Bevezetés

A WEB-adatbázis programozás igazából nem egy elszeparált, önálló terület az informatikán belül, hanem sokkal inkább néhány más terület kombinációjából adódó alkalmazási ág. Hogy így külön foglalkozunk vele, arra az ad okot, hogy a mai gyakorlatban az alkalmazások jellemző architektúrája nagyban eltolódott a klasszikus kliens-szerver architektúrák felől a legalább 3 rétegű felépítés irányába. Ez utóbbi esetek 95%-ban pedig a kliens oldali réteg valamilyen WEB-es felület, a „leghátsó" szerver oldali réteg pedig egy adatbázis.

A klasszikus numerikus és szöveges adatmegjelenítő –kereső –módosító alkalmazások mellett nagyon érdekes terület a multimédiás WEB-es adatbázisok világa. Erre külön koncentrálni fogunk a félév során

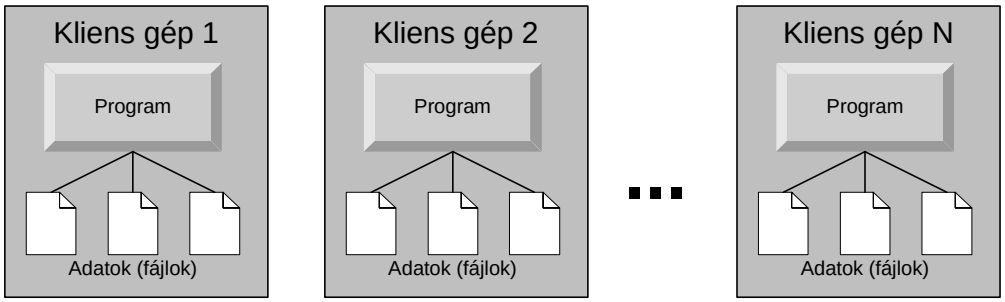
Tekintsük át, hogy milyen fontos területekkel kell foglalkoznunk, hogy aztán magunk is a mostani kornak megfelelő WEB-adatbázis alkalmazásokat fejleszthessünk! Az egyes pontok után 'E' betűvel jelöltem, hogy ez a tudás a mostani tárgyunk előfeltétele, és csak ismétlésképpen foglalkozunk vele, illetve annyiban, amennyiben azt szükséges átlátnunk, hogy az adott pont hogy illeszkedik más területekhez a WEB-adatbázis programozáson belül. Az 'F' betű azt reprezentálja, hogy részletesen foglalkozunk vele a félév során.

- WEB-es prezentációs, hálózati, szerver és kliens oldali megoldások (HTML nyelv, TCP/IP és HTTP protokoll és működése, WEB szerverek, böngészők, kliens oldali WEB programozás alapjai (pl. JavaScript)) ['E']
- Adatbázis-kezelés (a relációs modell, adatmodellezés, SQL) ['E']
- Rendszerek közti adatkommunikáció „önleíró" dokumentum nyelven = XML (XML felépítése, használata, kapcsolódó technológiák érintőlegesen: DTD, XSD, XSL ill. XSLT) ['F']
- XML alapú adatbázisok (XML adattárolás alapjai, lekérdező nyelvek: XPath, XQuery) ['F']
- Multimédiás adatbázisok (nagy méretű multimédiás anyagok tárolása adatbázisokban, visszakeresés, hatékonyság) ['F']
- Programozási módszertan ['E']
- WEB programozás (módszerek, beágyazott script-nyelvek általában, PHP részletesen) ['E']
- Multimédiás WEB programozás – bináris tartalmak (stream-ek, header, letöltés, feltöltés) ['F']
- Multimédiás WEB programozás – vektorgrafikus és programozott tartalmak (SVG, Flash) ['E'/'F']
- Informatikai biztonság (adatvédelem, kommunikációs vonalak védelme, védelem illetéktelen behatolásokkal szemben, meghibásodások elleni védelem) ['E'/'F']

Architektúrák

Egygépes alkalmazások

Architekturális szempontból a legegyszerűbb informatikai rendszerek az egygépes (un. standalone) alkalmazások.

