SZAKDOLGOZAT

Rózsa László

Debrecen

2007

Debreceni Egyetem

Informatika Kar

XML ALAPÚ WEBALKALMAZÁS FEJLESZTÉSE A PEDAGÓGUSOK TÚLMUNKÁJÁNAK ELSZÁMOLÁSÁRA

Témavezető:
Adamkó Attila
számítástechnikai munkatárs

Készítette:
Rózsa László
pedagógus szakvizsgázó hallgató

Debrecen

2007

Tartalomjegyzék

I. Bevezetés

2006. szeptemberétől a pedagógusok munkájának elszámolására új szabályozás lépett életbe. Régebben az elszámolás alapja a kötelező órák megtartása után járó fix munkabér, valamint a kötelező órákon túl megtartott túlórák után fizetett átalány, vagy tételesen elszámolt óradíj volt.

Ennek a rendszernek természetesen voltak előnyei és hátrányai egyaránt. Gyakran okozott például vitát az, hogy ha egy adott napon tanórák maradtak el valamilyen rendezvény miatt, akkor volt, akinek ezek az órák a kötelező óráiból maradtak el, s ez az óraelmaradás a fix munkabérét nem érintette, volt viszont, akinek túlórája maradt el, s így az elmaradt óra után nem kapta meg a túlóra díját.

Mindezek mellett legrosszabbul az iskola fenntartója járt, mert az elmaradt órát a pedagógusok többségének ki kellett fizetni, miközben ténylegesen megtartott óra nem volt mögötte, hanem az elmaradt órák helyett értekezleten, vagy mondjuk egy szavalóversenyen tartózkodtak a kollégák.

Mivel ez az elszámolás csak a túlórák megtartását, vagy elmaradását vette figyelembe, ezért előfordulhatott, valakinek elmaradtak a kötelező órái, tehát nem dolgozta le a heti kötelező negyven munkaórát, és ennek ellenére mégis volt túlórája. Ezeket a visszásságokat kívánták orvosolni az új rendszerrel, olyan módon, hogy túlmunkát csak abban az esetben lehessen elszámolni, ha a kötelező heti óraszámot a pedagógus teljesítette. Ez egyben azt is jelenteti, hogy a pedagógusok elvégzett munkáját sokkal pontosabban kell adminisztrálni, hiszen e nélkül nem tudjuk eldönteni, hogy valójában hány órát dolgozott a pedagógus. Minden pedagógusnak 20 +2 kötelező órája van -természetesen ebből levonhatók az órakedvezmények-, s csak miután ezt hiánytalanul megtartotta, az ezen felül végzett munkájáért jár bérezés. A törvény így fogalmaz:

> *„...A pedagógus-munkakörben foglalkoztatottak munkáját - a munkaidőkeretre vonatkozó rendelkezések [Mt. 118/A. §] alapul vételével - oly módon kell megszervezni, hogy a pedagógus a heti kötelező óraszáma egy tanítási évre jutó*

időkeretét teljesíteni tudja. Ehhez a munkáltató a pedagógus-munkakörben foglalkoztatottak munkáját kéthavi tanítási időkeret kialakításával szervezi meg. A tanítási időkeretet a következők szerint kell megállapítani: a két hónapra jutó tanítási napok számát meg kell szorozni a pedagógus-munkakörre megállapított heti kötelező óraszám egyötödével. A tanítási időkeretet a munkakörre megállapított heti kötelező óraszám egyötödével csökkenteni kell minden olyan kieső tanítási nap után, amely az Mt. 151. §-nak (2) bekezdésében megjelölt távollét napjaira vagy a keresőképtelenség időtartamára esik. A tanítási időkeret teljesítésénél a ténylegesen megtartott, továbbá a pedagógus heti kötelező órájának teljesítésébe beszámítható órák vehetők figyelembe. A tanítási időkeret teljesítésébe a II/18. pont alapján egyéni foglalkozás, szabadidős foglalkozás, tanórán kívüli foglalkozás megtartására elrendelt többlettanításból be kell számítani az egy tanítási héten a két órát meghaladó többlettanítási órát. A rendes munkaidőn belül végzett tanításért óradíj a tanítási időkereten felül teljesített többlet tanításért állapítható meg." [1]-ből a 2006 évi LXXI tv. 20§ (2) bekezdése alapján.

Különös figyelmet érdemel itt a kötelezően elrendelt plusz 2 óra, melyet felzárkóztatása, tehetséggondozásra, iskolán kívüli tevékenységre kell fordítani. Nagyon sok esetben a tanárok túlórája nem tanítási órákból ered, hanem szakkörök, korrepetálások, felzárkóztató foglalkozások tartásából. Most, akinek a túlórái ilyen jellegű tevékenységből adódtak, két órát köteles díjazás nélkül megtartani, s csak az ezen felüli órákért kaphat túlóradíjat. Érdekesség, hogy akinek nem ilyen jellegű a túlmunkája, ugyan foglalkozást ő is köteles tartani, de ennek elmaradása esetén is kifizetik a túlóradíját. További érdekessége a rendszernek, hogy akinek a kötetező óraszáma sincs meg, az alapbérét ez esetben is megkapja. Nem mondhatjuk tehát, hogy az új rendszer a valóban megtartott órák alapján díjazza a dolgozókat, de mindenképpen pontosabb, mint a korábbi. Előnyei mellett hiányosságai természetesen ennek a rendszernek is vannak, a gyakorlati tapasztalatok még csak most gyűlnek, s ennek fényében módosítják a rendszert, például legutóbb januárban történt változás az időkeretből levonható órák számát illetően. Továbbra is hátrányként említhető viszont, hogy a tanárokat arra ösztönzi, hogy szakköröket, felzárkóztató foglalkozásokat ne tartsanak, erre is kell majd valami megoldást

keresni. A rendszer minősítése viszont nem feladata e dolgozatnak, hátrányokat és előnyöket kizárólag példaként említünk az új rendszer bemutatása céljából. Számunkra mindezeknek annyi következménye van, hogy az elszámolás, az adminisztráció a fentiekből is láthatóan elég bonyolult, ami többletterhet rak a pedagógusok nyakába.

Ez a dolgozat arra vállalkozik, hogy ezt a terhet csökkentse egy helyi hálózatban -intraneten- működő web alapú alkalmazás elkészítésével. A tanárok bármelyik kliens gépről elérhetik az alkalmazást, az elszámolt órák egy adatbázisban lesznek tárolva, így az ellenőrzést végző igazgatóhelyettes, vagy gazdaságvezető szintén könnyedén hozzáférhet az adatokhoz.

II. A feladat leírása

Az elszámolási rendszer lényege az elszámolási időszak. Ez általában egy két hónapos intervallum. Ilyen időközönként összesítik a kötelező órákat, a ténylegesen megtartott órákat, szakköröket, helyettesítéseket. Ez lesz az alapja a túlórák elszámolásának.

Nálunk -s a legtöbb iskolában- ez úgy valósul meg, hogy a tanárok miden héten készítenek egy elszámoló lapot, melyre minden lényeges adatot -megtartott, elmaradt órák, helyettesítés, szakkör- felvisznek naponkénti bontásban (ld.: 1 sz. melléklet). Ezekből a heti összesítőkből áll össze az időszaki összesítés, amiben a tanár kötelező órai és az időszak tanítási napjai alapján meghatározzák tanáronként a tanítási időkeretet -ez mutatja meg, hány órát kellett a tanárnak az adott időszakban megtartani.

Nézzük meg részletesen, ennek nyilvántartásához milyen mezőket, milyen adatszerkezetet kell kialakítanunk:

A tanárok a heti elszámolólapon tartják nyilván a megtartott órarendi óráik számát, a helyettesítéseiket, vagyis azt, hogy hány órát helyettesített szakszerűen illetve nem szakszerűen az adott napon. Ennek a három mezőnek az összege lesz a ténylegesen megtartott tanítási órák száma. A tanítási órákon felül a csoportos, illetve az egyéni foglalkozások számát kell még nyilvántartani. Az órarendi órák és az órarenden kívüli egyéni foglalkozások összege számít majd be az elszámolható órákba.

Az időszaki elszámolásnál adják meg a tanárok, hogy mennyi a heti kötelező óraszámuk, valamint, hogy az adott időszak hivatalos tanítási napjait. Ez csökkenthető azokkal a napokkal, melyeken a tanár hivatalosan távol volt, ez adja a tényleges tanítási napok számát. A kötelező óraszám és a tényleges tanítási napok alapján számítható ki a tanítási időkeret: a heti kötelező óraszámot osztjuk 5-tel -megkapjuk a napi kötelező órák számát- és ezt szorozzuk a tényleges tanítási napok számával. Ennyi órát kellett volna megtartani a tanárnak az adott időszakban. Amennyiben volt olyan óra, ami nem a tanár távolléte miatt maradt el -például az

iskola hangversenyre ment, vagy a diákok kompetenciamérésen vettek részt- az levonható a tanítási időkeretből, s így kapjuk meg a tényleges keretet. Ezek után a heti adatok összesítéséből kiszámítjuk, hogy a tanár hány órarendi, illetve órarenden kívüli órát tartott, s ebből meghatározható -a tényleges kerettel összevetve-, hogy van-e többletórája (túlórája). A többletórákból levonva a helyettesítéseket, megkapjuk a tanár állandó túlóráit, bár erre az adatra csak adminisztratív okok miatt van szükség.

Az adatokat naponként fogjuk tárolni egy táblában. A napok táblában létrehozunk egy id mezőt a nap egyértelmű azonosítására, ez lesz az elsődleges kulcs. Ennek segítségével kapcsoljuk a napok mezőt a hetek táblához, a hetek táblát pedig az időszakok táblához. Ezen kívül egy tanárok táblára lesz még szükségünk, ahol nyilvántartjuk a tanárok nevét, jelszavát -hálózatos alkalmazásról lévén szó, azonosításra szükség van-, valamint a jogait, ami később meghatározhatja, hogy milyen adatokhoz fér hozzá a táblában. Egyelőre nem definiálunk különböző jogokkal rendelkező felhasználókat, hiszen a programok úgy készítjük el, hogy minden tanár csak a saját maga által létrehozott adatokhoz fér hozzá. A későbbiek során azonban -feltéve, hogy az újonnan bevezetett rendszer hosszútávon is fennmarad- szükség lehet olyan felhasználókra, akik mondjuk olvashatják a tanárok által bevitt adatokat, de módosíthatnak abban -pl.: a gazdaságvezető betekinthet a túlmunka elszámolásába. Esetleg az is elképzelhető, hogy a felügyeletet gyakorló alkalmazott -mondjuk az igazgató, vagy a z igazgatóhelyettes- nem csak betekinthet, hanem módosíthat is adatokat a tanárok túlóraelszámolásában. Jelenleg mindez papír alapon történik, de ha a rendszer beválik, és minden tanár áttér elektronikus elszámolásra, akkor felügyelet is sokat egyszerűbbé válhat, s akkor jó hasznát vehetjük, hogy előre gondoltunk a jogosultságok kezelésére.

A felhasználókat az időszakok táblához fogjuk kapcsolni. Ez azt jelenti, hogy a felhasználó először egy időszakot hoz létre, az időszakban heteket definiál, végül a heteken belül napokat.

Az adatok módosítását heti szinten tervezzük megoldani, mivel a gyakorlatban a tanárok hetenként, a hét utolsó napján viszik fel az adatokat. Ez annyit jelent, hogy

egyszerre egy hét adatait lehet módosítani, feldolgozni. A naponkénti adatbevitel és rögzítés nagyon körülményes lenne, az időszakonkénti pedig nagy mennyiségű adat mozgatását igényli akár egyetlen érték megváltozása miatt is.

A következő fejezetben megvizsgáljuk, milyen technológiákat használunk a a feladat megoldásánál, illetve rátérünk a konkrét megvalósításra is.

III. Alkalmazott technológiák

Az XML

XML, (eXtensible Markup Language), a W3C (World Wide Web Consortium) által létrehozott metanyelv. Alkalmas más nyelvek definiálására, illetve gyakran használják adatszerkezetek leírására is. Ezért nem meglepő, hogy az XML-t egyre szélesebb körben támogatják a legkülönfélébb szoftverek. A különböző Office változatok (MS-Office, OpenOffice.org, Star Office) mindegyike képes adatait XML formátumban menteni. A programozási nyelvek többsége szintén komoly támogatást nyújt XML dokumentumokkal történő munkához, egyre népszerűbb az adatbázis-kiszolgálók körében is (Pl.: Oracle, MS-SQL Server), és természetesen a web böngészők szinte mindegyike támogatja az XHTML-t, ami szintén XML.

