Debreceni Egyetem
Informatikai Kar

Szoftverfejlesztés Delphiben

Témavezető:
Dr. Bölcskei András
Egyetemi Docens

Készítette:
Nagy Zsolt
Programozó Matematikus

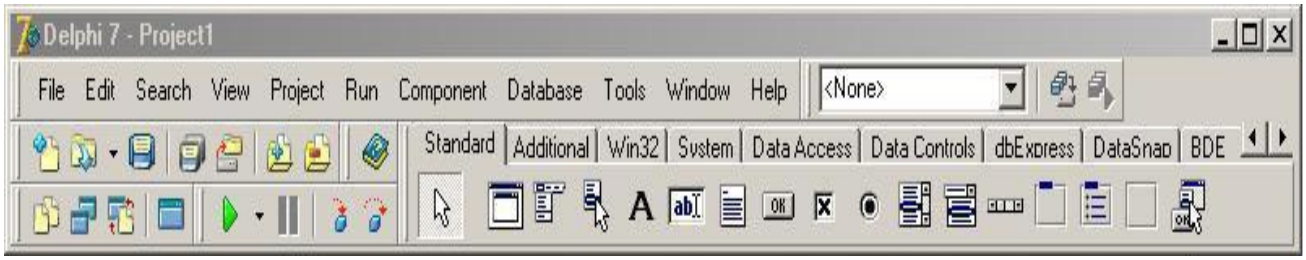
Debrecen
2007

Tartalomjegyzék

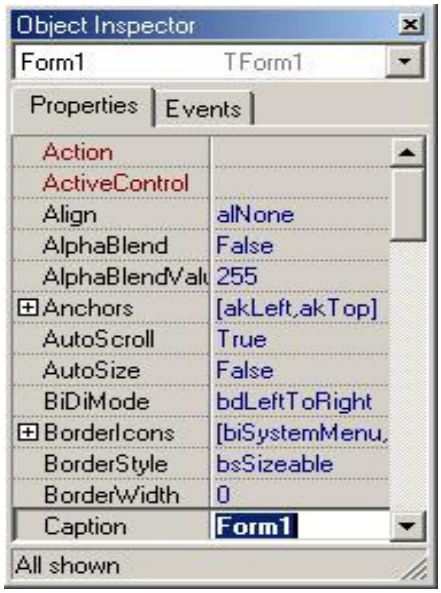
1. Bevezetés

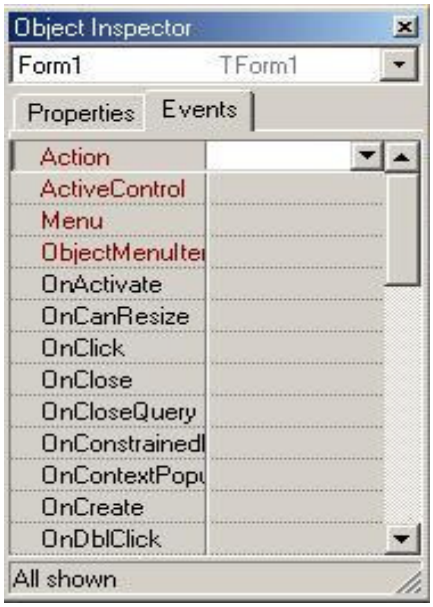
Szakdolgozatom célja, hogy bemutassam a Delphi fejlesztőrendszer lehetőségeit, amely egyesíti magában a Windows alkalmazások készítésére szolgáló grafikus fejlesztői környezetet és a teljesen objektum orientált programnyelvi fordítót. A dolgozat további részeiben betekintést nyerhetünk a Delphi programozási nyelv lehetőségeibe.

A Windows rendszer alapvetően objektumorientált akárcsak a Delphi. Az alkalmazások ablakokat jelentetnek meg melyek objektumként kezelendők, hiszen vannak adataik és vannak viselkedésük. Adatnak tekinthető az ablak címe viselkedésnek pedig a külső és belső eseményekre való reakciók. Amikor a felhasználó a billentyűzet vagy az egér segítségével használja a programot egy esemény jön létre, melyet a Windows üzenetté alakít. Ez tulajdonképpen egy rekord. Az üzenetek több típusa is van például külső és belső. Így tehát a Windows vezérlése egyfajta üzenet alapú kommunikációra épül. Ilyen Windows alkalmazások készíthetők a Delphi rendszerrel rendkívül egyszerűen, mégpedig az előre megírt komponensek segítségével és a hozzájuk tartozó eseményekkel. A Delphi egy Pascal szintaktikájú objektum orientált nyelv. Egy Delphi alkalmazásban megtalálható a *Projektállomány*, mely az alkalmazás főprogramjának felel meg. Ez *.DPR kiterjesztésű mely a **D**elphi **Pr**oject rövidítése. Az *Űrlaptervek,* melyek tulajdonképpen a *.DFM(= **D**elphi **F**or**m**) és a *.PAS kiterjesztésű állományainkat jelentik. Továbbá vannak a *rutinkönyvtáraink* és a *külső erőforrások*. A Delphi egy vizuális fejlesztői eszköz, ami azt jelenti, hogy az ablakok, gombok már a tervezés során láthatóak. A Delphi önállóan futtatható állományt generál, ami nem jellemző a többi más Windows alkalmazásfejlesztőkre. Általában egy kis vagy közepes méretű alkalmazásnál elég csak a célgépre átmásolni a fordított EXE fájlt mivel ezek nem használnak extra állományokat. Az integrált fejlesztői környezet maga több ablakból épül fel. A képernyő felső részén található a főmenü, de a fontosabb menüpontok ikonokról is elérhetőek melyek az eszköztárban találhatóak. Itt helyezkedik el az úgynevezett komponenspaletta, mely különböző lapokat tartalmaz számtalan lehetőségekkel bizonyos rendszer szerint csoportosítva.



A programok egy úgynevezett *form*-on futnak, ami nem más mint egy ablakobjektum. Bármely komponens választható először az adott fülön majd a kiválasztott komponensen való kattintással. Ha kiválasztottunk egy komponenst a *form*-on kattintva megjelenik Általában a baloldalon található az *objektum felügyelő*(*Object Inspector*), melyen a *form* és a rá kerülő objektumok tulajdonságait állíthatjuk be, illetve eseményeit adhatjuk meg.



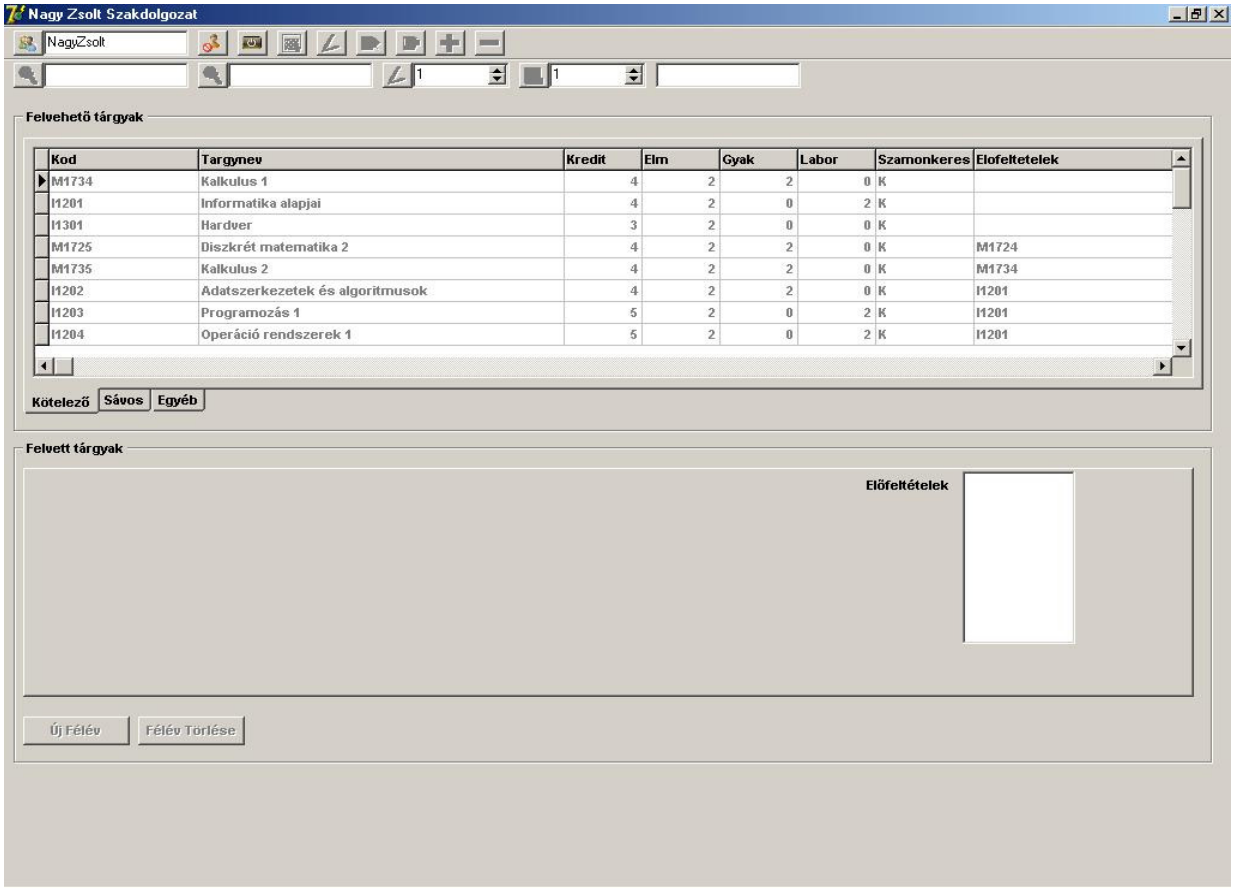


A szakdolgozatomhoz azért a Delphi rendszert választottam, mert számtalan lehetőségekkel rendelkezik a grafikus megvalósítás terén, valamint kiemelkedően jól és egyszerűen, érthetően kezelhetőek vele az adatbázisok.

