Debreceni Egyetem

Informatika Kar

XML adatbázis kezelésének lehetőségei

Témavezető:
Dr. Adamkó Attila
egyetemi adjunktus

Készítette:
Buka Balázs
programtervező informatikus

Debrecen
2010

Tartalomjegyzék

Bevezetés

Amikor szóba kerül, az XML mint technológia, a legtöbb informatika világában minimálisan jártas ember tekintete előtt egy HTML szerű nyelv jelenik meg, melyet leggyakrabban alkalmazások konfigurációs beállításának tárolására használnak. De valójában jóval többet hordoz magában. Hiszen XML dokumentumban tárolt adataink konzisztens megjelenítésére, segítségül szolgál az XML stíluslap definíció, mellyel formázhatjuk adatinkat. Az XML dokumentumok létrehozására is többféle technológia áll rendelkezésünkre, az egyszerűbb a DTD technológia, míg a komplexebb az XSD. Az utóbbi XML alapokon nyugvó kollekciók, illetve objektum típusoknak megfeleltethető adattárolást tesz lehetővé. Ennek következtében már körvonalazódik, hogy többek között adatbázis funkcionalitást is képes betölteni az XML technológia. És még számos kevésbé ismert technológia létezik, melyeknek nagytöbbsége XML nyelven íródik. Ugyanakkor ebből következik, hogy léteznek olyan XML nyelven írt rendszerek, melyek már az adott technológia határait feszegetik, és esetleg már nem is olyan hatékonyak. Ettől függetlenül az XML technológiák teljes mértékben tekinthetők platfrom független technológiáknak.

Fő témaként, az XML-ben rejlő adatbázis képességeket emelném ki, az egyéb technológiák mellett. Szeretném körvonalazni a natív XML adatbázis adta lehetőségeket, és ugyanakkor bemutatni az XML technológiát egy modern relációs alapú adatbázis rendszerben. Ennek demonstrálásához az Oracle adatbázis kezelő rendszert választottam. Természetesen mind a két féle megközelítésnek léteznek korlátai, teljesítményi és megvalósítási egyaránt.

Több technológia létezik, az XML dokumentumban tárolt adatok elérésére, szűrésére, vagy akár módosítására is. Ilyen például az XPath illetve az XQuery. Ezek alapvetően natív adatbázisokhoz lettek kifejlesztve, viszont az Oracle kiválóan alkalmazza rendszerében, saját beépített programcsomagjaival összhangban. Mivel az alkalmazásom Oracle alapokon nyugszik, próbálom bemutatni e programcsomagok nagy részét, hiszen alkalmazhatóságban közel olyan szinten állnak, mint a natív technológiák. És főleg itt tudok rávilágítani az általam felfedezett és hiányosságnak vélt problémákra.

Az internet egyre nagyobb ütemű, és szélesebb körű elterjedésével az XML egyre nagyobb teret kap. Fontosnak tartottam, e technológiák valamilyen gyakorlati alkalmazását, hogy saját kézből tapasztalhassam a lehetőségeket. Illetve hogy milyen szintű a támogatottsága egy-egy napjainkban ismertebb komponens csomagban, technológiában. Az alkalmazásomban Java Enterprise technológiát alkalmazó webes alkalmazást fejlesztettem, Java Server Faces megjelenítési réteggel, és Hibernate adatbázis kezelő komponens segítségével. Ezek a fajta alkalmazások is egyre több helyen fellelhetők, melynek ésszerű okai vannak. Többek között nem igényelnek külön telepítést, valamint a hasonló technológiák fejlődésével, vetekednek az asztali alkalmazások megjelenítési, stabilitási tulajdonságaival.

Natív XML adatbázis

XML

Az XML az Extensible Markup Language szavak rövidítése. Ebből nyilvánvalóvá válik, hogy az XML kiterjeszthető és változtatható. Lehetővé teszi a változtatható adatok tárolását weblapoknál, amelyek bármikor módosíthatóak. Ebből adódik a lehetőség a honlapok generálására futás közben. Leggyakrabban az XML-t a HTML-hez (Hypertext Markup Language) hasonlítják, mivel leginkább egy kiterjesztett HTML Form-hoz hasonlítható. De ez a nyelv nem teszi lehetővé a honlapok változtatását, effektíve megragadtak abban az időben mikor készítették őket, és nem engedélyeznek semmiféle változást a böngészőben való megnyitás közben.

Bármilyen információt elhelyezhetünk XML oldalakon. Az XML inkább maga az adat, minthogy egy tároló az adatokhoz. Ugyanakkor integrálható honlapokba, és leginkább adat szervezésűnek nevezhető. Teljesen flexibilis, nem egy cég vagy vállalat birtokolja a szabványt, és definiálja, hogy milyen tagokat használhatunk. Talán egy teljesen generikus programozási nyelvhez hasonlítható. Az adatok sorrendje lényegtelen, hiszen teljesen mindegy hogy milyen adat jelenik meg egy XML oldalban, ha használunk XSL-hez hasonlót. Az XSL egy formázó nyelv, amely segít rendezni a következetesen ismétlődő adatokat az XML-ben. (Extensible Style Sheets)

DTD

A Documnet Type Definition lehetővé teszi az XML dokumentumok érvényesítését. Segítségével létrehozhatunk például egy vállalat minden dokumentumával konzisztens struktúrát. Képes biztosítani egy strukturális érvényesítő metódust, amely természetesen bármilyen adatbázis fontos részét képezi. Mindamellett feleslegessé is válhat, ez főként az XML fájlok létrehozásától, generálásától függ. Ha manuálisan lettek létrehozva, akkor hasznos lehet egy DTD-hez hasonló eszköz. Persze statikus adathoz elég egyszeri érvényesítés. Statikus adatnak nevezzük azt az adatot, ami ritkán változik az adatbázisban, vagy egyáltalán nem. Tranzakciós vagy dinamikus adat, ami rendszeresen módosul. De

valószínűleg ezeket az XML fájlok alkalmazás generáltak lesznek, ilyenkor felesleges a DTD általi érvényesítés hisz az alkalmazás megteszi ezt.

XML szintaxis

Az XML szintaktikai szabályai nagyon egyszerűek, de ugyanakkor nagyon szigorúak is. Az első sor leírja az éppen használatban lévő verzió számot.

```
<?xml version=”1.0”?>
```

A következő sor tartalmazza az XML fa struktúrájának gyökér elemét, amely tartalmazza az össze többit, ami a dokumentumban megtalálható, közvetlenül vagy közvetetten. Az utolsó sornak pedig tartalmaznia kell a gyökér elem záró tagját.

```
<root>
     ..
</root>
```

Minden XML elemnek szüksége van záró tagra, ennek elhagyása hibát okoz. Az elemeknél különbséget teszünk, kis és nagy betűk között. Lehetnek attribútumaik, amelyek finomítják az elem aspektusát. Az attribútumokat és értékeit név-érték párosoknak nevezzük. több ilyen párosa is lehet egy elemnek. Az elemek neve tartalmazhat bármilyen karaktert, nem kezdődhet számmal vagy írásjellel, nem tartalmazhatnak szóközöket, nem kezdődhetnek „xml”-el.

A gyökér elem az egyetlen, amelynek csak gyerekei lehetnek, a többi elemnek van egy szülő eleme is. Az XML elemeknek lehet egyszerű tartalmuk (szöveg), attribútumaik és tartalmazhatnak gyermek elemeket. Az attribútumokat könnyen ki lehet váltani, úgy hogy az általuk hordozott információt inkább gyermek elemekben tároljuk. Ekkor ezek az elemek egy struktúrát definiálnak, így létrehozhatóak többértékű attribútumok” is, a programozás egyszerűbb, valamint sokkal egyszerűbb a későbbiekben a gyermekelemeket módosítani, mint az attribútumokat.

Létezik egy speciális szekció, az úgynevezett CDATA szekció az XML-ben. Ebben a részben az XML mindent mellőz, nincsenek hibák, szintaktika ellenőrzés. Ezt a részt használják más nyelven íródott scriptek beágyazására, mint például Javascript.

Az XML maga egy adatbázis

Egy XML dokumentum tartalmaz adatot és meta adatot egyaránt. A legalapvetőbb definíciója az adatbázisnak, egy tároló az adatoknak. A tároló tartalmaz információt, adatot úgy, mint egy cég eladási statisztikáit, ügyfeleinek listáját stb. Továbbá az alapvető adatbázis struktúra leírja az adatbázisban tárolt információkat, adatokat. A struktúra leíró adat, az adatbázis meta adatait jelenti, ez segít leírni a tárolt adatokat, e nélkül nem lenne értelme a tárolt adatoknak.

Az XML egy önleíró nyelv, mivel tartalmazza az adatokat és a hozzájuk tartozó meta adatokat, amelyek meghatározzák az adatok struktúráját, is egy helyen. Ez a meta adat megtalálható az elemek nevében és attribútumaiban valamint magában az XML dokumentum hierarchiájában is. A meta adat leíró mind skalár, mind strukturális értelemben is. A valóságban az XML rengeteg információt tárolhat egy helyen, nem ritkán egy dokumentumba sűrítve. Ez hatalmas problémát okozhat, a sok adat, sok felhasználó, és bármilyen megszokott háttérfolyamattal együtt. Az XML adatbázisok főként kevés felhasználó, adat és kis teljesítmény igény esetén alkalmazandó.

Definíció alapján natív XML adatbázis (NXD) lehet XML dokumentum vagy adat típus. Az XML adattípus egy specializált tároló létesítmény, amely egy relációs adatbázisban található. Ebből következik, hogy natív XML adatbázis, bármilyen metódus, ami XML dokumentumban tárol adatot. Továbbá XML adat típust használva a relációs adatbázisokban, natív XML adatbázis kompatibilissé teszi. Lényegében ahhoz, hogy egy XML adatbázist, natívként írjunk le, ahhoz az kell, hogy az XML dokumentumokat XML dokumentumként tároljuk.

A natív XML adatbázisok sokszor tárolnak és kezelnek több XML dokumentumot, mint az XML töredékek egy kollekcióját. Egy egyszerű natív XML adatbázis több különböző típusú kollekciót tartalmazhat, ahol a különböző típusú XML dokumentumok különböző alárendelt témákat fednek le, és így rengeteg oda nem tartozó adatot. Ezen felül, kollekciónként az

egyéni XML töredék részeknek nem kell az egész kollekcióban konzisztensnek lenniük. Ténylegesen, minden egyes egyszerű töredék résznek egy kollekcióban különböző struktúrája lehet az összes többi töredéktől az adott kollekcióban. Ennek az eredménye a strukturális és séma független XML adat.

A séma-független XML kivételesen rugalmas, amely gyorsabb és könnyebb fejlesztést eredményez. Viszont, a flexibilitás ára az adat integritás és a hibák kockázatának terjedése az adatbázison jelentkezik. A valóságban, számtalan felfogás létezik az XML fájlok kollekciójának, natív XML adatbázissá tételére, a legjobb tárhely és teljesítmény kihasználásra törekedve. Kétségtelen az elképesztő rugalmasság az XML fájlok tárolásra való használatánál. Szemlátomást, rengeteg hibához vezethet, például túl sok adat többszöri előfordulása, túlzott vagy kevés strukturális komplexitás. A potenciális buktatók listája olyan hosszú, mint az összes lehetséges különböző variációja egy témának, amit egy XML-hez hasonlóan rugalmas eszköz segítségével létre lehet hozni.

