SZAKDOLGOZAT

Molnár János

Debrecen

2009

Debreceni egyetem
Informatikai kar

Adatbázis alapú alkalmazás fejlesztése Borland Delphi környezetben

Témavezető:
Dr. Bajalinov Erik
Tudományos főmunkatárs

Készítette:
Molnár János
Programozó Matematikus Szak

Debrecen
2009

Tartalomjegyzék

1. BEVEZETÉS

Molnár János vagyok, jelenleg recepciósként dolgozok a hajdúszoboszlói négycsillagos Apolló Termalhotel-ben. Munkám során megismerkedtem több szállodai szoftverrel is melyek több-kevesebb sikerrel igyekeztek felváltani valamint kiegészíteni a hagyományosan papír alapon folyó tevékenységeket és kijavítani azok hibáit. Természetesen a korai verziókat nem a szállodai szakemberek készítették így várható is volt hogy nem fognak teljes mértékben megfeleni a felhasználók igényeinek. Másrészt a régi hardware okozta lassúság - egy bonyolultabb lekérdezés, listakészítés vagy listanyomtatás mátrixos nyomtatón - és inkompatibilitás is borzolhatta az emberek kedélyállapotát, a programok működési hibái mellett. Ilyen rendszerek voltak a ma már jelentős mértékben javított és egyéb platformokon is megjelenő Hostware vagy Flexys DOS-os elődei.

Az idők során elkészült újabb rendszerek már kiküszöböltek sok kis apró vagy nagyobb hibát, de még a ma használt programok sem elégítenek ki maradéktalanul minden felhasználói igényt. A tapasztalat azt mutatja, hogy a hibák és a félkész megoldások miatt az emberek nem szívesen használnak számítógépet, maradnak inkább az időigényesebb papírmunkánál, így kihasználatlanul marad segítőtársunk pozitív oldala mely az adatok tárolásában és feldolgozásában nyilvánul meg.

A probléma

A program készítésének gondolata akkor vetődött fel bennem amikor a szálloda orvosának asszisztense már sokadik alkalommal adta le hibásan illetve félig kitöltve azon vendégek névsorát akik különböző kezeléseket igényeltek. Néhol a kezelés kódja és megnevezése nem volt megfelelő, vagy a kezelések összege volt hibás, máshol a név és szobaszám nem egyezett, egyszóval minden olyan hibát elkövetett ami emberileg lehetséges volt

A probléma megoldása

Egy olyan programra van tehát szükség amelyben kevesebb hibalehetőséggel és lehetőleg kevesebb idő alatt elvégezhető ugyanaz a munka ami azelőtt papíron történt. Képes legyen eltárolni a vendégek bizonyos adatait (név, szobaszám, stb.) melyek segítségével beazonosíthatók, valamint az általuk igényelt kezeléseket és azok összegét. Továbbá lehetőség legyen ezen adatok visszakeresésére, megtekintésére valamint módosítására, amennyiben erre szükség lenne, természetesen csak az arra jogosultaknak. Az adatok ugyan nem titkosak de illik bizalmasan kezelni és védeni őket az illetéktelen hozzáférésektől.

2. AZ ADATBÁZIS

Hétköznapian: a bennünket érdeklő dolgok adatainak szervezett összessége.
Tárolás szempontjából: az adatbázis több, egymással kapcsolatban álló table összessége.

Definíció: Az adatbázis véges számú egyed-előfordulásnak, azok véges számú tulajdonság-előfordulásának és kapcsolat-előfordulásának az adatmodell által szervezett együttese.

Az adatmodell: Az adatmodellek tükrözik az általunk leképzett rendszer legfontosabb információ-elemeit (egyedeit), és ezen információ elemek kapcsolatait.

Definíció:Az adatmodell véges számú egyedtípusnak, azok véges számú tulajdonságtípusának és kapcsolatának szervezett együttese.

Kapcsolatok: Az adatok több táblába csoportosításával, a felesleges adatok elhagyásával elérhető, hogy az adattárolás gazdaságosabb és biztonságosabb legyen.

Elsődleges kulcs: A kapcsolatok létrehozásához kulcsokat kell definiálni.

Definíció: Az egyedtípus azon tulajdonságtípusát vagy tulajdonságtípus együttesét, amely minden egyed-előfordulásra eltérő értéket vesz fel, az egyedtípus azonosítójának, más néven elsődleges kulcsnak nevezzük.

Szabály: Az azonosító értéke egyetlen egyed-előfordulásban sem lehet definiálatlan. Minden egyedtípusnak egy és csak egy azonosítója van. Idegen kulcs Definíció:Az egyedtípus azon tulajdonságtípusát, vagy tulajdonságtípus együttesét, amely egy másik egyedtípusban elsődleges kulcs szerepet tölt be és az adott egyedtípusban csak a két egyedtípus közötti kapcsolat létrehozásához szükséges, idegen kulcsnak nevezzük. Azt, hogy egy mező idegen kulcs, nem kell definiálni a táblában, ezt a kapcsolat létrehozásakor kell csak megadni. A tulajdonságtípus együttes arra utal, hogy a kulcs nem feltétlenül egyetlen tulajdonság, hanem gyakran több tulajdonság összekapcsolásból alakul ki (pl. Csnév, Knév és Szülidő).

Kapcsolatok:

Definíció: Két egyedtípus akkor és csak akkor áll közvetlen kapcsolatban egymással, ha az egyik egyedtípus elsődleges kulcsát idegen kulcsként tartalmazza a másik egyedtípus.

Definíció: Az egyedtípusok között lévő viszonyokat, amelyeket az elsődleges kulcs-idegen kulcs tulajdonságok írnak le, kapcsolattípusnak nevezzük.

Definíció: Az egy kapcsolattípusba tartozó konkrét kapcsolatokat kapcsolat-előfordulásnak nevezzük.

Egy a többhöz (1:N) kapcsolatnak ha két egyedtípus egyed-előfordulásai között nem kölcsönösen egyértelmű hozzárendelés van, azaz az egyik egyedtípus bármely egyed-előfordulásához több egyed-előfordulás tartozhat a másik egyedtípusban , míg a másik egyedtípus bármely egyed-előfordulásához az egyikben egy és csak egy.

Szabály: Több a többhöz (M:N) kapcsolatot mindig csak közvetve, egy kapcsolótáblával és két ellentétes irányú egy a többhöz kapcsolattal hozhatunk létre. A kapcsolótábla két idegen kulcsként tartalmazza a másik két tábla elsődleges kulcsait.

Egy az egyhez (1:1) kapcsolatnak nevezzük, ha két egyedtípus egyed-előfordulásai között kölcsönösen egyértelmű hozzárendelés van, azaz mindkét egyedtípus bármely egyed-előfordulásához egy, és csak egy egyed-előfordulás tartozhat a másik egyedtípusban.

Hivatkozási Integritás: A hivatkozási integritás biztosítja, hogy két gyedtípus közötti kapcsolatban az egyes egyed-előfordulások kapcsolatai ne sérüljenek meg, a kapcsolt táblák kapcsoló mezőinek tartalma mindig megfelelő legyen.

Szabály: Nem rögzíthető olyan idegen kulcs érték, amely a kapcsolt táblában mint elsődleges kulcs nem létezik. Nem szüntethető meg olyan elsődleges kulcs érték, amely a kapcsolt táblá(k)ban idegen kulcsként létezik. Nem módosítható olyan elsődleges kulcs érték, amely a kapcsolt táblá(k)ban idegen kulcsként létezik.

3. ADATBÁZISKEZELÉS

3.1. Adatbázis-kezelő rétegek:

Egy adatbázis-kezelő alkalmazás esetén alapvetően három különböző réteget különböztetünk meg, melyek a következők:

Az adatszolgáltatások rétege (Data Processing)

Ez a réteg felelős az adatok fizikai eléréséért, feldolgozásáért. E réteg feladata az adatbázis állományok nyitása, zárása, újadat felvitele, törlése, módosítása, indexek kezelése, zárolási konfliktushelyzetek feloldása, és így tovább.

Az alkalmazáslogika rétege (Bussines Logic)

Az alkalmazáslogika rétege az adatbázisra vonatkozó szabályok összességét tartalmazza. Gyakorlatilag ebbe a rétegbe tartoznak azok a funkciók, műveletek, amelyek meghatározzák egy adatbázis működését. Ilyen szabályok a mező illetve rekordszintű ellenőrzések (mezőszintű ellenőrzés, pl. ha egy tanuló érdemjegyeinek felvitelekor a program csak egy és öt közötti értéket enged felvinni), a hivatkozási függőségek ellenőrzése (pl. egy könyvet csak akkor lehessen eladni, ha az szerepel a könyvesbolt árukészletén) stb.

Megjelenítési réteg (User Interface)

Mely a tulajdonképpeni kezelői felület, ahol a felhasználónak lehetőséget biztosítunk az adatbázishoz való hozzáférésre. Természetesen a fent tárgyalt rétegek a különböző adatbázis architektúrákban eltérő módokon valósulhatnak meg, attól függően, hogy a struktúra milyen lehetőségeket kínál.

3.2. Adatbázis architektúrák

Egygépes megvalósítás (Local Databases)

Az adatbázisoknak ez a lehető legegyszerűbb megvalósítási módja. Az alkalmazás egyetlen gépre íródott, az adatbázis és az azt feldolgozó program ugyanazon a gépen helyezkedik el, az adatbázist csak egyetlen program használja egy időben. Ebben az esetben mindhárom réteg egyazon gépen helyezkedik el.

File-kiszolgáló (File-Server) architektúra

Ebben az esetben az adatbázis állományok átkerülnek egy központi szerverre és egy időben több program is használhatja őket hálózaton keresztül. A szerver csak az adatok tárolására szolgál. Ezen megoldás esetén, ha a felhasználó akármilyen egyszerű adatműveletet akar is végrehajtani, az adatrekordoknak el kellett jutniuk a felhasználóhoz a hálózaton. Ez nagy adatforgalommal jár, ami a hálózat túlterheléséhez vezethet. Ezt a megoldást leginkább az xBase alapú (Dbase,FoxPro,Clipper..) adatbázis-kezelőkkel használják. Delphiben is írhatunk ilyen alkalmazásokat, de csak akkor érdemes, ha nincs szükség nagy teljesítményre, aránylag kevés felhasználója van a programnak, és olcsón meg akarjuk úszni a dolgot, (mivel egy adatbázisszerver nem olcsó mulatság). Szintén mindhárom fentebb tárgyalt réteg egyazon gépen helyezkedik el.

Ügyfél-kiszolgáló (Client/Server) Architektúra

Az adatbázisok implementálásának e formájában az alkalmazás két részre bomlik. Az adatok közvetlen kezeléséért egy adatbázis-szervernek nevezett software a felelős, (pl. MS SQL Server, Oracle, Informix, Sybase, InterBase stb.), míg a felhasználóval való kapcsolattartás az ügyfél program feladata. Az adatbázis-servert készen vásárolhatjuk meg, míg a kliens programot mi magunk írhatjuk meg valamilyen programozási nyelven. Az alkalmazás logikának egy részét magába az adatbázisba, a többit a kliens programba tudjuk beépíteni. Hogy ez hogyan is történik arról majd később még lesz szó. A client/server technológiában az ügyfél utasítja a szervert, pl. adatokat kér le, és erre a szerver visszaküldi

az eredményt. Tehát nem kell a hálózaton a feldolgozandó adatoknak rekordról-rekordra átmenni a klienshez, hanem egy rövid parancs hatására, csak a ténylegesen kért, hasznos adatok fognak a szervertől a kliensig utazni, ezáltal jelentősen csökkentve a hálózati forgalmat. Így az adatfeldolgozást a szerver végzi a kliens parancsainak hatására. E parancsok számára kidolgoztak egy szabványos nyelvet, ez az SQL. Tehát az adatbázis-szervereket ilyen SQL parancsokkal tudjuk munkára bírni. Az adatbázis szervereknek a hálózati forgalom csökkentésén kívül számos más előnyük is van, biztosítják az egyidejű adatelérést (egyszerre nagy számú felhasználó kiszolgálására képesek), az adatbiztonságot, központilag kezeli a felhasználói jogosultságokat, stb.

Több rétegű (Multi-Tier) adatbázis architektúra

Ebben az esetben a kliens nem közvetlenül az adatbázis-szerverhez, hanem egy vagy több köztes ún. applikációs szerverhez kapcsolódik, és végül az applikációs szerver kapcsolódik az adatbázis-szerverhez. Tehát a kliensnek a középen elhelyezkedő applikációs szervertől kapják az adatokat, ezért ezt a réteget adatszolgáltatónak (Data Broker) is nevezik. Így az adatbázis logikát el lehet helyezni a középső rétegben, és a kliens feladata csak a felhasználóval való kapcsolattartás lesz. Az ilyen kliens-t "sovány" (thin) kliensnek nevezzük, hiszen a munka nagy részét az applikációs szerver végzi. Tehát ebben az esetben a három réteg, fizikailag is három különböző helyen helyezkedhet el.

4. A DELPHI

4.1. Bevezető

Windows alapú alkalmazások fejlesztéséhez jó darabig csak a C nyelv állt rendelkezésre. Mikor a Microsoft kihozta Visual Basic nevű, komponens alapú fejlesztő rendszerét, a Borland is elérkezettnek látta az idejét, hogy - miután már korábban megjelentette népszerű Turbo Pascal programnyelvét Windowsos változatban - vizuális fejlesztői környezettel lássa el a Pascal-t. Így született meg a Delphi, egy integrált fejlesztői környezetben (Integrated Development Environment) futó valós idejű fejlesztőrendszer. Könnyedén és gyorsan hozhatunk létre segítségével igényes, bonyolult Windows környezetben futó programokat, köszönhetően a RAD (Rapid Application Development) felületnek. A korszerű fordító technika eredményeként optimalizált és gyors futású programot nyerünk. A Delphi ugyanis a forráskód lefordítása után valódi, végrehajtható gépi kódot állít elő (szemben pl. a Visual Basic interpreter típusú programmal, mely a forráskódot futási időben értelmezi).

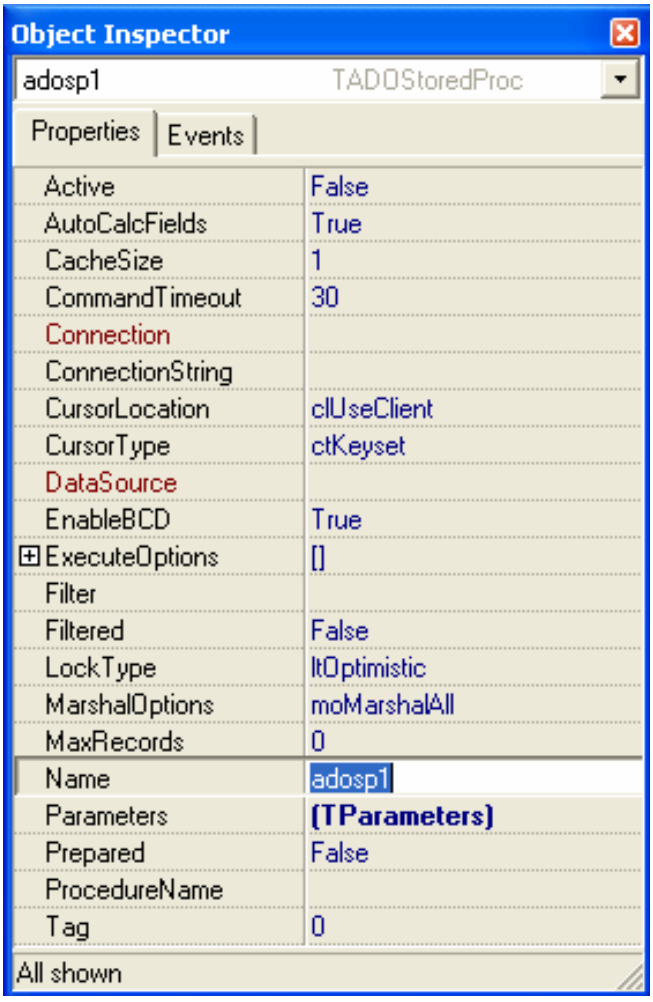
4.2. Részei

4.2.1. A komponenspaletta

A komponenspalettán alapesetben 15 lapon, tematikus csoportokba gyűjtve találhatók a Delphi komponensei. Már a Standard lapon található elemekkel felépíthető egy átlagos form, azaz Windows ablak (menü, gombok, jelölő négyzetek, rádiógombok stb. felhasználásával), az esetleg még szükséges komponensek a további lapokon találhatók. A form szerkesztése úgy történik, hogy a komponenseket az adott gombra kattintva kiválasztjuk, majd a form egy adott helyére kattintva elhelyezzük a form felületén. Az egyes elemek tulajdonságait az Object Inspector-ban (objektum-felügyelő) állíthatjuk be.