2. Funkcionális leírás

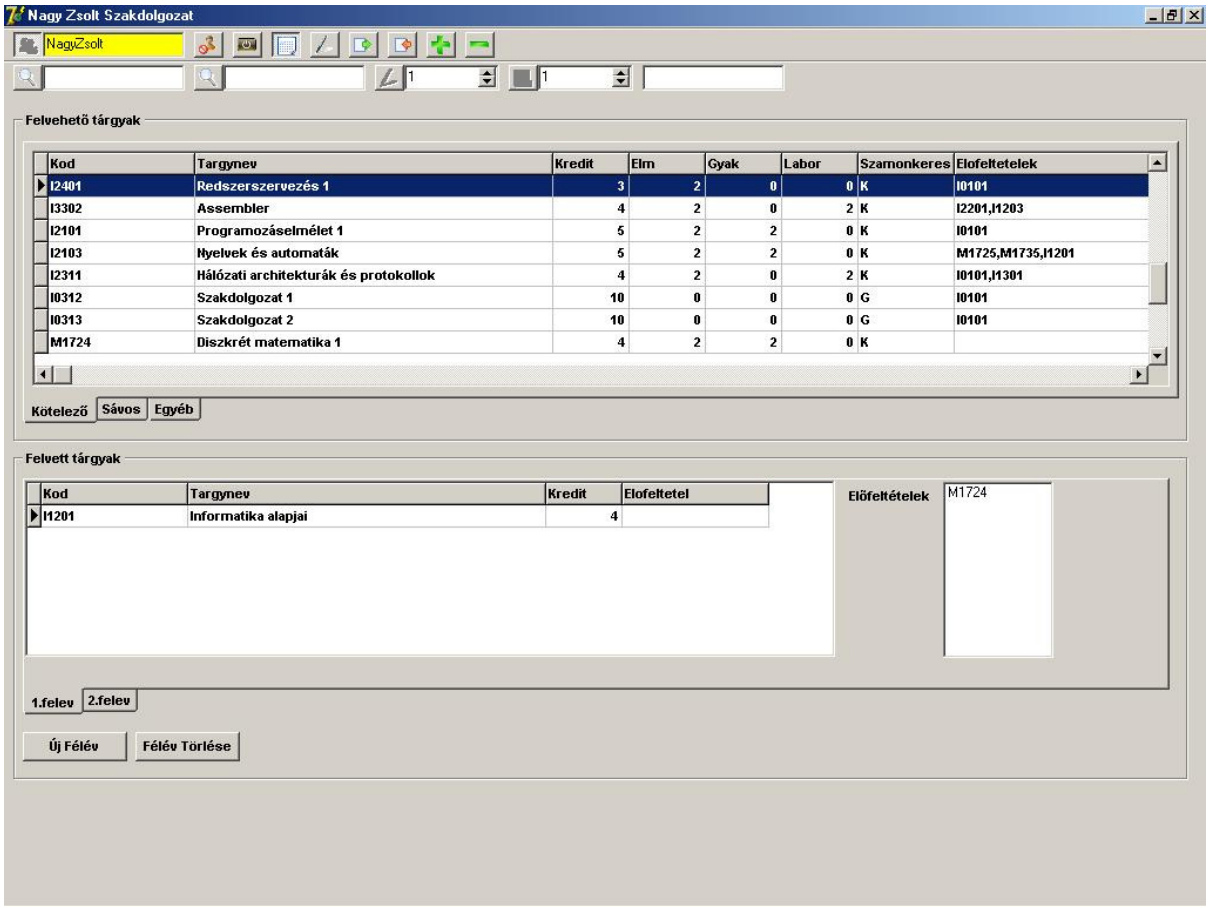
Programom egy tanulmányi előrehaladást tervező és figyelő rendszer, melynek célja hogy számom tartsa, hogy hol is tartunk a diploma megszerzéséhez vezető úton. Programomban igyekeztem különféle komponensek használatára törekedni, ezzel a Delphi által kínált lehetőségek közül minél többet bemutatva. Természetesen a felhasznált komponensek csak a töredékét teszik ki a Delphi szinte végtelennek mondható komponenspalettájának. A felhasznált komponensek között vannak bizonyos komponensek melyek csak futási időben jönnek létre és csakis ekkor léteznek, figyelve a lefoglalt tárhely felszabadítására a biztonságos programozás megvalósításának érdekében. A megírás során törekedtem az egyszerű kezelhetőségre és ebből kifolyólag a program használatának minél gyorsabban történő elsajátíthatóságára. Egy tantárgy felvétele, például megoldható egy egyszerű dupla kattintással, de ikonsorból is vezérelhetjük a történéseket.

A program indításakor szinte minden gomb és minden lehetőség le van tiltva. Egyedül a belépésre illetve a kilépésre van lehetőségünk, valamint a belépési azonosító megválasztására.



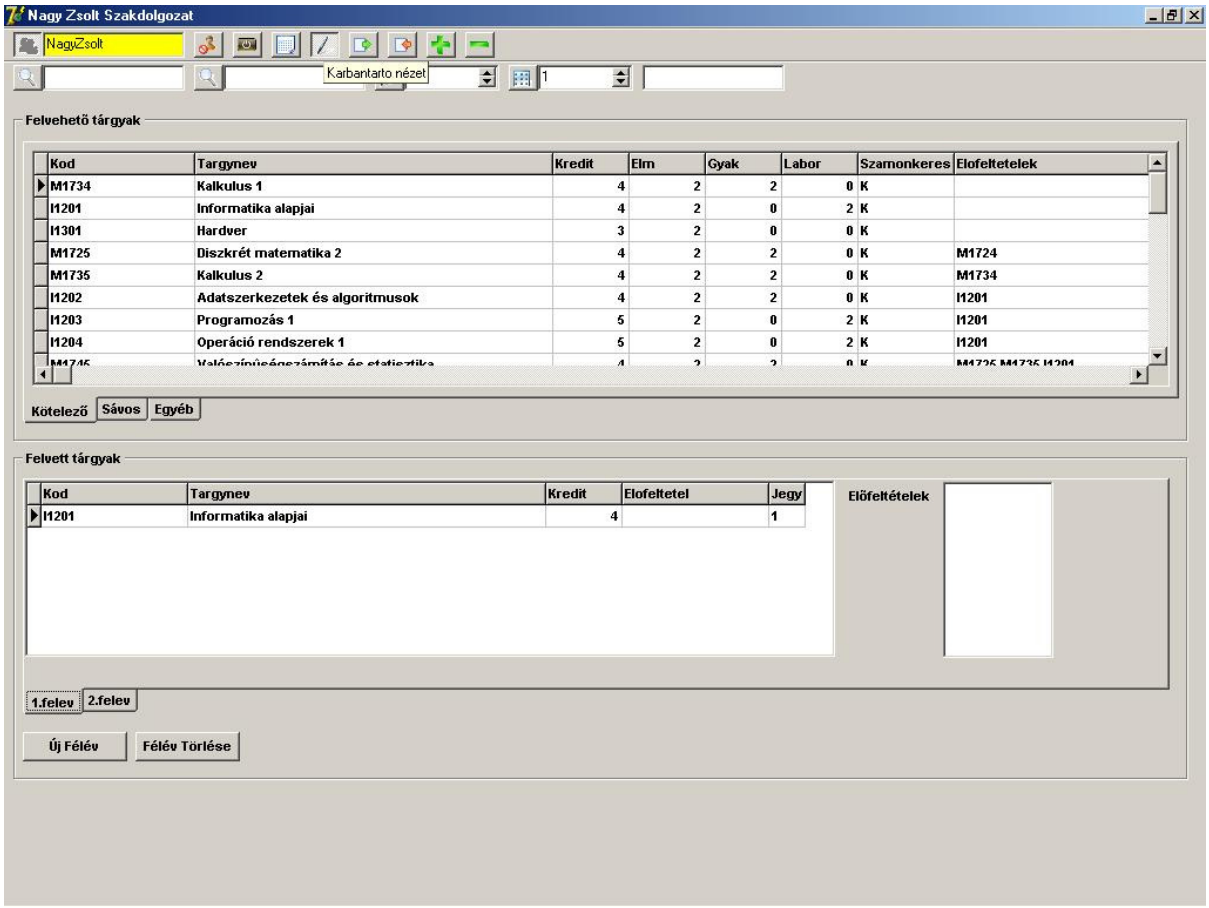
Ha begépeltük a választott azonosítónkat és a belépés gombra kattintunk vagy sikeresen belépünk (ha már korábban létrehoztuk ezt az azonosítót) vagy pedig ha még nem létezik

rákérdez a program, hogy biztosan létre akarjuk-e hozni. Belépés után bizonyos komponensek még mindig letiltott állapotban vannak.