Az XML adatbázisoknak különböző indexelési metódusai léteznek. A legkézenfekvőbb metódus XML dokumentumok indexelésére egy relációs adatbázisban, egy különálló struktúra ami tartalmazza az indexelendő XML elemeket. az index tartalmazza egy egyszerű mező másolatát egy tábla minden rekordjából. Továbbá az index tartalmaz néhány mutatót, ami közvetlen elérést biztosít az index és a tábla közt, egy I/O szintű merevlemez cím. Más szavakkal, az adatbázisnak minden elemhez az XML dokumentumban hozzá kell rendelnie egy merevlemez címet. Ekkor az index tartalmazza ennek a mutatónak a címét. Az eredmény pedig a következő, mikor megtalál egy régiót az indexben, az adott régióhoz tartozó mutató átadódik egy függvénynek, ami megtalálja a rekordot egy táblában vagy XML dokumentumban a belépési rekord merevlemez címe alapján. A merevlemez címek a teljes adathalmaz vagy talán az index létrehozásakor rendelődnek a tábla vagy az XML dokumentum elemeihez. A folyamat és a folyamat lépéseinek sorrendje teljes egészében a natív XML adatbázis eléréséhez használt szoftvertől függ, vagy attól, ami egy egyszerű XML dokumentumot, vagy XML dokumentumok halmazát egy kollekciók halmazának tartja fent. (Ezek a kollekciók XML-ként vannak tárolva természetesen.)

4 indexelési típus létezik az XML dokumentumokra:

- Strukturális index:
  Az elemek és attribútumaik indexe, valamint a helye a többi elemhez viszonyítva a dokumentumban.
- Érték index:
  Általában szöveg és attribútum értékeket keresnek az XML dokumentumban, így ez kialakít egy indexet néhány,mind vagy szöveg és attribútum értékek kombinációján.
- „Teljes-szöveg index” – Full-text index:
  Ez kihasználja azt hogy egy XML dokumentumok kollekciójában egy speciális érték keresése után visszatér egy részhalmaz kollekcióval. Ez valójában egy hatalmas érték index, rengeteg XML dokumentumon vagy töredék részen egy kollekcióban.
- Összefüggés index:
  Ez egy általánosabb formája az indexelésnek, talán egy kicsit elavult, ahol sok dokumentum úgy van indexelve, hogy az index tartalmaz valamilyen értéket, ami egyértelműen beazonosítja az XML dokumentumot. Az indexek egy úgy nevezett „side” táblában vannak tárolva, majd pedig erre a táblára és kerül egy index. A végeredmény egy gyors indexelt elérés XML dokumentumok kollekciójába. Ez a megközelítés elég fárasztó. Jobb ha az XML dokumentum tartalmára kerül index. Ez olyan mintha nagy és rendezetlen XML fájlokat használnánk, kisebb könnyen kategorizálható töredék részek helyett. Bármilyen manuális kategorizálás nagyon idő és erőforrás igényes a modern adatbázisokban, az információ tiszta fizikai méretének köszönhetően.

XML DOM

Az XML DOM egy XML dokumentumként megírt generikus struktúra. Az eredmény pedig, az XML DOM. XML dokumentumok elérésére használható program szinten, tekintet nélkül az XML dokumentum adat és meta adat tartalmára. Ez azt jelenti hogy nincs szükség az XML dokumentum tartalmának, összefüggéseinek, alárendelt témájának, vagy a fő témájának az ismeretére, ahhoz hogy XML DOM-ot használjunk programozási szinten. Nyilvánvalóan, a generikus hozzáférés az adatokhoz generikus kimenetet eredményez, a programozás speciálisságától függően. Az XML DOM tárolható a natív XML adatbázisban. Így a jövőben is használhatjuk az XML DOM struktúrát generikus programozásra és feldolgozásra, XML adatok adatbázisból való kinyerésénél.

A Document Object Model egy olyan platform-és nyelv-független interfész, amely lehetővé teszi programok és szkriptek számára, a dinamikus hozzáférést a dokumentum tartalmának,

szerkezetének és stílusának, eléréséhez és módosításához. A dokumentum feldolgozható, és az eredmény a továbbiakban visszaépíthető.

Az XML DOM, bármilyen egyéni dokumentumhoz viszonyítva, általában nagyobb mint a dokumentum maga, a generált kódban és a memória használatában kifejezve. XML DOM natív XML bázisban való használata, ami az XML DOM adott adatbázisban való tárolását igényli, komoly teljesítménybeli romlást tud okozni. Ez igaz kisebb és nagyobb dokumentumokra egyaránt, ez feltehetően idegesítő a legkisebb XML dokumentumoknál. Az általános szakvélemény megegyezése szerint, XSL és XML DOM használata esetén, futási időben es web vagy applikációs szerveren használjuk.

XSLT

Az *Extensible Stylesheet Language Transformations* egy XML-alapú fájlformátumot illetve a hozzá tartozó feldolgozó rendszert jelöli. Az XSLT feldolgozót XML dokumentumok más, emberi szem számára olvashatóbb formátumra alakításához használják. A konverzió során az eredeti file megmarad, s annak tartalma alapján létrejön egy új fájl a célformátumban. A keletkező dokumentum formátuma lehet többek között XML, HTML, XHTML, PDF vagy sima szöveg például. Az XSLT-t leggyakrabban különböző sémájú XML dokumentumok közötti konverzióra és dinamikus weboldalak létrehozására használják.

The XSLT feldolgozó tipikus esetben két inputfájlt olvas be, egy forrásfájlt és egy XSLT stíluslapot, majd egy outputfájlt hoz létre. A forrásfájl jellemzően XML formátumú, de a specifikáció nem zár ki más formátumokat, például DOM modellt sem.

Natív XML adatbázisok tartalom alapján csoportosítva

Az XML dokumentumok tartalmaznak adatot, meta adatot, és némi szemantikát a benne rejlő hierarchikus struktúrában. A dokumentum központú dokumentumok, emberi felhasználásra alkalmasabbak. De lehetnek adat központúak is. ezek általában számítógépek között megosztott adatok. Ezek generikusabbak, és főként szkript nyelvek általi feldolgozásra használható.

Dokumentum központú XML:

Számítógépek számára nem a legkönnyebben feldolgozhatók, már ha ez lehetséges. Ezek azok a dokumentumok, amiket kézzel írtak, úgy mint egy Word vagy egy PDF dokumentumot. Néha indexeltek, indexelt keresésekhez, például technikai dokumentumok könyvtárakhoz. Másik oldalról nézve léteznek, olyan technikai dokumentum adatbázisok léteznek, amelyek hajlamosak keverni a dokumentum és adat centrikus dokumentumokat.

Egy speciális típusa a dokumentum centrikus natív XML adatbázisnak a Content Management System-nek hívott rendszer. A tartalomkezelő adatbázisok engedélyeznek némi irányítást és kezelést emberek írta XML adatokon keresztül, amelyek egy natív XML adatbázisban XML típusként vannak tárolva.

Adat-centrikus XML:

Ezeket a dokumentumokat a legtisztább formában, számítógépek közötti adatátvitelre használják. A valóságban az XML dokumentumok keverékei a két típusnak. Egyik része a dokumentum központú, ember által írt és ember által olvasható forma. Az adat középpontú szekció, ami generikus és program elérésű mert ismétlődő.

Natív XML adatbázis használata

Tárolás XML adatbázisokban:

Az XML dokumentumok tárolhatóak egy adatbázisban úgy mint szöveg, bináris objektum, vagy valamilyen XML adat típus. Néhány relációs adatbázis engedélyezi 4000 karakter hosszúságú egyszerű szövegek tárolását. Az XML dokumentumok természetéből adódóan, a hosszuk kiszámíthatatlan, ezért ez a tárolási módszer helytelen. A bináris objektumok lehetnek egyenesen bináris tárolásúak más néven BLOB típusúak, vagy lehetnek még speciális bináris tárolásúak, nagy szöveges objektumok amelyeket CLOB-oknak hívunk. A CLOB-ok mérete fizikailag korlátolt, 4GB maximális méretben, ellentétben az egyszerű szöveg típusúak hosszával. Továbbá eltérően az egyszerű szövegektől és BLOB típusú objektumoktól, a CLOB típus általában engedélyez valamilyen szöveg keresést illetve minta illesztést. Azonban, az XML támogatással bíró XML adat típusoknak nincs párja a CLOB minta illesztések között. Néhány relációs adatbázis biztosítja az XML dokumentumok teljes egészében XML adat típusban való tárolását, teljes XML támogatással. Természetesen egy cél szerint létrehozott natív XML adatbázisnak kellene biztosítania teljes XML funkcionalitást, az XML adat tárolás részeként.

Konkurencia, zárás és tranzakció kezelés:

A natív XML adatbázisok, XML adat típusok formájában a relációs adatbázisokban, vagy máshogy csak XML dokumentum szinten támogatják a zárakat. A legnagyobbak az XML dokumentumok, a legrosszabbak a konkurencia és a több felhasználós kapacitás lesznek. XML dokumentumok csomópont szintje persze egy lehetőség, de csak képzeljük el az implementálását. Csomópont szintű zárolásoknak szüksége lenne jóváhagyásra és végrehajtott sémákra, és ezek a rugalmasság rovására mennének. És ez egyre csak rosszabb, hiszen az XML dokumentumok természetesen hierarchikusak, így minden csomópont szint záráshoz, szükséges lenne az összes szülő csomópont zárására egy időben. Az XML tároló felépítése valójában az alkalmazás igényeitől függ.

Natív XML adatbázisok olvasása

Natív XML adatbázisokból, vagy XML adat típusokból való olvasás speciális eszközökkel elvégezhető. Ezek az eszközök tartalmaznak segédeszközöket úgy mint az XPath, XQuery.

Tartalom cseréje natív XML adatbázisokban: Számtalan eszköz és funkció elérhető ilyen célra. Ilyen például a továbbiakban leírt XQuery és XPath.

XML az informatikában

Mire képes az XML?

Rengeteg különféle dolog megalkotására, főként adatátvitel könnyítésére és komplexitás kezelésére használják. sokkal inkább olvasható, mint a relációs adatbázisok. Valójában a gépek számára is könnyebben érthető, mert egy univerzálisan megérthető nyelv, amely platform független. Az XML problémái közül az egyik, hogy lehetetlen versenyezni a modern relációs adatbázisok hatalmas méretével és nagy mennyiségű feldolgozási képességeivel. Néha viszont a különösen komplex adatszerkezetek forradalmi kezelő metódusának nevezik az XML-t.

Az XML objektum struktúrájú és elméletileg használható arra amire szánták, ténylegesen egy objektum adatbázis. Nyilvánvalóan egy teljesen alkalmas objektum adatbázis tartalmaz minden objektumorientált technikát, úgy mint az öröklődést. Tovább haladva, ki kell terjeszteni speciális eszközökkel az XML-t, úgymint DTD-k, SXD-k. Az objektum adatbázisok sokkal inkább alkalmasak különösen komplex szerkezetű adatok kezelésére. Az igazi indok az érvelés mögött az hogy egy objektum terv kisebb részekre tördeli a dolgokat, ezáltal a programoknak könnyebben kezelhetővé és hatékonyabbá válik. Az ellenkezője pedig gyakran igaz relációs adatbázisokra. Kereskedelmi környezetekben, a relációs adatbázisok de normalizálása megszokott a teljesítmény fokozása érdekében.