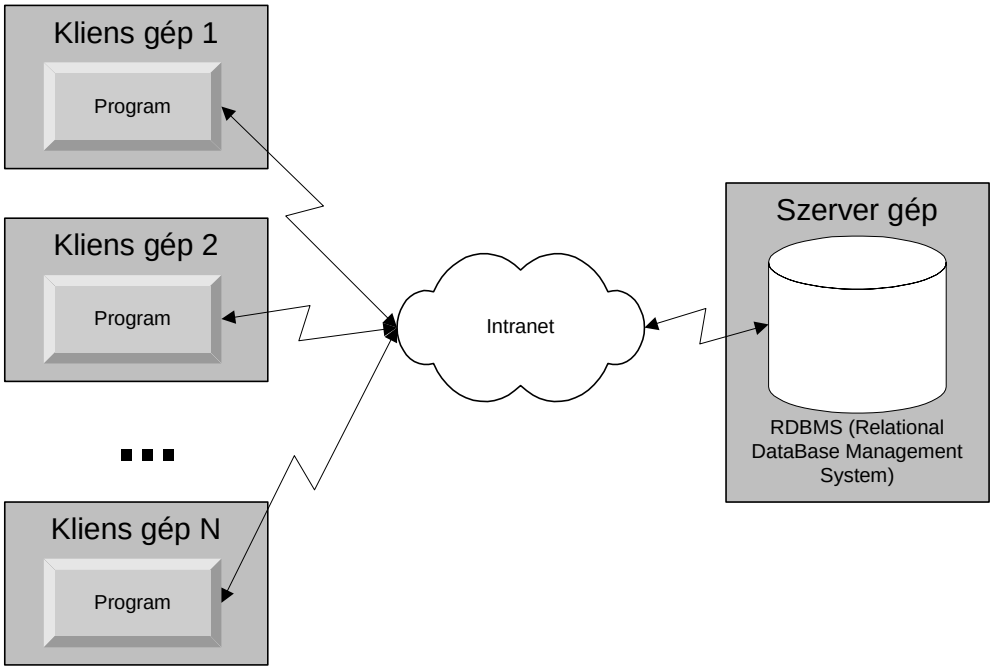


Jellemzői:

- A program teljes egészében azon a munkaállomáson fut.
- Az adatok ugyanitt tárolódnak.
- Egyszerre csak egy felhasználó használhatja.
- Semmilyen hálózati kapcsolat nincs, a különálló programok közti adatszinkronizáció meglehetősen nehézkes.

Egyszerű kliens-szerver alkalmazások

Hosszú ideig az ilyen jellegű rendszerek alkották „a modern, centralizált rendszerek” kategóriát.



Jellemzői:

- Egy vagy több (de tipikusan egy) szerver gép a saját erőforrásait (jellemzően adatait) valamilyen hálózati kapcsolat segítségével megosztja a kliensek között. Ez a megosztás jobb esetben on-line.
- Az alkalmazás egy része (tipikusan az adatbázis-kezelő rendszer) a szerven fut.
- Az alkalmazás logikát implementáló rész a kliens gépeken fut. Ezért ezeket a programokat „***vastag kliens rendszerek***nek” nevezzük.
- Egy adatbázist többféle kliens program is használhat.
- Az előző miatt az adatbázis módosítások nagyon nehezen megoldhatók.
- Szerver oldali „közös” programrészek készítésére csak kevés és gyenge eszköz van.
- Egyszerre több konkurrens felhasználó használhatja.
- Jellemzően intranet-es alkalmazásoknál használatos, az Internet-en különböző biztonsági megfontolásokból elég ritkán.
- Terheli a kliens gép erőforrásait, gyakran „izmos” kliensekre van szükség.
- Gyakran mindenféle driver-ek telepítését igényli a kliens gépeken (pl. adatbázis kliens szoftver)
- Verziófrissítés alkalmával az összes kliens-en frissíteni kell a programot.
- A RAD (Rapid Application Development) sok eszközzel támogatott, számos jó vizuális fejlesztőkörnyezet létezik, amelyben gyorsan „összekattintgathatunk” és leprogramozhatunk aránylag komoly alkalmazásokat.

