret meghaladja a 4000 bájtot akkor futási idejű hiba következik be.

`getNumberVal`	A XMLTYPE által mutatott numerikus adatot számként adja vissza. Numerikus értéket tartalmazó létező szöveges csomópontot kell tartalmaznia az adott XMLTYPE példánynak. Visszatérési érték az XMLTYPE példány által meghatározott szövegből létrehozott numerikus érték.
`updateXML`	Az ORACLE 11g verziójában lehetőség van arra, hogy módosítsuk az XML adatokat úgy, hogy nem kell az egész dokumentumot kicserélni. Ez a függvény egy XML csomópontot cserél ki egy másikra.
`insertChildXML`	Ez a függvény egy adott csomópont gyerekeként szúr be egy másikcsomópontot vagy attribútumot. Ez a függvény az ORACLE 11g verzió óta használható.
`insertXMLbefore`	Ez a függvény XML csomópontot, vagy csomópontokat szúr be egy adott XML csomópont elé. Ez a függvény az ORACLE 11g verzió óta használható.
`appendChildXML`	Ez a függvény egy adott csomópont utolsó gyermekeként szúr be egy új csomópontot. Ez a függvény az ORACLE 11g verzió óta használható.
`deleteXML`	Ez a függvény egy adott XML csomópontot töröl. Ez a függvény az ORACLE 11g verzió óta használható.

5 PDF generáló alkalmazás

5.1 Bemutatás

Az ORACLE XML támogatottságának tanulmányozása során egy olyan alkalmazást fejlesztettem ki, amely kihasználja az XML nyelvezetben rejlő lehetőségeket, ezzel egy lehetséges példával szolgálva a szinte végtelen számú felhasználási területre.
Egy olyan Java nyelven íródott webes alkalmazást készítettem, ami PDF dokumentumok generálását teszi lehetővé XML alapokon. Ezt úgy éri el, hogy minden tényleges adat, ami a dokumentumok előállításához szükségesek csupán egy XML dokumentumban elérhető. Ezek a dokumentumok pedig adatbázisban vannak letárolva.
A PDF dokumentumainkban lehetőség van dinamikusan változó mezők elhelyezésére, amiknek a segítségével egy adott típusú dokumentum vázhoz több az eredeti dokumentummal formailag megegyező de tartalmilag különböző, új PDF dokumentumot hozzunk létre.
A program lehetőséget ad arra, hogy egy adott dokumentumnak felderítse a dinamikusan változó mezőit, és ezekhez egy a felhasználó által könnyen kezelhető webes felületen vihessen fel adatokat.

5.1.1 Esetleges felhasználási területek

Az alkalmazás nem egy konkrét feladat megvalósítására született, inkább a technológiákban rejlő lehetőségek kiaknázására. Mivel nem voltak kötött követelmények, így a felhasználási területek száma igen nagynak mondható. Általánosságban véve egy olyan webes alkalmazás, ami szinte bármilyen weboldal PDF generáló motorja lehet. Mivel a programhoz készített felhasználói felület csak tesztelési céllal készült, igyekeztem az alkalmazás hátterét minél könnyebben hozzáférhetőnek és felparaméterezhetőnek kialakítani.
Ezzel egy olyan szerver oldali alkalmazás született ami, más weboldalak készítői könnyen használatba tudnak venni, mivel nem szükséges számukra, hogy tisztába legyenek azzal, hogy hogyan működik az XML dokumentumok feldolgozása, majd ezek összefűzése a dinamikus adatokkal és hogy hogyan megy végbe a PDF transzformáció. Semmi másra nincs szükségük

csak arra, hogy ismerjék, hogy milyen szolgáltatásra van szükségük, és ezt milyen HTTP linken és milyen paraméterekkel vehetik igénybe.
Az alkalmazás egy lehetséges és egy már megvalósult felhasználási területe egy olyan szerződésgeneráló program, ami egy adott szerkezetű szerződés szabadon hagyott részeit tölti fel automatikusan, úgy hogy a felhasználó csak azt választja ki, hogy milyen típusú szerződésre van szüksége, és hogy az adott típusú szerződés mely ügyfél adataival kerüljenek kitöltésre.
Az, hogy az adatok letárolása XML alapokon történik, egy olyan technológia előnyhöz juttatja ezt az alkalmazást a többi egyszerű „körlevélkészítő" programmal szemben, hogy az adatbázisban tárolt adatok indexelhetők, így a hozzáférhetőség nagyon gyorsan végrehajtható.
A fent említett szerződésgeneráló alkalmazásban például lehetőség van arra, hogy az adatbázisban tárolt adatok közül egy keresési feltétel segítségével kiválasszuk a számunkra fontosnak ítélt adatokat és ezekből generáljuk le, a dokumentumainkat.
Egy példát tekintve, nagyon egyszerűen megoldható hogy a szerződések keltére, ami egy dinamikusan változóadat egy indexet hozunk létre. Így ha szükségünk van az idei év Augusztus 10. napja óta megkötött szerződésekre, azokat nagyon gyorsan legenerálhatjuk, még akár úgy is, ha több száz van belőle.