Ha röviden szeretnénk összefoglalni, azt mondhatnánk, hogy az XML -és a hozzá kapcsolódó technológiák- lényege az, hogy elválasztja egymástól az adatot, az adatszerkezetet a programozási logikát és a megjelenést. Mindez áttekinthetőbbé teszi adatainkat, szabványos nyelvet használunk az adatok leírására, s ez a hordozhatóságot is elősegíti. Nagyon nagy előnye ennek a technológiának, hogy az adatok megjelenítését rugalmasan alakíthatjuk, miközben a a megírt programkódhoz nem kell hozzányúlni. Ez fordítva is működik, ha a programkódot változtatjuk, esetleg másik nyelvet használunk, a dokumentum és a megjelenés nem kell, hogy megváltozzon, vagyis dokumentumunk nyelvfüggetlen lesz. Természetesen az is megoldható, hogy egy XML dokumentumot több különböző formátumban jelenítsünk meg a felhasználói igényeknek megfelelően. Így például egy időben férhetünk hozzá adatainkhoz web illetve wap felületen, vagy tölthetjük le pdf formátumban ugyanazt a programkódot használva.

Természetesen nem csak előnyei vannak ennek a technológiának. Az egyik gyakran emlegetett „hibája”, hogy terjengős. Ez egyedül az XML dokumentum elkészítésekor jelent kicsit több munkát, ma már az átvitelnél, feldolgozásnál nincs különösebb jelentősége, bár mindkettőre kihatással van.

Sokkal komolyabb gondot jelent a technológia támogatottsága. Ez kicsit meglepő kijelentés, de nem túl régi technológiáról van szó. Igaz, az XML-t 1997-ben definiálták, ám nagyon sokáig csak lassan terjedt, minden szoftvercég ígérte az XML támogatást, ám mindig csak a következő verzióban. Azért az elmúlt 10 év természetesen elegendő volt az elterjedéséhez, azonban sok különféle támogatás látott napvilágot. Van, aki tömbként dolgozza fel az XML dokumentumot, van, aki egy fa gráfként kezeli, és létezik, aki mintaillesztést használ. A feldolgozás sokfélesége még nem jelentene problémát, de ezek támogatottsága a különböző programozási nyelvekben különböző fokon áll.

Egy másik „hátrányként" említhetjük, hogy az XML alapokon történő adatfeldolgozáshoz magasan képzett szakemberekre van szükség. A saját példánknál maradva, ahhoz, hogy XML alapokon tudjunk adatokat feldolgozni szükségünk lesz a később felsorolt technológiák mindegyikére: XML, DTD vagy XML Schema, Xpath, XSL, valamint az ezeket megvalósító programozási megoldásokra PHP-ben és MySQL-ben.

Elmondhatjuk, hogy ennek ellenére igazán nagy hátránya nincs ennek a technológiának, lassan tisztázódnak a technológia újszerűségéből adódó hátrányok, s az előnyök messze kárpótolnak bennünket egy kis többletmunkáért. Inkább azt kell meglátni, hogy nem minden esetben célszerű ezt a többletmunkát befektetni, el kell tudnunk dönteni, mikor érdemes és mikor nem érdemes használni. Itt jegyezzük meg, hogy már több protokoll is támogatja az XML kommunikációs célú használatát (pl.: SOAP, vagy Web-DAV)

Vizsgáljuk meg, hogyan néz ki egy XML dokumentum, milyen szabályok szerint épül fel, illetve hogyan tudjuk használni, feldolgozni az XML dokumentumokat.

Az XML-nek jóformán nincsenek szabályai. Három olyan feltétel van, amit egy dokumentumnak teljesíteni kell ahhoz, hogy megfeleljen az XML elvárásainak. Az XML-ben, mint a legtöbb leírónyelvben az adatokat úgynevezett tag-ek között adjuk meg, erre vonatkozóan az alábbiakat kell betartani:

- egy XML dokumentumnak pontosan egy gyökéreleme (tag-je) van

- minden tag-nek létezik egy, a tag-et lezáró párja
- a tag-ek szigorúan egymásba ágyazottak, tilos a tag-ek átlapolása

Amelyik dokumentum megfelel ezeknek a követelményeknek, azt „jól formált" dokumentumnak nevezzük.

A fenti szabályokon túl mi magunk is állíthatunk fel szabályokat saját dokumentumunk részére. Leírhatjuk, hogy milyen tag-ek fordulhatnak elő benne, hányszor ismétlődhet egy-egy tag, milyen attribútumok szerepelhetnek benne, és milyen típusú adatot tartalmazhatnak. Ha egy XML dokumentum megfelel ezeknek az általunk definiált szabályoknak, akkor érvényes (valid). Az érvényesség ellenőrzését érdemes elvégezni, bár nem kötelező. Biztonsági okok miatt a webes alkalmazásoknál különösen ajánlott.

XML Schema

Egy XML dokumentum nyelvtani szabályainak leírására két lehetőségünk is kínálkozik. Az egyik , ami a kezdetektől része az XML specifikációnak, a DTD (Document Type Definition). A DTD nem XML formátumot használ az adatok leírására, s ennek vannak hátrányai, de ettől függetlenül jól használható XML dokumentumok szabályainak leírására. Ugyan vannak problémák például a névterek és a hierarchia kezelésében, de még mindig sokan használják.

A másik lehetőség az XML Schema. A sémák használatával sokkal részletesebben szabályozhatjuk a dokumentumok tartalmára vonatkozó megkötéseinket. Az XMLSchema később került be az W3C ajánlásai közé, azonban teljesen az XML szintaktikájára épít, és mivel egy újabb ajánlás, már csak e miatt is javasolható ennek a használata. A feladat megoldása során mi is ezt a leírást fogjuk használni.

A sémák, mivel az XML szintaktikáját követik, feldolgozhatók XML dokumentumként is, így ellenőrizni lehet magának a sémának az érvényességét is -amit természetesen DTD segítségével definiáltak. Ez komoly segítség lehet a séma írásakor, hogy sémánk valóban megfelel a szintaktikai szabályoknak.

XSL, XSLT

A XSL (eXtensible Stylesheet Language), ugyan erősen kötődik az XML-hez, mégis egy különálló dolog. Nem kötelező használni, más módon is megoldható a transzformáció, illetve ha nem akarjuk az XML fájlt más formátumba alakítani, akkor nincs is szükségünk XSL-re. A nyelvet szintén a W3C definiálta leginkább azzal a céllal, hogy egyszerűvé tegye az XML dokumentumok formázását, és az XML – XML átalakítást. Számunkra fontosabb az XSLT, ami egy XSL transzformációt definiál, mindezt XML formátumban. Az XSLT segítségével tudunk átalakítani egy szöveges XML fájlt egy másik szöveges formátumba, ami már nem is kötelezően XML alapú. Így egy meglévő XML fájlból készíthetünk HTML, XHTML, PDF, RTF vagy akár TXT formátumú fájlt is.

XPath

Az Xpath (XML Path Language) feladata, hogy tudjunk navigálni XML dokumentumokban. Az Xpath minden -jól formált- XML dokumentumot fa gráfként kezel. Egy Xpath kifejezéssel ennek a fának egy részfáját jelölhetjük ki. Az XSLT-ben vesszük például ennek nagy hasznát, mivel ezzel egy-egy részfára vonatkozóan tudunk feldolgozási utasításokat adni. Az XSLT-n kívül természetesen még nagyon sok alkalmazás támogatja az Xpath használatát, így például a MySql gyengécske XML támogatásában kizárólag ezt használják az XML dokumentum értékeinek elérésére, de megtalálható a XPath támogatás a lentebb említett DOM osztályok metódusai között is.

PHP

A PHP sokak által kedvelt programozási nyelv, leginkább webes alkalmazásoknál használják szerver-oldali szkriptek írására. Ez egy kicsit ízlés dolga, hogy ki melyik nyelvet használja. A PHP-ben több módja is létezik az XML dokumentumok kezelésének. Ezek közül némelyik külön modul telepítését igényli, némelyik elavult -esetleg az újabb verziókban már nem is támogatott. Mi a középutat választjuk, nem a legújabb, alapból még nem támogatott módszert használjuk, hanem egy régtől

működő, az 5-ös verzióban is teljes támogatottságot élvező DOM (Document Object Model) függvények segítségével kezeljük majd az XML fájlokat. Megjegyezzük, hogy hasonlóan jó választás lehet a SAX (Simple API for XML) alapú XML támogatás használata is, ez talán még régebbtől változatlan, ezért a régen írt kódok is jól használhatók -ha szerencsénk volt és ezt a módszert használtuk, de ez egy kód írásának a pillanatában megmondani, hogy melyik módszer lesz leghosszabb távon támogatott, lehetetlen.

Az XML sokféle támogatását láthatjuk a PHP-ben -mindenki megtalálja a saját kedvére valót- és természetesen nem csak az XML, hanem a hozzá kapcsolódó technikák -mint például a DTD-k, sémák, XSLT- támogatása szintén megtalálható a nyelvben.

DOM

A DOM szintén a W3C által elfogadott szabvány, így támogatottsága széleskörű, és várhatóan hosszútávú a legtöbb nyelvben. A DOM nem csak XML dokumentumokat képes kezelni, arra tervezték, hogy egy dokumentum tartalmát és szerkezetét tudjuk kezelni. A DOM egy fa gráf segítségével modellezi a dokumentumokat, ugyanakkor objektumként kezeli, így léteznek a létrehozott dokumentumnak osztályai -pl.: document, element, node-, az osztályoknak tulajdonságai -pl.: doctype, vagy encoding-, metódusai -pl.: load() vagy setAttribute() -, s ezek segítségével tudjuk kezelni a dokumentumot, amit aztán XML, HTML, stb. formátumra alakíthatunk.

MySQL

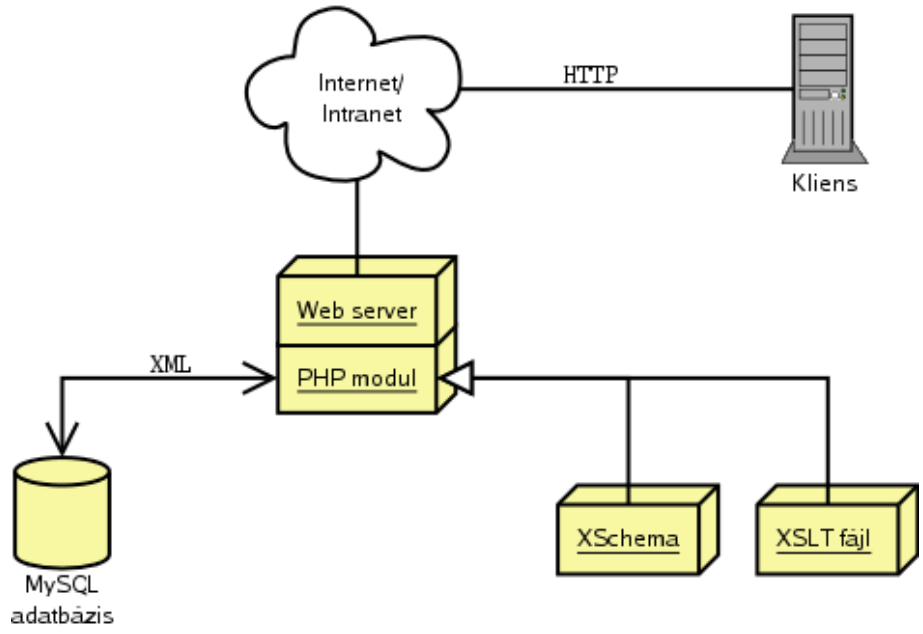
A MySql az egyik legnépszerűbb adatbázis-kiszolgáló a webes alkalmazások esetében. Egyszerű, könnyen használható, és -igaz, hogy léteznek komolyabb tudású és gyorsabb adatbázis-kiszolgálók- a MySql tudása pontosan elegendő a web alkalmazások legtöbbjénél. Érdemes megfigyelni azt is, hogy a MySql a legtöbb ingyenes tárhely-szolgáltatónál elérhető, és gyakran kizárólag a MySql-t használhatjuk ilyen esetben.

E miatt a MySql nem rossz döntés egyszerű web alapú alkalmazáshoz adatbázis-kiszolgálónak. Viszont, mindaz amit a PHP-nél elmondhatunk XML támogatás ügyében, a MySql esetében ezeket az állításokat nyugodtan negálhatjuk. A MySql egyáltalán nem támogatja az XML-t. Egyelőre. Mert a legújabb 5.1 verzióban -mely jelenleg béta állapotában érhető el- már megjelent két függvény, ami natív XML támogatást nyújt. Az viszont, hogy egy szoftver nem támogatja az XML-t, még nem jelenti azt, hogy nem is tudunk XML-t használni, hiszen az XML egy szöveges fájl. Annyi többletmunkát igényel, hogy magunknak kell gondoskodni az adatok eléréséről. Pont ebben kapunk segítséget az említett függvényekkel, s ha ez nem is nevezhető tökéletes támogatásnak, azért együtt lehet vele élni. Megjegyezzük itt is, hogy léteztek eddig is MySQL alatt XML alapú megoldások, ezeket valamilyen külső feldolgozó -mondjuk Perl szkript- segítségével valósították meg.

