SZAKDOLGOZAT

Gali Gyula

Debrecen

2008

Debreceni Egyetem

Informatikai Kar

Informatikai Rendszerek és Hálózatok Tanszék

Adatnyilvántartó program fejlesztése ADO.NET segítségével

Témavezető:
Prof. Dr. Sztrik János
Full Professor

Készítette:
Gali Gyula
mérnök informatikus

Debrecen
2008

*Ezúton szeretnék köszönetet mondani konzulensemnek, **Prof. Dr. Sztrik Jánosnak**, a dolgozat megírásához nyújtott hasznos tanácsaiért, támogatásáért, töretlen bizalmáért.*

Tartalomjegyzék

Bevezetés

A szakdolgozati programom egy komplett adatbázis kezelő és nyilvántartó rendszer mely ADO.NET segítségével készült hoteleknek és kollégiumoknak kiszolgálási elemzéssel.

A szakdolgozatom kapcsán azért esett a választásom a nyilvántartó rendszerekre, mert egy kihívást éreztem benne, mivel egy összetett probléma halmazt kell felölelni a megvalósítás érdekében. Egy nyilvántartó rendszer, nem csak adatokat tárol, hanem kezeli is azokat, megfelelően képes megjeleníteni és műveleteket végezni rajtuk. Az általam elkészített program egy komplett adatbázis kezelő rendszer, mely az adatbázisban lévő táblákat, lekérdezéseket és a táblák közötti kapcsolatokat képes program szinten kezelni és az adatokat a megfelelő módon kiolvasni és megjeleníteni.

Kiegészítettem egy kimutatást végző résszel is, melynek a lényege, hogy a felhasználó által kijelölt időtartamban listázza ki a napokra eső kihasználtságot és adjon meg egy százalékos reprezentatív kimutatást a szállóhely átlagos foglaltságával kapcsolatban. Ezeket úgy valósítottam meg, hogy az időtartamot alkotó minden napra lekérdeztem a programban az adott napra vonatkozó foglaltságokat, vagyis hány vendég vett ki aznap szobát?

A vendégeken belül készítettem még egy másik csoportot is, amelyet a hallgatók alkotnak, így beköltözéskor amennyiben a vendég egyben hallgató is, akkor szükséges megadni az oktatási intézmény nevét, szakját és évfolyamát. Ezért használható a program kollégiumi nyilvántartásra is. Ezt a plusz funkciót az elemzés során is felhasználom kimutatás céljára, hogy láthatóvá váljon, a vendégek közül hány hallgató vette igénybe a szállóhely szolgáltatásait.

Az adatbázissal a kapcsolatot dinamikusan hozom létre a programban, csak akkor építem ki, ha mentésre, vagy közvetlen adatelérésre, olvasásra van szükség. A művelet elvégzését követően pedig azonnal zárom a nyitott kapcsolatot. Ez azért jó, mert így nincs állandó kapcsolat, kisebb a hibalehetőség és a program nem omlik össze ha egy kis időre megszakadna az adatbázissal az összeköttetés.

Az adatok védelme érdekében jelszavas védelemre is lehetőség van, melyet a programban kezelek, így kívülről történő elérése az adatbázisban lévő adatoknak nem lehetséges.

A program egy stabil alapot nyújt az esetleges kiegészítésekre és továbbfejlesztésre, melyet a jövőben bármikor meg lehet tenni a program áttekinthető szerkezete miatt.

1. Rendszerterv

Hardver és szoftver környezet

A program az alábbi rendszertulajdonságokkal rendelkező számítógépen készült:

Fejlesztő szoftverek:

Microsoft Visual Studio 2005 Team Edition for Software Developers
Version 8.0

Microsoft .NET Framework
Version 2.0 SP1

Operációs rendszer:

Microsoft Windows XP Professional
2002-es verzió
Szervizcsomag 2

Számítógép:

Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20 GHz
2.19 GHz
2 GB RAM

A program futásához szükséges minimális rendszerkövetelmény:

Operációs rendszer:

Microsoft Windows 98

Számítógép:

Intel Pentium 2 CPU 400 MHz
400MHz
256 MB RAM

Adatbázis bemutatása

A szakdolgozati programom egy adatbázist kezelő program. Ez az adatbázis a rendszer legfőbb részét képezi, mivel a program innen olvassa ki és írja be a futásához szükséges adatokat. A program a futása közben az adatbázissal dinamikusan építi ki a kapcsolatot. Csak akkor kapcsolódik az adatbázishoz, ha az adatok mentése vagy olvasása azt szükségessé teszi.

Az adatbázis több táblát, lekérdezést és táblák közötti kapcsolatot tartalmaz.
Az adatbázis létrehozására és karbantartására több adatbázis-kezelő program szolgál. Én a Microsoft Office programcsomagban található adatbázis-kezelőt használtam.
A Microsoft Access adatbázis-kezelő programmal dolgoztam az adatbázisom kialakítása során. Az Access egy relációs adatbázis-kezelő rendszer. Hatalmas előrelépés a korábbi adatbázis-kezelő szoftverekkel szemben, hogy teljesen Windows-ra illeszkedő grafikus felülete könnyebb kezelhetőséget biztosít a felhasználó számára. Itt nem szükséges a rengeteg parancs, fájl-szerkezet, függvény és adattípusok teljes körű ismerete, elég ha a felhasználónak van némi jártassága a számítógép kezelhetőségét illetően és alapszintű ismerete az adatbázisokkal kapcsolatban.

A Microsoft Access együtt működik más adatbázis-kezelőkkel, így a Foxpro, dBase és egyéb SQL adatbázisok adatai kényelmesen beimportálhatóak, csatolhatóak a már esetlegesen meglévő Access adatbázisunkhoz. Hasonló módon, ahogy együttműködik más adatbázis-kezelőkkel, úgy dolgozik együtt egyéb Microsoft-os termékekkel. A Microsoft Word dokumentumokban és Excel munkalapokon kitűnően használhatjuk a már Access-ben elkészített adatbázisunk adatait.

Az Accessben használt alap objektumok

Tábla: a relációs adatbázis alapobjektuma, az adatok megfelelő nyilvántartását teszi lehetővé. Egy táblába mindig valamilyen közös témájú adatok kerülnek. A tábla adatai oszlopokra (mezők) és sorokra (rekordok) vannak bontva. Az oszlopok elemei az adatok egy tulajdonságát írja le, míg a rekord az egy adott dologról tárolt információk összessége.

Lekérdezés: a táblákban tárolt adatok feldolgozását, elemzését, esetleg módosítását segítik elő. A lekérdezés legtöbbször egy egyszerűen megfogalmazható kérdés, melyet adatainkkal kapcsolatban teszünk fel. Vagy másképpen fogalmazva, az adatoknak egy bizonyos, általunk felállított kritériumnak való megfeleltetése.

Űrlap: az adatok kényelmes, gyors, esztétikus karbantartására szolgál. Látványosabb megjelenítést, módosítást és felvitelt tesz lehetővé. Az űrlap adatforrása tábla, vagy lekérdezés lehet.

Jelentés: a jelentés célja az adatbázis adatainak papíron való megtekintése. Lehetőségünk van adataink csoportosítására, rendezésére, rész- és végösszegek esztétikus formában történő megjelenítésére. Készíthetünk levélcímkét, összesítéseket tartalmazó listákat, kimutatásokat.

Az *elsődleges kulcs* névre hallgató mező minden táblában helyet kell, hogy foglaljon, annak érdekében, hogy a tábla rekordjait egymástól egyértelműen meg lehessen különböztetni. Ezt a mezőt a tábla elsődleges kulcsának nevezzük, és egy táblában csak egy elsődleges kulcs szerepelhet. Ha a tábla tartalmaz elsődleges kulcsot, akkor annak segítségével a tábla bármelyik elemét pontosan elérhetjük.
Az elsődleges kulcs tulajsonsággal rendelkező mező értéke sohasem ismétlődhet és nem veheti fel a NULL értéket sem.

A táblák létrehozása és adatokkal való feltöltése után kényelmesen tudunk a táblák között *kapcsolatokat* kialakítani. Két tábla közötti kapcsolatot a táblák azonos típusú mezőinek egymással való megfeleltetése adja, általában ugyanazzal a mezőnévvel is rendelkeznek a kapcsolatban résztvevő mezők.

Majd *lekérdezések* segítségével böngészhetünk az általunk megadott feltételeknek eleget tevő adataink között. Kiszűrhetjük a számunkra szükséges adatokat a már feltöltött, kapcsolatokkal ellátott táblákból. Lehetőségünk van rendezni, csoportosítani és összesíteni a kapott adathalmaz elemeit.

Ezek után űrlapokat és jelentéseket is létrehozhatunk, amelyekkel a különböző táblák adatait egyszerre jeleníthetjük meg, a már fentebb említett formákban.
Az űrlap és a jelentés sok hasonlóságot, de sok különbséget rejt magába. Mindkettő egy esztétikusabb, felhasználóbarátabb környezetbe kényszeríti az adatokat.
Az *űrlap* elsősorban inkább az adatok bevitelét szolgálja, de lehetőség van a már meglévő adatok megtekintésére, esetleges módosítására is.

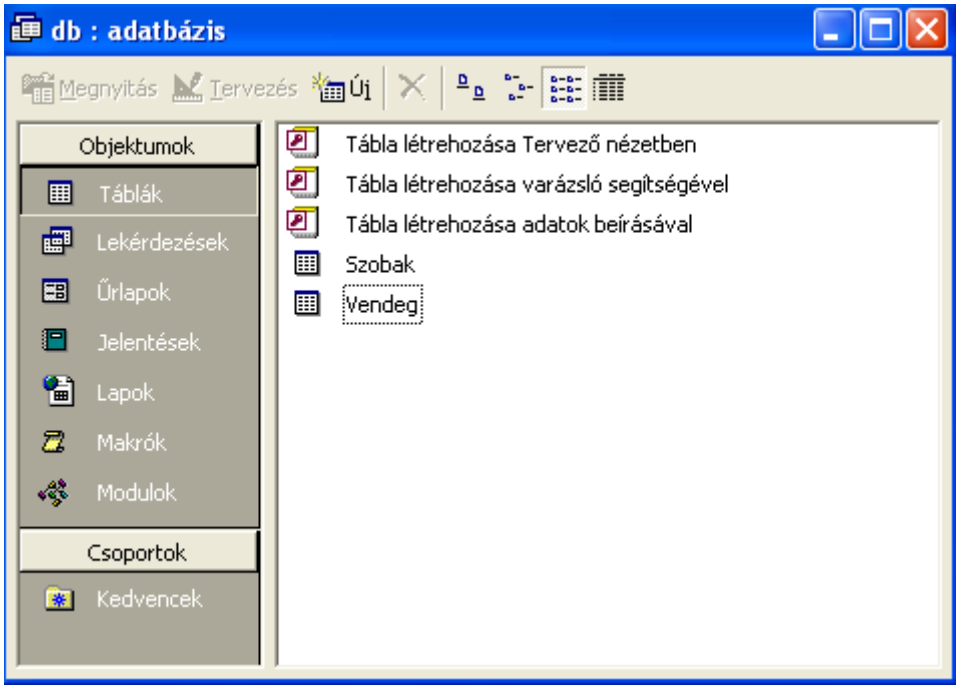
A *jelentés* a formai megjelenítés eszközeit felhasználva szolgáltat információt az adatbázis adatairól. A jelentés a megadott tábla vagy lekérdezés rekordjait csoportosítja és a megadott csoport szinteken összesítést végez.

Táblák bemutatása

Az adattábla az adatbázis egy elhanyagolhatatlan rész, ezért különös figyelmet kell fordítani az elkészítésére, mert a megfelelő tervezéssel megkönnyíthetjük a későbbi munkáinkat az adatbázisban. A hasonló témájú adatoknak mindig azonos tábla a gyűjtőhelye. Táblákból akármennyit létrehozhatunk, de célszerű minél kevesebbet, hogy áttekinthető maradhasson az adatbázisunk.

A szakdolgozatom adatbázisában 2 táblát hoztam létre, amelyek között kapcsolatokat valósítottam meg.(1.kép) Ezáltal az egyik tábla eleme képes hivatkozni a másik tábla elemére és fordítva. Minden tábla rendelkezik névvel, az én tábláimnak a nevei:

- *Szobak* tábla,
- *Vendeg* tábla.



1. kép: A „db.mdb” adatbázis Tábla nézete

A táblák bemutatását először a *Vendeg* táblával kezdem, majd a *Szobak* táblával folytatom.

A *Vendeg* tábla tartalmát tekintve a vendégek összes adatát tartalmazza. Elsősorban itt foglal helyet a Vendégek azonosítására szolgáló *vendégkód* mező, amely elsődleges kulcsként is szolgál a táblában. Számláló típusú, mely azt a célt szolgálja, hogy új vendég felvitelénél az adatbázis növeli eggyel az értékét, (ami sose csökken,) így kizárva azt a hibalehetőséget, hogy két vendég azonos vendégkóddal rendelkezzen. Arra az esetre, hogy elég sok vendég elférhessen az adatbázisban, a vendégkód mező méretének hosszú egészt választottam.(2. kép)

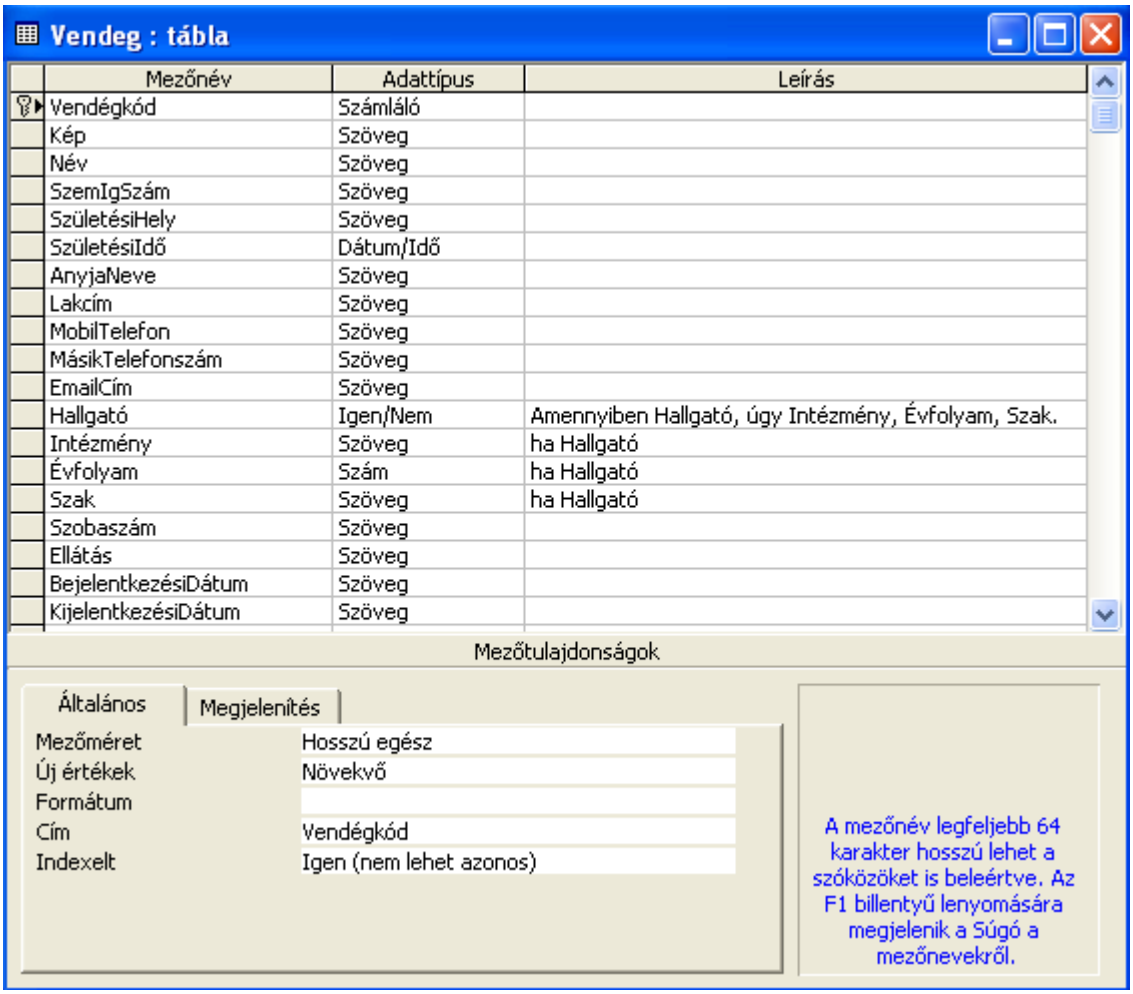
Továbbá a vendég adatainak megfelelően tartalmazza a tábla a többi mezőt is. A *Kép* mező típusát tekintve Szöveg, mely a vendégek igazolvány képeinek elérési útját fogják

tartalmazni ezen mező sorai. A mezőméret 255 karakterre lett beállítva, ezzel maximalizálva az elérési út szöveges hosszát.

A soron következő mező a *Név* mező, melynek tipusát tekintve Szöveg, méretét tekintve pedig 50 karakter. A fixre adott maximális mérettel behatárolva egy név átlagos hosszát úgy gondolom, hogy elegendő helyet biztosítok ezzel az 50 karakter lehetőséggel, ha mégse akkor a vendég nevében rövidítést kell végrehajtani.