Ez annak köszönhető, hogy a program két módban használható. Az egyikben az egyetemi éveinket lehet megtervezni, vagyis hogy hány félévre tervezzünk, melyik tárgyat melyik félévben szeretnénk felvenni és teljesíteni. Nevezzük ezt tervező módnak. A másik módban pedig a megtervezett félévek karbantartására van lehetőségünk, a megszerzett jegyek beírására, átlagunk számítására egy-egy teljesített félév után. Ezt karbantartó módnak neveztem el. Belépéskor tervező módba kerülünk, mivel első dolgunk úgyis a tantervünk meghatározása lesz. A kezdeti képernyőn egy vagy kettő úgynevezett *DBGrid* található, melyek az adatbázisok megjelenítéséért felelősek. Több lehetőség közül választhatunk attól függően, hogy hol is tartunk a tervezésben. Hozzáadhatunk új félévet a már meglévőkhöz, de akár törölhetjük is a féléveinket, de csakis akkor, ha nincsenek benne tárgyak. Egy tárgyat a már

korábban említett dupla kattintással lehet felvenni, de a dupla kattintás csakis a választható tantárgylistában értendő. Ha a felvett tárgyakat megjelenítendő *DBGrid*-ben kattintunk az megfelel a törlésnek. Természetesen ezt is vezérelhetjük az ikonsorból. A felvehető félévek száma le lett korlátozva 20-ra. Ennyi idő alatt azért bármilyen szakon lehet végezni még akkor is, ha valaki minden évet kétszer jár. Lehetőségünk van a keresésre mind név mind pedig tantárgykód alapján. A keresésnél nem szükséges a teljes név vagy kód megadása, elég egy részszót megadnunk. Ha módot szeretnénk váltani azt könnyen megtehetjük a karbantartó nézet gombra kattintva. Ekkor már minden gomb aktívvá válik.



Ilyenkor az alsó *DBGrid*-ben megjelenik egy új tulajdonság a jegy. Az aktuálisan kiválasztott tantárgyhoz megadhatjuk az általunk megszerzett jegyet, mégpedig úgy, hogy a második ikonsorban a toll képes ikonra kattintunk. Ekkor bejegyzi az ikon mellett szereplő jegyet, melynek felvehető értékei korlátozva vannak egytől ötig. A megadható jegy mellett jobbról egy kis számológép ikon található mely lenyomására megadja számunkra a megadott félévben elért

átlagunkat. Ha nem szerepel a megadott félévben egyetlen tantárgy sem vagy érvénytelen félévet adunk meg akkor nulla értéket fog szolgáltatni. Az alsó *DBGrid mellett* található egy úgynevezett *listbox* melyben információt közlök egy-egy sikertelen tárgyfelvétel után, hogy milyen előfeltételek hiányoznak.

A megírás során igyekeztem az egyértelmű használhatóságra törekedni valamint minden hibás lépésről tájékoztatni a felhasználót. Például, ha nincs meg egy tantárgy előfeltétele azt ne lehessen felvenni vagy ha megvan ne lehessen az előfeltételek megszerzésével egy időben illetve annál korábban teljesíteni. Ugyanígy a tantárgyak törlése csak akkor lehetséges, ha nincs a későbbi félévekben rá épülő tantárgy. A félévek törlésénél mindig csak az utolsó félévet törölhetjük, de csakis akkor, ha nincs benne tantárgy.

3. Telepítési útmutató

A program telepítése meglehetősen egyszerű, hiszen mint már korábban említettem a Delphi automatikusan generál egy futtatható állományt. Elég volna ezt az állományt másolni, de a programban két adatbázist is felhasználunk, így ezeknek a másolása is szükséges. Mivel az adatbázist együtt kell másolnunk a futtatható állománnyal, ezért egy olyan megoldást választottam, aminek segítségével csak egy könyvtárat kell másolnunk. Mégpedig ez a c:\Access könyvtár, mely tartalmazza a fordított *.EXE fájlt, az adatbázisokat és a felhasznált ikonokat is. A program úgy lett készítve hogy ennek a könyvtárnak tetszőleges helyre történő másolása estén minden esetben megtalálja a felhasznált adatbázisokat. Az adatbázisok a Microsoft Office 2003 –as programcsomaggal lettek készítve, valamint a program a Delphi 7.0-ás verziójával készült és Microsoft Windows XP SP2-es rendszerén lett fordítva.

4. Komponensek

Számos komponens felhasználásra került a különböző komponenslapokról, melyeket néhány rövid mondatban ismertetek a fő funkciójuk és főbb tulajdonságaik és eseményeik alapján.
A standard lapról mindenképpen meg kell említenünk a *Panel* és a *GroupBox* komponenseket melyek más komponensek tárolására alkalmasak. Őket úgy is szokás nevezni, hogy tárolók. Az ezen elhelyezett komponenseknek ha később meg akarjuk változtatni az elhelyezkedését, akkor elég ennek a komponensek mozgatása. Ez úgymond összefogja őket. A komponensnek nevet a *name* tulajdonsággal adhatunk és a rajtuk megjeleníthető szöveget a *caption* tulajdonságnál adhatjuk meg. Egy másik fontos komponens az úgynevezett nyomógomb (*Button*). Ennek szintén létezik a name és caption tulajdonság ugyanazon funkciókkal. Ezt a két tulajdonságot a továbbiakban egy komponensnél sem említem csak ha eltérő jelentéssel bírnak az előbbiekben említettektől. A B*utton* komponens legfontosabb eseménye az *OnClick*, mely a gombon való kattintáskor hívódik meg. Az itt megadott utasítások a gombon való kattintás után végrehajtódnak. Itt található még a *Label* komponensünk melyet szövet elhelyezésére vagy futás közbeni megjelenítésére használhatunk. Ekkor a caption tulajdonságot használva megjeleníthetünk bárhol a form-on tetszőleges szöveget, tetszőleges betűtípussal és betűméret beállításokkal. Ezen beállításokat a font tulajdonságnál végezhetjük el. Egy másik szöveg megjelenítésére alkalmas komponens az *Edit*. Ez a komponens nem rendelkezik *caption* tulajdonsággal, viszont van helyette egy *text* tulajdonság, ami szinte azonos jellemzőkkel bír. Továbbá fontos tulajdonság a *ReadOnly*, mely egy *boolean* értéket vehet fel. Ha ezt *true*-ra állítjuk, akkor csak olvasható lesz ez az Edit vagyis nem tudunk bele írni, csak megjelenítésre lesz használható. Megadhatjuk a szöveg maximális hosszát a *MaxLength* tulajdonsággal. Ha értéke nulla, akkor a szöveg hossza határozatlan. Ezen lapon található még az egyszerű listaablak(*ListBox*), mely szintén nem rendelkezik *Caption* tulajdonsággal. A *ListBox*-ban tételek vagy sorok helyezhetőek el. Kezdőérték adható neki *Items* tulajdonság segítségével, mely előhozza a *String list editor*-t. Fontos tulajdonsága ai *ItemIndex* mely jelzi, hogy melyik tétel került kiválasztásra. A *Sorted* határozza meg, hogy a tételek rendezetten jelenjenek-e meg. A *MultiSelect* beállításával egynél több tétel is kiválaszthatóvá válik. Ehhez a *Ctrl+bal* egér gomb vagy a *Shift+bal* egér gomb hívható segítségül. A listaablak tételeihez újakat adhatunk hozzá az Items tulajdonság *Add* metódusával, tételt beszúrhatunk az *Insert* metódusával, törölhetünk a *Delete* metódussal. Ezen kívül az *Items*-nek van még egy fontos metódusa az

IndexOf, mellyel meghatározható, hogy egy tétel szerepel a ListBox-ban vagy sem. A függvény -1-el tér vissza, ha nincs benne ilyen tétel. Egy adott tételt kijelölhetünk a *Selected* tulajdonság true értékre állításával. Az Items-nek van egy úgynevezett *Count* tulajdonsága, mely megadja a *ListBox* tételeinek a számát. Ezen tulajdonságok és metódusok alapján tekinthető egyfajta listának a *ListBox*, mivel van eleje és vége, minden elemnek van megelőzője és rákövetkezője, kivéve az elsőt és az utolsót. A delphi a rendszerüzeneteket, rendszereseményeket úgynevezett Windows message-ként (röviden WM) fogadja. Ilyen esemény az egérgörgetés, billentyűleütés vagy az alapvető előre definiált perifériákról érkező inputok. Ezen események kezelésére a delphi előre beépített eseménykezelőket használ. Esetünkben a görgetés a *WM_MouseWheel* eseményt váltja ki. Ennek a folyamatnak a megváltoztatására használjuk az *ApplicationEvents* komponenst. Görgetéskor a komponens OnMessage eljárása hívódik meg, melyben megváltoztathatjuk az előre definiált eljárást.

A Win32 lapról felhasználtam az *ImageList* komponenst, mely a felhasznált ikonokat importálja és tárolja. Ez a komponens futási időben nem látható, hiszen tárolási szerepet játszik, a megjelenítéshez közvetlenül nincs köze. Szintén innen használtam fel az úgynevezett *ToolBar*-t. Ennek a segítségével valósítható meg az eszköztár. Tetszőleges számú gomb és más komponens helyezhető el rajta. Szabványosítható a rajta elhelyezett gombok mérete és helyzete, így nem kell mindegyiknél külön-külön beállításokat végeznünk. A *ToolBar*-on jobb egér klikkel előjön egy helyi menü, melyben tetszőlegesen választhatunk, hogy *Button*-t vagy *Separator*-t adjunk hozzá. A *Separator*-nak elválasztó szerepe van. Mivel egy adatbázison belül több táblánk van ezért több *DBGrid*-re van szükségünk melyek közötti váltást a *TabControl* komponenssel oldjuk meg. Ennek fontos metódusa az *OnChange* mely a *TabControl* fülei között történő váltás során hívódik meg.