Viszont az XML nem alkalmas nagy mennyiségű adat kezelésére, még akkor sem ha az adat szerkezete komplikált. A relációs adatbázisok mindenre képesek, még akkor is ha az adat komplexitás közepes szintű. Az indokok nagyon egyszerűek, az XML és objektum struktúrák túl komplexek nagy mennyiségű információhoz, és a relációs adatbázisok jobban bevizsgáltak, behangoltak, mint bármilyen más adatbázis modellezési hozzáállás.

Adat séma változtatása

A legtöbb fejlett relációs adatbázis biztosít viszonylagosan egyszerű séma változtatásokat, beépített parancsok használatával. Néhány relációs adatbázis dinamikus változtatásokat is enged a meta adat objektumokban, és még a konkurenciák kezeléséről is gondoskodik. Az XML dokumentumoknál tény, hogy a dokumentum tartalmazza mind az adatot, mind pedig a meta adatot is. Így könnyen hozzáférhet a program, és manipulálhatja azt. Míg relációs

adatbázisokba való belépés és módosítás egy kicsit komplexebb, mivel az adat módosítása a meta adattól elszeparáltan történik. Továbbá bármilyen meta adat módosítása, mint például táblák, maga után vonja az adatok módosítását is. Ez XML-ben nem mindig követelmény, hiszen néhány meta adat változtatás csak a struktúrát alakítja nem pedig az adatot. Talán az XML-ben könnyebb a sémák változtatása, mert mikor elég kicsi az adatbázis hatásosabb olvasni az egészet, mint egy kis részét, valamint az XML kezelése sokkal látványosabb, és kevésbé bonyolult.

Az XML kereskedelmi implementációja

Néhány cég az információ standardizáltsága miatt használja az XML-t. Különböző emberek, számítógépek, vagy cégek közötti küldött információ könnyen megérthető minden résztvevő számára. Mindegyikőjük XML-t használ, és talán valamilyen típusú XML szótárat az adat értelmezéséhez. A másik indok nyilvánvaló indok a természetes objektum és hierarchikus struktúra szempontja az XML-nek. A kereskedelmi környezetben nem ritka a nagy mennyiségű kontrollálatlan adat. A relációs adatbázisok önmagukat tartalmazó részekre vannak bontva, és az adatokat lekérdező utasítások nagyon komplexek tudnak lenni. Ez nagyon gyenge teljesítményhez vezet. Több különböző általános felhasználási területe létezik az XML adatbázisoknak.

B2B:

„Business to Business”, információs portálok vagy adat küldési mechanizmusok az üzleti világban.

Katalógus és dokumentumkezelés:

Az adat katalógusoknak, például tudományos szöveges dokumentumok valamint fejlesztési dokumentumok, alapos keresési képességekre van szükségük a címekre és a dokumentum tartalmára vonatkozóan egyaránt.

Gyártás terület:

A gyártásnak mindig is komplex problémája volt a relációs adatbázisokkal és bármilyen alkalmazás fejlesztésével. A magas szintű komplexitás az egyik fő indok az XML használatára.

Bioinformatika és genetika:

Az adatok ezen formája különösen részletes és illékony. A kutatás bizonyos részei drámaian meg tudnak változni, akár ismétlődő változások is bekövetkezhetnek. Az XML rugalmassága és komplexitás kezelése végtelenül hasznos ezen részek számára. Együttműködést biztosítanak különböző platformokon futó szoftverek között.

Web szolgáltatások: Bármilyen web alkalmazás valós idejű adatokat szolgáltat, amit az XML standardizálni tud mind adatátvitelre, mind pedig megjelenítésre. A web szolgáltatás alkalmazások közötti adatcserére szolgáló protokollok és szabványok gyűjteménye.

SOAP

Ez egy üzenetküldésre használt, XML alapú formátum. A SOAP eredetileg a Simple Object Access Protocol rövidítése volt, de az 1.1-es verziótól önállósult. Napjaink alkalmazásai RPC(Remote Procedure Call)-k segítségével kommunikálnak objektumok között, például COBRA, DCOM. Ezeknek van egy kompatibilitási és biztonsági problémája, a tűzfalak és proxy szerverek általában blokkolják az ilyen jellegű adatforgalmat. A megoldás a http protokoll, mivel minden böngésző és szerver támogatja.

A SOAP üzenet egyfajta XML dokumentum, ami a következő elemket tartalmazza:

- „Envelope” elem, ami SOAP üzenetként azonosítja az XML dokumentumot. Kötelező a „soap-envelope”, valamint a „soap-encoding” névtér használata.

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

- Egy „soap:Header” elem, ami fejléc információkat tartalmaz.
- Egy „soap:Body” elem, ami tartalmazza a hívási és reagálási információkat.
- Egy „soap:Fault” elem, ami tartalmazza a hibákat, és státusz információkat.

A következő példa, az xmethods.net tőzsdei árfolyam-lekérdező SOAP szolgáltatását alkalmazza.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body xmlns:m="http://www.example.org/stock">
     <m:GetStockPrice>
       <m:StockName>IBM</m:StockName>
     </m:GetStockPrice>
   </soap:Body>
</soap:Envelope>
```

A válasz pedig:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body xmlns:m="http://www.example.org/stock">
     <m:GetStockPriceResponse>
       <m:Price>34.5</m:Price>
     </m:GetStockPriceResponse>
   </soap:Body>
</soap:Envelope>
```

XPath

Az XPath egy specializált kifejező nyelv, XML csomópontokon keresztüli elemzésre és nagyobb adathalmazok kinyerésére. Az XPath jelentős része az XSLT-nek. Az XPath kifejezések biztosítják a navigációt az XML dokumentum elemein és attribútumain keresztül a szöveges vagy attribútum értékig.

Az XPath teljesít néhány feladatot:

- Definiálja egy XML dokumentum részeit
- Felhasználja a kifejezéseket, hogy navigáljon az XML dokumentumokon keresztül. A kifejezés csomópontok halmazát adja az XML dokumentumból.
- Számtalan beépített funkciót használ, hogy elősegítse a megelőző feladatok feldolgozását. A beépített funkciók segítenek a típusok közötti konvertálásban, és a csomópontok vagy ezek halmaza is manipulálható vele.

XPath csomópontok és kifejezés szintaktika

Az XPath számos különböző típusát felismeri az XML dokumentumokban található csomópontoknak. Ezek a csomópontok elem, attribútum, szöveg érték, névtér, feldolgozási információt tartalmazó, megjegyzés és dokumentum csomópontokat foglal magában. Az XPath csomópontok kapcsolata egyszerűen definiált, precíz és nyilvánvaló kapcsolat, különböző típusú csomópontok között egy XML dokumentumban. Egy elemi vagy szöveg értéknek nincs kapcsolata bármilyen más csomóponttal. Az atomi értékek valódi adat értékek, nem pedig meta információk.

A szülő csomópont minden elem vagy attribútum csomópont szülőjeként van definiálva. A testvér csomópontoknál a szülő és a csomópontok közötti hierarchiában a szintjük azonos. Az ős csomópontok azok, amelyek feljebb találhatóak a hierarchiában, például a szülő csomópont szülője. Nyilvánvalóan, a leszármazottai egy csomópontnak, a gyermekei és azok gyermekei egyaránt.

Az XPath „út kifejezés”-eket használ, hogy csomópontokat nyerjen ki egy XML dokumentumból. Ez majdnem szó szerint egy kifejezés, határozott, mint egy út, egy hierarchikus struktúrán keresztül. A kifejezés szó használata helyénvaló, mivel ez az út egy

minta, ami egy XML dokumentum struktúrájára lett illesztve. Ez megtalál minden utat, ami illeszkedik a kifejezésre.

Predikátumok segítségével képesek vagyunk speciális értékek megtalálására. A predikátumok ekvivalensek az SQL-ben használt szűrőkkel, vagy a WHERE kikötéssel. A predikátumokkal minősíteni tudjuk hogy mely csomópontokat adja vissza keresési halmazból.

Általában a predikátum egy szűrő, amiben szükség van egy operátorra hogy matematikai úton kiszűrje az eredményeket. A többszörös út kifejezések biztosítják hogy egyszerre 2 adat halmazt is vissza adhasson. Más szavakkal, először keres egyet, majd egy másikat, aztán összefűzi, és azt adja vissza. A | operátor többszörös XPath kifejezések eredményeinek összefűzésére szolgál. Ez az operátor néhány programozási nyelvben a csővezeték vagy összefűzés operátorként ismert.

Példák

Az alkalmazásban cégek nyílván tartására használt XML-t fogom használni a megfelelő szemléltetéshez.

```
<?xml version="1.0" encoding="UTF-8"?>
<company xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="file:/D:/SZAKDOGA/xmlsNEW/company.xsd
">
      <id>1</id>
      <name>Cégnév</name>
      <address>Cég címe</address>
      <phone>06301112233</phone>
      <email>ceg@email.hu</email>
      <info>Németországi székhelyű cég.</info>
</company>
```

Cégnévre vonatkozó XPath utasítások:

```
company/name - Ilyen módon a teljes elérési utat megadva érhetjük el
a megfelelő csomópontot.
//name - A <name> tag minden előfordulására utal.
```

Tegyük fel hogy a <phone> elemnek több előfordulása is lehet, ekkor hivatkozhatunk az összes vagy akár egy adott sorszámú elemére is.

```
//phone - Ez az összes előfordulásra mutat.
company/phone[1] - Így az előforduló telefonszámok közül az elsőt kapjuk eredményül. (IE 5 esetén az első elemet a 0 indexszel kapjuk meg)
```

Ha attribútum értékre szeretnénk hivatkozni, hasonlóan kell rá hivatkoznunk, mint csomópontnál, viszont egy „@” jellel kell kezdenünk az attribútum nevét. Tegyük fel hogy az <address> elemnek van egy „country” attribútuma, ekkor a következőképp hivatkozhatunk rá:

```
<address country=”Németország”>Cég címe</address>
company/address@country
```

Lehetőség van egy adott csomópont szülő csomópontját megtalálni.

```
//phone/.. - Így megkapjuk a <country> elemet amelyhez az adott <phone> elem tartozik.
```

Predikátumok segítségével lehetőség van szűrő feltételek megadására is, attribútumok és csomópontok esetén egyaránt.

```
country[id=1]/address - Ez annak a cégnek a címét adja vissza amelynek azonosítója 1.
country/id[. > 1] - Vissza adja azon cégek azonosítóját, ahol az azonosító nagyobb mint 1.
country/address[@country=”Németország”] - Ennek segítségével megkaphatjuk a németországi cégek címét.
```

XPath tengelyek, és funkciók

Az XPath tengelyek elérést definiálnak és adnak bármilyen csomóponthoz egy XML dokumentumban. Ez kicsit olyan mintha irányt mutatnánk a navigációhoz az XML dokumentumon keresztül, XPath adat modellt használva. Egy XPath tengely irányt ad az érvényes csomóponttól függetlenül, az aktuális csomópont XML dokumentumbeli aktuális helyétől és környezetétől függően.