A vendég azonosítására szolgáló személyes adatainak következő mezője a *SzemIgSzám* mező, mely értelemszerűen a Személyazonosító igazolvány okmány számát tartalmazza 10 karakter hosszra maximalizálva.

Az eddig ismertetett mezők közül egyiknek sem kötelező a kitöltése adatbázis szinten, viszont a programban egy függvénnyel vizsgáltatom meg az adatok hiánytalan felvitelét. Erre azért van ilyen módon szükség, mert ezáltal pontosabban nyomon követhetőek az adatok.



2. kép: A „Vendeg” tábla Tervező nézete

A további azonosításra szolgáló *SzületésiHely* mező 100 karakteres felső határral várja a vendég születési helyét. Ehhez kapcsolódva a *SzületésiIdő* mező Rövid Dátum típusban lett létrehozva, így formátuma pl.:1988.10.25. lehet.

A személyes adatok utolsó részét képezi az *AnyjaNeve* elnevezésű mező, amely értelemszerűen a vendég édesanyjának a nevét tárolja 50 karakter hosszan. Ezen mező kitöltése sem kötelező adatbázis szinten, mint az előbbiek esetén is, viszont a programban használt, előre megírt ellenőrző függvény fog majd gondoskodni a hiánytalan kitöltésről.

Az Elérhetőség résznél a Vendégtől Lakcímet, Mobiltelefon számot, esetleg másik telefonszámot és e-mail címet várunk. Ezen adatokat Szöveg típusként tároljuk, ezáltal elérve azt, hogy egyéb szimbólumokat és különleges karaktereket is tudjon használni a program az adatokban, amennyiben ez szükséges. Sorrendben *Lakcím*, *MobilTelefon*, *MásikTelefonszám*, *EmailCím* mezőneveket használtam. A mezőméretet tekintve pedig a Lakcím 255, a MobilTelefon, MásikTelefonszám 30-30 és az EmailCím pedig 50 karakternyi helyet foglalhat maximálisan.

Továbbá, amennyiben hallgatóként menthető el a vendég, (ez akkor jelentős, ha kollégium használja a programot) akkor egy logikai típussal deklarált *Hallgató* mező kitöltése válik kötelezővé. Amelyben „True” értéket beállítva adható meg a Hallgatóra vonatkozó további tulajdonságok. Az Intézmény, Évfolyam és Szak mezőket tölthetjük ki a megfelelő adatokkal, a szakdolgozati program segítségével. Az *Intézmény* mező és a *Szak* mező adatainak maximális méretét tekintve az 100 karakternyi hosszan engedélyezett, Szöveges formátumban. Az *Évfolyam* mezőnek pedig Bájt nagyságú Szám típust használok. Ez annyit tesz, hogy értéke 0-255 lehet.

A következő mező a *Szobaszám* nevet kapta, amely a Vendég szobájának számát fogja tárolni 50 karakter hosszan, Szöveges típusban. Azért, mert gondolva arra, hogy a jövőben további fejlesztéseket és kiegészítéseket figyelembe véve a munka megkönnyítése végett lehessen akár a félszobákat és szobarészeket is kiadni. Pl.:1101/A és 1101/B.

Az *Ellátás* mező a Vendég által igénybe vett szolgáltatások csoportját tartalmazza. Típusa ennek is szöveg és 50 karakter hosszan maximalizált. Programban egyenlőre az alábbi felsorolásból fog majd tudni választani ellátást a vendég:

-Ellátás Nélkül

-Reggeli

-Félpanzió

-Teljes Panzió

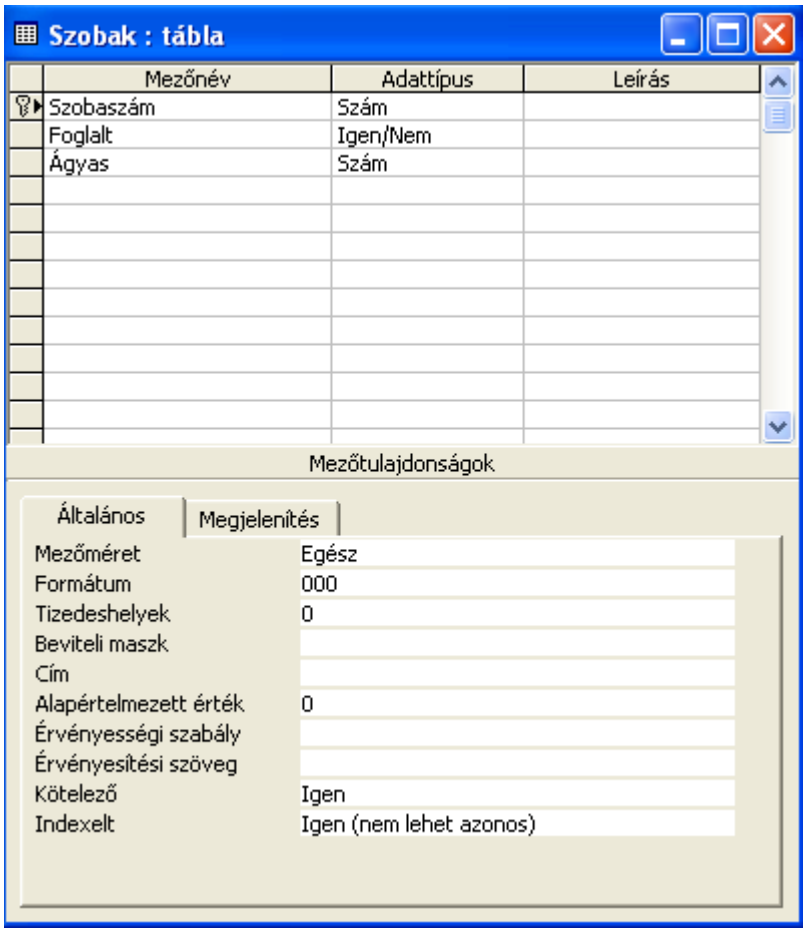
Ezen lehetőségek bővíthetősége meg maradt a programban.

Az utolsóelőtti mező neve a *BejelentkezésiDátum*, mely maximum 50 karakter hosszú lehet és Szöveg típusú. Jogosan merül fel az olvasóban a kérdés, hogy miért használok itt Szöveg típust, amikor Dátum értéket tárolok. Erre a válaszom, hogy a kezdeti verziók során még Dátum típussal dolgoztam, de komolyabb kiolvasások és átalakítások során váltanom kellett a fejlesztés közben. A dátum formátum kiolvasása és adatbázisban történő rögzítése a programban történő átalakítások után elég bonyolulttá válik, így a fejlesztés során inkább String-t használva könnyebb levágni C# controllerei által generált dátum felesleges részeit, (óra:perc:másodperc) mivel ezekre nincs szükség. Az általam elvárt dátum csonkítására pedig nem nyújtott lehetőséget a környezet. Ezért változtattam meg az adatbázisban is a típusát és használok String-t, habár így nagyobb ellenőrzést igényel ez a rész a programban.

A *KijelentkezésiDátum*-mal hasonló a helyzet, annyi különbség viszont mégis van, hogy a Bejelentkezési dátum a vendég felvitele során automatikusan generálódik a mai dátumból, így feltételezve, hogy amikor a vendég az adatbázisba bekerül, akkor költözik és jelentkezik be a szállóhelyre. Ennek módosítására természetesen lehetőséget biztosítottam még a vendég elmentése előtt. Így akár feltételezésem ellenkező esetében is a megfelelő módon használható a Bejelentkezés Dátuma.

A következőkben a *Szobak* nevű tábla mezőit ismertetem.(3. kép)

Legelső mező a *Szobaszám* névre hallgat, amely a szállóhely összes szobájának a számait tartalmazza. Ez a mező típusát tekintve Egész típusú. Azért választottam ekkora értéket, mert gondolva a további bővítésekre, vagy csak feltételezve, hogy a szállóhely több száz, vagy akár ezer szobát is tartalmazhat, kevés lett volna a Byte típus, ami maximum 255 szobát jelent. Ebben a táblában ezt a mezőt választottam Elsődleges kulcsként, mivel egy szoba azonosítására nincs is jobb eszköz, mint a szobaszáma. Természetesen a mező kitöltését kötelezőnek vettem, így nem lehet az adatbázisban létrehozni olyan szobát, mely nem rendelkezik számmal. Továbbá említésre szorul a formátuma is, ahol „000”-t adtam meg, emiatt nem lesz olyan szobaszám, hogy 1 vagy 2, helyettük 001 vagy 002. Ez bevált szokás szállóhelyek szobaszámainál. A szobaszámok mivel elsődleges kulcsként szolgálnak, soha sem lehetnek azonosak, ezt az Indexelt tulajdonságnál állítottam be.



3. kép: A „Szobak” tábla Tervező nézete

Következő mező a *Foglalt* nevet viseli, mely a szállóhely minden szobájához rendelve egy logikai értéket, tárolja a szoba foglaltságát.
Egy szoba akkor foglalt, ha a szobaszáma köthető a Vendég tábla valamelyik vendégének Szobaszám mezőjének értékéhez.
Alapértelmezett értéke „Hamis”, mely annyit tesz, hogy kezdetben minden szoba üres.
Értéke akkor vált át „Igaz” állapotba, ha a vendég ebbe a szobába költözik be. Kitöltése szintén kötelező, viszont az majd a programban fog történni. A vezérlőelem megjelenését tekintve egy Jelölőnégyzet.

Az *Ágyas* nevű mező minden szobához a szoba kapacitásához mérten a benne elhelyezkedő ágyak számát tartalmazza. Kezdetben minden szobát 1 ágyasra választottam, viszont az igények felmerülése után ez változtatható. Ekkor a programban egy rövid kis részletet beépítve vizsgálni kell a szoba Ágyas mezőjének a tartalmát is, mely alapján eldönthető, hogy az adott szoba teljesen foglalt, vagy van még szabad ágy, illetve, ha többen egy szobában szeretnének helyet foglalni, akkor van-e rá lehetőségük és van-e még olyan szabad szoba, mely nem csak egy ágyat tartalmaz.
Ezt a lehetőséget is azért helyeztem el az adatbázisba, hogy egy megfelelő alapot biztosítva a programból több irányba is lehessen tovább indulni.
A mező adattípusa Bájt, ami annyit tesz, hogy a szobában lévő ágyak száma maximum 255 lehet. Azért választottam ezt az adattípust, mert ez volt a választható legkisebb és nem vehet fel negatív értéket, ami az ágyak számában nem utolsó szempont.

A mezőnevek „beszélőnevek”, ami azt jelenti, hogy utalnak a tartalmukra. A típusok pedig úgy lettek megválasztva, hogy először figyelembe vettem a lehetséges felvehető értékeket (számok esetén: pozitív, negatív) és azok nagyságát. Ahol kellett, ott kötelezővé tettem a mező kitöltését, ezzel is elkerülve az esetlegesen fellépő hibákat.

Kapcsolatok típusai

Táblák között kapcsolatokat hozhatunk létre, úgy hogy az egyik tábla megfelelő mezőjét kapcsoljuk össze a másik tábla megfelelő mezőjével. Csak és kizárólag azonos típusú adatokon keresztül kapcsolhatunk össze két táblát. A megfelelő kapcsolat elérése érdekében érdemes a két tábla elsődleges kulcsait összekapcsolni.

A kapcsolat típusát tekintve megkülönböztetünk
„Egy az Egyhez”(1-1), „Egy a Többhöz”(1-N) és „Több a Többhöz”(N-M) kapcsolatot.

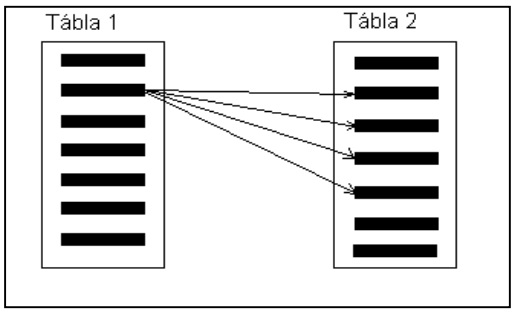
„Egy az Egyhez” kapcsolat

„Az „Egy az Egyhez” kapcsolat esetén az egyik egyed egyik példányához egy és csak egy példányát kapcsoljuk a másik egyednek.”[1] Vagyis az „A” tábla minden egyes rekordjához legfeljebb egy rekord tartozhat a „B” táblában, és a „B” tábla minden egyes rekordjához is csak legfeljebb egy rekord tartozhat az „A” táblában. „Ez a fajta kapcsolat nem túl gyakran használatos, mert a legtöbb információ, amelyet ilyen módon írunk le, leírható egyetlen táblán belül is. Az „Egy az Egyhez” kapcsolat akkor lehet hasznos, ha egy sok rekordból álló táblát több kisebb, könnyebben kezelhető táblára kívánunk felosztani, ha egy tábla valamely részét adatvédelmi megfontolásból külön kívánjuk tárolni, vagy ha az egyik táblában olyan adatokat szeretnénk tárolni, amely a főtáblában csak bizonyos rekordokra érvényes.”[2]

„Egy a Többhöz” kapcsolat

„Az „Egy a Többhöz” kapcsolat a leggyakrabban használatos kapcsolati típus. Ebben a kapcsolatban az „A” tábla valamely rekordjához több rekord tartozhat a „B” táblában, de a „B” tábla valamennyi rekordjához csak egy-egy rekord tartozhat az „A” táblában.”[3]

Tehát az „Egy a Többhöz” kapcsolat lényege röviden, hogy az első tábla egy rekordjához a második tábla több rekordját rendelem, de a második tábla egy rekordjához csak egy rekord tartozik az első táblából.



1. ábra: Az „Egy a Többhöz” kapcsolat

[1] Internet: (Budapesti Műszaki Egyetem) http://www.agt.bme.hu/szakm/adatb/er/er.html

[2] Internet: (ECDLweb) http://ecdlweb.hu/index.php?title=Access_2000_-_Táblák [Egy az Egyhez kapcsolat]

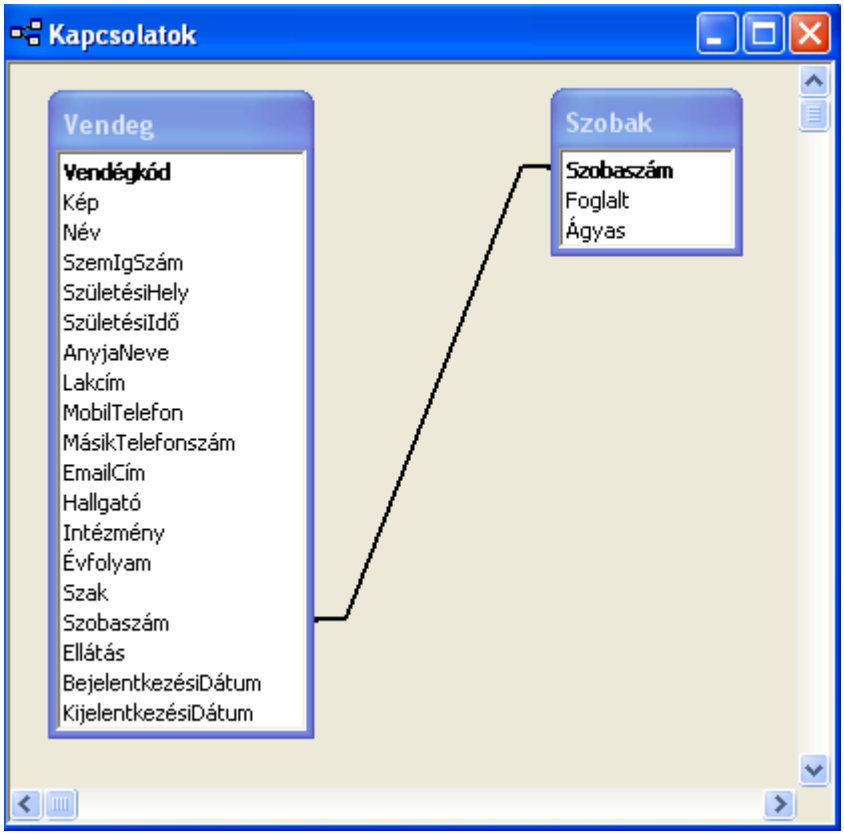
[3] Internet: (ECDLweb) http://ecdlweb.hu/index.php?title=Access_2000_-_Táblák [Egy a Többhöz kapcsolat]

„Több a Többhöz” kapcsolat

A „Több a Többhöz” kapcsolat esetén az első tábla egy rekordjához több rekord tartozik a második táblából és ez kölcsönösen visszafelé is igaz. A második tábla egy rekordjához is több rekord tartozhat az első táblából.

Közvetlen megvalósítása ennek a kapcsolatnak a relációs adatbázis kezelő rendszerekben nem lehetséges. A „Több a Többhöz” kapcsolat csak két „Egy a Többhöz” kapcsolattal reprezentálható. Ehhez viszont igénybe kell vennünk egy harmadik táblát, amelyet illesztő táblának nevezünk, és ennek segítségével kapcsolhatjuk össze a két táblánk Több-több elemét. Ennek az illesztő táblának az elsődleges kulcsa két mezőt tartalmaz: az első tábla és a második tábla külső kulcsát.

A szakdolgozatom adatbázisában „Egy a Többhöz” kapcsolatot használok a *Vendeg* tábla és a *Szobak* tábla között.(2. ábra) A vastag betűs mezőnevek mind elsődleges kulcsot jelölnek.