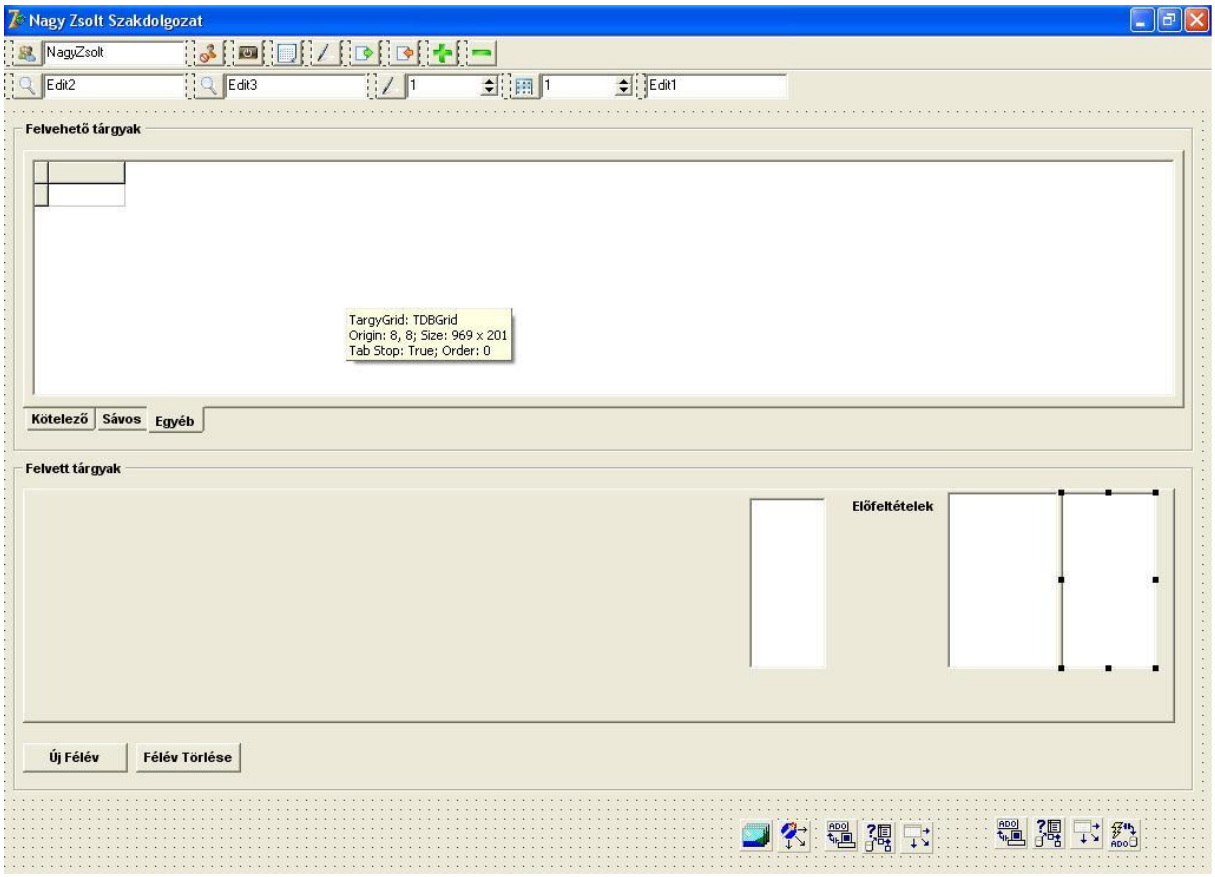
A *Data Access* lapról a *DataSource* komponens került felhasználásra, melynek feladata az adatbázis adattáblái és az adatbázis-kezelő alkalmazás közötti kapcsolat teremtése. Egyfajta közvetítő szerepet lát el az adatelérő és adatmegjelenítő komponensek között. Az összekapcsolást a *DataSet* tulajdonságával végezzük. Ez a komponens nem vizuális komponens nincs szerepe a program ablakának kialakításában.

Az *ADO* lapról felhasználtam az *ADODataSet*-t, ami kicsivel többet tud, mint egy szimpla *DataSet*. Ugyanis van egy úgynevezett *CommandText* tulajdonsága, ami lehetőséget ad SQL-es parancsok felhasználására. Ezzel az adatbázis adatait könnyedén lehet korlátozni, amivel bizonyos részfeladatok egyszerűbben megoldhatók, és időben gyorsabbá válik a lefutása. Az

előbbi *DataSource DataSet* tulajdonságának ennek a komponensnek a nevét kell értékül adnunk. Innen került még felhasználásra az *ADOConnection*, mely a biztonságos kapcsolatért felelős. Az előbbi *ADODataSet* komponens *Connection* tulajdonságának ennek a komponensnek a nevét kell értékül adnunk. Ezek szintén nem vizuális komponensek.
Az adatok tényleges megjelenítéséért a *Data Controls* lapról a már korábban említett *DBGrid* a felelős. Ennek a komponensnek a *DataSource* tulajdonságának kell értékül adnunk a kapcsolatért felelős *DataSource* komponens nevét. A megjelenítés teljesen automatikus módon zajlik, a komponens felismeri a mezők és rekordok számát egy adott táblán belül, melyeket ezek után egy rácsszerkezetben jelenít meg.

5. Függvények

A program lefuttatásakor az első függvényünk, amely mindenképpen le fog futni a *Form* ablakobjektum *FormCreate* metódusa. A neve utal a szerepére. Lényegében megalkotja a *Form*-ot és mivel elsőként ez fut le ide kerülhetnek az inicializáló utasítások, az adatbázisok előkészítése. A *Form*-on elhelyezkedő objektumok és a futtatás után megalkotott objektumok így festenek a futtatás előtt:



A metódusban található 4 db lokális változó. Ebből kettő egész típusú melyeket ciklusváltozóként használtam fel. Egy *String* típusú és egy *TIniFile* típusú mely a különböző tanulók tárolása miatt szükséges. A függvény egy kivételkezelővel indul, melyben megalkotjuk az Ini file-t. Ezután a USER szekciójából kiolvassuk a már meglévő felhasználókat, melyeket egy *ListBox*-ban fogunk tárolni. A *ListBox* neve a szerepére utalva *Users*. A fájlban a megadott felhasználónév=table'i' módon tárolódnak soronként annak *USER* szekciójában. Az adatbázisban pedig table'i' néven jönnek létre a táblák Az i 0-tól végtelen sok értéket vehet fel.

A kiolvasás első lépésében mindent kiolvas, ami a fájlban van, de egy ciklus segítségével gyorsan megoldható, hogy csak a felhasználónevek maradjanak. Ezután ha már megvannak a felhasználók, felszabadítjuk az *Ini* nevű változónkat a kivételkezelő *finally* ágával mely minden esetben lefut. Majd beállítjuk a *ConnectionString*-eket és megadjuk, hogy hol is találhatóak az adatbázisok. Ugyanis az adatbázisok egy külön könyvtárban helyezkednek el. Aztán beállítjuk a tárgyak adatbázisunkból, hogy miket láthassunk a már említett SQL utasítások segítségével. Majd pedig megnyitjuk az adatbázist egy előre megírt private metódus segítségével. Ezek után beállítjuk a megfelelő kezdőértékre a globális változóinkat. A db változónknak a félévek megjelenítésében lesz szerepe, az *akt_db* változónknak a félévek törlésében és felszabadításában, a nézet változó a kiválasztott módnak megfelelő megjelenítésért felelős, a *UserNr* a felhasználók számát tárolja azért, ha új felhasználót hoznánk létre tudjuk hányadik számú táblát rendeljük a nevéhez. Aztán *TargyTab* komponens *TabIndex*-ének is értéket adunk majd hívjuk ezen komponens *OnChange* metódusát, melyet később részletezek. Ezután pedig beállítom a komponensek *Enabled* tulajdonságát, valamint a szövegmegjelenítők *ReadOnly* tulajdonságát, melyek megadják, hogy a megalkotott *Form*-on mely komponenseket tudom használni. Illetve még bizonyos komponensek láthatóságát is korlátozom a *Visible* tulajdonság *false* értékre állításával. Ezek később természetesen változni fognak a lépéseinktől függően.
Az adatbázisok megnyitásáért a *TargyOpen* és *UserOpen* private metódusok felelősek. Mindkettőben szerepel egy lokális boolean típusú változó, mely értéke megadja hogy a megnyitás sikeres volt vagy sem. Alaphelyzetben ezt az értéket true-ra állítjuk, majd egy kivételkezelőben nyitjuk meg az adatbázisokat a következő módon.

```
try
TargyConnection.Connected:=True;
TargyDataSet.Active:=True;
except
Open:=False;
MessageDlg('Nem sikerült csatlakozni!', mtWarning, [mbOK],0);
end;
```

Ha a *try* utáni utasítások hiba nélkül lefutnak az except ág kimarad. Ha hibát okoznak lefut az az ág is. Ezután a *Result* változónak értékül adjuk az *Open* változó által felvett értéket és ez lesz a függvény visszatérési értéke. Közben még a *TargyDataSet* komponensünk

IndexFieldNames tulajdonságát üres értékre állítjuk, amire azért van szükség, ha táblát váltanánk az adatbázisunkban akkor ne okozzon ismeretlen oszlopnév hibát.
A *MessageDlg* függvényünk egy ablakot jelenít meg és a string-ként megadott szöveget kiírja, melynek célja a felhasználó tájékoztatása. Utána pedig különböző paraméterek szerepelnek melyek közül érdemes megemlíteni [mbOK]-ot. Ez egy *BitButton* objektumot helyez el a *form*-on, melynek megvan az a tulajdonság, hogy a "leokézás" után nyugtázza a végrehajtott módosításokat. Magyarul előre meg van írva a gomb kattintásához tartozó metódus.
Az adatbázisok lezárása hasonlóképpen történik, mint a megnyitása csak éppen a lokális változó neve más és a műveleteket fordított sorrendben kell végrehajtanunk, mégpedig a következő módon.

```
try
TargyDataSet.Active:=False;
TargyConnection.Connected:=False;
except
Close:=False;
MessageDlg('Nem sikerült bezárni!', mtWarning, [mbOK],0);
end;
```

Látható, hogy a megnyitáshoz képest itt először az *Active* tulajdonságot állítjuk false értékre és csak utána állítjuk a *Connected* tulajdonságot false értékre, tehát pontosan fordított sorrendben.
A *FormCreate* metódus ellentétje a *FormDestroy*. Ez fut le utoljára, amiben mindössze csak az adatbázisok lezárását végezzük.
A *TabControl* komponens fülei közötti váltások a tárgy adatbázisunknál különböző táblák megjelenítését jelenti. Ennek a megvalósítására TargyTab komponens *OnChange* metódusát használtuk, melyet *TargyTabChange*-nek neveztem el a táblára való utalása miatt. A függvény nem tartalmaz lokális változót. Tartalmaz három külső feltételt, melyeket egyenként ellenőriz, de mindig csak egy fog végrehajtódni attól függően, hogy a komponens *TabIndexe* milyen értékkel rendelkezik. Ezután feltételben vizsgáljuk, hogy sikerült-e az adatbázisunkat bezárni és ha igen az adatbázis *CommandText*-jének csak azt a táblát állítjuk be, amelyikre éppen szükségünk van. Majd újra megnyitjuk az adatbázist feltételben vizsgálva a sikeres megnyitást.