5.1.2 Felhasználói felület

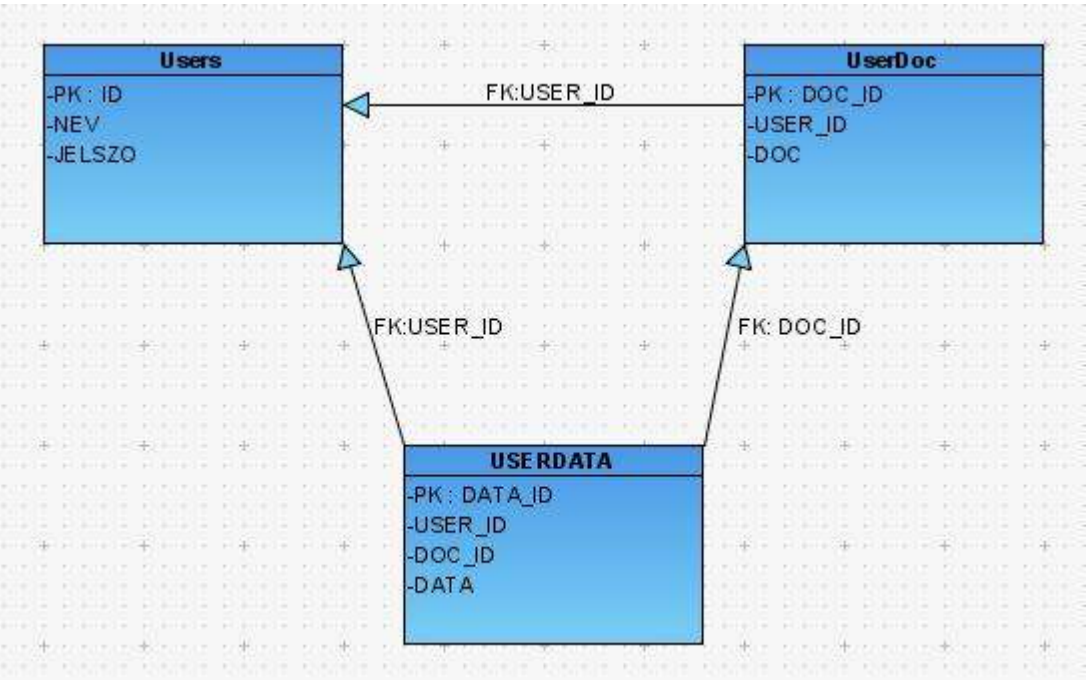
Az alkalmazásban több felhasználó kezelése megoldott úgy, hogy az egyes felhasználók nem láthatják egymás adatait. Így az alkalmazás nyitó oldala egy bejelentkezési képernyő, ahol egy felhasználói nevet és jelszót kér a program a felhasználótól. Amennyiben nem rendelkezik a felhasználó még érvényes hozzáféréssel így, lehetőség van saját felhasználói név és jelszó létrehozására. Miután átjutottunk a bejelentkezési képernyőn, három lehetőségünk van.
Lehetőség van új a felhasználó által létrehozott XML dokumentumok feltöltésére az adatbázisba, fontos hogy a feltöltött dokumentumok megfeleljenek a jólformáltság kritériumának, valamint az alkalmazás által értelmezhető XML adatot kell feltölteni, amiben az ehhez készített XML séma dokumentum válik segítségünkre.

További lehetőség, hogy az általunk feltöltött XML dokumentum dinamikus mezőihez adatokat vigyünk fel. Fontos hogy megtaláljuk, hogy mely mezők változnak a dokumentumban, és melyek maradnak változatlanul. A program teljesen automatikusan kérdez rá ezekre az adatokra, így a felhasználó könnyen vihet fel új adatokat. Ha olyan dokumentumot választottunk, amelyek nem tartalmaznak változó részt, ezt a program jelzi.
Az utolsó lehetőség, a PDF dokumentumok legenerálása. Ebben a menüpontban a feltöltött dokumentumok közül választhatunk valamilyen szűrési feltétel alapján, majd a kiválasztott dokumentumhoz, ha rendelkezik dinamikus mezővel, ki kell választanunk, hogy mely adatokkal történjen meg a PFD dokumentumok legenerálása, ha nem rendelkezik változó mezővel, akkor rögtön a dokumentum előállítása megy végbe

5.2 Technikai háttér

5.2.1 Az alkalmazás ORACLE háttere

A program adatbázisként az ORACLE 10g Express Edition változatát használja, az adatok tárolására három táblát használ fel, melyek közötti konzisztenciát az elsődleges és külső kulcsok megadásával értem el. A táblák szerkezete az alábbi ábrával szemléltethető:



Külön táblában vannak letárolva a felhasználó adatai, ez a tábla a „Users”, aminek egy elsődleges kulcsa van az „ID” és egy felhasználónév és egy jelszó tárolására szolgáló oszlop. Az elsődleges kulcs az adatbázis által van legenerálva egy szekvencia segítségével. A táblába való beszúrás előtt mindig lefut egy TRIGGER és ebből a szekvenciából a következő értéket kivéve mindig egyedi lesz az új felhasználó ID értéke.
A tábla definíciója az alábbi:

```
CREATE TABLE  "USERS"
   (   "ID" NUMBER NOT NULL ENABLE,
       "NEV" VARCHAR2(4000) NOT NULL ENABLE,
       "JELSZO" VARCHAR2(4000) NOT NULL ENABLE,
        CONSTRAINT "USERS_PK" PRIMARY KEY ("ID") ENABLE
   )
/

CREATE SEQUENCE   "USERS_SEQ"  MINVALUE 1 MAXVALUE
999999999999999999999999999 INCREMENT BY 1 START WITH 21 CACHE
20 NOORDER  NOCYCLE
/

CREATE OR REPLACE TRIGGER  "BI_USERS"
  before insert on "USERS"
  for each row
begin
    select "USERS_SEQ".nextval into :NEW.ID from dual;
end;
/

ALTER TRIGGER  "BI_USERS" ENABLE
/
```

