Webes Alkalmazás-fejlesztés Borland Delphiben

Szabó Zsolt

2007. május 8.

Tartalomjegyzék

1. fejezet

Bevezetés

A szakdolgozatom címe: **Webes Alkalmazás-fejlesztés Borland Delphiben.** A címből kiderül, hogy webes alkalmazást készítek, tehát elkészül egy olyan script amely weblapokat tartalmaz. De a program alapja egy asztali alkalmazás lesz, amelyet csak kiegészít egy webes felület.

Napjainkban nagy teret hódítanak a webes alkalmazások, és ezek elkészítéséhez számtalan eszköz áll rendelkezésre. Tanulmányaim során jó néhány programozási-nyelvbe nyertem betekintést. Ezek közé tartozik a Delphi nyelv is. A Delphi és az Adatbázis-kezelés Borland Delphiben tárgyak elegendő ismerettel vérteztek fel ahhoz, hogy szakdolgozatom témájául ezt a nyelvet válasszam. Lehet, hogy webes alkalmazás-fejlesztéshez sokan más eszközt, más programozási nyelvet választottak volna, de én úgy gondoltam, hogy az általam szabadon választott program megírásához tökéletes eszköz a Delphi fejlesztői környezet.

Természetesen programomban óriási szerepet kap az Adatbázis-kezelés. Nehezen tudnék elképzelni olyan alkalmazást ahol valamilyen szerepet ne kapna az adatok kezelése. Az én programomban is komoly szerepet kap az adatok bevitele és struktúrált lekérdezése. Lehetőségeimhez képest tervezem a program további fejlesztését és javítgatását.

2. fejezet

Objektum Orientált programfejlesztés Delphiben

A Delphi fejlesztőkörnyezet a Pascal programozási nyelv objektumközpontú kiterjesztésére, az Object Pascal nyelvre épül. A legtöbb modern programozási nyelv támogatja az objektum-orientált vagy objektum-központú programozást (OOP).

Az OOP nyelvek három pillére a betokozás (*Encapsulation* - ezt általában osztályokkal valósítják meg), az öröklődés (*Inheritance*) és a többalakúság (*Polymorphism*, vagy késői kötés).

2.1. Osztályok definiálása, objektumok példányosítása

Az osztály kifejezés az objektumorientált programozás sarokköve. Az osztályok adatokból és eljárásokból állnak. Ha az osztályhoz adatok tartoznak, akkor megállapodás szerint ezeket mezőknek (*Field*), az osztályokhoz tartozó eljárásokat pedig metódusoknak (*Method*) nevezzük. Bármi, ami valamely osztály tagjaként van definiálva, *Attribute*.

Az osztályok tagjaira vonatkozó nyelvhasználat bevezetésének célja annak hangsúlyozása, hogy az osztályokba sorolt adatok és eljárások nem azonosak a közönséges adatokkal és eljárásokkal. Helyes használatukhoz több információ szükséges.

A Delphi megalkotásakor az Object Pascal programnyelvet használták, tehát architektúrája objektumorientált, törzsosztálya pedig a *TObject*. A Delphi minden osztályának őse a *TObject* osztály.

2.1.1. Az osztályok alapvető szintaxisa

Minden osztálydeklaráció ugyanazon egyszerű szintaxis szerint történik.

```
TMyClass = class
end;
```

A fenti lista egy attribútumok nélküli *TMyClass* osztályt definiál.

A következő definíció egyenlő az előbbivel:

```
TMyClass = class (TObject)
end;
```

Az első lista az osztálydeklaráció, a második pedig egy alosztály, azaz az öröklés alapvető szintaxisát mutatja be. Ebben az esetben mindkét osztály megegyezik, mivel a Delphiben minden osztály implicit őse a *TObject*.

A *TObject* osztály rendelkezik egy *ClassName* osztálymetódus névvel, amely megadja az osztálysztring nevét. Ezt a metódust *RunTime Type Identification*nek (RTTI) nevezzük.

2.1.2. Állapotok leírása

Az osztályok egyik fő feladata, hogy tartalmazzák példányaik tervrajzát. Az állapotot leíró attribútumot adatattribútumnak nevezzül.

Az objektumorientált tervezés egyik legnagyobb kihívása a problémát megfelően leíró tulajdonságok kiválasztása. Az állapotattribútumok kódolása a tulajdonságok elnevezéséből és a megfelelő adattípus hozzárendeléséből áll.

```
TUser = class
  Nev: String;
  Lakohely: String;
  Eletkor: Integer;
  Email: String;
end;
```

Ebben a modelben egy felhasználó legalapvetőbb tulajdonságait tároljuk. Természetesen, tetszőleges bonyolultságú modellt készíthetünk ahol azokat a tulajdonságokat rögzítjük amelyek fontosak a modell állapotának tárolásához.

2.1.3. Képességek

Ha az adat azt definiálja, hogy az osztály mely példányai mit tudnak, akkor a képesség (*Capability*) azt adja meg, az osztály mire képes. Egy osztály képességeit a metódusok határozzák meg. A metódusok osztályok tagjaként deklarált eljárások és függvények, tehát a metódus deklarációja a **class** utasítás első sora és az **end** kulcsszó között található.

```
TUser = class
  Nev: String;
  Lakohely: String;
  Eletkor: Integer;
```

```
  Email: String;

  procedure Koltozik;
  procedure Leveletkuld;
end;
```

Ha az osztály forrásdefiníciójában a metódusokat a mezők és az adatok deklarációja előtt deklaráljuk *"Field definition not allowed after methods or properties"* hibajelentést kapunk.

Az implementációs szakaszban a metódusokat az osztályba nem tartozó eljárások definíciójával csaknem megegyező módon definiáljuk.

```
Implementation

procedure TUser.Koltozik;
begin
  // Ide jön a kód.
end;
```

Az összes eljárás definíciója az implementációs szakaszba, kódjuk pedig a **begin end** utasítások közé kerül. Az osztályokhoz tartozó metódusokra ugyanazok a kanonikus szabályok vonatkoznak, mint az osztályokhoz nem tartozó eljárásokra és függvényekre. Az eljárás esetén érvényes argumentum általában érvényes metódus esetén is.

2.1.4. Objektumpéldány létrehozása és megszüntetése

Amint definiáltunk egy osztályt, létre kell hoznunk a hozzá tartozó, objektumnak nevezett példányt is, az objektum adatait és képességeit csak ezután használhatjuk.

A példány létrehozása mindig azonos: *válzozónév := osztálynév.create*. Például egy *TUser* példány létrehozása:

```
var
  user: TUser;

begin
  user := TUser.Create;
  user.Nev := "Kiss József";
  user.Free;
end;
```

A *Create* metódus meghívása az objektumpéldány számára memóriát foglal le, a *Free* hívása pedig felszabadítja az adott memória területet.

2.2. Adatrejtés

Minden programozó két kő között örlődik: mikor kódot ír, ő maga a szerző, de ezzel egyidejűleg felhasználó is. Ha azonban a szerző és a felhasználó nem azonos, akkor is bizonyos kapcsolat áll fenn közöttük.

2.2.1. Private

Minden, a **private** kulcsszó után következő attribútum rejtve van az általános felhasználó számára, hozzáférése korlátozott, nem nyilvános. A **private** interfészt implementációs részeknek nevezzük, ezek azok a részletek, amik az osztály működését irják le. Amikor adatrejtésről beszélünk, ezek a privát interfészben található attribútumok azok, amelyek a felhasználó számára nem láthatók. A nyilvános interfész az, amit a felhasználó osztálynak lát, a privát pedig az, ahogyan az osztály működik.

Az objektumorientált programozás egyik kitétele, hogy minden adat legyen a privát interfészben, az adatokhoz való hozzáférés pedig csak a tulajdonságokon keresztül legyen lehetséges.

2.2.2. Protected

Egy osztály egy példányán keresztül nem férünk hozzá a védett metódusokhoz, adatokhoz és tulajdonságokhoz. Egy alosztály definiálásával azonban már lehetséges a közvetlen hozzáférés. Ha korlátozni akarjuk a védett attribútumokhoz való felhasználói hozzáférést, de a fejlesztőknek lehetőséget kell hagyni azok bővítésére vagy a hozzáférésre, az adatokat és metódusokat az osztály védett területén helyezzük le.

2.2.3. Public

Az alapértelmezés szerint az Object Pascal osztályok, ha az osztály vagy annak ősét nem a $M+ direktívával fordították, nyílvánosak. Az osztály egy részét a definícióban elhelyezett **public** kulcsszóval közvetlenül nyilvánossá tehetjük, így a **public** kulcsszó és az **end** között előforduló minden attribútum nyilvános. Az osztály nyilvános részében található attribútumokat **public** interface-nek nevezzük.

2.2.4. Published

A közzétett hozzáférésnek csak RAD eszközök esetén van értelme, mivel gyakorlatilag ugyanolyan, mint a nyilvános hozzáférés, de olyan attribútumok esetén használjuk, amelyeket tervezési időben is el kell érnünk. A közzétett tulajdonságokat és eseménytulajdonságokat komponensosztályokkal kell használni, ezek azok az attribútumok, melyek megjelennek az *Object Inspectorban.*

2.3. Öröklődés

Az öröklődés a már létező kód új attribútumokkal való ellátását jelenti. Amikor kódot írunk, azt ki kell próbálni, és végre kell hajtani a hibakeresést is. Ha módosítjuk a kódunkat, újra kell tesztelni és a hibakeresés is ismét előttünk áll, sőt, minden olyan kódot is ki kell próbálnunk, amely a módosított kódot használja.

Minderre azonban nincs szükség, ha már a létező viselkedést örökítjük át, azaz egy új osztályon, az alosztályon keresztül bővítünk.

Az alosztályok származtatásának több előnye is van. Először is nem a létező osztály kódját kell megváltoztatnunk, tehát elkerülhetjük a tesztelést éd az újbóli hibakeresést. Másodszor, csak az alosztály új kódját kell kipróbálnunk. Mivel az alosztály új, nincs tőle függő kód, amelyet szintén tesztelni kellene.

Az öröklést a következő szintaxis mutatja be:

```
type
  TÚjOsztály = class(TSzülőOsztály)
    // Új jellmezők.
    // Új viselkedés.
end;
```

Ha nem adunk meg szülő osztályt, az új osztály a *TObject* alosztálya, így minden új osztály definiálásakor öröklést használunk.

A **Delphi** (*C++* -val ellentétben) az egyedüli öröklést támogatja. Ez azt jelenti, hogy minden osztálynak csak egy közvetlen szülő osztálya van.

2.4. Polimorfizmus

Az Object Pascalban a se virtuálisként, se dinamikusként nem deklarált metódusok statikus metódusok, nincs rájuk külön fordítói direktíva. A virtuális és a dinamikus metódusok gyakorlati szempontból megegyeznek. A virtuális vagy dinamikus metódusok létrehozása összesen annyiból áll, hogy a metódusdeklaráció végére még odaírjuk a virtuális vagy dinamikus fordítói direktívát.

A virtuális metódusok a **Virtuális Metódus Tábla** (VMT) részei, míg a dinamikus metódusokat a **Dinamikus Metódus Tábla** (DMT) tartalmazza. Ha egy metódust virtuálisnak deklarálunk, az már magában rejti azt a szándékunkat, hogy a későbbiekben az osztályból alosztályokat származtatunk és az alosztályon belül a virtuális viselkedést majd felülbíráljuk. A statikus metódusokat nem lehet felülbírálni. A polimorfizmuson a virtuális metódusok mechanizmusát értjük.

2.5. Eseménykezelés

A Delphi komponenseit tulajdonságokon, tagfüggvényeken és eseményeken keresztül programozhatjuk. A legtöbb Delphi esemény az operációs rendszer üzeneteinek hatására váltódik ki, de nem feleltethetők meg egyértelműen az üzeneteknek. A Delphi események magasabb szintűek a rendszerüzeneteknél, ráadásul a Delphi komponensek közötti üzeneteket is használ.

Elméleti szempontból nézve az esemény az ablaknak küldött üzenet eredménye, és ez az ablak válaszolhat az üzenetre. A Delphiben egy komponens eseménykezelője általában a komponenst tartalmazó formhoz, és nem magához a komponenshez tartozik. Más szóval, a komponens az őt birtokló formra bízza eseményeinek kezelését. Ez a fajta átruházás a Delphi komponens alapú modelljének egyik alapköve.

Az események kezelése a nyelv egy másik szolgáltatására, a tagfüggvénymutatóra épül. A tagfüggvénymutató egy olyan eljárástípushoz hasonlítható, amely egy tagfüggvényre hivatkozik, de rendelkezik *Self* paraméterrel is. Más szóval, a tagfüggvénymutató két címet tárol: a tagfüggvény kódjának, illetve az objektumpéldánynak a címét. Ha ennek a tagfüggvénymutatónak a segítségével hívjuk meg a tagfüggvény kódját, akkor az objektumpéldányra a tagfüggvény törzsében, a *Self* kulcsszóval hivatkozhatunk.

Egy példa ahogy a Delphi egy gomb eseménykezelőjét és a hozzá kapcsolódó tagfüggvényt meghatározza:

```
type
  TNotifyEvent = procedure  (Sender: TObject) of object;

  MyButton = class
    OnClick: TNotifyEvent;
  end;

  TForm1 = class (TForm)
    procedure Button1Click (Sender: TObject);
    Button1: MyButton;
  end;

var
  Form1: TForm1;
```

Az eljárás belsejébe a következő sort írhatjuk:

```
MyButton.OnClick := Form1.Button1Click;
```

2.6. Kivételkezelés

A Delphi fontos jellegzetessége, hogy támogatja a kivételek kezelését. A kivételekkel, amelyek szabványos módot biztosítanak a hibák és a váratlan helyzetek kezelésére, stabilabbá tehetjük programunkat. A kivételek segítségével a kód magját teljesen elválaszthatjuk a hibakezelő kódtól, így a program sokkal áttekinthetőbbé és logikusabbá válik, valamint tömörebb is lesz, nem szabdalják szét a program tényleges céljától eltérő hibakezelő kódrészletek.

