Debreceni Egyetem
Informatikai Kar

Vendéglátásban alkalmazható, szobákat és azok foglalásait nyilvántartó szoftver készítése

Témavezető:
Dr. Végh János
DSc, egyetemi tanár

Készítette:
Horváth Gergő
mérnök informatikus
Varga Tamás
mérnök informatikus

Debrecen
2008

Nyilatkozat

Alulírott Horváth Gergő, a Debreceni Egyetem Informatikai Karának mérnök informatikus hallgatója kijelentem, hogy a „Vendéglátásban alkalmazható, szobákat és azok foglalásait nyilvántartó szoftver készítése” című szakdolgozat, és a hozzá kapcsolódó szoftver közös szerzemény, melyet Varga Tamással, szintén a Debreceni Egyetem Informatikai Karának mérnök informatikus hallgatójával közösen készítettünk el.

Debrecen, 2008. november 20.

Horváth Gergő

Tartalomjegyzék

1. Bevezetés

Mérnök informatikus hallgatóként lehetőségünk nyílt betekinteni az informatika különböző ágazataiba. Az elmúlt három és fél évben bővült elméleti tudásunk, azonban a gyakorlati részre sajnos kevesebb idő jutott. Ennek ellenére hozzánk mégis a feladatok konkrét, gyakorlatban használható megoldása áll közelebb. Ezért választottunk diplomamunkánk témájának egy ténylegesen működő, piacképes szoftver elkészítését.

Az ötletet egy hétvégi kirándulásból merítettük. Egy néhány szobás vendégházban szálltunk meg, amely családi vállalkozásként működik. A szállásfoglalás zökkenőmentesnek tűnt, azonban érkezésünkkor saját bőrünkön tapasztalhattuk, hogy ez nem mindig fedi a valóságot. A foglalásokat ugyanis egy kis füzetben vezette a tulajdonos. Egy beszélgetés során derült ki, hogy szükség lenne egy olyan eszközre, ami megoldja az ebből adódó adminisztratív problémákat. Így jött az ötletünk, hogy készítsünk egy szoftvert, ami lehetővé teszi a szobák, ügyfelek, és foglalások nyilvántartását, és kezelése nem okoz gondot egy informatikában kevésbé járatos személynek sem.

A szakdolgozat célja bemutatni az elkészített szoftver tervezésének, fejlesztésének fázisait, valamint annak részletes működését, funkcióit.

A tervezés és megvalósítás folyamán szem előtt tartottuk a beszélgetés során szerzett információkat. Ezek az információk rávilágítottak a szükséges funkciókra és szolgáltatásokra, melyek megvalósítása esetén megfelelő szoftvert tudunk készíteni. Így a feladatunk nem egy általunk kitalált célkitűzés megvalósítása, hanem a véletlen folytán felmerülő tényleges igény kielégítése lett.

Bár már léteznek olyan programok, melyek megoldást nyújtanak, de úgy gondoltuk, hogy ez a feladat megfelelő kihívás számunkra. Hiszen már a tervezés előtt rájöttünk, hogy célszerű lenne egy olyan programot készíteni, melyet a vendéglátásban működő kisebb és nagyobb egységek is megfelelően tudnak használni. Így a programot nem az imént említett vendégház konkrét méreteire terveztük, hanem úgy, hogy alkalmas legyen más jellegű, nagyobb egységek kiszolgálására is.

Úgy gondoltuk, hogy az elsőre – funkcióiban - egyszerűnek tűnő program elkészítése komoly átgondolást, tervezést igényel, valamint a megvalósításában is minden részletre ügyelni kell. Rájöttünk, hogy az adatok kezelését és tárolását legmegfelelőbben egy adatbázis-szerver használatával tudjuk megoldani. A kialakítandó adatbázist pedig egy, a mai igényeknek megfelelő grafikus felületen keresztül lehet elérni, amely megvalósításához egy fejlett, magas szintű programozási nyelv (C#) lehetőségeit használtuk ki.

Ezek alapján alkalmasnak véltük a problémát arra, hogy csapatban, ketten valósítsuk meg a megoldást. Így a feladatot kisebb részfeladatokra bontottuk, egy-egy részt külön, viszont egymás munkáját segítve oldottuk meg. Úgy éreztük, az ezáltali csapatmunkával szerzett tapasztalatok segíthetik a majdani beilleszkedést egy munkahelyi csoportba.

Manapság ugyanis ha bekerül az ember egy vállalathoz, akkor egyre inkább előtérbe kerül a csoportos munkavégzés. Szükség van az együttműködésre, a másokkal való munkavégzésbe is bele kell tanulni. Egy-egy nagyobb fejlesztés során több ember tevékenysége adódik össze, tudnunk kell alkalmazkodni és egymást segíteni a munka során.

A szoftver elkészítése olyan programozási nyelven történt, mely nem képezte a tananyag részét egyetemi éveink alatt. Számunkra ez plusz kihívást és motivációt jelentett, hiszen a kötelező feladat elkészítésében megláttuk az önképzés további lehetőségeit egy gyakorlati példán keresztül. Véleményünk szerint a folyamatosan fejlődő informatikában a sikerességhez elengedhetetlen feltétel, hogy ez a szemlélet kísérje végig munkánkat.

Szakdolgozatunk végigvezet a fejlesztés szakaszain, bemutatva ezzel egy fejlesztési projekt lépéseit. Ezen felül bemutatja azokat az eszközöket, amelyeket felhasználtunk a készítés során. A dolgozat végére világossá válik a szoftverfejlesztés menete, a program funkcionalitása, működése, és a mögötte álló technológiák alapja. A szakdolgozat további részeiben szoftverünkre gyakran az általunk kitalált, **Room Manager** névvel hivatkozunk.

2. A program

Ebben a fejezetben bemutatjuk a program használatához szükséges feltételeket, valamint a készítés szakaszait a tervezéstől a megvalósításon át egészen a tesztelésig.

2.1. Rendszerkövetelmények

A szoftvert Windows operációs rendszerre fejlesztettük. A program futtatásához szükség van bizonyos eszközök meglétére a számítógépen, amelyek nélkül a program el sem indulna.

Az egyik ilyen elengedhetetlen komponens a Microsoft .NET keretrendszer. A programot a Microsoft Visual Studio 2008 szoftverrel fejlesztettük, amely önmagában integrálja a .NET eszközeit. A futtatás feltétele, hogy a használt számítógépen installálva legyen a .NET keretrendszer 3.5-ös verziója, amit bárki ingyenesen letölthet az internetről.

Mivel a program egy MySQL szerverhez kapcsolódik és annak egy adatbázisát manipulálja, elengedhetetlen, hogy a számítógépen megtalálható legyen egy ilyen szerver. Persze nem kell feltétlenül egy számítógépen futnia az alkalmazásnak és a MySQL szolgáltatásnak, ha hálózatban szeretnénk dolgozni. Erről részletesebben a Fejleszthetőség fejezetben lesz szó.
A legújabb és ingyenesen letölthető verziója ennek a szoftvernek a MySQL 5. A fejlesztés és tesztelés folyamán mi is ezt használtuk.

A MySQL telepítése és konfigurálása nagyon egyszerű, nem igényel különleges informatikai jártasságot, többek közt emiatt is esett erre a választásunk.

A MySQL szervert a telepítés után úgy kell konfigurálni, hogy a programunk kapcsolatba tudjon vele lépni. A szerver alapértelmezésként a 3306-os portot használja a kliensekkel való kommunikációra, de mi ezt biztonsági okokból megváltoztattuk. A módosítás oka abban rejlik, hogy egy külső támadó nyilván az alapértelmezett porton keresztül próbálja meg először manipulálni az adatbázisunkat. Új értéknek a 3333-as portszámot választottuk. Ennek

megfelelően a MySQL szerver konfigurációja közben ezt is be kell állítani, mert különben nem tudjuk igénybe venni a szerver által nyújtott szolgáltatásokat.
A konfiguráció során meg kell adnunk a *'root'* felhasználó jelszavát is. Tekintettel arra, hogy egy megrendelés esetén a megrendelő szabadon választhat jelszót, ezt a forráskódban elegendő egy helyen átírni és a szoftver gondtalanul használható. Ez igaz arra az esetre is, amikor a megrendelő használ már esetleg más célra is MySQL szervert, amelynek root jelszavát számunkra megadva könnyen tudjuk módosítani a forráskódot ennek megfelelően és így a kész program csatlakozni tud a szerverhez.

A fent említett két feltétel mellett szükség van még egy harmadik összetevőre is, amely biztosítja a kapcsolat felépítésének lehetőségét a .NET eszközeit használó alkalmazás és a MySQL szerver között. Ez a MySQL Connector Net 5.2.3-as változata, amely hiányában szintén működésképtelen a program. A MySQL Connector Net szintén ingyen letölthető a MySQL honlapjáról, telepítése mindössze pár kattintás. Megléte azért fontos, mert az általa definiált osztályokat felhasználva hoztuk létre a kapcsolatot az alkalmazás és a MySQL szerver között.

A potenciális megrendelőnek tehát a használat előtt gondoskodnia kell az említett követelmények teljesítéséről, illetve azok hosszú távú meglétéről az esetleges problémák elkerülése érdekében.

2.2. Tervezés

A fejlesztés elkezdése előtt pontosan át kellett gondolnunk, hogy hogyan is épüljön fel a szoftverünk struktúrája. Ebben a fejezetben bemutatjuk, hogyan állítottuk össze az adatbázist, miként képzeltük el programunk kinézetét, funkcióit.

2.2.1. Az adatbázis

Az adatbázis-tervezés egy olyan folyamat, amely lépések sorozatából tevődik össze. Első lépésként az adatbázisban leképezendő rendszert a számunkra fontos szempontok szerint elemzésnek vetjük alá és meghatározzuk a tárolandó adatok körét, azok kapcsolatait és az adatbázissal szemben felmerülő igényeket.

Ezután következik a rendszertervezés, melynek eredménye az adatbázis logikai modellje.

Végül fizikai szinten képezzük le a logikai adatbázis-modellt a felhasználható szoftver és hardver függvényében.

A tervezés során szem előtt tartottuk az adatbázis-rendszerekről tanultakat, és ezen szabályok betartásával valósítottuk meg az adatbázist. Gondolunk itt olyan fogalmakra, mint funkcionális függés, redundancia kezelése, illetve a normálformák kritériumainak teljesítése.

Adatok között akkor áll fenn funkcionális kapcsolat, ha egy vagy több adat konkrét értékéből más adatok egyértelműen következnek. Például esetünkben a szobák azonosítója az emelet- és szobaszámból tevődik össze, és így az azonosító és a két másik mező között funkcionális kapcsolat áll fenn, mivel minden szobának egyedi szobaazonosítója van.

Redundanciáról akkor beszélünk, ha valamely tényt vagy a többi adatból levezethető mennyiséget ismételten (többszörösen) tároljuk az adatbázisban. A redundancia, a szükségtelen tároló terület lefoglalása mellett komplikált adatbázis frissítési és karbantartási műveletekhez vezet, melyek könnyen az adatbázis inkonzisztenciáját okozhatják.

Normalizálás alatt bizonyos szabályok alkalmazását értjük, melyek nagyban megkönnyítik az adatbázisunk fenntartását.

A bevezetésben említett vendégház tulajdonosát ismét megkerestük, és segítségével, az ő tapasztalatait felhasználva alakítottuk ki az adatbázis tábláinak szerkezetét.

Minden táblában szükség van egy elsődleges kulcsra, ami alapján a sorok egyértelműen azonosíthatók. Ennek az értéke minden sorban különböző kell, hogy legyen. Minden mező kap egy típust. Ez azt definiálja, hogy milyen értékű lehet az adott mező. Például születési helynek nem célszerű szám típust választani. Szót kell még ejteni a NULL értékről. Akkor használjuk, ha egy adott mezőnek nincs értéke, nem ismerjük az értékét, vagy opcionális, nem kötelezően kitöltendő mezőről van szó. Például e-mail cím megadása esetén, ha valaki nem rendelkezik vele, akkor NULL érték kerül a mezőbe.