A felhasználókhoz külön kell kezelni a feltöltött dokumentumokat, ezért ehhez a táblához egy külső kulcs segítségével van hozzárendelve a felhasználók adatait tartalmazó tábla. Elsődleges kulcsnak egy „DOC_ID” nevű oszlop lett létrehozva, és erre egy szekvencia.
A dokumentumok ebben a táblában „DOC” nevű XMLTYPE típusú oszlopban vannak letárolva. Ebben a táblában letárolt dokumentum dinamikus mezői még nincsenek letárolva. A dinamikus mezők tartalmának letárolása egy külön táblában történik.

```
CREATE TABLE  "USERDOC"
   (   "USER_ID" NUMBER NOT NULL ENABLE,
       "DOC_ID" NUMBER NOT NULL ENABLE,
```

```
      "DOC" "SYS"."XMLTYPE" ,
       CONSTRAINT "USERDOC_PK" PRIMARY KEY ("DOC_ID") ENABLE,
       CONSTRAINT "USERDOC_FK" FOREIGN KEY ("USER_ID")
        REFERENCES  "USERS" ("ID") ENABLE
   )
/

CREATE INDEX  "USERDOC_CTX1" ON  "USERDOC"
(SYS_MAKEXML("SYS_NC00004$"))
   INDEXTYPE IS "CTXSYS"."CONTEXT"
/

CREATE OR REPLACE TRIGGER  "BI_USERDOC"
  before insert on "USERDOC"
  for each row
begin
    select "USERDOC_SEQ".nextval into :NEW.DOC_ID from dual;
end;

/
ALTER TRIGGER  "BI_USERDOC" ENABLE
/
```

A dokumentumok dinamikusan változó mezőihez az adatok a „USERDATA” nevű táblában vannak letárolva. Fontos, hogy egy adott nyers adat csak egy felhasználóhoz és egy nyers fix dinamikus adatokat nem tartalmazó dokumentumhoz lehessen hozzárendelni, valamint ezek az adatok egyediek legyenek. Ezért két külső kulcs tartozik ehhez a táblához, az egyik a „USERS” táblához kapcsolja „USER_ID alapján”, a másik a „USERDOC” táblához a „DOC_ID” alapján kapcsolja külső kulcs segítségével.

```
CREATE TABLE  "USERDATA"
   (   "USER_ID" NUMBER NOT NULL ENABLE,
       "DOC_ID" NUMBER NOT NULL ENABLE,
       "DATA_ID" NUMBER NOT NULL ENABLE,
       "DATA" "SYS"."XMLTYPE" ,
        CONSTRAINT "USERDATA_PK" PRIMARY KEY ("DATA_ID")
ENABLE,
        CONSTRAINT "USERDATA_FK" FOREIGN KEY ("USER_ID")
         REFERENCES  "USERS" ("ID") ENABLE,
        CONSTRAINT "USERDATA_FK2" FOREIGN KEY ("DOC_ID")
         REFERENCES  "USERDOC" ("DOC_ID") ENABLE
   )
/
```

```
CREATE INDEX  "USERDATA_CTX1" ON  "USERDATA"
(SYS_MAKEXML("SYS_NC00005$"))
   INDEXTYPE IS "CTXSYS"."CONTEXT"
/

CREATE OR REPLACE TRIGGER  "BI_USERDATA"
  before insert on "USERDATA"
  for each row
begin
    select "USERDATA_SEQ".nextval into :NEW.DATA_ID from dual;
end;

/
ALTER TRIGGER  "BI_USERDATA" ENABLE
/
```

A fejlesztés során egy az XML lekérdezésekhez szükség volt egy olyan tárolt függvényre, ami egy adatbázisban letárolt XML dokumentumhoz tartozó fában megszámolja, hogy hány testvér csúcs van egy megadott ágon.

```
create or replace function getSorokSzama(s varchar2, tname
varchar2,id varchar2) return number is
     i number;
     ossz number := 0;
begin
     loop
     execute immediate 'select 1 from ' || tname || ' t ' ||
     'where  t.doc.existsnode('''||  s  ||  '''  ||  ''['' ||
     to_char(' || ossz || '+1) || '']'')=1 AND doc_id='|| id
     ||''
     into i;
     ossz := ossz + 1;
     end loop;
exception
     when no_data_found then return ossz;
end;
```

A fejlesztés során a tesztelés alatt olyan esettel is találkoztam, hogy túl sok folyamat próbált hozzáférni az adatbázishoz. Ez akkor fordult elő, amikor egy nagyméretű XML dokumentumot dolgoztam fel, és sok dokumentum töredéket kellett lekérdezni, a feldolgozás miatt.
A változtatások eléréséhez az egyszerre futtatható folyamatok számát kellet megnövelni, az alábbi módon [5].

Csatlakozni az adatbázishoz SQL*Plus-ból SYSDBA-ként.

```
alter system set processes=100 scope=spfile;
```

utasítással a folyamtok számának növelése 100-ra.
Az adatbázis újraindítása. (`shutdown immediate` és a `startup` utasítással)
Az futtatható folyamatok maximális számának lekérdezése (`SHOW PARAMETER PROCESSES`)

5.2.2 Az alkalmazás XML háttere

Az alkalmazáshoz egy új XML dokumentumot fejlesztettem, aminek a segítségével könnyen le lehet írni egy átlagos szöveges dokumentumot. Ehhez az XML dokumentumhoz készítettem egy XML Schema dokumentumot is, ami az XML dokumentumok előállításakor ad sok segítséget.
A nyers XML kétféle szöveges objektumot tud leírni, amiből később a PDF dokumentumot könnyen elő lehet állítani. Az egyik a sima szöveges adat, a másik a táblázat. Mindkettő objektumnak kötelezően egy „**sor**” nevű elemben kell elhelyezkednie. A „**sor**” elemből tetszőlegesen sok szerepelhet. A kész dokumentum pedig sorokból épül fel. Egy szöveges sor az alábbi módon néz ki.

```
<sor>
    <szoveg betu_tipus="Normal" dinamikus="false" igazitas="Bal" meret="10">Első
    sor
    </szoveg>
</sor>
```

A „**szoveg**” elem értéke lesz a megjelenítendő szöveg, az attribútumai pedig a formázást adják meg. A „**dinamikus**” attribútum értéke adja meg, hogy az adott mező dinamikus-e vagy sem. Ha igen akkor az értéke „true”, ha nem akkor „false”. A „**meret**” attribútum értéke a betűméretet adja meg, ez az érték *6* és *30* közötti lehet. Az „**igazitas**” attribútum a szöveg igazítását adja meg, szabályos értékei: „*Bal*”, „*Jobb*”, „*Kozep*” és „*Sorkizart*”. A „*betu_típus*” attribútum a kiírandó szöveg betű típusát adja meg, szabályos értékei: „*Normal*”, „*Felkover*”, „*Alahuzott*”, „*Dolt*”, „*FelkoverAlahuzott*”, „*FelkoverDolt*”, „*AlahuzottDolt*” és „*FelkoverAlahuzottDolt*”.