Futásidőben a Delphi könyvtárak akkor váltanak ki kivételeket, ha valami hiba történik. Ha a kódot jól írtuk meg, akkor az felismeri a problémát és megpróbálja megoldani, ellenkező esetben a kivétel átadódik a hívó kódnak, és így tovább. Végül, amennyiben kódunk egyetlen része sem tudta kezelni a kivételt, a Delphi fogja kezelni azt, méghozzá úgy, hogy megjelenít egy szabványos hibaüzenetet, majd megpróbálja folytatni a program futását.

A kivételkezelés négy kulcsszóra épül:

- A *try* jelöli ki a védett kódblokk kezdetét.
- Az *except* jelöli a védett kódblokk végét. Ezt már a kivételkezelő utasítások követik.
- A *finally* olyan kódblokkot jelöl, amely akkor is végrehajtódik, ha hiba történt, és akkor is, ha nem. Ebbe a blokkba általában olyan tisztogató műveletek kerülnek, amelyeket mindenképpen végre kell hajtani.
- A *raise* utasítással mi magunk válthatunk ki kivételeket. A legtöbb kivételt, amelyekkel a **Delphi** programozása során találkozunk majd, a rendszer váltja ki, de mi is készíthetünk saját kivételeket a kódban, ha érvénytelen vagy ha egymással összhangban nem álló adatokkal találkozunk futásidőben. A raise kulcsszót a kivételkezelőn belül a kivétel újbóli kiváltására használhatjuk, ezzel a módszerrel továbbadhatjuk a kivételt a következő kezelőnek.

Példa egy kivételkezelőre:

```
try

  //Valami tevékenység.

except
  on  //valami hiba típus.  do
  begin
    //A hiba kezelése.
  end;

  on  //Valami hiba típus.  do
```

```
  begin
    //A hiba kezelése.
  end;

finally
  //Amit mindenféleképpen végre akarunk hajtani.
end;
```

3. fejezet

Adatbáziskezelés Delphiben

3.1. Adabázis-kezelési architektúrák

Minden adatbázis-kezelési architektúrában három fő funkcionális egységet különböztetünk meg:

- A közvetlen adatkezelés (*Data processing*) Az alkalmazásnak ez a része végzi el a tárolt adatok fizikai feldolgozását: állományok nyitása, zárása, indexelések,a lekérdezések futtatása, új adatok felvitele, a meglévők törlése, módosítása. Ennek a résznek a megvalósítása függ az adatok tárolási módjától, azaz a használt adatbázisformátumtól.

- Az alkalmazás logika (*Business logic*) Ez a rész felelős a teljes alkalmazás helyes működéséért: biztosítja az adatok védelmét, elronthatatlanságát, hatékony és kényelmes kezelését.

- A felhasználói felület (*User interface*) Ez a rész a felhasználóval való közvetlen kapcsolattartásért felelős. A felületnek minél tetszetősebbnek, barátságosnak, és ugyanakkor elronthatatlannak kell lennie. Az adatok helyességéért nem a felhasználói felület, hanem az alkalmazás-logika felel.

3.1.1. Fájl/szerver architektúra

A fájl-szerver architektúrában az alkalmazás mindhárom része egyetlen gépen helyezkedik el. Ameddig az adatok és az alkalmazás ugyanazon a számítógépen találhatók, addig ez a megoldás nem rendelkezik különösebb hátránnyal. A valós igények megkövetelik a közös "központi" gépen elhelyezkedő adatokon történő többfelhasználós munkát. A feldolgozandó adatok a hálózaton keresztül mindíg átkerülnek a célgépre, ahol a felhasználók saját gépükön használják ezeket.

Ebből fakad e technika nagy hátránya: a teljes adathalmaz tekintélyes méreteket ölthet, ezek átvitele akár többször is leterhelheti a hálózatot, vagy egy közös erőforrást.

Így az egyes felhasználók munkája gyakran a többiek munkáját gátolja. Emiatt szükség volt más architektúrák kidolgozására. Az új technikák jellemzőiből kiindúlva, utólag a fájl-szerver technikát egy rétegűnek (*single-tier*) nevezték el, mivel minden feladatot egy gép végez el.

3.1.2. Kliens/szerver architektúra

Ebben a techikában az alkalmazás két részre bomlik: az adatok közvetlen kezelése az adatok tárolásáért felelős központi gépen történik egy adatbázis-szervernek nevezett szoftver által. A kliens gépeken már csak az adatbázisba be nem épített alkalmazás-logika és a felhasználói felület kerül.

Ezt a modelt még ügyfél-kiszolgálónak is nevezik: az ügyfél gépen futó kliens alkalmazás lekérdez bizonyos jellemzőjű adatokat, a kérés elutazik a szerverhez, aki ezt a leválogatást elvégzi, és az eredményt visszaküldi az ügyfélhez. Tehát a hálózaton nem a feldolgozandó adatok közlekednek, hanem előbb elmegy a szerverhez a megfelelő adatfeldolgozó parancs, és csak a parancs eredménye utazik vissza. Emiatt a hálózati átvitel teljesítménye sokat javul.

Ezeknek az adatfeldolgozó parancsoknak kidolgoztak egy szabványos nyelvet, ezt nevezik **SQL**-nek (**Structured Query Language**). Tehát a kliensek **SQL** utasításokkal késztetik rá a szervert arra, hogy a megfelelő műveleteket ott helyben elvégezze, és az eredményt visszaküldje. Mivel ebben a technikában az alkalmazás két részre bomlik, ezt az architektúrát később két rétegűnek (*two-tier*) nevezték el.

3.1.3. A többrétegű architektúra

Ebben a technikában az alkalmazás részei kettőnél is több gépen helyezkednek el. *Adatbázis-szerver* ahol az adatok tárolása és közvetlen feldolgozása történik. A középső réteg ahol az alkalmazás logika foglal helyet. *Kliens alkalmazás* ahol a felhasználói felület található. A több rétegre való felosztás további előnyökkel jár (munkamegosztás, adminisztráció könnyítése). A kliens oldalon már ténylegesen csak a felhasználói interfész található, ezért ezt az alkalmazást "sovány" (*thin*) kliensnek is nevezzük.

3.2. Nyilt adatbázisú kapcsolhatóság (ODBC)

Az ODBC-t az 1990-es években tették ismertté, megalapozva egy protokolt az alkalmazások adatbázisokkal történő kommunikációjára. Ez egy API-meghatározás. Minden szoftverszállító felelős a DLL-ek építéséért, amelyek azonos API-eljáráscsomagokat alkalmaznak az ODBC API-val konzisztensen. Egy megadott szállító API-ja konzisztens módot nyújt arra, hogy a fejlesztőprogramok útján tudjon kommunikálni a szállító adatbázis-motorjával.

Az ODBC mögötti cél az, hogy az alkalmazások az ODBC API-ra íródjanak és a fejlesztők adatbázis-motorokat cseréljenek a kód kicserélése nélkül. Ez a lehetőség még mindíg jó ok az ODBC használatára.

3.2.1. ODBC DataSource Administrator

A *Microsoft* a 90-es évek elejétől kezdte el az ODBC adatforrás-adminisztrátort szállítani a *Windows* operációs rendszerrel. Az ODBC-adminisztrátor ODBC-meghajtókkal kapcsolatos neveket, bizonyos adatbázis fájlokat és bármely más információt tárol, amire egy bizonyos adatbázismotornak szüksége van. Amikor az elkészített nevet egy programban használják, a megfelelő ODBC DLL-t lehet használni a megfelelő típusú adatbázissal.

3.2.2. A kapcsolat tesztelése

Egy ODBC kapcsolat teszteléséhez nyissuk meg az *SQL Exploret*-t. Ez lehetővé teszi, hogy megtekintsük az ODBC kapcsolatokat és a táblákat, amire egy *alias* utal. Természetesen ezenkívül még sok más funkciója is van.

3.3. Borland Database Engine

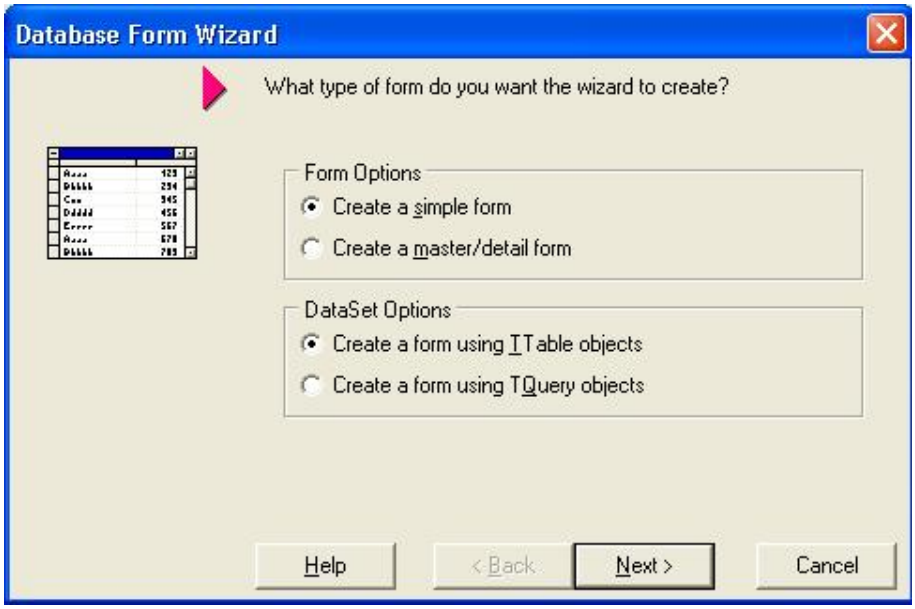
A Borland adatbázismotor (**Borland Database Engine**) egy API, ami belső adatbázistámogatást ad *Inprise* alkalmazásokhoz, a Delphi-t is beleértve. A **BDE Administrator** a Control Panel kialkalmazásai között van, ami lehetővé teszi olyan aliasok megadását, amelyek a Native BDE adatbázis-meghajtókat használják minden támogatott adatbázisra és ODBC aliasokra.

3.3.1. ODBC-Alias készítése

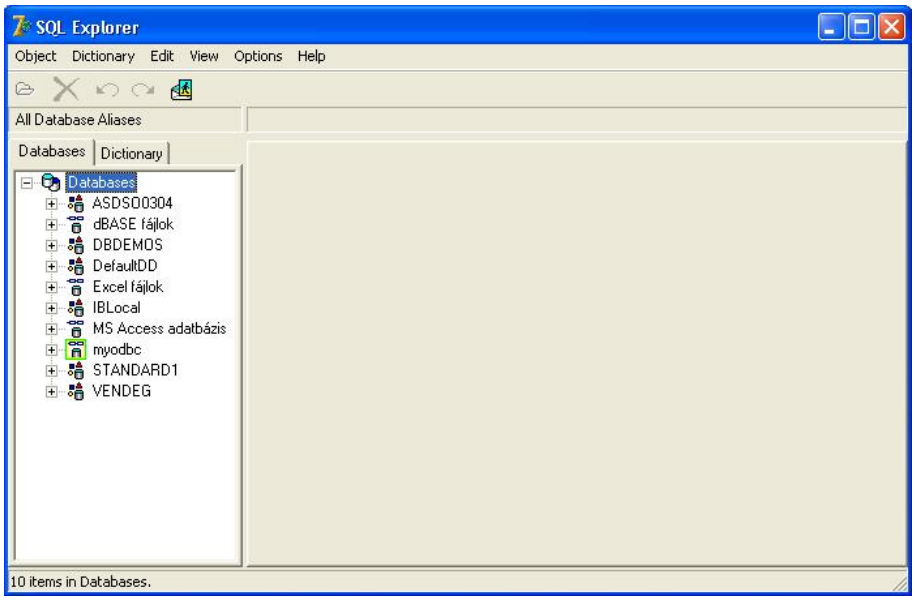
A Database Form Wizard lehetővé teszi, hogy egyszerű adatbázis-alkalmazásokat készítsünk, amelyek bemutatják a kétrétegű adatbázis minimálisan szükséges elemeit.

3.3.2. Az SQL explorer használata

BDE aliast hozhatunk létre a *Database, Explorer* menüvel a Delphiben. Ha a Delphi *Standard* verziója van meg, akkor ez a menüpont a Database Explorer-t nyitja meg. A Professional és Enterprise verzióknél az SQL Explorer kerül megnyitásra. Segítségével BDE-aliasokat hozhatunk létre.



3.1. ábra. *Database Form Wizard*



3.2. ábra. *SQL Explorer*

3.4. Adat-hozzáférési komponensek

Az adatelérési komponensek elég magasszintű absztrakciót tesznek lehetővé, az adatbázis elérést megkönnyítve. A *TDataSet* a *TComponent* közvetlen leszármazotja, ami bevezeti az adatbázisadat mint osztály fogalmát.

3.4.1. A TDataSet osztály

A *TDataSet* osztály bevezeti a lényeges tulajdonságokkal és metódusokkal rendelkező legkisebb közös többszöröst az adatbázishoz való kapcsolat megkönnyítéséhez, és a rekordok, illetve mezők kezeléséhez. Felsorolja azokat az eseményeket amelyeket a *TTable* és a *TQuery* örököl. A *TBDEDataSet* és *TDBDataSet* komponensek.

A *TBDEDataSet* a *TDataSet* következő leszármazotja. A *TBDEDataSet* komponens azért jött létre, hogy beillessze az eredeti **BDE** API működését a *TDataSet* működésébe.