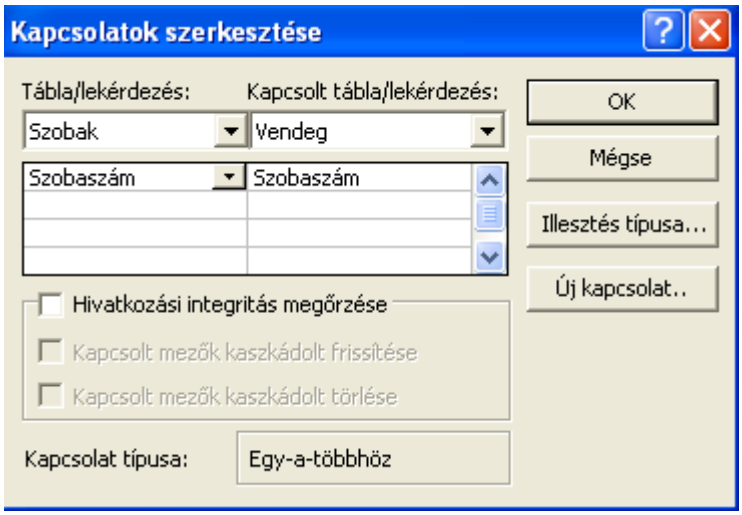
2. ábra: Kapcsolatok

A kapcsolatok létrehozásakor az Access automatikusan felismeri a kapcsolatok irányát a már fentebb említett elsődleges kulcsok miatt.

Látható, hogy a két táblát a Szobaszám mezőjük köti össze.

A kapcsolat bővebben megtekinthető az alábbi képen:(4.kép)

A kapcsolatra vonatkozó további beállításokat megadhatjuk a Kapcsolatok szerkesztése párbeszédablakban, amelyet a kapcsolatot szimbolizáló vastag vonalra történő dupla kattintással érhetünk el.



4. kép: Kapcsolatok szerkesztése

A szakdolgozatomban Access lekérdezést használok, így meg tudom jeleníteni a vendégek adatainak módosítása során a megfelelő mezők tartalmát. Erre azért van szükség, mert így a programban adatforrásként nem az egész adatbázist kell majd betölteni, hanem elég csak ezt a lekérdezést, amiből a fontosabb információt megtudhatjuk egy vendég megfelelő azonosításához, mivel ez a lekérdezés, alkotó elemeit tekintve nem változik.

A lekérdezések lényege, hogy a már meglévő tábláinkkal kapcsolatos kérdéseket tehetünk fel, esetleg azok adatait változtathatjuk meg csoportosan.

Lekérdezések csoportosítása:

Választó lekérdezés:

Ezek a leggyakrabban használt lekérdezés típusok. „A választó lekérdezés egy vagy több táblából olvassa ki az adatokat bizonyos feltételek alapján, majd a kívánt sorrendben megjeleníti azokat.”[4]

Akció lekérdezés:

„Az akció lekérdezések a választó lekérdezésekkel ellentétben valamilyen változtatást hajtanak végre valamely tábla rekordjain (akár egyszerre több rekordon is).”[5]

Az akció lekérdezéseket attól függően, hogy mi ez a változtatás, 3 kategóriába soroljuk:

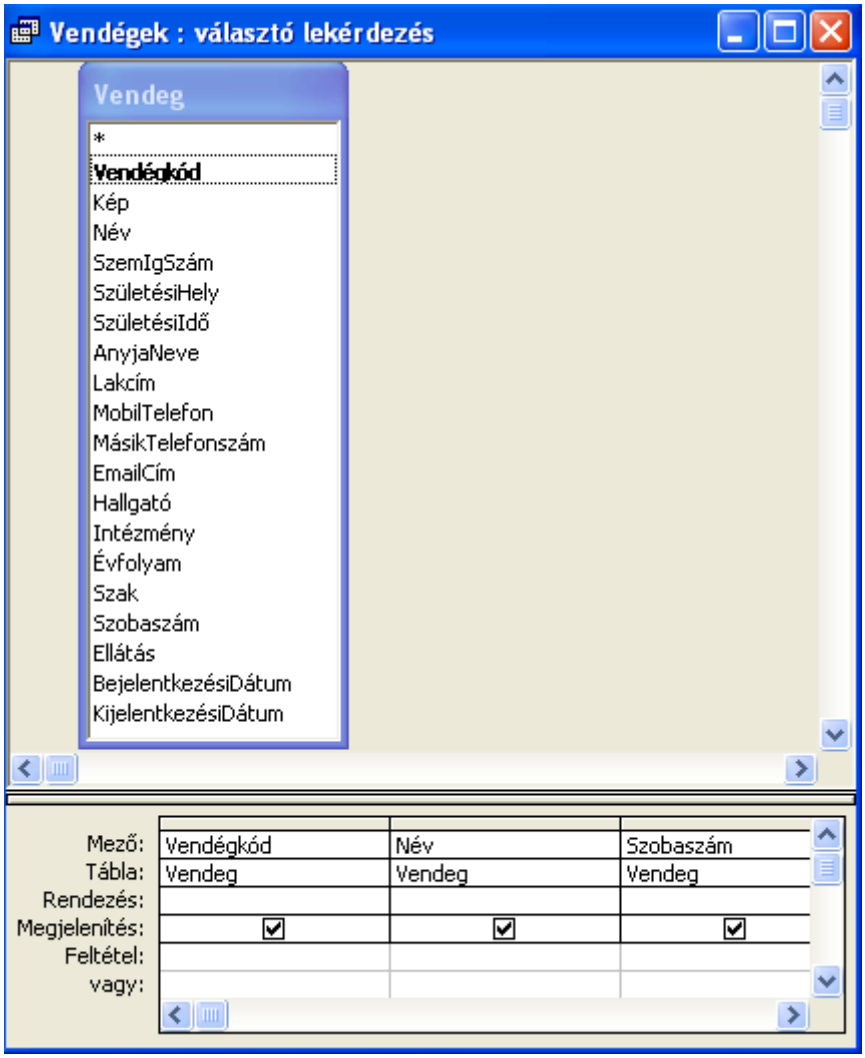
[4] Internet: (ECDLweb) http://ecdlweb.hu/index.php?title=Access_2000_-_Lekérdezések [Választó lekérdezés]

[5] Internet: (ECDLweb) http://ecdlweb.hu/index.php?title=Access_2000_-_Lekérdezések [Akció lekérdezés]

- Törlő lekérdezés: Ez törli az összes olyan rekordot, melyek megfelelnek a feltételnek.
- Frissítő lekérdezés: A feltételnek megfelelő rekordok mezőinek módosítása.
- Hozzáfűző lekérdezés: Segítségével rekordokat másolhatunk át egyik táblából egy másik táblába, ill. új rekordot vihetünk fel egy adott táblába.

A „*Vendégek*” nevű lekérdezés egy választó lekérdezés, mely a „Vendeg” tábla megfelelő mezőinek kiválasztásával (amelyeken a lekérdezést szeretnénk végrehajtani) hozható létre. „Vendeg” tábla 3 mezőjét használtam fel a lekérdezéshez:(5. kép)

- Vendégkód
- Név
- Szobaszám



5. kép: „Vendégek” lekérdezés

Microsoft .NET keretrendszer

„A **Microsoft .NET** szoftverek olyan összessége, amely segít összekapcsolni

⇒ az információt,

⇒ az embereket,

⇒ a számítógéprendszereket és

⇒ a hardvereszközöket.

Részeit képzik a kliensek (például Windows XP, Windows CE, Microsoft Office XP), a szerverek (például Microsoft Windows Server 2003, Microsoft SQL Server) és a fejlesztői eszközök (például a Microsoft Visual Studio .NET 2003).

A Microsoft által készített *.NET keretrendszer* (.NET Framework) gyors alkalmazásfejlesztést (RAD), platformfüggetlenséget és hálózati átlátszóságot támogató szoftverfejlesztői platform."[6]

Több új funkciót és eszközt vezetett be az API-ba, melyek képessé teszik a fejlesztők számára Windowsos és webes alkalmazások, valamint komponensek és szolgáltatások (webszolgáltatás) fejlesztését. A .NET új objektum-orientált API-t tesz elérhetővé.

„Az API alkalmazásprogramozási felület vagy alkalmazásprogramozási interfész (angolul Application Programming Interface) egy program vagy rendszerprogram azon eljárásainak (szolgáltatásainak) és azok használatának dokumentációja, amelyet más programok felhasználhatnak. Egy nyilvános API segítségével lehetséges egy programrendszer szolgáltatásait használni anélkül, hogy annak belső működését ismerni kellene."[7]

A .NET keretrendszert elég általánosnak tervezték, hogy több különböző magas szintű nyelvet legyen képes fordítani. Több fejlesztői segédeszköz áll rendelkezésre kifejezetten a .NET platformon történő fejlesztéshez. A legfontosabb példa a Visual Studio .NET, a Microsoft által nyújtott integrált fejlesztő környezet (IDE).

Röviden a C#-ról

„A Microsoft Visual C# komponens alapú, hatékony, mégis egyszerű programozási nyelv, melyet főleg a .NET-keretrendszerben alkalmazásokat író fejlesztőknek szántak."[8]

---

[6] Internet: (Wikipédia) http://hu.wikipedia.org/wiki/Microsoft_.NET

[7] Internet: (Wikipédia) http://hu.wikipedia.org/wiki/API

[8] Internet: (Wikipédia) http://hu.wikipedia.org/wiki/C_Sharp_programozási_nyelv

A **C#** a Microsoft által a .NET keretrendszer részeként kifejlesztett objektumorientált programozási nyelv. A C#-ot úgy tervezték, hogy meglegyen az egyensúly a fejlesztő nyelvi szabadsága és a gyors alkalmazásfejlesztés lehetősége között. A C# az a programozási nyelv, ami a legközvetlenebb módon tükrözi az alatta működő, minden .NET programot futtató .NET keretrendszert.

A primitív adattípusai, objektumok, a .NET típusok megfelelői. Szemétgyűjtést (Garbage collection) használ, valamint az absztrakcióinak többsége (osztályok, interfészek, delegáltak, kivételek…) a .NET futtatórendszert használja közvetlen módon.

A C vagy C++ nyelvhez hasonlítva a C# több korlátozást és továbbfejlesztést is tartalmaz.

A teljesség igénye nélkül csupán néhányat a lehetőségei közül:

- ✓ Az objektumok nem szabadíthatók fel közvetlen módon, ehelyett a szemétgyűjtő szabadítja fel őket, mikor már nincs rájuk hivatkozás. Ez a módszer kizárja a nem létező objektumokra való hivatkozás lehetőségét.
- ✓ A tömbdeklaráció szintaxisa eltérő (*int[] a = new int[5]* az *int a[5]* helyett).
- ✓ Tulajdonságok (*Properties*) használhatók, amelyek úgy tesznek lehetővé kódfuttatást mezők beállításakor és olvasásakor, mintha mezőhozzáférés történne.

Említésképpen a kódkönyvtárakról néhány mondatban:

A legtöbb programozási nyelvtől eltérően a C# megvalósítások nem rendelkeznek önálló, eltérő osztály- vagy függvénykönyvtárakkal. E helyett a C# szorosan kötődik a .NET keretrendszerhez, amitől a C# kapja a futtató osztályait és függvényeit.

A kód névterekbe van rendezve, mely a hasonló funkciót ellátó osztályokat fogja össze. Például System.Drawing a grafikai, System.Collections az adatstruktúra és System.Windows.Forms a Windows Forms funkciókat fogja össze.

Példa:

Egy egyszerű C# program:

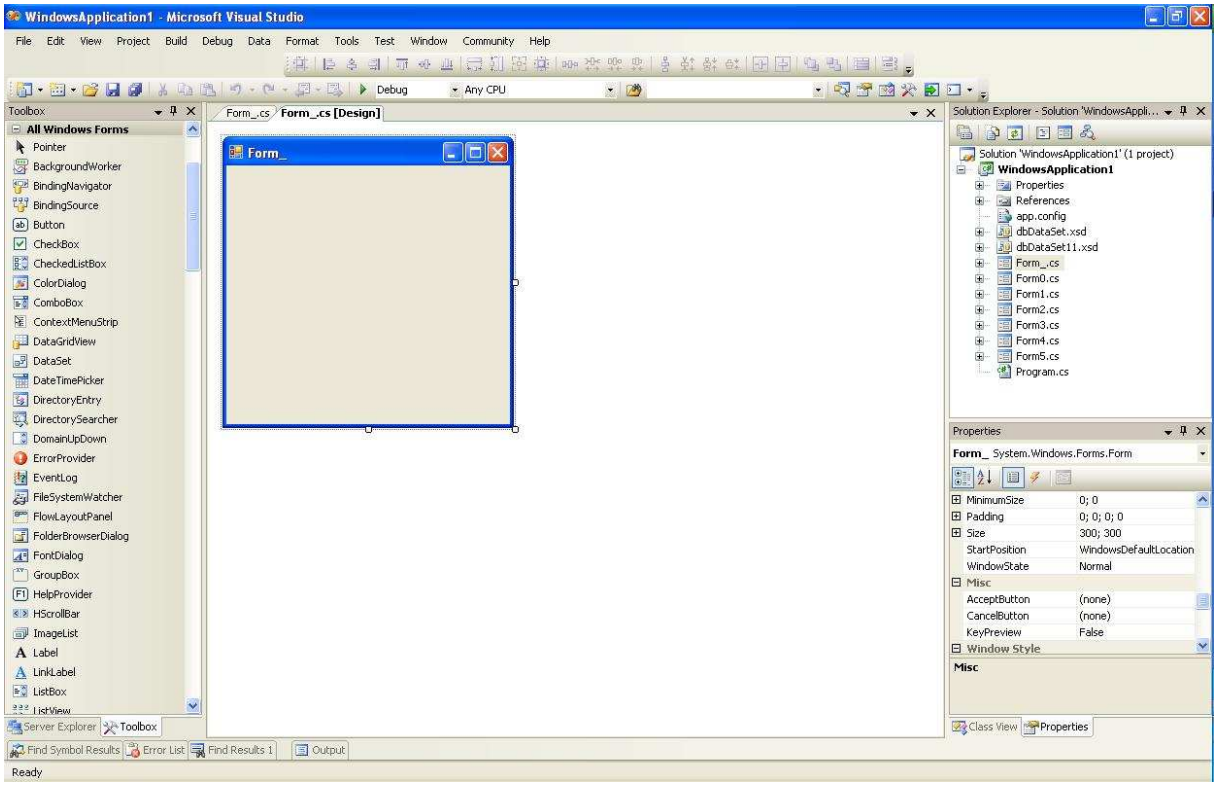
```
public class ExampleClass      //Ez az osztálydeklaráció. Publikus, azaz bármely más projekt
{                                                          //szabadon használhatja az osztályt.
  public static void Main()    //Ez a program belépési pontja, a program végrehajtása kezdődik.
  {
    System.Console.WriteLine("Helló világ!");   //A  feladat végrehajtása.
  }                            //A Console egy rendszerobjektum, ami egy parancssort jelképez.
                               //A Console objektum WriteLine metódusának meghívása a
                               //paraméterként átadott szöveget kiírja a parancssorba.
}
```

A Microsoft Visual Studio fejlesztői környezete

A Microsoft Windows™ operációs rendszer alatt futó alkalmazások készítésének talán legnépszerűbb és leggyorsabb eszköze a Microsoft Visual Studio™ fejlesztőrendszer.

„A Visual Studio 2005 fejlesztői környezetben minden szükséges eszközt és funkcionalitást megtalálunk, így akár nagyobb C#-projektek készítésére is alkalmas. Akár egyéb nyelvekben írt modulokat is gond nélkül használhatunk benne.”[9]

A szakdolgozatom elkészítéséhez általam használt *Microsoft Visual Studio 2005 Team Edition for Software Developers* teljes mértékben segítséget nyújtott, szabadkezet adva minden olyan megvalósítás során, amit más programozási rendszerrel illetve nyelvvel egyáltalán nem vagy esetleg elég körülményesen lehetett volna megvalósítani.(6. kép)



6. kép: Microsoft Visual Studio 2005 kezelő felülete

A képernyő tetején látható *menüsávval* elérhetjük a programozási környezet használatához szükséges funkciókat. Ahogy minden Windows-ban használt programban megszokhattuk, a menüt egérrel és billentyűzettel is elérhetjük. A menüsáv alatt található *eszköztár* gombjai megkönnyítik a leggyorsabban használt parancsok előhívását. A képernyő legnagyobb részét a *Code and Text Editor* ablak foglalja el. Ebben láthatjuk a forrásfájl tartalmát. Többfájlos projekt esetén minden egyes forrásfájl saját füllel rendelkezik, melyen a forrásfájl neve

[9] John Sharp (2005): Microsoft® Visual C# 2005 lépésről lépésre: Első fejezet, 3.oldal

látható. Ha valamelyik forrásfájlt meg szeretnénk nyitni a Code and Text Editor ablakban, kattintsunk az azonos nevű fülre. Más elemek mellett a *Solution Explorer* megjeleníti a projektben használt fájlok nevét. Ha a Code and Text Editor ablakban szeretnénk nyitni valamelyik fájlt, innen is megtehetjük: kattintsunk kétszer annak nevére.

Az ablak jobb oldalán, a *Solution Explorer*-ben láthatjuk a projekt által használt fájlokat, legfelső szinten a „megoldásfájlt", ebből alkalmazásonként mindig csak egy van. Minden megoldásfájl hivatkozik legalább egy projektfájlra.