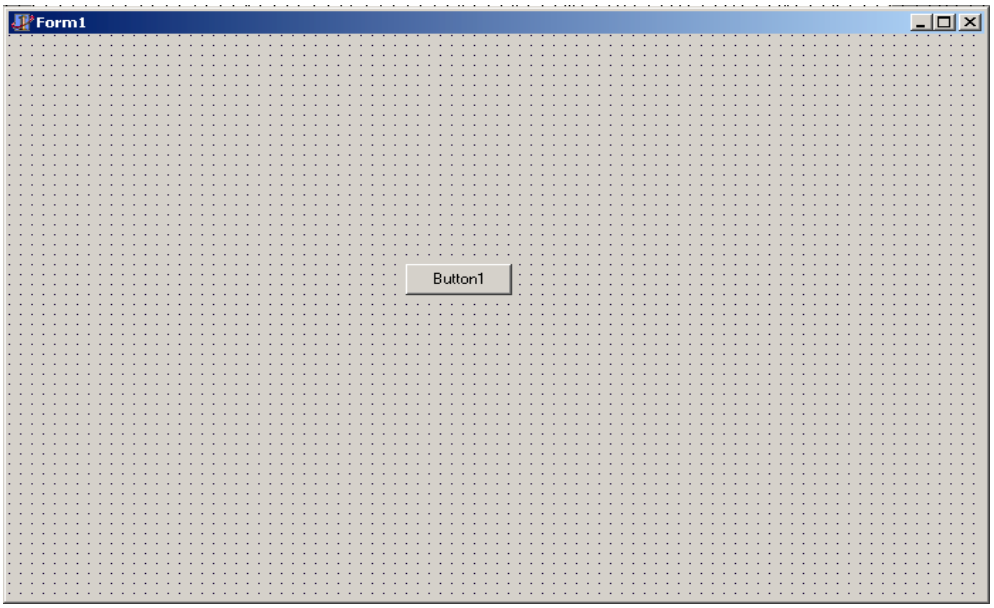
4.2.2. Az objektum-felügyelő

Az Object Inspector voltaképpen egy kétlapos párbeszéd-ablak, ahol beállíthatjuk a form és a komponensek tulajdonságait (név, méret, szín, betűk stb.). Ez történik a Properties nevű lapon. Az Events (események) lapon pedig az egyes elemekhez kapcsolódó eseményekről (pl. egérkattintás, billentyűleütés stb.) intézkedhetünk. Az objektum-felügyelőben mindig a formszerkesztőn aktív (vagyis kijelölt) elem tulajdonságai állíthatók be, illetvemódosíthatóak.



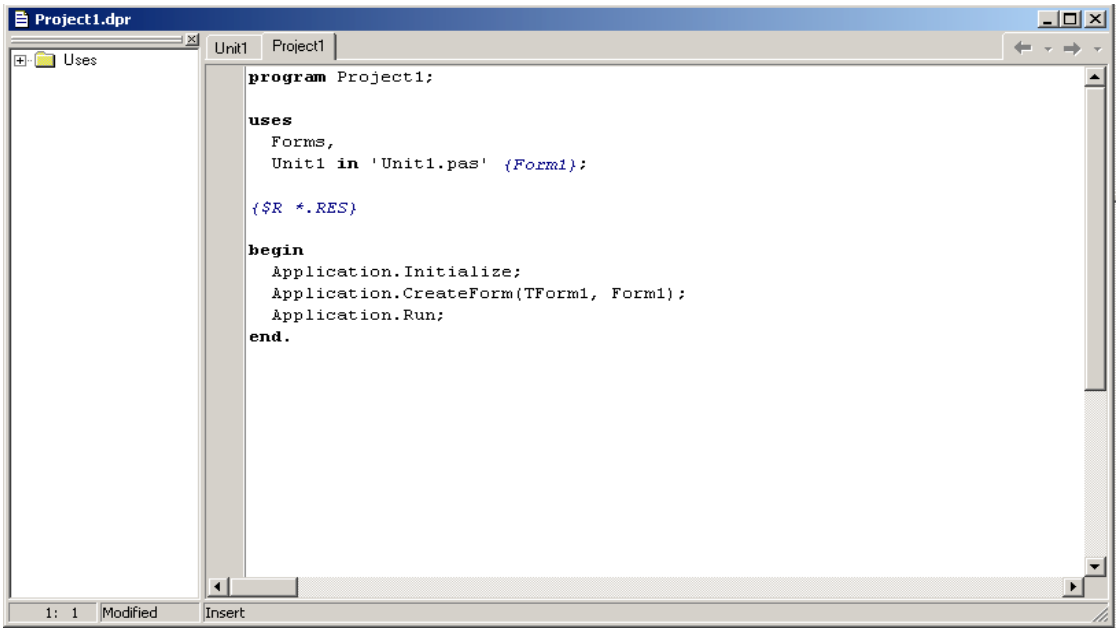
4.2.3. A formszerkesztő

A formszerkesztő az a felület, amelyen a tervezett Windows ablakot kialakítjuk. A formon (űrlapon) kell elhelyezni a menüt, gombokat stb., az objektum-felügyelőben pedig megadhatjuk ezek tulajdonságait. Már maga a form is egy teljes, ámbár üres Windows ablakot takar. Ha lefuttatnánk, egy olyan üres Windows ablakot kapnánk, amit méretezhetünk, áthelyezhetünk, használhatnánk az ablakgombokat, de még a rendszermenüt is. Az ablakmodellek leírását a Delphi .DFM (Delphi Form) kiterjesztésű állományokban tárolja.



4.2.4. A kódszerkesztő ablak

A kód- (szöveg-) szerkesztő ablak alapesetben a formszerkesztő alatt alig látható. Akkor tűnik elő, ha a kilógó alsó részére kattintunk, vagy megnyomjuk az F12 funkcióbillentyűt. Ennek eredményeként a fent elhelyezkedő űrlapunk előtt megjelenik az addig a háttérben meghúzódó UNIT ablak a következő ábra szerinti formában:



Az ablakban az eddig "megírt" Object Pascal nyelvű kódszöveget láthatjuk. A Delphi ugyanis az általunk vizuális módon szerkesztett form, illetve a komponensek Pascal-kódját rögtön beilleszti az aktuális unit megfelelő helyére. A unitok fájljai .PAS kiterjesztésűek. Minden egyes formhoz tartozik egy unit (de természetesen létezik form nélküli unit is). Egy program alá általában több form és több unit is tartozik. Maga a program (azaz a project) kódja is megtekinthető, ha a View menü UNIT parancsát választjuk, és ott kiválasztjuk a Project1-et. A projekt forráskódja .DPR kiterjesztésű állományban tárolódik.

4.2.5. Kódszerkesztés a Delphiben

A Windows egész filozófiája az ablakokra van alapozva, ezért a Windowsos felületre tervezett Delphi alkalmazásokat is ablakokban (formokban) kell elképzelni. A program (project) különböző részei, műveletei egy-egy, arra a feladatra tervezett ablakban játszódnak le. Magyarul: a program felhasználói felületét az ablakok szolgáltatják.

A Delphiben kétféle programozási stílussal dolgozhatunk. Az egyik a komponensekkel való programozás, aminek az alapjain már lassan túljutunk. A komponensek előre elkészített építőelemek, amelyekből úgy állítjuk össze a programot, mint egy Lego-játékot az építőkockáiból. A másik programozási mód a "hagyományos": a problémamegoldás Object

Pascal programnyelven való megfogalmazása. A Delphi a forráskód egy részét saját maga készíti el, a tulajdonképpeni algoritmusokat nekünk kell megírnunk. De mik is a komponensek? Adott feladatra tervezett elemek, amelyeket Object Pascal-ban írtak meg, s amelyek forráskódjához a Delphi bizonyos verzióiban hozzá is férhetünk. A komponenseket a nevük elé írt T betűről ismerhetjük fel. A komponenseket az űrlapon (formon) helyezhetjük el. Maga a form komponens neve TForm, a ráírt szöveg (címke) TLabel, a ráhelyezett nyomógomb pedig TButton stb. Ha eseményeket szeretnénk rendelni az objektumainkhoz, meg kell keresnünk az objektum-kezelőben az objektum Events fülén lévő valamely eseményt, és végre valóban „programozhatunk”, a korábban már elsajátított Pascal utasításainkra alapozva.

5. A DELPHI ADATBÁZIS-KEZELÉSE

A Delphi rendszer egyik kiemelkedő erőssége a különböző tipusú adatbázisok kezéléséhez szükséges technológiák és eszközök sokoldalú támogatása. Az adatbázis-kezelő alkalmazások fejlesztése a Delphi korábbi és a jelenlegi változataiban egy speciális programozási felületen, a BDE *(Borland Database Engine)* API-n (régebbi neve az IDAPI, *Integrated Database Application Programming Interface)* keresztül történik. A Delphi rendszer, illetve a Delphi-ben fejlesztett adatbázis-kezelő alkalmazások egyaránt a BDE alkalmazás-programozói felületet használjak.

A BDE *(Borland Database Engine)*

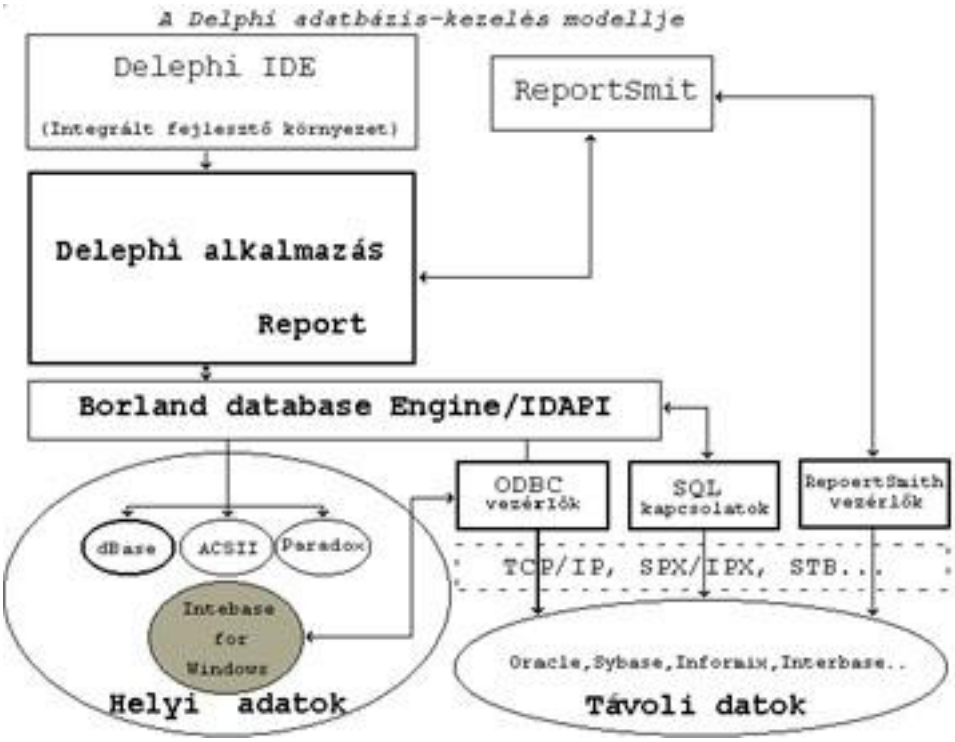
A BDE mint programozási felület sok olyan alapfeladat elvégzése alól mentesíti az adatbázis-kezelő alkalmazások programozóit, mint az adattáblák és bejegyzések lezárása, bejegyzések frissítése, alap 1/O műveletek stb. Ezért a fejlesztő több figyelmet szentelhet magának az adatnak és a vele elégezendő műveleteknek.

A BDE API közel 200 eljárást és függvényt tartalmaz *(BDE* modul), amelyeknek közvetlen meghívására azonban csak igen ritkán van szükség. Ehelyett a BDE-használat általában a Delphi adatelérési komponensein keresztül történik (az eszközpaletta *Data Access* lapja), melyek az egyszerű kezelhetőség mellett a BDE API hívásainak többségét is megvalósítják.

Mivel technikailag a BDE az adatbázis-eléréshez szükséges dinamikusan szerkeszthető könyvtárak gyűjteménye, telepitése után párhuzamosan több alkalmazás is osztozhat rajta. Ezzel együtt - mivel az adatelérésekhez szükséges kód másolata csak egyszer van jelen a memóriában - nem csak az összesített RAM-foglalás mértéke csökken, hanem a BDE-t használó alkalmazások mérete is kisebb az adatbázis-elérést végző kódot is tartalmazó alkalmazásokénál.

Ha az alkalmazásunk írásakor a BDE lehetőségeit használjuk, mind a gépünk lokális - például *dBASE, Paradox* - adattábláit, mind a távoli adátbázis-szerveren levő, illetve az

ODBC *(Open DataBase Connectivity)* meghajtók által támogatott állományokat is könnyen elérhetjük.



A BDE egyik legfontosabb tulajdonsága, hogy alkalmazásunk írása során nem kell kötődnünk egyik adatbázis-szabványhoz sem. A fejlesztés során az adatbázis-elérő műveleteket, a Delphiben megszokott módon a vezérlőelemek kezelőfelületén keresztül végezhetjük el. Ha a későbbiekben át kell térnünk egy másik adatbázis-típusra, az esetek többségében az alkalmazás újrafordítása megspórolható. Ez azért lehetséges, mert az adatok helyére és elérésére vonatkozó beállítások az alkalmazáson kívülről is megadhatók, a BDE Administrator alkalmazás felhasználásával.

Logikai nevek (aliasok) létrehozása BDE Administrator segítségével

A BDE Administrator segítségével beállíthatjuk (Start/Beállítások/Vezérlőpult/BDE Administrator) hogy a programunk az elérni kívánt adatokat hol találja meg, melyik

megahjtóprogramot kell alkalmazni az adateléréshez.

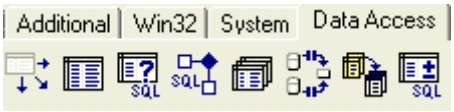
A BDE Administrator ablakban bal oldalt található panel, Datebase lapján az un. logikai nevek (aliasok) a Configuration lapján pedig a meghajtók és a rendszerbeállítások felsorolása található. A jobb oldali panelben a kijelölt elemre vonatkozó paraméterek jelennek meg.

A logikai (alias) vagy adatbázisnév, ill. az általa takart paraméterkészlet az adathalmaz elérhetőségét és fizikai elhelyezését írja le. A BDE alkalmazásokban a logikai neveket az un külső, azaz az adott számítógépen futó összes alkalmazás számára elérhető adatbázissal való kapcsolat megteremtésére használjuk. A lokális adatbázisok esetén eldönthetjük, hogy logikai névvel, vagy közvetlenül az elérési út megjelölésével hivatkozunk az adatforrásra. Az SQL adatbázisokra való hivatkozást csak logikai név használatával oldhatjuk meg.

Az adatforrásunkhoz új logikai nevet többféleképpen is létrehozhatunk. Használhatjuk a BDE Administrator Object/New, illetve a Databases panel felbukkanó menüjének New menüpontját. Ugyanitt módosíthatjuk a neveket, ill. az adatok lezárása (Close) után átnevezhetjük (Rename), ill. törölhetjük (Delete) is őket.

5.1. Adatelérési komponensek

Az adatelérési (Data Access) komponensek az adattáblák elérését teszik lehetővé. Tulajdonképpen a BDE megfelelő moduljaival ezek tartják a kapcsolatot. A Delphi változatától függ, hogy itt milyen elemeket láthatunk, a Standard változatban az alábbi palettával találhatjuk szemben magunkat:



Nézzük meg, hogy mire is szolgálnak az itt látható, előre elkészített komponensek, amelyek többsége nem vizuális komponens, vagyis csak tervezési időben látszanak.

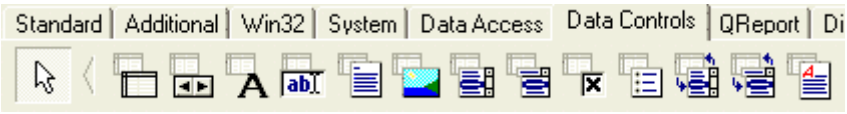
	Az adatbázisok eléréséhez szükségünk van egy DataSource komponensre, amely erre szolgál. Fontos, hogy ez nem közvetlenül kapcsolódik a fizikai táblához, mivel ennek bemenete lehet akár egy tábla, akár egy lekérdezés eredménye, de akár tárolt eljárások is szolgáltathatják a bemenetét. Az adatmegjelenítési komponensek ettől kapják az adatokat. Ez a megoldás rendkívül rugalmas és hatékony eszközt ad a kezünkbe, mert így az adatok és a megjelenítési komponensek függetlenné válhatnak egymástól.
	A Table komponens egy relációs táblával tartja a kapcsolatot a BDE-n keresztül. A paramétereit rendszerint tervezési időben határozzuk meg, néhány tulajdonságának kivételével.
	A Query komponenssel a relációs táblákból kérdezhetünk le rekordokat az SQL nyelv használatával. Szintén tervezési időben állítjuk be a legtöbb tulajdonságát, de az SQL utasításokat az esetek döntő részében csak futási időben adjuk át.
	A StoredProc szerver-kliens adatbázis-szerkezetnél használatos, amikor a kliens az adatbázis-szerveren eljárásokat szeretne tárolni, illetve azokhoz hozzáférni.

	A Database komponens rendszerint szerver-kliens architektúráknál használatos, lehetővé teszi a kapcsolatok ellenőrzését, különböző biztonsági műveletek elvégzését csakúgy, mint a kapcsolat-ellenőrzést.
	A Session komponens legfontosabb tulajdonsága, hogy egy eseményt biztosít az adatbázisokban történő bejelentkezések testre szabásához. Mi nem fogjuk használni.
	A BatchMove segítségével, mint ahogy a nevéből is látszik, kötegelt műveletek végrehajtásra nyílik lehetőségünk. Ilyen művelet lehet akár másolás, akár mozgatás, vagy törlés is.
	Az UpdateSQL segítségével adatfrissítő műveleteket végezhetünk SQL lekérdezések segítségével egy csak olvasható Query komponensen. Rendszerint a táblák és a lekérdezések UpdateObject értékeként használjuk.

5.2. Adatmegjelenítési komponensek

Az adatok párbeszédpanelen (FORM) megjelenítésére számos komponens áll rendelkezésünkre, amelyek rengeteg előnyös tulajdonsággal rendelkeznek. Ha ezek nem lennének megfelelő, az interneten számtalan komponenshez férhetünk hozzá, amelyek egy része ingyenes, míg másokat meg kell vásárolnunk.