Szükségünk volt egy táblára, ami tárolja a vendéglátó egység szobáinak adatait. Elsődleges kulcsnak a szobaazonosítót (*szoba_ID*) választottuk. Ennek értéke abból adódik, hogy hányadik emeleten, hányadik szobáról van szó. Ezeket az értékeket az *emelet* és a *szam* mezők tartalmazzák. Fontos szempont egy szoba esetén, hogy hány személy fér el, hány ágy áll rendelkezésre, és esetleg van-e lehetőség pótágy elhelyezésére. A pótágy és a TV lehetősége nem feltétlenül áll minden szobában rendelkezésre, így azt logikai értékként helyeztük el a táblában.

Mezők	Típus	Lehet-e NULL
szoba_ID	**varchar(4)**	**nem**
emelet	tinyint unsigned	nem
szam	tinyint unsigned	nem
agyak_szama	tinyint unsigned	nem
ferohely	tinyint unsigned	nem
potagy	Bit	igen
TV	Bit	igen

1. táblázat

A *szoba_ID* négy karakterből áll, az első két karaktere az emelet, a fennmaradó részben a szobaszám szerepel (pl.: 0104: első emelet, négyes szoba).

Az *ugyfelek* táblában tároljuk az érkező vendégek adatait. Az ügyfelek azonosítása céljából alapvető fontosságú a nevük tárolása, viszont számoltunk azzal, hogy több azonos nevű ügyfél is lehet. Így a névből és születési dátumból generáltunk egy harmadik mezőt, amely betölti a táblában az elsődleges kulcs szerepét. Ezen információk mellett szükség van az ügyfél postai és telefonos elérhetőségeire, és ha van, akkor e-mail címére is. Gondoltunk arra az eshetőségre, ha olyan ügyfél érkezik, aki nem maga finanszírozza szállásának költségeit. Emiatt döntöttünk a számlázási adatok külön rögzítésére, mely az esetek többségében redundanciát okoz, viszont ha kell, speciális igényt elégít ki. Extra funkcióként számon tartjuk a foglalások számát, ennek függvényében kedvezményes árat lehet biztosítani a visszatérő ügyfelek számára.

Mező	Típus	Lehet-e NULL?
ugyfel_ID	**varchar(12)**	**nem**
nev	varchar(100)	nem
szul_datum	date	nem
ir_szam	smallint unsigned	nem
varos	varchar(100)	nem
cím	varchar(100)	nem
telefon	varchar(11)	nem
e_mail	varchar(100)	igen
sz_nev	varchar(100)	nem
sz_ir_szam	smallint unsigned	nem
sz_varos	varchar(100)	nem
sz_cim	varchar(100)	nem
foglalasok	tinyint unsigned	nem

2. táblázat

Az *ugyfel_ID* mező tizenkét karakterből épül fel. Az első négy karakter megegyezik a *nev* mező azonos karaktereivel, a maradék nyolc karakter pedig a születési dátum számjegyeiből áll össze. Új ügyfél felvétele esetén a foglalások számához mindig nulla érték kerül.

A program fő feladata a foglalások nyilvántartása, így kezelésükre létrehoztuk a *foglalasok* táblát. Ennél a táblánál meg kell említenünk az összetett kulcs fogalmát: a kulcs legalább két attribútumból áll. Előfordulhat az is, hogy az összes oszlop szerepel a kulcsban. Jelen esetben az *ugyfel_ID*, *szoba_ID* és a *kezdete* mezők alkotnak összetett kulcsot. Ez azt jelenti, hogy a táblában nem létezhet két olyan sor, ahol mind a három mező értéke megegyezik.

Ezzel biztosítjuk azt, hogy egy szobát azonos időpontban ne foglaljanak le kétszer, viszont így egy ügyfélnek egyszerre több foglalása is lehet.

Mező	Típus	Lehet-e NULL?
ugyfel_ID	**varchar(12)**	**nem**
szoba_ID	**varchar(4)**	**nem**
kezdete	**date**	**nem**
vege	date	nem
letszam	tinyint unsigned	nem
ar	mediumint unsigned	nem
fizetve	bit	nem

3. táblázat

Az *ugyfel_ID* mező értékének szerepelnie kell az *ugyfelek* táblában, a *szoba_ID* értékének pedig a *szobak* táblában. Ez a két mező biztosítja a kapcsolatot a táblák között, tehát külső kulcs szerepét tölti be.

Értelemszerűen egy foglalásnak van kezdete, van egy tervezett kijelentkezési dátuma, valamint rögzítve van, hogy hány fő szeretne tartózkodni a szobában. A foglalás tartalmazza az árat is, ami a végösszeget jelöli. A fizetve mező tartalmazza, hogy a számla rendezése megtörtént-e. Erre a programban megvalósított bevételi jelentések miatt volt szükség, valamint a keresési lehetőségeket is bővítette.

Az alapvető funkciók elkészítéséhez a fenti három táblára volt szükségünk. A szoftver többoldalúsága miatt gondoltunk arra, hogy nem csak egy felhasználó fogja kezelni a programot. Emiatt szükségünk volt egy felhasználókat tároló táblára. Mivel hosszú távú használatra terveztük, biztosítani kellett az árak változásának követését is. Így létrehoztunk egy táblát, ami tartalmazza az aktuális árakat.

Mező	Típus
tetel	**varchar(20)**
ar	int

4. táblázat

Mint látható teljesen egyszerű a tábla felépítése. Minden olyan dolog, melynek ára van a vendégházban, külön sorként szerepel a táblában. Az elsődleges kulcs a *tetel*, nem adhatunk meg két különböző árat ugyanannak a tételnek.

Mező	Típus
usernev	varchar(10)
jelszo	varchar(128)

5. táblázat

A *usernev* tartalmazza a felhasználónevet, ez az elsődleges kulcs, hogy ne szerepelhessen két azonos felhasználónév a rendszerben. Ennek a mezőnek az értéke maximum tíz karakter hosszúságú lehet. A *jelszo* mező tartalmazza a beállított jelszót, MD5 kódolással. Az MD5 megvalósítása pár soros művelet a forráskódban, és számunkra megfelelő biztonságot nyújtó kódolási eljárás. Így illetéktelen nem tudja kiolvasni a felhasználók jelszavát.

2.2.2. A felhasználói felület

A felhasználói felület megalkotásánál az átláthatóság volt a legfőbb cél. Éppen ezért kerestünk olyan fejlesztői környezetet, mellyel a mai igényeknek megfelelő, grafikus felületet tudunk létrehozni.

A program úgynevezett **form**okból épül fel. Minden **form** egy-egy külön ablakot jelent. A szoftver egyes funkciói külön ablakban érhetőek el.

A felhasználónak gyakorlatilag csak kattintgatnia kell az egérrel, de természetesen a billentyűzetről is vezérelhető a működés. A beviteli adatmezőket egyértelmű feliratozással láttuk el, így nem okoz gondot a szoftver használatának gyors elsajátítása.

Mindig egy aktív ablakban tud dolgozni a felhasználó. Addig nem tud tovább- vagy visszalépni, amíg az aktuális műveletet be nem fejezte, vagy meg nem szakította. Ezt azért tartjuk hasznosnak, mert így nem lehet „elveszni” a nyitott ablakok között.

Az egyes mezőket úgy próbáltuk elhelyezni, hogy a logikailag egymáshoz kapcsolódó részek egy csoportot alkossanak, és a különálló csoportok ne helyezkedjenek el túl közel egymáshoz, az átláthatóság érdekében.

A könnyebb használat miatt a vezérlőgombokat színes háttérrel láttuk el, minden gomb a funkciójához közel álló színt kapott. A visszalépő gombok pirosak, a rögzítést, változtatást elfogadó gombok pedig zöld színűek, vizuálisan is segítve a felhasználót a program kezelésében.

Az ebben a fejezetben felvázolt szempontok szerint készítettük el a programot. A megvalósítás részleteit a dolgozat további fejezeteiben tárgyaljuk.

2.3. Megvalósítás

2.3.1. Mit-miért?

Miután az adatbázis logikai modellje elkészült, el kellett döntenünk, hogy milyen környezetben valósítjuk azt meg. Rövid tájékozódás és a lehetőségek felmérése után döntöttünk a MySQL használata mellett.

A MySQL egy többfelhasználós, többszálú, SQL-alapú, relációs adatbázis-kezelő szerver.

Számunkra a rendszer többfelhasználós, többszálú tulajdonsága azért fontos, hogy a fejleszthetőségnél leírt lehetőségek megvalósulhassanak. Fontos ugyanis, hogy ha a jövőben több kliens csatlakozik a szerverhez, akkor – akár egyidejű – kiszolgálásuk ne okozzon problémát.

A relációs adatbázis-kezelők általában az SQL (Structured Query Language: Strukturált lekérdező nyelv) nyelven programozhatók. A nyelvi elemeket szokásos adatdefiníciós (Data Definition Language, DDL) és adatkezelési (Data Manipulation Language, DML) részekre bontani. A nyelvben az utasításokat pontosvessző választja el egymástól. A DDL utasítások segítségével lehet táblákat létrehozni, módosítani, törölni. Az adatok felvitelére, törlésére, módosítására a DML nyelv használható.

A MySQL az egyik legelterjedtebb adatbázis-kezelő. Egyik legfőbb előnye, hogy nyílt forráskódú, ingyenesen elérhető szoftver. Több operációs rendszert és programnyelvet támogat.

Az adatbázis-kezelés módjának kiválasztása után el kellett döntenünk, hogy milyen programozási nyelvben szeretnénk a programot elkészíteni. A döntés meghozatala előtt több alternatíva is szóba került.

Legelső gondolatként a Java nyelv használata merült fel, mivel tanulmányaink során ebben több tapasztalatot szereztünk, mint más nyelvekkel kapcsolatban. Úgy gondoltuk, hogy a

grafikus felület létrehozásának támogatása a Java-ban számunkra elég szegényes, és bonyolultnak tűnő.

Az egyetemi képzésen belül egy féléven keresztül volt szerencsénk megismerkedni a Microsoft .NET keretrendszerrel. A Microsoft által készített .NET keretrendszer (a .NET Framework) gyors alkalmazásfejlesztést, platformfüggetlenséget és hálózati átlátszóságot támogató szoftverfejlesztői platform. A Microsoft Visual Studio tökéletes környezetet nyújt a fejlesztéshez, integráltan tartalmazza a .NET keretrendszert. Ebben a fejlesztői környezetben több programozási nyelv használatára van lehetőség. Órai keretek között a Visual Basic és a C# használatával ismerkedtünk meg. Mivel szintaktikailag a C# közelebb áll a Java-hoz, ezért döntöttünk úgy, hogy programunkat e nyelv használatával készítjük el.

Mivel C#-ban előtte csak konzol alkalmazások írásával foglalkoztunk, ezért szükség volt tudásunk továbbfejlesztésére.

A nyelv alapjául a C++ és a Java szolgált. 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 nyelv primitív adattípusai objektumok, a .NET típusok megfelelői. Hulladékgyűjtést használ, valamint az absztrakcióinak többsége (osztályok, interfészek, kivételek…) a .NET futtatórendszert használja közvetlen módon.

2.3.2. A táblák elérése

A Tervezés részben tárgyalt adatbázis és a felhasználói felület között meg kellett teremteni a kapcsolatot.