Az érvényes csomópont, az a csomópont, amelyet éppen a legsűrűbben használt XPath kifejezés talált meg. Technológiailag, bármelyik csomópont megtalálható, bármelyik csomópontból indulva.

```
axis::tesztelendő-csomópont[preditkátum]
```

Az XPath funkciók aktuálisan megtalálhatók az XQuery-ben és az XPath-ban egyaránt. Az összes funkció elérhető az „fn” névtér hozzáadásával:

```
xmlns:fn=http://www.w3.org/2005/02/xpath-functions
```

Az XPath 2.0 funkciói kategóriákra osztva, néhány példával

- Hozzáférhetőségi funkciók: Ezek hozzáférést biztosítanak ahhoz hogy privát módon érhessük el az adatot, egy funkció amely engedélyezi hogy privát módon lekérdezzük és állítsuk be a tulajdonságainak értékét, anélkül hogy közvetlen hozzáférést biztosítana a tulajdonságokat tartalmazó objektumhoz. Egy csak olvasható tulajdonságnak csak egy ilyen függvénye létezik, ami engedélyezi a tulajdonság értékének a lekérdezését.
  - fn:node-name(node) – Csomópont neve
  - fn:string – Visszaadja a csomópont értékét stringként.
  - fn:data(item [, item,…]) – Szekvenciává alakítja az elemeket,
- Hibák és nyomkövetés: Hibák kiváltása, és megoldása futás közbeni nyomkövetés segítségével.

  fn:error(error, description, object) – Kivált egy kivételt.
- Konstruktorok: Ezek állítják elő az objektum példányokat egy osztályból, vagy definiálnak egy típust.

  xs:date(), xs:string(), xs:Name(), xs:token()
- Numerikus függvények: Mint bármely programozási nyelvben, egy numerikus függvény végez valamilyen konverziót illetve kalkulációt, amely után egy számot ad eredményül. Számos numerikus operátor elérhető az XPath használata közben. Rengeteg ezek közül úgy mint a „numeric-add”, elérhető a megszokottabb szimbolikus használati formája is(+).
  - fn:round(num) - Kerekítő függvény.
  - fn:abs(num) – Abszolútérték függvény.
  - fn:number(arg) – Stringből numerikus típussá konvertál.
- Szöveges függvények: A műveletek általában szövegen hajtják végre, és legtöbbször ugyanilyen típusú a visszatérési érték is, de nem minden esetben.
  - fn:concat(string [, string..]) – Összefűzi a stringeket.
  - fn:string-length([| string]) – A string hossza, vagy ha nincs megadva paraméter akkor az aktuális csomóponté.

    - fn:starts-with(string1, string2)/ fn:ends-with(string1, string2)
    - fn:contains(string1, string2) – Tartalmazza e az adott stringet.
    - fn:replace(string, pattern, replace) – Lecseréli a stringben az előfordulásokat.
- URI függvények: URI-kat dolgoznak fel, illetve módosítanak. Az egyetlen függvény ebben a szekcióban:

  fn:resolve-uri(relative, base) – Feloldja a relatív URI-kat abszolúttá.
- Boolean függvények
    - fn:boolean(arg) – String, numerikus illetve csomópont értékek boolean értéke.
    - fn:not(arg) – A boolean függvény inverze.
- Idő és dátum funkciók
    - fn:dateTime(date, time) – Egy TimeStamp típusú objektumot ad vissza.
    - fn:year-from-date(date) – Az adott dátumtól számított időtartam években. Hasonló formában létezik hónapokra, napokra is.
    - fn:hours-from-time(time) - Az adott időponttól számított időtartam órákban. Hasonló formában létezik percekre, másodpercekre is.
    - fn:adjust-dateTime-to-timezone(datetime, timezone) – Egy timestamp objektumot adott időzónának megfelelőre konvertál.

      Léteznek dátumok összegzését, kivonását, hasonlítását szolgáló függvények is.
- QName funkciók: (Qualified Name) A QName egy korlátozott név. Egy QName tartalmaz egy névtér URI-t. A QName egy opcionális előtag és egy kettőspont után következik a „helyi” név vagy egy URI és a lokális vagy attribútum név.
    - fn:QName() – Visszadja a QName értékét az adott csomópontnak.
    - fn:local-name-from-QName() – A lokális nevet adja vissza a QName attribútumból.
    - fn:namespace-uri-from-QName() – Névtérrel és URI-val tér vissza.
- Csomópont függvények
    - fn:name([nodeset]) – Az aktuális csomópont neve, vagy a csomópont kollekció első eleme.
    - fn:root([node]) – Visszaadja a gyökér elemet.
    - op:is-same-node – Igazat ad vissza, ha a két csomópont egyenlő.
- Szekvencia funkciók: A szekvencia ténylegesen egy lista a nem létező vagy még lehetségesen ismétlődő elemek egy szülő csomóponton belül. A szekvencia ennek következtében egy kollekció. Szekvencia függvények alkalmazhatóak egy egyszerű kollekcióra teljes egészében, vagy a kollekció valamely tagjára.
    - fn:count(collection) - Kollekció elemeinek száma
    - fn:max(collection) - Kollekció elemeinek maximuma
    - fn:avg(collection) – Kollekció elemeinek átlaga
    - fn:empty(collection) – Igazat ad vissza ha üres a kollekció.
    - fn:exists(collection) – Nem üres kollekciókra igazat ad.
- Tartalmi funkciók: Ezek a funkciók a jelentésekkel dolgoznak, így a meta adatokat dolgozzák fel
    - fn:last() – Megtalálja az utolsó elemet a kollekcióban.

    - fn:current-date()/fn:current-time() – Aktuális dátum illetve időpont.
    - fn:implicit-timezone() – Megtalálja az időzónára vonatkozó értéket.
- Egyéb típusú függvények: 64bites számokon alapuló operátorok, XML jelölésekre vonatkozó függvények és operátorok. Az XML jelölések(notation) egy formája az XML-nek, amely dialektus specifikus, és nagyon speciális alkalmazásokhoz lett készítve XML-ben úgy, mint a MathXML, CML. A speciális XML dialektus jelölések szabványokat is fektetnek le, és a regionális szemantikát alkalmazzák az egyébként általános XML adathoz.

XQuery

Az XQuery olyan az XML-nél, mint az SQL a relációs adatbázisoknál. Az XML dokumentumok olvasásra használják. Arra lett tervezve hogy bármit lekérdezzünk ami XML adatként jelenik meg, még relációs adatbázisban tárolt XML adat típus struktúrát is. Lényegében adat modellből és az adat modellen használható lekérdező operátorokból áll. Inkább leíró, mint procedurális programozási nyelv, csakúgy, mint az SQL. Procedurális programozási nyelv, lépésről lépésre halad a feldolgozásban hogy megoldja a problémát, ahol a különböző programkód sorok egymásnak értékeket adhatnak át és kaphatnak vissza. A leíró programozási nyelv egyszerűen definiál határvonalakat, feltételeket és megszorításokat. Majd a számítógép keres valamilyen megoldást, amely megfelel a specifikált megszorításoknak. Egy leíró nyelv sokkal magasabb szintű, mind egy procedurális programozási nyelv.

Az XQuery átvizsgálja az XML dokumentumot (vagy egy részét), felhasználja a megszorításokat a lekérdezéshez (úgy mint szűrés, predikátumok), és vissza adja azokat az adatokat amelyek illeszkednek a lekérdezés által specifikált határvonalaknak, feltételeknek és megszorításoknak. A megoldás egy részhalmaza az eredetileg vizsgált XML dokumentumnak, XML formában. Technikailag, az XQuery mindenhol használható ahol az XML is. Az XML célja hogy létrehozzon egy univerzálisan érthető, rendszer független adat forrást, így az XQuery követve a formát, megkísérel egy univerzálisan használható XML dokumentum lekérdező nyelvet létrehozni.

Az Xpath és XQuery ugyanazokat a funkciókat és operátorokat használja, és az adat modell is közös. Valójában az XQuery, XPath kifejezésekkel végez lekérdezéseket XML dokumentumokon.

XQuery alapjai

A legfontosabb tény, hogy az XQuery nem XML-ben íródott, bár a lekérdezések beágyazhatóak XSL stílus lapokkal. XQuery parancsok HTML fájlokba is beágyazhatók.

7 különböző fajta XQuery csomópont létezik: elem, attribútum, névtér, feldolgozási információ, megjegyzés, dokumentum.

- Atomi érték: Egy szöveges érték, nincs szülő vagy gyermek csomópontja.
- Darab érték: Ez lehet atomi érték vagy csomópont.
- Szülő: Egy csomópont szülőjére hivatkozik, általában csomópontokra és nem atomi értékekre utal.
- Gyermek: Ellentéte a szülőnek, ahol a csomópontnak 0 vagy több csomópontja lehet. Egy csomópont aminek lehet gyermeke, de nincs, egy üres csomópontra hivatkozik.
- Testvér: Egy csomópont abban a kollekcióban amelynek ugyanaz a szülő csomópontja.
- Ős: Bármely csomópont az érvényes csomópont felett a fában.
- Leszármazott: Bármely csomópont az érvényes csomópont alatt.

XQuery szintakszis

Néhány általános szabály:

- A parancsokban és kódokban általános megkülönböztetjük a kis és nagy betűket, csak úgy, mint az XML esetében.

- Minden csomópontnak érvényesnek kell lennie mint XML név

- Változó definiálása $ karakter segítségével történik, például $változónév.

- Az elágaztató utasítások szintaktikája a következő:

```
if(…)
then
else ...
```

- Többszörös értékek hasonlítása szabványos aritmetikai operátorokkal történik úgy, mint =, !=, >, <=

- Egyéni értékek hasonlítása: eq, ne, gt, it stb.

Lehetőség van saját funkciók létrehozására is, a következő szintakszis használatával, ami eléggé következetes szintaktikájú rengeteg egyéb programozási leíró és procedurális nyelvel egyaránt.

```
declare function prefix: függvény_neve ($paraméter AS adatTípus) AS
visszatérésiAdatTípus
{
      {(: … :)}
};
```

Egy egyénileg definiált függvény ugyanúgy meghívható, mint bármely másik. Ha a funkció előtaggal együtt lett definiálva, akkor előtaggal együtt kell meghívni.

XQuery FLWOR

Ez a kifejezés a „for” ciklus egy formája. Programozási formában, a for ciklus ismétlődő feldolgozást engedélyez, egy kollekció minden elemére,más szóval engedélyezi ugyanazt a folyamatot lefuttatni minden elemre egy kollekcióból. A FLWOR szó a „For, Let, Where, Order by, Return” szavak kezdőbetűiből tevődik össze. Ez azt jelenti hogy a „for” ciklus engedélyezi az egész kollekció elemenkénti feldolgozását. A „where” feltétel, szűrési lehetőségeket ad, hogy csak a szükséges elemeket dolgozzuk fel. Az „Order by” feltétel rendezettséget, biztosit az eredményhalmazban, amelyet a „return” kikötés ad vissza. A „return” kikötés irányítja a speciális adat elemeket, amelyek a „for” ciklusból lettek vissza adva.