Ezek a kliens-szerver alkalmazások, bár lokálisan általában megoldják az adatcentralizálás problémáját, mégis rendelkeznek néhány –a jellemzőkben felsorolt– negatív tulajdonsággal.

Többrétegű hálózati alkalmazások

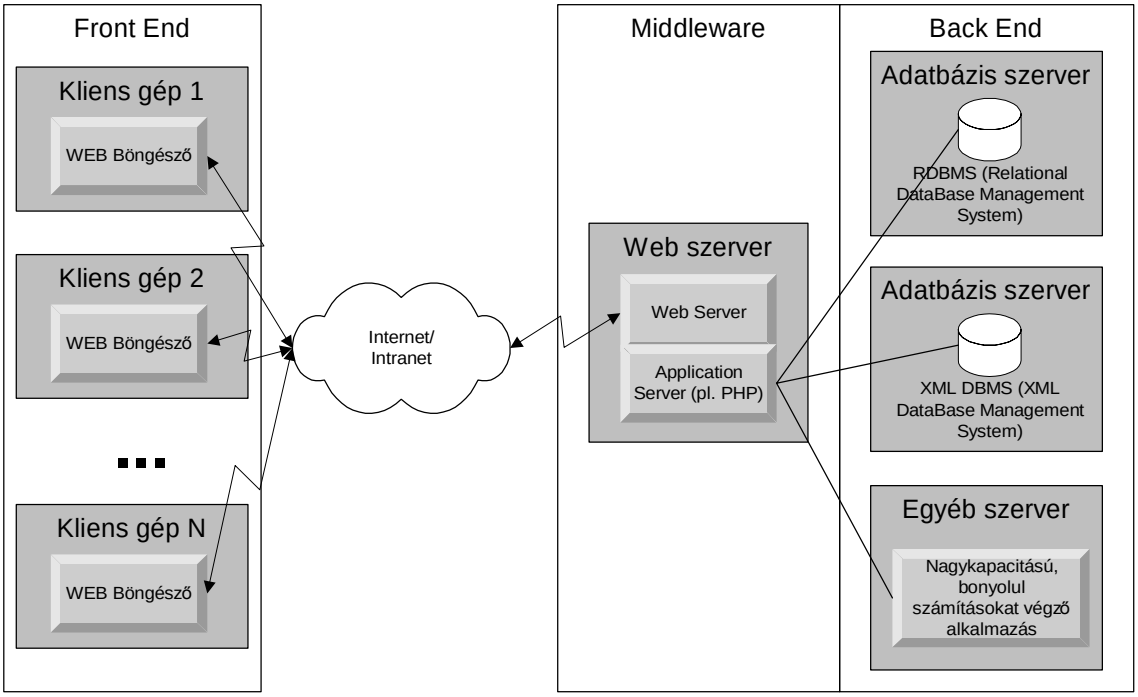
Ez a technológia szól a jelenről és a jövőről. A WEB-adatbázis programozás tipikusan ilyen architektúrára épít (többrétegű = multitier).