A *TBDEDataSet* képes a frissítéseket a gyorsítótárba helyezni, ami lehetővé teszi, hogy alkalmazásaink egyszerre több rekord módosíthatóságát végezhessék el. Ez csökkenti a hálózat túlterheltségét. A *TDBDataSet* viszi be a *TBDEDataSet* komponensbe az adatbázis-kommunikációt. Azokat a komponenseket, amelyeket az adatérzékeny alkalmazások tervezésekor használunk, egyenesen származtatjuk a *TDBDataSet* komponensből. Mivel rendelkezésünkre áll az összes szülő osztályoktól örökölt tulajdonság és eljárás, programírás közben csak ezekre a komponensekre kell koncentrálnunk.

3.4.2. A TTable táblázatkomponens

A *TTable* komponens egy adatbázis-táblázatot reprezentál. Közvetlenül a *TDBDataSet* komponensből származik, az őseitől örökölt minden képességet, adatokra és eseményekre vonatkozó tulajdonságot: *TObject*, *TPersistent*, *TComponent*, *TDataSet*, *TBDEDataSet* és *TDBDataset*.

A *TTable* komponens egy láthatatlan komponens vagyis futáskor nincs látható megfelelője, amely egy fizikai adattáblázatot vagy adatkészletet képvisel egy adatbázison belül. A *TTable* komponens használatához mindössze csak annyit kell tennünk, hogy levesszük a komponenspaletta *Data Access* lapjáról és az *Object Inspectorban* megadjuk a *DatabaseName* és a *TableName* tulajdonságot.

- A *SessionName* és a *DatabaseName* tulajdonságok. Minden táblázatkomponensnek van *SessionName* és *DatabaseName* tulajdonsága. A *DatabaseName* vagy egy alias, amely az adatbázist reprezentálja, vagy egy *TDatabase* komponens neve. A globális *Session* objektumból egyetlen példány van, az ilyen objektumokat nevezzük egyedüli objektumoknak. Az egyedüli objektum olyan globális objektum, amelynek egyetlen példánya működik egyszerre. A *Session*ből persze a *TSession* osztály segítségével több példányt is létrehozhatunk.

  Megadhatjuk a *Session* komponens *SessionName* tulajdonságát, de inkább hagyjuk üresen, vagy a legördülő listából válasszuk ki a *Default* nevet. Ha nem használunk *Session* komponenst, akkor a program a *DBTables.pas* unit kezdeti beállítások részében keletkező globális *Session* objektumot fogja használni.

- A táblázatok paraméterei. A *TTable* komponens *MasterSource*, *MasterFields* és *FieldIndexNames* tulajdonságai lehetővé teszik *master/detail* kapcsolatok létesíté-

sét. A *Mastersource* tulajdonság egy adatforrás-komponenste (*DataSource*) hivatkozik. A *MasterFields* tulajdonság a mester-adatkészlet erre a táblázatra vonatkozó adatmezőit tartalmazza. A field *IndexNames* a mester-adatkészlet indexmezőire hivatkozik.

– Adatmezők. A *TDataSet* komponensnél megjelenik a *Fields* kollekció, amely az adatkészlet adatmezőit tartalmazza. Az adatmező ebben a szövegösszefüggésb enegyetlen adatsor, és a táblázat egy oszlopának a metszetét jelenti. A táblázat egyes adatmezőihez hozzáférhetünk a *FieldByName* metódus és a *Fields* kollekció segítségével is. Mind a *Fields* kollekció, mind a *FieldByName* metódus egy *TField* objektumot ad eredményül. Ha tervezési időben nem hozunk létre adatmező-komponenseket, akkor futáskor az adatkészlet-objektum létre hozza őket, mielőtt az adatkészlet megnyílna.

3.4.3. A TQuery komponens

A *TQuery* komponens a *TTable* komponens ikertestvére. Ahelyett, hogy volna egy *TableName* tulajdonsága, amely egy adatkészletet jelöl ki, ennek a komponensnek egy SQL-tulajdonsága van. Az SQL-tulajdonság *TStrings* típusú és egy érvényes SQL-utasítássorozatot tartalmaz. Az aktuális SQL-utasítások, amiket a szöveg-lista szerkesztőben írunk, függenek az éppen használt adatbázismotortól. A táblázat és lekérdezéskomponenseket egymást váltva is használhatjuk. Egy adatbázisban a *TQuery* és a *TTable* komponenssel is létrehozhatunk és törölhetünk, szerkeszthetünk egy adattáblát.

– A select SQL utasítás.

```
SELECT mezőnév1 [ , mezőnév2, mezőnévn | *] FROM táblázatnév
```

alakú.

A *SELECT* és a *FROM* az SQL nyelv kulcsszavai. A dőlt betűs mezőnév paraméterek az eredmény-adatkészletnek visszaadott mezőket jelölik. A "táblázatnév" jelöli azt a forrásadattáblát, amelyből az eredmény származik.

– *OPEN* vagy *EXECSQL* ? A lekérdezéskomponensnek van egy a táblázatkomponensnél nem szereplő utasítása, az *ExecSQL*. Amikor a lekérdezéskomponens egy *SELECT* SQL-utasítást tartalmaz, akkor benne az eredmény-adatkészlet található. A *SELECT* utasítások végrehajtásához használjuk az *Open* metódust. Az olyan esetekben, amikor az *INSERT*, *DELETE*, *UPDATE*, *CREATE TABLE* illetve a *DROP TABLE* utasításokat, vagy más nem-select utasítást használunk, amelyek nem adnak vissza eredmény-adatkészletet, használjuk az *ExecSQL*-t.

– A *PARAMS* tulajdonság.

A lekérdezéskomponens rendelkezik *Params* tulajdonsággal, amely az SQ-utasításban helyettesíthető paramétereket tartalmazza. Ezeket kitölthetjük az SQL-utasítás definiálása után.

Miután az SQL-utasítást a *Strings* szöveglista-szerkesztőben megadtuk, a kollekciószerkesztőben meg kell adnunk a *Param* tulajdonságot. A paramétereknek a lekérdezésobjektum *ParamByName* metódusával adhatunk értéket.

3.4.4. A TDataSource komponens

A *TDataSource* komponens köti össze az adatérzékeny vezérlőket az adatkészlettel. Minden adatérzékeny vezérlőnek van egy *DataSorce* tulajdonsága, de az adatkészletre nem tartalmaz közvetlen hivatkozást. Az adatkészlettel a kapcsolat a *DataSorce* adatforráson keresztül valósul meg. A *DataSet* tulajdonságon felül az adatforrásnak van még *AutoEdit*, *Enabled* és *State* tulajdonsága és *Create*, *Destroy*, *Edit* és *IsLinkedTo* metódusa.

Az adatforrásnak speciális eseménykezelő metódusai is vannak: *OnDataChange*, *OnStateChange* és *OnUpdateData*.

Egy alapvető adatérzékeny form számára szükséges az adatkészlet, az adatforráskomponensek és az adatérzékeny vezérlők, amelyek által a felhasználó látja és módosítani tudja az adatokat.

3.4.5. A TDatabase komponens

Az adatbázis-komponens a fizikai adatbázis alkalmazásbeli reprezentációja. Amikor az alkalmazásunk egy adatkészlet-komponenst hoz létre, akkor is létrejön egy példánya, ha *explicite* nem is deklaráltunk egy adatbázis-komponenst. Az adatbázis-komponens tartalmazza az *AliasName*, a *DatabaseName* és a *Connected* tulajdonságokat. Vannak olyan metódusai, amelyek kötegelt frissítést és tranzakció-feldolgozást hajtanak végre.

3.4.6. A TSession komponens

A *Session* komponens több adatbázis-kapcsolat kezelését teszi lehetővé egy alkalmazáson belül. A *Session* objektumokat használják szabványos adatbázis-alkalmazások, olyan alkalmazások, amelyek több, a hálózat különböző pontjain lévő táblázatot használnak és többszálú adatbázis-alkalmazások is.

3.5. Adatérzékeny vezérlők

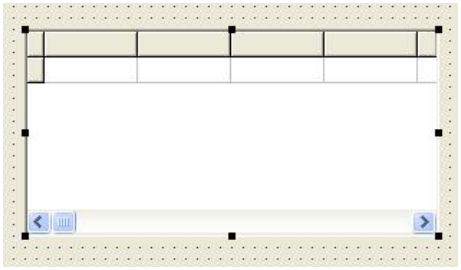
A komponenspaletta *Data Controls* lapján sok olyan vezérlő található, mint a paletta *Standard* lapján. Az adatérzékeny vezérlők az egyszerű szabványos vezérlőktől származnak, de tartalmaznak egy *TFieldDataLink* objektumot, amely által az adatforrással kapc-

solódnak össze. Az egyes vezérlők szükségleteitől függően egyéb kiegészítő tulajdonságok és eseménykezelők is adódnak a komponenshez.

3.5.1. DBGrid

A *TDBGrid* komponens valójában egy *TCustomGrid*. Adatsorokat és oszlopokat tartalmaz, minden sora az adatkészlet egy sorát reprezentálja. Egy relációs adatbázisnál egy sor több adattáblából is összeállhat.

A *DBGrid*-nél bevezetésre kerül az oszlopgyűjtemény és a tervezési fázisban használható oszlopszerkesztő, amellyel az adatbázis adatai megjeleneíthetők a táblában.



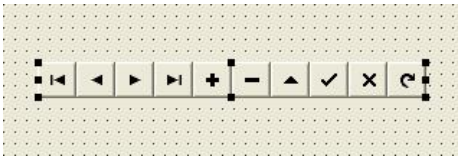
3.3. ábra. *DBGrid*

3.5.2. DBNavigator

A *DBNavigator* egy sor gombbal ellátott *TCustomPanel* komponens. Egy adatforráshoz kapcsolódik, és a gombok vizuális metaforái az adatkészlet egyes metódusainak.

A műveletek, amelyeket a gombok reprezentálnak balról jobbra: *First*, *Prior*, *Next*, *Last*, *Delete*, *Edit*, *Post*, *Cancel* és *Refresh*.

A navigátor nem maga valósítja meg ezeket a funkciókat, hanem rendelkezik egy *DataSource* tulajdonsággal. A *DBNavigator BtnClick* metódusa meghatározza, hogy a felhasználó melyik gombot nyomta meg, és meghívja a megfelelő metódust a *DataSource.Dataset* objektumra.



3.4. ábra. *DBNavigator*

3.5.3. DBText

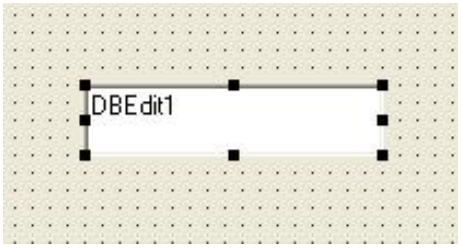
A *TDBText* a *TCustomLabel*-ből származtatott vezérlő, amely különösen hasznos az adatbázis csak olvasható részeinek megjelenítésére. A felhasználó a *TDBText* vezérlőben közvetlenül nem szerkesztheti a szöveget.

Ahhoz, hogy adatérzékeny vezérlőt adatmezőhöz kapcsoljunk, az adatforrást annak a *DataSource* tulajdonságához kell rendelnünk, a táblázat vagy lekérdezés adatmezőjének nevét pedig a *DataField* tulajdonságához. A lehetséges adatmezőnevek listája az adatforráshoz kapcsolt adatkészletből származik.

3.5. ábra. *DBText*

3.5.4. DBEdit

A *DBEdit* a *TCustomEdit* leszármaztatottja, általános célú adatérzékeny vezérlő, bármilyen szövegformában reprezentálható adat szerkeszthető vele. Az adatbázisok szöveges és numerikus adatmezői mind hozzárendelhetők.

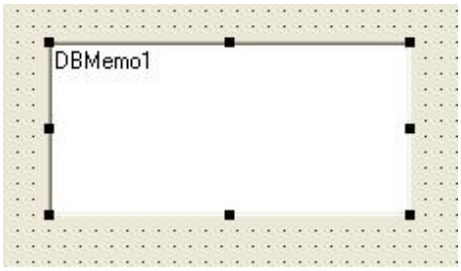


3.6. ábra. *DBEdit*

3.5.5. DBMemo

A *DBMemo* a *TCustomMemo* komponensből származik. A megfelelő adatmezőtípus, amely a *TDBMemo*-ban szerepelhet, a "memo" típusú mező. A megfelelő adattípusok, amelyeket a *TMemoField* típusként lehet használni, az egyes adatbázisok saját "memo" típusai, amelyek támogatják ezt a "memo" típusú vagy változó hosszúságú karakteres

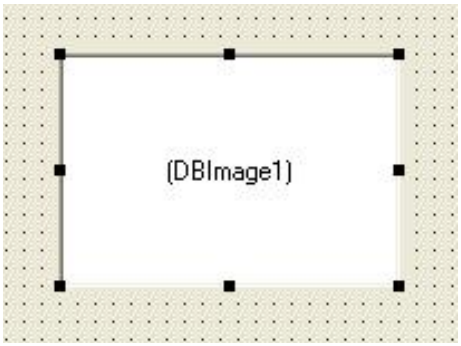
mezőt, ilyen például a **varchar** típus.