Amikor megnyitjuk a Delphi Standard Data Controls (adat vezérlő) palettáját, akkor ehhez hasonló választékkal találjuk szemben magunkat:



	A DBGrid egy táblázatot jelenít meg a panelen, amelyen a táblákból származó adatokat láthatjuk. A táblázat automatikusan annyi oszlopot tartalmaz, amennyi mezője van a relációnak, de ezt felül is bírálhatjuk. Áttekintő listák készítéséhez nagyon jól használható.
	A rekordokműveleteket segíti elő a DBNavigator. Segítségével a rekordmutató léptetésén túl, felvehetünk új rekordot, törölhetjük az aktuális sort, vagy akár szerkeszthetjük is azt. A gombok tetszőlegesen ki és bekapcsolhatók, annak megfelelően, hogy melyikre van szükségünk.
	Abban az esetben, ha egy statikus, vagyis a formon nem módosítható szöveget kell megjeleníteni egy relációs táblából, akkor használjuk a DBLabel komponenst. Az éppen aktuális rekord hozzárendelt mezőjének az értékét jeleníti meg.
	Az egyik leggyakrabban használt vezérlőelem a beviteli mező adatbázisokhoz illesztett változata, a DBEdit. Nem csak a mező értékét képes megjeleníteni, hanem módosíthatjuk is az, emennyiben engedélyeztük ezt az adatbázis műveletet.

	Számos esetben előfordulhat, hogy nem elegendő egy soros beviteli mező, ilyen esetekben használhatjuk a DBMemo komponenst, amely a DBEdit többsoros változata.
	Ha képet szeretnénk megjeleníteni a párbeszédpanelünkön, akkor erre a DBImage komponenst felhasználva nagyon egyszerűen lehetőséget kapunk. A képek megjelenítésénél még azt is meghatározhatjuk, hogy a nagy (vagy éppen kicsi) képekkel mi történjen, megnyújthatjuk, kicsinyíthetjük, vagy éppen levághatjuk a kilógó részeket.
	A DBListBox egy listaelem, amely az lehetséges értékeit nekünk kell feltölteni tervezési- vagy futási időben. Ebből a listából kiválasztott elem a hozzárendelt mezőbe kerül tárolásra.
	A DBListBox komponenshez nagyon hasonló vezérlőelem a DBComboBox, azonban itt a választható elemeket egy legördülő listából választhatjuk ki.
	Abban az esetben, ha egy mező értéke csak igaz, vagy hamis lehet, akkor használhatjuk a DBCheckBox vezérlőelemet. A logikai érték kerül eltárolásra az adattábla megfelelő mezőjében.
	Egy adatbázishoz kapcsolódó választógomb-csoportot hoz létre a DBRadioGroup komponens. A gombok egy listában tárolódnak és a kiválasztott elem sorszáma kerül be a tábla mezőjébe.
	Számos esetben előfordul, hogy egy listaelem sorait egy másik táblából kellene feltölteni. Ennek biztosítására készült a DBLookupListBox komponens, amely rendkívül jól használható több táblás adatbázisok esetében.
	A DBLookupListBox vezérlőelem egy speciális változata a DBLookupComboBox, amikor is az adatok egy legördülő listába kerülnek be.

	A DBRichEdit komponens nagyon hasonlít a DBMemo vezérlőelemhez, mivel itt is több soros információt tárolhatunk, azonban az ebben megjelenített tartalomhoz formátumot is hozzárendelhetünk. Lehetőség van szövegek stílusát, betűtípusát, betűméretét megváltoztatni.

5.3. A Komponensek használata

- File\New Data Module (TDataModule): speciális form.

- TTable: a komponens kapcsolatot teremt a tábla és a komponens között.
  - DatabaseName: adatbázis elérési útja (vagy alias).
  - TableName: tábla neve.
  - Active: az adatbázis aktiváltságát mutatja (boolean).
  - RecNo: aktuális rekord sorszáma.
  - RecordCount: összes rekord száma.
  - Exclusive: a tábla kizárólagos, más program nem férhet hozzá.
  - ReadOnly: a tábla csak olvasható.
  - DisableControls/EnableControls: a tábla és a DBGrid kapcsolatának megszűntetése, engedélyezése.
  -Fields: tömb, amely a tábla mezőit tartalmazza.
  - FieldCount: a tömb elemeinek száma.

- TDataSource: a Table és adatbázis-kezelő komponensek közötti kapcsolatot tartja fönn.
  - DataSet: a TTable neve.

- TDBGrid: táblázat megjelenítője.
  - DataSource: a TDataSource komponens neve.

Adatbázis mezői

Egy mező tulajdonságai:

- Értéke: Value: érték, AsString: stringbe konvertálás, AsInteger: integerbe váltá, AsFloat: lebegőpontosba váltás, AsBoolean: logikai, AsDateTime: dátum és idő, AsCurrency: pénzformátum, AsVariant: bármilyen típust felvehet futás közben.
- IsNull (boolean): azt mutatja, hogy a mező üres-e.
- FieldName: a mező táblabeli neve.
- Name: amező nevét mutatja (alapban táblanév és mezőnév konkatenálása).
- Alignment: a mezőérték megjelenésének igazítása.
- DisplayFormat: a mező megjelenítési formátuma (pl. 0 darab: "0 darab", # darab: "darab").
- DisplayLabel: a mező megjelenési neve.
- Displ: a mező szélessége
- MinValue, MaxValue: a mező minimális és maximális értéke.
- ReadOnly (boolean): a mező csak olvasható.
- Visible (boolean): a mező láthatóságát adja meg.
- EditMask: a mező bemeneti formátuma (pl. EditMask:=!(999)000-000;1;_ - az "1" azt jelenti, hogy a zárójelek és szóközök is tárolásra kerülnek, a "_" helyettesítő karakter, megjelenése: "(___)___ -___).
- CustomConstraint: a mező beviteli értékét határozza meg (pl. CustomConstraint:=(Nem='Férfi') or (Nem='Nő')).
- ConstraintE: ha a CustomConstraint jellemzőbe írt feltétel nem teljesül, hibaüzenetet ad (pl. ConstraintErrorMessage:='A személy neme csak "Férfi" vagy "Nő" lehet!").
- RecordSize: az aktuális rekord mérete byte-ban.
- FieldByName(mező): a mező sorszámát adja.
- FieldValues: az adott rekord mezőinek tartalmát adja.
- ClearFields: A mező összes értékének törlése.

Keresés

- Locate: ha megtalálja a keresett rekordot, akkor aktuálissá teszi.

Locate (const KeyFields:string; const KeyValues:variant; Options:TLocateOptions):boolean;

- KeyFields: keresendő mező neve.
- KeyValues: keresett érték.
- Options: loCaseInsitive (a kis- és nagybetűket nem különbözteti meg), loPartialKey (részszóra keres).

- LookUp: a keresett rekord megadott értékeit adja vissza.

LookUp (const KeyFields:string; const KeyValues:variant; const ResultFields:string:variant);

Resutl: azoknak a mezőknek a nevei, melyeket visszatérési értékként akarunk kapni.

Adatok szerkesztése

- Állapot lekérdezése: State, State: tábla állapota.
  - dsInactive: lezárt állapot,
  - dsBrowse: alapállapot,
  - dsEdit: szerkesztés alatt áll,
  - dsInsert: az aktuális rekord új (beszúrás alatt áll).

- Új rekord beszúrása: Append (indexelt esetén a pozíció után, egyébként a végére - a State dsInsert-re vált)

```
DataModule1.Table1.Append;
DataModule1.Table1KOD.Value:='0012';
```

- BeforeInsert: beszúrás előtti elem.
- AfterInsert: beszúrás utáni elem.
- OnNewRecord: minden beszúráskor létrejövő esemény (pl. alapérték-beállításra való).

Pl. DataModule1.Table1FIZMOD.Value:=1;

- AppendRecord(const Values: array of const): megadhatjuk a beszúrt rekord értékét .

Pl. DataModule2.Table1.AppendRecord(['0200',Date,'971211',]);

- Új rekord beszúrása az aktuális rekord elé: Insert (State=dsInsert)

DataModule1.Table1.Insert;

DataModule1.Table1KOD.Value:='0012';

- Aktuális rekord szerkesztése: Edit (State=dsEdit)

DataModule1.Table1.Edit;

DataModule2.Table1.SetFields(['990417',Date]); - Mező típusának és értékének beállítása.

- Adatbázis változtatásának mentése: Post

DataModule1.Table1.Post;

Hiba figyelése annak, hogy módosítható vagy szerkeszthető-e a rekord:

If DataModule2.Table1.State in[dsEdit, dsInsert] then DataModule2.Table1.Post;

- OnPostError: Post-hiba esetén létrejövő esemény.

- Adatbázis változtatásának visszavonása: Cancel

DataModule1.Table1.Cancel;

- Aktuális rekord törlése: Delete (nem kérdez rá)

- OnDeleteError: Delete-hiba esetén létrejövő esemény.
- DataSet: aktuális adatbázis.
- EDataBaseEr: adatbázis-hiba esetén létrejövő esemény.
- Action: daFail (hibajelentés angolul),
  daAbort (törlés megszüntetése hibaüzenet nélkül),
  daRetry (törlés újra megpróbálása).

Adatbázis megnyitása, lezárása

Az aktiváltság is hasoló tulajdonság - True esetén nyitott, Fasle esetén zárt (de nem megnyitott és lezárt).

- Megnyitás: Open
  - BeforeOpen: megnyitás előtt létrejövő esemény.
  - AfterOpen: megnyitás után létrejövő esemény.

- Bezárás: Close
  - BeforeClose: bezárás előtt létrejövő esemény.
  - AfterClose: bezárás után létrejövő esemény.

Pozícionálás

Ha nem alapállapotban van a tábla (State<>dsBrowse) akkor Post-ot hajtanak végre.
- Első rekordra ugrás: First, DataModule1.Table1.First;
- Utolsó rekordra ugrás: Last, DataModule1.Table1.Last;
- Előző rekordra ugrás: Prior, DataModule1.Table1.Prior;
- Követező rekordra ugrás: Next, DataModule1.Table1.Next;
- Több rekordnyi ugrás: MoveBy(elem:integer), DataModule1.Table1.MoveBy(-5)

- BeforeScroll: léptetés előtt létrejövő esemény.
- AfterScroll: léptetés után létrejövő esemény.

- BOF (boolean): a tulajdonság azt mutatja, hoelső-e az elem.
- EOF (boolean): a tulajdonság azt mutatja, hogy utolsó-e az elem.

- Könyvjelző elhelyezése: GetBookMark, bo:= DataModule2.Table1.GetBookmark;

- Ugrás a könyvjelzőre: GotoBookMark(könyvjelző)
DataModule2.Table1.GotoBookMark(bo);

- Könyvjelző törlése: FreeBookMark(könyvjelző)
DataModule2.Table1.FreeBookMark(bo);

Két adatbázis összekapcsolása

Két adatbázist -egy mezőn keresztül kapcsolunk össze, a segéd adatbázistól kiindulva.
1. MasterSource=TDataSource (a fő adatbázis).
2. Property\Detail Fields =KOD (az összekapcsoló mező - lehet más nevű, mint a MasterFiled).
3. MasterField: KOD (az összekapcsoló mező).
2. Add gombbal létre kell hozni a kapcsolatot.

Számított mezők (Calculated Field)

A kalkulált mező olyan mező, mely másik mezők értékéből számítható ki. Nem kell új mezőt fizikailag létrehoznunk.

- TTable menüszerkesztő ablakában (2 klikk a komponensen) jobb egérgomb, majd New Field: Type=Float (ha lebegőpontos kell), FieldType=Calculated.

- TTable Event\OnCalcField: annyiszor jön létre, ahány rekordra kell kiszámítani az adott értéket, értékeket, így egy-egy eseménykor mindig az aktuális rekord értékeiből számíthatjuk ki a kívánt értékeket.

A kalkulált mező csak olvasható!

Kapcsolt mezők (Lookup Field)

Egy mező adatait felesleges lehet állandóan ismétlődően megadni (pl. NEME='Férfi'). Hatékonyabb, ha kódot rendelünk hozzá, hiszen így kisebb az adatbázis. Ehhez az aktuális táblát össze kell kapcsolni a másik (kereső)táblával egy-egy összekapcsoló mezőn keresztül, és a keresőtábla kívánt mezőjéből átvesszük az értékét.

- Új mező létrehozása: TTable\New Field\Type=LookUp, KeyFields: az alaptábla összekapcsoló mezője, Dataset: a keresőtábla neve, LookUp Keys: a keresőtábla összekapcsoló mezője, Result Field: a keresőtáblának ebből a mezőjéből vesszük át az eredményt.

Mezők értékének korlátozása

- TTable property-re kattintva előjön a segédablak, ahol feltételeket adhatunk (Add).
- CustomConstraint: szöveges feltételadás (lásd a II. alcím alatt).
- DisableConstraints/EnableConstraints: korlátozás átmeneti tiltása/engedélyezése.

Rekord végleges törlése: DbiPackTable, dBase-nél: Pack

A Delete nem fizikailag törli a rekordot, valójában csak törlésre jelöli ki.

- DbiPackTable;
E művelet előtt az adatbázist kizárólagossá kell tenni (ilyenkor nem férhet hozzá más program): TTable.Exclusive:=True;

- A törölt rekordok megjelenítése:
  - DbiSetProp;
  - iPropValue (boolean): a törölt elemek láthatósága.
  - Refresh: elemek láthatóságának frissítése.

Adatbázis létrehozása és törlése

- Adatbázis törlése: DeleteTable;
- Adatbázis létrehozása: CreateTable;

  Előtte:

  - A táblázat adatait megadjuk.
  - FieldDefs propertiesben definiáljuk a mezőket (a FieldDefs properties TFiledDef típusú).
  - IndexDefs: propertiesben megadjuk az indexeket.

- Újmező létrehozása:

  Add(Name;DataType; Size; Required);

  - Name: az új mező neve.
  - DataType: az új mező típusa.
  - Size: az új mező mérete (nem kötelező minden esetben megadni).
  - Required: maradhat-e üresen.

- Clear: az összes felvett elem törlése.

7. Count: az aktuálisan megadott mezők száma.

Items properties: egy TFieldDefs típusú elemeket tároló tömb, melyben pl. amezők neveit tárolhatunk (.Name esetén) – 0..n-1. Az Items-ben a Find-del kereshetünk. IndexOf(Name): azt mutatja, hogy a Name hányadik elem az indexben.

Rekordok törlése Master-Detail kapcsolatban

1. A segédtábla törlése rekordonként (a kapcsolt rekordok látszanak).
2. A főtábla törlése.

TDataSource

Az Table és a DBGrid közötti kapcsolatot tarja fenn.

- State (állapot):
  - doInactive: inaktív,
  - doBrowse: megjelenítés alatt,
  - doEdit: szerkesztés alatt,
  - doInsert: új elem beszúrása történik,
  - SetKey: keresés vagy nszűrés alatt,
  - doCalcFields: kalkulált mező kiszámítása történik
  - doFilter: szűrés alatt.

- Események (Events):
  - onStateChange: az állapot megváltozott,
  - onDataChange: az adatbázis tartalma megváltozott,
  - onUpdateData: Post esemény létrejött,
  - AutoEdit (boolean): Ha True bármikor változtatható, ha False, akkor az Edit parancs után.

TBachMovie: több adatbázis lemásolása

1. Source: itt kell megadni, hogy honnan vegye a rekordokat (TDataSet).
2. Destination: itt kell megadni, hogy hova tegye a rekordokat (TTable).
3. DataBaseName: az adatbázis neve.
4. TableName: a tábla neve.
5. Mode: mód (Indexeltek legyenek az adatbázisok.)
   - batAppend: a forrásadatbázis összes rekordja hozzáfűződik a céladatbázishoz.
   - vatUpdate: a forrásadatbázis egyező mezőit átírja a más értékeket tartalmazó céladatbázisba.
   - batCopy: akár nemlétező cél adatbázisba írja a forrásadatbázist.
   - batDelete: a céladatbázisban törlődnek azok a rekordok, amelyek egyeznek a

forrásadatbázis rekordjaival.

6. Execute: végrehajtás

- Programfutás alatt:

Pl.

```
With BatchMove1 do
begin
  Source:=Query1;
  Destination:=Table1;
  Mode:=batCopy;
  Execute;
end;
```

- Fejlesztési időben: Jobb egérgomb a TBatchMove-n\Execute.

Egyéb lehetőségek:

- ChangedTableName: itt állománynevet adhatunk meg, amely néven létrehoz egy Paradox adatbázist (Execute utasítás esetén), amelybe a változtatott rekordok kerülnek.
- ProblemTableName: az itt megadott állománynéven létrejön egy Paradox adatbázis, amelybe a hibás rekordok kerülnek.
- KeyViolTableName: az itt megadott állománynével létrejön egy Paradox tábla, amelybe a forrásadatbázisból a rekordok bekerülnek.
- AbortOnProblem: (boolean): ha True, akkaz első hibás rekord másolása után a folyamat megszakad.
- AbortOnKeyViol: ha True, akkor az első rekord másolása után kilép.
- MovedCount: azt mutatja, hogy hány rekord került módosításra vagy másolásra a céladatbázisban.
- ChangedCount: azt mutatja, hogy a ChangedTableName-ben megadott adatbázisba hány rekord került.
- ProblemCount: azt mutatja, hogy hány hibás rekord van.
- KeyViolCount: megadja a hibás rekordok számát.
- CommitCount: azt mutatja, hány rekord került át sikeresen az új táblába.
- RecordCount: itt korlátozhatjuk, hogy maximum mennyi rekord kerülhet a

forrásadatbázisból a céladatbázisba. 0 esetén nincs korlátozás.
- Mappings: jó, ha a forrás- és céladatbázis mezőnevei nem azonosak. Itt megadhatjuk, hogy a forrás- és céladatbázis mely mezőit tekintse azonosnak.

