Szakdolgozat

Sarkadi Csaba

Debrecen
2009

Debreceni Egyetem
Informatikai Kar

Alkalmazásfejlesztés Lotus Notes alapokon

Témavezető:
Bodroginé Dr. Zichar Marianna
Egyetemi adjunktus

Készítette:
Sarkadi Csaba
Programtervező Informatikus szak

Debrecen
2009

Tartalomjegyzék

0. Bevezetés 4
1. A Lotus Notesról általánosságban 5
1.1. A Lotus Notes és történelme 5
1.2. Előnyök és hátrányok 6
2. Design elemek és használatuk 8
2.1. Form 11
2.1.1. Profil form 13
2.1.2. Subform 13
2.1.3. Form típusok – Document, Response, Response to Response 14
2.2. View 15
2.3. Page 17
2.4. Agent 18
2.5. Frameset 20
2.6. Outline 21
2.7. Folder 21
3. Programozás 22
3.1. Egyszerű parancsok 23
3.2. Formula nyelv 24
3.3. Lotus script 25
3.4. Java 26
3.5. Library-k 27
4. Kapcsolódási lehetőségek más adatbázis kezelőkkel 28
4.1. LEI 28
4.2. LSX 29
4.3. JDBC 30
5. Biztonság 31
5.1. ACL 31
5.2. Biztonsági / hozzáférési szintek 32
6. Tippek, trükkök, tanácsok 33
6.1. Java agent debuggolása külső fejlesztői eszközökkel (Eclipse és Netbeans) 33
7. Egy példa alkalmazás tervezése és fejlesztése 38
7.1. Az alkalmazás által használt formok 38
7.1.1. Profil formok az adatbázisban 38
7.1.2. Az effektív adattárolásra szolgáló formok 39
7.2. Kapcsolatok az egyes formok között 40
7.3. Műveletek 41
7.4. Szerepkörök 42
8. Az újdonságok 43
8.1. 7.0-ban 43
8.2. 8.0-ban 44
8.3. 8.5-ben 44
9. Irodalomjegyzék 45
10. Függelék 46
11. Rövidítések, szómagyarázó 51
12. Utószó 52

Bevezetés

Egyetemistaként sok mindent tanultam, mégis miután kézhez kaptam az első diplomámat, kiderült, hogy a megszerzett tudásom az informatikai szegmensben nem rendelkezik megfelelő piaci értékkel.

Friss diplomásként az első munkahelyemen olyan nagyvállalati technológiákkal találkoztam, amelyek piacvezetőek, mégis egyetemi tanulmányaim során nem hallhattunk róluk semmit (kivétel ez alól egyedül az Oracle Applications, melyet az egyetem és a National Instruments szervezésében ismerhettünk meg minimális szinten).

Az egyik ilyen piacvezető technológia volt a Lotus Notes is, mely egy frissen végzett hallgatónak elsőre nagyon nehezen érthető és használható, szakirodalom hiányában nehezen felfogható. További nehézség, hogy az egyetemeken adatbázis-kezelési alapok terén csak a hagyományos, relációs elveket oktatják, illetve ha egy hallgatónak éppen szerencséje van, akkor részt vehetett olyan kurzusokon, melyek az objektum orientált adatbázis-kezelők alapjait mutatják be.

Szeretnék elkészíteni egy olyan segédletet, mely azon programozók és egyetemisták számára nyújtana egy rövid, tömör áttekintést, akik még csak most ismerkednek a Lotus Notes alapjaival. Ezen kívül szeretném megosztani azon irányú tapasztalataimat is, melyeket az első munkahelyemen szereztem. Például hogyan nem szabad fejleszteni és programozni, mit eredményez az, ha csak a határidő létezik és nem a minőségi munka.

Egy példaalkalmazáson keresztül bemutatom továbbá a saját tapasztalataimat, illetve, hogy meglátásom szerint hogyan ajánlott Lotus Notes alapokon alkalmazásokat fejleszteni.

Mindezek mellett egy pár nagyon különleges trükköt is megosztok az olvasókkal, ezek közül talán az egyik legfontosabb, hogy Javaban írt kódokat hogyan lehet debuggolni Java fejlesztői eszközön keresztül.

Szakdolgozatomban vegyesen fogom használni az angol és a magyar szakszavakat, mivel meglátásom szerint nagyon sok kifejezést egyszerűen nem lehet magyarra fordítani, vagy ha igen, akkor csak nagyon erőltetetten, ami sokak fülét bántja (ilyen fordítások pl. az „ügynökök”, „terv elemek”, „kód könyvtárak”).

1. A Lotus Notes-ról általánosságban

1.1 A Lotus Notes és történelme

A Lotus Notes egy kollaborációs, nagyvállalati platformnak készült, hálózati kapcsolaton keresztül egymással kommunikáló munkaállomások számára 1989-ben. A mai napig a legsikeresebb ilyen termék ebben a szegmensben, nemzetközileg több mint 100 millió eladott felhasználói licensszel. Sikerének egyik titka abban áll, hogy a Lotus fejlesztői gárdája mindig is igyekezett lépést tartani az aktuális informatikai trenddel. A másik nagy titka a RAD, azaz a gyors alkalmazásfejlesztés támogatása.

A platform hatalmas sikerét megirigyelve az IBM 1995-ben vásárolta fel, és ezután az új verziók egyre nagyobb és nagyobb vásárlói kört jelentettek a kék óriás számára. A 6.0-ás verzió idején már 80 millióra becsülték az eladott licenszek számát. Ezen fejezet írása idején a 8.5-ös verzió megjelenése várható (pontosabban 2008 decemberében), mely igen jól használható újításokat fog bevezetni, de erről részletesebben a 8. fejezetben írok.

1.2 Előnyök és hátrányok

Ebben a fejezetben összesítem a Lotus Notes alkalmazásfejlesztés és a Domino infrastruktúra előnyeit és hátrányait, illetve bemutatom, hogy a hagyományos adatbázis-kezelés és alkalmazás-fejlesztéshez képest mire számíthatunk, ha Lotus Notes környezetet használunk.

Előnyök

- „lazán kapcsolt" adatelemek – a hagyományos relációs adatbázis-kezelőkkel ellentétben itt nincsenek olyan megszorításaink, mint pl. elsődleges kulcs, normál formák, vagy egyértékű mezők. Sokak szemében ez egy hátránynak tűnhet, és a hagyományos felfogás móddal szemben nehéz hozzászokni.
- A tároló adatmodell – a „jegyzet" adatbázis
- Egyszerű adatbázis létrehozás, és könnyített adatbeviteli módszerek.
- Az előző pontban taglaltakhoz tartozik, hogy a hagyományos adatbázis-kezeléssel ellentétben a programozónak nem kell SQL utasításokat kódolnia, azt a Domino lekezeli helyette.
- Hordozhatóság és távoli hozzáférés – azaz a replikáció. Ez azt jelenti konkrétan, hogy egyrészt a felhasználó lemásolhat magának a saját számítógépére egy olyan adatbázist, amihez megfelelő jogosultsággal rendelkezik, és azt bármikor használhatja hálózati kapcsolat nélkül. Másrészről pedig replikáció alatt értjük azt is, amikor az adott vállalaton belül lévő szervereken található adatbázisok tartalma bizonyos időközönként (ez a replikációs idő) szinkronizálódik, azaz előáll egy olyan pillanat, amikor az adatbázisok tartalma megegyezik.
- Használhatunk többértékű mezőket.
- Sokféle programozási nyelven fejleszthetőek az egyes komponensek (Formula nyelv, Lotus Script, Java Script, Java, C++)
- A Java nyelven írt kódok nagyon hatékonyan futnak a Domino infrastruktúrán, és így nagyon jól kiegészítik azt.
- Gyors alkalmazás-fejlesztés (RAD) támogatása.
- 4.6-os verziótól a Domino server támogatja a HTML-t, és a dokumentumokat képes web böngészőn keresztül, szabvány HTML kóddal megjeleníteni.

Hátrányok

- A korábban említett előny miszerint nem relációs adatbázis-kezelő, egyben az egyik igazi nagy hátránya is. A Lotus Notes felépítéséből adódóan nem tudunk olyan lekérdezéseket készíteni, mint a relációs adatbázis-kezelőkben (értem ezen a JOIN-t, mint alapvető műveletet), így a redundáns adattárolás nagy mértékű.
- Habár támogatja a Java programozási nyelvet, igazán elmaradott szinten teszi azt. Az 5-ös, a 6-os és a 6.5-ös verziók még csak az 1.3-as Java JDK-t támogatják, a 7.0-ás, 8-as, és 8.5-ös verziók pedig még csak az 1.4-es JDK-t. Itt hozzátenném azt, hogy a 6.0-ás verzió megjelenésekor már a piacon az 1.5-ös verziójú JDK megjelenés előtt állt éppen, a 8.0-ás verzió esetében pedig már az 1.6-os verzió is elérhető volt (1.6 update 4).
- Nem cserélhető a JVM a Domino server alatt.

- Nagy (milliós) mértékű adattárolásra nem igazán használható hatékonyan a felépítéséből adódóan.
- A tökéletes webes megjelenítéshez szükséges a magas szintű Java script ismeret.
- A fejlesztést csak Windows környezetben tudjuk elvégezni.
- Sok helyen a felhasználók és a fejlesztők elvből ellenzik, ahelyett, hogy megismernék.

2. Design elemek

2.0 Domino Designer – A fejlesztői felület

A fejlesztéshez a legfontosabb eszközünk a Domino Designer lesz.

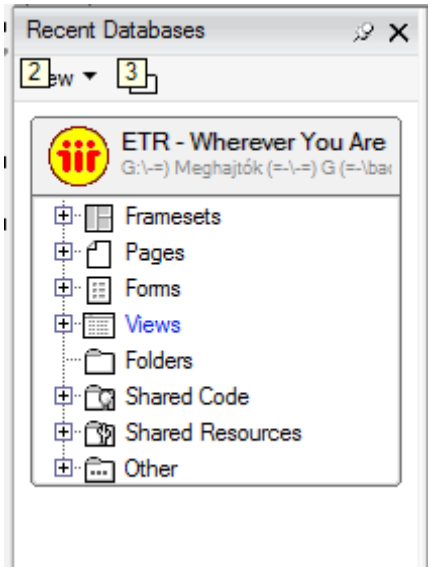
Első lépésként ajánlatos egy új adatbázist készíteni, amiben a Domino fejlesztéssel ismerkedő programozók kipróbálhatják az egyes design elemeket és a különböző programozási nyelveket. Ezt megtehetjük a File → Create new database → a megjelenő ablakon írjunk be egy állomány nevet, és válasszuk ki a blank template-t.

A teljes felületről a függelékben található egy kép.

Ezután a baloldalt látható design pane megjelenik számunkra egy rövid összesítő az adatbázisban elhelyezhető (létező adatbázis megnyitása esetén a már létrehozottak is) design elemekről.

Itt egy hierarchikus rendezésű menüben láthatjuk a fejlesztéshez használható eszközöket.

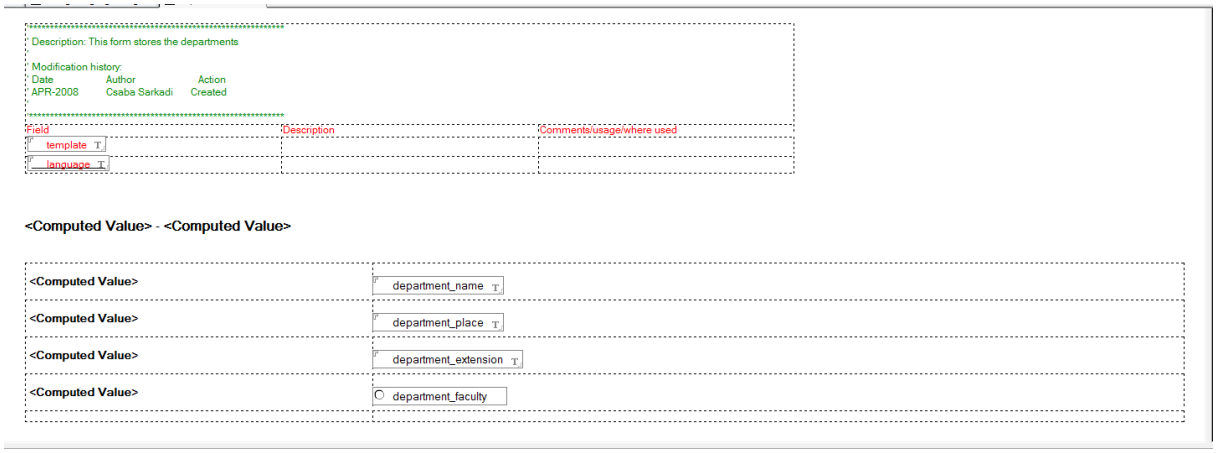
A felület további része még a menüsor. Itt találhatjuk az alapvető szerkesztési, megnyitási, mentési műveleteket, illetve több kontextus-érzékeny menüpontot is.



1. Ábra a design pane

2. Ábra a menüsor

A felületen található még a work pane, a munkafelület. Ez több részre tagolódik, amennyiben egy adott design elemet szerkesztünk éppen. Láthatunk egy grafikus felületet, egy információs listát és egy script szerkesztőt.

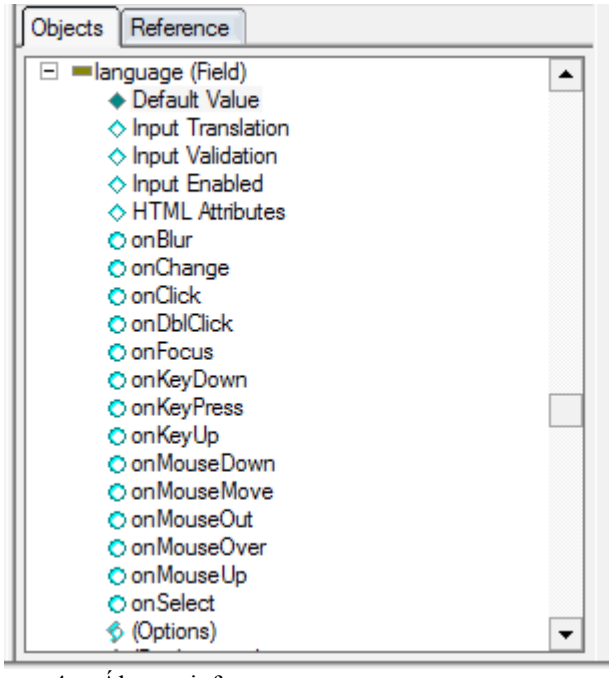


3. Ábra a work pane grafikus felülete

Az információs lista segítségével az egyes design elemekhez tartozó objektumokat (elsődlegesen az eseményeket, mezőket és gombokat) és a referenciákat tekinthetjük meg.

Ezek közül a legfontosabbak az egyes objektumokhoz kapcsolódó események lesznek.

Érdemes megfigyelni, hogy a mezőkhöz adhatunk validációs formulát is (ellenőrzi az adott mező helyességét, formátumát), illetve fordítási (translation) formulát, ami a beírt adatokat alakítja az általunk kívánt formátumra.



4. Ábra az info pane

A designer felület utolsó nagyobb, eddig nem tárgyalt része a script editor. Ide írhatjuk az egyes gombokhoz, mezőkhöz, eseményekhez tartozó kódokat. A felületen választhatunk, hogy a kód webes, vagy kliens környezetben fusson, illetve lotus script, javascript és formula nyelv lehetőségeket választhatunk.

A következő ábrán látható ez a felület, és az opciók választásának listái.

5. Ábra a script szerkesztő

2.1 A form

A form az alapvető adatbeviteli egysége a Lotus Notes-nak. Relációs környezetben 1 form megfelel egy sémának, 1 dokumentum pedig a táblázat 1 sorának. Az adatok definiálásán kívül az adatok megjelenítését is a form elkészítésével tudjuk megoldani.

Ugyanazon adatok megjelenítését több különféle módon is megoldhatjuk.

1. A Page-en helyezzük el a megjelenítendő mezőket.
2. Subformon
3. Készítünk egy újabb formot, aminek ugyanazt az alias nevet adjuk, mint az eredeti formnak. Ez a leggyakoribb megvalósítás, így is szokták előkészíteni a dokumentumok webes megjelenítését.

Mindezt könnyen és gyorsan megtehetjük a Designerből.

Új form létrehozásához válasszuk ki a Form feliratot a designer pane-ről, majd a megjelenő lista tetején kattintsunk a New Form gombra. Ezután adjuk meg a nevét (esetleg egy aliast is úgy, hogy a név után a pipe jel „|" elhelyezése után írjuk be).
Létrehozás után megadhatjuk a form címét is, ami a létrejött dokumentumokon lesz látható. Ez lehet konstans szöveg, vagy formulák segítségével számított cím is.