A feladat megoldása során a MySql fent említett béta verzióját használjuk, eddig a működés során hiba nem volt tapasztalható, remélhetőleg ez a verzió hamarosan végleges állapotban jelenik majd meg.

IV. Megvalósítás

A megvalósításhoz ismernünk kell a webes alkalmazások működési vázlatát. Ezt az alábbi ábrán szemléltetjük.



A séma létrehozása

Mivel a web-szerver és az adatbázis-kezelő közötti kommunikáció XML alapokon zajlik, a megvalósítás első lépése az adatszerkezet létrehozása kell legyen, vagyis megtervezzük, hogyan fog felépülni az az XML dokumentum, amit küldözgetünk az alkalmazások között.

Ennek alapját a feladat leírásánál már tárgyalt adatszerkezet adja. Az XML dokumentumunk gyökéreleme a <tulora> tag lesz. Ezen belül vagy időszakokat, vagy napokat, vagy felhasználókat küldünk az alkalmazások között. Ezt az XML tartalmát korlátozó sémában is definiáljuk:

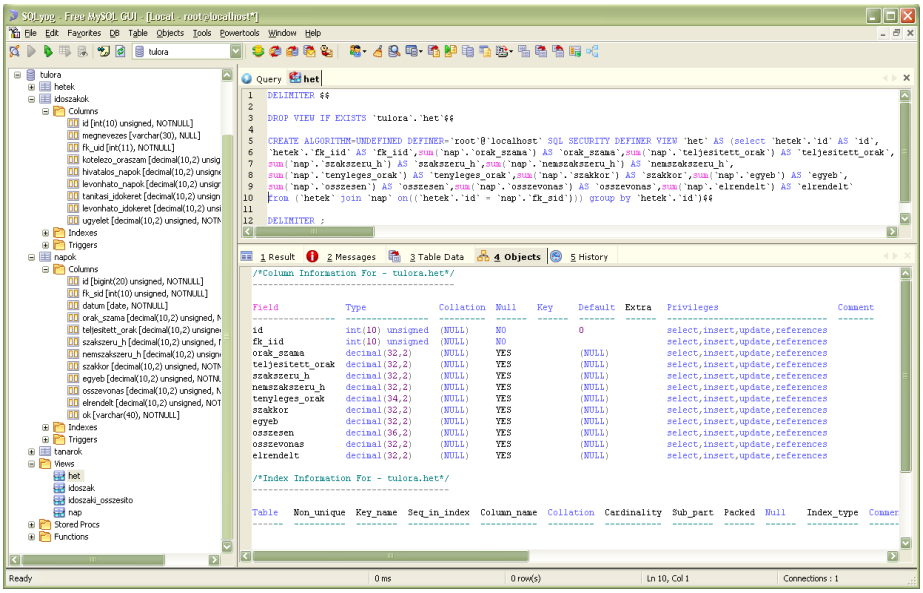
```
<?xml version="1.0" encoding="ISO-8859-2" standalone="yes"?>
```

```
  <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema"  targetNamespace
  = "http://www.berzeviczy.sulinet.hu/tulora"  xmlns:tulora =
  "http://www.berzeviczy.sulinet.hu/tulora" >

<xs:element name = "tulora" >
     <xs:annotation>
        <xs:documentation>A gyoker tag</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:all maxOccurs = '1'>
        <xs:element name = "tanarok" type = "tulora:tanarokType" minOccurs =
         "0" maxOccurs = "1" />
        <xs:element name = "idoszakok" type = "tulora:idoszakType" minOccurs
         = "0" maxOccurs = "1" />
        <xs:element name = "napok" type = "tulora:napokType" minOccurs = "0"
         maxOccurs = "1" />
      </xs:all>
     </xs:complexType>
</xs:element>

</xs:schema>
```

Hasonlóképpen hozzuk létre a hivatkozott tanarokType, idoszakType, és napokType típusokat. Rendelkezünk afelől is, hogy ezek tag-ek hányszor fordulhatnak elő az XML-ben. A sémát természetesen használni fogjuk az előállított XML érvényességének ellenőrzésére. Sajnos, erre csak a PHP szkriptben van lehetőség, mert a MySQL nem támogatja a validitás ellenőrzését sem sémák, sem DTD használatával. Azért kétségbe esni nem kell, az, hogy nem tudjuk ellenőrizni a MySql-ben az érvényességet, nem olyan komoly probléma, hiszen mielőtt elküldjük az adatokat, ellenőrizzük PHP-ben. Megjegyezzük hogy még, hogy a sémák használata a DTD-vel szemben nem csak azért biztonságosabb, mert jobban lehet szabályozni az adatok, a szerkezet tulajdonságait, hanem azért is, mert a sémák csak külső fájlként jelenhetnek meg, míg a DTD akár az XML-be ágyazva is. Ez utóbbi változat kerülendő, mert ilyen módon egy nemkívánatos fájlt is érvényessé lehet tenni.

Az adatbázis

A következő lépés a megvalósítás során az adatbázis létrehozása. Ha már definiáltuk az XML szerkezetét, akkor az alapján aránylag egyértelmű, hogy milyen táblákat kell létrehozni. A munkához a a Webyog Softworks SQLyog nevű szoftverét fogjuk használni, ami egy kellemes grafikus felületet (GUI) biztosít a MySql eléréséhez.



1. ábra: View létrehozása az SQLyog kezelőfelület segítségével

A táblák mellé készítünk „nézeteket” (view-kat), ami tulajdonképpen egy nevesített lekérdezés (1. ábra). Ez azért előnyös, mert elég sok számított mező található a feladatban, amit nem tárolunk, viszont a nézetek segítségével bármikor elérhetőek a számított adatok is. A view-ban, mint bármelyik lekérdezésben, rendezhetjük is az adatokat. Jelen esetben mindig dátum szerinti rendezettségre lesz szükségünk, ezt a napokat listázó view-ban definiáljuk. A továbbiakban az SQL lekérdezésekben is általában a létrehozott view-kra fogunk támaszkodni.

Miután a táblákat elkészítettük, elkezdjük feltölteni az adatbázist. Mivel az adatokat egy PHP szkript szolgáltatja XML formában, a feltöltés előtt ki kell nyerni az értékeket az XML sztringből. Ezt tárolt eljárásokkal fogjuk megvalósítani. A tárolt eljárások tulajdonképpen SQL parancsok, függvények, néhány vezérlési szerkezettel kiegészítve. Ezeket az eljárásokat hívjuk meg majd a PHP szkriptben.

Tárolt eljárásaink -legalábbis, melyek a PHP-vel kommunikálnak- általában két paramétert tartalmaznak, egy input XML sztringet, és egy output XML sztringet. Ennek a megoldásnak -mint magának az XML-nek is- az az előnye, hogy később nyugodtan változtathatunk az eljáráson, csupán arra kell figyelni, hogy az eredmény egy adott sémának megfeleljen.

Látni fogjuk, hogy a MySql XML támogatása még gyerekcipőben jár. Ezért a PHP által küldött adatok érvényességét például nem tudjuk ellenőrizni. Azonban, hogy mégse kapjunk érvénytelen adatokat, még az eljáráshívás előtt megtesszük ezt magában a PHP szkriptben. Az azonban mindenki számára világos, hogy szerencsésebb lenne a kapott XML érvényességét a MySQL-ben ellenőrizni.

De térjünk vissza az új elemek létrehozására. Létrehozhat a felhasználó napokat egy adott héten belül, heteket egy időszakban, illetve időszakokat. Ha napokat, illetve heteket akar létrehozni, akkor ismernünk kell annak a hétnek, illetve időszaknak az azonosítóját, ahová be akarja illeszteni. Ezeket az értékeket az eljárások ugyanolyan XML sztringben kapják meg, mint amiben később az adatokat is -mármint ha nem új elemet hozunk létre.

A MySql az ExtractValue(*IN xml text, IN xpath text*) függvényt használja az XML-ben történő navigáláshoz. A megadott Xpath kifejezésre illeszkedő részfa text() típusú csomópontjainak értékét kapjuk visszatérési értékként egy sztringben. Ez azt jelenti, hogy az ExtractValue(*xml, "//nap"*) és az ExtractValue(*xml, "//nap/text()"*) függvényhívások azonos eredményt adnak. Itt megint csak találkozunk a MySql hiányos XML támogatásával. Vegyük még e mellé, hogy a Xpath kifejezés nem tartalmazhat jó néhány függvényt, például name(), string(), last(); nem tartalmazhatja a következő tengelyeket: following, following-sibling, preceding, preceding-sibling; valamint nem tartalmazhat relatív meghatározást, csak / vagy // jellel kezdődhet.

Miután tudomásul vettük a MySql ezen hiányosságait, elkészítjük eljárásainkat, melyek *ujnap, ujhet, ujidoszak* névre hallgatnak. Ezeket egymásba ágyazzuk, és a PHP szkriptből minden esetben csak az *ujidoszak* eljárást fogjuk meghívni, bármit szeretnénk is létrehozni. Az átküldött XML sztring alapján tudjuk majd eldönteni, hogy szükséges-e új időszakot vagy új hetet készíteni, mégpedig olyan módon, hogy ha az időszak azonosítója, vagy a hét azonosítója nincs megadva -üres-, akkor létre kell hozni ezeket az új elemeket, egyébként pedig a meglévő időszakhoz hozunk létre új hetet, vagy a meglévő héthez új napot. Ez annyi megkötést jelent, hogy ha új időszakot hozunk létre, abban automatikusan létrejön egy új hét, s az új hét egy új

napot is fog tartalmazni. Ez ugye nem komoly hátrány -inkább előny. Viszont egyértelmű előnyt jelent, hogy csak az *ujidoszak* eljárásban kell XML dokumentumokkal bíbelődni.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `tulora`.`ujidoszak`$$

CREATE PROCEDURE `tulora`.`ujidoszak`( IN user_xml text, OUT idoszak_xml
    text )
BEGIN
   DECLARE het_id, iid, nap_id, uid, owner int;
   DECLARE nev, eredm text;
   SET uid = extractvalue( user_xml, "/tulora:tulora/tanarok/tanar/@id" );
   SET nev = extractvalue( user_xml,
       "/tulora:tulora/idoszakok/idoszak/megnevezes" );
   SET iid = extractvalue( user_xml,
       "/tulora:tulora/idoszakok/idoszak/@id" );
   SET het_id = extractvalue( user_xml,
       "/tulora:tulora/idoszakok/idoszak/hetek/het/@id" );
   IF iid = "" THEN
     insert into `idoszakok`
       (`id`,`megnevezes`,`fk_uid`,`kotelezo_oraszam`,`hivatalos_napok`,
       `levonhato_napok`,`tanitasi_idokeret`,`levonhato_idokeret`,`ugyelet`)
       values ( NULL,nev,uid,'0.00','0.00','0.00','0.00','0.00','0.00');
     set iid = LAST_INSERT_ID() ;
   END IF;
   select idoszakok.fk_uid from idoszakok where id = iid into owner;
   IF owner = uid THEN
      call ujhet ( iid, het_id, nap_id );
   END IF;
   SET eredm = "<tulora:tulora xmlns:tulora =
   'http://www.berzeviczy.sulinet.hu/tulora'>\n";
   set eredm = concat( eredm, "<idoszakok>\n<idoszak id = '", iid, "' fk_uid
       = '", uid, "'>\n</idoszak>\n</idoszakok>\n" );
   set eredm = concat( eredm, "<tanarok>\n<tanar id = '", uid,
       "'>\n</tanar>\n</tanarok>\n" );
   SET eredm = concat( eredm, "\n</tulora:tulora>\n\n" );
   call idoszak_lista_xml( eredm, idoszak_xml );
END$$

DELIMITER ;
```

Csak összehasonlítás végett álljon itt az *ujhet* eljárás kódja is:

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `tulora`.`ujhet`$$

CREATE PROCEDURE `tulora`.`ujhet`( IN iid int, INOUT het_id int, OUT nap_id
     int )
BEGIN
  IF het_id = "" THEN
      insert into `hetek` ( `fk_iid`,`megnevezes` ) values ( iid, NULL );
      set het_id = LAST_INSERT_ID() ;
  END IF;
  call ujnap ( het_id, nap_id );
END$$

DELIMITER ;
```

A nyilvánvaló különbség az XML kezelés megvalósítása, pedig a kimeneti XML sztringet frappánsan egy már létező, a felhasználó teljes időszakát listázó

idoszak_lista_xml eljárás meghívásával oldottuk meg -igaz egy bemeneti XML-t így is létre kellett hozni, ami azonosítja a felhasználót és az időszakot.