TDBGrid

Az adatbázis táblázatos megjelenítésére szolgál. 1. elem: 0, utolsó elem: Count-1.

1. DataSource-szel kapcsolni kell a Table-höz.

Tulajdonságok:
- Columns: fejléc (oszlopok címkéje - mezők) beállítása.
  Új fejléc hozzáadása: DBGrid.Colums.Add;
  - FieldName: a D-szel kapcsolódó tábla egy mezője.
  - Title: a megjelenő fejléc formázása és szövege.
- Items: a TColumn típusú tömb elemei ezen keresztül érhetőek el.
- Count: azt mutatja, hogy hány oszlop van.
  Pl.
  For i:=0 to DBGrid1.Columns.Count-1 do
  begin
    DBGrid1.Columns.Items[i].Color:=clRed;
  end;
- Clear: elem törlése (az adatbázisból is törli - nem fizikailag).
- FieldName: mezők nevei.
  Pl.
  DBGrid.Columns.Add.FieldName:='Datum';
  DBGrid.Columns.Items[].FieldName:=Fizmod';
- Field: az adaázis adott mezőjét érhetjük el vele.
  Pl. Label1.Caption:=DBGrid.Columns.Items[].Field.AsString;
- Color: a táblázat adott oszlopának a háttérszíne.
  Pl. DBGrid1.Columns.Items[].Color:=clRed;
- Font: a táblázat adott oszlopának a betűtípusa.

  - Style: stílus,
  - Color: karakterszín.
  Pl. Font.Style:=[];
- Alignment: szöveg igazítása.
  - taLeftJustify: balra igazítás,
  - taRightJustify: jobbra igazítás,
  - taCenterJustify: középre igazítás.
- PopupMenu: legördülő menüt rendelhetünk egy oszlophoz (futás közben jobb egérgombbazl érhetjük el).
- ReadOnly: a kiválasztott oszlop csak olvasható.
- ButtonStyle: itt megadhatjuk, hogy mi történjen, ha a megadott oszlopba lépünk.
  - cbsName: alapértelmezett neve,
  - cbsEllipsis: egy nyomógomb jelenik meg, amelyben egy onEditButtonClick esemény keletkezik.
  - cbsAuto: oszlopba lépéskor ComboBox jön létre, ahonnan adatokat juttathatunk a cellába. PickList-ben írhatjuk le, mit tartalmazzon a ComboBox. A DropDownRows-ban megadhatjuk, hogy egyszerre hány elem látszódjon a ComboBox-ban.
- Width: itt az oszlop szélességét korlátozhatjuk.
   Pl. DBGrid1.Columns.Items[].Width:=20;
- Title: fejléc szövege.
  - Caption: a fejléc megjelenési neve,
  - Align: szöveg igazítása,
  - Color: háttér színe,
  - Font: betűtípus.
  Pl.
  DBGrid1.Columns.Items[].Title.Caption:='Dátum';
  DBGrid1.Columns.Items[].Title.Color:=clRed;
- FieldCount: a mezők száma.
- SelectField: aktuális mező
- SelectedIndex: az aktuális oszlop sorszáma.
  Pl. Label1.Caption:=DBGrid1.SelectedField.AsString;
- TitleFont: fejléc betűtípusa (az összes oszlop fejlécére érvényes).

Pl. DBGrid2.TitleFont:=FontDialog1.Font;

- EditorMode: azt mutatja, hogy szerkesztés alatt áll-e a grid.
- Option: beállítások (boolean típusú elemek)
  - dgEditing: ha True, akkor a gridben is szerkeszthetjük az adatokat,
  - dgAlwaysShowEditor: True esetén automatikusan szerkeszthető a mező, False esetén Enter-re vagy F2-re,
  - dgTitles: ha True, a fejléc látható,
  - dgIndicator: True esetén a grid széli indikátor látható,
  - dgColumnResize: ha True, az oszlopok mérete változtatható,
  - dgColLines: True esetén az oszlopelválasztó vonal látható,
  - dgRowLines: True esetén a sorelválasztó vonal látható,
  - dgTabs: ha True, akkor a gridben érvényes a TAB és a SHIFT+,
  - dgConfirmDelete: True esetén a CTRL+-tel törölhető az aktuális sor,
  - dgMultiSelect: ha True, akkor a CTRL+nyilakkal (részenként) vagy a SHIFT+nyilakkal (egységként) több elem is kijelölhető,

Események:

- onTitleClick: oszlop fejlécére való kattintás eseménye. TColumn változó jön létre, amely az aktuális oszlop számát adja meg.

  Pl.

  Label1.Caption:=Column.Title.Caption;

  Column.Title.Color:=clRed;
- onColEnter: ez az esemény akkor jön létre, amikor egy oszlopra lépünk.
- onColExit: akkor jön létre ez az esemény, ha egy oszlopról kilépünk.

  Pl. Label1.Caption:='Ez a táblázat'+(DBGrid1.SelectedIndex)+'.oszlopa';
- onCellClick: cellán való kattintás eseménye. Létrejön egy TColumn változó, amely az aktuális oszlop számát adja meg.
- onColumnMoved: akkor jön létre ez az esemény, ha egy oszlopot áthelyezünk. Létrejön egy FromIndex és egy ToIndex változó, melyek azt mutatják, hogy hányadik oszlop hányadik oszlop helyére kerül.

TDBNavigator

Adatbázis-mozgató gombsor.

1. Létre kell hozni a kapcsolatot a DataSource-ön keresztül.

TNavigateBtn típusú konstansok:
- nbFirst: első rekordra ugrik (TTable.First).
- nbPrior: egyel visszalép (TTable.Prior).
- nbNext: következő rekordra lép (TTable.Next).
- nbLast: utolsó rekordra ugrik (TTable.Last).
- nbInsert: új rekordot szúr be.
- nbDelete: aktuális rekordot törli.
- nbEdit: szerkesztőmódba lép.
- nbPost: elmenti a változtatásokat.
- nbCancel: érvényteleníti a változtatásokat.
- nbRefresh: újraolvassa az adatokat (ha másképpen is megváltozhatott az adatbázis).

További lehetőségek:
- VisibleButton: azt mutatja, melyik gombok látszódnak.
- Hint: az egész komponens magarázószövege.
- Hints: nyomógombok magyarázószövege (sorrendben egy-egy sor egy-egy nyomógombhoz tartozik).
- ConfirmDelete: megerősítéskérést végez a törléshez (angolul).
- BtnClick(TNavigateBtn): valamely gomb megnyomásának szimulálása.
  Pl. DBNavigator1.BtnClick(nbFirst);
- Flat: True esetén a nyomógomb körvonala csak akkor látszódik, ha fölötte állunk az egérrel, és nincs letiltva.
- BeforeAction: ez az esemény akkor jön létre, ha lenyomtunk egy gombot, de még nem hajtódott végre a parancs. Kapunk egy TNavigateBtn típusú Button változót, amely azt mutatja, hogy melyik gombot nyomtuk meg.
  Pl.

```
If Button=nbDelete then
begin
  If MessageDlg('Törlés megerősítése', mtConfirmation, mbOkCancel, 0)<>mrOk then
Abort;
```

- onClick: nyomógomb lenyomása, és a parancs végrehajtása utáni esemény. Button változót kapunk, amely megadja, hogy melyik gomb lett lenyomva.

TDBText

Az aktuális mező megjelenítése szolgál.

- DataSource: adatbázis.
- DataField: mező.
- Field: program futása alatt a mezőt ezzel érhetjük el.

```
Pl.
DBText1.DataSource1.DataSet.Edit;
DBText1.Field.Value:=Date;
```

- AutoSize (boolean): Tue esetén a komponens szélessége automatikusan változik.
- Transparent (boolean): True esetén a DBText mögött elhelyezkedő komponens is látható.

TDBEdit

Az aktuális mezőt megjeleníthetjük és szerkeszthetjük a beviteli mezőben. A komponensbe írt stringre (vagy konvertált értékre) változtatja a mezőt, ha a tábla módosítható.

- DataSource: adatbázis.
- DataField: mező.
- CharCase: a beírt szöveg megjelenési módja.
  - ecLowerCase: minden betűt kisbetűre vált,
  - ecNormal: alap,
  - e: minden betűt nagybetűre vált.
- ReadOnly (boolean): True esetén csak olvasható.

- PasswordChar: ha nem #0, akkor az ide beírt karakter fog megjelenni begépelés közben (pl. *). Az adatbázisban az eredeti adat fog megjelenni.
- MaxLength: a bevitmező maximum hosszát adja (ha nem 0).
- AutoSelect (boolean): ha True, akkor a komponensre fókuszálás után automatikusan kijelöli a benne foglalt teljes stringet.
- Modified (boolean): azt mutatja, hogy megváltozott-e a szöveg tartalma.
  Pl. If Edit1.Modified then...
- SelText: a kijelölt szöveget tárolja el.
  - SelStart: a kijelölés első karaktere,
  - SelLength: a kijelölt string hossza.
- Text: a DBEdit-et Text stringgel változtathatjuk meg és kérhetjük le.
- Field: egy mezőt érhetünk el vele
  Pl. DBEdit.Field.AsString:='Hahó!';
- Clear: törli a komponens tartalmát.
- SelectAll: kijelöli a teljes szöveget.
- ClerSelection: a kijelölt szövegrészt törli.
- CopyToClipboard: a vágólapra másolja a kijelölt szöveget.
- CutToClipboard: kivágja a vágólapra a kijelölt szöveget.
- PasteFromClipboard: a vágólapra másolt szöveget beilleszti a megadott komponensbe.
- onChange: létrejön ez az esemény, ha a DBEdit tartalma megváltozik.

TDBMemo

Többsoros beviteli mező.

- DataSource: adatbázis.
- DataField: ő (az adatbázis mezője Memo típusú legyen!).
- Field: egy mezőt érhetünk el vele.
  DBMemo1.Field.DisplayLabel:=’Ez egy mező’;
- AutoDisplay (boolean): ha True, akkor automatikusan megjelenik a DBMemo tartalma, amikor az adatbázisban lépkdünk.
- ReadOnly (boolean): ha True, akkor csak olvasható.

- AlignMent: szöveg igazítása.
  - taLeftJustify: balra igazítás,
  - taCenter: középre igazítás,
  - taRightJustify: jobbra igazítás,
- MaxLength: a maximálisan megadható karakterek számát adja meg.
- ScrollBars: görgetősávok.
  - ssNone: nincs,
  - ssHorizontal: vízszintes,
  - ssVertical: függőleges,
  - ssBoth: vízszintes és függőleges.
- WantTabs (boolean): True esetén a TAB lenyomását a DBMemo-n belül érzékeli.
- WordWarp (boolean): True esetén a sor végélemaradt szövegrész átkerül a következő sorba, False esetén az egész szó.
- Lines: a DBMemo sorai.
  - Add(string): sor hozzáadása.
- Modified (boolean):a szöveg megváltozottságát jelzi.
- LoadMemo: egy file tartalmával tölti fel a DBMemo-t.
- SelectAll: kijelöli a teljes szöveget a komponensen belül.
- Clear: törli a DBMemo tartalmát.
- ClearSelection: a kijelölt szöveget törli.
- CopyToClipboard/CutToClipboard/PasteFromClipboard: vágólap használata.

TDBImage

Az adatbázisban lévő képeket jeleníti meg.

- DataSource: adatbázis.
- DataField: mező (az adatbázis mezője dBase esetén binary, egyéb esetén egyéb képformátumú legyen!).
  Az adatbázismezőbe új képet beilleszteni, illetve onnan képet kimásolni a vágólapon keresztül lehet (CTRL+C, CTRL+X, CTRL+V).
- Field: egy mezőt érhetünk el vele.

- AutoDisplay (boolean): True esetén mindig megjeleníti az aktuális rekord DBImage-ét, False esetén a LoadImage eljárással kell betöltenünk.
- Stretch (boolean): ha True, a kép méretét a komponens méretéhez igazítja (nyújtja).
- QuickDraw (boolean): True esetén nem használ speciális palettát a kép kirajzolásához, hanem csak 256 színt. (Alapesetben True.)
- BorderStyle: itt azt lehet beállítani, legyen-e keret.
  - bsNone: nincs keret,
  - bsSingle: van keret.
- Picture: közvetlenül elérhetjük vele a képet.
  Pl.
  BitBtn(PaintBox1.Canvas.Handle, 0, 0, DBImage1.Picture.Width, DBImage1.Height, DBImage1.Bitmap.Canvas.Handle, 0, 0, srcCopy);
- ReadOnly (boolean): ha True, akkor csak olvasható.
- LoadImage: egy file tartalmával tölti fel a DBImage-et.
- CopyToClipboard/CutToClipboard/PasteFromClipboard: vágólap használata.

TDBListBox

Adatbázishoz igazodó listadoboz. A listadobozban kiválasztott elem kerül a rekordmezőbe. Ha a rekord kiválasztott eleme megegyezik a listadoboz egyik elemével, akkor az az elem a listadobozban is ki lesz választva.

- DataSource: adatbázis.
- DataField: mező.
- Field: egy mezőt érhetünk el vele.
- Items: a listadoboz elemei, a rekord mezői a listadobozban.
  - Add(elemnév): új elem hozzáadása.
- ReadOnly (boolean): ha True, akkor csak olvasható.
- onDrawItem: olyan esemény, amely akkor jön létre, ha kirajzol egy listaelemet.
- Style: stílus.
  - lbStandard: alap,
  - lbOwnerDrawFixed: állandó méretű elemek,

- lbOwnerVariable: változó méretű elemek.
- ItemHeight: az egyes elemek magassága (Style=lbStandard esetén nem kell változtatnunk).
- IntegralVariable: a listadoboz szélessége a listaelemek egészszámú megjelenéséhez igazodik.
- ItemIndex: a kiválasztott elem sorszámát adja meg (0-tól Count-1-ig), ha értéke –1, akkor nincs kiválasztott elem.
- Selected (boolean tömb): minden egyes elem kiválasztottságát mutatja (True esetén a megadott elem kiválasztott).

TDBComboBox

Adatbázishoz igazodó ComboBox. A listából kiválasztott elem kerüa mezőbe. Ha megváltozik az aktuális rekord, akkor komponens tartalma is megváltozik arra az értékre, amelyet az adatbázis tartalmaz. Ha az adatbázisban olyan érték van, amely nem szerepel a ComboBox-ban, akkor semmi sem jelenik meg komponensben. Ha egy másik elemet választunk ki, akkor az adatbázis automatikusan szerkeszthető üzemmódba kerül.

- DataSource: adatbázis.
- DataField: mező.
- Field: egy mezőt érhetünk el vele.
- Items: a listadoboz elemei, a rekord mezői a listadobozban.
- ReadOnly (boolean): ha True, akkor csak olvasható.
- ItemIndex: a kiválasztott elem sorszámát adja meg (0-tól Count-1-ig), ha értéke –1, akkor nincs kiválasztott elem.
- Sorted (boolean): True esetén rendezett a lista.

TDBCheckBox

Adatbázishoz igazodó, két- vagy háromelemű doboz (logikai típusú értékekhez ajánlják).

- DataSource: adatbázis.
- DataField: mező.
- Field: egy mezőt érhetünk el vele.
  DBCheckBox1.Field.DisplayValues:=’igaz; hamis’;
- ReadOnly (boolean): ha True, akkor csak olvasható.
- AllowGrayed (boolean): esetén háromértékű (iksz/szürke iksz/üres), False esetén kétértékű.
- Checked: elemek kiválasztottságát mutatja.
- State: háromelemű esetén az állása.
  - cbUnchecked: nem kiválasztott,
  - cbChecked: kiválasztott,
  - cbGrayed: harmadik érték (szürkített).
- ValueChecked: itt lehet megadni, hogy milyen szöveges érték feleljen meg az igaz értéknek (pontosvesszővel több értéket is megadhatunk).
  Pl. CheckBox1.ValueChecked:=’igaz;i’;
- ValueUnchecked: a hamis érték szöveges megfelelője.

TDBRadioGroup

Erre a komponensre akkor lehet szükség, ha véges számú elem közül csak egyet kell mkiválasztani.

- DataSourcs: adatbázis.
- DataField: mező.
- ReadOnly (boolean): a komponens csak olvasható.
- Items: a komponens elemei.
- Columns: a megjelenítő oszlopok száma.
- Values: az értékek soronkénti hozzárendelése az elemekhez.
- Caption: a fejléc szövege.

- ItemIndex: megmutatja, hogy hányadik az aktuális elem (0-tól kezdődik a számolás).
- CanModify (boolean): True esetén az adott mező értékét megváltoztathatjuk.

TDBLookUpListBox

A lista elemeit nem mi adjuk meg, hanem egy másik adatbázis szolgáltatja.

- DataSource: a főadatbázis, ahová az elemek kerülnek.
- DataField: a főadatbázis azon mezője, ahová a ListaSource-ben és a KeyField-ben megadott mező tartalma kerül.
- ListSource: a segédadatbázis, ahol a lista elemeit tároljuk.
- ListField: a lista elemeit tároló segédadatbázis mezője (több mezőt is megadhatunk ';'-vel - több oszlopban jelenik meg).

```
Pl.
With DBLookUpListBox1 do
begin
  ListField:='NEV;CÍM;TELEFON';
end;
```

- KeyField: a segédadatbázisban ennek a mezőnek az értéke kerül be a főadatbázisba.

```
Pl.
With DBLookUpListBox1 do
begin
  ListSource:=DataSource2;
  ListField:='NEV';
  KeyField:='KOD';
  DataSource:=DataSource1;
  DataField:='VEVOKOD';
end;
```

- Field: a főadatbázis egy mezője.
- ListFieldIndex: több mező megadása esetén a mező száma (0-tól).