Ahhoz, hogy bármilyen módosítást végrehajtsunk az adatbázison, először kapcsolódni kell hozzá. A kapcsolódás megvalósítását nagyban megkönnyítette a MySQL Connector által

biztosított osztályok és típusok használata. A kapcsolat felépítéséhez meg kell adnunk egy connectionstringet. Ennek felépítése a következő:

```
static string host = "localhost";
static string database = "MySQL";
static string user = "root";
static string password = "szakdoga";
static string connectionString = "Server=" + host + ";Port=3333;Database="
+ database + ";Uid=" + user + ";Pwd=" + password;
```

Mivel egy gépen futtattuk a MySQL szervert, és a Room Managert is, ezért hostnak a localhostot használtuk. A database jelöli azt az adatbázist, amelyhez kapcsolódni szeretnénk. Alapértelmezettként ez a mysql. Az első futtatásnál mi is ehhez kapcsolódunk, mivel ekkor még nincs kész a saját adatbázisunk. Meg kell adnunk a kapcsolódáshoz használt portot is, mivel a szerver csak ezen a porton keresztül szolgálja ki a klienseket. Értelemszerűen a user és password változókba kerül a felhasználónév és jelszó, ezt nem szabad összetéveszteni a programunkba történő belépéshez használt felhasználónévvel és jelszóval. Ezeket az értékeket egy string típusú változóba összefűzve adjuk át a Connector által definiált `MySqlConnection` osztály konstruktorának. Az ez által létrehozott objektum `Open()` metódusát meghívva nyithatjuk meg a kapcsolatot.

```
mysqlCon = new MySqlConnection(connectionString);
mysqlCon.Open();
```

Sikeres kapcsolódás után bármilyen SQL parancsot kiadhatunk az adatbázis tábláira vonatkozóan. A parancsok létrehozásához és végrehajtásához egy `MySqlCommand` tipusú objektumra van szükségünk.

```
sqlParancs = "CREATE TABLE felhasznalok (usernev VARCHAR(10) PRIMARY KEY
NOT NULL, jelszo VARCHAR(150) DEFAULT 'NULL')";
MySqlCommand mysqlCmd = new MySqlCommand(sqlParancs, mysqlCon);
```

Amint látható, az SQL parancsot egyszerű stringként kell megadnunk, ugyanolyan szintaktikával, mintha csak az adatbázisban dolgoznánk parancssoron keresztül. Ezt a stringet, valamint egy `MySqlConnection` típusú objektumot kell átadni a `MySqlCommand` osztály konstruktorának. Ezzel a kapcsolatot és a parancsot definiáltuk, viszont a parancs végrehajtása még nem történt meg. Kétféle végrehajtást használtunk. Az olyan esetekben, amikor nem

szeretnénk adatokat kinyerni az adatbázisból – ilyen a törlés, módosítás, sorok beszúrása – a következő módon jártunk el:

```
mysqlCmd.ExecuteNonQuery();
```

Ez a metódus egész értékkel, az érintett sorok számával tér vissza, de számunkra ez nem volt fontos, így nem használtuk.

Azokban az esetekben, amikor lekérdezést végeztünk – például a szabad szobák keresésénél – a `MySqlDataReader` osztályt használtuk.

```
MySqlCommand select = new MySqlCommand("SELECT usernev FROM felhasznalok",
Startup.mysqlCon);
            MySqlDataReader reader = select.ExecuteReader();
            while (reader.Read())
            {
                if (textBox1.Text.Equals(reader.GetString(0)))
                {
                    megvan = true;
                    break;
                }
            }
            reader.Close();
```

A fenti kódrészlet tökéletes példa egy select utasítás által visszaadott eredmény kiolvasására egy `MySqlDataReader` objektum felhasználásával. A *felhasznalok* táblából keressük ki a *usernev*eket. Mielőtt adatot olvasnánk ki, meg kell hívnunk ennek az objektumnak a `Read()` metódusát. Ha ezt a hívást egy ciklus feltételében helyezzük el, akkor addig fog olvasni, amíg be nem olvassa az összes visszaadott sort. Így a ciklus magja egy sor feldolgozásának felel meg. A számunkra fontos adatot, jelen példában a *usernev*eket, típusnak megfelelő `Get()` metódusokkal kaphatjuk meg, mely paraméterének meg kell adni, hogy az adott sor hányadik mezőjében szerepel az adat. Az adatok kiolvasása után meg kell hívni a `Close()` metódust, ezzel bezárjuk a reader objektumot.
Az így kapott adatokat a felhasználó számára szemléletesen meg is kell tudni jeleníteni a képernyőn. Ennek megvalósításához `DataTable` objektumokat hoztunk létre, melyekbe dinamikusan szúrtuk be a kívánt sorokat. A `DataTable` megjelenítése `DataGridView` objektummal történik. Az így megjelenített táblázatok tükrözik az adatbázisban szereplő táblák felépítését.

A bemutatott technikákkal bármilyen parancs végrehajtható az adatbázison. A program futása során az említett kapcsolat folyamatosan nyitva van, így nem kell minden SQL utasítás végrehajtása előtt felépíteni újra és újra. Abban az esetben, ha a kapcsolat megszakad a szerverrel, akkor a programban használt kivételkezelésnek köszönhetően nem áll le a szoftver működése, hanem egy üzenetben értesül a felhasználó a hibáról. A programból való kilépéskor zárjuk le a kapcsolatot az adatbázissal.

2.3.3. Dinamikusság

A program fejlesztésének elkezdésekor még csak egy kezdeti elképzelésünk volt a leendő funkcióiról. Kezdetben felvázoltuk azokat az alapvető dolgokat, melyekről úgy gondoltuk, hogy elengedhetetlen kellékei egy ilyen jellegű szoftvernek.

Ahogy haladtunk előre a Room Manager fejlesztésével, folyamatosan egyre több ötletünk támadt, hogy mivel lehetne kibővíteni a program funkcióit.

Az első elképzelésünk az volt, hogy egy konkrét vendéglátó-ipari egység igényeinek megfelelően készítjük el az adatbázist, és ehhez kapcsolódóan hozzá alakítjuk a program funkcióit, szolgáltatásait. Később azonban arra jutottunk, hogy sokkal hatékonyabb, ha dinamikusan, paraméterezhetően készítjük el a programot.
Természetesen, ha érkezik egy felkérés, hogy valahol ezt a szoftvert szeretnék használni, akkor felmérjük az igényeket, és ennek megfelelően hozzuk létre a szobák adatbázisát. Arra is gondoltunk, hogy – igaz nem jellemző, de előfordulhat – egy szoba ideiglenesen nem elérhető, esetleg bővül a felszereltsége, vagy talán bővül a kiadható szobák száma. Ilyen esetben nem szükséges semmilyen adatbázis-kezelői ismeret a módosításhoz, bővítéshez, esetleges törléshez. A programban elhelyeztünk egy karbantartási opciót. Itt szabadon manipulálható az adatbázis. Egyértelműen, szemléletesen készítettük el ezt a funkciót, így nem okoz gondot az átlátása.

A felhasználói adatbázis folyamatosan bővül, azonban ennek exportálására és importálására lehetőséget biztosítunk a programban. Így le van védve a felhasználó egyrészt az adatbázis felől, másrészt a saját gépre mentett biztonsági másolattal is.

A MySQL adatbázis futhat azon a gépen, amelyen a programot használják, így megfelelő akár egy kis vendégháznak is, ahol egyetlen számítógép áll rendelkezésre.

Arra az esetre is felkészültünk, ha egy nagyobb egység, például egy kollégium szeretné használni ezt a szoftvert. Ha van egy központi szerver, amin már esetleg fut egy MySQL szerver, akkor a programban megadható ennek a szervernek a címe, ezáltal nincs szükség saját adatbázis-szerver telepítésére.

A Room Managert két felhasználói szinttel láttuk el.

Első indításkor tudjuk beállítani az *adminisztrátor*i szint jelszavát, ilyen jogosultságú felhasználóból csak egy létezik, az ő *usernev*e 'admin'. Neki van lehetősége elérni a karbantartási funkciókat, valamint új felhasználót létrehozni.

Az egyszerű felhasználó nem éri el a karbantartási funkciókat, csak az ügyfeleket és foglalásokat tudja rögzíteni és kezelni.
Azért láttuk szükségességét több felhasználó kezelésének, mert így nagyobb személyzettel rendelkező egységek is tudják használni a programot. Azt is szem előtt tartottuk, hogy az egyszerű szintű, kellő informatikai jártassággal nem rendelkező felhasználó véletlen se tudjon kárt tenni az adatbázisban.

2.3.4. Verziókövetés

A verziókövető rendszerek legfőbb célja, hogy egy program fejlesztése során készített programkódokat hatékonyabban és biztonságosabban tudjuk kezelni.
Verziókövetést használhatunk akkor is, ha egyénileg fejlesztünk egy szoftvert, de legfőképpen a csoportmunkában készített projektek esetén érezhetjük hasznosságát. Ilyen esetben előfordul, hogy több ember használ egyszerre egy forrásfájlt, módosítja annak tartalmát. A verziókövető rendszerek használatával kiküszöbölhető, hogy felülíródjon, és ez által elvesszen valamelyik fejlesztő munkája.

Tulajdonképpen a verziókövető szoftver egy szerver-alkalmazás, ami nyilvántartja az aktuális forráskódokat. Ezeket a kódokat tudja a fejlesztő letölteni a saját gépére. Elvégzi a szükséges műveleteket, fejlesztéseket, javításokat, majd visszatölti a szerverre a forrást. A szerver felügyeli, hogy egyszerre többen ne írják ugyanazt a fájlt. Ha többen is küldenek be ugyanazon fájl ugyanazon verziójához változást, akkor megpróbálja a változásokat összefésülni. Ha nem sikerül, akkor a konkurens verziókövető rendszerek az ütközéseket a később jóváhagyó ember számára jól észrevehetően jelzik.
Abban az esetben, ha eltérnek a módosítások, a szoftver kiemeli a konfliktusos részeket, ezáltal is segítve a fejlesztők munkáját. A verziókövető nyilvántartja a korábbi módosításokat, előző verzióit a programnak. Így bármikor elérhető egy korábbi állapot, abban az esetben, ha esetleg valamit később elrontunk.

Mivel mi Windows alatt fejlesztettük a diplomamunkát, ezért a CS-RCS nevű verziókövető szoftvert használtuk. Nem ez a leghatékonyabb, legokosabb ilyen jellegű program, viszont a mi céljainknak teljesen megfelelt. Egyszerű, könnyen megérthető és kezelhető szoftver, installálás után tökéletesen beépül a rendszerbe.

Nagymértékben segítette a munkánkat, mivel így folyamatosan tudtuk fejleszteni a programot, nem volt szükség rá, hogy „bevárjuk” egymás munkáját.

Verziókövető rendszerhez kapcsolódó **fogalmak**:

Repository: Az a hely, (katalógus) ahol a verziókövetésre kijelölt fájlok, dokumentumok változatait tárolják. Néha depot-nak vagy röviden repo-nak nevezik.
Working copy : Munkamásolat. A repository egyik verziójának fájljairól készült helyi másolat. A repositoryba kerülő fájlok innen származnak.
Branch: Magyarul **ág**, esetleg **fork**. A verziókövető rendszerben tárolt fájlok egy csoportja, amit leágaztattak az eredeti forrásból és azután külön életet él. Például egy szoftverből készítenek egy speciális változatot az egyik ügyfélnek. Azok a fájlok átmásolódnak a másik ágba és megkezdik különálló életüket.
Revision: Magyarul **verzió** vagy **version**, egy változat (állapot) a változatok között. Nem összetévesztendő a szoftver verziószámával, ezért szándékosan kapott más elnevezést.
Tag: Egy megjelölt, fontos változat (állapot) a verziók között. Ez általában egy jól megjegyezhető beszédes név vagy kód.
Konfliktus: Konfliktus akkor fordul elő, ha ketten módosították ugyanazt a fájlt, és mindketten megpróbálják véglegesíteni a változásokat. Csak az elsőnek sikerül a feladás. A konfliktus feloldására általában emberi beavatkozásra van szükség.

Műveletek

Checkout: Röviden **co.** Készít egy munkamásolatot a repositoryból. Ez aktuális verzió, de lehet egy régebbi meghatározott verzió is.
Commit: Másnéven **check-in**, **ci.** Egy változat (a helyi munkamásolaton történt változások) elhelyezése a repositoryba.
Update: Az **update** vagy **sync** frissíti a munkamásolatot. Bedolgozza, összefésüli a repositoryban történt újdonságokat a helyi munkamásolatba.
Merge: Összefésülés. Két változat összefésülése
Resolve: A frissítések közben jelentkező konfliktusok feloldása.