3.7. ábra. *DBMemo*

3.5.6. DBImage

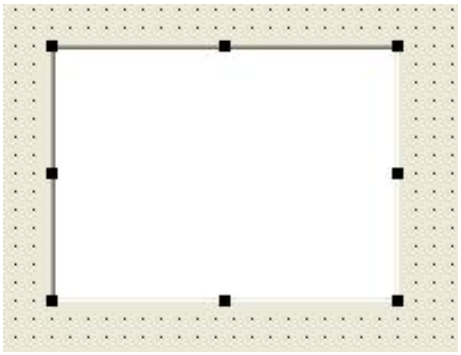
A *DBImage* vezérlő a *TCustomControl* komponenstől származik, és nem a *TImage*-től. Ez a vezérlő bármely olyan képtípust képes megjeleníteni, amely *TPicture* típusúnak sorolható be. A **Picture** típusú objektumok közé tartozik a "bitkép", az "ikon" és a "metafájl" grafika. A fájlok kiterjesztésük szerint lehetnek **.jpg**, **.jpeg**, **.bmp**, **.ico**.



3.8. ábra. *DBImage*

3.5.7. DBListBox

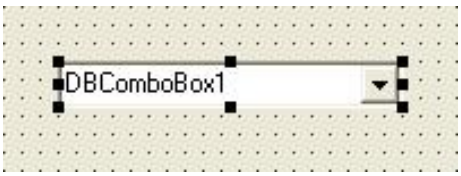
A *DBListBox* vezérlő az állandó értéklistából kiválasztott elemet a *DataField* tulajdonsága és az adatforrás által meghatározott adatmezőhöz rendeli. A *DBListBox*-ban felsorolt lehetőségeket vagy a tervezési fázisban vagy futási időben töltjük ki a vezérlő *TStrings* tipusú *Items* tulajdonságába. Ha dinamikus, táblázatból kikeresett értékekre van szükség a listában, akkor használjuk a *TDBLookupListBox* vezérlőt.



3.9. ábra. *DBListBox*

3.5.8. DBComboBox

A *DBComboBox* a *TCustomComboBox* komponens leszármaztatottja. A választási lehetőségek a *TString* típusú *Items* tulajdonságba kerülnek, az eredmény pedig a *DataSource* és a *DataField* tulajdonságok által meghatározott adatmezőbe kerülnek.



3.10. ábra. *DBComboBox*

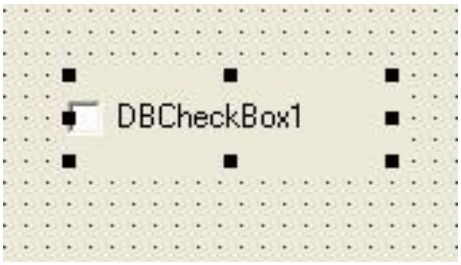
3.5.9. DBCheckBox

A *DBCheckBox* az igaz és a hamis választási lehetőséget adó legördülő lista alternatívája. Az igaz értéket a pipa jelenléte, a hamisat a hiánya jelenti. A *DataField* és *DataSource* tulajdonság mellett szerepel még két kiegészítő tulajdonság, a *ValueChecked* és a *ValueUnchecked*, amelyek mind a *DBCheckBox* által képviselt adatmező jelöltségét vagy jelöletlenségét fejezik ki.

3.5.10. DBLookupListBox

A *DBLookupListbox* komponens tulajdonságai a már jól ismert, a lefedett adatmezőre mutató két tulajdonság a *DataField* és a *DataSource*, és néhány új *ListSource*, a *ListField* és a *KeyField*.

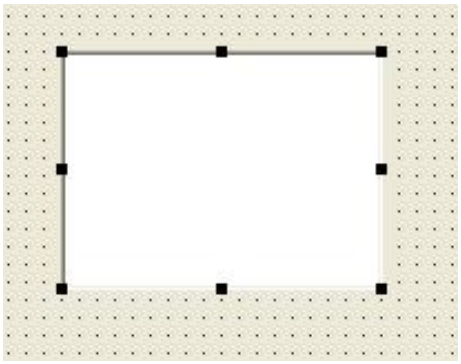
Ez a három tulajdonság azt mutatja meg, hogy a választási lehetőségek honnan származnak. A *ListSource* az adatforrás, ahonnan a lista adatai származnak. A *ListSource* és



3.11. ábra. *DBCheckBox*

a *DataSource* tulajdonságok nem mutathatnak ugyanarra az adatforrásra. Ez azt jelenti, hogy a *DBLookupListBox* használatához két adatkészletre és két adatforrásra van szükség.

A *ListField* a listaablakban megjelenő választási lehetőségeket adja meg, a *KeyField* pedig azt, hogy a lista adatkészlet melyik mezője kerüljön a *DataSorce* és a *DataField* által mutatott eredmény adatmezőbe.



3.12. ábra. *DBLookupListBox*

3.5.11. DBLookupComboBox

A *DBLookupComboBox* a *DBLookupListBox* komponens párja. Van *ListSource*, *ListField* és *KeyField* tulajdonsága, a legördülő listáját a *ListSource ListField* oszlopának értékei töltik ki.

Amikor egy listaelemet kiválasztunk, akkor az ehhez a sorhoz tartozó, a *KeyField* által meghatározott mező értéke íródik a *DataSource* által mutatott adatállomány *DataField* mezőjébe.

3.13. ábra. *DBLookupComboBox*

3.6. SQL programozás Delphiben

Az adatbázis-kezelés legáltalánosabb feladatai közé az adatok felvitele, eltávolítása és módosítása tartozik. Utasítások igen hatékony gyűjteményét állíthatjuk össze abból a néhány parancsból, mely ezen alapfeladatokat valósítja meg.

3.6.1. Select

A *SELECT* utasítást egy vagy több tábla adatainak visszakeresésére használjuk. A szabványos *SELECT* utasítás kanonikus alakja a következő:

```
SELECT mezőlista From táblanév.
```

A *SELECT* kulcsszóval kezdődik az utasítás. A mezőlista vesszővel elválasztott mezőnevek sorozata vagy pedig a csillag karakter (*), mely utóbbi jelentése minden. A *FROM* utasításelem után adjuk meg a mezőket tartalmazó tábla vagy táblák nevét.

A *SELECT* utasítással az a célunk, hogy olyan utasítást fogalmazzunk meg, amellyel egy vagy több táblából bármikor bizonyos sorokat vagy akár minden sort visszakapunk.

3.6.2. Delete

A *DELETE* utasítás rendkívül egyszerű.
A

```
DELETE FROM táblanév
```

utasítás kitörli az összes sort a táblanév megnevezésű táblából. Általában azonban adott sorok törlésére van szükség. Ekkor a *WHERE* feltétel járul az utasításhoz. Fontos, hogy a *DELETE* nem törli a fizikai táblát, csak a benne lévő sort, vagy sorokat.

3.6.3. Insert

Az *INSERT* utasítás már bonyolultabb. Egy táblába sorokat illeszthetünk be vele. Az *INSERT* utasítással értéket adhatunk minden egyes mezőnek, vagy csak bizonyos, az utasításban megadott mezőknek. E célból az *INSERT*-hez szükség van táblanévre, mezőlistára, és a mezőknek megfelelő típusú értékekre.

```
INSERT INTO "animals.dbf" VALUES
 ("Turtle", 7, 5, "Wetlands", NULL)
```

3.6.4. Update

Az *UPDATE* utasítással létező rekordokat módosíthatunk. A *WHERE* feltétel nélküli alaputasítás minden rekordot módosít. A legtöbb esetben valószínűleg szűkíteni szeretnénk a módosítandó rekordok körét, de minden *UPDATE* az alapformával indul, amely a következő.

```
UPDATE táblanév SET mező1 = érték1 [, mező2 = érték2, mezőn = értékn]
```

Az *UPDATE* utasítás hatékonysága abban rejlik, hogy módosíthatunk adott sorokat, de külső követelményeken alapuló feltételekhez is köthetjük a módosítás végrehajtását.

3.6.5. SQL és TQuery

A Delphi, SQL és egy adatbázisszerver között szükség van valamilyen kapcsolatmódra. Többféle komponens is rendelkezésünkre áll, hogy egy adatbázishoz kapcsolódjunk, de a *TQuery* komponenst kifejezetten a Delphi és egy adatbázis-szerver közötti SQL-kapcsolat megvalósítására tervezték.

A *TQuery* komponensszámos tulajdonságát kell beállítani ahhoz, hogy SQL utasítást küldjünk egy adatbázisszerver felé. A *TQuery.Database* tulajdonság a BDE vagy ODBC aliast jelöli, amely a fizikai adatbázis helyére utal.

A *DataSource* tulajdonság dinamikusan használható arra, hogy paraméterezett értéket adjunk a lekérdezésekhez.

A *Params* tulajdonsággal paramétereket határozunk meg a lekérdezésekhez, az *Sql* tulajdonság pedig az SQL-utasításszövegét tartalmazza.

4. fejezet

Webszerver készítése Delphiben

4.1. Webszerver készítése a WEB Brokerrel

Web Brokernek hívják a vizuális eszközök azon csoportját, amelyek segítségével webszerver-alkalmazások készíthetők. A Web Broker segítségével **ISAPI**, **NSAPI** vagy CGI protokolokat támogató webszerver készíthető. A fejlesztők szempontjából fontos, hogy a Web Broker megkönnyítit a webalkalmazások készítését a legnépszerűbb internetszerverek által használt protokollokra.

Minden webszerver tartalmaz egy *TWebModule* vagy egy *TDataModule* és egy *TWebDispatcher* komponenst.

4.2. HTML alapok

Az URL(*Uniform Resource Locator*) képviselte útvonalban benne kell lennie a webszerver nevének is. A Web Brokerrel létrehozott szerver az URL-kérés tartalmától és a webszerver felépítésétől függően ad vissza oldalakat. A válaszok formája általában HTML-dokumentum. A dokumentum tartalmazhat hiperhivatkozásokat más weboldalakra wagy webszolgáltatásokra.

Egy HTML-dokumentum általános felépítése:

```
<html>
 <head>
  <title>
  </title>
 </head>
 <body>
 </body>
</html>
```

4.2.1. A cím használata

A *<title> </title>* cím tag-pár segítségével egy címet adhatunk minden oldalnak. A cím tag általában a *<body>* tag elé a *<html>* nyitó tag után kerül.

4.2.2. Hivatkozások hozzáadása

Az * display text * tag egy hiperhivatkozást épít a HTML-dokumentum szövegébe. Ha a felhasználó rákattint az aktuális hiperhivatkozásra, akkor átirányítódik a megadott URL-hez.

4.2.3. Képek beillesztése

A weboldalakra sokféle grafika beszerkeszthető, használhatók **GIF**, **JPEG**, és **BMP** formátumú képek is. A grafikák beszerkesztése az *<img src="grafikus fájl útvonal">* tag használható. A képméret korlátozható, megadhatunk vízszintes és függőleges méreteket is.

4.2.4. Táblázatok definiálása

A *<TABLE>* tag lehetővé teszi, hogy az oldal egész kiterjedését különálló szervezet részekké osszuk fel. A *<table> </table>* tag-ok jelzik a táblázat határait. A *<tr></tr>* tag-ok egy sort határoznak meg, míg a *<td></td>* tag-ok egy soron belüli oszlopokat.

4.3. WEB Broker komponensek használata

4.3.1. Helyettesíthető paraméter TAG-ok

A profi weboldalak készítése lényegesen több erőfeszítést és a HTML lehetőségeinek szélesebb körű kihasználását kívánja. Egy dinamikus vagy változó weboldalnál az alapvető weboldal elkészítésén felül szükség lesz még a helyettesíthető tag-ok kijelölésére is. Ezekre a helyekre teszi a webszerver a változó adatokat. A helyettesíthető paraméterek a könyvjelzőkhöz hasonlóak.

A helyettesíthető paraméterek a Delphi webszerver oldalakon a

```
<#tagnév>
```

jelölést használják.

A HTML egyedülálló tulajdonsága, hogy a *<>* jelek közé írt szöveget, ha az nem valamely érvényes tag nevét tartalmazza, egyszerűen figyelmen kívül hagyja. A Delphi-komponensek számára ilyen helyeken speciális kódok helyezhetők el.

4.3.2. A WebDispatcher komponens

Minden webszerver-alkalmazás tartalmaz egy *WebDispatcher* komponenst, amelyet vagy a *TWebModul* tartalmaz, amikor egy új projektet indítunk a webszerver-alkalmazás varázslóból vagy egy létező adatmodul komponenshez adjuk hozzá a komponenspaletta **Internet** füléről.

A *TWebDispatcher* komponens a *TWebActionItems* objektumok listáját tartalmazza. Ezek teszik lehetővé, hogy az **URL** válaszok és kérések számára útvonalinformációt definiáljunk.

Ha a webszerver-alkalmazás varázslót használjuk akkor a **Delphi** automatikusan létrehoz egy webmodult, amelyben benne van a *WebDispatcher* illesztő. Egy webszerverhez csak egy *WebDispatcher*re van szükség, tehát vagy adjunk egy adatmodulhoz egy *TWebDispatcher* komponenst, vagy hozzunk létre egy *WebModule* komponest.

4.3.3. A WebModule komponens

A *WebModule* a *DataModule* osztályból származtatott komponens, amely lehetővé teszi a webszerver **Web Broker** csomaggal való készítéséhez szükséges *WebActionItems* objektumok és egyéb nem látható vezérlők kezelését. A webmodulhoz hozzáadunk még más **VCL**-vezérlőt is, például az oldalkészítő komponenst. Az oldalkészítő az **URL**-útvonal információja és lekérdezési adatai által megfogalmazott kérésre válaszol.

4.3.4. A WebActionItem komponens

A webakció-szerkesztő lehetővé teszi a webszerver részére útvonal-információk definiálását. Amikor a webszerver útvonal és lekérdezésinformációt kap a http-szervertől, akkor megegyező útvonalat és metódustípust keres az akcólistában. Ha ilyet nem talál, akkor veszi az alapértelmezett akciót. Ha viszont talált megfelelő akciót, akkor meghívja az *OnAction* eseményt. Az *OnAction* eseménykezelő a hívó objektumra kap hivatkozást, kap egy *TWebRequest*, egy *TWebResponse* objektumot és egy logikai változó paramétert.