Ez alatt található a *Properties* ablakrész, amely az éppen aktuális komponens tulajdonságait foglalja össze.

Az ablak bal oldalán a *Toolbox* foglal helyet, ahol a különböző komponensek, vezérlő elemek és egyéb eszközök találhatóak.

ADO.NET bemutatása

„Az **ADO.NET** a .NET keretrendszer része, egy olyan osztálykönyvtár, ami az (elsősorban relációs) adatbázisokhoz való hozzáférést támogatja.
Az **A**ctiveX **D**ata **O**bjects (ADO) továbbfejlesztett verziója."[10]

A számítógépes alkalmazások legnagyobb része adatvezérelt. Az utóbbi időben a kisebb lokális adatbázisok helyét egyre inkább a központosított adatbázisok vették át. Így szükség volt az adatelérési technológiák átalakítására is.

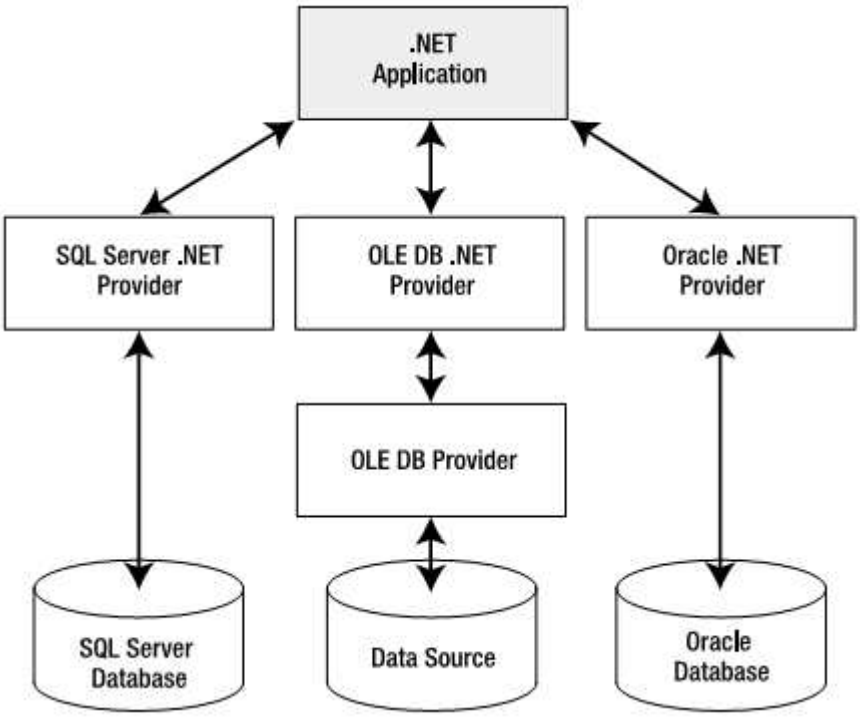
„A .Net-keretrendszer bevezetésével a Microsoft úgy döntött, hogy frissíti adatbázis-hozzáférési modelljét, az ActiveX adatobjektumokat (ADO), és létrehozta az ADO.NET-et."[11] Az eredeti ADO architektúrához képest az ADO.NET lényegesen sok fejlesztést tartalmaz és így jobb együttműködési képességgel és teljesítménnyel rendelkezik. „Nem létezik már RecordSet típus, a Microsoft létrehozta a kapcsolat nélküli adathozzáférést és –műveleteket támogató, TableAdapter és DataSet osztályokat. Ezek jobb méretezhetőséget biztosítanak, mivel többé nincs szükség a folyamatos adatbázis-kapcsolatra. Alkalmazásaink így nem használnak majd annyi erőforrást. Az ADO.NET connection pooling mechanizmusával az

[10] Internet: (Wikipédia) http://hu.wikipedia.org/wiki/ADO.NET

[11] John Sharp (2005): Microsoft® Visual C# 2005 lépésről lépésre: Huszonharmadik fejezet, 451.oldal

adatbázis-kapcsolatokat több alkalmazás is újra használhatja, így csökken az adatbázisokhoz kapcsolódások és lekapcsolódások száma."[12]

Az ADO.NET olyan osztályokat tartalmaz, amelyek segítségével lehetőség nyílik adatforrásokhoz - legtöbb esetben relációs adatbázisokhoz - való kapcsolódásra, adatbázis parancsok futtatására és a már kinyert adatok adatbázis-kapcsolat nélküli kezelésére.
Az ADO.NET architektúrát a 3. ábra szemlélteti.



3. ábra: Az ADO.NET architektúra

Az ADO.NET *szétkapcsolt adatarchitektúrát* használ. Ez azt jelenti, hogy az alkalmazásunk nincs folyamatosan beszélő viszonyban a DB szerverrel. Ennek egyértelmű előnye az, hogy lecsökkentjük az adatforgalmat, illetve az adatbázis kiszolgálót tehermentesíthetjük, amit már korábban említettem. Hátránya az, hogy nem konzisztens (tehát nem a legfrissebb) adatokkal tudunk dolgozni. Felvetődhet bennünk az a kérdés, hogy ha nincs folyamatos kapcsolat, akkor mégis hogyan képes működni? A válasz egyszerű, létrehozunk az adatbázisból egy másolatot, mellyel dolgozunk, majd a változtatásokat visszavezetjük a **DB** szerverbe. Az adatbázis másolatot **adatkészlet**nek (**DataSet)** fogjuk hívni. Az adatszinkronizációért felelős objektumot pedig **adatillesztő**nek (**DataAdapter)** nevezzük.

[12] John Sharp (2005): Microsoft® Visual C# 2005 lépésről lépésre: Huszonharmadik fejezet, 452.oldal

Az ADO.NET eszközeit alapvetően két típusba sorolhatjuk:

- kapcsolat-alapú objektumok,
- tartalom-alapú objektumok.

A kapcsolat-alapú objektumok csoportjába tartozik például a Connection, a Command, a DataReader vagy a DataAdapter, a tartalom-alapú objektumok csoportjának a része a DataSet vagy a DataRelation. Fontos különbség közöttük, hogy míg a kapcsolat-alapú objektumok adatforrás specifikusak, tehát alkalmazásukat befolyásolja az, hogy honnan szeretnénk adatot kinyerni, addig a tartalom-alapúak adatforrástól függetlenek.

„A **DataAdapter** adatillesztő egy olyan objektum, mely kiadja a megfelelő parancsot (utasításokat) az adatbázisnak, majd a válaszul kapott eredményt eltárolja egy adatkészletben. Tehát az adatszolgáltató csak akkor fog kapcsolódni az adatbázishoz, ha az adatillesztővel valamilyen műveletet akarunk végrehajtani, majd miután ez megtörtént, a kapcsolat megszakad. (Valójában csak a Connection objektum csatlakozik le a csatornáról, mely csatorna utána visszakerül egy verembe újra felhasználásra. A vermet connection pool-nak hívják, az ezt kezelő szolgáltatás, pedig a már korábban említett **connection pooling** -nak). Az adatillesztő egy eléggé összetett objektum. A komplexitásának első jele az, hogy ő képes kezelni automatikusan a connection objektumot, tehát a csatorna megnyitásával, lezárásával, sőt egyáltalán a connnection objektummal már nem is kell törődnünk.”[13]

A szakdolgozati programomban használva mindkét megoldást, megtanultam a módszerek megfelelő használatát. Külön létrehoztam a connection objektumot és kezeltem a kapcsolatot, de van olyan rész is a programban, ahol pedig a DataAdapter-re bíztam ennek lekezelését.

”A DataAdapter magába foglal 4 Command objektumot is
(SelectCommand, InsertCommand, DeleteCommand, UpdateCommand),
annak érdekében, hogy ő automatikusan vissza tudja vezetni az adatbázisban a változásokat. Természetesen nem kötelező megadni az összeset, kivéve a SelectCommand-ot, hogy fel tudja tölteni a DataSet-et.”[14]

Röviden a **DataSet**-ről is néhány gondolt:
„Az adatkészlet egy *xml alapú* dolog, amely valójában több fájlból áll. Ez egy *mini adatbázis*. Ezért szinte minden olyan dologgal rendelkezik, amivel a nagy testvére(DB) is. Egyszerre

[13] Internet: (msportal) http://msportal.hu/blogs/csalap/default.aspx?PageIndex=2 [ADO.NET Adatelérési modell]

[14] Internet: (msportal) http://msportal.hu/blogs/csalap/default.aspx?PageIndex=2 [ADO.NET Adatelérési modell]

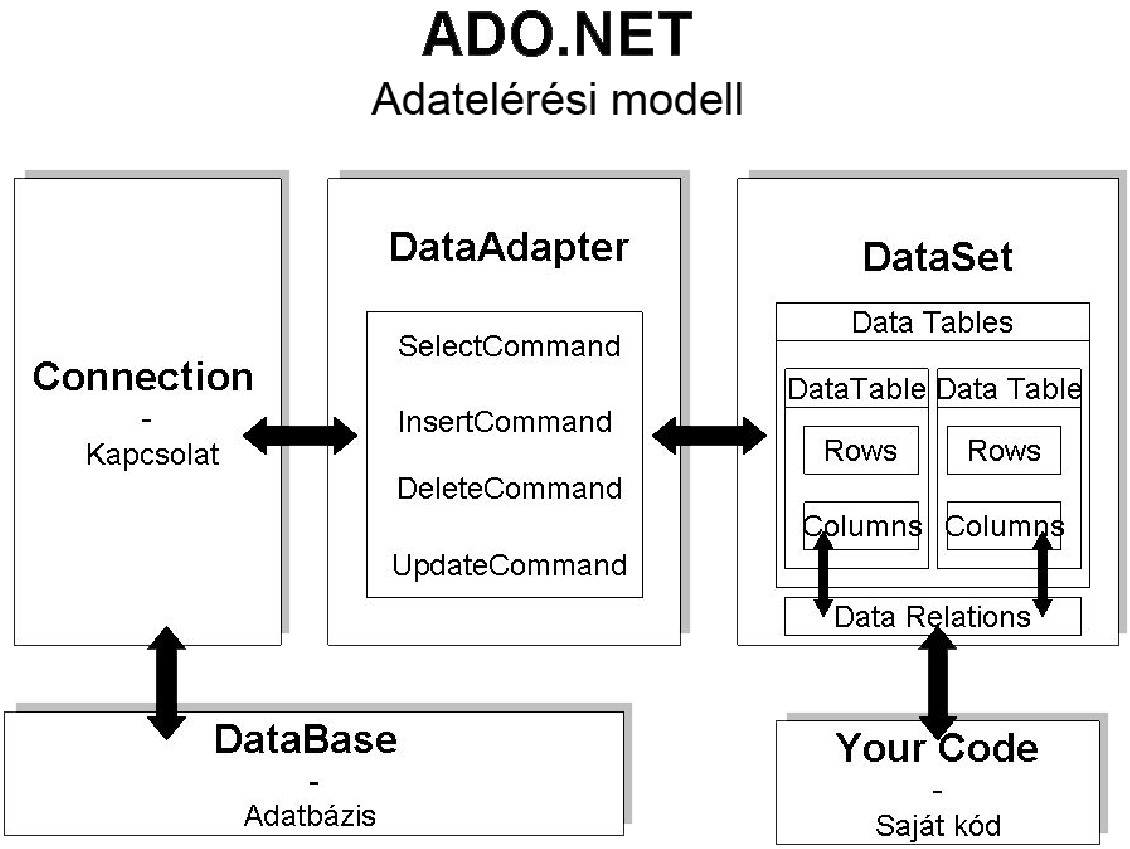
több táblát tárolhatunk benne (DataTables), illetve ezek között relációkat szabhatunk meg (DataRelations).

A DataSet-et adatokkal a DataAdapter Fill metódusának segítségével tölthetjük fel. A változások visszavezetéséhez pedig az Update metódust használjuk!”[15]

Természetesen ez jóval erőforrás igényesebb, mintha egy adatolvasóval nyernénk ki adatokat az adatbázisból. Így törekedjünk arra, hogy minél kevesebbszer kelljen az adatbázishoz kapcsolódnunk. Összefoglalásként a 4. ábra szolgál.

Két ajánlás a szerzőtől:

- Olyan esetekben használjunk a *DataReader*-t, amikor csak adatokat akarunk kiolvasni az adatbázisból, vagy esetleg minimális módosításokat szeretnénk végrehajtatni (pl.: 1 Delete)!
- Csak több adatmanipulációs lépés elvégezése esetén, használjunk DataSet-et!



4. ábra: ADO.NET Adatelérési modell

[15] Internet: (msportal) http://msportal.hu/blogs/csalap/default.aspx?PageIndex=2 [ADO.NET Adatelérési modell]

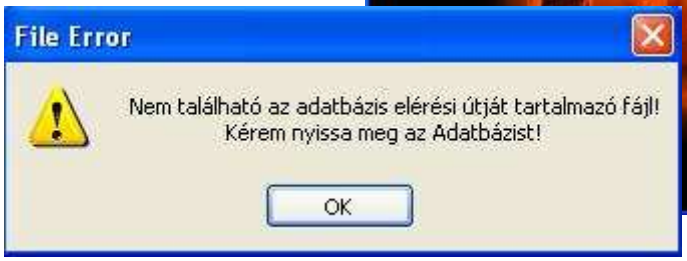
2. Felhasználói dokumentáció

Ebben a fejezetben részletesen ismertetni fogom a program működését, kifejezetten a felhasználók számára. Külön kitérek az esetleges hibalehetőségekre és azokra a dolgokra, amikre figyelni kell a program futtatása során. A felhasználó ebben a fejezetben kellő ismeretet szerezhet a program megfelelő használatához.

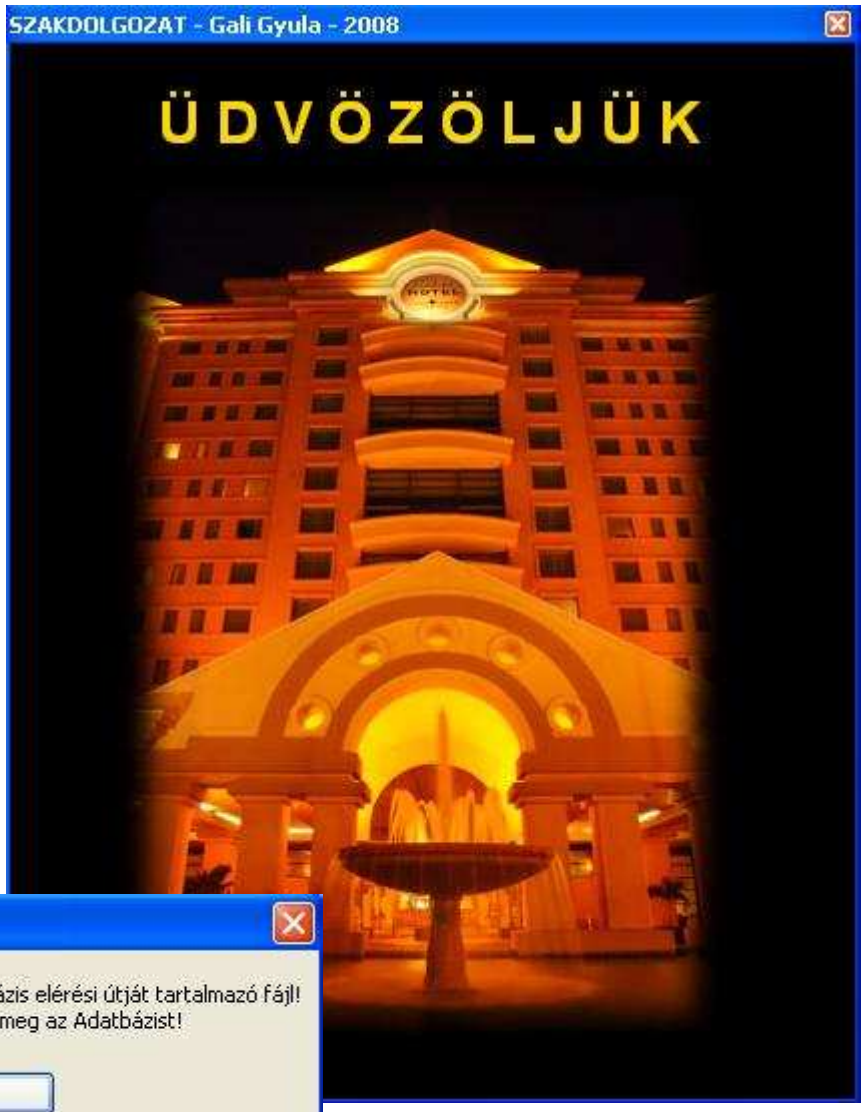
Program használata

A program elindítását követően egy *Üdvözlő ablak* fogad minket (7.kép), amelyre kattintva, vagy bezárva tudunk tovább lépni. Már ekkor a háttérben zajló folyamatok az adatbázis elérésének ellenőrzését végzik.

Ha az ellenőrzés során nem találta meg a program az adatbázist a legutóbbi használatból, vagy ha ez az első használat, úgy az Üdvözlő ablakot elhagyva ki kell jelölnünk egyet a további munkáinkhoz. (8.kép)



7. kép: Üzenet az adatbázis kijelölésére



8. kép: Üdvözlő ablak

Továbblépve a szoftver Főmenüjébe érkezünk (9. kép), ahol az alábbi lehetőségek közül választhatunk:

⇒ Új vendég felvitele: Egy új vendég adatainak rögzítése az adatbázisban.