Formok elnevezése esetén nincs sok megkötés, nem is nagyon van elterjedt konvenció. Alias esetén viszont törekedjünk arra, hogy ne tartalmazzon szóközöket, azokat pótoljuk _ jellel, és törekedjünk ebben az esetben a rövidségre.

Itt rögtön megjegyezném az alias használat jelentőségét, illetve az értelmét. Segítségével egy dokumentumnak többféle megjelenítést tudunk kölcsönözni, környezettől függően. A Domino 8.5-ös verzió előtt ezt a lehetőséget leginkább arra használták, hogy a webes és a klienses megjelenítést elválasszák egymástól. Ilyenkor a megegyező aliasú formok fognak megfelelni egymásnak. Nyilván a mező nevek egyezősége ajánlott, egyéb esetben nem fogjuk látni azokat az adatokat, amiket szeretnénk.
Megjegyzés: alias névnek ajánlatos rövid, tömör és szóközmentes nevet adni. Segítségével az alkalmazásunkat is könnyen többnyelvűsíthetjük.

A formok esetén érdemes megjegyezni, hogy relációs környezetben egy sor beszúrása a különböző programozási platformokon plusz kódolással jár, legtöbbször egy INSERT INTO kifejezés és a hozzá tartozó kódrészlet megírásával. A Lotus Notes környezet ezt a munkát megspórolja nekünk, ezzel nekünk nem kell törődnünk, így csökken a hiba lehetősége, a későbbi bővítések esetén a plusz munka, és gyorsul a fejlesztés üteme.
(Összehasonlításképpen ehhez hasonló technológia a Java EE 1.5-ös verzióval jelent meg a persistence apival).

Fontos megjegyezni, hogy egy apróbb hátránnyal is jár ez a funkció. Azaz amennyiben bővítünk egy formot, pl. új mezőket veszünk rá fel, akkor az azon formmal létrehozott dokumentumok esetén manuálisan kell hozzáadnunk az új design elemeket.
Ez történhet agent segítségével, vagy a dokumentumokon végrehajtott egyszeri mentés segítségével.

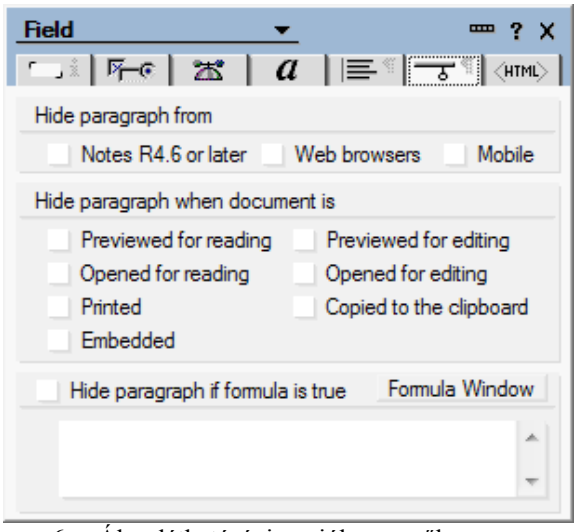
A formoknak többféle típusa is lehet, amit a form tulajdonságai ablak első fülén tudunk állítani. Ezen lehetőségek a Document, Response, Response to Response. Erről bővebben a 2.1.3-as fejezetben olvashatunk.

A form szerkesztő felület nagyfokú rugalmasságot nyújt számunkra a megjelenítési opciókban. A form egyes elemeit, szövegrészleteket, gombokat, választási listákat, mezőket elrejthetünk és megjeleníthetünk, teljesen programozottan. Ez a gyakorlatban jelentheti azt, hogy a form egyes részeit elrejtjük a megfelelő hozzáférési szinttel nem rendelkező felhasználók elől (a pénzügyi részlegek kedvence), vagy dinamikus kérdőívet állíthatunk elő (pl. ha a kitöltő felhasználó férfi, akkor a következő kérdés nem a férj neve lesz, hanem a felesége neve).

Ezt a funkciót a megfelelő elem kiválasztása után a tulajdonságok menüjének láthatósági panelén állíthatjuk be.

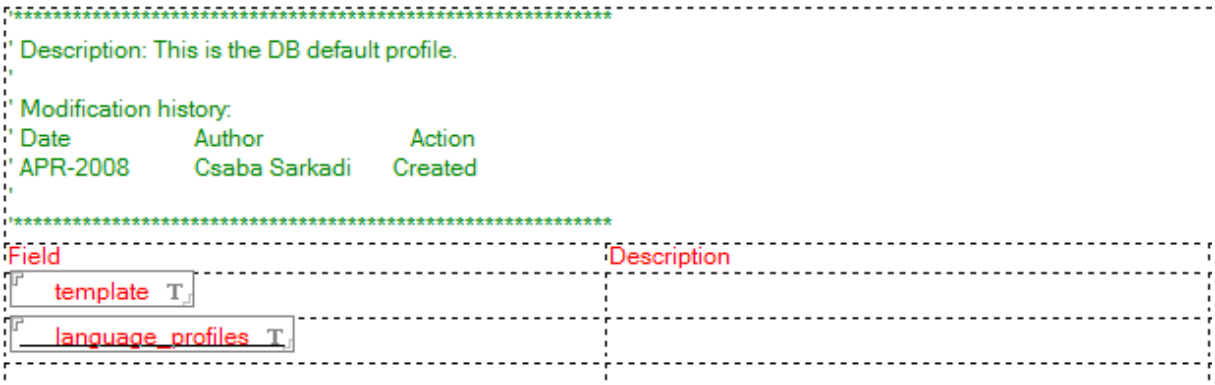
Az opciókat megfigyelve a rejtés (hide-when opció) kiterjedhet a környezetre (Web, mobil készülék, LN 4.6-nál régebbi kliens), egyes események teljesülésére (pl. a szerkesztésre vagy olvasásra van megnyitva).

A programozott elrejtéshez a Hide paragraph if formula is true opció bekapcsolása után egy formulát kell írnunk, melynek igaz vagy hamis értéket kell eredményeznie.



6. Ábra láthatósági opciók a mezőkre

A formok megtervezésekor ajánlott azok tetejére egy rejtett területet létrehozni, melynek tartalma egy módosítási történet (modification history) legyen, hogy nyomon követhessük, mikor ki és mit módosított a form arculatán. A rejtett mezőket (melyek a használat során mindvégig rejtve maradnak és csak programozási okokból fontosak) szintén ajánlott ide elhelyezni egy táblázatba. A bal oszlop tartalmazza a mezőket, a jobb pedig a hozzájuk tartozó magyarázatot, ami a többi programozó jövőbeni munkáját segítheti.



7. Ábra a form történet, a rejtett mezők és a hozzájuk tartozó leírás.

2.1.1 A profil form

Ahhoz, hogy a „hardcodeolást”, azaz a konstans értékek használatát kiküszöbölhessük a program kódokból, a Lotus Notes a profil formot adja számunkra, mint programozói eszközt. Ez gyakorlatilag egy formot jelent, amin a konstans értékeket tároló mezőket helyezzük el. Annyi kikötés van összesen, hogy a profil dokumentumot egy speciális paranccsal kell létrehoznunk (@Command([EditProfile];”form_neve”);), és a profil form mezői nem szerepelhetnek nézetekben.

Általánosan elmondható, hogy egyféle profil formból egy dokumentum példány létezhet az adatbázisban. Ez alól úgy tudunk kivételt képezni, hogy egy adott típusú profil formhoz megadhatunk egy kulcsot létrehozásakor, és mikor a mezőire hivatkozunk, akkor ezt a kulcsot szintén meg kell adnunk. Ez olyan esetekben jó, mikor többszörösen akarunk tárolni egy konstanst több különböző értékkel. Ilyen eset, amikor a különböző felhasználókkal kapcsolatban szeretnénk tárolni egyéni beállításokat, vagy jól jöhet olyan esetekben, ha egy adatbázisban több különböző nyelvi felületet is szeretnénk prezentálni a felhasználóknak.

A profilhoz a hozzáférést ajánlatos korlátozni. Ez történhet speciális mezőkkel (readers vagy writers típus), illetve a formhoz tartozó Queryopen eseményben.

A profil dokumentumok használatának egy további hasznos tulajdonsága az, hogy az adatbázis megnyitása után a profil dokumentum teljes tartalmát beolvassa a kliens (cacheli), így az egyes értékekhez való hozzáférés a lehető leggyorsabb lesz.

2.1.2 A subformok

A fejlesztés során sokszor előfordul, hogy bizonyos form részletek (pl. egy bejelentkező ablak, vagy mondjuk egy weboldal menüsora, header vagy footer része) több formon is előfordulnak. Ezen formok egyszerűsítésére használhatjuk a subformokat, mint nem önálló (azaz önmagában nem szerepelhet, csak egy formhoz illesztve) design elemeket.

Használata felettébb egyszerű. A baloldali menüből válasszuk ki a subforms menüpontot, hozzunk létre egy újat. A szükséges mezőket adjuk hozzá, illetve a szükséges megjelenítési elemeket.

Ha ezzel kész vagyunk, akkor mentés után a már létező subformot bármely formon elhelyezhetjük az Insert shared elements → subform menüt választva.

Érezhető, hogy egy ilyen design elem nagyon hasznos tud lenni a gyakorlatban, rengeteg időt megspórolhat nekünk.

2.1.3 Form típusok – Document, Response, Response to Response

Mint korábban említettem az egyes formoknak 3 típusa van. Ezen típusok lényege, hogy az egyes dokumentumok között egyfajta hierarchiát tudunk létrehozni.

Egy hétköznapi példával élve, egy internetes fórumon egy új témának megfelel egy új dokumentum, az arra adott válaszoknak pedig egy válasz (response) dokumentum. A válaszra adott újabb válaszra pedig a válasz a válaszra (response to response) dokumentum.

Ez Lotus Notes környezetben úgy történik, hogy az egyes típusokkal létre kell hoznunk 1-1 formot. Nyilván a document típusú form nem lehet válasz másik formra, illetve a response dokumentum csak document típusúra lehet válasz.

Ezen dokumentum típusok létrehozását a Lotus Notes nagyon egyszerű programozói parancsokkal támogatja, nekünk gyakorlatilag csak annyit kell tennünk, hogy kiválasztunk egy dokumentumot, megnyomunk egy gombot, és a háttérben a Notes mindent lekezel a programozó helyett. Egyúttal létrejön egy mező is, aminek a segítségével később programkódból fel tudjuk dolgozni az egyes dokumentumok hierarchiákat, a szülő dokumentumtól a gyermek dokumentumokig haladva.

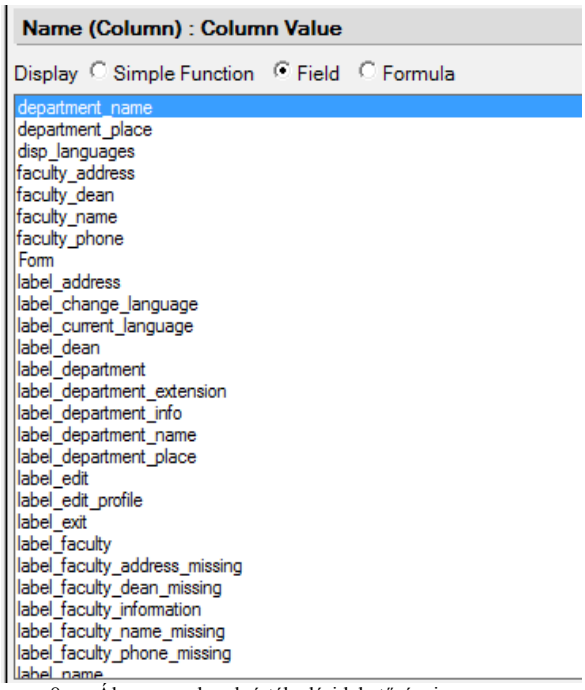
A fordított irány feldolgozásához már nekünk is kell dolgoznunk, egy plusz mezőben el kell tárolnunk az adott formon a szülő dokumentum UNID-ját (lásd 11. fejezet).

2.2 Nézetek

A nézetek Lotus Notes környezetben ugyanazt jelentik, mint a relációs környezetben. Sajnos nem kapjuk ugyanazt a funkcionalitást, mint a relációs adatbázis-kezelőkben. Ahogy korábban említettem, nem tudunk olyan komplex lekérdezéseket készíteni, ahhoz, hogy egyszerre több formon szereplő adatokat jelenítsünk meg melyek valamilyen kapcsolatban állnak egymással, kénytelenek vagyunk a szükséges mezőket redundánsan tárolni az adatbázisban.

Nézetet a design pane-n a Views menüre kattintva a megjelenő lista tetején lévő New View gombbal tudunk létrehozni.

Egy nézeten belül tetszőleges számú oszlopot hozhatunk létre. Az oszlopok tartalma lehet egy mező tartalma vagy egy számított formula.



8. Ábra az oszlopok értékadási lehetőségei

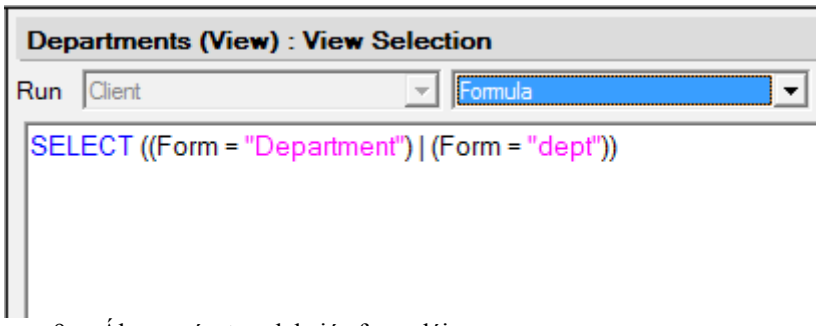
Az oszlopok tartalmát a script editor segítségével adhatjuk meg. Választhatunk mezőt, számítási formulát, vagy beépített függvényt is (pl. létrehozó, létrehozás dátuma, a dokumentum mérete, utolsó módosítás dátuma, az adott dokumentumra adott válaszok száma).

Mint az ábrán látható, a választható mezők listája az összes, az adatbázisban szereplő mező nevét tartalmazza.

Formula esetén az egyes mezőket a nevük alapján hivatkozhatjuk (pl. lét mező tartalmát egyben jeleníthetjük meg).
Az egyes oszlopok tulajdonság menüjében elég sok opciót találhatunk. Elrejthetünk egy oszlopot, ugyanúgy, ahogy a formok esetén a mezőkkel tettük ezt. Állíthatunk méretet, rendezést, stílust, betűkészletet, de egy weben használt nézet esetén az oszlop tartalmát linkként jeleníthetjük meg.

A nézetekben megjelenő sorokat a szelekciós formula segítségével szűkíthetjük vagy bővíthetjük. Ehhez a script editort használhatjuk. Megadhatunk egyszerű keresési feltételeket is (Simple Search), ahol egy adott formhoz tartozó dokumentumokat, vagy egy adott mintának megfelelő dokumentumokat választhatunk ki.

A formulák használatával összetettebb keresési feltételeket adhatunk meg, amikre az egyszerű keresés nem ad lehetőséget.



9. Ábra a nézet szelekciós formulája

Mint korábban említettem, az egyes oszlopokat rendezhetjük valamilyen sorrend szerint (növekvő, csökkenő), illetve rendezés esetén kategorizálhatjuk is az egyes sorokat, aminek segítségével egy lenyíló menüs csoportba rendezhetjük az ugyanazon oszlop értékkel rendelkező sorokat.

A rendezéssel kapcsolatban megadhatunk olyan opciót is, hogy egy oszlop rendezettségét a felhasználó szabadon választhassa meg (Click on column header to short).

Fontos azonban tudni, hogy az egyes rendezettséggel rendelkező oszlopok 1-1 indexet generálnak az adatbázisban, mely hatással van az adatbázis méretére és a sebességre.

Egy szélsőséges példának tudnám felhozni egy korábbi munkám során tapasztalt problémát, melyben egy nézet összesen 10, minden irányban szabadon rendezhető oszlopot tartalmazott, amik miatt az adatbázis indexeinek mérete a felét tette ki az adatbázis teljes méretének. Ezen oszlopok rendezettségének eltávolítása után (gyakorlatilag csak 1 felhasználónak lett volna rá szüksége) a használat érezhetően gyorsabb lett, és az adatbázis újraszervezése után a mérete is jelentősen csökkent.

Az index problémára egy átmeneti megoldást nyújthat a nézet tulajdonságainak módosítása. Választhatjuk azt, hogy addig ne készüljön index, amíg valaki meg nem nyitja az adott nézetet (ilyenkor az első megnyitás lassú lehet), illetve a nézet indexeket el is dobhatjuk, ha a nézetet nem használták megadott számú napig.