A fenti kódból jól látszik hogyan tudunk XML sztringekből adatokat elérni, illetve hogyan tudunk -fapados módszerrel- XML sztringeket előállítani. Ami még egy kis magyarázatra szorul, az *ujhet* eljárás meghívása. Ezt az eljárást mindenképpen meg kellene hívni, akár új időszakot hoztunk létre, akár már meglévőben szeretnénk új hetet vagy napot létrehozni. Azt viszont nem árt ellenőrizni, hogy az az időszak, amelyikben a felhasználó új hetet vagy napot kíván létrehozni, az a saját időszaka-e. Ne felejtsük el, hogy egyedül az időszakok vannak felhasználókhoz rendelve. Üres időszakot minden felhasználó létrehozhat, de csak a saját azonosítójával. A majdan ellenőrzést végző vagy a felügyeletet gyakorló felhasználók sem hozhatnak létre mások számára új időszakot, bár az időszakok adatait -ellenőrzési jogukból adódóan- módosíthatják.

Adatainkat az adatbázisból XML formátumban kell továbbküldeni, ezért a lekérdezés eredményét át kell alakítanunk ilyen formátumra. Mivel a MySQL nem nyújt támogatást XML dokumentumok létrehozásához, magunknak kell az XML tag-eket elkészíteni. Ez annyiban jelent kis nehézséget, hogy nem kérdezhetjük le egyszerre az összes, adott feltételnek megfelelő napot, hanem egyenként, a kapott adatokat tag-ek közé helyezve kell megoldani a problémát.

Ehhez definiálunk egy CURSOR-t a kívánt lekérdezéssel, majd egy ciklusban a FETCH parancs segítségével a CURSOR által definiált lekérdezés eredményének egy-egy sorát mezőnként bepakoljuk a megfelelő változókba. Gondot jelent még, hogy nem tudjuk előre, hány soros eredményhalmazt kapunk a CURSOR által definiált lekérdezésből, ezért nem lehet megmondani, hogy mikor álljon le a ciklus. Ezt úgy oldjuk meg, hogy definiálunk egy hibakezelőt. Ha a FETCH már nem tud több sort elérni a hivatkozott CURSOR-ból, akkor SQLSTATE 02000-es számú hibát generál (Fetch No Data). Ehhez a hibához kell definiálni egy hibakezelőt, mégpedig olyat, ami nem állítja meg a program futását (continue handler). Ez a hibakezelő egyetlen változó értékét állítja át igazra, s ez jelzi, hogy a ciklust be kell fejezni. Kicsit bonyolult megoldás.

```
CREATE PROCEDURE `napok_lista_xml`( IN uid int, IN sid int, IN fejlec
boolean, OUT eredm text )
                                        .
                                        .
                                        .
  DECLARE lista CURSOR for select nap.id, nap.fk_sid, nap.datum,
    nap.orak_szama, nap.teljesitett_orak, nap.szakszeru_h,
    nap.nemszakszeru_h, nap.szakkor, nap.egyeb, nap.osszevonas,
    nap.elrendelt, nap.ok, nap.tenyleges_orak, nap.osszesen from nap inner
    join het on fk_sid = `het`.`id` inner join idoszak on `het`.`fk_iid` =
   `idoszak`.`id` where idoszak.fk_uid = uid and `nap`.`fk_sid` = sid;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET vege = 1;
  open lista;
  SET eredm = "";
                                        .
                                        .
                                        .
  REPEAT
    FETCH lista INTO azon_, sid_, datum_, orak_szama_, teljesitett_orak_,
    szakszeru_h_, nemszakszeru_h_, szakkor_, egyeb_, osszevonas_,
    elrendelt_, ok_, tenyleges_orak_, osszesen_;
    IF not vege THEN
        set eredm = concat( eredm, "\n<nap id = '", azon_, "' fk_sid = '",
        sid_, "'>\n" );
        set eredm = concat( eredm, "<datum>", datum_,...
                                        .
                                        .
                                        .
    END IF;
    UNTIL vege
  END REPEAT;
  set eredm = concat( eredm, " \n</napok>\n\n" );
```

A fenti eljárás részlet egy adott felhasználó kiválasztott hetéhez tartozó napokat listázza. Az eljárás bemeneti paraméterei: a felhasználó azonosítója (uid), a lekérdezett hét azonosítója (sid), egy logikai változó, hogy akarunk-e XML fejlécet, és végül egy kimenti érték, ami maga az előállított XML string. Ezt a sztringet kapja meg a hívó *hetek_lista_xml* eljárás, az előállítja a hetek XML kódját, végül az *idoszak_lista_xml,* ami annyiban különbözik az előző kettőtől, hogy amint korábban is tettük az egymásba ágyazható eljárások esetén csak a külső eljárást készítjük fel a PHP-vel való XML alapú kommunikációra. Ez azt jelenti, hogy az *idoszak_lista_xml* eljárásnak két argumentuma lesz, mindkettő XML sztring, és ebben az eljárásban történik a beérkező XML adatok kezelése, valamint a kimenő XML sztring előállítása.

Egy pár szó a biztonságról. Az eljárás során nem ellenőrizzük, hogy a felhasználónak van-e joga az adott napot lekérdezni. A PHP szkriptből ugyanis úgy kerül majd meghívásra ez az eljárás, hogy előtte a felhasználó jelszavát ellenőrizzük. A PHP szkripten belül belül pedig már nem változhat meg a felhasználó id-je külső támadás

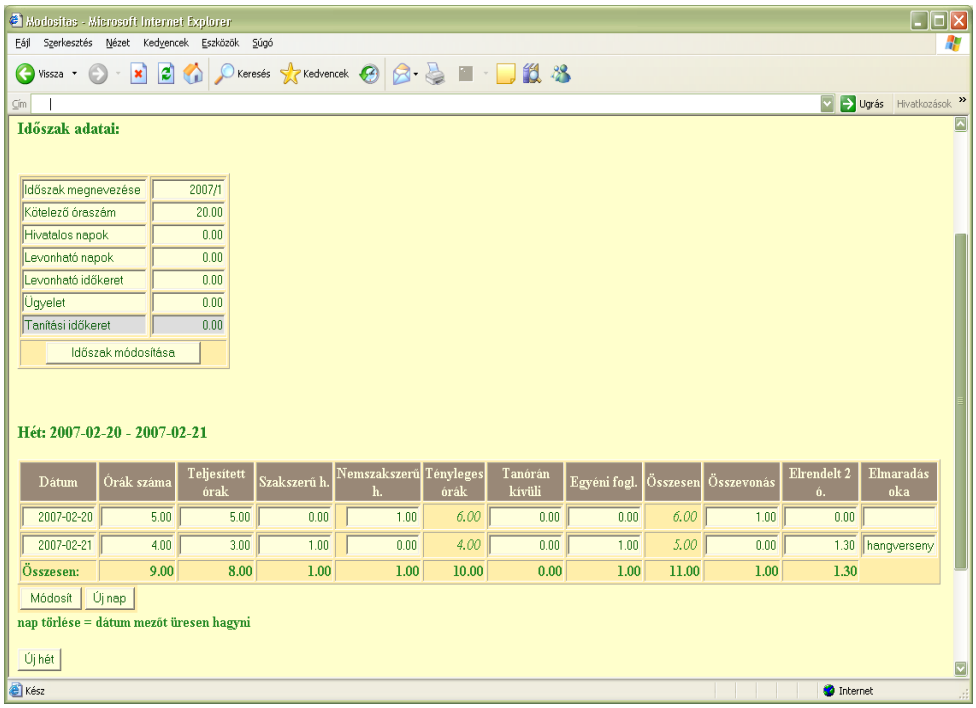
esetén sem. Lekérdezni pedig minden felhasználó lekérdezheti a saját időszakához tartozó napokat. Igaz, hogy az eljárás a felhasználó azonosítója mellett a hét azonosítóját is kéri, de a beállított feltételek miatt, ha megadott adatok inkonzisztensek, akkor az eredmény üreshalmaz lesz, vagyis más felhasználóhoz tartozó heteket nem lehet az eljárással kilistázni.

Ezek után teljesen hasonló módon megírhatjuk azokat az eljárásokat, melyek a tanárokat, az időszakokat, illetve a heteket kérdezik le. Ezek az eljárások egymásra épülnek, hiszen az időszakokban a heteket, a hetekben a napokat már a fenti eljárás meghívásával illesztjük be, ez esetben viszont a napokhoz nem kérünk XML fejlécet.

Meg kell írnunk azokat az eljárásokat, melyek az adatok módosítását illetve törlését végzik. Ezt a két műveletet egy eljárásban valósítjuk meg. Itt már szükség lesz annak ellenőrzésére, hogy a felhasználó jogosult-e módosítani a megadott napot. Ezt olyan módon tesszük, hogy lekérdezzük, a felhasználó jogait, valamint azt, hogy az adott nap melyik felhasználóhoz tartozik. Ha felhasználó jogosultsága mindenre kiterjed (supervisor jog), vagy a lekérdezett és a kapott user id megegyezik, akkor módosítunk. Ezt ismételjük, amíg el nem fogynak a hét napjai. Lássuk, hogyan működik ez XML adatok esetén. Az alábbi részlet a hét i-ik napja esetén végzi az ellenőrzést, ahol az i-ik nap adatainak kinyerése lehet érdekes egy XML sztringből:

```
select jogok from tanarok where id = uid into rights;
set xpath = concat ("/tulora:tulora/napok/nap[position() = '", i, "']");
select idoszakok.fk_uid from napok inner join hetek on fk_sid =
  `hetek`.`id` inner join idoszakok on `hetek`.`fk_iid` = `idoszakok`.`id`
  where napok.id = extractvalue(xml,concat( xpath,  "/@id" ) ) into user_id;
set pass = ( ( uid = user_id ) ) or ( rights = "supervisor" );
```

Mivel az adatok felvitele általában hetenként egyszer, a hét utolsó napján történik, ezért a felhasználó heteket tud majd módosítani, egyszerre az egész időszak összes hetét nem, noha ez az eljárás akár egy komplett időszak, vagy egy felhasználóhoz tartozó összes időszak módosítását is el tudná végezni (2. ábra). Csupán azért korlátozzuk a módosításokat hetekre, hogy egyetlen adat megváltoztatása miatt ne kelljen egy egész időszak adatmennyiségét XML-ben átküldeni, valamint a felhasználói gyakorlat is ezt követi.



2. ábra: A hetenkénti módosítás megvalósítása

A módosítás természetesen nem csak a napokra vonatkozhat, hasonlóképpen készítjük el a felhasználók, és az időszak adatainak módosítását végző eljárásokat is.

A felvitt adatok törlése a felhasználó részéről igen egyszerűen működik majd: csak napokat törölhet. Ha egy hét összes napját kitörölte, akkor törlődik a hét is, ha pedig egy időszak minden hetét -annak minden napját- kitörölte, törlődik az időszak is. Ez annyiból hátrányos megoldás, hogy az időszak törlése kicsit több egérkattintást igényel, a gyakorlatban viszont igen ritkán -mondhatni soha nem- kell egy teljes, kitöltött időszakot törölni. Az előnye viszont az, hogy nem lehet egyetlen egérkattintással több hónap adatát kitörölni, valamint a megvalósítása is egyszerűbb. Visszatérve a teljes időszak törlésére, furcsán hangzik, hogy soha nem kell törölni. Igazán egyetlen eset van, mikor új tanév kezdődik. Ekkor viszont mindenki törli az összes időszakot, s ezt egyszerűbb a felügyeletet ellátó tanárra bízni, aki az adminisztrációs felületen egyszerűen kezdhet majd új évet -archiválja és „törli" mindenkinek minden időszakát.