2.4. Működés

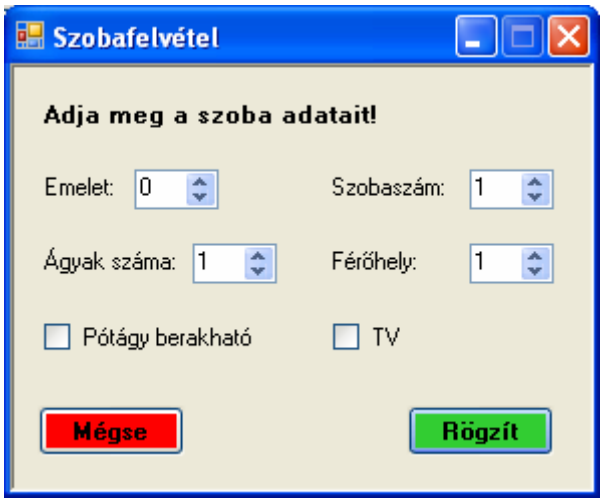
Ebben a fejezetben részletesen tárgyaljuk a szoftver funkcióinak működését. Annak érdekében, hogy szemléletesebbé váljon a bemutatás, az egyes részekhez képernyőképet is csatoltunk.

2.4.1. Szobák kezelése

A program használata közben a szobák kezelése nem szembetűnő, viszont nagyon fontos része a szoftvernek. Az első dolog, ami felmerül a szobákkal kapcsolatban, hogy tárolni kell az adataikat.

Kezdetben az adatbázisban nincsenek szobák. Új szoba hozzáadására csak az admin felhasználónak van lehetősége, mivel ez egy olyan funkció, amit csak nagyon ritkán használunk, és nincs is rá szükség, hogy bármely felhasználó megtehesse. Új szobák felvételére kétféle lehetőség van.

Az egyik a karbantartási ablakon elhelyezett **Új szoba felvétele** gombra kattintva, egy új ablakban adhatjuk meg a felvenni kívánt helyiség jellemzőinek értékét.



1. ábra

Az adatok megadása után a **Rögzít** gombra kattintva a program ellenőrzi az adatokat. Ide tartozik, hogy nem lehet egy emeleten két azonos számú szoba, és nagyban eltérő ágyszám és férőhely sem adható meg. Amennyiben valamelyik megadott érték nem megfelelő, hibaüzenetet kapunk. Ha minden adat kifogástalan, a szoftver értesíti a használóját a szoba bekerüléséről az adatbázisba.

A másik lehetőség szobák felvételére a szintén a karbantartási ablakban elhelyezett importálás. Ekkor egy számítógépen tárolt fájlból olvassuk be a szobák adatait. A program kizárólag CSV állományok megnyitását teszi lehetővé. A CSV fájlnak soronként egy szoba adatait kell tartalmaznia, ahol a sorok felépítése a következő:

```
szobaazonosító; emelet; szobaszám; ágyak száma; férőhely; pótágy; van-e tv
```

Ebben a sorban a pótágy és a televízióra vonatkozó logikai értéket kizárólag 0-val vagy 1-essel adhatjuk meg. A jellemzők értékeinek elválasztását a pontosvessző valósítja meg.
Egy ilyen fájl könnyen elkészíthető a Microsoft Excel programmal, ha a cellákba egymás után megadjuk a megfelelő értékeket, és soronként egy szoba adatait írjuk be. Ezután a táblázat mentésénél kiválaszthatjuk a CSV formátumot. De elkészíthető a kívánt fájl egy egyszerű editorral is, csak arra kell figyelni, hogy mentéskor itt is CSV fájlt hozzunk létre, és a sorok felépítése megfelelő legyen.

2. ábra

A megnyitott állományból a program soronként beolvassa a benne található jellemzőket. Ha a fájlban olyan szobaazonosító szerepel, amely már létezik az adatbázisban, akkor azt

figyelmen kívül hagyja és a következő sorral folytatja az importálást. A művelet végén üzenetben értesül a felhasználó, hogy hány új szoba adatainak rögzítése sikerült.

Ehhez hasonlóan megoldottuk az adatok lementését is a számítógépre, ezzel biztosítva az adatok hordozhatóságát. Ez a funkció szintén a karbantartási műveleteknél található, és az **Exportálás** gombra kattintva érhető el. Ekkor egy olyan ablakhoz jutunk, ahol választhatunk, hogy a szobák, ügyfelek vagy a foglalások adatait szeretnénk menteni.

3. ábra

A mappa és a fájlnév megadása egy mentési ablakkal történik meg, amely csak CSV fájlként engedélyezi a mentést. Az állomány felépítése természetesen megegyezik az importálásnál elvárttal. Az **OK** gombra kattintással a program létrehozza a fájlt.

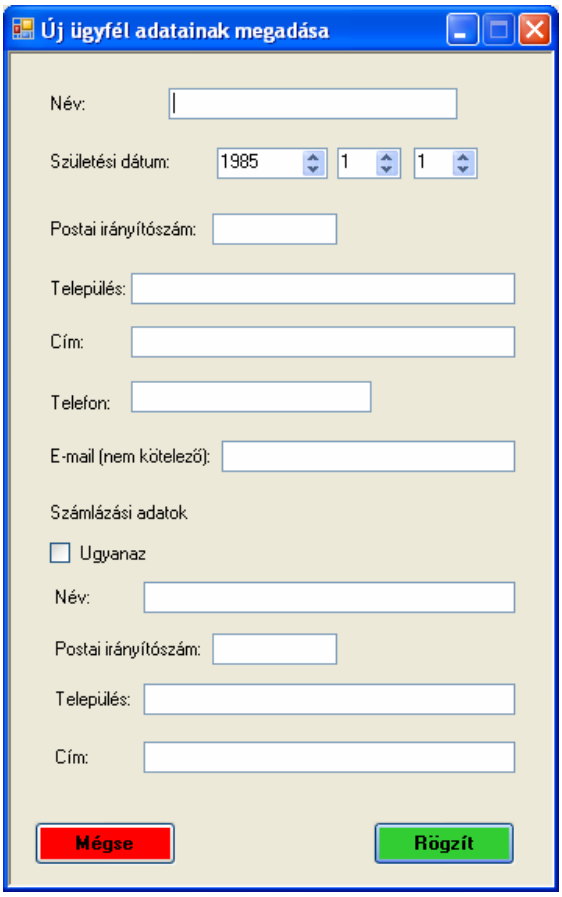
Szobák adatainak megadása viszont nem csak új szoba felvételénél lehetséges, hiszen a programban biztosítani kellett az adatmódosítás meglétét is. Ez a lehetőség a keresési funkciókon keresztül érhető el, ezért ezt abban a fejezetben tárgyaljuk.

Olyan esetben is használjuk a szobák adatait, amikor ebből a felhasználó gyakorlatilag nem tapasztal semmit, hiszen ezek a lekérdezések a háttérben futnak le. Ez megy végbe egy foglalás felvételénél, amikor az ügyfél igényeinek megfelelő szobát kell ajánlania a szoftvernek. Erről részletesebben a foglalások kezelésénél lesz szó.

2.4.2. Az ügyfelek nyilvántartása

A szoftvernek az adatbázisban naprakészen kell tárolnia az ügyfelek adatait, ezért ezek kezelése nagyon fontos. A program használatbavételekor természetesen még egyetlen ügyfél sincs rögzítve.

Az ügyfelek adatainak megadása a szobáknál látott két módon lehetséges. Ugyanolyan menete van az adatok kézzel való megadásának és az importálásuknak is. A CSV fájl felépítése természetesen eltér a szobáknál leírtaktól, az ügyfeleknél sorrendben az azonosító, név, születési dátum, irányítószám, város, cím, telefon, e-mail cím, számlázási név, számlázási város, számlázási cím és foglalások számának értékeinek kell szerepelniük pontosvesszővel elválasztva. Ennek megfelelően az exportálás esetén is ilyen formában készül el a CSV állomány.



4. ábra

A szobáknál látott módok mellett még egy harmadik lehetőség is van új ügyfél rögzítésére. Új foglalás esetén van lehetőség új ügyfél létrehozására. Ekkor a fenti ablak jelenik meg, amin a felhasználó megadhatja az új ügyfél adatait.

Az adatok kitöltése során a születési dátumnál biztosított, hogy csak valós dátumot lehessen megadni. Az irányítószám megadásánál is figyelni kell a helyes formátumra, mert ha nem megfelelő értéket adunk meg, akkor hibaüzenetet kapunk. Kényelmi lehetőségként megoldottuk, hogy ha a számlázási adatok megegyeznek az ügyfél adataival, akkor nem kell újra begépelni, csak egy kattintással átmásolhatóak. Ha nem adunk meg az e-mail cím mezőben semmit, akkor az adatbázisban ebbe a mezőbe NULL érték kerül.

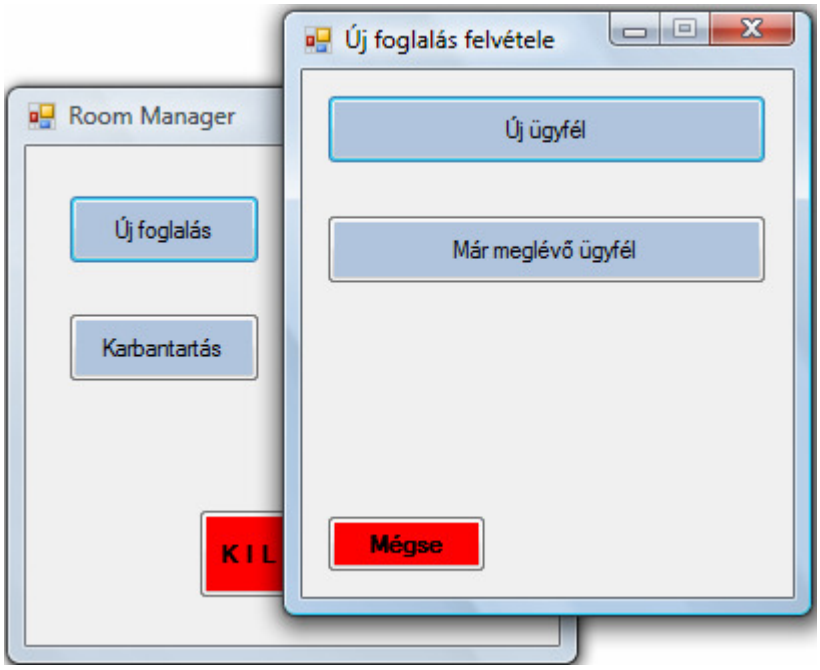
Van olyan mező is az *ugyfelek* táblában, amelynek értékét ezen az űrlapon nem lehet megadni. Ez a jellemző a foglalások száma. Minden új ügyfél esetén ez az érték értelemszerűen nulla lesz, és minden rögzített foglalásnál növeljük eggyel, illetve törlés esetén csökkentjük.

Az ügyfelek adatait természetesen tudnunk kell módosítani is, ezért ezt a feladatot is el kell látnia a szoftvernek. Ez a funkció az ügyfél megkeresése esetén érhető el, éppen ezért majd a Keresés és adatmódosítás című fejezetben tárgyaljuk részletesen.

2.4.3 Foglalások

Szoftverünk legfőbb célja a foglalások kezelése. Ebben a részben bemutatjuk, hogy miként valósítottuk meg ezt a funkciót.