A *Request WebRequest* típusú objektum tartalmaz minden információt, amit a http-szerver a kérésben elküldött. A *Response WebResponse* típusú objektum használható az oldal tartalmának visszaküldésére a szerverhez. Itt a *Content* tulajdonság használható egy egyszerű szöveg visszaküldésére.

4.3.5. A PageProducer komponens

A *PageProducer* komponensben megadható egy HTML-dokumentumsablon, amelyet a *HTMLDoc* nevű *TStrings* típusú tulajdonság tartalmaz, vagy a *HTMLFile* tulajdonsággal lehet rá hivatkozni. A *HTMLFile* és a *HTMLDoc* tulajdonságok egymást kizárják.

Ha az oldalkészítő sablonja tartalmaz a HTML számára átlátszó tag-okat, amelyeknél valamilyen adatokat kell behelyettesíteni, akkor létre kell hoznunk egy *TPagePro-*

ducer.OnHTMLTag eseménykezelőt, ami majd ezeket behelyettesítéseket dinamikusan elvégzi. Ha az oldalkészítő egy teljes HTML-oldalt tartalmaz, akkor nincs szükség az *OnHTMLTag* eseménykezelőre.

4.3.6. A DataSetPageProducer komponens

A *DataSetPageProducer* lehetővé teszi, hogy egy *DataSet* komponenst rendeljünk hozzá. Amikor az adatkészlet oldalkészítő tartalmát kérik, akkor az oldalkészítő végignézi a HTML-nek átlátszó tagokat és ahol a megfelelő adatmezőnevekkel találkozik, annak a tagnak a helyére behelyettesíti az adatmező aktuális értékét az adatkészletből.

Ha egyszerre több adatsort is szeretnénk megjeleníteni, ennek kényelmes módja, ha alkalmazzuk a *DataSetTableProducer* komponest.

4.3.7. A QueryTableProducer komponens

Az adatkészlet táblázatkészítőt a több adatsort is szemléltető oldalak készítésére találták ki. Lehet vele táblázatnézetet definiálni az adatokhoz és több adatsort is megjelenít, ha megfelelő kérés érkezik. A megjelenített sorok aktuális számát a táblázatkészítő *MaxRows* tulajdonsága adja meg. Az adatkészlet táblázatkészítő sokféle tulajdonsággal rendelkezik, amelyekkel az oldal külalakja vagy tartalma szabályozható, lehet fejlécet, láblécet és a cím tartalmát megadni vagy attribútumokat rendelni a sorokhoz.

5. fejezet

A program célja és funkciói

A szakdolgozathoz mellékelt program egy kisebb internet kávéző (pc-barlang) alapvető nyilvántartási feladatait hivatott ellátni. A program nem törekszik egy mindent átfogó, maximális igény kielégítésére. Munkahelyemen, ahol már évek óta dolgozom, elérkezett az az idő, hogy a vendégek (felhasználók) napi nyilvántartása, a bejelentkezések rögzítése és az esetleges ingyenjátékok vezetése ne papíron történjen, hanem egy program segítségével.

A felhasználók száma az utóbbi időben minimálisra csökkent. Szinte minden második háztartásban bekötésre került valamilyen internetkapcsolat. Ennek ellenére még mindíg van létjogosultsága az ilyen helyeknek. Azok akik nem rendelkeznek internet eléréssel, és azok is szívesen látogatják akik együtt, lokális hálózaton szeretnének egymás ellen játszani.

A program nyilvántartást vezet azokrók akik, ha csak egyszer is, de igénybe vették a klub szolgáltatásait. Azonkívűl segítséget nyújt a helyfoglalások rögzítésében is. További lehetőség az ingyenjátékkal rendelkező felhasználók igyenjátékainak, esetleges tartozásainak ellenörzése is. A program neve *Webes Alakalmazás-fejlesztés Borland Delphiben*, így nyilvánvaló, hogy webes felülettel is rendelkezik. Ennek a felületnek a jelentősége abban áll, hogy akár otthonról is betekintést nyerhetnek a felhasználók a klub napi történéseibe, ezen kívűl mindenki információt kaphat az ingyenjátékainak számáról is.

Természetesen egy ilyen nyilvántartó program nem lehet meg adatbázis-kezelés nélkül. Így a program legnagyobb része adatbázkezelési-feladatok ellátásáról és különböző formok elkészítéséből áll. A programnak ez az 1. verziója, így bizonyos dolgok még valószínűleg javításra, esetleg megváltoztatásra kerülnek majd, ha már használatban lesz. Minden esetre megpróbáltam a Delphi 5 adta lehetőségeket kihasználni mind az adatbázis, mind a weboldalak elkészítésekor.

6. fejezet

A program megvalósítása

6.1. Adatbázis elkészítése

Az adatbázis tervezésekor az egyik legfontosabb feladat az adatbázis-kezelő rendszer megválasztása volt. Több szempontot kellett figyelmbe vennem. Mivel a program saját részre készült így nem kellett figyelnem arra, hogy a felhasználó rendelkezik-e a kívánt adatbázis-kezelő rendszerrel. Csak azokat a szempontokat kellett vizsgálnom amelyek nekem előnyösek. Így a döntésem, az ingyenes MYSQL adatbázis-kezelő rendszer lett. Egyrészről már használtam ezt a rendszert PHP oldalak elkészítésekor, másrészről az elkészített adatbázis így újra felhasználható, ellenben a Delphi rendszerbe integrált, eddig használatos, Paradox rendszerrel szemben.

A Delphi és a MYSQL közötti kapcsolat létrehozásához szügség volt egy ODBC connector-ra, ezt a feladatot a MYSQL 3.51 -es verziójú driver-e látta el. A BDE Administrator segítségével egy "myodbc" aliast rendeltem a kapcsolathoz, így a felhasználónév és a jelszó megadása után már lehetőség nyílt az adatok elérésére.

A táblák elkészítéséhez a MYSQL Administrator 1.2.9. verzióját használtam. A grafikus felületű "Administratorral" gyerekjáték a táblák és a hozzájuk tartozó kapcsolatok elkészítése. Ezen kívül lehetőségünk van a táblák exportálására ill. importálására is. Az adatbázis neve "joystick", a szügséges táblák elkészítését az alábbi pontokban ismertetem.

6.1.1. A FELHASZNÁLÓ tábla

- A "felhasznalo" táblában a klubban lévő felhasználókat modeleztem le. A tábla "fazonosito" kulcs mezője azonosítja egyértelműen a felhasználókat. Megszorítása *NOT NULL*, és *AUTO INCREMENT*.

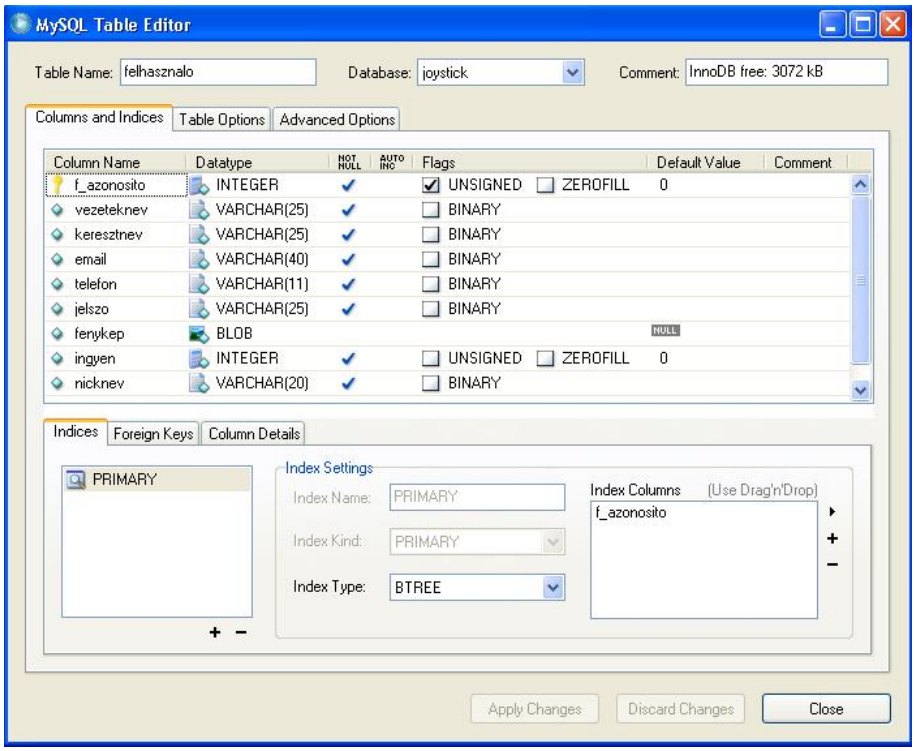
- A "vezeteknev", "keresztnev" és a "nicknev" oszlopok a felhasználok neveit jelentik. Megszorítása *NOT NULL*.

– Az "email" adja meg, hogy az adott felhasználó milyen e-mail címen érhető el. Megszorítása *NOT NULL*.

– A "jelszo" oszlop egy előre generált jelszó a belépéshez szügséges. A webes felhasználói felületen bármikor megváltoztatható. Megszorítása *NOT NULL*.

– A "telefon oszlop" a felhaszáló telefonos elérhetőségét tartalmazza. Megszorítása *NOT NULL*.

– Az "ingyen" oszlop a felhasználó ingyenjátékainak a számát tartalmazza. Megszorítása *NOT NULL*, *DEFAULT VALUE* = 0;



6.1. ábra. *Felhasználó tábla*

6.1.2. A JÁTÉK tábla

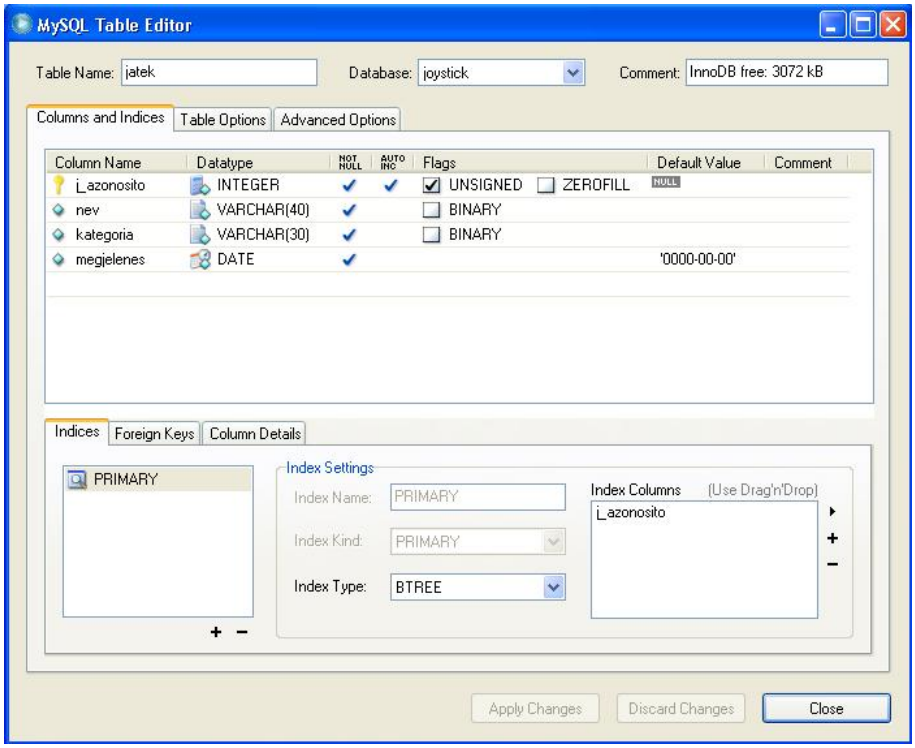
A "jatek" táblában a klub számítógépein játszható pc-játékok adatai szerepelnek. A táblának tájékoztató szerepe van.

– A "jazonosito" a tábla kulcs mezője, egyértelműen beazonosítja az adott recordot. Megszorítása *NOT NULL*, *AUTO INCREMENT*.

– A "nev" oszlop a játék hivatalos nevét tartalmazza.

– A "kategoria" megadja, hogy a játék milyen kategóriába sorolható. *NOT NULL.*

– A "megjelenes" egy dátum típusú mező amely a játék hivatalos megjelenési dátumát tartalmazza. *NOT NULL*, *DEFAULT VALUE* = 0000-00-00.



6.2. ábra. *Játék tábla*

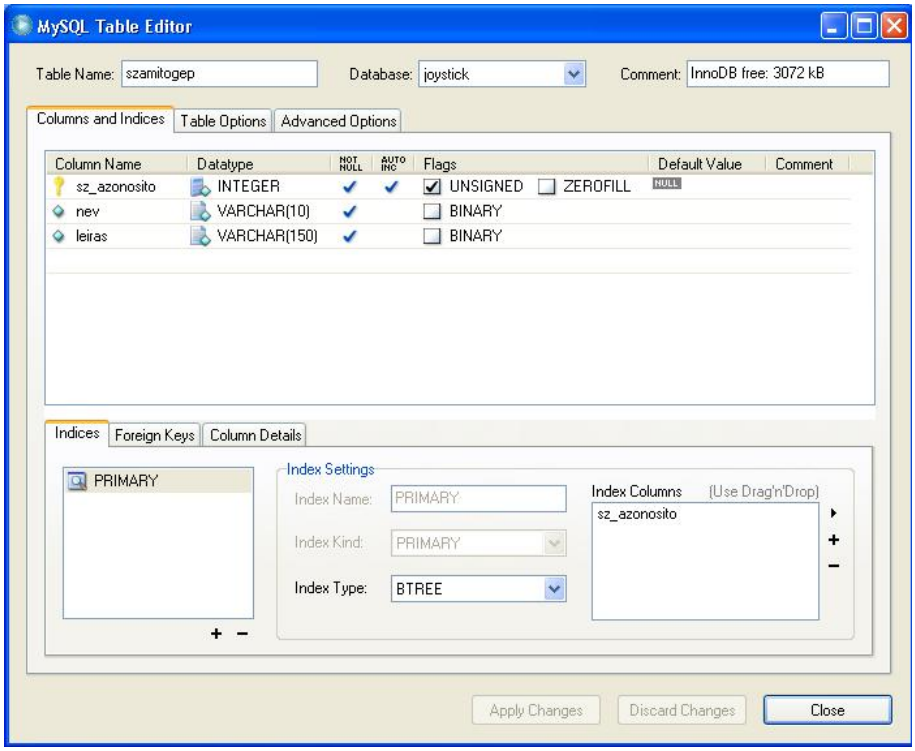
6.1.3. A SZÁMÍTÓGÉP tábla

A "szamitogep" tábla a klub számítógépeinek hardver adatait tartalmazza. E táblából megtudhatjuk, hogy hány és milyen konfigurációjú számítógépek találhatók a játékteremben.