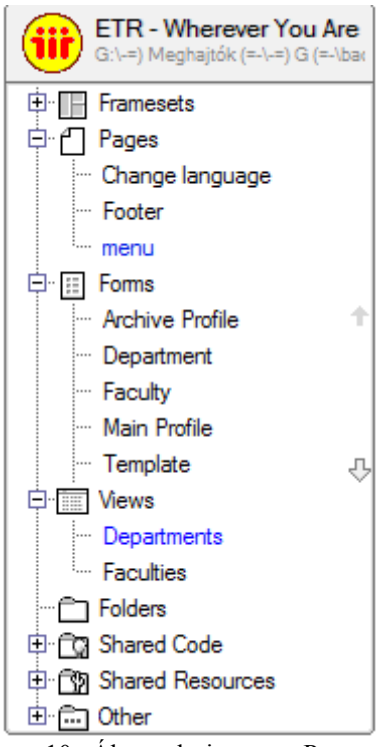
Ehhez kapcsolódóan a Lotus Notes 8.0 egy újdonságot is hozott, mely segítségével egy oszlop indexére állíthatjuk be, hogy csak akkor jöjjön létre, ha használják is azt (azaz a felhasználó módosít az oszlop rendezettségén), ehhez válasszuk az oszlop tulajdonságai közül a „defer index creation until first use” opciót.

2.3 Page

A pagek, vagy oldalak segítségével olyan design elemeket hozhatunk létre, melyek nagyon hasonlítanak a formokra, de segítségükkel az adatokat csak megjeleníteni tudjuk, elmenteni nem.

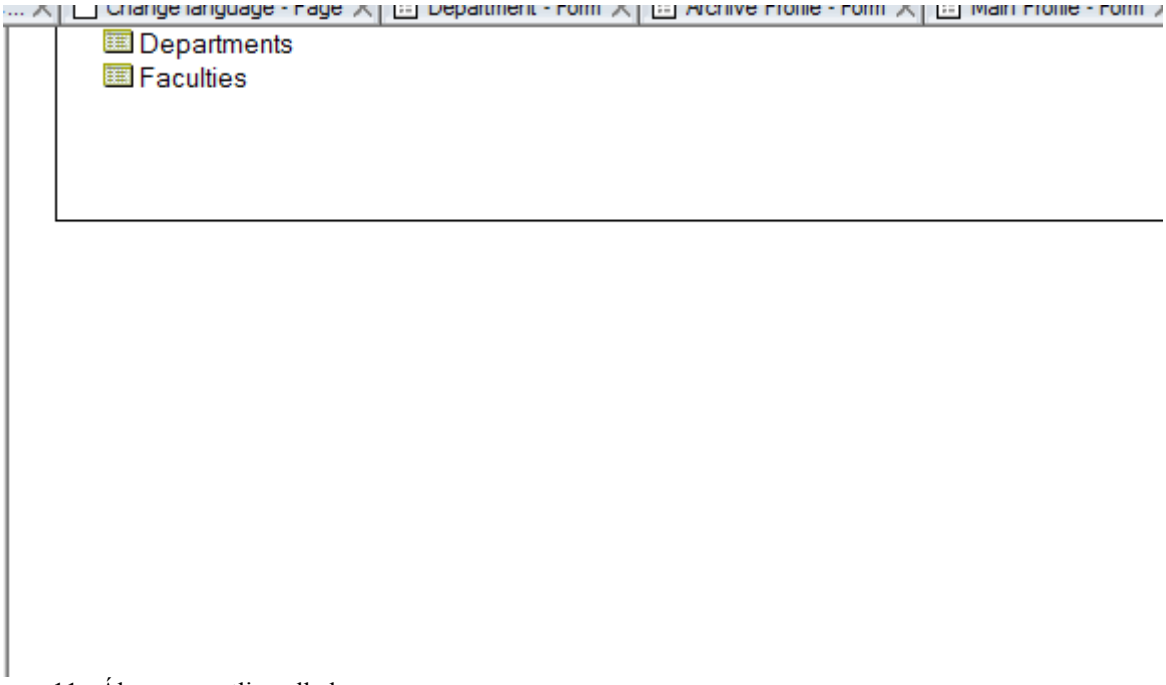
Használata leginkább webes környezetben ajánlott.

Létrehozásához a design pane-ről válasszuk a Pages menüpontot, majd create new.



10. Ábra a design pane Page menüje

A webes megjelenítés mellett a page-ek másik elterjedt felhasználási területe a framesetekhez kapcsolódik. A menüpontokat tartalmazó outlinet egy page-re kell beillesztenünk, és utána tudjuk az adott framesethez beilleszteni a menüt.



11. Ábra egy outline elhelyezve egy page-n.

2.4 Agentek

Az agentek a leggyakoribb megjelenési területei a programkódoknak.
Típusaik szerint lehetnek időzítettek és manuális indításúak. Az időzített agentek olyan programok, amelyek valamilyen szabály szerint megadott időpontokban futnak. Ez az időpont lehetőség lehet gyakori (naponta többször, ebben az esetben meg kell adni az ismétlési gyakoriságot), napi, heti, havi, és még ennél is ritkább futású.

Napi futású agentek esetén megadhatjuk, hogy a nap mely időpontjában fusson, illetve azt is, hogy esetleg csak munkanapokon.

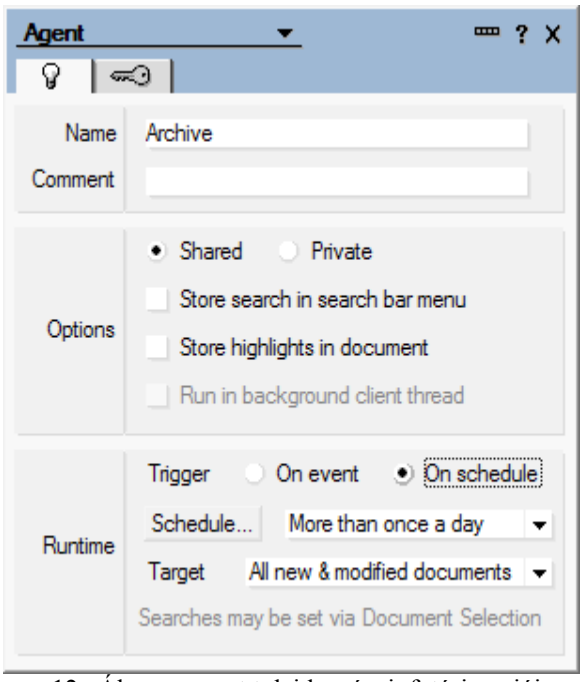
Heti időzítés esetén a hét azon napját, óráját és percét kell megadnunk, amikor szeretnénk, hogy fusson.

Havi futású agent esetén a hónap x.-edik napjára időzíthetjük a kód futását.

Időzített agentek gyakorlati haszna a rendszeres műveletek automatizálásában van, pl. egy pénzügyi alkalmazásban a havi zárás emberi beavatkozás nélküli megoldása jó ötlet lehet.

Kézi indítású agentek esetén megadhatjuk a felhasználónak az indítási lehetőséget, vagy program kód segítségével indíthatunk egy agentet egy esemény hatására is.

Az agentek időzítésének megadásához a script editorból válasszuk az agent tulajdonságai menüpontot.

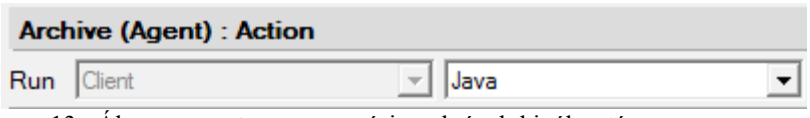


12. Ábra az agent tulajdonságai, futási opciói

Programozási nyelvek használatára 5 lehetőségünk van agentek esetén.
Ezek az egyszerű parancsok (simple command), melyek előre definiált, egyszerű és gyakori parancsokat takarnak (pl. hozzunk létre dokumentumot egy adott form segítségével, vagy küldjünk valakinek egy levelet), a formula kódok, lotus script, javascript és java agentek.

Az éppen aktuális programozási nyelvet a script editor megfelelő legördülő listájából választhatjuk ki.

Az egyes agenteknek amellett, hogy indíthatóak kézzel, vagy futtathatóak időzítetten, megadhatunk egy eseményt, amire végrehajtódhatnak. Ilyen események lehetnek a levelekkel (mielőtt, vagy miután egy levél érkezett az adatbázisba), dokumentumokkal (dokumentum létrejött, vagy módosult, vagy beillesztődött egy nézetbe), illetve felhasználói eseményekkel kapcsolatosak (a felhasználó kiválasztotta az agent vagy az action menüből az agentet, és úgy futtatja kliens oldalon).



13. Ábra az agent programozási nyelvének kiválasztása

Megjegyzésképpen említeném, mint azt már korábban említettem, az időzített agentek mind szerver oldalon futnak, a többi agent pedig általában kliens oldalon (ezek alól kivétel pl. a webről indított agentek).
Ennek ellenére készíthetünk olyan agentet is, mely egyszerűen csak kiválaszt az adatbázisból egy másik agentet, és azt úgy indítja el, hogy az a szerveren fusson. Ez a lehetőség nagy segítséget jelenthet az adminisztrátoroknak és a fejlesztőknek.

Bővebben a 3. fejezetben olvashatunk a programozásról, illetve az agentekről.

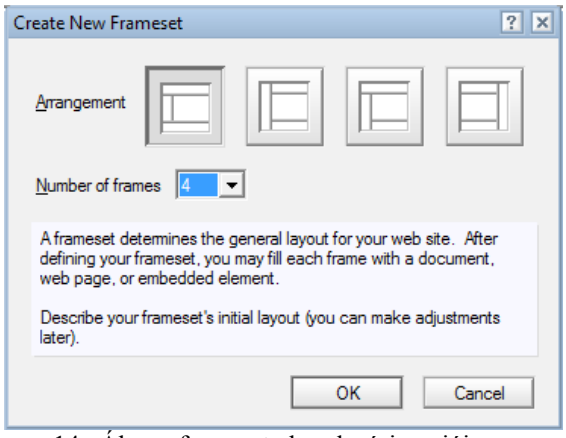
2.5 Frameset

A frameset design elem a html fejlesztésből lehet ismerős sokunk számára. Egy frameset, vagy gyűjtemény, több frameből áll, amik az alkalmazás webes vagy lotusos megjelenésért felelhetnek. Egy frame természetesen tovább bontható további framekre, de megjeleníthetnek formokat, könyvtárakat, oldalakat (lotusost és webest is), outlinet (navigációs menühöz).

A létrehozáshoz válasszuk a design pane-ről a Frameset → New Frameset opciót. Ezután szabadon adhatjuk hozzá az újabb frameket, vagy az új frameset választása után megjelenő ablakban kiválaszthatunk egy nekünk tetsző elrendezést, az általunk megadott számú frame-mel.

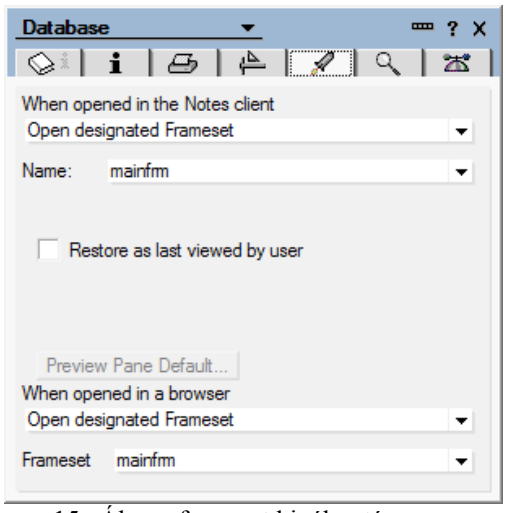
Megjelenítési lehetőségként ugyanazokkal az opciókkal rendelkezünk, mint a weben: újraméretezhető framek, lapozhatóság. Az egyes frameknek adhatunk nevet, és így egymás között hivatkozhatjuk őket, megadhatjuk, hogy egy adott frameben kiválasztott dokumentum melyik másikban nyíljon meg.

A gyakorlatban a legelterjedtebb a 4 frames felosztás, ahol a baloldalon 3 kisebb framet találunk, jobb oldalt pedig egy nagyobbat, ami megjeleníti a nézeteket, formokat. A 3 kisebb közül az egyik menüként funkcionál, a másik kettő pedig információs panel (pl. az adatbázis neve – melyik szerveren található).



14. Ábra a frameset elrendezési opciói

Ahhoz, hogy az adatbázisunk megnyitásakor az általunk létrehozott frameset tartalma nyíljon meg, az adatbázis tulajdonságai közül válasszuk ki a When opened in notes client részből az Open designated frameset opciót, és adjuk meg a frameset nevét.



15. Ábra a frameset kiválasztása

2.6 Outline

Az outlineok segítségével elegáns menüket készíthetünk alkalmazásunknak. Előnyük a programozható, hierarchikus megjelenés. Segítségével csoportokba rendezhetjük a menüpontokat, amik segítségével nézeteket, könyvtárakat, dokumentumokat, agenteket nyithatunk és hívhatunk meg.

Az egyes menüpontokhoz társíthatunk képeket is (pl. többszintű, hierarchikus menüpont mellé egy saját „+” ikont).

Létrehozásához válasszuk design pane-ről a Shared Code → Outlines menüpontot, majd a create new-t.

Létrehozás után egy formon, vagy egy oldalon kell beillesztenünk (Create → Embedded element → Outline), hogy utána később a framesethez csatolhassuk.

Az outline framesetbe illesztése után kiválaszthatjuk, hogy az egyes menüpontok által megnyitott elemek (dokumentumok, nézetek, oldalak), melyik framesetben nyíljanak meg.

A megjelenítési stílus könnyen változtatható, a felhasznált képeket és stílusokat az adatbázis design-jának shared resources részében helyezhetjük el, így kinézet változtatáshoz elég csak ezeken módosítani.

Az outlineok teljesen programozhatóak, így megtehetjük azt is, hogy általunk megadott logika és elvek alapján renderelődik az alkalmazásban (pl. kliens vagy webes oldalon vagyunk).

2.7 Folder

Azaz könyvtár, a már megszokott informatikai jelentésében. Segítségével az egyes, általunk sokat használt dokumentumokat rendezhetjük. A foldert létrehozhatjuk egy adatbázison belül is, mint privát könyvtárat (ehhez elegendő egy Lotus Notes kliens is), vagy, mint megosztott mappát (ekkor vagy designeri jogokkal kell rendelkeznünk az adatbázis felett, vagy Domino Designert kell alkalmaznunk.

Egy folderbe dokumentumokat többféleképpen is helyezhetünk, egyrészt a klasszikus drag & drop módszerrel, másrészről pedig programozottan is (agent vagy action segítségével).

A folderek kinézete és tartalma ugyanúgy szerkeszthető, mint egy nézeté, oszlopokat adhatunk hozzá és ezzel szabályozhatjuk a megjelenített adatokat.

Nagy előnye még a könyvtáraknak, hogy az átlag felhasználók is létrehozhatják őket, így kicsit testre szabhatják magunknak az alkalmazást, összegyűjthetik a napi munkát.

3. Programozás

A Lotus Notes alkalmazás-fejlesztés során több programozási nyelvet is használhatunk. A kódokat 3 főbb helyen tároljuk általában. Az agentekben, a script librarykban és a formokon.

Ahhoz, hogy eldönthessük, hogy melyik programozási nyelvet szeretnénk használni a lehetőségek közül (egyszerű parancsok, formula, lotus script, javascript, java), elsődlegesen meg kell vizsgálni a futási eseményeket, azaz milyen eseményre fusson az agent. Ez lehet gombnyomásra, form megnyitása előtt és bezárása után, vagy akár időzített is.

Mivel elég sok lehetőséget kínál nekünk a Lotus Notes, ezért szükségünk lehet egy kis segítségre, hogy milyen programozási nyelvet használjunk / használhatunk az adott feladathoz.