Egy részletes prezentáció a kétrétegű és háromrétegű alkalmazások összehasonlításáról:

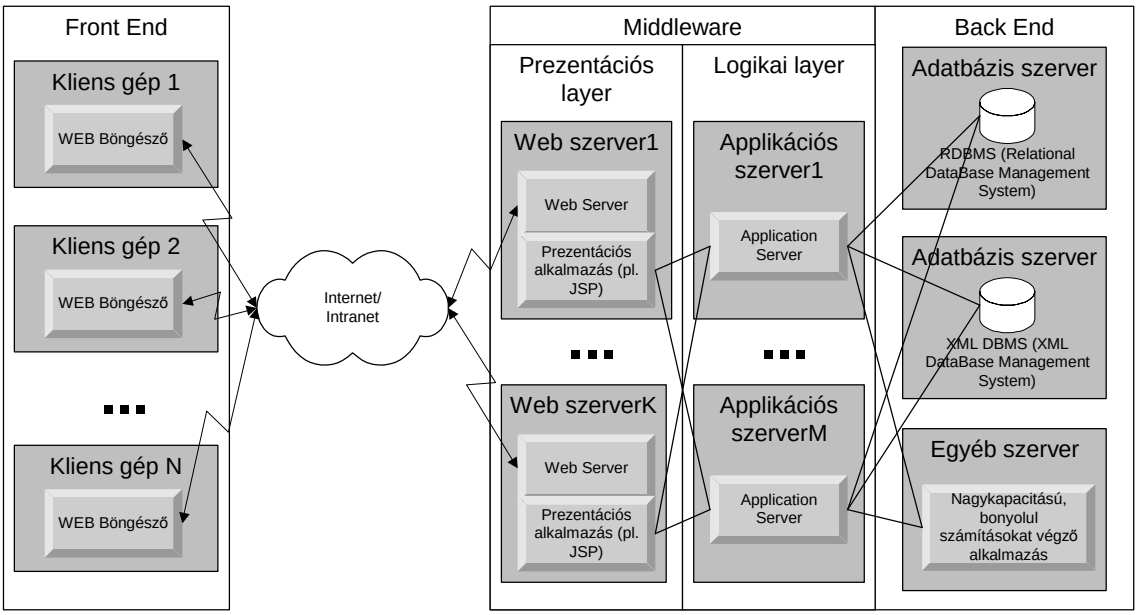
http://www.inf.bme.hu/~frigo/frme8/tiers.ppt (Copyright Frigó József, BME)

Minimálisan három réteg létezik:

- Front End = kliens oldali felhasználói réteg (általában egy WEB böngészőben)
- Middleware = szerver oldali prezentációs és logikai réteg (általában egy WEB szerveren beágyazott script-ekben összeolvasztva a megjelenítés és az egyszerűbb logika)
- Back End = hátsó szerver oldali nagykapacitású tároló (adatbázis szerver) vagy számoló réteg



Szofisztikáltabb rendszerekben a middleware prezentációs és logikai részét szétbontják, így kapjuk a négyrétegű alkalmazásokat.



Ezen az ábrán már azt is feltűntettük, hogy lehetnek teljesen azonos funkciókat ellátó szerverek az architektúrában (pl. Web szerver 1..K és Applikációs szerver 1..M). Itt az azonos funkciókat ellátó gépek a terhelést egymás között megosztják (load balacing). Néhány környezet (pl. JAVA J2EE) hatalmas előnye, hogy az alkalmazás tervezésénél ilyen aspektusokkal nem kell foglalkozni. Ez praktikusan annyit jelent, hogy ha azt látom, hogy a WEB és prezentációs szerverem nagyon terhelt (pl. állandóan 90%-on fut), és lassú reakcióidőket produkál, akkor egyszerűen üzembe állítok egy újabb WEB és prezentációs szervert. Ezután a két gép a terhelést egymás között automatikusan megosztja úgy, hogy a kliens-en a felhasználók nem is tudják, hogy valójában fizikailag melyik WEB szerver

szolgálta ki őket. Ugyanilyen load balancing lehetséges a logikai réteg alkalmazás szerverein is.

A dolog fordítva is igaz, nem csak szétbontani, hanem összevonni is lehet: a Middleware és a Back End különböző alkalmazásai nem feltétlenül vannak fizikailag külön számítógépen, az egész lehet egy nagy „vas” is.