Ha a megnyitás nem sikerülne, akkor a *MessageDlg* függvényt felhasználva ezt tudatjuk a felhasználóval.
Az egyik eset a következő:

```
if TargyTab.TabIndex=0 then
if TargyClose then
begin
TargyDataSet.CommandText:='Select * From Targy';
if not TargyOpen then  MessageDlg('Nem sikerült megnyitni!', mtWarning, [mbOK],0);
end;
```

Ekkor a *Targy* táblánk minden elemét láthatjuk.
Ugyanígy a *User* adatbázisunkhoz is írtam egy hasonló függvényt, de ez másképp működik, mint az előbbi, mert több jelentős különbség is van. Az előbbinél a füleken történő váltás a táblák megjelenítése közötti váltást jelentette. Itt viszont egy táblánk van minden felhasználónál, amiben tárolva vannak a tantárgyaink, így az előbbi megoldás itt nem működik. Valamint ez nem statikus adatbázis, mint az előző esetben, hanem itt folyamatosan változik a félévek száma attól függően, hogy éppen hozzá adunk vagy éppen törlünk egyet. Valamint a két nézet miatt a *CommandText*-et is különböző módon kell megadnunk. Lokális változóra itt sincs szükségünk. Először bezárjuk az adatbázist, majd a nézet változónak megfelelően beállítjuk a *CommandText*-et. Ez után feltételben vizsgáljuk, hogy sikerült e az adatbázis újbóli megnyitása. És itt jön be a jelentős eltérés:

```
felev[db].DataSource:=UserDataSource;
```

A *felev* tömbünk egy *DbGrid*-eket tartalmazó tömb, mely 0-tól a maximális félévszám-1-ig vehet fel értékeket. Az éppen db-nak megfelelő számú *DBGrid*-nek itt állítjuk be a *DataSource*-át mellyel megteremtjük a kapcsolatot a táblák és az alkalmazás között. Erre azért van itt szükség, mert a *TabControl* elemei futási időben jönnek létre, ahogyan a *DbGrid*-ek is, így ilyenkor kell gondoskodnunk a kapcsolatteremtésről. Ezt a metódust többször is fogjuk hívni a futás közben, míg az előzőt elég volt egyszer meghívnunk a *FormCreate* metódusban.

A tárgyak közötti válogatást megkönnyítheti számunkra az oszlopok szerinti rendezés. Ehhez nem kell mást tennünk, mint hogy rákattintunk a *DBGrid* valamely oszlopának a címkéjére. Minden kattintásnál ellenkezőjére változik a rendezés iránya. Ez a *TargyGrid OnTitleClick* metódusában lett megírva a következő módon:

```
Bookmark:= TargyDataSet.Bookmark;
if TargyDataSet.IndexFieldNames = '[' + Column.FieldName + ']' then
TargyDataSet.IndexFieldNames:= '[' + Column.FieldName + '] DESC'
else
TargyDataSet.IndexFieldNames:= '[' + Column.FieldName + ']';
Clmn:= Column;
TargyDataSet.Bookmark:= Bookmark;
```

A megadott felhasználónévvel való belépés a *Toolbar*-on elhelyezett *Belepes* gombra való kattintással történhet meg. Ezt a metódust *SBBelepClick*-nek neveztem. Három lokális változója van a metódusnak. Egy egész típusú, melyet ciklusváltozónak használtam fel. Egy *boolean* típusú melyet arra a célra tartottam fenn, hogy figyeljem új felhasználót hoztam-e létre. Valamint egy *TIniFile* típusút mely az *Ini* fájl kezeléséhez szükséges. A metódus elején a nézet változót egyre állítom, amire a több felhasználós mód miatt van szükség. Ugyanis az elején egyre van inicializálva, de ha kilépek egy adott felhasználótól, amelyik éppen 2-es nézetben használta és belépek egy új felhasználónévvel akkor 2-es nézetben lenne a program. Ezt kiküszöbölve egyes nézetben indul a program minden felhasználónál külön külön. A db változót nullaértékűre állítom. A belépés után több gomb is aktívvá válik melyeket az Enabled tulajdonság *true* értékre történő állításával tettem meg. A User adatbázis megjelenítéséért felelős *TabControl*-t letisztázom a *Clear* metódusának a hívásával. Aztán meghívtam a *Tárgy* adatbázis megjelenítésért felelős *TabControl OnChange* metódusát mellyel aktualizálódnak a változások. Ezután a felszabadítom a *DBGrid*-ek tárolását végző tömböt, majd a *new* logikai típusú változónak *false* értéket adok. Ez után megvizsgálom, hogy a beírt felhasználónév létezik-e vagy sem és az alapján közlök üzenetet a felhasználóval.

A vizsgálatot végző feltétel:

```
If  MessageDlg('Ilyen felhasználó nem létezik
Létrehozzam?',mtInformation,[mbOK,mbCancel],0)=mrOK then
```

A feltételben a *MessageDlg* függvényt használtam fel, de itt nem csak egy gombot jelenítek meg kettőt. Az *mbOK* mellett szerepel az *mbCancel* gomb is megjelenik. Ugyanúgy előre megírt metódusa van tehát ennek a megírásával már nem kell foglakoznunk. Ha nem létezik megkérdezem létrehozzam-e. Ha nem visszalépek a program indítása utáni állapotba.
Az indítási állapotba kerüléshez következő komponensek letiltása és különböző beállítása szükséges:

```
LoginEdit.Color:=clWhite;
SBBelep.Down:=False;   SBLogot.Enabled:=False;
SBTervezo.Down:=False; SBBelep.Enabled:=True;
SBBelep.Enabled:=True; SBTervezo.Enabled:=false;
SBBeiro.Enabled:=False; DeleteTab.Enabled:=false;
SBNewFelev.Enabled:=False; SBFelevDelete.Enabled:=False
SBAdd.Enabled:=false;  SBKod.Enabled:=False;
SBNev.Enabled:=false; TargyGrid.Enabled:=False;
SBJegy.Enabled:=false; SBAtlag.Enabled:=false;
TargyTab.Enabled:=false; NewTab.Enabled:=false;
Edit1.ReadOnly:=true;  Edit2.ReadOnly:=true;  Edit3.ReadOnly:=true;
SpinJegy.ReadOnly:=true; SpinAtlag.ReadOnly:=true;
```

Egyébként pedig ha az *OK* gombra nyomunk létrehozom a felhasználót. Először is növelem a *UserNr* globális változó számát egyel. Majd létrehozom az adatbázisban a soron következő táblát table'i' néven.

A létrehozás a *CommandText*-nek megadott SQL utasítással történik:

```
UserCommand.CommandText:='CREATE TABLE Table'+IntToStr(UserNr)+' ('+
'Felev INTEGER,'+
'Kod TEXT(20) PRIMARY KEY,'+
'Targynev TEXT(50),'+
'Kredit INTEGER, '+
'Elofeltetel TEXT(20),'+
'Jegy TEXT(2))';
UserCommand.Execute;
```

Ezek után a logikai *Ini* fájl-t megalkotom és beleírom az adott felhasználónevet egyenlővé téve a létrehozott táblanévvel, valamint hozzáadom a felhasználóneveket tartalmazó *ListBox*-hoz. Mindezt természetesen egy *try* kivételkezelőben melynél a *finally* ágban felszabadítom az Ini állományt. Aztán a logikai változó értékének *true*-t adok ezzel jelezve, hogy létrehoztam egy új felhasználót, majd ezt üzenetben is közlöm a felhasználóval. Ezután megvizsgálva a logikai változónkat annak igaz értéke esetén rekurzív módon újra meghívom ezt a metódust melynél az eddig leírt rész már nem fog lefutni, mivel a felhasználónév biztosan létezni fog hiszen most hoztuk létre. Ezért az ezt vizsgáló feltétel *else* ágáról folytatjuk a programot. Rögtön egy kivételkezelővel kezdtem mely az eddig ismert módon létrehozza a logikai Ini fájlt és kiolvassa belőle a fájlból az adott felhasználónévhez tartozó táblanevet. Ezek után felszabadítom a logikai állományt. Ezután bezárom az adatbázist, a *CommandText*-nek megadom azon tábla nevét, amelyik az adott felhasználóhoz tartozik és megnyitom. A begépelt azonosító háttérszínét megváltoztattam ezzel is jelezve, hogy a felhasználó be van lépve. Aztán a db változónak értékül adom, hogy hány félév van az adott táblában jelenleg eltárolva. Erre egy private függvényt alkalmazok melyet később ismertetek. Ezek után letisztázom a *TabControl*-t a *Clear* metódusával és létrehozok rajta pontosan annyit fület, ahány félév le volt tárolva a táblában. Valamint ugyanennyi *DBGrid*-et is létrehozok melyeket az ennek fenntartott tömbben gyűjtök össze. A létrehozáskor beállítom a *DBGrid*-ek pozícióját, az *Options* tulajdonságukat, melyeknél különböző apró beállítások vannak felsorolva. Valamint az *OnDoubleClick* metódusokhoz hozzárendelem az általam megírt *UserDoubleClick* metódust. Erre azért van szükség, mert mivel futási időben jönnek létre a *DBGrid*-ek nem lehet a

megszokott módon az *Object Inspector* segítségével. A hozzárendelés úgy történik, mint egy értékadás csak itt az érték maga a függvény:

```
felev[i].Options:=[dgTitles,dgIndicator,dgColumnResize,dgColLines,dgRowLines,dgTabs,dgR
owSelect,dgConfirmDelete,dgCancelOnExit];
felev[i].OnDblClick:=UserDBClick;
```