Tovább egyszerűsíthetjük helyzetünket, ha egy nap törlését az üresre állított dátum mező jelenti. Amikor a felhasználó új napot hoz létre, a dátum mező automatikusan az aktuális dátum értékét veszi fel, tehát az üres dátum mező mindenképpen szándékosságot sugall. De hogy tényleg kizárjuk a véletlen törlést -noha egyetlen nap véletlen törlése még nem katasztrófa- egy Javascript programmal ellenőrizzük az üres dátum mezőt, és csak felhasználói megerősítés után küldjük tovább a PHP szkriptnek. A MySQL eljárásban így már csak azt kell ellenőrizni, hogy a törölni kívánt nap egy hét utolsó napja-e mert ez esetben a hetet is törölni kell.

```
IF extractvalue( xml,concat( xpath,  "/datum" ) ) = "" THEN
  SET napid = extractvalue( xml,concat( xpath,  "/@id" ) );
  SET hetid = extractvalue( xml,concat( xpath,  "/@fk_sid" ) );
  select idoszakok.id from napok inner join hetek on fk_sid =
    `hetek`.`id` inner join idoszakok on `hetek`.`fk_iid` =
    `idoszakok`.`id` where napok.id = napid into idoszakid;
  delete from `napok` where `id` = napid;
  select count(  napok.id ) from napok join hetek on napok.fk_sid =
    hetek.id where hetek.id = hetid into nap_db;
  IF nap_db = 0 THEN
     delete from hetek where id = hetid;
     select count( hetek.id) from hetek join idoszakok on hetek.fk_iid =
       idoszakok.id where idoszakok.id = idoszakid into het_db;
     IF het_db = 0 THEN
       delete from idoszakok where id = idoszakid;
     END IF;
  END IF;
ELSE
```

Itt jegyezzük meg, hogy a törlés, miként a módosítás is egy heti ciklusokban történik, vagyis a felhasználó, ha egy teljes hetet kíván törölni, akkor -ugyan naponként kell kitörölnie a dátumot-, de az adatok rögzítését egyszer kell megtennie.

Gondoskodni kell még egy egyszerű hibakezelésről. Mi történjen, ha felhasználónak nincs joga módosítani az adott napot -bár ez a helyzet egyelőre nem állhat elő, ezért nem bonyolítjuk túl a hibakezelést. Ilyen esetben a nap dátumát visszaküldjük a hívó szkriptnek, és a feldolgozás leáll. Alapvetően ugyanis csak olyan eset lehetséges, hogy az eljárást hívó szkript egy meglévő nap adatait írta ki a felhasználó képernyőjére, ez a nap a felhasználó uid-je alapján lett lekérdezve, tehát módosítható is, mivel a nap id-je a szkripten belül nem változhat meg. Egy másik eset lehet majd, ha felhasználó más felhasználó adatait listázta, mert rendelkezik ilyen jogokkal. Ez egyelőre nincs megvalósítva, de kicsit előregondolva az ilyen eseteknél előfordulhat jogosulatlan kérés. Minden más eset nem várt esemény, így egyszerűen leállítjuk a

feldolgozást, ha ilyennel találkozunk. Részletezni nem érdemes, egy egyszerű IF-es szerkezet.

Végezetül az új felhasználók felvételét kell még megvalósítanunk. Nem tűnik bonyolult dolognak, de végig kell gondolni néhány dolgot. Például azt, hogy a felhasználó által megadott jelszót a password() függvény segítségével kódolva érdemes tárolni. A felhasználó jelszavát 20 karakterben limitáltuk, de könnyen pórul járhatunk, ha mezőméretnek is ezt a limitet választjuk, hiszen a password() által előállított hash több mint 40 jegyű.

Érdemes ellenőrizni az eljárás során, hogy létezik-e ugyanolyan néven már felhasználó. A név ugyan nem elsődleges kulcs, a felhasználók azonban ez alapján azonosítják magukat, s nem szerencsés, ha a listában két Gipsz Jakab van. Az persze ritka, hogy ugyanolyan nevű felhasználóból több legyen egy tanári karban, azonban a felhasználók regisztrációjakor néhányszor a lassú böngésző frissítés gombjára kattintva hamar gyárthatunk azonos nevű felhasználókat. Érdemes hát ezeket is kizárni.

Végül az SQL szerver által küldött üzenetet teljesen más fájl fogja megkapni, mint a korábbi esetekben, hiszen a felhasználót most hozzuk létre, nyilván nem ugyanabból a szkriptből, amiben a bejelentkezett felhasználók az adataikat kezelik. Ezért érdemes megfontolni azt is, hogy más jellegű XML-t küldjünk át a fájlnak, amivel semmi más dolga nincs, minthogy kiírja a képernyőre -tehát egyszerűsödhet az XSL transzformáció.

Az új felhasználót létrehozó *newuser* eljárást nem részletezzük, az előzőekben megismert módon, a fentiek figyelembevételével készült el.

A PHP szkriptek

Miután az adatbázist létrehoztuk, továbbléphetünk a feldolgozás következő szintjére. A PHP modul a felhasználó és az SQL szerver között helyezkedik el. Ezen a szinten történik az adatok értelmezése, átalakítása, kiírása vagy éppen továbbküldése az SQL szerver részére. Adatainkat XML formátumban kapjuk meg a

MySQL szervertől, ezt kell a felhasználó számára érthető formában -jelenesetben HTML-ben- megjeleníteni. Ehhez szükségünk lesz XML-t kezelő függvényekre. Ebből több gyűjtemény is létezik a PHP-ben, és ahogyan említettük, mi a DOM függvényeket fogjuk erre a célra használni. Vizsgáljuk meg egy példán, hogyan tudjuk mondjuk egy időszak adatait lekérni az SQL szervertől, és azt megjeleníteni a felhasználó számára.

Mivel az SQL szerverrel közölnünk kell, hogy melyik felhasználó melyik időszakát szeretnénk lekérdezni, ezért az első lépés egy XML sztring előállítása, mely tartalmazza a tanár és az időszak azonosítóját:

```
$user_xml = new DomDocument( "1.0" );
$root = $user_xml->createElementNS(
  "http://www.berzeviczy.sulinet.hu/tulora", "tulora:tulora" );
$user_xml->appendChild( $root );
$tanarok = $user_xml->createElement( "tanarok" );
$root->appendChild( $tanarok );
$tanar = $user_xml->createElement( "tanar" );
$tanarok->appendChild( $tanar );
$tanar->setAttribute( "id", $id );
$idoszakok = $user_xml->createElement( "idoszakok" );
$root->appendChild( $idoszakok );
$idoszak = $user_xml->createElement( "idoszak" );
$idoszakok->appendChild( $idoszak );
$idoszak->setAttribute( "id", "" );
$idoszak->setAttribute( "fk_uid", $id );
```

A kód szinte önmagáért beszél, nem is kell magyarázni. Miután a $user_xml változóban létrejött a kívánt XML sztring, elküldhetjük az SQL szervernek, meghívva a kívánt eljárást. Viszont minden adatküldés előtt -ahogyan adatfogadás után is- erősen javasolt ellenőrizni az XML sztring érvényességét nem kevés bosszúságtól kímélve meg magunkat. Az érvényesség ellenőrzése nem csak azért fontos, hogy a kívülről érkező nemkívánatos támadásokat védjük, hanem a felhasználó által megadott adatok helyességét, meglétét, esetleg hiányát is tudjuk ellenőrizni. Egy érvénytelen XML sztringet nem érdemes elküldeni az SQL szervernek, mert valószínűleg hibára futunk. A dokumentum érvényességét a DOM objektum schemaValidate() metódusával végezzük.

Miután elvégeztük az ellenőrzést, elindíthatjuk a lekérdezést, ami egy eljáráshívás lesz. Az eljárás egy globális MySQL változóba helyezi a visszatérési XML sztringet, nekünk ezt a változót kell átvennünk feldolgozásra a PHP szkriptben:

```
if ( $user_xml->schemaValidate( $schemafile ) )
```

```
{
    mysql_query( "call idoszak_lista_xml( '". $user_xml->saveXML() ."',
      @b );" ) or die( 'sikertelen query' . mysql_error() );
    $eredmeny=mysql_query( "select @b;" ) or die( 'sikertelen query' .
       mysql_error() );
    $xml = mysql_fetch_row( $eredmeny );
    $inputdom = new DomDocument( "1.0", "iso-8859-2" );
    $inputdom->loadXML( $xml[0] );
    if ($inputdom->schemaValidate($schemafile))
    {
          $xslfile = "./process_modify.xsl";
          $xsl = new DomDocument( "1.0", "iso-8859-2" );
          $xsl->loadXML ( iconv ( "iso-8859-2", "utf-8", file_get_contents
            ( $xslfile ) ));
          $proc = new XsltProcessor();
          $proc->importStylesheet( $xsl  );
          if ($idoszakaz != "" ) { $proc->setParameter( "",
            "valasztott_idoszak", $idoszakaz ); }
          $outputdom = $proc->transformToDoc( $inputdom );
          echo $outputdom->saveHTML();
    } //end if7
}
```

Jól látható a fenti példában, hogy a visszakapott XML sztringet egy új DOM dokumentumba töltjük. Miután itt is megtörténik az érvényesség ellenőrzése, szintén a DOM segítségével létrehozunk egy $xsl dokumentumot, melybe a lemezen található xslt fájlt olvassuk be. A beolvasás itt kicsit furcsán történik, mivel nem a fájlt olvassuk be, hanem azt sztringgé alakítva át kell konvertálni UTF-8 kódolásúvá, mivel a DOM csak ezt a kódolást hajlandó elfogadni.

Most tehát van egy $inputdom objektumunk, amiben az XML fájl tartalma van, és egy $xsl objektumunk, amiben pedig az átalakításhoz szükséges xslt utasítások. Szükségünk lesz még egy XSLT feldolgozóra, amit a $proc változóban hozunk létre. Ebbe olvassuk be az importstylesheet metódus segítségével az $xsl objektumban lévő utasításokat. És kezdődhet is az átalakítás. Azaz, csak kezdődhetne, ha minden adat rendelkezésre állna. Azonban az SQL-ben meghívott idoszak_lista_xml függvény a felhasználó összes időszakát listázza. Nekünk viszont csak egy, a legutóbb listázott időszak kell -már ha van ilyen. Mivel a fenti részlet listázza a módosítás utáni állapotot is, nem nézne ki jól, hogy a módosítás után hirtelen egy másik időszakban találnánk magunkat -vagyis minden módosítás után ki kellene választani azt az időszakot, amit módosítottunk. Az összes időszak listázására pedig azért van szükség, mert az XSLT átalakításkor szeretnénk kiíratni egy legördülő listát, amiben felsoroljuk a felhasználó által létrehozott összes időszakot. Ebből tudja majd kiválasztani, hogy melyik időszakra kíváncsi.

Tárolnunk kell tehát az utoljára látott időszak azonosítóját -már ha van ilyen, mert rögtön bejelentkezés után például nincs. Ezt az $idoszakaz változó tartalmazza. Ha ez a változó üres, akkor az XSLT feldolgozó automatikusan a legfrissebb dátummal rendelkező időszakot listázza majd. Ha nem üres, akkor pedig át kell adni ennek a változónak a tartalmát a feldolgozó számára. Nos, ez egy érdekes művelet. Az érdekessége abban rejlik, hogy az XSLT fájlban léteznie kell egy ugyanilyen nevű globális paraméternek, ez kapja meg az átadott értéket.

Miután mindent beállítottunk, maga az átalakítás a $proc objektum transformToDoc metódusával történik, ahol megadjuk az átalakítani kívánt DOM objektumot. Így kapjuk meg a $outputDom változóban az előállított HTML formátumú objektumot, amit az echo $outputDom->saveHTML() utasítással kiküldünk a felhasználó képernyőjére.

Láthatjuk az XML előnyeit, a PHP szkriptben szinte alig van dolgunk a lekérdezésből kapott adatokkal -tulajdonképpen nem is kellett egyetlen adatot sem kiolvasnunk az XML sztringből, egyszerűen csak továbbküldtük az XSL feldolgozónak. Az egész hókuszpókusz csak amiatt van, mert az XSL feldolgozó nem két sztringet, hanem két DOM objektumot vár.

Most, hogy láttuk, hogyan történik az XML -> HTML átalakítás -tehát ki tudjuk küldeni a felhasználó számára elfogadható formában a kapott adatokat-, gondoljuk végig, mit kell még megvalósítanunk a PHP-ben. Dolgunk lesz bőven, hiszen adatainkat módosítani kell tudni, mind a napok, mind az időszakok esetén, kell tudni új napot, új hetet, új időszakot létrehozni, valamint munkánk végeztével szükség lesz kijelentkezésre is.

Mivel a HTML kódok nem jelennek meg a PHP dokumentumban -hiszen azt az XSL fájl tartalmazza, nyugodtan megoldhatjuk mindezek kezelését egyetlen PHP fájl segítségével -pontosabban azokat a részleteket tesszük csak külön fájlba, amit máshol is felhasználunk. Így a készülő programunk magja egy switch szerkezet lesz. A felhasználó különböző nyomógombok segítségével tud majd módosítani, kijelentkezni, új elemeket létrehozni. Az űrlapok submit gombjai mind-mind

ugyanannak a PHP fájlnak küldik el az adatokat, amelyikben maga az űrlap is található, csupán a gomb felirata alapján választva történik majd eltérő végrehajtás. Így nem fogunk elveszni az egymásra hivatkozó PHP fájlok rengetegében, ugyanakkor az XSLT-nek köszönhetően a kódunk is tiszta marad.