– A tábla egy sorát az "szazonosito" mezővel lehet egyértelműen azonosítani. *NOT NULL*, *AUTO INCREMENT.*

– A "nev" mező a számítógép jelképes nevét tartalmazza. *NOT NULL.*

– A "leiras" mező az adott számítógép adatait tartalmazza. *NOT NULL.*



6.3. ábra. *Számítógép tábla*

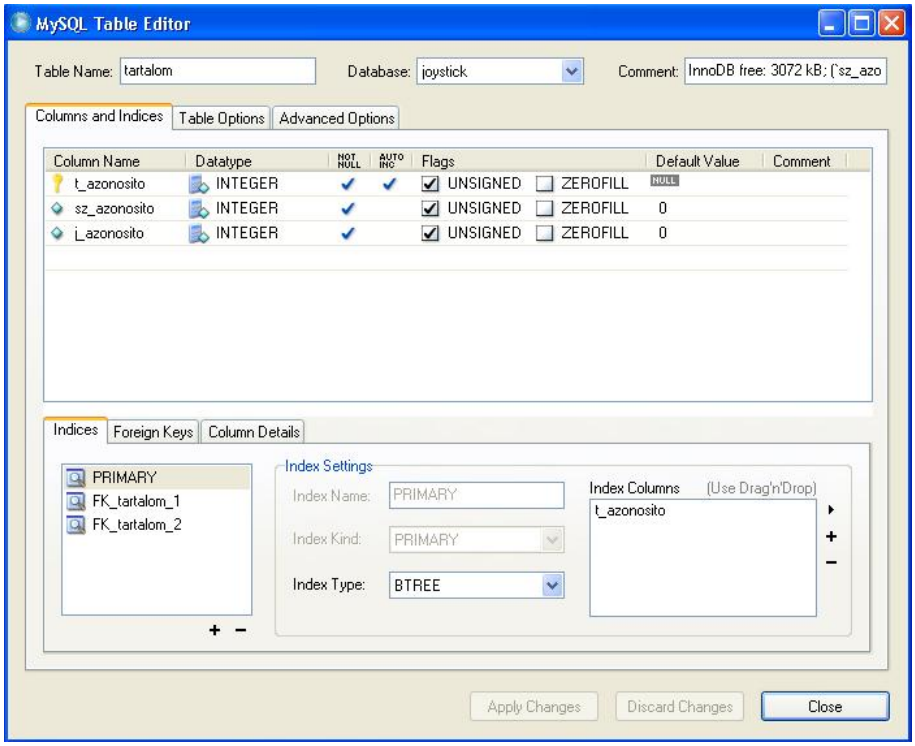
6.1.4. A TARTALOM tábla

A "tartalom" tábla információt nyújt, hogy melyik gépen milyen játék található. Szügségességét az adja, hogy a relációs adatbázis-kezelő rendszerek nem használnak többértékű tulajdonságokat. Így mivel, hogy egy gépen több játék is telepítésre kerülhet szügség volt egy kapcsoló tábla létrehozása. Az adatbázis normalizálása indokolja a tábla létrehozását. A tábla több-több számosságú kapcsolatot reprezentál.

- A "tazonosito" egyértelműen beazonosít egy kapcsolatot. *NOT NULL*, *AUTO INCREMENT*.
- Az "szazonosito" mezőnek külső kulcs szerepe van megadja, hogy melyik számítógép szerepel a kapcsolatban. *NOT NULL*.
- A "jazonosito" szintén külső kulcs, a játék tábla egy adott sorát azonosítja. *NOT NULL*.

6.1.5. A GÉPHASZNÁLAT tábla

A "gephasznalat" tábla segítségével lehet nyilvántartani, hogy melyik gépen, ki, mikor, mennyi időre veszi igénybe akár játék, vagy akár böngészés céljából.

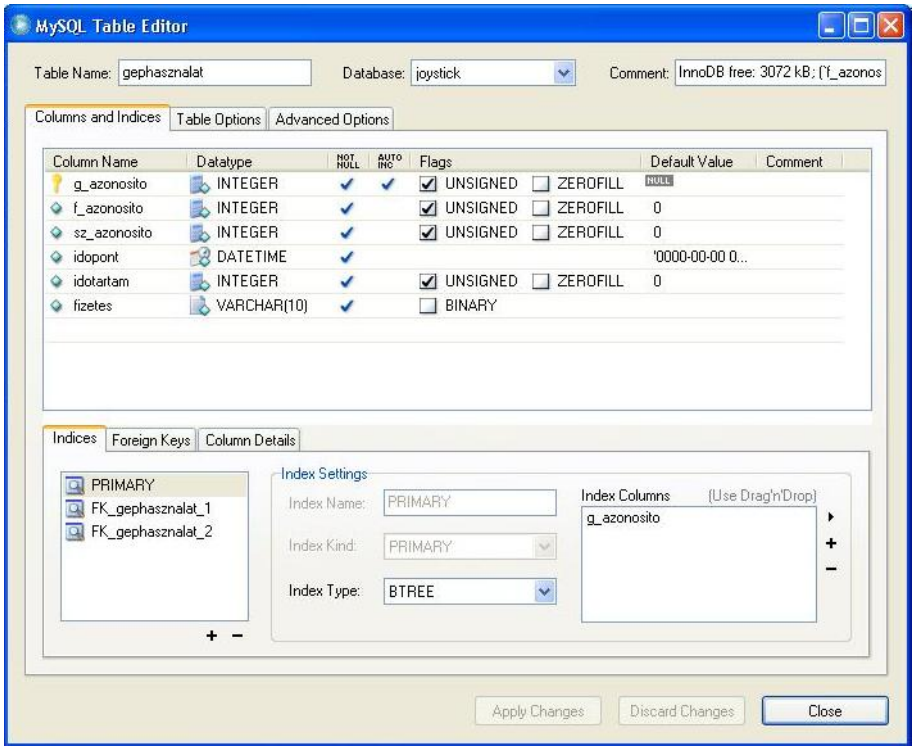


6.4. ábra. *Tartalom tábla*

- A táblát egyértelműen a "gazonosito" mező azonosítja. *NOT NULL*, *AUTO INCREMENT*.
- Az "fazonosito" mező azonosítja az adott felhasználót, ez a külső kulcs. *NOT NULL*.
- A "gazonosito" adja meg az adott számítógép azonosítóját, külső kulcs. *NOT NULL*.
- Az "idopont" mező tárolja a kezdés pontos idejét. *NOT NULL*.
- Az "idotartam" oszlop megadja a géphasználat időtartamát. *NOT NULL*.
- A "fizetes" mező tárolja, hogy a géphasználat készpénzzel, ingyennel, vagy esetleg hitelből van fedezve. *NOT NULL*.

6.1.6. A VENDÉGKÖNV tábla

A vendégkönyv tábla csak webes felületről érhető el. Az oldalt meglátogató felhasználók megjegyzéseket, különböző észrevételeket jegyezhetnek be a vendégkönyvbe.



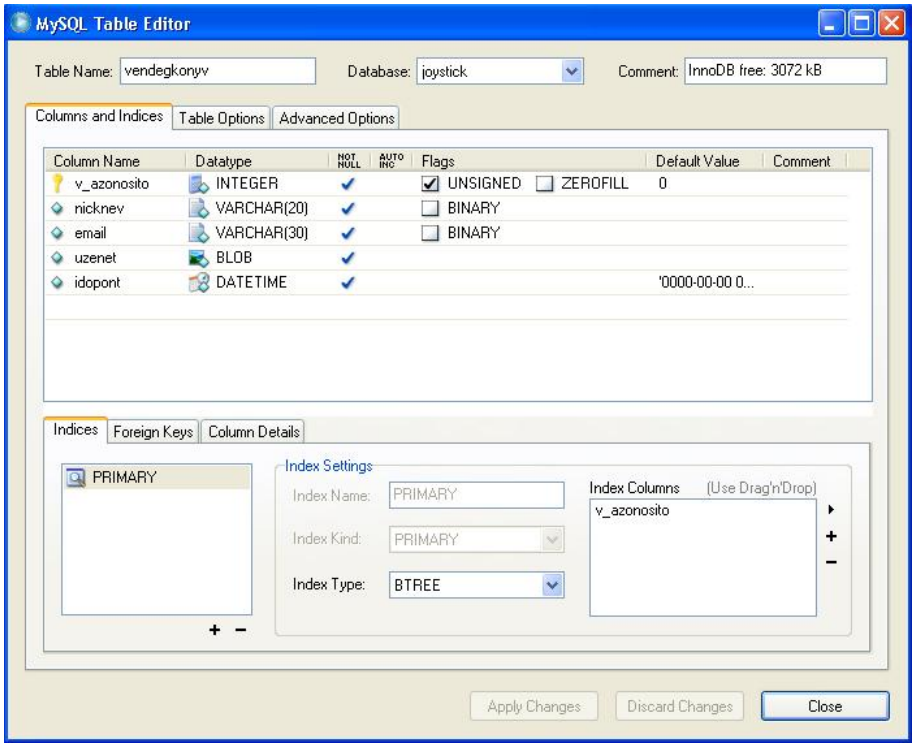
6.5. ábra. *Géphasználat tábla*

- A "vendegkonyv" tábla sorait egyértelműen a "vazonosito" mező szolgáltatja. *NOT NULL*, *AUTO INCREMENT*.
- A "fazonosito" adja meg, hogy ki az üzenet küldője, külső kulcs. *NOT NULL*.
- Az "email" mező az üzenet küldőjének az e-mail címe. *NOT NULL*.
- Az "uzenet" mező az üzenet tartalmát jelenti. *NOT NULL*.
- Az "idopont" az üzenet elküldésének időpontja. *NOT NULL*.

6.2. A TMYGEP saját osztály

A program futása folyamán az aktuális nap gáphasználatának adatait a programnak a memóriában kell tartania. Ennek hatékony megvalósítására hoztam létre a *TMyGep* saját osztályt. Ezt az oszályt a *MyUnit* nevű unit tartalmazza.

Az osztály, egy példány egy adott számítógép egy adott napjának a helyfoglalásait tartalmazza. Tehát a program indulásakor mindíg annyi pédány jön létre ahány számítógép van. A program elindítása történjen bármikor, mindíg az adott nap bejegyzéseivel töltödnek fel a példányok.



6.6. ábra. *Vendégkönyv tábla*

6.2.1. Mezők

- *Idopontok* : array[0..30] of String; Az "Idopontok" tömb string elemi reggel 09.00-tól kezve, fél órával mindíg növelve a kezdési időpontokat jelentik, teljesen záróráig.
- *GepNev* : String; A "Gepnev" a példány számítógép nevét tartalmazza.
- *NapHossz* : Integer; A "NapHossz" megadja a nyitvatartás hosszát.

6.2.2. Metódusok

- *constructor TMyGep.Create(Nev: String; Hossz: Integer);* Az osztály felülbírált konstruktora. Az alapértelmezettől eltérő konstruktorra van szügség, mert a példányt seciális módon kell inicializálni. Minden példányt felkell tölteni az aznapi adatbázis adatokkal. Így bármikor, bármennyiszer inditjuk a programot mindíg az aznapi adatok töltődnek be. A program futása során végig ezeket a példányokat használja ellenörzés céljából. Ha a tábla adatai módosulnak akkor módosul a példány állapota is.
- *function TMyGep.GetNev: String;* Ez a metódus visszadja a példány számítógép nevét.

- *procedure TMyGep.SetNev(Nev: String);* Ez a metódus beállítja a példány számítógép nevét.
- *function TMyGep.Get(Naphossz : Integer);* Ez a metódus visszaadja a nyitvatartás hosszát.
- *procedure TMyGep.SetNapHossz(Hossz: Integer);* Ez a metódus beállítja a nyitvatartás hosszát.
- *function TMyGep.GetThisTime(X: Integer): String;* Ez a metódus visszaadja az "Idopontok" tömb x-edik elemét.
- *procedure TMyGep.SetThisTime(Time: String; X: Integer);* Ez a metódus beállítja az "Idopontok" tomb x-edik elemét a "Time" értékre.
- *function TMyGep.GetHely(Time: String) : Integer;* A program futása során szügség van arra, hogy tudjuk, hogy egy adott időpont után hány félóra szad hely van. Ez a metódus ezt az értéket számolja ki.
- *procedure TMyGep.SetHely(Time: String; X: Integer);* Amikor valamelyk felhasználó helyet foglal bizonyos számú félórára, akkor ennek megfelelően nullázni kell az "Idopontok" tömb adott számú elemeit.