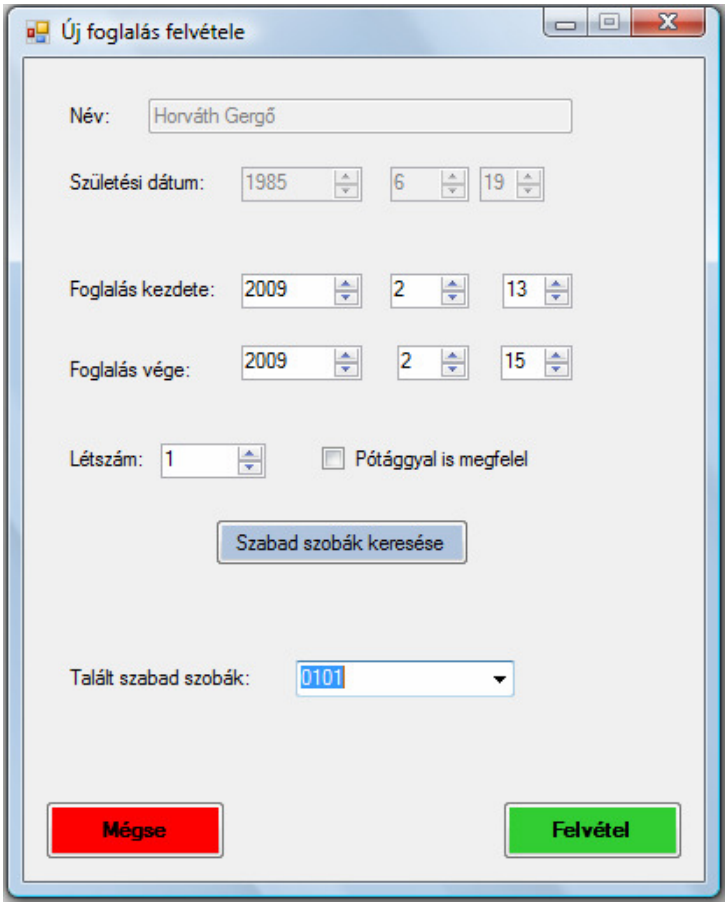
A főképernyőn látható az **Új foglalás** gomb. Ezt megnyitva lehetőségünk van már meglévő ügyfél foglalását rögzíteni, illetve a foglalás elhelyezése előtt vihetünk fel új ügyfelet az adatbázisba.



5. ábra

Ha már létező ügyfélnek foglalunk szobát, akkor megadva nevét és születési dátumát, a program kikeresi az adatbázisból, így adatait nem kell újra rögzíteni. Új ügyfél esetén értelemszerűen felvesszük az adatokat, ezután következhet a foglalás. Az előző fejezetben már tárgyaltuk az ügyfelek nyilvántartását, így erre most nem térünk ki külön.

Az ügyféladatok megadása után a következő képernyőre kerülünk:



6. ábra

Értelemszerűen kitöltjük a mezőket. A program biztosítja, hogy csak valós dátumot adhassunk meg, valamint a foglalás vége később kell, hogy legyen, mint a foglalás kezdete. A Szabad szobák keresése gomb megnyomásával a háttérben egy lekérdezés megy végbe az adatbázisban. A lekérdezés azokat a szobákat szűri ki, amelyek megfelelnek a létszámnál beállított követelménynek, valamint az adott időszakban nem foglaltak.

A Talált szabad szobák lenyíló menüben megjelennek az adott időpontban rendelkezésre álló, megfelelő férőhellyel rendelkező szobák.

A foglalás végelegesítése a Felvétel gombbal történik. Egy felugró ablakban még ellenőrizhetjük, hogy minden adat helyesen lett-e megadva. Ha mindent rendben találunk, akkor a program nyugtázza a foglalást, és felajánlja a lehetőséget további foglalások leadására, ugyanazon ügyfél számára.

A foglalásokat lehet törölni, módosítani, valamint lehetőség van a foglaláshoz kapcsolódó számla nyomtatására. Mivel ezek a keresésen keresztül érhetők el, ezért bővebben abban a fejezetben tárgyaljuk.

2.4.4. Keresés és adatmódosítás

A programban szükség volt keresési lehetőségek biztosítására a többoldalú használhatóság érdekében. Ez a funkció a kezdőablakról közvetlenül elérhető a Keresés gombra kattintással.

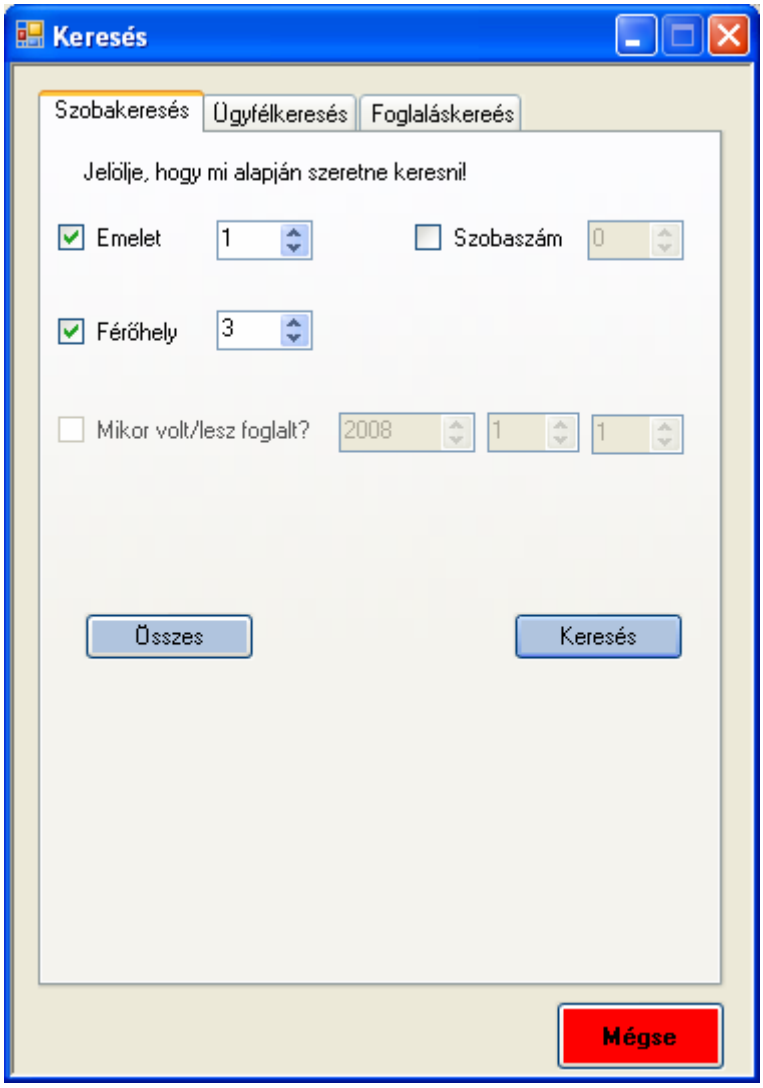
A funkció kiválasztása után megjelenik a keresési ablak, amelyen különválasztottuk az adatbázisban külön tárolt adatokra vonatkozó kereséseket.

A felhasználónak döntenie kell, hogy a vendéglátóhely szobái, ügyfelei vagy a rögzített foglalások közül szeretné kiszűrni a feltételeknek megfelelőeket. Ezt a három lehetőséget külön paneleken helyeztük el, így a felhasználó egyszerre csak a megfelelő keresés lehetséges beállításait látja a képernyőn.

Szobakeresés

Szoba keresése esetén választhatunk bizonyos keresési feltételek közül. A választást rádiógombok használatával oldottuk meg. Ha figyelembe szeretnénk venni a jelölőnégyzet által jelölt feltételt a keresés folyamán, akkor ki kell pipálnunk ezt a négyzetet és így adhatunk meg értéket a kritériumnak.
Lehetőség van egy adott emelet szobáinak keresésére, illetve kereshetünk szobaszám alapján is. Ha a két lehetőség közül csak az egyiket szeretnénk használni, akkor még megadhatjuk a szoba férőhelyét is, mint keresési feltételt. Ha viszont megadjuk az emeletet és a szobaszámot is, akkor erre nincs mód, mivel így egyértelműen azonosítottuk a szobát. Így helytelen férőhely beállítása esetén a keresés eredménye üres halmaz lenne.
Ha csak arra vagyunk kíváncsiak, hogy egy szoba foglalt volt-e egy adott napon, akkor meg kell adnunk azt, hogy hányadik emeleten található, és a számát. Ekkor elérhetővé válik, hogy egy dátumot adjunk meg. Ekkor a keresés eredménye természetesen csak egy szobát

tartalmazhat, amit mi adtunk meg. Ha az adott napon erre a szobára nem rögzítettek foglalást, akkor a keresés üres halmazt fog eredményezni.

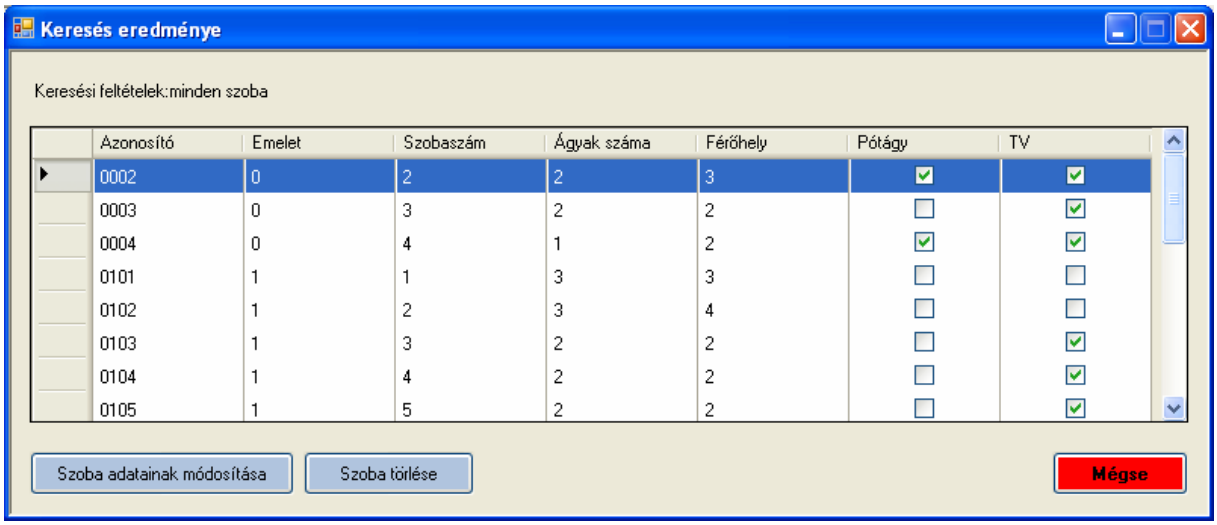


7. ábra

Ha minden szoba adatait szeretnénk egyszerre megtekinteni, nem kell megadnunk keresési feltételeket, mivel elhelyeztünk egy gombot a panelen, amelynek feladata éppen ez. Ha nem adunk meg egy feltételt sem, akkor a Keresés gomb nem is használható, ilyenkor inaktív állapotban van. Viszont a feltételek megadása nem zárja ki a lehetőséget, hogy minden szobát megjelenítsünk.

A fenti képen látható, hogyan néz ki a szobakeresést megvalósító ablak. A megadott esetben az első emeleten kerestünk olyan szobákat, amelyek három személy elszállásolását teszik lehetővé.

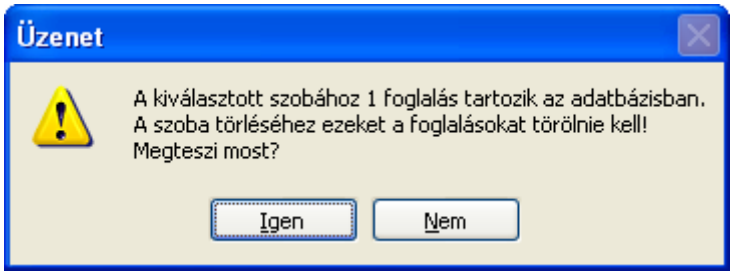
A keresési feltételeknek eleget tevő szobák mindig új ablakban jelennek meg.
Fontosnak tartottuk, hogy az eredményt megjelenítő oldalon is láthatóak legyenek a keresésnél megadott feltételek.



8. ábra

A kapott szobahalmaz elemei egy táblázatban jelennek meg, hűen tükrözve az adatbázisban lévő *szobak* tábla felépítését.
A táblázatban szereplő adatokat nem lehet megváltoztatni, viszont az admin felhasználó számára a program ezen részéről érhető el a szoba adatainak felülírása, illetve lehetőség van szobák törlésére is. Az erre a két műveletre létrehozott nyomógombok a második szintű felhasználók számára nem láthatóak és számukra ez a funkció a program más részéből sem érhető el.