Pl. ListField:='NEV;CÍM;TELEFON';
Ha a ListFieldIndex=1, akkor a CÍM mező alapján keresi a beírt értéket.

- KeyValue: KeyField-ben megadott mező értéke.
  Pl. Label1.Caption:=DBLookUpListBox1.KeyValue;
- SelectedItems: az aktuálisan kiválasztott elem szövege.
- RowCount: az ablak sorainak száma.

TDBLookUpComboBox

- DataSource: a főadatbázis, ahová az elemek kerülnek.
- DataField: a főadatbázis azon mezője, ahová a ListaSource-ben és a KeyField-ben megadott mező tartalma kerül.
- ListSource: a segédadatbázis, ahol a lista elemeit tároljuk.
- ListField: a lista elemeit tároló segédadatbázis mezője (több mezőt is megadhatunk ';'-vel - több oszlopban jelenik meg).
- KeyField: a segédadatbázisban ennek a mezőnek az értéke kerül be a főadatbázisba.
- Field: a főadatbázis egy mezője.
- ListFieldIndex: több mező megadása esetén a mező száma (0-tól).
- KeyValue: a KeyField-benmegadott mező értéke.
  Pl. Label1.Caption:=DBLookUpListBox1.KeyValue;
- DropDownAlign: az elemek igazítása.
  - daLeft: balra igazítás,
  - daRight: jobbra igazítás,
  - daCenter: középre igazítás.
- DropDownRows: a lista megjelenő elemeinek száma.
- DropDownWidth: a lista szélessége.
- ListVisible (boolean): True esetén a lista látható.
- Text: az aktuálisan szerkesztett mező tartalma.

TDBRichEdit

Adatbázisban tárolt Rtf file-ok megjelenítésére szolgál.

- DataSource: adatbázis.
- DataField: adatbázis mezője.
- Field: a mező elérése.
- ReadOnly (boolean): csak olvasható.

TDBCtrlGrid

Az adatbázis többrekeszes megjelenítését teszi lehetővé. Egy rekesz több mezőt is tartalmazhat.

- TDBText
- TDBEdit
- TDBMemo
- TDBImage
- TDBCheckBox
- TDBComboBox
- TDBLookUpComboBox
- DataSource: itt azt lehet megadni, hogy a megjelenítő eszközök mely adatbázishoz tartoznak.
- ColCount: a táblázat oszlopainak számát adhatjuk itt meg.
- RowCount: a táblázat sorainak számát adhatjuk itt meg.
- AllowDelete (boolean): True esetén CTRL+-tel törölhetjük a mező tartalmát.
- AllowInsert (boolean): True esetén CTRL+Insert-tel új elemet szúrhatunk be.
- EditMode (boolean): True esetén az adatbázis a CtrlGrid-en keresztül is szerkeszthető.
- Orie: azt lehet itt megadni, hogy függőlegesen vagy vízszintesen helyezkedjenek-e el.
- PanelBorder: a rácsozást adhatjuk meg itt.
- PanelCount: az egyidőben látszó rekordok számát adja meg.
- PanelIndex: azt mutatja, hogy hányadik panelon található az aktuális rekord.

- SelectedColor: a fókuszált panelek színe.
- ShowFocus (boolean): True esetén látszani fog, hogy melyik az aktuális rekord.
- DoKey: események előidézése.
 - gkNul: semmi,
 - gkEditMode: szerkesztő üzemmódba kapcsol,
 - gkPriorTab: előző cellába lép,
 - gkNextTab: következő cellába lép,
 - gkLeft: a balra lévő cellába lép,
 - gkRight: a jobbra lévő cellába lép,
 - gkUp: a fenti cellába lép,
 - gkDown: a lenti cellába lép,
 - gkScrollUp: egy sorral feljebb görget,
 - gkScrollDown: egy sorral lejjebb görget,
 - gkPageUp: egy lappal feljebb görget,
 - gkPageDown: egy lappal lejjebb görget,
 - gkHome: az első rekordra ugrik,
 - gkEnd: az utolsó rekordra ugrik,
 - gkInsert: új rekordot szúr az aktuális rekord elé, és szerkesztőmódba kapcsol,
 - gkAppend: új rekordot szúr az utolsó elem után, és szerkesztőmódba kapcsol,
 - gkDelete: törli az aktuális rekordot,
 - gkCancel: szerkesztő üzemmód esetén nem menti el, és kilép,
- onPaintPanel: minden cella kirajzolása esetén létrejövő esemény. Létrejön egy Index változó, amely az aktuális cella sorszámát mutatja. Egyéni rajzoláshoz a Canvas-t használjuk.

TDBChart

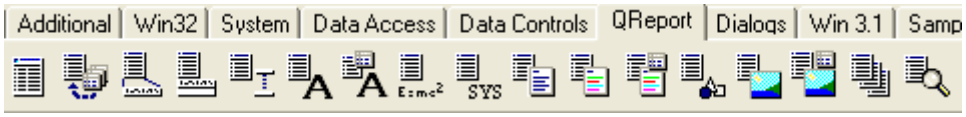
Grafikon megjelenítését teszi lehetővé az adatbázisból.

1. Kapcsolat megjelenítése a TDataModule-ban, és a TDBChart rátétele a TForm-ra.
2. Dupla klikk a TDBChart-on. Előjön egy dialógusablak.
   2.1. Add gombbal hozzunk létre egy grafikont.
   2.2. Meg kell adni a grafikon típusát.
   2.3. Series.
   2.4. DataSet-ben a tábla nevét meg kell adni.
2. A Labels-ben megadott érték meg fog jelenni minden oszlop alatt.
3. A Bar-ben megadott érték lesz minden oszlop magassága.

6. A QUICKREPORT

Az adatbázis-kezeléssel egyidős az az igény, hogy egy adatfeldolgozási folyamat eredménye jól áttekinthető formában papíron, vagy esetleg FAX-on elküldhető formában is megjelenjen. Az adatbázis-kezelő alkalmazások papírra irányuló kimenetét jelentésnek hívjuk. A Delphi rendszerben a QReport komponenslap komponenseit használjuk jelentések összeállításához. A QucikReport segítségével adatainkról az egyszerű listáktól kezdve a bonyolult, pl. diagrammokat, különbözı kimutatásokat is tartalmazó jelentésig számos nyomtatvány elkészítésére nyílik lehetőségünk.

Az egész QuickReport alapját a QuickRep komponens képzi. Amikor egy jelentést készítünk, mindig először egy QuickRep komponenst kell a form-ra helyeznünk, majd erre tesszük a többi komponenst, melyek a jelentésünket meg fogják jeleníteni. Egy jelentés különbözı szakaszokból (pl. fejléc, törzs, lábléc stb.) épül fel, és ezeken a szakaszokon helyezzük el azokat a komponenseket, amelyek az adatokat, szövegeket, képeket ténylegesen megjelenítik. Ez a komponenspaletta látható az alábbi ábrán.



6.1. A quickreport komponensei

	QuickRep komponens, amely tulajdonképpen maga a riport lesz, amely nyomtatásban meg fog jelenni. Ez egy vizuális komponens, ami azt jelenti, hogy a formon megjelenik egy lap képe, amit a formon a kívánt helyre illeszthetünk. Minden nyomtatandó anyagnak erre kell kerülnie.
	A QRSubDetail komponens arra szolgál, hogy másik adatbázisból vagy adattáblából adatokat jeleníthessünk meg a riportunkban. Tipikus felhasználási területe a táblák összekapcsolásakor létrejövő Master-Detail kapcsolat.
	QRBand egy nagyon gyakran használt komponens, mivel segítségével lehet a nyomtatandó lapon szakaszokat létrehozni. Az, hogy milyen szerepet töltsön be, a

BandType tulajdonsága fogja meghatározni. Ezek lehetnek:

rbTitle	A nyomtatandó anyag címe, csak az első oldalon jelenik meg. Ha a lapfejléc (PageHeader) be van kapcsolva, akkor a cím (Title) utána fog megjelenni.
rbPageHeader	A lap fejléce, ez minden lap tetején meg fog jelenni.
rbDetail	Ebben jelenítjük meg az adatsorokat. A Detail minden rekord esetében megjelenik egyszer. Ebben a blokkban helyezzük el a mezőneveket, amelyekből az adatokat vesszük ki. Erre szolgál a QRdbText.
rbPageFooter	Lap lábléce, minden lap alján nyomtatva.
rbSummary	Egyszer jelenik meg a nyomtatandó anyag utolsó lapjának az alján. Amennyiben van lap lábléc (PageFooter), akkor az fölött.
rbGroupHeader	Csoportfejléc, aminek segítségével a rekordok közül csoportokat hozhatunk létre valamilyen szempont szerint.
rbGroupFooter	A csoport lábléce. Minden csoport után nyomatatásra kerül. Ebben lehet például különböző kimutatásokat készíteni.
rbSubDetail	Egy speciális típus, mivel ezt a QRSubDetail komponens használja. Mi manuálisan soha ne állítsuk be ezt a típust, használjuk az említett komponenst!
rbColoumnHeader	A nyomtatandó anyag minden oszlopa felett jelenik meg. Kiválóan használható többoszlopos jelentések készítésénél az oszlopok azonosítására.
rbOverlay	Ez igazából már nem használt lehetőség, a visszafelé kompatibilitást szolgálja.

	rbChild	Szintén egy speciális típus,a QRChildBand komponenshez tartozik. Ne állítsuk be ezt a típust, használjuk az említett komponenst.
	A QRChildBand segítségével a blokkok méretét terjeszthetjük ki.	
	A QRGroup komponenssel csoportokat hozhatunk létre, ami azt jelenti, hogy a mezőket bizonyos szempont szerint csoportosítva jelenítjük meg. Minden csoporthoz tartozik fejléc, amelyben az a mező, vagy kifejezés van, amely alapján a csoportosítást el szeretnénk végezni.	
	A QRText eszköz segítségével egyszerű szöveget helyezhetünk el a lapon.	
	A QRDBText adattáblából szedi az információkat, ennek megfelelően természetesen rendelkezik DataField tulajdonsággal.	
	A QRExpr komponenst kell akkor használnunk, amikor számított értékeket szeretnénk előállítani az adattáblában tárolt adatokból. A kívánt kifejezést az Expression tulajdonságba kell illesztenünk. A segítségünkre van az Expresson Builder nevű ablak, amelyben grafikus vezérlőelemek segítségével összeállíthatjuk a kívánt kifejezést.	
	A QRSysData komponens a rendszer által szolgáltatott információk megjelenítésére használható. Azt, hogy mit mutasson, a Data tulajdonságában kell beállítani. Itt a következő értékek szerepelhetnek: qrsColumnNo — Az oszlopok számát adja vissza. qrsDate — Az aktuális rendszerdátum. qrsDateTime — Az aktuális dátum és idő. qrsDetailCount — A rekordok számát jeleníti meg.	

	qrsDetailNo	Az éppen aktuális rekord száma.
	qrsPageNumber	Az aktuális lap száma.
	qrsPageCount	Az összes oldal száma.
	qrsReportTitle	A nyomtatandó anyagunk címe.
	qrsTime	Az éppen aktuális idő.
	A QRMemo komponenssel többsoros szöveget helyezhetünk el a nyomtatandó dokumentumon.	
	A QRRichEdit komponens nagyon hasonlít a QRMemohoz, de itt már formázhatjuk is a szöveget.	
	A QRDBRichEdit -el adattáblából származó információt jeleníthetünk meg.	
	A QRShape segítségével keretet helyezhetünk el a lapon. A Shape tulajdonságban állíthatjuk be, hogy milyen keretet szeretnénk.	
	A QRImage használatával képet tehetünk a lapra.	
	Ha a táblában képet is tárolunk, annak megjelenítését teszi lehetővé a QRDBImage.	
	A nyomtatás előnézetét jeleníti meg ez a komponens.	

7.A PROGRAM BEMUTATÁSA

A bevezetőben felvázolt probléma megoldásához az adatbázist a következőképpen kell kialakítani:

Szükségünk lesz egy olyan adattáblára mely a vendégeket azonosítja be egy egyedi kód, jelen esetben a TAJ segítségével, valamint a kezelések megnevezését és egységárát tartalmazó és a kiírt kezeléseket tartalmazó táblára. Továbbá szükséges egy olyan tábla létrehozása mely kapcsoló funkciót tölt be a vendégadatok és a kiírt kezelések között. Ez a tábla tárolja a különböző időpontokat (amikor a vendég itt járt) és állítja elő a kulcsot a kiírt kezelések leválasztására a részltező táblázatban. Ez a kulcs egy összetett kulcs ami a TAJ-ból és a Dátumból tevődik össze, tehát a vendégazonosító és a dátum együtt határozzák meg a vendég által egy bizonyos időpontban felvett kezeléseket. Az időpontok tábla a könnyebb azonosítás céljából többletinformációként a szobaszámot is tartalmazza. Mivel a vendégek vissza is jöhetnek ezért fontos hogy dátum alapján elkülönítve kezeljük az adatokat. A legfontosabb táblázat tehát a kiírt kezeléseké mely tárolja a kulcsot, a kezelés kódját és egységárát is a későbbi árváltozások miatt.

A táblák a következőképpen kapcsolódnak egymáshoz:

fő tábla	részletező tábla	a részletezés alapja
TAJ	Időpontok	TAJ
Időpontok	Kiírt kezelések	TAJ+Dátum

A TAJ táblázatban kiválasztott sor TAJ száma alapján az Időpontok táblázatban megjelennek a különböző dátumok és mellettük a szobaszámok, míg az időpontokból válogatva megjelennek az adott napon kiírt kezelések az erre a célra készült táblázatban.

Szükséges állományok

Állomány neve	Mező neve	Típusa	Egyéb tulajdonság
TAJ.dbf			
	TAJ	Numerikus	kapcsoló mező
	Név	Karakteres	
Kezelesek.dbf			
	Kód	Karakteres	kapcsoló mező
	Kezelés	Numerikus	
	Egységár	Numerikus	
Idopontok.dbf			
	TAJ	Numerikus	kapcsoló mező
	Dátum	Dátum	kapcsoló mező
	Szobaszám	Karakteres	
	Fizetendő	Numerikus	
Kiirtkezelesek.dbf			
	TAJ	Numerikus	kapcsoló mező
	Dátum	Dátum	kapcsoló mező
	Kód	Karakteres	
	Kezelés	Karakteres	
	Egységár	Numerikus	
	Db	Numerikus	
	Összesen	Numerikus	számított mező

8. A PROGRAM HASZNÁLATA

8.1. Telepítés

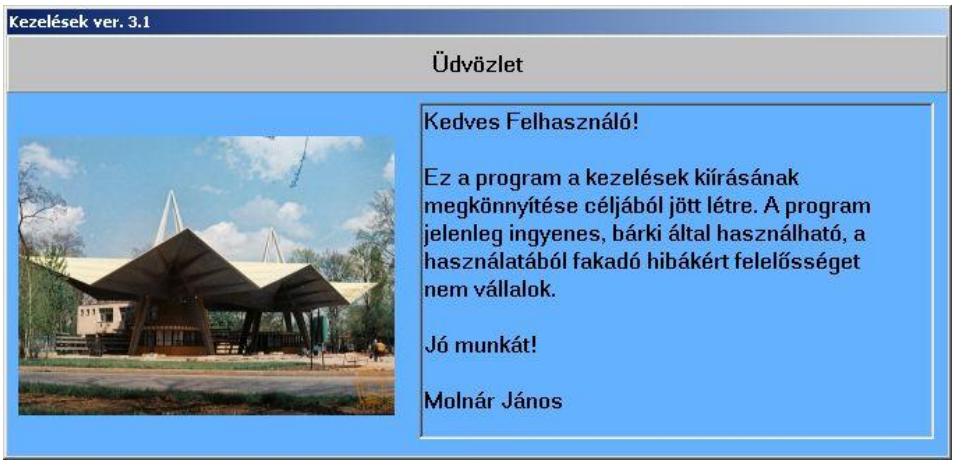
A program használatához telepíteni kell a BDE-t (Borland Database Engine) majd egyszerű másolással bárhova beilleszthető,.saját almappájába menti az állományokat. A felhasználók felvételéhez rendszergazdai jog szükséges.

8.2. Belépés

A belépni szándékozó legördülő menüből választhatja ki a supervisor által számára kijelölt, vagy általa választott 'nick'-nevet,.

Téves jelszó begépelése esetén a program egy hibaüzenetet jelenít meg.

Megfelelő név és jelszó megadásakor üdvözlőüzenetet kap a felhasználó.