6.3. A megjelenítés formjai

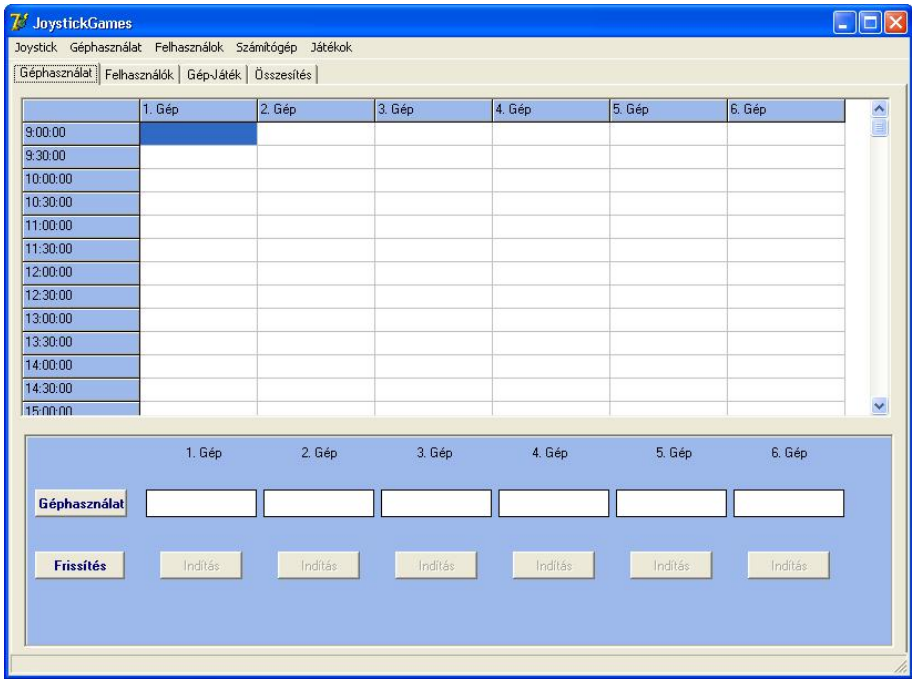
A megjelenésnek, megjelenítésnek óriási szerepe van egy programban. A felhasználónak szügsége van arra, hogy egy kényelmes felületen keresztül végezze az adatok bevitelét ill. feldolgozását. Mivel a program saját részre készül így olyan színeket választottam amiket én szeretek.

Az elemek elrendezésében, a nyomógombok elhelyezésében próbáltam egy logikus irányt követni, hogy minden kéznél legyen és ne kelljen semmit sem keresgélni. Az adatmegjelenítési formok minden olyan információt tartalmaznak amit a nevük sugalhat, így nem keverednek az adatok a logikailag más formra tartozó adatokkal.

6.3.1. Géphasználat

A "Géphasználat" form információt szolgáltat a klub egy adott napi helyfoglalásairól. Tartalmaz egy táblázatot amit egy általam készített algoritmus tölt fel. A lényege, hogy az adatbázisba bevitt helyfoglalások abban az oszlopban íródjanak ki amelyik gépre a helyfoglalás történt. A felhasználó neve abban a sorban kezdődik amikor elkezdi a szolgáltatás igénybevételét és annyi oszlopon keresztül ismétlődik ahány félórát eltölt a klubban. A formon ezenkívűl, a form alján található még egy olyan panel amelyen *TGauge* típusú

elemek és indító gombok találhatók. Szerepük, hogy a *Gauge* objektum vizuálisan is mutatja az eddig eltelt időt, és gomb segítségével lehet indítani a vizuális számlálást. A panel baloldalán található egy "Új Helyfoglalás" címkéjű gomb melynek segítségével lehet új helyfoglalást rögzíteni.



6.7. ábra. *Géphasználat*

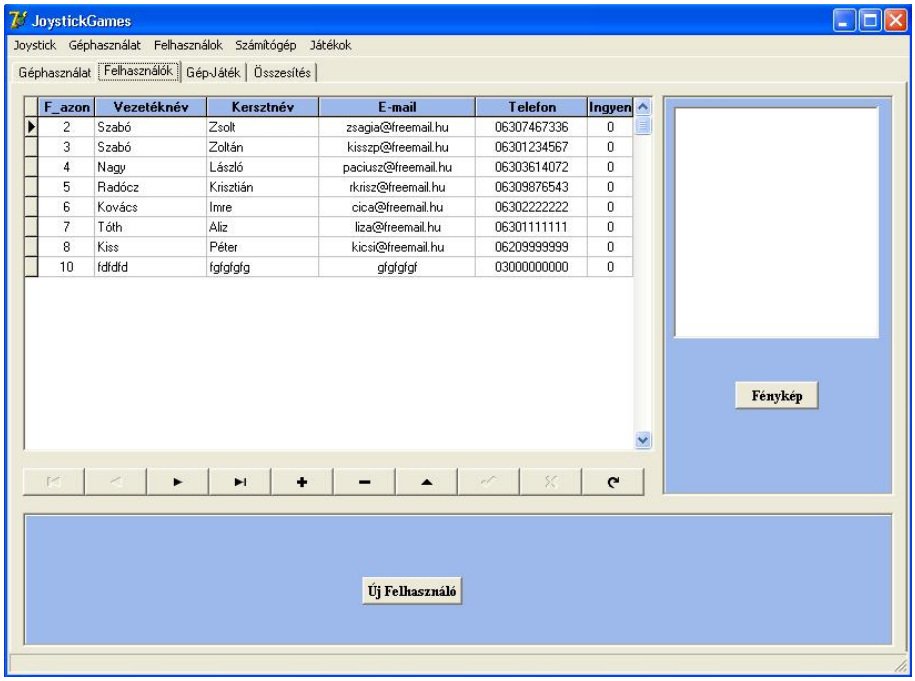
6.3.2. Felhasználók

A "Felhasználók" formon egy táblázat található, amely megjeleníti a klubban nyilvántartott személyek fontosabb adatait. A táblázat egy adatbázis-táblából veszi az elemeket, így különösebb kirajzolási algoritmus nélkül jeleníti meg az adatait.

Egy *DBNavigator* komponens is tartozik a formhoz amelnek segítségével lehet a táblában lépkedni, elemet törölni, illetve a táblázat tartalmát frissíteni. A táblázat alján helyezkedik el egy "Új Felhasználó" nyomógomb, amely az új felhasználók hozzáadásában segédkezik.

6.3.3. Gép-Játék

A "Gép-Játék" form egy összetett form. Ezen a formon információ található a klubban lévő számítógépekről, a megtalálható játékokról, és egy táblázat árulkodik arról, hogy melyik gépen melyik játék került telepítésre.



6.8. ábra. *Felhasználók*

A gép és a játék táblázatokhoz egyaránt tartozik egy-egy navigátor objektum. Segítségével lehet módosítani a táblázatok adatait.

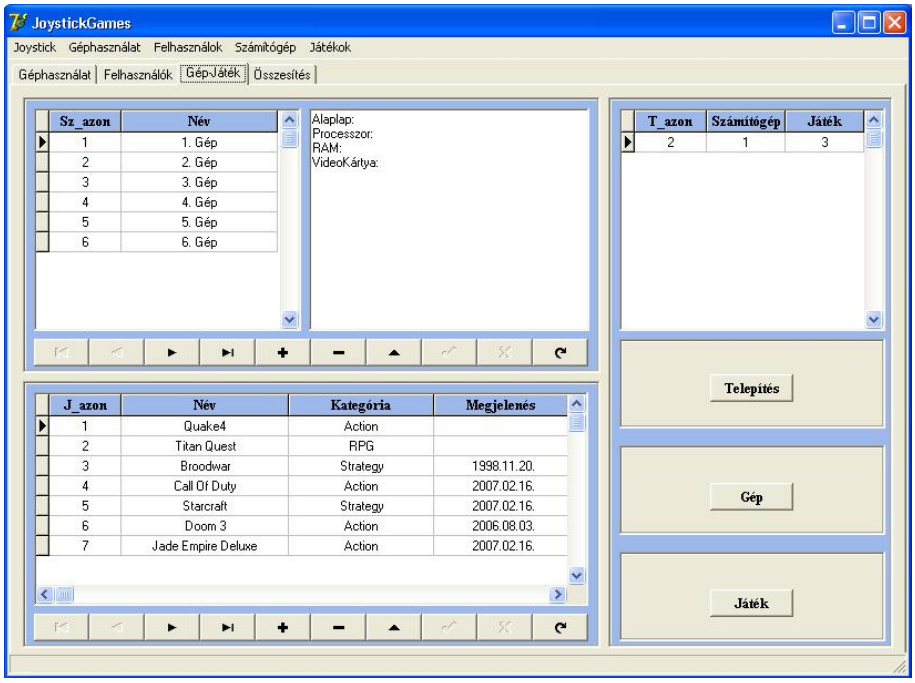
A form jobb oldalán található 3 nyomógomb, amelyeknek segítségével adhatunk az adatbázishoz új számítógépet, új játékot és a meglévő játékokat telepíthetjük bármely gépre.

6.3.4. Összesítés

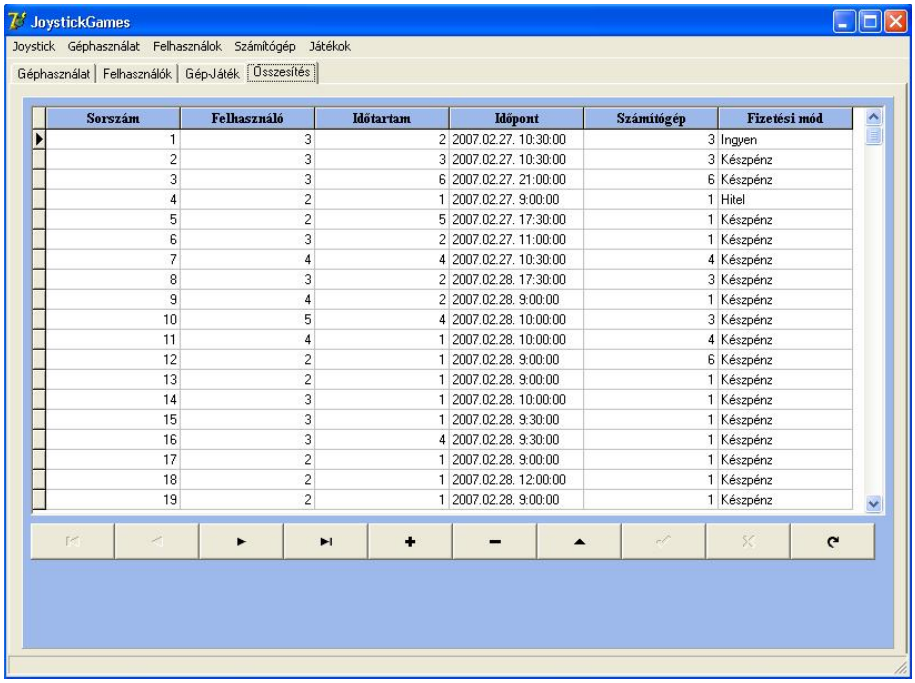
Az "Összesítés" form az eddigi formok legegyszerübb változata. Segítségével egy listát kaphatunk a klubban az eddigi helyfoglalásokról. Ebben a táblázatban megjelenik a felhasználó azonosítója, hogy melyik gépen, mennyit, és mikor játszott. Szerepe inkább csak formai, de komoly statisztikai nyilvántartások készíthetők belőle.

6.4. Formok az adatok beviteléhez

Itt azok a formok kerülnek bemutatásra, amelyek az adattáblák feltöltésében segítkeznek. Törekedtem arra, hogy adatbevitel után minél kevesebb ellenörzést keljen végeznem. Ezt úgy értem el, hogy azoknál a beviteli komponenseknél amelyeknél lehetséges és van értelme előre feltöltött adatokból lehet kiválasztani a megfelelőt. Természetesen ezt a megoldás nem lehet minden komponensnél alkalamazni, mert a beviteli mezők csak egy részénél alkalmazható ez a technika.



6.9. ábra. *Gép - Játék*



6.10. ábra. *Összesítés*

6.4.1. Új Géphasználat

Az "Új Géphasználat" form segítségével rögzítem a helyfoglalásokat. Ezt a formot úgy alakítottam ki, hogy egyáltalán ne lehessen helytelen adatokat bevinni. Ehhez a technikához *ComboBox* és *DBLookupComboBox* komponenseket használtam. Ezeknek a komponenseknek a feltöltése dinamikusan töténik.

Az első kiválasztandó adat a felhasználó neve, itt csak azok kerülhetnek kiválasztásra akik már rögzítve vannak az adatbázisban. Ha olyan személy szeretne helyet foglalni aki még nem regisztrált, akkor először az "Új Felhasználó" form segítségével regisztrálni kell.