Egy táblázatot leíró sor az alábbi módon néz ki.

```
<sor>
    <tabla oldalszel_szazalekaban="100" tablaszelesseg_egysegben="5">
        <cella betu_tipus="Normal" szelesseg="1" szegely="Alul" meret="10"
        igazit="Jobb" dinamikus="true">Cella1</cella>
        <cella betu_tipus="Normal" szelesseg="4" szegely="Alul" meret="10"
        igazit="Jobb" dinamikus="true">Cella2</cella>
    </tabla>
</sor>
```

Egyetlen „**sor**” elemen belül csak egy „**tabla**” elem szerepelhet, de egy „**tabla**” elemen belül tetszőleges számú „**cella**” elem szerepelhet.

A „**tabla**” elem két attribútummal rendelkezik, az egyik a „**oldalszel_szazalekaban**” ami *1* és *100* közötti értékkel megadja, hogy a táblázat szélessége az oldal szélességének hány százaléka, a másik a „**tablaszelesseg_egysegben**” ez azt adja meg, hogy hány egységre osztjuk fel a táblázatot. Ez az értéket a cella szélesség megadáskor használjuk.

A „**cella**” elem nagyban hasonlít a fent tárgyalt „**szoveg**” elemre, de ennek még van két további attribútuma, az egyik a „**szelesseg**” ami cella szélességét adja meg felhasználva a „**tablaszelesseg_egysegben**” attribútum értékét úgy hogy ennek az értékénél nem lehet nagyobb és az egy sorba kerülő cellák szélességének összege pontosan, ezzel az értékkel kell megegyeznie. A másik attribútum a „**szegely**” ami azt adja meg hogy milyen szegélyel szerepeljen az adott cella. A következő értékeket veheti fel: „*Bal*”, „*Jobb*”, „*Alul*”, „*Felul*”, „*Box*” és „*Semmi*”.

Az PDF dokumentumok vázát ez az XML dokumentum adja. Amikor egy nyers dokumentumot töltünk fel, akkor lényegében csak egy XML leírást adunk meg a program számára. Mivel ilyenkor bármilyen szerkezetű a program számára értelmezhetetlen adat lenne felvihető, ezért az alkalmazáshoz készült egy XML Schema dokumentum is. Ez egyfajta biztosíték az alkalmazás számára. A séma megvédi az alkalmazást a számára értelmezhetetlen, vagy rosszul strukturált adatoktól.

Tekintsük az alábbi XML séma leírást a fenti XML struktúrához:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="dokumentum">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="sor" minOccurs="1"
                    maxOccurs="unbounded" type="sor_tipus"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="sor_tipus" >
        <xsd:sequence>
            <xsd:choice>
            <xsd:element name="szoveg" type="szoveg_tipus"
                    minOccurs="0" maxOccurs="1"/>
            <xsd:element name="tabla" type="tabla_tipus"
                    minOccurs="0" maxOccurs="1"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

Az a kódrészlet az XML struktúra csak egy részletét írja le. A kód első fele azt írja le, hogy az XML fa gyökere egy „*dokumentum*” nevű elem. A további leírásból az derül ki, hogy a dokumentum elemnek legalább egy „*sor*” elemmel kell rendelkeznie. A maximális számra nincs megkötés. A kódrészletből az is kiderül, hogy egy új típust hoztam létre a „*sor*” elemek elírására.
A „*sor_tipus*” eleme kétféle lehet. A gyermek elem neve vagy „*szoveg*” lehet vagy „*tabla*”. Erre a két típusra is van egy megkötés, egy soron belül vagy csak egy „*szoveg*” elem vagy csak egy „*tabla*” elem szerepelhet. A kódból az is kiderül, hogy ennek a két elemnek a leírására, két külön típus lett létrehozva.
Látható, hogy ez a kódrészlet első ránézésre kicsit nehezen átlátható, de ez a leírás nagy előnye, hogy minden részletet pontosan megadhatunk az XML struktúráról.

5.2.3 Az alkalmazás Java háttere

Az alkalmazás lényegében egy Szervlet. A Szervlet lényegében egy speciális Java program, ami a webszerverrel együttműködve a szerver oldalon lehetővé teszi HTML oldalak dinamikus létrehozását, illetve paraméterezését például HTTP protokollon keresztül. A Szervletekkel lényegében a webszerver funkcionalitását növelhetjük tovább, mivel ha a kliens egy olyan weboldalt kér a webszervertől, amelyet egy Szervlet állít elő, akkor az csak delegálja a kérést a Szervlet felé, majd a Szervlet által generált oldalt továbbítja a kliensnek.
A Szervletek többek között olyan előnyökkel rendelkeznek, mint a webszerveren állandóan futó virtuális gép sokkal gyorsabb kiszolgálást eredményez.
A *javax.servlet* csomag tartalmazza az összes szervletspecifikus osztályt és interfészt. Minden Szervlet a *javax.servlet.genericServlet* leszármazottja, vagy a *javax.servlet.Servlet* interfészt implementálja. A HTTP protokollt használó Szervlet mindig a *javax.servlet.http.HttpServlet* leszármazottjai. A program használata során ezt a Szervletet használtam.
A Szervlet paramétereit POST-olt változókban lettek átadva, ezeket a Java oldalon nagyon egyszerű kiolvasni, csak a *HttpServletRequest getParameter*(*attribútum név*) metódusát kell használni. Amikor egy állományt adunk át a Szervletnek paraméterként az egy HTTP formban az alábbi módon lehet megoldani.

```
<form method=”post” action=”loaddata” enctype=”multipart/form-data”>
        <input type=file name="file" size="40"><br>
        <input type=submit><br>
</form>
```