Példák:

Ebben az esetben egy hosszabb XML dokumentumot szeretnék használni, melyben a cégek nyílván tartására használt XML dokumentumbeli „<company>” elem többször is előfordulhat.

A következő példa a cégek neveit hivatott vissza adni.

```
for $i in doc("companys.xml")/company
return $i/name

for $i in doc("companys.xml")/company/name
return $i
```

Természetesen lehetőség van, szűrési és rendezési feltétel megadására is.

```
for $i in doc("companys.xml")/company
order by $i/name
return $i/name

for $i in doc("companys.xml")/company/
where $i/phone='0630*'
return $i/name
```

Az „at” kulcsszó használatával lehetőség nyílik az iteráció számosságának meghatározására.

```
for $x at $i in doc("companys.xml")/company/phone
return <phone>{$i}. {data($x)}</phone>
-------------------------------------------------
<phone>1. 06112223333</phone>
<phone>2. 06112223334</phone>
<phone>3. …</phone>
…
```

A „let” kulcsszó segítségével, egy kifejezést többször is megjeleníthetünk

```
let $x := (1 to 5)
return <test>{$x}</test>
-------------------------
<test>1 2 3 4 5</test>

for $x := (1 to 5)
return <test>{$x}</test>
-------------------------
<test>1</test>
<test>2</test>
<test>3</test>
<test>4</test>
<test>5</test>
```

A következő példa, függvény hívását mutatja. Kiveszi a pont előfordulásokat, a cím elemből.

```
for $x in doc("companys.xml")/company
return <address>{translate($x/address),'.', ''}</address>
-----------------------------------------------------------------
<address>Db Egyetem tér 1</address>
<address>Db ...</address>
...
```

Több ciklus is használható egy lekérdezésben.

```
for $x in (1, 2), $y in (1, 2)
return <szamok>{$x} {$y}</szamok>
---------------------------------
<szamok>1 1</szamok>
<szamok>1 2</szamok>
<szamok>2 1</szamok>
<szamok>2 2</szamok>

for $x in doc('companys.xml')/company
return <ceg>
           <nev>{$x/name/text()}</nev>
           {for $y in $x//phone
           return <szam>{$x/text()}</szam>}
       </ceg>
-----------------------------------------------
<ceg>
    <nev>Kiss BT.</nev>
    <szam>0652111111</szam>
    <szam>0652111112</szam>
</ceg>
<ceg>
    ...
</ceg>
..
```

A következő példa, email cím helyett telefonszámot ad meg, ha az nincs megadva.

```
for $x in doc("companys.xml")/company
return if (count($x//email) < 1)
then <phone>{$x//phone/text()}</phone>
else <email>{$x//email/text()}</email>
```

Az olyan cégek neveit, ahol csak magyarországi telefonszám van megadva, a következőképp kaphatjuk meg. Módosítva az „every” kulcsszót, „some”-ra, minden olyan céget megkapunk, ahol van legalább egy magyar telefonszám megadva.

```
for $x in doc("companys.xml")/company
where every $y in $x/phone satisfies
     starts-with($y/text(), "06")
     or starts-with($y/text(), "+36")
return $x/name
```

Oracle-ben is használhatunk XQuery-t, dokumentumokra és XMLType objektumokra egyaránt. A következő példa visszaadja az 1-es azonosítóval rendelkező cég telefonszámait.

```
SELECT XMLQuery(
           'for $i in /company
            where  $i/id/text() = "1"
            return $i/phone/text()'
            PASSING OBJECT_VALUE RETURNING CONTENT)
FROM COMPANYS;
```

.

Ha séma validált XMLType típusú objektumról van szó, akkor a következőképp is nézhet ki a lekérdezés:

```
SELECT phones.COLUMN_VALUE
FROM COMPANYS, XMLTable(
                      'for $i in /company
                       where  $i/id/text() = "1"
                       return $i/phone/text()'
                       PASSING OBJECT_VALUE) phones;
```

És relációs táblák adatainak lekérdezésére is van lehetőség, ehhez tekintsük a cégek nyílván tartására szolgáló tábla relációs változatát.

```
SELECT *
FROM XMLTable('for $i in ora:view("COMPANYS_TABLE")/ROW
               where $i/ID = "1"
               return $i/NAME');
--------------------------------------------------------
<NAME>...</NAME>
<NAME>...</NAME>
...
```

XQueryX

Ez egy variációja a XQuery XML dokumentum lekérdező nyelvének, XML-ben írva. A probléma a következő, a komplexitás eszközökbe lett programozva, mindegyiknek tartalmaznia kell az XML dokumentum és az XQuery lekérdezés struktúráját is, ami elég átláthatatlan tud lenni. Az XQueryX lekérdezések kódolása nem komplikált, de mikor összehasonlítjuk az XQuery egyszerűségével, érezhető a különbség.

Az XQueryX XML dokumentum konstrukciójának szüksége van minden adat, meta adat, és programozási konstrukció definíciójára. Amikor hozzá adunk egy „for” ciklust, le kell kódolnunk a ciklust, a változó definíciókat, és a bejövő és kimenő ciklus paramétereket. Szinte a tokenizált programozás világába lép be. Ez túl sok az XML-nek, és túl sokat kér a fejlesztőktől. Szép hogy mindent egy nyelven csináljon, de a lekérdezések esetében, az XQueryX nem túl gyakorlatias és talán még egy kicsit túlbuzgó az általánosított XML felhasználása a saját programjában.

Az XQuery rossz oldala hogy nem XML-ben lett írva, ez valójában nem probléma. A szépsége a legtöbb W3C XML szabványainak, hogy XML-ben lett írva, ebből látszik igazán hogy az XML programozás platform és forgalmazó független.

Egyéb XML Technológiák

Sok magas szintű XML szabvány főként nem adatbázis specifikus, viszont annál inkább Front-endalkalmazás specifikus.

XLink és XPointer

Az XML változik a kimeneti eredményeknek vonatkozásában, úgy, mint egy transzformáció XSL-en keresztül, vagy XQuery segítségével olvasni az adatbázisból. A fő indokok egyike a z internet hatalmas sikerének, az a lehetőség, amellyel a honlapok hivatkozhatnak egymásra az egész interneten keresztül. Az XML-nek muszáj ugyanezt a képességet biztosítania. Ez az azért van, mert az XML dokumentumok nem csak adatokat szolgáltatnak, hanem meta adatokat is, amelyek értelmet adnak az adatoknak. Ezáltal az XML dokumentum adatokat használó hivatkozásnak is rugalmasnak kell lennie, ha lehetséges kiszámíthatóbbnak, de ez nem követelmény.

A weblapok azon hivatkozásai, amelyek XML segítségével menedzseltek, 2 fő komponenst használnak, az XLinket, és az XPointert. Az XLink definiálja azt a szabványt, amely megadja, hogy mely honlap hivatkozások készüljenek el. Az XPointer pedig kiegészíti az XLinket, lehetővé téve hogy XML töredékre hivatkozhassunk egy nagy XML dokumentumon belül.

Az XLink lehetővé teszi a hivatkozások generálását, HTML-éhez hasonló módon. Mindazonáltal az XLink sokkal alkalmazhatóbb, mivel bármilyen XML dokumentum elem létrehozható hivatkozásként. Továbbá az lehetőség nyílik több forrás együttes összekapcsolására. Az eredmény pedig, bármely XML elem egy dokumentumban, hivatkozható egy másik honlapról. Az XLink generálható az XML dokumentum tartalmára alapozva, szükségtelenné téve az emberi vagy programozói beavatkozást.

Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<xlinkexamples xmlns:xlink="http://www.w3.org/TR/xlink/">
<xlinkex1 xlink:type="simple"
xlink:href="http://www.google.com">Google</xlinkex1>
 <xlinkex2 xlink:type="simple"
xlink:href="http://www.yahoo.com">Yahoo></xlinkex2>
</xlinkexamples>
```

4 fő attribútuma van.

- xlink:type - Ezt mindig meg kell adni, a „href” attribútummal együtt természetesen. Egyszerű hivatkozásnál „simple” az értéke, valamint specifikus értékek ha egy specifikus forrásra akarunk hivatkozni, avagy egy hivatkozó elemre akár.
- xlink:href – Maga a hivatkozó URL.
- xlink:show – Hol legyen megnyitva a hivatkozás, beágyazni a meglévő tartalomba, újként megnyitni, lecserélni a jelenlegit, stb.
- xlink:actuate – Mikor legyen a hivatkozás aktiválva, betöltéskor, kattintáskor, semmikor, vagy akár hagyhatjuk más szoftverre is.

XForms

Megteremti a lehetőséget a végfelhasználóknak adatok bevitelére alkalmazásba, esetleg adatbázis adatinak módosítására. Az XForms ugyanazt biztosítja, mint a HTML formok, kivéve hogy ez XML alapú. Ezáltal könnyen lehet adatvezérelt, specifikusabb, komplexebb beviteli űrlapok megjelenítését, úgy, mint egyedi képernyők bizonyos felhasználóknak, illetve felhasználói csoportoknak.

Példa:

```
<html xmlns:xf="http://www.w3.org/2002/xforms">
<head>
     <xf:model>
          <xf:instance>
               <felhasznalo>
                    <nev/>
                    <szuldat/>
               </felhasznalo >
          </xf:instance>
     <xf:submissionid="form1" action="submitted.asp" method="get"/>
     </xf:model>
</head>
<body>
     <xf:input ref="nev">
          <xf:label>Név</xf:label>
     </xf:input><br/>
     <xf:input ref=" szuldat ">
          <xf:label>Születési dátum</xf:label>
     </xf:input><br/>
     <xf:submit submission="form1">
          <xf:label>Submit</xf:label>
     </xf:submit>
</body>
</html>
```

Az <instance> elem adja meg az eredmény XML dokumentum struktúráját. A <submit> tag írja le hogyan lesz elküldve az adat.

Az <input> tag adja meg a bemenet specifikációit. Ilyen lehet a <secret> tag jelszavakhoz, <output> megjelenítéshez, <trigger> valamilyen eljárás, függvény végrehajtáshoz, stb

```
<secret ref="nev/jelszo"><label>Jelszó:</label></secret>
```

Az „upload” elem segítségével, fájlok feltöltését biztosíthatjuk a szerverre.

```
<upload bind="name">
      <label>A fájl neve: </label>
      <filename bind="FájlNeve"/>
      <mediatype bind="media"/>
</upload>
```

Lehetőség van típusok használatára így a név, és születési dátum a következőképpen néz ki:

```
…
<felhasznalo>
      <nev xsi:type="xsd:string"/>
      <szuldat xsi:type="xsd:date"/>
</felhasznalo>
…
```

Különböző megszorítások adhatók meg:

- constraint: Egyfajta megszorítás, mint például csak speciális értékek fogadhatók el.
- required: Kötelező megadni értéket.
- type: Adattípus.
- calculate: Végrehajt egy számítást.
- readonly: Csak olvasható az adott tag.

Különböző műveletek hajthatók végre, a specifikus események hatására. Ilyen például egy tooltip felhozása, a kitöltés segítéséhez.