Az alábbi kérdőív a [4] –ből származik:

16. Formot tervezel, több mezővel, melyeknek szüksége van kezdeti értékre, amiket akár a felhasználó is változtathat. Más mezőknek számított értéke van, néhány elem pedig rejtett a felhasználók elől.
    Válasz: Formula
17. Az alkalmazásnak dialógus ablakokra van szüksége.
    Válasz: Formula, Lotus script, Javascript
18. A programnak szüksége van arra, hogy más dokumentumokhoz is hozzáférjen azon kívül, amit éppen nyitva tartunk.
    Válasz: Java, Formula, Lotus script
19. Egy olyan alkalmazás tervezel, melyhez nagymértékű interakcióra van szükség a felhasználó felé. A fejlesztő kevesebb programozói tapasztalattal rendelkezik.
    Válasz: Formula, Javascript
20. Az alkalmazásnak nagyfokú interakcióra van szükséges a felhasználó felé, mező validálásra, és kurzor pozicionálásra.
    Válasz: Formula, Javascript, Lotus script
21. A tervezés alatt álló összetett folyamat, ciklusokat és feltételes elágazásokat tartalmaz. Nem lehet tudni, hogy ez a folyamat hányszor fog lefutni.
    Válasz: Formula, Javascript, Lotus script
22. Egy alkalmazás teljesítménye nem olyan, mint amit a felhasználók elvárnának, igaz, nagyon felhasználó barát. Alapos elemzés után úgy tűnik, hogy a formulák közül több lassan fut, milyen programozási nyelvekkel lehetne növelni a sebességet?
    Válasz: Lotus script, Javascript

Ehhez a listához még azt tenném hozzá, hogy szerver oldali agentek esetén érdemes elgondolkodni az alábbiakon:

1. Az agent kódja leszűkíthető-e egy nézetben található dokumentumokra, és egyszerűbb műveleteket akarunk-e végrehajtani (tehát csak épp az adott nézet dokumentumain akarunk operálni)?
   Ha igen, akkor használjunk **Formulát.**
2. Amennyiben nem:
   2.1. Ha egyszerűbb mégis az agent kódja, és tudjuk, hogy nem futna sok dokumentumra (max 1-2000 db.), akkor nyugodtan használjunk **Lotus script**et,

ami könnyen fejleszthető, és ellenőrizhető a futása, de nagyobb mennyiségű dokumentumra a sebessége nem elfogadható.

2.2. Nagy mennyiségű adat komplex feldolgozására használjunk **Java** agentet. Előnye a Lotus scripthez képest a nagyobb sebesség (tapasztalatok alapján akár 5-10-szeren is gyorsabb lehet, mint ugyanaz a feladat Lotus scriptben megvalósítva). Hátránya, hogy több fejlesztési időt igényel (lassabb a tesztelése és a hibaellenőrzése), de **6.1.** fejezet sokat segíthet nekünk ilyen téren.

3.1 Az egyszerű parancsok

Az egyszerű parancsokat (Simple Action) agentekben használhatjuk. Ez igazándiból nem programozást jelent, hiszen e parancsok nem sok mindenre használhatóak, olyanokra képesek, hogy elindítanak egy agentet, vagy küldenek egy levelet.

Gyakorlati hasznuk nem igazán van e parancsoknak, talán annyi, hogy a felhasználók ezeket könnyen elsajátíthatják, és az egyes adatbázisokban egyszerűbb akciókat készíthetnek maguknak.

Érdemes viszont figyelni ezeknél a parancsoknál, hogy az agent tulajdonságainál mit adunk meg futási célpontnak. Egy korábbi tapasztalat: Volt egy időzített egyszerű parancs, aminek a feladata az volt, hogy indítson el egy másik agentet, majd utána küldjön 4 levelet, ha az lefutott. Viszont a futási céljának az volt megadva, hogy az adatbázis minden dokumentumára fusson (kb. 150000 db.), ami azt eredményezte, hogy egy éjszaka alatt 600000 dokumentummal telítődött a postaládája, melyek miatt az túllépte a 200MB-os limitet (4GB lett), és ehhez kapcsolódóan a csak levelezésre beállított server is „meghalt”, nem számítva a további levelező szervereket, melyeknek a routerei megtelítődtek a hatalmas mennyiségű levéllel, amiket csak kézzel lehetett kiüríteni.

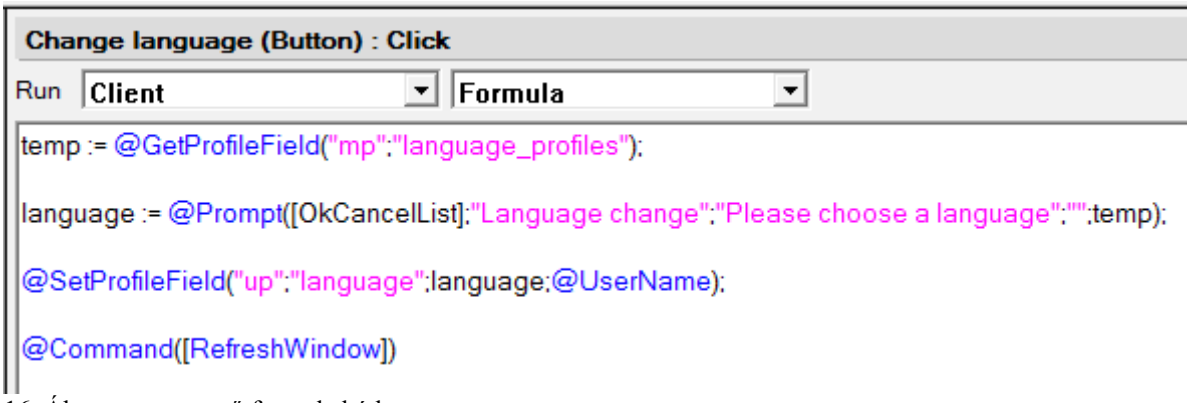
Ezen parancsokkal történt rossz tapasztalataim miatt nem igazán ajánlom senkinek sem a használatukat, mivel futásuk nem ellenőrizhető, illetve egyszerűségük miatt sokkal könnyebb bennük hibát elejteni, ami nagyon extrém eseményeket eredményezhet.

3.2 Formula nyelv

A formula nyelv szintaktikájában, megjelenésében és utasításaiban nagyon hasonlít a Lisp-re.

Előnye a könnyen megtanulhatóság, és a gyors végrehajtás (a domino formula motorja sokkal gyorsabban képes végrehajtani a programkódokat, mint a lotus script motor). Hátránya az, hogy funkcionalitásban és tudásban nem éri el a Lotus Scriptet, és nem lehet benne megfelelően komplex alkalmazás logikát kódolni.

Alkalmazási területeinek leginkább a felhasználói interakciókat (gombok, dialógus ablakok) lehet megemlíteni, illetve az egyes formokon a számított értékeket és a mezők rejtését.



16. Ábra egy egyszerű formula kód

További előnye még a formuláknak, hogy nagyon beszédesek, illetve sokrétűek. Az egyszerű interakciók könnyen és gyorsan programozhatóak. Nagyon sok feladatot csak formulával tudunk megoldani, ilyen pl. a számított értékek a formokon, vagy egy legördülő lista elemeinek a dinamikus előállítása, esetleg nézetek oszlop értékeinek megadása.

Mint már említettem, a formula nyelv nagyon hasonlít a Lisp-hez. Ez a szintaxison kívül abban is megnyilvánul, hogy a legtöbb formula listát is kezel a szimpla értékek mellett. Így adhatunk dinamikusan egy legördülő, vagy checkbox listának dinamikusan választási lehetőségeket.

Egyszerűsége miatt jól jöhet az is, hogy a formulákat futtathatjuk lotus script vagy java kódból is (session.evaluate() segítségével).

Nagy hátrány viszont, hogy nem tudunk komplexebb logikát kódolni formula nyelvvel, illetve nem tudunk készíteni egyszerűbb függvényeket sem, ami eléggé megnehezíti a dolgunkat, ha formulát kell alkalmaznunk.

Egy alkalmazás létrehozásánál ajánlatos shared action-ök használata, melyek gyakori műveleteket takarhatnak, így csak egyszer kell megírnunk azokat, később pedig csak az adott design elembe (pl. egy formba) beleilleszteni. Ezen shared action-ök leggyakoribb megvalósítása a formula nyelvben történik, mert 1-1 egyszerűbb parancshoz csak 1 sornyi programkódra van szükségünk. (pl. mentés: @Command([FileSave]);).

3.3 Lotus script

A lotus script a Domino alkalmazásfejlesztés egyik legnagyobb tudású és használhatóságú eleme, nagyon sok esetben csak e programnyelv használható. Ilyen helyek pl. a form események is.

Nagy előnye, hogy segítségével minden felmerülő probléma, legyen az háttér folyamat, vagy felhasználói interakció, programozható. Mindemellett a lotus script kódok gyorsan elkészíthetőek és tesztelhetőek, ami sok esetben jelenthet időmegtakarítást.

Nagy hátránya viszont a már korábban is említett lotus script motor, vagy interpreter, ami magát a végrehajtást nagyon lelassítja. Összehasonlításképpen egy érdekes teszt eredményét osztanám meg, melynek során Oracle adatbázis-kezelő környezetből importáltunk adatokat. Ugyanazon táblából 1200 sornyi adat importálása Java kód számára 5 és 30 másodperc közötti időt emésztett fel (hálózati kihasználtságtól függően), Lotus script esetén kb. 2-3 percnyi időt.

A lotus script objektum orientált programozási nyelv, ami egy egyszerűbb osztály hierarchiával rendelkezik. Ezt meg lehet tekinteni a súgóban, vagy az IBM hivatalos leírásaiban. A legfontosabb osztályok listáját le lehet szűkíteni az alábbiakra:

- NotesSession: ez a felhasználó aktuális munkafolyamata, ezen keresztül érjük el a NotesDatabase objektumot
- NotesDatabase: ezen objektumon keresztül érjük el az adatbázisunkat, az egyes design elemeket (pl. agentek, nézetek)
- NotesView: egy, az adatbázisban található nézetet használhatunk rajta keresztül. Feldolgozhatjuk azt, módosíthatjuk (pl. a szelekciós formuláját), elérhetjük a benne szereplő dokumentumokat
- NotesDocument: egy konkrét dokumentum példányt érhetünk el vele, vagy a NotesDatabase-n (UNID alapján), vagy NotesView-n (a nézet n.-edik eleme), vagy egy másik NotesDocument-en (hierarchikus kapcsolat) keresztül. Mivel a Lotus Notes rendszer központjában a dokumentumok állnak, így érthető, hogy ez lesz a leggyakrabban használt osztály.
- NotesItem: a dokumentum egy item-jét érhetjük el vele, azaz 1 mezőt, annak értékét és tulajdonságait.

Lotus script esetén korábbi tapasztalatom, hogy előfordulhatnak hibás, gyári függvények, melyek a programozó munkáját megkeseríthetik. Ilyen függvényekkel én a 6.5-ös verzióban többel is találkoztam, ezek leginkább a dátumkezelő függvények voltak. A hiba elsődlegesen a dátum és idő konverziós függvényekben jelentkezett (időzónák), a különböző időzónák közötti váltásoknál (ez nyilván egy multinacionális cég környezetében jelent problémát). Ilyenkor nyilván csak kerülő problémával oldható meg a probléma, ami sok plusz munkát és gondolkodást jelent a fejlesztőnek.
Hangsúlyozom, hogy Lotus Notes 6.5-ben találkoztam ezen hibákkal, azóta nem volt alkalmam ellenőrizni ezen függvények működését. Illetve nem sok függvényről van szó, összesen talán 2-3-ról, de én mindenképpen zavarónak találtam.

3.4 Java

A java, mint programozási nyelv eléggé ismert és népszerű, így fordulhatott elő az, hogy a Lotus Notes 5.0-ás verziójába bekerült támogatott programozási platformként.

Fontos tudni, hogy milyen lehetőségekkel számolhatunk Java terén. Lotus Notes 5.0, 6.0 és 6.5 esetén csak a Java 1.3 elemeit használhatjuk, Lotus Notes 7.0 és 8.0 esetén pedig az 1.4-et, ami ha belegondolunk nagyfokú korlátokat jelent.

Mindezek ellenére én határozottan ajánlom a Java használatát Domino-s alkalmazásokban, mivel szerver oldalon sokkal hatékonyabb és gyorsabb minden téren, mint a többi programozási nyelv a Lotus Notes platformon.

A szabvány Java SE osztályok mellé (nem számítva a grafikus osztályokat) a Lotus Notes által nyújtott osztályokat használhatjuk. Ezen osztályok gyakorlatilag ugyanazok, mint Lotus script környezetben, amennyiben az elnevezésüket nem nézzük (de nem tartalmazza a UI osztályokat).

Az egyes osztályok elnevezése annyiban különbözik a lotus scriptes elnevezésektől, hogy azok mind Notes- előtaggal kezdődnek. Így egyszerűen egyeztethetőek egymással az osztályok, illetve a kódok is könnyen átírhatóak egyik programozási nyelvről a másikra.

Amennyiben egy agent kódját egy java fejlesztői környezetben (eclipse, netbeans) szeretnénk megírni (nyilván mert átláthatóbb, mint a designer szerkesztője, és egyéb előnyökkel is rendelkezik), úgy a létrehozott javas project-hez adjuk hozzá külső library-ként a Notes.jar-t, amit a Lotus Notes telepítési könyvtárunkban találunk (esetleg szükségünk lehet még az ncso.jar és ncsow.jar fileokra, ha külső alkalmazásból szeretnénk használni Dominos adatbázist).

A kész java kódot kétféleképpen helyezhetjük el az adatbázisunkban. Egyrészt bemásolhatjuk a kódot (copy-paste), másrészt, ha programozási nyelvként imported java-t választunk, akkor a kód helyett csak a lefordított .class filet szükséges kitallóznunk, így nyilván elrejthetjük az agent kódját mások elől.

Mivel a java sokkal komolyabb programozási nyelv, mint a Lotus script, ezért az objektum orientáltságot is jobban figyelembe kell venni, amikor java agentet készítünk. Azok az adattagok, tulajdonságok, melyeket Lotus script alatt csak egyszerű osztály tulajdonságként hivatkoztunk, java esetében mindet .get vagy .is vagy .has metódusokkal tudunk csak elérni.

Amennyiben webre fejlesztünk, használhatunk .jsp technológiát is, előre definiált domino-s tagekkel. Ez nagy könnyebbséget jelenthet számunkra, amennyiben webes alkalmazásokat és szolgáltatásokat készítünk. Használhatunk a domino szerver mellett egy külön alkalmazás szervert is. Elsődlegesen a Webspheret, mint szintén IBM terméket ajánlja a Lotus Notes, de más alkalmazás szervernek sem lesz gondja vele. Bővebb információt az [1]-ben találunk.

3.5 Libraryk

A libraryk, vagy kódkönyvtárak mindenkinek ismerősen hangzanak, más programozási környezetekben is találkozhattunk már velük.

Libraryt 3 programozási nyelven készíthetünk: Lotus script, java, javascript.

Lotus script kód esetén a (Declarations) részből hivatkozhatjuk a már kész kódkönyvtárat a %Include '*library_neve*' direktíva segítségével.
Java kódból a már megszokott import *csomag_neve* segítségével.
Javascript kód könyvtárat nyilván olyan design elemeken tudunk megjeleníteni, melyeket elsődlegesen weben fogunk használni. Ehhez válasszuk ki a design elem form tulajdonságok és események (info pane) listájából a JS Header-t, majd a Create menüből az Insert Resource-t választva illesszük be a már létező libraryt.

A libraryk használatát csak ajánlani tudom, hátrányai pedig egyáltalán nem léteznek. Elsődlegesen megkönnyíti a munkánkat, másrészt az egyes kód könyvtárakat örököltethetjük más adatbázisokból is, így rengeteg fölösleges munkát spórolhatunk meg magunknak. (Öröklés esetén, ha engedjük, akkor a későbbiekben, ha módosulnak a másik adatbázisban az abból örökölt a könyvtár, akkor a miénkben is frissülne, így egyetlen hibajavítással több adatbázisbeli hibát javíthatunk).

Tapasztalataim szerint, az előnyei ellenére sok helyen nem, vagy alig használják a libraryket, és nagy általánosságban csak a Lotus script, esetleg javascript opciókat alkalmazzák.

Habár hátrányokat nem tudok felsorolni a libraryk használatával kapcsolatban, pár hiányosságot viszont igen:

- nem lehet formula libraryt készíteni
- egy java nyelvű könyvtárból nem tudok Lotus scriptes vagy javascriptes függvényt hivatkozni

4. Kapcsolódási lehetőségek más adatbázis-kezelőkkel

4.1 LEI

Az LEI a Lotus Enterprise Integrator rövidítése. Jelentősége, hogy nem szükséges hozzá programozói tudás, az egészet egy adatbázison keresztül (LEI Admin) tudjuk konfigurálni, egyszerű műveletek és kapcsolatok definiálásával.

A már definiált kapcsolatok újrahasznosíthatóak és a létrehozott feladatokban felhasználhatóak.

Előnye, mint említettem, hogy nem kell hozzá programozói tudás, egy „mezei” adminisztrátor is képes ellátni a konfigurációs feladatokat. Hátránya az egyszerűségéből adódik, egyrészt nincs semmiféle irányításunk a folyamat felett, másrészt pedig a háttér folyamatok LSX segítéségével, Lotus Scriptben vannak kódolva, így a sebesség igen sarkalatos pontja.