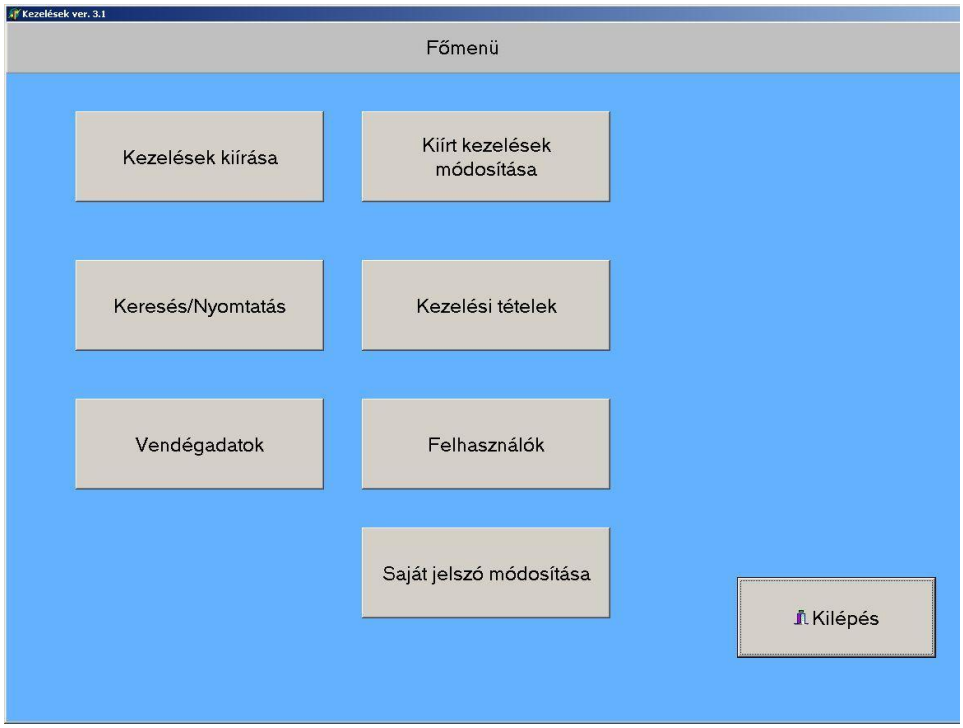


8.3. Főmenü:

A főmenü tartalmazza a felhasználók által elérhető funkciókat melyek a megfelelő gombok megnyomásával érhetők el:

- Kezelések Kiírása
- Kiírt kezelések módosítása
- Keresés/nyomtatás
- Kezelési tételek
- Vendégadatok
- Felhasználók
- Saját jelszó módosítása

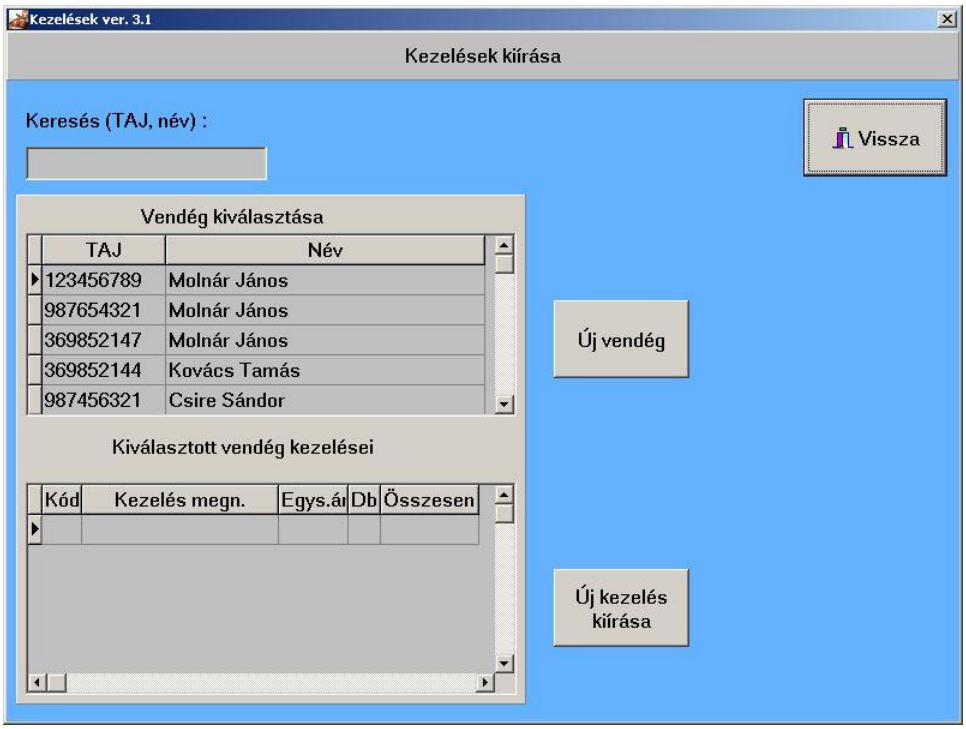
Amennyiben valamely funkcióhoz nincs az adott felhasználónak jogosultsága úgy az adott gomb letíltott állapotú, világosszürke színű lesz.



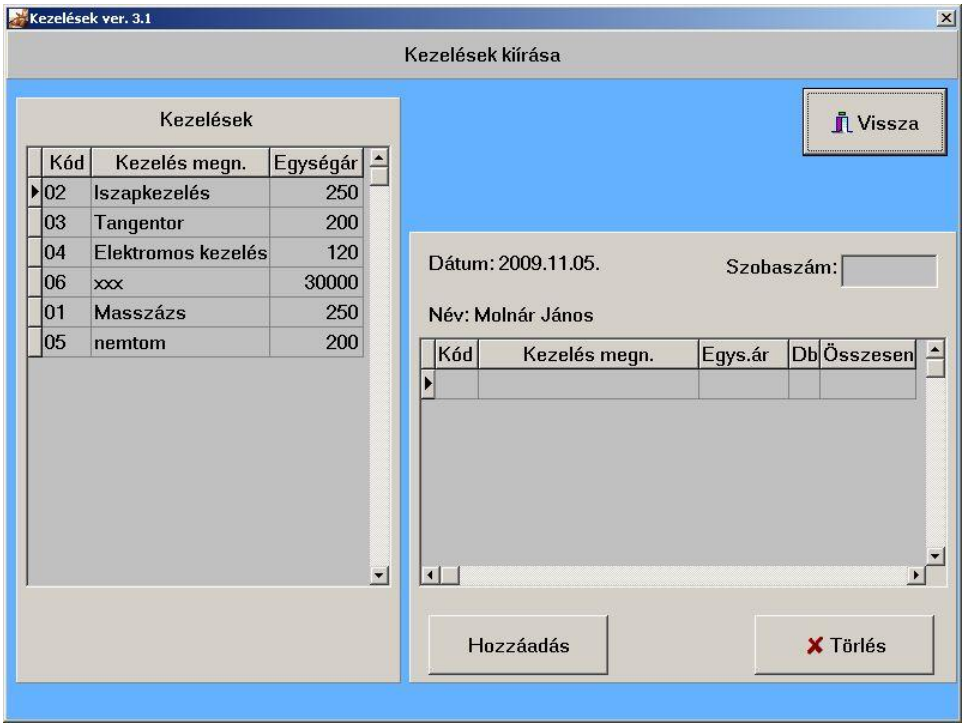
8.4. Almenük

8.4.1. Kezelések kiírása:

A keresési ablakban van lehetőség TAJ és Név alapján keresni. Ha a rendszerben már rögzítettek a megadott vendég adatok akkor a találatok a felső ablakban jelennek meg, míg az alsó ablak a mai dátummal kiírt kezelések megjelenítésére szolgál. Amennyiben nem szerepelnek a vendég adatai az adatbázisban úgy lehetőség van azokat hozzáadni az ’Új vendég’ gomb segítségével. A vendég kiválasztása után lehetőség nyílik a kezelések kiírására.



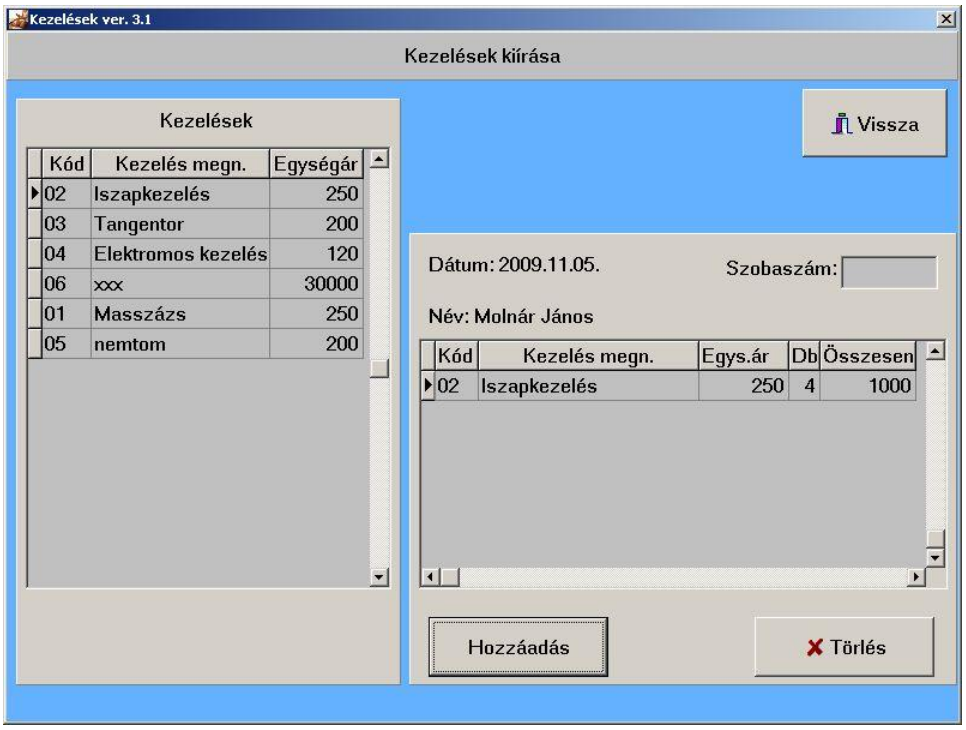
Az ’Új kezelés kiírása’ gombra kattintva egy újabb form jelenik meg mely tartalmazza a kiírható kezelések listáját táblázatban, valamint egy segéd táblázatot mely tartalmazza a felhasználó által kiírt kezeléseket.



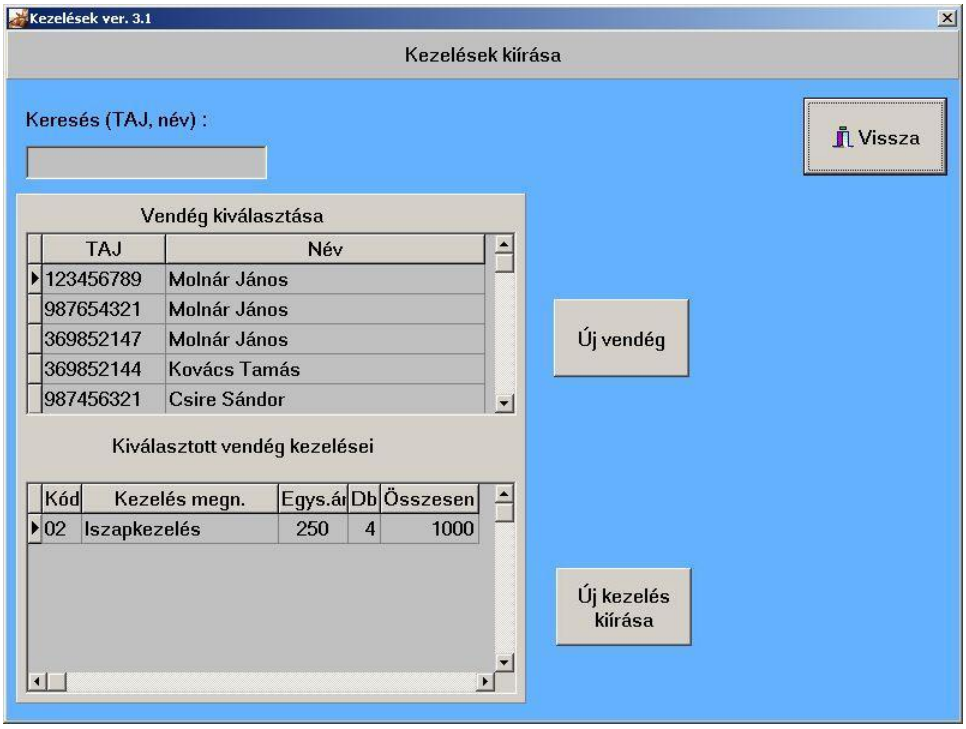
Kezelés hozzáadása a ’Kezelések’ táblázatban egér dupla kattintásával vagy a ’Hozzáadás’ gomb segítségével lehetséges, mindkét esetben az alábbi ablak jelenik meg:

A ’Hozzáad’ gombra kattintva a rendszer rákérdez a letárolásra.

Letárolást választva a segéd táblázatban megjelennek az adatok a kezelésről, ami ha nem megfelelő akkor a billentyűzet Delete gombjával vagy a ’Törlés’ gombra kattintva törölhető.

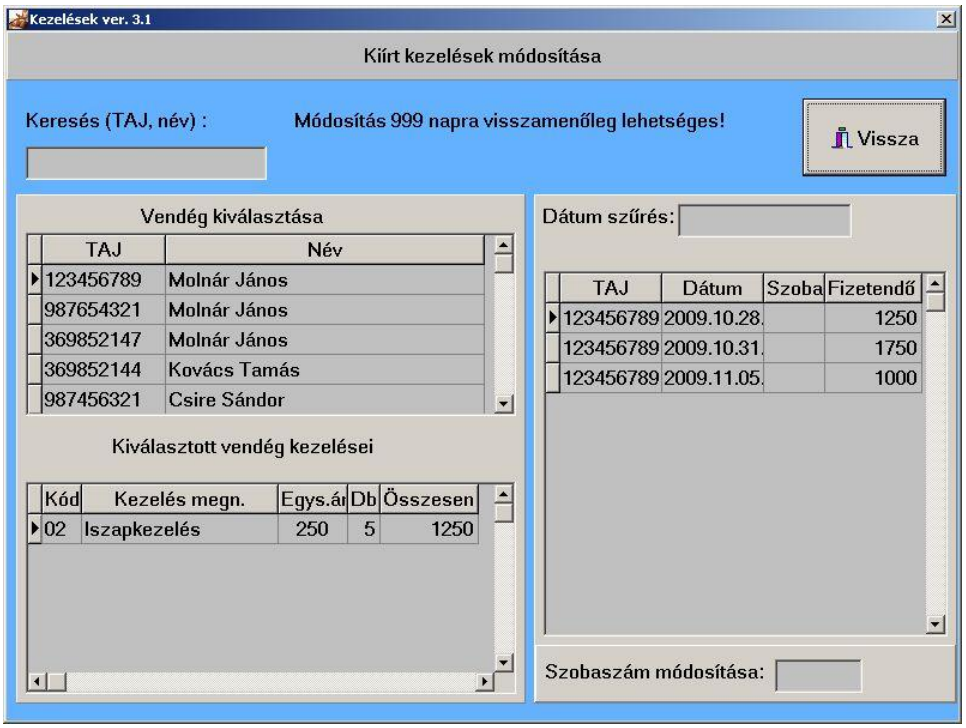


Visszalépés után az alsó táblázatban láthatóak a kiírt kezelések.



8.4.2. Kiírt kezelések módosítása:

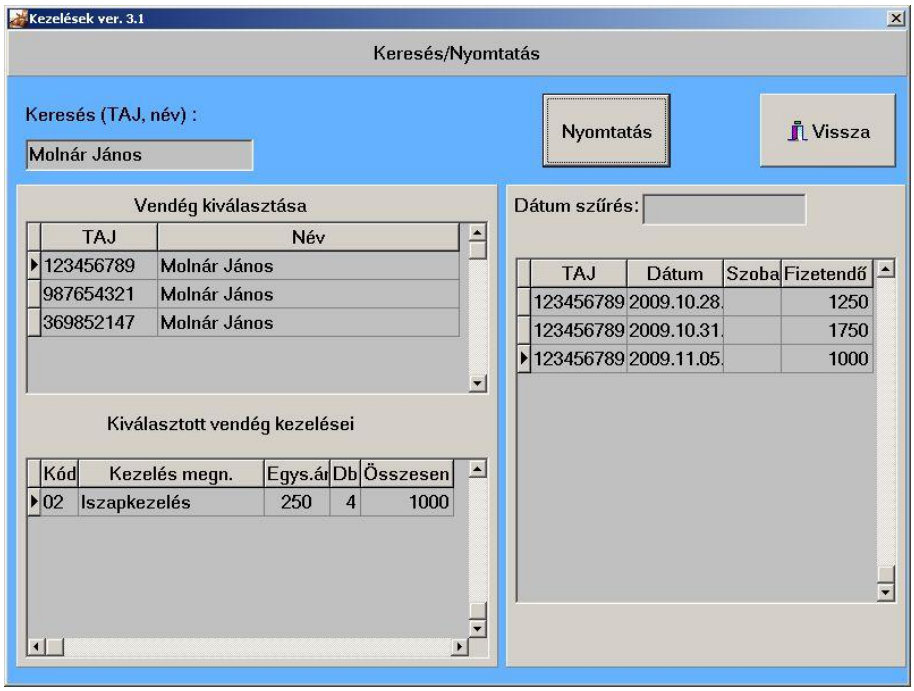
Amennyiben tévesen lett kiírva egy kezelés, lehetőség van annak módosítására az arra jogosult személynek a megadott határidőn belül. Itt lehetséges új kezelést kiírni vagy törölni a meglévőt, ezért ennek a jogosultságnak a korlátozása a napok számát tekintve nagyon fontos. A 'kiválasztott vendég kezelései' ablakban delete gombbal egyesével törölhető a már kiírt kezelés, insert gombbal pedig új adható hozzá. Az összes kezelés törlésére is van lehetőség a jobb oldali ablak megfelelő sorában a delete gomb segítségével. Természetesen törlés előtt megerősítésre van szükség. Lehetőség van még a szobaszám átírására is mely könnyebb beazonosítást tesz lehetővé más munkaterületek részére.