Kezdjük a napok módosításával. Miután listáztuk a felhasználónak az adatokat, ha módosít rajta, azt át kell küldenünk az SQL szervernek. Most ugyanazt kell tennünk, mint amit az előbb tettünk. Készítünk a felhasználó által elküldött -egy form-ból átvett- adatokból egy XML sztringet, ennek felhasználásával meghívjuk az eljárást az SQL szerveren, ami elvégzi a módosítást, lekezeljük az esetlegesen fellépő hibákat, majd a fenti program-részlet segítségével kiíratjuk a módosított állapotot.

Ha nem akarunk sokat kínlódni, érdemes a form elemeinek olyan nevet adni, mint amit az XML sztringben kell kapniuk. Ekkor ugyanis egy foreach ciklus segítségével, ami a form által küldött változókat tartalmazó $_GET tömb elemein fut végig, nemcsak a változó értékét, hanem a változó nevét is könnyen kezelhetjük. A form elemeinek a neve az XSL átalakításkor kerül meghatározásra, vagyis könnyen megoldható, hogy kívánt nevet kapják. Még ha változtatni kellene is az XML szerkezetén, a fenti megoldást még ez sem feltétlenül érinti, ami nagyon előnyös. Hiszen az SQL szerver által küldött XML-ben szereplő tag-eket használjuk a form elemek nevénél, a form elemeinek a neveit pedig az újabb XML sztring előállításánál.

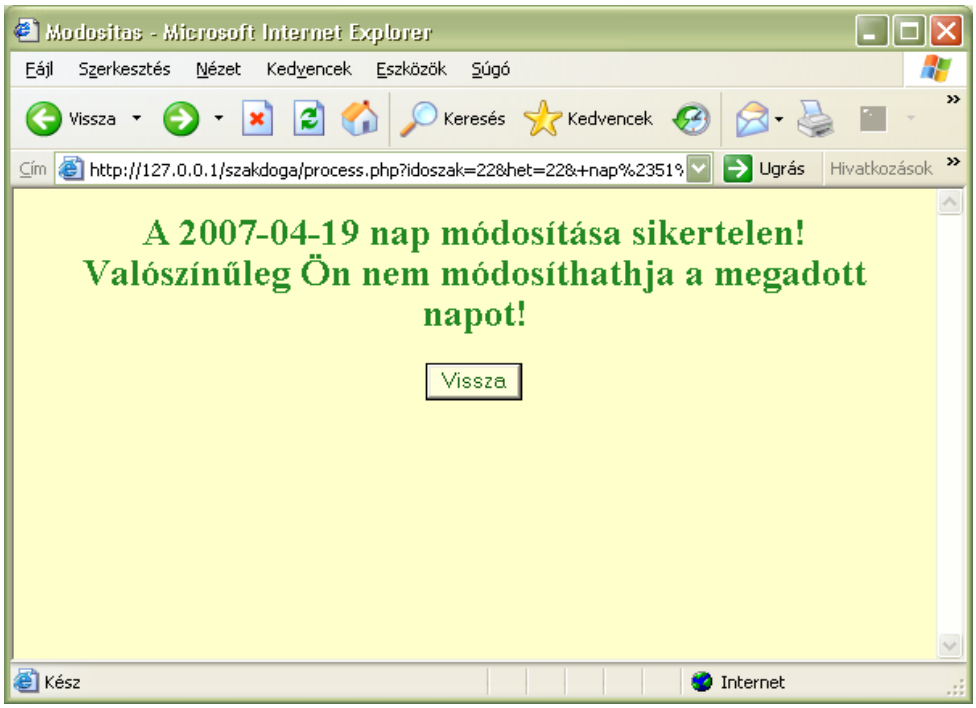
Rögtön adódik is egy probléma. Az XML-ben, például a napok az alábbi szerkezetet követik:

```
…
<napok>
<nap id = „1” fk_sid = „1”>
<datum>…</datum>
<orak_szama>…</orak_szama>
…
</nap>

<nap id = „2” fk_sid = „1”>
<datum>…</datum>
<orak_szama>…</orak_szama>
…
</nap>
…
</napok>
…
```

Ez azt jelenti, hogy minden napnak ugyanolyan nevű tag-jei vannak, ami a form-ok esetében így nem használható. Természetesen az XSL átalakításnál majd kitérünk ennek megoldására, egyelőre annyit, hogy a kézenfekvő nap#nap_id#tag-neve megjelölést fogjuk használni, pl.: nap#2#datum. Ebből az azonosítóból kell majd kinyerni a nap azonosítóját, valamint a tag nevét. A nap# prefixum feleslegesnek látszik, s ebben a példában az is, de az XSLT a szülő tag nevét teszi majd ide, ami pedig szintén szükséges.

A módosítás során keletkezhetnek hibák. Ezek a hibák, mivel az elküldött XML fájl érvényes, igazából nem adódhatnak az adatok helytelen voltából, -ezért a rendszerhibáktól eltekintve, aminek a kezelése nem feladata e programnak- csak jogosultságbeli, esetleg működés közben fellépő szemantikai hibával állhatunk szemben. A kettő között nem teszünk különbséget, a felhasználó felé jogosultsági hibát fogunk jelezni, mivel másfajta hibát úgysem tud orvosolni -és a programozó sem tudja előre kezelni a jövőben fellépő szemantikai hibákat.

A hibáról természetesen egy XML sztring segítségével értesülünk. Ha hiba lépett fel az SQL eljárás során, akkor annak a napnak az adatait kapjuk vissza, ahol a feldolgozás leállt, míg hibátlan futás esetén ez a sztring üres, így könnyen eldönthető, hogy mit kell tenni. A hiba kiírása egy függvény segítségével történik, mivel több helyen is ugyanígy, egy XML sztring tartalmaként kapjuk meg a hibát okozó adatokat. Ez a függvény pedig semmi mást nem tesz, mint a fentebb megismert módon egy hiba.xsl fájl segítségével átalakítja a kapott XML-t HTML-lé, kiírva a hibaüzenetet, és a hibát okozó adatokat (3. ábra).



3. ábra: Sikertelen módosítás eredménye

Az időszak adatait külön eljárásban módosítjuk. Most is, de máskor is: új napok, új hetek, új időszak, a képlet ugyanaz. Elő kell állítani a megfelelő XML sztringet, átadni az SQL szerver egy tárolt eljárásának, majd a válaszként kapott XML-t egy XSL átalakítás után megjeleníteni a felhasználó képernyőjén. Megvalósítani azért egy kicsit bonyolultabb, hiszen minden esetben más az XML felépítése, más eljárást kell meghívni, de részletezni nem érdemes.

Az egyetlen dolog, ami említése érdemel, az új felhasználók regisztrációja. Ezt már egy teljesen másik PHP fájlban hajtjuk végre, hiszen az előbbi programunk feltételezte a sikeres bejelentkezést. Az új fájlhoz egy másik XSL fájlt kell létrehozni, mert a megjelenítés is teljesen más, nincs szükség az időszakok adataira, így az SQL szerver is más XML-t küld majd. Ezektől eltekintve viszont ebben sincs semmi újdonság, a feldolgozás a szokásos forgatókönyv szerint zajlik.

XSL átalakítás

Az utolsó fázis az adatok megjelenítése. Itt nagy előnye lesz annak, hogy adatainkat XML formátumban kapjuk meg az SQL szervertől. Ahogyan láttuk, a PHP szkriptben az adatok megjelenítésével nem kellett foglalkozni, egyszerűen definiáltunk egy XSLT feldolgozót, s az elvégezte az átalakítást egy előre megadott XSL fájl alapján. Most ennek az XSL fájlnak az elkészítése lesz a feladatunk.

Az XSL felfogható akár egy programozási nyelvként is, sok olyan eszköz, melyeket programnyelvekben használnak itt is a rendelkezésünkre áll: változók, vezérlési szerkezetek, valamint a XPath által definiált függvények. Ezekről érdemes ejteni néhány mondatot.

Változók: itt nem beszélhetünk valódi változókról, inkább amolyan nevesített konstans szerepet töltenek be. A változók általában egy XPath kifejezés által meghatározott értéket vesznek fel, ami lehet konkrét adat, vagy egy XPath részfa. Ahogyan a programnyelvekben megszokhattuk, a változóknak van hatóköre. Minden változó, amit valamilyen sablonon belül definiálunk, lokális, amit az XSL fájl elején definiálunk globális hatókörrel bír.

Vezérlési szerkezetek: ezek segítségével tudunk ciklust szervezni, s végigjárni mondjuk egy elem összes gyermekelemét, köthetjük feltételhez egy kód végrehajtását, illetve -ami szintén elágazást valósít meg- választhatunk több végrehajtási ág közül az <xsl:choose> szerkezet segítségével.

XPath függvények: csomópont (node-set), sztring, szám és logikai műveletek végzésére használjuk. Segítségükkel tudjuk könnyedén meghatározni mondjuk egy elem nevét, pozícióját, gyermekeinek számát, stb.

Mint az a fentiekből is kitűnik, az XSL az XML-ben történő navigációhoz az XPath lehetőségei használja. A legegyszerűbb módja annak, hogy kiírassuk egy XML fájl tartalmát az <xsl:template> tag alkalmazása a dokumentum gyökérelemére. Ezt illesztett sablonnak is nevezik. Hogy szemléletesek legyünk, a mi esetünkben ez így nézne ki:

```
<xsl:template match="/tulora:tulora">
     <xsl:apply-templates />
</xsl:template>
```

Ezzel annyit érnénk el, hogy az XML dokumentum tag-jei közé írt adatokat, mint szöveget kapjuk meg a képernyőn -már ha a kiíratás a képernyőre történik, mert ne felejtsük el, hogy az XSLT nem csak HTML átalakításra használható, hanem tetszőleges XML – XML, vagy akár XML – SGML átalakításra is. Ha a fenti szövegből mondjuk HTML formátumú dokumentumot akarunk készíteni, akkor a kódunkba be kell írni a szükséges HTML tag-eket:

```
<xsl:template match="/tulora:tulora">
  <head>
     <title>Modositas</title>
     <link href="design.css" rel="stylesheet" type="text/css"/>
  </head>
  <body>
     <xsl:apply-templates />
  </body>
</xsl:template>
```

Így kerül a dokumentum tartalma a <body> és </body> tag-ek közé. Természetesen, ha más XML tag-ekre is hozunk létre szabályokat, akkor elérhetjük, hogy mondjuk a <nap> tag-re illeszkedő elemek egy táblázatban, az <idoszak> tag-re illeszkedő megnevezések egy legördülő listában (kombi panel) jelenjenek meg. Itt azonban vigyázni kell, mert hiába hozunk létre sablont valamelyik gyermekelemre, ha a szülőre is definiáltunk sablont, és nem használjuk az <xsl:apply-templates /> utasítást, akkor a gyermek elemek transzformációit a XSL értelmező nem végzi el.

Még mielőtt létrehoznánk az XSL fájlt, érdemes végiggondolni, hogy a formázási műveleteket -igazítás, betűtípusok, színek- hol érdemes elvégezni. Magában az XSL dokumentumban is leírhatjuk, mikor egy-egy XML tag-re illeszkedő utasításokat hajtunk végre, hogy ezt vagy azt igazítsuk középre, írjuk piros színnel, vagy félkövér kiemeléssel. Ekkor azonban egy későbbi módosításnál az XSL dokumentumban kell majd turkálni, ráadásul többletmunkát is adtunk magunknak, hiszen várhatóan szövegeink, táblázataink, nyomógombjaink ugyanúgy kell, hogy kinézzenek függetlenül attól, hogy egy nap, vagy egy időszak adatait tartalmazzák. Ilyen megfontolásból mindenképpen érdemes kaszkádolt stíluslapokat, CSS-t használni, ahol definiáljuk, hogy alapértelmezésként hogyan nézzenek ki a táblázatok, fejlécek, stb. Így csak az ezektől eltérő esetekben kell direkt módon szabályozni a

megjelenítést, de még ezekre is használjunk inkább nevesített stílusokat, -osztályokat- amit szintén a CSS-ben definiálhatunk. Az XSL fájlban így formázást alig kell végeznünk.

Így már nekikezdhetünk az XSL fájl létrehozásának. Azzal kezdjük, hogy megállapítjuk, melyik időszakot is kell majd megjeleníteni. A PHP szkripteknél már volt szó arról, hogy a MySQL által visszaadott XML sztring egy adott felhasználó összes időszakát tartalmazza, s ezek közül csak egyet -a felhasználó által kiválasztott időszakot- kell a képernyőre kiíratni. Erre a célra hozunk létre egy $valsztott_idoszak nevű paramétert. Ennek a paraméternek az értékét tudja majd megváltoztatni a PHP szkript. Ha a PHP nem állítaná be ennek az értékét -nincs választott időszak, mert a felhasználó most jelentkezett be-, akkor a legelső időszak kerül listázásra.

```
<xsl:param name = "valasztott_idoszak" >
    <xsl:value-of select = "//idoszak[position() = 1]/@id"/>
</xsl:param>
```

Megjegyezzük, hogy a MySQL fordított időrendi sorrendben adja át az időszakokat, így tehát az utoljára létrehozott időszak lesz az első, ami jó választásnak tűnik. Ha a sorrend nem lenne megfelelő, esetleg más szempont szerint kellene rendezni, XSL utasítások segítségével ez is megoldható lenne.