A *UserDoubleClick* metódust később ismertetem. Ezután csökkentem a db változó számát egyel mivel a létrehozáskor az indexelés 0-tól kezdődik és ezt az eltérést ki kell küszöbölni a további félévek létrehozása, törlése miatt. A végén még meghívom a *TabControl OnChange* metódusát.
A *kilépes* gombra kattintva meghívódik az *SBKilep* gomb *OnClick* metódusa, melynek egyetlen utasítása befejezteti a programot.
A *private* láthatóságú *Felevmeghat* függvény arra szolgál, hogy meghatározza az adott táblában hány félév van eddig letárolva. Visszatérési értéke a legmagasabb félévszám lesz. Egy lokális változó használtam fel benne, mely a visszatérési értéket szolgáltatja majd. A függvény elején ezt -1-re inicializáltam. Aztán bezárom az adatbázisomat majd beállítom a *CommandText*-ét, hogy biztosan megjelenjen az összes adat, ami a táblában található. Aztán újra megnyitom és a tábla első elemére pozícionálok. Majd egy *while* ciklussal végigmegyek a tábla elemein és vizsgálom, hogy a tábla első mezőjében találok-e az eddig letároltnál nagyobb számot. Ha találok akkor kicserélem a nagyobbra. Ebből kikövetkeztethető, hogy a tábla első oszlopában azt tárolom, hogy melyik tárgyat melyik félévben vettem fel. A ciklus után a *b* értékét értékül adom a *Result*-nak és így ez lesz a függvényünk visszatérési értéke.
A függvényből egy részlet:

```
while not UserDataSet.Eof do
begin
if UserDataSet.Fields[0].AsInteger>b then
b:=UserDataSet.Fields[0].AsInteger;
UserDataSet.Next;
end;
Result:=b;
```

Új félévet létrehozni az új félév gomb *OnClick* metódusával lehet. Lokális változója nincs. A függvény elején megvizsgáljuk, hogy elértük-e már a maximális félévszámot. Ha igen ezt közöljük a felhasználóval. Ha nem növeljük a db számot, majd a féléveket megjelenítő *TabControl*-on létrehozunk egy újabb fület. A *felev* tömbünkbe létrehozunk egy újabb *DBGrid*-et, mely *db*-adik helyen lesz eltárolva. Beállítom a szülőt és a megfelelő pozíciót, az *Options* tulajdonságot valamint hozzárendelem az *OnDoubleClick* metódusához a *UserDBClick* metódust, mely a tantárgyak törléséért lesz felelős. Majd a végén meghívom a *TabConrol OnChange* metódusát.

A félévek törlése bonyolultabb feladat mint azoknak a létrehozása. Figyelni kell a pontos felszabadításra, valamint vizsgálni kell, hogy csak olyan félév törölhető melyben nincsenek tantárgyak. Félévet mindig a végéről törlünk hiszen furcsán mutatna ha törölnénk az első félévet, de második félévünk viszont lenne. Először is meghatározzuk, hogy melyik a legmagasabb félévszámunk, melyet az *akt_db* változóban fogunk tárolni. Majd megnézzük, hogy a *db* szám nagyobb-e 0-nál, mivel a két esetet külön kezeljük. Aztán vizsgáljuk, hogy a *db* nagyobb-e mint az *akt_db* változónk. Ha nagyobb azt jelenti, hogy nyugodtan törölhetjük a félévet. Először csökkenteni kell-e a *TabControl* indexét. Erre azért van szükség, mert nem csak a *DBGrid*-et kell törölnünk, hanem a *TabControl* egy fülét is és nem akarjuk hogy az indexe nem létező fülre mutasson. Majd felszabadítjuk a *felev* tömb *db*-adik elemét, töröljük a *TabControl db*-adik elemét, csökkentjük a *db* változó értékét eggyel. Az aktualizáláshoz hívjuk a *TabControl OnChange* metódusát. Ha nem törölhető a félév szintén hívjuk az *OnChange* metódust és tudatjuk a felhasználóval, hogy miért nem törölhető. Ha a *db* változó értéke egyenlő nullával, akkor szintén megvizsgáljuk, hogy a *db* változó értéke nagyobb vagy egyenlő az *akt_db* változó értékével. Ha igaz a feltételünk felszabadítjuk a tomb *db*-adik elemét, kitisztázzuk a *TabControl*-t a *Clear* metódusával és csökkentjük a *db* változó értékét eggyel. Az *OnChange* metódus hívása itt hibát okozna, mivel a tömbünk üres és nem tudna kapcsolatot teremteni az adatbázis és a megjelenítők között. Ha a feltételünk hamis közöljük a felhasználóval, hogy miért nem tudja törölni az utolsó félévet. A metódus nem használt lokális változókat. A következő programrészlet bemutatja azon esetet, ahol a db változó értéke nagyobb mint 0:

```
if (db>=akt_db) then
begin
if db=Tabs.TabIndex then Tabs.TabIndex:=Tabs.Tabindex-1;
felev[db].Free;
Tabs.Tabs.Delete(db);
Dec(db);
Tabs.OnChange(Self);
end else
begin
Tabs.OnChange(Self);
Showmessage('Nem törölhető mert tárgyak vannak benne');
end;      //else
end          //if
```

Tárgyat felvenni legegyszerűbben a tárgyon való dupla kattintással lehet. Ezt a tárgyakat megjelenítő *TargyGrid OnDoubleClick* metódusa intézi. A metódus tartalmaz egy egész típusú, három *String* és két logikai típusú lokális változót. Az egész típusút ciklusváltozóként használtam fel. A logikai változók különböző események figyelésre vannak, a *String* típusúak pedig az előfeltételek kódjainak szétbontásásra, tárolására vannak fenntartva. Az egész metódus egy feltételben van beágyazva ami figyeli, hogy csak akkor lehessen törölni, ha már van miből. Ha tárgyat veszünk fel a két nézet közül mindig tervező nézetben kell lennünk, hiszen egy tárgyfelvétel mindenképpen a tervezéshez tartozik. Ennek megfelelően az elején beállítom a nézetet jelző gombokat majd törlöm azon *ListBox*-ok elemeit melyeket fel fogok használni az előfeltételek megvizsgálására. Az *s* stringbe beolvasom a kiválasztott tantárgy előfeltételeit. Az előfeltételek azon tantárgyak kódjai melyek teljesítése szükséges a választott tárgy felvételéhez. Ezek egy mezőben helyezkednek el tetszőleges számú ,-vel és szóközzel elválasztva. A beolvasás tulajdonképpen nem más mint egy értékadás, hiszen a kattintással automatikusan kiválasztunk egy sort a táblából, és ezen sor előfeltétel mezőjét adjuk értékül az *s* változónak. Ezután megvizsgálom, hogy van-e egyáltalán előfeltétele az adott tantárgynak. Ha van szétbontom a stringemet a következő módon. A stringemben minden szóközt vesszőre cseréle, így ezek után már csak egy karaktert kell figyelnem és kiszűrnöm. Aztán levágom a

string elején és végén lévő felesleges vesszőket, ha van. Majd egy ciklussal végigmegyek a stringen, melyhez a *pos* függvényt használtam fel. Aztán kimásolom egy *ListBox*-ba a tantárgy kódját. Ezután törlöm a , karaktereket a stringből ha még voltak. Ha voltak azt jelenti van még van másik előfeltételünk is, melyet az előbb leírt módon szintén kimentek. Ezek után az előfeltételeink egy *ListBox*-ba kerültek tételenként letárolva, mely kezelése így már rendkívül egyszerű. A szétbontás programrészlete:

```
s:=TargyDataSet.FieldByName('Elofeltetelek').AsString;
if s<>'' then
begin
while (pos(' ',s)>0) do s[pos(' ',s)]:=',';
while (s[1]=',') do delete(s,1,1);
while (s[length(s)]=',') do delete(s,length(s),1);
while pos(',',s)>0 do
Begin
EloFeltetel.Items.Add(copy(s,1,pos(',',s)-1));
Delete(s,1,pos(',',s));
while (s[1]=',') do delete(s,1,1);
end;
EloFeltetel.Items.Add(s);
end;
```

Láthatjuk, hogy a ciklusok nem ciklusváltozókkal megadott értékig haladnak, hanem beépített függvények szolgáltatják a kilépési feltételt. A *pos* függvény az első paraméterben megadott karakterig halad a második paraméterben megadott stringben. A *delete* függvény első paramétere, hogy melyik stringben kell törölnünk. A második paramétere, hogy honnan kezdjük a törlést, a harmadik pedig, hogy hány karaktert kell törölnünk. A *copy* függvénnyel megadjuk, hogy melyik stringből mettől meddig másoljunk. Bezárjuk az adatbáziunkat, beállítjuk a *CommandText*-et és megnyitjuk újra. A *van* logikai változónkat *false*-ra állítom. Ennek *true* értéke fogja jelezni, hogy már felvettük a tárgyat. A felvett string típusú változó értékét kinullázom. Ebben fogom tárolni, hogy melyik félévben vettem már fel a tárgyat, ha ez