Azt látjuk tehát, hogy a rendszer logikai architektúrája független a fizikai számítógépes megvalósítástól és a hálózattól. Összes szint egy gépen, vagy minden szint külön gépen, vagy szintenként az azonos funkciók több gépre duplikálva; logikailag nem számít (az alkalmazás tervezésénél és programozásánál)!

Nagyobb rendszerekben a logikai rész még több szintre osztódhat, ílymódon akárhány réteg is születhet.

Azoknak a hallgatóknak, akik szeretnének bővebben ismerkedni a multi-tier alkalmazásokkal, a következő két technológiával való ismerkedést javaslom. A fentiek mindegyikben (de főleg az elsőben!) szépen megvalósításra kerültek:

- J2EE (JAVA Enterprise technológia): http://java.sun.com/j2ee
- Microsoft .Net Framework és szerver oldali megoldások: http://www.microsoft.com/net/

Mi maradunk a háromszintű architektúránál és a PHP-nál. A PHP az előző kettőnél jóval kevesebb lehetőséget nyújt a funkciók szétválasztására, azonban mégis nagyon elterjedt világszerte. Ennek oka elsősorban az, hogy az Apache WEB szerverrel együtt nagyon hatékony, ténylegesen képes akár több ezer konkurrens (párhuzamosan beérkező) kérést kiszolgálni, a programozóknak egyszerű megtanulni, a nyelvet direkt erre találták ki.

A funkciók nehéz szétválaszthatósága elsősorban a prezentációra és a logikára vonatkozik. Beágyazott script nyelveknél, ahol pl. egy PHP oldal tartalmaz egyrészt statikus HTML tartalmat (mint prezentáció), másrészt pedig kódrészleteket, amik dinamikus HTML tartalmat állítanak elő (mint prezentáció és logika összegyúrva); elég nehéz e kettőt szétválasztani. Maga a környezet nem is kínál erre külön eszközöket, hanem általában a programozók saját maguktól jönnek rá egy bizonyos komplexitás után, hogy jó tervezés és tudatos szétválasztás hiányában nem tudják szépen struktúrálni az alkalmazást, és kezd az egész kód átláthatatlanná válni számukra. Ez számos veszélyt jelent, hiszen a szoftver tesztelése jóval bonyolultabb lehet, nem is beszélve arról, hogy a team munka nagyon nehézkes. Ez a témakör a félév során néhányszor előkerül majd. Alapvetőn azt tudjuk javasolni, hogy azok a PHP kódok, amik végül megfelelnek majd egy-egy generált HTML fájlnak (prezentáció) csak olyan egyszerű PHP programsorokat tartalmazzanak, amik meghívják a logikát tartalmazó PHP-kat, és az eredményt kiírják. A többi PHP fájl (logika) pedig csak adatfeldolgozással, adatbázis-eléréssel, stb. foglalkozzon, a HTML output-tal ne. Ez ebben a formában elég egyszerűnek hangzik, de a gyakorlatban nem is olyan egyszerű… Természetesen az olyan alapvető WEB tervezési iránymutatások, miszerint használjunk CSS-eket, stb., itt is érvényesek.

Ezek után nézzük a többszintű architektúra jellemzőit:

- Egyszerre nagyon sok felhasználó használhatja. A szerver oldali terhelés a speciális technológiáknak köszönhetően sok konkurens felhasználó esetén jóval kisebb lehet, mint ugyanannyi felhasználónál a klasszikus kétszintű architektúra szerverén. Erre eklatáns példa, hogy a WEB szerverek direkt arra lettek optimalizálva, hogy nagyon sok párhuzamos HTTP kérést szolgáljanak ki.