Ami számunkra lényeges az „**enctype=multipart/form-data”**. Java oldalon az ilyen paraméterek beolvasása az alábbi módon történik:

```
String fileName = (String) request.getParameter("file");
boolean isMultipart =
                ServletFileUpload.isMultipartContent(request);
FileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload upload = new ServletFileUpload(factory);
List /* FileItem */ items = upload.parseRequest(request);
Iterator iter = items.iterator();
while (iter.hasNext()) {
     FileItem item = (FileItem) iter.next();
     xml_to_store=new String(item.get(),"utf-8");
}
```

Az így megkapott XML dokumentumokat az adatbázisban tárolja le a program. Az így megkapott XML dokumentumot sztring ként átadva egy INSERT SQL utasításnak válik beszúrhatóvá. Azonban ha ennek a sztringnek a mérete meghaladja a 4000 bájtot, az SQL utasítás nem hajtható végre, mivel az XMLTYPE createXML(XMLdata IN VARCHAR2) VARCHAR2 vár paraméterként, aminek a maximális mérete 4000 bájt. A megoldás az, hogy az XMLTYPE azon createXML(XMLdata IN CLOB) metódusát használjuk, amelyik CLOB ként várja a bemenő paramétert. Így Java oldalon kell azt megoldani, hogy a letárolni kívánt sztringben lévő XML dokumentumot CLOB típusúvá alakítjuk, és ezt adjuk át az SQL utasításnak. Az ezt leíró Java kód a következő:

```
private CLOB getCLOB(String xmlData, Connection conn) throws
SQLException{
        CLOB tempClob = null;
        try{
create new tempClob = CLOB.createTemporary(conn, true,
CLOB.DURATION_SESSION);
            tempClob.open(CLOB.MODE_READWRITE);
            Writer tempClobWriter =
 tempClob.getCharacterOutputStream();
            tempClobWriter.write(xmlData);
            tempClobWriter.flush();
            tempClobWriter.close();
            tempClob.close();
        } catch(SQLException sqlexp){
            tempClob.freeTemporary();
            sqlexp.printStackTrace();
        } catch(Exception exp){
            tempClob.freeTemporary();
            exp.printStackTrace();
        }
        return tempClob;
    }

    public void insertXML(String xmlData, Connection
conn,PrintWriter out,String query) {
        CLOB clob = null;
        PreparedStatement pstmt = null;
        try{
            pstmt = conn.prepareStatement(query);
            clob = getCLOB(xmlData, conn);
            pstmt.setObject(1, clob);
            if (pstmt.executeUpdate() == 1) {
                ;
```

```
                out.println("Successfully inserted!");
            }
        } catch(SQLException sqlexp){
            out.println(sqlexp);
            sqlexp.printStackTrace();
        } catch(Exception exp){
            out.println(exp);
            exp.printStackTrace();
        }
    }
```

Az `insertXML` eljárás négy paramétert vár, az első beszúrandó XML adat, a második egy Connection ami, az adatbázissal való kapcsolatot építi ki, a harmadik egy Printwriter, ami a megadott kimenetre ír ki üzenetet, vagy hiba esetén kivételt, a negyedik paraméter egy SQL utasítás, amiben egy INSERT utasítás van megadva.

```
INSERT INTO PDF_GEN (PRINT) VALUES (XMLTYPE(?))
```

A „?” helyére kerül beírásra azaz XML adat amit a

```
CLOB getCLOB(String xmlData, Connection conn)
```

függvény segítségével kapunk.

A PDF generálás az ingyenes PDF generáló iText API [10] segítségével történik. Ez az API teljesen ingyenes, és a használatához szükséges összes csomag letölthető a *http://www.lowagie.com/iText/* címről.
Az API használata rendkívül egyszerű. A PDF dokumentum egy *Document* típusú objektumban lesz letárolva. Ennek az objektumnak a konstruktora egy oldalméretet és a négy margót várja paraméterként. Ehhez a *Document* objektumhoz egyszerűen adhatunk hozzá elemeket. Elemek alatt, itt szöveget tartalmazó sorokat, vagy táblázatot értek.

Egy szöveget tartalmazó sor hozzáadására a *Document* típusú objektum *add*(*Paragraph* p) metódusát használja a program, ami egy *Paragraph* típusú objektumot vár paraméterként. A *Paragraph* konstruktora egy *Stringet* és egy *Fontot* vár paraméterként ahol a *String* tartalmazza a kiírandó szöveget tartalmazza. A paraméterként várt Font írja le a kiírandó szöveg stílusát és méretét. A *Paragraph setAlignment* metódusa a szöveg igazítását adhatja meg.

```
Document pdf_document =
new Document(PageSize.A4, 40, 40, 40, 40);
Font font =
new Font(BaseFont.createFont("TIMES.TTF",
BaseFont.CP1250, BaseFont.NOT_EMBEDDED),size);
Paragraph sor=
new Paragraph(”Szöveg”,font);
sor.setAlignment(Element.ALIGN_RIGHT);
pdf_doc.add(sor);
```