Fontosnak tartom megemlíteni itt egy korábbi tapasztalatomat, amikor is egy ékezetes betűket is tartalmazó Oracle alapú táblázatból importált adatokat egy LEI folyamat, bizonyos karakterek esetén ?-ek jelentek meg a valódi karakter helyett. Ezen probléma korrekt megoldása végül Java környezetben, jdbc használatával volt kivitelezhető.

Konfigurálása egy Lotus Notes-os adatbázison keresztül történhet, amit LEI adminnak hívnak. Ajánlatos ezen adatbázist részletesebben átvizsgálni, mivel az adatbázis-kapcsolatokon kívül más jól használható funkciói is vannak.

4.2 LSX

Az LSX interfészt az IBM készíti el a különböző adatbázis-kezelő rendszerekhez, jelentése: Lotus Script Extensions, azaz egy kiegészítés a Lotus Scripthez.

Az LSX-et lotus script környezetből használhatjuk, előnye az egyszerű programozhatóság, hátránya a már korábban is említett lassabb végrehajtási sebesség, illetve az, hogy a különböző adatbázis-kezelőkhöz való kapcsolódás esetén a kódon változtatni kell (minimálisan).

Használata ajánlatos az egyszerűbb, illetve kisebb hálózati erőforrást igénylő feladatok programozására.

Megjegyezném még, hogy az LEI kapcsolatok az LSX interfészt használják végrehajtáshoz.

Segítségével az alábbi adatbázis-kezelőkhöz kapcsolódhatunk:

- Lotus Notes
- IBM DB2
- File rendszer
- ODBC interfészek
- Oracle
- OLE DB
- Sybase

Egyik gyakorlati hátrányának említeném még, hogy sokan nem ismerik, vagy nem használják, így mások számára nehézkes lehet az LSX használatával írt kódok módosítása, javítása, így azt tudnám tanácsolni, hogy igyekezzünk egyszerűbb és kevés adatfeldolgozással járó feladatokra használni.

4.3 JDBC

A jdbc egy interfészt jelent java fejlesztési környezetben, melyhez a konkrét implementációt és a vezérlőket az egyes adatbázis-kezelők készítőinek kötelessége elkészíteni. Előnye az egységesség, azaz a különböző adatbázisokhoz való kapcsolódás esetén a kódon nem kell változtatni, csak a vezérlőt kell lecserélni, továbbá a már korábban említett nagyobb végrehajtási sebesség.

Egyetlen hátrányának talán azt tudnám megemlíteni, hogy kódolása nehezebb, mint az LSX-é. Cserébe viszont elméletben létezik olyan, hogy ANSI SQL, így a jdbc-n keresztül elért adatbázisok típusa nem lenne szabad, hogy befolyásolja a kódunkat.

Ez a gyakorlatban általában ott bukik meg, hogy ha valaki egy kiadott sql utasításban limitálni akarja a visszakapott sorok számát (Oracle alatt rownum, Mysql alatt limit utasítás).

Viszont érezhető, hogy jdbc segítségével gyakorlatilag minden olyan adatbáziskezelő-rendszerhez tudunk csatlakozni, melyeknek létezik saját jdbc drivere.

A gyakorlatban viszont be kell látni, hogy a Lotus Notes inkább nagyvállalati platform, így ahol alkalmazzák, nagy valószínűséggel a másik adatbáziskezelő-rendszer Oracle, DB2 vagy esetleg SAP lesz.

4.4 Web szolgáltatások

Nem tartozik szorosan ezen fejezet témájához ez az alfejezet. A jdbc kapcsán el lehet gondolkozni, hogy addig nagyon egyszerű a dolgunk, amíg Lotus Notesból akarunk csatlakozni más rendszerekhez, de mi a helyzet akkor, ha ezt fordított irányban kell elérnünk?

Erre a legegyszerűbb megoldás a web szolgáltatások használata lesz, és azokat kell megfelelően felparamétereznünk, hogy más adatbázis-kezelőket kiszolgálhassunk. Ez viszont erősen alkalmazás specifikus lesz, nagyon jól át kell gondolnunk, hogy milyen lehetséges műveletekkel látunk el egy web szolgáltatást.

Felmerülhet, hogy vajon tudunk-e hívni más rendszerekből web szolgáltatást? Igen, tudunk, mind a Lotus script, mint a java rendelkezik ehhez szükséges osztályokkal és programozhatósággal.

Megjegyezném még, hogy más rendszerekből Lotus Notes alkalmazásokat elérni nem sok lehetőségünk van a web szolgáltatásokon kívül, de van még lehetőség. Megoldhatjuk file rendszeren keresztüli kommunikációval, amennyiben nem fontos, hogy az adatok azonnal elérhetőek legyenek (ez nyilván elég morbid megoldás). Az utolsó lehetőségünk pedig ismételten a Java. Lehetséges olyan kód írása külső alkalmazásokban, melyek egy felhasználói .id fileon keresztül be tudnak jelentkezni az adatbázisba (ehhez szükség lesz a Notes.jar, NCSO.jar fileokra), és onnantól a Lotus Notes osztályain keresztül tud kommunikálni a két rendszer.

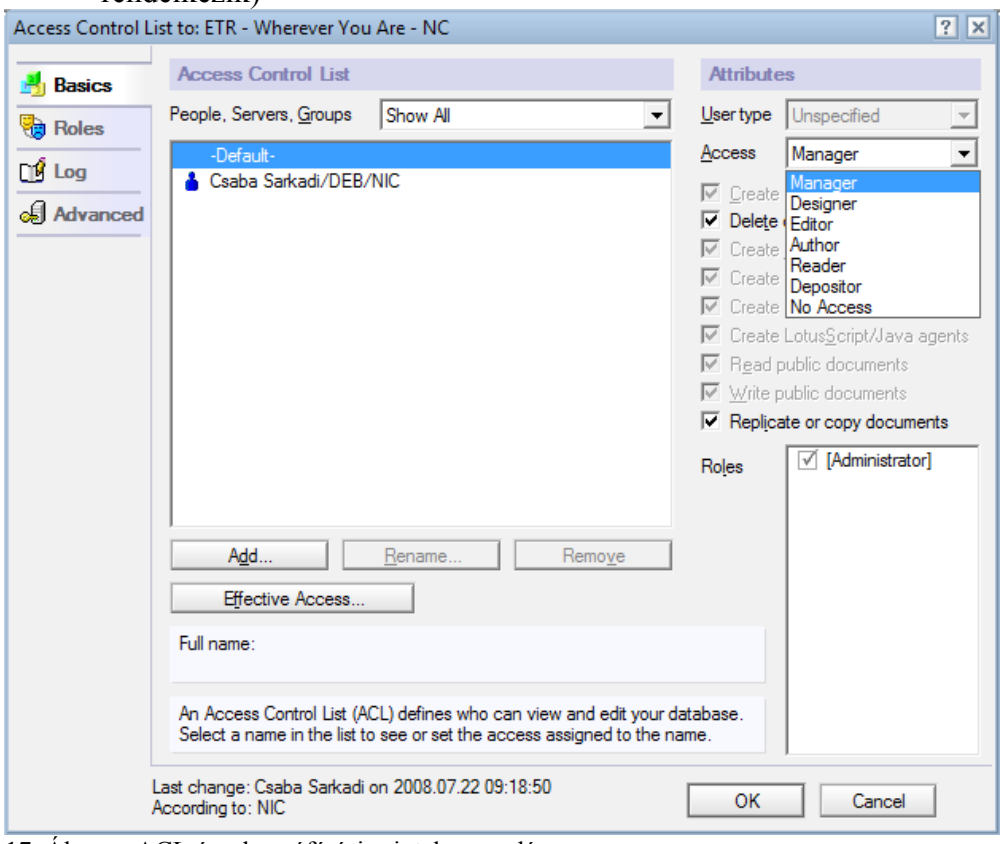
5. Biztonság

5.1 ACL

Az ACL segítségével szabályozhatjuk az egyes adatbázisokhoz a felhasználók hozzáférési szintjét. Megtekintéséhez válasszuk a Lotus Notes vagy Domino Designer kliensekből a File → Database → Access Control menüpontot.

Itt az alábbiakra van lehetőségünk:

- felhasználó hozzáadása a listához
- felhasználói csoport hozzáadása a listához
- felhasználói hozzáférési szint módosítása (lásd következő fejezet)
- felhasználói szerepkörök módosítása (User role)
- felhasználói jogosultságok megtekintése
- megtekinthetjük az adatbázis logját
- adminisztrációs servert adhatunk meg (erre a replikával rendelkező adatbázisok esetén van szükség, bővebb információt a Designer súgójában találhatunk)
- replikák esetén lehetőségünk van arra is, hogy az egyes replikák ACL-ét egymással megegyezővé tegyük
- internetes használat és bejelentkezés esetén a felhasználó maximum mekkora hozzáférési szinttel rendelkezzen az adatbázishoz (pl. egy adminisztrátor kliens oldalon rendelkezik maximális jogosultsággal, de elképzelhető, hogy a weben keresztül bejelentkezve az adatbázis felületére már csak mezei felhasználói jogokkal rendelkezik)



17. Ábra az ACL és a hozzáférési szintek megadása

5.2 Hozzáférési szintek

No Access = Az adott felhasználónak nincs joga az adatbázis megnyitásához.

Depositor = A felhasználó létrehozhat dokumentumokat, illetve szabályozható, hogy a már létező, publikus dokumentumokat olvashatja, szerkesztheti, vagy akár replikálhatja-e.

Reader = A felhasználónak olvasási joga van a publikus dokumentumokhoz. Ezen felül opcionálisan adhatunk lehetőséget privát agentek, folderek és nézetek létrehozására.
De nincs joga dokumentumok létrehozására.

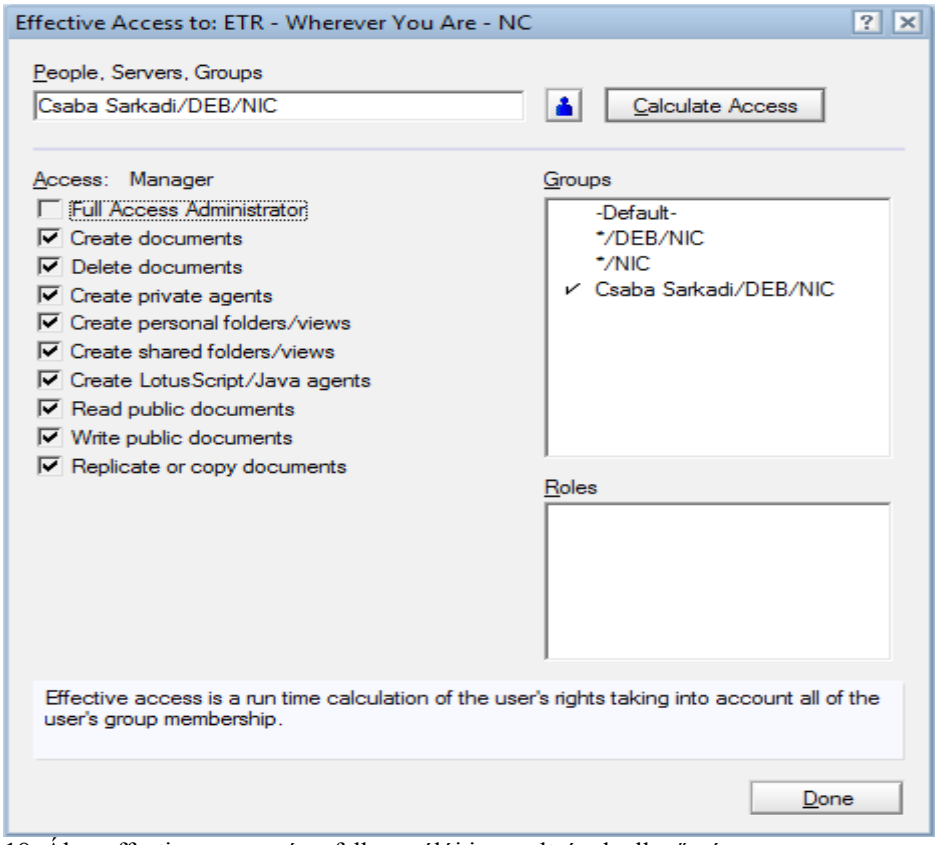
Author = A readertől annyival van több joga, hogy lehetőségként kaphat dokumentumlétrehozási és törlési jogot.

Editor = Az author jogtól annyiban különbözik, hogy nem visszavonható a dokumentum létrehozási joga, illetve esetleges megosztott nézet / foldert is létrehozhat.

Designer = Hozzáférése van az adatbázis designhoz, az egyedüli visszavonható opciók számára a dokumentumok törlése, illetve a Java/Lotusscript agentek létrehozásának joga.

Manager = Teljes hozzáféréssel rendelkezik az adatbázis felett, szabadon oszthatja és visszavonhatja a felhasználói jogosultságokat.

Egy felhasználó hozzáférési szintjét egyszerűen lekérhetjük az effective access gomb használatával.



18. Ábra effective access, és a felhasználói jogosultságok ellenőrzése

6. Tippek, trükkök, tanácsok

6.1 Java agent debuggolása Eclipseből

A most következő leírásban részletesen szemléltetni fogom, hogy hogyan használhatunk egy Java fejlesztői környezetet (pl Eclipse vagy Rational) agent debuggolására.

Ezt a segítséget a legjobban arra tudjuk kihasználni, hogy láthassuk az agent működése közben bekövetkező folyamatot, illetve az egyes objektumokról a lehető legrészletesebb információkat kapjuk.

A szükséges softwarek:

- Java fejlesztői eszköz (ajánlott Eclipse vagy Rational)
- Domino Designer
- Amennyiben 6.0-ás vagy 6.5-ös Designerrel dolgozunk, szükség van külön telepített 1.3-as java JRE-re (http://java.sun.com/products/archive/j2se/1.3.1_20/index.html)

A softwarek telepítése után el kell döntenünk, hogy az agentet milyen oldalon szeretnénk futtatni, illetve az adatbázist milyen oldalon tároljuk. Az első lehetőség a kliens oldal, ez a kényelmesebb, a második lehetőség a server oldal.

Amennyiben ezt sikerült eldönteni, a megfelelő oldalon az alábbi beállításokra lesz szükségünk a notes.ini fileban:

JavaEnableDebug=1

Ezután tapasztalataim szerint mindenképp szükséges a kliens/server újraindítása.

Ahhoz, hogy még kényelmesebb legyen a művelet, szükségünk lesz egy agentre, aminek összesen annyi a feladata, hogy a java ügynököt elindítja a serveren. Minta kód:

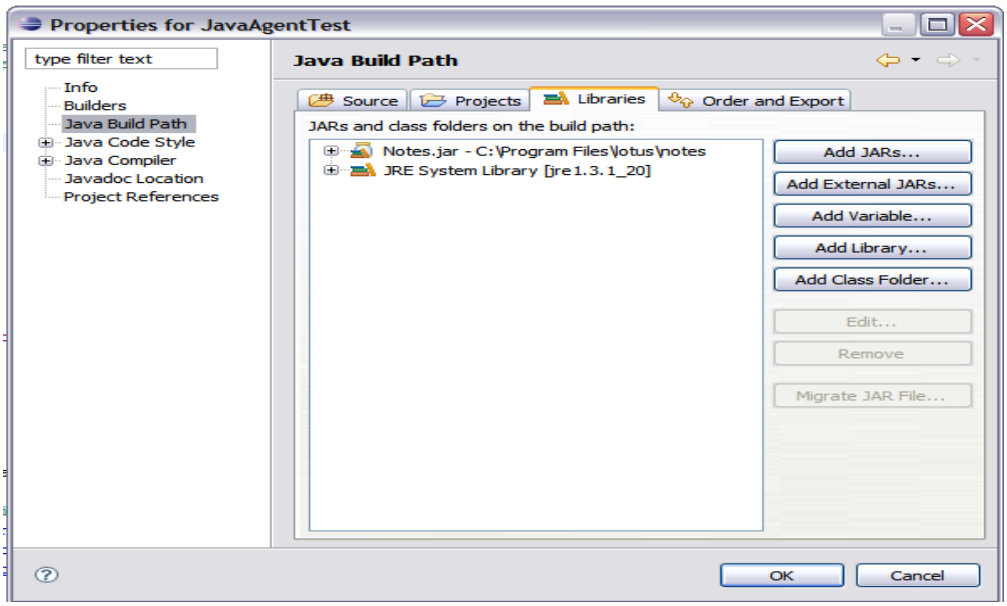
```
Sub Initialize

Dim ns As New NotesSession

Dim db As NotesDatabase

Dim agent As NotesAgent

Set db = ns.CurrentDatabase

Set agent = db.GetAgent("Ezt akarom futtatni")

Call agent.RunOnServer

End Sub
```

A most következő lépéseket a kliens oldali futtatáshoz írom le, de amennyiben server oldalon szeretnénk futtatni az agentet, a különbség csak annyi lesz, hogy a fejlesztői környezetben a localhost helyett a server címét kell majd megadnunk.