A $valasztott_idoszak paraméter tehát felveszi az első időszak azonosítóját, illetve a PHP szkript által beállított értéket. Ennek segítségével egy legördülő listában összegyűjtjük az időszakok nevét úgy, hogy a választott időszak legyen kijelölve. Ehhez egy ciklus segítségével végigszaladunk az időszakokon. Az XSL-ben ciklusszervező „utasítás” az <xsl:for-each select = ”XPath kifejezés”> tag, ami a megadott Xpath kifejezés összes elemén végigmegy. Alapvető különbség a korábban megismert <xsl:template match = ”XPath kif.”> tag-gel szemben, hogy ugyan az is illeszkedik minden elemre, ami a kifejezésnek megfelel, nem ciklusszerűen megy végig az elemeken, így az egyes elemek elérése nehézkes, valamint fontos különbség még, hogy sablonokat nem lehet egymásba ágyazni, míg ciklusoknál ez nem jelent gondot. Jellemzően úgy alkalmazzuk ezeket, hogy sablont illesztünk a szülő tag-re, és annak leszármazottain futtatunk végig egy ciklust. Most is így járunk el, a

gyökérelemre illesztett sablonban végigfutunk az összes időszakon, és a megnevezésüket összegyűjtjük egy listába:

```
<form action = "process.php">
<select name = "idoszak" onChange = "javascript:form.submit()">
<xsl:for-each select = "//idoszak">
   <option value = "{@id}">
      <xsl:if test = "@id = $valasztott_idoszak">
           <xsl:attribute name = "selected" />
      </xsl:if>
   <xsl:value-of select = "megnevezes"/>
   </option>
</xsl:for-each>
</select>
</form>
```

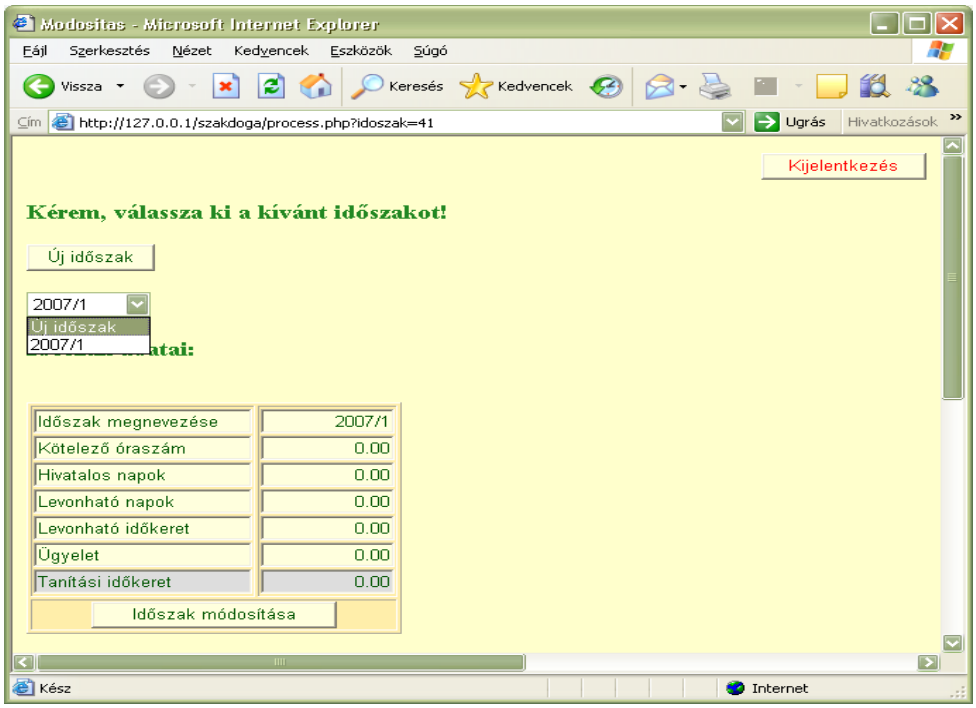
Az <xsl:value-of ...> tag illeszti be a megadott XPath-t, mely általában egy elem tartalmát, míg az <xsl:if ...> szerkezettel lehet a feltételes végrehajtást megvalósítani.

Miután a listánk elkészült, rátérhetünk az időszak adatainak listázására. Ezeknek az adatoknak a többsége módosítható, ezért a listázás alatt itt egy input elembe történő kiírást érjük. A képlet itt is egyszerű, végigfutunk a kiválasztott <idoszak> tag minden elemén. Azaz, hogy nem minden elemén. Ki kellene zárni azokat az elemeket, melyek a hetek adatait tartalmazzák, illetve az összesítést. Szerencsére az XPath-ban tehetünk megkötéseket predikátumok segítségével arra nézve, hogy mely elemekre illeszkedjen a kifejezés. Így akár megnevezés, akár pozíció alapján ki tudjuk zárni a szükségtelen elemeket, vagy csupán csak azért, hogy valamelyik elemet a lista végére helyezzük. Az alábbi példában a tanítási időkeretet tesszük a lista végére, mert az egy számított érték, így nem módosítható. Ezt a tulajdonságát be is állítjuk az <xsl:attribute name = ”readonly”> tag segítségével, illetve, hogy a felhasználónak is egyértelmű legyen, szürkére színezzük (4. ábra):

```
<xsl:for-each select="child::*[local-name() != 'tanitasi_idokeret' and
  local-name() != 'hetek' and local-name() != 'idoszak_osszesites']">
        <input type = "text" name = " #{ local-name( self::*) }" value =
        "{self::*}"size="10" /><br/>
</xsl:for-each>
<input type = "text" name = "#tanitasi_idokeret" value = "{child::*[local-
  name() = 'tanitasi_idokeret']}" size="10" >
   <xsl:attribute name = "READONLY" />
   <xsl:attribute name = "style"> background-color: #dddddd </xsl:attribute>
</input>
```

Ez egy nagy előnye az XSL feldolgozásnak, hogy tetszőleges sorrendben jeleníthetjük meg adatainkat -természetesen tetszőleges formázással-, akár ki is hagyhatunk adatokat a feldolgozásból. Ugyanannak az XML dokumentumnak egy

másik XSL fájl segítségével teljesen más külsőt varázsolhatunk, ami nem csak formázásban, hanem sorrendben, de akár tartalmában is eltérhet a korábbitól. Nem csoda ezek után, hogy ezt a technológiát előszeretettel alkalmazzák web felületek készítésekor, hiszen később nagyon rugalmasan szabhatjuk át a megjelenítést, sokkal rugalmasabban, mint egy stíluslap alkalmazásával.



4. ábra: Eddigi munkánk eredménye

A hetek adatait hasonló módon listázzuk, hiszen ott is az adatok többsége módosítható. Problémát okozhat viszont a napok azonosítása. Míg az időszakoknál ez egyszerűen egy paraméter beállításával megoldható volt, a heteknél ez kicsit másképp fog működni. Leginkább azért, mert a képernyőn egy időszak adatai láthatóak, azaz, ha valaki módosít az időszak adatain, egyértelmű, hogy melyik időszakot kell megváltoztatni. Viszont a választott időszak több hetet, hetenként több napot is tartalmazhat, ezért ezek azonosítása nem olyan egyértelmű. A PHP szkripteknél már utaltunk arra, hogy az azonosításhoz az <input> elem nevébe bevisszük a nap azonosítóját, ami a <nap> tag egy attribútuma az XML-ben. Általában javasolt az XML dokumentumok készítésénél, hogy tag-ek közé olyan értékeket tegyünk, melyre adatként van szükségünk, attribútumokban pedig

olyanokat tároljunk, melyek valamilyen tulajdonságot, jellemzőt tartalmaznak [3]. Persze el lehet térni ezen ajánlástól -ahogyan mi is tettük az összesítés esetén-, hiszen a tag-ek, illetve attribútumok használata szinte midig felcserélhető egymással. Mi viszont az azonosítót attribútumként definiáltuk, így a feldolgozás során érdemes egy változóba elhelyezni ezt az értéket, mivel a <nap> tag egy összetett elem, és a gyermekelemein fogunk végigmenni egy ciklussal, s mindig vissza kellene utalni a szülő @id attribútumára. Az <input> elem nevének meghatározása az alábbi programrészleten látható:

```
<xsl:template match="//nap">
      <xsl:variable name = "napaz">
           <xsl:value-of select = "attribute::id"/>
      </xsl:variable>
                                        .
                                        .
                                        .
      <xsl:for-each select="child::*">
                                        .
                                        .
                                        .
        <input type = "text" name = " { local-name( parent::*) }
        #{ $napaz }#{ local-name( self::*) }" value = "{self::*}" size =
       "10" /> <br/>
                                        .
                                        .
                                        .
```

A hetek napjait táblázatba foglaljuk, majd az utolsó sorban elkészítjük az összesítést, amit még az SQL-ben egy view segítségével számítottunk ki. Ez a módszer ugyan kicsit terjengős, hiszen az XML tartalmazza az összesített adatokat, amit itt az XSL átalakításkor is ki tudnánk számolni, de az az előnye mindenképpen meg van, hogy ha módosítani kell a számítási módon, akkor nem az XSL vagy a PHP fájlban kell bogarászni, melyeknek szerencsésebb, ha nincs köze az adatok számítgatásához. Illetve, ha több alkalmazásban is felhasználjuk az SQL által előállított XML-t, akkor mindegyik alkalmazásban változtatni kellene a számítás módján, s ez könnyen belátható, hogy nem túl előnyös. Ezért nekünk most számolgatni nem kell, csupán az értékeket kell kiíratnunk a képernyőre. Ezek az értékek -ahogyan korábban is utaltunk rá- egy elem, a <heti_osszesites> attribútumaiként jelennek meg az XML-ben. Mivel ezek az értékek nem módosíthatóak, egyszerű táblázatcellaként jelennek meg a képernyőn

A hetek végén nem csak összesítést kell készíteni, hanem -mivel az adatok módosítása heti bontásban történik- itt kell definiálni azt a form-ot, ami a

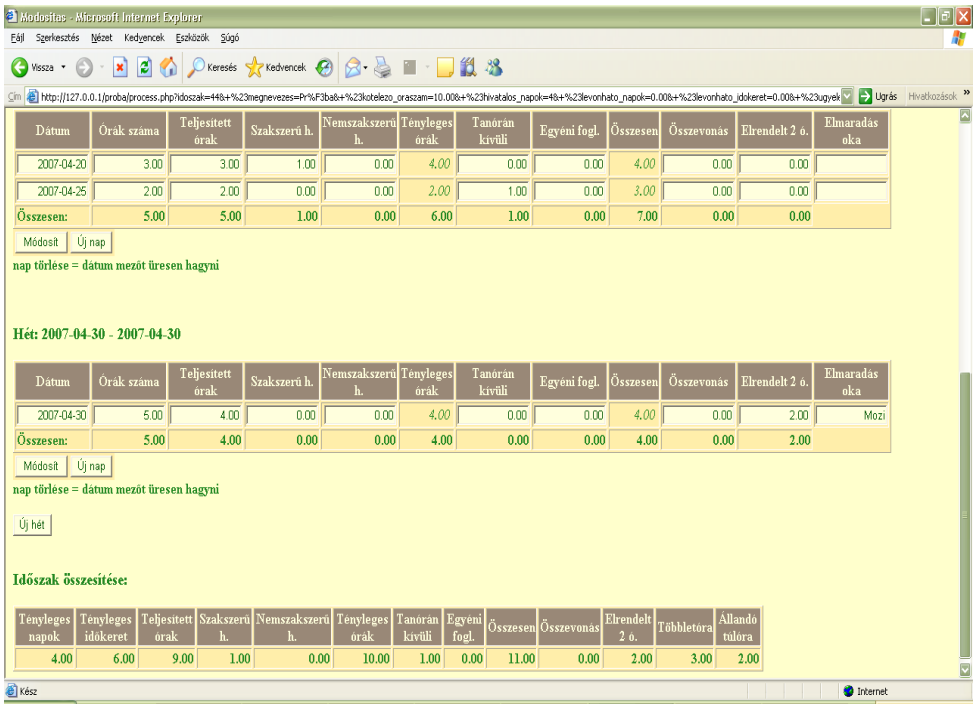
módosítandó adatokat tartalmazza. Ehhez persze szükség lesz a hét azonosítójára is, amit a napoknál tárgyalt módon, szintén egy változóban tárolunk. Ezeken felül tudnunk kell létrehozni új napot is az adott héthez.

```
<xsl:template match="//het">
    <xsl:variable name = "hetaz">
      <xsl:value-of select = "attribute::id"/>
    </xsl:variable>
    <form action = "process.php" name = "modosit" id = "{$hetaz}" >
    <input type = "hidden" name = "idoszak" value = "{$valasztott_idoszak}"
    /> <br/>
    <input type = "hidden" name = "het" value = "{$hetaz}" /> <br/>
    <h3>
        Hét:
        <xsl:value-of select = "descendant::nap[ position() = 1 ]/datum"/>
        -
        <xsl:value-of select = "descendant::nap[ position() = last() ]
        /datum"/>
    </h3>
    <table width="80%" border="1">
     <tr>
        <th>Dátum</th>
        <th>Órák száma</th>
        <th>Teljesített órak</th>
        <th>Szakszerű h.</th>
        <th>Nemszakszerű h.</th>
        <th>Tényleges órák</th>
        <th>Tanórán kívüli</th>
        <th>Egyéni fogl.</th>
        <th>Összesen</th>
        <th>Összevonás</th>
        <th>Elrendelt 2 ó.</th>
        <th>Elmaradás oka</th>
     </tr>
     <xsl:apply-templates />
     <tr>
       <td><b>Összesen:</b></td>
       <xsl:for-each select = "heti_osszesites/@*" >
         <xsl:variable name = "ertek">
            <xsl:value-of select = "."/>
         </xsl:variable>
         <td align = "right">
         <b>
            <xsl:value-of select = "$ertek" />
         </b>
         </td>
       </xsl:for-each>
     </tr>
     </table>
     <table>
     <tr>
       <td><input type = "submit" name = "submit" value = "Módosít" style =
       " text-align:center;" onclick = "ellenoriz( this.form )"/> </td>
       <td><input type = "submit" name = "submit" value = "Új nap" style =
       " text-align:center;"/> </td>
     </tr>
     </table>
     <b>nap törlése = dátum mezőt üresen hagyni</b>
     </form>
</xsl:template>
```