A Document add metódusa nem csak szöveget leíró *Paragraph* objektumot kaphat paraméterként, hanem táblázat leírására szolgáló *PdfPTable* objektumot is.

```
PdfPTable tabla = new PdfPTable(oszlopszam);
Font font =
new Font(BaseFont.createFont("TIMES.TTF",
BaseFont.CP1250, BaseFont.NOT_EMBEDDED),size);
PdfPCell cella =
new PdfPCell(new Paragraph(data,font));
cella.setColspan(szelesseg);
cella.setBorder(Rectangle.NO_BORDER);
cella.setHorizontalAlignment(Element.ALIGN_RIGHT);
tabla.addCell(cella);
```

A *PdfPTable* konstruktora egy oszlopszámot vár paraméterként. Ehhez a *PdfPTable* objektumhoz tetszőleges számú *PdfPCell* cellát adhatunk hozzá. A *PdfPCell* egy Paragraph

objektumot vár paraméterként, aminek az előállítása teljesen hasonló módon történik, mint a szöveges tartalom esetén. Ennek a *PdfPCell* objektumnak vannak olyan beállító metódusai, mint *setColspan,* aminek a segítségével a cella szélességét lehet beállítani, a *setBorder,* aminek a segítségével a cella szegélyét lehet beállítani vagy *setHorizontalAlignment,* aminek a segítségével a cella tartalmának igazítását lehet beállítani.

5.3 XML lekérdezések tesztelése Java és ORACLE oldalról

Az alkalmazás fejlesztésekor két különböző módszert hasonlítottam össze, az XML adatok lekérdezésére. Az egyik az ORACLE oldali SQL utasítás belüli lekérdezések, a másik a Java oldali lekérdezések egy DOM fa legenerálásával, és ennek a fának a bejárásával.
Az alkalmazást kiegészítve egy ilyen tesztelővel, összehasonlítottam a két lekérdezést, melyik dolgozik gyorsabban és hatékonyabban. Valamint melyik lekérdezés melyen korlátokkal rendelkezik.

5.3.1 XML lekérdezések Java oldalról

A lekérdezések Java oldalon a DOM fa felépítésével történik, majd a lekérdezés ezen a DOM fán operál. Ha több adatot akarunk lekérdezni, akkor az adatbázishoz egy alakalommal nyúl, amikor felépíti a DOM fát. A későbbiekben, ha újabb lekérdezésre van szükség, akkor nem épül fel kapcsolat az adatbázissal, hanem a keresés csak a DOM fában megy végbe. A DOM fa felépítése az alábbi módon történik:

```
Connection con = new dbConnection().getConnection();
String utasitas = "select XMLTYPE(d.doc.getclobval()) from
              userdoc d where doc_id='"+doc_id+"'";
PreparedStatement stmt = con.prepareStatement(utasitas);
ResultSet rs = stmt.executeQuery();
rs.next();
org.w3c.dom.Document xml_db =
((XMLTYPE)rs.getObject(1)).getDOM();
```

```
NodeList dbnl=xml_db.getChildNodes();
```

A SELECT utasításban az egész dokumentumot elkérjük az adatbázistól CLOB típusú változóként, majd Java oldalon ezt XMLTYPE típusú objektummá konvertálva elérhetővé válik az XMLTYPE getDOM() metódusa, ami egy DOM dokumentumot ad vissza. Későbbiekben ezt a DOM dokumentumot kell bejárnunk a lekérdezések során. Az alábbi kód a gyökér elem címkéjét kérdezi le.

```
dbnl.item(0).getLocalName();
```

A későbbiekben ezt a fát felhasználva könnyen bejárhatóvá válik az egész XML dokumentum.

5.3.2 XML lekérdezések ORACLE oldalról

Az alkalmazás az adatbázis oldalon is képes kinyerni adatokat egy XMLTYPE ként tártolt XML dokumentumból. Ezek sokkal egyszerűbben hajthatók végre a lekérdezések, ugyanis mindent megold helyettünk az ORACLE. Az egész lekérdezés beágyazható egyetlen SQL utasításba, így a fejlesztőnek csak az értékes adat lekérdezését kell megoldania. Egy „*dokumentum*” gyökérrel rendelkező második „*sor*” gyermeknek „*szoveg*” gyermekének a „*meret*” attribútuma az alábbi módon érhető el.

```
SELECT D.DOC.EXTRACT('/dokumentum/sor[2]/szoveg/@meret') FROM
USERDOC D;
```

5.3.3 Összehasonlítás

Az XML adatok lekérdezésére az adatbázisból, több lehetséges módszer van. Az előző két pontban tárgyalt módon lehetőség van az adatok kinyerésére csupán az adatbázis oldalról, de lehetőség van erre a Java oldaláról is. A két lekérdezés gyakorlatilag ugyanazt tudja megoldani, bár az ORACLE-beli lekérdezések egyszerűbbnek tűnnek.

Mivel nem lehetett pusztán technikai oldalról eldönteni, hogy melyik technológia a hatékonyabb, ezért elkészült mindkét verzió, és összehasonlítottam, hogy melyik verzió a hatékonyabb ennél az alkalmazásnál.
Tekintsük az alábbi két lekérdezést:

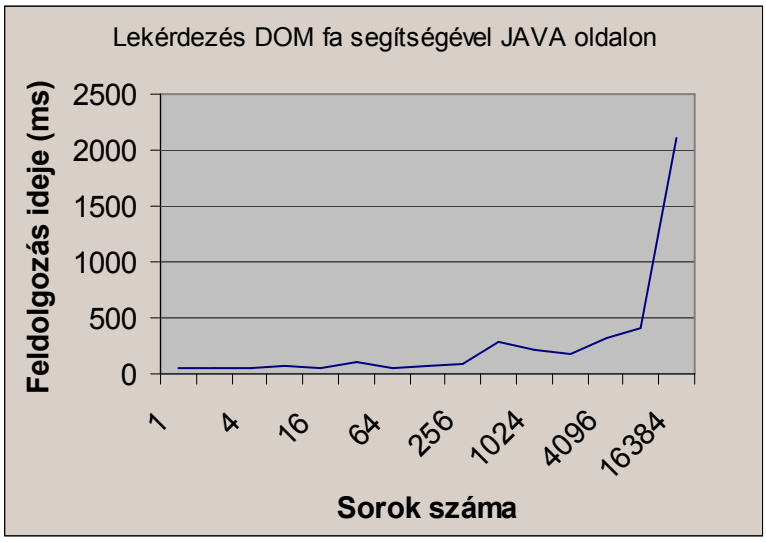
```
SELECT  D.DOC.EXTRACT('/dokumentum/sor[2]/szoveg/@meret')  FROM
USERDOC D;
```