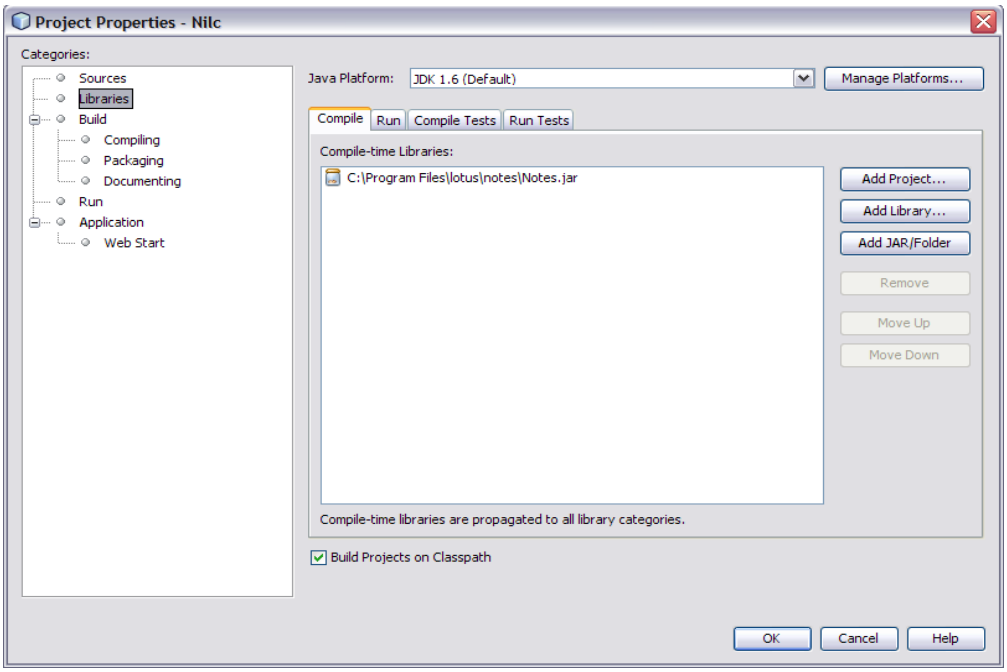
Ezek után nézzük a szükséges lépéseket!

1. A fejlesztői környezet beállítása

Készítsünk egy új projektet. Használjuk az egér jobb gombját a projekt nevén, a tulajdonságai megtekintéséhez.

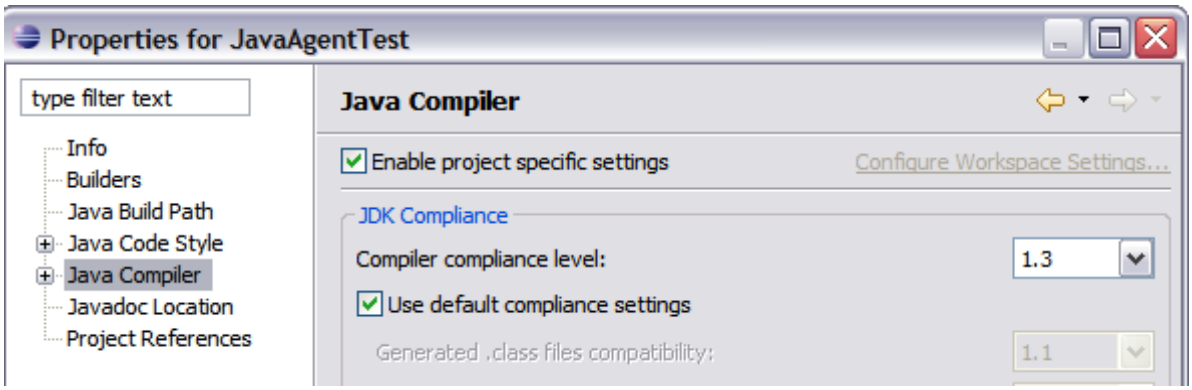


(eclipse)

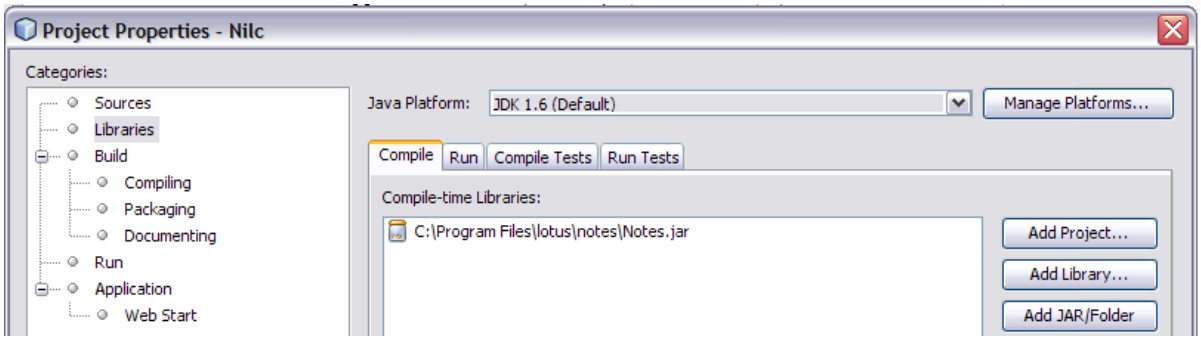


(netbeans)

Most adjuk hozzá a Notes.jar -t a külső kódkönyvtárakhoz. Amennyiben 7.0-ás verzió előtti Designert használunk, állítsuk be a fordítási szinthez a JDK 1.3-at. Ellenkező esetben az 1.4-et.



(eclipse)



(netbeans)

Ezen lépés segítségével a feljesztői környezetünket ugyanúgy használhatjuk, mint a Designer Java felületét.

2. Lépés

A létrehozott projekthez adjuk hozzá az ügynökünk kódját, majd fordítsuk le.

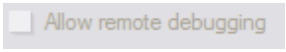
3. Lépés

A lefordított ügynök futtatható verzióját (.class fájl) keressük meg a projekt mappájában.

Térjünk vissza a Domino Designerhez ezután. Hozzunk létre egy ügynököt Importált Java típusúként és használjuk a Reimport class files gombot, és tallózuk ki az előzőleg létrehozott .class filet.

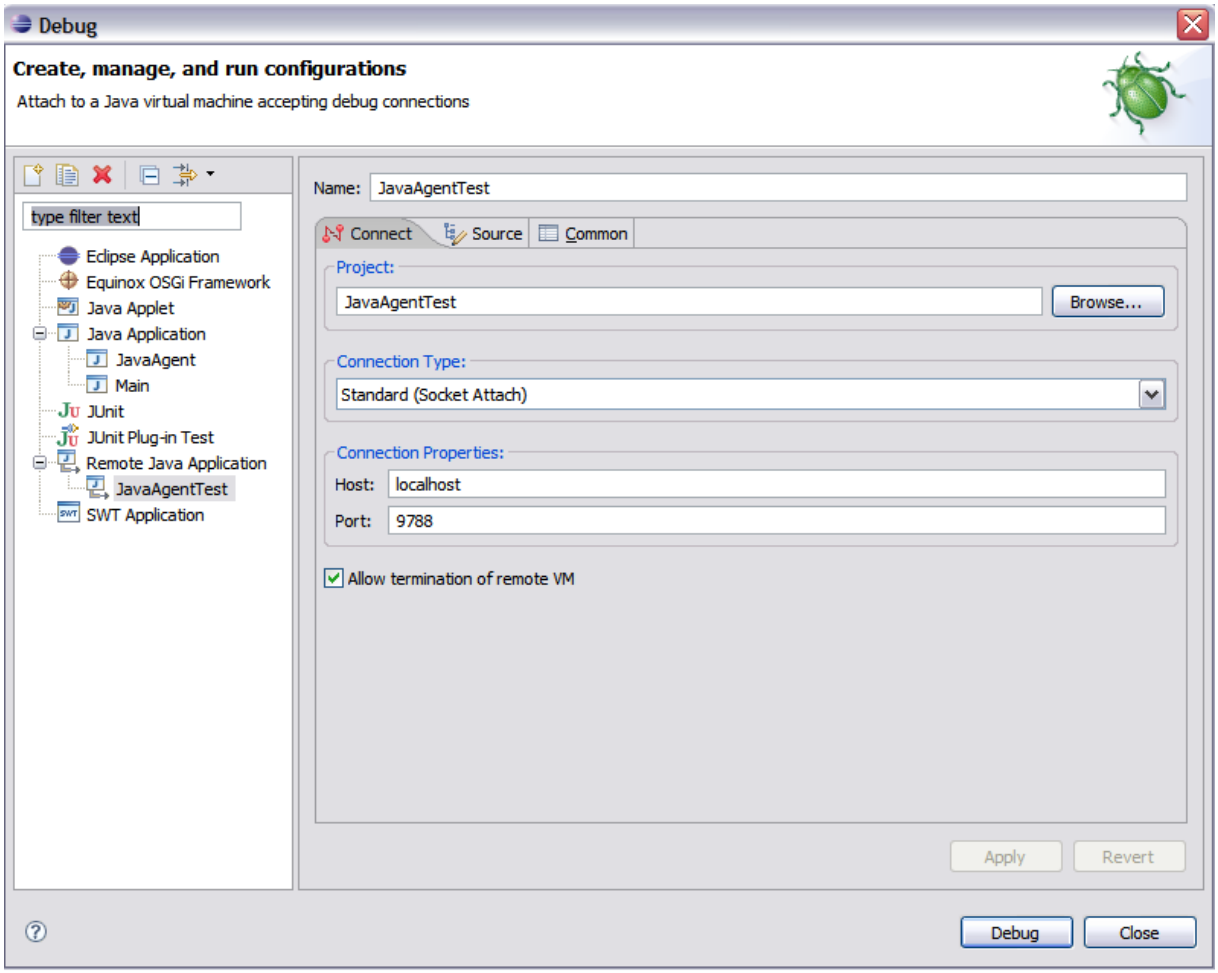
Ezután két lehetőségünk van: beállíthatjuk az ügynököt időzítettre (nem ajánlott), vagy használhatjuk a már előzőleg létrehozott másik ügynököt arra, hogy futtassuk a kódot.

Amennyiben server oldalon dolgozunk, az ügynök tulajdonságainál állítsuk be az

opciót. 

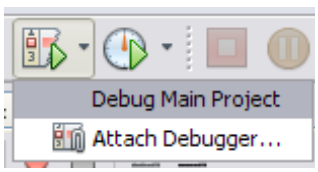
4. Lépés – Futtatás

A projektünkre az egér jobb gombjával kattintva válasszuk a debug as → debug → Remote Java Application opciót. Kattintsunk az Új gombra (az eclipse ikon alatt a jobb felső sarokban), és töltsük ki az alábbiak szerint a mezőket:

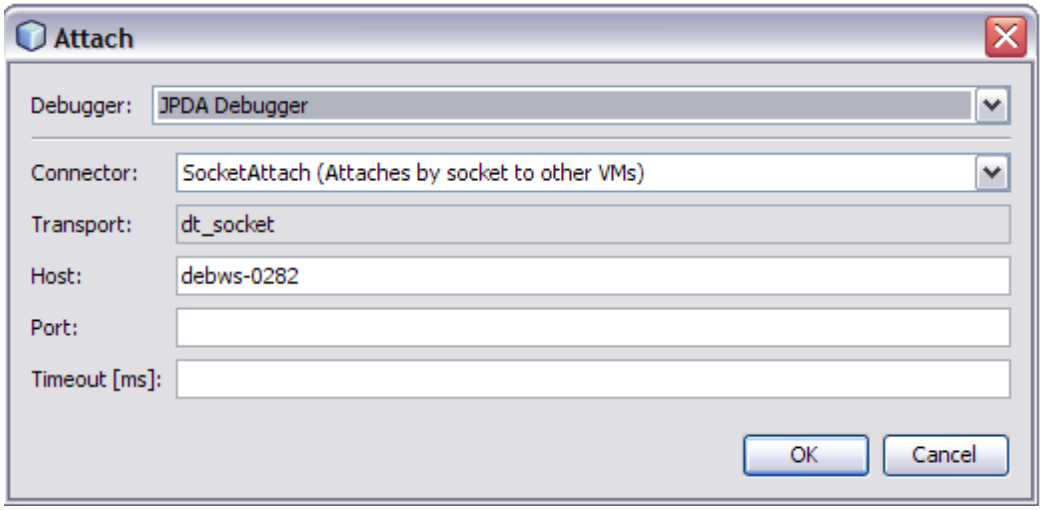


(eclipse)

Válasszuk az Attach Debugger opciót a projekten.

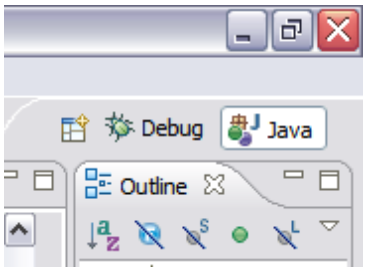


(netbeans)



A hosthoz írjuk be, hogy localhost, a port pedig 9788 legyen.

Ha ezzel megvagyunk, akkor Eclipse alatt válasszuk ki a debug felületet:



Netbeans esetén csak nyomjuk meg az OK gombot.

Ha ezzel kész vagyunk, és elindul az agent (akár mi indítottuk el a második ügynök segítségével, akár az időzítés hatására indult el a serveren), akkor a serveren a kód első soránál meg áll a futása, és innentől a fejlesztői eszköz vezérli a kód futását. Használhatjuk a watch expression/inspect expression, run to line eszközöket is.

Megjegyzések

Ez a bemutató az alábbi cikkek alapján íródott:

http://www.tlcc.com/admin/tips.nsf/0/A4D8A25A0BD48B4685256F5B0058917E

http://www.ibm.com/developerworks/lotus/library/notes-eclipse/

Saját tapasztalataim alapján ez a módszer abban az esetben működik a legegyszerűbben, ha az adatbázisunk kliens oldalon van és Eclipse/Rational fejlesztői eszközt használunk. Ettől függetlenül a leírt metódusnak ugyanúgy kell működnie server oldalon is, mint kliens oldalon.

7. Egy példa alkalmazás tervezése és fejlesztése

Egyetemistaként nagyon sok hibáját és hiányosságát láttuk a tanulmányi rendszereknek, melyek használatukat nehézzé tették. Ezen egyszerű okok vezettek rá arra, hogy vajon milyen elemekből is állhat egy tanulmányi rendszer, illetve milyen belső funkciókra van szükség a működéshez.

Az alkalmazáskor pillanatképeket a 10. fejezetben lehet találni.

7.1 Az alkalmazás által használt formok

Az adattároláshoz különböző formokra lesz szükségünk, ezeket ebben a fejezetben szeretném ismertetni.

7.1.1. A profil típusú formok

Mint korábban, a 2.1.1.-es fejezetben, a profil formokat konstansok, adatbázis és felhasználó függő beállítások tárolására használhatjuk.

A main profile form fogja tárolni az adatbázissal kapcsolatos beállításokat.

A user profile az egyes felhasználókkal kapcsolatos egyéni beállításokat fogja tartalmazni, például milyen nyelvi beállításokat használ az adott felhasználó.

A translation profile az egyes nyelvekhez tartozó címkéket, fordítási elemeket tartalmazza. Az alkalmazás lefordítása egy újabb nyelvre egyszerű, csak a main profilet kell megnyitnunk, és az „Add new language” gombra kattintva már szerkeszthetjük is a címékéket.

Az archive profile az archiválási beállításokhoz szükséges. Az adatbázis tartalmaz egy univerzális archiváló kódot, mely segítségével egy tetszőleges adatbázisból tudunk adatok menteni egy erre a célra szolgáló adatbázisba. Nagy előnye, hogy Java-ban írt kód, így futási sebessége jobb, mint a lotus scriptben írt változatoknak. Maga a kód egy multinacionális vállalat több adatbázisában is működik hosszú ideje, éles rendszereken, probléma és hiba nélkül.

7.1.2. Az effektív adattárolásra szolgáló formok

A konstans adatokon és beállításokon kívül nyilván szeretnénk adatokat is tárolni a rendszerben.

Az adatok készülhetnek egyrészt személyekről (oktatók és hallgatók), másrészt tanszékekről és karokról.