- A kliens gépen csak egy böngésző található, szinte minden logika a szervereken található. Ezért ezeket az alkalmazásokat „***vékony kliens rendszerek***nek” nevezzük.
- Az a minimális logika, amit a vékony kliensre helyezünk, többnyire csak a beviteli adatok validálására, valamint a lapok speciális megjelenítésére szolgál (pl. JavaScript).
- A szerveren elkülönül az adattárolás, a logika és a prezentáció. Az egyes szinteket azok a programozók implementálhatják, akik arra specializálódtak (pl. adatbázis alkalmazás készítő, web programozó, stb.), így specializálódott szerepkörök miatt professzionálisabb alkalmazások készíthetők.
- A rendszer különböző szintjei önmagukban is tesztelhetők.
- A rendszer egyes komponensei több célra vagy újra felhasználhatók. Az applikációs logika pl. kiszolgálhatja más rendszerek kéréseit (pl. XML-ben jön egy kérés, és XML-ben válaszolunk), és a prezentációs szint kéréseit egyaránt.
- Enterprise rendszereknél lehetőség van olyan speciális megoldásokra, mint a load balancing vagy a connection pooling.
- Az adatbázis módosítások könnyebben megoldhatók.
- A vékony kliensek miatt nagyon gyenge kliens gépek is elegendők, valamint a technológia platformfüggetlen.
- Mivel csak egy böngészőre van szükség, semmilyen driver-t a kliensre telepíteni nem kell.
- A verziófrissítés csak a szervert érinti, a sok-sok klienst nem.
- Sajnos egyelőre elég kevés eszköz támogatja a RAD-ot (Rapid Application Development). Bár vannak elsőre egész jónak tűnő fejlesztő környezetek (pl. Sun Studio, Borland JBuilder, Visual Studio .Net), azonban komoly, többszintű alkalmazások esetén a fejlesztői munka mégis sokkal nehézkesebb, mint a kétrétegű alkalmazásoknál. A nehézkesség arra is értendő, hogy egy adott dolgot architekturálisan sokféleképpen meg lehet oldani, és a számos lehetőség közül történő kiválasztásban a környezet kevés segítséget nyújt a programozónak.

Az utolsó pont gondolatmenetét folytatva, az a tapasztalat, hogy érdemes a sokféle megoldás közül saját magunk számára „házi szabványokat” bevezetni, és a jellemző részfeladatokat mindig eszerint megvalósítani. Persze a leghatásosabb módszer mindig csak hosszú tapasztalat után bontakozik ki a programozó előtt.

Néha a programozók elkezdenek saját keretrendszert építeni. Ez egy evidens gondolat, mikor az ember felfedezi, hogy a WEB-adatbázis programozásban vannak tipikus felhasználó oldali elemek. Szerepeljen itt erre egy példa: Nagyon gyakori az olyan három WEB oldalas alkalmazásrészlet, ahol a felhasználónak azt akarjuk megengedni, hogy egy nagy adathalmazból (pl. adattábla) keressen ki valamilyen rekordot, annak részleteit megtekintse, esetleg módosítson. A három oldal a következő:

- Első oldalon a felhasználó megad egy feltételt (pl. kitölt bizonyos mezőket egy HTML form-on).
- A második képernyőn megjelennek a feltételnek megfelelő sorok táblázatos formában. A táblázat csak a legfontosabb információkat tartalmazó oszlopokat tartalmazza, és az oszlop fejlécére kattintva lehet rendezni az adott attribútum szerint. A táblázat minden sorában szerepelhet pl. egy piros X, amivel a rekordot törölni lehet. Mivel egy feltételnek nagyon sok rekord is megfelelhet, ezért ezt az oldalt szokás un. lapozásos

technikával megoldani. Ennek lényege, hogy csak az első N darab sor jelenik meg, és alul link-ekkel lehet ugrani a következőre, előzőre, legelsőre, legutolsóra, stb. Ha a felhasználó egy adott sorra kattint, akkor a harmadik oldalra jut.