⇒ Vendég adatbázis: Az adatbázisban szereplő vendégek adatainak módosítása, törlése, adatok között történő rendezés, szűrés.

⇒ Kiszolgálási elemzés: Egy adott időtartamra nézve a vendégek milyen gyakorisággal költöztek be és ki.

⇒ Jelenleg használt adatbázis lecserélése: az aktuálisan kijelölt adatbázis helyett egy másik kijelölése. A program képes kezelni a távoli helyen lévő adatbázist is, így akár nyugodtan használhatjuk hálózati adatbázis kezelő, adatnyilvántartóként is.

9. kép: Főmenü

A program ismertetése során minden részletre ki fogok térni, így nem éri majd semmilyen meglepetés a felhasználót a szoftver használata közben. Külön figyelmet fordítok a Vendégek felvitele és módosítása során jelentkező feltétel szabályozásra, ami annyit tesz, hogy megfelelő módon kiszűri a hibás formátumú vagy helytelenül felvinni szándékozó adatokat. A program interaktív módon, a felhasználói bevitel által irányítottan képes különbséget tenni a hiányos vagy hibás adatot tartalmazó mezők között attól függően, hogy a felhasználó elírás során rontotta el azt, vagy szándékosan. Ekkor figyelmeztet a program a hibára és jelzi annak helyét is.

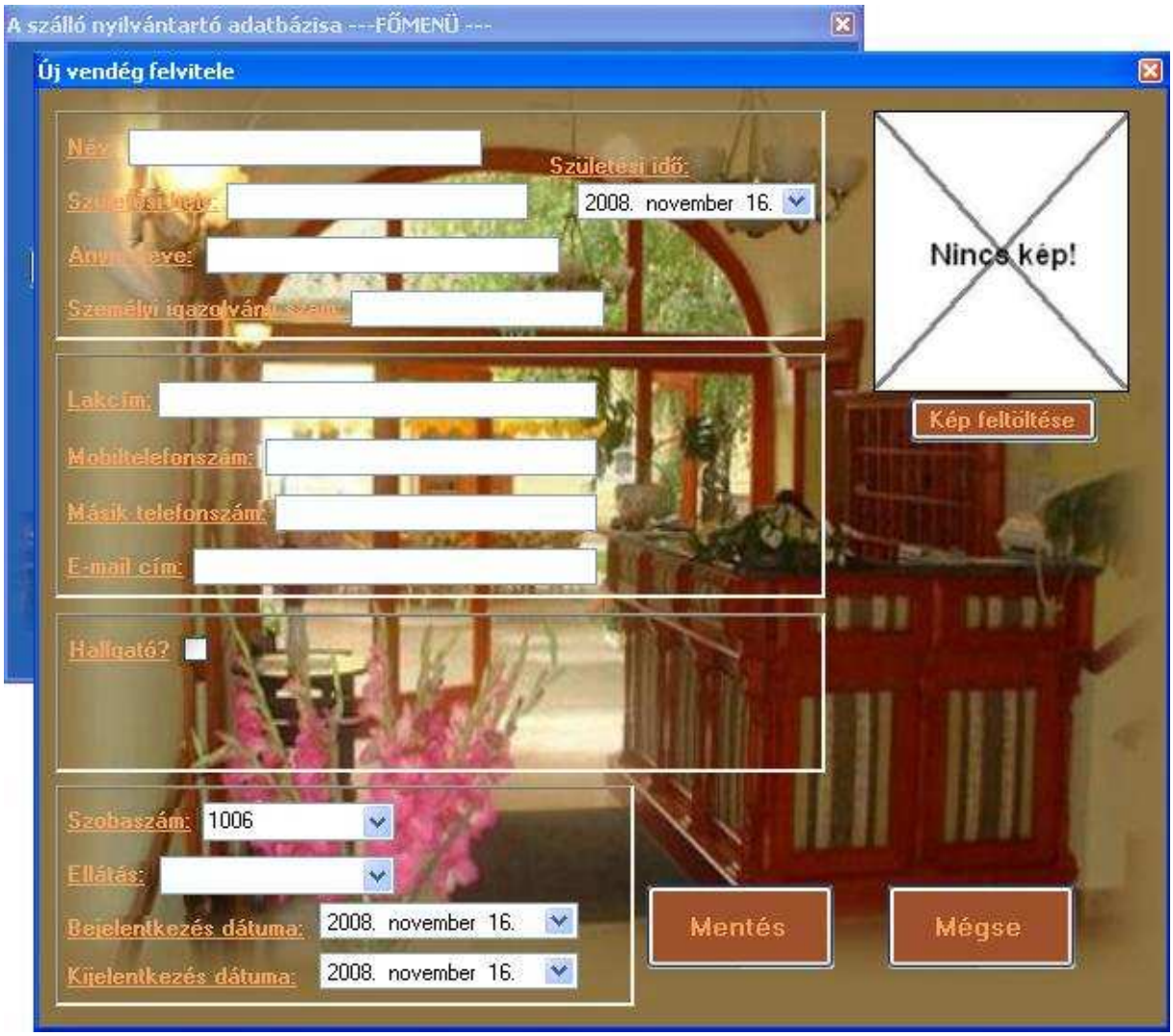
Új vendég felvitele

Az adatbázisba Új vendég rögzítése esetén lényeges az adatmezők megfelelő és hiánytalan kitöltése. A Vendég adatait illetően azok különböző csoportokba tartozhatnak, így szeparáltam el én is ezeket 4 részre az adatlapon, 4 panelbe foglalva.(10. kép)

A felső panel tartalmazza a vendég *személyes adatait*:

- Név,
- Születési hely,
- Születési idő,
- Anyja neve,
- Személyi igazolvány száma.

Ezek a vendég megfelelő azonosításához szükségesek. Mind szöveges mező, kivéve a Születési idő, amely egy naptár belsőprogram segítségével állítja be a kívánt időpontot.



10. kép: „Új vendég felvitele” adatlap üresen

A következő adatcsoportot tartalmazó panel az *elérhetőségeket* menti, az itt lévő mezők kitöltése szintén kötelező:

- Lakcím,
- Mobil telefonszám,
- Másik telefonszám,
- E-mail cím.

Mind szöveges mező, figyelembe véve azt, hogy más-más formátumú mobilszámok, lakcímek és e-mail címek lehetnek. Nem szabtam feltételnek azt, hogy a lakcím esetén egyéb [emelet, ajtó] adatokat is megadjanak, mert zavaró lehet azok számára, akik nem ilyen lakásban laknak, csak utca, házszám az elérhetőségük. Tévhitnek tartom azt, hogy az lenne a jó megoldás, ha minden adatot elkérünk az ügyféltől, mivel kitöltése közben akár belebonyolódhat abba. Így én szabadkezet adok az adatok kitöltését illetően, nem szabok meg különböző formátumbeli feltételeket, ezzel „felhasználóbarátabbá” és könnyebben kezelhetővé teszem a programomat.

A harmadik adatcsoport a Vendég foglalkozására utalva vár el adatokat, amennyiben *hallgató* az illető. Ha ez teljesül, vagyis valamely oktatási intézmény tanulója, akkor a következő adatokat kell megadnia:

- Intézmény,
- Évfolyam,
- Szak.

Erre azért volt szükség, mert így kollégiumok részére is teljes mértékben használhatóvá válik adatnyilvántartásra a szoftver.

A legalsó panel által közbezárt adatok pedig a *szállóhely szobájának* igénybevételével kapcsolatosak:

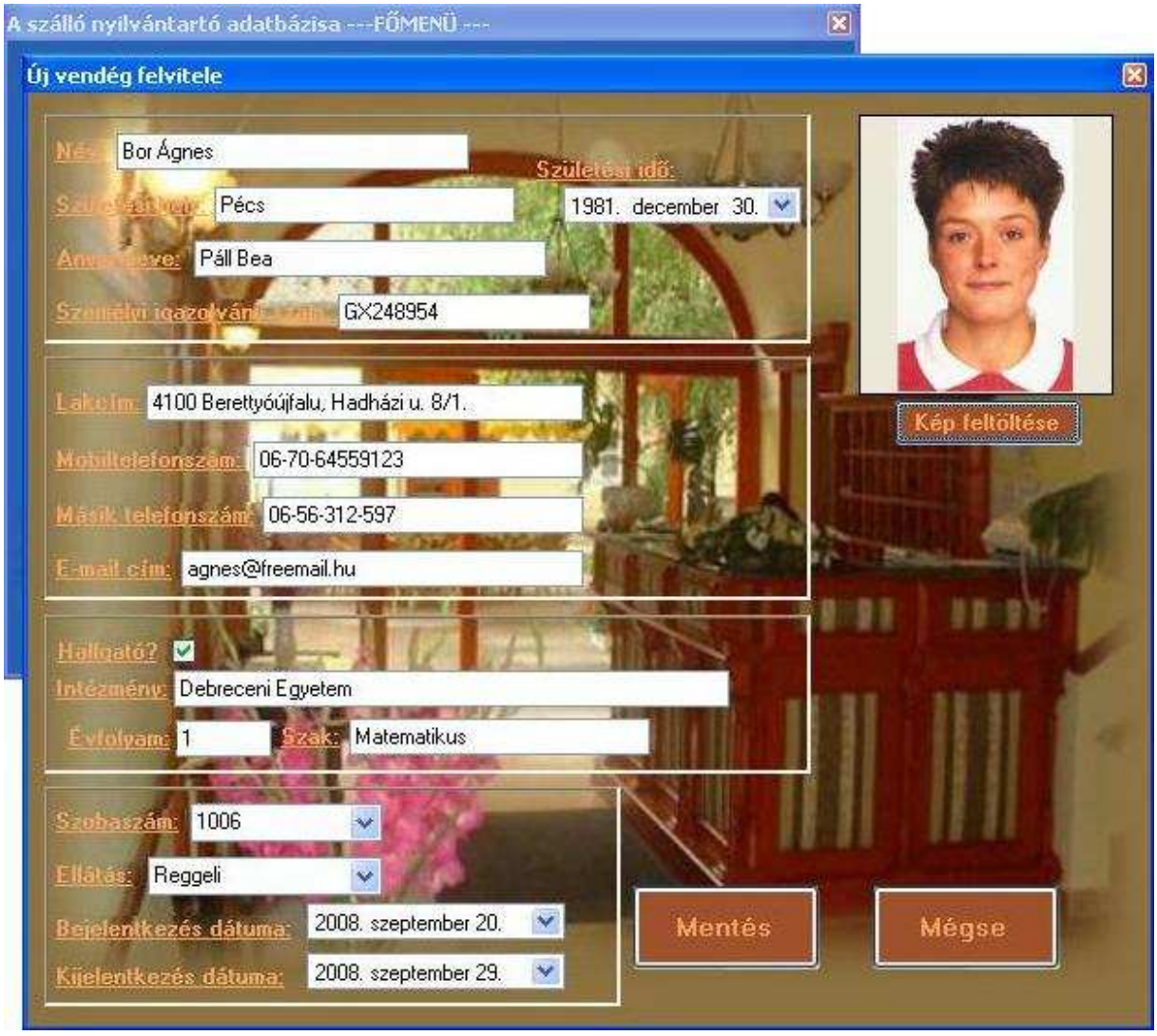
- Szobaszám,
- Ellátás,
- Bejelentkezés dátuma,
- Kijelentkezés dátuma.

Az itt kitöltendő mezők közül a Szobaszám kiválasztására egy legördülő lista szolgál, mely csak az üres szobákat tartalmazza. Alatta, az Ellátással kapcsolatban, szintén egy legördülő lista segít a gyors kiválasztásban. Majd a Bejelentkezés dátum és a Kijelentkezés dátum

beállítására, a már korábban említett naptár belsőprogram segít a munkánk könnyítésében, ahogy a Születési Idő esetében is.

Lehetőség van *fénykép* feltöltésére is, ami a további azonosítást szolgálhatja, de ennek igénybevétele nem kötelező.

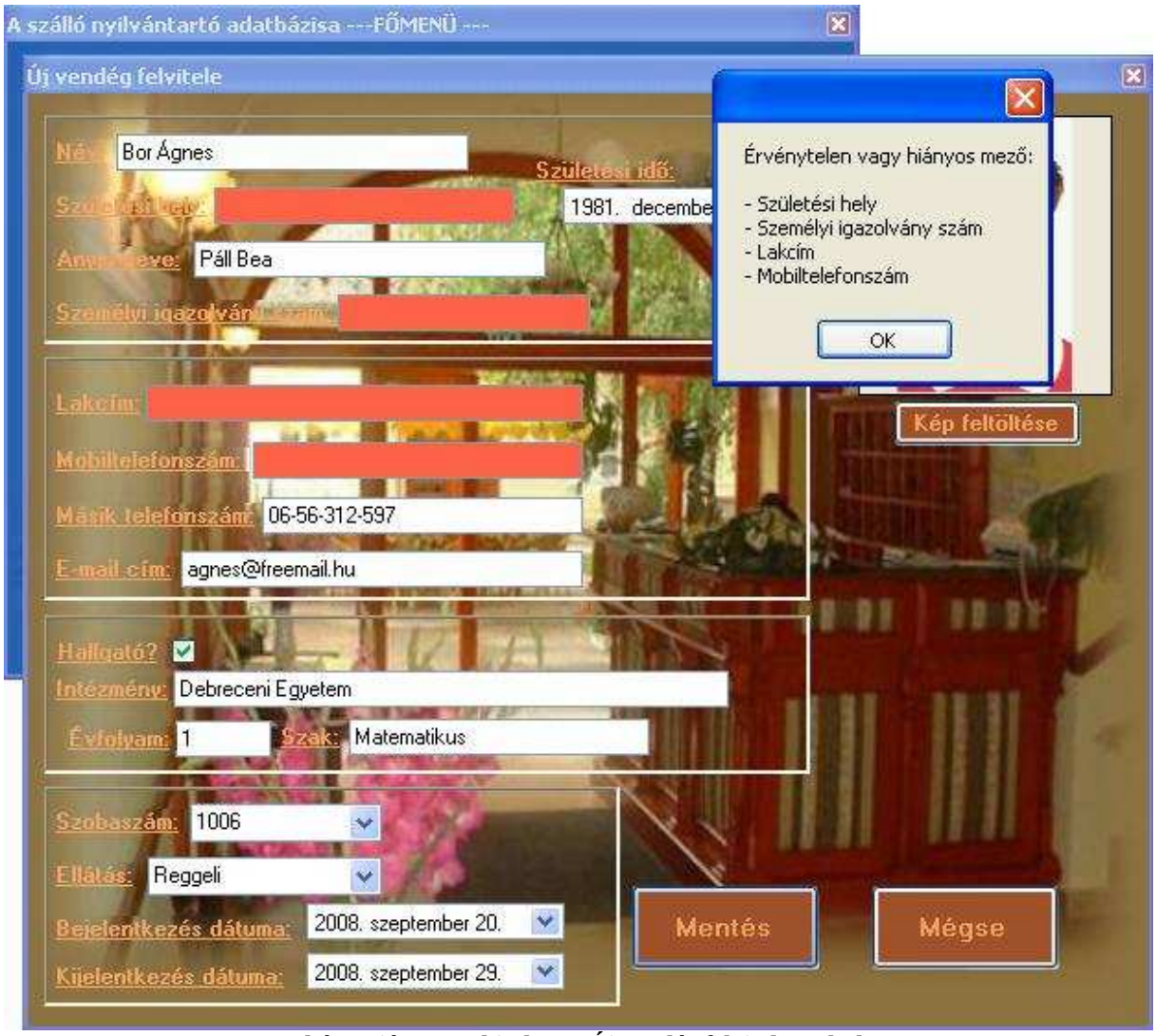
Egy teljesen kitöltött vendég adatlap a következőképpen néz ki:(11. kép)



11. kép: „Új vendég felvitele” adatlap kitöltve

Természetesen a Hallgató rész (3. panel) kitöltése opcionális, amennyiben a Vendég nem tanulója egy oktatási intézménynek sem. Viszont, ha ezt a részt megjelöljük (vagyis „pipát teszünk a Hallgató rész kis négyzetébe”), akkor a program kötelez minket az „Intézmény”, „Évfolyam” és „Szak” mezők kitöltésére is.

Amennyiben valahol hiányosan hagytunk egy vagy több mezőt, ott a program figyelmeztet a korrekcióra és jelzi a hibás vagy hiányosan kitöltött mezők helyét is:(12. kép)



12. kép: Hiányosan kitöltött „Új vendég felvitele” adatlap

Vendég adatbázis

A főmenüben(9. kép) a „Vendég adatbázis” gombra való kattintással nyithatjuk meg az adatbázisban helyet foglaló Vendégek listáját egy táblázatban. A táblázat az alábbi információkat szolgáltatja:(13. kép)

- Vendégkód,
- Név,
- Szobaszám.

Alapértelmezetten a táblázat sorai a Vendégkód oszlop szerint vannak rendezve.

13. kép: Vendég adatbázis

Rendezés és szűrés

Elsőként a *rendezést* ismertetem. Lehetőség van Vendégkód, Név vagy Szobaszám alapján rendezni az adatokat. Ezekre a gombokra történő kattintással érhetjük el azt, hogy a táblázat sorai a kívánt szempont szerint legyenek sorrendbe rakva.(13. kép)

A megszokott „A→Z” rendezés mellett a program képes a fordított sorrendű „Z→A” rendezésre is. Ehhez csupán az előző gombra kell ismét kattintani és a sorrend megfordul.