Az egyes karok adatait a Faculty formmal rendelkező dokumentumok fogják tárolni. Kulcsfontosságú információként a nevét, címét, telefonszámát és a dékán nevét szükséges elérhetővé tenni. A létrehozása a Faculties nézetből elérhető, és csak az [Administrator] szerepkörrel rendelkező felhasználók számára.

A tanszékek adatait a Department form segítségével fogja kezelni az alkalmazás. Az egyszerűség kedvéért csak a nevet, telefonszámot, címet és a kar nevét kell megadnunk, amelyiknek a része (ez választható a már létrehozott karok listájából).

Az oktatók adatait az Instructor, a hallgatókét pedig a Student formmal létrehozott dokumentumok fogják tárolni. A tárolt adatok gyakorlatilag megegyeznek: név, azonosító (vagy ismertebb nevén „Neptun kód”), telefonszám, cím, email cím, kar, tanszék.

A tantárgyi adatokat a Subject form segítségével fogja tárolni az alkalmazás. Szükséges a név, leírás, oktató, a kredit szám, és a tantárgy típusának tárolása (gyakorlat, előadás, labor).

A Subject form csak megjelenítési célokat szolgál, azaz ezeket böngészhetik az egyes felhasználók. Ezen formnak a típusa dokumentum.

Az egyes hallgatók által felvett tárgyakat viszont egy másik formmal fogom megvalósítani, ami response típusú. Ezt Class-nak nevezem az adatbázisban. Ez a megvalósítás azért lesz használható, mert így egy dokumentum hierarchia alakulhat ki a hallgató és a felvett tárgyai között. Ennek eredménye, hogy van egy fix kapcsolat, és a hallgatói felvett tárgyai egyszerűen feldolgozhatóak programkódból.

A Class form ugyanazokat a mezőket tartalmazza, mint a Subject form, de kiegészül egy rejtett mezővel, ami a hallgató nevét tartalmazza.

A név mezők mind Names típusúak. Ez a Lotus Notesban egyértelmű azonosítást tesz lehetővé, mivel két egyforma nevű ember nem lehet sem az adatbázisban, sem a tartománykezelőben (LDAP). Amennyiben két egyforma nevű ember is létezne egy adott munkahelyen, úgy azokat meg kell különböztetni, erre a legelterjedtebb módszer a számozás. (Például: Nagy Gábor 2, Nagy Gábor 3 …)

7.2. Kapcsolatok az egyes formok és dokumentumok között

Az egyes formok között nyilván van szükségünk valamilyen kapcsolatra, csak úgy, mint relációs környezetben az egyes sémák közötti relációkra.

Ez a kapcsolat többféle módon is megvalósulhat, a lényeg, hogy az azonosító adat egyedi legyen.

Az alkalmazásban többféle megoldást fogok alkalmazni. Az egyik ilyen, a dokumentum hierarchia, amit korábban már tárgyaltam a 2.1.1., 2.1.3 és az előző alfejezetben.

Konkrét megvalósítása az alkalmazásban a hallgató és a felvett tárgyai közötti hierarchia megvalósítása lesz. A hallgató felvett tárgyai így egyszerűen elérhetőek lesznek program kódból (konkrétabban a NotesDocument.Responses dokumentum tulajdonságon keresztül).

A kapcsolat másik gyakori megvalósítása az UNID-en keresztül történhet, ezt szintén programkódon keresztül használom. Segítségével egy adott azonosítójú dokumentumot azonnal elérünk a NotesDatabase.GetDocumentByUNID metóduson keresztül.

A dokumentumok között kapcsolatot teremthetünk még más technikákkal is. Megoldás lehet a kulcsként használható mezőérték (azaz olyan mező, melynél tudjuk biztosítani, hogy az adott formú dokumentumok közül csak egy rendelkezik azzal az értékkel). Ennek egyik különleges változata a „Names” típusú mező használata. Mint már említettem, az ilyen típusú mezők esetén a rendszer, illetve az adminisztrátorai gondoskodnak az egyedi nevekről, így ez a megoldás minden esetben használható.

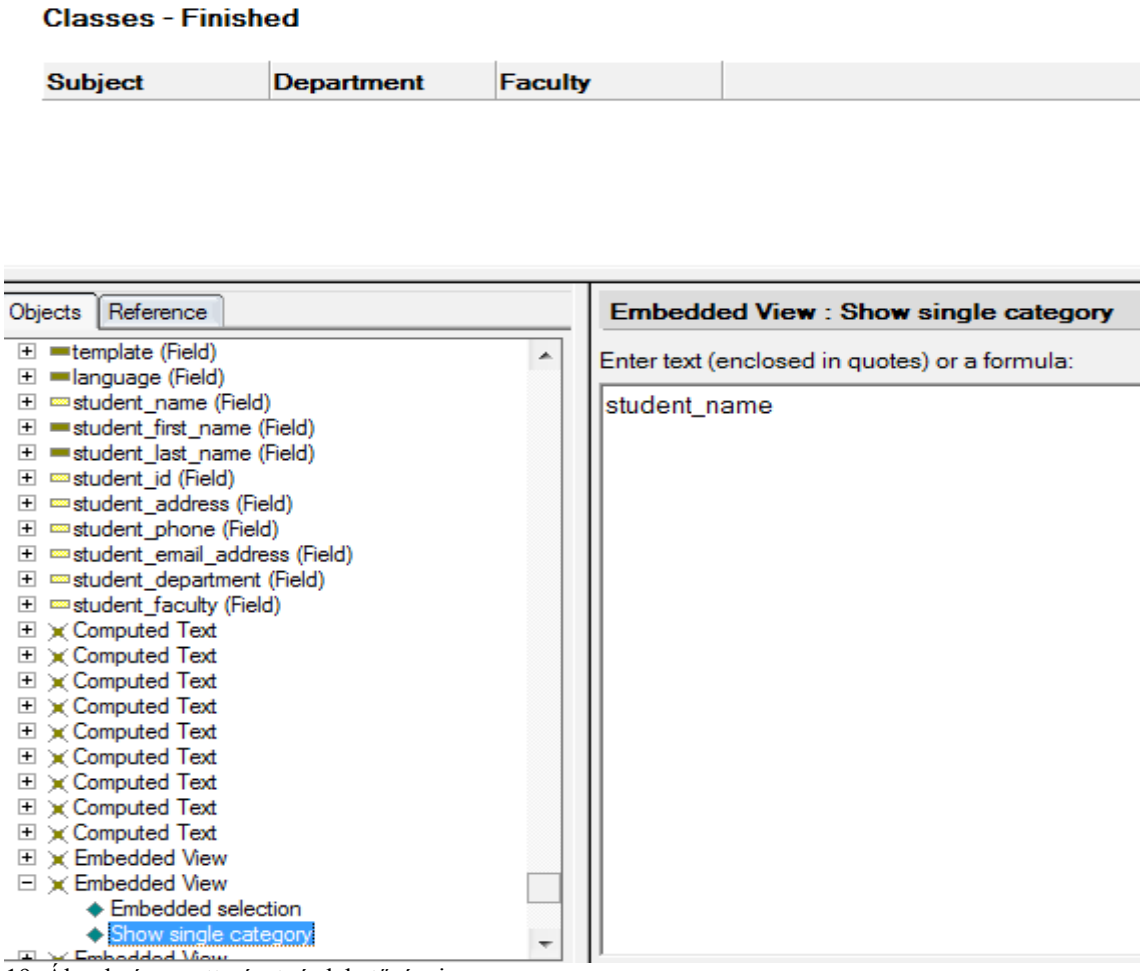
A dokumentumok közti kapcsolatot is többféleképpen tudjuk megjeleníteni. A hierarchikus megoldás esetén a grafikus megoldást a Lotus Notes belső mechanizmusa szolgáltatja számunkra.

A vizuális megjelenítés történhet még beágyazott nézeten keresztül. Ez a megoldás azt jelenti, hogy egy nézet tartalma egy dokumentumra beágyazva jelenik meg. Egy további opcióval ezen tudunk finomítani. Ez az opció pedig a nézet beágyazása után jelenik meg számunkra, és „Show single category” néven találjuk meg.

Ez a megoldás azt fogja eredményezni számunkra, hogy a beágyazott nézetből csak a megadott kulcsú dokumentumok fognak megjelenni, egyfajta szűrést eredményezve a nézeten.

Ahhoz, hogy ez a megoldás működjön, 3 lépésre van szükségünk.

0. A beágyazni kívánt nézetben legyen az első oszlop rejtett, tartalmazza a kulcs mező értékeit, és legyen az oszlop típusa kategorizált.
1. Ágyazzuk be a nézetet a formra.
2. A „Show single category” opciónál adjuk meg a kulcs értéket tartalmazó mező nevét (itt van még lehetőségünk további szűrésre, mivel ez az opció valójában egy formulát vár)



19. Ábra beágyazott nézet, és lehetőségei.

Az előbb felvázolt beágyazott nézet lehetőséget az Instructor és Student formok esetén használom. Így egyszerűen meg lehet jeleníteni, hogy egy adott oktatónak milyen felvehető tárgyai vannak, illetve egy hallgató milyen tárgyakkal rendelkezik (teljesített, felvett, nem teljesített).

7.3 Műveletek

A szakdolgozat keretein belül kevésbé van lehetőség terjengősebb alkalmazást készíteni, nem is ez a cél, így én – a tanulmányi rendszerekben található funkciók közül – csak a legsűrűbben használtakat fogom megvalósítani.

A leggyakrabban használt funkciók az adminisztrációs feladatok, tantárgy felvétel, jegybeírás, üzenetküldés, listázások.

Az üzenetküldési funkciót egy Lotus Notes rendszer esetén egyszerűen megoldhatjuk a mail adatbázison keresztül. Az egyes listázásokat a különböző nézeteken keresztül egyszerűen, programozás nélkül el lehet készíteni.

A nézeteken keresztül az alábbi listázásokat kapjuk:

- elérhető (felvehető) kurzusok az alábbi mezők szerint rendezve:
    - oktató
    - név
    - tanszék
    - kar
- felvett kurzusok az alábbiak szerint rendezve:
    - oktató
    - hallgató
    - tantárgyanként az egyes hallgatók listája
- teljesített tárgyak listája az alábbi mezők szerint rendezve:
    - oktató
    - név
    - tanszék
    - kar
- nem teljesített tárgyak listája az alábbi mezők szerint rendezve:
    - oktató
    - név
    - tanszék
    - kar
- oktatók listája az alábbi mezők szerint rendezve:
    - név
    - kar
    - tanszék
- hallgatók listája az alábbi mezők szerint rendezve:
    - név
    - kar
    - tanszék

7.4 Szerepkörök

Az adatbázisban az egyes funkciók szerepkörökhöz kötöttek. Erre azért van szükség, hogy egy hallgató pl. ne írhasson be jegyet magának, vagy ne hozhasson létre kurzus információkat.

[Administrator] – A karbantartási feladatokért felelős, adatbázis beállítások, adminisztráció.
[Instructor] – Oktatói szerepkör, tagjai jegyeket írhatnak be a hallgatóknak.
[Student] – Hallgatói szerepkör, tantárgyat vehet fel saját magának.

8. Újdonságok az egyes verziókban

8.1 Lotus Notes 7.0

- A 7.0-ás verziótól kezdve a Domino serveren 1.4.2-es Java JVM fut, ez fejlesztési szempontból hatalmas előnyt jelent.
- Habár a 6.0-ás verzióban már létezett a Web Alkalmazások támogatása, a 7.0-ás verziótól kezdve ezeket könnyen hozhatjuk létre, és azokat a domino serveren futtathatjuk mindenféle gond nélkül.
- Shared (megosztott) oszlopok – azaz létrehozhatunk olyan oszlopokat, melyeket létrehozásuk után bármelyik nézetben elhelyezhetünk.
- Automatikus mentés
- Forráskód nyomtathatósága a Programmer's pane-ről
- Framek és Framesetek nyomtathatósága Notes-ból
- Jobb használhatósága a Hide-When formuláknak a mobil készülékekre
- Az egyes agentekről és web szolgáltatásokról futási időben adatokat gyűjthetünk, elsődlegesen a teljesítményről, illetve, hogy az egyes kódrészeken a futtató rendszer mennyi időt töltött el (Profiling)
- Jobban használható Lotus script debugger
- A 6.1.-ben található leírásom még Lotus Notes 6.0 és 6.5-ös verziókhoz készült eredetileg, a súgóban a 7.0-ás verziótól találunk utalást rá, hogy ilyen lehetséges
- Új formulák (teljes listájuk megtalálható a Designer súgó What's new… szekciójában)
- Kibővített Lotus script osztályok, elsődlegesen a DOM kezeléshez
- A java osztályok is bővültek.

8.2 Lotus Notes 8.0

- A kliens az Eclipse nevezetű Java fejlesztői eszköz motorjára épül teljesen.
- Kompozit alkalmazások megjelenése, melyekkel egyszerűen hozhatunk létre egységes felületet különböző adatbázisok között.
- Új Formulák
- Új metódusok a java és Lotus script osztályokban.

8.3 Lotus Notes 8.5

- A Domino Designer is megkapja az Eclipse-es, egységes felületet.
- Xpages megjelenése. Segítségével könnyebben készíthetünk olyan formot, mellyel weben jeleníthetjük meg az adatokat, előnye az, hogy az egészet a grafikus felület mellett konfigurálhatjuk úgy is, hogy a létrejött XML filet alakítjuk át az igényeink szerint.
- A programnyelvek használhatósága sokkal bővebb lett, mint az előző verziókban, most már szinte „mindent használhatunk mindenütt”. Azaz nagyon sok helyen, ahol eddig pl. csak lotus scriptet használhattunk (pl. form események), ott most már használhatunk akár formula nyelvet, Javat, vagy Javascriptet is.
- Dojo és Ajax technológiák megjelenése
- Új funkciók a kompozit alkalmazásokban
- Új formulák
- Új metódusok a java és Lotus script osztályokban.
- Nagyon sok új funkcióval találkozhatunk az Eclipse-es felületű Designerben
- Témák használata
- Jobb és gyorsabb html kódgenerálás szerver oldalon
- Újabb verziójú javascript támogatása

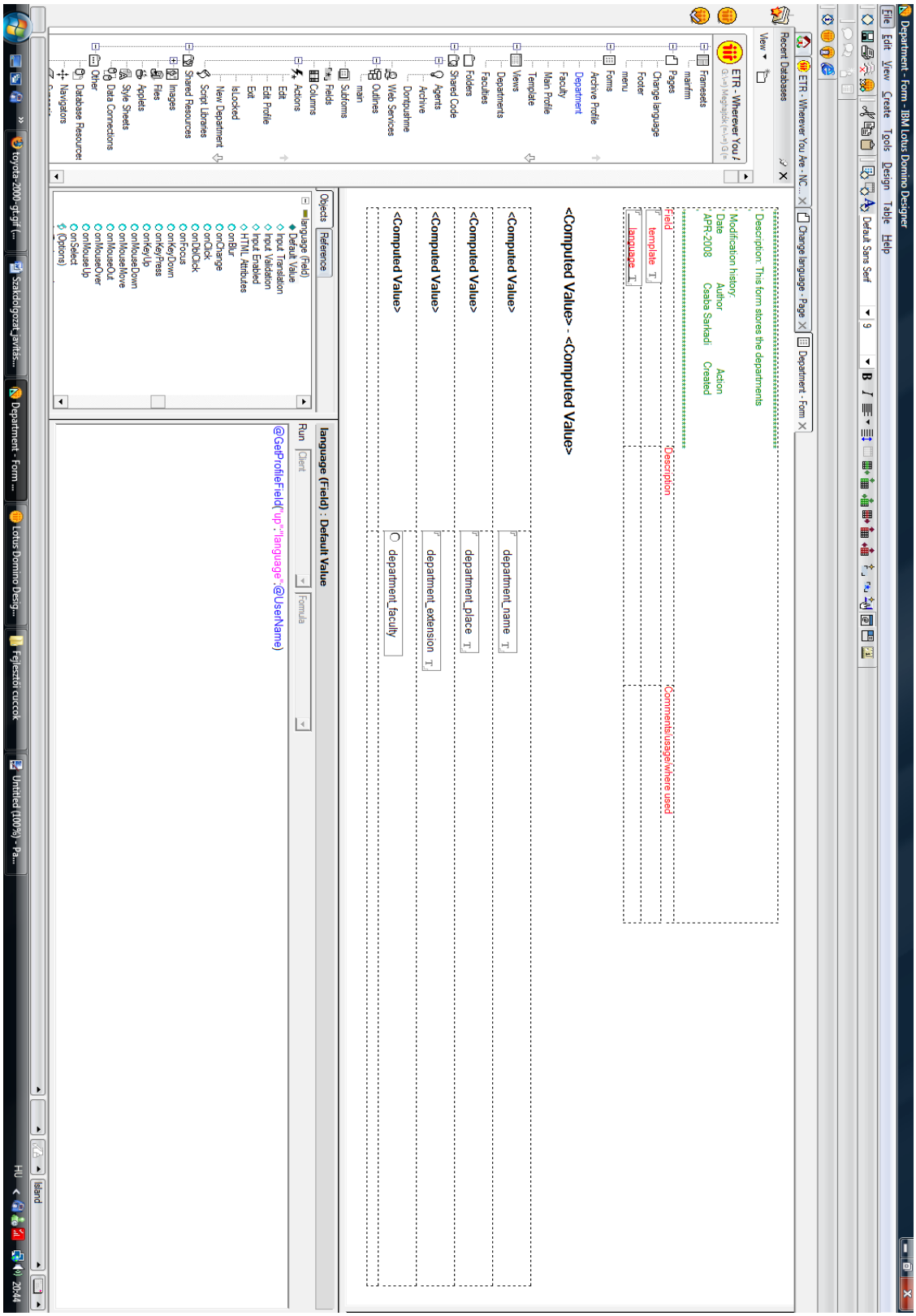
9. Irodalomjegyzék

[1] Séra – Sudár : Alkalmazásfejlesztés Lotus Notes 5.0-ban, Computerbooks, 2001
[2] Domino Designer 7.0 Help, IBM, 2006
[3] Domino Designer 8.0 Help, IBM, 2007
[4] Domino Designer 8.5 Help, IBM, 2008
[5] Fundamentals of IBM Lotus Domino 7.0 Application Development – Instructor Guide, IBM, 2006