```
…
<input ref="nev">
      <label>Név</label>
      <message level="ephemeral" event="DOMFocusIn">
            Írja be a nevét!
      </message>
</input>
…
```

Az „ephemeral” a rövid időtartamot jelzi, így mikor a név mező előtérbe kerül, megjelenik az üzenet, majd eltűnik.

Úgy mint a többi XML technológiának, az XForms-nak is van közvetlen elérése az XPath funkciókhoz, de természetesen saját függvényei is léteznek. Ilyen például a property(<string>), amely egy attribútum aktuális értékét adja vissza.

Oracle

XMLType adattípus

XML dokumentumok létrehozása adatbázis specifikusan, általában a rendelkezésünkre álló speciális eszközök használatával történik. Legalapvetőbb formában, az Oracle SQL-ben, az XML magában foglalja az XMLType adattípust és néhány metódust a típushoz kötve. Tulajdonképpen az XMLType egy osztály. Az XMLType tárolja az XML dokumentumok szövegét, és lehetőség van elérni az XML dokumentum objektum modelljét. Az objektum modellen keresztül elérhetünk minden XML dokumentum elemet.

Ezt a speciális adattípust egyaránt használják tárolásra és XML dokumentumok feldolgozására is. Ezenkívül egy ilyen típus tárolható egy speciálisan létrehozott táblában bináris objektumként(CLOB). A CLOB egy szöveges objektum, amely főképp belső keresésre alkalmas. Ez azt jelenti, hogy megtalálható egy kis szövegrész is a bináris objektumon belül.

A következőknél érdemes az XMLType használata:

- Erős típusok SQL utasításokon és PL/SQL funkciókon belül. Sokkal jobb ha az értékek XMLben vannak tárolva, mintha valamilyen egyszerű szöveges típusként.
- Megóvni az alkalmazásokat a tárolási modellektől. Később sokkal könnyebben lehet más változatos tárolási eljárásokat bevezetni, anélkül hogy módosítanánk az alkalmazásban szereplő DML illetve lekérdező utasításokat.
- Felkészülhetünk vele a jövőbeni optimalizálásra. Kiváló optimalizáló és indexelő technikák érhetők el,amelyeket folyamatosan fejlesztenek, így később az alkalmazás módosítása nélkül javíthatunk a teljesítményen.

	LOB tárolási mód(Oracle szövegindexeléssel)	Strukturált tárolás (B* indexeléssel)
Adatbázis séma rugalmasság	Séma változásnál nagyon rugalmas.	Korlátozott változtathatóság, ALTER TABLE megszorításaihoz hasonló.
Adat pontosság és integritás	Az eredeti XML dokumentummal byte szinten megegyező.	Lehet benne új sor, szóköz és az adat típus a nem szöveges adatokhoz elvész. De a DOM-hoz ragaszkodik.
DML futtatási teljesítmény	Közepes	Kiváló
Hozzáférés SQL funkcionalitáshoz	Korlátozott.	Jó hozzáférhetőség a meglévő lehetőségekhez, úgy mint indexek, megszorítások stb.
Helyigény	Jelentős tárterületet igényelhet.	Kevesebb helyet foglal, ha regisztrált XML sémával használjuk az adatbázist.

A következőképpen lehet létrehozni:

```
CREATE TABLE XMLEX1 OF XMLTYPE;

CREATE TABLE XMLEX2(
      ID    NUMBER NOT NULL,
      XML   XMLTYPE,
)
```

Valamint PL/SQL-ben is használható:

```
DECLARE
      XML XMLTYPE;
BEGIN
      NULL;
END;
```

Számos metódus(alprogram) létezik, ami XMLType adattípuson végrehajtható. Néhány fontosabb ezek közül:

A következő példákhoz az alkalmazásomban, cégek nyílván tartására használt XML dokumentumot fogom használni.

```
<?xml version="1.0" encoding="UTF-8"?>
<company       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">  <id>0001</id>
     <name>company</name>
     <address>address</address>
     <phone>phone</phone>
     <email>email</email>
     <info>info</info>
</company>
```

- EXISTSNODE(XPath-expression, [namespace]) : Egy XPath kifejezést használ, hogy megvizsgálja létezik e a csomópont az adott XMLType típusú objektumban.

```
SELECT c.getClobVal()
FROM companys c
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]') = 1;
--------------------------------------------------------------------
-------------------
<?xml version="1.0" encoding="UTF-8"?>
<company  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">
     <id>0040</id>
     <name>company</name>
     <address>address</address>
     <phone>phone</phone>
     <email>email</email>
     <info>info</info>
</company>
```

- EXTRACT(XPath-expression):

```
SELECT extract(OBJECT_VALUE, '/company/name')
FROM companys
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]') = 1;
--------------------------------------------------------------------
<name>company</name>
```

```
SELECT extract(OBJECT_VALUE, '/company/name/text()')
FROM companys c
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]') = 1;
--------------------------------------------------------------------
company
```

- EXTRACTVALUE(XPath-expression): Az extract függvényen használható getStringVal() illetve `getnumberval()` kiváltására szolgál.

```
SELECT extract(OBJECT_VALUE, '/company/name')
FROM companys
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]') = 1;
--------------------------------------------------------------
company
```

- TRANSFORM(XSL-stylesheet-script): Az XMLType típusú objektumot egy XSL stíluslap segítségével formázott XML dokumentummá transzformálja.

XML létrehozása adatbázisból

XML dokumentumok létrehozásánál az adatbázisból fogunk olvasni adatokat, pontosabban a táblákból olvasunk rekordokat. Ehhez számos különböző alprogram áll rendelkezésünkre.

- SQL/XML szabvány: Alapvető funkciók elemek létrehozására, attribútumok hozzárendeléséhez. INCITS (International Committee for Information Technology Standards) által létrehozott és támogatott szabvány.
- DBMS_XMLGEN csomag: Egy komplex csomag mely segítségével egy lekérdezésből XML dokumentumot hozhatunk létre.
- XSU segédprogram: Kifejezetten Javahoz készült.

SQL/XML szabvány

Néhány alapvető funkcionalitás, amely elérhető Oracle adatbázisból. Legyen adott egy relációs tábla a következő mezőkkel.

FELHASZNALOK:

- ID
- NEV
- SZUL_DAT
- CÍM

- **XMLELEMENT**([NAME] id [, attrbts] [, xprssn [, ..]]) : XML elem létrehozása.

```
SELECT XMLElement("SzuletesiDatum", SZUL_DAT)
FROM FELHASZNALOK
WHERE ID = 1;
----------------------------
<SzuletesiDatum>1994-06-07</SzuletesiDatum>
```

```
SELECT XMLElement("felhasznalo",
  XMLElement("nev", NEV),
  XMLElement("szuldat", SZUL_DAT),
  XMLElement("cim", CIM))
FROM FELHASZNALOK
WHERE ID = 2;
----------------------------
<felhasznalo>
<nev>Nagy Elemér</nev>
<szuldat>1994-06-07</szuldat>
<cim>Debrecen Egyetem tér 1.</cim>
</felhasznalo>
```

- **XMLATTRIBUTES** (xprssn [AS alias] [, …]) : Értéket rendel az attribútumokhoz.

```
SELECT XMLElement("felhasznalo", XMLAttributes( id as "ID"))
FROM FELHASZNALOK
WHERE ID = 1;
----------------------------
<felhasznalo ID = 1></felhasznalo>
```

- **XMLCONCAT** (XMLType obj) .. : XML elemeket konkatenál.

  Ennek a funkciónak két formája van:

  Az első amikor egy `XMLSequenceType(XMLType tipusú elemek tömbje)` értéket kap, és egy egyszerű `XMLType` típusú elemet ad vissza értékül.

```
SELECT XMLConcat(XMLSequenceType(
     XMLType('<id>1236</id>'),
     XMLType('<nev>Nagy István</nev>'))).getClobVal()
FROM DUAL;
-----------------------------------------------------
<id>1236</id>
<nev>Nagy István</nev>
```

- **XMLAGG** (XMLType obj [ORDER BY …]) : Egy egyszerű oszlopot vagy kifejezést készít több sorból, a sorok egy sorba és XML elembe aggregálásával .

```
SELECT XMLElement("Nevek",
     XMLAgg(XMLElement("Nev", NEV) ORDER BY ID))
FROM FELHASZNALOK
------------------
<Nevek>
     <Nev>Kiss József</Nev>
     <Nev>Nagy István</Nev>
     <Nev>Szél Pál</Nev>
     ...
</Nevek>
```

- **XMLPI** : XML feldolgozási instrukciók adhatóak meg.

```
SELECT      XMLPI(NAME      "OrderAnalysisComp",      'imported,
reconfigured')
FROM DUAL;
-----------------------------------------------------------
<?OrderAnalysisComp imported, reconfigured, disassembled?>
```

- **XMLCOMMENT**

```
SELECT XMLComment('Megjegyzés')
FROM DUAL;
-------------------------
<!--Megjegyzés-->
```

- **XMLROOT**

```
SELECT XMLRoot(XMLType('<id>4</id>'), VERSION '1.0', STANDALONE
YES)
FROM DUAL;
-------------------------------------
<?xml version="1.0" standalone="yes"?>
<id>4</id>
```

- **XMLPARSE :** Vissza térési értéke egy XMLtype példány.

  Content/Document: Ha document akkor egy gyökér elemű jól formázott dokumentumnak kell lennie, míg „content”-nél csak a jól formázottságnak kell teljesülnie.
  Wellformed: Ha meg van adva, azt jelenti az adatbázis kezelő számára, hogy jól formázott a dokumentum, nem kell ellenőriznie.

```
SELECT XMLParse(CONTENT
     '124 <felhasznalo>
          <nev>Nagy István</nev>
          <szuldat>1987-05-05</szuldat>
     </felhasznalo>'
     WELLFORMED)
FROM DUAL d;
-----------------------------------------------
124
<felhasznalo>
     <customerName>Nagy István</customerName>
     <szuldat>1987-05-05</szuldat>
</felhasznalo>
```

- **XMLSERIALIZE**

```
SELECT XMLSerialize(DOCUMENT XMLType('<id>17</id>') AS CLOB)
FROM DUAL;
-------------------
<id>17</id>
```

- **XMLFOREST** (xprssn [AS alias] [, …]) : Több XMLELEMENT végrehajtást eredményez.

  Akár felhasználói típusokból is készíthetünk XML dokumentumokat.
- **XMLTRANSFORM** (XMLType obj, XMLType obj) :

Oracle specifikus SQL funkciók:

- **XMLSEQUENCE**

```
SELECT value(T).getStringVal()
FROM table (XMLSequence
     (extract(XMLType(
          '<A><B>V1</B><B>V2</B><B>V3</B></A>'),'/A/B')
          )
     ) T;
----------------------
<B>V1</B>
<B>V2</B>
<B>V3</B>
```

Két formája létezik, az első amikor egy XMLType példányt kap paraméterül, és csomópontok kollekcióját adja vissza(Ez nem Oracle specifikus.) A másik formája mikor egy REFCURSOR példányt kap, és opcionálisan megadható mellé egy XMLFormat objektum is, és a kurzor sorait konvertálja XMLType típusú varray kollekcióra.

- **XMLCOLATTVAL** (xprssn [AS alias] [, …]) : Minden nevével megadott oszlopnak a neve attribútuma lesz az oszlop elemnek.