A *szűrést* a táblázatban szereplő vendégeken végezhetjük el. Név alapján vagy Szobaszám alapján hajthatunk végre szűrést, miután megadtuk a szűrendő betűt, számot, szót, kifejezést a Szűrés mezőben. Pl.: a vendégek között keresve a Dóra nevűeket, hajthatunk végre egy szűrést, mellyel csak azon vendégek lesznek kilistázva, akiknek a nevükben szerepel az, hogy „Dóra”.(14. kép)

A „Szűrés bekapcsolása” feliratú gombra történő kattintással indítjuk el a szűrést a táblázatban. A gomb felirata a „Szűrés kikapcsolás”-ra vált át, ha éppen szűrést hajtunk végre. Ekkor a gombra történő kattintásra a szűrés előtti, eredeti táblázat jelenik meg. A szűrés lényege, hogy megkönnyíti a keresést a táblázat sorai között a nagy adatbázisok esetén.

14. kép: Vendég adatbázis – Szűrés

Vendég adatainak módosítása

Lehetőség van a vendégek adatainak módosítására és törlésére. Első lépésben meg kell határozni, hogy melyik vendégnek az adatait akarjuk módosítani. Ehhez elegendő csupán kijelölni azt a sort vagy cellát (vagy belekattintani), amelyik a vendég adatait tartalmazza. Ekkor az ablak alsó részén a Vendégkód mezőben kiírásra kerül a módosításra szánt vendég Vendégkódja, mely alapján megfelelően azonosíthatóvá válik a vendég. Majd a „Módosítás” feliratú gombra kattintva egy új ablakban megnyílik a kiválasztott vendég adatlapja.(15. kép) A vendég adatlapján elhelyezkedő mezőkben hajthatunk végre módosításokat az adatain. Ezek a módosítások csak akkor kerülnek rögzítésre, ha a „Mentés” gombra kattintunk. A „Mégse” feliratú gombra történő kattintás esetén mentés nélkül visszajutunk a főmenübe.

15. kép: Vendég adatlapja

Vendég törlése

A Vendég adatbázis ablak(13. kép) alsó részén található „Törlés” feliratú gombra kattintva töröljük az adatbázisból a táblázatban aktuálisan kijelölt vendéget. A táblázatban a vendég kijelölésének menete már korábban ismertetve lett.

Statisztikai rész

A főmenüben(9. kép) a „Kiszolgálási elemzés” feliratú gombra kattintva megnyílik a szállóhely kihasználtsága ablak, amely egy kijelölt időintervallumnak a vendégekre vonatkozó helyfoglaltságát segít elemezni.(16. kép) Az időszak kijelölésére egy naptár belsőprogram nyújt megfelelő segítséget. A kezdő és vég dátum beállítására szabott feltétel alapján nem lehet a kezdő dátum > vég dátum. Vagyis a program kiszűri a hibásan megadni kívánt dátum sorrendet és a „Probléma a dátumokkal” szövegű hibaüzenettel válaszol.

A kezdő és vég dátum esetén az egyenlőség megengedett, ezt egy napra vonatkozó kimutatásként veszi. A dátumok kijelölése után „A szállóhely időszaki kihasználtsága” gombra kattintva a program megjeleníti az adott időszak napi lebontását. Kiírásra kerül minden adott napra vonatkoztatva a szállóban lakó vendégek száma, és azon belül, amennyiben van köztük hallgató, azok száma is. Az ablak alsó felén pedig megjelenítésre kerül a szálló teljes kihasználtsága a kijelölt időszakban.

16. kép: A szállóhely kihasználtsága

3. Fejlesztői dokumentáció

A fejlesztői dokumentáció fejezetben nem csak a program részletes, kódbeli bemutatásával ismertetem meg az olvasót, hanem a szoftver működtetésének különböző fajtáiból eredő teszteseteivel is. A műszaki leírás során igyekszem kitérni minden apró belső részletre, amely a program jelenlegi felépítéséért és működéséért felel. A tesztelés résznél pedig képekkel és példákkal szeretném illusztrálni a program adatbázissal való megfelelő kapcsolatát és használatának majdnem minden lehetőségét.

Program működésének műszaki leírása

A program elindításakor egy Üdvözlő ablak jelenik meg.(7.kép) Ennek az ablaknak a célja, hogy köszöntse a felhasználót, és képi elemeivel utaljon a program témájára. Az ablak nem rendelkezik semmilyen ablakot vezérlő eszközzel, nem lehet kicsinyíteni, nagyítani, tálcára lehúzni, azért, hogy ne tévessze meg a felhasználót, eredeti rendeltetésétől. Az üdvözlő ablakot két megjelenítő alkotja, melyek mindegyike csupán látvány elemként szolgál, viszont fontosságuk a programról alkotható összkép pozitív benyomásában rejlik.

Form0 / Form0.Designer.cs / InitializeComponent() eljárás kódrészlete:

```
this.pictureBox1 = new System.Windows.Forms.PictureBox();
this.label1 = new System.Windows.Forms.Label();
// pictureBox1
this.pictureBox1.Image =
((System.Drawing.Image)(resources.GetObject("pictureBox1.Image")));
this.pictureBox1.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.StretchImage;
// label1
this.label1.BackColor = System.Drawing.Color.Transparent;
this.label1.Font = new System.Drawing.Font("Arial", 21.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
this.label1.ForeColor = System.Drawing.Color.Gold;
this.label1.Text = "Ü D V Ö Z Ö L J Ü K";
```

Az üdvözlő ablakon való kattintás esetén a Form0 / Form0.cs / Click() metódus kódrészlete:

```
private void form_udv_Click(object sender, EventArgs e)
{
Close();
}
```

Ez előbbi kódrészlet miatt az Üdvözlő ablak nem csak a jobb felső sarokban lévő piros X-re kattintva zárható be, hanem az ablakon való kattintás következtében meghívott *Click()* metódus is felel az ablak zárásáért.

Az üdvözlő ablak bezárását kezdeményezve a program tényleges elindulása fog zajlani, amely már a program főmenüjének betöltődése előtt, a *form_menu_Load()* metódusban elkezdődik. Tartalma egy *try-catch()* blokkban van közrezárva. Először az adatbázis helyét tartalmazó fájlból próbálja meg kiolvasni az abszolút elérési utat a *BinaryReader* osztály nyújtotta lehetőségekkel:

```
BinaryReader br = new BinaryReader(new FileStream(url, FileMode.Open));
```

Megfigyelhető a br-t létrehozásakor, hogy elérési módként csak Olvashatóságot definiálok, ennek segítségével pedig a

```
db_url = br.ReadString();
```

sor hatására a db_url változó a korábban elmentett elérési utat fogja tárolni szövegként. Az elérési utat pedig a programban az adatbázis kapcsolatok létrehozásakor használom fel, úgy hogy átadom minden ablaknak ezt az értéket. Ennek az abszolút útnak a helyességét a *Check_adatb()* metódussal ellenőriztetem. Itt ellenőrzi a program az adatbázis hibátlan elérhetőségét és figyelmeztet amennyiben szükséges annak javítása, az útvonal ellenőrzése mellett egy csatlakozási kísérletet is végez az adatbázis irányába. Ha nem megfelelő a korábban elmentett útvonal, akkor nekünk kell kijelölni azt a következő sorok segítségével:

```
string FileName = null;
OpenFileDialog ofn = new OpenFileDialog();
ofn.Filter = "Database Files (*.mdb)|*.mdb|All Files (*.*)|*.*";
if (ofn.ShowDialog() == System.Windows.Forms.DialogResult.OK)
      db_url = ofn.FileName;
```

Az *OpenFileDialog* nevű osztály segítségével létrejött **ofn** vezérlőelem Filter tulajdonságának megadtam, hogy milyen kiterjesztésű állományok között szeretnék böngészni, a *FileDialog ShowDialog()* metódusával pedig megnyithatjuk az állomány kijelölésére alkalmas dialógus ablakot. Amikor a metódus visszatér, megkapjuk a *FileName* tulajdonságát, ha a felhasználó rákattintott az Open (Megnyitás) gombra. Az **ofn** *FileName* tulajdonsága az aktuálisan kijelölt fájl neve, amit elmentünk a string típusú, publikus db_url változónkba.

Ezután a program főmenüjének ablaka nyitódik be az alábbi sor hatására:

```
Application.Run(new form_menu());
```

A főmenüben(9. kép) olvasható a program által jelenleg használt adatbázis és a készítő neve. Az ablakban 4 gomb helyezkedik el, melyek közül a felső 3 az alábbi ablakokba irányítanak tovább:

⇒ Új vendég felvitele,

⇒ Vendég adatbázis,

⇒ Kiszolgálási elemzés.

Az alsó gomb segítségével, melynek a felirata „Adatbázis kijelölése”, lehet megváltoztatni a jelenleg használt adatbázist.

A gombokra kattintás eseményeként az alábbi blokkok érvényesek:

Új vendég felvitele gomb:

```
private void button1_Click(object sender, EventArgs e)
        {
            form_uj form = new form_uj();
            form.db_url = db_url;
            form.Show();
        }
```

Vendég adatbázis gomb:

```
private void button2_Click(object sender, EventArgs e)
        {
            form_adatb form = new form_adatb();
            form.db_url = db_url;
            form.Show();
        }
```

Kiszolgálási elemzés gomb:

```
private void button3_Click(object sender, EventArgs e)
        {
            form_statistic form = new form_statistic();
            form.db_url = db_url;
            form.Show();
        }
```

Ezen gombok Click() metódusainak tartalmát tekintve az általuk nyíló ablakok származtatása folyik, mely a Show() eljárás segítségével kerül megjelenítésre és az újonnan nyitott form db_url lokális változója felveszi a jelenlegi, *Főmenü_form* db_url lokális változó értékét. Erre azért volt így szükség, mivel ezek az ablakok a program futása közben kerülnek deklarálásra és kódolás közben még nem látják a globális változókat sem. Csak úgy oldható meg az

értékátadás, ha közvetlenül létrehozásuk után adom át a helyi változójának. Ez akkor lehet végzetes hiba, ha a *Főmenü_form* db_url lokális változója nem tartalmaz akkor még semmilyen értéket, de ezt kikerülve már korábban ellenőriztetem.

Az „Adatbázis kijelölése” feliratú gomb Click() metódusának tartalma az Open_adatb() eljárást hívja meg, amely már korábban ismertetésre került az adatbázis általunk történő kijelölése kapcsán.

Az „**Új vendég felvitele**” ablak betöltődésekor az alábbi sorok parancsai hajtódnak végre:

```
DateTime maidatum = DateTime.Now.ToUniversalTime();
```

A mai dátum meghatározása után azt a naptár belsőprogram kezdőértékként veszi fel:

```
            dateTimePicker1.Value = maidatum;
            dateTimePicker2.Value = maidatum;
            dateTimePicker3.Value = maidatum;
```

Azért kapnak így értéket, hogy elkerüljük a Tipus eltérési hibát:

```
            txb_intezmeny.Text = "0";
            txb_evf.Text = "0";
            txb_szak.Text = "0";
```

Jelszavas kapcsolódás az adatbázishoz, az elérési út pedig a db_url-ből olvasva:

```
OleDbConnection myConnection1 = new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Jet OLEDB:Database
Password='" + password + "'; Data Source = '" + db_url + "'");
```

Az alábbi sorok hatására a szobákat tartalmazó combobox vezérlőhöz, csak a szabad szobák kerülnek átadásra:

```
string SzobakCommand="SELECT * FROM Szobak " + "WHERE Foglalt LIKE False";
//Egy DataAdapter létrehozása azért, hogy a SzobakCommand által
//tartalmazott SQL utasítást és a myConnection1 OleDb kapcsolatot
//kezelhessük.
OleDbDataAdapter SzobakAdapter = new OleDbDataAdapter(SzobakCommand,
myConnection1);
//DataSet létrehozása:
DataSet dset = new DataSet();
//Elkérni a gyűjteményt, melyet a főtábla szolgáltat a forrás tábla és a
//DataTable között:
      SzobakAdapter.TableMappings.Add("Table", "Szobak");
//A SzobakAdapter nevű data adapter objektum Fill() tulajdonságával
```

```
//feltöltjük a DataAdapter-t a DataSet vagy DataTable objektum adataival,
//amiket a SELECT parancs ad vissza:
      SzobakAdapter.Fill(dset);
//a combobox adatkötéséhez a dset nevű DataSet.DefaultViewManager
//tulajdonságát használjuk, azt adva a combobox adatforrásának:
            DataViewManager dviewmanager = dset.DefaultViewManager;
            cobox_szoba.DataSource = dviewmanager;
//a combobox-ban megjelenítjük "Szobak.Szobaszám" mező értékeit:
            cobox_szoba.DisplayMember = "Szobak.Szobaszám";

//a használt objektumok felszabadítása:
            myConnection1.Dispose();
            SzobakAdapter.Dispose();
            dset.Dispose();
```

Ezáltal a vendég felviteli ablakban a szobaszámokat tartalmazó combobox vezérlő már fel lett töltve a szabad szobák számait reprezentáló adatokkal. Tehát ennek eléréséhez egy jól megírt SQL utasítás és a megfelelő adatbázis kapcsolat összerendelése szükséges, majd pedig az így kapott adattábla sorait átadva, csak a számunkra hasznos mező értékeit felhasználva mentjük azokat a vezérlő adatainak gyűjteményébe.

Az adatok kitöltése egyszerűen, az adatlapon elhelyezett „TextBox”-okba írva lehetséges, ez adatbázis kapcsolat nélkül folyik, mindaddig, amíg a felhasználó a „Mentés” feliratú gombra nem kattint. Ekkor egy vizsgálat hajtódik végre az összes adaton, mely kiszűri, hogy hol nincs, vagy hibásan van megadva a várt érték. A *felvitel_ok()* logikai értéket visszaadó függvényt felhasználva vizsgálódunk, és amennyiben hiba van, ott kijelezzük a következő sorok segítségével:

```
string hiba = null;
if (txb_nev.TextLength == 0)  //ha üres a txb_nev TextBox tartalma
            {
              txb_nev.BackColor = Color.Tomato;      //színezzük a vezérlőt
              hiba += "- " + lbl_nev.Text.Trim(':') + "\n";
//a hiba szövegéhez hozzáadjuk a kitöltendő adatmező nevét,
// „:” nélkül, gondolatjellel kiegészítve =>pl.: -Név
            }
```

A *hiba* nevű string változó fogja tartalmazni a hiányosan kitöltött mezők neveit. Ezt a feltételt minden adatmezőre meghívjuk, azután pedig a következő néhány sort futtatjuk:

```
    if (hiba == null) //Ha nincs hiba
        return true; //mehetünk tovább a mentésre
```

```
    else //Ha van olyan mező, amelyet hiányosan töltöttek ki
       {
       MessageBox.Show("Érvénytelen vagy hiányos mező:\n\n" + hiba);
       return false; //szükség van még javításra
      }
```

Tehát a korábban említett „Mentés” gomb Click() metódusa ezt a két sort tartalmazza:

```
if (felvitel_ok())  //ha minden mező megfelelően ki van töltve, akkor
   Ment();              //mentjük az adatokat az adatbázisba:
```

A „*Ment()*” eljárás feladata, hogy megnyissa az adatbázissal a kapcsolatot és a helyesen kitöltött adatlap minden mezőjéből kiolvasva az értékeket, azokat mentse az adatbázis „Vendeg” tábla következő sorába.

A következő sorokban magyarázattal ellátott kódrészleteket mutatok be a Ment() eljárásból.

A három DateTime típusú változó sorra kiolvassa a hozzájuk tartozó naptár belsőprogram segítségével kiválasztott értékeket, melyek később kerülnek felhasználásra:

```
            DateTime szulido = dateTimePicker1.Value;
            DateTime bejelentdatum = dateTimePicker2.Value;
            DateTime kijelentdatum = dateTimePicker3.Value;
```

A korábban már ismertetett *OleDb* kapcsolat létrehozása az adatbázissal és később, a *VendegAdapter*-ben a *VendegCommand* SQL paranccsal való összerendelése:

```
//jelszóval védett kapcsolódás az adatbázishoz, az elérési út pedig a
//db_url-ből olvasva:
OleDbConnection myConnection2 = new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Jet OLEDB:Database
Password='" + password + "'; Data Source = '" + db_url + "'");
```

Következőleg a *VendegCommand* SQL utasítás megadását ismertetem. Az *INSERT* parancs után a „Vendeg” táblát megadva egy zárójelben felsorolom azon mezőket, amelyekben az értékadást hajtjuk majd végre. Utána ugyanilyen sorrendben az értékeket:

```
string VendegCommand = "INSERT into
Vendeg(Kép,Név,SzemIgSzám,SzületésiHely,SzületésiIdő,AnyjaNeve,Lakcím,Mobil
Telefon,MásikTelefonszám,EmailCím,Hallgató,Intézmény,Évfolyam,Szak,Szobaszá
m,Ellátás,BejelentkezésiDátum,KijelentkezésiDátum) " + " VALUES
('" + kep + "','" + txb_nev.Text + "','" + txb_szemig.Text + "','" +
txb_szulhely.Text + "','" + szulido + "','" + txb_anyja.Text + "','" +
txb_lakcim.Text + "','" + txb_mobil.Text + "','" + txb_telefon.Text + "','"
+ txb_email.Text + "','" + hallgato + "','" + txb_intezmeny.Text + "','" +
txb_evf.Text + "','" + txb_szak.Text + "','" +
cobox_szoba.GetItemText(cobox_szoba.SelectedItem) + "','" +
```

```
cobox_ellatas.GetItemText(cobox_ellatas.SelectedItem) + "','" +
bejelentdatum + "','" + kijelentdatum + "')";
```

Az itt felsorolt vezérlők által tartalmazott értékek kerülnek majd mentésre az adatbázisban.