- A harmadik oldalon a kiválasztott rekord összes részletes adata megjelenik adatlap formában, ahol esetleg módosítani is lehet. Ugyanez az oldal nem harmadikként, hanem elsőként és üresen jön fel, ha új rekordot akarunk beszúrni.

Ez egy aránylag kultúrált felhasználó oldali megoldása annak, hogy a user-ek egy táblában a négy fontos DML műveletet végre tudják hajtani.

Ezt a feladatot meg lehet általánosan oldani (pl. PHP-ben). Az ember később csak beparaméterezi, hogy melyik táblából jöjjenek az adatok, továbbá hogy az egyes oszlopokat hogy szeretnénk hívni a felületen, stb., és egy általános kód működhet tetszőleges adattáblára.

A megoldásnak van viszont egy hatalmas hátránya, amelybe sok programozó és informatikai megoldásszállító beleesik. Az élet általában nem ilyen egyszerű, mint a fenti példa, és a felhasználó hirtelen elkezd speciális dolgokat igényelni (pl. a második oldalon felül jelenjen meg valami speciális adat valami más adatforrásból, és a táblázatból lehessen ugrani valami más funkcióra, és a harmadik oldalon ne csak text mezők legyenek, hanem adott esetben egy listából lehessen kiválasztani, stb.). Itt két nagy hiba követhető el:

1. A felhasználónak megmagyarázni, hogy jó ez úgy, ahogy van. Ez azért nagy hiba, mert az elegáns informatikai megoldás oltárán feláldozzuk a valós felhasználói igényeket. Holott egy igazi megoldás szállító számára ez kell legyen a legfontosabb prioritás.
2. A végtelenségig általánosítani az alkalmazást, hogy mindenféle spéci kérésnek eleget tudjon tenni. Ez egy idő után annyira komplexszé teszik a saját keretrendszert, hogy annak paraméterezése egy adott feladatra esetleg bonyolultabb lesz, mint az egészet nulláról lefejleszteni. Itt még hatékonysági és tesztelési problémák is felléphetnek.

Akkor mi a megfelelő megoldás? Erre nincs általánosan jó válasz. A válasz sajnos az, hogy „attól függ…”. Néhány szempontra igyekszünk rávilágítani a félév során, azonban hangsúlyozzuk, hogy igazi WEB-adatbázis programozónak csak az nevezheti magát, aki a saját tapasztalati alapján adott szituációban meg tudja válaszolni a fenti kérdést.

Multimédiás adatbázisok

A fentiekben keveset szóltunk a multimédiás tartalmak kezeléséről. Azért kell ezzel külön foglalkozni a WEB-adatbázis programozáson belül, mert a nagyméretű bináris adatok kezelése az összes szinten (kliens, prezentációs, logikai, adatbázis) eltér attól, mint ahogy az elemi numerikus és kis szöveges adatokat kezelni kell.

- Kliens: A böngészőn esetleg speciális plug-in szükséges a tartalom megjelenítéséhez.
- Prezentációs és logikai szint: A tartalmakat nem tudjuk egyszerűen változókba tölteni, mint elemi típusok esetén, hanem stream-ekkel kell dolgozni. Adott környezetben (pl. PHP) ennek a technikáját meg kell ismerni. Néha a HTTP protokoll részletes ismerete is szükséges, hiszen alap dinamikus WEB oldalakat általában a környezet default beállításaival le tudunk küldeni a kliens-re, míg multimédiás tartalom esetén a HTTP header-t gyakran módosítani kell.
- Adatbázis: A nagyméretű bináris adatok tárolása pl. relációs adatbázisokban; valamint a visszakeresés, módosítás egy külön nagy kutatási területet jelent az adatbázisokkal

foglalkozó ágon belül. Ez korrelál bizonyos mesterséges intelligenciabeli módszerekkel is.