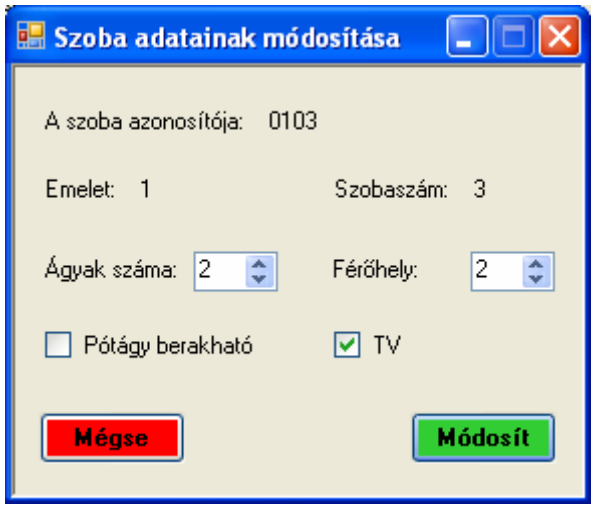
Az adatmódosítás és a szoba törlése mindig a kijelölt sorban szereplő szobára vonatkozik. A törlés esetén fontos, hogy egy szobához az adatbázisban akár több foglalás is tartozhat. Ha van foglalás regisztrálva az adott szobához, akkor a szoftver választás elé állítja használóját.



9. ábra

Nincs lehetőség ugyanis a szoba törlésére, amíg ahhoz egyetlen foglalás is tartozik. A felhasználónak tehát döntenie kell, hogy a szobával együtt törli a kapcsolódó foglalásokat is, vagy továbbra is szerepelni fog a szoba az adatbázisban.

Az adatmódosítás a szobaszámon és az emeleten kívül bármely jellemzőre vonatkozhat. Az adatok megváltoztatása esetén a program nem biztosítja a felhasználót, hogy a már regisztrált jövőbeli foglalások ide vonatkozó adatai mindenben megfelelnek a szoba új adatainak. Ezt szükség esetén saját kezűleg kell ellenőrizni, különben előfordulhat például, hogy a szoba férőhelye csökkent, miközben már létezik olyan foglalás, amely kihasználta volna a régi, maximális kapacitást is.

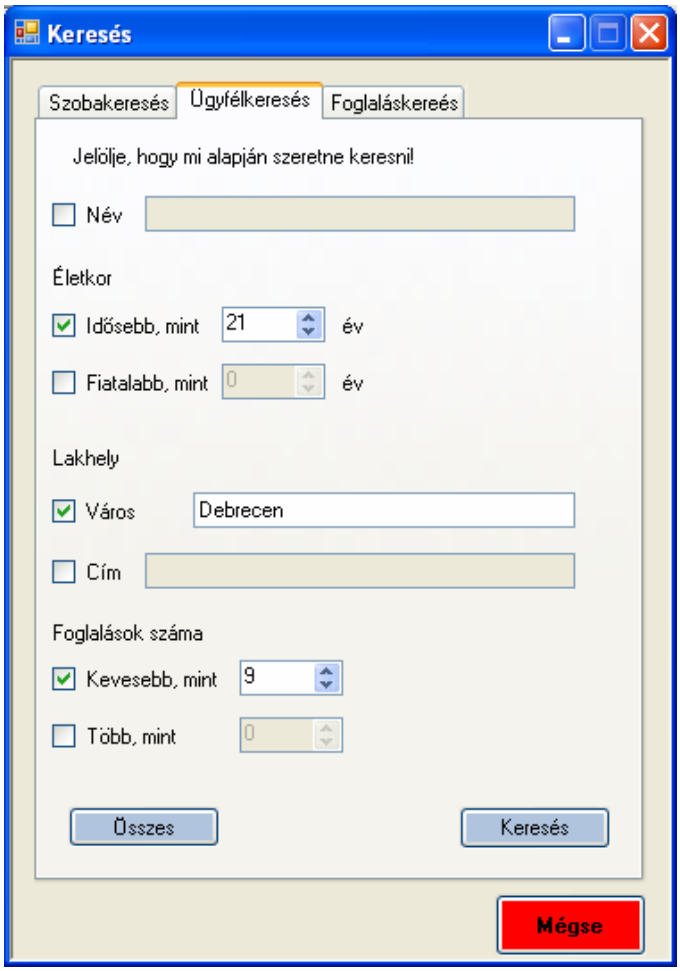


10. ábra

Az új értékek megadása után a Módosít gombra kattintva megtörténik az adatok ellenőrzése, és ha minden megfelelő, üzenetet kapunk a módosítások sikeres elvégzéséről, majd visszajutunk a keresési ablakhoz.

Ügyfélkeresés

A keresési ablak második lehetősége, hogy ügyfelek adatait szűrjük ki a megadott feltételek alapján. Lehetőségünk van az ügyfél nevének egy részlete alapján, életkor adatok, lakhely és foglalások száma szerinti keresésre is.
Az életkor és a foglalások számának megadása esetén nincs szükség rá, hogy ismerjük a pontos adatokat, ugyanis alsó és felső határokat adhatunk meg. A lakhely feltüntetésénél különválasztottuk a város és a cím mezőket, mivel az adattáblában is külön mezőként szerepelnek, illetve itt a keresésnél is több lehetőséget nyújtanak külön-külön.

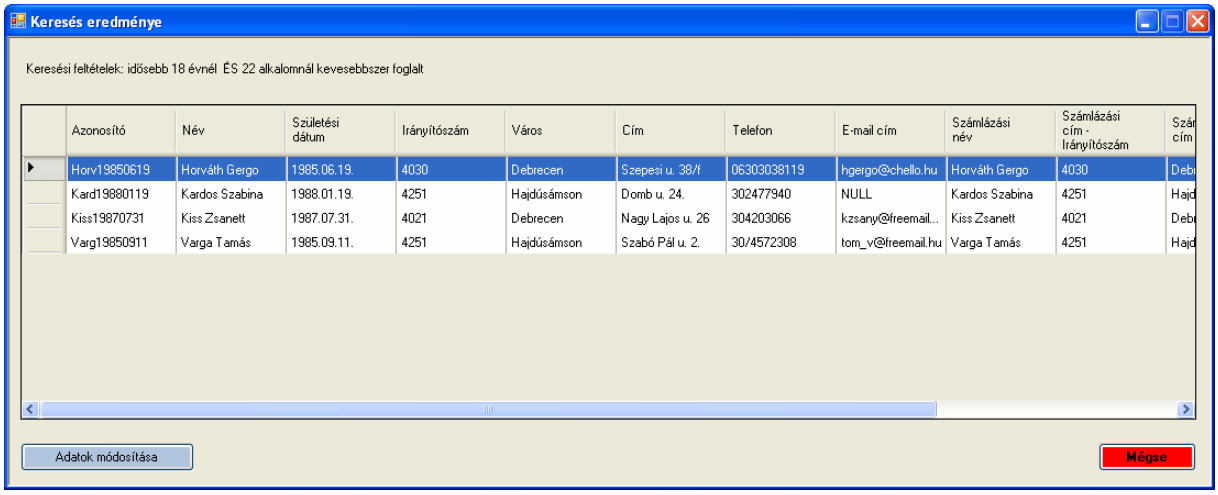


11. ábra

Természetesen itt is megtehetjük, hogy a keresési feltételek megadása nélkül megjelenítünk minden ügyfelet. Ennek módja teljesen megegyezik a szobakeresésnél leírt használattal.

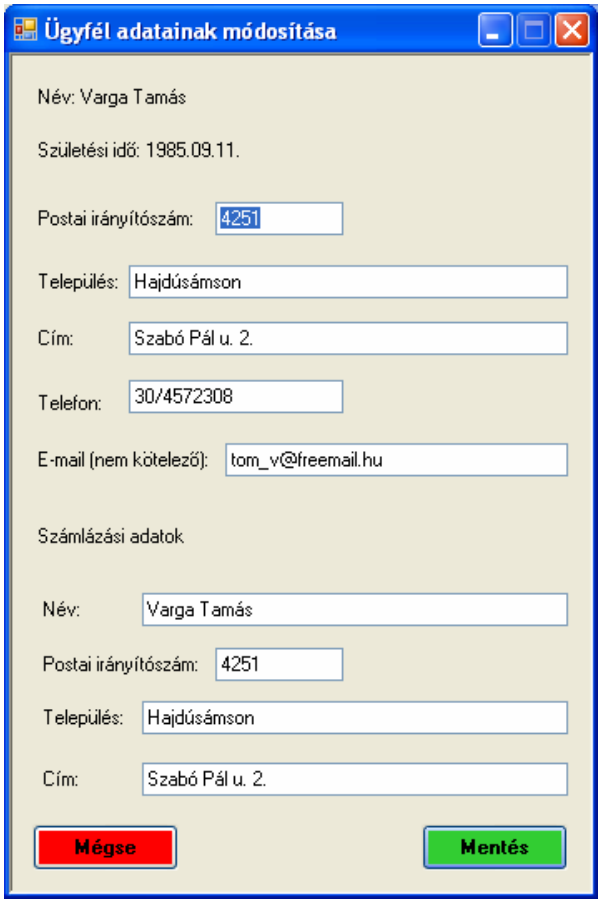
Az eredmény megtekintése is ugyanúgy lehetséges, külön ablakban kapjuk meg a feltételeknek megfelelő ügyfelek halmazát.

Az ablak felső részében ekkor is megjelennek a beállított keresési feltételek, illetve itt is lehetőségünk van már meglevő adatok felülírására. Erre sokszor szükség lehet, gondoljunk például egy költözésre, vagy a telefonszám megváltozására.



12. ábra

Az adatmódosítást itt már bármely felhasználó elvégezheti, mivel egy alkalmazottnak is tudnia kell az ügyfél új adatait rögzíteni. Természetesen ebben a táblázatban is egyszerre csak egy sort jelölhetünk ki, és a módosítás minden esetben a kijelölt ügyfélre fog vonatkozni.

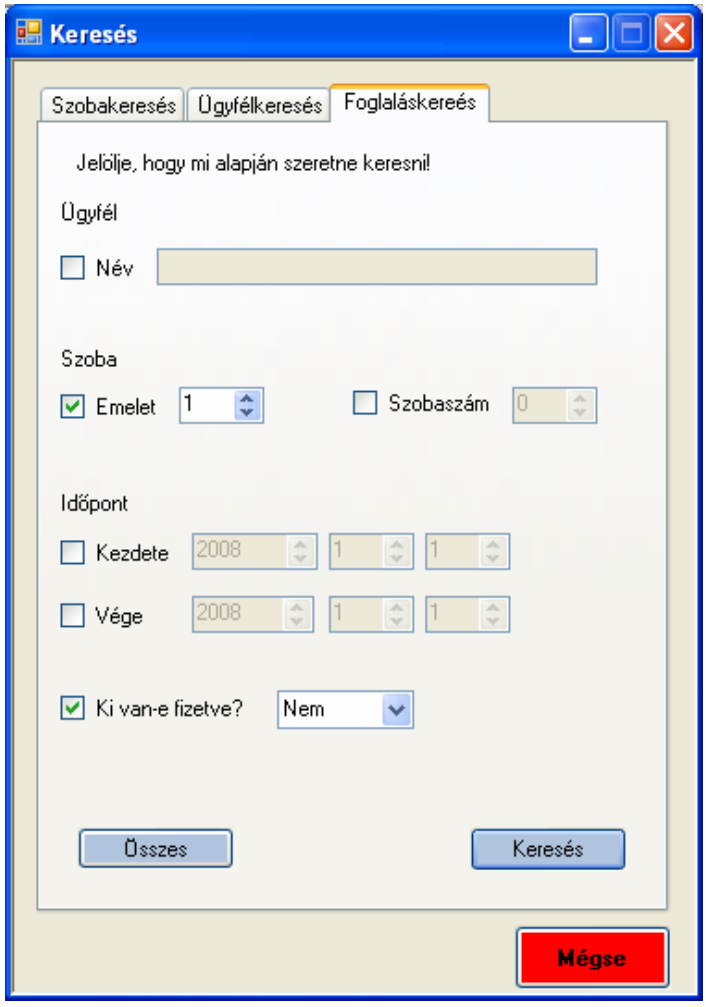


13. ábra

A kiválasztott személy adatai közül értelemszerűen nem módosítható a név és a születési dátum. A többi értéket mindenféle következmény nélkül felülírhatjuk, a sikeres változtatások után itt is üzenetet kapunk.