A *VendegCommand* SQL parancs és *myConnection2* kapcsolat összerendelése a *VendegAdapter* segítségével:

```
OleDbDataAdapter VendegAdapter = new OleDbDataAdapter(VendegCommand,
myConnection2);
```

Ezek után a *VendegAdapter.Fill()* metódusát használjuk a *dataSet*, adatokkal való feltöltésére.

Amikor valaki beköltözik egy szobába, akkor azt a szobát foglalttá kell tenni, amit a következő kódrészlet segítségével hajtottam végre:

```
string SzobaUpdateCommand = "UPDATE Szobak "
      + " SET Foglalt=True"
      + " WHERE Szobaszám LIKE '" +
cobox_szoba.GetItemText(cobox_szoba.SelectedItem) + "'";
OleDbDataAdapter SzobaUpdateAdapter = new
      OleDbDataAdapter(SzobaUpdateCommand, myConnection2);
SzobaUpdateAdapter.Fill(dataSet);
```

A *SzobaUpdateCommand* SQL utasítássorban először megadtam az *UPDATE* parancsot, a frissítés elérése miatt, majd a szoba foglaltságának beállításához a *SET Foglalt = True*-t. Végül feltételként a Szobaszámát, ami azonosítja a szobát az adatbázis „Szobak” táblájában. A fentebb már ismertetett *OleDbDataAdapter* osztályt használva rendelem össze az SQL utasítást és a kapcsolatot egy *DataAdapter*-ben, amelynek segítségével feltöltöm a *dataSet*-et.

A következő bemutatásra szánt, „**Vendég adatbázis**” ablak működésének műszaki hátterét fogom ismertetni. Ez a *Form* több vezérlőből és megjelenítőből áll, amelyek egy részével már az ablak betöltődésekor kezdő értékadási műveletek hajtódnak végre. Az ablak közepén elhelyezkedő megjelenítő objektum, a DataGridView-hoz egy Load metódust párosítva végzem el annak adatokkal való feltöltését.(13. kép) A Load() eljárásban kiépítem az adatbázissal a kapcsolatot és beállítom a megjelenítő eszközhöz tartozó adatforrást:

```
this.dataGridView1.DataSource = this.dbDataSet1.Vendégek;
```

A *DataGridView1* adatmegjelenítő objektum adatforrásaként a „Vendégek” lekérdezést állítom be, amely a vendég

- kódját,
- nevét,

- szobaszámát

foglalja egy táblába össze.

A *DataGridView1* megjelenítőn különböző megjelenési és felsorolási lehetőségeket készítettem a program használójának.

Elsősorban a *Rendezés* bemutatására a következő kódrészlet szolgál:

```
DataGridViewColumn sortColumn = dataGridView1.Columns[0];
ListSortDirection direction;
//irány meghatározása:
sort1 *= -1;      //+1,-1,+1,-1,+1,-1,...
if (sort1 == 1)
      direction = ListSortDirection.Ascending;
else
      direction = ListSortDirection.Descending;

dataGridView1.Sort(sortColumn, direction);
```

Az első sorban a *DataGridViewColumn* osztály tulajdonságának adom meg azt az oszlopot, ami szerint akarom a rendezést végrehajtani, jelen esetben a „Vendégkód” oszlop szerint. A következő sorban a *ListSortDirection* enum osztályt felhasználva fogom a „direction”-nal definiálni azt, hogy majd később milyen rendezési irányt választok. Az irány változtathatóságát a „sort1” változó segítségével teszem meg, a számértékétől pedig maga az irány fog majd függeni. Pozitív „sort1” érték esetén a rendezés iránya növekvő, negatívnál csökkenő lesz. Rendezés végrehajtásáért pedig az adatmegjelenítő *Sort()* metódusa fog felelni, amely argumentumainak a „sortColumn” és „direction” változókat várja.

A 3 különböző rendezési típus közötti eltérés a kódjukban csupán a *DataGridViewColumn* oszlop kijelölésénél van.

A következő funkció, mellyel a „Vendég adatbázis” Form-ot láttam el a Szűrés művelete. Itt a „txb_filter” nevű TextBox-ba megadott karakter vagy karaktersorozat alapján történik a szűrés, amely működéséhez az alábbi kódrészlet tartozik:

```
      string searchString = txb_filter.Text;
      DataView dv = new DataView(this.dbDataSet1.Vendégek, "", "",
DataViewRowState.CurrentRows);
//Szűrés a Név alapján
      if(rbtn_name.Checked)
            dv.RowFilter = "Név LIKE '%" + searchString + "%'";
//Szűrés a Szobaszám alapján
```

```
        if(rbtn_room.Checked)
        dv.RowFilter = "Szobaszám LIKE '%" + searchString + "%'";
dataGridView1.DataSource = dv;
```

A „searchString” nevű szövegváltozó fogja tartalmazni a szűrendő feltételt. A *DataView* osztály segítségével a „dv”-t a szűrés kimenetének adatforrásaként fogom használni.

Amennyiben név alapján hajtjuk végre a szűrést, úgy a „Név” mezőben lévő adatokat hasonlítjuk a „searchString”-el a *Név LIKE* parancs segítségével, a szobaszám esetén pedig a „Szobaszám” mezőben lévőket. Az utolsó sorban a „dataGridView1” megjelenítő adatforrásaként megadjuk a „dv” *DataView*-t, ezzel végrehajtva a szűrést. Visszatérni az eredeti, „teljes nézetes” táblához következő néhány sor segítségével lehet:

```
//nincs szűrés, vagyis mutassa az összes vendéget
DataView dv = new DataView(this.dbDataSet1.Vendégek, "", "",
DataViewRowState.CurrentRows);
dataGridView1.DataSource = dv;
```

Mivel nem adtunk meg szűrési feltételt és a *DataView.RowFilter* tulajdonságának sem adtunk meg oszlopot, ami szerint szűrni kellene, így azt alapértelmezettként veszi.

A *DataGridView1* megjelenítő komponensen történő cella, vagy sor kijelölése esetén az adott sor által tartalmazott vendég „Vendégkódja” kerül kiolvasásra, ennek megvalósításáért az alábbi kódblokk felel:

```
{
int rowindex = e.RowIndex;    //az „e” egy paraméterként kapott sor
string cellValue = dataGridView1["Vendégkód", rowindex].Value.ToString();
txb_vendegkod.Text = cellValue;
}
```

A „Vendég adatbázis” ablak utolsó két gombjának „Click” eseményének ismertetése előtt szükség volt a „Vendégkód” kiolvasására szolgáló kódrészlet bemutatására, mivel a „Módosítás” és „Törlés” funkciók működésének alapja a „Vendégkód” helyes meghatározása. Előbb a módosítás program szinti műveletének ismertetését végzem el.

Egy vendég adatainak módosítása a vendég meghatározása után a *Módosít* gombra kattintva a *btn_modify_Click()* metódusnál kezdődik el.

```
private void btn_modify_Click(object sender, EventArgs e)
        {
                form_adatlap form = new form_adatlap();
                form.Vendegkod = Vendegkod_szam;
                form.db_url = db_url;
```

```
                form.Show();
                Close();
        }
```

Az első sorban, létrehozom a vendég adatait tartalmazó *form*-t, amelynek a vendég adatait tartalmazó mezőiben lehet majd a módosításokat elvégezni.

Az előbb kiolvasott „Vendegkod” változó értékét átadva lehet beazonosítani a módosítandó vendéget, a „db_url” pedig az adatbázis elérési útját fogja tartalmazni. Az „adatlap_form” műszaki leírását a „Törlés” művelete után fogom ismertetni.

A törlés műveletét a már jól megszokott SQL utasítást és kapcsolat párt használva hajtom végre.

```
OleDbConnection myConnection1 = new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Jet OLEDB:Database
Password='" + password + "'; Data Source = '" + db_url + "'");
//a kiválasztott vendéget töröljük - azonosítás a vendég kód alapján, mivel
//elsődleges kulcsként szolgál
string VendegAdatokCommand = "DELETE FROM Vendeg " + "WHERE Vendégkód LIKE
'" + Vendegkod_szam + "'";
```

A törölt vendég szobáját üressé teszem, a szoba azonosítására pedig a „Vendégkód”-ot használom fel:

```
string RegiSzobaUpdateCommand = "UPDATE Szobak "
      + " SET Foglalt=False"
      + " WHERE Szobaszám LIKE '" + regiSzobaszam + "'";
OleDbDataAdapter RegiSzobaUpdateAdapter = new
OleDbDataAdapter(RegiSzobaUpdateCommand, myConnection1);
            RegiSzobaUpdateAdapter.Fill(dsetVendeg);
```

A „Módosítás” gomb Click() eseményének következtében újonnan megnyílt „**Vendég adatlapja**” ablak már a betöltődésekor a kiválasztott vendég adataival fog megjelenni, ennek ismertetéséhez egy kódrészletet illesztek be:

```
//kapcsolódás az adatbázishoz:
OleDbConnection myConnection1 = new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Jet OLEDB:Database
Password='" + password + "'; Data Source = '" + db_url + "'");
string VendegAdatokCommand = "SELECT * FROM Vendeg " + "WHERE Vendégkód
LIKE '" + Vendegkod + "'";
OleDbDataAdapter VendegAdatokAdapter = new
OleDbDataAdapter(VendegAdatokCommand, myConnection1);
```

```
DataSet dsetVendeg = new DataSet();
VendegAdatokAdapter.TableMappings.Add("Table", "Vendeg");
VendegAdatokAdapter.Fill(dsetVendeg);
//VENDEG Táblában lévő adatok kiolvasása:
//1,Kép elérési útjának kiolvasása, ha van:
pbox_kep.DataBindings.Add("ImageLocation", dsetVendeg, "Vendeg.Kép");
//és megjelenítése ezáltal.
//2,Név adatának kiolvasása a txb_nev-be
txb_nev.DataBindings.Add("Text", dsetVendeg, "Vendeg.Név");
txb_szemig.DataBindings.Add("Text", dsetVendeg, "Vendeg.SzemIgSzám");
txb_szulhely.DataBindings.Add("Text", dsetVendeg, "Vendeg.SzületésiHely");
dateTimePicker1.DataBindings.Add("Text", dsetVendeg,
"Vendeg.SzületésiIdő");
txb_anyja.DataBindings.Add("Text", dsetVendeg, "Vendeg.AnyjaNeve");
txb_lakcim.DataBindings.Add("Text", dsetVendeg, "Vendeg.Lakcím");
```

A kódrészlet első néhány sora az adatbázishoz való kapcsolódást és az SQL utasítás kiadását mutatja be, amit már korábban ismertettem, majd ezek *DataAdapter* segítségével történő összerendelése zajlik. Ezután a „VendegAdatokAdapter”-rel a „dsetVendeg” feltöltését követően az adatok kiolvasása folyik. A vendég adatával feltölteni kívánt komponens „DataBindings” tulajdonságát hívtam meg. A „DataBindings”-en belül az *Add()* metódussal végezhető el az adatkinyerése a függvény paraméterként többek között megadott „dsetVendeg”-ből. A függvény egyéb paramétereket is vár, amiknek segítségével tudja behatárolni a kívánt adatmezőt. Példaként megemlítve a vendég nevének kiolvasását az első paraméter az adat típusát, vagyis az adatbázisban lévő érték tulajdonságát jelenti, a „Vendeg.Név” adattag pedig az adatbázis „Vendeg” táblájának „Név” mezőjét.

A vendég adatain való módosítás hatására a módosított komponens érték az *UPDATE* SQL parancs segítségével kerül majd mentésre az adatbázisban, ennek formája a következő:

```
string VendegUpdateCommand = "UPDATE Vendeg " + " SET Kép='" + kep + "',
Név='" + txb_nev.Text + "', SzemIgSzám='" + txb_szemig.Text + "',
SzületésiHely='" + txb_szulhely.Text + "', SzületésiIdő='" + szulido + "',
AnyjaNeve='" + txb_anyja.Text + "', Lakcím='" + txb_lakcim.Text + "',
MobilTelefon='" + txb_mobil.Text + "', MásikTelefonszám='" +
txb_telefon.Text + "', EmailCím='" + txb_email.Text + "', Hallgató='" +
hallgato + "', Intézmény='" + txb_intezmeny.Text + "', Évfolyam='" +
txb_evf.Text + "', Szak='" + txb_szak.Text + "', Szobaszám='" +
jelenlegiSzobaSzam + "', Ellátás='" +
cobox_ellatas.GetItemText(cobox_ellatas.SelectedItem) + "',
```

```
BejelentkezésiDátum='" + bejelentdatum + "', KijelentkezésiDátum='" +
kijelentdatum + "' "
+ " WHERE Vendégkód LIKE '" + Vendegkod + "'";
```

A *SET* parancs után felsorolásra kerül az összes „Vendeg” tálba mező neve, amelyek sorai adatot tárolnak, és a hozzájuk tartozó komponensek értékei. Az SQL parancs végén pedig a vendég azonosítására az előző („form_adatb” nevű) *form*-tól kapott „Vendegkod” változó értékét használom.

A vendég másik szobába való átköltöztetésének lehetőségét is beépítettem a programba.

```
//Ha átköltözött másik szobába, akkor a régi szobaszámát felszabadítjuk:
//vagyis a régi szoba szabaddá tétele:
if(atkoltozott)
     if (regiSzobaszam !=
        Int32.Parse(cobox_szoba.GetItemText(cobox_szoba.SelectedItem)))
       {
       string RegiSzobaUpdateCommand = "UPDATE Szobak "
       + " SET Foglalt=False"
       + " WHERE Szobaszám LIKE '" + regiSzobaszam + "'";
       OleDbDataAdapter RegiSzobaUpdateAdapter = new
          OleDbDataAdapter(RegiSzobaUpdateCommand, myConnection2);
       RegiSzobaUpdateAdapter.Fill(dataSet);
       RegiSzobaUpdateAdapter.Dispose();
       }

//az új szoba foglalttá tétele:
string SzobaUpdateCommand = "UPDATE Szobak "
       + " SET Foglalt=True"
       + " WHERE Szobaszám LIKE '" + jelenlegiSzobaSzam + "'";
OleDbDataAdapter SzobaUpdateAdapter = new
       OleDbDataAdapter(SzobaUpdateCommand, myConnection2);
SzobaUpdateAdapter.Fill(dataSet);
```

Amennyiben a vendég átköltözik egy másik szobába, úgy a régi szoba számát ideiglenesen eltárolom, és később felhasználom arra, hogy azt a szobát szabaddá tegyem. Az SQL utasítások során azért használok *UPDATE* parancsot, mert ennél a résznél új adat nem fog érkezik, csupán meglévő adatok frissítésére kell számolni.

Az adatbázisba történő felvitel előtt itt is használok „felviteli ellenőrzést”, amit a „felvitel_ok()” nevű függvény segítségével kezelek le, az „Új vendég felvitele form”-hoz hasonló módon.

A szoftver működésének műszaki leírását a „**Statisztikai rész”** bemutatásával zárom, ahol kiszolgálási elemzést hajt végre a program, a szállóhely szobáinak kihasználtságát illetően. Az ablak betöltődésekor a komponensek inicializálását követően a mai dátumot veszik fel kezdőértékként a naptár belső programok, amelyek a megfelelő dátum kijelölésében segítenek a felhasználónak. A következő kódblokk ezt mutatja be.

```
{
      InitializeComponent();
      DateTime maidatum = DateTime.Now.ToUniversalTime();
      dateTimePicker1.Value = maidatum;
      dateTimePicker2.Value = maidatum;
}
```

Az vizsgált időszak határait alkotó dátumok megfelelő kijelölése után a „szállóhely időszaki kihasználtsága” feliratú gombra kattintva elindul az elemzés. Átvizsgálva a köztes napokat, meghatározásra kerül a vendégek száma minden adott napra nézve.

```
//Azon Vendégek száma, akiknek a BejelentkezésiDátum és KijelentkezésiDátum
//közé esik az adott dátum:
string DatumCommand = "SELECT COUNT(Név) FROM Vendeg WHERE '" + datum + "'
BETWEEN BejelentkezésiDátum AND KijelentkezésiDátum";
myConnection1.Open();
OleDbCommand cmdCount = new OleDbCommand(DatumCommand, myConnection1);
object objCounted = cmdCount.ExecuteScalar();
int objToInt = Convert.ToInt32(objCounted);
//a kiválasztott dátum hány embernek esik a Bejelentkezési és
//Kijelentkezési dátuma közé:
emberek = objToInt;
```

A vendégek közül a hallgatók száma is külön kiírásra kerül.

```
//Azon Vendégek száma, akiknek Hallgatóként a BejelentkezésiDátum és
//KijelentkezésiDátum közé esik az adott dátum:
string HallgatoCommand = "SELECT COUNT(Név) FROM Vendeg WHERE '" + datum +
"' BETWEEN BejelentkezésiDátum AND KijelentkezésiDátum AND Hallgató =
TRUE";
OleDbCommand cmdCount2 = new OleDbCommand(HallgatoCommand, myConnection1);
object objCounted2 = cmdCount2.ExecuteScalar();
int objToInt2 = Convert.ToInt32(objCounted2);
//azokból az emberekből mennyi a hallgató:
hallgatok = objToInt2;
```

Az SQL utasításban a *COUNT()* parancsot használva lehetséges a feltételnek eleget tevő mezők darabszámát lekérdezni. A már megszokott *DataAdapter* helyett ebben az esetben az *OleDbCommand* osztály segítségével lehet egy lekérdezéssé összerendelni az SQL parancsot a kapcsolattal, majd ennek az *ExecuteScalar()* tulajdonságával hívható meg a lekérdezett darabszám. Ennek az *object* alaptípusként lementett értéknek a számmá konvertálása a *Convert.ToInt32()* metódussal végezhető el.