A fenti három közül a legérdekesebb az, ami az adatbázisokkal kapcsolatos. Itt is a hatékony keresési módszerek megvalósítása a legizgalmasabb (pl. keressük meg az összes olyan képet, amin alma van → egy adott pixelmátrixhoz valamilyen szempont szerint „legközelebb” eső képek kikeresése, a hatékony kereséshez szükséges index struktúrák kezelése, a feladat komplexitása, időigény stb.). Természetesen nekünk az adott módszerek megvalósításával nem kell foglalkoznunk, azonban érdemes ismerni, hogy bizonyos adatbázis-kezelők (pl. Oracle) hogy oldják meg ezt, és mit kell tudni ahhoz, hogy mi is képesek legyünk ezeket a funkciókat az adatbázisban használni.

Nyilván a legegyszerűbb módja a multimédiás tartalmak közti keresésnek az, amikor minden elérhető anyaghoz tartozik valamilyen szöveges alapú leírás is. Ilyenkor a keresés nem binárisan a képen történik, hanem a leíró szövegrészben. Ez az egyik oka, hogy az XML, mint általános adatleíró nyelv fontos lesz számunkra.

A másik oka pedig az, hogy multimédiás tartalmak nemcsak binárisak lehetnek, hanem valamilyen elemi adatokkal kódoltak is (pl. vektorgrafikus kép, flash, stb.). Ugyanitt beszélünk majd az MPEG7-ről.

Már itt a bevezetőben érdemes megemlíteni, hogy a multimédiás alkalmazások készítésének a WEB-en jóval erősebb korlátai vannak, mint az elemi adatokkal dolgozó alkalmazások készítésének. Tipikusan olyan programokat, amikkel a kliensen multimédiás anyagokban transzformációkat akarunk elvégezni, nagyon-nagyon nehéz, szinte lehetetlen készíteni. Ha csak egy olyan egyszerű funkcióra gondolunk, mint pl. egy video anyag feliratozása (itt a feladat az, hogy adott képkockától adott képkockáig logikailag hozzárendelünk egy feliratot a video részlethez), azt látjuk, hogy vastag kliens megoldással ez gyorsan leprogramozható, míg WEB-es technológiával szinte reménytelen. Ennek oka, hogy a böngészőnkben videót a plugin-jeinkkel lényegében csak lejátszani tudunk, de képkockánként sorszámozottan megjeleníteni nem. Rossz esetben ilyenkor születnek olyan faramuci megoldások, hogy minden N. képkockát képenként (jpg) letöltik az egy WEB oldalra, és így lehet elvégezni a feliratozást (kijelölni a felirat elejét és végét). Mivel azonban ekkor hang nincs, egy olyan anyagnál, ami pl. egy interjú (végig egy ember feje látszik, tehát kvázi az összes képkockán ugyanazt a fejet látom), elég nehéz a feliratot a megfelelő helyre betenni.

XML és tárolás, visszakeresés

Az XML, mint általános adatleíró nyelv, egyre elterjedőben van az informatikai világban. Hatalmas előnye, hogy „önleíró”, tehát nem csak az adatokat tartalmazza, hanem azt is, hogy mi az az adat („Kiss József” helyett <név>Kiss József</név>).

Az adatok egymásba ágyazása miatt minden ilyen dokumentum egy hierarchikus struktúrát alkot. A kérdés tehát, hogy XML adatokat hogyan lehet jól relációs adatbázisban tárolni, illetve érdemes-e egyátalán a relációs adatmodellre építkezni (hiszen a táblázatos és a hierarchikus ábrázolásmód eléggé távol áll egymástól, így a konverziók oda-vissza néha nagyon sok programozást, később pedig hosszú futási időt vesznek igénybe), illetve a szokásos XML technológiák hogy érvényesíthetők egy adatbázison belül (validálás: XSD, transzformáció: XSLT).

Az XML-re, mint alap leíró nyelvre több grafikus formátum is épül. Amit mi részletesen nézünk, az az SVG (Scaleable Vector Graphics), és amit érintőlegesen vizsgálunk, az az MPEG7.