```
SELECT XMLElement("felhasznalo",
     XMLAttributes(NEV AS "nev" ),
     XMLColAttVal(SZUL_DAT, CIM AS "lakcim"))
FROM FELHASZNALOK
WHERE ID = 1;
-------------------------------------------------------------
<felhasznalo nev="Nagy elemér">
     <column name = "SZUL_DAT">1994-06-07</column>
     <column name = "lakcim">Debrecen Egyetem tér 1.</column>
</felhasznalo>
```

- **XMLCDATA**:

```
SELECT XMLElement("felhasznalo",
     XMLElement("cim",
     XMLCDATA('Egyetem tér 1.'),
     XMLElement("varos", 'Debrecen')))
FROM DUAL;
--------------------------
<felhasznalo>
     <cim>
          <![CDATA[Egyetem tér 1.]]>
          <varos>Debrecen</varos>
     </cim>
</felhasznalo>
```

- **SYS_XMLGEN**

```
SELECT sys_XMLGen(nev).getStringVal()
FROM FELHASZNALOK
WHERE id = 1;
----------------------------------------
<?xml version="1.0"?>
<NEV>Nagy István</NEV>
```

```
CREATE TYPE FELH AS OBJECT(
     nev   varchar2(50),
     szul_dat   date,
     cím   varchar2(200)
)

SELECT sys_XMLGen(
     FELH(NEV,SZUL_DATE,CIM)).getStringVal()
FROM FELHASZNALOK
WHERE ID = 1;
----------------------------------------
<?xml version="1.0"?>
<ROW>
     <NEV>Nagy István</NEV
     <SZUL_DAT>1994-06-07</SZUL_DAT>
     <CIM>Debrecen Egyetem Tér 1.</CIM>
</ROW>
```

DBMS_XMLGEN csomag

SQL lekérdezések eredményéből képes XML dokumentumot készíteni. XMLType vagy CLOB típusú XML dokumentummal tér vissza. Biztosít egy fetch interfészt, amelyben beállíthatjuk a maximum lekérdezett sorokat, vagy éppen azokat a sorokat, amelyeket kihagyjon. Ez különösen hasznos weboldalak lapokra bontásánál.

1. qryCtx := dbms_xmlgen.newContext ('SELECT * FROM FELHASZNALOK');
   ahol
   qryCtx DBMS_XMLGEN.ctxHandle;
2. Állítsuk be a csomag függvényeinek és eljárásainak segítségével a használni kívánt opciókat. pl: DBMS_XMLGEN.setRowSetTag(ctxHandle, 'felhasznalok'); DBMS_XMLGEN.setRowTag(ctxHandle, 'felhasznalo');
3. Szerezzük meg az XML eredményt a getXML() vagy getXMLType() függvények használatával.

A setMaxRows használatával beállíthatjuk a maximálisan feldolgozható sorok számát, valamint a getNumRowsProcessed() segítségével pedig megkaphatjuk az eddig feldolgozott sorok számát.

4. Szabadítsuk fel az erőforrásokat a closeContext() segítségével.

```
SELECT DBMS_XMLGEN.getXML
   ('SELECT *
   FROM felhasznalok
   WHERE id = 1')
FROM dual;
------------------------------------------------------------
<?xml version="1.0"?>
<ROWSET>
   <ROW>
        <ID>1</ID>
        <NEV>Nagy István</NEV>
        <SZUL_DAT>1994-06 -04</SZUL_DAT>
        <CIM>Debrecen Egyetem tér 1.</CIM>
   </ROW>
</ROWSET>
```

Egyéb függvények és eljárások a csomagban:

- setSkipRows()

  Hasznos lehet weboldalak tagolásánál. Az első oldalnál beállítjuk nullára és megadjuk hogy mennyi az a maximum amit lekérdezünk és egyúttal ki írhatunk egy oldalon. Majd a következő oldalnál beállítjuk arra a számra amit már kilistáztunk egyszer.

  Hiba következik be, ha egy olyan context-re hívjuk meg, amelyet a newContextFromHierarchy() segítségével kreáltunk.

- setConvertSpecialChars(ctx IN ctxHandle, conv IN BOOLEAN);

  Hogyha tudjuk hogy nem tartalmaz semmilyen speciális(végjelző, stb.) karaktert akkor beállíthatjuk igazra, javíthatja a teljesítményt.

- restartQuery()

  Újra indítja a lekérdezést, és ez által újragenerálja az XML-t az első sortól kezdve.

- setNullHandling(ctx IN ctxHandle, flag IN NUMBER);

  ```
  DROP_NULLS CONSTANT NUMBER := 0; Ez az alapértelmezett, kihagyja azt
  az elemet.
  NULL_ATTR CONSTANT NUMBER := 1; Ez beállítja :xsi:nil=”true”
  EMPTY_TAG CONSTANT NUMBER := 2; pl: <cím/>
  ```

- useNullAttributeIndicator(ctx IN ctxHandle,attrind IN BOOLEAN := TRUE); Ez gyakorlatilag setNullHandling(ctx, NULL_ATTR)rövid formája.

- setBindValue(ctx IN ctxHandle, bindVariableName IN VARCHAR2, bindValue IN VARCHAR2);

  Beállítja a paraméterek értékét.
- clearBindValue(ctx IN ctxHandle); Törli az összes paraméter értékét.

XMLtype típusú adatok módosítása

UPDATEXML() függvény:

Bármilyen típusú XML csomópont cseréjére alkalmas. Az eredeti adat módosítatlan marad, annak másolatával dolgozik a függvény, és azt is adja vissza.

Paraméterek:

- a módosítandó dokumentum, XMLType típusú
- Egy vagy több xpath – helyettesítési páros.
    - XPath utasítás ami azonosítja a cserélendő csomópontokat. Hogyha üres csomópontokra mutat, akkor nem történik módosítás.
    - Az új értéknek olyan típusúnak kell lenni, mint amilyen értékre az XPath utasítás mutat. Ez lehet csomópont, egyszerű érték. Attribútum esetén csak az értéknek kell szerepelnie.
- Névtér, opcionálisan megadható az XPath utasításhoz.

Példák:

```
UPDATE companys
SET  OBJECT_VALUE  =  updateXML(OBJECT_VALUE,  'company/name/text()',
'Új név')
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]') = 1;

select c.getClobVal()
from companys c
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]') = 1;
-----------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<company        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">
      <id>0040</id>
      <name>Új név</name>
      <address>address</address>
      <phone>phone</phone>
      <email>email</email>
      <info>info</info>
</company>
```

```
UPDATE companys
SET OBJECT_VALUE = updateXML(OBJECT_VALUE, 'company/name ',
      XMLType('<name>Új név 2</name>'))
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]') = 1;
```

```
SELECT c.getClobVal()
FROM COMPANYS C
WHERE existsNode(OBJECT_VALUE, '/company[id="0040"]' ) = 1;
--------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<company       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">
     <id>0040</id>
     <name>Új név 2</name>
     <address>address</address>
     <phone>phone</phone>
     <email>email</email>
     <info>info</info>
</company>
```

- Hogyha egy csomópontot NULL értékűre változtatunk, az attribútumai és a gyermek csomópontjai törlődnek, és az elem üres lesz. Az elem típus és a névtér tulajdonságai megmaradnak
- Ha attribútum értéket állítunk NULL-ra, akkor az üres stringként fog megjelenni.
- Ha az elem tartalmát frissítem NULL értékre, akkor az elem megmarad üres elemként.

INSERTCHILDXML függvény:

Gyermek csomópont, illetve attribútum beszúrására képes szülő csomópontok alá. A kiegészítendő XML dokumentum lehet séma alapú, vagy séma nélküli is. A dokumentum módosított másolatával tér vissza a függvény, az eredeti adat változatlan marad.

Paraméterei:

- Az XML dokumentum amely tartalmazza a módosítandó szülő csomópontot.
- Egy XPath utasítás, amely a csomópontra mutat. Hogyha üres csomópontokra mutat, akkor nem történik módosítás.
- Maga a beszúrandó csomópont neve, ha attribútumról van szó, akkor a „@” jellel kezdjük.
- A beszúrandó adat. Lehet VARCHAR2 vagy XMLType típusú érték is.
    - Hogyha XMLType típusú, és vannak csomópontjai, akkor a legfelső szintű csomópontnak meg kell egyeznie a paraméterként megadott csomópont névvel.
    - Ha attribútumot akarunk beszúrni, akkor VARCHAR2 típust kell használnunk. Ha már meg van adva az adott nevű attribútum akkor hiba váltódik ki.
- Névtér a szülő csomópont XPath utasításához, és a gyermek elemhez.

NULL értékek:

- Ha a gyermek neve null, hiba váltódik ki.

- Ha a cél dokumentum vagy a csomópontokra mutató XPath utasítás NULL, akkor NULL értékkel tér vissza.
- Ha a beszúrandó adat NULL:
    - Ha a név elemre vonatkozik, akkor nem történik módosítás.
    - Viszont attribútum esetén, üres attribútum érték kerül beszúrásra.

Tegyük fel hogy a cégek nyílván tartására szolgáló XML dokumentumban többször is feltűnhet a telefonszám, illetve email csomópont.

```
UPDATE COMPANYS
SET OBJECT_VALUE =
      insertChildXML(OBJECT_VALUE,'/company', 'phone',
      XMLType('<phone>06305353456</phone>'))
WHERE existsNode(OBJECT_VALUE,'/company[id="0040"]')= 1;
```

Ezek után a következőképp néz ki a dokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<company xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">
        <id>0040</id>
        <name>Új név 2</name>
        <address>address</address>
        <phone>phone</phone>
        <phone>06305353456</phone>
        <email>email</email>
        <info>info</info>
</company>
```

INSERTXMLBEFORE függvény

Egy vagy több, bármilyen típusú csomópont beszúrása közvetlenül egy csomópont elé, ami nem attribútum. A kiegészítendő XML dokumentum lehet séma alapú, vagy séma nélküli is. A dokumentum módosított másolatával tér vissza a függvény, az eredeti adat változatlan marad.

Paraméterei:

- A módosítandó XML dokumentum.
- XPath utasítás, ez által hivatkozott csomópontok elé történik meg a beszúrás.
- XMLType típusú csomópont, amit be szeretnénk szúrni.
- Névtér.

NULL értékek:

- Ha a cél dokumentum vagy a csomópontokra mutató XPath utasítás NULL, akkor NULL értékkel tér vissza.
- Ha a beszúrandó adat NULL, akkor nem történik módosítás.

```
UPDATE companys
SET OBJECT_VALUE = insertXMLbefore(OBJECT_VALUE,
      '/company[phone="phone"]/phone',
      XMLType('<phone>06704444456</phone>'))
WHERE existsNode(OBJECT_VALUE,'/company[id="0040"]')= 1;
```

Ezek után a következőképpen néz ki a dokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<company        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">
      <id>0040</id>
      <name>Új név 2</name>
      <address>address</address>
      <phone>06704444456</phone>
      <phone>phone</phone>
      <phone>06305353456</phone>
      <email>email</email>
      <info>info</info>
</company>
```

APPENDCHILDXML függvény

Egy vagy több, bármilyen típusú csomópont beszúrása egy csomópont utolsó gyermekének. A kiegészítendő XML dokumentum lehet séma alapú, vagy séma nélküli is. A dokumentum módosított másolatával tér vissza a függvény, az eredeti adat változatlan marad.

Paraméterei:

- A módosítandó XML dokumentum.
- XPath utasítás, ez által hivatkozott csomópontok gyermek csomópontjai lesznek a beszúrandó elemek
- XMLType típusú csomópont, amit be szeretnénk szúrni.
- Névtér.

NULL értékek:

- Ha a cél dokumentum vagy a csomópontokra mutató XPath utasítás NULL, akkor NULL értékkel tér vissza.
- Ha a beszúrandó adat NULL, akkor nem történik módosítás.