A statisztikai adatok kiszámolásához a szobák darabszámát is meg kell határozni. Erre a következő lekérdezés szolgál:

```
string SzobakCommand = "SELECT COUNT(*) FROM Szobak";
OleDbCommand cmdCount4 = new OleDbCommand(SzobakCommand, myConnection1);
object objCounted4 = cmdCount4.ExecuteScalar();
int objToInt4 = Convert.ToInt32(objCounted4);
//ÖsszSzobaszám:
OsszSzoba = objToInt4;
```

A fentebb ismertetett *OleDbComman.ExecuteScalar()* tulajdonsággal a lekérdezett szobák összes darabszámát hívhatjuk meg és mentjük el *object* alaptípusként. Ezt egésszé konvertálás után felhasználhatjuk, mint *Integer*-t.

Az időszaki kihasználtság átlag kiszámolására a

```
NapiKihasznAtl = NapiKihasznAtl + ((double)emberek / (double)OsszSzoba);
double OsszKihasznAtl = ((double)NapiKihasznAtl / (double)napok_db) * 100;
```

parancsokat használom.

Kiírásnál az alábbi módon járok el:

```
//Az időszak átl.kihasználtsága: napi átl.kihasználtságok / napok db_száma
lbl_adat.Text = OsszKihasznAtl.ToString("0.##") + " %";
```

Az „lbl_adat” nevű komponens mutatja a felhasználó számára az időszak átlagos szállóhelyi kihasználtságát.

Az adatbázis eléréshez és kezeléshez a fejlesztés során minden esetbe a *System.Data.OleDb* névteret használtam fel, amelynek az osztályai nélkül nem lehetett volna megfelelően elérni és kezelni ADO.NET-ben az adatbázisokat.

Tesztelés

Adatbázis teszt

Az adatbázis teszt során a program által végrehajtott adatbázis műveleteket vizsgáljuk meg. Ellenőrizzük azokat az értékeket, amelyeket a program az adatbázisban ír vagy éppen módosít. Az adatbázisban lévő értékeket a program

⇒ Új vendég felvitelével,

⇒ a Vendégek adatainak módosításával vagy

⇒ a Vendégek törlésével

változtatja. Mind a három lehetőséget példán keresztül mutatom be.

Vigyünk fel ez adatbázisba egy új vendéget, ezt a Főmenü „Új vendég felvitele” gombjával tehetjük meg.(11. kép) Az adatlap megfelelő és hiánytalan kitöltése után a Mentés gombra kattintva a program rögzíti a vendég adatait az adatbázis „Vendeg” táblájának egy új sorában, a vendégkód egyel történő növelésével.(17. kép)

Vendeg : tábla

Vendégkód	Kép	Név	Személyigaz	SzületésiHely	Születésidő	AnyjaNeve	Lakcím	Mobiltelefon	Másik telefons	E-mail
1		Tóth Gyula	AS252769	Püspökladány	1985.03.02.	Gaál Magdolna	4150 Püspökladány, Petőfi u. 6.	06-20-4718762	06-54-453929	tgyula3@fre
2		Balogh Dóra	AB142356	Püspökladány	1988.10.25.	Horváth Krisztir	4150 Püspökladány, Zöldfa u. 8.	06-30-7778889	06-54-453-256	bdori@freem
3		Horváth Julian	AR532344	Debrecen	1983.06.04.	Kovács Emília	4150 Püspökladány, Kossuth u. 24.	06-30-6594889	06-54-452-745	hjulia@freer
4		Kiss József	NA157979	Berettyóújfalu	1972.03.21.	Nagy Anna	9600 Sárvár, Bem u. 9.	06-30-2182459	06-52-964-832	cdgbc@free
5		Nagy Imre	MS526749	Budapest	1983.06.09.	Szabó Enikő	2040 Budapest, Péterfia u. 10.	06-20-6723591	06-1-354-226	Nagyimre@
6	C:\igazolvá	Tóth Béla	TT245654	Debrecen	1981.12.02.	Kovács Bea	4100 Debrecen, Hadházi u. 12.	06-30-4539123	06-52-432-597	tbela@freem
7		Soha Ferenc	DE489512	Sopron	1980.04.06.	Kiss Nóra	7064 Győr, József A. u. 35.	06-20-3246987	06-60-632-123	soha@freem
8		Kádár Dóra	GY103245	Győr	1978.05.25.	Nagy Enikő	5700 Gyula, Deák F. u. 26.	06-20-6542987	06-42-321-594	dora@freem
9		Arany István	PR702254	Hajdúszoboszló	1979.09.22.	Nádházi Éva	3300 Eger, Derék u. 8.	06-30-9248316	06-31-694-523	pista@freem
10		Papp Emma	BE901103	Püspökladány	1975.06.30.	Inke Judit	7600 Pécs, Nagy S. u. 62.	06-30-2593164	06-72-698-156	emma@free
11		Győri Péter	GF829963	Budapest	1979.01.08.	Bő Dóra	7064 Győr, Kardos u. 26.	06-30-2136492	06-60-216-230	peti@freema
12		Nagy László	KS134569	Dombóvár	1988.11.30.	Horváth Edit	7064 Győr, Mókus u. 26.	06-30-2365419	06-60-312-265	laci@freema
13		Sélye Zsolt	WR448520	Pécs	1977.12.05.	Győri Vanda	4100 Berettyóújfalu, Erdő u. 68.	06-20-1654893	06-56-321-288	zsolti32@fre
14		Csáti Péterné	LJ456987	Szeged	1979.09.14.	Kiss Nóra	9600 Sárvár, Rét u. 6/2.	06-20-3456982	06-52-123-354	gioaelto213(
15		Nemes Emil	EI159357	Barcs	1979.12.03.	Nagy Petra	5700 Gyula, Petőfi u. 62.	06-70-2164523	06-42-326-012	emil20@free
16		Balogh Dóra	OL556874	Paks	1981.02.25.	Király Anett	4200 Paks, Zrínyi u. 42.	06-20-1265923	06-53-203-656	karcsi30@fr
17	C:\igazolvá	Tóth Géza	AP2463591	Debrecen	1983.12.16.	Szabó Etelka	4150 Püspökladány, Görgey u. 98.	06-20-866-48-78	06-54-453-896	tothgeza@g
18	C:\igazolvá	Bor Ágnes	GX248954	Pécs	1981.12.30.	Páll Bea	4100 Berettyóújfalu, Hadházi u. 8/1.	06-70-6455912	06-56-312-597	agnes@free

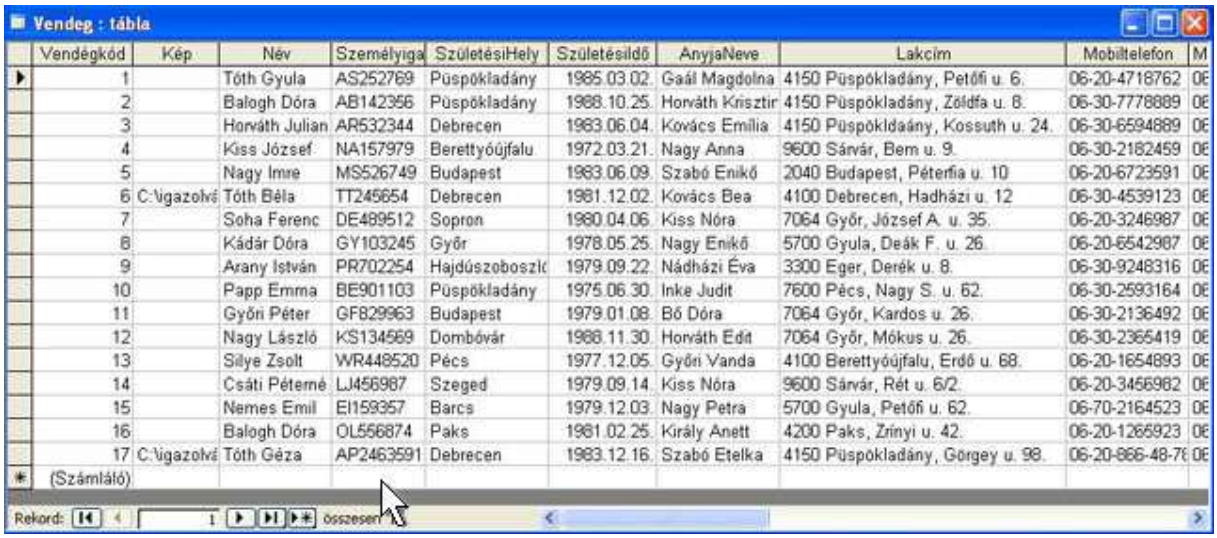
Rekord: 1 összesen

17. kép: „Vendeg” tábla Adat nézete

Nézzünk példát a Vendég adatainak módosítására is. Ennél az adatbázis tesztnél a program által, az adatbázisban tárolt vendég adatain történő módosítások életbe lépését vizsgáljuk. Fontos megjegyezni, hogy itt az adatok frissítése folyik. Ehhez a programban az adatelérési modell *UpdateCommand* nevű DataAdapter-t használom.
A főmenüben a „Vendég adatbázis” gombra kattintva nyithatjuk meg a vendégek listáját és az ott kiválasztott vendég adatain végezhetünk módosításokat. A módosítás gombra kattintva egy új ablakban megnyílik a kiválasztott vendég adatlapja. A vendég adatlapján elhelyezkedő mezőkben hajthatunk végre módosításokat a vendég adatain. Ezek a módosítások csak akkor

kerülnek rögzítésre, ha a „Mentés” gombra kattintunk.(15. kép) Ekkor a program az adatbázishoz csatlakozva frissíti a „Vendeg” tábla megfelelő sorát a megváltozott adatokkal.(17. kép)

Következő példában töröljünk egy vendéget a meglévő adatbázisból. Töröljük ki a korábban felvitt „Bor Ágnes” nevű vendéget. Első lépésben nyissuk meg a vendégek adatbázisát a főmenüben a „Vendég adatbázis” gombra kattintva. Majd válasszuk ki a példa által megjelölt „Bor Ágnes” nevű vendéget az adatbázisból és kattintsunk a „Törlés” feliratú gombra. Ekkor az adatbázisban ez a vendég véglegesen törlésre került.



Program teszt

A Program teszt alatt a program működése közben előforduló eseményeket és megfelelő adat módosulásokat értem. Külön figyelmet fordítva a programban minden olyan eseménynek, ami valami változást hajt végre a program működését illetően. Itt elsősorban olyan dolgokra gondolok, hogy az adott parancs kiadását követően a program mindig a várt eredményt hozza-e. Ennek a vizsgálatnak egy része már a korábbi „Adatbázis teszt” alfejezetben ismertetésre került a különböző adatbázis műveletek kapcsán, és már a Program teszt egy részét is bemutattam. Így most a kimaradt Program teszteseteket vizsgáljuk át.

Az adatbázis műveleteken kívül a program egy Kiszolgálási elemző résszel is rendelkezik. Itt nem végez az adatbázisban semmilyen módosító műveletet, csupán lekérdezést hajt végre a táblákon.

Kezdjük a tesztet azzal, hogy a főmenüben(9. kép) a „Kiszolgálási elemzés” feliratú gombra kattintva megnyitjuk a szállóhely kihasználtsága ablakot. Amely egy kijelölt

időintervallumnak a vendégekre vonatkozó helyfoglaltságát segít elemezni. Az időszak kijelölését a **„Felhasználói dokumentáció/Program használata/Statisztikai rész”** alfejezetben ismertetett módon hajtjuk végre. Vagyis a kezdő és vég dátum megfelelő beállítása után a „A szállóhely időszaki kihasználtsága” gombra kattintva a program megjeleníti az adott időszak napi lebontását, és kiírásra kerül minden adott napra vonatkoztatva a szállóban lakó vendégek és azon belül a hallgatók száma. Az ablak alsó felén pedig megjelenítésre kerül a szálló teljes kihasználtsága a kijelölt időszakban.(16. kép)
A példában bemutatott kiszolgálási elemzés során felhasznált adatbázis „Vendeg” táblájának részlete lentebb látható.(18. kép)

Vendeg : tábla

Hallgató	Intézmény	Évfolyam	Szak	Szobaszám	Ellátás	Bejelentkezés	Kijelentkezés
☑	Debreceni Egye	4	Matematikus	1001	Reggeli	2008.09.10.	2008.09.15.
☑	Debreceni Egye	2	Biológus	1002	Félpanzió	2008.09.21.	2008.09.28.
☐		0		1003	Reggeli	2008.10.01.	2008.10.14.
☐		0		1004	Teljes Pan	2008.10.05.	2008.10.08.
☐		0		1005	Félpanzió	2008.08.30.	2008.09.16.
☑	Debreceni Egye	1	Matematikus	1006	Reggeli	2008.09.20.	2008.09.29.
☐		0		1007	Ellátás Né	2008.09.11.	2008.09.22.
☐		0		1008	Teljes Pan	2008.10.10.	2008.11.05.
☐		0		1009	Reggeli	2008.09.20.	2008.09.27.
☐		0		1010	Félpanzió	2008.09.11.	2008.09.23.
☑	Debreceni Egye	1	Programozó	1011	Reggeli	2008.10.20.	2008.10.28.
☐		0		1012	Reggeli	2008.09.30.	2008.10.02.
☐		0		1013	Reggeli	2008.09.23.	2008.09.30.
☑	Debreceni Egye	3	Biológus	1014	Teljes Pan	2008.10.03.	2008.10.16.
☐		0		1015	Reggeli	2008.09.06.	2008.09.14.
☐	0	0	0	1016	Reggeli	2008.10.05.	2008.10.10.
☑	Debreceni Egye	3	Programozó	1017	Félpanzió	2008.11.14.	2008.11.22.
☐		0					

Rekord: 1 összesen 17

18. kép: „Vendeg” tábla részlete

4. Összefoglalás

Szakdolgozatomban, eddigi tapasztalataimra támaszkodva és irodalmi forrásokat felhasználva igyekeztem bemutatni elsősorban az ADO.NET osztálykönyvtárat. Programozói ismeretek elsajátítására szántam a .NET keretrendszer részeként nyilvántartott fejlesztő eszköz bemutatását, a Microsoft Visual Studio-t, melyet a szakdolgozatom elkészítéséhez használtam. Néhány mondatban próbáltam minél érthetőbben megismertetni az olvasóval a ma egyik legelterjedtebb programozási nyelvet, a C#-ot. Remélem kellő ismeretanyagot nyújtottam a szakdolgozatom alapját képező adatbázis rendszer megértéséhez és átlátásához. Igyekeztem rávilágítani arra, hogy mennyi lehetőség rejlik a programozásban és ebből milyen könnyű egy kicsit is elsajátítani kellő gyakorlással.

A szoftverem fejlesztése eredménye elérte célját, de nem érte el a végső fázist. Tehát a szakdolgozati programom korántsem egy befejezett változat. Sok funkciót hordoz már magában, de további verziók elkészítésére adtam lehetőséget a nyílt forráskóddal.
További terveim között szerepel többek között

- a szállóhely kihasználtságának elemzésénél egy oszlop diagram készítése,
- a felhasználói szinten elvégezhető adathalmaz bővíthetőségének lehetősége,
- több felhasználó kezelésének lehetősége, akár egy időben,
- az adatbázishoz való felhasználói, jelszavas hozzáférés.
- nyomtatási lehetőségek beépítése.

Irodalomjegyzék

[1] **John Sharp**, *Microsoft® Visual C# 2005 lépésről lépésre*, Szak kiadó (2005)

[2] **Rádi György**, *Számítástechnika Windows és Office XP alapokon ECDL, számítógép-kezelő és számítástechnikai szoftverüzemeltető tanfolyamokhoz*, PSZF-SALGÓ Kft.(2002)

[3] **Mahesh Chand**, *A Programmer's Guide to ADO.NET in C#*, Apress (2003)

[4] **Tom Barnaby**, *Distributed .NET Programming in C#*, Apress (2002)

[5] **Carsten Thomsen**, *Database Programming with C#*, Apress (2002)

[6] **William R. Vaughn with Peter Blackburn**, *ADO.NET Examples and Best Practices for C# Programmers*, Apress (2002)

[7] **Christian Gross**, *Beginning C# 2008: From Novice to Professional*, Apress (2008)

Internetes irodalmak:

Wikipédia –a szabad enciklopédia–:

[8] **Microsoft .NET**
(http://hu.wikipedia.org/wiki/Microsoft_.NET)

[9] **C# programozási nyelv**
(http://hu.wikipedia.org/wiki/C_Sharp_programozási_nyelv)

[10] **Microsoft Visual Studio**
(http://hu.wikipedia.org/wiki/Microsoft_Visual_Studio)

[11] **ADO.NET**
(http://hu.wikipedia.org/wiki/ADO.NET)