Internetes források:
[6] IBM Lotus Notes redbook: http://www.redbooks.ibm.com/lotus/
[7] IBM Lotus fórum: http://www.ibm.com/developerworks/lotus/community/
[8] www.tlcc.com
[9] www.openntf.org

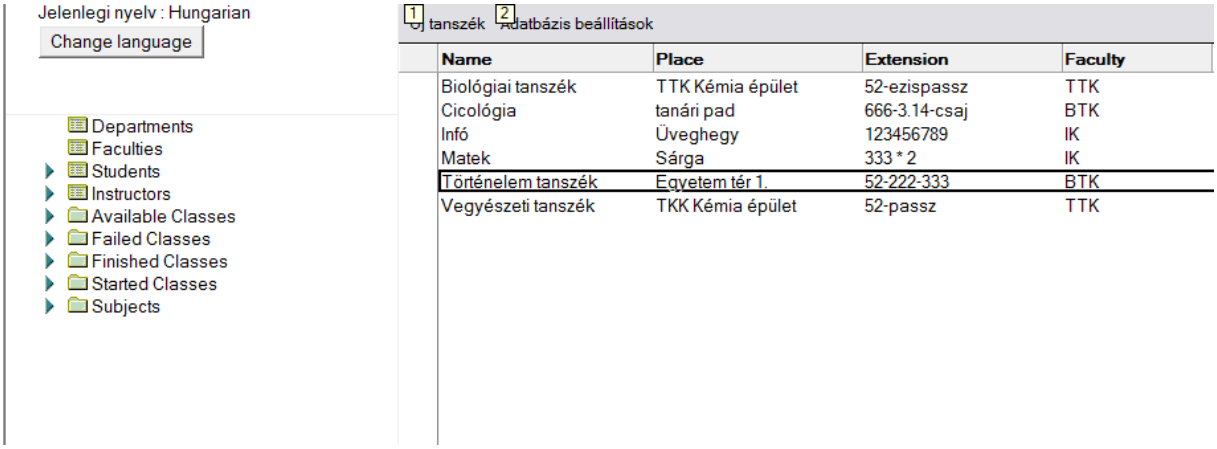
10. Függelék

20. Ábra a designer felület

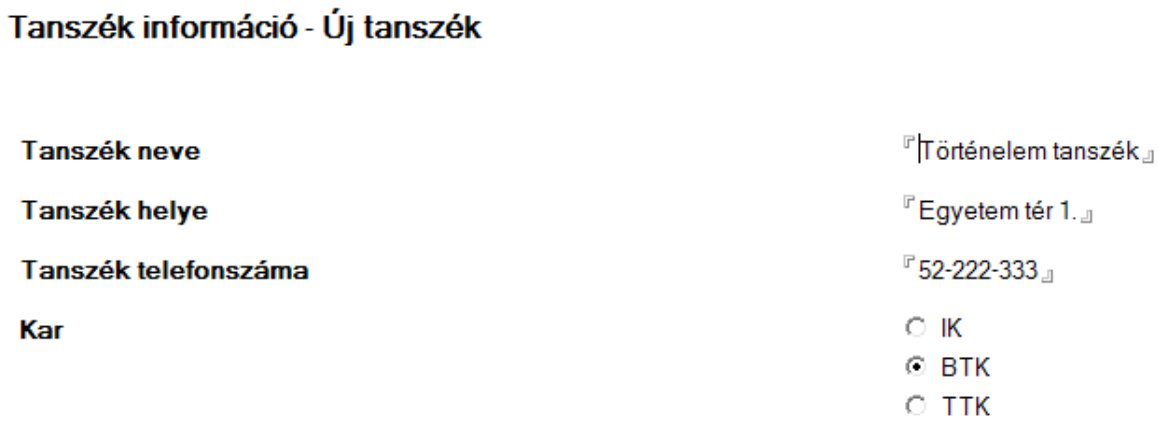


21. Ábra a designer súgója

22. Ábra az alkalmazás felülete



23. Ábra tanszék információ bevitele az alkalmazásba



24. Ábra adatbázis nyelvi beállítások

25. Ábra a nyelvi beállítások

Name	Név
Address	Cím
Phone	Telefonszám
Dean	Dékán
Faculty	Kar
Department	Tanszék
Department info	Tanszék információ
New Department	Új tanszék
Department name	Tanszék neve
Department place	Tanszék helye
Department extension	Tanszék telefonszáma
Sign up for subject	Jelentkezés
Give credit	Jegybeírás
Credit value	Kredit
Subject type	Tárgy típusa
Subject types	Aláírás; Gyakorlat; Kollokvium
Given credit	Kredit érték

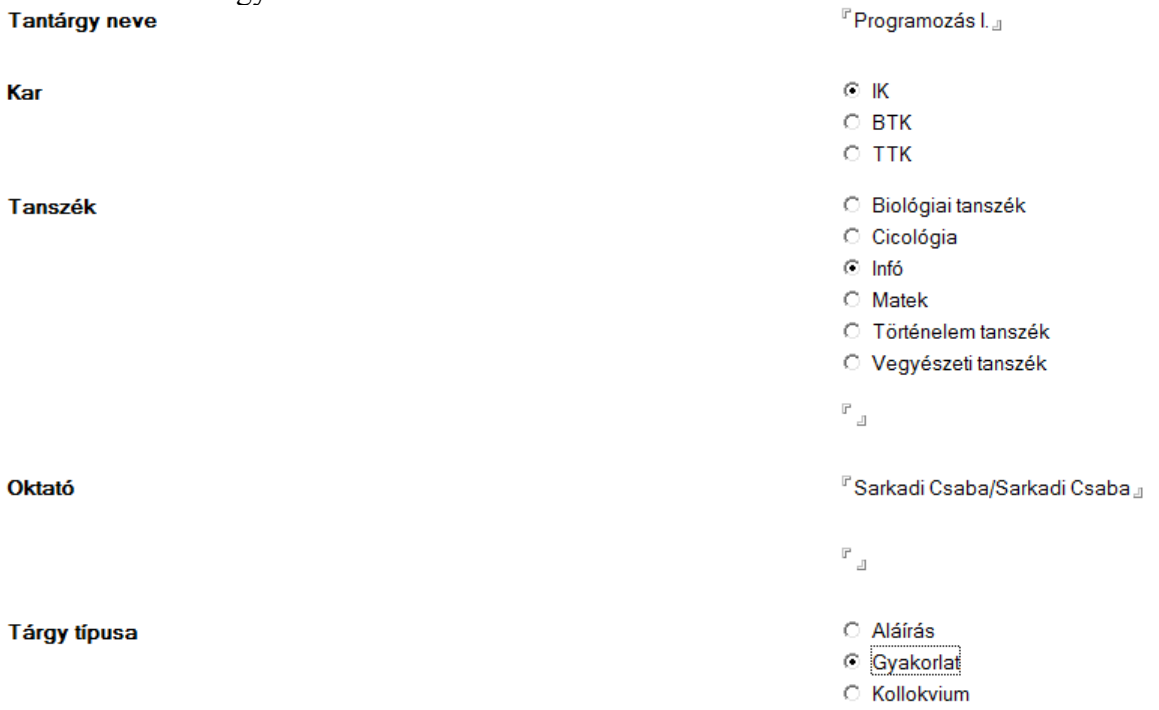
New Faculty	Új Tanszék
Faculty Information	Kar információ
Faculty Name missing	Kar neve hiányzik
Faculty address missing	Kar címe hiányzik
Faculty phone missing	Kar telefonszáma hiányzik
Faculty dean missing	Kar dékánja hiányzik
First name	Vezetéknév
Last name	Keresztnév
Student ID	Hallgatói azonosító
Email address	Email cím
New Student	Új hallgató
New Instructor	Új oktató
New Subject	Új tantárgy
Subject Name	Tantárgy neve
Subject Department	Tanszék
Subject Faculty	Kar
Subject Instructor	Oktató

26. Ábra kari információk

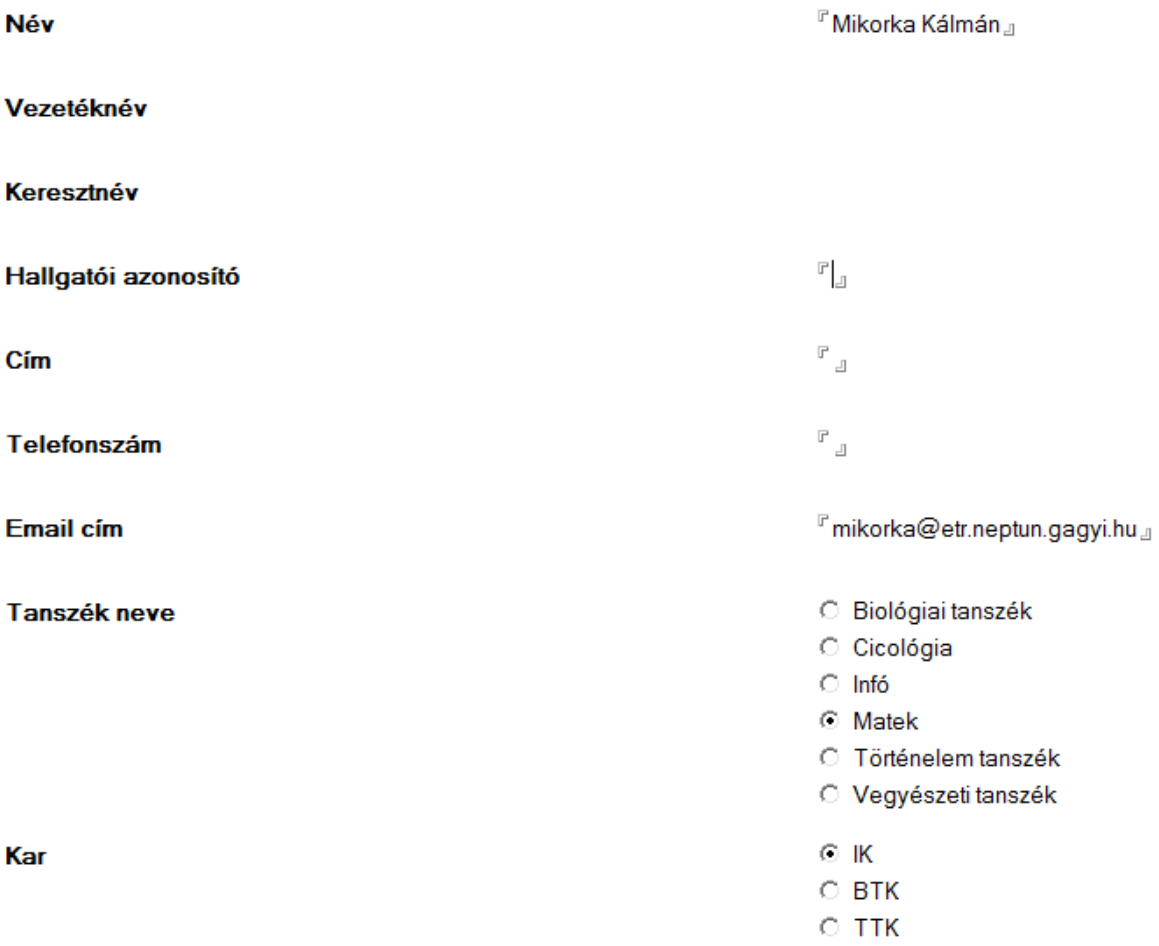
Kar információ - TTK

Név	TTK
Cím	Egyetem tér 1.
Telefonszám	52-nemtudom
Dékán	Rektor Géza

27. Ábra tantárgyi információk



28. Ábra hallgatói információ



11. Rövidítések, szómagyarázat

UNID – Universal ID. A Lotus Notes rendszerben minden dokumentumnak létezik egy egyedi, 32 byteos azonosítója, ezt hívják UNID-nek. Segítségével minden dokumentum egyértelműen beazonosítható, így megfelel a relációs adatbázis kezelők kulcs fogalmának. Dokumentumnak számítanak a design elemek is, így azoknak is létezik egy UNID mezőjük. Az azonosító létrehozásával nem kell törődnünk, azt a háttérben a Domino server megteszi helyettünk. Agent segítségével bizonyos esetekben ez az ID módosítható.

ACL – Access control list. Segítségével szabályozhatjuk az egyes adatbázisokhoz az egyes felhasználók, vagy felhasználói csoportok hozzáférési szintjét.

Replikáció – Az egyes adatbázisok között bizonyos feltételekkel történő adat szinkronizáció. Általános felhasználási területe a több központtal rendelkező cégek esetén a nagyobb kihasználtságú adatbázisok több központban történő elhelyezése, és azok közötti folyamatos szinkronizáció. Az egyes adatbázisok tartalma így gyakorlatilag majdnem megegyezik, a felhasználók pedig boldogabbak lesznek, mert gyorsabb az alkalmazások elérése.

Utószó

A célom egy olyan szakdolgozat elkészítése volt, melyben röviden be tudom mutatni a Lotus Notes rendszert, megoszthatok néhány tapasztalatomat a használatával kapcsolatban és egy mintaalkalmazást prezentálok. Az elképzelésem úgy érzem sikerült, egy szakdolgozat keretein és lehetőségein belül.

Remélem a dolgozatom egy megfelelő képet nyújt a Lotus Notes platformról, annak használhatóságáról, és gyakorlati életképességéről. Mint érezhető írásomból, igyekeztem hangsúlyozni a Java fejlesztési lehetőséget, is, mint lehetséges alternatívát, illetve felhasználható programozói forrást. Tapasztalataim szerint a két technológia jól ki tudja egészíteni egymást, és kibővíti az elérhető funkciókat.

Mivel csak egy rövid bemutatót készítettem, így az egyes fejezetek után ajánlom elolvasni az [2] és [6] –ból az idevágó részeket.
Programozási nyelveket nem ismertettem részletesebben, nyilván azért, mert csak a Lotus script, vagy Formula nyelvekről 1-1 könyvnyi információt lehetne összeszedni. Ezen információk pedig megtalálhatóak a [4]-ben is, nagyon jó példákkal.

Az egyes fejezetek során próbáltam hangsúlyozni az előnyöket, az esetleges hátrányokat, így egy támpontot nyújtva azok számára, akik nagyobb fejlesztés előtt állnak, és egyelőre nincsenek konkrét elképzeléseik a fejlesztési környezetről. Az egyes fejezetek kapcsán próbáltam ismertetni korábbi tapasztalataimat, illetve, hogy az adott témakörben mire érdemes figyelni. Meglátásaim szerint e tapasztalatok még egy tapasztaltabb programozó számára is hasznosak lehetnek.

Az egyes témák kidolgozása során nagy segítséget jelentettek a korábbi és a jelenlegi kollégáim is, akiktől rengeteget tanultam az elmúlt években a Lotus Notes alkalmazásfejlesztés terén. Ezúton is szeretnék köszönetet mondani nekik, hogy még a leglehetetlenebb időpontokban is hajlandóak voltak segíteni, és válaszolni a felmerült kérdéseimre.

A legnagyobb köszönettel a témavezetőnek, dr Zichar Mariannának tartozom, az alapos munkájáért és a rengeteg felfedezett hibáért.