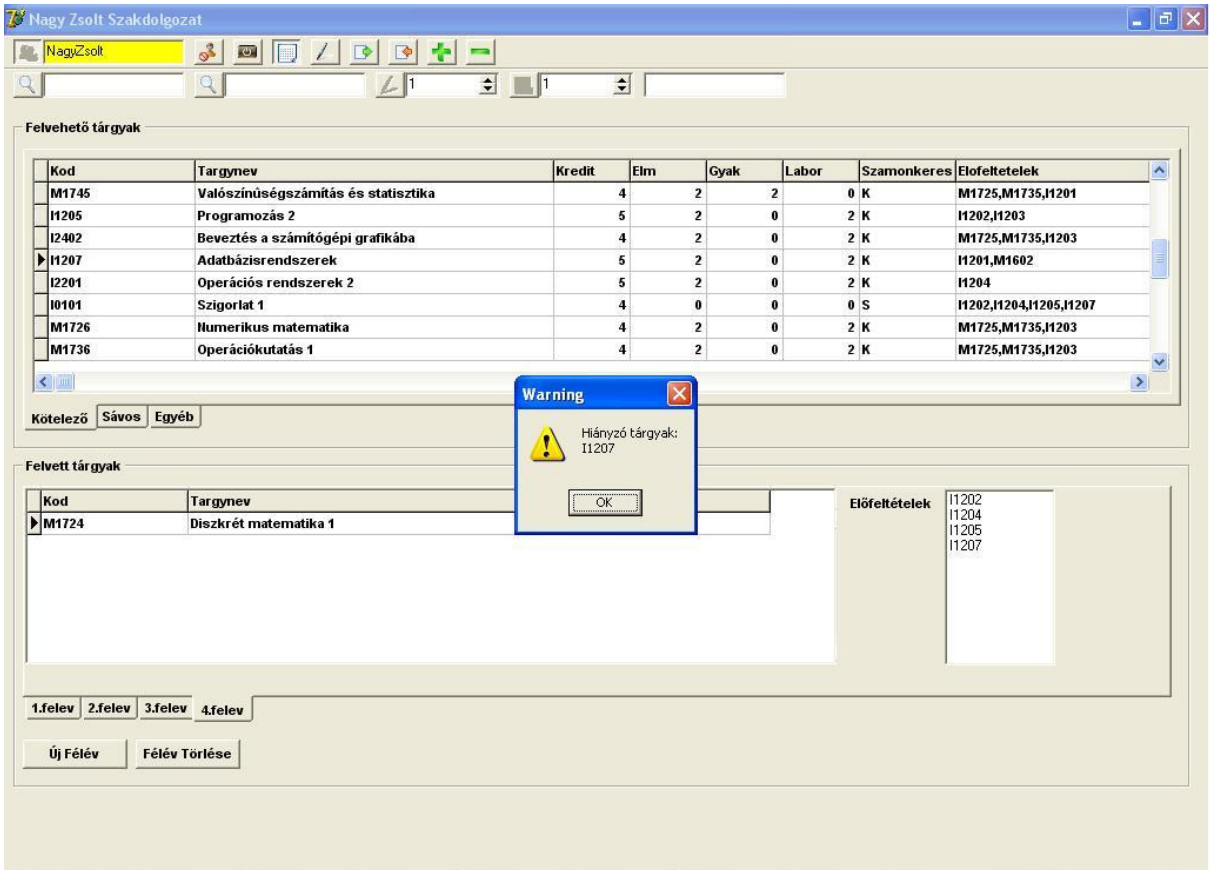
az eset áll fenn. Ezután a *User* adabázisban egy keresést hajtok végre a *Locate* metódussal. Paraméterként meg kell adni, hogy melyik mezőben keressen, mit és hogyan.

```
if UserDataSet.Locate('Kod',TargyGrid.Fields[0].AsString,[loPartialKey]) then
```

Ha megtaláltuk ezen tantárgykódot akkor a *nezet* változónkat egy értékűre állítom, kimentem, hogy hanyadik félévben vettem fel a tantárgyat. A *van* logikai változó értékét *true*-ra állítom, hívom a *TabControl OnChange* metódusát és üzenetben közlöm a felhasználóval, hogy már felvette ezt a tárgyat. Segítség képpen közlöm, hogy melyikben. Majd egy feltételben vizsgálom, ha még nem vettem fel a tantárgyat, akkor megnézem megvan-e minden előfeltétele. Ehhez először is be kell zárjam az adatbázist. A *CommandText*-t pedig úgy állítom be, hogy csak azok a tantárgyak jelenjenek meg a megnyitáskor, melyek azon félév előtt lettek teljesítve, amelyikben a tárgyat fel akarom venni. Ezzel a megoldással kiküszöbölöm azon hibákat, hogy a tárgy előfeltétele teljesítve van, de későbbi félévben, mint amikor fel akarom venni. Ugyanezen megoldás figyel arra is, hogy ne vehessek fel egy tárgyat ugyanabban a félévben amikor az előfeltételét csinálom. Egy ciklussal végigmegyek az előfeltételeken. Ha egy tantárgy kódját nem találom meg, akkor a *feltetel* nevű változót *false*-ra állítom, ezzel jelezve hogy van nem teljesített tantárgy. Ezután ráállok a hiányzó előfeltételre és a *Hiany* nevű *ListBox*-hoz hozzáadom a tantárgykódot. Ezek után egy feltételben ellenőrzöm a logikai változók értékét. Ha még nem vettem fel a tantárgyat és megvan minden előfeltétele, akkor az adatbázisban létrehozok egy új üres rekordot és a mezőit egyenként átmásolom. A másolás után az adatbázisra alkalmazom a *Post* metódust mely véglegesíti a változásokat. Az új elem létrehozása és adatokkal való feltöltése:

```
UserDataset.Insert;
UserDataSet.FieldByName('Felev').AsInteger:=Tabs.TabIndex+1;
UserDataSet.FieldByName('Kod').AsString:=TargyGrid.Fields[0].AsString;
UserDataSet.FieldByName('Targynev').AsString:=TargyGrid.Fields[1].AsString;
UserDataSet.FieldByName('Kredit').AsInteger:=TargyGrid.Fields[2].AsInteger;
UserDataSet.FieldByName('Elofeltetel').AsString:=TargyGrid.Fields[7].AsString;
UserDataSet.Post;
```

Ha ez megvan bezárom az adatbázist beállítom a *CommandText*-et és újra megnyitom. Lehet olyan eset is mikor még nem vettem fel egy tantárgyat, de nincs meg az előfeltétele. Ezt szintén a logikai változók vizsgálatával nézem meg. Ebben az esetben tájékoztatom a felhasználót a hibáról, kiírom hogy melyik tantárgyak kódjait kell megkeresni, mivel azok még nincsenek teljesítve. A függvény végén még letiltottam két gombot. Erre azért volt szükség, mert ha karbantartó nézetben voltam éppen a gombok még aktívak voltak, de automatikusan átkerülök tárgyfelvételkor tervező nézetbe és akkor már a gombok nem lehetnek aktívak.
Egy hibaüzenetet adó kép melyen látható, hogy a *Listbox*-ban tárolva vannak a választott tárgy előfeltételei és a hibaüzenet tartalmazza a hiányzó előfeltételt is.Érdemes megfigyelni, hogy a hiányzó előfeltételre pozícionált a program az adott táblában.



Tantárgyak törlésére a legegyszerűbb megoldás a tantárgyon való dupla kattintás. Mivel a féléveket megjelenítő *DBGrid*-ek futási időben jönnek létre, ezért nem lehet az *Object Inspector* segítségével megadni a metódust. Írni kell egy *private* láthatóságú metódust és azt

kell hozzárendelni a *DBGrid*-ekhez. A metódus neve *UserDBClick*. A hozzárendelést már korábban leírtam. Három lokális változója van, ebből kettő *String* típusú egy pedig *Boolean* típusú. A metódus elején inicializálom a változókat, törlöm az elemeit a felhasználni kívánt *ListBox*-nak és kimentem a *kod* nevű *String* típusú változómba a törölni kívánt tantárgy kódját. Ezután bezárom az adatbázist és beállítom a *CommandText*-et. A beállítást úgy végzem, hogy csak azok a tantárgyak jelenjenek meg a megnyitáskor, melyek későbbi félévekben vannak felvéve, mint ahol a törölni kívánt tantárgy található. Ezután megnyitom az adatbázist. Megnyitás után végigmegyek az adatbázison egy ciklus segítségével. Közben minden egyes tantárgy előfeltételében vizsgálom, hogy nem szerepel-e a törölni kívánt kód. Ezt a következő képpen néztem meg. Egy *String* típusú változóban eltárolom a tantárgy előfeltételeit. Ezt a korábban ismertetett módon részekre bontom és egy üres *ListBox*-ban tárolom. Majd megvizsgálom, hogy szerepel-e a *Listbox*-ban a törölni kívánt tantárgy kódja. Ha szerepel akkor a *torol* logikai típusú változónak *false* értéket adok. A *Hiany* nevű *ListBox*-ban pedig tárolom, hogy melyik tárgyak épülnek erre a tantárgyra. Ha nem szerepel benne akkor a *torol* értéke változatlan marad, azaz *true*. Ezután a vizsgálatot kezdem előről a soron következő tantárgynál. Majd hívom a *Tabs TabControl OnChange* metódusát. Ezután feltételben vizsgálom a *torol* logikai változó értékét. Ha *true* értékkel rendelkezik akkor rápozícionálok a törlendő tárgyra. Azért kell újra rápozícionálni, mert a keresés során az utolsó elem lett az aktuális. Aztuán törlöm azt és hogy láthassuk az eredményt hívom ismét a *Tabs TabControl OnChange* metódusát. Ha a *torol* változónk *false* értékkel rendelkezik hívom a *Tabs TabControl OnChange* metódusát és jelzem, hogy miért nem törölhető a kiválasztott tántárgy. A törlés az adatbázisból a következőképpen néz ki:

```
if (torol=true) then
if UserDataSet.Locate('Kod',kod,[loPartialKey])=true then
begin
UserDataSet.Delete;
Tabs.OnChange(Self);
end;
```