A hétnek az időszakokkal ellentétben nincs neve. Azonosításuk dátum alapján történik, s hogy ez automatikus legyen, itt az XSL átalakítás során „kérdezzük le” a

hét első és utolsó napjának a dátumát. Miután ezt kiírtuk, következik a táblázat fejléce, s utána jönnek a napok. A napok listázása természetesen nem jelenik meg a heteknél, azt egy külön sablon végzi, s a fenti listában a <xsl:apply-templates /> sor helyén kerül beillesztésre. Eztán jön a heti összesítő a fent tárgyalt módon megvalósítva, majd két nyomógomb, a módosítást, illetve az új nap beillesztését megvalósító gomb.

Végezetül az időszak összesítését kell elvégeznünk, ami a heti összesítéshez hasonlóan szintén az XML-ben megtalálható. A megoldás a heteknél tárgyaltakhoz hasonló módon történik, az időszakokra illesztett sablon <xsl:apply-templates /> sora után végezzük el, ahová az új hét beszúrását végző nyomógombot is elhelyezzük..



5. ábra: A hetek és az időszak összesítése

Ezzel tulajdonképpen az átalakítást elvégeztük, a programmal elkészültünk. Már csak annyi feladatunk van, hogy elkészítsünk egy egyszerű bejelentkezési képernyőt, egy adminisztrációs felületet, ahol új felhasználókat tudunk felvenni, esetleg törölni, valamint meg kell még írni a hibakezeléshez szükséges XSL fájlt is.

Összegzés

Most, hogy programunk végére értünk tekintsük át, hogy mennyire tudtuk teljesíteni a kitűzött célt. Egy egyszerűen használható, az intézmény dolgozói számára könnyen hozzáférhető és kezelhető alkalmazás fejlesztése volt a feladat. Azt megítélni, hogy a célt mennyire sikerült megvalósítani egy programfejlesztő nem tudja tökéletesen. Ezt a gyakorlati alkalmazás dönti el. Munkánk során viszont igyekeztünk a felhasználói igényeket, szokásokat messzemenőkig figyelembe venni, a legújabb módszereket, technológiákat beépíteni, ami minden bizonnyal a felhasználók számára is előnyös lesz. Mivel alkalmazásunkat web felületre fejlesztettük, így biztosított az egyszerű hozzáférés, valamint a kezelőfelület sem ismeretlen a felhasználók többségének: űrlapokat kell kitölteni, s azt egy nyomógomb segítségével elküldeni a szervernek feldolgozásra. Ez nem lehet bonyolult feladat. Új elemeket -időszakot, hetet, napot, felhasználókat- szintén ezzel a módszerrel lehet létrehozni. Talán egyedül ahhoz kell egy kicsit hozzászokni a felhasználóknak, hogy a módosítás -így a törlés is- heti bontásban valósul meg, ezért nem muszáj naponként módosítani vagy törölni, lehet egyszerre egy egész heti adatmennyiséget. Ez persze nem zárja ki a naponkénti módosítást, csak gyorsabb, kényelmesebb egyszerre több adatot felvinni, s ezt érdemes a felhasználóknak is megszokni munkájuk gyorsítása érdekében.

Úgy ítéljük ezek után, hogy célunkat sikerült megvalósítani, de hogy mennyire fog tetszeni a célközönségnek az alkalmazás, nehéz előre megmondani. Mivel kötelezővé tenni egy ilyen elektronikus elszámolást nem javasolt, csak abban bízhatunk, hogy a felhasználók felfedezik benne azt az eszközt, amit hatékonyan tudnak használni munkájuk során. Ekkor érdemes majd a felhasználóktól tapasztalatokat gyűjteni, a javaslatokat végiggondolni, s a programot továbbfejleszteni.

A fejlesztés egyik iránya már most is látszik. Ha az alkalmazást kellően sok kolléga fogja használni, akkor célszerű lehet az ellenőrzést is elektronikus úton végezni. Tudjuk, hogy teljesen elektronikus elszámolásra még sokat kell várnunk, papír

alapon zajlik majdnem minden. Viszont az, hogy valamit meg kell őrizni papíron is, még nem jelenti azt, hogy ne lehetne minden lépést elektronikus úton végezni, s csak az ellenőrzött végeredményt megjeleníteni nyomtatott formában. Ez tehát az egyik fejlesztési irány.

Az alkalmazott technológiák lehetővé teszik, hogy a képernyőre kikerülő adatok megjelenését egyszerűen megváltoztassuk. Így igény esetén könnyen készíthetünk megjelenési sablonokat, és mindenki megválaszthatja, milyen elrendezésben, színösszeállításban szeretné látni az adatait. Jelenleg például önkényesen eldöntöttük, hogy az időszakok változó adatai a lap elején, az időszakok összesítése pedig a lap végén helyezkedjen el. Amennyiben valaki jobban szeretné ezeket az adatokat egy helyen látni, egy másik XSL fájl alkalmazásával ezen igénye gyorsan kielégíthető akár oly módon is, hogy más elrendezés jelenjen meg a képernyőn, más a nyomtatón. Ez lehet egy másik fejlesztési irány. Ilyen módon az alkalmazott új technológiák nemcsak a program fejlesztése során jelentettek könnyebbséget számunkra, hanem azok hatása a továbbfejlesztés, módosítás fázisában is előnyként jelentkezik.

Az új technológiáknak köszönhetően programunk könnyen módosítható és hordozható lesz. Adatainkat könnyedén alakíthatjuk olyan formátumba, mely akár egy táblázatkezelő, akár egy szövegszerkesztő program számára feldolgozható, így egyszerűen állíthatunk elő év végi kimutatásokat, statisztikákat nem csak saját intézményünk számára, hanem a pénzügyi ellenőrzést végzők, vagy például a fenntartó számára.

Szintén fontos megemlékezni arról, hogy a túlórák elszámolására az új módszer 2006-ban került bevezetésre, s ez a módszer nem tekinthető teljesen kiforrottnak, többszöri módosítása várható. Így különösen előnyös lehet olyan kódot készíteni, melynek módosítása egyszerűbb. A mi esetünkben ez azt jelenti, hogy ha például az elszámolás módját változtatják meg, akkor sem a PHP, sem az XSL fájl tartalmát nem kell módosítani, kizárólag az SQL tárolt eljárásaiban, vagy nézeteiben (view-kban) történik módosítás, mivel a feldolgozás szintjei jól elválnak egymástól, s nem egy

kódban történik az adatstruktúra meghatározása, a programozási logika, és a megjelenítés, mint némely korábbi technológia esetén.

Mindent összevetve megállapíthatjuk, hogy amellett, hogy célunkat elértük, az alkalmazott technológiáknak köszönhetően egy rugalmasan kezelhető, az igényeknek megfelelően több irányba is fejleszthető alkalmazást sikerült készíteni, mely remélhetőleg a hétköznapokban is megállja a helyét, és segíteni fogja a pedagógusok túlmunkájának adminisztrálását.

Irodalomjegyzék

[1] Segédanyag a pedagógusok kéthavi tanítási időkeretének megállapításához
http://www.okm.gov.hu/main.php?folderID=723&articleID=227569&ctag=articlelist&iid=1, 2006. 08

[2] KIR-dev team: XML, XSLT
http://sztk.sch.bme.hu/files/eloadasok/92/xml.pdf, 2007. 04

[3] Brett McLaughlin: Java és XML, Kossuth kiadó, 2001

[4] MySQL 5.1 Reference Manual, www.mysql.org, 2007. 04

[5] PHP manual
http://www.php.net, 2007. 04

[6] Bevezetés az XML-be
XML-sémák használata
www.prog.hu/cikkek/?aid=892&nfr=1&auth=67dc0d3e83a, 2007. 04

Melléklet

Heti elszámolólap és időszaki összesítő

Berzeviczy Gergely Kereskedelmi és Vendéglátóipari Szakközépiskola, Miskolc, Hősök tere 1.

Pedagógus teljesítmény nyilvántartó lap

Pedagógus neve: Gipsz Jakab

2007 9. hét

Napok:	Órarendi órák száma	Teljesített órarendi órák száma	Helyettesítés		Ténylegesen teljesített tanítási órák száma	Tanórán kívüli órák (szakkör, tömegsport stb.)	Egyéni fogl. (tehetség-gondozás)	Összesen	Összevo-nás	Elrendelt 2 óra	Óraelmara-dás oka
			Szakszerű	Nem szakszerű							
	1.	2.	3.	4.	5=2+3+4	6.	7.	8=5+6+7	9.	10.	11.
2007. február 26. hétfő	3	3			3			3			
2007. február 27. kedd	5	5	2		7	1		8			
2007. február 28. szerda	4	4		1	5			5		2	
2007. március 1. csütörtök	5	3			3		1	4			továbbképzés
2007. március 2. péntek	6	6			6			6			
Összesen	**23**	**21**	**2**	**1**	**24**	**1**	**1**	**26**		**2**	

2007-03-02
dátum

pedagógus aláírása

ellenőrizte

Berzeviczy Gergely Kereskedelmi és Vendéglátóipari Szakközépiskola, Miskolc, Hősök tere 1.

Teljesítmény mutató kiszámítása (kéthavi elszámolás)

2007

Időszak: 2007. 1-9 hét

Kötelező óraszám:	18	Hivatalos tanítási napok száma:	49	Tanítási időkeret:	172,8
		Levonható (betegség, anyasági, hivatalos):	1	Levonható keret:	2
		Tényleges tanítási napok száma:	48	**Tényleges keret:**	170,8

Hetek:	Teljesített órarendi órák száma	Helyettesítés		Ténylegesen teljesített tanítási órák száma	Tanórán kívüli órák (szakkör, tömegsport stb.)	Egyéni fogl. (tehetség-gondozás)	Összesen	Összevo-nás
		Szakszerű	Nem szakszerű					
9. hét	21	2	1	24	1	1	26	
10. hét	22			22			22	
11. hét	20			20			20	
12. hét	21			21			21	
13. hét	22			22			22	
14. hét	22			22			22	
15. hét	22			22			22	
16. hét	23			23			23	
17. hét	23			23			23	
18. hét	9			9			9	
Összesen	**205**	**2**	**1**	**208**	**1**	**1**	**210**	

Kifizetések:		
Többletóra:		39,2
Ebből	Állandó túlóra	37,2
	Szakszerű helyettesítés:	
	Nem szakszerű helyettesítés:	2
	Szakszerű összevonás:	
Hiányzások:		
	Távollét	1
	Ügyelet	

2007-04-08
dátum

pedagógus aláírása

ellenőrizte

számfejtette

igazgató

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek Adamkó Attilának, aki tapasztalatával és tanácsaival segítette munkámat.