Foglaláskeresés

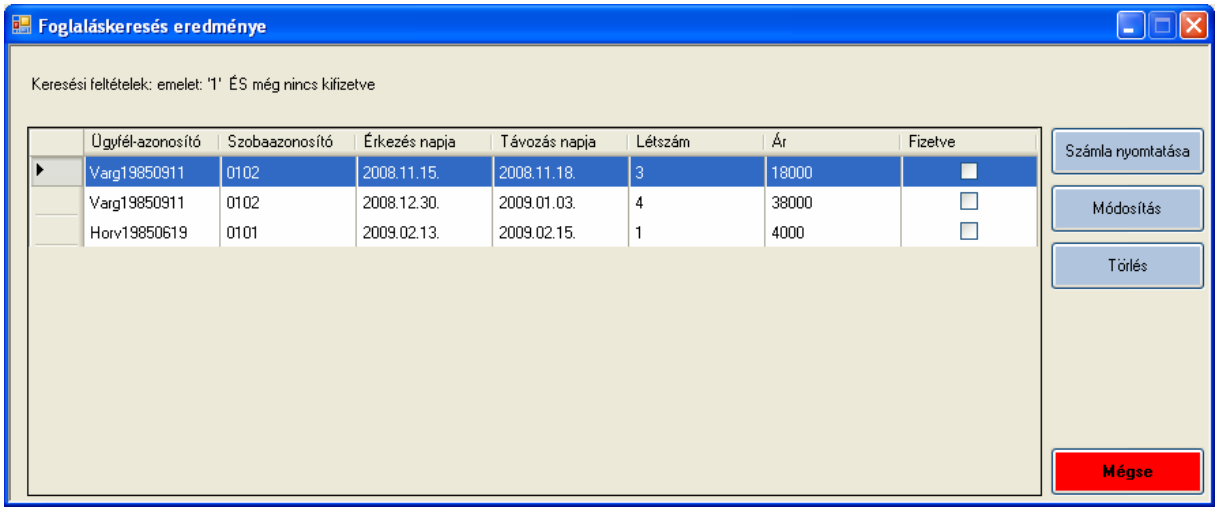
A keresési funkciók közül a legfontosabb a foglalások keresése. Ahogy látható, ez a funkció a keresési ablak harmadik lapfülén érhető el.



14. ábra

A rögzített foglalások közül kinyerhetjük a kívánt sorokat a foglaló személy neve, a foglalt szoba és a foglalás időpontja alapján is, valamint kereshetünk a szerint is, hogy a számla rendezése megtörtént-e már.

Ahogy a szobák és az ügyfelek esetében, itt is működik a feltételek nélküli összes tétel megmutatása, és az eredmény itt is külön ablakban látható.



15. ábra

Az eredményablak legsűrűbben használt funkciója a számlanyomtatás, mivel ezt minden fizetéskor meg kell tenni. A gombra kattintás után megtekinthetjük a nyomtatási képet, majd ha rendben találtuk az adatokat, elindítható a nyomtatás. A művelethez tartozik még, hogy az adatbázisban rögzíti a szoftver a bevételt, így később már látni fogjuk, hogy megtörtént a fizetés.

Egy foglalás esetén előfordulhat az is, hogy valamilyen okból az ügyfél változtatni szeretné az időpontot vagy a létszámot. Erre nyújt lehetőséget a Módosítás gomb használata. A kiválasztás után ekkor is egy új ablakhoz jutunk.

16. ábra

Az érkezés és távozás időpontjának módosítása esetén a program ellenőrzi az adatbázisban, hogy a megadott napok között nem foglalt-e a szoba. Ütközés esetén értesíti a felhasználót. Létszámnak megadható bármekkora szám, de a módosítás végrehajtásakor ellenőrzésre kerül, hogy nagyobb szám lett-e beállítva, mint a szobában elszállásolható személyek maximális száma. Ilyen hiba esetén a program felajánlja a maximális létszámra való módosítást és további szobafoglalásokkal lehet megoldani, hogy elég hely álljon az ügyfél rendelkezésére az adott napok között.

Biztosítani kellett a foglalás törlésének lehetőségét is a program funkciói között. A program semmilyen ellenőrzést nem végez ilyenkor, bármely foglalás törölhető, csak egy megerősítési üzenetre kell megfelelően válaszolni.

17. ábra

2.4.5. Egyéb funkciók

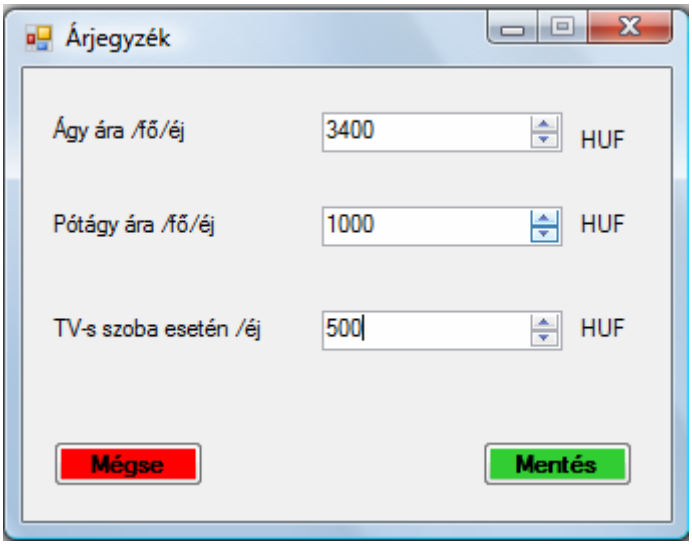
Ebben a részben a program nem alapvető, viszont lényeges funkcióit tárgyaljuk.

Elsőként az árjegyzék beállításáról lesz szó. Mivel a vendéglátásban – akárcsak a piac többi szegmensében – az árak folyamatosan változhatnak, ezért nem láttuk értelmét a forráskódba beépíteni az épp aktuális díjakat.

Megoldásként a Tervezés részben már említett módon létrehoztunk egy táblát az adatbázisban, ami tartalmazza a különböző tételek árait. Konkrétan három ilyen tétel létezik a jelenlegi verzióban:

- egy ágy ára /fő/éj
- pótágy ára /éj
- TV-s szoba esetén felár /éj

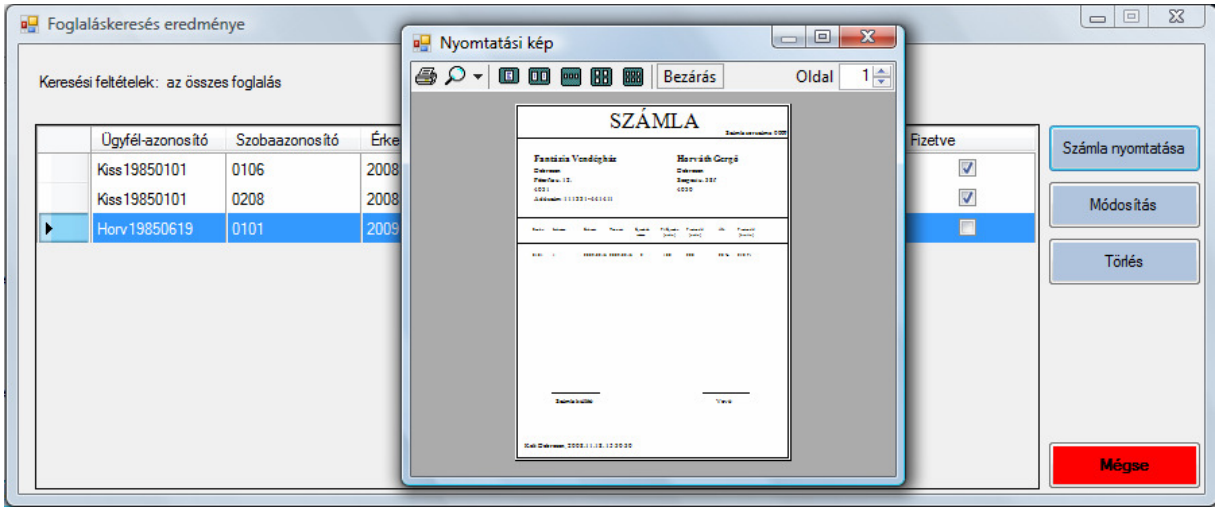
Az első indításkor lehetőségünk van megadni ezeket az értékeket a programban. Ha az így beállított árakat a későbbiekben módosítani szeretnénk, akkor megtehetjük a karbantartási lehetőségeken belül, az Árjegyzék gombra kattintva. Ehhez csak az adminisztrátornak van joga.



18. ábra

Fontos szerepe van a nyomtatási lehetőségnek. Alapvető elvárás, hogy egy foglalásról számlát tudjon kiállítani a vendéglátó. Ezt megkönnyítendő építettük be a Room Managerbe ezt a funkciót.

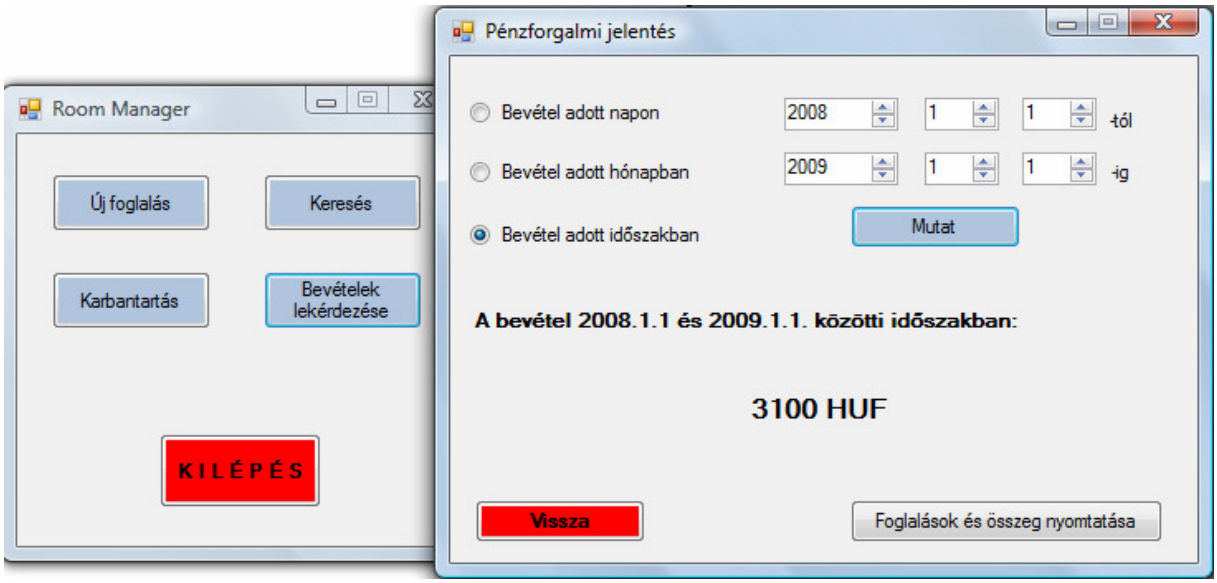
Ha rákeresünk egy foglalásra, akkor a Számla nyomtatása gomb segítségével regisztrálhatjuk a fizetést. Ekkor megjelenik egy nyomtatási kép a számláról, majd ki is nyomtathatjuk a számlát.