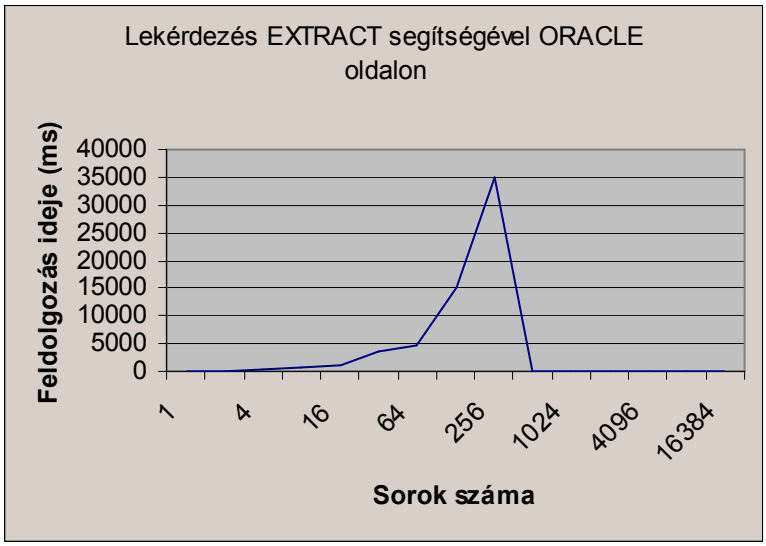
Ugyan ezt Java oldalon sokkal bonyolultabb megoldani.

```
dbnl.item(0).getChildNodes().item(1).getFirstChild().
     getAttributes().getNamedItem("meret").getNodeValue();
```

Látható hogy ugyan ez a lekérdezés Java oldalon sokkal bonyolultabb és nehezebben átlátható.

Tekintsük az alábbi két diagramot:





A diagramban szereplő „Sorok száma” a XML dokumentumok bonyolultságát jelöli, pontosan azt hogy a PDF dokumentumban egy sor előállítása mennyi időbe kerül.
A lekérdezések idejének a mérésére Java oldalon történik, az alábbi kód segítségével:

```
Calendar naptar = Calendar.getInstance();
int start=naptar.getTimeInMillis();
getxml_doc_from_db_with_select(user_id,doc_id,data_id,txtout,p
df_document);
int eltelt_ido=start - naptar.getTimeInMillis();
```

Az adatbázis oldali lekérdezés abból a szempontból előnyösebb, hogy könnyebben átlátható és nem igényel semmilyen extra memóriát a dokumentum fa felépítéséhez.

5.3.4 Konklúzió

A két diagramból kiderül, ha sok dokumentum töredéket kell lekérdezni az adatbázis oldali lekérdezések nem mondhatók hatékonynak, sőt ha túl sok dokumentum töredéket próbálunk egyszerre lekérdezni, akkor könnyen túlterhelhetjük vele az adatbázist. A második diagramon látszik, hogy 512 sor feletti lekérdezés után már olyan sok kapcsolat keletkezik az adatbázis és a webszerver között, hogy az leáll. Erre a problémára több megoldás is van, az egyik a

folyamatok maximális számának a növelése, vagy a lekérdezések között szünetek beiktatása, hogy legyen ideje az adatbázisnak a még nyitva lévő, de már nem használt kapcsolatok lezárására

A Diagramból az is kiderül, hogy nem érdemes az ilyen jellegű problémákkal foglakozni, mert ha sok dokumentum töredékkel kell dolgozni, akkor sokkal hatékonyabb a Java oldalon felépíteni egy DOM fát és ebben keresni mivel látható, hogy míg ORACLE oldalon lekérdezve 16 dokumentum töredéket az körülbelül 1500ms ideig tartott, ugyan ennyi idő alatt java oldalon 16384 dokumentum töredék került feldolgozásra.

A teszt eredmények lényegében alá támasztják azt, amit az XML dokumentumok tárolási jellemzőinél tárgyaltam. A CLOB ként tárolt dokumentum töredékek lekérdezése nem hatékony, még úgy is ha indexeket hozunk létre az ilyen adatok lekérdezésére. Ehelyett érdemesebb valamilyen más tárolási módot választani, például az OBJECT RELATIONAL modellt.

Csak akkor érdemes az adatbázis oldalon végrehajtani a lekérdezéseket, ha csak egy dokumentum töredékre, vagy egész dokumentum lekérdezésére van szükségünk.

5.4 További fejlesztési lehetőségek

A további fejlesztési lehetőségek közt olyan lehetőségek szerepelnek, hogy nyers dokumentumot tartalmazó XML előállítására ne csak egy séma dokumentum legyen segítségünkre, hanem egy olyan felhasználói interfész ahol a végfelhasználó egy szövegszerkesztőhöz hasonló felületen készítheti el a dokumentumait és az XML dokumentumok legenerálása a háttérben, teljesen automatikusan menjen végbe.

További lehetőségek közt szerepel olyan tesztek elkészítése, amelyek az ORACLE 11g lehetőségeit tárja fel.

Idő hiányában nem valósult meg, de egyszerűen elkészíthető lenne egy olyan kapcsolat egy felhasználókat, és adataikat tartalmazó adatbázissal, amiből adatokat lekérdezve be lehetne mutatni egy szolgáltató számlázását, amit minden hónap végén kiküld az ügyfelei részére.