A kereséseket két egyszerű függvénnyel írtam meg. Mindkettő egy *OnClick* metódus. A név szerinti keresés az *SBNev*, a kód szerinti keresés az *SBKod* nevű *SpeedButton OnClick* metódusa. Mindkettőben a megadott szöveget vizsgálom, de annyi különbséggel hogy az egyiknél a *Kod* nevű oszlopban keresek, a másiknál viszont a *Targynev* nevű oszlopban. Ha megtalálom a keresett stringet, nem közlök információt, hogy feleslegesen ne szakítsam meg a felhasználó munkáját. Ha nem találom meg csak abban az esetben közlöm, hogy nincs ilyen nevű, vagy kódú tantárgy. A keresés nem teljes keresés, lehet bizonyos részstringekre is keresni.
A két keresés kódja a következő:

```
if not TargyDataSet.Locate('Kod',Edit2.Text,[loPartialKey])then
Showmessage('Nincs ilyen kódú tantárgy');

if not TargyDataSet.Locate('Targynev',Edit3.Text,[loPartialKey])then
Showmessage('Nincs ilyen nevű tantárgy');
```

A *ToolBar*-ról is lehet vezérelni a félévek hozzáadását vagy törlését. Ilyenkor nem kell újra megírni a függvényt, hanem csak egy hivatkozást kell tennünk rá a következő módon.

```
NewTab.Click;
```

Ekkor meghívódik az *NewTab* nevű gomb *OnClick* nevű metódusa. Az egész metódusunk ilyenkor egyetlen sorból áll.
A félév törlése szintén vezérelhető ugyanolyan módon, mint ahogyan azt az előbb is leírtam. Annyi különbség van hogy itt a *DeleteTab* nevű gomb *OnClick* metódusára hivatkozok.
A tantárgy törlése és felvétele szintén vezérelhető a *ToolBar*-ról. A kód ugyanúgy egy metódusra hivatkozik, de itt kicsit más a helyzet. Mivel eddig a dupla kattintással jelöltük ki, hogy melyik legyen a törölni illetve a felvenni kívánt tantárgy. Mivel nincs kattintás az éppen aktuálisan kijelölt tantárgy lesz felvéve illetve törölve. A *DBGrid*-ek jobb oldalán látható egy kis kék háromszög mely jelöli az aktuális rekordot.
A nézetek közötti váltást két gombbal tehetjük meg. Ezeknek a gomboknak az *OnClick* metódusa fut le ilyenkor. Ha a tervező nézetet választjuk akkor rögtön a metódus elején le lesz

tiltva a jegy beírását és az átlag számolását szolgáló két gomb. Mivel mindkettő nyomógomb, tehát lenyomásuk után lent is maradnak gondoskodnunk kell arról, hogy ha az egyik le van nyomva a másik ne legyen. Ezért a másik gomb *Down* tulajdonságát *false* értékűre állítom. Ezután megnézem, ha van megnyitott *UserDBGrid* vagyis a *db* változó értéke nagyobb vagy egyenlő mint 0 akkor a *nezet* változó értékét egyre állítom és meghívom a *Tabs TabControl OnChange* metódusát.

A karbantaró nézetre történő váltásnál is nagyjából ez a helyzet, de pont az ellenkezője történik mint eddig. Vagyis hogy engedélyezzük a jegyek beírásáért és az átlag számításáért felelős gombokat, majd ha van megnyitott félévünk, akkor a *nezet* változó értékét kettőre állítom és hívom *Tabs TabControl OnChange* metódusát.

A jegy beírása szintén egy *OnClick* metódussal történik. A beírása csak akkor történik meg, ha az aktuális elemünk nem üres. Pontosabban a tárgykódja nem lehet üres.

Ha a feltételünk igaz kijelöljük módosításra az aktuális elemet az adatbázis *Edit* metódusával, majd a *Jegy* nevű mezőhöz beírjuk a megadott értéket. Ezután a *Post* metódus hívásával érvényesíttetjük a változásokat.

Az átlag kiszámítása csak karbantartó nézetben lehetséges. Ezt az *SBAtlag* gomb *OnClick* metódusával tehetjük meg. A metódusban felhasználtam két lokális változót. Mindkettő egész típusú. A metódus elején inicializálom a változókat, melyeket használni akarok. Ezek között van az *atlag* nevű *real* típusú változó is, melyet az átlag tárolására fogok felhasználni. Ezután megnézem, hogy van-e egyáltalán megjelenített félév, mert ha nincs nem is számolok átlagot és nulla értéket fogok szolgáltatni. Ha van félévünk akkor eltároljuk mely félév átlagára vagyunk kíváncsiak. Ezután bezárjuk az adatbázist, beállítjuk a *CommandText*-et és újra megnyitjuk, majd az első elemére pozícionálunk. Egy ciklussal végigmegyünk az elemein. Közben vizsgáljuk, ha valamely elem az adott félévhez tartozik melynek átlagára kíváncsiak vagyunk, akkor annak kreditszámát hozzáadjuk az összkredithez, a jegy érték pedig szorozva a kredittel hozzáadjuk az *atlag* változóhoz. Ezután megnézzük, ha az összkreditszámunk nulla, akkor az átlagunk is nulla lesz. Ezt azt jelenti, hogy vannak tárgyak a félévben, de még nincsenek hozzá jegyek beírva. Különben pedig az *atlag* változónkat osztjuk az összkredit értékünkkel. A kapott értéket stringgé konvertáljuk és megjelenítjük. A végén hívjuk a *Tabs TabControl OnChange* metódusát.

A tényleges számolást végző programrészlet:

```
while not UserDataSet.Eof do
begin
if UserDataSet.Fields[0].AsInteger=fev then
if UserDataSet.Fields[5].AsString<>'' then
begin
okredit:=okredit+UserDataSet.Fields[3].AsInteger;
atlag:=atlag+(UserDataSet.Fields[3].AsInteger*StrToInt(UserDataSet.Fields[5].AsString));
end;
UserDataSet.Next;
end;    //while
```

Az egér görgetését lekezelő eljárás a *AppEventsMessage*. Az eljárás egy darab lokális változót használ melynek típusa *SmallInt*. Az eseményt felépítő belső függvény statikus felépítésű és ezért lényeges a változó típusok egyezése. Más típusokkal funkcionálisan nem működne az eljárás. Az eljárás egy feltétellel indul, amely az egész eljárást keretbe foglalja. A feltételben vizsgáljuk, hogy a delphinek küldött üzenet egérgörgetés-e. Ha nem az, akkor nem foglalkozunk vele, ha igen akkor átalakítjuk az üzenetet billentyűleütéssé. Az üzenet egy osztály. A *wparam* tulajdonság vizsgálatával döntjük el, hogy fel illetve le görgetés történt-e. Az értékét egész típusúvá konvertáljuk. Ha ez az érték nagyobb, mint nulla, akkor felfelé görgetés történt, ha kisebb akkor pedig lefelé. Az eredeti *WM_MouseWheel* eseményt kicseréljük ennek megfelelően *VK_Up* és *VK_Down* eseményekre. Ezek a fel és a le nyilak lenyomását jelentik a billentyűzeten. A végén a *Handled* tulajdonság értékét *false*-ra állítjuk, amire azért van szükség, hogy az új *VK_Up* és *VK_Down* eseményeket ismét lekezelje a program.
Az eljárás kódja a következő:

```
if Msg.message=WM_MOUSEWHEEL then
begin
Msg.message:=WM_KEYDOWN;
Msg.lParam:=0;
```

```
i:=HiWord(Msg.wParam);
if i>0 then Msg.wParam:=VK_UP
else Msg.wParam:=VK_DOWN;
Handled:=False;
end;
```

6. Összefoglalás

A programban rengeteg előre megírt komponens és tulajdonságaik kerültek felhasználásra, melyeknek köszönhetően a programozás gyorsan és egyszerűen történhet. Rövid idő alatt elkészíthetőek az alkalmazások, ám bizonyos előre megírt tulajdonságok úgymond javításra szorulnak. Valamint a különféle hivatkozások sokszor hibákat okoznak a programban, melyek megtalálása és lekezelése sok időt vesz igénybe. Illetve fellépnek olyan problémák is, melyeket bizonyos komponensek használta idéz elő. Ilyen például az egér görgőjének a lekezelése.

A már előre megírt adatbziskezelő komponenseken túl könnyedén készíthetünk új teljesen igényeinknek megfelelő új komponenst az objektum orientált rendszer kihasználásával. Ennek nagy előnye a jó átláthatóság és egyszerű frissítési lehetőségek.

Az adatbázisok kezelése valóban nagyon egyszerűen történt, köszönhetően a megírt komponenseknek. Az előre definiált komponensek könnyedén összekapcsolhatóak ezzel egyszerűvé téve az adatbázis feldolgozását és megjelenítését. Az ADO komponenscsalád remekül illeszkedik szinte minden manapság haszált adatbázis típushoz. Esetünkben az Access adatbázisunkon könnyen elvégezhetünk bármilyen műveletet. A megjelenítésre a felhasználtakon kívül még rengeteg lehetőség van, melyekhez különféle keresési módok kapcsolódnak, melyek jelentősen megkönnyíthetik az adatbázisok használatát.

7. Irodalomjegyzék

Baga Edit: Delphi másképp

Benkő Tiborné, Benkő László, Dr. Tamás Péter: Windows alkalmazások fejlesztése Delphi 3 rendszerében

Marco Cantú:Delphi 7 I-II. kötet

8. Köszönetnyilvánítás

Ezen a helyen szeretném megköszönni Dr. Bölcskei András tanár úr sokoldalú segítségét, aki a vizsgálandó témára a figyelmemet felhívta, megismertetett az alapvető szakirodalommal és a dolgozat végső formájának kialakításához sok hasznos tanácsot adott.