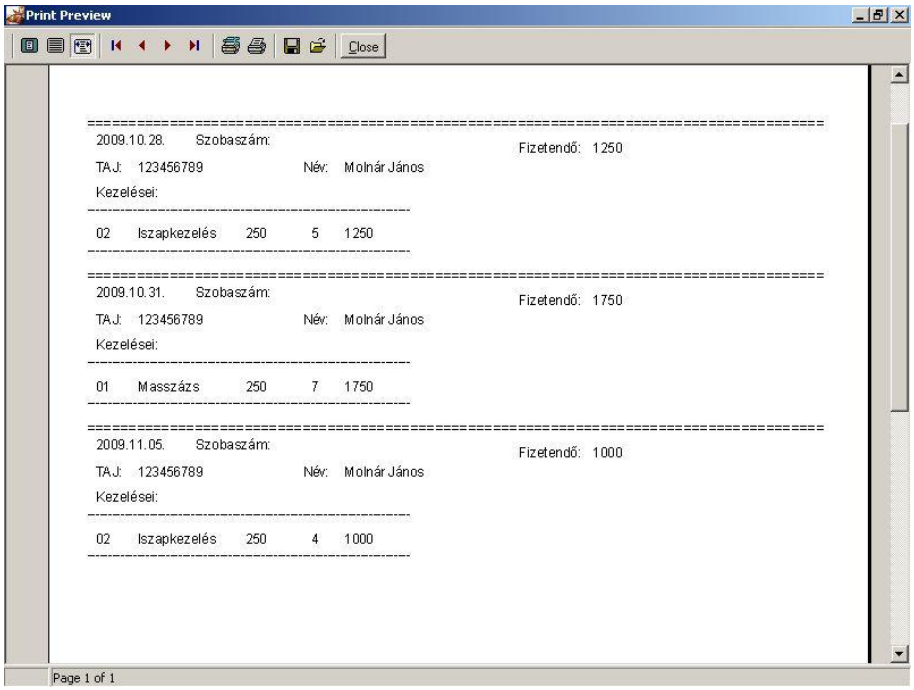


8.4.3. Keresés, nyomtatás:

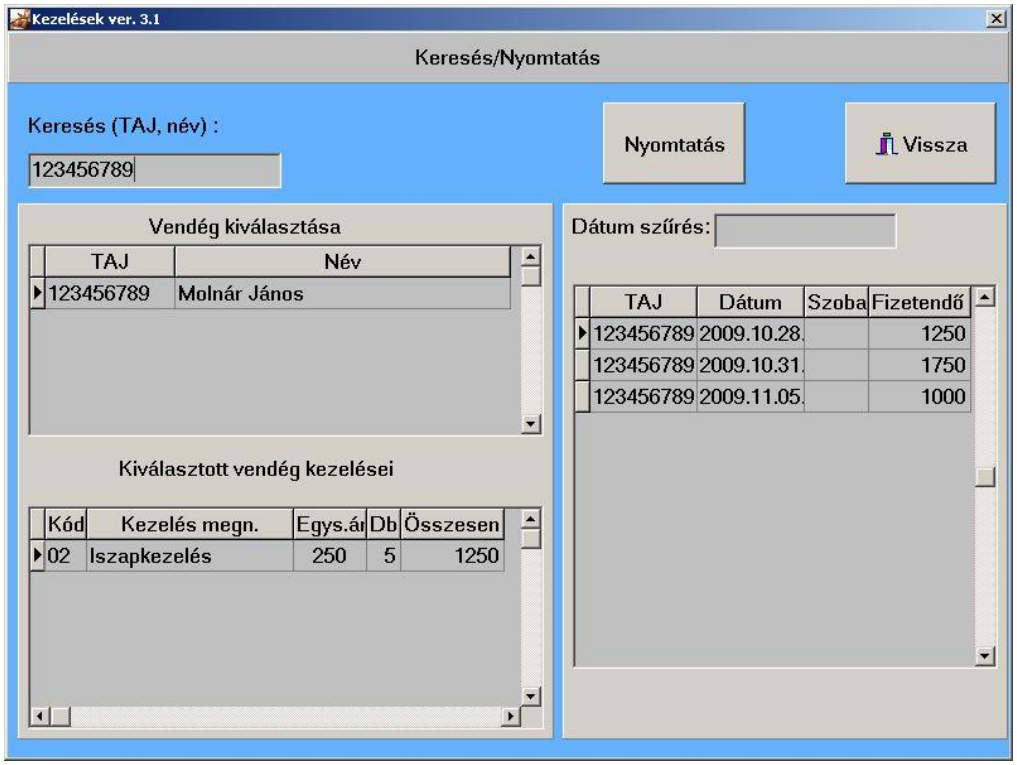
A képernyőn ugyanazok a komponensek jelennek meg mint a kezelések kiírásakor annyi különbséggel hogy a táblák adatait nem lehet módosítani. A keresés és nyomtatás különböző módjaira a megfelelő információcsere érdekében van szükség.

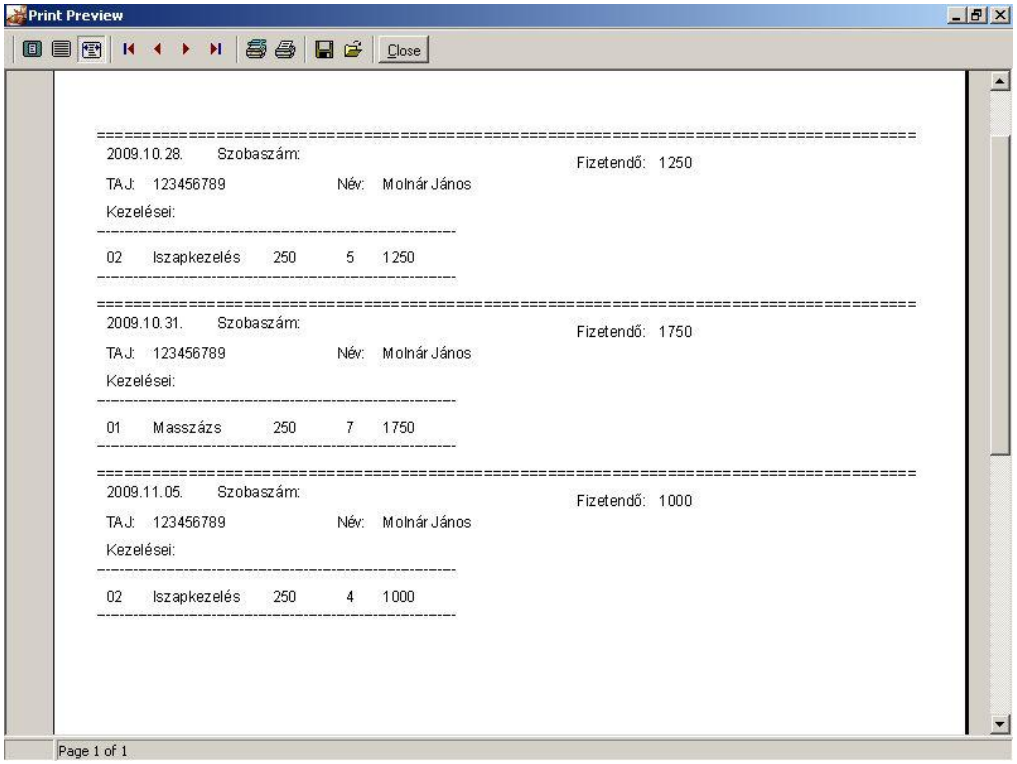
Keresés név alapján, és nyomtatási képe:



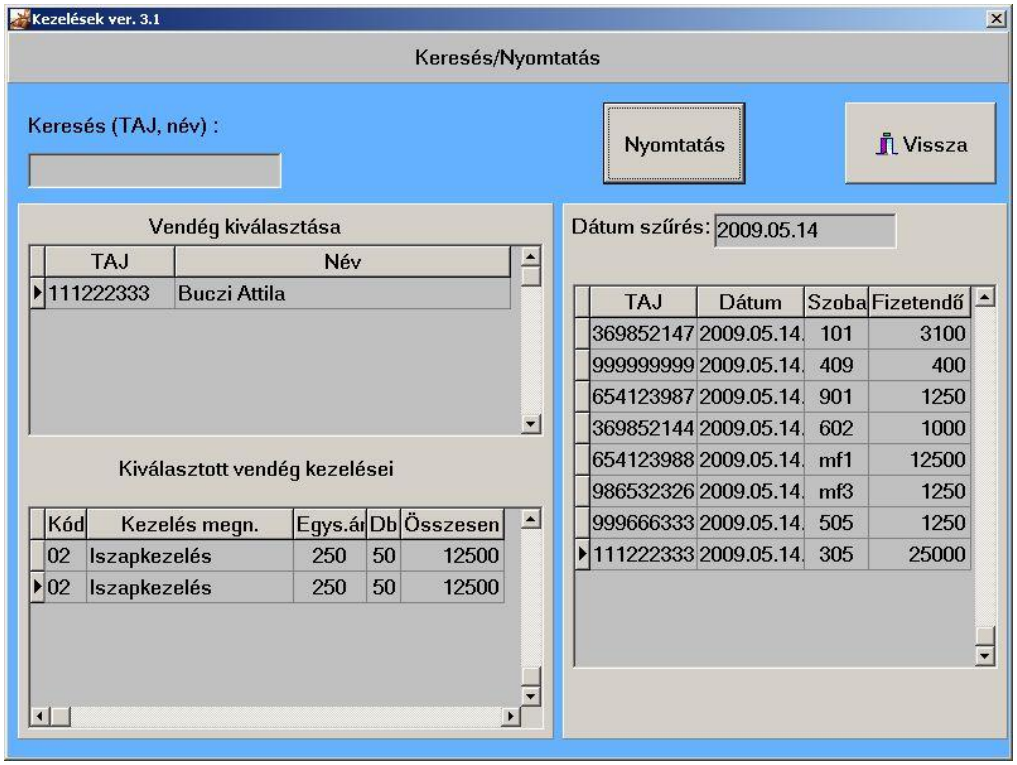


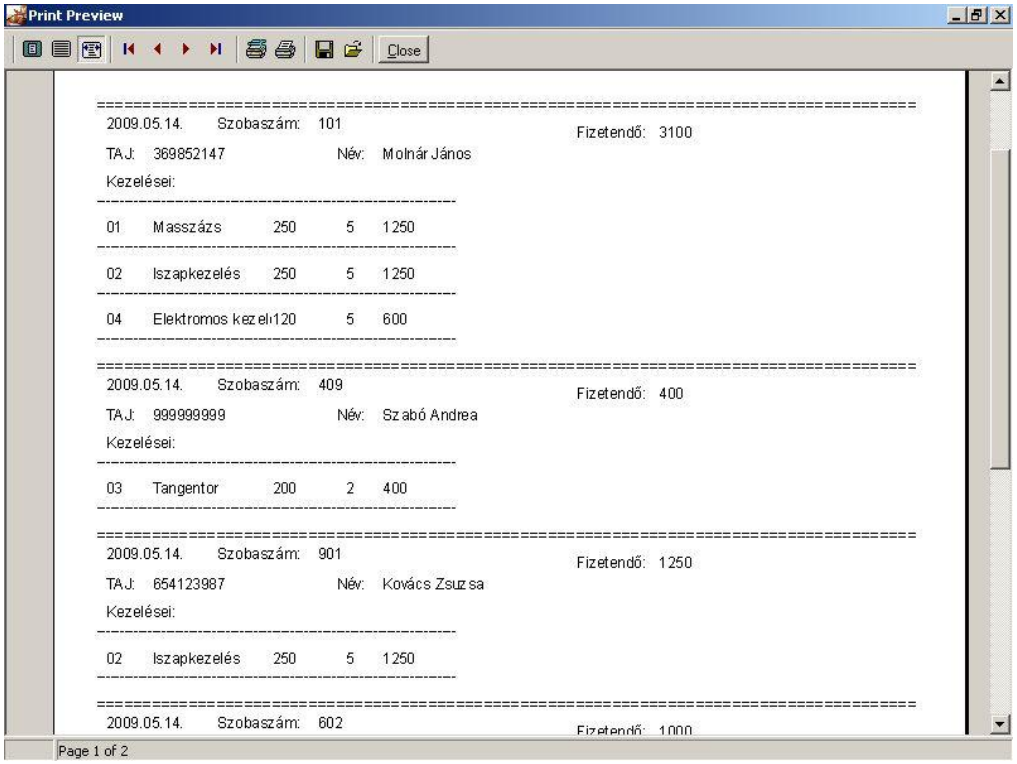
Keresés TAJ alapján, és nyomtatási képe:



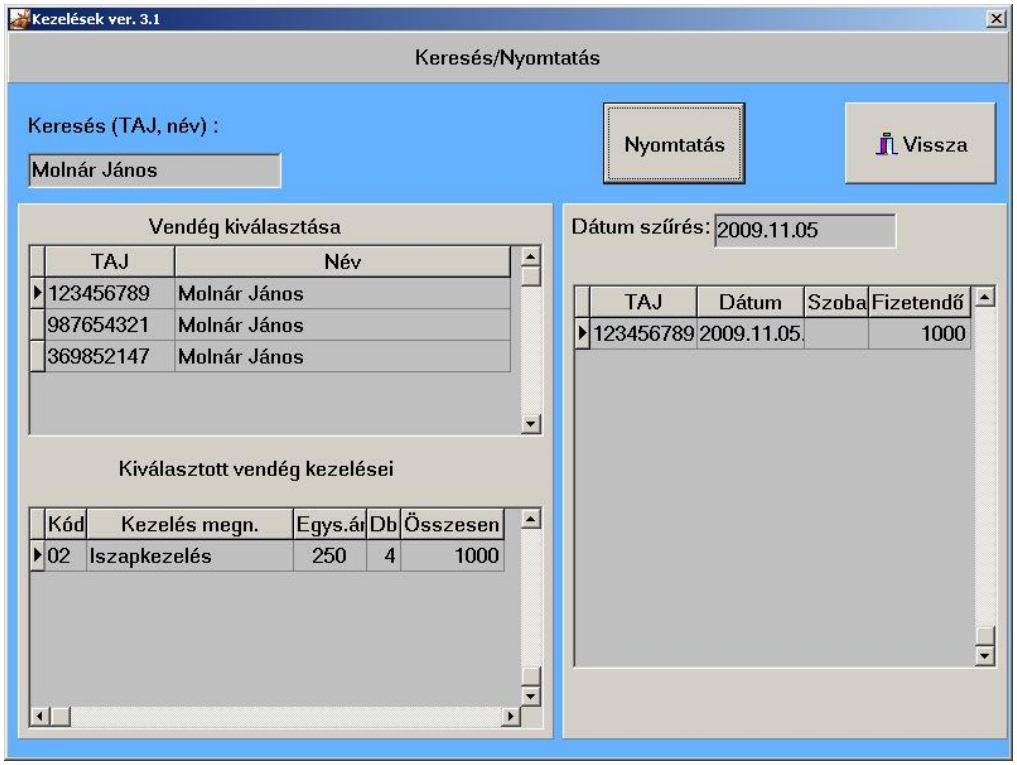


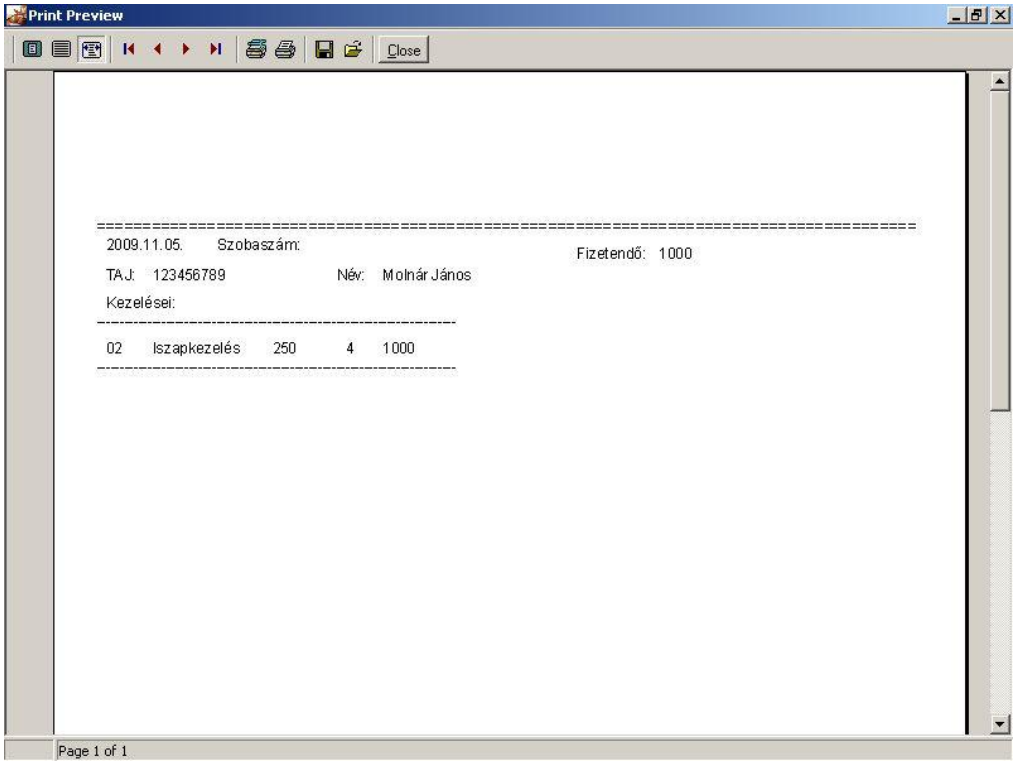
Keresés dátum alapján, és nyomtatási képe:



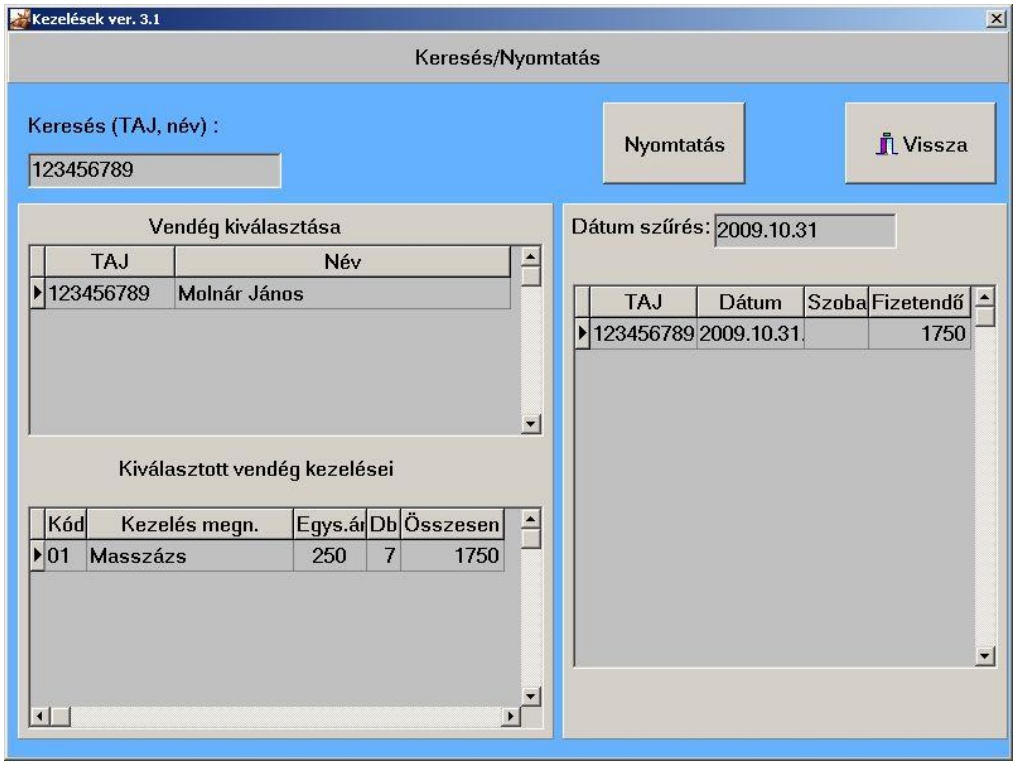


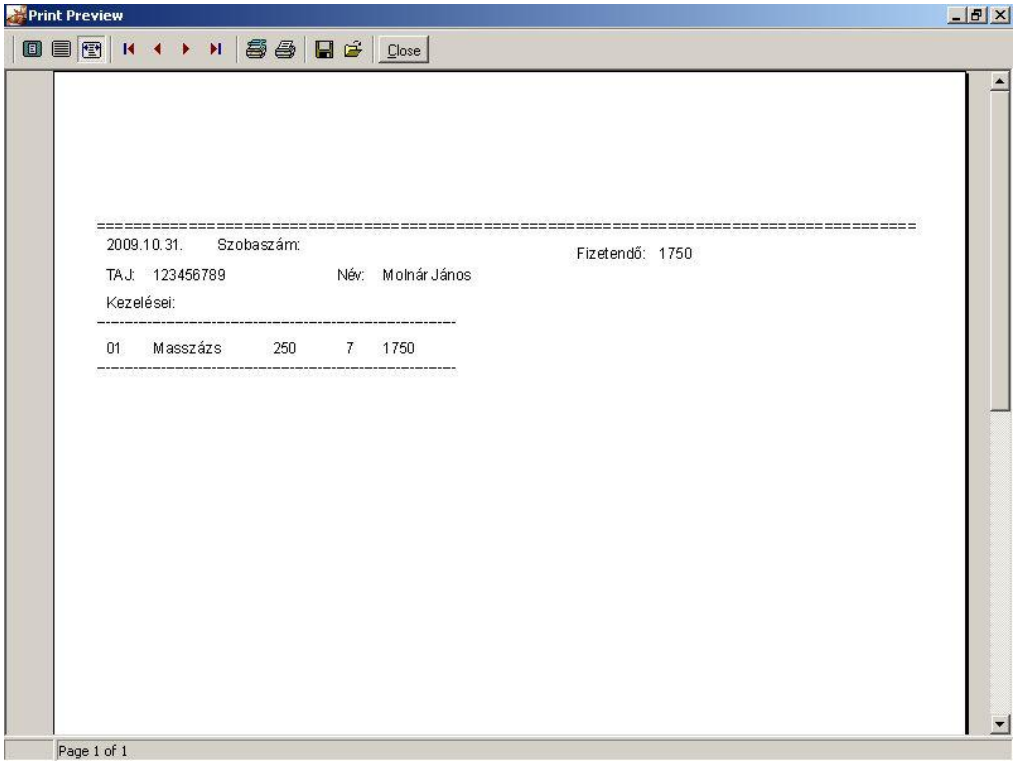
Keresés név alapján dátummal szűrve, és nyomtatási képe:





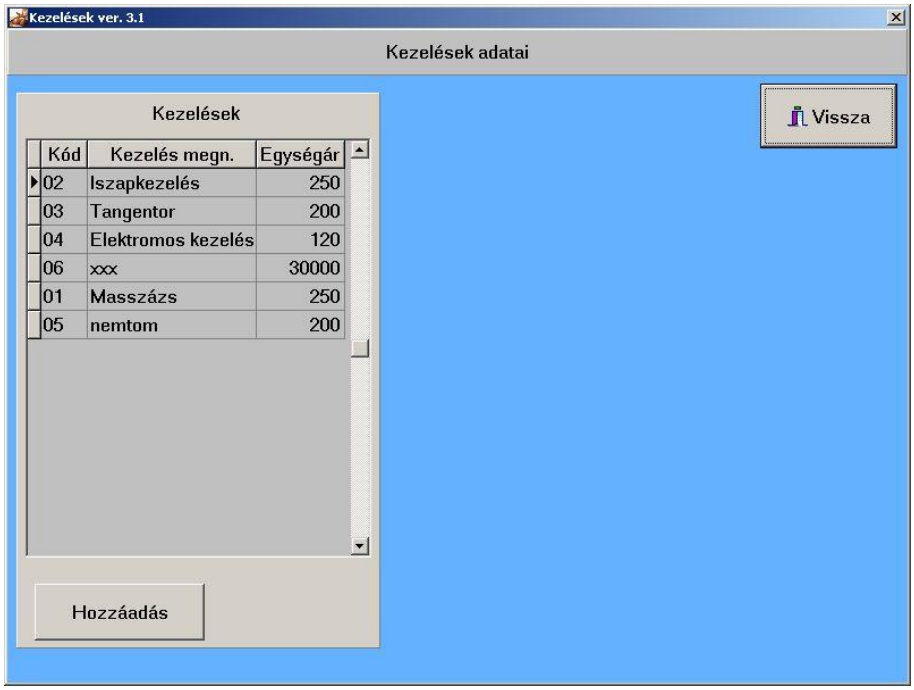
Keresés TAJ alapján dátummal szűrve, és nyomtatási képe:





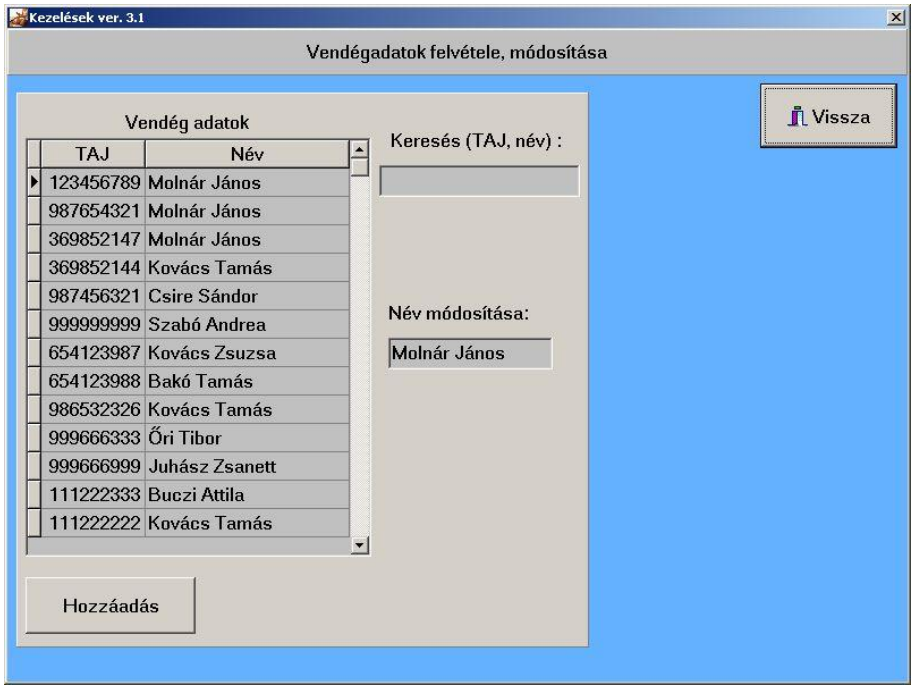
8.4.4. Kezelési tételek:

Az alábbi formon van lehetőség megadni a kezelések kódját, nevét és árát.



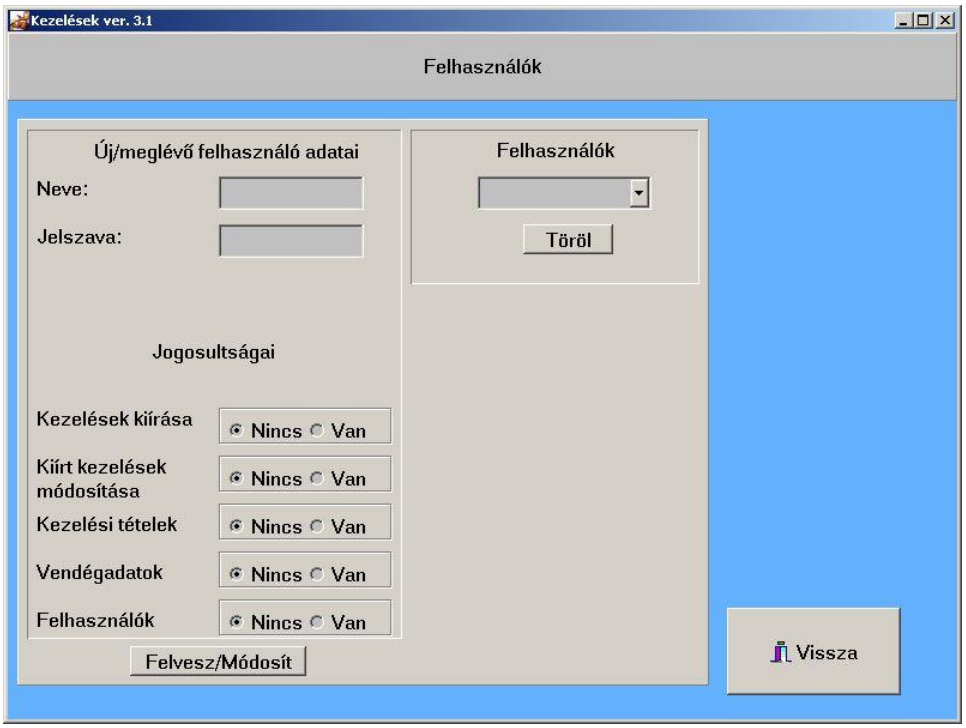
8.4.5. Vendégadatok:

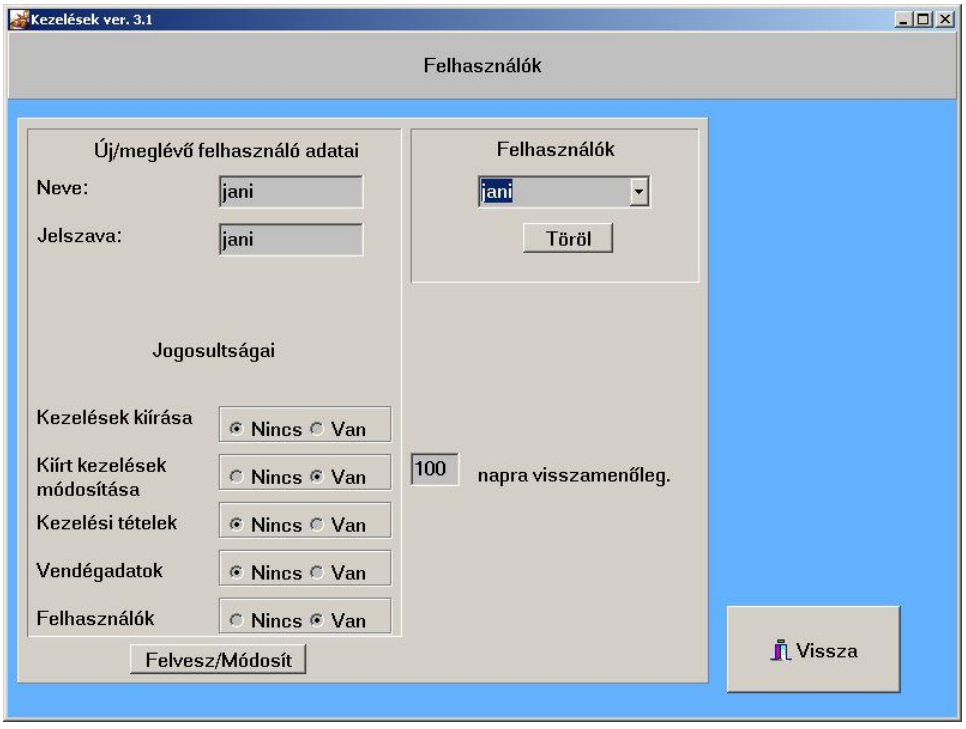
A már bevitt adatok módosítása és új adatok bevitele itt lehetséges.



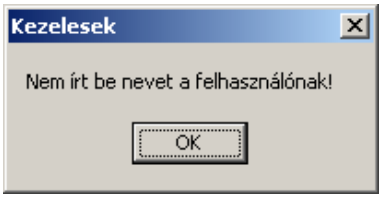
8.4.6. Felhasználók felvétele, törlése:

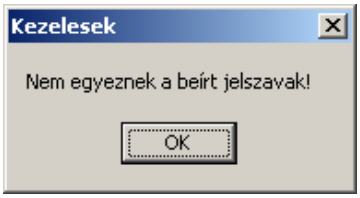
Felhasználók felvételére és eltávolítására az alábbi almenüben nyílik lehetősége az arra jogosultnak.





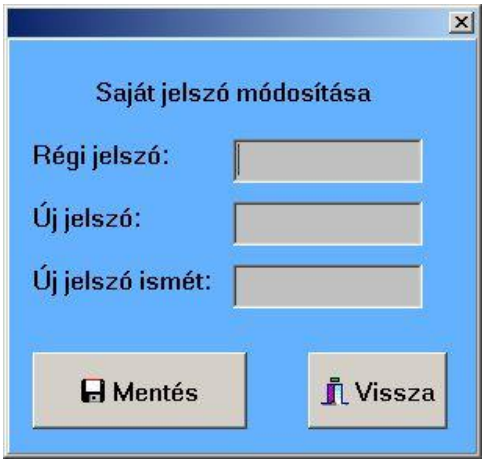
Tévesen írt adatok alapján itt is hibaüzenet érkezik:



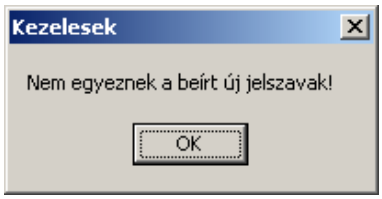


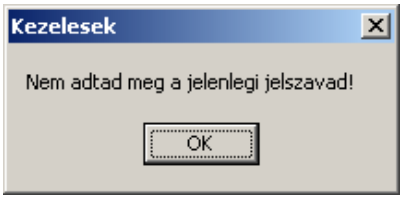
8.4.7. Saját jelszó módosítása

Minden felhasználó módosíthatja saját jelszavát.



Rossz régi jelszó, vagy valamely ki nem töltött új jelszó mező esetén hibaüzenet kap a Felhasználó:





9. ÖSSZEFOGLALÁS

A program kialakításánál az egyszerűséget és a használhatóságot tekintettem elsődleges alapelvnek, mind formai, funkcionális és színvilágát tekinve. Ezen alapelvek mentén készült el az alkalmazás, tekintettel a leendő felhasználókra, akik nem biztos, hogy az információs társadalomban születtek, és ha egy kicsit számitástechnikai analfabéták is, akkor is tudják kezelni a programot.

Az általam készített alkalmazás létrejöttének oka a jelenleg használt rendszer hiányosságaira vezethető vissza. Amíg a fejlesztők nem integrálnak hasonló problémamegoldó eszközt, addig ez a programocska kitöltheti a keletkezett űrt. Bár nem integrálható egyetlenegy jelenleg használt szállodai szoftverbe sem, de ettől még önállóan kiegészítésként bárhol alkalmazható, hiszen jelentős könnyebbség hogy a számításokat már a számítógép végzi el, végső soron ez is lenne a feladata.

Források

Markó Imre: Adatok nyomtatása - 4. rész
http://codexonline.hu/CodeX6/alap/delphi/MarkoImre/Delphi4.htm

Markó Imre: Adatbázis-kezelés Delphi segítségével
http://www.codexonline.hu/CodeX3/Alap/MONTH/DelphiDB.htm

Szűcs Tamás - A Delphi adatbázis-kezelése

A delphi helpje

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani tanáraimnak, akik lehetővé tették azt, hogy az ő tudásuknak legalább egy töredékét én is birtokolhassam, valamint külön köszönetet érdemel témavezető tanárom Dr. Bajalinov Erik, mert az óráin nem csak a szakmai tudásáról volt lehetőségem meggyőződni hanem emberségéről is.