További lehetőségek között szerepel egy webszolgáltatás elkészítése. Ez egy olyan alkalmazáslogika, amely adatokat és szolgáltatásokat biztosít más programok számára. Ez lényegében egy olyan felület, amihez tartozik egy formális XML leírás, ennek segítségével

leírjuk, hogyan vehetjük igénybe a webszolgáltatást. A felület elrejti a szolgáltatás megvalósításának részleteit, így független marad az azt megvalósító hardver és szoftver környezettől. Ha valaki egy webszolgáltatást akar használni, nem kell mást tennie csak megszerezni a használatához szükséges leírást.

6 Összefoglalás

A diplomamunkámban bemutattam a ma már széles körben elismert és használt XML nyelvezetet. Áttekintettem a nyelvezet alapjait, valamint bemutattam két elterjedt XML-t leíró API-t, az egyik a DTD a másik az XML Schema. Ezek a leíró nyelvek egyfajta védelmi vonalként foghatók fel az alkalmazás és a külvilág között, mivel megakadályozzák, hogy rosszul strukturált vagy szabálytalan adatokat adjunk meg bemenetként.

Bemutattam az ORACLE XML támogatottságát, hogy az XML dokumentumok tárolására milyen új típust vezetett be, valamint ezen típus, az XMLTYPE milyen új tárolási jellemzőket vezettet be a nem rég megjelent legújabb ORACLE verzióban, a 11g-ben. Bemutattam, hogy milyen lehetőségek állnak számunkra az adatbázis oldalon arra, hogy lekérdezzük, módosítsuk vagy töröljük az XML dokumentumainkat.

Nagyon érdekes volt megnézni, hogy menyiben változhat az XML feldolgozás sebessége, attól függően, hogy milyen tárolási jellemzőt választottunk valamint, hogy bizonyos műveletek, mint az XML darabonkénti módosítása nem mindegyik modellnél elérhető.

Az XML technológiát bemutattam a fejlesztői oldalról is. Milyen reprezentációban van jelen egy ilyen dokumentum, és a két XML feldolgozó API segítségével, hogyan lehet hozzáférni programozói oldalról.

Végül bemutattam az általam készített PDF generáló alkalmazást, amiben szemléltettem az XML technológia kihasználhatóságát fejlesztői és adatbázis oldalról. Ez az alkalmazás egy olyan XML alapokon dolgozó PDF dokumentumgeneráló program, amiben a felhasználó egyszerűen helyezhet el dinamikusan változó részeket, azáltal hogy egy dokumentum vázat ad, és a későbbiekben a változó részekhez egyszerűen tud felvinni adatokat, így lehetősége nyílik arra, hogy akár több száz azonos vázzal rendelkező, de különböző tartalmú adatot, tartalmazó dokumentumot állítson elő rendkívül gyorsan.

Az alkalmazás nem csak bemutató céllal készült, hanem hogy teszteljem, hogy az XML lekérdezéseket mikor hol érdemes végrehajtani, ugyanis erre lehetőségünk van mind szerver, mind fejlesztői (Java) oldalon is.

Érdekes volt megfigyelni, hogy több dokumentumtöredék lekérdezése Java oldalon, míg egyetlen dokumentumtöredéké inkább adatbázis oldalon hatékonyabb, mivel nem kell felépíteni a dokumentumot leíró fát.

Azt is fontos megemlíteni, hogy a Java oldalon hatékonyabban felépíthető fa kezelése, bejárása sokkal bonyolultabb és nehezebben átlátható, mint az adatbázis oldalán.

7 Irodalomjegyzék

[1] Gábor András, Gunda Lénárd, Juhász István, Kollár Lajos, Mohai Gábor, Vágner Anikó,
Az ORACLE és a web, 2003

[2] Michael Morrison,
Tanuljuk meg az XML használatát, 2006

[3] Chris Bates,
XML elmélet és gyakorlat, 2003

[4] Nyékiné G. Judit,
Java 2 útikalauz programozóknak, 2001

[5] Christoper Jones, Alison Holloway,
The Underground PHP and ORACLE Manual, 2007

[6] M. Gralike,
ORACLE 11g- XMLTYPE Storage Options, 2007

[7] Document Object Model
http://www.w3.org/DOM/

[8] XML Schema
http://www.w3.org/XML/Schema

[9] XML Path Language
http://www.w3.org/TR/xpath

[10] iText API
http://itext.ugent.be/library/api/

8 Függelék

FELDOLGOZÁSI HATÉKONYSÁG TESZTELÉSE		
Sorok száma	Lekérdezés DOM fa segítségével Java oldalon	Lekérdezés EXTRACT segítségével ORACLE oldalon
1	47 ms	62 ms
2	47 ms	109 ms
4	47 ms	469 ms
8	78 ms	641 ms
16	47 ms	1028 ms
32	109 ms	3453 ms
64	62 ms	4797 ms
128	78 ms	14984 ms
256	94 ms	35000 ms
512	281 ms	NINCS ADAT
1024	218 ms	NINCS ADAT
2048	171 ms	NINCS ADAT
4096	313 ms	NINCS ADAT
8192	406 ms	NINCS ADAT
16384	2109 ms	NINCS ADAT

9 Köszönetnyilvánítás

Sokan vannak, akik tapasztalataik átadásával segítették ezen munka elkészítését. Köszönettel tarozom Dr. Juhász Istvánnak a programozási ismeretek elsajátításáért, Jeszenszky Péternek az XML technológia használata során szerzett tapasztalok átadásért. Külön köszönettel tartozom témavezetőmnek Kollár Lajosnak, aki sokat segített a program fejlesztése során a tanácsaival, ami végül segített a diplomamunkám elkészítésében.