```
UPDATE companys
SET OBJECT_VALUE = appendChildXML(OBJECT_VALUE,'/company',
     XMLType('<phone>06905353456</phone>'))
WHERE existsNode(OBJECT_VALUE,'/company[id="0040"]')= 1;
```

Ezek után a következőképpen néz ki a dokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<company       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">
     <id>0040</id>
     <name>Új név 2</name>
     <address>address</address>
     <phone>06704444456</phone>
     <phone>phone</phone>
     <phone>06305353456</phone>
     <email>email</email>
     <info>info</info>
     <phone>06905353456</phone>
</company>
```

DELETEXML függvény

Egy vagy több, bármilyen típusú csomópont törlésére alkalmas. A kiegészítendő XML dokumentum lehet séma alapú, vagy séma nélküli is. A dokumentum módosított másolatával tér vissza a függvény, az eredeti adat változatlan marad.

Paraméterei:

- A módosítandó XML dokumentum.
- XPath utasítás, a törlendő elemekre mutat
- Névtér.

```
UPDATE companys
SET                          OBJECT_VALUE                              =
     deleteXML(OBJECT_VALUE,'/company/phone["06905353456"]')
WHERE existsNode(OBJECT_VALUE,'/company[id="0040"]')= 1;
```

Ezek után a következőképpen néz ki a dokumentum:

```
<?xml version="1.0" encoding="UTF-8"?>
<company        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="company.xsd">
      <id>0040</id>
      <name>Új név 2</name>
      <address>address</address>
      <phone>06704444456</phone>
      <phone>phone</phone>
      <phone>06305353456</phone>
      <email>email</email>
      <info>info</info>
</company>
```

Alkalmazás

A program maga egy egyszerű raktározási program, amely egy kisebb cég készlet nyílván tartási feladatait képes ellátni. Valójában nyomon lehet vele követni, hogy a termékek, hogyan kerültek be, mikor, és milyen termékekkel egy időben.

Adatbázis séma kialakítása

5 fő tábla van az adatbázisban:

- COMPANYS
  Cégek adatainak nyílván tartására szolgál.
- DRIVERS
  Sofőrök adatainak nyílván tartására szolgál.
- PRODUCTS
  Termékek adatainak nyílván tartására szolgál.
- PRODUCT_MOVEMENTS
  Termék mozgások adatainak nyílván tartására szolgál.
- ITEMS
  Termékmozgásokhoz tartozó tételek nyílván tartására szolgál.

Minden táblához tartozik ellenőrző séma, amelyet regisztráltam az adatbázisba, majd pedig azok felhasználásával hoztam létre a táblákat.

```
CREATE TABLE ITEMS OF XMLType
XMLSCHEMA "item.xsd"
ELEMENT "item";
```

Az elsődleges és külső kulcsok megadása a következőképp történt:

```
ALTER TABLE ITEMS
ADD CONSTRAINT PK_ITEMS PRIMARY KEY (xmldata."id");

ALTER TABLE ITEMS
ADD CONSTRAINT FK_ITEM_PRODUCT FOREIGN KEY (xmldata."id_product")
REFERENCES PRODUCTS(xmldata."id");
```

Az eredeti terv szerint, nem lett volna különválasztva a termékmozgásokat és a hozzájuk tartozó tételeket nyílván tartó tábla. Az eredeti termékmozgás séma a következő:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="productmovement">
    <xs:complexType>
      <xs:sequence>
         <xs:element name="id" type="xs:ID"/>
         <xs:element name="id_driver" type="xs:ID"/>
         <xs:element name="type" type="xs:string"/>
         <xs:element name="date" type="xs:dateTime"/>
         <xs:element name="item" maxOccurs="unbounded">
             <xs:complexType>
                 <xs:sequence>
                    <xs:element name="id_product" type="xs:ID"/>
                    <xs:element name="amount"
                    type="xs:nonNegativeInteger"/>
                 </xs:sequence>
             </xs:complexType>
         </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:key name="productmovement_id">
      <xs:selector xpath="."></xs:selector>
      <xs:field xpath="id"></xs:field>
    </xs:key>
    <xs:unique name="pm_id_product">
      <xs:selector xpath="./item"></xs:selector>
      <xs:field xpath="id_product"></xs:field>
    </xs:unique>
   </xs:element>
</xs:schema>
```

Ebből látható, hogy a tételek, magán a termékmozgáson belül, mint egyfajta kollekció jelentek volna meg.

Az Oracle, az XMLType-ként tárolt XML dokumentumokat típusok segítségével tárolja. Az egyszerű típusokhoz automatikusan skalár típusokat társít, míg a „complexType” típusú elemekhez saját illetve generált objektum típusokat társít. Azok az elemek, amelyeknél a „maxOccurs” paraméter nagyobb mint egy, kollekcióként tárolja.

Erre két lehetőség van:

- LOB-ként
- VARRAY-ként, beágyazott táblában (NESTED TABLE)

Mivel nagyméretű objektumként tárolva, nem lehetséges a külső kulcs megszorítás hozzáadása, a következőképpen próbáltam létrehozni a táblát.

```
CREATE TABLE PRODUCT_MOVEMENTS OF XMLType
XMLSCHEMA "productmovement.xsd"
ELEMENT "productmovement"
VARRAY "XMLDATA"."ITEM" STORE AS ITEMS_NTAB
```

Ezután pedig, a következő üzenetű hiba lépett fel, külső kulcs megszorítás megadása közben, ami szerint nem lehet megvalósítani az általam kigondolt struktúrát.

```
Oracle Database Error Code ORA-30730 Description :
referential constraint not allowed on nested table column
```

Adatkezelés

Az adatbázis kezeléséhez Hibernate-t használtam. A Hibernate egy nagy teljesítményű objektumrelációs perzisztencia és adatbázis lekérdező (query) szolgáltatás. A Hibernate lehetővé teszi az objektum-orientált elvet követő perzisztens osztályok létrehozását beleértve az asszociációt, öröklődést, polimorfizmust, kompozíciót, kollekciót. A Hibernate segítségével saját, hordozható SQL kiterjesztésében (HQL) is történhet a lekérdezés, valamint natív SQL-ben is, vagy pedig objektum-orientált Criteria és Example API segítségével.

Egy adatbázis táblához létrehozni perzisztens Java objektumot, csak olyan táblákon lehet amelyekhez létezik elsődleges kulcs. De a Hibernate nem támogatja az Oracle XMLType típusú táblákban, az objektumon belüli elhelyezkedő azonosítókat, ezért Natív SQL nyelvel értem el az adatokat, így jobban megismerhettem az Oracle, XMLType típushoz elérhető függvényeit.

```
...
Session s = StorageHibernateUtil.getSessionFactory().openSession();
Transaction t = s.beginTransaction();
SQLQuery select = s.createSQLQuery(
     "SELECT " +
     "extractValue(VALUE(P),'/product/id'), " +
     "extractValue(VALUE(P),'/product/serial'), " +
     "extractValue(VALUE(P),'/product/name'), " +
     "extractValue(VALUE(P),'/product/manufacturer'), " +
     "extractValue(VALUE(P),'/product/info'), " +
     "extractValue(VALUE(P),'/product/amount') " +
     "FROM PRODUCTS P ");
java.util.List result = select.list();
s.close();
...
```

Ezzel a lekérdezéssel, töltöttem fel az alkalmazásokban a „Termék típusok” lapon található táblát. A „list()” függvény, egy Object tömb típusú elemekből álló listát ad vissza. Melyből a megfelelő konverziókkal, könnyen létre lehetett hozni a megfelelő típusú objektumokat. Majd ezen objektumokat „ArrayList”-ként átadni az IceFaces GUI csomag, „datatable” komponensének. Az adatok módosítása is hasonlóképpen történt:

```
Session s = StorageHibernateUtil.getSessionFactory().openSession();
Transaction t = s.beginTransaction();
SQLQuery insert = s.createSQLQuery(
                  "INSERT INTO PRODUCTS VALUES(XMLType(':xml'))");
insert.setParameter("xml", XMLBuilder.createProductXML(product));
insert.executeUpdate();
t.commit();
s.close();
```

Egy String típusúvá alakított XML dokumentumot adunk át, a „list()” függvény helyett az „executeUpdate()” hívódik meg, majd pedig véglegesítjük a változtatásokat.

Összefoglalás

Dolgozatom írása közben, számos érdekes technológiával, koncepcióval és megoldással találkoztam, próbáltam megtalálni a leghasználhatóbbakat.

Világossá vált számomra hogy az XML-nek jelentős a létjogosultsága, főleg platform és gyártó független mivolta miatt. Az Oracle kiválóan alkalmazza az XML technológiák nagy részét, ugyanakkor nem minden lehetőséget sikerült megvalósítaniuk, amire egy natív XML adatbázis képes.

Ugyanez a helyzet, a manapság használatos keretrendszerekkel, mint például a Hibernate. Kiválóan használható több gyártó adatbázis kezelő termékéhez, viszont ennek következtében nem minden adatbázis specifikus lehetőség kerül bele a rendszerbe. Támogatottság híján, nem tudtam kipróbálni az esetleges teljesítménybeli illetve lehetőségbeli különbségeket a HQL és SQL között.

A dolgozatom során inkább került előtérbe a minél több, változatos XML technológia megismerése, bemutatása. És még így is van olyan rész, amit nem sikerült igazán megismernem, ilyen például az Oracle XML DB. Ennek megismerésére szeretnék időt szánni a jövőben.

Alkalmazásom fejlesztése során, megismerkedhettem a Java Enterprise környezettel, ezen belül a Java Server Faces technológiával. Megjelenítési minőségben közel azonos szintet hoz, egy asztali alkalmazás szintjével, megbízhatóságban, hibatűrő képességben viszont jelenleg még elmarad. Remélem a későbbiekben, munkám során is lesz szerencsém találkozni ezen technológiák valamelyikével, hogy mélyebben megismerhessem őket, nálam jóval tapasztaltabb programozóktól tanulva.

Köszönetnyilvánítás

Ezen szakdolgozat elkészítésében nyújtott segítségért, irányadó tanácsokért, és útmutatásért köszönettel tartozom témavezetőmnek, dr. Adamkó Attilának.

Irodalomjegyzék

Beginning XML databases - Gavin Powell

Beginning XML - David Hunter

Beginning Database Design - Gavin Powell

XForms Essentials – Micah Dubinko

XForms: XML Powered Web Forms – T. V. Raman

XQuery – Priscilla Walmsley

XSLT 2.0 and XPath 2.0 Programmer's Reference - Michael Kay

Beginning XSLT and XPath: Transforming XML Documents and Data – Ian Williams

XML Problem Design Solution - Mitch Amiano

Effective XML: 50 Specific Ways to Improve Your XML - Elliotte Rusty Harold

Oracle® XML DB Developer's Guide 10*g* Release 2 (10.2) – http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14259/toc.htm