19. ábra

A fenti, nyomtatási képen látható számlán a vevő adatait az adatbázisból tölti ki a program. A számla sorszámát szintén az adatbázisban tároltuk le egy külön táblában. Ez az érték minden számlakiállításnál eggyel növekszik. Az eladó adatait a forráskódba építjük be, ennek megváltoztatásának lehetőségét nem tartottuk szükségesnek a szoftveren belül.

Lehetőség van a bevételek lekérdezésére is, ez a funkció a főablakon keresztül érhető el. A bevételeket lekérdezhetjük egy adott napra, hónapra szűkítve, vagy akár tetszőleges időintervallumot is megadhatunk.



20. ábra

Természetesen lehetőség van ezen foglalások nyomtatására is, így részletes jelentést kapunk az adott időszak foglalásairól és bevételeiről.

2.5. Tesztelés

Mielőtt éles működésbe helyezzük a szoftvert, tesztelés során kell megbizonyosodni róla, hogy az alkalmazás megfelelően működik-e minden lehetséges helyzetben. Ennek érdekében külön-külön mindketten próbára tettük programunk funkcióit.

Mivel a fejlesztés során folyamatosan teszteltük az éppen aktuálisan elkészített részfeladatot, ezért ebbe a fázisba érve már a tényleges működéssel kapcsolatban nem voltak kétségeink. Természetesen ez nem volt elegendő számunkra, így tesztelési folyamat alá vetettük a végeredményt is.

A funkcionális tesztünk folyamán úgy jártunk el, mintha egy saját vendégházat üzemeltetnénk. Ennek megfelelően első lépésként létrehoztunk két felhasználót. Felvezettük a képzelt árakat, majd rögzítettük az általunk kitalált szobák adatait. A keresési funkciók által bizonyosodtunk meg ennek sikerességéről.

Következő lépésként egy képzeletbeli ügyfélkört alkottunk. Ellenőriztük a felvitel során, hogy a program megfelelően kezeli-e, ha helytelen adatot próbálunk megadni. Mivel figyelmesen jártunk el a készítés során, ezért ezek a hibák nem okoznak gondot, mert a rossz adat javítása nélkül nem enged továbblépni az alkalmazás.

Szükség volt foglalásokra is. Tesztünk során adtunk meg szilvesztert magába foglaló tartózkodást, módosítottunk, töröltünk foglalást, minden szerintünk szóba jöhető esetet megvizsgáltunk.

Kipróbáltunk minden egyéb funkciót, beleértve a nyomtatást, ügyfelek adatainak módosítását, ezek a műveletek is megfelelően működnek.

A csoportmunka előnye, hogy ketten, két különböző adatbázissal tudtuk elvégezni a tesztelést. Mivel szoftverünk támogatja az importálást és exportálást, ezért - a foglalásokon kívül – minden általunk rögzített adatot megcseréltünk egymás között, majd az így kapott adatokkal újrateszteltük a programot.

A grafikus felületen túl az adatbázis változásait figyelemmel kísértük a MySQL parancssoros kliensprogramja segítségével is.

A tesztelés végére elégedetten kijelenthetjük, hogy sikerült olyan szoftvert készítenünk, mely stabilan működik, és különleges helyzetek sem okoznak gondot a használata során.

3. Fejleszthetőség

3.1. Hálózaton keresztüli működés

A fejleszthetőség egyik iránya lehet, hogy a MySQL szerver nem az általunk fejlesztett programmal azonos gépen fut. Előfordulhat ez például akkor, ha a programot egy kollégium szeretné használni, ahol már van egy szerver számítógép, amin a MySQL szervert szeretnék futtatni. Ehhez képessé kell tenni a szoftvert arra, hogy egy távoli számítógépen futtatott szolgáltatáshoz, jelen esetben a MySQL-hez kapcsolódjon.

Ahhoz, hogy a program ebben a környezetben is megfelelően működjön, meg kell változtatni a kapcsolódás módját. Ebben az esetben általános megoldás, hogy a bejelentkezési képernyőn választhassuk ki azt a távoli számítógépet, amelyhez kapcsolódni szeretnénk. Az alkalmazásnak ehhez célszerű tárolnia azokat a kapcsolatokat, amelyek esetleg már sikeresen megnyitásra kerültek az előző futások esetén, illetve ha nem volt még ilyen, akkor tudnia kell új kapcsolatot kialakítania és azt tesztelnie. A kapcsolat létrehozása ekkor IP cím alapján történik.

Természetesen, ha hálózaton keresztül kell igénybe venni a MySQL által nyújtott szolgáltatásokat, akkor elképzelhető, hogy egyszerre több helyről is szükség lenne az adatbázisban való módosítások lehetőségére. Ebben az esetben gondoskodni kell a konkurens hozzáférés megfelelő biztosításáról. Ha ugyanis egyszerre több helyről is elérhető egy adatbázis, akkor vigyázni kell arra, hogy a módosítások egyidejű végrehajtási kísérlete ne okozzon problémát. Erre megfelelő megoldást nyújthat a timeout értékek helyes megválasztása és a MySQL szerver konfigurálásakor az egyidejűleg engedélyezett kliensek átgondolt maximalizálása. Ebben a tekintetben a legbiztosabb megoldás az, ha egyszerre egy kapcsolatot engedélyezünk csak, amely minél kevesebb ideig foglalja a szerver erőforrásait.

Hálózati adatátvitel közben az is előfordulhat, hogy valamilyen hiba következik be a kommunikációt folytató felek közötti összeköttetésben. Ezt az alkalmazásnak tudnia kell felismerni, illetve kezelni. Ilyen esetekben kényelmi lehetőség lehet a felhasználónak, ha egy

esetleges kapcsolatszakadás után a program nem „száll el” azonnal, hanem megpróbál újra kapcsolódni, és ha ez is sikertelen, csak akkor értesíti a felhasználót.

A sikertelen műveletek elmentése is segítheti a program használatát. Megvalósítható, hogy emlékeztetőket hozzunk létre ezekhez, vagy az is, hogy egy későbbi sikeres kapcsolódás esetén automatikusan történjen meg a módosítás az adatbázisban.

3.2. Web

Mára az internet életünk elengedhetetlen részévé vált. Egyre inkább elterjedt, hogy ha keresünk valamit, információra van szükségünk, akkor nem a telefonhoz, vagy könyvekhez nyúlunk, hanem leülünk a számítógép elé. Éppen ezért, ha egy cég nem képviseli magát a világhálón, akkor azt az emberek nem fogják komoly piaci partnernek tekinteni.

A fenti tények ösztönöztek minket arra, hogy ne ragadjunk le egy egyszerű desktop alkalmazás készítésénél. Bár gyakorlatban az internetes elérést nem valósítottuk meg, tökéletes elképzelésünk van a fejlesztés ezen irányáról.

Az elképzelés egy olyan honlap készítése, amely online felhasználó-regisztrációt és szobafoglalást tesz lehetővé. A webes felületen lévő mezők megfelelnének a programunkban megtalálható részeknek.

Szoftverünket .NET keretrendszerben készítettük el, az online elérhető verziót pedig ASP.NET rendszerben készítenénk el.

A legkényelmesebb megoldás az lenne, ha a HTML oldalak általában statikus részét hagyományos módon, akár egy kényelmes, grafikus szerkesztővel készíthetnénk, és csak a dinamikus részt kellene programozni. Az Active Server Pages (ASP) elve pontosan ez: amikor azt mondjuk, ASP, tulajdonképpen egy HTML kódba ágyazott, speciális programozási módszerről beszélünk.

Fontos, hogy az ASP nem egy programozási nyelv, hanem csak egy keretrendszer. Az ASP oldal végrehajtásakor a webkiszolgáló végigszalad az oldal tartalmán, és ha abban ASP scriptrészletet talál, végrehajtja. A HTML oldal és a script által esetleg visszaadott kódrészletek együttesen képezik az eredményt.

A MySQL adatbázis kezelése ebben a formában is egyszerűen megoldható. Az így létrejövő honlap természetesen egy időben tudná ellátni a funkcióit az asztali alkalmazással, ami például egy recepció gépén futhatna.

4. Összefoglalás

Célunk egy piacképes, szobák, ügyfelek és foglalások adatait menedzselő szoftver elkészítése volt. Ennek érdekében felmértünk egy valós igényt, és saját ötleteinkkel kiegészítve haladtunk végig a tervezés és megvalósítás lépésein.

Munkánk során megtapasztaltuk, hogy mennyire fontosak az elsőre apró, lényegtelennek tűnő dolgok is. Ha ugyanis valamilyen apró hibát ejtünk, vagy elsiklunk egy kisebb részfeladat fölött, akkor az kihatással lehet a munkánk végeredményére.

Mivel tanulmányaink során viszonylag rövid határidővel, kisebb házi feladatokat, beadandókat kellett megoldanunk, ezért ez a feladat tökéletes volt arra, hogy más oldalról is megtapasztaljuk a szoftverfejlesztés folyamatát. A feladat összetettsége miatt szükség volt több részfeladat meghatározására, melyeket meghatározott sorrendben kellett megoldanunk. Mielőtt munkához láttunk volna, meg kellett tervezni, hogy pontosan hogyan és milyen eszközökkel kívánjuk elkészíteni szoftverünket.

A program elkészítése során tudásunk több területen is gyarapodott, mivel a számunkra eddig ismeretlen C# nyelvet használtuk és megismerkedtünk az adatbázisprogramozás menetével illetve fortélyaival is. A MySQL használatával lehetőségünk nyílt az egyetemi képzés alatt elsajátított elméleti tudásunk alkalmazására egy konkrét adatbázis létrehozásával és menedzselésével.

Fontos tapasztalatokat szereztünk a csapatban való együttműködéssel kapcsolatban, ugyanakkor az önálló munkavégzési készségünk is fejlődött, mivel a részfeladatokat külön-külön oldottuk meg. A közös munka és egymás segítése nem okozott problémát. Hasonló módon gondolkodunk, így ha valamelyikünk elakadt, akkor a felmerülő nehézséget ketten könnyebben tudtuk leküzdeni.

A Room Manager készítése, és a szakdolgozat megírása során a kitűzött cél elérésén felül a közös projektfejlesztés technikájának elsajátítását, valamint egymás munkájának segítését

tartottuk szem előtt. Ebből kifolyólag nem tudjuk az egyes részeket kettévágni, mindkettőnk munkája és ötlete megtalálható a program egészében.

Véleményünk szerint az elkészült program élesben, a mindennapi használat során is megállná a helyét. Legjobb tudásunknak megfelelően készítettük el, és a tesztelés során próbáltunk minden eshetőséget számba venni.

Mivel a bevezetésben említett vendégház csak a nyári időszakban működik, és egyelőre más esetleges megrendelő nem ismeri szoftverünket, ezért a tényleges használatra még nem került sor. Így lehetőségünk van követni az igények változását, valamint a Fejleszthetőség fejezetben leírt ötletek esetleges gyakorlati kivitelezésére is.

Táblázatok és ábrák jegyzéke

Irodalomjegyzék

[1] Hatvany Béla Csaba: MySQL.NET (BBS-Info Kft., 2007)

[2] Tim King – Gerge Reese – Randy J. Yarger: A MySQL kezelése és használata (Kossuth Kiadó zRt., 2002)

[3] Benkő Tiborné – Tóth Bertalan: Együtt könnyebb a programozás: C# (COMPUTERBOOKS, 2008)

[4] Siki Zoltán: Adatbáziskezelés és tervezés http://www.agt.bme.hu/szakm/adatb/db3.htm

[5] Verziókezelő rendszer – HupWiki http://wiki.hup.hu/index.php/Verzi%C3%B3kezel%C5%91_rendszer

[6] Az ASP alapjai http://www.archiweb.hu/aspseged/tankonyv/page1.htm

Köszönetnyilvánítás

Ezúton szeretnénk köszönetet mondani témavezetõnknek, Dr. Végh Jánosnak, amiért elvállalta a témavezetéssel járó feladatokat, valamint szakdolgozatunk elkészítésében segített és útmutatást nyújtott.