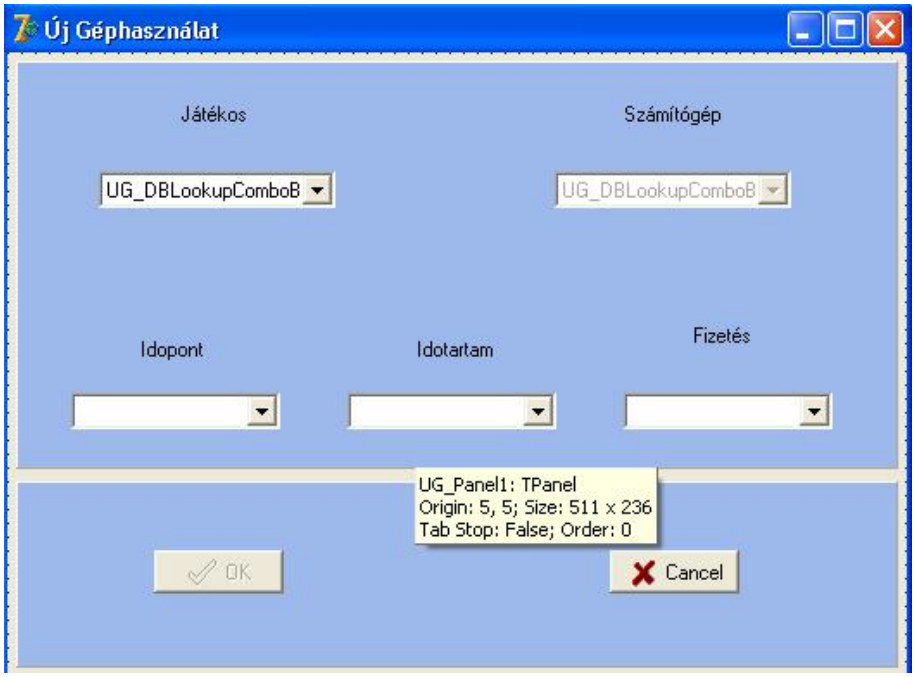
A második kiválasztandó adat a gép neve, amin a tevékenységet el akarja végezni.

Ezután kell megadni a kezdési időpontot. Ebben a *ComboBox*-ban csak azok az időpontok jelennek meg amelyek még szabadak.

A negyedik pont az időtartam amely a lehetséges, a gép szabad-hely kapacitásától függ.

Végezetűl a fizetési mód következik, ennél a beviteli komponensnél a választás lehet : készpénz, ingyen és hitel. Ha a felhasználó nem rendelkezik elegendő ingyenjátékkal akkor egy figyelmeztető üzenetet kap és javítania kell az adatokon.

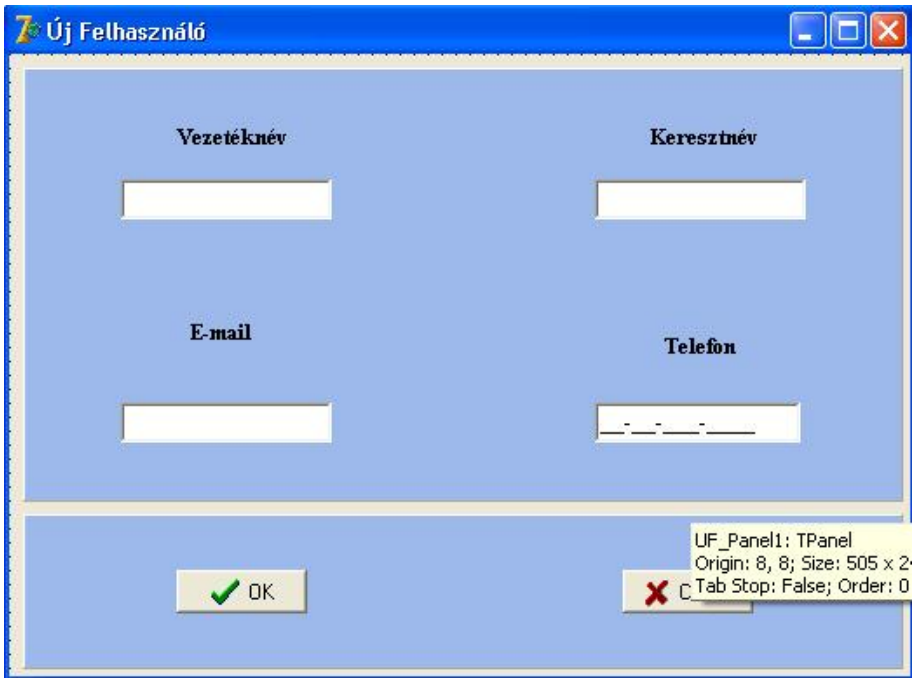
Ha minden adat megfelelő akkor az új géphasználat rögzítésre kerül az adatbázisban, ezen felül a táblázatban is kiírásra kerülnek az új adatok.



6.11. ábra. *Új Géphasználat*

6.4.2. Új Felhasználó

Az "Új Felhasználó" form az új felhasználók adatbázisbeli rögzítését segíti. Meg kell adni a felhasználó vezetéknevét, keresztnevét, e-mail címét. Ezek a beviteli mezők *TEdit* típusúak. Eznfelül még meg kell adni a telefonszámot, aminek a bevitele egy *TMaskEdit* komponensben történik. A MaskEdit 00-00-000-0000 formában fogadja el a számokat,és csak numerikus értéket lehet megadni.



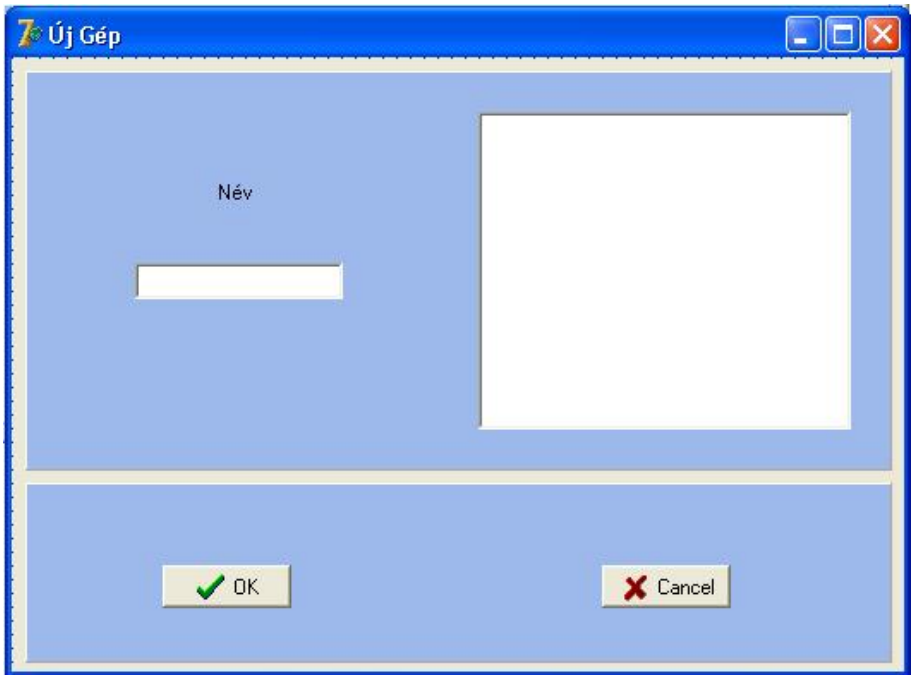
6.12. ábra. *Új Felhasználó*

6.4.3. Új Számítógép

Az új számítógép beviteléhez mindösszesen csak két beviteli komponens szükséges. Az egyik egy sima *TEdit* amiben a számítógép fantázianevét adhatjuk meg. Az új gép hardver paramétereinek megadása egy *TMemo* komponesben történik. Itt a fontosabb adatok kerülnek bejegyzésre mint pl.: alaplap, processzor, videókártya, merevlemez, memória. Az itt megadott tulajdonságok később, egy esetleges harver csere miatt, megváltoztathatók.

6.4.4. Telepítés

Ennek a formnak a neve csak jelképes telepítést jelent. Igazából egy kapcsoló táblát tölt fel, hogy melyik gépre milyen játék kerül. A formon két *DBLookupComboBox* található. Az egyik a "szamitogep"' táblából veszi az adatokat, a másik pedig a "'játék"'



6.13. ábra. *Új Számítógép*

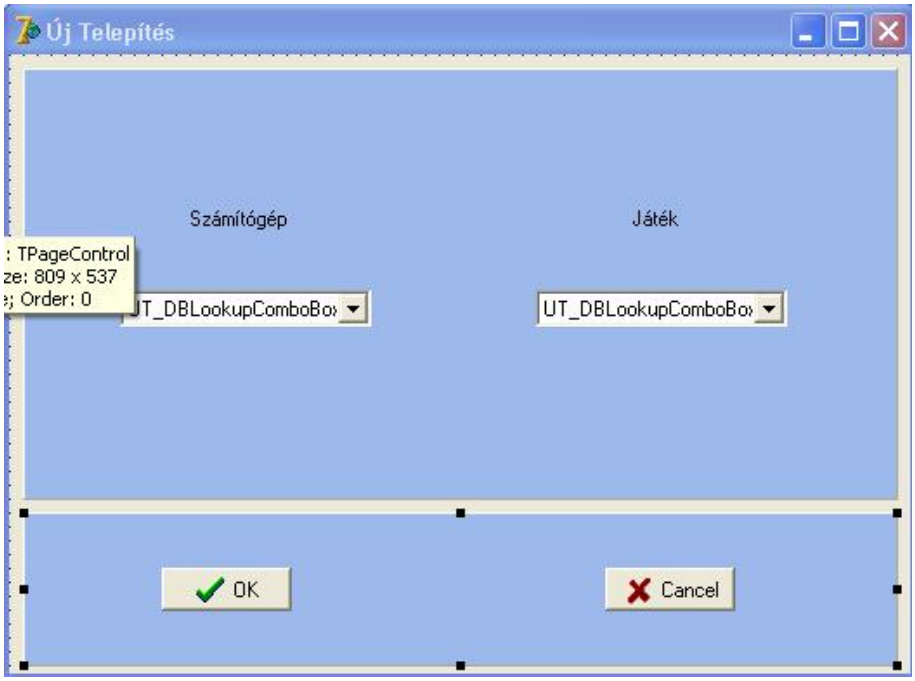
táblából. Ezzel a technikával itt sem lehet hibás adatot megadni. A felhasználók ennek a tábla segítségével kiválaszthatják, hogy melyik gépen foglaljanak helyet.

6.5. Menük

Ebben a programban nem szántam különösebb szerepet a menüknek. A megvalósítandó funkciók mind elérhetők arról a panelről amihez logikailag tartoznak. A program finomításakor, fejlesztésekor talán lehetőség lesz bizonyos funkciók menün keresztül való elérésére is. A program tartalmaz egy főmenü komponenst, de csak az alapvető, formok létrehozását elősegítő menüpontok szerepelnek rajta.

6.6. Weblapok készítése

A programot egy webes szolgáltatás egészíti ki. Ezen a weboldaLon a látogatók tájékoztatást kapnak az üzletben történő eseményekről, bajnokságokról, rendezvényekről, a játékvilág aktuális híreiről. Egy Vendégkönyv egészíti ki a webalkalmazást, amelyen a látogatók egymásnak hagyhatnak üzeneteket.



6.14. ábra. *Új Telepítés*

6.6.1. Kezdőoldal

A kezdőoldal az az oldal amivel a felhasználó először találkozik. Itt kaphatnak a látogatók információkat az üzlet eseményeiről.

6.6.2. Játék Hírek

A Játék Hírek oldalon az üzletben játszható játékok aktuális híreit olvashatjátok. De ezenkívül más érdekességek és újdonságok is napvilágot látnak. A játékok világában naponta több száz friss hír kerül fel a különböző játékokkal foglalkozó oldalakra. Nehéz kihámozni azokat a híreket amelyek azokkal a játékokkal foglalkoznak amivel a játéktéremben játszhatsz. Ezen az oldalon megoldom ezt a problémát, kiválogatom a hírek közül az idevalókat.

6.6.3. Játék Bemutatók

Ezen az oldalon a kendvenc játékainkról egy teljeskörű leírás olvasható. Itt minden olyan információ megtalálható amelyek a játék megkelenésétől a mai napig előkerült. Azok a felhasználók akik még kicsit ismeretlenül mozognak a számítógépes játékok világában, megismerhetik azokat a játékokat amelyek meghatározzák a mai játékvilágot. Mivel, hogy több ezer játék közül választhatunk így egy kezdő játékos csak a fejét kapkodja a sok név

hallatán. Ha ezeket a játékokat tanulmányozza, akkor könnyen eldöntheti, hogy mivel játszon.

6.6.4. Vendégkönyv

A Vendégkönyv oldalon a felhasználók észrevételeiket, megjegyzéseiket és véleményüket irhatják le egymásról, az oldalról, vagy a játékteremről. Ez az oldal chat-ként is funkcionálhat, a felhasználók így akár másodpercek alatt tudnak üzenetet váltani. Az üzenet elküldéséhez megkell adni az e-mail címet és az üzenetet. Ezek után ha az elküld gombra kattintassz akkor az üzenet beíródik az adatbázisba és innentől kezdve mindenki olvashatja.

6.6.5. Linkek

Kedvenc játékaink webes elérhetőségeit tartalmazza. Ezenkívűl a játékvilág érdekesebb weboldalai is megtalálhatók itt.

7. fejezet

Tervek és célok

A program elkészültekor világossá vált előttem, hogy a tervezésekor nem voltam elég körültekintő. Egy esetleges következő munkát mindenképpen alaposabb tervezésnek kell megelőznie. Értem itt az adatbázis tervet, de ide sorolnám a felhasználóval történő kapcsolattartáshoz szügséges formok megtervezését is.

Természetesen azzal is tisztában vagyok, hogy bizonyos hibák, hiányosságok majd csak a program tényleges, mindennapi használatakor derülnek ki. Mindenfélekép tervezem a program továbbfejlesztését, amely az adminisztratív munka megkönnyítését jelentené. Ezen kívűl a felhasználókkal való kapcsolattartás javítása érdekében egy levelező rendszer integrálását is szeretném megoldani. Néhány elem bevezetése komoly programozási ismeretet feltételez, így ezek megvalósítása érdekében szeretnék minél jobb betekintést nyerni a Delphi nyelv rejtelmeibe.

8. fejezet

Irodalomjegyzék

- Baga Edit: *Delphi másképp* 1998
- Paul Kimmel: *Delphi 6* Panem kiadó 2002.
- Marco Cantú: *Delphi Mesteri szinten* Kiskapu Kft. 2004.